TensorLayer Documentation

Release 2.0.1

TensorLayer contributors

May 23, 2019

CONTENTS

Documentation Version: 2.0.1

Good News: We won the **Best Open Source Software Award** @ACM Multimedia (MM) 2017.

TensorLayer is a Deep Learning (DL) and Reinforcement Learning (RL) library extended from Google TensorFlow. It provides popular DL and RL modules that can be easily customized and assembled for tackling real-world machine learning problems. More details can be found here.

Note: If you got problem to read the docs online, you could download the repository on GitHub, then go to `/docs/_build/html/index.html` to read the docs offline. The `_build` folder can be generated in `docs` using `make html`.

CHAPTER

ONE

USER GUIDE

The TensorLayer user guide explains how to install TensorFlow, CUDA and cuDNN, how to build and train neural networks using TensorLayer, and how to contribute to the library as a developer.

1.1 Installation

TensorLayer has some prerequisites that need to be installed first, including TensorFlow , numpy and matplotlib. For GPU support CUDA and cuDNN are required.

If you run into any trouble, please check the TensorFlow installation instructions which cover installing the TensorFlow for a range of operating systems including Mac OX, Linux and Windows, or ask for help on tensorlayer@gmail.com or FAQ.

1.1.1 Step 1 : Install dependencies

TensorLayer is build on the top of Python-version TensorFlow, so please install Python first.

Note: We highly recommend python3 instead of python2 for the sake of future.

Python includes `pip` command for installing additional modules is recommended. Besides, a virtual environment via `virtualenv` can help you to manage python packages.

Take Python3 on Ubuntu for example, to install Python includes `pip`, run the following commands:

```
sudo apt-get install python3
sudo apt-get install python3-pip
sudo pip3 install virtualenv
```

To build a virtual environment and install dependencies into it, run the following commands: (You can also skip to Step 3, automatically install the prerequisites by TensorLayer)

```
virtualenv env
env/bin/pip install matplotlib
env/bin/pip install numpy
env/bin/pip install scipy
env/bin/pip install scikit-image
```

To check the installed packages, run the following command:

```
env/bin/pip list
```

After that, you can run python script by using the virtual python as follow.

```
env/bin/python *.py
```

1.1.2 Step 2 : TensorFlow

The installation instructions of TensorFlow are written to be very detailed on TensorFlow website. However, there are something need to be considered. For example, TensorFlow officially supports GPU acceleration for Linux, Mac OX and Windows at present.

> **Warning:** For ARM processor architecture, you need to install TensorFlow from source.

1.1.3 Step 3 : TensorLayer

For latest code for TensorLayer 2.0, please build from the source.

```
# First clone the repository and change the current directory to the newly cloned␣
↪repository
git clone https://github.com/zsdonghao/tensorlayer2.git
cd tensorlayer2

# Install virtualenv if necessary
pip install virtualenv
# Then create a virtualenv called `venv`
virtualenv venv

# Activate the virtualenv

## Linux:
source venv/bin/activate

## Windows:
venv\Scripts\activate.bat

# basic installation
pip install .

# ============= IF TENSORFLOW IS NOT ALREADY INSTALLED ============= #

# for a machine **without** an NVIDIA GPU
pip install -e ".[all_cpu_dev]"

# for a machine **with** an NVIDIA GPU
pip install -e ".[all_gpu_dev]"
```

If you want install TensorLayer 1.X, the simplest way to install TensorLayer 1.X is as follow. It will also install the numpy and matplotlib automatically.

```
[stable version] pip install tensorlayer
[master version] pip install git+https://github.com/tensorlayer/tensorlayer.git
```

However, if you want to modify or extend TensorLayer 1.X, you can download the repository from Github and install it as follow.

```
cd to the root of the git tree
pip install -e .
```

This command will run the `setup.py` to install TensorLayer. The `-e` reflects editable, then you can edit the source code in `tensorlayer` folder, and `import` the edited TensorLayer.

1.1.4 Step 4 : GPU support

Thanks to NVIDIA supports, training a fully connected network on a GPU, which may be 10 to 20 times faster than training them on a CPU. For convolutional network, may have 50 times faster. This requires an NVIDIA GPU with CUDA and cuDNN support.

CUDA

The TensorFlow website also teach how to install the CUDA and cuDNN, please see TensorFlow GPU Support.

Download and install the latest CUDA is available from NVIDIA website:

- CUDA download and install

If CUDA is set up correctly, the following command should print some GPU information on the terminal:

```
python -c "import tensorflow"
```

cuDNN

Apart from CUDA, NVIDIA also provides a library for common neural network operations that especially speeds up Convolutional Neural Networks (CNNs). Again, it can be obtained from NVIDIA after registering as a developer (it take a while):

Download and install the latest cuDNN is available from NVIDIA website:

- cuDNN download and install

To install it, copy the `*.h` files to `/usr/local/cuda/include` and the `lib*` files to `/usr/local/cuda/lib64`.

1.1.5 Windows User

TensorLayer is built on the top of Python-version TensorFlow, so please install Python first. NoteWe highly recommend installing Anaconda. The lowest version requirements of Python is py35.

Anaconda download

GPU support

Thanks to NVIDIA supports, training a fully connected network on a GPU, which may be 10 to 20 times faster than training them on a CPU. For convolutional network, may have 50 times faster. This requires an NVIDIA GPU with CUDA and cuDNN support.

1. Installing Microsoft Visual Studio

You should preinstall Microsoft Visual Studio (VS) before installing CUDA. The lowest version requirements is VS2010. We recommend installing VS2015 or VS2013. CUDA7.5 supports VS2010, VS2012 and VS2013. CUDA8.0 also supports VS2015.

2. Installing CUDA

Download and install the latest CUDA is available from NVIDIA website:

CUDA download

We do not recommend modifying the default installation directory.

3. Installing cuDNN

The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. Download and extract the latest cuDNN is available from NVIDIA website:

cuDNN download

After extracting cuDNN, you will get three folders (bin, lib, include). Then these folders should be copied to CUDA installation. (The default installation directory is *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0*)

Installing TensorLayer

For TensorLayer 2.0, please refer to the steps mentioned above.

For TensorLayer 1.X, you can easily install Tensorlayer 1.X using pip in CMD

```
pip install tensorflow        #CPU version
pip install tensorflow-gpu    #GPU version (GPU version and CPU version just choose␣
↪one)
pip install tensorlayer       #Install tensorlayer
```

Test

Enter "python" in CMD. Then:

```
import tensorlayer
```

If there is no error and the following output is displayed, the GPU version is successfully installed.

```
successfully opened CUDA library cublas64_80.dll locally
successfully opened CUDA library cuDNN64_5.dll locally
successfully opened CUDA library cufft64_80.dll locally
successfully opened CUDA library nvcuda.dll locally
successfully opened CUDA library curand64_80.dll locally
```

If there is no error, the CPU version is successfully installed.

1.1.6 Issue

If you get the following output when import tensorlayer, please read FQA.

```
_tkinter.TclError: no display name and no $DISPLAY environment variable
```

1.2 Examples

1.2.1 Basics

- Multi-layer perceptron (MNIST), simple usage. Classification task, see tutorial_mnist_simple.py.
- Multi-layer perceptron (MNIST), dynamic model. Classification with dropout using iterator, see tutorial_mnist_mlp_dynamic.py method2.
- Multi-layer perceptron (MNIST), static model. Classification with dropout using iterator, see tutorial_mnist_mlp_static.py.
- Convolutional Network (CIFAR-10). Classification task, see tutorial_cifar10_cnn_static.py.
- TensorFlow dataset API for object detection see here.
- Merge Keras into TensorLayer. tutorial_keras.py.
- Data augmentation with TFRecord. Effective way to load and pre-process data, see tutorial_tfrecord*.py and tutorial_cifar10_tfrecord.py.
- Data augmentation with TensorLayer. See tutorial_fast_affine_transform.py (for quick test only).

1.2.2 Pretrained Models

- VGG 16 (ImageNet). Classification task, see tutorial_models_vgg16.
- VGG 19 (ImageNet). Classification task, see tutorial_models_vgg19.py.
- SqueezeNet (ImageNet). Model compression, see tutorial_models_squeezenetv1.py.
- MobileNet (ImageNet). Model compression, see tutorial_models_mobilenetv1.py.
- All pretrained models in pretrained-models.

1.2.3 Vision

- Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, see examples.
- ArcFace: Additive Angular Margin Loss for Deep Face Recognition, see InsignFace.
- BinaryNet. Model compression, see mnist cifar10.
- Ternary Weight Network. Model compression, see mnist cifar10.
- DoReFa-Net. Model compression, see mnist cifar10.
- QuanCNN. Model compression, sees mnist cifar10.
- Wide ResNet (CIFAR) by ritchieng.
- Spatial Transformer Networks by zsdonghao.

- U-Net for brain tumor segmentation by zsdonghao.
- Variational Autoencoder (VAE) for (CelebA) by yzwxx.
- Variational Autoencoder (VAE) for (MNIST) by BUPTLdy.
- Image Captioning - Reimplementation of Google's im2txt by zsdonghao.

1.2.4 Adversarial Learning

- DCGAN (CelebA). Generating images by Deep Convolutional Generative Adversarial Networks by zsdonghao.
- Generative Adversarial Text to Image Synthesis by zsdonghao.
- Unsupervised Image to Image Translation with Generative Adversarial Networks by zsdonghao.
- Improved CycleGAN with resize-convolution by luoxier.
- Super Resolution GAN by zsdonghao.
- BEGAN: Boundary Equilibrium Generative Adversarial Networks by 2wins.
- DAGAN: Fast Compressed Sensing MRI Reconstruction by nebulaV.

1.2.5 Natural Language Processing

- Recurrent Neural Network (LSTM). Apply multiple LSTM to PTB dataset for language modeling, see tutorial_ptb_lstm_state_is_tuple.py.
- Word Embedding (Word2vec). Train a word embedding matrix, see tutorial_word2vec_basic.py.
- Restore Embedding matrix. Restore a pre-train embedding matrix, see tutorial_generate_text.py.
- Text Generation. Generates new text scripts, using LSTM network, see tutorial_generate_text.py.
- Chinese Text Anti-Spam by pakrchen.
- Chatbot in 200 lines of code for Seq2Seq.
- FastText Sentence Classification (IMDB), see tutorial_imdb_fasttext.py by tomtung.

1.2.6 Reinforcement Learning

- Policy Gradient / Network (Atari Ping Pong), see tutorial_atari_pong.py.
- Deep Q-Network (Frozen lake), see tutorial_frozenlake_dqn.py.
- Q-Table learning algorithm (Frozen lake), see tutorial_frozenlake_q_table.py.
- Asynchronous Policy Gradient using TensorDB (Atari Ping Pong) by nebulaV.
- AC for discrete action space (Cartpole), see tutorial_cartpole_ac.py.
- A3C for continuous action space (Bipedal Walker), see tutorial_bipedalwalker_a3c*.py.
- DAGGER for (Gym Torcs) by zsdonghao.
- TRPO for continuous and discrete action space by jjkke88.

1.2.7 Miscellaneous

- TensorDB by fangde see tl_paper.
- A simple web service - TensorFlask by JoelKronander.

1.3 Contributing

TensorLayer is a major ongoing research project in Data Science Institute, Imperial College London. The goal of the project is to develop a compositional language while complex learning systems can be built through composition of neural network modules.

Numerous contributors come from various horizons such as: Tsinghua University, Carnegie Mellon University, University of Technology of Compiegne, Google, Microsoft, Bloomberg and etc.

There are many functions need to be contributed such as Maxout, Neural Turing Machine, Attention, TensorLayer Mobile and etc.

You can easily open a Pull Request (PR) on GitHub, every little step counts and will be credited. As an open-source project, we highly welcome and value contributions!

If you are interested in working with us, please contact us at: tensorlayer@gmail.com.

1.3.1 Project Maintainers

The TensorLayer project was started by Hao Dong at Imperial College London in June 2016.

For TensorLayer 2.x, it is now actively developing and maintaining by the following people who has more than 50 contributions*:

- **Hao Dong** (@zsdonghao) - https://zsdonghao.github.io
- **Jingqing Zhang** (@JingqingZ) - https://jingqingz.github.io
- **Rundi Wu** (@ChrisWu1997) - http://chriswu1997.github.io
- **Ruihai Wu** (@warshallrho) - https://warshallrho.github.io/

For TensorLayer 1.x, it was actively developed and maintained by the following people *(in alphabetical order)*:

- **Akara Supratak** (@akaraspt) - https://akaraspt.github.io
- **Fangde Liu** (@fangde) - http://fangde.github.io/
- **Guo Li** (@lgarithm) - https://lgarithm.github.io
- **Hao Dong** (@zsdonghao) - https://zsdonghao.github.io
- **Jonathan Dekhtiar** (@DEKHTIARJonathan) - https://www.jonathandekhtiar.eu
- **Luo Mai** (@luomai) - http://www.doc.ic.ac.uk/~lm111/
- **Simiao Yu** (@nebulaV) - https://nebulav.github.io

Numerous other contributors can be found in the Github Contribution Graph.

1.3.2 What to contribute

Your method and example

If you have a new method or example in terms of Deep learning and Reinforcement learning, you are welcome to contribute.

- Provide your layer or example, so everyone can use it.
- Explain how it would work, and link to a scientific paper if applicable.
- Keep the scope as narrow as possible, to make it easier to implement.

Report bugs

Report bugs at the GitHub, we normally will fix it in 5 hours. If you are reporting a bug, please include:

- your TensorLayer, TensorFlow and Python version.
- steps to reproduce the bug, ideally reduced to a few Python commands.
- the results you obtain, and the results you expected instead.

If you are unsure whether the behavior you experience is a bug, or if you are unsure whether it is related to TensorLayer or TensorFlow, please just ask on our mailing list first.

Fix bugs

Look through the GitHub issues for bug reports. Anything tagged with "bug" is open to whoever wants to implement it. If you discover a bug in TensorLayer you can fix yourself, by all means feel free to just implement a fix and not report it first.

Write documentation

Whenever you find something not explained well, misleading, glossed over or just wrong, please update it! The *Edit on GitHub* link on the top right of every documentation page and the *[source]* link for every documented entity in the API reference will help you to quickly locate the origin of any text.

1.3.3 How to contribute

Edit on GitHub

As a very easy way of just fixing issues in the documentation, use the *Edit on GitHub* link on the top right of a documentation page or the *[source]* link of an entity in the API reference to open the corresponding source file in GitHub, then click the *Edit this file* link to edit the file in your browser and send us a Pull Request. All you need for this is a free GitHub account.

For any more substantial changes, please follow the steps below to setup TensorLayer for development.

Documentation

The documentation is generated with Sphinx. To build it locally, run the following commands:

```
pip install Sphinx
sphinx-quickstart

cd docs
make html
```

If you want to re-generate the whole docs, run the following commands:

```
cd docs
make clean
make html
```

To write the docs, we recommend you to install Local RTD VM.

Afterwards, open `docs/_build/html/index.html` to view the documentation as it would appear on readthedocs. If you changed a lot and seem to get misleading error messages or warnings, run `make clean html` to force Sphinx to recreate all files from scratch.

When writing docstrings, follow existing documentation as much as possible to ensure consistency throughout the library. For additional information on the syntax and conventions used, please refer to the following documents:

- reStructuredText Primer
- Sphinx reST markup constructs
- A Guide to NumPy/SciPy Documentation

Testing

TensorLayer has a code coverage of 100%, which has proven very helpful in the past, but also creates some duties:

- Whenever you change any code, you should test whether it breaks existing features by just running the test scripts.
- Every bug you fix indicates a missing test case, so a proposed bug fix should come with a new test that fails without your fix.

Sending Pull Requests

When you're satisfied with your addition, the tests pass and the documentation looks good without any markup errors, commit your changes to a new branch, push that branch to your fork and send us a Pull Request via GitHub's web interface.

All these steps are nicely explained on GitHub: https://guides.github.com/introduction/flow/

When filing your Pull Request, please include a description of what it does, to help us reviewing it. If it is fixing an open issue, say, issue #123, add *Fixes #123*, *Resolves #123* or *Closes #123* to the description text, so GitHub will close it when your request is merged.

1.4 Get Involved in Research

1.4.1 Data Science Institute, Imperial College London

Data science is therefore by nature at the core of all modern transdisciplinary scientific activities, as it involves the whole life cycle of data, from acquisition and exploration to analysis and communication of the results. Data science is not only concerned with the tools and methods to obtain, manage and analyse data: it is also about extracting value from data and translating it from asset to product.

Launched on 1st April 2014, the Data Science Institute at Imperial College London aims to enhance Imperial's excellence in data-driven research across its faculties by fulfilling the following objectives.

The Data Science Institute is housed in purpose built facilities in the heart of the Imperial College campus in South Kensington. Such a central location provides excellent access to collabroators across the College and across London.

- To act as a focal point for coordinating data science research at Imperial College by facilitating access to funding, engaging with global partners, and stimulating cross-disciplinary collaboration.
- To develop data management and analysis technologies and services for supporting data driven research in the College.
- To promote the training and education of the new generation of data scientist by developing and coordinating new degree courses, and conducting public outreach programmes on data science.
- To advise College on data strategy and policy by providing world-class data science expertise.
- To enable the translation of data science innovation by close collaboration with industry and supporting commercialization.

If you are interested in working with us, please check our vacancies and other ways to get involved , or feel free to contact us.

1.5 FAQ

1.5.1 How to effectively learn TensorLayer

No matter what stage you are in, we recommend you to spend just 10 minutes to read the source code of TensorLayer and the Understand layer / Your layer in this website, you will find the abstract methods are very simple for everyone. Reading the source codes helps you to better understand TensorFlow and allows you to implement your own methods easily. For discussion, we recommend Gitter, Help Wanted Issues, QQ group and Wechat group.

Beginner

For people who new to deep learning, the contributors provided a number of tutorials in this website, these tutorials will guide you to understand autoencoder, convolutional neural network, recurrent neural network, word embedding and deep reinforcement learning and etc. If your already understand the basic of deep learning, we recommend you to skip the tutorials and read the example codes on Github , then implement an example from scratch.

Engineer

For people from industry, the contributors provided mass format-consistent examples covering computer vision, natural language processing and reinforcement learning. Besides, there are also many TensorFlow users already implemented product-level examples including image captioning, semantic/instance segmentation, machine translation, chatbot and etc., which can be found online. It is worth noting that a wrapper especially for computer vision Tf-Slim can be connected with TensorLayer seamlessly. Therefore, you may able to find the examples that can be used in your project.

Researcher

For people from academia, TensorLayer was originally developed by PhD students who facing issues with other libraries on implement novel algorithm. Installing TensorLayer in editable mode is recommended, so you can extend your methods in TensorLayer. For research related to image processing such as image captioning, visual QA and etc., you may find it is very helpful to use the existing Tf-Slim pre-trained models with TensorLayer (a specially layer for connecting Tf-Slim is provided).

1.5.2 Exclude some layers from training

You may need to get the list of variables you want to update, TensorLayer provides two ways to get the variables list.

The first way is to use the all_params of a network, by default, it will store the variables in order. You can print the variables information via `tl.layers.print_all_variables(train_only=True)` or `network.print_params(details=False)`. To choose which variables to update, you can do as below.

```
train_params = network.trainable_weights[3:]
```

The second way is to get the variables by a given name. For example, if you want to get all variables which the layer name contains `dense`, you can do as below.

```
train_params = network.get_layer('dense').trainable_weights
```

After you get the variable list, you can define your optimizer like that so as to update only a part of the variables.

```
train_weights = network.trainable_weights
optimizer.apply_gradients(zip(grad, train_weights))
```

1.5.3 Logging

TensorLayer adopts the Python logging module to log running information. The logging module would print logs to the console in default. If you want to configure the logging module, you shall follow its manual.

1.5.4 Visualization

Cannot Save Image

If you run the script via SSH control, sometimes you may find the following error.

```
_tkinter.TclError: no display name and no $DISPLAY environment variable
```

If this happens, run `sudo apt-get install python3-tk` or `import matplotlib` and `matplotlib.use('Agg')` before `import tensorlayer as tl`. Alternatively, add the following code into the top of `visualize.py` or in your own code.

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

1.5.5 Install Master Version

To use all new features of TensorLayer, you need to install the master version from Github. Before that, you need to make sure you already installed git.

```
[stable version] pip install tensorlayer
[master version] pip install git+https://github.com/tensorlayer/tensorlayer.git
```

1.5.6 Editable Mode

- 1. Download the TensorLayer folder from Github.
- 2. Before editing the TensorLayer `.py` file.
  - If your script and TensorLayer folder are in the same folder, when you edit the `.py` inside TensorLayer folder, your script can access the new features.
  - If your script and TensorLayer folder are not in the same folder, you need to run the following command in the folder contains `setup.py` before you edit `.py` inside TensorLayer folder.

```
pip install -e .
```

1.5.7 Load Model

Note that, the `tl.files.load_npz()` can only able to load the npz model saved by `tl.files.save_npz()`. If you have a model want to load into your TensorLayer network, you can first assign your parameters into a list in order, then use `tl.files.assign_params()` to load the parameters into your TensorLayer model.

1.6 Define a model

TensorLayer provides two ways to define a model. Static model allows you to build model in a fluent way while dynamic model allows you to fully control the forward process.

1.6.1 Static model

```
import tensorflow as tf
from tensorlayer.layers import Input, Dropout, Dense
from tensorlayer.models import Model

def get_model(inputs_shape):
    ni = Input(inputs_shape)
    nn = Dropout(keep=0.8)(ni)
    nn = Dense(n_units=800, act=tf.nn.relu, name="dense1")(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=800, act=tf.nn.relu)(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=10, act=tf.nn.relu)(nn)
    M = Model(inputs=ni, outputs=nn, name="mlp")
    return M

MLP = get_model([None, 784])
```

(continues on next page)

(continued from previous page)

```
MLP.eval()
outputs = MLP(data)
```

1.6.2 Dynamic model

In this case, you need to manually input the output shape of the previous layer to the new layer.

```
class CustomModel(Model):

    def __init__(self):
        super(CustomModel, self).__init__()

        self.dropout1 = Dropout(keep=0.8)
        self.dense1 = Dense(n_units=800, act=tf.nn.relu, in_channels=784)
        self.dropout2 = Dropout(keep=0.8)#(self.dense1)
        self.dense2 = Dense(n_units=800, act=tf.nn.relu, in_channels=800)
        self.dropout3 = Dropout(keep=0.8)#(self.dense2)
        self.dense3 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)

    def forward(self, x, foo=False):
        z = self.dropout1(x)
        z = self.dense1(z)
        z = self.dropout2(z)
        z = self.dense2(z)
        z = self.dropout3(z)
        out = self.dense3(z)
        if foo:
            out = tf.nn.relu(out)
        return out

MLP = CustomModel()
MLP.eval()
outputs = MLP(data, foo=True) # controls the forward here
outputs = MLP(data, foo=False)
```

1.6.3 Reuse weights

For static model, call the layer multiple time in model creation

```
# create siamese network

def create_base_network(input_shape):
      '''Base network to be shared (eq. to feature extraction).
      '''
      input = Input(shape=input_shape)
      x = Flatten()(input)
      x = Dense(128, act=tf.nn.relu)(x)
      x = Dropout(0.9)(x)
      x = Dense(128, act=tf.nn.relu)(x)
      x = Dropout(0.9)(x)
      x = Dense(128, act=tf.nn.relu)(x)
      return Model(input, x)
```

(continues on next page)

(continued from previous page)

```
def get_siamese_network(input_shape):
      """Create siamese network with shared base network as layer
      """
      base_layer = create_base_network(input_shape).as_layer() # convert model as␣
↪layer

      ni_1 = Input(input_shape)
      ni_2 = Input(input_shape)
      nn_1 = base_layer(ni_1) # call base_layer twice
      nn_2 = base_layer(ni_2)
      return Model(inputs=[ni_1, ni_2], outputs=[nn_1, nn_2])

siamese_net = get_siamese_network([None, 784])
```

For dynamic model, call the layer multiple time in forward function

```
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.dense_shared = Dense(n_units=800, act=tf.nn.relu, in_channels=784)
        self.dense1 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)
        self.dense2 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)
        self.cat = Concat()

    def forward(self, x):
        x1 = self.dense_shared(x) # call dense_shared twice
        x2 = self.dense_shared(x)
        x1 = self.dense1(x1)
        x2 = self.dense2(x2)
        out = self.cat([x1, x2])
        return out

model = MyModel()
```

1.6.4 Print model information

```
print(MLP) # simply call print function

# Model(
#   (_inputlayer): Input(shape=[None, 784], name='_inputlayer')
#   (dropout): Dropout(keep=0.8, name='dropout')
#   (dense): Dense(n_units=800, relu, in_channels='784', name='dense')
#   (dropout_1): Dropout(keep=0.8, name='dropout_1')
#   (dense_1): Dense(n_units=800, relu, in_channels='800', name='dense_1')
#   (dropout_2): Dropout(keep=0.8, name='dropout_2')
#   (dense_2): Dense(n_units=10, relu, in_channels='800', name='dense_2')
# )
```

1.6.5 Get specific weights

We can get the specific weights by indexing or naming.

```
# indexing
all_weights = MLP.all_weights
some_weights = MLP.all_weights[1:3]

# naming
some_weights = MLP.get_layer('dense1').all_weights
```

1.6.6 Save and restore model

We provide two ways to save and restore models

Save weights only

```
MLP.save_weights('./model_weights.h5') # by default, file will be in hdf5 format
MLP.load_weights('./model_weights.h5')
```

Save model architecture and weights(optional)

```
# When using Model.load(), there is no need to reimplement or declare the␣
↪architecture of the model explicitly in code
MLP.save('./model.h5', save_weights=True)
MLP = Model.load('./model.h5', load_weights=True)
```

1.6.7 Customizing layer

The fully-connected layer is

z = f(x*W+b)

```
class Dense(Layer):
    def __init__(self, n_units, act=None, in_channels=None, name=None):
        super(Dense, self).__init__(name)

        self.n_units = n_units
        self.act = act
        self.in_channels = in_channels

        # for dynamic model, it needs the input shape to get the shape of W
        if self.in_channels is not None:
            self.build(self.in_channels)
            self._built = True

    def build(self, inputs_shape):
        if self.in_channels is None and len(inputs_shape) != 2:
            raise AssertionError("The input dimension must be rank 2, please reshape␣
↪or flatten it")
        if self.in_channels:
            shape = [self.in_channels, self.n_units]
        else:
            self.in_channels = inputs_shape[1]
            shape = [inputs_shape[1], self.n_units]
```

(continues on next page)

(continued from previous page)

```
        self.W = self._get_weights("weights", shape=tuple(shape))
        if self.b_init:
            self.b = self._get_weights("biases", shape=(self.n_units, ))

    @tf.function
    def forward(self, inputs):
        z = tf.matmul(inputs, self.W)
        if self.b_init:
            z = tf.add(z, self.b)
        if self.act:
            z = self.act(z)
        return z
```

1.7 Advanced features

1.7.1 Pre-trained CNN

Get entire CNN

```
import tensorflow as tf
import tensorlayer as tl
import numpy as np
from tensorlayer.models.imagenet_classes import class_names

vgg = tl.models.vgg16(pretrained=True)
img = tl.vis.read_image('data/tiger.jpeg')
img = tl.prepro.imresize(img, (224, 224)).astype(np.float32) / 255
output = vgg(img, is_train=False)
```

Get a part of CNN

```
# get VGG without the last layer
cnn = tl.models.vgg16(end_with='fc2_relu', mode='static').as_layer()
# add one more layer and build a new model
ni = Input([None, 224, 224, 3], name="inputs")
nn = cnn(ni)
nn = tl.layers.Dense(n_units=100, name='out')(nn)
model = tl.models.Model(inputs=ni, outputs=nn)
# train your own classifier (only update the last layer)
train_params = model.get_layer('out').all_weights
```

Reuse CNN

```
# in dynamic model, we can directly use the same model
# in static model
vgg_layer = tl.models.vgg16().as_layer()
ni_1 = tl.layers.Input([None, 224, 224, 3])
ni_2 = tl.layers.Input([None, 224, 224, 3])
a_1 = vgg_layer(ni_1)
```

(continues on next page)

(continued from previous page)

```
a_2 = vgg_layer(ni_2)
M = Model(inputs=[ni_1, ni_2], outputs=[a_1, a_2])
```

CHAPTER

TWO

API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API - Activations

To make TensorLayer simple, we minimize the number of activation functions as much as we can. So we encourage you to use TensorFlow's function. TensorFlow provides `tf.nn.relu`, `tf.nn.relu6`, `tf.nn.elu`, `tf.nn.softplus`, `tf.nn.softsign` and so on. More TensorFlow official activation functions can be found here. For parametric activation, please read the layer APIs.

The shortcut of `tensorlayer.activation` is `tensorlayer.act`.

2.1.1 Your activation

Customizes activation function in TensorLayer is very easy. The following example implements an activation that multiplies its input by 2. For more complex activation, TensorFlow API will be required.

```
def double_activation(x):
    return x * 2
```

A file containing various activation functions.

leaky_relu(x[, alpha, name])	leaky_relu can be used through its shortcut: `tl.act.lrelu()`.
leaky_relu6(x[, alpha, name])	*leaky_relu6()* can be used through its shortcut: `tl.act.lrelu6()`.
leaky_twice_relu6(x[, alpha_low, . . .])	*leaky_twice_relu6()* can be used through its shortcut: `:func:`tl.act.ltrelu6()`.
ramp(x[, v_min, v_max, name])	Ramp activation function.
swish(x[, name])	Swish function.
sign(x)	Sign function.
hard_tanh(x[, name])	Hard tanh activation function.
pixel_wise_softmax(x[, name])	Return the softmax outputs of images, every pixels have multiple label, the sum of a pixel is 1.

2.1.2 Ramp

`tensorlayer.activation.`**`ramp`**(*x*, *v_min=0*, *v_max=1*, *name=None*)

Ramp activation function.

Reference: [tf.clip_by_value]<https://www.tensorflow.org/api_docs/python/tf/clip_by_value>

Parameters

- **x** (*Tensor*) – input.
- **v_min** (*float*) – cap input to v_min as a lower bound.
- **v_max** (*float*) – cap input to v_max as a upper bound.
- **name** (*str*) – The function name (optional).

Returns A `Tensor` in the same type as `x`.

Return type Tensor

2.1.3 Leaky ReLU

`tensorlayer.activation.`**`leaky_relu`**(*x*, *alpha=0.2*, *name='leaky_relu'*)

leaky_relu can be used through its shortcut: `tl.act.lrelu()`.

This function is a modified version of ReLU, introducing a nonzero gradient for negative input. Introduced by the paper: Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]

The function return the following results:

- When x < 0: `f(x) = alpha_low * x`.
- When x >= 0: `f(x) = x`.

Parameters

- **x** (*Tensor*) – Support input type `float`, `double`, `int32`, `int64`, `uint8`, `int16`, or `int8`.
- **alpha** (*float*) – Slope.
- **name** (*str*) – The function name (optional).

Examples

```
>>> import tensorlayer as tl
>>> net = tl.layers.Input([10, 200])
>>> net = tl.layers.Dense(n_units=100, act=lambda x : tl.act.lrelu(x, 0.2), name=
↪'dense')(net)
```

Returns A `Tensor` in the same type as `x`.

Return type Tensor

References

- Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]

2.1.4 Leaky ReLU6

`tensorlayer.activation.leaky_relu6`(*x*, *alpha=0.2*, *name='leaky_relu6'*)

leaky_relu6() can be used through its shortcut: `tl.act.lrelu6()`.

This activation function is a modified version *leaky_relu()* introduced by the following paper: Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]

This activation function also follows the behaviour of the activation function `tf.nn.relu6()` introduced by the following paper: Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

The function return the following results:

- When x < 0: `f(x) = alpha_low * x`.
- When x in [0, 6]: `f(x) = x`.
- When x > 6: `f(x) = 6`.

Parameters

- **x** (*Tensor*) – Support input type `float`, `double`, `int32`, `int64`, `uint8`, `int16`, or `int8`.
- **alpha** (*float*) – Slope.
- **name** (*str*) – The function name (optional).

Examples

```
>>> import tensorlayer as tl
>>> net = tl.layers.Input([10, 200])
>>> net = tl.layers.Dense(n_units=100, act=lambda x : tl.act.leaky_relu6(x, 0.2),
↪name='dense')(net)
```

Returns A `Tensor` in the same type as `x`.

Return type Tensor

References

- Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]
- Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

2.1.5 Twice Leaky ReLU6

`tensorlayer.activation.leaky_twice_relu6`(*x*, *alpha_low=0.2*, *alpha_high=0.2*, *name='leaky_relu6'*)

leaky_twice_relu6() can be used through its shortcut: `:func:`tl.act.ltrelu6()`.

This activation function is a modified version *leaky_relu()* introduced by the following paper: Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]

This activation function also follows the behaviour of the activation function `tf.nn.relu6()` introduced by the following paper: Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

This function push further the logic by adding *leaky* behaviour both below zero and above six.

The function return the following results:

- When x < 0: `f(x) = alpha_low * x`.
- When x in [0, 6]: `f(x) = x`.
- When x > 6: `f(x) = 6 + (alpha_high * (x-6))`.

Parameters

- **x** (*Tensor*) – Support input type `float`, `double`, `int32`, `int64`, `uint8`, `int16`, or `int8`.
- **alpha_low** (*float*) – Slope for x < 0: `f(x) = alpha_low * x`.
- **alpha_high** (*float*) – Slope for x < 6: `f(x) = 6 (alpha_high * (x-6))`.
- **name** (*str*) – The function name (optional).

Examples

```
>>> import tensorlayer as tl
>>> net = tl.layers.Input([10, 200])
>>> net = tl.layers.Dense(n_units=100, act=lambda x : tl.act.leaky_twice_relu6(x,␣
↪0.2, 0.2), name='dense')(net)
```

Returns A `Tensor` in the same type as `x`.

Return type Tensor

References

- Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]
- Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

2.1.6 Swish

`tensorlayer.activation.`**`swish`**(*x*, *name='swish'*)

Swish function.

See Swish: a Self-Gated Activation Function.

Parameters

- **x** (*Tensor*) – input.
- **name** (*str*) – function name (optional).

Returns A `Tensor` in the same type as `x`.

Return type Tensor

2.1.7 Sign

`tensorlayer.activation.sign(x)`

Sign function.

Clip and binarize tensor using the straight through estimator (STE) for the gradient, usually be used for quantizing values in *Binarized Neural Networks*: https://arxiv.org/abs/1602.02830.

Parameters **x** (*Tensor*) – input.

Returns A `Tensor` in the same type as `x`.

Return type Tensor

References

- ***Rectifier Nonlinearities Improve Neural Network Acoustic Models, Maas et al. (2013)*** http://web.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf
- ***BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, Courbariaux et al. (2*** https://arxiv.org/abs/1602.02830

2.1.8 Hard Tanh

`tensorlayer.activation.hard_tanh(x, name='htanh')`

Hard tanh activation function.

Which is a ramp function with low bound of -1 and upper bound of 1, shortcut is *htanh*.

Parameters

- **x** (*Tensor*) – input.
- **name** (*str*) – The function name (optional).

Returns A `Tensor` in the same type as `x`.

Return type Tensor

2.1.9 Pixel-wise softmax

`tensorlayer.activation.pixel_wise_softmax(x, name='pixel_wise_softmax')`

Return the softmax outputs of images, every pixels have multiple label, the sum of a pixel is 1.

> **Warning: THIS FUNCTION IS DEPRECATED:** It will be removed after after 2018-06-30. *Instructions for updating:* This API will be deprecated soon as tf.nn.softmax can do the same thing.

Usually be used for image segmentation.

Parameters

- **x** (*Tensor*) –

  input.

  – For 2d image, 4D tensor (batch_size, height, weight, channel), where channel >= 2.

  – For 3d image, 5D tensor (batch_size, depth, height, weight, channel), where channel >= 2.
- **name** (*str*) – function name (optional)

Returns A `Tensor` in the same type as `x`.

Return type Tensor

Examples

```
>>> outputs = pixel_wise_softmax(network.outputs)
>>> dice_loss = 1 - dice_coe(outputs, y_, epsilon=1e-5)
```

References

- tf.reverse

2.1.10 Parametric activation

See `tensorlayer.layers`.

2.2 API - Array Operations

A file containing functions related to array manipulation.

`alphas`(shape, alpha_value[, name])	Creates a tensor with all elements set to *alpha_value*.
`alphas_like`(tensor, alpha_value[, name, . . .])	Creates a tensor with all elements set to *alpha_value*.

2.2.1 Tensorflow Tensor Operations

tl.alphas

`tensorlayer.array_ops.alphas`(*shape*, *alpha_value*, *name=None*)

Creates a tensor with all elements set to *alpha_value*. This operation returns a tensor of type *dtype* with shape *shape* and all elements set to alpha.

Parameters

- **shape** (A list of integers, a tuple of integers, or a 1-D *Tensor* of type *int32*.) – The shape of the desired tensor
- **alpha_value** (*float32*, *float64*, *int8*, *uint8*, *int16*, *uint16*, int32‘, *int64*) – The value used to fill the resulting *Tensor*.
- **name** (*str*) – A name for the operation (optional).

Returns

Return type A *Tensor* with all elements set to alpha.

Examples

```
>>> tl.alphas([2, 3], tf.int32)  # [[alpha, alpha, alpha], [alpha, alpha, alpha]]
```

tl.alphas_like

`tensorlayer.array_ops.`**`alphas_like`**(*tensor*, *alpha_value*, *name=None*, *optimize=True*)

Creates a tensor with all elements set to *alpha_value*. Given a single tensor (*tensor*), this operation returns a tensor of the same type and shape as *tensor* with all elements set to *alpha_value*.

Parameters

- **`tensor`** (*`tf.Tensor`*) – The Tensorflow Tensor that will be used as a template.
- **`alpha_value`** (*float32*, *float64*, *int8*, *uint8*, *int16*, *uint16*, int32', *int64*) – The value used to fill the resulting *Tensor*.
- **`name`** (*`str`*) – A name for the operation (optional).
- **`optimize`** (*`bool`*) – if true, attempt to statically determine the shape of 'tensor' and encode it as a constant.

Returns

Return type A *Tensor* with all elements set to *alpha_value*.

Examples

```
>>> tensor = tf.constant([[1, 2, 3], [4, 5, 6]])
>>> tl.alphas_like(tensor, 0.5)  # [[0.5, 0.5, 0.5], [0.5, 0.5, 0.5]]
```

2.3 API - Cost

To make TensorLayer simple, we minimize the number of cost functions as much as we can. So we encourage you to use TensorFlow's function, , see TensorFlow API.

Note: Please refer to Getting Started for getting specific weights for weight regularization.

`cross_entropy`(output, target[, name])	Softmax cross-entropy operation, returns the TensorFlow expression of cross-entropy for two distributions, it implements softmax internally.
`sigmoid_cross_entropy`(output, target[, name])	Sigmoid cross-entropy operation, see `tf.nn.sigmoid_cross_entropy_with_logits`.
`binary_cross_entropy`(output, target[, . . .])	Binary cross entropy operation.
`mean_squared_error`(output, target[, . . .])	Return the TensorFlow expression of mean-square-error (L2) of two batch of data.
`normalized_mean_square_error`(output, target)	Return the TensorFlow expression of normalized mean-square-error of two distributions.
`absolute_difference_error`(output, target[, . . .])	Return the TensorFlow expression of absolute difference error (L1) of two batch of data.

Continued on next page

Table 3 – continued from previous page

dice_coe(output, target[, loss_type, axis, ...])	Soft dice (Sørensen or Jaccard) coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e.
dice_hard_coe(output, target[, threshold, ...])	Non-differentiable Sørensen–Dice coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e.
iou_coe(output, target[, threshold, axis, ...])	Non-differentiable Intersection over Union (IoU) for comparing the similarity of two batch of data, usually be used for evaluating binary image segmentation.
cross_entropy_seq(logits, target_seqs[, ...])	Returns the expression of cross-entropy of two sequences, implement softmax internally.
cross_entropy_seq_with_mask(logits, ...[, ...])	Returns the expression of cross-entropy of two sequences, implement softmax internally.
cosine_similarity(v1, v2)	Cosine similarity [-1, 1].
li_regularizer(scale[, scope])	Li regularization removes the neurons of previous layer.
lo_regularizer(scale)	Lo regularization removes the neurons of current layer.
maxnorm_regularizer([scale])	Max-norm regularization returns a function that can be used to apply max-norm regularization to weights.
maxnorm_o_regularizer(scale)	Max-norm output regularization removes the neurons of current layer.
maxnorm_i_regularizer(scale)	Max-norm input regularization removes the neurons of previous layer.
huber_loss(output, target[, is_mean, delta, ...])	Huber Loss operation, see https://en.wikipedia.org/wiki/Huber_loss .

2.3.1 Softmax cross entropy

tensorlayer.cost.**cross_entropy**(*output*, *target*, *name=None*)

Softmax cross-entropy operation, returns the TensorFlow expression of cross-entropy for two distributions, it implements softmax internally. See `tf.nn.sparse_softmax_cross_entropy_with_logits`.

Parameters

- **output** (*Tensor*) – A batch of distribution with shape: [batch_size, num of classes].
- **target** (*Tensor*) – A batch of index with shape: [batch_size,].
- **name** (*string*) – Name of this loss.

Examples

```
>>> import tensorlayer as tl
>>> ce = tl.cost.cross_entropy(y_logits, y_target_logits, 'my_loss')
```

References

- About cross-entropy: https://en.wikipedia.org/wiki/Cross_entropy.
- The code is borrowed from: https://en.wikipedia.org/wiki/Cross_entropy.

2.3.2 Sigmoid cross entropy

`tensorlayer.cost.sigmoid_cross_entropy`(*output*, *target*, *name=None*)
Sigmoid cross-entropy operation, see `tf.nn.sigmoid_cross_entropy_with_logits`.

Parameters

- **output** (*Tensor*) – A batch of distribution with shape: [batch_size, num of classes].
- **target** (*Tensor*) – A batch of index with shape: [batch_size,].
- **name** (*string*) – Name of this loss.

2.3.3 Binary cross entropy

`tensorlayer.cost.binary_cross_entropy`(*output*, *target*, *epsilon=1e-08*, *name='bce_loss'*)
Binary cross entropy operation.

Parameters

- **output** (*Tensor*) – Tensor with type of *float32* or *float64*.
- **target** (*Tensor*) – The target distribution, format the same with *output*.
- **epsilon** (*float*) – A small value to avoid output to be zero.
- **name** (*str*) – An optional name to attach to this function.

References

- ericjang-DRAW

2.3.4 Mean squared error (L2)

`tensorlayer.cost.mean_squared_error`(*output*, *target*, *is_mean=False*, *axis=-1*, *name='mean_squared_error'*)
Return the TensorFlow expression of mean-square-error (L2) of two batch of data.

Parameters

- **output** (*Tensor*) – 2D, 3D or 4D tensor i.e. [batch_size, n_feature], [batch_size, height, width] or [batch_size, height, width, channel].
- **target** (*Tensor*) – The target distribution, format the same with *output*.
- **is_mean** (*boolean*) –

  Whether compute the mean or sum for each example.

  – If True, use `tf.reduce_mean` to compute the loss between one target and predict data.

  – If False, use `tf.reduce_sum` (default).
- **axis** (*int or list of int*) – The dimensions to reduce.
- **name** (*str*) – An optional name to attach to this function.

References

- Wiki Mean Squared Error

2.3.5 Normalized mean square error

`tensorlayer.cost.`**`normalized_mean_square_error`**(*output*, *target*, *axis=-1*, *name='normalized_mean_squared_error_loss'*)

Return the TensorFlow expression of normalized mean-square-error of two distributions.

Parameters

- **output** (*Tensor*) – 2D, 3D or 4D tensor i.e. [batch_size, n_feature], [batch_size, height, width] or [batch_size, height, width, channel].
- **target** (*Tensor*) – The target distribution, format the same with *output*.
- **axis** (*int or list of int*) – The dimensions to reduce.
- **name** (*str*) – An optional name to attach to this function.

2.3.6 Absolute difference error (L1)

`tensorlayer.cost.`**`absolute_difference_error`**(*output*, *target*, *is_mean=False*, *axis=-1*, *name='absolute_difference_error_loss'*)

Return the TensorFlow expression of absolute difference error (L1) of two batch of data.

Parameters

- **output** (*Tensor*) – 2D, 3D or 4D tensor i.e. [batch_size, n_feature], [batch_size, height, width] or [batch_size, height, width, channel].
- **target** (*Tensor*) – The target distribution, format the same with *output*.
- **is_mean** (*boolean*) –

  Whether compute the mean or sum for each example.

  - If True, use `tf.reduce_mean` to compute the loss between one target and predict data.
  - If False, use `tf.reduce_sum` (default).
- **axis** (*int or list of int*) – The dimensions to reduce.
- **name** (*str*) – An optional name to attach to this function.

2.3.7 Dice coefficient

`tensorlayer.cost.`**`dice_coe`**(*output*, *target*, *loss_type='jaccard'*, *axis=(1, 2, 3)*, *smooth=1e-05*)

Soft dice (Sørensen or Jaccard) coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e. labels are binary. The coefficient between 0 to 1, 1 means totally match.

Parameters

- **output** (*Tensor*) – A distribution with shape: [batch_size, ...], (any dimensions).
- **target** (*Tensor*) – The target distribution, format the same with *output*.
- **loss_type** (*str*) – `jaccard` or `sorensen`, default is `jaccard`.

- **axis** (*tuple of int*) – All dimensions are reduced, default `[1,2,3]`.
- **smooth** (*float*) –

  This small value will be added to the numerator and denominator.

  - If both output and target are empty, it makes sure dice is 1.
  - If either output or target are empty (all pixels are background), dice = ``smooth/(small_value + smooth)`, then if smooth is very small, dice close to 0 (even the image values lower than the threshold), so in this case, higher smooth can have a higher dice.

Examples

```
>>> import tensorlayer as tl
>>> outputs = tl.act.pixel_wise_softmax(outputs)
>>> dice_loss = 1 - tl.cost.dice_coe(outputs, y_)
```

References

- Wiki-Dice

2.3.8 Hard Dice coefficient

`tensorlayer.cost.`**`dice_hard_coe`**(*output*, *target*, *threshold=0.5*, *axis=(1, 2, 3)*, *smooth=1e-05*)

Non-differentiable Sørensen–Dice coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e. labels are binary. The coefficient between 0 to 1, 1 if totally match.

Parameters

- **output** (*tensor*) – A distribution with shape: [batch_size, ...], (any dimensions).
- **target** (*tensor*) – The target distribution, format the same with *output*.
- **threshold** (*float*) – The threshold value to be true.
- **axis** (*tuple of integer*) – All dimensions are reduced, default `(1,2,3)`.
- **smooth** (*float*) – This small value will be added to the numerator and denominator, see `dice_coe`.

References

- Wiki-Dice

2.3.9 IOU coefficient

`tensorlayer.cost.`**`iou_coe`**(*output*, *target*, *threshold=0.5*, *axis=(1, 2, 3)*, *smooth=1e-05*)

Non-differentiable Intersection over Union (IoU) for comparing the similarity of two batch of data, usually be used for evaluating binary image segmentation. The coefficient between 0 to 1, and 1 means totally match.

Parameters

- **output** (*tensor*) – A batch of distribution with shape: [batch_size, ...], (any dimensions).

- **target** (*tensor*) – The target distribution, format the same with *output*.
- **threshold** (*float*) – The threshold value to be true.
- **axis** (*tuple of integer*) – All dimensions are reduced, default (1,2,3).
- **smooth** (*float*) – This small value will be added to the numerator and denominator, see dice_coe.

Notes

- IoU cannot be used as training loss, people usually use dice coefficient for training, IoU and hard-dice for evaluating.

2.3.10 Cross entropy for sequence

tensorlayer.cost.**cross_entropy_seq**(*logits*, *target_seqs*, *batch_size=None*)

Returns the expression of cross-entropy of two sequences, implement softmax internally. Normally be used for fixed length RNN outputs, see PTB example.

Parameters

- **logits** (*Tensor*) – 2D tensor with shape of *[batch_size * n_steps, n_classes]*.
- **target_seqs** (*Tensor*) – The target sequence, 2D tensor *[batch_size, n_steps]*, if the number of step is dynamic, please use tl.cost.cross_entropy_seq_with_mask instead.
- **batch_size** (*None or int.*) –

  Whether to divide the cost by batch size.

  – If integer, the return cost will be divided by *batch_size*.

  – If None (default), the return cost will not be divided by anything.

Examples

```
>>> import tensorlayer as tl
>>> # see `PTB example <https://github.com/tensorlayer/tensorlayer/blob/master/
↪example/tutorial_ptb_lstm.py>`__.for more details
>>> # outputs shape : (batch_size * n_steps, n_classes)
>>> # targets shape : (batch_size, n_steps)
>>> cost = tl.cost.cross_entropy_seq(outputs, targets)
```

2.3.11 Cross entropy with mask for sequence

tensorlayer.cost.**cross_entropy_seq_with_mask**(*logits*, *target_seqs*, *input_mask*, *return_details=False*, *name=None*)

Returns the expression of cross-entropy of two sequences, implement softmax internally. Normally be used for Dynamic RNN with Synced sequence input and output.

Parameters

- **logits** (*Tensor*) – 2D tensor with shape of [batch_size * ?, n_classes], *?* means dynamic IDs for each example. - Can be get from *DynamicRNNLayer* by setting return_seq_2d to *True*.

- **target_seqs** (*Tensor*) – int of tensor, like word ID. [batch_size, ?], *?* means dynamic IDs for each example.
- **input_mask** (*Tensor*) – The mask to compute loss, it has the same size with *target_seqs*, normally 0 or 1.
- **return_details** (*boolean*) –

  Whether to return detailed losses.

  – If False (default), only returns the loss.

  – If True, returns the loss, losses, weights and targets (see source code).

Examples

```
>>> import tensorlayer as tl
>>> import tensorflow as tf
>>> import numpy as np
>>> batch_size = 64
>>> vocab_size = 10000
>>> embedding_size = 256
>>> ni = tl.layers.Input([batch_size, None], dtype=tf.int64)
>>> net = tl.layers.Embedding(
...         vocabulary_size = vocab_size,
...         embedding_size = embedding_size,
...         name = 'seq_embedding')(ni)
>>> net = tl.layers.RNN(
...         cell =tf.keras.layers.LSTMCell(units=embedding_size, dropout=0.1),
...         return_seq_2d = True,
...         name = 'dynamicrnn')(net)
>>> net = tl.layers.Dense(n_units=vocab_size, name="output")(net)
>>> model = tl.models.Model(inputs=ni, outputs=net)
>>> input_seqs = np.random.randint(0, 10, size=(batch_size, 10), dtype=np.int64)
>>> target_seqs = np.random.randint(0, 10, size=(batch_size, 10), dtype=np.int64)
>>> input_mask = np.random.randint(0, 2, size=(batch_size, 10), dtype=np.int64)
>>> outputs = model(input_seqs, is_train=True)
>>> loss = tl.cost.cross_entropy_seq_with_mask(outputs, target_seqs, input_mask)
```

2.3.12 Cosine similarity

`tensorlayer.cost.`**`cosine_similarity`**(*v1*, *v2*)

Cosine similarity [-1, 1].

Parameters **v2** (*v1,*) – Tensor with the same shape [batch_size, n_feature].

References

- Wiki.

2.3.13 Regularization functions

For `tf.nn.l2_loss`, `tf.contrib.layers.l1_regularizer`, `tf.contrib.layers.l2_regularizer` and `tf.contrib.layers.sum_regularizer`, see tensorflow API. Maxnorm ^^^^^^^^^^ .. autofunction:: maxnorm_regularizer

Special

tensorlayer.cost.**li_regularizer**(*scale*, *scope=None*)

Li regularization removes the neurons of previous layer. The *i* represents *inputs*. Returns a function that can be used to apply group li regularization to weights. The implementation follows TensorFlow contrib.

Parameters

- **scale** (*float*) – A scalar multiplier *Tensor*. 0.0 disables the regularizer.
- **scope** (*str*) – An optional scope name for this function.

Returns

Return type A function with signature *li(weights, name=None)* that apply Li regularization.

:raises ValueError : if scale is outside of the range [0.0, 1.0] or if scale is not a float.:

tensorlayer.cost.**lo_regularizer**(*scale*)

Lo regularization removes the neurons of current layer. The *o* represents *outputs* Returns a function that can be used to apply group lo regularization to weights. The implementation follows TensorFlow contrib.

Parameters **scale** (*float*) – A scalar multiplier *Tensor*. 0.0 disables the regularizer.

Returns

Return type A function with signature *lo(weights, name=None)* that apply Lo regularization.

:raises ValueError : If scale is outside of the range [0.0, 1.0] or if scale is not a float.:

tensorlayer.cost.**maxnorm_o_regularizer**(*scale*)

Max-norm output regularization removes the neurons of current layer. Returns a function that can be used to apply max-norm regularization to each column of weight matrix. The implementation follows TensorFlow contrib.

Parameters **scale** (*float*) – A scalar multiplier *Tensor*. 0.0 disables the regularizer.

Returns

Return type A function with signature *mn_o(weights, name=None)* that apply Lo regularization.

:raises ValueError : If scale is outside of the range [0.0, 1.0] or if scale is not a float.:

tensorlayer.cost.**maxnorm_i_regularizer**(*scale*)

Max-norm input regularization removes the neurons of previous layer. Returns a function that can be used to apply max-norm regularization to each row of weight matrix. The implementation follows TensorFlow contrib.

Parameters **scale** (*float*) – A scalar multiplier *Tensor*. 0.0 disables the regularizer.

Returns

Return type A function with signature *mn_i(weights, name=None)* that apply Lo regularization.

:raises ValueError : If scale is outside of the range [0.0, 1.0] or if scale is not a float.:

Huber Loss

tensorlayer.cost.**huber_loss**(*output*, *target*, *is_mean=True*, *delta=1.0*, *dynamichuber=False*, *reverse=False*, *axis=-1*, *epsilon=1e-05*, *name=None*)

Huber Loss operation, see `https://en.wikipedia.org/wiki/Huber_loss` . Reverse Huber Loss operation, see ''https://statweb.stanford.edu/~owen/reports/hhu.pdf''. Dynamic Reverse Huber Loss operation, see ''https://arxiv.org/pdf/1606.00373.pdf''.

Parameters

- **output** (*Tensor*) – A distribution with shape: [batch_size, ...], (any dimensions).
- **target** (*Tensor*) – The target distribution, format the same with *output*.
- **is_mean** (*boolean*) – Whether compute the mean or sum for each example. - If True, use `tf.reduce_mean` to compute the loss between one target and predict data (default). - If False, use `tf.reduce_sum`.
- **delta** (*float*) – The point where the huber loss function changes from a quadratic to linear.
- **dynamichuber** (*boolean*) – Whether compute the coefficient c for each batch. - If True, c is 20% of the maximal per-batch error. - If False, c is delta.
- **reverse** (*boolean*) – Whether compute the reverse huber loss.
- **axis** (*int or list of int*) – The dimensions to reduce.
- **epsilon** – Eplison.
- **name** (*string*) – Name of this loss.

2.4 API - Data Pre-Processing

`affine_rotation_matrix`([angle])	Create an affine transform matrix for image rotation.
`affine_horizontal_flip_matrix`([prob])	Create an affine transformation matrix for image horizontal flipping.
`affine_vertical_flip_matrix`([prob])	Create an affine transformation for image vertical flipping.
`affine_shift_matrix`([wrg, hrg, w, h])	Create an affine transform matrix for image shifting.
`affine_shear_matrix`([x_shear, y_shear])	Create affine transform matrix for image shearing.
`affine_zoom_matrix`([zoom_range])	Create an affine transform matrix for zooming/scaling an image's height and width.
`affine_respective_zoom_matrix`([w_range, h_range])	Get affine transform matrix for zooming/scaling that height and width are changed independently.
`transform_matrix_offset_center`(matrix, y, x)	Convert the matrix from Cartesian coordinates (the origin in the middle of image) to Image coordinates (the origin on the top-left of image).
`affine_transform`(x, transform_matrix[, ...])	Return transformed images by given an affine matrix in Scipy format (x is height).
`affine_transform_cv2`(x, transform_matrix[, ...])	Return transformed images by given an affine matrix in OpenCV format (x is width).
`affine_transform_keypoints`(coords_list, ...)	Transform keypoint coordinates according to a given affine transform matrix.
`projective_transform_by_points`(x, src, dst)	Projective transform by given coordinates, usually 4 coordinates.
`rotation`(x[, rg, is_random, row_index, ...])	Rotate an image randomly or non-randomly.
`rotation_multi`(x[, rg, is_random, ...])	Rotate multiple images with the same arguments, randomly or non-randomly.
`crop`(x, wrg, hrg[, is_random, row_index, ...])	Randomly or centrally crop an image.
`crop_multi`(x, wrg, hrg[, is_random, ...])	Randomly or centrally crop multiple images.
`flip_axis`(x[, axis, is_random])	Flip the axis of an image, such as flip left and right, up and down, randomly or non-randomly,

Continued on next page

Table 4 – continued from previous page

flip_axis_multi(x, axis[, is_random])	Flip the axises of multiple images together, such as flip left and right, up and down, randomly or non-randomly,
shift(x[, wrg, hrg, is_random, row_index, . . .])	Shift an image randomly or non-randomly.
shift_multi(x[, wrg, hrg, is_random, . . .])	Shift images with the same arguments, randomly or non-randomly.
shear(x[, intensity, is_random, row_index, . . .])	Shear an image randomly or non-randomly.
shear_multi(x[, intensity, is_random, . . .])	Shear images with the same arguments, randomly or non-randomly.
shear2(x[, shear, is_random, row_index, . . .])	Shear an image randomly or non-randomly.
shear_multi2(x[, shear, is_random, . . .])	Shear images with the same arguments, randomly or non-randomly.
swirl(x[, center, strength, radius, . . .])	Swirl an image randomly or non-randomly, see scikit-image swirl API and example.
swirl_multi(x[, center, strength, radius, . . .])	Swirl multiple images with the same arguments, randomly or non-randomly.
elastic_transform(x, alpha, sigma[, mode, . . .])	Elastic transformation for image as described in [Simard2003].
elastic_transform_multi(x, alpha, sigma[, . . .])	Elastic transformation for images as described in [Simard2003].
zoom(x[, zoom_range, flags, border_mode])	Zooming/Scaling a single image that height and width are changed together.
respective_zoom(x[, h_range, w_range, . . .])	Zooming/Scaling a single image that height and width are changed independently.
zoom_multi(x[, zoom_range, flags, border_mode])	Zoom in and out of images with the same arguments, randomly or non-randomly.
brightness(x[, gamma, gain, is_random])	Change the brightness of a single image, randomly or non-randomly.
brightness_multi(x[, gamma, gain, is_random])	Change the brightness of multiply images, randomly or non-randomly.
illumination(x[, gamma, contrast, . . .])	Perform illumination augmentation for a single image, randomly or non-randomly.
rgb_to_hsv(rgb)	Input RGB image [0~255] return HSV image [0~1].
hsv_to_rgb(hsv)	Input HSV image [0~1] return RGB image [0~255].
adjust_hue(im[, hout, is_offset, is_clip, . . .])	Adjust hue of an RGB image.
imresize(x[, size, interp, mode])	Resize an image by given output size and method.
pixel_value_scale(im[, val, clip, is_random])	Scales each value in the pixels of the image.
samplewise_norm(x[, rescale, . . .])	Normalize an image by rescale, samplewise centering and samplewise centering in order.
featurewise_norm(x[, mean, std, epsilon])	Normalize every pixels by the same given mean and std, which are usually compute from all examples.
channel_shift(x, intensity[, is_random, . . .])	Shift the channels of an image, randomly or non-randomly, see numpy.rollaxis.
channel_shift_multi(x, intensity[, . . .])	Shift the channels of images with the same arguments, randomly or non-randomly, see numpy.rollaxis.
drop(x[, keep])	Randomly set some pixels to zero by a given keeping probability.
array_to_img(x[, dim_ordering, scale])	Converts a numpy array to PIL image object (uint8 format).
find_contours(x[, level, fully_connected, . . .])	Find iso-valued contours in a 2D array for a given level value, returns list of (n, 2)-ndarrays see skimage.measure.find_contours.

Continued on next page

Table 4 – continued from previous page

pt2map([list_points, size, val])	Inputs a list of points, return a 2D image.
binary_dilation(x[, radius])	Return fast binary morphological dilation of an image.
dilation(x[, radius])	Return greyscale morphological dilation of an image, see skimage.morphology.dilation.
binary_erosion(x[, radius])	Return binary morphological erosion of an image, see skimage.morphology.binary_erosion.
erosion(x[, radius])	Return greyscale morphological erosion of an image, see skimage.morphology.erosion.
obj_box_coord_rescale([coord, shape])	Scale down one coordinates from pixel unit to the ratio of image size i.e.
obj_box_coords_rescale([coords, shape])	Scale down a list of coordinates from pixel unit to the ratio of image size i.e.
obj_box_coord_scale_to_pixelunit(coord[, shape])	Convert one coordinate [x, y, w (or x2), h (or y2)] in ratio format to image coordinate format.
obj_box_coord_centroid_to_upleft_butright(coord)	Convert one coordinate [x_center, y_center, w, h] to [x1, y1, x2, y2] in up-left and botton-right format.
obj_box_coord_upleft_butright_to_centroid(coord)	Convert one coordinate [x1, y1, x2, y2] to [x_center, y_center, w, h].
obj_box_coord_centroid_to_upleft(coord)	Convert one coordinate [x_center, y_center, w, h] to [x, y, w, h].
obj_box_coord_upleft_to_centroid(coord)	Convert one coordinate [x, y, w, h] to [x_center, y_center, w, h].
parse_darknet_ann_str_to_list(annotations)	Input string format of class, x, y, w, h, return list of list format.
parse_darknet_ann_list_to_cls_box(annotations)	Parse darknet annotation format into two lists for class and bounding box.
obj_box_horizontal_flip(im[, coords, …])	Left-right flip the image and coordinates for object detection.
obj_box_imresize(im[, coords, size, interp, …])	Resize an image, and compute the new bounding box coordinates.
obj_box_crop(im[, classes, coords, wrg, …])	Randomly or centrally crop an image, and compute the new bounding box coordinates.
obj_box_shift(im[, classes, coords, wrg, …])	Shift an image randomly or non-randomly, and compute the new bounding box coordinates.
obj_box_zoom(im[, classes, coords, …])	Zoom in and out of a single image, randomly or non-randomly, and compute the new bounding box coordinates.
keypoint_random_crop(image, annos[, mask, size])	Randomly crop an image and corresponding keypoints without influence scales, given by keypoint_random_resize_shortestedge.
keypoint_resize_random_crop(image, annos[, …])	Reszie the image to make either its width or height equals to the given sizes.
keypoint_random_rotate(image, annos[, mask, rg])	Rotate an image and corresponding keypoints.
keypoint_random_flip(image, annos[, mask, …])	Flip an image and corresponding keypoints.
keypoint_random_resize(image, annos[, mask, …])	Randomly resize an image and corresponding keypoints.
keypoint_random_resize_shortestedge(image, annos)	Randomly resize an image and corresponding keypoints based on shorter edgeself.

Continued on next page

Table 4 – continued from previous page

pad_sequences(sequences[, maxlen, dtype, ...])	Pads each sequence to the same length: the length of the longest sequence.
remove_pad_sequences(sequences[, pad_id])	Remove padding.
process_sequences(sequences[, end_id, ...])	Set all tokens(ids) after END token to the padding value, and then shorten (option) it to the maximum sequence length in this batch.
sequences_add_start_id(sequences[, ...])	Add special start token(id) in the beginning of each sequence.
sequences_add_end_id(sequences[, end_id])	Add special end token(id) in the end of each sequence.
sequences_add_end_id_after_pad(sequences[, ...])	Add special end token(id) in the end of each sequence.
sequences_get_mask(sequences[, pad_val])	Return mask for sequences.

2.4.1 Affine Transform

Python can be FAST

Image augmentation is a critical step in deep learning. Though TensorFlow has provided `tf.image`, image augmentation often remains as a key bottleneck. `tf.image` has three limitations:

- Real-world visual tasks such as object detection, segmentation, and pose estimation must cope with image meta-data (e.g., coordinates). These data are beyond `tf.image` which processes images as tensors.
- `tf.image` operators breaks the pure Python programing experience (i.e., users have to use `tf.py_func` in order to call image functions written in Python); however, frequent uses of `tf.py_func` slow down TensorFlow, making users hard to balance flexibility and performance.
- `tf.image` API is inflexible. Image operations are performed in an order. They are hard to jointly optimize. More importantly, sequential image operations can significantly reduces the quality of images, thus affecting training accuracy.

TensorLayer addresses these limitations by providing a high-performance image augmentation API in Python. This API bases on affine transformation and `cv2.wrapAffine`. It allows you to combine multiple image processing functions into a single matrix operation. This combined operation is executed by the fast `cv2` library, offering 78x performance improvement (observed in openpose-plus for example). The following example illustrates the rationale behind this tremendous speed up.

Example

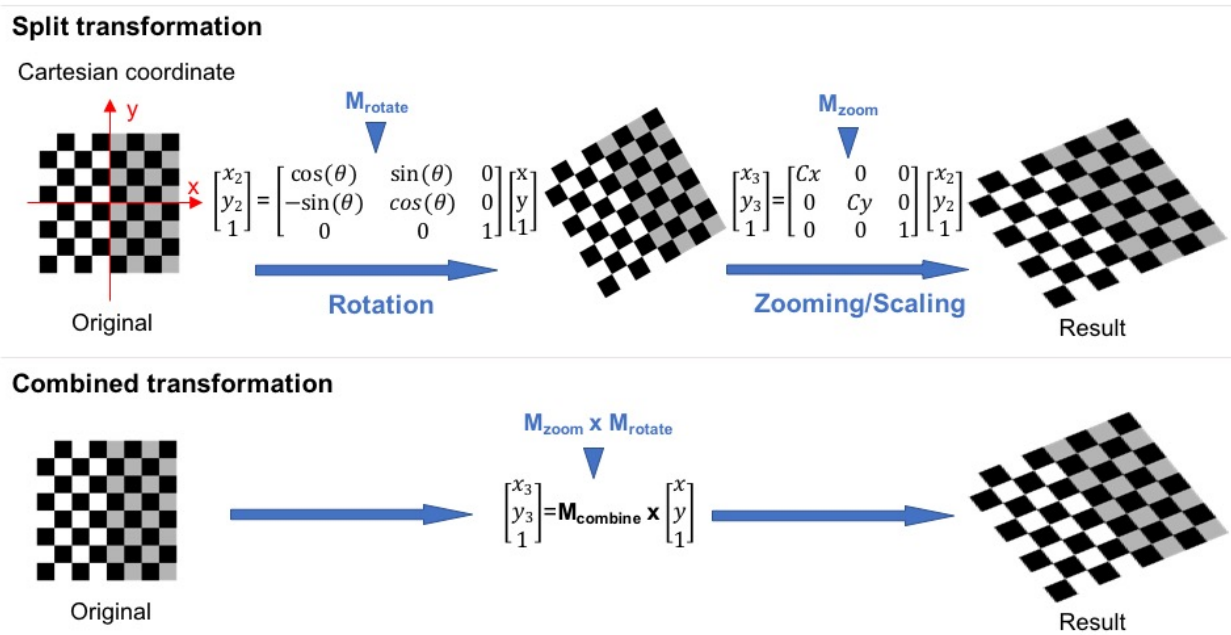
The source code of complete examples can be found here. The following is a typical Python program that applies rotation, shifting, flipping, zooming and shearing to an image,

```
image = tl.vis.read_image('tiger.jpeg')

xx = tl.prepro.rotation(image, rg=-20, is_random=False)
xx = tl.prepro.flip_axis(xx, axis=1, is_random=False)
xx = tl.prepro.shear2(xx, shear=(0., -0.2), is_random=False)
xx = tl.prepro.zoom(xx, zoom_range=0.8)
xx = tl.prepro.shift(xx, wrg=-0.1, hrg=0, is_random=False)

tl.vis.save_image(xx, '_result_slow.png')
```

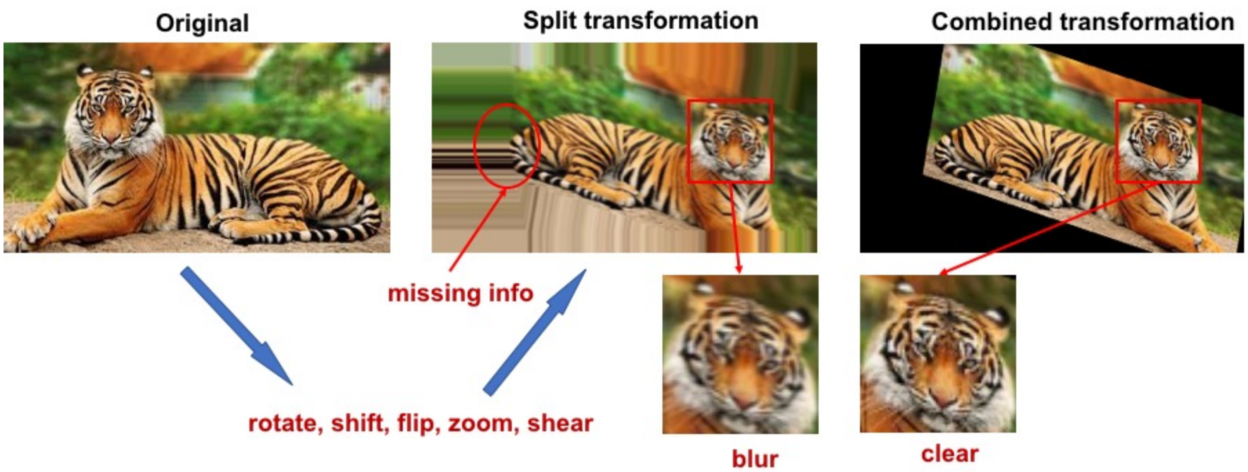
However, by leveraging affine transformation, image operations can be combined into one:

```
# 1. Create required affine transformation matrices
M_rotate = tl.prepro.affine_rotation_matrix(angle=20)
M_flip = tl.prepro.affine_horizontal_flip_matrix(prob=1)
M_shift = tl.prepro.affine_shift_matrix(wrg=0.1, hrg=0, h=h, w=w)
M_shear = tl.prepro.affine_shear_matrix(x_shear=0.2, y_shear=0)
M_zoom = tl.prepro.affine_zoom_matrix(zoom_range=0.8)

# 2. Combine matrices
# NOTE: operations are applied in a reversed order (i.e., rotation is performed first)
M_combined = M_shift.dot(M_zoom).dot(M_shear).dot(M_flip).dot(M_rotate)

# 3. Convert the matrix from Cartesian coordinates (the origin in the middle of image)
# to image coordinates (the origin on the top-left of image)
transform_matrix = tl.prepro.transform_matrix_offset_center(M_combined, x=w, y=h)

# 4. Transform the image using a single operation
result = tl.prepro.affine_transform_cv2(image, transform_matrix)  # 76 times faster

tl.vis.save_image(result, '_result_fast.png')
```

The following figure illustrates the rational behind combined affine transformation.



Using combined affine transformation has two key benefits. First, it allows you to leverage a pure Python API to achieve orders of magnitudes of speed up in image augmentation, and thus prevent data pre-processing from becoming a bottleneck in training. Second, performing sequential image transformation requires multiple image interpolations. This produces low-quality input images. In contrast, a combined transformation performs the interpolation only once, and thus preserve the content in an image. The following figure illustrates these two benefits:



The major reason for combined affine transformation being fast is because it has lower computational complexity. Assume we have `k` affine transformations `T1, ..., Tk`, where `Ti` can be represented by 3x3 matrixes. The sequential transformation can be represented as `y = Tk (... T1(x))`, and the time complexity is `O(k N)` where `N` is the cost of applying one transformation to image `x`. `N` is linear to the size of `x`. For the combined transformation `y = (Tk ... T1) (x)` the time complexity is `O(27(k - 1) + N) = max{O(27k), O(N)} = O(N)` (assuming 27k << N) where 27 = 3^3 is the cost for combining two transformations.

Get rotation matrix

`tensorlayer.prepro.`**`affine_rotation_matrix`**(*angle=(-20, 20)*)

Create an affine transform matrix for image rotation. NOTE: In OpenCV, x is width and y is height.

Parameters **angle** (*int/float or tuple of two int/float*) –

Degree to rotate, usually -180 ~ 180.

- int/float, a fixed angle.
- tuple of 2 floats/ints, randomly sample a value as the angle between these 2 values.

Returns An affine transform matrix.

Return type numpy.array

Get horizontal flipping matrix

`tensorlayer.prepro.`**`affine_horizontal_flip_matrix`**(*prob=0.5*)

Create an affine transformation matrix for image horizontal flipping. NOTE: In OpenCV, x is width and y is height.

Parameters **prob** (*float*) – Probability to flip the image. 1.0 means always flip.

Returns An affine transform matrix.

Return type numpy.array

Get vertical flipping matrix

`tensorlayer.prepro.`**`affine_vertical_flip_matrix`**(*prob=0.5*)

Create an affine transformation for image vertical flipping. NOTE: In OpenCV, x is width and y is height.

Parameters **prob** (*float*) – Probability to flip the image. 1.0 means always flip.

Returns An affine transform matrix.

Return type numpy.array

Get shifting matrix

`tensorlayer.prepro.`**`affine_shift_matrix`**(*wrg=(-0.1, 0.1)*, *hrg=(-0.1, 0.1)*, *w=200*, *h=200*)

Create an affine transform matrix for image shifting. NOTE: In OpenCV, x is width and y is height.

Parameters

- **wrg** (*float or tuple of floats*) –

  Range to shift on width axis, -1 ~ 1.

  - float, a fixed distance.
  - tuple of 2 floats, randomly sample a value as the distance between these 2 values.

- **hrg** (*float or tuple of floats*) –

  Range to shift on height axis, -1 ~ 1.

  - float, a fixed distance.
  - tuple of 2 floats, randomly sample a value as the distance between these 2 values.

- **h** (*w,*) – The width and height of the image.

Returns An affine transform matrix.

Return type numpy.array

Get shearing matrix

`tensorlayer.prepro.`**`affine_shear_matrix`**(*x_shear=(-0.1, 0.1)*, *y_shear=(-0.1, 0.1)*)

Create affine transform matrix for image shearing. NOTE: In OpenCV, x is width and y is height.

Parameters **shear** (*tuple of two floats*) – Percentage of shears for width and height directions.

Returns An affine transform matrix.

Return type numpy.array

Get zooming matrix

`tensorlayer.prepro.`**`affine_zoom_matrix`**(*zoom_range=(0.8, 1.1)*)

Create an affine transform matrix for zooming/scaling an image's height and width. OpenCV format, x is width.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **zoom_range** (*float or tuple of 2 floats*) –

  The zooming/scaling ratio, greater than 1 means larger.

  - float, a fixed ratio.
  - tuple of 2 floats, randomly sample a value as the ratio between these 2 values.

Returns An affine transform matrix.

Return type numpy.array

Get respective zooming matrix

`tensorlayer.prepro.`**`affine_respective_zoom_matrix`**(*w_range=0.8*, *h_range=1.1*)

Get affine transform matrix for zooming/scaling that height and width are changed independently. OpenCV format, x is width.

Parameters

- **w_range** (*float or tuple of 2 floats*) –

  The zooming/scaling ratio of width, greater than 1 means larger.

  - float, a fixed ratio.
  - tuple of 2 floats, randomly sample a value as the ratio between 2 values.

- **h_range** (*float or tuple of 2 floats*) –

  The zooming/scaling ratio of height, greater than 1 means larger.

  - float, a fixed ratio.
  - tuple of 2 floats, randomly sample a value as the ratio between 2 values.

Returns An affine transform matrix.

Return type numpy.array

Cartesian to image coordinates

`tensorlayer.prepro.`**`transform_matrix_offset_center`**(*matrix*, *y*, *x*)

Convert the matrix from Cartesian coordinates (the origin in the middle of image) to Image coordinates (the origin on the top-left of image).

Parameters

- **matrix** (*numpy.array*) – Transform matrix.
- **and y** (*x*) – Size of image.

Returns The transform matrix.

Return type numpy.array

Examples

- See `tl.prepro.rotation`, `tl.prepro.shear`, `tl.prepro.zoom`.

Apply image transform

`tensorlayer.prepro.`**`affine_transform_cv2`**(*x*, *transform_matrix*, *flags=None*, *border_mode='constant'*)

Return transformed images by given an affine matrix in OpenCV format (x is width). (Powered by OpenCV2, faster than `tl.prepro.affine_transform`)

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **transform_matrix** (*numpy.array*) – A transform matrix, OpenCV format.
- **border_mode** (*str*) –
  - *constant*, pad the image with a constant value (i.e. black or 0)
  - *replicate*, the row or column at the very edge of the original is replicated to the extra border.

Examples

```
>>> M_shear = tl.prepro.affine_shear_matrix(intensity=0.2, is_random=False)
>>> M_zoom = tl.prepro.affine_zoom_matrix(zoom_range=0.8)
>>> M_combined = M_shear.dot(M_zoom)
>>> result = tl.prepro.affine_transform_cv2(image, M_combined)
```

Apply keypoint transform

tensorlayer.prepro.**affine_transform_keypoints**(*coords_list*, *transform_matrix*)

Transform keypoint coordinates according to a given affine transform matrix. OpenCV format, x is width.

Note that, for pose estimation task, flipping requires maintaining the left and right body information. We should not flip the left and right body, so please use `tl.prepro.keypoint_random_flip`.

Parameters

- **coords_list** (*list of list of tuple/list*) – The coordinates e.g., the keypoint coordinates of every person in an image.
- **transform_matrix** (*numpy.array*) – Transform matrix, OpenCV format.

Examples

```
>>> # 1. get all affine transform matrices
>>> M_rotate = tl.prepro.affine_rotation_matrix(angle=20)
>>> M_flip = tl.prepro.affine_horizontal_flip_matrix(prob=1)
>>> # 2. combine all affine transform matrices to one matrix
>>> M_combined = dot(M_flip).dot(M_rotate)
>>> # 3. transfrom the matrix from Cartesian coordinate (the origin in the middle␣
↪of image)
>>> # to Image coordinate (the origin on the top-left of image)
>>> transform_matrix = tl.prepro.transform_matrix_offset_center(M_combined, x=w,␣
↪y=h)
>>> # 4. then we can transfrom the image once for all transformations
>>> result = tl.prepro.affine_transform_cv2(image, transform_matrix)  # 76 times␣
↪faster
>>> # 5. transform keypoint coordinates
>>> coords = [[(50, 100), (100, 100), (100, 50), (200, 200)], [(250, 50), (200,␣
↪50), (200, 100)]]
>>> coords_result = tl.prepro.affine_transform_keypoints(coords, transform_matrix)
```

2.4.2 Images

Projective transform by points

`tensorlayer.prepro.projective_transform_by_points(x, src, dst, map_args=None, output_shape=None, order=1, mode='constant', cval=0.0, clip=True, preserve_range=False)`

Projective transform by given coordinates, usually 4 coordinates.

see scikit-image.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **src** (*list or numpy*) – The original coordinates, usually 4 coordinates of (width, height).
- **dst** (*list or numpy*) – The coordinates after transformation, the number of coordinates is the same with src.
- **map_args** (*dictionary or None*) – Keyword arguments passed to inverse map.
- **output_shape** (*tuple of 2 int*) – Shape of the output image generated. By default the shape of the input image is preserved. Note that, even for multi-band images, only rows and columns need to be specified.
- **order** (*int*) –

  The order of interpolation. The order has to be in the range 0-5:

  - 0 Nearest-neighbor
  - 1 Bi-linear (default)
  - 2 Bi-quadratic
  - 3 Bi-cubic
  - 4 Bi-quartic
  - 5 Bi-quintic
- **mode** (*str*) – One of *constant* (default), *edge*, *symmetric*, *reflect* or *wrap*. Points outside the boundaries of the input are filled according to the given mode. Modes match the behaviour of numpy.pad.
- **cval** (*float*) – Used in conjunction with mode *constant*, the value outside the image boundaries.
- **clip** (*boolean*) – Whether to clip the output to the range of values of the input image. This is enabled by default, since higher order interpolation may produce values outside the given input range.
- **preserve_range** (*boolean*) – Whether to keep the original range of values. Otherwise, the input image is converted according to the conventions of img_as_float.

Returns A processed image.

Return type numpy.array

Examples

Assume X is an image from CIFAR-10, i.e. shape == (32, 32, 3)

```
>>> src = [[0,0],[0,32],[32,0],[32,32]]     # [w, h]
>>> dst = [[10,10],[0,32],[32,0],[32,32]]
>>> x = tl.prepro.projective_transform_by_points(X, src, dst)
```

References

- scikit-image : geometric transformations
- scikit-image : examples

Rotation

`tensorlayer.prepro.rotation`(*x*, *rg=20*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Rotate an image randomly or non-randomly.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **rg** (*int or float*) – Degree to rotate, usually 0 ~ 180.
- **is_random** (*boolean*) – If True, randomly rotate. Default is False
- **col_index and channel_index** (*row_index*) – Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).
- **fill_mode** (*str*) – Method to fill missing pixel, default *nearest*, more options *constant*, *reflect* or *wrap*, see scipy ndimage affine_transform
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0
- **order** (*int*) – The order of interpolation. The order has to be in the range 0-5. See `tl.prepro.affine_transform` and scipy ndimage affine_transform

Returns A processed image.

Return type numpy.array

Examples

```
>>> x --> [row, col, 1]
>>> x = tl.prepro.rotation(x, rg=40, is_random=False)
>>> tl.vis.save_image(x, 'im.png')
```

`tensorlayer.prepro.rotation_multi`(*x*, *rg=20*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Rotate multiple images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).

- **others** (*args*) – See `tl.prepro.rotation`.

Returns A list of processed images.

Return type numpy.array

Examples

```
>>> x, y --> [row, col, 1]  greyscale
>>> x, y = tl.prepro.rotation_multi([x, y], rg=90, is_random=False)
```

Crop

`tensorlayer.prepro.`**`crop`**(*x*, *wrg*, *hrg*, *is_random=False*, *row_index=0*, *col_index=1*)

Randomly or centrally crop an image.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **wrg** (*int*) – Size of width.
- **hrg** (*int*) – Size of height.
- **is_random** (*boolean,*) – If True, randomly crop, else central crop. Default is False.
- **row_index** (*int*) – index of row.
- **col_index** (*int*) – index of column.

Returns A processed image.

Return type numpy.array

`tensorlayer.prepro.`**`crop_multi`**(*x*, *wrg*, *hrg*, *is_random=False*, *row_index=0*, *col_index=1*)

Randomly or centrally crop multiple images.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).
- **others** (*args*) – See `tl.prepro.crop`.

Returns A list of processed images.

Return type numpy.array

Flip

`tensorlayer.prepro.`**`flip_axis`**(*x*, *axis=1*, *is_random=False*)

Flip the axis of an image, such as flip left and right, up and down, randomly or non-randomly,

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **axis** (*int*) –

  Which axis to flip.

  – 0, flip up and down

- 1, flip left and right
- 2, flip channel

- **is_random** (*boolean*) – If True, randomly flip. Default is False.

Returns A processed image.

Return type numpy.array

`tensorlayer.prepro.`**`flip_axis_multi`**(*x*, *axis*, *is_random=False*)

Flip the axises of multiple images together, such as flip left and right, up and down, randomly or non-randomly,

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).
- **others** (*args*) – See `tl.prepro.flip_axis`.

Returns A list of processed images.

Return type numpy.array

Shift

`tensorlayer.prepro.`**`shift`**(*x*, *wrg=0.1*, *hrg=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shift an image randomly or non-randomly.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **wrg** (*float*) – Percentage of shift in axis x, usually -0.25 ~ 0.25.
- **hrg** (*float*) – Percentage of shift in axis y, usually -0.25 ~ 0.25.
- **is_random** (*boolean*) – If True, randomly shift. Default is False.
- **col_index and channel_index** (*row_index*) – Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).
- **fill_mode** (*str*) – Method to fill missing pixel, default *nearest*, more options *constant*, *reflect* or *wrap*, see scipy ndimage affine_transform
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.
- **order** (*int*) – The order of interpolation. The order has to be in the range 0-5. See `tl.prepro.affine_transform` and scipy ndimage affine_transform

Returns A processed image.

Return type numpy.array

`tensorlayer.prepro.`**`shift_multi`**(*x*, *wrg=0.1*, *hrg=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shift images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).

- **others** (*args*) – See `tl.prepro.shift`.

Returns A list of processed images.

Return type numpy.array

Shear

`tensorlayer.prepro.`**shear**(*x*, *intensity=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear an image randomly or non-randomly.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **intensity** (*float*) – Percentage of shear, usually -0.5 ~ 0.5 (is_random==True), 0 ~ 0.5 (is_random==False), you can have a quick try by shear(X, 1).
- **is_random** (*boolean*) – If True, randomly shear. Default is False.
- **col_index and channel_index** (*row_index*) – Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).
- **fill_mode** (*str*) – Method to fill missing pixel, default *nearest*, more options *constant*, *reflect* or *wrap*, see and scipy ndimage affine_transform
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.
- **order** (*int*) – The order of interpolation. The order has to be in the range 0-5. See `tl.prepro.affine_transform` and scipy ndimage affine_transform

Returns A processed image.

Return type numpy.array

References

- Affine transformation

`tensorlayer.prepro.`**shear_multi**(*x*, *intensity=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).
- **others** (*args*) – See `tl.prepro.shear`.

Returns A list of processed images.

Return type numpy.array

Shear V2

`tensorlayer.prepro.`**`shear2`**(*x*, *shear=(0.1, 0.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear an image randomly or non-randomly.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **shear** (*tuple of two floats*) – Percentage of shear for height and width direction (0, 1).
- **is_random** (*boolean*) – If True, randomly shear. Default is False.
- **col_index and channel_index** (*row_index*) – Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).
- **fill_mode** (*str*) – Method to fill missing pixel, default *nearest*, more options *constant*, *reflect* or *wrap*, see scipy ndimage affine_transform
- **cval** (*float*) – Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.
- **order** (*int*) – The order of interpolation. The order has to be in the range 0-5. See `tl.prepro.affine_transform` and scipy ndimage affine_transform

Returns A processed image.

Return type numpy.array

References

- Affine transformation

`tensorlayer.prepro.`**`shear_multi2`**(*x*, *shear=(0.1, 0.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).
- **others** (*args*) – See `tl.prepro.shear2`.

Returns A list of processed images.

Return type numpy.array

Swirl

`tensorlayer.prepro.`**`swirl`**(*x*, *center=None*, *strength=1*, *radius=100*, *rotation=0*, *output_shape=None*, *order=1*, *mode='constant'*, *cval=0*, *clip=True*, *preserve_range=False*, *is_random=False*)

Swirl an image randomly or non-randomly, see scikit-image swirl API and example.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).

- **center** (*tuple or 2 int or None*) – Center coordinate of transformation (optional).
- **strength** (*float*) – The amount of swirling applied.
- **radius** (*float*) – The extent of the swirl in pixels. The effect dies out rapidly beyond radius.
- **rotation** (*float*) – Additional rotation applied to the image, usually [0, 360], relates to center.
- **output_shape** (*tuple of 2 int or None*) – Shape of the output image generated (height, width). By default the shape of the input image is preserved.
- **order** (*int, optional*) – The order of the spline interpolation, default is 1. The order has to be in the range 0-5. See skimage.transform.warp for detail.
- **mode** (*str*) – One of *constant* (default), *edge*, *symmetric reflect* and *wrap*. Points outside the boundaries of the input are filled according to the given mode, with *constant* used as the default. Modes match the behaviour of numpy.pad.
- **cval** (*float*) – Used in conjunction with mode *constant*, the value outside the image boundaries.
- **clip** (*boolean*) – Whether to clip the output to the range of values of the input image. This is enabled by default, since higher order interpolation may produce values outside the given input range.
- **preserve_range** (*boolean*) – Whether to keep the original range of values. Otherwise, the input image is converted according to the conventions of img_as_float.
- **is_random** (*boolean,*) –

  If True, random swirl. Default is False.

  - random center = [(0 ~ x.shape[0]), (0 ~ x.shape[1])]
  - random strength = [0, strength]
  - random radius = [1e-10, radius]
  - random rotation = [-rotation, rotation]

Returns A processed image.

Return type numpy.array

Examples

```
>>> x --> [row, col, 1] greyscale
>>> x = tl.prepro.swirl(x, strength=4, radius=100)
```

tensorlayer.prepro.**swirl_multi**(*x*, *center=None*, *strength=1*, *radius=100*, *rotation=0*, *output_shape=None*, *order=1*, *mode='constant'*, *cval=0*, *clip=True*, *preserve_range=False*, *is_random=False*)

Swirl multiple images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).

- **others** (*args*) – See `tl.prepro.swirl`.

Returns A list of processed images.

Return type numpy.array

Elastic transform

`tensorlayer.prepro.`**`elastic_transform`**(*x*, *alpha*, *sigma*, *mode='constant'*, *cval=0*, *is_random=False*)

Elastic transformation for image as described in [Simard2003].

Parameters

- **x** (*numpy.array*) – A greyscale image.
- **alpha** (*float*) – Alpha value for elastic transformation.
- **sigma** (*float or sequence of float*) – The smaller the sigma, the more transformation. Standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.
- **mode** (*str*) – See scipy.ndimage.filters.gaussian_filter. Default is *constant*.
- **cval** (*float,*) – Used in conjunction with *mode* of *constant*, the value outside the image boundaries.
- **is_random** (*boolean*) – Default is False.

Returns A processed image.

Return type numpy.array

Examples

```
>>> x = tl.prepro.elastic_transform(x, alpha=x.shape[1]*3, sigma=x.shape[1]*0.07)
```

References

- Github.
- Kaggle

`tensorlayer.prepro.`**`elastic_transform_multi`**(*x*, *alpha*, *sigma*, *mode='constant'*, *cval=0*, *is_random=False*)

Elastic transformation for images as described in [Simard2003].

Parameters

- **x** (*list of numpy.array*) – List of greyscale images.
- **others** (*args*) – See `tl.prepro.elastic_transform`.

Returns A list of processed images.

Return type numpy.array

Zoom

tensorlayer.prepro.**zoom**(*x*, *zoom_range=(0.9*, *1.1)*, *flags=None*, *border_mode='constant'*)
Zooming/Scaling a single image that height and width are changed together.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **zoom_range** (*float or tuple of 2 floats*) –

  The zooming/scaling ratio, greater than 1 means larger.

  – float, a fixed ratio.

  – tuple of 2 floats, randomly sample a value as the ratio between 2 values.
- **border_mode** (*str*) –

  – *constant*, pad the image with a constant value (i.e. black or 0)

  – *replicate*, the row or column at the very edge of the original is replicated to the extra border.

Returns A processed image.

Return type numpy.array

tensorlayer.prepro.**zoom_multi**(*x*, *zoom_range=(0.9*, *1.1)*, *flags=None*, *border_mode='constant'*)
Zoom in and out of images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).
- **others** (*args*) – See `tl.prepro.zoom`.

Returns A list of processed images.

Return type numpy.array

Respective Zoom

tensorlayer.prepro.**respective_zoom**(*x*, *h_range=(0.9*, *1.1)*, *w_range=(0.9*, *1.1)*, *flags=None*, *border_mode='constant'*)
Zooming/Scaling a single image that height and width are changed independently.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **h_range** (*float or tuple of 2 floats*) –

  The zooming/scaling ratio of height, greater than 1 means larger.

  – float, a fixed ratio.

  – tuple of 2 floats, randomly sample a value as the ratio between 2 values.
- **w_range** (*float or tuple of 2 floats*) –

  The zooming/scaling ratio of width, greater than 1 means larger.

  – float, a fixed ratio.

– tuple of 2 floats, randomly sample a value as the ratio between 2 values.

- **border_mode** (*str*) –
  – *constant*, pad the image with a constant value (i.e. black or 0)
  – *replicate*, the row or column at the very edge of the original is replicated to the extra border.

Returns A processed image.

Return type numpy.array

Brightness

tensorlayer.prepro.**brightness**(*x*, *gamma=1*, *gain=1*, *is_random=False*)
Change the brightness of a single image, randomly or non-randomly.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **gamma** (*float*) –

  Non negative real number. Default value is 1.

  – Small than 1 means brighter.
  – If *is_random* is True, gamma in a range of (1-gamma, 1+gamma).
- **gain** (*float*) – The constant multiplier. Default value is 1.
- **is_random** (*boolean*) – If True, randomly change brightness. Default is False.

Returns A processed image.

Return type numpy.array

References

- skimage.exposure.adjust_gamma
- chinese blog

tensorlayer.prepro.**brightness_multi**(*x*, *gamma=1*, *gain=1*, *is_random=False*)
Change the brightness of multiply images, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpyarray*) – List of images with dimension of [n_images, row, col, channel] (default).
- **others** (*args*) – See `tl.prepro.brightness`.

Returns A list of processed images.

Return type numpy.array

Brightness, contrast and saturation

`tensorlayer.prepro.illumination(x, gamma=1.0, contrast=1.0, saturation=1.0, is_random=False)`

Perform illumination augmentation for a single image, randomly or non-randomly.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **gamma** (*float*) –

  Change brightness (the same with `tl.prepro.brightness`)

  - if is_random=False, one float number, small than one means brighter, greater than one means darker.
  - if is_random=True, tuple of two float numbers, (min, max).
- **contrast** (*float*) –

  Change contrast.

  - if is_random=False, one float number, small than one means blur.
  - if is_random=True, tuple of two float numbers, (min, max).
- **saturation** (*float*) –

  Change saturation.

  - if is_random=False, one float number, small than one means unsaturation.
  - if is_random=True, tuple of two float numbers, (min, max).
- **is_random** (*boolean*) – If True, randomly change illumination. Default is False.

Returns A processed image.

Return type numpy.array

Examples

Random

```
>>> x = tl.prepro.illumination(x, gamma=(0.5, 5.0), contrast=(0.3, 1.0),␣
↪saturation=(0.7, 1.0), is_random=True)
```

Non-random

```
>>> x = tl.prepro.illumination(x, 0.5, 0.6, 0.8, is_random=False)
```

RGB to HSV

`tensorlayer.prepro.rgb_to_hsv(rgb)`

Input RGB image [0~255] return HSV image [0~1].

Parameters **rgb** (*numpy.array*) – An image with values between 0 and 255.

Returns A processed image.

Return type numpy.array

HSV to RGB

`tensorlayer.prepro.hsv_to_rgb(hsv)`

Input HSV image [0~1] return RGB image [0~255].

Parameters **hsv** (*numpy.array*) – An image with values between 0.0 and 1.0

Returns A processed image.

Return type numpy.array

Adjust Hue

`tensorlayer.prepro.adjust_hue(im, hout=0.66, is_offset=True, is_clip=True, is_random=False)`

Adjust hue of an RGB image.

This is a convenience method that converts an RGB image to float representation, converts it to HSV, add an offset to the hue channel, converts back to RGB and then back to the original data type. For TF, see tf.image.adjust_hue.and tf.image.random_hue.

Parameters

- **im** (*numpy.array*) – An image with values between 0 and 255.
- **hout** (*float*) –

  The scale value for adjusting hue.

  – If is_offset is False, set all hue values to this value. 0 is red; 0.33 is green; 0.66 is blue.

  – If is_offset is True, add this value as the offset to the hue channel.
- **is_offset** (*boolean*) – Whether *hout* is added on HSV as offset or not. Default is True.
- **is_clip** (*boolean*) – If HSV value smaller than 0, set to 0. Default is True.
- **is_random** (*boolean*) – If True, randomly change hue. Default is False.

Returns A processed image.

Return type numpy.array

Examples

Random, add a random value between -0.2 and 0.2 as the offset to every hue values.

```
>>> im_hue = tl.prepro.adjust_hue(image, hout=0.2, is_offset=True, is_
↪random=False)
```

Non-random, make all hue to green.

```
>>> im_green = tl.prepro.adjust_hue(image, hout=0.66, is_offset=False, is_
↪random=False)
```

References

- tf.image.random_hue.
- tf.image.adjust_hue.
- StackOverflow: Changing image hue with python PIL.

Resize

`tensorlayer.prepro.`**`imresize`**(*x*, *size=None*, *interp='bicubic'*, *mode=None*)

Resize an image by given output size and method.

Warning, this function will rescale the value to [0, 255].

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **size** (*list of 2 int or None*) – For height and width.
- **interp** (*str*) – Interpolation method for re-sizing (*nearest*, *lanczos*, *bilinear*, *bicubic* (default) or *cubic*).
- **mode** (*str*) – The PIL image mode (*P*, *L*, etc.) to convert image before resizing.

Returns A processed image.

Return type numpy.array

References

- scipy.misc.imresize

Pixel value scale

`tensorlayer.prepro.`**`pixel_value_scale`**(*im*, *val=0.9*, *clip=None*, *is_random=False*)

Scales each value in the pixels of the image.

Parameters

- **im** (*numpy.array*) – An image.
- **val** (*float*) –

  The scale value for changing pixel value.

  - If is_random=False, multiply this value with all pixels.
  - If is_random=True, multiply a value between [1-val, 1+val] with all pixels.
- **clip** (*tuple of 2 numbers*) – The minimum and maximum value.
- **is_random** (*boolean*) – If True, see `val`.

Returns A processed image.

Return type numpy.array

Examples

Random

```
>>> im = pixel_value_scale(im, 0.1, [0, 255], is_random=True)
```

Non-random

```
>>> im = pixel_value_scale(im, 0.9, [0, 255], is_random=False)
```

Normalization

tensorlayer.prepro.**samplewise_norm**(*x*, *rescale=None*, *samplewise_center=False*, *samplewise_std_normalization=False*, *channel_index=2*, *epsilon=1e-07*)

Normalize an image by rescale, samplewise centering and samplewise centering in order.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **rescale** (*float*) – Rescaling factor. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (before applying any other transformation)
- **samplewise_center** (*boolean*) – If True, set each sample mean to 0.
- **samplewise_std_normalization** (*boolean*) – If True, divide each input by its std.
- **epsilon** (*float*) – A small position value for dividing standard deviation.

Returns A processed image.

Return type numpy.array

Examples

```
>>> x = samplewise_norm(x, samplewise_center=True, samplewise_std_
↪normalization=True)
>>> print(x.shape, np.mean(x), np.std(x))
(160, 176, 1), 0.0, 1.0
```

Notes

When samplewise_center and samplewise_std_normalization are True. - For greyscale image, every pixels are subtracted and divided by the mean and std of whole image. - For RGB image, every pixels are subtracted and divided by the mean and std of this pixel i.e. the mean and std of a pixel is 0 and 1.

tensorlayer.prepro.**featurewise_norm**(*x*, *mean=None*, *std=None*, *epsilon=1e-07*)

Normalize every pixels by the same given mean and std, which are usually compute from all examples.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **mean** (*float*) – Value for subtraction.
- **std** (*float*) – Value for division.
- **epsilon** (*float*) – A small position value for dividing standard deviation.

Returns A processed image.

Return type numpy.array

Channel shift

tensorlayer.prepro.**channel_shift**(*x*, *intensity*, *is_random=False*, *channel_index=2*)

Shift the channels of an image, randomly or non-randomly, see numpy.rollaxis.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **intensity** (*float*) – Intensity of shifting.
- **is_random** (*boolean*) – If True, randomly shift. Default is False.
- **channel_index** (*int*) – Index of channel. Default is 2.

Returns A processed image.

Return type numpy.array

tensorlayer.prepro.**channel_shift_multi**(*x*, *intensity*, *is_random=False*, *channel_index=2*)
Shift the channels of images with the same arguments, randomly or non-randomly, see numpy.rollaxis. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

- **x** (*list of numpy.array*) – List of images with dimension of [n_images, row, col, channel] (default).
- **others** (*args*) – See `tl.prepro.channel_shift`.

Returns A list of processed images.

Return type numpy.array

Noise

tensorlayer.prepro.**drop**(*x*, *keep=0.5*)
Randomly set some pixels to zero by a given keeping probability.

Parameters

- **x** (*numpy.array*) – An image with dimension of [row, col, channel] or [row, col].
- **keep** (*float*) – The keeping probability (0, 1), the lower more values will be set to zero.

Returns A processed image.

Return type numpy.array

Numpy and PIL

tensorlayer.prepro.**array_to_img**(*x*, *dim_ordering=(0, 1, 2)*, *scale=True*)
Converts a numpy array to PIL image object (uint8 format).

Parameters

- **x** (*numpy.array*) – An image with dimension of 3 and channels of 1 or 3.
- **dim_ordering** (*tuple of 3 int*) – Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).
- **scale** (*boolean*) – If True, converts image to [0, 255] from any range of value like [-1, 2]. Default is True.

Returns An image.

Return type PIL.image

References

PIL Image.fromarray

Find contours

`tensorlayer.prepro.`**`find_contours`**(*x*, *level=0.8*, *fully_connected='low'*, *positive_orientation='low'*)

Find iso-valued contours in a 2D array for a given level value, returns list of (n, 2)-ndarrays see skimage.measure.find_contours.

Parameters

- **x** (*2D ndarray of double.*) – Input data in which to find contours.
- **level** (*float*) – Value along which to find contours in the array.
- **fully_connected** (*str*) – Either *low* or *high*. Indicates whether array elements below the given level value are to be considered fully-connected (and hence elements above the value will only be face connected), or vice-versa. (See notes below for details.)
- **positive_orientation** (*str*) – Either *low* or *high*. Indicates whether the output contours will produce positively-oriented polygons around islands of low- or high-valued elements. If *low* then contours will wind counter-clockwise around elements below the iso-value. Alternately, this means that low-valued elements are always on the left of the contour.

Returns Each contour is an ndarray of shape (n, 2), consisting of n (row, column) coordinates along the contour.

Return type list of (n,2)-ndarrays

Points to Image

`tensorlayer.prepro.`**`pt2map`**(*list_points=None*, *size=(100, 100)*, *val=1*)

Inputs a list of points, return a 2D image.

Parameters

- **list_points** (*list of 2 int*) – [[x, y], [x, y]..] for point coordinates.
- **size** (*tuple of 2 int*) – (w, h) for output size.
- **val** (*float or int*) – For the contour value.

Returns An image.

Return type numpy.array

Binary dilation

`tensorlayer.prepro.`**`binary_dilation`**(*x*, *radius=3*)

Return fast binary morphological dilation of an image. see skimage.morphology.binary_dilation.

Parameters

- **x** (*2D array*) – A binary image.
- **radius** (*int*) – For the radius of mask.

Returns A processed binary image.

Return type numpy.array

Greyscale dilation

`tensorlayer.prepro.dilation(x, radius=3)`
Return greyscale morphological dilation of an image, see skimage.morphology.dilation.

Parameters

- **x** (*2D array*) – An greyscale image.
- **radius** (*int*) – For the radius of mask.

Returns A processed greyscale image.

Return type numpy.array

Binary erosion

`tensorlayer.prepro.binary_erosion(x, radius=3)`
Return binary morphological erosion of an image, see skimage.morphology.binary_erosion.

Parameters

- **x** (*2D array*) – A binary image.
- **radius** (*int*) – For the radius of mask.

Returns A processed binary image.

Return type numpy.array

Greyscale erosion

`tensorlayer.prepro.erosion(x, radius=3)`
Return greyscale morphological erosion of an image, see skimage.morphology.erosion.

Parameters

- **x** (*2D array*) – A greyscale image.
- **radius** (*int*) – For the radius of mask.

Returns A processed greyscale image.

Return type numpy.array

2.4.3 Object detection

Tutorial for Image Aug

Hi, here is an example for image augmentation on VOC dataset.

```
import tensorlayer as tl

## download VOC 2012 dataset
imgs_file_list, _, _, _, classes, _, _,\
    _, objs_info_list, _ = tl.files.load_voc_dataset(dataset="2012")
```

(continues on next page)

(continued from previous page)

```
## parse annotation and convert it into list format
ann_list = []
for info in objs_info_list:
    ann = tl.prepro.parse_darknet_ann_str_to_list(info)
    c, b = tl.prepro.parse_darknet_ann_list_to_cls_box(ann)
    ann_list.append([c, b])

# read and save one image
idx = 2  # you can select your own image
image = tl.vis.read_image(imgs_file_list[idx])
tl.vis.draw_boxes_and_labels_to_image(image, ann_list[idx][0],
     ann_list[idx][1], [], classes, True, save_name='_im_original.png')

# left right flip
im_flip, coords = tl.prepro.obj_box_horizontal_flip(image,
        ann_list[idx][1], is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_flip, ann_list[idx][0],
        coords, [], classes, True, save_name='_im_flip.png')

# resize
im_resize, coords = tl.prepro.obj_box_imresize(image,
        coords=ann_list[idx][1], size=[300, 200], is_rescale=True)
tl.vis.draw_boxes_and_labels_to_image(im_resize, ann_list[idx][0],
        coords, [], classes, True, save_name='_im_resize.png')

# crop
im_crop, clas, coords = tl.prepro.obj_box_crop(image, ann_list[idx][0],
         ann_list[idx][1], wrg=200, hrg=200,
         is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_crop, clas, coords, [],
         classes, True, save_name='_im_crop.png')

# shift
im_shfit, clas, coords = tl.prepro.obj_box_shift(image, ann_list[idx][0],
        ann_list[idx][1], wrg=0.1, hrg=0.1,
        is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_shfit, clas, coords, [],
        classes, True, save_name='_im_shift.png')

# zoom
im_zoom, clas, coords = tl.prepro.obj_box_zoom(image, ann_list[idx][0],
        ann_list[idx][1], zoom_range=(1.3, 0.7),
        is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_zoom, clas, coords, [],
        classes, True, save_name='_im_zoom.png')
```

In practice, you may want to use threading method to process a batch of images as follows.

```
import tensorlayer as tl
import random

batch_size = 64
im_size = [416, 416]
n_data = len(imgs_file_list)
jitter = 0.2
def _data_pre_aug_fn(data):
```

(continues on next page)

(continued from previous page)

```
    im, ann = data
    clas, coords = ann
    ## change image brightness, contrast and saturation randomly
    im = tl.prepro.illumination(im, gamma=(0.5, 1.5),
             contrast=(0.5, 1.5), saturation=(0.5, 1.5), is_random=True)
    ## flip randomly
    im, coords = tl.prepro.obj_box_horizontal_flip(im, coords,
             is_rescale=True, is_center=True, is_random=True)
    ## randomly resize and crop image, it can have same effect as random zoom
    tmp0 = random.randint(1, int(im_size[0]*jitter))
    tmp1 = random.randint(1, int(im_size[1]*jitter))
    im, coords = tl.prepro.obj_box_imresize(im, coords,
            [im_size[0]+tmp0, im_size[1]+tmp1], is_rescale=True,
             interp='bicubic')
    im, clas, coords = tl.prepro.obj_box_crop(im, clas, coords,
             wrg=im_size[1], hrg=im_size[0], is_rescale=True,
             is_center=True, is_random=True)
    ## rescale value from [0, 255] to [-1, 1] (optional)
    im = im / 127.5 - 1
    return im, [clas, coords]

# randomly read a batch of image and the corresponding annotations
idexs = tl.utils.get_random_int(min=0, max=n_data-1, number=batch_size)
b_im_path = [imgs_file_list[i] for i in idexs]
b_images = tl.prepro.threading_data(b_im_path, fn=tl.vis.read_image)
b_ann = [ann_list[i] for i in idexs]

# threading process
data = tl.prepro.threading_data([_ for _ in zip(b_images, b_ann)],
              _data_pre_aug_fn)
b_images2 = [d[0] for d in data]
b_ann = [d[1] for d in data]

# save all images
for i in range(len(b_images)):
    tl.vis.draw_boxes_and_labels_to_image(b_images[i],
             ann_list[idexs[i]][0], ann_list[idexs[i]][1], [],
             classes, True, save_name='_bbox_vis_%d_original.png' % i)
    tl.vis.draw_boxes_and_labels_to_image((b_images2[i]+1)*127.5,
             b_ann[i][0], b_ann[i][1], [], classes, True,
             save_name='_bbox_vis_%d.png' % i)
```

Image Aug with TF Dataset API

- Example code for VOC here.

Coordinate pixel unit to percentage

`tensorlayer.prepro.`**`obj_box_coord_rescale`**`(`*coord=None*, *shape=None*`)`

Scale down one coordinates from pixel unit to the ratio of image size i.e. in the range of [0, 1]. It is the reverse process of `obj_box_coord_scale_to_pixelunit`.

Parameters

- **coords** (*list of 4 int or None*) – One coordinates of one image e.g. [x, y, w, h].

- **shape** (*list of 2 int or None*) – For [height, width].

Returns New bounding box.

Return type list of 4 numbers

Examples

```
>>> coord = tl.prepro.obj_box_coord_rescale(coord=[30, 40, 50, 50], shape=[100,␣
↪100])
  [0.3, 0.4, 0.5, 0.5]
```

Coordinates pixel unit to percentage

tensorlayer.prepro.**obj_box_coords_rescale**(*coords=None*, *shape=None*)

Scale down a list of coordinates from pixel unit to the ratio of image size i.e. in the range of [0, 1].

Parameters

- **coords** (*list of list of 4 ints or None*) – For coordinates of more than one images .e.g.[[x, y, w, h], [x, y, w, h], . . .].
- **shape** (*list of 2 int or None*) – height, width].

Returns A list of new bounding boxes.

Return type list of list of 4 numbers

Examples

```
>>> coords = obj_box_coords_rescale(coords=[[30, 40, 50, 50], [10, 10, 20, 20]],␣
↪shape=[100, 100])
>>> print(coords)
  [[0.3, 0.4, 0.5, 0.5], [0.1, 0.1, 0.2, 0.2]]
>>> coords = obj_box_coords_rescale(coords=[[30, 40, 50, 50]], shape=[50, 100])
>>> print(coords)
  [[0.3, 0.8, 0.5, 1.0]]
>>> coords = obj_box_coords_rescale(coords=[[30, 40, 50, 50]], shape=[100, 200])
>>> print(coords)
  [[0.15, 0.4, 0.25, 0.5]]
```

Returns New coordinates.

Return type list of 4 numbers

Coordinate percentage to pixel unit

tensorlayer.prepro.**obj_box_coord_scale_to_pixelunit**(*coord*, *shape=None*)

Convert one coordinate [x, y, w (or x2), h (or y2)] in ratio format to image coordinate format. It is the reverse process of `obj_box_coord_rescale`.

Parameters

- **coord** (*list of 4 float*) – One coordinate of one image [x, y, w (or x2), h (or y2)] in ratio format, i.e value range [0~1].

- **shape** (*tuple of 2 or None*) – For [height, width].

Returns New bounding box.

Return type list of 4 numbers

Examples

```
>>> x, y, x2, y2 = tl.prepro.obj_box_coord_scale_to_pixelunit([0.2, 0.3, 0.5, 0.
↪7], shape=(100, 200, 3))
  [40, 30, 100, 70]
```

Coordinate [x_center, x_center, w, h] to up-left button-right

tensorlayer.prepro.**obj_box_coord_centroid_to_upleft_butright**(*coord*, *to_int=False*)

Convert one coordinate [x_center, y_center, w, h] to [x1, y1, x2, y2] in up-left and botton-right format.

Parameters

- **coord** (*list of 4 int/float*) – One coordinate.
- **to_int** (*boolean*) – Whether to convert output as integer.

Returns New bounding box.

Return type list of 4 numbers

Examples

```
>>> coord = obj_box_coord_centroid_to_upleft_butright([30, 40, 20, 20])
  [20, 30, 40, 50]
```

Coordinate up-left button-right to [x_center, x_center, w, h]

tensorlayer.prepro.**obj_box_coord_upleft_butright_to_centroid**(*coord*)

Convert one coordinate [x1, y1, x2, y2] to [x_center, y_center, w, h]. It is the reverse process of `obj_box_coord_centroid_to_upleft_butright`.

Parameters **coord** (*list of 4 int/float*) – One coordinate.

Returns New bounding box.

Return type list of 4 numbers

Coordinate [x_center, x_center, w, h] to up-left-width-high

tensorlayer.prepro.**obj_box_coord_centroid_to_upleft**(*coord*)

Convert one coordinate [x_center, y_center, w, h] to [x, y, w, h]. It is the reverse process of `obj_box_coord_upleft_to_centroid`.

Parameters **coord** (*list of 4 int/float*) – One coordinate.

Returns New bounding box.

Return type list of 4 numbers

Coordinate up-left-width-high to [x_center, x_center, w, h]

tensorlayer.prepro.**obj_box_coord_upleft_to_centroid**(*coord*)

Convert one coordinate [x, y, w, h] to [x_center, y_center, w, h]. It is the reverse process of `obj_box_coord_centroid_to_upleft`.

Parameters **coord** (*list of 4 int/float*) – One coordinate.

Returns New bounding box.

Return type list of 4 numbers

Darknet format string to list

tensorlayer.prepro.**parse_darknet_ann_str_to_list**(*annotations*)

Input string format of class, x, y, w, h, return list of list format.

Parameters **annotations** (*str*) – The annotations in darkent format "class, x, y, w, h . . . ." seperated by "\n".

Returns List of bounding box.

Return type list of list of 4 numbers

Darknet format split class and coordinate

tensorlayer.prepro.**parse_darknet_ann_list_to_cls_box**(*annotations*)

Parse darknet annotation format into two lists for class and bounding box.

Input list of [[class, x, y, w, h], . . .], return two list of [class . . .] and [[x, y, w, h], . . .].

Parameters **annotations** (*list of list*) – A list of class and bounding boxes of images e.g. [[class, x, y, w, h], . . .]

Returns

- *list of int* – List of class labels.
- *list of list of 4 numbers* – List of bounding box.

Image Aug - Flip

tensorlayer.prepro.**obj_box_horizontal_flip**(*im*, *coords=None*, *is_rescale=False*, *is_center=False*, *is_random=False*)

Left-right flip the image and coordinates for object detection.

Parameters

- **im** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **coords** (*list of list of 4 int/float or None*) – Coordinates [[x, y, w, h], [x, y, w, h], . . .].
- **is_rescale** (*boolean*) – Set to True, if the input coordinates are rescaled to [0, 1]. Default is False.
- **is_center** (*boolean*) – Set to True, if the x and y of coordinates are the centroid (i.e. darknet format). Default is False.
- **is_random** (*boolean*) – If True, randomly flip. Default is False.

Returns

- *numpy.array* – A processed image
- *list of list of 4 numbers* – A list of new bounding boxes.

Examples

```
>>> im = np.zeros([80, 100])    # as an image with shape width=100, height=80
>>> im, coords = obj_box_left_right_flip(im, coords=[[0.2, 0.4, 0.3, 0.3], [0.1,␣
↪0.5, 0.2, 0.3]], is_rescale=True, is_center=True, is_random=False)
>>> print(coords)
  [[0.8, 0.4, 0.3, 0.3], [0.9, 0.5, 0.2, 0.3]]
>>> im, coords = obj_box_left_right_flip(im, coords=[[0.2, 0.4, 0.3, 0.3]], is_
↪rescale=True, is_center=False, is_random=False)
>>> print(coords)
  [[0.5, 0.4, 0.3, 0.3]]
>>> im, coords = obj_box_left_right_flip(im, coords=[[20, 40, 30, 30]], is_
↪rescale=False, is_center=True, is_random=False)
>>> print(coords)
  [[80, 40, 30, 30]]
>>> im, coords = obj_box_left_right_flip(im, coords=[[20, 40, 30, 30]], is_
↪rescale=False, is_center=False, is_random=False)
>>> print(coords)
  [[50, 40, 30, 30]]
```

Image Aug - Resize

tensorlayer.prepro.**obj_box_imresize**(*im*, *coords=None*, *size=None*, *interp='bicubic'*, *mode=None*, *is_rescale=False*)

Resize an image, and compute the new bounding box coordinates.

Parameters

- **im** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **coords** (*list of list of 4 int/float or None*) – Coordinates [[x, y, w, h], [x, y, w, h], . . .]
- **interp and mode** (*size*) – See `tl.prepro.imresize`.
- **is_rescale** (*boolean*) – Set to True, if the input coordinates are rescaled to [0, 1], then return the original coordinates. Default is False.

Returns

- *numpy.array* – A processed image
- *list of list of 4 numbers* – A list of new bounding boxes.

Examples

```
>>> im = np.zeros([80, 100, 3])    # as an image with shape width=100, height=80
>>> _, coords = obj_box_imresize(im, coords=[[20, 40, 30, 30], [10, 20, 20, 20]],␣
↪size=[160, 200], is_rescale=False)
>>> print(coords)
  [[40, 80, 60, 60], [20, 40, 40, 40]]
```

(continues on next page)

(continued from previous page)

```
>>> _, coords = obj_box_imresize(im, coords=[[20, 40, 30, 30]], size=[40, 100], 
↪is_rescale=False)
>>> print(coords)
  [[20, 20, 30, 15]]
>>> _, coords = obj_box_imresize(im, coords=[[20, 40, 30, 30]], size=[60, 150], 
↪is_rescale=False)
>>> print(coords)
  [[30, 30, 45, 22]]
>>> im2, coords = obj_box_imresize(im, coords=[[0.2, 0.4, 0.3, 0.3]], size=[160, 
↪200], is_rescale=True)
>>> print(coords, im2.shape)
  [[0.2, 0.4, 0.3, 0.3]] (160, 200, 3)
```

Image Aug - Crop

tensorlayer.prepro.**obj_box_crop**(*im*, *classes=None*, *coords=None*, *wrg=100*, *hrg=100*, *is_rescale=False*, *is_center=False*, *is_random=False*, *thresh_wh=0.02*, *thresh_wh2=12.0*)

Randomly or centrally crop an image, and compute the new bounding box coordinates. Objects outside the cropped image will be removed.

Parameters

- **im** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **classes** (*list of int or None*) – Class IDs.
- **coords** (*list of list of 4 int/float or None*) – Coordinates [[x, y, w, h], [x, y, w, h], . . .]
- **hrg and is_random** (*wrg*) – See tl.prepro.crop.
- **is_rescale** (*boolean*) – Set to True, if the input coordinates are rescaled to [0, 1]. Default is False.
- **is_center** (*boolean, default False*) – Set to True, if the x and y of coordinates are the centroid (i.e. darknet format). Default is False.
- **thresh_wh** (*float*) – Threshold, remove the box if its ratio of width(height) to image size less than the threshold.
- **thresh_wh2** (*float*) – Threshold, remove the box if its ratio of width to height or vice verse higher than the threshold.

Returns

- *numpy.array* – A processed image
- *list of int* – A list of classes
- *list of list of 4 numbers* – A list of new bounding boxes.

Image Aug - Shift

`tensorlayer.prepro.`**`obj_box_shift`**(*im, classes=None, coords=None, wrg=0.1, hrg=0.1, row_index=0, col_index=1, channel_index=2, fill_mode='nearest', cval=0.0, order=1, is_rescale=False, is_center=False, is_random=False, thresh_wh=0.02, thresh_wh2=12.0*)

Shift an image randomly or non-randomly, and compute the new bounding box coordinates. Objects outside the cropped image will be removed.

Parameters

- **im** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **classes** (*list of int or None*) – Class IDs.
- **coords** (*list of list of 4 int/float or None*) – Coordinates [[x, y, w, h], [x, y, w, h], . . .]
- **hrg row_index col_index channel_index is_random fill_mode cval and order** (*wrg,*) –
- **is_rescale** (*boolean*) – Set to True, if the input coordinates are rescaled to [0, 1]. Default is False.
- **is_center** (*boolean*) – Set to True, if the x and y of coordinates are the centroid (i.e. darknet format). Default is False.
- **thresh_wh** (*float*) – Threshold, remove the box if its ratio of width(height) to image size less than the threshold.
- **thresh_wh2** (*float*) – Threshold, remove the box if its ratio of width to height or vice verse higher than the threshold.

Returns

- *numpy.array* – A processed image
- *list of int* – A list of classes
- *list of list of 4 numbers* – A list of new bounding boxes.

Image Aug - Zoom

`tensorlayer.prepro.`**`obj_box_zoom`**(*im, classes=None, coords=None, zoom_range=(0.9, 1.1), row_index=0, col_index=1, channel_index=2, fill_mode='nearest', cval=0.0, order=1, is_rescale=False, is_center=False, is_random=False, thresh_wh=0.02, thresh_wh2=12.0*)

Zoom in and out of a single image, randomly or non-randomly, and compute the new bounding box coordinates. Objects outside the cropped image will be removed.

Parameters

- **im** (*numpy.array*) – An image with dimension of [row, col, channel] (default).
- **classes** (*list of int or None*) – Class IDs.
- **coords** (*list of list of 4 int/float or None*) – Coordinates [[x, y, w, h], [x, y, w, h], . . .].
- **row_index col_index channel_index is_random fill_mode cval and order** (*zoom_range*) –

- **is_rescale** (*boolean*) – Set to True, if the input coordinates are rescaled to [0, 1]. Default is False.
- **is_center** (*boolean*) – Set to True, if the x and y of coordinates are the centroid. (i.e. darknet format). Default is False.
- **thresh_wh** (*float*) – Threshold, remove the box if its ratio of width(height) to image size less than the threshold.
- **thresh_wh2** (*float*) – Threshold, remove the box if its ratio of width to height or vice verse higher than the threshold.

Returns

- *numpy.array* – A processed image
- *list of int* – A list of classes
- *list of list of 4 numbers* – A list of new bounding boxes.

2.4.4 Keypoints

Image Aug - Crop

tensorlayer.prepro.**keypoint_random_crop**(*image*, *annos*, *mask=None*, *size=(368, 368)*)

Randomly crop an image and corresponding keypoints without influence scales, given by `keypoint_random_resize_shortestedge`.

Parameters

- **image** (*3 channel image*) – The given image for augmentation.
- **annos** (*list of list of floats*) – The keypoints annotation of people.
- **mask** (*single channel image or None*) – The mask if available.
- **size** (*tuple of int*) – The size of returned image.

Returns

Return type preprocessed image, annotation, mask

Image Aug - Resize then Crop

tensorlayer.prepro.**keypoint_resize_random_crop**(*image*, *annos*, *mask=None*, *size=(368, 368)*)

Reszie the image to make either its width or height equals to the given sizes. Then randomly crop image without influence scales. Resize the image match with the minimum size before cropping, this API will change the zoom scale of object.

Parameters

- **image** (*3 channel image*) – The given image for augmentation.
- **annos** (*list of list of floats*) – The keypoints annotation of people.
- **mask** (*single channel image or None*) – The mask if available.
- **size** (*tuple of int*) – The size (height, width) of returned image.

Returns

Return type preprocessed image, annos, mask

Image Aug - Rotate

`tensorlayer.prepro.`**`keypoint_random_rotate`**(*image*, *annos*, *mask=None*, *rg=15.0*)

Rotate an image and corresponding keypoints.

Parameters

- **image** (*3 channel image*) – The given image for augmentation.
- **annos** (*list of list of floats*) – The keypoints annotation of people.
- **mask** (*single channel image or None*) – The mask if available.
- **rg** (*int or float*) – Degree to rotate, usually 0 ~ 180.

Returns

Return type preprocessed image, annos, mask

Image Aug - Flip

`tensorlayer.prepro.`**`keypoint_random_flip`**(*image*, *annos*, *mask=None*, *prob=0.5*, *flip_list=(0, 1, 5, 6, 7, 2, 3, 4, 11, 12, 13, 8, 9, 10, 15, 14, 17, 16, 18)*)

Flip an image and corresponding keypoints.

Parameters

- **image** (*3 channel image*) – The given image for augmentation.
- **annos** (*list of list of floats*) – The keypoints annotation of people.
- **mask** (*single channel image or None*) – The mask if available.
- **prob** (*float, 0 to 1*) – The probability to flip the image, if 1, always flip the image.
- **flip_list** (*tuple of int*) – Denotes how the keypoints number be changed after flipping which is required for pose estimation task. The left and right body should be maintained rather than switch. (Default COCO format). Set to an empty tuple if you don't need to maintain left and right information.

Returns

Return type preprocessed image, annos, mask

Image Aug - Resize

`tensorlayer.prepro.`**`keypoint_random_resize`**(*image*, *annos*, *mask=None*, *zoom_range=(0.8, 1.2)*)

Randomly resize an image and corresponding keypoints. The height and width of image will be changed independently, so the scale will be changed.

Parameters

- **image** (*3 channel image*) – The given image for augmentation.
- **annos** (*list of list of floats*) – The keypoints annotation of people.
- **mask** (*single channel image or None*) – The mask if available.
- **zoom_range** (*tuple of two floats*) – The minimum and maximum factor to zoom in or out, e.g (0.5, 1) means zoom out 1~2 times.

Returns

Return type preprocessed image, annos, mask

Image Aug - Resize Shortest Edge

`tensorlayer.prepro.`**`keypoint_random_resize_shortestedge`**(*image*, *annos*, *mask=None*, *min_size=(368, 368)*, *zoom_range=(0.8, 1.2)*, *pad_val=(0, 0, numpy.random.uniform)*)

Randomly resize an image and corresponding keypoints based on shorter edgeself. If the resized image is smaller than *min_size*, uses padding to make shape matchs *min_size*. The height and width of image will be changed together, the scale would not be changed.

Parameters

- **image** (*3 channel image*) – The given image for augmentation.
- **annos** (*list of list of floats*) – The keypoints annotation of people.
- **mask** (*single channel image or None*) – The mask if available.
- **min_size** (*tuple of two int*) – The minimum size of height and width.
- **zoom_range** (*tuple of two floats*) – The minimum and maximum factor to zoom in or out, e.g (0.5, 1) means zoom out 1~2 times.
- **pad_val** (*int/float, or tuple of int or random function*) – The three padding values for RGB channels respectively.

Returns

Return type preprocessed image, annos, mask

2.4.5 Sequence

More related functions can be found in `tensorlayer.nlp`.

Padding

`tensorlayer.prepro.`**`pad_sequences`**(*sequences*, *maxlen=None*, *dtype='int32'*, *padding='post'*, *truncating='pre'*, *value=0.0*)

Pads each sequence to the same length: the length of the longest sequence. If maxlen is provided, any sequence longer than maxlen is truncated to maxlen. Truncation happens off either the beginning (default) or the end of the sequence. Supports post-padding and pre-padding (default).

Parameters

- **sequences** (*list of list of int*) – All sequences where each row is a sequence.
- **maxlen** (*int*) – Maximum length.
- **dtype** (*numpy.dtype or str*) – Data type to cast the resulting sequence.
- **padding** (*str*) – Either 'pre' or 'post', pad either before or after each sequence.
- **truncating** (*str*) – Either 'pre' or 'post', remove values from sequences larger than maxlen either in the beginning or in the end of the sequence
- **value** (*float*) – Value to pad the sequences to the desired value.

Returns **x** – With dimensions (number_of_sequences, maxlen)

Return type numpy.array

Examples

```
>>> sequences = [[1,1,1,1,1],[2,2,2],[3,3]]
>>> sequences = pad_sequences(sequences, maxlen=None, dtype='int32',
...                  padding='post', truncating='pre', value=0.)
[[1 1 1 1 1]
 [2 2 2 0 0]
 [3 3 0 0 0]]
```

Remove Padding

`tensorlayer.prepro.`**`remove_pad_sequences`**(*sequences*, *pad_id=0*)

Remove padding.

Parameters

- **sequences** (*list of list of int*) – All sequences where each row is a sequence.
- **pad_id** (*int*) – The pad ID.

Returns The processed sequences.

Return type list of list of int

Examples

```
>>> sequences = [[2,3,4,0,0], [5,1,2,3,4,0,0,0], [4,5,0,2,4,0,0,0]]
>>> print(remove_pad_sequences(sequences, pad_id=0))
[[2, 3, 4], [5, 1, 2, 3, 4], [4, 5, 0, 2, 4]]
```

Process

`tensorlayer.prepro.`**`process_sequences`**(*sequences*, *end_id=0*, *pad_val=0*, *is_shorten=True*, *remain_end_id=False*)

Set all tokens(ids) after END token to the padding value, and then shorten (option) it to the maximum sequence length in this batch.

Parameters

- **sequences** (*list of list of int*) – All sequences where each row is a sequence.
- **end_id** (*int*) – The special token for END.
- **pad_val** (*int*) – Replace the *end_id* and the IDs after *end_id* to this value.
- **is_shorten** (*boolean*) – Shorten the sequences. Default is True.
- **remain_end_id** (*boolean*) – Keep an *end_id* in the end. Default is False.

Returns The processed sequences.

Return type list of list of int

Examples

```
>>> sentences_ids = [[4, 3, 5, 3, 2, 2, 2, 2],  <-- end_id is 2
...                  [5, 3, 9, 4, 9, 2, 2, 3]]  <-- end_id is 2
>>> sentences_ids = precess_sequences(sentences_ids, end_id=vocab.end_id, pad_
↪val=0, is_shorten=True)
[[4, 3, 5, 3, 0], [5, 3, 9, 4, 9]]
```

Add Start ID

`tensorlayer.prepro.sequences_add_start_id`(*sequences*, *start_id=0*, *remove_last=False*)

Add special start token(id) in the beginning of each sequence.

Parameters

- **sequences** (*list of list of int*) – All sequences where each row is a sequence.
- **start_id** (*int*) – The start ID.
- **remove_last** (*boolean*) – Remove the last value of each sequences. Usually be used for removing the end ID.

Returns The processed sequences.

Return type list of list of int

Examples

```
>>> sentences_ids = [[4,3,5,3,2,2,2,2], [5,3,9,4,9,2,2,3]]
>>> sentences_ids = sequences_add_start_id(sentences_ids, start_id=2)
[[2, 4, 3, 5, 3, 2, 2, 2, 2], [2, 5, 3, 9, 4, 9, 2, 2, 3]]
>>> sentences_ids = sequences_add_start_id(sentences_ids, start_id=2, remove_
↪last=True)
[[2, 4, 3, 5, 3, 2, 2, 2], [2, 5, 3, 9, 4, 9, 2, 2]]
```

For Seq2seq

```
>>> input = [a, b, c]
>>> target = [x, y, z]
>>> decode_seq = [start_id, a, b] <-- sequences_add_start_id(input, start_id,␣
↪True)
```

Add End ID

`tensorlayer.prepro.sequences_add_end_id`(*sequences*, *end_id=888*)

Add special end token(id) in the end of each sequence.

Parameters

- **sequences** (*list of list of int*) – All sequences where each row is a sequence.
- **end_id** (*int*) – The end ID.

Returns The processed sequences.

Return type list of list of int

Examples

```
>>> sequences = [[1,2,3],[4,5,6,7]]
>>> print(sequences_add_end_id(sequences, end_id=999))
[[1, 2, 3, 999], [4, 5, 6, 999]]
```

Add End ID after pad

`tensorlayer.prepro.`**`sequences_add_end_id_after_pad`**(*sequences*, *end_id=888*, *pad_id=0*)

Add special end token(id) in the end of each sequence.

Parameters

- **sequences** (*list of list of int*) – All sequences where each row is a sequence.
- **end_id** (*int*) – The end ID.
- **pad_id** (*int*) – The pad ID.

Returns The processed sequences.

Return type list of list of int

Examples

```
>>> sequences = [[1,2,0,0], [1,2,3,0], [1,2,3,4]]
>>> print(sequences_add_end_id_after_pad(sequences, end_id=99, pad_id=0))
[[1, 2, 99, 0], [1, 2, 3, 99], [1, 2, 3, 4]]
```

Get Mask

`tensorlayer.prepro.`**`sequences_get_mask`**(*sequences*, *pad_val=0*)

Return mask for sequences.

Parameters

- **sequences** (*list of list of int*) – All sequences where each row is a sequence.
- **pad_val** (*int*) – The pad value.

Returns The mask.

Return type list of list of int

Examples

```
>>> sentences_ids = [[4, 0, 5, 3, 0, 0],
...                  [5, 3, 9, 4, 9, 0]]
>>> mask = sequences_get_mask(sentences_ids, pad_val=0)
[[1 1 1 1 0 0]
 [1 1 1 1 1 0]]
```

2.5 API - Files

A collections of helper functions to work with dataset. Load benchmark dataset, save and restore model, save and load variables.

TensorLayer provides rich layer implementations trailed for various benchmarks and domain-specific problems. In addition, we also support transparent access to native TensorFlow parameters. For example, we provide not only layers for local response normalization, but also layers that allow user to apply `tf.nn.lrn` on `network.outputs`. More functions can be found in TensorFlow API.

load_mnist_dataset([shape, path])	Load the original mnist.
load_fashion_mnist_dataset([shape, path])	Load the fashion mnist.
load_cifar10_dataset([shape, path, plotable])	Load CIFAR-10 dataset.
load_cropped_svhn([path, include_extra])	Load Cropped SVHN.
load_ptb_dataset([path])	Load Penn TreeBank (PTB) dataset.
load_matt_mahoney_text8_dataset([path])	Load Matt Mahoney's dataset.
load_imdb_dataset([path, nb_words, ...])	Load IMDB dataset.
load_nietzsche_dataset([path])	Load Nietzsche dataset.
load_wmt_en_fr_dataset([path])	Load WMT'15 English-to-French translation dataset.
load_flickr25k_dataset([tag, path, ...])	Load Flickr25K dataset.
load_flickr1M_dataset([tag, size, path, ...])	Load Flick1M dataset.
load_cyclegan_dataset([filename, path])	Load images from CycleGAN's database, see this link.
load_celebA_dataset([path])	Load CelebA dataset
load_voc_dataset([path, dataset, ...])	Pascal VOC 2007/2012 Dataset.
load_mpii_pose_dataset([path, is_16_pos_only])	Load MPII Human Pose Dataset.
download_file_from_google_drive(ID, destination)	Download file from Google Drive.
save_npz([save_list, name])	Input parameters and the file name, save parameters into .npz file.
load_npz([path, name])	Load the parameters of a Model saved by tl.files.save_npz().
assign_weights(weights, network)	Assign the given parameters to the TensorLayer network.
load_and_assign_npz([name, network])	Load model from npz and assign to a network.
save_npz_dict([save_list, name])	Input parameters and the file name, save parameters as a dictionary into .npz file.
load_and_assign_npz_dict([name, network, skip])	Restore the parameters saved by `tl.files.save_npz_dict()`.
save_weights_to_hdf5(filepath, network)	Input filepath and save weights in hdf5 format.
load_hdf5_to_weights_in_order(filepath, network)	Load weights sequentially from a given file of hdf5 format
load_hdf5_to_weights(filepath, network[, skip])	Load weights by name from a given file of hdf5 format
save_any_to_npy([save_dict, name])	Save variables to *.npy* file.
load_npy_to_any([path, name])	Load *.npy* file.
file_exists(filepath)	Check whether a file exists by given file path.
folder_exists(folderpath)	Check whether a folder exists by given folder path.
del_file(filepath)	Delete a file by given file path.
del_folder(folderpath)	Delete a folder by given folder path.
read_file(filepath)	Read a file and return a string.

Continued on next page

Table 5 – continued from previous page

`load_file_list`([path, regx, printable, ...])	Return a file list in a folder by given a path and regular expression.
`load_folder_list`([path])	Return a folder list in a folder by given a folder path.
`exists_or_mkdir`(path[, verbose])	Check a folder by given name, if not exist, create the folder and return False, if directory exists, return True.
`maybe_download_and_extract`(filename, ...[, ...])	Checks if file exists in working_directory otherwise tries to dowload the file, and optionally also tries to extract the file if format is ".zip" or ".tar"
`natural_keys`(text)	Sort list of string with number in human order.

2.5.1 Load dataset functions

MNIST

tensorlayer.files.**load_mnist_dataset**(*shape=(-1, 784), path='data'*)

Load the original mnist.

Automatically download MNIST dataset and return the training, validation and test set with 50000, 10000 and 10000 digit images respectively.

Parameters

- **shape** (*tuple*) – The shape of digit images (the default is (-1, 784), alternatively (-1, 28, 28, 1)).
- **path** (*str*) – The path that the data is downloaded to.

Returns X_train, y_train, X_val, y_val, X_test, y_test – Return splitted training/validation/test set respectively.

Return type tuple

Examples

```
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_mnist_
↪dataset(shape=(-1,784), path='datasets')
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_mnist_
↪dataset(shape=(-1, 28, 28, 1))
```

Fashion-MNIST

tensorlayer.files.**load_fashion_mnist_dataset**(*shape=(-1, 784), path='data'*)

Load the fashion mnist.

Automatically download fashion-MNIST dataset and return the training, validation and test set with 50000, 10000 and 10000 fashion images respectively, examples.

Parameters

- **shape** (*tuple*) – The shape of digit images (the default is (-1, 784), alternatively (-1, 28, 28, 1)).
- **path** (*str*) – The path that the data is downloaded to.

Returns **X_train, y_train, X_val, y_val, X_test, y_test** – Return splitted training/validation/test set respectively.

Return type tuple

Examples

```
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_fashion_mnist_
↪dataset(shape=(-1,784), path='datasets')
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_fashion_mnist_
↪dataset(shape=(-1, 28, 28, 1))
```

CIFAR-10

`tensorlayer.files.`**`load_cifar10_dataset`**(*shape=(-1, 32, 32, 3)*, *path='data'*, *plotable=False*)

Load CIFAR-10 dataset.

It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Parameters

- **shape** (*tupe*) – The shape of digit images e.g. (-1, 3, 32, 32) and (-1, 32, 32, 3).
- **path** (*str*) – The path that the data is downloaded to, defaults is `data/cifar10/`.
- **plotable** (*boolean*) – Whether to plot some image examples, False as default.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,␣
↪32, 32, 3))
```

References

- CIFAR website
- Data download link
- https://teratail.com/questions/28932

SVHN

`tensorlayer.files.`**`load_cropped_svhn`**(*path='data'*, *include_extra=True*)

Load Cropped SVHN.

The Cropped Street View House Numbers (SVHN) Dataset contains 32x32x3 RGB images. Digit '1' has label 1, '9' has label 9 and '0' has label 0 (the original dataset uses 10 to represent '0'), see ufldl website.

Parameters

- **path** (*str*) – The path that the data is downloaded to.
- **include_extra** (*boolean*) – If True (default), add extra images to the training set.

Returns **X_train, y_train, X_test, y_test** – Return splitted training/test set respectively.

Return type tuple

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cropped_svhn(include_
↪extra=False)
>>> tl.vis.save_images(X_train[0:100], [10, 10], 'svhn.png')
```

Penn TreeBank (PTB)

`tensorlayer.files.load_ptb_dataset(path='data')`

Load Penn TreeBank (PTB) dataset.

It is used in many LANGUAGE MODELING papers, including "Empirical Evaluation and Combination of Advanced Language Modeling Techniques", "Recurrent Neural Network Regularization". It consists of 929k training words, 73k validation words, and 82k test words. It has 10k words in its vocabulary.

Parameters **path** (*str*) – The path that the data is downloaded to, defaults is `data/ptb/`.

Returns

- **train_data, valid_data, test_data** (*list of int*) – The training, validating and testing data in integer format.
- **vocab_size** (*int*) – The vocabulary size.

Examples

```
>>> train_data, valid_data, test_data, vocab_size = tl.files.load_ptb_dataset()
```

References

- `tensorflow.models.rnn.ptb import reader`
- Manual download

Notes

- If you want to get the raw data, see the source code.

Matt Mahoney's text8

`tensorlayer.files.load_matt_mahoney_text8_dataset(path='data')`

Load Matt Mahoney's dataset.

Download a text file from Matt Mahoney's website if not present, and make sure it's the right size. Extract the first file enclosed in a zip file as a list of words. This dataset can be used for Word Embedding.

Parameters **path** (*str*) – The path that the data is downloaded to, defaults is `data/mm_test8/`.

Returns The raw text data e.g. [.... 'their', 'families', 'who', 'were', 'expelled', 'from', 'jerusalem', ...]

Return type list of str

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> print('Data size', len(words))
```

IMBD

`tensorlayer.files.`**`load_imdb_dataset`**(*path='data', nb_words=None, skip_top=0, maxlen=None, test_split=0.2, seed=113, start_char=1, oov_char=2, index_from=3*)

Load IMDB dataset.

Parameters

- **path** (*str*) – The path that the data is downloaded to, defaults is `data/imdb/`.
- **nb_words** (*int*) – Number of words to get.
- **skip_top** (*int*) – Top most frequent words to ignore (they will appear as oov_char value in the sequence data).
- **maxlen** (*int*) – Maximum sequence length. Any longer sequence will be truncated.
- **seed** (*int*) – Seed for reproducible data shuffling.
- **start_char** (*int*) – The start of a sequence will be marked with this character. Set to 1 because 0 is usually the padding character.
- **oov_char** (*int*) – Words that were cut out because of the num_words or skip_top limit will be replaced with this character.
- **index_from** (*int*) – Index actual words with this index and higher.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_imdb_dataset(
...                                 nb_words=20000, test_split=0.2)
>>> print('X_train.shape', X_train.shape)
(20000,)  [[1, 62, 74, ... 1033, 507, 27],[1, 60, 33, ... 13, 1053, 7]..]
>>> print('y_train.shape', y_train.shape)
(20000,)  [1 0 0 ..., 1 0 1]
```

References

- Modified from keras.

Nietzsche

`tensorlayer.files.`**`load_nietzsche_dataset`**`(`*path='data'*`)`

Load Nietzsche dataset.

Parameters **path** (*str*) – The path that the data is downloaded to, defaults is `data/nietzsche/`.

Returns The content.

Return type str

Examples

```
>>> see tutorial_generate_text.py
>>> words = tl.files.load_nietzsche_dataset()
>>> words = basic_clean_str(words)
>>> words = words.split()
```

English-to-French translation data from the WMT'15 Website

`tensorlayer.files.`**`load_wmt_en_fr_dataset`**`(`*path='data'*`)`

Load WMT'15 English-to-French translation dataset.

It will download the data from the WMT'15 Website (10^9-French-English corpus), and the 2013 news test from the same site as development set. Returns the directories of training data and test data.

Parameters **path** (*str*) – The path that the data is downloaded to, defaults is `data/wmt_en_fr/`.

References

- Code modified from /tensorflow/models/rnn/translation/data_utils.py

Notes

Usually, it will take a long time to download this dataset.

Flickr25k

`tensorlayer.files.`**`load_flickr25k_dataset`**`(`*tag='sky'*, *path='data'*, *n_threads=50*, *printable=False*`)`

Load Flickr25K dataset.

Returns a list of images by a given tag from Flick25k dataset, it will download Flickr25k from the official website at the first time you use it.

Parameters

- **tag** (*str or None*) –

  What images to return.

  – If you want to get images with tag, use string like 'dog', 'red', see Flickr Search.

  – If you want to get all images, set to `None`.

- **path** (*str*) – The path that the data is downloaded to, defaults is `data/flickr25k/`.
- **n_threads** (*int*) – The number of thread to read image.
- **printable** (*boolean*) – Whether to print infomation when reading images, default is `False`.

Examples

Get images with tag of sky

```
>>> images = tl.files.load_flickr25k_dataset(tag='sky')
```

Get all images

```
>>> images = tl.files.load_flickr25k_dataset(tag=None, n_threads=100,␣
↪printable=True)
```

Flickr1M

`tensorlayer.files.load_flickr1M_dataset`(*tag='sky'*, *size=10*, *path='data'*, *n_threads=50*, *printable=False*)

Load Flick1M dataset.

Returns a list of images by a given tag from Flickr1M dataset, it will download Flickr1M from the official website at the first time you use it.

Parameters

- **tag** (*str or None*) –

  What images to return.

  - If you want to get images with tag, use string like 'dog', 'red', see Flickr Search.
  - If you want to get all images, set to `None`.
- **size** (*int*) – integer between 1 to 10. 1 means 100k images . . . 5 means 500k images, 10 means all 1 million images. Default is 10.
- **path** (*str*) – The path that the data is downloaded to, defaults is `data/flickr25k/`.
- **n_threads** (*int*) – The number of thread to read image.
- **printable** (*boolean*) – Whether to print infomation when reading images, default is `False`.

Examples

Use 200k images

```
>>> images = tl.files.load_flickr1M_dataset(tag='zebra', size=2)
```

Use 1 Million images

```
>>> images = tl.files.load_flickr1M_dataset(tag='zebra')
```

CycleGAN

`tensorlayer.files.load_cyclegan_dataset` (*filename='summer2winter_yosemite'*, *path='data'*)

Load images from CycleGAN's database, see this link.

Parameters

- **filename** (*str*) – The dataset you want, see this link.
- **path** (*str*) – The path that the data is downloaded to, defaults is *data/cyclegan*

Examples

```
>>> im_train_A, im_train_B, im_test_A, im_test_B = load_cyclegan_dataset(filename=
↪'summer2winter_yosemite')
```

CelebA

`tensorlayer.files.load_celebA_dataset` (*path='data'*)

Load CelebA dataset

Return a list of image path.

Parameters **path** (*str*) – The path that the data is downloaded to, defaults is `data/celebA/`.

VOC 2007/2012

`tensorlayer.files.load_voc_dataset` (*path='data'*, *dataset='2012'*, *contain_classes_in_person=False*)

Pascal VOC 2007/2012 Dataset.

It has 20 objects: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor and additional 3 classes : head, hand, foot for person.

Parameters

- **path** (*str*) – The path that the data is downloaded to, defaults is `data/VOC`.
- **dataset** (*str*) – The VOC dataset version, *2012*, *2007*, *2007test* or *2012test*. We usually train model on *2007+2012* and test it on *2007test*.
- **contain_classes_in_person** (*boolean*) – Whether include head, hand and foot annotation, default is False.

Returns

- **imgs_file_list** (*list of str*) – Full paths of all images.
- **imgs_semseg_file_list** (*list of str*) – Full paths of all maps for semantic segmentation. Note that not all images have this map!
- **imgs_insseg_file_list** (*list of str*) – Full paths of all maps for instance segmentation. Note that not all images have this map!
- **imgs_ann_file_list** (*list of str*) – Full paths of all annotations for bounding box and object class, all images have this annotations.
- **classes** (*list of str*) – Classes in order.

- **classes_in_person** (*list of str*) – Classes in person.
- **classes_dict** (*dictionary*) – Class label to integer.
- **n_objs_list** (*list of int*) – Number of objects in all images in `imgs_file_list` in order.
- **objs_info_list** (*list of str*) – Darknet format for the annotation of all images in `imgs_file_list` in order. `[class_id x_centre y_centre width height]` in ratio format.
- **objs_info_dicts** (*dictionary*) – The annotation of all images in `imgs_file_list`, `{imgs_file_list : dictionary for annotation}`, format from TensorFlow/Models/object-detection.

Examples

```
>>> imgs_file_list, imgs_semseg_file_list, imgs_insseg_file_list, imgs_ann_file_
↪list,
>>>     classes, classes_in_person, classes_dict,
>>>     n_objs_list, objs_info_list, objs_info_dicts = tl.files.load_voc_
↪dataset(dataset="2012", contain_classes_in_person=False)
>>> idx = 26
>>> print(classes)
['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair',
↪'cow', 'diningtable', 'dog', 'horse', 'motorbike', 'person', 'pottedplant',
↪'sheep', 'sofa', 'train', 'tvmonitor']
>>> print(classes_dict)
{'sheep': 16, 'horse': 12, 'bicycle': 1, 'bottle': 4, 'cow': 9, 'sofa': 17, 'car
↪': 6, 'dog': 11, 'cat': 7, 'person': 14, 'train': 18, 'diningtable': 10,
↪'aeroplane': 0, 'bus': 5, 'pottedplant': 15, 'tvmonitor': 19, 'chair': 8, 'bird
↪': 2, 'boat': 3, 'motorbike': 13}
>>> print(imgs_file_list[idx])
data/VOC/VOC2012/JPEGImages/2007_000423.jpg
>>> print(n_objs_list[idx])
2
>>> print(imgs_ann_file_list[idx])
data/VOC/VOC2012/Annotations/2007_000423.xml
>>> print(objs_info_list[idx])
14 0.173 0.461333333333 0.142 0.496
14 0.828 0.542666666667 0.188 0.594666666667
>>> ann = tl.prepro.parse_darknet_ann_str_to_list(objs_info_list[idx])
>>> print(ann)
[[14, 0.173, 0.461333333333, 0.142, 0.496], [14, 0.828, 0.542666666667, 0.188, 0.
↪594666666667]]
>>> c, b = tl.prepro.parse_darknet_ann_list_to_cls_box(ann)
>>> print(c, b)
[14, 14] [[0.173, 0.461333333333, 0.142, 0.496], [0.828, 0.542666666667, 0.188, 0.
↪594666666667]]
```

References

- Pascal VOC2012 Website.
- Pascal VOC2007 Website.

MPII

`tensorlayer.files.load_mpii_pose_dataset`(*path='data'*, *is_16_pos_only=False*)

Load MPII Human Pose Dataset.

Parameters

- **path** (*str*) – The path that the data is downloaded to.
- **is_16_pos_only** (*boolean*) – If True, only return the peoples contain 16 pose keypoints. (Usually be used for single person pose estimation)

Returns

- **img_train_list** (*list of str*) – The image directories of training data.
- **ann_train_list** (*list of dict*) – The annotations of training data.
- **img_test_list** (*list of str*) – The image directories of testing data.
- **ann_test_list** (*list of dict*) – The annotations of testing data.

Examples

```
>>> import pprint
>>> import tensorlayer as tl
>>> img_train_list, ann_train_list, img_test_list, ann_test_list = tl.files.load_
↪mpii_pose_dataset()
>>> image = tl.vis.read_image(img_train_list[0])
>>> tl.vis.draw_mpii_pose_to_image(image, ann_train_list[0], 'image.png')
>>> pprint.pprint(ann_train_list[0])
```

References

- MPII Human Pose Dataset. CVPR 14
- MPII Human Pose Models. CVPR 16
- MPII Human Shape, Poselet Conditioned Pictorial Structures and etc
- MPII Keyponts and ID

Google Drive

`tensorlayer.files.download_file_from_google_drive`(*ID*, *destination*)

Download file from Google Drive.

See `tl.files.load_celebA_dataset` for example.

Parameters

- **ID** (*str*) – The driver ID.
- **destination** (*str*) – The destination for save file.

2.5.2 Load and save network

TensorFlow provides `.ckpt` file format to save and restore the models, while we suggest to use standard python file format `hdf5` to save models for the sake of cross-platform. Other file formats such as `.npz` are also available.

```
## save model as .h5
tl.files.save_weights_to_hdf5('model.h5', network.all_weights)
# restore model from .h5 (in order)
tl.files.load_hdf5_to_weights_in_order('model.h5', network.all_weights)
# restore model from .h5 (by name)
tl.files.load_hdf5_to_weights('model.h5', network.all_weights)

## save model as .npz
tl.files.save_npz(network.all_weights , name='model.npz')
# restore model from .npz (method 1)
load_params = tl.files.load_npz(name='model.npz')
tl.files.assign_weights(sess, load_params, network)
# restore model from .npz (method 2)
tl.files.load_and_assign_npz(sess=sess, name='model.npz', network=network)

## you can assign the pre-trained parameters as follow
# 1st parameter
tl.files.assign_weights(sess, [load_params[0]], network)
# the first three parameters
tl.files.assign_weights(sess, load_params[:3], network)
```

Save network into list (npz)

`tensorlayer.files.`**`save_npz`**(*save_list=None*, *name='model.npz'*)

Input parameters and the file name, save parameters into .npz file. Use tl.utils.load_npz() to restore.

Parameters

- **save_list** (*list of tensor*) – A list of parameters (tensor) to be saved.
- **name** (*str*) – The name of the *.npz* file.

Examples

Save model to npz

```
>>> tl.files.save_npz(network.all_weights, name='model.npz')
```

Load model from npz (Method 1)

```
>>> load_params = tl.files.load_npz(name='model.npz')
>>> tl.files.assign_weights(load_params, network)
```

Load model from npz (Method 2)

```
>>> tl.files.load_and_assign_npz(name='model.npz', network=network)
```

References

Saving dictionary using numpy

Load network from list (npz)

`tensorlayer.files.load_npz(path='', name='model.npz')`

Load the parameters of a Model saved by tl.files.save_npz().

Parameters

- **path** (*str*) – Folder path to *.npz* file.
- **name** (*str*) – The name of the *.npz* file.

Returns A list of parameters in order.

Return type list of array

Examples

- See `tl.files.save_npz`

References

- Saving dictionary using numpy

Assign a list of parameters to network

`tensorlayer.files.assign_weights(weights, network)`

Assign the given parameters to the TensorLayer network.

Parameters

- **weights** (*list of array*) – A list of model weights (array) in order.
- **network** (`Layer`) – The network to be assigned.

Returns

- *1) list of operations if in graph mode* – A list of tf ops in order that assign weights. Support sess.run(ops) manually.
- *2) list of tf variables if in eager mode* – A list of tf variables (assigned weights) in order.

Examples

References

- Assign value to a TensorFlow variable

Load and assign a list of parameters to network

`tensorlayer.files.load_and_assign_npz(name=None, network=None)`

Load model from npz and assign to a network.

Parameters

- **name** (*str*) – The name of the *.npz* file.
- **network** (`Model`) – The network to be assigned.

Examples

- See `tl.files.save_npz`

Save network into dict (npz)

`tensorlayer.files.save_npz_dict(save_list=None, name='model.npz')`

Input parameters and the file name, save parameters as a dictionary into .npz file.

Use `tl.files.load_and_assign_npz_dict()` to restore.

Parameters

- **save_list** (*list of parameters*) – A list of parameters (tensor) to be saved.
- **name** (*str*) – The name of the *.npz* file.

Load network from dict (npz)

`tensorlayer.files.load_and_assign_npz_dict(name='model.npz', network=None, skip=False)`

Restore the parameters saved by `tl.files.save_npz_dict()`.

Parameters

- **name** (*str*) – The name of the *.npz* file.
- **network** (`Model`) – The network to be assigned.
- **skip** (*boolean*) – If 'skip' == True, loaded weights whose name is not found in network's weights will be skipped. If 'skip' is False, error will be raised when mismatch is found. Default False.

Save network into OrderedDict (hdf5)

`tensorlayer.files.save_weights_to_hdf5(filepath, network)`

Input filepath and save weights in hdf5 format.

Parameters

- **filepath** (*str*) – Filename to which the weights will be saved.
- **network** (`Model`) – TL model.

Load network from hdf5 in order

`tensorlayer.files.load_hdf5_to_weights_in_order(filepath, network)`

Load weights sequentially from a given file of hdf5 format

Parameters

- **filepath** (*str*) – Filename to which the weights will be loaded, should be of hdf5 format.
- **network** (`Model`) – TL model.
- **Notes** – If the file contains more weights than given 'weights', then the redundant ones will be ignored if all previous weights match perfectly.

Load network from hdf5 by name

`tensorlayer.files.load_hdf5_to_weights(filepath, network, skip=False)`

Load weights by name from a given file of hdf5 format

Parameters

- **filepath** (*str*) – Filename to which the weights will be loaded, should be of hdf5 format.
- **network** (*Model*) – TL model.
- **skip** (*bool*) – If 'skip' == True, loaded weights whose name is not found in 'weights' will be skipped. If 'skip' is False, error will be raised when mismatch is found. Default False.

2.5.3 Load and save variables

Save variables as .npy

`tensorlayer.files.save_any_to_npy(save_dict=None, name='file.npy')`

Save variables to *.npy* file.

Parameters

- **save_dict** (*directory*) – The variables to be saved.
- **name** (*str*) – File name.

Examples

```
>>> tl.files.save_any_to_npy(save_dict={'data': ['a','b']}, name='test.npy')
>>> data = tl.files.load_npy_to_any(name='test.npy')
>>> print(data)
{'data': ['a','b']}
```

Load variables from .npy

`tensorlayer.files.load_npy_to_any(path='', name='file.npy')`

Load *.npy* file.

Parameters

- **path** (*str*) – Path to the file (optional).
- **name** (*str*) – File name.

Examples

- see tl.files.save_any_to_npy()

2.5.4 Folder/File functions

Check file exists

`tensorlayer.files.file_exists(filepath)`
Check whether a file exists by given file path.

Check folder exists

`tensorlayer.files.folder_exists(folderpath)`
Check whether a folder exists by given folder path.

Delete file

`tensorlayer.files.del_file(filepath)`
Delete a file by given file path.

Delete folder

`tensorlayer.files.del_folder(folderpath)`
Delete a folder by given folder path.

Read file

`tensorlayer.files.read_file(filepath)`
Read a file and return a string.

Examples

```
>>> data = tl.files.read_file('data.txt')
```

Load file list from folder

`tensorlayer.files.load_file_list(path=None, regx='\\.jpg', printable=True, keep_prefix=False)`
Return a file list in a folder by given a path and regular expression.

Parameters

- **path** (*str or None*) – A folder path, if *None*, use the current directory.
- **regx** (*str*) – The regx of file name.
- **printable** (*boolean*) – Whether to print the files infomation.
- **keep_prefix** (*boolean*) – Whether to keep path in the file name.

Examples

```
>>> file_list = tl.files.load_file_list(path=None, regx='w1pre_[0-9]+\.(npz)')
```

Load folder list from folder

`tensorlayer.files.load_folder_list(path='')`

Return a folder list in a folder by given a folder path.

Parameters **path** (*str*) – A folder path.

Check and Create folder

`tensorlayer.files.exists_or_mkdir(path, verbose=True)`

Check a folder by given name, if not exist, create the folder and return False, if directory exists, return True.

Parameters

- **path** (*str*) – A folder path.
- **verbose** (*boolean*) – If True (default), prints results.

Returns True if folder already exist, otherwise, returns False and create the folder.

Return type boolean

Examples

```
>>> tl.files.exists_or_mkdir("checkpoints/train")
```

Download or extract

`tensorlayer.files.maybe_download_and_extract(filename, working_directory, url_source, extract=False, expected_bytes=None)`

Checks if file exists in working_directory otherwise tries to dowload the file, and optionally also tries to extract the file if format is ".zip" or ".tar"

Parameters

- **filename** (*str*) – The name of the (to be) dowloaded file.
- **working_directory** (*str*) – A folder path to search for the file in and dowload the file to
- **url** (*str*) – The URL to download the file from
- **extract** (*boolean*) – If True, tries to uncompress the dowloaded file is ".tar.gz/.tar.bz2" or ".zip" file, default is False.
- **expected_bytes** (*int or None*) – If set tries to verify that the downloaded file is of the specified size, otherwise raises an Exception, defaults is None which corresponds to no check being performed.

Returns File path of the dowloaded (uncompressed) file.

Return type str

Examples

```
>>> down_file = tl.files.maybe_download_and_extract(filename='train-images-idx3-
↪ubyte.gz',
...                                                working_directory='data/',
...                                                url_source='http://yann.lecun.com/
↪exdb/mnist/')
>>> tl.files.maybe_download_and_extract(filename='ADEChallengeData2016.zip',
...                                             working_directory='data/',
...                                             url_source='http://sceneparsing.
↪csail.mit.edu/data/',
...                                             extract=True)
```

2.5.5 Sort

List of string with number in human order

`tensorlayer.files.`**`natural_keys`**(*text*)

Sort list of string with number in human order.

Examples

```
>>> l = ['im1.jpg', 'im31.jpg', 'im11.jpg', 'im21.jpg', 'im03.jpg', 'im05.jpg']
>>> l.sort(key=tl.files.natural_keys)
['im1.jpg', 'im03.jpg', 'im05', 'im11.jpg', 'im21.jpg', 'im31.jpg']
>>> l.sort() # that is what we dont want
['im03.jpg', 'im05', 'im1.jpg', 'im11.jpg', 'im21.jpg', 'im31.jpg']
```

References

- link

2.5.6 Visualizing npz file

`tensorlayer.files.`**`npz_to_W_pdf`**(*path=None*, *regx='w1pre_[0-9]+\\.(npz)'*)

Convert the first weight matrix of *.npz* file to *.pdf* by using *tl.visualize.W()*.

Parameters

- **path** (*str*) – A folder path to *npz* files.
- **regx** (*str*) – Regx for the file name.

Examples

Convert the first weight matrix of w1_pre…npz file to w1_pre…pdf.

```
>>> tl.files.npz_to_W_pdf(path='/Users/.../npz_file/', regx='w1pre_[0-9]+\.(npz)')
```

2.6 API - Iteration

Data iteration.

minibatches([inputs, targets, batch_size, …])	Generate a generator that input a group of example in numpy.array and their labels, return the examples and labels by the given batch size.
seq_minibatches(inputs, targets, batch_size, …)	Generate a generator that return a batch of sequence inputs and targets.
seq_minibatches2(inputs, targets, …)	Generate a generator that iterates on two list of words.
ptb_iterator(raw_data, batch_size, num_steps)	Generate a generator that iterates on a list of words, see PTB example.

2.6.1 Non-time series

tensorlayer.iterate.**minibatches**(*inputs=None*, *targets=None*, *batch_size=None*, *allow_dynamic_batch_size=False*, *shuffle=False*)

Generate a generator that input a group of example in numpy.array and their labels, return the examples and labels by the given batch size.

Parameters

- **inputs** (*numpy.array*) – The input features, every row is a example.
- **targets** (*numpy.array*) – The labels of inputs, every row is a example.
- **batch_size** (*int*) – The batch size.
- **allow_dynamic_batch_size** (*boolean*) – Allow the use of the last data batch in case the number of examples is not a multiple of batch_size, this may result in unexpected behaviour if other functions expect a fixed-sized batch-size.
- **shuffle** (*boolean*) – Indicating whether to use a shuffling queue, shuffle the dataset before return.

Examples

```
>>> X = np.asarray([['a','a'], ['b','b'], ['c','c'], ['d','d'], ['e','e'], ['f','f
↪']])
>>> y = np.asarray([0,1,2,3,4,5])
>>> for batch in tl.iterate.minibatches(inputs=X, targets=y, batch_size=2,␣
↪shuffle=False):
>>>     print(batch)
(array([['a', 'a'], ['b', 'b']], dtype='<U1'), array([0, 1]))
(array([['c', 'c'], ['d', 'd']], dtype='<U1'), array([2, 3]))
(array([['e', 'e'], ['f', 'f']], dtype='<U1'), array([4, 5]))
```

Notes

If you have two inputs and one label and want to shuffle them together, e.g. X1 (1000, 100), X2 (1000, 80) and Y (1000, 1), you can stack them together (*np.hstack((X1, X2))*) into (1000, 180) and feed to `inputs`. After getting a batch, you can split it back into X1 and X2.

2.6.2 Time series

Sequence iteration 1

tensorlayer.iterate.**seq_minibatches**(*inputs*, *targets*, *batch_size*, *seq_length*, *stride=1*)
Generate a generator that return a batch of sequence inputs and targets. If *batch_size=100* and *seq_length=5*, one return will have 500 rows (examples).

Parameters

- **inputs** (*numpy.array*) – The input features, every row is a example.
- **targets** (*numpy.array*) – The labels of inputs, every element is a example.
- **batch_size** (*int*) – The batch size.
- **seq_length** (*int*) – The sequence length.
- **stride** (*int*) – The stride step, default is 1.

Examples

Synced sequence input and output.

```
>>> X = np.asarray([['a','a'], ['b','b'], ['c','c'], ['d','d'], ['e','e'], ['f','f
↪']])
>>> y = np.asarray([0, 1, 2, 3, 4, 5])
>>> for batch in tl.iterate.seq_minibatches(inputs=X, targets=y, batch_size=2,␣
↪seq_length=2, stride=1):
>>>     print(batch)
(array([['a', 'a'], ['b', 'b'], ['b', 'b'], ['c', 'c']], dtype='<U1'), array([0,␣
↪1, 1, 2]))
(array([['c', 'c'], ['d', 'd'], ['d', 'd'], ['e', 'e']], dtype='<U1'), array([2,␣
↪3, 3, 4]))
```

Many to One

```
>>> return_last = True
>>> num_steps = 2
>>> X = np.asarray([['a','a'], ['b','b'], ['c','c'], ['d','d'], ['e','e'], ['f','f
↪']])
>>> Y = np.asarray([0,1,2,3,4,5])
>>> for batch in tl.iterate.seq_minibatches(inputs=X, targets=Y, batch_size=2,␣
↪seq_length=num_steps, stride=1):
>>>     x, y = batch
>>>     if return_last:
>>>         tmp_y = y.reshape((-1, num_steps) + y.shape[1:])
>>>     y = tmp_y[:, -1]
>>>     print(x, y)
[['a' 'a']
['b' 'b']
['b' 'b']
['c' 'c']] [1 2]
[['c' 'c']
['d' 'd']
['d' 'd']
['e' 'e']] [3 4]
```

Sequence iteration 2

`tensorlayer.iterate.`**`seq_minibatches2`**(*inputs*, *targets*, *batch_size*, *num_steps*)

Generate a generator that iterates on two list of words. Yields (Returns) the source contexts and the target context by the given batch_size and num_steps (sequence_length). In TensorFlow's tutorial, this generates the *batch_size* pointers into the raw PTB data, and allows minibatch iteration along these pointers.

Parameters

- **inputs** (*list of data*) – The context in list format; note that context usually be represented by splitting by space, and then convert to unique word IDs.
- **targets** (*list of data*) – The context in list format; note that context usually be represented by splitting by space, and then convert to unique word IDs.
- **batch_size** (*int*) – The batch size.
- **num_steps** (*int*) – The number of unrolls. i.e. sequence length

Yields *Pairs of the batched data, each a matrix of shape [batch_size, num_steps].*

:raises ValueError : if batch_size or num_steps are too high.:

Examples

```
>>> X = [i for i in range(20)]
>>> Y = [i for i in range(20,40)]
>>> for batch in tl.iterate.seq_minibatches2(X, Y, batch_size=2, num_steps=3):
...     x, y = batch
...     print(x, y)
```

[[0. 1. 2.] [10. 11. 12.]] [[20. 21. 22.] [30. 31. 32.]]

[[3. 4. 5.] [13. 14. 15.]] [[23. 24. 25.] [33. 34. 35.]]

[[6. 7. 8.] [16. 17. 18.]] [[26. 27. 28.] [36. 37. 38.]]

Notes

- Hint, if the input data are images, you can modify the source code *data = np.zeros([batch_size, batch_len)* to *data = np.zeros([batch_size, batch_len, inputs.shape[1], inputs.shape[2], inputs.shape[3]]).*

PTB dataset iteration

`tensorlayer.iterate.`**`ptb_iterator`**(*raw_data*, *batch_size*, *num_steps*)

Generate a generator that iterates on a list of words, see PTB example. Yields the source contexts and the target context by the given batch_size and num_steps (sequence_length).

In TensorFlow's tutorial, this generates *batch_size* pointers into the raw PTB data, and allows minibatch iteration along these pointers.

Parameters

- **raw_data** (*a list*) – the context in list format; note that context usually be represented by splitting by space, and then convert to unique word IDs.
- **batch_size** (*int*) – the batch size.
- **num_steps** (*int*) – the number of unrolls. i.e. sequence_length

Yields

- *Pairs of the batched data, each a matrix of shape [batch_size, num_steps].*
- *The second element of the tuple is the same data time-shifted to the*
- *right by one.*

:raises ValueError : if batch_size or num_steps are too high.:

Examples

```
>>> train_data = [i for i in range(20)]
>>> for batch in tl.iterate.ptb_iterator(train_data, batch_size=2, num_steps=3):
>>>     x, y = batch
>>>     print(x, y)
[[ 0  1  2] <---x                          1st subset/ iteration
 [10 11 12]]
[[ 1  2  3] <---y
 [11 12 13]]
```

[[3 4 5] <— 1st batch input 2nd subset/ iteration [13 14 15]] <— 2nd batch input

[[4 5 6] <— 1st batch target [14 15 16]] <— 2nd batch target

[[6 7 8] 3rd subset/ iteration [16 17 18]]

[[7 8 9] [17 18 19]]

2.7 API - Layers

TensorLayer provides rich layer implementations trailed for various benchmarks and domain-specific problems. In addition, we also support transparent access to native TensorFlow parameters. For example, we provide not only layers for local response normalization, but also layers that allow user to apply `tf.nn.lrn` on `network.outputs`. More functions can be found in TensorFlow API.

2.7.1 Layer list

`Layer`([name])	The basic `Layer` class represents a single layer of a neural network.
`Input`(shape[, dtype, name])	The `Input` class is the starting layer of a neural network.
`OneHot`([depth, on_value, off_value, axis, ...])	The `OneHot` class is the starting layer of a neural network, see `tf.one_hot`.
`Word2vecEmbedding`(vocabulary_size, ...[, ...])	The `Word2vecEmbedding` class is a fully connected layer.
`Embedding`(vocabulary_size, embedding_size[, ...])	The `Embedding` class is a look-up table for word embedding.
`AverageEmbedding`(vocabulary_size, embedding_size)	The `AverageEmbedding` averages over embeddings of inputs.
`Dense`(n_units[, act, W_init, b_init, ...])	The `Dense` class is a fully connected layer.

Continued on next page

Table 7 – continued from previous page

Dropout(keep[, seed, name])	The *Dropout* class is a noise layer which randomly set some activations to zero according to a keeping probability.
GaussianNoise([mean, stddev, is_train, . . .])	The *GaussianNoise* class is noise layer that adding noise with gaussian distribution to the activation.
DropconnectDense([keep, n_units, act, . . .])	The *DropconnectDense* class is *Dense* with DropConnect behaviour which randomly removes connections between this layer and the previous layer according to a keeping probability.
UpSampling2d(scale[, method, antialias, . . .])	The *UpSampling2d* class is a up-sampling 2D layer.
DownSampling2d(scale[, method, antialias, . . .])	The *DownSampling2d* class is down-sampling 2D layer.
Conv1d([n_filter, filter_size, stride, act, . . .])	Simplified version of `Conv1dLayer`.
Conv2d([n_filter, filter_size, strides, . . .])	Simplified version of `Conv2dLayer`.
Conv3d([n_filter, filter_size, strides, . . .])	Simplified version of `Conv3dLayer`.
DeConv2d([n_filter, filter_size, strides, . . .])	Simplified version of `DeConv2dLayer`, see tf.nn.conv3d_transpose.
DeConv3d([n_filter, filter_size, strides, . . .])	Simplified version of `DeConv3dLayer`, see tf.nn.conv3d_transpose.
DepthwiseConv2d([filter_size, strides, act, . . .])	Separable/Depthwise Convolutional 2D layer, see tf.nn.depthwise_conv2d.
SeparableConv1d([n_filter, filter_size, . . .])	The *SeparableConv1d* class is a 1D depthwise separable convolutional layer.
SeparableConv2d([n_filter, filter_size, . . .])	The *SeparableConv2d* class is a 2D depthwise separable convolutional layer.
DeformableConv2d([offset_layer, n_filter, . . .])	The *DeformableConv2d* class is a 2D Deformable Convolutional Networks.
GroupConv2d([n_filter, filter_size, . . .])	The *GroupConv2d* class is 2D grouped convolution, see here.
PadLayer([padding, mode, name])	The *PadLayer* class is a padding layer for any mode and dimension.
PoolLayer([filter_size, strides, padding, . . .])	The *PoolLayer* class is a Pooling layer.
ZeroPad1d(padding[, name])	The *ZeroPad1d* class is a 1D padding layer for signal [batch, length, channel].
ZeroPad2d(padding[, name])	The *ZeroPad2d* class is a 2D padding layer for image [batch, height, width, channel].
ZeroPad3d(padding[, name])	The *ZeroPad3d* class is a 3D padding layer for volume [batch, depth, height, width, channel].
MaxPool1d([filter_size, strides, padding, . . .])	Max pooling for 1D signal.
MeanPool1d([filter_size, strides, padding, . . .])	Mean pooling for 1D signal.
MaxPool2d([filter_size, strides, padding, . . .])	Max pooling for 2D image.
MeanPool2d([filter_size, strides, padding, . . .])	Mean pooling for 2D image [batch, height, width, channel].
MaxPool3d([filter_size, strides, padding, . . .])	Max pooling for 3D volume.
MeanPool3d([filter_size, strides, padding, . . .])	Mean pooling for 3D volume.
GlobalMaxPool1d([data_format, name])	The *GlobalMaxPool1d* class is a 1D Global Max Pooling layer.
GlobalMeanPool1d([data_format, name])	The *GlobalMeanPool1d* class is a 1D Global Mean Pooling layer.
GlobalMaxPool2d([data_format, name])	The *GlobalMaxPool2d* class is a 2D Global Max Pooling layer.

Continued on next page

Table 7 – continued from previous page

GlobalMeanPool2d([data_format, name])	The GlobalMeanPool2d class is a 2D Global Mean Pooling layer.
GlobalMaxPool3d([data_format, name])	The GlobalMaxPool3d class is a 3D Global Max Pooling layer.
GlobalMeanPool3d([data_format, name])	The GlobalMeanPool3d class is a 3D Global Mean Pooling layer.
CornerPool2d([mode, name])	Corner pooling for 2D image [batch, height, width, channel], see here.
SubpixelConv1d([scale, act, in_channels, name])	It is a 1D sub-pixel up-sampling layer.
SubpixelConv2d([scale, n_out_channels, act, ...])	It is a 2D sub-pixel up-sampling layer, usually be used for Super-Resolution applications, see SRGAN for example.
SpatialTransformer2dAffine([in_channels, ...])	The SpatialTransformer2dAffine class is a 2D Spatial Transformer Layer for 2D Affine Transformation.
transformer(U, theta, out_size[, name])	Spatial Transformer Layer for 2D Affine Transformation , see SpatialTransformer2dAffine class.
batch_transformer(U, thetas, out_size[, name])	Batch Spatial Transformer function for 2D Affine Transformation.
BatchNorm([decay, epsilon, act, is_train, ...])	The BatchNorm is a batch normalization layer for both fully-connected and convolution outputs.
BatchNorm1d([decay, epsilon, act, is_train, ...])	The BatchNorm1d applies Batch Normalization over 3D input (a mini-batch of 1D inputs with additional channel dimension), of shape (N, L, C) or (N, C, L).
BatchNorm2d([decay, epsilon, act, is_train, ...])	The BatchNorm2d applies Batch Normalization over 4D input (a mini-batch of 2D inputs with additional channel dimension) of shape (N, H, W, C) or (N, C, H, W).
BatchNorm3d([decay, epsilon, act, is_train, ...])	The BatchNorm3d applies Batch Normalization over 5D input (a mini-batch of 3D inputs with additional channel dimension) with shape (N, D, H, W, C) or (N, C, D, H, W).
LocalResponseNorm([depth_radius, bias, ...])	The LocalResponseNorm layer is for Local Response Normalization.
InstanceNorm([act, epsilon, beta_init, ...])	The InstanceNorm is an instance normalization layer for both fully-connected and convolution outputs.
InstanceNorm1d([act, epsilon, beta_init, ...])	The InstanceNorm1d applies Instance Normalization over 3D input (a mini-instance of 1D inputs with additional channel dimension), of shape (N, L, C) or (N, C, L).
InstanceNorm2d([act, epsilon, beta_init, ...])	The InstanceNorm2d applies Instance Normalization over 4D input (a mini-instance of 2D inputs with additional channel dimension) of shape (N, H, W, C) or (N, C, H, W).
InstanceNorm3d([act, epsilon, beta_init, ...])	The InstanceNorm3d applies Instance Normalization over 5D input (a mini-instance of 3D inputs with additional channel dimension) with shape (N, D, H, W, C) or (N, C, D, H, W).
LayerNorm([center, scale, act, epsilon, ...])	The LayerNorm class is for layer normalization, see tf.contrib.layers.layer_norm.
GroupNorm([groups, epsilon, act, ...])	The GroupNorm layer is for Group Normalization.

Continued on next page

Table 7 – continued from previous page

SwitchNorm([act, epsilon, beta_init, ...])	The SwitchNorm is a switchable normalization.
RNN(cell[, return_last_output, ...])	The RNN class is a fixed length recurrent layer for implementing simple RNN, LSTM, GRU and etc.
BiRNN(fw_cell, bw_cell[, return_seq_2d, ...])	The BiRNN class is a fixed length Bidirectional recurrent layer.
retrieve_seq_length_op(data)	An op to compute the length of a sequence from input shape of [batch_size, n_step(max), n_features], it can be used when the features of padding (on right hand side) are all zeros.
retrieve_seq_length_op2(data)	An op to compute the length of a sequence, from input shape of [batch_size, n_step(max)], it can be used when the features of padding (on right hand side) are all zeros.
retrieve_seq_length_op3(data[, pad_val])	An op to compute the length of a sequence, the data shape can be [batch_size, n_step(max)] or [batch_size, n_step(max), n_features].
Flatten([name])	A layer that reshapes high-dimension input into a vector.
Reshape(shape[, name])	A layer that reshapes a given tensor.
Transpose([perm, conjugate, name])	A layer that transposes the dimension of a tensor.
Shuffle(group[, name])	A layer that shuffle a 2D image [batch, height, width, channel], see here.
Lambda(fn[, fn_weights, fn_args, name])	A layer that takes a user-defined function using Lambda.
Concat([concat_dim, name])	A layer that concats multiple tensors according to given axis.
Elementwise([combine_fn, act, name])	A layer that combines multiple Layer that have the same output shapes according to an element-wise operation.
ElementwiseLambda(fn[, fn_weights, fn_args, ...])	A layer that use a custom function to combine multiple Layer inputs.
ExpandDims(axis[, name])	The ExpandDims class inserts a dimension of 1 into a tensor's shape, see tf.expand_dims() .
Tile([multiples, name])	The Tile class constructs a tensor by tiling a given tensor, see tf.tile() .
Stack([axis, name])	The Stack class is a layer for stacking a list of rank-R tensors into one rank-(R+1) tensor, see tf.stack().
UnStack([num, axis, name])	The UnStack class is a layer for unstacking the given dimension of a rank-R tensor into rank-(R-1) tensors., see tf.unstack().
Sign([name])	The SignLayer class is for quantizing the layer outputs to -1 or 1 while inferencing.
Scale([init_scale, name])	The Scale class is to multiple a trainable scale value to the layer outputs.
BinaryDense([n_units, act, use_gemm, ...])	The BinaryDense class is a binary fully connected layer, which weights are either -1 or 1 while inferencing.
BinaryConv2d([n_filter, filter_size, ...])	The BinaryConv2d class is a 2D binary CNN layer, which weights are either -1 or 1 while inference.
TernaryDense([n_units, act, use_gemm, ...])	The TernaryDense class is a ternary fully connected layer, which weights are either -1 or 1 or 0 while inference.
TernaryConv2d([n_filter, filter_size, ...])	The TernaryConv2d class is a 2D ternary CNN layer, which weights are either -1 or 1 or 0 while inference.

Continued on next page

Table 7 – continued from previous page

DorefaDense([bitW, bitA, n_units, act, ...])	The DorefaDense class is a binary fully connected layer, which weights are 'bitW' bits and the output of the previous layer are 'bitA' bits while inferencing.
DorefaConv2d([bitW, bitA, n_filter, ...])	The *DorefaConv2d* class is a 2D quantized convolutional layer, which weights are 'bitW' bits and the output of the previous layer are 'bitA' bits while inferencing.
PRelu([channel_shared, in_channels, a_init, ...])	The *PRelu* class is Parametric Rectified Linear layer.
PRelu6([channel_shared, in_channels, ...])	The *PRelu6* class is Parametric Rectified Linear layer integrating ReLU6 behaviour.
PTRelu6([channel_shared, in_channels, ...])	The *PTRelu6* class is Parametric Rectified Linear layer integrating ReLU6 behaviour.
flatten_reshape(variable[, name])	Reshapes a high-dimension vector input.
initialize_rnn_state(state[, feed_dict])	Returns the initialized RNN state.
list_remove_repeat(x)	Remove the repeated items in a list, and return the processed list.

2.7.2 Base Layer

class tensorlayer.layers.**Layer**(*name=None*, **args*, ***kwargs*)

The basic *Layer* class represents a single layer of a neural network.

It should be subclassed when implementing new types of layers.

Parameters name (*str or None*) – A unique layer name. If None, a unique name will be automatically assigned.

__init__()

Initializing the Layer.

__call__()

(1) Building the Layer if necessary. (2) Forwarding the computation.

all_weights()

Return a list of Tensor which are all weights of this Layer.

trainable_weights()

Return a list of Tensor which are all trainable weights of this Layer.

nontrainable_weights()

Return a list of Tensor which are all nontrainable weights of this Layer.

build()

Abstract method. Build the Layer. All trainable weights should be defined in this function.

forward()

Abstract method. Forward computation and return computation results.

2.7.3 Input Layers

Input Layer

tensorlayer.layers.**Input**(*shape*, *dtype=tensorflow.float32*, *name=None*)

The *Input* class is the starting layer of a neural network.

Parameters

- **shape** (*tuple (int)*) – Including batch size.
- **name** (*None or str*) – A unique layer name.

One-hot Layer

class tensorlayer.layers.**OneHot**(*depth=None*, *on_value=None*, *off_value=None*, *axis=None*, *dtype=None*, *name=None*)

The *OneHot* class is the starting layer of a neural network, see tf.one_hot. Useful link: *https://www.tensorflow.org/api_docs/python/tf/one_hot*.

Parameters

- **depth** (*None or int*) – If the input indices is rank N, the output will have rank N+1. The new axis is created at dimension *axis* (default: the new axis is appended at the end).
- **on_value** (*None or number*) – The value to represnt *ON*. If None, it will default to the value 1.
- **off_value** (*None or number*) – The value to represnt *OFF*. If None, it will default to the value 0.
- **axis** (*None or int*) – The axis.
- **dtype** (*None or TensorFlow dtype*) – The data type, None means tf.float32.
- **name** (*str*) – A unique layer name.

Examples

```
>>> import tensorflow as tf
>>> import tensorlayer as tl
>>> net = tl.layers.Input([32], dtype=tf.int32)
>>> onehot = tl.layers.OneHot(depth=8)
>>> print(onehot)
OneHot(depth=8, name='onehot')
>>> tensor = tl.layers.OneHot(depth=8)(net)
>>> print(tensor)
tf.Tensor([...], shape=(32, 8), dtype=float32)
```

Word2Vec Embedding Layer

class tensorlayer.layers.**Word2vecEmbedding**(*vocabulary_size*, *embedding_size*, *num_sampled=64*, *activate_nce_loss=True*, *nce_loss_args=None*, *E_init=<tensorlayer.initializers.RandomUniform object>*, *nce_W_init=<tensorlayer.initializers.TruncatedNormal object>*, *nce_b_init=<tensorlayer.initializers.Constant object>*, *name=None*)

The *Word2vecEmbedding* class is a fully connected layer. For Word Embedding, words are input as integer index. The output is the embedded word vector.

The layer integrates NCE loss by default (activate_nce_loss=True). If the NCE loss is activated, in a dynamic model, the computation of nce loss can be turned off in customised forward feeding by setting use_nce_loss=False when the layer is called. The NCE loss can be deactivated by setting activate_nce_loss=False.

Parameters

- **vocabulary_size** (*int*) – The size of vocabulary, number of words
- **embedding_size** (*int*) – The number of embedding dimensions
- **num_sampled** (*int*) – The number of negative examples for NCE loss
- **activate_nce_loss** (*boolean*) – Whether activate nce loss or not. By default, True If True, the layer will return both outputs of embedding and nce_cost in forward feeding. If False, the layer will only return outputs of embedding. In a dynamic model, the computation of nce loss can be turned off in forward feeding by setting use_nce_loss=False when the layer is called. In a static model, once the model is constructed, the computation of nce loss cannot be changed (always computed or not computed).
- **nce_loss_args** (*dictionary*) – The arguments for tf.nn.nce_loss()
- **E_init** (*initializer*) – The initializer for initializing the embedding matrix
- **nce_W_init** (*initializer*) – The initializer for initializing the nce decoder weight matrix
- **nce_b_init** (*initializer*) – The initializer for initializing of the nce decoder bias vector
- **name** (*str*) – A unique layer name

outputs

The embedding layer outputs.

Type Tensor

normalized_embeddings

Normalized embedding matrix.

Type Tensor

nce_weights

The NCE weights only when activate_nce_loss is True.

Type Tensor

nce_biases

The NCE biases only when activate_nce_loss is True.

Type Tensor

Examples

Word2Vec With TensorLayer (Example in *examples/text_word_embedding/tutorial_word2vec_basic.py*)

```
>>> import tensorflow as tf
>>> import tensorlayer as tl
>>> batch_size = 8
>>> embedding_size = 50
>>> inputs = tl.layers.Input([batch_size], dtype=tf.int32)
>>> labels = tl.layers.Input([batch_size, 1], dtype=tf.int32)
>>> emb_net = tl.layers.Word2vecEmbedding(
>>>     vocabulary_size=10000,
>>>     embedding_size=embedding_size,
>>>     num_sampled=100,
>>>     activate_nce_loss=True, # the nce loss is activated
```

(continues on next page)

(continued from previous page)

```
>>>     nce_loss_args={},
>>>     E_init=tl.initializers.random_uniform(minval=-1.0, maxval=1.0),
>>>     nce_W_init=tl.initializers.truncated_normal(stddev=float(1.0 / np.
↪sqrt(embedding_size))),
>>>     nce_b_init=tl.initializers.constant(value=0.0),
>>>     name='word2vec_layer',
>>> )
>>> print(emb_net)
Word2vecEmbedding(vocabulary_size=10000, embedding_size=50, num_sampled=100,␣
↪activate_nce_loss=True, nce_loss_args={})
>>> embed_tensor = emb_net(inputs, use_nce_loss=False) # the nce loss is turned␣
↪off and no need to provide labels
>>> embed_tensor = emb_net([inputs, labels], use_nce_loss=False) # the nce loss␣
↪is turned off and the labels will be ignored
>>> embed_tensor, embed_nce_loss = emb_net([inputs, labels]) # the nce loss is␣
↪calculated
>>> outputs = tl.layers.Dense(n_units=10, name="dense")(embed_tensor)
>>> model = tl.models.Model(inputs=[inputs, labels], outputs=[outputs, embed_nce_
↪loss], name="word2vec_model") # a static model
>>> out = model([data_x, data_y], is_train=True) # where data_x is inputs and␣
↪data_y is labels
```

References

https://www.tensorflow.org/tutorials/representation/word2vec

Embedding Layer

class tensorlayer.layers.**Embedding**(*vocabulary_size*, *embedding_size*, *E_init=<tensorlayer.initializers.RandomUniform object>*, *name=None*)

The *Embedding* class is a look-up table for word embedding.

Word content are accessed using integer indexes, then the output is the embedded word vector. To train a word embedding matrix, you can used *Word2vecEmbedding*. If you have a pre-trained matrix, you can assign the parameters into it.

Parameters

- **vocabulary_size** (*int*) – The size of vocabulary, number of words.
- **embedding_size** (*int*) – The number of embedding dimensions.
- **E_init** (*initializer*) – The initializer for the embedding matrix.
- **E_init_args** (*dictionary*) – The arguments for embedding matrix initializer.
- **name** (*str*) – A unique layer name.

outputs

The embedding layer output is a 3D tensor in the shape: (batch_size, num_steps(num_words), embedding_size).

Type tensor

Examples

```
>>> import tensorflow as tf
>>> import tensorlayer as tl
>>> input = tl.layers.Input([8, 100], dtype=tf.int32)
>>> embed = tl.layers.Embedding(vocabulary_size=1000, embedding_size=50, name=
↪'embed')
>>> print(embed)
Embedding(vocabulary_size=1000, embedding_size=50)
>>> tensor = embed(input)
>>> print(tensor)
tf.Tensor([...], shape=(8, 100, 50), dtype=float32)
```

Average Embedding Layer

class tensorlayer.layers.**AverageEmbedding**(*vocabulary_size, embedding_size, pad_value=0, E_init=<tensorlayer.initializers.RandomUniform object>, name=None*)

The *AverageEmbedding* averages over embeddings of inputs. This is often used as the input layer for models like DAN[1] and FastText[2].

Parameters

- **vocabulary_size** (*int*) – The size of vocabulary.
- **embedding_size** (*int*) – The dimension of the embedding vectors.
- **pad_value** (*int*) – The scalar padding value used in inputs, 0 as default.
- **E_init** (*initializer*) – The initializer of the embedding matrix.
- **name** (*str*) – A unique layer name.

outputs

The embedding layer output is a 2D tensor in the shape: (batch_size, embedding_size).

Type tensor

References

- [1] Iyyer, M., Manjunatha, V., Boyd-Graber, J., & Daum'e III, H. (2015). Deep Unordered Composition Rivals Syntactic Methods for Text Classification. In Association for Computational Linguistics.
- [2] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification.

Examples

```
>>> import tensorflow as tf
>>> import tensorlayer as tl
>>> batch_size = 8
>>> length = 5
>>> input = tl.layers.Input([batch_size, length], dtype=tf.int32)
>>> avgembed = tl.layers.AverageEmbedding(vocabulary_size=1000, embedding_size=50,
↪ name='avg')
```

(continues on next page)

(continued from previous page)

```
>>> print(avgembed)
AverageEmbedding(vocabulary_size=1000, embedding_size=50, pad_value=0)
>>> tensor = avgembed(input)
>>> print(tensor)
tf.Tensor([...], shape=(8, 50), dtype=float32)
```

2.7.4 Activation Layers

PReLU Layer

class tensorlayer.layers.**PRelu**(*channel_shared=False*, *in_channels=None*, *a_init=<tensorlayer.initializers.TruncatedNormal object>*, *name=None*)

The *PRelu* class is Parametric Rectified Linear layer. It follows f(x) = alpha * x for x < 0, f(x) = x for x >= 0, where alpha is a learned array with the same shape as x.

Parameters

- **channel_shared** (*boolean*) – If True, single weight is shared by all channels.
- **in_channels** (*int*) – The number of channels of the previous layer. If None, it will be automatically detected when the layer is forwarded for the first time.
- **a_init** (*initializer*) – The initializer for initializing the alpha(s).
- **name** (*None or str*) – A unique layer name.

Examples

```
>>> inputs = tl.layers.Input([10, 5])
>>> prelulayer = tl.layers.PRelu(channel_shared=True)
>>> print(prelulayer)
PRelu(channel_shared=True,in_channels=None,name=prelu)
>>> prelu = prelulayer(inputs)
>>> model = tl.models.Model(inputs=inputs, outputs=prelu)
>>> out = model(data, is_train=True)
```

References

- Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
- Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

PReLU6 Layer

class tensorlayer.layers.**PRelu6**(*channel_shared=False*, *in_channels=None*, *a_init=<tensorlayer.initializers.TruncatedNormal object>*, *name=None*)

The *PRelu6* class is Parametric Rectified Linear layer integrating ReLU6 behaviour.

This Layer is a modified version of the *PRelu*.

This activation layer use a modified version `tl.act.leaky_relu()` introduced by the following paper: Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]

This activation function also use a modified version of the activation function `tf.nn.relu6()` introduced by the following paper: Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

This activation layer push further the logic by adding *leaky* behaviour both below zero and above six.

The function return the following results:

- When x < 0: `f(x) = alpha_low * x`.
- When x in [0, 6]: `f(x) = x`.
- When x > 6: `f(x) = 6`.

Parameters

- **channel_shared** (*boolean*) – If True, single weight is shared by all channels.
- **in_channels** (*int*) – The number of channels of the previous layer. If None, it will be automatically detected when the layer is forwarded for the first time.
- **a_init** (*initializer*) – The initializer for initializing the alpha(s).
- **name** (*None or str*) – A unique layer name.

References

- Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
- Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]
- Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

PTReLU6 Layer

class tensorlayer.layers.**PTRelu6**(*channel_shared=False*, *in_channels=None*, *a_init=<tensorlayer.initializers.TruncatedNormal object>*, *name=None*)

The `PTRelu6` class is Parametric Rectified Linear layer integrating ReLU6 behaviour.

This Layer is a modified version of the `PRelu`.

This activation layer use a modified version `tl.act.leaky_relu()` introduced by the following paper: Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]

This activation function also use a modified version of the activation function `tf.nn.relu6()` introduced by the following paper: Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]

This activation layer push further the logic by adding *leaky* behaviour both below zero and above six.

The function return the following results:

- When x < 0: `f(x) = alpha_low * x`.
- When x in [0, 6]: `f(x) = x`.
- When x > 6: `f(x) = 6 + (alpha_high * (x-6))`.

This version goes one step beyond `PRelu6` by introducing leaky behaviour on the positive side when x > 6.

Parameters

- **channel_shared** (*boolean*) – If True, single weight is shared by all channels.

- **in_channels** (*int*) – The number of channels of the previous layer. If None, it will be automatically detected when the layer is forwarded for the first time.
- **a_init** (*initializer*) – The initializer for initializing the alpha(s).
- **name** (*None or str*) – A unique layer name.

References

- Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
- Convolutional Deep Belief Networks on CIFAR-10 [A. Krizhevsky, 2010]
- Rectifier Nonlinearities Improve Neural Network Acoustic Models [A. L. Maas et al., 2013]

2.7.5 Convolutional Layers

Convolutions

Conv1d

class tensorlayer.layers.**Conv1d**(*n_filter=32, filter_size=5, stride=1, act=None, padding='SAME', data_format='channels_last', dilation_rate=1, W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

Simplified version of Conv1dLayer.

Parameters

- **n_filter** (*int*) – The number of filters
- **filter_size** (*int*) – The filter size
- **stride** (*int*) – The stride step
- **dilation_rate** (*int*) – Specifying the dilation rate to use for dilated convolution.
- **act** (*activation function*) – The function that is applied to the layer activations
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **data_format** (*str*) – "channel_last" (NWC, default) or "channels_first" (NCW).
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 100, 1], name='input')
>>> conv1d = tl.layers.Conv1d(n_filter=32, filter_size=5, stride=2, b_init=None, ␣
↪in_channels=1, name='conv1d_1')
>>> print(conv1d)
>>> tensor = tl.layers.Conv1d(n_filter=32, filter_size=5, stride=2, act=tf.nn.
↪relu, name='conv1d_2')(net)
>>> print(tensor)
```

Conv2d

class tensorlayer.layers.**Conv2d**(*n_filter=32, filter_size=(3, 3), strides=(1, 1), act=None, padding='SAME', data_format='channels_last', dilation_rate=(1, 1), W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

Simplified version of Conv2dLayer.

Parameters

- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **strides** (*tuple of int*) – The sliding window strides of corresponding input dimensions. It must be in the same order as the shape parameter.
- **dilation_rate** (*tuple of int*) – Specifying the dilation rate to use for dilated convolution.
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **W_init** (*initializer*) – The initializer for the the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 400, 400, 3], name='input')
>>> conv2d = tl.layers.Conv2d(n_filter=32, filter_size=(3, 3), stride=(2, 2), b_
↪init=None, in_channels=3, name='conv2d_1')
>>> print(conv2d)
>>> tensor = tl.layers.Conv2d(n_filter=32, filter_size=(3, 3), stride=(2, 2),␣
↪act=tf.nn.relu, name='conv2d_2')(net)
>>> print(tensor)
```

Conv3d

class tensorlayer.layers.**Conv3d**(*n_filter=32, filter_size=(3, 3, 3), strides=(1, 1, 1), act=None, padding='SAME', data_format='channels_last', dilation_rate=(1, 1, 1), W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

Simplified version of `Conv3dLayer`.

Parameters

- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **strides** (*tuple of int*) – The sliding window strides of corresponding input dimensions. It must be in the same order as the `shape` parameter.
- **dilation_rate** (*tuple of int*) – Specifying the dilation rate to use for dilated convolution.
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **data_format** (*str*) – "channels_last" (NDHWC, default) or "channels_first" (NCDHW).
- **W_init** (*initializer*) – The initializer for the the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 20, 20, 20, 3], name='input')
>>> conv3d = tl.layers.Conv2d(n_filter=32, filter_size=(3, 3, 3), stride=(2, 2,␣
↪2), b_init=None, in_channels=3, name='conv3d_1')
>>> print(conv3d)
>>> tensor = tl.layers.Conv2d(n_filter=32, filter_size=(3, 3, 3), stride=(2, 2,␣
↪2), act=tf.nn.relu, name='conv3d_2')(net)
>>> print(tensor)
```

Deconvolutions

DeConv2d

class tensorlayer.layers.**DeConv2d**(*n_filter=32, filter_size=(3, 3), strides=(2, 2), act=None, padding='SAME', dilation_rate=(1, 1), data_format='channels_last', W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

Simplified version of `DeConv2dLayer`, see tf.nn.conv3d_transpose.

Parameters

- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **strides** (*tuple of int*) – The stride step (height, width).
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **act** (*activation function*) – The activation function of this layer.
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **dilation_rate** (*int of tuple of int*) – The dilation rate to use for dilated convolution
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([5, 100, 100, 32], name='input')
>>> deconv2d = tl.layers.DeConv2d(n_filter=32, filter_size=(3, 3), strides=(2, 2),
↪ in_channels=32, name='DeConv2d_1')
>>> print(deconv2d)
>>> tensor = tl.layers.DeConv2d(n_filter=32, filter_size=(3, 3), strides=(2, 2),␣
↪name='DeConv2d_2')(net)
>>> print(tensor)
```

DeConv3d

class tensorlayer.layers.**DeConv3d**(*n_filter=32, filter_size=(3, 3, 3), strides=(2, 2, 2), padding='SAME', act=None, data_format='channels_last', W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

Simplified version of `DeConv3dLayer`, see tf.nn.conv3d_transpose.

Parameters

- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (depth, height, width).
- **strides** (*tuple of int*) – The stride step (depth, height, width).
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **act** (*activation function*) – The activation function of this layer.
- **data_format** (*str*) – "channels_last" (NDHWC, default) or "channels_first" (NCDHW).
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip bias.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([5, 100, 100, 100, 32], name='input')
>>> deconv3d = tl.layers.DeConv3d(n_filter=32, filter_size=(3, 3, 3), strides=(2,␣
↪2, 2), in_channels=32, name='DeConv3d_1')
>>> print(deconv3d)
>>> tensor = tl.layers.DeConv3d(n_filter=32, filter_size=(3, 3, 3), strides=(2, 2,
↪ 2), name='DeConv3d_2')(net)
>>> print(tensor)
```

Deformable Convolutions

DeformableConv2d

class tensorlayer.layers.**DeformableConv2d**(*offset_layer=None, n_filter=32, filter_size=(3, 3), act=None, padding='SAME', W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The *DeformableConv2d* class is a 2D Deformable Convolutional Networks.

Parameters

- **offset_layer** (*tf.Tensor*) – To predict the offset of convolution operations. The shape is (batchsize, input height, input width, 2*(number of element in the convolution kernel)) e.g. if apply a 3*3 kernel, the number of the last dimension should be 18 (2*3*3)
- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **W_init** (*initializer*) – The initializer for the weight matrix.

- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.InputLayer([5, 10, 10, 16], name='input')
>>> offset1 = tl.layers.Conv2d(
...     n_filter=18, filter_size=(3, 3), strides=(1, 1), padding='SAME', name=
↪'offset1'
... )(net)
>>> deformconv1 = tl.layers.DeformableConv2d(
...     offset_layer=offset1, n_filter=32, filter_size=(3, 3), name='deformable1'
... )(net)
>>> offset2 = tl.layers.Conv2d(
...     n_filter=18, filter_size=(3, 3), strides=(1, 1), padding='SAME', name=
↪'offset2'
... )(deformconv1)
>>> deformconv2 = tl.layers.DeformableConv2d(
...     offset_layer=offset2, n_filter=64, filter_size=(3, 3), name='deformable2'
... )(deformconv1)
```

References

- The deformation operation was adapted from the implementation in here

Notes

- The padding is fixed to 'SAME'.
- The current implementation is not optimized for memory usgae. Please use it carefully.

Depthwise Convolutions

DepthwiseConv2d

class tensorlayer.layers.**DepthwiseConv2d**(*filter_size=(3, 3), strides=(1, 1), act=None, padding='SAME', data_format='channels_last', dilation_rate=(1, 1), depth_multiplier=1, W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

Separable/Depthwise Convolutional 2D layer, see tf.nn.depthwise_conv2d.

Input: 4-D Tensor (batch, height, width, in_channels).

Output: 4-D Tensor (batch, new height, new width, in_channels * depth_multiplier).

Parameters

- **filter_size** (*tuple of 2 int*) – The filter size (height, width).
- **strides** (*tuple of 2 int*) – The stride step (height, width).
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **dilation_rate** (*tuple of 2 int*) – The dilation rate in which we sample input values across the height and width dimensions in atrous convolution. If it is greater than 1, then all values of strides must be 1.
- **depth_multiplier** (*int*) – The number of channels to expand to.
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip bias.
- **in_channels** (*int*) – The number of in channels.
- **name** (*str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 200, 200, 32], name='input')
>>> depthwiseconv2d = tl.layers.DepthwiseConv2d(
...     filter_size=(3, 3), strides=(1, 1), dilation_rate=(2, 2), act=tf.nn.relu,␣
↪depth_multiplier=2, name='depthwise'
... )(net)
>>> print(depthwiseconv2d)
>>> output shape : (8, 200, 200, 64)
```

References

- tflearn's grouped_conv_2d
- keras's separableconv2d

Group Convolutions

GroupConv2d

class tensorlayer.layers.**GroupConv2d**(*n_filter=32, filter_size=(3, 3), strides=(2, 2), n_group=2, act=None, padding='SAME', data_format='channels_last', dilation_rate=(1, 1), W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The *GroupConv2d* class is 2D grouped convolution, see here.

Parameters

- **n_filter** (*int*) – The number of filters.

- **filter_size** (*tuple of int*) – The filter size.
- **strides** (*tuple of int*) – The stride step.
- **n_group** (*int*) – The number of groups.
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **dilation_rate** (*tuple of int*) – Specifying the dilation rate to use for dilated convolution.
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 24, 24, 32], name='input')
>>> groupconv2d = tl.layers.QuanConv2d(
...     n_filter=64, filter_size=(3, 3), strides=(2, 2), n_group=2, name='group'
... )(net)
>>> print(groupconv2d)
>>> output shape : (8, 12, 12, 64)
```

Separable Convolutions

SeparableConv1d

class tensorlayer.layers.**SeparableConv1d**(*n_filter=100, filter_size=3, strides=1, act=None, padding='valid', data_format='channels_last', dilation_rate=1, depth_multiplier=1, depthwise_init=None, pointwise_init=None, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The *SeparableConv1d* class is a 1D depthwise separable convolutional layer.

This layer performs a depthwise convolution that acts separately on channels, followed by a pointwise convolution that mixes channels.

Parameters

- **n_filter** (*int*) – The dimensionality of the output space (i.e. the number of filters in the convolution).
- **filter_size** (*int*) – Specifying the spatial dimensions of the filters. Can be a single integer to specify the same value for all spatial dimensions.

- **strides** (*int*) – Specifying the stride of the convolution. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- **padding** (*str*) – One of "valid" or "same" (case-insensitive).
- **data_format** (*str*) – One of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width).
- **dilation_rate** (*int*) – Specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
- **depth_multiplier** (*int*) – The number of depthwise convolution output channels for each input channel. The total number of depthwise convolution output channels will be equal to num_filters_in * depth_multiplier.
- **depthwise_init** (*initializer*) – for the depthwise convolution kernel.
- **pointwise_init** (*initializer*) – For the pointwise convolution kernel.
- **b_init** (*initializer*) – For the bias vector. If None, ignore bias in the pointwise part only.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 50, 64], name='input')
>>> separableconv1d = tl.layers.Conv1d(n_filter=32, filter_size=3, strides=2,␣
↪padding='SAME', act=tf.nn.relu, name='separable_1d')(net)
>>> print(separableconv1d)
>>> output shape : (8, 25, 32)
```

SeparableConv2d

class tensorlayer.layers.**SeparableConv2d**(*n_filter=100, filter_size=(3, 3), strides=(1, 1), act=None, padding='valid', data_format='channels_last', dilation_rate=(1, 1), depth_multiplier=1, depthwise_init=None, pointwise_init=None, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The *SeparableConv2d* class is a 2D depthwise separable convolutional layer.

This layer performs a depthwise convolution that acts separately on channels, followed by a pointwise convolution that mixes channels. While *DepthwiseConv2d* performs depthwise convolution only, which allow us to add batch normalization between depthwise and pointwise convolution.

Parameters

- **n_filter** (*int*) – The dimensionality of the output space (i.e. the number of filters in the convolution).

- **filter_size** (*tuple/list of 2 int*) – Specifying the spatial dimensions of the filters. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** (*tuple/list of 2 int*) – Specifying the strides of the convolution. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- **padding** (*str*) – One of "valid" or "same" (case-insensitive).
- **data_format** (*str*) – One of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width).
- **dilation_rate** (*integer or tuple/list of 2 int*) – Specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
- **depth_multiplier** (*int*) – The number of depthwise convolution output channels for each input channel. The total number of depthwise convolution output channels will be equal to num_filters_in * depth_multiplier.
- **depthwise_init** (*initializer*) – for the depthwise convolution kernel.
- **pointwise_init** (*initializer*) – For the pointwise convolution kernel.
- **b_init** (*initializer*) – For the bias vector. If None, ignore bias in the pointwise part only.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 50, 50, 64], name='input')
>>> separableconv2d = tl.layers.Conv1d(n_filter=32, filter_size=(3, 3),␣
↪strides=(2, 2), act=tf.nn.relu, padding='VALID', name='separableconv2d')(net)
>>> print(separableconv2d)
>>> output shape : (8, 24, 24, 32)
```

SubPixel Convolutions

SubpixelConv1d

class tensorlayer.layers.**SubpixelConv1d**(*scale=2*, *act=None*, *in_channels=None*, *name=None*)

It is a 1D sub-pixel up-sampling layer.

Calls a TensorFlow function that directly implements this functionality. We assume input has dim (batch, width, r)

Parameters

- **scale** (*int*) – The up-scaling ratio, a wrong setting will lead to Dimension size error.
- **act** (*activation function*) – The activation function of this layer.

- **in_channels** (*int*) – The number of in channels.
- **name** (*str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 25, 32], name='input')
>>> subpixelconv1d = tl.layers.SubpixelConv1d(scale=2, name='subpixelconv1d')(net)
>>> print(subpixelconv1d)
>>> output shape : (8, 50, 16)
```

References

Audio Super Resolution Implementation.

SubpixelConv2d

class tensorlayer.layers.**SubpixelConv2d**(*scale=2, n_out_channels=None, act=None, in_channels=None, name=None*)

It is a 2D sub-pixel up-sampling layer, usually be used for Super-Resolution applications, see SRGAN for example.

Parameters

- **scale** (*int*) – The up-scaling ratio, a wrong setting will lead to dimension size error.
- **n_out_channel** (*int or None*) – The number of output channels. - If None, automatically set n_out_channel == the number of input channels / (scale x scale). - The number of input channels == (scale x scale) x The number of output channels.
- **act** (*activation function*) – The activation function of this layer.
- **in_channels** (*int*) – The number of in channels.
- **name** (*str*) – A unique layer name.

Examples

With TensorLayer

```
>>> # examples here just want to tell you how to set the n_out_channel.
>>> net = tl.layers.Input([2, 16, 16, 4], name='input1')
>>> subpixelconv2d = tl.layers.SubpixelConv2d(scale=2, n_out_channel=1, name=
↪'subpixel_conv2d1')(net)
>>> print(subpixelconv2d)
>>> output shape : (2, 32, 32, 1)
```

```
>>> net = tl.layers.Input([2, 16, 16, 4*10], name='input2')
>>> subpixelconv2d = tl.layers.SubpixelConv2d(scale=2, n_out_channel=10, name=
↪'subpixel_conv2d2')(net)
>>> print(subpixelconv2d)
>>> output shape : (2, 32, 32, 10)
```

```
>>> net = tl.layers.Input([2, 16, 16, 25*10], name='input3')
>>> subpixelconv2d = tl.layers.SubpixelConv2d(scale=5, n_out_channel=10, name=
↪'subpixel_conv2d3')(net)
>>> print(subpixelconv2d)
>>> output shape : (2, 80, 80, 10)
```

References

- Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network

2.7.6 Dense Layers

Dense Layer

class tensorlayer.layers.**Dense**(*n_units*, *act=None*, *W_init=<tensorlayer.initializers.TruncatedNormal object>*, *b_init=<tensorlayer.initializers.Constant object>*, *in_channels=None*, *name=None*)

The *Dense* class is a fully connected layer.

Parameters

- **n_units** (*int*) – The number of units of this layer.
- **act** (*activation function*) – The activation function of this layer.
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of channels of the previous layer. If None, it will be automatically detected when the layer is forwarded for the first time.
- **name** (*None or str*) – A unique layer name. If None, a unique name will be automatically generated.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([100, 50], name='input')
>>> dense = tl.layers.Dense(n_units=800, act=tf.nn.relu, in_channels=50, name=
↪'dense_1')
>>> print(dense)
Dense(n_units=800, relu, in_channels='50', name='dense_1')
>>> tensor = tl.layers.Dense(n_units=800, act=tf.nn.relu, name='dense_2')(net)
>>> print(tensor)
tf.Tensor([...], shape=(100, 800), dtype=float32)
```

Notes

If the layer input has more than two axes, it needs to be flatten by using *Flatten*.

Drop Connect Dense Layer

class tensorlayer.layers.**DropconnectDense**(*keep=0.5*, *n_units=100*, *act=None*, *W_init=<tensorlayer.initializers.TruncatedNormal object>*, *b_init=<tensorlayer.initializers.Constant object>*, *in_channels=None*, *name=None*)

The `DropconnectDense` class is `Dense` with DropConnect behaviour which randomly removes connections between this layer and the previous layer according to a keeping probability.

Parameters

- **keep** (*float*) – The keeping probability. The lower the probability it is, the more activations are set to zero.
- **n_units** (*int*) – The number of units of this layer.
- **act** (*activation function*) – The activation function of this layer.
- **W_init** (*weights initializer*) – The initializer for the weight matrix.
- **b_init** (*biases initializer*) – The initializer for the bias vector.
- **in_channels** (*int*) – The number of channels of the previous layer. If None, it will be automatically detected when the layer is forwarded for the first time.
- **name** (*str*) – A unique layer name.

Examples

```
>>> net = tl.layers.Input([None, 784], name='input')
>>> net = tl.layers.DropconnectDense(keep=0.8,
...         n_units=800, act=tf.nn.relu, name='relu1')(net)
>>> net = tl.layers.DropconnectDense(keep=0.5,
...         n_units=800, act=tf.nn.relu, name='relu2')(net)
>>> net = tl.layers.DropconnectDense(keep=0.5,
...         n_units=10, name='output')(net)
```

References

- Wan, L. (2013). Regularization of neural networks using dropconnect

2.7.7 Dropout Layers

class tensorlayer.layers.**Dropout**(*keep*, *seed=None*, *name=None*)

The `Dropout` class is a noise layer which randomly set some activations to zero according to a keeping probability.

Parameters

- **keep** (*float*) – The keeping probability. The lower the probability it is, the more activations are set to zero.
- **seed** (*int or None*) – The seed for random dropout.
- **name** (*None or str*) – A unique layer name.

2.7.8 Extend Layers

Expand Dims Layer

class tensorlayer.layers.**ExpandDims**(*axis*, *name=None*)

The *ExpandDims* class inserts a dimension of 1 into a tensor's shape, see tf.expand_dims() .

Parameters

- **axis** (*int*) – The dimension index at which to expand the shape of input.
- **name** (*str*) – A unique layer name. If None, a unique name will be automatically assigned.

Examples

```
>>> x = tl.layers.Input([10, 3], name='in')
>>> y = tl.layers.ExpandDims(axis=-1)(x)
[10, 3, 1]
```

Tile layer

class tensorlayer.layers.**Tile**(*multiples=None*, *name=None*)

The *Tile* class constructs a tensor by tiling a given tensor, see tf.tile() .

Parameters

- **multiples** (*tensor*) – Must be one of the following types: int32, int64. 1-D Length must be the same as the number of dimensions in input.
- **name** (*None or str*) – A unique layer name.

Examples

```
>>> x = tl.layers.Input([10, 3], name='in')
>>> y = tl.layers.Tile(multiples=[2, 3])(x)
[20, 9]
```

2.7.9 Image Resampling Layers

2D UpSampling

class tensorlayer.layers.**UpSampling2d**(*scale*, *method='bilinear'*, *antialias=False*, *data_format='channel_last'*, *name=None*)

The *UpSampling2d* class is a up-sampling 2D layer.

See tf.image.resize_images.

Parameters

- **scale** (*int/float or tuple of int/float*) – (height, width) scale factor.
- **method** (*str*) –

  The resize method selected through the given string. Default 'bilinear'.

  – 'bilinear', Bilinear interpolation.

- 'nearest', Nearest neighbor interpolation.
- 'bicubic', Bicubic interpolation.
- 'area', Area interpolation.

- **antialias** (*boolean*) – Whether to use an anti-aliasing filter when downsampling an image.
- **data_format** (*str*) – channels_last 'channel_last' (default) or channels_first.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> ni = tl.layers.Input([None, 50, 50, 32], name='input')
>>> ni = tl.layers.UpSampling2d(scale=(2, 2))(ni)
>>> output shape : [None, 100, 100, 32]
```

2D DownSampling

class tensorlayer.layers.**DownSampling2d**(*scale*, *method='bilinear'*, *antialias=False*, *data_format='channel_last'*, *name=None*)

The *DownSampling2d* class is down-sampling 2D layer.

See tf.image.resize_images.

Parameters

- **scale** (*int/float or tuple of int/float*) – (height, width) scale factor.
- **method** (*str*) –

  The resize method selected through the given string. Default 'bilinear'.

  - 'bilinear', Bilinear interpolation.
  - 'nearest', Nearest neighbor interpolation.
  - 'bicubic', Bicubic interpolation.
  - 'area', Area interpolation.

- **antialias** (*boolean*) – Whether to use an anti-aliasing filter when downsampling an image.
- **data_format** (*str*) – channels_last 'channel_last' (default) or channels_first.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> ni = tl.layers.Input([None, 50, 50, 32], name='input')
>>> ni = tl.layers.DownSampling2d(scale=(2, 2))(ni)
>>> output shape : [None, 25, 25, 32]
```

2.7.10 Lambda Layers

Lambda Layer

class tensorlayer.layers.**Lambda**(*fn*, *fn_weights=None*, *fn_args=None*, *name=None*)

A layer that takes a user-defined function using Lambda. If the function has trainable weights, the weights should be provided. Remember to make sure the weights provided when the layer is constructed are SAME as the weights used when the layer is forwarded. For multiple inputs see `ElementwiseLambda`.

Parameters

- **fn** (*function*) – The function that applies to the inputs (e.g. tensor from the previous layer).
- **fn_weights** (*list*) – The trainable weights for the function if any. Optional.
- **fn_args** (*dict*) – The arguments for the function if any. Optional.
- **name** (*str or None*) – A unique layer name.

Examples

Non-parametric and non-args case This case is supported in the Model.save() / Model.load() to save / load the whole model architecture and weights(optional).

```
>>> x = tl.layers.Input([8, 3], name='input')
>>> y = tl.layers.Lambda(lambda x: 2*x, name='lambda')(x)
```

Non-parametric and with args case This case is supported in the Model.save() / Model.load() to save / load the whole model architecture and weights(optional).

```
>>> def customize_func(x, foo=42): # x is the inputs, foo is an argument
>>>     return foo * x
>>> x = tl.layers.Input([8, 3], name='input')
>>> lambdalayer = tl.layers.Lambda(customize_func, fn_args={'foo': 2}, name=
↪'lambda')(x)
```

Any function with outside variables This case has not been supported in Model.save() / Model.load() yet. Please avoid using Model.save() / Model.load() to save / load models that contain such Lambda layer. Instead, you may use Model.save_weights() / Model.load_weights() to save / load model weights. Note: In this case, fn_weights should be a list, and then the trainable weights in this Lambda layer can be added into the weights of the whole model.

```
>>> vara = [tf.Variable(1.0)]
>>> def func(x):
>>>     return x + vara
>>> x = tl.layers.Input([8, 3], name='input')
>>> y = tl.layers.Lambda(func, fn_weights=a, name='lambda')(x)
```

Parametric case, merge other wrappers into TensorLayer This case is supported in the Model.save() / Model.load() to save / load the whole model architecture and weights(optional).

```
>>> layers = [
>>>     tf.keras.layers.Dense(10, activation=tf.nn.relu),
>>>     tf.keras.layers.Dense(5, activation=tf.nn.sigmoid),
>>>     tf.keras.layers.Dense(1, activation=tf.identity)
>>> ]
```

(continues on next page)

(continued from previous page)

```
>>> perceptron = tf.keras.Sequential(layers)
>>> # in order to compile keras model and get trainable_variables of the keras␣
↪model
>>> _ = perceptron(np.random.random([100, 5]).astype(np.float32))
```

```
>>> class CustomizeModel(tl.models.Model):
>>>     def __init__(self):
>>>         super(CustomizeModel, self).__init__()
>>>         self.dense = tl.layers.Dense(in_channels=1, n_units=5)
>>>         self.lambdalayer = tl.layers.Lambda(perceptron, perceptron.trainable_
↪variables)
```

```
>>>     def forward(self, x):
>>>         z = self.dense(x)
>>>         z = self.lambdalayer(z)
>>>         return z
```

```
>>> optimizer = tf.optimizers.Adam(learning_rate=0.1)
>>> model = CustomizeModel()
>>> model.train()
```

```
>>> for epoch in range(50):
>>>     with tf.GradientTape() as tape:
>>>         pred_y = model(data_x)
>>>         loss = tl.cost.mean_squared_error(pred_y, data_y)
```

```
>>>     gradients = tape.gradient(loss, model.trainable_weights)
>>>     optimizer.apply_gradients(zip(gradients, model.trainable_weights))
```

ElementWise Lambda Layer

class tensorlayer.layers.**ElementwiseLambda**(*fn*, *fn_weights=None*, *fn_args=None*, *name=None*)

A layer that use a custom function to combine multiple `Layer` inputs. If the function has trainable weights, the weights should be provided. Remember to make sure the weights provided when the layer is constructed are SAME as the weights used when the layer is forwarded.

Parameters

- **fn** (*function*) – The function that applies to the inputs (e.g. tensor from the previous layer).
- **fn_weights** (*list*) – The trainable weights for the function if any. Optional.
- **fn_args** (*dict*) – The arguments for the function if any. Optional.
- **name** (*str or None*) – A unique layer name.

Examples

Non-parametric and with args case This case is supported in the Model.save() / Model.load() to save / load the whole model architecture and weights(optional).

z = mean + noise * tf.exp(std * 0.5) + foo >>> def func(noise, mean, std, foo=42): >>> return mean + noise * tf.exp(std * 0.5) + foo

```
>>> noise = tl.layers.Input([100, 1])
>>> mean = tl.layers.Input([100, 1])
>>> std = tl.layers.Input([100, 1])
>>> out = tl.layers.ElementwiseLambda(fn=func, fn_args={'foo': 84}, name=
↪'elementwiselambda')([noise, mean, std])
```

Non-parametric and non-args case This case is supported in the Model.save() / Model.load() to save / load the whole model architecture and weights(optional).

z = mean + noise * tf.exp(std * 0.5) >>> noise = tl.layers.Input([100, 1]) >>> mean = tl.layers.Input([100, 1]) >>> std = tl.layers.Input([100, 1]) >>> out = tl.layers.ElementwiseLambda(fn=lambda x, y, z: x + y * tf.exp(z * 0.5), name='elementwiselambda')([noise, mean, std])

Any function with outside variables This case has not been supported in Model.save() / Model.load() yet. Please avoid using Model.save() / Model.load() to save / load models that contain such ElementwiseLambda layer. Instead, you may use Model.save_weights() / Model.load_weights() to save / load model weights. Note: In this case, fn_weights should be a list, and then the trainable weights in this ElementwiseLambda layer can be added into the weights of the whole model.

z = mean + noise * tf.exp(std * 0.5) + vara >>> vara = [tf.Variable(1.0)] >>> def func(noise, mean, std): >>> return mean + noise * tf.exp(std * 0.5) + vara >>> noise = tl.layers.Input([100, 1]) >>> mean = tl.layers.Input([100, 1]) >>> std = tl.layers.Input([100, 1]) >>> out = tl.layers.ElementwiseLambda(fn=func, fn_weights=vara, name='elementwiselambda')([noise, mean, std])

2.7.11 Merge Layers

Concat Layer

class tensorlayer.layers.**Concat**(*concat_dim=-1*, *name=None*)

A layer that concats multiple tensors according to given axis.

Parameters

- **concat_dim** (*int*) – The dimension to concatenate.
- **name** (*None or str*) – A unique layer name.

Examples

```
>>> class CustomModel(tl.models.Model):
>>>     def __init__(self):
>>>         super(CustomModel, self).__init__(name="custom")
>>>         self.dense1 = tl.layers.Dense(in_channels=20, n_units=10, act=tf.nn.
↪relu, name='relu1_1')
>>>         self.dense2 = tl.layers.Dense(in_channels=20, n_units=10, act=tf.nn.
↪relu, name='relu2_1')
>>>         self.concat = tl.layers.Concat(concat_dim=1, name='concat_layer')
```

```
>>>     def forward(self, inputs):
>>>         d1 = self.dense1(inputs)
>>>         d2 = self.dense2(inputs)
>>>         outputs = self.concat([d1, d2])
>>>         return outputs
```

ElementWise Layer

class tensorlayer.layers.**Elementwise**(*combine_fn=tensorflow.minimum*, *act=None*, *name=None*)

A layer that combines multiple `Layer` that have the same output shapes according to an element-wise operation. If the element-wise operation is complicated, please consider to use `ElementwiseLambda`.

Parameters

- **combine_fn** (*a TensorFlow element-wise combine function*) – e.g. AND is `tf.minimum` ; OR is `tf.maximum` ; ADD is `tf.add` ; MUL is `tf.multiply` and so on. See TensorFlow Math API . If the combine function is more complicated, please consider to use `ElementwiseLambda`.
- **act** (*activation function*) – The activation function of this layer.
- **name** (*None or str*) – A unique layer name.

Examples

```
>>> class CustomModel(tl.models.Model):
>>>     def __init__(self):
>>>         super(CustomModel, self).__init__(name="custom")
>>>         self.dense1 = tl.layers.Dense(in_channels=20, n_units=10, act=tf.nn.
↪relu, name='relu1_1')
>>>         self.dense2 = tl.layers.Dense(in_channels=20, n_units=10, act=tf.nn.
↪relu, name='relu2_1')
>>>         self.element = tl.layers.Elementwise(combine_fn=tf.minimum, name=
↪'minimum', act=tf.identity)
```

```
>>>     def forward(self, inputs):
>>>         d1 = self.dense1(inputs)
>>>         d2 = self.dense2(inputs)
>>>         outputs = self.element([d1, d2])
>>>         return outputs
```

2.7.12 Noise Layer

class tensorlayer.layers.**GaussianNoise**(*mean=0.0*, *stddev=1.0*, *is_train=True*, *seed=None*, *name=None*)

The `GaussianNoise` class is noise layer that adding noise with gaussian distribution to the activation.

Parameters

- **mean** (*float*) – The mean. Default is 0.0.
- **stddev** (*float*) – The standard deviation. Default is 1.0.
- **is_train** (*boolean*) – Is trainable layer. If False, skip this layer. default is True.
- **seed** (*int or None*) – The seed for random noise.
- **name** (*str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([64, 200], name='input')
>>> net = tl.layers.Dense(n_units=100, act=tf.nn.relu, name='dense')(net)
>>> gaussianlayer = tl.layers.GaussianNoise(name='gaussian')(net)
>>> print(gaussianlayer)
>>> output shape : (64, 100)
```

2.7.13 Normalization Layers

Batch Normalization

class tensorlayer.layers.**BatchNorm**(*decay=0.9*, *epsilon=1e-05*, *act=None*, *is_train=False*, *beta_init=<tensorlayer.initializers.Zeros object>*, *gamma_init=<tensorlayer.initializers.RandomNormal object>*, *moving_mean_init=<tensorlayer.initializers.Zeros object>*, *moving_var_init=<tensorlayer.initializers.Zeros object>*, *num_features=None*, *data_format='channels_last'*, *name=None*)

The `BatchNorm` is a batch normalization layer for both fully-connected and convolution outputs. See `tf.nn.batch_normalization` and `tf.nn.moments`.

Parameters

- **decay** (*float*) – A decay factor for *ExponentialMovingAverage*. Suggest to use a large value for large dataset.
- **epsilon** (*float*) – Eplison.
- **act** (*activation function*) – The activation function of this layer.
- **is_train** (*boolean*) – Is being used for training or inference.
- **beta_init** (*initializer or None*) – The initializer for initializing beta, if None, skip beta. Usually you should not skip beta unless you know what happened.
- **gamma_init** (*initializer or None*) – The initializer for initializing gamma, if None, skip gamma. When the batch normalization layer is use instead of 'biases', or the next layer is linear, this can be disabled since the scaling can be done by the next layer. see Inception-ResNet-v2
- **moving_mean_init** (*initializer or None*) – The initializer for initializing moving mean, if None, skip moving mean.
- **moving_var_init** (*initializer or None*) – The initializer for initializing moving var, if None, skip moving var.
- **num_features** (*int*) – Number of features for input tensor. Useful to build layer if using BatchNorm1d, BatchNorm2d or BatchNorm3d, but should be left as None if using BatchNorm. Default None.
- **data_format** (*str*) – channels_last 'channel_last' (default) or channels_first.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 50, 32], name='input')
>>> net = tl.layers.BatchNorm()(net)
```

Notes

The *BatchNorm* is universally suitable for 3D/4D/5D input in static model, but should not be used in dynamic model where layer is built upon class initialization. So the argument 'num_features' should only be used for subclasses *BatchNorm1d*, *BatchNorm2d* and *BatchNorm3d*. All the three subclasses are suitable under all kinds of conditions.

References

- Source
- stackoverflow

Batch Normalization 1D

class tensorlayer.layers.**BatchNorm1d**(*decay=0.9*, *epsilon=1e-05*, *act=None*, *is_train=False*, *beta_init=<tensorlayer.initializers.Zeros object>*, *gamma_init=<tensorlayer.initializers.RandomNormal object>*, *moving_mean_init=<tensorlayer.initializers.Zeros object>*, *moving_var_init=<tensorlayer.initializers.Zeros object>*, *num_features=None*, *data_format='channels_last'*, *name=None*)

The *BatchNorm1d* applies Batch Normalization over 3D input (a mini-batch of 1D inputs with additional channel dimension), of shape (N, L, C) or (N, C, L). See more details in *BatchNorm*.

Examples

With TensorLayer

```
>>> # in static model, no need to specify num_features
>>> net = tl.layers.Input([None, 50, 32], name='input')
>>> net = tl.layers.BatchNorm1d()(net)
>>> # in dynamic model, build by specifying num_features
>>> conv = tl.layers.Conv1d(32, 5, 1, in_channels=3)
>>> bn = tl.layers.BatchNorm1d(num_features=32)
```

Batch Normalization 2D

class tensorlayer.layers.**BatchNorm2d**(*decay=0.9*, *epsilon=1e-05*, *act=None*, *is_train=False*, *beta_init=<tensorlayer.initializers.Zeros object>*, *gamma_init=<tensorlayer.initializers.RandomNormal object>*, *moving_mean_init=<tensorlayer.initializers.Zeros object>*, *moving_var_init=<tensorlayer.initializers.Zeros object>*, *num_features=None*, *data_format='channels_last'*, *name=None*)

The *BatchNorm2d* applies Batch Normalization over 4D input (a mini-batch of 2D inputs with additional channel dimension) of shape (N, H, W, C) or (N, C, H, W). See more details in *BatchNorm*.

Examples

With TensorLayer

```
>>> # in static model, no need to specify num_features
>>> net = tl.layers.Input([None, 50, 50, 32], name='input')
>>> net = tl.layers.BatchNorm2d()(net)
>>> # in dynamic model, build by specifying num_features
>>> conv = tl.layers.Conv2d(32, (5, 5), (1, 1), in_channels=3)
>>> bn = tl.layers.BatchNorm2d(num_features=32)
```

Batch Normalization 3D

class tensorlayer.layers.**BatchNorm3d**(*decay=0.9*, *epsilon=1e-05*, *act=None*, *is_train=False*, *beta_init=<tensorlayer.initializers.Zeros object>*, *gamma_init=<tensorlayer.initializers.RandomNormal object>*, *moving_mean_init=<tensorlayer.initializers.Zeros object>*, *moving_var_init=<tensorlayer.initializers.Zeros object>*, *num_features=None*, *data_format='channels_last'*, *name=None*)

The *BatchNorm3d* applies Batch Normalization over 5D input (a mini-batch of 3D inputs with additional channel dimension) with shape (N, D, H, W, C) or (N, C, D, H, W). See more details in *BatchNorm*.

Examples

With TensorLayer

```
>>> # in static model, no need to specify num_features
>>> net = tl.layers.Input([None, 50, 50, 50, 32], name='input')
>>> net = tl.layers.BatchNorm3d()(net)
>>> # in dynamic model, build by specifying num_features
>>> conv = tl.layers.Conv3d(32, (5, 5, 5), (1, 1), in_channels=3)
>>> bn = tl.layers.BatchNorm3d(num_features=32)
```

Local Response Normalization

class tensorlayer.layers.**LocalResponseNorm**(*depth_radius=None*, *bias=None*, *alpha=None*, *beta=None*, *name=None*)

The *LocalResponseNorm* layer is for Local Response Normalization. See `tf.nn.local_response_normalization` or `tf.nn.lrn` for new TF version. The 4-D input tensor is a 3-D array of 1-D vectors (along the last dimension), and each vector is normalized independently. Within a given vector, each component is divided by the weighted square-sum of inputs within depth_radius.

Parameters

- **depth_radius** (*int*) – Depth radius. 0-D. Half-width of the 1-D normalization window.
- **bias** (*float*) – An offset which is usually positive and shall avoid dividing by 0.
- **alpha** (*float*) – A scale factor which is usually positive.
- **beta** (*float*) – An exponent.
- **name** (*None or str*) – A unique layer name.

Instance Normalization

class tensorlayer.layers.**InstanceNorm**(*act=None*, *epsilon=1e-05*, *beta_init=<tensorlayer.initializers.Zeros object>*, *gamma_init=<tensorlayer.initializers.RandomNormal object>*, *num_features=None*, *data_format='channels_last'*, *name=None*)

The *InstanceNorm* is an instance normalization layer for both fully-connected and convolution outputs. See `tf.nn.batch_normalization` and `tf.nn.moments`.

Parameters

- **act** (*activation function.*) – The activation function of this layer.
- **epsilon** (*float*) – Eplison.
- **beta_init** (*initializer or None*) – The initializer for initializing beta, if None, skip beta. Usually you should not skip beta unless you know what happened.
- **gamma_init** (*initializer or None*) – The initializer for initializing gamma, if None, skip gamma. When the instance normalization layer is use instead of 'biases', or the next layer is linear, this can be disabled since the scaling can be done by the next layer. see Inception-ResNet-v2
- **num_features** (*int*) – Number of features for input tensor. Useful to build layer if using InstanceNorm1d, InstanceNorm2d or InstanceNorm3d, but should be left as None if using InstanceNorm. Default None.
- **data_format** (*str*) – channels_last 'channel_last' (default) or channels_first.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 50, 32], name='input')
>>> net = tl.layers.InstanceNorm()(net)
```

Notes

The *InstanceNorm* is universally suitable for 3D/4D/5D input in static model, but should not be used in dynamic model where layer is built upon class initialization. So the argument 'num_features' should only be used for subclasses *InstanceNorm1d*, *InstanceNorm2d* and *InstanceNorm3d*. All the three subclasses are suitable under all kinds of conditions.

Instance Normalization 1D

class tensorlayer.layers.**InstanceNorm1d**(*act=None*, *epsilon=1e-05*, *beta_init=<tensorlayer.initializers.Zeros object>*, *gamma_init=<tensorlayer.initializers.RandomNormal object>*, *num_features=None*, *data_format='channels_last'*, *name=None*)

The *InstanceNorm1d* applies Instance Normalization over 3D input (a mini-instance of 1D inputs with additional channel dimension), of shape (N, L, C) or (N, C, L). See more details in *InstanceNorm*.

Examples

With TensorLayer

```
>>> # in static model, no need to specify num_features
>>> net = tl.layers.Input([None, 50, 32], name='input')
>>> net = tl.layers.InstanceNorm1d()(net)
>>> # in dynamic model, build by specifying num_features
>>> conv = tl.layers.Conv1d(32, 5, 1, in_channels=3)
>>> bn = tl.layers.InstanceNorm1d(num_features=32)
```

Instance Normalization 2D

class tensorlayer.layers.**InstanceNorm2d**(*act=None, epsilon=1e-05, beta_init=<tensorlayer.initializers.Zeros object>, gamma_init=<tensorlayer.initializers.RandomNormal object>, num_features=None, data_format='channels_last', name=None*)

The `InstanceNorm2d` applies Instance Normalization over 4D input (a mini-instance of 2D inputs with additional channel dimension) of shape (N, H, W, C) or (N, C, H, W). See more details in `InstanceNorm`.

Examples

With TensorLayer

```
>>> # in static model, no need to specify num_features
>>> net = tl.layers.Input([None, 50, 50, 32], name='input')
>>> net = tl.layers.InstanceNorm2d()(net)
>>> # in dynamic model, build by specifying num_features
>>> conv = tl.layers.Conv2d(32, (5, 5), (1, 1), in_channels=3)
>>> bn = tl.layers.InstanceNorm2d(num_features=32)
```

Instance Normalization 3D

class tensorlayer.layers.**InstanceNorm3d**(*act=None, epsilon=1e-05, beta_init=<tensorlayer.initializers.Zeros object>, gamma_init=<tensorlayer.initializers.RandomNormal object>, num_features=None, data_format='channels_last', name=None*)

The `InstanceNorm3d` applies Instance Normalization over 5D input (a mini-instance of 3D inputs with additional channel dimension) with shape (N, D, H, W, C) or (N, C, D, H, W). See more details in `InstanceNorm`.

Examples

With TensorLayer

```
>>> # in static model, no need to specify num_features
>>> net = tl.layers.Input([None, 50, 50, 50, 32], name='input')
>>> net = tl.layers.InstanceNorm3d()(net)
>>> # in dynamic model, build by specifying num_features
>>> conv = tl.layers.Conv3d(32, (5, 5, 5), (1, 1), in_channels=3)
>>> bn = tl.layers.InstanceNorm3d(num_features=32)
```

Layer Normalization

class tensorlayer.layers.**LayerNorm**(*center=True, scale=True, act=None, epsilon=1e-12, begin_norm_axis=1, begin_params_axis=-1, beta_init=<tensorlayer.initializers.Zeros object>, gamma_init=<tensorlayer.initializers.Ones object>, data_format='channels_last', name=None*)

The `LayerNorm` class is for layer normalization, see tf.contrib.layers.layer_norm.

Parameters

- **prev_layer** (`Layer`) – The previous layer.
- **act** (*activation function*) – The activation function of this layer.
- **others** – tf.contrib.layers.layer_norm.

Group Normalization

class tensorlayer.layers.**GroupNorm**(*groups=32, epsilon=1e-06, act=None, data_format='channels_last', name=None*)

The `GroupNorm` layer is for Group Normalization. See tf.contrib.layers.group_norm.

Parameters

- **prev_layer** (*#*) –
- **The previous layer.** (*#*) –
- **groups** (*int*) – The number of groups
- **act** (*activation function*) – The activation function of this layer.
- **epsilon** (*float*) – Eplison.
- **data_format** (*str*) – channels_last 'channel_last' (default) or channels_first.
- **name** (*None or str*) – A unique layer name

Switch Normalization

class tensorlayer.layers.**SwitchNorm**(*act=None, epsilon=1e-05, beta_init=<tensorlayer.initializers.Constant object>, gamma_init=<tensorlayer.initializers.Constant object>, moving_mean_init=<tensorlayer.initializers.Zeros object>, data_format='channels_last', name=None*)

The `SwitchNorm` is a switchable normalization.

Parameters

- **act** (*activation function*) – The activation function of this layer.
- **epsilon** (*float*) – Eplison.
- **beta_init** (*initializer or None*) – The initializer for initializing beta, if None, skip beta. Usually you should not skip beta unless you know what happened.
- **gamma_init** (*initializer or None*) – The initializer for initializing gamma, if None, skip gamma. When the batch normalization layer is use instead of 'biases', or the next layer is linear, this can be disabled since the scaling can be done by the next layer. see Inception-ResNet-v2

- **moving_mean_init** (*initializer or None*) – The initializer for initializing moving mean, if None, skip moving mean.
- **data_format** (*str*) – channels_last 'channel_last' (default) or channels_first.
- **name** (*None or str*) – A unique layer name.

References

- Differentiable Learning-to-Normalize via Switchable Normalization
- Zhihu (CN)

2.7.14 Padding Layers

Pad Layer (Expert API)

Padding layer for any modes.

class tensorlayer.layers.**PadLayer**(*padding=None*, *mode='CONSTANT'*, *name=None*)

The *PadLayer* class is a padding layer for any mode and dimension. Please see tf.pad for usage.

Parameters

- **padding** (*list of lists of 2 ints, or a Tensor of type int32.*) – The int32 values to pad.
- **mode** (*str*) – "CONSTANT", "REFLECT", or "SYMMETRIC" (case-insensitive).
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 224, 224, 3], name='input')
>>> padlayer = tl.layers.PadLayer([[0, 0], [3, 3], [3, 3], [0, 0]], "REFLECT",
↪name='inpad')(net)
>>> print(padlayer)
>>> output shape : (None, 106, 106, 3)
```

1D Zero padding

class tensorlayer.layers.**ZeroPad1d**(*padding*, *name=None*)

The *ZeroPad1d* class is a 1D padding layer for signal [batch, length, channel].

Parameters

- **padding** (*int, or tuple of 2 ints*) –
  - If int, zeros to add at the beginning and end of the padding dimension (axis 1).
  - If tuple of 2 ints, zeros to add at the beginning and at the end of the padding dimension.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 1], name='input')
>>> pad1d = tl.layers.ZeroPad1d(padding=(2, 3))(net)
>>> print(pad1d)
>>> output shape : (None, 106, 1)
```

2D Zero padding

class tensorlayer.layers.**ZeroPad2d**(*padding*, *name=None*)

The *ZeroPad2d* class is a 2D padding layer for image [batch, height, width, channel].

Parameters

- **padding** (*int, or tuple of 2 ints, or tuple of 2 tuples of 2 ints.*) –
  - If int, the same symmetric padding is applied to width and height.
  - If tuple of 2 ints, interpreted as two different symmetric padding values for height and width as (symmetric_height_pad, symmetric_width_pad).
  - If tuple of 2 tuples of 2 ints, interpreted as ((top_pad, bottom_pad), (left_pad, right_pad)).
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 100, 3], name='input')
>>> pad2d = tl.layers.ZeroPad2d(padding=((3, 3), (4, 4)))(net)
>>> print(pad2d)
>>> output shape : (None, 106, 108, 3)
```

3D Zero padding

class tensorlayer.layers.**ZeroPad3d**(*padding*, *name=None*)

The *ZeroPad3d* class is a 3D padding layer for volume [batch, depth, height, width, channel].

Parameters

- **padding** (*int, or tuple of 2 ints, or tuple of 2 tuples of 2 ints.*) –
  - If int, the same symmetric padding is applied to width and height.
  - If tuple of 2 ints, interpreted as two different symmetric padding values for height and width as (symmetric_dim1_pad, symmetric_dim2_pad, symmetric_dim3_pad).
  - If tuple of 2 tuples of 2 ints, interpreted as ((left_dim1_pad, right_dim1_pad), (left_dim2_pad, right_dim2_pad), (left_dim3_pad, right_dim3_pad)).

- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 100, 100, 3], name='input')
>>> pad3d = tl.layers.ZeroPad3d(padding=((3, 3), (4, 4), (5, 5)))(net)
>>> print(pad3d)
>>> output shape : (None, 106, 108, 110, 3)
```

2.7.15 Pooling Layers

Pool Layer (Expert API)

Pooling layer for any dimensions and any pooling functions.

class tensorlayer.layers.**PoolLayer**(*filter_size=(1, 2, 2, 1)*, *strides=(1, 2, 2, 1)*, *padding='SAME'*, *pool=tensorflow.nn.max_pool*, *name=None*)

The *PoolLayer* class is a Pooling layer. You can choose `tf.nn.max_pool` and `tf.nn.avg_pool` for 2D input or `tf.nn.max_pool3d` and `tf.nn.avg_pool3d` for 3D input.

Parameters

- **filter_size** (*tuple of int*) – The size of the window for each dimension of the input tensor. Note that: len(filter_size) >= 4.
- **strides** (*tuple of int*) – The stride of the sliding window for each dimension of the input tensor. Note that: len(strides) >= 4.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **pool** (*pooling function*) – One of `tf.nn.max_pool`, `tf.nn.avg_pool`, `tf.nn.max_pool3d` and `f.nn.avg_pool3d`. See TensorFlow pooling APIs
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 50, 32], name='input')
>>> net = tl.layers.PoolLayer()(net)
>>> output shape : [None, 25, 25, 32]
```

1D Max pooling

class tensorlayer.layers.**MaxPool1d**(*filter_size=3*, *strides=2*, *padding='SAME'*, *data_format='channels_last'*, *dilation_rate=1*, *name=None*)

Max pooling for 1D signal.

Parameters

- **filter_size** (*int*) – Pooling window size.

- **strides** (*int*) – Stride of the pooling operation.
- **padding** (*str*) – The padding method: 'VALID' or 'SAME'.
- **data_format** (*str*) – One of channels_last (default, [batch, length, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 32], name='input')
>>> net = tl.layers.MaxPool1d(filter_size=3, strides=2, padding='SAME', name=
↪'maxpool1d')(net)
>>> output shape : [None, 25, 32]
```

1D Mean pooling

class tensorlayer.layers.**MeanPool1d**(*filter_size=3, strides=2, padding='SAME', data_format='channels_last', dilation_rate=1, name=None*)

Mean pooling for 1D signal.

Parameters

- **filter_size** (*int*) – Pooling window size.
- **strides** (*int*) – Strides of the pooling operation.
- **padding** (*str*) – The padding method: 'VALID' or 'SAME'.
- **data_format** (*str*) – One of channels_last (default, [batch, length, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 32], name='input')
>>> net = tl.layers.MeanPool1d(filter_size=3, strides=2, padding='SAME')(net)
>>> output shape : [None, 25, 32]
```

2D Max pooling

class tensorlayer.layers.**MaxPool2d**(*filter_size=(3, 3), strides=(2, 2), padding='SAME', data_format='channels_last', name=None*)

Max pooling for 2D image.

Parameters

- **filter_size** (*tuple of int*) – (height, width) for filter size.
- **strides** (*tuple of int*) – (height, width) for strides.

- **padding** (*str*) – The padding method: 'VALID' or 'SAME'.
- **data_format** (*str*) – One of channels_last (default, [batch, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 50, 32], name='input')
>>> net = tl.layers.MaxPool2d(filter_size=(3, 3), strides=(2, 2), padding='SAME
↪')(net)
>>> output shape : [None, 25, 25, 32]
```

2D Mean pooling

class tensorlayer.layers.**MeanPool2d**(*filter_size=(3, 3), strides=(2, 2), padding='SAME', data_format='channels_last', name=None*)

Mean pooling for 2D image [batch, height, width, channel].

Parameters

- **filter_size** (*tuple of int*) – (height, width) for filter size.
- **strides** (*tuple of int*) – (height, width) for strides.
- **padding** (*str*) – The padding method: 'VALID' or 'SAME'.
- **data_format** (*str*) – One of channels_last (default, [batch, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 50, 32], name='input')
>>> net = tl.layers.MeanPool2d(filter_size=(3, 3), strides=(2, 2), padding='SAME
↪')(net)
>>> output shape : [None, 25, 25, 32]
```

3D Max pooling

class tensorlayer.layers.**MaxPool3d**(*filter_size=(3, 3, 3), strides=(2, 2, 2), padding='VALID', data_format='channels_last', name=None*)

Max pooling for 3D volume.

Parameters

- **filter_size** (*tuple of int*) – Pooling window size.
- **strides** (*tuple of int*) – Strides of the pooling operation.
- **padding** (*str*) – The padding method: 'VALID' or 'SAME'.

- **data_format** (*str*) – One of channels_last (default, [batch, depth, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Returns A max pooling 3-D layer with a output rank as 5.

Return type tf.Tensor

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 50, 50, 32], name='input')
>>> net = tl.layers.MaxPool3d(filter_size=(3, 3, 3), strides=(2, 2, 2), padding=
↪'SAME')(net)
>>> output shape : [None, 25, 25, 25, 32]
```

3D Mean pooling

class tensorlayer.layers.**MeanPool3d**(*filter_size=(3, 3, 3)*, *strides=(2, 2, 2)*, *padding='VALID'*, *data_format='channels_last'*, *name=None*)

Mean pooling for 3D volume.

Parameters

- **filter_size** (*tuple of int*) – Pooling window size.
- **strides** (*tuple of int*) – Strides of the pooling operation.
- **padding** (*str*) – The padding method: 'VALID' or 'SAME'.
- **data_format** (*str*) – One of channels_last (default, [batch, depth, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Returns A mean pooling 3-D layer with a output rank as 5.

Return type tf.Tensor

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 50, 50, 50, 32], name='input')
>>> net = tl.layers.MeanPool3d(filter_size=(3, 3, 3), strides=(2, 2, 2), padding=
↪'SAME')(net)
>>> output shape : [None, 25, 25, 25, 32]
```

1D Global Max pooling

class tensorlayer.layers.**GlobalMaxPool1d**(*data_format='channels_last'*, *name=None*)

The *GlobalMaxPool1d* class is a 1D Global Max Pooling layer.

Parameters

- **data_format** (*str*) – One of channels_last (default, [batch, length, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 30], name='input')
>>> net = tl.layers.GlobalMaxPool1d()(net)
>>> output shape : [None, 30]
```

1D Global Mean pooling

class tensorlayer.layers.**GlobalMeanPool1d**(*data_format='channels_last'*, *name=None*)

The *GlobalMeanPool1d* class is a 1D Global Mean Pooling layer.

Parameters

- **data_format** (*str*) – One of channels_last (default, [batch, length, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 30], name='input')
>>> net = tl.layers.GlobalMeanPool1d()(net)
>>> output shape : [None, 30]
```

2D Global Max pooling

class tensorlayer.layers.**GlobalMaxPool2d**(*data_format='channels_last'*, *name=None*)

The *GlobalMaxPool2d* class is a 2D Global Max Pooling layer.

Parameters

- **data_format** (*str*) – One of channels_last (default, [batch, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 100, 30], name='input')
>>> net = tl.layers.GlobalMaxPool2d()(net)
>>> output shape : [None, 30]
```

2D Global Mean pooling

class tensorlayer.layers.**GlobalMeanPool2d**(*data_format='channels_last'*, *name=None*)

The *GlobalMeanPool2d* class is a 2D Global Mean Pooling layer.

Parameters

- **data_format** (*str*) – One of channels_last (default, [batch, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 100, 30], name='input')
>>> net = tl.layers.GlobalMeanPool2d()(net)
>>> output shape : [None, 30]
```

3D Global Max pooling

class tensorlayer.layers.**GlobalMaxPool3d**(*data_format='channels_last'*, *name=None*)

The *GlobalMaxPool3d* class is a 3D Global Max Pooling layer.

Parameters

- **data_format** (*str*) – One of channels_last (default, [batch, depth, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 100, 100, 30], name='input')
>>> net = tl.layers.GlobalMaxPool3d()(net)
>>> output shape : [None, 30]
```

3D Global Mean pooling

class tensorlayer.layers.**GlobalMeanPool3d**(*data_format='channels_last'*, *name=None*)

The *GlobalMeanPool3d* class is a 3D Global Mean Pooling layer.

Parameters

- **data_format** (*str*) – One of channels_last (default, [batch, depth, height, width, channel]) or channels_first. The ordering of the dimensions in the inputs.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 100, 100, 100, 30], name='input')
>>> net = tl.layers.GlobalMeanPool3d()(net)
>>> output shape : [None, 30]
```

2D Corner pooling

class tensorlayer.layers.**CornerPool2d**(*mode='TopLeft'*, *name=None*)

Corner pooling for 2D image [batch, height, width, channel], see here.

Parameters

- **mode** (*str*) – TopLeft for the top left corner, Bottomright for the bottom right corner.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([None, 32, 32, 8], name='input')
>>> net = tl.layers.CornerPool2d(mode='TopLeft',name='cornerpool2d')(net)
>>> output shape : [None, 32, 32, 8]
```

2.7.16 Quantized Nets

This is an experimental API package for building Quantized Neural Networks. We are using matrix multiplication rather than add-minus and bit-count operation at the moment. Therefore, these APIs would not speed up the inferencing, for production, you can train model via TensorLayer and deploy the model into other customized C/C++ implementation (We probably provide users an extra C/C++ binary net framework that can load model from TensorLayer).

Note that, these experimental APIs can be changed in the future.

Sign

class tensorlayer.layers.**Sign**(*name=None*)

The `SignLayer` class is for quantizing the layer outputs to -1 or 1 while inferencing.

Parameters **name** (*a str*) – A unique layer name.

Scale

class tensorlayer.layers.**Scale**(*init_scale=0.05*, *name='scale'*)

The *Scale* class is to multiple a trainable scale value to the layer outputs. Usually be used on the output of binary net.

Parameters

- **init_scale** (*float*) – The initial value for the scale factor.

- **name** (*a str*) – A unique layer name.
- **Examples** –
- **----------** –
- **inputs = tl.layers.Input([8, 3])** (*>>>*) –
- **dense = tl.layers.Dense(n_units=10)(inputs)** (*>>>*) –
- **outputs = tl.layers.Scale(init_scale=0.5)(dense)** (*>>>*) –
- **model = tl.models.Model(inputs=inputs, outputs=[dense, outputs])** (*>>>*) –
- **dense_out, scale_out = model(data, is_train=True)** (*>>>*) –

Binary Dense Layer

class tensorlayer.layers.**BinaryDense**(*n_units=100, act=None, use_gemm=False, W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The `BinaryDense` class is a binary fully connected layer, which weights are either -1 or 1 while inferencing.

Note that, the bias vector would not be binarized.

Parameters

- **n_units** (*int*) – The number of units of this layer.
- **act** (*activation function*) – The activation function of this layer, usually set to `tf.act.sign` or apply `Sign` after `BatchNorm`.
- **use_gemm** (*boolean*) – If True, use gemm instead of `tf.matmul` for inference. (TODO).
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of channels of the previous layer. If None, it will be automatically detected when the layer is forwarded for the first time.
- **name** (*None or str*) – A unique layer name.

Binary (De)Convolutions

BinaryConv2d

class tensorlayer.layers.**BinaryConv2d**(*n_filter=32, filter_size=(3, 3), strides=(1, 1), act=None, padding='SAME', use_gemm=False, data_format='channels_last', dilation_rate=(1, 1), W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The `BinaryConv2d` class is a 2D binary CNN layer, which weights are either -1 or 1 while inference.

Note that, the bias vector would not be binarized.

Parameters

- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **strides** (*tuple of int*) – The sliding window strides of corresponding input dimensions. It must be in the same order as the `shape` parameter.
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **use_gemm** (*boolean*) – If True, use gemm instead of `tf.matmul` for inference. TODO: support gemm
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **dilation_rate** (*tuple of int*) – Specifying the dilation rate to use for dilated convolution.
- **W_init** (*initializer*) – The initializer for the the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 100, 100, 32], name='input')
>>> binaryconv2d = tl.layers.QuanConv2d(
...     n_filter=64, filter_size=(3, 3), strides=(2, 2), act=tf.nn.relu, in_
↪channels=32, name='binaryconv2d'
... )(net)
>>> print(binaryconv2d)
>>> output shape : (8, 50, 50, 64)
```

Ternary Dense Layer

TernaryDense

class tensorlayer.layers.**TernaryDense**(*n_units=100, act=None, use_gemm=False, W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The *TernaryDense* class is a ternary fully connected layer, which weights are either -1 or 1 or 0 while inference.

Note that, the bias vector would not be tenaried.

Parameters

- **n_units** (*int*) – The number of units of this layer.
- **act** (*activation function*) – The activation function of this layer, usually set to `tf.act.sign` or apply `SignLayer` after `BatchNormLayer`.

- **use_gemm** (*boolean*) – If True, use gemm instead of `tf.matmul` for inference. (TODO).
- **W_init** (*initializer*) – The initializer for the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of channels of the previous layer. If None, it will be automatically detected when the layer is forwarded for the first time.
- **name** (*None or str*) – A unique layer name.

Ternary Convolutions

TernaryConv2d

class tensorlayer.layers.**TernaryConv2d**(*n_filter=32, filter_size=(3, 3), strides=(1, 1), act=None, padding='SAME', use_gemm=False, data_format='channels_last', dilation_rate=(1, 1), W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The `TernaryConv2d` class is a 2D ternary CNN layer, which weights are either -1 or 1 or 0 while inference.

Note that, the bias vector would not be tenarized.

Parameters

- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **strides** (*tuple of int*) – The sliding window strides of corresponding input dimensions. It must be in the same order as the `shape` parameter.
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **use_gemm** (*boolean*) – If True, use gemm instead of `tf.matmul` for inference. TODO: support gemm
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **dilation_rate** (*tuple of int*) – Specifying the dilation rate to use for dilated convolution.
- **W_init** (*initializer*) – The initializer for the the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 12, 12, 32], name='input')
>>> ternaryconv2d = tl.layers.QuanConv2d(
...     n_filter=64, filter_size=(5, 5), strides=(1, 1), act=tf.nn.relu, padding=
↪'SAME', name='ternaryconv2d'
... )(net)
>>> print(ternaryconv2d)
>>> output shape : (8, 12, 12, 64)
```

DoReFa Convolutions

DorefaConv2d

class tensorlayer.layers.**DorefaConv2d**(*bitW=1, bitA=3, n_filter=32, filter_size=(3, 3), strides=(1, 1), act=None, padding='SAME', use_gemm=False, data_format='channels_last', dilation_rate=(1, 1), W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The `DorefaConv2d` class is a 2D quantized convolutional layer, which weights are 'bitW' bits and the output of the previous layer are 'bitA' bits while inferencing.

Note that, the bias vector would not be binarized.

Parameters

- **bitW** (*int*) – The bits of this layer's parameter
- **bitA** (*int*) – The bits of the output of previous layer
- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **strides** (*tuple of int*) – The sliding window strides of corresponding input dimensions. It must be in the same order as the `shape` parameter.
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **use_gemm** (*boolean*) – If True, use gemm instead of `tf.matmul` for inferencing. TODO: support gemm
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **dilation_rate** (*tuple of int*) – Specifying the dilation rate to use for dilated convolution.
- **W_init** (*initializer*) – The initializer for the the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 12, 12, 32], name='input')
>>> dorefaconv2d = tl.layers.QuanConv2d(
...     n_filter=32, filter_size=(5, 5), strides=(1, 1), act=tf.nn.relu, padding=
↪'SAME', name='dorefaconv2d'
... )(net)
>>> print(dorefaconv2d)
>>> output shape : (8, 12, 12, 32)
```

DoReFa Convolutions

DorefaConv2d

class tensorlayer.layers.**DorefaConv2d**(*bitW=1, bitA=3, n_filter=32, filter_size=(3, 3), strides=(1, 1), act=None, padding='SAME', use_gemm=False, data_format='channels_last', dilation_rate=(1, 1), W_init=<tensorlayer.initializers.TruncatedNormal object>, b_init=<tensorlayer.initializers.Constant object>, in_channels=None, name=None*)

The `DorefaConv2d` class is a 2D quantized convolutional layer, which weights are 'bitW' bits and the output of the previous layer are 'bitA' bits while inferencing.

Note that, the bias vector would not be binarized.

Parameters

- **bitW** (*int*) – The bits of this layer's parameter
- **bitA** (*int*) – The bits of the output of previous layer
- **n_filter** (*int*) – The number of filters.
- **filter_size** (*tuple of int*) – The filter size (height, width).
- **strides** (*tuple of int*) – The sliding window strides of corresponding input dimensions. It must be in the same order as the `shape` parameter.
- **act** (*activation function*) – The activation function of this layer.
- **padding** (*str*) – The padding algorithm type: "SAME" or "VALID".
- **use_gemm** (*boolean*) – If True, use gemm instead of `tf.matmul` for inferencing. TODO: support gemm
- **data_format** (*str*) – "channels_last" (NHWC, default) or "channels_first" (NCHW).
- **dilation_rate** (*tuple of int*) – Specifying the dilation rate to use for dilated convolution.
- **W_init** (*initializer*) – The initializer for the the weight matrix.
- **b_init** (*initializer or None*) – The initializer for the the bias vector. If None, skip biases.
- **in_channels** (*int*) – The number of in channels.
- **name** (*None or str*) – A unique layer name.

Examples

With TensorLayer

```
>>> net = tl.layers.Input([8, 12, 12, 32], name='input')
>>> dorefaconv2d = tl.layers.QuanConv2d(
...     n_filter=32, filter_size=(5, 5), strides=(1, 1), act=tf.nn.relu, padding=
↪'SAME', name='dorefaconv2d'
... )(net)
>>> print(dorefaconv2d)
>>> output shape : (8, 12, 12, 32)
```

2.7.17 Recurrent Layers

Common Recurrent layer

All recurrent layers can implement any type of RNN cell by feeding different cell function (LSTM, GRU etc).

RNN layer

class tensorlayer.layers.**RNN**(*cell*, *return_last_output=False*, *return_seq_2d=False*, *return_last_state=True*, *in_channels=None*, *name=None*)

The *RNN* class is a fixed length recurrent layer for implementing simple RNN, LSTM, GRU and etc.

Parameters

- **cell** (*TensorFlow cell function*) –

  A RNN cell implemented by tf.keras

  - E.g. tf.keras.layers.SimpleRNNCell, tf.keras.layers.LSTMCell, tf.keras.layers.GRUCell
  - Note TF2.0+, TF1.0+ and TF1.0- are different

- **return_last_output** (*boolean*) –

  Whether return last output or all outputs in a sequence.

  - If True, return the last output, "Sequence input and single output"
  - If False, return all outputs, "Synced sequence input and output"
  - In other word, if you want to stack more RNNs on this layer, set to False

  In a dynamic model, *return_last_output* can be updated when it is called in customised forward(). By default, *False*.

- **return_seq_2d** (*boolean*) –

  Only consider this argument when *return_last_output* is *False*

  - If True, return 2D Tensor [batch_size * n_steps, n_hidden], for stacking Dense layer after it.
  - If False, return 3D Tensor [batch_size, n_steps, n_hidden], for stacking multiple RNN after it.

  In a dynamic model, *return_seq_2d* can be updated when it is called in customised forward(). By default, *False*.

- **return_last_state** (*boolean*) – Whether to return the last state of the RNN cell. The state is a list of Tensor. For simple RNN and GRU, last_state = [last_output]; For LSTM, last_state = [last_output, last_cell_state]
  - If True, the layer will return outputs and the final state of the cell.
  - If False, the layer will return outputs only.

  In a dynamic model, *return_last_state* can be updated when it is called in customised forward(). By default, *False*.
- **in_channels** (*int*) – Optional, the number of channels of the previous layer which is normally the size of embedding. If given, the layer will be built when init. If None, it will be automatically detected when the layer is forwarded for the first time.
- **name** (*str*) – A unique layer name.

Examples

For synced sequence input and output, see PTB example

A simple regression model below. >>> inputs = tl.layers.Input([batch_size, num_steps, embedding_size]) >>> rnn_out, lstm_state = tl.layers.RNN(>>> cell=tf.keras.layers.LSTMCell(units=hidden_size, dropout=0.1), >>> in_channels=embedding_size, >>> return_last_output=True, return_last_state=True, name='lstmrnn' >>>)(inputs) >>> outputs = tl.layers.Dense(n_units=1)(rnn_out) >>> rnn_model = tl.models.Model(inputs=inputs, outputs=[outputs, rnn_state[0], rnn_state[1]], name='rnn_model') >>> # If LSTMCell is applied, the rnn_state is [h, c] where h the hidden state and c the cell state of LSTM.

A stacked RNN model. >>> inputs = tl.layers.Input([batch_size, num_steps, embedding_size]) >>> rnn_out1 = tl.layers.RNN(>>> cell=tf.keras.layers.SimpleRNNCell(units=hidden_size, dropout=0.1), >>> return_last_output=False, return_seq_2d=False, return_last_state=False >>>)(inputs) >>> rnn_out2 = tl.layers.RNN(>>> cell=tf.keras.layers.SimpleRNNCell(units=hidden_size, dropout=0.1), >>> return_last_output=True, return_last_state=False >>>)(rnn_out1) >>> outputs = tl.layers.Dense(n_units=1)(rnn_out2) >>> rnn_model = tl.models.Model(inputs=inputs, outputs=outputs)

Notes

Input dimension should be rank 3 : [batch_size, n_steps, n_features], if no, please see layer `Reshape`.

Bidirectional layer

class `tensorlayer.layers.`**`BiRNN`**(*fw_cell*, *bw_cell*, *return_seq_2d=False*, *return_last_state=False*, *in_channels=None*, *name=None*)

The `BiRNN` class is a fixed length Bidirectional recurrent layer.

Parameters

- **fw_cell** (*TensorFlow cell function for forward direction*) – A RNN cell implemented by tf.keras, e.g. tf.keras.layers.SimpleRNNCell, tf.keras.layers.LSTMCell, tf.keras.layers.GRUCell. Note TF2.0+, TF1.0+ and TF1.0- are different
- **bw_cell** (TensorFlow cell function for backward direction similar with *fw_cell*) –
- **return_seq_2d** (*boolean.*) – If True, return 2D Tensor [batch_size * n_steps, n_hidden], for stacking Dense layer after it. If False, return 3D Tensor [batch_size, n_steps,

n_hidden], for stacking multiple RNN after it. In a dynamic model, *return_seq_2d* can be updated when it is called in customised forward(). By default, *False*.

- **return_last_state** (*boolean*) –

  Whether to return the last state of the two cells. The state is a list of Tensor.

  – If True, the layer will return outputs, the final state of *fw_cell* and the final state of *bw_cell*.

  – If False, the layer will return outputs only.

  In a dynamic model, *return_last_state* can be updated when it is called in customised forward(). By default, *False*.

- **in_channels** (*int*) – Optional, the number of channels of the previous layer which is normally the size of embedding. If given, the layer will be built when init. If None, it will be automatically detected when the layer is forwarded for the first time.

- **name** (*str*) – A unique layer name.

Examples

A simple regression model below. >>> inputs = tl.layers.Input([batch_size, num_steps, embedding_size]) >>> # the fw_cell and bw_cell can be different >>> rnnlayer = tl.layers.BiRNN(>>> fw_cell=tf.keras.layers.SimpleRNNCell(units=hidden_size, dropout=0.1), >>> bw_cell=tf.keras.layers.SimpleRNNCell(units=hidden_size + 1, dropout=0.1), >>> return_seq_2d=True, return_last_state=True >>>) >>> # if return_last_state=True, the final state of the two cells will be returned together with the outputs >>> # if return_last_state=False, only the outputs will be returned >>> rnn_out, rnn_fw_state, rnn_bw_state = rnnlayer(inputs) >>> # if the BiRNN is followed by a Dense, return_seq_2d should be True. >>> # if the BiRNN is followed by other RNN, return_seq_2d can be False. >>> dense = tl.layers.Dense(n_units=1)(rnn_out) >>> outputs = tl.layers.Reshape([-1, num_steps])(dense) >>> rnn_model = tl.models.Model(inputs=inputs, outputs=[outputs, rnn_out, rnn_fw_state[0], rnn_bw_state[0]])

A stacked BiRNN model. >>> inputs = tl.layers.Input([batch_size, num_steps, embedding_size]) >>> rnn_out1 = tl.layers.BiRNN(>>> fw_cell=tf.keras.layers.SimpleRNNCell(units=hidden_size, dropout=0.1), >>> bw_cell=tf.keras.layers.SimpleRNNCell(units=hidden_size + 1, dropout=0.1), >>> return_seq_2d=False, return_last_state=False >>>)(inputs) >>> rnn_out2 = tl.layers.BiRNN(>>> fw_cell=tf.keras.layers.SimpleRNNCell(units=hidden_size, dropout=0.1), >>> bw_cell=tf.keras.layers.SimpleRNNCell(units=hidden_size + 1, dropout=0.1), >>> return_seq_2d=True, return_last_state=False >>>)(rnn_out1) >>> dense = tl.layers.Dense(n_units=1)(rnn_out2) >>> outputs = tl.layers.Reshape([-1, num_steps])(dense) >>> rnn_model = tl.models.Model(inputs=inputs, outputs=outputs)

Notes

Input dimension should be rank 3 : [batch_size, n_steps, n_features]. If not, please see layer `Reshape`.

Advanced Ops for Dynamic RNN

These operations usually be used inside Dynamic RNN layer, they can compute the sequence lengths for different situation and get the last RNN outputs by indexing.

Compute Sequence length 1

`tensorlayer.layers.retrieve_seq_length_op(data)`

An op to compute the length of a sequence from input shape of [batch_size, n_step(max), n_features], it can be used when the features of padding (on right hand side) are all zeros.

Parameters **data** (*tensor*) – [batch_size, n_step(max), n_features] with zero padding on right hand side.

Examples

Single feature

```
>>> data = [[[1],[2],[0],[0],[0]],
>>>          [[1],[2],[3],[0],[0]],
>>>          [[1],[2],[6],[1],[0]]]
>>> data = tf.convert_to_tensor(data, dtype=tf.float32)
>>> length = tl.layers.retrieve_seq_length_op(data)
[2 3 4]
```

Multiple features

```
>>> data = [[[1,2],[2,2],[1,2],[1,2],[0,0]],
>>>           [[2,3],[2,4],[3,2],[0,0],[0,0]],
>>>           [[3,3],[2,2],[5,3],[1,2],[0,0]]]
>>> data = tf.convert_to_tensor(data, dtype=tf.float32)
>>> length = tl.layers.retrieve_seq_length_op(data)
[4 3 4]
```

References

Borrow from TFlearn.

Compute Sequence length 2

`tensorlayer.layers.retrieve_seq_length_op2(data)`

An op to compute the length of a sequence, from input shape of [batch_size, n_step(max)], it can be used when the features of padding (on right hand side) are all zeros.

Parameters **data** (*tensor*) – [batch_size, n_step(max)] with zero padding on right hand side.

Examples

```
>>> data = [[1,2,0,0,0],
>>>          [1,2,3,0,0],
>>>          [1,2,6,1,0]]
>>> data = tf.convert_to_tensor(data, dtype=tf.float32)
>>> length = tl.layers.retrieve_seq_length_op2(data)
[2 3 4]
```

Compute Sequence length 3

`tensorlayer.layers.`**`retrieve_seq_length_op3`**`(`*data*, *pad_val=0*`)`

An op to compute the length of a sequence, the data shape can be [batch_size, n_step(max)] or [batch_size, n_step(max), n_features].

If the data has type of tf.string and pad_val is assigned as empty string (‘’), this op will compute the length of the string sequence.

Parameters

- **data** (*tensor*) – [batch_size, n_step(max)] or [batch_size, n_step(max), n_features] with zero padding on the right hand side.
- **pad_val** – By default 0. If the data is tf.string, please assign this as empty string (‘’)

Examples

```
>>> data = [[[1],[2],[0],[0],[0]],
>>>          [[1],[2],[3],[0],[0]],
>>>          [[1],[2],[6],[1],[0]]]
>>> data = tf.convert_to_tensor(data, dtype=tf.float32)
>>> length = tl.layers.retrieve_seq_length_op3(data)
[2, 3, 4]
>>> data = [[[1,2],[2,2],[1,2],[1,2],[0,0]],
>>>          [[2,3],[2,4],[3,2],[0,0],[0,0]],
>>>          [[3,3],[2,2],[5,3],[1,2],[0,0]]]
>>> data = tf.convert_to_tensor(data, dtype=tf.float32)
>>> length = tl.layers.retrieve_seq_length_op3(data)
[4, 3, 4]
>>> data = [[1,2,0,0,0],
>>>          [1,2,3,0,0],
>>>          [1,2,6,1,0]]
>>> data = tf.convert_to_tensor(data, dtype=tf.float32)
>>> length = tl.layers.retrieve_seq_length_op3(data)
[2, 3, 4]
>>> data = [['hello','world','','',''],
>>>          ['hello','world','tensorlayer','',''],
>>>          ['hello','world','tensorlayer','2.0','']]
>>> data = tf.convert_to_tensor(data, dtype=tf.string)
>>> length = tl.layers.retrieve_seq_length_op3(data, pad_val='')
[2, 3, 4]
```

2.7.18 Shape Layers

Flatten Layer

class `tensorlayer.layers.`**`Flatten`**`(`*name=None*`)`

A layer that reshapes high-dimension input into a vector.

Then we often apply Dense, RNN, Concat and etc on the top of a flatten layer. [batch_size, mask_row, mask_col, n_mask] —> [batch_size, mask_row * mask_col * n_mask]

Parameters **name** (*None or str*) – A unique layer name.

Examples

```
>>> x = tl.layers.Input([8, 4, 3], name='input')
>>> y = tl.layers.Flatten(name='flatten')(x)
[8, 12]
```

Reshape Layer

class tensorlayer.layers.**Reshape**(*shape*, *name=None*)
A layer that reshapes a given tensor.

Parameters

- **shape** (*tuple of int*) – The output shape, see `tf.reshape`.
- **name** (*str*) – A unique layer name.

Examples

```
>>> x = tl.layers.Input([8, 4, 3], name='input')
>>> y = tl.layers.Reshape(shape=[-1, 12], name='reshape')(x)
(8, 12)
```

Transpose Layer

class tensorlayer.layers.**Transpose**(*perm=None*, *conjugate=False*, *name=None*)
A layer that transposes the dimension of a tensor.

See tf.transpose() .

Parameters

- **perm** (*list of int*) – The permutation of the dimensions, similar with `numpy.transpose`. If None, it is set to (n-1…0), where n is the rank of the input tensor.
- **conjugate** (*bool*) – By default False. If True, returns the complex conjugate of complex numbers (and transposed) For example [[1+1j, 2+2j]] –> [[1-1j], [2-2j]]
- **name** (*str*) – A unique layer name.

Examples

```
>>> x = tl.layers.Input([8, 4, 3], name='input')
>>> y = tl.layers.Transpose(perm=[0, 2, 1], conjugate=False, name='trans')(x)
(8, 3, 4)
```

Shuffle Layer

class tensorlayer.layers.**Shuffle**(*group*, *name=None*)
A layer that shuffle a 2D image [batch, height, width, channel], see here.

Parameters

- **group** (*int*) – The number of groups.

- **name** (*str*) – A unique layer name.

Examples

```
>>> x = tl.layers.Input([1, 16, 16, 8], name='input')
>>> y = tl.layers.Shuffle(group=2, name='shuffle')(x)
(1, 16, 16, 8)
```

2.7.19 Spatial Transformer

2D Affine Transformation

class tensorlayer.layers.**SpatialTransformer2dAffine**(*in_channels=None, out_size=(40, 40), name=None*)

The *SpatialTransformer2dAffine* class is a 2D Spatial Transformer Layer for 2D Affine Transformation.

Parameters

- **in_channels** –
- **out_size** (*tuple of int or None*) –
  - The size of the output of the network (height, width), the feature maps will be resized by this.
- **name** (*str*) –
  - A unique layer name.

References

- Spatial Transformer Networks
- TensorFlow/Models

2D Affine Transformation function

tensorlayer.layers.**transformer**(*U, theta, out_size, name='SpatialTransformer2dAffine'*)

Spatial Transformer Layer for 2D Affine Transformation , see *SpatialTransformer2dAffine* class.

Parameters

- **U** (*list of float*) – The output of a convolutional net should have the shape [num_batch, height, width, num_channels].
- **theta** (*float*) – The output of the localisation network should be [num_batch, 6], value range should be [0, 1] (via tanh).
- **out_size** (*tuple of int*) – The size of the output of the network (height, width)
- **name** (*str*) – Optional function name

Returns The transformed tensor.

Return type Tensor

References

- Spatial Transformer Networks
- TensorFlow/Models

Notes

To initialize the network to the identity transform init.

```
>>> import tensorflow as tf
>>> # ``theta`` to
>>> identity = np.array([[1., 0., 0.], [0., 1., 0.]])
>>> identity = identity.flatten()
>>> theta = tf.Variable(initial_value=identity)
```

Batch 2D Affine Transformation function

`tensorlayer.layers.`**`batch_transformer`**(*U*, *thetas*, *out_size*, *name='BatchSpatialTransformer2dAffine'*)

Batch Spatial Transformer function for 2D Affine Transformation.

Parameters

- **U** (*list of float*) – tensor of inputs [batch, height, width, num_channels]
- **thetas** (*list of float*) – a set of transformations for each input [batch, num_transforms, 6]
- **out_size** (*list of int*) – the size of the output [out_height, out_width]
- **name** (*str*) – optional function name

Returns Tensor of size [batch * num_transforms, out_height, out_width, num_channels]

Return type float

2.7.20 Stack Layer

Stack Layer

class `tensorlayer.layers.`**`Stack`**(*axis=1*, *name=None*)

The *Stack* class is a layer for stacking a list of rank-R tensors into one rank-(R+1) tensor, see tf.stack().

Parameters

- **axis** (*int*) – New dimension along which to stack.
- **name** (*str*) – A unique layer name.

Examples

```
>>> import tensorflow as tf
>>> import tensorlayer as tl
>>> ni = tl.layers.Input([None, 784], name='input')
>>> net1 = tl.layers.Dense(10, name='dense1')(ni)
>>> net2 = tl.layers.Dense(10, name='dense2')(ni)
```

(continues on next page)

(continued from previous page)

```
>>> net3 = tl.layers.Dense(10, name='dense3')(ni)
>>> net = tl.layers.Stack(axis=1, name='stack')([net1, net2, net3])
(?, 3, 10)
```

Unstack Layer

class tensorlayer.layers.**UnStack**(*num=None*, *axis=0*, *name=None*)

The *UnStack* class is a layer for unstacking the given dimension of a rank-R tensor into rank-(R-1) tensors., see tf.unstack().

Parameters

- **num** (*int or None*) – The length of the dimension axis. Automatically inferred if None (the default).
- **axis** (*int*) – Dimension along which axis to concatenate.
- **name** (*str*) – A unique layer name.

Returns The list of layer objects unstacked from the input.

Return type list of *Layer*

Examples

```
>>> ni = Input([4, 10], name='input')
>>> nn = Dense(n_units=5)(ni)
>>> nn = UnStack(axis=1)(nn)  # unstack in channel axis
>>> len(nn)  # 5
>>> nn[0].shape  # (4,)
```

2.7.21 Helper Functions

Flatten tensor

tensorlayer.layers.**flatten_reshape**(*variable*, *name='flatten'*)

Reshapes a high-dimension vector input.

[batch_size, mask_row, mask_col, n_mask] —> [batch_size, mask_row x mask_col x n_mask]

Parameters

- **variable** (*TensorFlow variable or tensor*) – The variable or tensor to be flatten.
- **name** (*str*) – A unique layer name.

Returns Flatten Tensor

Return type Tensor

Initialize RNN state

`tensorlayer.layers.initialize_rnn_state(state, feed_dict=None)`
Returns the initialized RNN state. The inputs are *LSTMStateTuple* or *State* of *RNNCells*, and an optional *feed_dict*.

Parameters

- **state** (*RNN state.*) – The TensorFlow's RNN state.
- **feed_dict** (*dictionary*) – Initial RNN state; if None, returns zero state.

Returns The TensorFlow's RNN state.

Return type RNN state

Remove repeated items in a list

`tensorlayer.layers.list_remove_repeat(x)`
Remove the repeated items in a list, and return the processed list. You may need it to create merged layer like Concat, Elementwise and etc.

Parameters **x** (*list*) – Input

Returns A list that after removing it's repeated items

Return type list

Examples

```
>>> l = [2, 3, 4, 2, 3]
>>> l = list_remove_repeat(l)
[2, 3, 4]
```

2.8 API - Models

TensorLayer provides many pretrained models, you can easily use the whole or a part of the pretrained models via these APIs.

Model([inputs, outputs, name])	The *Model* class represents a neural network.
VGG16([pretrained, end_with, mode, name])	Pre-trained VGG16 model.
VGG19([pretrained, end_with, mode, name])	Pre-trained VGG19 model.
SqueezeNetV1([pretrained, end_with, name])	Pre-trained SqueezeNetV1 model (static mode).
MobileNetV1([pretrained, end_with, name])	Pre-trained MobileNetV1 model (static mode).

2.8.1 Base Model

class `tensorlayer.models.Model(inputs=None, outputs=None, name=None)`
The *Model* class represents a neural network.

It should be subclassed when implementing a dynamic model, where 'forward' method must be overwritten. Otherwise, please specify 'inputs' tensor(s) and 'outputs' tensor(s) to create a static model. In that case, 'inputs' tensors should come from tl.layers.Input().

Parameters

- **inputs** (*a Layer or list of Layer*) – The input(s) to the model.
- **outputs** (*a Layer or list of Layer*) – The output(s) to the model.
- **name** (*None or str*) – The name of the model.

__init__(*self*, *inputs=None*, *outputs=None*, *name=None*)
Initializing the Model.

inputs()
Get input tensors to this network (only avaiable for static model).

outputs()
Get output tensors to this network (only avaiable for static model).

__call__(*inputs*, *is_train=None*, ***kwargs*)
Forward input tensors through this network.

all_layers()
Get all layer objects of this network in a list of layers.

weights()
Get the weights of this network in a list of tensors.

train()
Set this network in training mode. (affect layers e.g. Dropout, BatchNorm).

eval()
Set this network in evaluation mode.

as_layer()
Set this network as a ModelLayer so that it can be integrated into another Model.

release_memory()
Release the memory that was taken up by tensors which are maintained by this network.

save_weights(*self*, *filepath*, *format='hdf5'*)
Save the weights of this network in a given format.

load_weights(*self*, *filepath*, *format=None*, *in_order=True*, *skip=False*)
Load weights into this network from a specified file.

save(*self*, *filepath*, *save_weights=True*)
Save the network with/without weights.

load(*filepath*, *save_weights=True*)
Load the network with/without weights.

Examples

```
>>> import tensorflow as tf
>>> import numpy as np
>>> from tensorlayer.layers import Input, Dense, Dropout
>>> from tensorlayer.models import Model
```

Define static model

```
>>> class CustomModel(Model):
>>>     def __init__(self):
>>>         super(CustomModel, self).__init__()
>>>         self.dense1 = Dense(n_units=800, act=tf.nn.relu, in_channels=784)
>>>         self.dropout1 = Dropout(keep=0.8)
>>>         self.dense2 = Dense(n_units=10, in_channels=800)
>>>     def forward(self, x):
>>>         z = self.dense1(x)
>>>         z = self.dropout1(z)
>>>         z = self.dense2(z)
>>>         return z
>>> M_dynamic = CustomModel()
```

Define static model

```
>>> ni = Input([None, 784])
>>> nn = Dense(n_units=800, act=tf.nn.relu)(ni)
>>> nn = Dropout(keep=0.8)(nn)
>>> nn = Dense(n_units=10, act=tf.nn.relu)(nn)
>>> M_static = Model(inputs=ni, outputs=nn, name="mlp")
```

Get network information

```
>>> print(M_static)
... Model(
...   (_inputlayer): Input(shape=[None, 784], name='_inputlayer')
...   (dense): Dense(n_units=800, relu, in_channels='784', name='dense')
...   (dropout): Dropout(keep=0.8, name='dropout')
...   (dense_1): Dense(n_units=10, relu, in_channels='800', name='dense_1')
... )
```

Forwarding through this network

```
>>> data = np.random.normal(size=[16, 784]).astype(np.float32)
>>> outputs_d = M_dynamic(data)
>>> outputs_s = M_static(data)
```

Save and load weights

```
>>> M_static.save_weights('./model_weights.h5')
>>> M_static.load_weights('./model_weights.h5')
```

Save and load the model

```
>>> M_static.save('./model.h5')
>>> M = Model.load('./model.h5')
```

Convert model to layer

```
>>> M_layer = M_static.as_layer()
```

2.8.2 VGG16

`tensorlayer.models.`**`VGG16`**(*pretrained=False*, *end_with='outputs'*, *mode='dynamic'*, *name=None*)

Pre-trained VGG16 model.

Parameters

- **pretrained** (*boolean*) – Whether to load pretrained weights. Default False.
- **end_with** (*str*) – The end point of the model. Default `fc3_relu` i.e. the whole model.
- **mode** (*str.*) – Model building mode, 'dynamic' or 'static'. Default 'dynamic'.
- **name** (*None or str*) – A unique layer name.

Examples

Classify ImageNet classes with VGG16, see tutorial_models_vgg.py With TensorLayer

```
>>> # get the whole model, without pre-trained VGG parameters
>>> vgg = tl.models.vgg16()
>>> # get the whole model, restore pre-trained VGG parameters
>>> vgg = tl.models.vgg16(pretrained=True)
>>> # use for inferencing
>>> output = vgg(img, is_train=False)
>>> probs = tf.nn.softmax(output)[0].numpy()
```

Extract features with VGG16 and Train a classifier with 100 classes

```
>>> # get VGG without the last layer
>>> cnn = tl.models.vgg16(end_with='fc2_relu', mode='static').as_layer()
>>> # add one more layer and build a new model
>>> ni = Input([None, 224, 224, 3], name="inputs")
>>> nn = cnn(ni)
>>> nn = tl.layers.Dense(n_units=100, name='out')(nn)
>>> model = tl.models.Model(inputs=ni, outputs=nn)
>>> # train your own classifier (only update the last layer)
>>> train_params = model.get_layer('out').trainable_weights
```

Reuse model

```
>>> # in dynamic model, we can directly use the same model
>>> # in static model
>>> vgg_layer = tl.models.vgg16().as_layer()
>>> ni_1 = tl.layers.Input([None, 224, 244, 3])
>>> ni_2 = tl.layers.Input([None, 224, 244, 3])
>>> a_1 = vgg_layer(ni_1)
>>> a_2 = vgg_layer(ni_2)
>>> M = Model(inputs=[ni_1, ni_2], outputs=[a_1, a_2])
```

2.8.3 VGG19

`tensorlayer.models.`**`VGG19`**(*pretrained=False*, *end_with='outputs'*, *mode='dynamic'*, *name=None*)

Pre-trained VGG19 model.

Parameters

- **pretrained** (*boolean*) – Whether to load pretrained weights. Default False.
- **end_with** (*str*) – The end point of the model. Default `fc3_relu` i.e. the whole model.
- **mode** (*str.*) – Model building mode, 'dynamic' or 'static'. Default 'dynamic'.
- **name** (*None or str*) – A unique layer name.

Examples

Classify ImageNet classes with VGG19, see tutorial_models_vgg.py With TensorLayer

```
>>> # get the whole model, without pre-trained VGG parameters
>>> vgg = tl.models.vgg19()
>>> # get the whole model, restore pre-trained VGG parameters
>>> vgg = tl.models.vgg19(pretrained=True)
>>> # use for inferencing
>>> output = vgg(img, is_train=False)
>>> probs = tf.nn.softmax(output)[0].numpy()
```

Extract features with VGG19 and Train a classifier with 100 classes

```
>>> # get VGG without the last layer
>>> cnn = tl.models.vgg19(end_with='fc2_relu', mode='static').as_layer()
>>> # add one more layer and build a new model
>>> ni = Input([None, 224, 224, 3], name="inputs")
>>> nn = cnn(ni)
>>> nn = tl.layers.Dense(n_units=100, name='out')(nn)
>>> model = tl.models.Model(inputs=ni, outputs=nn)
>>> # train your own classifier (only update the last layer)
>>> train_params = model.get_layer('out').trainable_weights
```

Reuse model

```
>>> # in dynamic model, we can directly use the same model
>>> # in static model
>>> vgg_layer = tl.models.vgg19().as_layer()
>>> ni_1 = tl.layers.Input([None, 224, 244, 3])
>>> ni_2 = tl.layers.Input([None, 224, 244, 3])
>>> a_1 = vgg_layer(ni_1)
>>> a_2 = vgg_layer(ni_2)
>>> M = Model(inputs=[ni_1, ni_2], outputs=[a_1, a_2])
```

2.8.4 SqueezeNetV1

tensorlayer.models.**SqueezeNetV1**(*pretrained=False*, *end_with='out'*, *name=None*)

Pre-trained SqueezeNetV1 model (static mode). Input shape [?, 224, 224, 3], value range [0, 1].

Parameters

- **pretrained** (*boolean*) – Whether to load pretrained weights. Default False.
- **end_with** (*str*) – The end point of the model [conv1, maxpool1, fire2, fire3, fire4, …, out]. Default out i.e. the whole model.
- **name** (*None or str*) – Name for this model.

Examples

Classify ImageNet classes, see tutorial_models_squeezenetv1.py

```
>>> # get the whole model
>>> squeezenet = tl.models.SqueezeNetV1(pretrained=True)
>>> # use for inferencing
```

(continues on next page)

(continued from previous page)

```
>>> output = squeezenet(img1, is_train=False)
>>> prob = tf.nn.softmax(output)[0].numpy()
```

Extract features and Train a classifier with 100 classes

```
>>> # get model without the last layer
>>> cnn = tl.models.SqueezeNetV1(pretrained=True, end_with='drop1').as_layer()
>>> # add one more layer and build new model
>>> ni = Input([None, 224, 224, 3], name="inputs")
>>> nn = cnn(ni)
>>> nn = Conv2d(100, (1, 1), (1, 1), padding='VALID', name='conv10')(nn)
>>> nn = GlobalMeanPool2d(name='globalmeanpool')(nn)
>>> model = tl.models.Model(inputs=ni, outputs=nn)
>>> # train your own classifier (only update the last layer)
>>> train_params = model.get_layer('conv10').trainable_weights
```

Returns

Return type static SqueezeNetV1.

2.8.5 MobileNetV1

`tensorlayer.models.`**`MobileNetV1`**(*pretrained=False*, *end_with='out'*, *name=None*)

Pre-trained MobileNetV1 model (static mode). Input shape [?, 224, 224, 3], value range [0, 1].

Parameters

- **`pretrained`** (*`boolean`*) – Whether to load pretrained weights. Default False.
- **`end_with`** (*`str`*) – The end point of the model [conv, depth1, depth2 … depth13, globalmeanpool, out]. Default `out` i.e. the whole model.
- **`name`** (*`None or str`*) – Name for this model.

Examples

Classify ImageNet classes, see tutorial_models_mobilenetv1.py

```
>>> # get the whole model with pretrained weights
>>> mobilenetv1 = tl.models.MobileNetV1(pretrained=True)
>>> # use for inferencing
>>> output = mobilenetv1(img1, is_train=False)
>>> prob = tf.nn.softmax(output)[0].numpy()
```

Extract features and Train a classifier with 100 classes

```
>>> # get model without the last layer
>>> cnn = tl.models.MobileNetV1(pretrained=True, end_with='reshape').as_layer()
>>> # add one more layer and build new model
>>> ni = Input([None, 224, 224, 3], name="inputs")
>>> nn = cnn(ni)
>>> nn = Conv2d(100, (1, 1), (1, 1), name='out')(nn)
>>> nn = Flatten(name='flatten')(nn)
>>> model = tl.models.Model(inputs=ni, outputs=nn)
>>> # train your own classifier (only update the last layer)
>>> train_params = model.get_layer('out').trainable_weights
```

Returns

Return type static MobileNetV1.

2.9 API - Natural Language Processing

Natural Language Processing and Word Representation.

`generate_skip_gram_batch`(data, batch_size, ...)	Generate a training batch for the Skip-Gram model.
`sample`([a, temperature])	Sample an index from a probability array.
`sample_top`([a, top_k])	Sample from `top_k` probabilities.
`SimpleVocabulary`(vocab, unk_id)	Simple vocabulary wrapper, see create_vocab().
`Vocabulary`(vocab_file[, start_word, ...])	Create Vocabulary class from a given vocabulary and its id-word, word-id convert.
`process_sentence`(sentence[, start_word, ...])	Seperate a sentence string into a list of string words, add start_word and end_word, see `create_vocab()` and `tutorial_tfrecord3.py`.
`create_vocab`(sentences, word_counts_output_file)	Creates the vocabulary of word to word_id.
`simple_read_words`([filename])	Read context from file without any preprocessing.
`read_words`([filename, replace])	Read list format context from a file.
`read_analogies_file`([eval_file, word2id])	Reads through an analogy question file, return its id format.
`build_vocab`(data)	Build vocabulary.
`build_reverse_dictionary`(word_to_id)	Given a dictionary that maps word to integer id.
`build_words_dataset`([words, ...])	Build the words dictionary and replace rare words with 'UNK' token.
`save_vocab`([count, name])	Save the vocabulary to a file so the model can be reloaded.
`words_to_word_ids`([data, word_to_id, unk_key])	Convert a list of string (words) to IDs.
`word_ids_to_words`(data, id_to_word)	Convert a list of integer to strings (words).
`basic_tokenizer`(sentence[, _WORD_SPLIT])	Very basic tokenizer: split the sentence into a list of tokens.
`create_vocabulary`(vocabulary_path, ...[, ...])	Create vocabulary file (if it does not exist yet) from data file.
`initialize_vocabulary`(vocabulary_path)	Initialize vocabulary from file, return the *word_to_id* (dictionary) and *id_to_word* (list).
`sentence_to_token_ids`(sentence, vocabulary)	Convert a string to list of integers representing token-ids.
`data_to_token_ids`(data_path, target_path, ...)	Tokenize data file and turn into token-ids using given vocabulary file.
`moses_multi_bleu`(hypotheses, references[, ...])	Calculate the bleu score for hypotheses and references using the MOSES ulti-bleu.perl script.

2.9.1 Iteration function for training embedding matrix

`tensorlayer.nlp.`**`generate_skip_gram_batch`**(*data*, *batch_size*, *num_skips*, *skip_window*, *data_index=0*)

Generate a training batch for the Skip-Gram model.

See Word2Vec example.

Parameters

- **data** (*list of data*) – To present context, usually a list of integers.
- **batch_size** (*int*) – Batch size to return.
- **num_skips** (*int*) – How many times to reuse an input to generate a label.
- **skip_window** (*int*) – How many words to consider left and right.
- **data_index** (*int*) – Index of the context location. This code use *data_index* to instead of yield like `tl.iterate`.

Returns

- **batch** (*list of data*) – Inputs.
- **labels** (*list of data*) – Labels
- **data_index** (*int*) – Index of the context location.

Examples

Setting num_skips=2, skip_window=1, use the right and left words. In the same way, num_skips=4, skip_window=2 means use the nearby 4 words.

```
>>> data = [1,2,3,4,5,6,7,8,9,10,11]
>>> batch, labels, data_index = tl.nlp.generate_skip_gram_batch(data=data, batch_
↪size=8, num_skips=2, skip_window=1, data_index=0)
>>> print(batch)
[2 2 3 3 4 4 5 5]
>>> print(labels)
[[3]
[1]
[4]
[2]
[5]
[3]
[4]
[6]]
```

2.9.2 Sampling functions

Simple sampling

`tensorlayer.nlp.`**`sample`**(*a=None*, *temperature=1.0*)

Sample an index from a probability array.

Parameters

- **a** (*list of float*) – List of probabilities.
- **temperature** (*float or None*) –

  The higher the more uniform. When a = [0.1, 0.2, 0.7],

  - temperature = 0.7, the distribution will be sharpen [0.05048273, 0.13588945, 0.81362782]
  - temperature = 1.0, the distribution will be the same [0.1, 0.2, 0.7]

- temperature = 1.5, the distribution will be filtered [0.16008435, 0.25411807, 0.58579758]
- If None, it will be `np.argmax(a)`

Notes

- No matter what is the temperature and input list, the sum of all probabilities will be one. Even if input list = [1, 100, 200], the sum of all probabilities will still be one.
- For large vocabulary size, choice a higher temperature or `tl.nlp.sample_top` to avoid error.

Sampling from top k

`tensorlayer.nlp.`**`sample_top`**(*a=None*, *top_k=10*)

Sample from `top_k` probabilities.

Parameters

- **a** (*list of float*) – List of probabilities.
- **top_k** (*int*) – Number of candidates to be considered.

2.9.3 Vector representations of words

Simple vocabulary class

class `tensorlayer.nlp.`**`SimpleVocabulary`**(*vocab*, *unk_id*)

Simple vocabulary wrapper, see create_vocab().

Parameters

- **vocab** (*dictionary*) – A dictionary that maps word to ID.
- **unk_id** (*int*) – The ID for 'unknown' word.

Vocabulary class

class `tensorlayer.nlp.`**`Vocabulary`**(*vocab_file*, *start_word='<S>'*, *end_word='</S>'*, *unk_word='<UNK>'*, *pad_word='<PAD>'*)

Create Vocabulary class from a given vocabulary and its id-word, word-id convert. See create_vocab() and `tutorial_tfrecord3.py`.

Parameters

- **vocab_file** (*str*) – The file contains the vocabulary (can be created via `tl.nlp.create_vocab`), where the words are the first whitespace-separated token on each line (other tokens are ignored) and the word ids are the corresponding line numbers.
- **start_word** (*str*) – Special word denoting sentence start.
- **end_word** (*str*) – Special word denoting sentence end.
- **unk_word** (*str*) – Special word denoting unknown words.

vocab

A dictionary that maps word to ID.

Type dictionary

reverse_vocab
A list that maps ID to word.

Type list of int

start_id
For start ID.

Type int

end_id
For end ID.

Type int

unk_id
For unknown ID.

Type int

pad_id
For Padding ID.

Type int

Examples

The vocab file looks like follow, includes *start_word* , *end_word* …

```
>>> a 969108
>>> <S> 586368
>>> </S> 586368
>>> . 440479
>>> on 213612
>>> of 202290
>>> the 196219
>>> in 182598
>>> with 152984
>>> and 139109
>>> is 97322
```

Process sentence

`tensorlayer.nlp.`**`process_sentence`**(*sentence*, *start_word='<S>'*, *end_word='</S>'*)
Seperate a sentence string into a list of string words, add start_word and end_word, see `create_vocab()` and `tutorial_tfrecord3.py`.

Parameters

- **sentence** (*str*) – A sentence.
- **start_word** (*str or None*) – The start word. If None, no start word will be appended.
- **end_word** (*str or None*) – The end word. If None, no end word will be appended.

Returns A list of strings that separated into words.

Return type list of str

Examples

```
>>> c = "how are you?"
>>> c = tl.nlp.process_sentence(c)
>>> print(c)
['<S>', 'how', 'are', 'you', '?', '</S>']
```

Notes

- You have to install the following package.
- Installing NLTK
- Installing NLTK data

Create vocabulary

tensorlayer.nlp.**create_vocab**(*sentences*, *word_counts_output_file*, *min_word_count=1*)

Creates the vocabulary of word to word_id.

See `tutorial_tfrecord3.py`.

The vocabulary is saved to disk in a text file of word counts. The id of each word in the file is its corresponding 0-based line number.

Parameters

- **sentences** (*list of list of str*) – All sentences for creating the vocabulary.
- **word_counts_output_file** (*str*) – The file name.
- **min_word_count** (*int*) – Minimum number of occurrences for a word.

Returns The simple vocabulary object, see *Vocabulary* for more.

Return type *SimpleVocabulary*

Examples

Pre-process sentences

```
>>> captions = ["one two , three", "four five five"]
>>> processed_capts = []
>>> for c in captions:
>>>     c = tl.nlp.process_sentence(c, start_word="<S>", end_word="</S>")
>>>     processed_capts.append(c)
>>> print(processed_capts)
...[['<S>', 'one', 'two', ',', 'three', '</S>'], ['<S>', 'four', 'five', 'five', '
→</S>']]
```

Create vocabulary

```
>>> tl.nlp.create_vocab(processed_capts, word_counts_output_file='vocab.txt', min_
→word_count=1)
Creating vocabulary.
  Total words: 8
  Words in vocabulary: 8
  Wrote vocabulary file: vocab.txt
```

Get vocabulary object

```
>>> vocab = tl.nlp.Vocabulary('vocab.txt', start_word="<S>", end_word="</S>", unk_
↪word="<UNK>")
INFO:tensorflow:Initializing vocabulary from file: vocab.txt
[TL] Vocabulary from vocab.txt : <S> </S> <UNK>
vocabulary with 10 words (includes start_word, end_word, unk_word)
    start_id: 2
    end_id: 3
    unk_id: 9
    pad_id: 0
```

2.9.4 Read words from file

Simple read file

`tensorlayer.nlp.`**`simple_read_words`**(*filename='nietzsche.txt'*)

Read context from file without any preprocessing.

Parameters **filename** (*str*) – A file path (like .txt file)

Returns The context in a string.

Return type str

Read file

`tensorlayer.nlp.`**`read_words`**(*filename='nietzsche.txt'*, *replace=None*)

Read list format context from a file.

For customized read_words method, see `tutorial_generate_text.py`.

Parameters

- **filename** (*str*) – a file path.
- **replace** (*list of str*) – replace original string by target string.

Returns The context in a list (split using space).

Return type list of str

2.9.5 Read analogy question file

`tensorlayer.nlp.`**`read_analogies_file`**(*eval_file='questions-words.txt'*, *word2id=None*)

Reads through an analogy question file, return its id format.

Parameters

- **eval_file** (*str*) – The file name.
- **word2id** (*dictionary*) – a dictionary that maps word to ID.

Returns A `[n_examples, 4]` numpy array containing the analogy question's word IDs.

Return type numpy.array

Examples

The file should be in this format

```
>>> : capital-common-countries
>>> Athens Greece Baghdad Iraq
>>> Athens Greece Bangkok Thailand
>>> Athens Greece Beijing China
>>> Athens Greece Berlin Germany
>>> Athens Greece Bern Switzerland
>>> Athens Greece Cairo Egypt
>>> Athens Greece Canberra Australia
>>> Athens Greece Hanoi Vietnam
>>> Athens Greece Havana Cuba
```

Get the tokenized analogy question data

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> data, count, dictionary, reverse_dictionary = tl.nlp.build_words_
↪dataset(words, vocabulary_size, True)
>>> analogy_questions = tl.nlp.read_analogies_file(eval_file='questions-words.txt
↪', word2id=dictionary)
>>> print(analogy_questions)
[[ 3068  1248  7161  1581]
[ 3068  1248 28683  5642]
[ 3068  1248  3878   486]
...,
[ 1216  4309 19982 25506]
[ 1216  4309  3194  8650]
[ 1216  4309   140   312]]
```

2.9.6 Build vocabulary, word dictionary and word tokenization

Build dictionary from word to id

`tensorlayer.nlp.`**`build_vocab`**`(`*data*`)`

Build vocabulary.

Given the context in list format. Return the vocabulary, which is a dictionary for word to id. e.g. {'campbell': 2587, 'atlantic': 2247, 'aoun': 6746 .... }

Parameters **data** (*list of str*) – The context in list format

Returns that maps word to unique ID. e.g. {'campbell': 2587, 'atlantic': 2247, 'aoun': 6746 .... }

Return type dictionary

References

- tensorflow.models.rnn.ptb.reader

Examples

```
>>> data_path = os.getcwd() + '/simple-examples/data'
>>> train_path = os.path.join(data_path, "ptb.train.txt")
>>> word_to_id = build_vocab(read_txt_words(train_path))
```

Build dictionary from id to word

`tensorlayer.nlp.`**`build_reverse_dictionary`**(*word_to_id*)

Given a dictionary that maps word to integer id. Returns a reverse dictionary that maps a id to word.

Parameters **`word_to_id`** (*`dictionary`*) – that maps word to ID.

Returns A dictionary that maps IDs to words.

Return type dictionary

Build dictionaries for id to word etc

`tensorlayer.nlp.`**`build_words_dataset`**(*words=None*, *vocabulary_size=50000*, *printable=True*, *unk_key='UNK'*)

Build the words dictionary and replace rare words with 'UNK' token. The most common word has the smallest integer id.

Parameters

- **`words`** (*`list of str or byte`*) – The context in list format. You may need to do preprocessing on the words, such as lower case, remove marks etc.
- **`vocabulary_size`** (*`int`*) – The maximum vocabulary size, limiting the vocabulary size. Then the script replaces rare words with 'UNK' token.
- **`printable`** (*`boolean`*) – Whether to print the read vocabulary size of the given words.
- **`unk_key`** (*`str`*) – Represent the unknown words.

Returns

- **data** (*list of int*) – The context in a list of ID.
- **count** (*list of tuple and list*) –

  Pair words and IDs.

  – count[0] is a list : the number of rare words

  – count[1:] are tuples : the number of occurrence of each word

  – e.g. [['UNK', 418391], (b'the', 1061396), (b'of', 593677), (b'and', 416629), (b'one', 411764)]
- **dictionary** (*dictionary*) – It is *word_to_id* that maps word to ID.
- **reverse_dictionary** (*a dictionary*) – It is *id_to_word* that maps ID to word.

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> vocabulary_size = 50000
>>> data, count, dictionary, reverse_dictionary = tl.nlp.build_words_
↪dataset(words, vocabulary_size)
```

References

- tensorflow/examples/tutorials/word2vec/word2vec_basic.py

Save vocabulary

tensorlayer.nlp.**save_vocab**(*count=None*, *name='vocab.txt'*)

Save the vocabulary to a file so the model can be reloaded.

Parameters **count** (*a list of tuple and list*) – count[0] is a list : the number of rare words, count[1:] are tuples : the number of occurrence of each word, e.g. [['UNK', 418391], (b'the', 1061396), (b'of', 593677), (b'and', 416629), (b'one', 411764)]

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> vocabulary_size = 50000
>>> data, count, dictionary, reverse_dictionary = tl.nlp.build_words_
↪dataset(words, vocabulary_size, True)
>>> tl.nlp.save_vocab(count, name='vocab_text8.txt')
>>> vocab_text8.txt
UNK 418391
the 1061396
of 593677
and 416629
one 411764
in 372201
a 325873
to 316376
```

2.9.7 Convert words to IDs and IDs to words

These functions can be done by `Vocabulary` class.

List of Words to IDs

tensorlayer.nlp.**words_to_word_ids**(*data=None*, *word_to_id=None*, *unk_key='UNK'*)

Convert a list of string (words) to IDs.

Parameters

- **data** (*list of string or byte*) – The context in list format
- **word_to_id** (*a dictionary*) – that maps word to ID.
- **unk_key** (*str*) – Represent the unknown words.

Returns A list of IDs to represent the context.

Return type list of int

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> vocabulary_size = 50000
>>> data, count, dictionary, reverse_dictionary = tl.nlp.build_words_
↪dataset(words, vocabulary_size, True)
>>> context = [b'hello', b'how', b'are', b'you']
>>> ids = tl.nlp.words_to_word_ids(words, dictionary)
>>> context = tl.nlp.word_ids_to_words(ids, reverse_dictionary)
>>> print(ids)
[6434, 311, 26, 207]
>>> print(context)
[b'hello', b'how', b'are', b'you']
```

References

- tensorflow.models.rnn.ptb.reader

List of IDs to Words

`tensorlayer.nlp.`**`word_ids_to_words`**(*data*, *id_to_word*)

Convert a list of integer to strings (words).

Parameters

- **data** (*list of int*) – The context in list format.
- **id_to_word** (*dictionary*) – a dictionary that maps ID to word.

Returns A list of string or byte to represent the context.

Return type list of str

Examples

see `tl.nlp.words_to_word_ids`

2.9.8 Functions for translation

Word Tokenization

`tensorlayer.nlp.`**`basic_tokenizer`**(*sentence*, *_WORD_SPLIT=re.compile(b'([., !?"\':;)(])')*)

Very basic tokenizer: split the sentence into a list of tokens.

Parameters

- **sentence** (*tensorflow.python.platform.gfile.GFile Object*) –
- **_WORD_SPLIT** (*regular expression for word spliting.*) –

Examples

```
>>> see create_vocabulary
>>> from tensorflow.python.platform import gfile
>>> train_path = "wmt/giga-fren.release2"
>>> with gfile.GFile(train_path + ".en", mode="rb") as f:
>>>    for line in f:
>>>       tokens = tl.nlp.basic_tokenizer(line)
>>>       tl.logging.info(tokens)
>>>       exit()
[b'Changing', b'Lives', b'|', b'Changing', b'Society', b'|', b'How',
  b'It', b'Works', b'|', b'Technology', b'Drives', b'Change', b'Home',
  b'|', b'Concepts', b'|', b'Teachers', b'|', b'Search', b'|', b'Overview',
  b'|', b'Credits', b'|', b'HHCC', b'Web', b'|', b'Reference', b'|',
  b'Feedback', b'Virtual', b'Museum', b'of', b'Canada', b'Home', b'Page']
```

References

- Code from /tensorflow/models/rnn/translation/data_utils.py

Create or read vocabulary

tensorlayer.nlp.**create_vocabulary**(*vocabulary_path*, *data_path*, *max_vocabulary_size*, *tokenizer=None*, *normalize_digits=True*, *_DIGIT_RE=re.compile(b'\\d')*, *_START_VOCAB=None*)

Create vocabulary file (if it does not exist yet) from data file.

Data file is assumed to contain one sentence per line. Each sentence is tokenized and digits are normalized (if normalize_digits is set). Vocabulary contains the most-frequent tokens up to max_vocabulary_size. We write it to vocabulary_path in a one-token-per-line format, so that later token in the first line gets id=0, second line gets id=1, and so on.

Parameters

- **vocabulary_path** (*str*) – Path where the vocabulary will be created.
- **data_path** (*str*) – Data file that will be used to create vocabulary.
- **max_vocabulary_size** (*int*) – Limit on the size of the created vocabulary.
- **tokenizer** (*function*) – A function to use to tokenize each data sentence. If None, basic_tokenizer will be used.
- **normalize_digits** (*boolean*) – If true, all digits are replaced by *0*.
- **_DIGIT_RE** (*regular expression function*) – Default is re.compile(br"\d").
- **_START_VOCAB** (*list of str*) – The pad, go, eos and unk token, default is [b"_PAD", b"_GO", b"_EOS", b"_UNK"].

References

- Code from /tensorflow/models/rnn/translation/data_utils.py

tensorlayer.nlp.**initialize_vocabulary**(*vocabulary_path*)

Initialize vocabulary from file, return the *word_to_id* (dictionary) and *id_to_word* (list).

We assume the vocabulary is stored one-item-per-line, so a file will result in a vocabulary {"dog": 0, "cat": 1}, and this function will also return the reversed-vocabulary ["dog", "cat"].

Parameters **vocabulary_path** (*str*) – Path to the file containing the vocabulary.

Returns

- **vocab** (*dictionary*) – a dictionary that maps word to ID.
- **rev_vocab** (*list of int*) – a list that maps ID to word.

Examples

```
>>> Assume 'test' contains
dog
cat
bird
>>> vocab, rev_vocab = tl.nlp.initialize_vocabulary("test")
>>> print(vocab)
>>> {b'cat': 1, b'dog': 0, b'bird': 2}
>>> print(rev_vocab)
>>> [b'dog', b'cat', b'bird']
```

:raises ValueError : if the provided vocabulary_path does not exist.:

Convert words to IDs and IDs to words

tensorlayer.nlp.**sentence_to_token_ids**(*sentence*, *vocabulary*, *tokenizer=None*, *normalize_digits=True*, *UNK_ID=3*, *_DIGIT_RE=re.compile(b'\\d')*)

Convert a string to list of integers representing token-ids.

For example, a sentence "I have a dog" may become tokenized into ["I", "have", "a", "dog"] and with vocabulary {"I": 1, "have": 2, "a": 4, "dog": 7"} this function will return [1, 2, 4, 7].

Parameters

- **sentence** (*tensorflow.python.platform.gfile.GFile Object*) – The sentence in bytes format to convert to token-ids, see `basic_tokenizer()` and `data_to_token_ids()`.
- **vocabulary** (*dictionary*) – Mmapping tokens to integers.
- **tokenizer** (*function*) – A function to use to tokenize each sentence. If None, `basic_tokenizer` will be used.
- **normalize_digits** (*boolean*) – If true, all digits are replaced by 0.

Returns The token-ids for the sentence.

Return type list of int

tensorlayer.nlp.**data_to_token_ids**(*data_path*, *target_path*, *vocabulary_path*, *tokenizer=None*, *normalize_digits=True*, *UNK_ID=3*, *_DIGIT_RE=re.compile(b'\\d')*)

Tokenize data file and turn into token-ids using given vocabulary file.

This function loads data line-by-line from data_path, calls the above sentence_to_token_ids, and saves the result to target_path. See comment for sentence_to_token_ids on the details of token-ids format.

Parameters

- **data_path** (*str*) – Path to the data file in one-sentence-per-line format.
- **target_path** (*str*) – Path where the file with token-ids will be created.
- **vocabulary_path** (*str*) – Path to the vocabulary file.
- **tokenizer** (*function*) – A function to use to tokenize each sentence. If None, `basic_tokenizer` will be used.
- **normalize_digits** (*boolean*) – If true, all digits are replaced by 0.

References

- Code from `/tensorflow/models/rnn/translation/data_utils.py`

2.9.9 Metrics

BLEU

`tensorlayer.nlp.`**`moses_multi_bleu`**(*hypotheses*, *references*, *lowercase=False*)

Calculate the bleu score for hypotheses and references using the MOSES ulti-bleu.perl script.

Parameters

- **hypotheses** (*numpy.array.string*) – A numpy array of strings where each string is a single example.
- **references** (*numpy.array.string*) – A numpy array of strings where each string is a single example.
- **lowercase** (*boolean*) – If True, pass the "-lc" flag to the multi-bleu script

Examples

```
>>> hypotheses = ["a bird is flying on the sky"]
>>> references = ["two birds are flying on the sky", "a bird is on the top of the␣
↪tree", "an airplane is on the sky",]
>>> score = tl.nlp.moses_multi_bleu(hypotheses, references)
```

Returns The BLEU score

Return type float

References

- Google/seq2seq/metric/bleu

2.10 API - Initializers

To make TensorLayer simple, TensorLayer only warps some basic initializers. For more advanced initializer, e.g. `tf.initializers.he_normal`, please refer to TensorFlow provided initializers here.

Initializer	Initializer base class: all initializers inherit from this class.
Zeros	Initializer that generates tensors initialized to 0.
Ones	Initializer that generates tensors initialized to 1.
Constant([value])	Initializer that generates tensors initialized to a constant value.
RandomUniform([minval, maxval, seed])	Initializer that generates tensors with a uniform distribution.
RandomNormal([mean, stddev, seed])	Initializer that generates tensors with a normal distribution.
TruncatedNormal([mean, stddev, seed])	Initializer that generates a truncated normal distribution.
deconv2d_bilinear_upsampling_initializer(shape)	Returns the initializer that can be passed to DeConv2dLayer for initializing the weights in correspondence to channel-wise bilinear up-sampling.

2.10.1 Initializer

class tensorlayer.initializers.**Initializer**

Initializer base class: all initializers inherit from this class.

2.10.2 Zeros

class tensorlayer.initializers.**Zeros**

Initializer that generates tensors initialized to 0.

2.10.3 Ones

class tensorlayer.initializers.**Ones**

Initializer that generates tensors initialized to 1.

2.10.4 Constant

class tensorlayer.initializers.**Constant**(*value=0*)

Initializer that generates tensors initialized to a constant value.

Parameters **value** (*A python scalar or a numpy array.*) – The assigned value.

2.10.5 RandomUniform

class tensorlayer.initializers.**RandomUniform**(*minval=-0.05*, *maxval=0.05*, *seed=None*)

Initializer that generates tensors with a uniform distribution.

Parameters

- **minval** (*A python scalar or a scalar tensor.*) – Lower bound of the range of random values to generate.
- **maxval** (*A python scalar or a scalar tensor.*) – Upper bound of the range of random values to generate.
- **seed** (*A Python integer.*) – Used to seed the random generator.

2.10.6 RandomNormal

class tensorlayer.initializers.**RandomNormal**(*mean=0.0*, *stddev=0.05*, *seed=None*)

Initializer that generates tensors with a normal distribution.

Parameters

- **mean** (*A python scalar or a scalar tensor.*) – Mean of the random values to generate.
- **stddev** (*A python scalar or a scalar tensor.*) – Standard deviation of the random values to generate.
- **seed** (*A Python integer.*) – Used to seed the random generator.

2.10.7 TruncatedNormal

class tensorlayer.initializers.**TruncatedNormal**(*mean=0.0*, *stddev=0.05*, *seed=None*)

Initializer that generates a truncated normal distribution.

These values are similar to values from a *RandomNormal* except that values more than two standard deviations from the mean are discarded and re-drawn. This is the recommended initializer for neural network weights and filters.

Parameters

- **mean** (*A python scalar or a scalar tensor.*) – Mean of the random values to generate.
- **stddev** (*A python scalar or a scalar tensor.*) – Standard deviation of the andom values to generate.
- **seed** (*A Python integer.*) – Used to seed the random generator.

2.10.8 deconv2d_bilinear_upsampling_initializer

tensorlayer.initializers.**deconv2d_bilinear_upsampling_initializer**(*shape*)

Returns the initializer that can be passed to DeConv2dLayer for initializing the weights in correspondence to channel-wise bilinear up-sampling. Used in segmentation approaches such as [FCN](https://arxiv.org/abs/1605.06211)

Parameters **shape** (*tuple of int*) – The shape of the filters, [height, width, output_channels, in_channels]. It must match the shape passed to DeConv2dLayer.

Returns A constant initializer with weights set to correspond to per channel bilinear upsampling when passed as W_int in DeConv2dLayer

Return type tf.constant_initializer

2.11 API - Reinforcement Learning

Reinforcement Learning.

discount_episode_rewards([rewards, gamma, mode])	Take 1D float array of rewards and compute discounted rewards for an episode.

Continued on next page

Table 11 – continued from previous page

cross_entropy_reward_loss(logits, actions, ...)	Calculate the loss for Policy Gradient Network.
log_weight(probs, weights[, name])	Log weight.
choice_action_by_probs([probs, action_list])	Choice and return an an action by given the action probability distribution.

2.11.1 Reward functions

tensorlayer.rein.**discount_episode_rewards**(*rewards=None*, *gamma=0.99*, *mode=0*)

Take 1D float array of rewards and compute discounted rewards for an episode. When encount a non-zero value, consider as the end a of an episode.

Parameters

- **rewards** (*list*) – List of rewards
- **gamma** (*float*) – Discounted factor
- **mode** (*int*) –

  Mode for computing the discount rewards.

  – If mode == 0, reset the discount process when encount a non-zero reward (Ping-pong game).

  – If mode == 1, would not reset the discount process.

Returns The discounted rewards.

Return type list of float

Examples

```
>>> rewards = np.asarray([0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1])
>>> gamma = 0.9
>>> discount_rewards = tl.rein.discount_episode_rewards(rewards, gamma)
>>> print(discount_rewards)
[ 0.72899997  0.81        0.89999998  1.          0.72899997  0.81
0.89999998  1.          0.72899997  0.81        0.89999998  1.        ]
>>> discount_rewards = tl.rein.discount_episode_rewards(rewards, gamma, mode=1)
>>> print(discount_rewards)
[ 1.52110755  1.69011939  1.87791049  2.08656716  1.20729685  1.34144104
1.49048996  1.65610003  0.72899997  0.81        0.89999998  1.        ]
```

2.11.2 Cost functions

Weighted Cross Entropy

tensorlayer.rein.**cross_entropy_reward_loss**(*logits*, *actions*, *rewards*, *name=None*)

Calculate the loss for Policy Gradient Network.

Parameters

- **logits** (*tensor*) – The network outputs without softmax. This function implements softmax inside.

- **actions** (*tensor or placeholder*) – The agent actions.
- **rewards** (*tensor or placeholder*) – The rewards.

Returns The TensorFlow loss function.

Return type Tensor

Examples

```
>>> states_batch_pl = tf.placeholder(tf.float32, shape=[None, D])
>>> network = InputLayer(states_batch_pl, name='input')
>>> network = DenseLayer(network, n_units=H, act=tf.nn.relu, name='relu1')
>>> network = DenseLayer(network, n_units=3, name='out')
>>> probs = network.outputs
>>> sampling_prob = tf.nn.softmax(probs)
>>> actions_batch_pl = tf.placeholder(tf.int32, shape=[None])
>>> discount_rewards_batch_pl = tf.placeholder(tf.float32, shape=[None])
>>> loss = tl.rein.cross_entropy_reward_loss(probs, actions_batch_pl, discount_
↪rewards_batch_pl)
>>> train_op = tf.train.RMSPropOptimizer(learning_rate, decay_rate).minimize(loss)
```

Log weight

`tensorlayer.rein.`**`log_weight`**(*probs*, *weights*, *name='log_weight'*)

Log weight.

Parameters

- **probs** (*tensor*) – If it is a network output, usually we should scale it to [0, 1] via softmax.
- **weights** (*tensor*) – The weights.

Returns The Tensor after appling the log weighted expression.

Return type Tensor

2.11.3 Sampling functions

`tensorlayer.rein.`**`choice_action_by_probs`**(*probs=(0.5, 0.5)*, *action_list=None*)

Choice and return an an action by given the action probability distribution.

Parameters

- **probs** (*list of float.*) – The probability distribution of all actions.
- **action_list** (*None or a list of int or others*) – A list of action in integer, string or others. If None, returns an integer range between 0 and len(probs)-1.

Returns The chosen action.

Return type float int or str

Examples

```
>>> for _ in range(5):
>>>     a = choice_action_by_probs([0.2, 0.4, 0.4])
>>>     print(a)
0
1
1
2
1
>>> for _ in range(3):
>>>     a = choice_action_by_probs([0.5, 0.5], ['a', 'b'])
>>>     print(a)
a
b
b
```

2.12 API - Utility

fit(network, train_op, cost, X_train, y_train)	Training a given non time-series network by the given cost function, training data, batch_size, n_epoch etc.
test(network, acc, X_test, y_test, batch_size)	Test a given non time-series network by the given test data and metric.
predict(network, X[, batch_size])	Return the predict results of given non time-series network.
evaluation([y_test, y_predict, n_classes])	Input the predicted results, targets results and the number of class, return the confusion matrix, F1-score of each class, accuracy and macro F1-score.
class_balancing_oversample([X_train, . . .])	Input the features and labels, return the features and labels after oversampling.
get_random_int([min_v, max_v, number, seed])	Return a list of random integer by the given range and quantity.
dict_to_one(dp_dict)	Input a dictionary, return a dictionary that all items are set to one.
list_string_to_dict(string)	Inputs `['a', 'b', 'c']`, returns `{'a': 0, 'b': 1, 'c': 2}`.
flatten_list(list_of_list)	Input a list of list, return a list that all items are in a list.
exit_tensorflow([port])	Close TensorBoard and Nvidia-process if available.
open_tensorboard([log_dir, port])	Open Tensorboard.
clear_all_placeholder_variables([printable])	Clears all the placeholder variables of keep prob, including keeping probabilities of all dropout, denoising, dropconnect etc.
set_gpu_fraction([gpu_fraction])	Set the GPU memory fraction for the application.

2.12.1 Training, testing and predicting

Training

tensorlayer.utils.**fit**(*network, train_op, cost, X_train, y_train, acc=None, batch_size=100, n_epoch=100, print_freq=5, X_val=None, y_val=None, eval_train=True, tensorboard_dir=None, tensorboard_epoch_freq=5, tensorboard_weight_histograms=True, tensorboard_graph_vis=True*)

Training a given non time-series network by the given cost function, training data, batch_size, n_epoch etc.

- MNIST example click here.
- In order to control the training details, the authors HIGHLY recommend `tl.iterate` see two MNIST examples 1, 2.

Parameters

- **network** (*TensorLayer Model*) – the network to be trained.
- **train_op** (*TensorFlow optimizer*) – The optimizer for training e.g. tf.optimizers.Adam().
- **cost** (*TensorLayer or TensorFlow loss function*) – Metric for loss function, e.g tl.cost.cross_entropy.
- **X_train** (*numpy.array*) – The input of training data
- **y_train** (*numpy.array*) – The target of training data
- **acc** (*TensorFlow/numpy expression or None*) – Metric for accuracy or others. If None, would not print the information.
- **batch_size** (*int*) – The batch size for training and evaluating.
- **n_epoch** (*int*) – The number of training epochs.
- **print_freq** (*int*) – Print the training information every `print_freq` epochs.
- **X_val** (*numpy.array or None*) – The input of validation data. If None, would not perform validation.
- **y_val** (*numpy.array or None*) – The target of validation data. If None, would not perform validation.
- **eval_train** (*boolean*) – Whether to evaluate the model during training. If X_val and y_val are not None, it reflects whether to evaluate the model on training data.
- **tensorboard_dir** (*string*) – path to log dir, if set, summary data will be stored to the tensorboard_dir/ directory for visualization with tensorboard. (default None)
- **tensorboard_epoch_freq** (*int*) – How many epochs between storing tensorboard checkpoint for visualization to log/ directory (default 5).
- **tensorboard_weight_histograms** (*boolean*) – If True updates tensorboard data in the logs/ directory for visualization of the weight histograms every tensorboard_epoch_freq epoch (default True).
- **tensorboard_graph_vis** (*boolean*) – If True stores the graph in the tensorboard summaries saved to log/ (default True).

Examples

See tutorial_mnist_simple.py

```
>>> tl.utils.fit(network, train_op=tf.optimizers.Adam(learning_rate=0.0001),
...                  cost=tl.cost.cross_entropy, X_train=X_train, y_train=y_train,␣
↪acc=acc,
...                  batch_size=64, n_epoch=20, _val=X_val, y_val=y_val, eval_
↪train=True)
>>> tl.utils.fit(network, train_op, cost, X_train, y_train,
...             acc=acc, batch_size=500, n_epoch=200, print_freq=5,
...             X_val=X_val, y_val=y_val, eval_train=False, tensorboard=True)
```

Notes

'tensorboard_weight_histograms' and 'tensorboard_weight_histograms' are not supported now.

Evaluation

`tensorlayer.utils.test(network, acc, X_test, y_test, batch_size, cost=None)`

Test a given non time-series network by the given test data and metric.

Parameters

- **network** (*TensorLayer Model*) – The network.
- **acc** (*TensorFlow/numpy expression or None*) –

  Metric for accuracy or others.

  – If None, would not print the information.
- **X_test** (*numpy.array*) – The input of testing data.
- **y_test** (*numpy array*) – The target of testing data
- **batch_size** (*int or None*) – The batch size for testing, when dataset is large, we should use minibatche for testing; if dataset is small, we can set it to None.
- **cost** (*TensorLayer or TensorFlow loss function*) – Metric for loss function, e.g tl.cost.cross_entropy. If None, would not print the information.

Examples

See tutorial_mnist_simple.py

```
>>> def acc(_logits, y_batch):
...     return np.mean(np.equal(np.argmax(_logits, 1), y_batch))
>>> tl.utils.test(network, acc, X_test, y_test, batch_size=None, cost=tl.cost.
↪cross_entropy)
```

Prediction

`tensorlayer.utils.predict(network, X, batch_size=None)`

Return the predict results of given non time-series network.

Parameters

- **network** (*TensorLayer Model*) – The network.
- **X** (*numpy.array*) – The inputs.
- **batch_size** (*int or None*) – The batch size for prediction, when dataset is large, we should use minibatche for prediction; if dataset is small, we can set it to None.

Examples

See tutorial_mnist_simple.py

```
>>> _logits = tl.utils.predict(network, X_test)
>>> y_pred = np.argmax(_logits, 1)
```

2.12.2 Evaluation functions

`tensorlayer.utils.evaluation`(*y_test=None*, *y_predict=None*, *n_classes=None*)

Input the predicted results, targets results and the number of class, return the confusion matrix, F1-score of each class, accuracy and macro F1-score.

Parameters

- **y_test** (*list*) – The target results
- **y_predict** (*list*) – The predicted results
- **n_classes** (*int*) – The number of classes

Examples

```
>>> c_mat, f1, acc, f1_macro = tl.utils.evaluation(y_test, y_predict, n_classes)
```

2.12.3 Class balancing functions

`tensorlayer.utils.class_balancing_oversample`(*X_train=None*, *y_train=None*, *printable=True*)

Input the features and labels, return the features and labels after oversampling.

Parameters

- **X_train** (*numpy.array*) – The inputs.
- **y_train** (*numpy.array*) – The targets.

Examples

One X

```
>>> X_train, y_train = class_balancing_oversample(X_train, y_train,␣
↪printable=True)
```

Two X

```
>>> X, y = tl.utils.class_balancing_oversample(X_train=np.hstack((X1, X2)), y_
↪train=y, printable=False)
>>> X1 = X[:, 0:5]
>>> X2 = X[:, 5:]
```

2.12.4 Random functions

`tensorlayer.utils.get_random_int`(*min_v=0*, *max_v=10*, *number=5*, *seed=None*)

Return a list of random integer by the given range and quantity.

Parameters

- **min_v** (*number*) – The minimum value.
- **max_v** (*number*) – The maximum value.
- **number** (*int*) – Number of value.

- **seed** (*int or None*) – The seed for random.

Examples

```
>>> r = get_random_int(min_v=0, max_v=10, number=5)
[10, 2, 3, 3, 7]
```

2.12.5 Dictionary and list

Set all items in dictionary to one

tensorlayer.utils.**dict_to_one**(*dp_dict*)
Input a dictionary, return a dictionary that all items are set to one.

Used for disable dropout, dropconnect layer and so on.

Parameters dp_dict (*dictionary*) – The dictionary contains key and number, e.g. keeping probabilities.

Convert list of string to dictionary

tensorlayer.utils.**list_string_to_dict**(*string*)
Inputs `['a', 'b', 'c']`, returns `{'a': 0, 'b': 1, 'c': 2}`.

Flatten a list

tensorlayer.utils.**flatten_list**(*list_of_list*)
Input a list of list, return a list that all items are in a list.

Parameters list_of_list (*a list of list*) –

Examples

```
>>> tl.utils.flatten_list([[1, 2, 3],[4, 5],[6]])
[1, 2, 3, 4, 5, 6]
```

2.12.6 Close TF session and associated processes

tensorlayer.utils.**exit_tensorflow**(*port=6006*)
Close TensorBoard and Nvidia-process if available.

Parameters port (*int*) – TensorBoard port you want to close, *6006* as default.

2.12.7 Open TensorBoard

tensorlayer.utils.**open_tensorboard**(*log_dir='/tmp/tensorflow'*, *port=6006*)
Open Tensorboard.

Parameters

- **log_dir** (*str*) – Directory where your tensorboard logs are saved
- **port** (*int*) – TensorBoard port you want to open, 6006 is tensorboard default

2.12.8 Clear TensorFlow placeholder

tensorlayer.utils.**clear_all_placeholder_variables**(*printable=True*)
Clears all the placeholder variables of keep prob, including keeping probabilities of all dropout, denoising, dropconnect etc.

Parameters **printable** (*boolean*) – If True, print all deleted variables.

2.12.9 Set GPU functions

tensorlayer.utils.**set_gpu_fraction**(*gpu_fraction=0.3*)
Set the GPU memory fraction for the application.

Parameters **gpu_fraction** (*None or float*) – Fraction of GPU memory, (0 ~ 1]. If None, allow gpu memory growth.

References

- TensorFlow using GPU

2.13 API - Visualization

TensorFlow provides TensorBoard to visualize the model, activations etc. Here we provide more functions for data visualization.

read_image(image[, path])	Read one image.
read_images(img_list[, path, n_threads, ...])	Returns all images in list by given path and name of each image file.
save_image(image[, image_path])	Save a image.
save_images(images, size[, image_path])	Save multiple images into one single image.
draw_boxes_and_labels_to_image(image, ...[, ...])	Draw bboxes and class labels on image.
draw_mpii_pose_to_image(image, poses[, ...])	Draw people(s) into image using MPII dataset format as input, return or save the result image.
draw_weights([W, second, saveable, shape, ...])	Visualize every columns of the weight matrix to a group of Greyscale img.
CNN2d([CNN, second, saveable, name, fig_idx])	Display a group of RGB or Greyscale CNN masks.
frame([I, second, saveable, name, cmap, fig_idx])	Display a frame.
images2d([images, second, saveable, name, ...])	Display a group of RGB or Greyscale images.
tsne_embedding(embeddings, reverse_dictionary)	Visualize the embeddings by using t-SNE.

2.13.1 Save and read images

Read one image

`tensorlayer.visualize.read_image(image, path='')`

Read one image.

Parameters

- **image** (*str*) – The image file name.
- **path** (*str*) – The image folder path.

Returns The image.

Return type numpy.array

Read multiple images

`tensorlayer.visualize.read_images(img_list, path='', n_threads=10, printable=True)`

Returns all images in list by given path and name of each image file.

Parameters

- **img_list** (*list of str*) – The image file names.
- **path** (*str*) – The image folder path.
- **n_threads** (*int*) – The number of threads to read image.
- **printable** (*boolean*) – Whether to print information when reading images.

Returns The images.

Return type list of numpy.array

Save one image

`tensorlayer.visualize.save_image(image, image_path='_temp.png')`

Save a image.

Parameters

- **image** (*numpy array*) – [w, h, c]
- **image_path** (*str*) – path

Save multiple images

`tensorlayer.visualize.save_images(images, size, image_path='_temp.png')`

Save multiple images into one single image.

Parameters

- **images** (*numpy array*) – (batch, w, h, c)
- **size** (*list of 2 ints*) – row and column number. number of images should be equal or less than size[0] * size[1]
- **image_path** (*str*) – save path

Examples

```
>>> import numpy as np
>>> import tensorlayer as tl
>>> images = np.random.rand(64, 100, 100, 3)
>>> tl.visualize.save_images(images, [8, 8], 'temp.png')
```

Save image for object detection

tensorlayer.visualize.**draw_boxes_and_labels_to_image**(*image*, *classes*, *coords*, *scores*, *classes_list*, *is_center=True*, *is_rescale=True*, *save_name=None*)

Draw bboxes and class labels on image. Return or save the image with bboxes, example in the docs of `tl.prepro`.

Parameters

- **image** (*numpy.array*) – The RGB image [height, width, channel].
- **classes** (*list of int*) – A list of class ID (int).
- **coords** (*list of int*) –

  A list of list for coordinates.

  – Should be [x, y, x2, y2] (up-left and botton-right format)

  – If [x_center, y_center, w, h] (set is_center to True).
- **scores** (*list of float*) – A list of score (float). (Optional)
- **classes_list** (*list of str*) – for converting ID to string on image.
- **is_center** (*boolean*) –

  Whether the coordinates is [x_center, y_center, w, h]

  – If coordinates are [x_center, y_center, w, h], set it to True for converting it to [x, y, x2, y2] (up-left and botton-right) internally.

  – If coordinates are [x1, x2, y1, y2], set it to False.
- **is_rescale** (*boolean*) –

  Whether to rescale the coordinates from pixel-unit format to ratio format.

  – If True, the input coordinates are the portion of width and high, this API will scale the coordinates to pixel unit internally.

  – If False, feed the coordinates with pixel unit format.
- **save_name** (*None or str*) – The name of image file (i.e. image.png), if None, not to save image.

Returns The saved image.

Return type numpy.array

References

- OpenCV rectangle and putText.
- scikit-image.

Save image for pose estimation (MPII)

`tensorlayer.visualize.`**`draw_mpii_pose_to_image`**(*image*, *poses*, *save_name='image.png'*)

Draw people(s) into image using MPII dataset format as input, return or save the result image.

This is an experimental API, can be changed in the future.

Parameters

- **image** (*numpy.array*) – The RGB image [height, width, channel].
- **poses** (*list of dict*) – The people(s) annotation in MPII format, see `tl.files.load_mpii_pose_dataset`.
- **save_name** (*None or str*) – The name of image file (i.e. image.png), if None, not to save image.

Returns The saved image.

Return type numpy.array

Examples

```
>>> import pprint
>>> import tensorlayer as tl
>>> img_train_list, ann_train_list, img_test_list, ann_test_list = tl.files.load_
↪mpii_pose_dataset()
>>> image = tl.vis.read_image(img_train_list[0])
>>> tl.vis.draw_mpii_pose_to_image(image, ann_train_list[0], 'image.png')
>>> pprint.pprint(ann_train_list[0])
```

References

- MPII Keyponts and ID

2.13.2 Visualize model parameters

Visualize CNN 2d filter

`tensorlayer.visualize.`**`CNN2d`**(*CNN=None*, *second=10*, *saveable=True*, *name='cnn'*, *fig_idx=3119362*)

Display a group of RGB or Greyscale CNN masks.

Parameters

- **CNN** (*numpy.array*) – The image. e.g: 64 5x5 RGB images can be (5, 5, 3, 64).
- **second** (*int*) – The display second(s) for the image(s), if saveable is False.
- **saveable** (*boolean*) – Save or plot the figure.

- **name** (*str*) – A name to save the image, if saveable is True.
- **fig_idx** (*int*) – The matplotlib figure index.

Examples

```
>>> tl.visualize.CNN2d(network.all_params[0].eval(), second=10, saveable=True,␣
↪name='cnn1_mnist', fig_idx=2012)
```

Visualize weights

tensorlayer.visualize.**draw_weights**(*W=None*, *second=10*, *saveable=True*, *shape=None*, *name='mnist'*, *fig_idx=2396512*)

Visualize every columns of the weight matrix to a group of Greyscale img.

Parameters

- **W** (*numpy.array*) – The weight matrix
- **second** (*int*) – The display second(s) for the image(s), if saveable is False.
- **saveable** (*boolean*) – Save or plot the figure.
- **shape** (*a list with 2 int or None*) – The shape of feature image, MNIST is [28, 80].
- **name** (*a string*) – A name to save the image, if saveable is True.
- **fig_idx** (*int*) – matplotlib figure index.

Examples

```
>>> tl.visualize.draw_weights(network.all_params[0].eval(), second=10,␣
↪saveable=True, name='weight_of_1st_layer', fig_idx=2012)
```

2.13.3 Visualize images

Image by matplotlib

tensorlayer.visualize.**frame**(*I=None*, *second=5*, *saveable=True*, *name='frame'*, *cmap=None*, *fig_idx=12836*)

Display a frame. Make sure OpenAI Gym render() is disable before using it.

Parameters

- **I** (*numpy.array*) – The image.
- **second** (*int*) – The display second(s) for the image(s), if saveable is False.
- **saveable** (*boolean*) – Save or plot the figure.
- **name** (*str*) – A name to save the image, if saveable is True.
- **cmap** (*None or str*) – 'gray' for greyscale, None for default, etc.
- **fig_idx** (*int*) – matplotlib figure index.

Examples

```
>>> env = gym.make("Pong-v0")
>>> observation = env.reset()
>>> tl.visualize.frame(observation)
```

Images by matplotlib

tensorlayer.visualize.**images2d**(*images=None*, *second=10*, *saveable=True*, *name='images'*, *dtype=None*, *fig_idx=3119362*)

Display a group of RGB or Greyscale images.

Parameters

- **images** (*numpy.array*) – The images.
- **second** (*int*) – The display second(s) for the image(s), if saveable is False.
- **saveable** (*boolean*) – Save or plot the figure.
- **name** (*str*) – A name to save the image, if saveable is True.
- **dtype** (*None or numpy data type*) – The data type for displaying the images.
- **fig_idx** (*int*) – matplotlib figure index.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,
↪32, 32, 3), plotable=False)
>>> tl.visualize.images2d(X_train[0:100,:,:,:], second=10, saveable=False, name=
↪'cifar10', dtype=np.uint8, fig_idx=20212)
```

2.13.4 Visualize embeddings

tensorlayer.visualize.**tsne_embedding**(*embeddings*, *reverse_dictionary*, *plot_only=500*, *second=5*, *saveable=False*, *name='tsne'*, *fig_idx=9862*)

Visualize the embeddings by using t-SNE.

Parameters

- **embeddings** (*numpy.array*) – The embedding matrix.
- **reverse_dictionary** (*dictionary*) – id_to_word, mapping id to unique word.
- **plot_only** (*int*) – The number of examples to plot, choice the most common words.
- **second** (*int*) – The display second(s) for the image(s), if saveable is False.
- **saveable** (*boolean*) – Save or plot the figure.
- **name** (*str*) – A name to save the image, if saveable is True.
- **fig_idx** (*int*) – matplotlib figure index.

Examples

```
>>> see 'tutorial_word2vec_basic.py'
>>> final_embeddings = normalized_embeddings.eval()
>>> tl.visualize.tsne_embedding(final_embeddings, labels, reverse_dictionary,
...                   plot_only=500, second=5, saveable=False, name='tsne')
```

2.14 API - Database

This is the alpha version of database management system. If you have any trouble, please ask for help at tensorlayer@gmail.com .

2.14.1 Why Database

TensorLayer is designed for real world production, capable of large scale machine learning applications. TensorLayer database is introduced to address the many data management challenges in the large scale machine learning projects, such as:

1. Finding training data from an enterprise data warehouse.
2. Loading large datasets that are beyond the storage limitation of one computer.
3. Managing different models with version control, and comparing them(e.g. accuracy).
4. Automating the process of training, evaluating and deploying machine learning models.

With the TensorLayer system, we introduce this database technology to address the challenges above.

The database management system is designed with the following three principles in mind.

Everything is Data

Data warehouses can store and capture the entire machine learning development process. The data can be categorized as:

1. Dataset: This includes all the data used for training, validation and prediction. The labels can be manually specified or generated by model prediction.
2. Model architecture: The database includes a table that stores different model architectures, enabling users to reuse the many model development works.
3. Model parameters: This database stores all the model parameters of each epoch in the training step.
4. Tasks: A project usually include many small tasks. Each task contains the necessary information such as hyper-parameters for training or validation. For a training task, typical information includes training data, the model parameter, the model architecture, how many epochs the training task has. Validation, testing and inference are also supported by the task system.
5. Loggings: The logs store all the metrics of each machine learning model, such as the time stamp, loss and accuracy of each batch or epoch.

TensorLayer database in principle is a keyword based search engine. Each model, parameter, or training data is assigned many tags. The storage system organizes data into two layers: the index layer, and the blob layer. The index layer stores all the tags and references to the blob storage. The index layer is implemented based on NoSQL document database such as MongoDB. The blob layer stores videos, medical images or label masks in large chunk size, which is usually implemented based on a file system. Our database is based on MongoDB. The blob system is based on the GridFS while the indexes are stored as documents.

Everything is identified by Query

Within the database framework, any entity within the data warehouse, such as the data, model or tasks is specified by the database query language. As a reference, the query is more space efficient for storage and it can specify multiple objects in a concise way. Another advantage of such a design is enabling a highly flexible software system. Many system can be implemented by simply rewriting different components, with many new applications can be implemented just by update the query without modification of any application code.

2.14.2 Preparation

In principle, the database can be implemented by any document oriented NoSQL database system. The existing implementation is based on MongoDB. Further implementations on other databases will be released depending on the progress. It will be straightforward to port our database system to Google Cloud, AWS and Azure. The following tutorials are based on the MongoDB implementation.

Installing and running MongoDB

The installation instruction of MongoDB can be found at MongoDB Docs. There are also many MongoDB services from Amazon or GCP, such as Mongo Atlas from MongoDB User can also use docker, which is a powerful tool for deploying software . After installing MongoDB, a MongoDB management tool with graphic user interface will be extremely useful. Users can also install Studio3T(MongoChef), which is powerful user interface tool for MongoDB and is free for non-commercial use studio3t.

2.14.3 Tutorials

Connect to the database

Similar with MongoDB management tools, IP and port number are required for connecting to the database. To distinguish the different projects, the database instances have a `project_name` argument. In the following example, we connect to MongoDB on a local machine with the IP `localhost`, and port `27017` (this is the default port number of MongoDB).

```
db = tl.db.TensorHub(ip='localhost', port=27017, dbname='temp',
      username=None, password='password', project_name='tutorial')
```

Dataset management

You can save a dataset into the database and allow all machines to access it. Apart from the dataset key, you can also insert a custom argument such as version and description, for better managing the datasets. Note that, all saving functions will automatically save a timestamp, allowing you to load staff (data, model, task) using the timestamp.

```
db.save_dataset(dataset=[X_train, y_train, X_test, y_test], dataset_name='mnist',␣
↪description='this is a tutorial')
```

After saving the dataset, others can access the dataset as followed:

```
dataset = db.find_dataset('mnist')
dataset = db.find_dataset('mnist', version='1.0')
```

If you have multiple datasets that use the same dataset key, you can get all of them as followed:

```
datasets = db.find_all_datasets('mnist')
```

Model management

Save model architecture and parameters into database. The model architecture is represented by a TL graph, and the parameters are stored as a list of array.

```
db.save_model(net, accuracy=0.8, loss=2.3, name='second_model')
```

After saving the model into database, we can load it as follow:

```
net = db.find_model(sess=sess, accuracy=0.8, loss=2.3)
```

If there are many models, you can use MongoDB's 'sort' method to find the model you want. To get the newest or oldest model, you can sort by time:

```
## newest model

net = db.find_model(sess=sess, sort=[("time", pymongo.DESCENDING)])
net = db.find_model(sess=sess, sort=[("time", -1)])

## oldest model

net = db.find_model(sess=sess, sort=[("time", pymongo.ASCENDING)])
net = db.find_model(sess=sess, sort=[("time", 1)])
```

If you save the model along with accuracy, you can get the model with the best accuracy as followed:

```
net = db.find_model(sess=sess, sort=[("test_accuracy", -1)])
```

To delete all models in a project:

```
db.delete_model()
```

If you want to specify which model you want to delete, you need to put arguments inside.

Event / Logging management

Save training log:

```
db.save_training_log(accuracy=0.33)
db.save_training_log(accuracy=0.44)
```

Delete logs that match the requirement:

```
db.delete_training_log(accuracy=0.33)
```

Delete all logging of this project:

```
db.delete_training_log()
db.delete_validation_log()
db.delete_testing_log()
```

Task distribution

A project usually consists of many tasks such as hyper parameter selection. To make it easier, we can distribute these tasks to several GPU servers. A task consists of a task script, hyper parameters, desired result and a status.

A task distributor can push both dataset and tasks into a database, allowing task runners on GPU servers to pull and run. The following is an example that pushes 3 tasks with different hyper parameters.

```
## save dataset into database, then allow other servers to use it
X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_mnist_dataset(shape=(-
↪1, 784))
db.save_dataset((X_train, y_train, X_val, y_val, X_test, y_test), 'mnist',␣
↪description='handwriting digit')

## push tasks into database, then allow other servers pull tasks to run
db.create_task(
    task_name='mnist', script='task_script.py', hyper_parameters=dict(n_units1=800, n_
↪units2=800),
    saved_result_keys=['test_accuracy'], description='800-800'
)

db.create_task(
    task_name='mnist', script='task_script.py', hyper_parameters=dict(n_units1=600, n_
↪units2=600),
    saved_result_keys=['test_accuracy'], description='600-600'
)

db.create_task(
    task_name='mnist', script='task_script.py', hyper_parameters=dict(n_units1=400, n_
↪units2=400),
    saved_result_keys=['test_accuracy'], description='400-400'
)

## wait for tasks to finish
while db.check_unfinished_task(task_name='mnist'):
    print("waiting runners to finish the tasks")
    time.sleep(1)

## you can get the model and result from database and do some analysis at the end
```

The task runners on GPU servers can monitor the database, and run the tasks immediately when they are made available. In the task script, we can save the final model and results to the database, this allows task distributors to get the desired model and results.

```
## monitors the database and pull tasks to run
while True:
    print("waiting task from distributor")
    db.run_task(task_name='mnist', sort=[("time", -1)])
    time.sleep(1)
```

Example codes

See here.

2.14.4 TensorHub API

class tensorlayer.db.**TensorHub**(*ip='localhost'*, *port=27017*, *dbname='dbname'*, *username='None'*, *password='password'*, *project_name=None*)

It is a MongoDB based manager that help you to manage data, network architecture, parameters and logging.

Parameters

- **ip** (*str*) – Localhost or IP address.
- **port** (*int*) – Port number.
- **dbname** (*str*) – Database name.
- **username** (*str or None*) – User name, set to None if you do not need authentication.
- **password** (*str*) – Password.
- **project_name** (*str or None*) – Experiment key for this entire project, similar with the repository name of Github.

ip, port, dbname and other input parameters

See above.

Type see above

project_name

The given project name, if no given, set to the script name.

Type str

db

See pymongo.MongoClient.

Type mongodb client

check_unfinished_task(*task_name=None*, ***kwargs*)

Finds and runs a pending task.

Parameters

- **task_name** (*str*) – The task name.
- **kwargs** (*other parameters*) – Users customized parameters such as description, version number.

Examples

Wait until all tasks finish in user's local console

```
>>> while not db.check_unfinished_task():
>>>     time.sleep(1)
>>> print("all tasks finished")
>>> sess = tf.InteractiveSession()
>>> net = db.find_top_model(sess=sess, sort=[("test_accuracy", -1)])
>>> print("the best accuracy {} is from model {}".format(net._test_accuracy,␣
↪net._name))
```

Returns boolean

Return type True for success, False for fail.

create_task(*task_name=None*, *script=None*, *hyper_parameters=None*, *saved_result_keys=None*, ***kwargs*)

Uploads a task to the database, timestamp will be added automatically.

Parameters

- **task_name** (*str*) – The task name.
- **script** (*str*) – File name of the python script.
- **hyper_parameters** (*dictionary*) – The hyper parameters pass into the script.
- **saved_result_keys** (*list of str*) – The keys of the task results to keep in the database when the task finishes.
- **kwargs** (*other parameters*) – Users customized parameters such as description, version number.

Examples

Uploads a task >>> db.create_task(task_name='mnist', script='example/tutorial_mnist_simple.py', description='simple tutorial')

Finds and runs the latest task >>> db.run_top_task(sort=[("time", pymongo.DESCENDING)]) >>> db.run_top_task(sort=[("time", -1)])

Finds and runs the oldest task >>> db.run_top_task(sort=[("time", pymongo.ASCENDING)]) >>> db.run_top_task(sort=[("time", 1)])

delete_datasets(***kwargs*)

Delete datasets.

Parameters kwargs (*logging information*) – Find items to delete, leave it empty to delete all log.

delete_model(***kwargs*)

Delete model.

Parameters kwargs (*logging information*) – Find items to delete, leave it empty to delete all log.

delete_tasks(***kwargs*)

Delete tasks.

Parameters kwargs (*logging information*) – Find items to delete, leave it empty to delete all log.

Examples

```
>>> db.delete_tasks()
```

delete_testing_log(***kwargs*)

Deletes testing log.

Parameters kwargs (*logging information*) – Find items to delete, leave it empty to delete all log.

Examples

- see `save_training_log`.

delete_training_log(***kwargs*)
Deletes training log.

Parameters **kwargs** (*logging information*) – Find items to delete, leave it empty to delete all log.

Examples

Save training log >>> db.save_training_log(accuracy=0.33) >>> db.save_training_log(accuracy=0.44)

Delete logs that match the requirement >>> db.delete_training_log(accuracy=0.33)

Delete all logs >>> db.delete_training_log()

delete_validation_log(***kwargs*)
Deletes validation log.

Parameters **kwargs** (*logging information*) – Find items to delete, leave it empty to delete all log.

Examples

- see `save_training_log`.

find_datasets(*dataset_name=None*, ***kwargs*)
Finds and returns all datasets from the database which matches the requirement. In some case, the data in a dataset can be stored separately for better management.

Parameters

- **dataset_name** (*str*) – The name/key of dataset.
- **kwargs** (*other events*) – Other events, such as description, author and etc (optional).

Returns params

Return type the parameters, return False if nothing found.

find_top_dataset(*dataset_name=None*, *sort=None*, ***kwargs*)
Finds and returns a dataset from the database which matches the requirement.

Parameters

- **dataset_name** (*str*) – The name of dataset.
- **sort** (*List of tuple*) – PyMongo sort comment, search "PyMongo find one sorting" and collection level operations for more details.
- **kwargs** (*other events*) – Other events, such as description, author and etc (optinal).

Examples

Save dataset >>> db.save_dataset([X_train, y_train, X_test, y_test], 'mnist', description='this is a tutorial')

Get dataset >>> dataset = db.find_top_dataset('mnist') >>> datasets = db.find_datasets('mnist')

Returns dataset – Return False if nothing found.

Return type the dataset or False

find_top_model (*sort=None*, *model_name='model'*, ***kwargs*)
Finds and returns a model architecture and its parameters from the database which matches the requirement.

Parameters

- **sort** (`List of tuple`) – PyMongo sort comment, search "PyMongo find one sorting" and collection level operations for more details.
- **model_name** (`str or None`) – The name/key of model.
- **kwargs** (`other events`) – Other events, such as name, accuracy, loss, step number and etc (optinal).

Examples

- see `save_model`.

Returns network – Note that, the returned network contains all information of the document (record), e.g. if you saved accuracy in the document, you can get the accuracy by using `net._accuracy`.

Return type TensorLayer Model

run_top_task (*task_name=None*, *sort=None*, ***kwargs*)
Finds and runs a pending task that in the first of the sorting list.

Parameters

- **task_name** (`str`) – The task name.
- **sort** (`List of tuple`) – PyMongo sort comment, search "PyMongo find one sorting" and collection level operations for more details.
- **kwargs** (`other parameters`) – Users customized parameters such as description, version number.

Examples

Monitors the database and pull tasks to run >>> while True: >>> print("waiting task from distributor") >>> db.run_top_task(task_name='mnist', sort=[("time", -1)]) >>> time.sleep(1)

Returns boolean

Return type True for success, False for fail.

save_dataset (*dataset=None*, *dataset_name=None*, ***kwargs*)
Saves one dataset into database, timestamp will be added automatically.

Parameters

- **dataset** (*any type*) – The dataset you want to store.
- **dataset_name** (*str*) – The name of dataset.
- **kwargs** (*other events*) – Other events, such as description, author and etc (optinal).

Examples

Save dataset >>> db.save_dataset([X_train, y_train, X_test, y_test], 'mnist', description='this is a tutorial')

Get dataset >>> dataset = db.find_top_dataset('mnist')

Returns boolean

Return type Return True if save success, otherwise, return False.

save_model (*network=None*, *model_name='model'*, ***kwargs*)
Save model architecture and parameters into database, timestamp will be added automatically.

Parameters

- **network** (*TensorLayer Model*) – TensorLayer Model instance.
- **model_name** (*str*) – The name/key of model.
- **kwargs** (*other events*) – Other events, such as name, accuracy, loss, step number and etc (optinal).

Examples

Save model architecture and parameters into database. >>> db.save_model(net, accuracy=0.8, loss=2.3, name='second_model')

Load one model with parameters from database (run this in other script) >>> net = db.find_top_model(accuracy=0.8, loss=2.3)

Find and load the latest model. >>> net = db.find_top_model(sort=[("time", pymongo.DESCENDING)]) >>> net = db.find_top_model(sort=[("time", -1)])

Find and load the oldest model. >>> net = db.find_top_model(sort=[("time", pymongo.ASCENDING)]) >>> net = db.find_top_model(sort=[("time", 1)])

Get model information >>> net._accuracy … 0.8

Returns boolean

Return type True for success, False for fail.

save_testing_log (***kwargs*)
Saves the testing log, timestamp will be added automatically.

Parameters kwargs (*logging information*) – Events, such as accuracy, loss, step number and etc.

Examples

```
>>> db.save_testing_log(accuracy=0.33, loss=0.98)
```

save_training_log (***kwargs*)
Saves the training log, timestamp will be added automatically.

Parameters **kwargs** (*logging information*) – Events, such as accuracy, loss, step number and etc.

Examples

```
>>> db.save_training_log(accuracy=0.33, loss=0.98)
```

save_validation_log(***kwargs*)
Saves the validation log, timestamp will be added automatically.

Parameters **kwargs** (*logging information*) – Events, such as accuracy, loss, step number and etc.

Examples

```
>>> db.save_validation_log(accuracy=0.33, loss=0.98)
```

2.15 API - Optimizers

TensorLayer provides rich layer implementations trailed for various benchmarks and domain-specific problems. In addition, we also support transparent access to native TensorFlow parameters. For example, we provide not only layers for local response normalization, but also layers that allow user to apply `tf.nn.lrn` on `network.outputs`. More functions can be found in TensorFlow API.

TensorLayer provides simple API and tools to ease research, development and reduce the time to production. Therefore, we provide the latest state of the art optimizers that work with Tensorflow.

2.15.1 Optimizers List

AMSGrad([learning_rate, beta1, beta2, . . .])	Implementation of the AMSGrad optimization algorithm.

2.15.2 AMSGrad Optimizer

class tensorlayer.optimizers.**AMSGrad**(*learning_rate=0.01*, *beta1=0.9*, *beta2=0.99*, *epsilon=1e-08*, *use_locking=False*, *name='AMSGrad'*)

Implementation of the AMSGrad optimization algorithm.

See: On the Convergence of Adam and Beyond - [Reddi et al., 2018].

Parameters

- **learning_rate** (*float*) – A Tensor or a floating point value. The learning rate.
- **beta1** (*float*) – A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
- **beta2** (*float*) – A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
- **epsilon** (*float*) – A small constant for numerical stability. This epsilon is "epsilon

hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper.

- **use_locking** (*bool*) – If True use locks for update operations.
- **name** (*str*) – Optional name for the operations created when applying gradients. Defaults to "AMSGrad".

2.16 API - Distributed Training

(Alpha release - usage might change later)

Helper API to run a distributed training. Check these examples.

Trainer(training_dataset, . . . [, batch_size, . . .])	Trainer for neural networks in a distributed environment.

2.16.1 Distributed training

Trainer

`tensorlayer.distributed.`**Trainer**(*training_dataset*, *build_training_func*, *optimizer*, *optimizer_args*, *batch_size=32*, *prefetch_size=None*, *checkpoint_dir=None*, *scaling_learning_rate=True*, *log_step_size=1*, *validation_dataset=None*, *build_validation_func=None*, *max_iteration=inf*)

Trainer for neural networks in a distributed environment.

TensorLayer Trainer is a high-level training interface built on top of TensorFlow MonitoredSession and Horovod. It transparently scales the training of a TensorLayer model from a single GPU to multiple GPUs that be placed on different machines in a single cluster.

To run the trainer, you will need to install Horovod on your machine. Check the installation script at *tensorlayer/scripts/download_and_install_openmpi3_ubuntu.sh*

The minimal inputs to the Trainer include (1) a training dataset defined using the TensorFlow DataSet API, and (2) a model build function given the inputs of the training dataset, and returns the neural network to train, the loss function to minimize, and the names of the tensor to log during training, and (3) an optimizer and its arguments.

The default parameter choices of Trainer is inspired by the Facebook paper: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Parameters

- **training_dataset** (class TensorFlow `DataSet`) – The training dataset which zips samples and labels. The trainer automatically shards the training dataset based on the number of GPUs.
- **build_training_func** (*function*) – A function that builds the training operator. It takes the training dataset as an input, and returns the neural network, the loss function and a dictionary that maps string tags to tensors to log during training.
- **optimizer** (class TensorFlow `Optimizer`) – The loss function optimizer. The trainer automatically linearly scale the learning rate based on the number of GPUs.

- **optimizer_args** (*dict*) – The optimizer argument dictionary. It must contain a *learning_rate* field in type of float. Note that the learning rate is linearly scaled according to the number of GPU by default. You can disable it using the option *scaling_learning_rate*
- **batch_size** (*int*) – The training mini-batch size (i.e., number of samples per batch).
- **prefetch_size** (*int or None*) – The dataset prefetch buffer size. Set this parameter to overlap the GPU training and data preparation if the data preparation is heavy.
- **checkpoint_dir** (*None or str*) – The path to the TensorFlow model checkpoint. Note that only one trainer master would checkpoints its model. If None, checkpoint is disabled.
- **log_step_size** (*int*) – The trainer logs training information every N mini-batches (i.e., step size).
- **validation_dataset** (None or class TensorFlow `DataSet`) – The optional validation dataset that zips samples and labels. Note that only the trainer master needs to the validation often.
- **build_validation_func** (*None or function*) – The function that builds the validation operator. It returns the validation neural network (which share the weights of the training network) and a custom number of validation metrics.
- **scaling_learning_rate** (*Boolean*) – Linearly scale the learning rate by the number of GPUs. Default is True. This *linear scaling rule* is generally effective and is highly recommended by the practioners. Check Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
- **max_iteration** (*int*) – The maximum iteration (i.e., mini-batch) to train. The default is *math.inf*. You can set it to a small number to end the training earlier. This is usually set for testing purpose.

`tensorlayer.distributed.`**`training_network`**
The training model.

Type class TensorLayer `Layer`

`tensorlayer.distributed.`**`session`**
The training session tha the Trainer wraps.

Type class TensorFlow `MonitoredTrainingSession`

`tensorlayer.distributed.`**`global_step`**
The number of training mini-batch by far.

Type int

`tensorlayer.distributed.`**`validation_metrics`**
The validation metrics that zips the validation metric property and the average value.

Type list of tuples

Examples

See tutorial_mnist_distributed_trainer.py.

CHAPTER

THREE

COMMAND-LINE REFERENCE

TensorLayer provides a handy command-line tool *tl* to perform some common tasks.

3.1 CLI - Command Line Interface

The tensorlayer.cli module provides a command-line tool for some common tasks.

3.1.1 tl train

(Alpha release - usage might change later)

The tensorlayer.cli.train module provides the `tl train` subcommand. It helps the user bootstrap a TensorFlow/TensorLayer program for distributed training using multiple GPU cards or CPUs on a computer.

You need to first setup the CUDA_VISIBLE_DEVICES to tell `tl train` which GPUs are available. If the CUDA_VISIBLE_DEVICES is not given, `tl train` would try best to discover all available GPUs.

In distribute training, each TensorFlow program needs a TF_CONFIG environment variable to describe the cluster. It also needs a master daemon to monitor all trainers. `tl train` is responsible for automatically managing these two tasks.

Usage

tl train [-h] [-p NUM_PSS] [-c CPU_TRAINERS] <file> [args [args ...]]

```
# example of using GPU 0 and 1 for training mnist
CUDA_VISIBLE_DEVICES="0,1"
tl train example/tutorial_mnist_distributed.py

# example of using CPU trainers for inception v3
tl train -c 16 example/tutorial_imagenet_inceptionV3_distributed.py

# example of using GPU trainers for inception v3 with customized arguments
# as CUDA_VISIBLE_DEVICES is not given, tl would try to discover all available GPUs
tl train example/tutorial_imagenet_inceptionV3_distributed.py -- --batch_size 16
```

Command-line Arguments

- `file`: python file path.
- `NUM_PSS` : The number of parameter servers.

- `CPU_TRAINERS`: The number of CPU trainers.

  It is recommended that `NUM_PSS + CPU_TRAINERS <= cpu count`
- `args`: Any parameter after `--` would be passed to the python program.

Notes

A parallel training program would require multiple parameter servers to help parallel trainers to exchange intermediate gradients. The best number of parameter servers is often proportional to the size of your model as well as the number of CPUs available. You can control the number of parameter servers using the `-p` parameter.

If you have a single computer with massive CPUs, you can use the `-c` parameter to enable CPU-only parallel training. The reason we are not supporting GPU-CPU co-training is because GPU and CPU are running at different speeds. Using them together in training would incur stragglers.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F

G

M

N

O

P

R

S

T

U

V

W

Z