TensorLayer Documentation

发布 ***1.6.3***

TensorLayer contributors

2017 年 **09** 月 **22** 日

Contents

TensorLayer 是为研究人员和工程师设计的一款基于Google TensorFlow开发的深度学习与强化学习库。 它提供高级别的（Higher-Level）深度学习API，这样不仅可以加快研究人员的实验速度，也能够减少工程师在实际开发当中的重复工作。 TensorLayer非常易于修改和扩展，这使它可以同时用于机器学习的研究与应用。 此外，TensorLayer 提供了大量示例和教程来帮助初学者理解深度学习，并提供大量的官方例子程序方便开发者快速找到适合自己项目的例子。

这篇文档不仅仅是为了描述如何使用这个库也是一个遍历不同类型的神经网络， 深度强化学习和自然语言处理等内容的教程。 此外，TensorLayer的Tutorial包含了所有TensorFlow官方深度学习教程的模块化实现，因此你可以对照TensorFlow深度学习教程来学习 [英文] [极客学院中文翻译]

注解： 我们建议你在 Github 上star和watch 官方项目 ，这样当官方有更新时，你会立即知道。本文档为 官方RTD文档 的翻译版，更新速度会比英文原版慢，若你的英文还行，我们建议你直接阅读 官方RTD文档。

如果你阅读在线中文文档时有什么问题，你可以在github上下载这个项目， 然后去 `/docs/cn/_build/html/index.html` 阅读离线中文文档。 或者在 Read the docs 中阅读官方原文档。

CHAPTER 1

用户指南

TensorLayer用户指南说明了如何去安装TensorFlow,CUDA和cuDNN， 然后如何用TensorLayer建立和训练神经网络和如何作为（一个开发者支持这个库。）

1.1 安装 Installation

TensorLayer 需要一些预安装库，如 TensorFlow ， numpy 和 matplotlib。 对于 GPU 加速，需要安装 CUDA 和 cuDNN。

如果你遇到麻烦，可以查看 TensorFlow 安装手册 ，它包含了在不同系统中安装 TensorFlow 的步骤。或发邮件到 hao.dong11@imperial.ac.uk 询问。

1.1.1 步骤 1 : 安装依赖库 dependencies

TensorLayer 是运行在 python 版本的 TensorFlow 之上的，所以请先安装 python。

注解: 着眼于未来，我们建议使用 python3 而不是 python2

Python 的 `pip` 可以帮助您很快地安装库，此外 虚拟环境(Virtual environment) 如 `virtualenv` 可以帮助你管理 python 包。

以 python3 和 Ubuntu 为例，可如下安装 python 及 `pip`:

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

接着在虚拟环境中安装 dependencies 到虚拟环境如下: (您也可以跳过该部分，在步骤3中让 TensorLayer 自动安装 dependencies)

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

安装完后，若无报错，可以如下在命令窗口中显示列出安装好的包:

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

最后，你可以用虚拟环境中的 python 来运行 python 代码，如下:

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

1.1.2 步骤 2 : TensorFlow

TensorFlow 的安装步骤在 TensorFlow 官网中有非常详细的说明，不过有一些东西是需要考虑的。 如 TensorFlow 官方目前支持 Linux, Mac OX 和 Windows 系统。

警告：对于使用 ARM 架构的机器，你需要用源码来编译安装 TensorFlow。

1.1.3 步骤 3 : TensorLayer

最便捷安装 TensorLayer 只需要一个指令，如下:

```
pip install git+https://github.com/zsdonghao/tensorlayer.git
```

不过，若你是高级玩家，你想要在 TensorLayer 的基础上拓展或修改源码，你可以从 Github 中把整个项目下载下来， 然后如下安装。

```
cd 到项目文件
pip install -e .
```

这个命令会运行 `setup.py` 来安装 TensorLayer。 其中， `-e` 代表 可编辑的（editable），因此你可以直接修改 `tensorlayer` 文件夹中的源代码，然后 `import` 该文件夹来使用修改后的 TensorLayer。

1.1.4 步骤 4 : GPU 加速

非常感谢 NVIDIA 的支持，在 GPU 上训练全连接神经网络比在 CPU 上训练往往要快 10~20 倍。 对于卷积神经网络，往往会快 50 倍。这需要有一个 NIVIDA 的 GPU，以及安装 CUDA 和 cuDNN。

CUDA

TensorFlow 官网讲了如何安装 CUDA 和 cuDNN，TensorFlow GPU 支持。

可在 NIVIDIA 官网下载与安装最新版本的 CUDA。

- CUDA 下载与安装

若 CUDA 被正确地安装，下面的指令可以在命令窗口中打印出 GPU 的信息。

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

cuDNN

出了 CUDA，NVIDIA 还专门提供另一个库来加速神经网络的运算，特别是用来加速卷积神经网络。这个库也可以从 NIVIDIA 官网中下载安装，但你要先注册为 NIVIDA 开发者（这需要一些审核时间）。下载时，请在 Deep Learning Framework 处在 Other 中输入 TensorLayer。

最新 cuDNN 下载与安装链接：

- cuDNN 下载与安装

下载后，复制 `*.h` 文件到 `/usr/local/cuda/include` 以及复制 `lib*` 文件到 `/usr/local/cuda/lib64`。

1.1.5 Windows 用户

Tensorflow于2016年11月28日发布0.12版本，添加了windows版本支持，Tensorlayer使用Tensorflow作为后端，也因此支持windows版本。根据Tensorflow官网说明，windows版本的最低系统要求为windows7，最低语言版本要求为python3.5。可以选择CPU和GPU两个版本。

Python 环境搭建

Python环境搭建我们建议使用科学计算集成python发行版Anaconda，Anaconda里面集成了大量常用的科学计算库，并且自带matlab风格的IDE Spyder，方便平时的开发使用。当然用户也可以根据自己的使用习惯选择合适的安装方式，但是python的版本最低要求为python3.5。

Anaconda 下载地址

GPU 环境搭建

如果想使用GPU版本的 TensorLayer，需要安装GPU支持，而CPU版本不需要。

编译环境 Microsoft Visual Studio

安装NVIDIA的CUDA显卡驱动需要预安装编译环境VS，VS最低的版本要求为VS2010，但我们建议安装较新的版本VS2015或者VS2013。其中CUDA7.5仅支持2010、2012、2013，CUDA8.0同时支持2015版本。

CUDA 驱动安装

为了使用显卡进行GPU加速运算，需要安装NVIDIA的CUDA驱动，我们建议安装最新版的CUDA8.0，并根据操作系统下载对应的版本。我们建议使用local安装的方式，以防出现安装过程中因为网络中断造成安装失败的现象。安装过程中所有的选择直接选择默认，如果C盘空间足够，不建议手动更改安装目录。

CUDA下载地址

加速库 cuDNN 安装

cuDNN是NVIDIA针对深度学习计算的一个加速，建议安装。您可能需要注册一个账号才能下载cuDNN，然后根据CUDA的版本和windows的版本下载相应的cuDNN源码，我们建议下载最新版的cuDNN5.1。下载下来之后直接解压，解压之后有三个夹bin,lib,include，把解压之后的三个文件夹直接复制到CUDA的安装目录。（默认的安装目录为：*C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0*）

TensorLayer 框架搭建

首先我们需要安装TensorFlow框架，在CMD命令行直接用pip命令进行安装：

```
pip install tensorflow      # CPU 版本 (二选一)
pip install tensorflow-gpu  # GPU 版本 (二选一)
pip install tensorlayer     # 之后安装 TensorLayer 框架
```

测试

在CMD命令行输入python进入Python环境，输入：

```
import tensorlayer
```

如果未报错并且显示以下输出，则GPU版本安装成功

```
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
```

如果未报错则CPU版本安装成功。

1.1.6 困难

当你 import tensorlayer 时出现如下错误，请阅读 FQA 。

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

1.2 教程 Tutorials

对于深度学习，该教程会引导您使用MNIST数据集构建不同的手写数字的分类器， 这可以说是神经网络的 "Hello World" 。 对于强化学习，我们将让计算机根据屏幕输入来学习打乒乓球。 对于自然语言处理。我们从词嵌套（word embedding）开始，然后再实现语言建模和机器翻译。 此外，TensorLayer的Tutorial包含了所有TensorFlow官方深度学习教程的模块化实现，因此你可以对照TensorFlow深度学习教程来学习 [英文] [极客学院中文翻译] 。

注解： 若你已经对TensorFlow非常熟悉，阅读 `InputLayer` 和 `DenseLayer` 的源代码可让您很好地理解 TensorLayer 是如何工作的。

1.2.1 在我们开始之前

本教程假定您在神经网络和TensorFlow(TensorLayer在它的基础上构建的)方面具有一定的基础。 您可以尝试从 Deeplearning Tutorial 同时进行学习。

对于人工神经网络更系统的介绍，我们推荐Andrej Karpathy等人所著的 Convolutional Neural Networks for Visual Recognition 和Michael Nielsen Neural Networks and Deep Learning 。

要了解TensorFlow的更多内容，请阅读 TensorFlow tutorial 。 您不需要会它的全部，只要知道TensorFlow是如何工作的，就能够使用TensorLayer。 如果您是TensorFlow的新手，建议你阅读整个教程。

1.2.2 TensorLayer很简单

下面的代码是TensorLayer的一个简单例子，来自 `tutorial_mnist_simple.py` 。 我们提供了很多方便的函数（如： `fit()` , `test()` ），但如果你想了解更多实现细节，或想成为机器学习领域的专家，我们鼓励 您尽可能地直接使用TensorFlow原本的方法如 `sess.run()` 来训练模型，请参考 `tutorial_mnist.py` 。

```
import tensorflow as tf
import tensorlayer as tl

sess = tf.InteractiveSession()

# 准备数据
X_train, y_train, X_val, y_val, X_test, y_test = \
                                tl.files.load_mnist_dataset(shape=(-1,784))

# 定义 placeholder
x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')

# 定义模型
network = tl.layers.InputLayer(x, name='input_layer')
network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu1')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu2')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
network = tl.layers.DenseLayer(network, n_units=10,
                                act = tf.identity,
                                name='output_layer')
# 定义损失函数和衡量指标
# tl.cost.cross_entropy 在内部使用 tf.nn.sparse_softmax_cross_entropy_with_logits() 实现␣
↪softmax
y = network.outputs
cost = tl.cost.cross_entropy(y, y_, name = 'cost')
correct_prediction = tf.equal(tf.argmax(y, 1), y_)
acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
y_op = tf.argmax(tf.nn.softmax(y), 1)

# 定义 optimizer
train_params = network.all_params
train_op = tf.train.AdamOptimizer(learning_rate=0.0001, beta1=0.9, beta2=0.999,
                            epsilon=1e-08, use_locking=False).minimize(cost, var_
↪list=train_params)
```

```
# 初始化 session 中的所有参数
tl.layers.initialize_global_variables(sess)

# 列出模型信息
network.print_params()
network.print_layers()

# 训练模型
tl.utils.fit(sess, network, train_op, cost, X_train, y_train, x, y_,
            acc=acc, batch_size=500, n_epoch=500, print_freq=5,
            X_val=X_val, y_val=y_val, eval_train=False)

# 评估模型
tl.utils.test(sess, network, acc, X_test, y_test, x, y_, batch_size=None, cost=cost)

# 把模型保存成 .npz 文件
tl.files.save_npz(network.all_params , name='model.npz')
sess.close()
```

1.2.3 运行MNIST例子

在本教程的第一部分，我们仅仅运行TensorLayer内置的MNIST例子。 MNIST数据集包含了60000个28x28像素的手写数字图片，它通常用于训练各种图片识别系统。

我们假设您已经按照 安装 *Installation* 安装好了TensorLayer。 如果您还没有，请复制一个TensorLayer的source目录到终端中，并进入该文件夹， 然后运行 `tutorial_mnist.py` 这个例子脚本：

```
python tutorial_mnist.py
```

如果所有设置都正确，您将得到下面的结果：

```
tensorlayer: GPU MEM Fraction 0.300000
Downloading train-images-idx3-ubyte.gz
Downloading train-labels-idx1-ubyte.gz
Downloading t10k-images-idx3-ubyte.gz
Downloading t10k-labels-idx1-ubyte.gz

X_train.shape (50000, 784)
y_train.shape (50000,)
X_val.shape (10000, 784)
```

```
y_val.shape (10000,)
X_test.shape (10000, 784)
y_test.shape (10000,)
X float32   y int64

tensorlayer:Instantiate InputLayer input_layer (?, 784)
tensorlayer:Instantiate DropoutLayer drop1: keep: 0.800000
tensorlayer:Instantiate DenseLayer relu1: 800, relu
tensorlayer:Instantiate DropoutLayer drop2: keep: 0.500000
tensorlayer:Instantiate DenseLayer relu2: 800, relu
tensorlayer:Instantiate DropoutLayer drop3: keep: 0.500000
tensorlayer:Instantiate DenseLayer output_layer: 10, identity

param 0: (784, 800) (mean: -0.000053, median: -0.000043 std: 0.035558)
param 1: (800,) (mean: 0.000000, median: 0.000000 std: 0.000000)
param 2: (800, 800) (mean: 0.000008, median: 0.000041 std: 0.035371)
param 3: (800,) (mean: 0.000000, median: 0.000000 std: 0.000000)
param 4: (800, 10) (mean: 0.000469, median: 0.000432 std: 0.049895)
param 5: (10,) (mean: 0.000000, median: 0.000000 std: 0.000000)
num of params: 1276810

layer 0: Tensor("dropout/mul_1:0", shape=(?, 784), dtype=float32)
layer 1: Tensor("Relu:0", shape=(?, 800), dtype=float32)
layer 2: Tensor("dropout_1/mul_1:0", shape=(?, 800), dtype=float32)
layer 3: Tensor("Relu_1:0", shape=(?, 800), dtype=float32)
layer 4: Tensor("dropout_2/mul_1:0", shape=(?, 800), dtype=float32)
layer 5: Tensor("add_2:0", shape=(?, 10), dtype=float32)

learning_rate: 0.000100
batch_size: 128

Epoch 1 of 500 took 0.342539s
  train loss: 0.330111
  val loss: 0.298098
  val acc: 0.910700
Epoch 10 of 500 took 0.356471s
  train loss: 0.085225
  val loss: 0.097082
  val acc: 0.971700
Epoch 20 of 500 took 0.352137s
  train loss: 0.040741
  val loss: 0.070149
  val acc: 0.978600
Epoch 30 of 500 took 0.350814s
  train loss: 0.022995
  val loss: 0.060471
  val acc: 0.982800
Epoch 40 of 500 took 0.350996s
  train loss: 0.013713
  val loss: 0.055777
  val acc: 0.983700
...
```

这个例子脚本允许您从 `if__name__=='__main__':` 中选择不同的模型进行尝试，包括多层神经网络（Multi-Layer Perceptron），退出（Dropout），退出连接（DropConnect），堆栈式降噪自编码器（Stacked Denoising Autoencoder）和卷积神经网络（CNN）。

```
main_test_layers(model='relu')
main_test_denoise_AE(model='relu')
main_test_stacked_denoise_AE(model='relu')
main_test_cnn_layer()
```

1.2.4 理解MNIST例子

现在就让我们看看它是如何做到的！跟着下面的步骤，打开源代码。

序言

您可能会首先注意到，除TensorLayer之外，我们还导入了Numpy和TensorFlow：

```
import tensorflow as tf
import tensorlayer as tl
from tensorlayer.layers import set_keep
import numpy as np
import time
```

这是因为TensorLayer是建立在TensorFlow上的，TensorLayer设计的初衷是为了简化工作并提供帮助而不是取代TensorFlow。所以您会需要一起使用TensorLayer和一些常见的TensorFlow代码。

请注意，当使用降噪自编码器(Denoising Autoencoder)时，代码中的 `set_keep` 被当作用来访问保持概率(Keeping Probabilities)的占位符。

载入数据

下面第一部分的代码首先定义了 `load_mnist_dataset()` 函数。其目的是为了下载MNIST数据集（如果还未下载），并且返回标准numpy数列通过numpy array的格式。到这里还没有涉及TensorLayer，所以我们可以把它简单看作：

```
X_train, y_train, X_val, y_val, X_test, y_test = \
                  tl.files.load_mnist_dataset(shape=(-1,784))
```

`X_train.shape` 为 `(50000,784)`，可以理解成共有50000张图片并且每张图片有784个像素点。`Y_train.shape` 为 `(50000,)`，它是一个和 `X_train` 长度相同的向量，用于给出每幅图的数字标签，即这些图片所包含的位于0-9之间的数字（如果画这些数字的人没有想乱画别的东西）。

另外对于卷积神经网络的例子，MNIST还可以按下面的4D版本来载入：

```
X_train, y_train, X_val, y_val, X_test, y_test = \
            tl.files.load_mnist_dataset(shape=(-1, 28, 28, 1))
```

`X_train.shape` 为 `(50000,28,28,1)`，这代表了50000张图片，每张图片使用一个通道（Channel），28行，28列。通道为1是因为它是灰度图像，每个像素只能有一个值。

建立模型

来到这里，就轮到TensorLayer来一显身手了！TensorLayer允许您通过创建，堆叠或者合并图层(Layers)来定义任意结构的神经网络。由于每一层都知道它在一个网络中的直接输入层和（多个）输出接收层，[###] 所以通常这是我们唯一要传递给其他代码的内容。

正如上文提到的， `tutorial_mnist.py` 支持四类模型，[###]。 首先，我们将定义一个结构固定的多层次感知器（Multi-Layer Perceptron），所有的步骤都会详细的讲解。 然后，我们会实现一个去噪自编码器(Denosing Autoencoding)。 接着，我们要将所有去噪自编码器堆叠起来并对他们进行监督微调(Supervised Fine-tune)。 最后，我们将展示如何去创建一个卷积神经网络(Convolutional Neural Network)。

此外，如果您有兴趣，我们还提供了一个简化版的MNIST例子在 `tutorial_mnist_simple.py` 中，和一个对于 -10数据集的卷积神经网络(CNN)的例子在 `tutorial_cifar10_tfrecord.py` 中可供参考。

多层神经网络 (Multi-Layer Perceptron)

第一个脚本 `main_test_layers()` ，创建了一个具有两个隐藏层，每层800个单元的多层次感知器，并且具有10个单元的SOFTMAX输出层紧随其后。 它对输入数据采用20%的退出率(dropout)并且对隐藏层应用50%的退出率(dropout)。

为了提供数据给这个网络，TensorFlow占位符(placeholder)需要按如下定义。 在这里 `None` 是指在编译之后，网络将接受任意批规模(batchsize)的数据 `x` 是用来存放 `X_train` 数据的并且 `y_` 是用来存放 `y_train` 数据的。 如果你已经知道批规模，那就不需要这种灵活性了。您可以在这里给出批规模，特别是对于卷积层，这样可以运用TensorFlow一些优化功能。

```
x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')
```

在TensorLayer中每个神经网络的基础是一个 *InputLayer* 实例。它代表了将要提供(feed)给网络的输入数据。 值得注意的是 `InputLayer` 并不依赖任何特定的数据。

```
network = tl.layers.InputLayer(x, name='input_layer')
```

在添加第一层隐藏层之前，我们要对输入数据应用20%的退出率(dropout)。 这里我们是通过一个 *DropoutLayer* 的实例来实现的。

```
network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
```

请注意构造函数的第一个参数是输入层，第二个参数是激活值的保持概率(keeping probability for the activation value) 现在我们要继续构造第一个800个单位的全连接的隐藏层。 尤其是当要堆叠一个 *DenseLayer* 时，要特别注意。

```
network = tl.layers.DenseLayer(network, n_units=800, act = tf.nn.relu, name='relu1')
```

同样，构造函数的第一个参数意味着这我们正在 `network` 之上堆叠 `network` 。 `n_units` 简明得给出了全连接层的单位数。 `act` 指定了一个激活函数，这里的激活函数有一部分已经被定义在了 `tensorflow.nn` 和 *tensorlayer.activation* 中。 我们在这里选择了整流器(rectifier)，我们将得到ReLUs。 我们现在来添加50%的退出率，以及另外800个单位的稠密层(dense layer)，和50%的退出率：

```
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2')
network = tl.layers.DenseLayer(network, n_units=800, act = tf.nn.relu, name='relu2')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
```

最后，我们加入 `n_units` 等于分类个数的全连接的输出层。注意， `cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(y, y_))` 在内部实现 Softmax，以提高计算效率，因此最后一层的输出为 identity ，更多细节请参考 `tl.cost.cross_entropy()` 。

```
network = tl.layers.DenseLayer(network,
                              n_units=10,
                              act = tl.activation.identity,
                              name='output_layer')
```

如上所述，因为每一层都被链接到了它的输入层，所以我们只需要在TensorLayer中将输出层接入一个网络：

```
y = network.outputs
y_op = tf.argmax(tf.nn.softmax(y), 1)
cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(y, y_))
```

在这里，`network.outputs` 是网络的10个特征的输出(按照一个热门的格式)。 `y_op` 是代表类索引的整数输出， `cost` 是目标和预测标签的交叉熵。

降噪自编码器(Denoising Autoencoder)

自编码器是一种无监督学习（Unsupervisered Learning）模型，可从数据中学习出更好的表达， 目前已经用于逐层贪婪的预训练（Greedy layer-wise pre-train） 。 有关Vanilla自编码器，请参考教程 Deeplearning Tutorial。

脚本 `main_test_denoise_AE()` 实现了有50%的腐蚀率(corrosion rate)的去噪自编码器。 这个自编码器可以按如下方式定义，这里的 `DenseLayer` 代表了一个自编码器：

```
network = tl.layers.InputLayer(x, name='input_layer')
network = tl.layers.DropoutLayer(network, keep=0.5, name='denoising1')
network = tl.layers.DenseLayer(network, n_units=200, act=tf.nn.sigmoid, name='sigmoid1
↪')
recon_layer1 = tl.layers.ReconLayer(network,
                                    x_recon=x,
                                    n_units=784,
                                    act=tf.nn.sigmoid,
                                    name='recon_layer1')
```

训练 `DenseLayer` ，只需要运行 `ReconLayer.Pretrain()` 即可。 如果要使用去噪自编码器，腐蚀层(corrosion layer)(`DropoutLayer`)的名字需要按后面说的指定。 如果要保存特征图像，设置 `save` 为 True 。 根据不同的架构和应用这里可以设置许多预训练的度量(metric)

对于 sigmoid型激活函数来说，自编码器可以用KL散度来实现。 而对于整流器(Rectifier)来说，对激活函数输出的L1正则化能使得输出变得稀疏。 所以 `ReconLayer` 默认只对整流激活函数(ReLU)提供KLD和交叉熵这两种损失度量，而对sigmoid型激活函数提供均方误差以及激活输出的L1范数这两种损失度量。 我们建议您修改 `ReconLayer` 来实现自己的预训练度量。

```
recon_layer1.pretrain(sess,
                      x=x,
                      X_train=X_train,
                      X_val=X_val,
                      denoise_name='denoising1',
                      n_epoch=200,
                      batch_size=128,
                      print_freq=10,
                      save=True,
                      save_name='w1pre_')
```

此外，脚本 `main_test_stacked_denoise_AE()` 展示了如何将多个自编码器堆叠到一个网络，然后进行微调。

卷积神经网络(Convolutional Neural Network)

最后，`main_test_cnn_layer()` 脚本创建了两个CNN层和最大汇流阶段(max pooling stages)，一个全连接的隐藏层和一个全连接的输出层。

首先，我们需要添加一个 *Conv2dLayer* ， 它顶部有32个5x5的过滤器，紧接着在两个2个向量的同尺寸的最大汇流。[###]

注，`tutorial_mnist.py` 中介绍了针对初学者的简化版的 CNN API。 .. code-block:: python

network = tl.layers.InputLayer(x, name='input_layer') network = tl.layers.Conv2dLayer(network,

act = tf.nn.relu, shape = [5, 5, 1, 32], # 32 features for each 5x5 patch strides=[1, 1, 1, 1], padding='SAME', name ='cnn_layer1') # output: (?, 28, 28, 32)

network = tl.layers.PoolLayer(network, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', pool = tf.nn.max_pool, name ='pool_layer1',) # output: (?, 14, 14, 32)

network = tl.layers.Conv2dLayer(network, act = tf.nn.relu, shape = [5, 5, 32, 64], # 64 features for each 5x5 patch strides=[1, 1, 1, 1], padding='SAME', name ='cnn_layer2') # output: (?, 14, 14, 64)

network = tl.layers.PoolLayer(network, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', pool = tf.nn.max_pool, name ='pool_layer2',) # output: (?, 7, 7, 64)

network = tl.layers.FlattenLayer(network, name='flatten_layer') # output: (?, 3136)

network = tl.layers.DropoutLayer(network, keep=0.5, name='drop1') # output: (?, 3136)

network = tl.layers.DenseLayer(network, n_units=256, act = tf.nn.relu, name='relu1') # output: (?, 256)

network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2') # output: (?, 256)

network = tl.layers.DenseLayer(network, n_units=10, act = tl.identity, name='output_layer') # output: (?, 10)

注解: 对于专家们来说， `Conv2dLayer` 将使用 `tensorflow.nn.conv2d` ,TensorFlow默认的卷积方式来创建一个卷积层。

训练模型

在 `tutorial_mnist.py` 脚本的其余部分，在MNIST数据上对于只使用交叉熵的循环训练进行了设置并且运行[###]。

数据集迭代

一个在给定的项目数的最小批规模下的输入特征及其对应的标签的两个Numpy数列依次同步的迭代函数[###]。 更多有关迭代函数的说明，可以在 `tensorlayer.iterate` 中找到。

```
tl.iterate.minibatches(inputs, targets, batchsize, shuffle=False)
```

损失和更新公式

我们继续创建一个在训练中被最小化的损失表达式:

```
y = network.outputs
y_op = tf.argmax(tf.nn.softmax(y), 1)
cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(y, y_))
```

举 main_test_layers() 这个例子来说，更多的成本或者正则化方法可以被应用在这里。 如果要在权重矩阵中应用最大模(max-norm)方法，你可以添加下列代码：

```
cost = cost + tl.cost.maxnorm_regularizer(1.0)(network.all_params[0]) +
              tl.cost.maxnorm_regularizer(1.0)(network.all_params[2])
```

根据要解决的问题，您会需要使用不同的损失函数，更多有关损失函数的说明请见： *tensorlayer.cost*

有了模型和定义的损失函数之后，我们就可以创建用于训练网络的更新公式。 接下去，我们将使用TensorFlow的优化器如下：

```
train_params = network.all_params
train_op = tf.train.AdamOptimizer(learning_rate, beta1=0.9, beta2=0.999,
    epsilon=1e-08, use_locking=False).minimize(cost, var_list=train_params)
```

为了训练网络，我们需要提供数据和保持概率给 feed_dict。

```
feed_dict = {x: X_train_a, y_: y_train_a}
feed_dict.update( network.all_drop )
sess.run(train_op, feed_dict=feed_dict)
```

同时为了进行验证和测试，我们这里用了略有不同的方法。 所有的Dropout，退连(DropConnect)，腐蚀层(Corrosion Layers)都将被禁用。 tl.utils.dict_to_one 将会设置所有 network.all_drop 值为1。

```
dp_dict = tl.utils.dict_to_one( network.all_drop )
feed_dict = {x: X_test_a, y_: y_test_a}
feed_dict.update(dp_dict)
err, ac = sess.run([cost, acc], feed_dict=feed_dict)
```

最后，作为一个额外的监测量，我们需要创建一个分类准确度的公式：

```
correct_prediction = tf.equal(tf.argmax(y, 1), y_)
acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

下一步?

在 tutorial_cifar10_tfrecord.py 中我们还有更高级的图像分类的例子。 请阅读代码及注释，用以明白如何来生成更多的训练数据以及什么是局部响应正则化。 在这之后，您可以尝试着去实现 残差网络(Residual Network)。 小提示：您可能会用到*Layer.outputs*。

1.2.5 运行乒乓球例子

在本教程的第二部分，我们将运行一个深度强化学习的例子，它在Karpathy的两篇博客 Deep Reinforcement Learning:Pong from Pixels 有介绍。

```
python tutorial_atari_pong.py
```

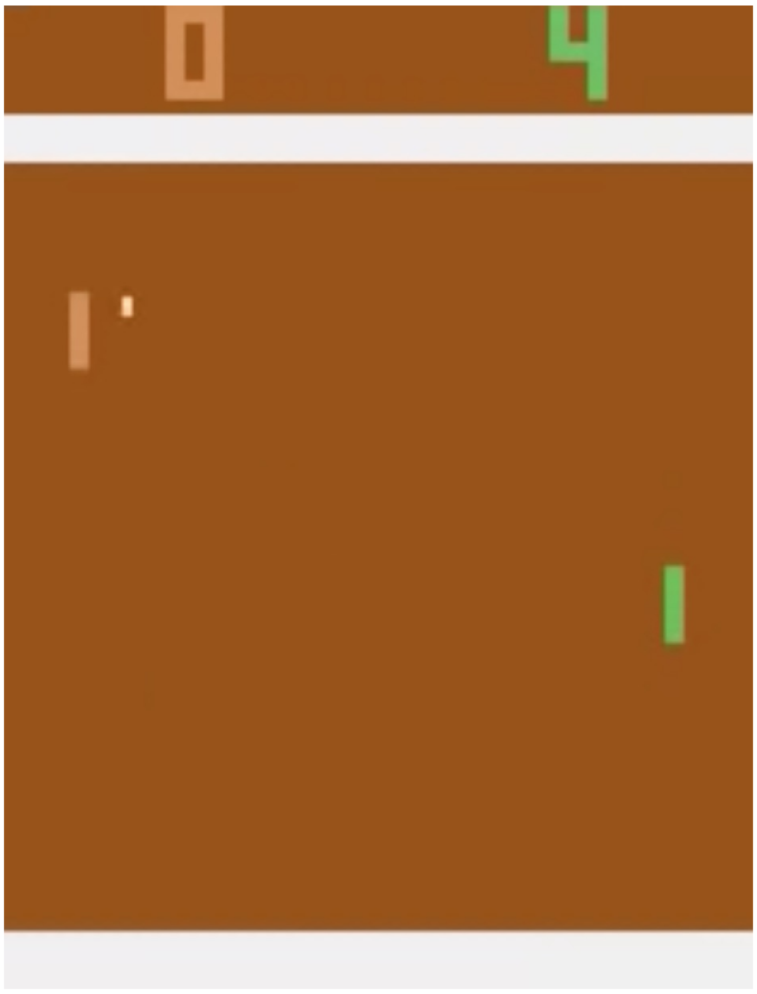
在运行教程代码之前 你需要安装 OpenAI gym environment ,它是强化学习的一个标杆。 如果一切设置正确，您将得到一个类似以下的输出：

```
[2016-07-12 09:31:59,760] Making new env: Pong-v0
  tensorlayer:Instantiate InputLayer input_layer (?, 6400)
  tensorlayer:Instantiate DenseLayer relu1: 200, relu
  tensorlayer:Instantiate DenseLayer output_layer: 3, identity
  param 0: (6400, 200) (mean: -0.000009, median: -0.000018 std: 0.017393)
  param 1: (200,) (mean: 0.000000, median: 0.000000 std: 0.000000)
  param 2: (200, 3) (mean: 0.002239, median: 0.003122 std: 0.096611)
  param 3: (3,) (mean: 0.000000, median: 0.000000 std: 0.000000)
  num of params: 1280803
  layer 0: Tensor("Relu:0", shape=(?, 200), dtype=float32)
  layer 1: Tensor("add_1:0", shape=(?, 3), dtype=float32)
episode 0: game 0 took 0.17381s, reward: -1.000000
episode 0: game 1 took 0.12629s, reward: 1.000000  !!!!!!!!
episode 0: game 2 took 0.17082s, reward: -1.000000
episode 0: game 3 took 0.08944s, reward: -1.000000
episode 0: game 4 took 0.09446s, reward: -1.000000
episode 0: game 5 took 0.09440s, reward: -1.000000
episode 0: game 6 took 0.32798s, reward: -1.000000
episode 0: game 7 took 0.74437s, reward: -1.000000
episode 0: game 8 took 0.43013s, reward: -1.000000
episode 0: game 9 took 0.42496s, reward: -1.000000
episode 0: game 10 took 0.37128s, reward: -1.000000
episode 0: game 11 took 0.08979s, reward: -1.000000
episode 0: game 12 took 0.09138s, reward: -1.000000
episode 0: game 13 took 0.09142s, reward: -1.000000
episode 0: game 14 took 0.09639s, reward: -1.000000
episode 0: game 15 took 0.09852s, reward: -1.000000
episode 0: game 16 took 0.09984s, reward: -1.000000
episode 0: game 17 took 0.09575s, reward: -1.000000
episode 0: game 18 took 0.09416s, reward: -1.000000
episode 0: game 19 took 0.08674s, reward: -1.000000
episode 0: game 20 took 0.09628s, reward: -1.000000
resetting env. episode reward total was -20.000000. running mean: -20.000000
episode 1: game 0 took 0.09910s, reward: -1.000000
episode 1: game 1 took 0.17056s, reward: -1.000000
episode 1: game 2 took 0.09306s, reward: -1.000000
episode 1: game 3 took 0.09556s, reward: -1.000000
episode 1: game 4 took 0.12520s, reward: 1.000000  !!!!!!!!
episode 1: game 5 took 0.17348s, reward: -1.000000
episode 1: game 6 took 0.09415s, reward: -1.000000
```

这个例子让电脑从屏幕输入来学习如何像人类一样打乒乓球。在经过15000个序列的训练之后，计算机就可以赢得20%的比赛。在20000个序列的训练之后，计算机可以赢得35%的比赛，我们可以看到计算机学的越来越快，这是因为它有更多的胜利的数据来进行训练。如果您用30000个序列来训练它，那么它会一直赢。

```
render = False
resume = False
```

如果您想显示游戏过程，那就设置 *render* 为 *True* 。当您再次运行该代码，您可以设置 *resume* 为 *True*,那么代码将加载现有的模型并且会基于它进行训练。



1.2.6 理解强化学习

乒乓球

要理解强化学习，我们要让电脑学习如何从原始的屏幕输入(像素输入)打乒乓球。 在我们开始之前，我们强烈建议您去浏览一个著名的博客叫做 Deep Reinforcement Learning:pong from Pixels , 这是使用python numpy库和OpenAI gym environment=来实现的一个深度强化学习的最简实现。

```
python tutorial_atari_pong.py
```

策略网络(Policy Network)

在深度强化学习中，Policy Network 等同于 深度神经网络。 它是我们的选手(或者说"代理人(agent)"），它的输出告诉我们应该做什么(向上移动或向下移动)： 在Karpathy的代码中，他只定义了2个动作，向上移动和向下移动，并且仅使用单个simgoid输出： 为了使我们的教程更具有普遍性，我们使用3个SOFTMAX输出来定义向上移动，向下移动和停止(什么都不做)3个动作。

```
# observation for training
states_batch_pl = tf.placeholder(tf.float32, shape=[None, D])
```

```
network = tl.layers.InputLayer(states_batch_pl, name='input_layer')
network = tl.layers.DenseLayer(network, n_units=H,
                                act = tf.nn.relu, name='relu1')
network = tl.layers.DenseLayer(network, n_units=3,
                        act = tl.activation.identity, name='output_layer')
probs = network.outputs
sampling_prob = tf.nn.softmax(probs)
```

然后我们的代理人就一直打乒乓球。它计算不同动作的概率，并且之后会从这个均匀的分布中选取样本(动作)。因为动作被1,2和3代表，但是softmax输出应该从0开始，所以我们从-1计算这个标签的价值。

```
prob = sess.run(
    sampling_prob,
    feed_dict={states_batch_pl: x}
)
# action. 1: STOP  2: UP  3: DOWN
action = np.random.choice([1,2,3], p=prob.flatten())
...
ys.append(action - 1)
```

策略逼近(Policy Gradient)

策略梯度下降法是一个end-to-end的算法，它直接学习从状态映射到动作的策略函数。一个近似最优的策略可以通过最大化预期的奖励来直接学习。策略函数的参数(例如，在乒乓球例子终使用的策略网络的参数)在预期奖励的近似值的引导下能够被训练和学习。换句话说，我们可以通过过更新它的参数来逐步调整策略函数，这样它能从给定的状态做出一系列行为来获得更高的奖励。

策略迭代的一个替代算法就是深度Q-learning(DQN)。他是基于Q-learning,学习一个映射状态和动作到一些值的价值函数的算法(叫Q函数)。DQN采用了一个深度神经网络来作为Q函数的逼近来代表Q函数。训练是通过最小化时序差分(temporal-difference)误差来实现。一个名为"再体验(experience replay)"的神经生物学的启发式机制通常和DQN一起被使用来帮助提高非线性函数的逼近的稳定性

您可以阅读以下文档，来得到对强化学习更好的理解：

- Reinforcement Learning: An Introduction. Richard S. Sutton and Andrew G. Barto
- Deep Reinforcement Learning. David Silver, Google DeepMind
- UCL Course on RL

强化深度学习近些年来最成功的应用就是让模型去学习玩Atari的游戏。AlphaGO同时也是使用类似的策略逼近方法来训练他们的策略网络而战胜了世界级的专业围棋选手。

- Atari - Playing Atari with Deep Reinforcement Learning
- Atari - Human-level control through deep reinforcement learning
- AlphaGO - Mastering the game of Go with deep neural networks and tree search

数据集迭代

在强化学习中，我们把每场比赛所产生的所有决策来作为一个序列 (up,up,stop,...,down)。在乒乓球游戏中，比赛是在某一方达到21分后结束的，所以一个序列可能包含几十个决策。然后我们可以设置一个批规模的大小，每一批包含一定数量的序列，基于这个批规模来更新我们的模型。在本教程中，我们把每批规模设置成10个序列。使用RMSProp训练一个具有200个单元的隐藏层的2层策略网络

损失和更新公式

接着我们创建一个在训练中被最小化的损失公式:

```
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)
...
...
sess.run(
    train_op,
    feed_dict={
        states_batch_pl: epx,
        actions_batch_pl: epy,
        discount_rewards_batch_pl: disR
    }
)
```

一batch的损失和一个batch内的策略网络的所有输出，所有的我们做出的动作和相应的被打折的奖励有关 我们首先通过累加被打折的奖励和实际输出和真实动作的交叉熵计算每一个动作的损失。 最后的损失是所有动作的损失的和。

下一步?

上述教程展示了您如何去建立自己的代理人，end-to-end。 虽然它有很合理的品质，但它的默认参数不会给你最好的代理人模型。 这有一些您可以优化的内容。

首先，与传统的MLP模型不同，比起 Playing Atari with Deep Reinforcement Learning 更好的是我们可以使用CNNs来采集屏幕信息

另外这个模型默认参数没有调整，您可以更改学习率，衰退率，或者用不同的方式来初始化您的模型的权重。

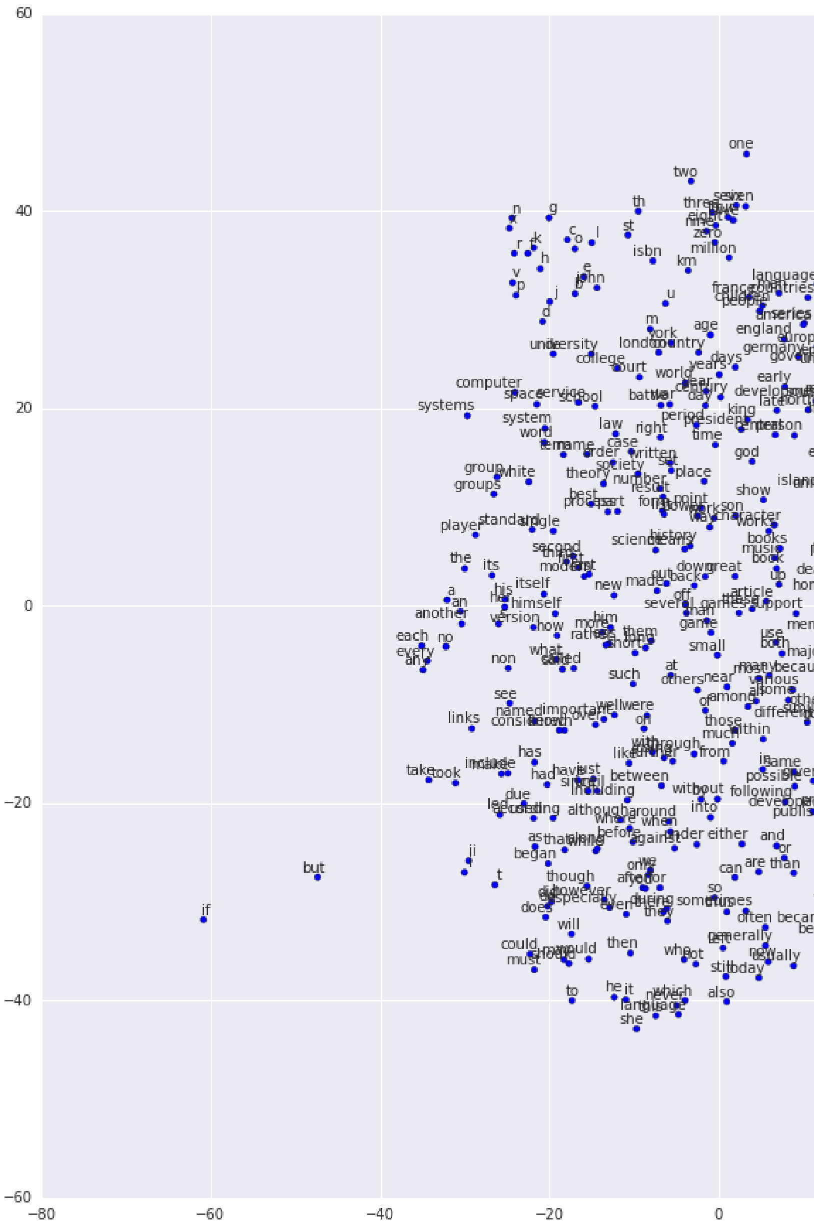
最后，您可以尝试不同任务(游戏)的模型。

1.2.7 运行Word2Vec例子

在教程的这一部分，我们训练一个词嵌套矩阵，每个词可以通过矩阵中唯一的行向量来表示。 在训练结束时，意思类似的单词会有相识的词向量。 在代码的最后，我们通过把单词放到一个平面上来可视化，我们可以看到相似的单词会被聚集在一起。

```
python tutorial_word2vec_basic.py
```

如果一切设置正确，您最后会得到如下的可视化图。



1.2.8 理解词嵌套(word embedding)

词嵌套（嵌入）

我们强烈建议您先阅读Colah的博客 Word Representations [中文翻译] ， 以理解为什么我们要使用一个向量来表示一个单词。更多Word2vec的细节可以在 Word2vec Parameter Learning Explained 中找到。

基本来说，训练一个嵌套矩阵是一个非监督学习的过程。一个单词使用唯一的ID来表示，而这个ID号就是嵌套矩阵的行号（row index），对应的行向量就是用来表示该单词的，使用向量来表示单词可以更好地表达单词的意思。比如，有4个单词的向量， `woman  man = queen - king`，这个例子中可以看到，嵌套矩阵中有一个纬度是用来表示性别的。

定义一个Word2vec词嵌套矩阵如下。

```python
# train_inputs is a row vector, a input is an integer id of single word.
# train_labels is a column vector, a label is an integer id of single word.
# valid_dataset is a column vector, a valid set is an integer id of single word.
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
valid_dataset = tf.constant(valid_examples, dtype=tf.int32)

# Look up embeddings for inputs.
emb_net = tl.layers.Word2vecEmbeddingInputlayer(
        inputs = train_inputs,
        train_labels = train_labels,
        vocabulary_size = vocabulary_size,
        embedding_size = embedding_size,
        num_sampled = num_sampled,
        nce_loss_args = {},
        E_init = tf.random_uniform_initializer(minval=-1.0, maxval=1.0),
        E_init_args = {},
        nce_W_init = tf.truncated_normal_initializer(
                          stddev=float(1.0/np.sqrt(embedding_size))),
        nce_W_init_args = {},
        nce_b_init = tf.constant_initializer(value=0.0),
        nce_b_init_args = {},
        name ='word2vec_layer',
    )
```

数据迭代和损失函数

Word2vec使用负采样（Negative sampling）和Skip-gram模型进行训练。噪音对比估计损失（NCE）会帮助减少损失函数的计算量，加快训练速度。Skip-Gram 将文本（context）和目标（target）反转，尝试从目标单词预测目标文本单词。我们使用 `tl.nlp.generate_skip_gram_batch` 函数来生成训练数据，如下：

```python
# NCE损失函数由 Word2vecEmbeddingInputlayer 提供
cost = emb_net.nce_cost
train_params = emb_net.all_params

train_op = tf.train.AdagradOptimizer(learning_rate, initial_accumulator_value=0.1,
          use_locking=False).minimize(cost, var_list=train_params)

data_index = 0
while (step < num_steps):
  batch_inputs, batch_labels, data_index = tl.nlp.generate_skip_gram_batch(
                data=data, batch_size=batch_size, num_skips=num_skips,
                skip_window=skip_window, data_index=data_index)
  feed_dict = {train_inputs : batch_inputs, train_labels : batch_labels}
  _, loss_val = sess.run([train_op, cost], feed_dict=feed_dict)
```

加载已训练好的的词嵌套矩阵

在训练嵌套矩阵的最后，我们保存矩阵及其词汇表、单词转ID字典、ID转单词字典。然后，当下次做实际应用时，可以想下面的代码中那样加载这个已经训练好的矩阵和字典，参考 `tutorial_generate_text.py`。

```
vocabulary_size = 50000
embedding_size = 128
model_file_name = "model_word2vec_50k_128"
batch_size = None

print("Load existing embedding matrix and dictionaries")
all_var = tl.files.load_npy_to_any(name=model_file_name+'.npy')
data = all_var['data']; count = all_var['count']
dictionary = all_var['dictionary']
reverse_dictionary = all_var['reverse_dictionary']

tl.nlp.save_vocab(count, name='vocab_'+model_file_name+'.txt')

del all_var, data, count

load_params = tl.files.load_npz(name=model_file_name+'.npz')

x = tf.placeholder(tf.int32, shape=[batch_size])
y_ = tf.placeholder(tf.int32, shape=[batch_size, 1])

emb_net = tl.layers.EmbeddingInputlayer(
                inputs = x,
                vocabulary_size = vocabulary_size,
                embedding_size = embedding_size,
                name ='embedding_layer')

tl.layers.initialize_global_variables(sess)

tl.files.assign_params(sess, [load_params[0]], emb_net)
```

1.2.9 运行PTB例子

Penn TreeBank（PTB）数据集被用在很多语言建模（Language Modeling）的论文中，包括"Empirical Evaluation and Combination of Advanced Language Modeling Techniques"和 “Recurrent Neural Network Regularization”。该数据集的训练集有929k个单词，验证集有73K个单词，测试集有82k个单词。 在它的词汇表刚好有10k个单词。

PTB例子是为了展示如何用递归神经网络（Recurrent Neural Network）来进行语言建模的。

给一句话 "I am from Imperial College London", 这个模型可以从中学习出如何从“from Imperial College”来预测出“Imperial College London”。也就是说，它根据之前输入的单词序列来预测出下一步输出的单词序列，在刚才的例子中 `num_steps (序列长度, sequence length)` 为 3。

```
python tutorial_ptb_lstm.py
```

该脚本提供三种设置(小，中，大)，越大的模型有越好的建模性能，您可以修改下面的代码片段来选择不同的模型设置。

```
flags.DEFINE_string(
    "model", "small",
    "A type of model. Possible options are: small, medium, large.")
```

如果您选择小设置，您将会看到：

```
Epoch: 1 Learning rate: 1.000
0.004 perplexity: 5220.213 speed: 7635 wps
```

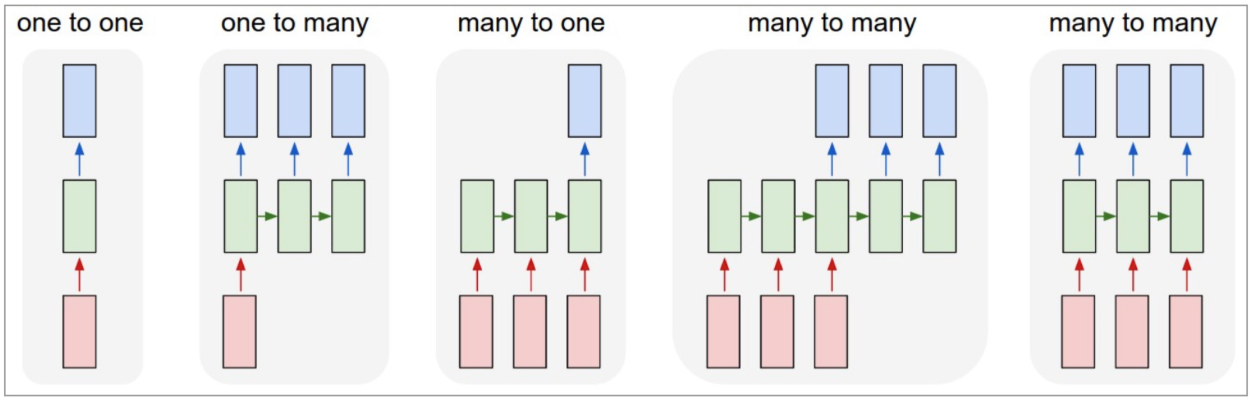
```
0.104 perplexity: 828.871 speed: 8469 wps
0.204 perplexity: 614.071 speed: 8839 wps
0.304 perplexity: 495.485 speed: 8889 wps
0.404 perplexity: 427.381 speed: 8940 wps
0.504 perplexity: 383.063 speed: 8920 wps
0.604 perplexity: 345.135 speed: 8920 wps
0.703 perplexity: 319.263 speed: 8949 wps
0.803 perplexity: 298.774 speed: 8975 wps
0.903 perplexity: 279.817 speed: 8986 wps
Epoch: 1 Train Perplexity: 265.558
Epoch: 1 Valid Perplexity: 178.436
...
Epoch: 13 Learning rate: 0.004
0.004 perplexity: 56.122 speed: 8594 wps
0.104 perplexity: 40.793 speed: 9186 wps
0.204 perplexity: 44.527 speed: 9117 wps
0.304 perplexity: 42.668 speed: 9214 wps
0.404 perplexity: 41.943 speed: 9269 wps
0.504 perplexity: 41.286 speed: 9271 wps
0.604 perplexity: 39.989 speed: 9244 wps
0.703 perplexity: 39.403 speed: 9236 wps
0.803 perplexity: 38.742 speed: 9229 wps
0.903 perplexity: 37.430 speed: 9240 wps
Epoch: 13 Train Perplexity: 36.643
Epoch: 13 Valid Perplexity: 121.475
Test Perplexity: 116.716
```

PTB例子证明了递归神经网络能够实现语言建模，但是这个例子并没有做什么实际的事情。 在做具体应用之前，您应该浏览这个例子的代码和下一章 "理解 LSTM" 来学好递归神经网络的基础。 之后，您将学习如何用递归神经网络来生成文本，如何实现语言翻译和问题应答系统。

1.2.10 理解LSTM

递归神经网络 (Recurrent Neural Network)

我们认为Andrey Karpathy的博客 Understand Recurrent Neural Network 是了解递归神经网络最好的材料。读完这个博客后，Colah的博客 Understand LSTM Network 能帮助你了解LSTM。 我们在这里不介绍更多关于递归神经网络的内容，所以在你继续下面的内容之前，请先阅读我们建议阅读的博客。



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

图片由Andrey Karpathy提供

同步输入与输出序列 (Synced sequence input and output)

PTB例子中的模型是一个典型的同步输入与输出，Karpathy 把它描述为 "(5) 同步序列输入与输出(例如视频分类中我们希望对每一帧进行标记)。"

模型的构建如下，第一层是词嵌套层（嵌入），把每一个单词转换成对应的词向量，在该例子中没有使用预先训练好的 嵌套矩阵。第二，堆叠两层LSTM，使用Dropout来实现规则化，防止overfitting。 最后，使用全连接层输出一序列的softmax输出。

第一层LSTM的输出形状是 [batch_size, num_steps, hidden_size]，这是为了让下一层LSTM可以堆叠在其上面。 第二层LSTM的输出形状是 [batch_size*num_steps, hidden_size]，这是为了让输出层（全连接层 Dense）可以堆叠在其上面。 然后计算每个样本的softmax输出，样本总数为 n_examples = batch_size*num_steps。

若想要更进一步理解该PTB教程，您也可以阅读 TensorFlow 官方的PTB教程 ，中文翻译请见极客学院。

```
network = tl.layers.EmbeddingInputlayer(
            inputs = x,
            vocabulary_size = vocab_size,
            embedding_size = hidden_size,
            E_init = tf.random_uniform_initializer(-init_scale, init_scale),
            name ='embedding_layer')
if is_training:
    network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop1')
network = tl.layers.RNNLayer(network,
            cell_fn=tf.nn.rnn_cell.BasicLSTMCell,
            cell_init_args={'forget_bias': 0.0},
            n_hidden=hidden_size,
            initializer=tf.random_uniform_initializer(-init_scale, init_scale),
            n_steps=num_steps,
            return_last=False,
            name='basic_lstm_layer1')
lstm1 = network
if is_training:
    network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop2')
network = tl.layers.RNNLayer(network,
            cell_fn=tf.nn.rnn_cell.BasicLSTMCell,
            cell_init_args={'forget_bias': 0.0},
            n_hidden=hidden_size,
            initializer=tf.random_uniform_initializer(-init_scale, init_scale),
            n_steps=num_steps,
            return_last=False,
            return_seq_2d=True,
            name='basic_lstm_layer2')
lstm2 = network
if is_training:
    network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop3')
network = tl.layers.DenseLayer(network,
            n_units=vocab_size,
            W_init=tf.random_uniform_initializer(-init_scale, init_scale),
            b_init=tf.random_uniform_initializer(-init_scale, init_scale),
            act = tl.activation.identity, name='output_layer')
```

数据迭代

batch_size 数值可以被视为并行计算的数量。 如下面的例子所示，第一个 batch 使用 0 到 9 来学习序列信息。 第二个 batch 使用 10 到 19 来学习序列。 所以它忽略了 9 到 10 之间的信息。 只当我们 bath_size 设为 1，它才使用 0 到 20 之间所有的序列信息来学习。

这里的 batch_size 的意思与 MNIST 例子略有不同。 在 MNIST 例子，batch_size 是每次迭代中我们使用的样本数量， 而在 PTB 的例子中，batch_size 是为加快训练速度的并行进程数。

虽然当 batch_size > 1 时有些信息将会被忽略， 但是如果你的数据是足够长的（一个语料库通常有几十亿个字），被忽略的信息不会影响最终的结果。

在PTB教程中，我们设置了 batch_size = 20，所以，我们将整个数据集拆分成 20 段（segment）。 在每一轮（epoch）的开始时，我们有 20 个初始化的 LSTM 状态（State），然后分别对 20 段数据进行迭代学习。

训练数据迭代的例子如下：

```
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, '\n',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]]
```

注解: 这个例子可以当作词嵌套矩阵的预训练。

损失和更新公式

损失函数是一系列输出cross entropy的均值。

```
# 更多细节请见 tensorlayer.cost.cross_entropy_seq()
def loss_fn(outputs, targets, batch_size, num_steps):
    # Returns the cost function of Cross-entropy of two sequences, implement
    # softmax internally.
    # outputs : 2D tensor [batch_size*num_steps, n_units of output layer]
    # targets : 2D tensor [batch_size, num_steps], need to be reshaped.
    # n_examples = batch_size * num_steps
    # so
    # cost is the averaged cost of each mini-batch (concurrent process).
    loss = tf.nn.seq2seq.sequence_loss_by_example(
        [outputs],
        [tf.reshape(targets, [-1])],
        [tf.ones([batch_size * num_steps])])
    cost = tf.reduce_sum(loss) / batch_size
    return cost

# Cost for Training
cost = loss_fn(network.outputs, targets, batch_size, num_steps)
```

在训练时，该例子在若干个epoch之后（由 `max_epoch` 定义），才开始按比例下降学习率（learning rate），新学习率是前一个epoch的学习率乘以一个下降率（由 `lr_decay` 定义）。此外，截断反向传播（truncated backpropagation）截断了

为使学习过程易于处理，通常的做法是将反向传播的梯度在（按时间）展开的步骤上照一个固定长度(`num_steps`)截断。通过在一次迭代中的每个时刻上提供长度为 `num_steps` 的输入和每次迭代完成之后反向传导，这会很容易实现。

```
# 截断反响传播 Truncated Backpropagation for training
with tf.variable_scope('learning_rate'):
    lr = tf.Variable(0.0, trainable=False)
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars),
                                  max_grad_norm)
optimizer = tf.train.GradientDescentOptimizer(lr)
train_op = optimizer.apply_gradients(zip(grads, tvars))
```

如果当前epoch值大于 `max_epoch`，则把当前学习率乘以 `lr_decay` 来降低学习率。

```
new_lr_decay = lr_decay ** max(i - max_epoch, 0.0)
sess.run(tf.assign(lr, learning_rate * new_lr_decay))
```

在每一个epoch的开始之前，LSTM的状态要被重置为零状态；在每一个迭代之后，LSTM状态都会被改变，所以要把最新的LSTM状态 作为下一个迭代的初始化状态。

```
# 在每一个epoch之前，把所有LSTM状态设为零状态
state1 = tl.layers.initialize_rnn_state(lstm1.initial_state)
state2 = tl.layers.initialize_rnn_state(lstm2.initial_state)
for step, (x, y) in enumerate(tl.iterate.ptb_iterator(train_data,
                                            batch_size, num_steps)):
```

```
    feed_dict = {input_data: x, targets: y,
                lstm1.initial_state: state1,
                lstm2.initial_state: state2,
                }
    # 启用dropout
    feed_dict.update( network.all_drop )
    # 把新的状态作为下一个迭代的初始状态
    _cost, state1, state2, _ = sess.run([cost,
                                    lstm1.final_state,
                                    lstm2.final_state,
                                    train_op],
                                    feed_dict=feed_dict
                                    )
    costs += _cost; iters += num_steps
```

预测

在训练完模型之后，当我们预测下一个输出时，我们不需要考虑序列长度了，因此 batch_size 和 num_steps 都设为 1 。然后，我们可以一步一步地输出下一个单词，而不是通过一序列的单词来输出一序列的单词。

```
input_data_test = tf.placeholder(tf.int32, [1, 1])
targets_test = tf.placeholder(tf.int32, [1, 1])
...
network_test, lstm1_test, lstm2_test = inference(input_data_test,
                      is_training=False, num_steps=1, reuse=True)
...
cost_test = loss_fn(network_test.outputs, targets_test, 1, 1)
...
print("Evaluation")
# 测试
# go through the test set step by step, it will take a while.
start_time = time.time()
costs = 0.0; iters = 0
# 与训练时一样，设置所有LSTM状态为零状态
state1 = tl.layers.initialize_rnn_state(lstm1_test.initial_state)
state2 = tl.layers.initialize_rnn_state(lstm2_test.initial_state)
for step, (x, y) in enumerate(tl.iterate.ptb_iterator(test_data,
                                        batch_size=1, num_steps=1)):
    feed_dict = {input_data_test: x, targets_test: y,
                lstm1_test.initial_state: state1,
                lstm2_test.initial_state: state2,
                }
    _cost, state1, state2 = sess.run([cost_test,
                                    lstm1_test.final_state,
                                    lstm2_test.final_state],
                                    feed_dict=feed_dict
                                    )
    costs += _cost; iters += 1
test_perplexity = np.exp(costs / iters)
print("Test Perplexity: %.3f took %.2fs" % (test_perplexity, time.time() - start_
↪time))
```

下一步?

您已经明白了同步序列输入和序列输出（Synced sequence input and output）。 现在让我们思考下序列输入单一输出的情况（Sequence input and one output）， LSTM 也可以学会通过给定一序列输入如 "我来自北京，我会说.." 来输出 一个单词 "中文"。

请仔细阅读并理解 `tutorial_generate_text.py` 的代码，它讲了如何加载一个已经训练好的词嵌套矩阵， 以及如何给定机器一个文档，让它来学习文字自动生成。

Karpathy的博客： "(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). "

1.2.11 运行机器翻译例子

```
python tutorial_translate.py
```

该脚本将训练一个神经网络来把英文翻译成法文。 如果一切正常，您将看到： - 下载WMT英文-法文翻译数据库，包括训练集和测试集。 - 通过训练集创建英文和法文的词汇表。 - 把训练集和测试集的单词转换成数字ID表示。

```
Prepare raw data
Load or Download WMT English-to-French translation > wmt
Training data : wmt/giga-fren.release2
Testing data : wmt/newstest2013

Create vocabularies
Vocabulary of French : wmt/vocab40000.fr
Vocabulary of English : wmt/vocab40000.en
Creating vocabulary wmt/vocab40000.fr from data wmt/giga-fren.release2.fr
  processing line 100000
  processing line 200000
  processing line 300000
  processing line 400000
  processing line 500000
  processing line 600000
  processing line 700000
  processing line 800000
  processing line 900000
  processing line 1000000
  processing line 1100000
  processing line 1200000
  ...
  processing line 22500000
Creating vocabulary wmt/vocab40000.en from data wmt/giga-fren.release2.en
  processing line 100000
  ...
  processing line 22500000

...
```

首先，我们从WMT'15网站上下载英语-法语翻译数据。训练数据和测试数据如下。 训练数据用于训练模型，测试数据用于评估该模型。

```
wmt/training-giga-fren.tar  <-- 英文－法文训练集 (2.6GB)
                                giga-fren.release2.* 从该文件解压出来
wmt/dev-v2.tgz              <-- 多种语言的测试集 (21.4MB)
                                newstest2013.* 从该文件解压出来
```

```
wmt/giga-fren.release2.fr   <-- 法文训练集 (4.57GB)
wmt/giga-fren.release2.en   <-- 英文训练集 (3.79GB)

wmt/newstest2013.fr         <-- 法文测试集 (393KB)
wmt/newstest2013.en         <-- 英文测试集 (333KB)
```

所有 `giga-fren.release2.*` 是训练数据， `giga-fren.release2.fr` 内容如下:

```
Il a transformé notre vie | Il a transformé la société | Son fonctionnement | La␣
↪technologie, moteur du changement Accueil | Concepts | Enseignants | Recherche |␣
↪Aperçu | Collaborateurs | Web HHCC | Ressources | Commentaires Musée virtuel du␣
↪Canada
Plan du site
Rétroaction
Crédits
English
Qu’est-ce que la lumière?
La découverte du spectre de la lumière blanche Des codes dans la lumière Le spectre␣
↪électromagnétique Les spectres d’émission Les spectres d’absorption Les années-
↪lumière La pollution lumineuse
Le ciel des premiers habitants La vision contemporaine de l'Univers L’astronomie pour␣
↪tous
Bande dessinée
Liens
Glossaire
Observatoires
...
```

`giga-fren.release2.en` 内容如下，我们可以看到单词或者句子用 | 或 `\n` 来分隔。

```
Changing Lives | Changing Society | How It Works | Technology Drives Change Home |␣
↪Concepts | Teachers | Search | Overview | Credits | HHCC Web | Reference | Feedback␣
↪Virtual Museum of Canada Home Page
Site map
Feedback
Credits
Français
What is light ?
The white light spectrum Codes in the light The electromagnetic spectrum Emission␣
↪spectra Absorption spectra Light-years Light pollution
The sky of the first inhabitants A contemporary vison of the Universe Astronomy for␣
↪everyone
Cartoon
Links
Glossary
Observatories
```

测试数据 `newstest2013.en` 和 `newstest2013.fr` 如下所示:

```
newstest2013.en :
A Republican strategy to counter the re-election of Obama
Republican leaders justified their policy by the need to combat electoral fraud.
However, the Brennan Centre considers this a myth, stating that electoral fraud is␣
↪rarer in the United States than the number of people killed by lightning.

newstest2013.fr :
Une stratégie républicaine pour contrer la réélection d'Obama
```

```
Les dirigeants républicains justifièrent leur politique par la nécessité de lutter␣
↪contre la fraude électorale.
Or, le Centre Brennan considère cette dernière comme un mythe, affirmant que la␣
↪fraude électorale est plus rare aux États-Unis que le nombre de personnes tuées par␣
↪la foudre.
```

下载完数据之后，开始创建词汇表文件。从训练数据 `giga-fren.release2.fr` 和 `giga-fren.release2.en``创建 ``vocab40000.fr` 和 `vocab40000.en` 这个过程需要较长一段时间，数字 `40000` 代表了词汇库的大小。

`vocab40000.fr` (381KB) 按下列所示地按每行一个单词的方式存储（one-item-per-line）。

```
_PAD
_GO
_EOS
_UNK
de
,
.
'
la
et
des
les
à
le
du
l
en
)
d
0
(
00
pour
dans
un
que
une
sur
au
0000
a
par
```

`vocab40000.en` (344KB) 也是如此。

```
_PAD
_GO
_EOS
_UNK
the
.
,
of
and
to
in
```

```
a
)
(
0
for
00
that
is
on
The
0000
be
by
with
or
:
as
"
000
are
;
```

接着我们开始创建英文和法文的数字化（ID）训练集和测试集。这也要较长一段时间。

```
Tokenize data
Tokenizing data in wmt/giga-fren.release2.fr  <-- Training data of French
  tokenizing line 100000
  tokenizing line 200000
  tokenizing line 300000
  tokenizing line 400000
  ...
  tokenizing line 22500000
Tokenizing data in wmt/giga-fren.release2.en  <-- Training data of English
  tokenizing line 100000
  tokenizing line 200000
  tokenizing line 300000
  tokenizing line 400000
  ...
  tokenizing line 22500000
Tokenizing data in wmt/newstest2013.fr        <-- Testing data of French
Tokenizing data in wmt/newstest2013.en        <-- Testing data of English
```

最后，我们所有的文件如下所示：

```
wmt/training-giga-fren.tar  <-- 英文－法文训练集 (2.6GB)
                                giga-fren.release2.* 从该文件解压出来
wmt/dev-v2.tgz              <-- 多种语言的测试集 (21.4MB)
                                newstest2013.* 从该文件解压出来

wmt/giga-fren.release2.fr   <-- 法文训练集 (4.57GB)
wmt/giga-fren.release2.en   <-- 英文训练集 (3.79GB)

wmt/newstest2013.fr         <-- 法文测试集 (393KB)
wmt/newstest2013.en         <-- 英文测试集 (333KB)

wmt/vocab40000.fr           <-- 法文词汇表 (381KB)
wmt/vocab40000.en           <-- 英文词汇表 (344KB)
```

```
wmt/giga-fren.release2.ids40000.fr   <-- 数字化法文训练集 (2.81GB)
wmt/giga-fren.release2.ids40000.en   <-- 数字化英文训练集 (2.38GB)

wmt/newstest2013.ids40000.fr         <-- 数字化法文训练集 (268KB)
wmt/newstest2013.ids40000.en         <-- 数字化英文测试集 (232KB)
```

现在，把数字化的数据读入buckets中，并计算不同buckets中数据样本的个数。

```
Read development (test) data into buckets
dev data: (5, 10) [[13388, 4, 949], [23113, 8, 910, 2]]
en word_ids: [13388, 4, 949]
en context: [b'Preventing', b'the', b'disease']
fr word_ids: [23113, 8, 910, 2]
fr context: [b'Pr\xc3\xa9venir', b'la', b'maladie', b'_EOS']

Read training data into buckets (limit: 0)
  reading data line 100000
  reading data line 200000
  reading data line 300000
  reading data line 400000
  reading data line 500000
  reading data line 600000
  reading data line 700000
  reading data line 800000
  ...
  reading data line 22400000
  reading data line 22500000
train_bucket_sizes: [239121, 1344322, 5239557, 10445326]
train_total_size: 17268326.0
train_buckets_scale: [0.013847375825543252, 0.09169638099257565, 0.3951164693091849,␣
↪1.0]
train data: (5, 10) [[1368, 3344], [1089, 14, 261, 2]]
en word_ids: [1368, 3344]
en context: [b'Site', b'map']
fr word_ids: [1089, 14, 261, 2]
fr context: [b'Plan', b'du', b'site', b'_EOS']

the num of training data in each buckets: [239121, 1344322, 5239557, 10445326]
the num of training data: 17268326
train_buckets_scale: [0.013847375825543252, 0.09169638099257565, 0.3951164693091849,␣
↪1.0]
```

最后开始训练模型，当 `steps_per_checkpoint = 10` 时，您将看到：

`steps_per_checkpoint = 10`

```
Create Embedding Attention Seq2seq Model

global step 10 learning rate 0.5000 step-time 22.26 perplexity 12761.50
  eval: bucket 0 perplexity 5887.75
  eval: bucket 1 perplexity 3891.96
  eval: bucket 2 perplexity 3748.77
  eval: bucket 3 perplexity 4940.10
global step 20 learning rate 0.5000 step-time 20.38 perplexity 28761.36
  eval: bucket 0 perplexity 10137.01
  eval: bucket 1 perplexity 12809.90
  eval: bucket 2 perplexity 15758.65
  eval: bucket 3 perplexity 26760.93
```

```
global step 30 learning rate 0.5000 step-time 20.64 perplexity 6372.95
  eval: bucket 0 perplexity 1789.80
  eval: bucket 1 perplexity 1690.00
  eval: bucket 2 perplexity 2190.18
  eval: bucket 3 perplexity 3808.12
global step 40 learning rate 0.5000 step-time 16.10 perplexity 3418.93
  eval: bucket 0 perplexity 4778.76
  eval: bucket 1 perplexity 3698.90
  eval: bucket 2 perplexity 3902.37
  eval: bucket 3 perplexity 22612.44
global step 50 learning rate 0.5000 step-time 14.84 perplexity 1811.02
  eval: bucket 0 perplexity 644.72
  eval: bucket 1 perplexity 759.16
  eval: bucket 2 perplexity 984.18
  eval: bucket 3 perplexity 1585.68
global step 60 learning rate 0.5000 step-time 19.76 perplexity 1580.55
  eval: bucket 0 perplexity 1724.84
  eval: bucket 1 perplexity 2292.24
  eval: bucket 2 perplexity 2698.52
  eval: bucket 3 perplexity 3189.30
global step 70 learning rate 0.5000 step-time 17.16 perplexity 1250.57
  eval: bucket 0 perplexity 298.55
  eval: bucket 1 perplexity 502.04
  eval: bucket 2 perplexity 645.44
  eval: bucket 3 perplexity 604.29
global step 80 learning rate 0.5000 step-time 18.50 perplexity 793.90
  eval: bucket 0 perplexity 2056.23
  eval: bucket 1 perplexity 1344.26
  eval: bucket 2 perplexity 767.82
  eval: bucket 3 perplexity 649.38
global step 90 learning rate 0.5000 step-time 12.61 perplexity 541.57
  eval: bucket 0 perplexity 180.86
  eval: bucket 1 perplexity 350.99
  eval: bucket 2 perplexity 326.85
  eval: bucket 3 perplexity 383.22
global step 100 learning rate 0.5000 step-time 18.42 perplexity 471.12
  eval: bucket 0 perplexity 216.63
  eval: bucket 1 perplexity 348.96
  eval: bucket 2 perplexity 318.20
  eval: bucket 3 perplexity 389.92
global step 110 learning rate 0.5000 step-time 18.39 perplexity 474.89
  eval: bucket 0 perplexity 8049.85
  eval: bucket 1 perplexity 1677.24
  eval: bucket 2 perplexity 936.98
  eval: bucket 3 perplexity 657.46
global step 120 learning rate 0.5000 step-time 18.81 perplexity 832.11
  eval: bucket 0 perplexity 189.22
  eval: bucket 1 perplexity 360.69
  eval: bucket 2 perplexity 410.57
  eval: bucket 3 perplexity 456.40
global step 130 learning rate 0.5000 step-time 20.34 perplexity 452.27
  eval: bucket 0 perplexity 196.93
  eval: bucket 1 perplexity 655.18
  eval: bucket 2 perplexity 860.44
  eval: bucket 3 perplexity 1062.36
global step 140 learning rate 0.5000 step-time 21.05 perplexity 847.11
  eval: bucket 0 perplexity 391.88
  eval: bucket 1 perplexity 339.09
  eval: bucket 2 perplexity 320.08
  eval: bucket 3 perplexity 376.44
global step 150 learning rate 0.4950 step-time 15.53 perplexity 590.03
  eval: bucket 0 perplexity 269.16
  eval: bucket 1 perplexity 286.51
  eval: bucket 2 perplexity 391.78
  eval: bucket 3 perplexity 485.23
global step 160 learning rate 0.4950 step-time 19.36 perplexity 400.80
  eval: bucket 0 perplexity 137.00
  eval: bucket 1 perplexity 198.85
  eval: bucket 2 perplexity 276.58
  eval: bucket 3 perplexity 357.78
global step 170 learning rate 0.4950 step-time 17.50 perplexity 541.79
  eval: bucket 0 perplexity 1051.29
  eval: bucket 1 perplexity 626.64
  eval: bucket 2 perplexity 496.32
  eval: bucket 3 perplexity 458.85
global step 180 learning rate 0.4950 step-time 16.69 perplexity 400.65
  eval: bucket 0 perplexity 178.12
  eval: bucket 1 perplexity 299.86
  eval: bucket 2 perplexity 294.84
  eval: bucket 3 perplexity 296.46
global step 190 learning rate 0.4950 step-time 19.93 perplexity 886.73
  eval: bucket 0 perplexity 860.60
  eval: bucket 1 perplexity 910.16
  eval: bucket 2 perplexity 909.24
  eval: bucket 3 perplexity 786.04
global step 200 learning rate 0.4901 step-time 18.75 perplexity 449.64
  eval: bucket 0 perplexity 152.13
  eval: bucket 1 perplexity 234.41
  eval: bucket 2 perplexity 249.66
  eval: bucket 3 perplexity 285.95
...
global step 980 learning rate 0.4215 step-time 18.31 perplexity 208.74
  eval: bucket 0 perplexity 78.45
  eval: bucket 1 perplexity 108.40
  eval: bucket 2 perplexity 137.83
  eval: bucket 3 perplexity 173.53
global step 990 learning rate 0.4173 step-time 17.31 perplexity 175.05
  eval: bucket 0 perplexity 78.37
  eval: bucket 1 perplexity 119.72
  eval: bucket 2 perplexity 169.11
  eval: bucket 3 perplexity 202.89
global step 1000 learning rate 0.4173 step-time 15.85 perplexity 174.33
  eval: bucket 0 perplexity 76.52
  eval: bucket 1 perplexity 125.97
  eval: bucket 2 perplexity 150.13
  eval: bucket 3 perplexity 181.07
...
```

经过350000轮训练模型之后，您可以将代码中的 `main_train()` 换为 `main_decode()` 来使用训练好的翻译器， 您输入一个英文句子，程序将输出一个对应的法文句子。

```
Reading model parameters from wmt/translate.ckpt-350000
>  Who is the president of the United States?
Qui est le président des États-Unis ?
```

1.2.12 理解机器翻译

Seq2seq

序列到序列模型（Seq2seq）通常被用来转换一种语言到另一种语言。 但实际上它能用来做很多您可能无法想象的事情，比如我们可以将一个长的句子翻译成意思一样但短且简单的句子， 再比如，从莎士比亚的语言翻译成现代英语。若用上卷积神经网络(CNN)的话，我们能将视频翻译成句子，则自动看一段视频给出该视频的文字描述（Video captioning）。

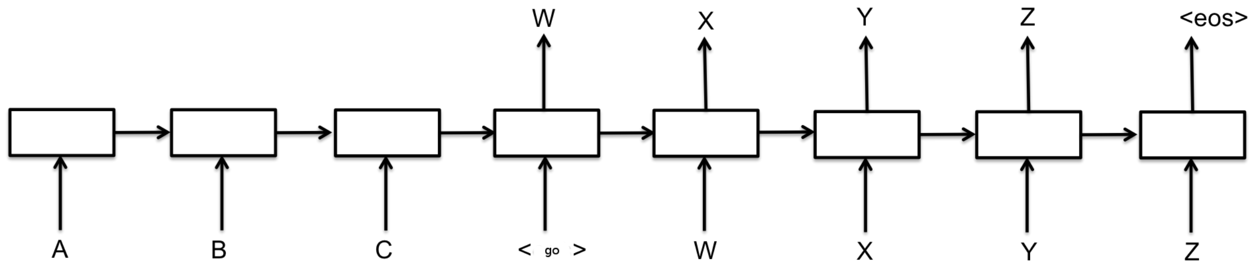
如果你只是想用 Seq2seq，你只需要考虑训练集的格式，比如如何切分单词、如何数字化单词等等。 所以，在本教程中，我们将讨论很多如何整理训练集。

基础

序列到序列模型是一种多对多（Many to many）的模型，但与PTB教程中的同步序列输入与输出(Synced sequence input and output）不一样，Seq2seq是在输入了整个序列之后，才开始输出新的序列（非同步）。该教程用了下列两种最新的方法来提高准确度： - 把输入序列倒转输入（Reversing the inputs） - 注意机制（Attention mechanism）

为了要加快训练速度，我们使用了： - softmax 抽样（Sampled softmax）

Karpathy的博客是这样描述Seq2seq的："(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)."



如上图所示，编码器输入（encoder input），解码器输入（decoder input）以及输出目标（targets）如下：

```
encoder_input =  A     B     C
decoder_input =  <go> W     X     Y     Z
targets       =  W     X     Y     Z     <eos>

 Note: 在代码实现中，targets的长度比decoder_input的长度小一，更多实现细节将在下文说明。
```

文献

该英语-法语的机器翻译例子使用了多层递归神经网络以及注意机制。 该模型和如下论文中一样：

- Grammar as a Foreign Language

该例子采用了 softmax 抽样（sampled softmax）来解决当词汇表很大时计算量大的问题。 在该例子中，`target_vocab_size=4000`，若词汇量小于 `512` 时用普通的softmax cross entropy即可。 Softmax 抽样在这篇论文的第三小节中描述:

- On Using Very Large Target Vocabulary for Neural Machine Translation

如下文章讲述了把输入序列倒转（Reversing the inputs）和多层神递归神经网络用在Seq2seq的翻译应用非常成功：

- Sequence to Sequence Learning with Neural Networks

如下文章讲述了注意机制（**Attention Mechanism**）让解码器可以更直接地得到每一个输入的信息：

- Neural Machine Translation by Jointly Learning to Align and Translate

如下文章讲述了另一种**Seq2seq**模型，则使用双向编码器（**Bi-directional encoder**）：

- Neural Machine Translation by Jointly Learning to Align and Translate

实现细节

Bucketing and Padding

Bucketing 是一种能有效处理不同句子长度的方法，为什么使用Bucketing，在 知乎 上已经有很好的回答了。

当将英文翻译成法文的时，我们有不同长度的英文句子输入（长度为 `L1 `` ）`，以及不同长度的法文句子输出，（长度为 ```L2 ）`。我们原则上要建立每一种长度的可能性，则有很多个 `(L1, L2+1)`，其中 `L2` 加一是因为有 GO 标志符。

为了减少 bucket 的数量以及为句子找到最合适的 bucket，若 bucket 大于句子的长度，我们则使用 PAD 标志符填充之。

为了提高效率，我们只使用几个 bucket，然后使用 padding 来让句子匹配到最相近的 bucket 中。 在该例子中，我们使用如下 4 个 buckets。

```
buckets = [(5, 10), (10, 15), (20, 25), (40, 50)]
```

如果输入的是一个有 `3` 个单词的英文句子，对应的法文输出有 `6` 个单词， 那么改数据将被放在第一个 bucket 中并且把 encoder inputs 和 decoder inputs 通过 padding 来让其长度变成 `5` 和 `10` 。 如果我们有 `8` 个单词的英文句子，及 `18` 个单词的法文句子，它们会被放到 `(20, 25)` 的 bucket 中。

换句话说，bucket `(I,O)` 是 (编码器输入大小`(encoder_input_size)`，解码器输入大小`(decoder_inputs_size)`)。

给出一对数字化训练样本 `[["I", "go", "."], ["Je", "vais", "."]]`，我们把它转换为 `(5,10)` 。 编码器输入（encoder inputs）的训练数据为 `[PAD PAD "." "go" "I"]`，而解码器的输入（decoder inputs）为 `[GO "Je" "vais" "." EOS PAD PAD PAD PAD PAD]`。 而输出目标（targets）是解码器输入（decoder inputs）平移一位。 `target_weights` 是输出目标（targets）的掩码。

. code-block:: text

> bucket = (I, O) = (5, 10) encoder_inputs = [PAD PAD "." "go" "I"] <– 5 x batch_size decoder_inputs = [GO "Je" "vais" "." EOS PAD PAD PAD PAD PAD] <– 10 x batch_size target_weights = [1 1 1 1 0 0 0 0 0 0] <– 10 x batch_size targets = ["Je" "vais" "." EOS PAD PAD PAD PAD PAD] <– 9 x batch_size

在该代码中，一个句子是由一个列向量表示，假设 `batch_size = 3` ， `bucket = (5, 10)` ，训练集如下所示。

```
encoder_inputs    decoder_inputs    target_weights    targets
0    0    0       1    1    1       1    1    1       87   71   16748
0    0    0       87   71   16748   1    1    1       2    3    14195
0    0    0       2    3    14195   0    1    1       0    2    2
0    0    3233    0    2    2       0    0    0       0    0    0
3    698  4061    0    0    0       0    0    0       0    0    0
                  0    0    0       0    0    0       0    0    0
                  0    0    0       0    0    0       0    0    0
                  0    0    0       0    0    0       0    0    0
                  0    0    0       0    0    0       0    0    0
                  0    0    0       0    0    0
```

```
其中 0 : _PAD     1 : _GO      2 : _EOS       3 : _UNK
```

在训练过程中，解码器输入是目标，而在预测过程中，下一个解码器的输入是最后一个解码器的输出。

在训练过程中，编码器输入（decoder inputs）就是目标输出（targets）； 当使用模型时，下一个编码器输入（decoder inputs）是上一个解码器输出（ decoder output）。

特殊标志符、标点符号与阿拉伯数字

该例子中的特殊标志符是：

```
_PAD = b"_PAD"
_GO = b"_GO"
_EOS = b"_EOS"
_UNK = b"_UNK"
PAD_ID = 0      <-- index (row number) in vocabulary
GO_ID = 1
EOS_ID = 2
UNK_ID = 3
_START_VOCAB = [_PAD, _GO, _EOS, _UNK]
```

```
        ID号     意义
_PAD    0       Padding, empty word
_GO     1       decoder_inputs 的第一个元素
_EOS    2       targets 的结束符
_UNK    3       不明单词（Unknown word），没有在词汇表出现的单词被标记为3
```

对于阿拉伯数字，建立词汇表时与数字化数据集时的 `normalize_digits` 必须是一致的，若 `normalize_digits=True` 所有阿拉伯数字都将被 `0` 代替。比如 `123` 被 `000` 代替，`9` 被 `0``代替，``1990-05` 被 `0000-00`` 代替，最后 ``000`，`0`，`0000-00` 等将在词汇库中(看 `vocab40000.en`)。

反之，如果 `normalize_digits=False`，不同的阿拉伯数字将会放入词汇表中，那么词汇表就变得十分大了。 本例子中寻找阿拉伯数字使用的正则表达式是 `_DIGIT_RE = re.compile(br"\d")` 。(详见 `tl.nlp.create_vocabulary()` 和 ``tl.nlp.data_to_token_ids()`)

对于分离句子成独立单词，本例子使用正则表达式 `_WORD_SPLIT = re.compile(b"([.,!?\"':;)(])")`，这意味着使用这几个标点符号 `[ . , ! ? " ' : ; ) ( ]` 以及空格来分割句子，详情请看 `tl.nlp.basic_tokenizer()` 。这个分割方法是 `tl.nlp.create_vocabulary()` 和 `tl.nlp.data_to_token_ids()` 的默认方法。

所有的标点符号，比如 `. , ) (` 在英文和法文数据库中都会被全部保留下来。

Softmax 抽样 (Sampled softmax)

softmax抽样是一种词汇表很大（Softmax 输出很多）的时候用来降低损失（cost）计算量的方法。 与从所有输出中计算 cross-entropy 相比，这个方法只从 `num_samples` 个输出中计算 cross-entropy。

损失和更新函数

`EmbeddingAttentionSeq2seqWrapper` 内部实现了 SGD optimizer。

下一步?

您可以尝试其他应用。

1.2.13 翻译对照

Stacked Denosing Autoencoder 堆栈式降噪自编吗器

Word Embedding 词嵌套、词嵌入

Iteration 迭代

Natural Language Processing 自然语言处理

Sparse 稀疏的

Cost function 损失函数

Regularization 规则化、正则化

Tokenization 数字化

Truncated backpropagation 截断反向传播

1.2.14 更多信息

TensorLayer 还能做什么？请继续阅读本文档。

最后，API 参考列表和说明如下：

layers (`tensorlayer.layers`),

activation (`tensorlayer.activation`),

natural language processing (`tensorlayer.nlp`),

reinforcement learning (`tensorlayer.rein`),

cost expressions and regularizers (`tensorlayer.cost`),

load and save files (`tensorlayer.files`),

operating system (`tensorlayer.ops`),

helper functions (`tensorlayer.utils`),

visualization (`tensorlayer.visualize`),

iteration functions (`tensorlayer.iterate`),

preprocessing functions (`tensorlayer.prepro`),

1.3 例子 Examples

1.3.1 Basics

- Multi-layer perceptron (MNIST). Classification task, see tutorial_mnist_simple.py.
- Multi-layer perceptron (MNIST). Classification using Iterator, see method1 and method2.

1.3.2 Computer Vision

- Denoising Autoencoder (MNIST). Classification task, see tutorial_mnist.py.
- Stacked Denoising Autoencoder and Fine-Tuning (MNIST). A MLP classification task, see tutorial_mnist.py.
- Convolutional Network (MNIST). Classification task, see tutorial_mnist.py.
- Convolutional Network (CIFAR-10). Classification task, see tutorial_cifar10.py and tutorial_cifar10_tfrecord.py.
- VGG 16 (ImageNet). Classification task, see tutorial_vgg16.py.
- VGG 19 (ImageNet). Classification task, see tutorial_vgg19.py.
- InceptionV3 (ImageNet). Classification task, see tutorial_inceptionV3_tfslim.py.
- Wide ResNet (CIFAR) by ritchieng.
- More CNN implementations of TF-Slim can be connected to TensorLayer via SlimNetsLayer.
- Spatial Transformer Networks by zsdonghao.
- U-Net for brain tumor segmentation by zsdonghao.
- Variational Autoencoder (VAE) for (CelebA) by yzwxx.
- Image Captioning - Reimplementation of Google's im2txt by zsdonghao.

1.3.3 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.

1.3.4 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.
- Super Resolution GAN by zsdonghao.

1.3.5 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 <https://github.com/ugo-nama-kun/gym_torcs>*) by zsdonghao.
- TRPO for continuous and discrete action space by jjkke88.

1.3.6 Special Examples

- Merge TF-Slim into TensorLayer. tutorial_inceptionV3_tfslim.py.
- 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_image_preprocess.py.
- TensorDB by fangde see here.
- A simple web service - TensorFlask by JoelKronander.

1.4 开发 Development

TensorLayer始于一个帝国理工的研究项目，目前由数位 GitHub 的贡献者负责维护和进行改进。

作为一个为研究员和工程师开发的开源项目，我们诚挚地欢迎您为对项目提供帮助。 每个微小的支持都会帮助我们并且会被记录下来。

1.4.1 理念

TensorLayer的想法源于组合TensorFlow的灵活性和正确的 building blocks的可用性来训练神经网络的需求。它依照下列设计目标所开发：

- **简洁**：易于使用，易于扩展与修改以便于研究和工程中使用。
- **快速**：在GPU的支持下运行速度与纯TensorFlow脚本速度一致。简洁但不牺牲性能。
- **兼容**：一个网络被抽象到正则化，成本和每一层的输出，易于与其他TensorFlow的库协作。
- **透明**：在抽象后面不隐藏TensorFlow，尽量依赖TensorFlow的功能并且遵循TensorFlow的约定。

如何参与我们

如果您在深度学习和强化学习方面发布了一种新的算法， 欢迎您分享您的算法给TensorLayer

- 说明它是怎么工作的，如果可以话请给出学术论文的链接。
- 尽可能地缩减其范围，以以致于便于实现

报告BUG

请您在 GitHub 上报告BUG。 如果您打算报告BUG，请包含以下内容：

- 您的TensorLayer和TensorFlow版本号
- 重现BUG的步骤，最好能减少到数个python 命令
- 您获得的结果和您期望的结果。

如果您不确定遇到的行为是否是BUG， 或者你不确定错误是否与TensorLayer或TensorFlow有关， 请您先在 our mailing list 查看下。

修复BUG

通过GitHub的问题栏(issues)来查看BUG报告。 任何被标记为BUG的项对所有想要修复它的人来说都是开放的。 如果您发现了TensorLayer的一个你可以自己修复的BUG， 您可以用任何方法来实现修复并且无需立即报告这个BUG。

编写文档

无论什么时候您发现有些文档没有解释清楚，存在误导，敷衍带过或者根本就是错的。 请及时更新它！ *Edit on GitHub*的链接就在每一篇文档的右上角 并且API引用列表中的每篇文档的*[source]*的链接可以帮助您快速地定位任何文字的根源。

1.4.2 如何参与我们

在GitHub上编辑

正如刚刚文档中修复BUG所说的简单方法， 点击文档右上角的*Edit on GitHub*链接或者API引用列表中的对象的*[source]*链接来打开GitHub中的源文件， 然后在你的浏览器中点击*Edit this file*链接并发送拉请求(Pull Request). 你只需要一个免费的GitHub账户就可以做到了。

对于任何较大幅度的修改，请遵循以下步骤来更新TensorLayer开发。

文档

文档由 Sphinx 生成。 如果要本地编辑它，运行下列命令： .. code:: bash

  cd docs make html

如果您想要重新生成整个文档，运行下列命令：

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

然后，打开 `docs/_build/index.html` 来查看会出现在 **‘readthedocs<http://tensorlayer.readthedocs/org/>‘_** 文档。 如果您更改了很多内容，并且似乎出现了许多误导性的错误信息或警告，运行‘‘make clean html‘‘来让Sphinx重新生成所有文件。

编写英文文档文字时，请尽可能地按照现有文档的文字习惯， 来保证整个库文字的一致性。所使用的语法及约定的相关信息，请参考以下文档：

- reStructuredText Primer

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

测试

TensorLayer有100%的代码覆盖率，这过去被证明是十分有用的的， 但也带来了一些责任：

- 每当您更改任何代码的时候，您应该运行测试脚本来测试它是否能优化现有属性。
- 您修改的每个BUG说明一个缺少的测试案例，

所以每个修复BUG的方案应该配置一个您没修复前的测试案例。

发送拉请求

当您对您添加的内容感到满意并且测试通过，文档规范，简明，不存在任何注释错误。 您可以将您的更改提交到一个新的分支(branch)，并且将这个分支与您的副本(fork)合并， 然后通过GitHub的网站发送一个拉请求(pull request)

所有的这些步骤在GitHub上有相当不错的说明： https://guides.github.com/introduction/flow/

当您提交拉请求时，请附带一个更改内容的说明，以帮助我们能更好的检阅它。 如果它是一个正在开放的问题(issue)，比如：issue#123，请在您的描述中添加 *Fixes#123*,*Resolves#123*或者*Closes#123*，这样当您的拉请求被接纳之后 GitHub会关闭那个问题。

1.5 更多 More

1.5.1 FQA

请见 英文网站 .

1.5.2 你的贡献

贡献方式有很多种，比如提供应用、实现新的TensorLayer层、回答 GitHub Issues 以及帮助完善翻译等等。每一点贡献都会署名，欢迎在 GitHub 上push。

1.5.3 招聘信息

帝国理工大学－数据科学院

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.

企鹅创新（北京）科技有限公司－算法工程师（30~60万人民币）

公司简介：我们是设计为主导，技术为基石的商业公司；致力于创造人工智能和万物互联时代最卓越的产品。 团队成员相继服务于Google、MUJI无印良品、Dyson戴森、PWC普华永道、D&AD、Oakley、DIAGEO及国内顶级科研院所。

岗位职责

- 使用深度学习解决机器视觉问题；
- 把深度学习算法部署到嵌入式系统。

岗位要求

- 具备2年以上深度学习、机器视觉、图像处理等相关背景；
- 精通C/C++ 和 Python；
- 熟悉TensorLayer和TensorFlow优先；
- 熟悉基于CUDA的算法设计与优化和并行优化经验者优先；
- 熟悉基本的数字图像与视频处理算法原理，熟练掌握OpenCV／OpenGL ES，了解图像拼接图像处理算法；
- 良好的团队合作意识；
- 能够长期稳定的供职。

地点：北京望京SOHO

年薪：30~60万人民币

联系方式：info@penguinsinnovate.com

CHAPTER 2

API目录

如果你正在寻找某个特殊的函数，类或者方法，这一列文档就是为你准备的。

2.1 API - 神经网络层

为了尽可能地保持TensorLayer的简洁性，我们最小化Layer的数量，因此我们鼓励用户直接使用 TensorFlow官方的函数。 例如，虽然我们提供local response normalization layer，但用户也可以在 `network.outputs` 上使用 `tf.nn.lrn()` 来实现之。 更多TensorFlow官方函数请看 这里。

2.1.1 了解层

所有TensorLayer层有如下的属性:

- `layer.outputs` : 一个 Tensor，当前层的输出。
- `layer.all_params` : 一列 Tensor, 神经网络每一个参数。
- `layer.all_layers` : 一列 Tensor, 神经网络每一层输出。
- `layer.all_drop` : 一个字典 {placeholder : 浮点数}, 噪声层的概率。

所有TensorLayer层有如下的方法:

- `layer.print_params()` : 打印出神经网络的参数信息（在执行 `tl.layers.initialize_global_variables(sess)` 之后）。另外，也可以使用 `tl.layers.print_all_variables()` 来打印出所有参数的信息。
- `layer.print_layers()` : 打印出神经网络每一层输出的信息。
- `layer.count_params()` : 打印出神经网络参数的数量。

神经网络的初始化是通过输入层实现的，然后我们可以像下面的代码那样把不同的层堆叠在一起，实现一个完整的神经网络，因此一个神经网络其实就是一个 `Layer` 类。 神经网络中最重要的属性有 `network.all_params`, `network.all_layers` 和 `network.all_drop`. 其中 `all_params` 是一个列表(list)，它按顺序保存了指向神经网络参数(variables)的指针，下面的代码定义了一个三层神经网络，则:

`all_params` = [W1, b1, W2, b2, W_out, b_out]

若需要取出特定的参数，您可以通过 `network.all_params[2:3]` 或 `get_variables_with_name()` 函数。然而 `all_layers` 也是一个列表(list)，它按顺序保存了指向神经网络每一层输出的指针，在下面的网络中，则：

`all_layers` = [drop(?,784), relu(?,800), drop(?,800), relu(?,800), drop(?,800)], identity(?,10)]

其中 ? 代表任意batch size都可以。 你可以通过 `network.print_layers()` 和 `network.print_params()` 打印出每一层输出的信息以及每一个参数的信息。 若想参看神经网络中有多少个参数，则运行 `network.count_params()` 。

```
sess = tf.InteractiveSession()

x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')

network = tl.layers.InputLayer(x, name='input_layer')
network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu1')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu2')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
network = tl.layers.DenseLayer(network, n_units=10,
                                act = tl.activation.identity,
                                name='output_layer')

y = network.outputs
y_op = tf.argmax(tf.nn.softmax(y), 1)

cost = tl.cost.cross_entropy(y, y_, name='ce')

train_params = network.all_params

train_op = tf.train.AdamOptimizer(learning_rate, beta1=0.9, beta2=0.999,
                            epsilon=1e-08, use_locking=False).minimize(cost, var_list␣
↪= train_params)

tl.layers.initialize_global_variables(sess)

network.print_params()
network.print_layers()
```

另外，`network.all_drop` 是一个字典，它保存了噪声层（比如dropout）的 keeping 概率。 在上面定义的神经网络中，它保存了三个dropout层的keeping概率。

因此，在训练时如下启用dropout层。

```
feed_dict = {x: X_train_a, y_: y_train_a}
feed_dict.update( network.all_drop )
loss, _ = sess.run([cost, train_op], feed_dict=feed_dict)
feed_dict.update( network.all_drop )
```

在测试时，如下关闭dropout层。

```
feed_dict = {x: X_val, y_: y_val}
feed_dict.update(dp_dict)
print("   val loss: %f" % sess.run(cost, feed_dict=feed_dict))
print("   val acc: %f" % np.mean(y_val ==
                        sess.run(y_op, feed_dict=feed_dict)))
```

更多细节，请看 MNIST 例子。

2.1.2 了解Dense层

在创造自定义层之前，我们来看看全连接（Dense）层是如何实现的。 若不存在Weights矩阵和Biases向量时，它新建之，然后通过给定的激活函数计算出 `outputs` 。 在最后，作为一个有新参数的层，我们需要把新参数附加到 `all_params` 中。

```
class MyDenseLayer(Layer):
  def __init__(
      self,
      layer = None,
      n_units = 100,
      act = tf.nn.relu,
      name ='simple_dense',
  ):
      # 校验名字是否已被使用（不变）
      Layer.__init__(self, name=name)

      # 本层输入是上层的输出（不变）
      self.inputs = layer.outputs

      # 输出信息（自定义部分）
      print("  MyDenseLayer %s: %d, %s" % (self.name, n_units, act))

      # 本层的功能实现（自定义部分）
      n_in = int(self.inputs._shape[-1])  # 获取上一层输出的数量
      with tf.variable_scope(name) as vs:
          # 新建参数
          W = tf.get_variable(name='W', shape=(n_in, n_units))
          b = tf.get_variable(name='b', shape=(n_units))
          # tensor操作
          self.outputs = act(tf.matmul(self.inputs, W) + b)

      # 获取之前层的参数（不变）
      self.all_layers = list(layer.all_layers)
      self.all_params = list(layer.all_params)
      self.all_drop = dict(layer.all_drop)

      # 更新层的参数（自定义部分）
      self.all_layers.extend( [self.outputs] )
      self.all_params.extend( [W, b] )
```

2.1.3 自定义层

一个简单的层

实现一个自定义层，你需要写一个新的Python类，然后实现 `outputs` 表达式。

下面的例子实现了把输入乘以2，然后输出。

```
class DoubleLayer(Layer):
    def __init__(
        self,
        layer = None,
        name ='double_layer',
    ):
        # 校验名字是否已被使用（不变）
        Layer.__init__(self, name=name)

        # 本层输入是上层的输出（不变）
        self.inputs = layer.outputs

        # 输出信息（自定义部分）
        print("  I am DoubleLayer")

        # 本层的功能实现（自定义部分）
        self.outputs = self.inputs * 2

        # 获取之前层的参数（不变）
        self.all_layers = list(layer.all_layers)
        self.all_params = list(layer.all_params)
        self.all_drop = dict(layer.all_drop)

        # 更新层的参数（自定义部分）
        self.all_layers.extend( [self.outputs] )
```

2.1.4 修改预训练行为

逐层贪婪预训练方法(Greedy layer-wise pretrain)是深度神经网络的初始化非常重要的一种方法， 不过对不同的网络结构和应用，往往有不同的预训练的方法。

例如 "普通"稀疏自编码器(Vanilla Sparse Autoencoder) 如下面的代码所示，使用 KL divergence 实现（对应于sigmoid)， 但是对于 深度整流神经网络(Deep Rectifier Network) ， 可以通过对神经元输出进行L1规则化来实现稀疏。

```
# Vanilla Sparse Autoencoder
beta = 4
rho = 0.15
p_hat = tf.reduce_mean(activation_out, reduction_indices = 0)
KLD = beta * tf.reduce_sum( rho * tf.log(tf.div(rho, p_hat))
        + (1- rho) * tf.log((1- rho)/ (tf.sub(float(1), p_hat))) )
```

预训练的方法太多了，出于这个原因，TensorLayer 提供了一种简单的方法来自定义自己的预训练方法。对于自编码器，TensorLayer 使用 `ReconLayer.__init__()` 来定义重构层（reconstruction layer）和损失函数。 要自定义自己的损失函数，只需要在 `ReconLayer.__init__()` 中修改 `self.cost` 就可以了。 如何写出自己的损失函数，请阅读 Tensorflow Math 。 默认情况下， 重构层(`ReconLayer`) 只使用 `self.train_params = self.all _params[-4:]` 来更新前一层的 Weights 和 Biases，这4个参数为 `[W_encoder, b_encoder, W_decoder, b_decoder]` ， 其中 `W_encoder, b_encoder` 属于之前的 Dense 层， `W_decoder, b_decoder]` 属于当前的重构层。 此外，如果您想要同时更新前 2 层的参数，只需要修改 `[-4:]` 为 `[-6:]`。

```
ReconLayer.__init__(...):
    ...
    self.train_params = self.all_params[-4:]
    ...
      self.cost = mse + L1_a + L2_w
```

2.1.5 层预览表

get_variables_with_name(name[, train_only, ...])	Get variable list by a given name scope.
get_layers_with_name([network, name, printable])	Get layer list in a network by a given name scope.
set_name_reuse([enable])	Enable or disable reuse layer name.
print_all_variables([train_only])	Print all trainable and non-trainable variables
initialize_global_variables([sess])	Excute `sess.run(tf.global_variables_initializer())` for TF12+ or sess.run(tf.initialize_all_variables()) for TF11.
Layer([inputs, name])	The *Layer* class represents a single layer of a neural network.
InputLayer([inputs, name])	The *InputLayer* class is the starting layer of a neural network.
OneHotInputLayer([inputs, depth, on_value, ...])	The *OneHotInputLayer* class is the starting layer of a neural network, see `tf.one_hot`.
Word2vecEmbeddingInputlayer([inputs, ...])	The *Word2vecEmbeddingInputlayer* class is a fully connected layer, for Word Embedding.
EmbeddingInputlayer([inputs, ...])	The *EmbeddingInputlayer* class is a fully connected layer, for Word Embedding.
DenseLayer([layer, n_units, act, W_init, ...])	The *DenseLayer* class is a fully connected layer.
ReconLayer([layer, x_recon, name, n_units, act])	The *ReconLayer* class is a reconstruction layer *DenseLayer* which use to pre-train a *DenseLayer*.
DropoutLayer([layer, keep, is_fix, ...])	The *DropoutLayer* class is a noise layer which randomly set some values to zero by a given keeping probability.
GaussianNoiseLayer([layer, mean, stddev, ...])	The *GaussianNoiseLayer* class is noise layer that adding noise with normal distribution to the activation.
DropconnectDenseLayer([layer, keep, ...])	The *DropconnectDenseLayer* class is `DenseLayer` with DropConnect behaviour which randomly remove connection between this layer to previous layer by a given keeping probability.
Conv1dLayer([layer, act, shape, stride, ...])	The *Conv1dLayer* class is a 1D CNN layer, see tf.nn.convolution.
Conv2dLayer([layer, act, shape, strides, ...])	The *Conv2dLayer* class is a 2D CNN layer, see tf.nn.conv2d.
DeConv2dLayer([layer, act, shape, ...])	The *DeConv2dLayer* class is deconvolutional 2D layer, see tf.nn.conv2d_transpose.
Conv3dLayer([layer, act, shape, strides, ...])	The *Conv3dLayer* class is a 3D CNN layer, see tf.nn.conv3d.
DeConv3dLayer([layer, act, shape, ...])	The *DeConv3dLayer* class is deconvolutional 3D layer, see tf.nn.conv3d_transpose.
PoolLayer([layer, ksize, strides, padding, ...])	The *PoolLayer* class is a Pooling layer, you can choose `tf.nn.max_pool` and `tf.nn.avg_pool` for 2D or `tf.nn.max_pool3d` and `tf.nn.avg_pool3d` for 3D.
PadLayer([layer, paddings, mode, name])	The *PadLayer* class is a Padding layer for any modes and dimensions.

Continued on next page

表 2.1 – continued from previous page

UpSampling2dLayer([layer, size, is_scale, ...])	The *UpSampling2dLayer* class is upSampling 2d layer, see tf.image.resize_images.
DownSampling2dLayer([layer, size, is_scale, ...])	The *DownSampling2dLayer* class is downSampling 2d layer, see tf.image.resize_images.
AtrousConv1dLayer(net[, n_filter, ...])	Wrapper for *AtrousConv1dLayer*, if you don't understand how to use *Conv1dLayer*, this function may be easier.
AtrousConv2dLayer([layer, n_filter, ...])	The *AtrousConv2dLayer* class is Atrous convolution (a.k.a.
SeparableConv2dLayer([layer, filters, ...])	The *SeparableConv2dLayer* class is 2-D convolution with separable filters, see tf.layers.separable_conv2d.
Conv1d(net[, n_filter, filter_size, stride, ...])	Wrapper for *Conv1dLayer*, if you don't understand how to use *Conv1dLayer*, this function may be easier.
Conv2d(net[, n_filter, filter_size, ...])	Wrapper for *Conv2dLayer*, if you don't understand how to use *Conv2dLayer*, this function may be easier.
DeConv2d(net[, n_out_channel, filter_size, ...])	Wrapper for *DeConv2dLayer*, if you don't understand how to use *DeConv2dLayer*, this function may be easier.
MaxPool1d(net, filter_size, strides[, ...])	Wrapper for tf.layers.max_pooling1d .
MeanPool1d(net, filter_size, strides[, ...])	Wrapper for tf.layers.average_pooling1d .
MaxPool2d(net[, filter_size, strides, ...])	Wrapper for *PoolLayer*.
MeanPool2d(net[, filter_size, strides, ...])	Wrapper for *PoolLayer*.
MaxPool3d(net, filter_size, strides[, ...])	Wrapper for tf.layers.max_pooling3d .
MeanPool3d(net, filter_size, strides[, ...])	Wrapper for tf.layers.average_pooling3d
SubpixelConv1d(net[, scale, act, name])	One-dimensional subpixel upsampling layer.
SubpixelConv2d(net[, scale, n_out_channel, ...])	The *SubpixelConv2d* class is a sub-pixel 2d convolutional ayer, usually be used for Super-Resolution applications, example code.
SpatialTransformer2dAffineLayer([layer, ...])	The *SpatialTransformer2dAffineLayer* class is a Spatial Transformer Layer for 2D Affine Transformation.
transformer(U, theta, out_size[, name])	Spatial Transformer Layer for 2D Affine Transformation , see *SpatialTransformer2dAffineLayer* class.
batch_transformer(U, thetas, out_size[, name])	Batch Spatial Transformer function for 2D Affine Transformation.
BatchNormLayer([layer, decay, epsilon, act, ...])	The *BatchNormLayer* class is a normalization layer, see `tf.nn.batch_normalization` and `tf.nn.moments`.
LocalResponseNormLayer([layer, ...])	The *LocalResponseNormLayer* class is for Local Response Normalization, see `tf.nn.local_response_normalization` or `tf.nn.lrn` for new TF version.
InstanceNormLayer([layer, act, epsilon, ...])	The *InstanceNormLayer* class is a for instance normalization.
LayerNormLayer([layer, center, scale, act, ...])	The *LayerNormLayer* class is for layer normalization, see tf.contrib.layers.layer_norm.
ROIPoolingLayer([layer, rois, pool_height, ...])	The *ROIPoolingLayer* class is Region of interest pooling layer.
TimeDistributedLayer([layer, layer_class, ...])	The *TimeDistributedLayer* class that applies a function to every timestep of the input tensor.

Continued on next page

表 2.1 – continued from previous page

RNNLayer([layer, cell_fn, cell_init_args, ...])	The RNNLayer class is a RNN layer, you can implement vanilla RNN, LSTM and GRU with it.
BiRNNLayer([layer, cell_fn, cell_init_args, ...])	The BiRNNLayer class is a Bidirectional RNN layer.
advanced_indexing_op(input, index)	Advanced Indexing for Sequences, returns the outputs by given sequence lengths.
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.
DynamicRNNLayer([layer, cell_fn, ...])	The DynamicRNNLayer class is a Dynamic RNN layer, see `tf.nn.dynamic_rnn`.
Seq2Seq([net_encode_in, net_decode_in, ...])	The Seq2Seq class is a Simple DynamicRNNLayer based Seq2seq layer without using tl.contrib.seq2seq.
PeekySeq2Seq([net_encode_in, net_decode_in, ...])	Waiting for contribution.
AttentionSeq2Seq([net_encode_in, ...])	Waiting for contribution.
FlattenLayer([layer, name])	The FlattenLayer class is layer which reshape high-dimension input to a vector.
ReshapeLayer([layer, shape, name])	The ReshapeLayer class is layer which reshape the tensor.
TransposeLayer([layer, perm, name])	The TransposeLayer class transpose the dimension of a teneor, see tf.transpose() .
LambdaLayer([layer, fn, fn_args, name])	The LambdaLayer class is a layer which is able to use the provided function.
ConcatLayer([layer, concat_dim, name])	The ConcatLayer class is layer which concat (merge) two or more tensor by given axis..
ElementwiseLayer([layer, combine_fn, name])	The ElementwiseLayer class combines multiple Layer which have the same output shapes by a given elemwise-wise operation.
ExpandDimsLayer([layer, axis, name])	The ExpandDimsLayer class inserts a dimension of 1 into a tensor's shape, see tf.expand_dims() .
TileLayer([layer, multiples, name])	The TileLayer class constructs a tensor by tiling a given tensor, see tf.tile() .
StackLayer([layer, axis, name])	The StackLayer class is layer for stacking a list of rank-R tensors into one rank-(R+1) tensor, see tf.stack().
UnStackLayer([layer, num, axis, name])	The *UnStackLayer* is layer for unstacking the given dimension of a rank-R tensor into rank-(R-1) tensors., see tf.unstack().
EstimatorLayer([layer, model_fn, args, name])	The EstimatorLayer class accepts `model_fn` that described the model.
SlimNetsLayer([layer, slim_layer, ...])	The SlimNetsLayer class can be used to merge all TF-Slim nets into TensorLayer.
KerasLayer([layer, keras_layer, keras_args, ...])	The KerasLayer class can be used to merge all Keras layers into TensorLayer.
PReluLayer([layer, channel_shared, a_init, ...])	The PReluLayer class is Parametric Rectified Linear layer.
MultiplexerLayer([layer, name])	The MultiplexerLayer selects one of several input and forwards the selected input into the output, see *tutorial_mnist_multiplexer.py*.

Continued on next page

表 2.1 – continued from previous page

EmbeddingAttentionSeq2seqWrapper(...[, ...])	Sequence-to-sequence model with attention and for multiple buckets (Deprecated after TF0.12).
flatten_reshape(variable[, name])	Reshapes high-dimension input to a vector.
clear_layers_name()	Clear all layer names in set_keep['_layers_name_list'], enable layer name reuse.
initialize_rnn_state(state)	Return the initialized RNN state.
list_remove_repeat([l])	Remove the repeated items in a list, and return the processed list.

2.1.6 名称与参数复用

这些函数用以帮助您在不同的 graph 中复用相同的参数，以及如何通过一个名字来获取相应的参数列表。 更多关于 TensorFlow parameters sharing 请点击 这里。

Get variables with name

tensorlayer.layers.**get_variables_with_name**(*name*, *train_only=True*, *printable=False*)
Get variable list by a given name scope.

Examples

```
>>> dense_vars = tl.layers.get_variable_with_name('dense', True, True)
```

Get layers with name

tensorlayer.layers.**get_layers_with_name**(*network=None*, *name="*, *printable=False*)
Get layer list in a network by a given name scope.

Examples

```
>>> layers = tl.layers.get_layers_with_name(network, "CNN", True)
```

Enable layer name reuse

tensorlayer.layers.**set_name_reuse**(*enable=True*)
Enable or disable reuse layer name. By default, each layer must has unique name. When you want two or more input placeholder (inference) share the same model parameters, you need to enable layer name reuse, then allow the parameters have same name scope.

Parameters **enable** : boolean, enable name reuse. (None means False).

Examples

```
>>> def embed_seq(input_seqs, is_train, reuse):
>>>     with tf.variable_scope("model", reuse=reuse):
>>>         tl.layers.set_name_reuse(reuse)
>>>         network = tl.layers.EmbeddingInputlayer(
...                     inputs = input_seqs,
...                     vocabulary_size = vocab_size,
...                     embedding_size = embedding_size,
...                     name = 'e_embedding')
>>>        network = tl.layers.DynamicRNNLayer(network,
...                     cell_fn = tf.contrib.rnn.BasicLSTMCell,
...                     n_hidden = embedding_size,
...                     dropout = (0.7 if is_train else None),
...                     initializer = w_init,
...                     sequence_length = tl.layers.retrieve_seq_length_op2(input_
↪seqs),
...                     return_last = True,
...                     name = 'e_dynamicrnn',)
>>>     return network
>>>
>>> net_train = embed_seq(t_caption, is_train=True, reuse=False)
>>> net_test = embed_seq(t_caption, is_train=False, reuse=True)
```

- see `tutorial_ptb_lstm.py` for example.

Print variables

`tensorlayer.layers.`**`print_all_variables`**(*train_only=False*)

Print all trainable and non-trainable variables without tl.layers.initialize_global_variables(sess)

Parameters train_only : boolean

If True, only print the trainable variables, otherwise, print all variables.

Initialize variables

`tensorlayer.layers.`**`initialize_global_variables`**(*sess=None*)

Excute `sess.run(tf.global_variables_initializer())` for TF12+ or sess.run(tf.initialize_all_variables()) for TF11.

Parameters sess : a Session

2.1.7 Basic 层

class `tensorlayer.layers.`**`Layer`**(*inputs=None*, *name='layer'*)

The *`Layer`* class represents a single layer of a neural network. It should be subclassed when implementing new types of layers. Because each layer can keep track of the layer(s) feeding into it, a network's output *`Layer`* instance can double as a handle to the full network.

Parameters inputs : a *`Layer`* instance

The *Layer* class feeding into this layer.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.8 输入层

class tensorlayer.layers.**InputLayer**(*inputs=None*, *name='input_layer'*)

The *InputLayer* class is the starting layer of a neural network.

Parameters **inputs** : a placeholder or tensor

The input tensor data.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.9 One-hot 输入层

class tensorlayer.layers.**OneHotInputLayer**(*inputs=None*, *depth=None*, *on_value=None*, *off_value=None*, *axis=None*, *dtype=None*, *name='input_layer'*)

The *OneHotInputLayer* class is the starting layer of a neural network, see `tf.one_hot`.

Parameters **inputs** : a placeholder or tensor

The input tensor data.

name : a string or None

An optional name to attach to this layer.

depth : 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 : If on_value is not provided, it will default to the value 1 with type dtype.

default, None

off_value : If off_value is not provided, it will default to the value 0 with type dtype.

default, None

axis : default, None

dtype : default, None

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.10 嵌入层+输入层

训练Word2vec的层

class `tensorlayer.layers.`**`Word2vecEmbeddingInputlayer`**(*inputs=None, train_labels=None, vocabulary_size=80000, embedding_size=200, num_sampled=64, nce_loss_args={}, E_init=<tensorflow.python.ops.init_ops.RandomUnifor object>, E_init_args={}, nce_W_init=<tensorflow.python.ops.init_ops.Truncated object>, nce_W_init_args={}, nce_b_init=<tensorflow.python.ops.init_ops.Constant object>, nce_b_init_args={}, name='word2vec_layer'*)

The *`Word2vecEmbeddingInputlayer`* class is a fully connected layer, for Word Embedding. Words are input as integer index. The output is the embedded word vector.

Parameters **inputs** : placeholder

For word inputs. integer index format.

train_labels : placeholder

For word labels. integer index format.

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.

nce_loss_args : a dictionary

The arguments for tf.nn.nce_loss()

E_init : embedding initializer

The initializer for initializing the embedding matrix.

E_init_args : a dictionary

The arguments for embedding initializer

nce_W_init : NCE decoder biases initializer

The initializer for initializing the nce decoder weight matrix.

nce_W_init_args : a dictionary

The arguments for initializing the nce decoder weight matrix.

nce_b_init : NCE decoder biases initializer

The initializer for tf.get_variable() of the nce decoder bias vector.

nce_b_init_args : a dictionary

The arguments for tf.get_variable() of the nce decoder bias vector.

name : a string or None

An optional name to attach to this layer.

References

- tensorflow/examples/tutorials/word2vec/word2vec_basic.py

Examples

- Without TensorLayer : see tensorflow/examples/tutorials/word2vec/word2vec_basic.py

```
>>> train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
>>> train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
>>> embeddings = tf.Variable(
...     tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
>>> embed = tf.nn.embedding_lookup(embeddings, train_inputs)
>>> nce_weights = tf.Variable(
...     tf.truncated_normal([vocabulary_size, embedding_size],
...                    stddev=1.0 / math.sqrt(embedding_size)))
>>> nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
>>> cost = tf.reduce_mean(
...    tf.nn.nce_loss(weights=nce_weights, biases=nce_biases,
...               inputs=embed, labels=train_labels,
...               num_sampled=num_sampled, num_classes=vocabulary_size,
...               num_true=1))
```

- With TensorLayer : see tutorial_word2vec_basic.py

```
>>> train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
>>> train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
>>> emb_net = tl.layers.Word2vecEmbeddingInputlayer(
...         inputs = train_inputs,
...         train_labels = train_labels,
...         vocabulary_size = vocabulary_size,
...         embedding_size = embedding_size,
...         num_sampled = num_sampled,
...        name ='word2vec_layer',
...    )
>>> cost = emb_net.nce_cost
>>> train_params = emb_net.all_params
>>> train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(
...                                             cost, var_list=train_params)
>>> normalized_embeddings = emb_net.normalized_embeddings
```

Attributes

nce_cost	(a tensor) The NCE loss.
outputs	(a tensor) The outputs of embedding layer.
normalized_embeddings	(tensor) Normalized embedding matrix

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

嵌入层(输入)

class `tensorlayer.layers.`**`EmbeddingInputlayer`**(*inputs=None, vocabulary_size=80000, embedding_size=200, E_init=<tensorflow.python.ops.init_ops.RandomUniform object>, E_init_args={}, name='embedding_layer'*)

The `EmbeddingInputlayer` class is a fully connected layer, for Word Embedding. Words are input as integer index. The output is the embedded word vector.

If you have a pre-train matrix, you can assign the matrix into it. To train a word embedding matrix, you can used class:*Word2vecEmbeddingInputlayer*.

Note that, do not update this embedding matrix.

Parameters **inputs** : placeholder

For word inputs. integer index format. a 2D tensor : [batch_size, num_steps(num_words)]

vocabulary_size : int

The size of vocabulary, number of words.

embedding_size : int

The number of embedding dimensions.

E_init : embedding initializer

The initializer for initializing the embedding matrix.

E_init_args : a dictionary

The arguments for embedding initializer

name : a string or None

An optional name to attach to this layer.

Examples

```
>>> vocabulary_size = 50000
>>> embedding_size = 200
>>> model_file_name = "model_word2vec_50k_200"
>>> batch_size = None
...
>>> all_var = tl.files.load_npy_to_any(name=model_file_name+'.npy')
>>> data = all_var['data']; count = all_var['count']
>>> dictionary = all_var['dictionary']
>>> reverse_dictionary = all_var['reverse_dictionary']
>>> tl.files.save_vocab(count, name='vocab_'+model_file_name+'.txt')
>>> del all_var, data, count
...
>>> load_params = tl.files.load_npz(name=model_file_name+'.npz')
>>> x = tf.placeholder(tf.int32, shape=[batch_size])
>>> y_ = tf.placeholder(tf.int32, shape=[batch_size, 1])
>>> emb_net = tl.layers.EmbeddingInputlayer(
...                 inputs = x,
...                 vocabulary_size = vocabulary_size,
...                 embedding_size = embedding_size,
...                 name ='embedding_layer')
>>> tl.layers.initialize_global_variables(sess)
>>> tl.files.assign_params(sess, [load_params[0]], emb_net)
>>> word = b'hello'
>>> word_id = dictionary[word]
>>> print('word_id:', word_id)
... 6428
...
>>> words = [b'i', b'am', b'hao', b'dong']
>>> word_ids = tl.files.words_to_word_ids(words, dictionary)
>>> context = tl.files.word_ids_to_words(word_ids, reverse_dictionary)
>>> print('word_ids:', word_ids)
... [72, 1226, 46744, 20048]
>>> print('context:', context)
... [b'i', b'am', b'hao', b'dong']
...
>>> vector = sess.run(emb_net.outputs, feed_dict={x : [word_id]})
>>> print('vector:', vector.shape)
... (1, 200)
>>> vectors = sess.run(emb_net.outputs, feed_dict={x : word_ids})
>>> print('vectors:', vectors.shape)
... (4, 200)
```

Attributes

outputs	(a tensor) The outputs of embedding layer. the outputs 3D tensor : [batch_size, num_steps(num_words), embedding_size]

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network

Continued on next page

表 2.6 – continued from previous page

`print_params([details])`	Print all info of parameters in the network

2.1.11 全连接层

全连接层

class `tensorlayer.layers.`**`DenseLayer`**(*layer=None, n_units=100, act=<function identity>, W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='dense_layer'*)

The `DenseLayer` class is a fully connected layer.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

n_units : int

The number of units of the layer.

act : activation function

The function that is applied to the layer activations.

W_init : weights initializer

The initializer for initializing the weight matrix.

b_init : biases initializer or None

The initializer for initializing the bias vector. If None, skip biases.

W_init_args : dictionary

The arguments for the weights tf.get_variable.

b_init_args : dictionary

The arguments for the biases tf.get_variable.

name : a string or None

An optional name to attach to this layer.

Notes

If the input to this layer has more than two axes, it need to flatten the input by using `FlattenLayer` in this case.

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DenseLayer(
...                 network,
...                 n_units=800,
...                 act = tf.nn.relu,
...                 W_init=tf.truncated_normal_initializer(stddev=0.1),
...                     name ='relu_layer'
...                     )
```

```
>>> Without TensorLayer, you can do as follow.
>>> W = tf.Variable(
...     tf.random_uniform([n_in, n_units], -1.0, 1.0), name='W')
>>> b = tf.Variable(tf.zeros(shape=[n_units]), name='b')
>>> y = tf.nn.relu(tf.matmul(inputs, W) + b)
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

训练Autoencoder的重构层

class tensorlayer.layers.**ReconLayer**(*layer=None*, *x_recon=None*, *name='recon_layer'*, *n_units=784*, *act=<function softplus>*)

The `ReconLayer` class is a reconstruction layer *DenseLayer* which use to pre-train a *DenseLayer*.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

x_recon : tensorflow variable

The variables used for reconstruction.

name : a string or None

An optional name to attach to this layer.

n_units : int

The number of units of the layer, should be equal to x_recon

act : activation function

The activation function that is applied to the reconstruction layer. Normally, for sigmoid layer, the reconstruction activation is sigmoid; for rectifying layer, the reconstruction activation is softplus.

Notes

The input layer should be *DenseLayer* or a layer has only one axes. You may need to modify this part to define your own cost function. By default, the cost is implemented as follow: - For sigmoid layer, the implementation can be UFLDL - For rectifying layer, the implementation can be Glorot (2011). Deep Sparse Rectifier Neural Networks

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DenseLayer(network, n_units=196,
...                                 act=tf.nn.sigmoid, name='sigmoid1')
>>> recon_layer1 = tl.layers.ReconLayer(network, x_recon=x, n_units=784,
...                                 act=tf.nn.sigmoid, name='recon_layer1')
>>> recon_layer1.pretrain(sess, x=x, X_train=X_train, X_val=X_val,
...                         denoise_name=None, n_epoch=1200, batch_size=128,
...                         print_freq=10, save=True, save_name='w1pre_')
```

Methods

pretrain(self, sess, x, X_train, X_val, denoise_name=None, n_epoch=100, batch_size=128, print_freq=10, save=True, save_name='w1pre_')	Start to pre-train the parameters of previous DenseLayer.

2.1.12 噪声层

Dropout层

class tensorlayer.layers.**DropoutLayer**(*layer=None*, *keep=0.5*, *is_fix=False*, *is_train=True*, *seed=None*, *name='dropout_layer'*)

The *DropoutLayer* class is a noise layer which randomly set some values to zero by a given keeping probability.

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer.

keep : float

The keeping probability, the lower more values will be set to zero.

is_fix : boolean

Default False, if True, the keeping probability is fixed and cannot be changed via feed_dict.

is_train : boolean

If False, skip this layer, default is True.

seed : int or None

An integer or None to create random seed.

name : a string or None

An optional name to attach to this layer.

Notes

- A frequent question regarding *DropoutLayer* is that why it donot have *is_train* like *BatchNormLayer*.

In many simple cases, user may find it is better to use one inference instead of two inferences for training and testing seperately, `DropoutLayer` allows you to control the dropout rate via *feed_dict*. However, you can fix the keeping probability by setting *is_fix* to True.

Examples

- Define network

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
>>> network = tl.layers.DenseLayer(network, n_units=800, act = tf.nn.relu, name=
↪'relu1')
>>> ...
```

- For training, enable dropout as follow.

```
>>> feed_dict = {x: X_train_a, y_: y_train_a}
>>> feed_dict.update( network.all_drop )     # enable noise layers
>>> sess.run(train_op, feed_dict=feed_dict)
>>> ...
```

- For testing, disable dropout as follow.

```
>>> dp_dict = tl.utils.dict_to_one( network.all_drop ) # disable noise layers
>>> feed_dict = {x: X_val_a, y_: y_val_a}
>>> feed_dict.update(dp_dict)
>>> err, ac = sess.run([cost, acc], feed_dict=feed_dict)
>>> ...
```

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

高斯噪声层

class tensorlayer.layers.**GaussianNoiseLayer**(*layer=None*, *mean=0.0*, *stddev=1.0*, *is_train=True*, *seed=None*, *name='gaussian_noise_layer'*)

The `GaussianNoiseLayer` class is noise layer that adding noise with normal distribution to the activation.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

mean : float

stddev : float

is_train : boolean

If False, skip this layer, default is True.

seed : int or None

An integer or None to create random seed.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

Dropconnect + 全链接层

class `tensorlayer.layers.`**`DropconnectDenseLayer`**(*layer=None, keep=0.5, n_units=100, act=<function identity>, W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='dropconnect_layer'*)

The *`DropconnectDenseLayer`* class is `DenseLayer` with DropConnect behaviour which randomly remove connection between this layer to previous layer by a given keeping probability.

Parameters layer : a *`Layer`* instance

The *Layer* class feeding into this layer.

keep : float

The keeping probability, the lower more values will be set to zero.

n_units : int

The number of units of the layer.

act : activation function

The function that is applied to the layer activations.

W_init : weights initializer

The initializer for initializing the weight matrix.

b_init : biases initializer

The initializer for initializing the bias vector.

W_init_args : dictionary

The arguments for the weights tf.get_variable().

b_init_args : dictionary

The arguments for the biases tf.get_variable().

name : a string or None

An optional name to attach to this layer.

References

- Wan, L. (2013). Regularization of neural networks using dropconnect

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DropconnectDenseLayer(network, keep = 0.8,
...         n_units=800, act = tf.nn.relu, name='dropconnect_relu1')
>>> network = tl.layers.DropconnectDenseLayer(network, keep = 0.5,
...         n_units=800, act = tf.nn.relu, name='dropconnect_relu2')
>>> network = tl.layers.DropconnectDenseLayer(network, keep = 0.5,
...         n_units=10, act = tl.activation.identity, name='output_layer')
```

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

2.1.13 卷积层(Pro)

1D卷积层

class tensorlayer.layers.**Conv1dLayer**(*layer=None, act=<function identity>, shape=[5, 1, 5], stride=1, dilation_rate=1, padding='SAME', use_cudnn_on_gpu=None, data_format='NWC', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='cnn_layer'*)

The *Conv1dLayer* class is a 1D CNN layer, see tf.nn.convolution.

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer, [batch, in_width, in_channels].

act : activation function, None for identity.

shape : list of shape

shape of the filters, [filter_length, in_channels, out_channels].

stride : an int.

The number of entries by which the filter is moved right at each step.

dilation_rate : an int.

Specifies the filter upsampling/input downsampling rate.

padding : a string from: "SAME", "VALID".

The type of padding algorithm to use.

use_cudnn_on_gpu : An optional bool. Defaults to True.

data_format : As it is 1D conv, default is ’NWC’.

W_init : weights initializer

The initializer for initializing the weight matrix.

b_init : biases initializer or None

The initializer for initializing the bias vector. If None, skip biases.

W_init_args : dictionary

The arguments for the weights tf.get_variable().

b_init_args : dictionary

The arguments for the biases tf.get_variable().

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2D卷积层

class tensorlayer.layers.**Conv2dLayer**(*layer=None, act=<function identity>, shape=[5, 5, 1, 100], strides=[1, 1, 1, 1], padding=’SAME’, W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, use_cudnn_on_gpu=None, data_format=None, name=’cnn_layer’*)

The `Conv2dLayer` class is a 2D CNN layer, see tf.nn.conv2d.

Parameters layer : a `Layer` instance

The *Layer* class feeding into this layer.

act : activation function

The function that is applied to the layer activations.

shape : list of shape

shape of the filters, [filter_height, filter_width, in_channels, out_channels].

strides : a list of ints.

The stride of the sliding window for each dimension of input.

It Must be in the same order as the dimension specified with format.

padding : a string from: "SAME", "VALID".

The type of padding algorithm to use.

W_init : weights initializer

The initializer for initializing the weight matrix.

b_init : biases initializer or None

The initializer for initializing the bias vector. If None, skip biases.

W_init_args : dictionary

The arguments for the weights tf.get_variable().

b_init_args : dictionary

The arguments for the biases tf.get_variable().

use_cudnn_on_gpu : bool, default is None.

data_format : string "NHWC" or "NCHW", default is "NHWC"

name : a string or None

An optional name to attach to this layer.

Notes

- shape = [h, w, the number of output channel of previous layer, the number of output channels]
- the number of output channel of a layer is its last dimension.

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 28, 28, 1])
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.Conv2dLayer(network,
...                 act = tf.nn.relu,
...                 shape = [5, 5, 1, 32],  # 32 features for each 5x5 patch
...                 strides=[1, 1, 1, 1],
...                 padding='SAME',
...                 W_init=tf.truncated_normal_initializer(stddev=5e-2),
...                 W_init_args={},
...                 b_init = tf.constant_initializer(value=0.0),
...                 b_init_args = {},
...                 name ='cnn_layer1')     # output: (?, 28, 28, 32)
>>> network = tl.layers.PoolLayer(network,
...                 ksize=[1, 2, 2, 1],
...                 strides=[1, 2, 2, 1],
...                 padding='SAME',
...                 pool = tf.nn.max_pool,
...                 name ='pool_layer1',)   # output: (?, 14, 14, 32)
```

```
>>> Without TensorLayer, you can implement 2d convolution as follow.
>>> W = tf.Variable(W_init(shape=[5, 5, 1, 32], ), name='W_conv')
>>> b = tf.Variable(b_init(shape=[32], ), name='b_conv')
>>> outputs = tf.nn.relu( tf.nn.conv2d(inputs, W,
...                   strides=[1, 1, 1, 1],
...                   padding='SAME') + b )
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2D反卷积层

class `tensorlayer.layers.`**`DeConv2dLayer`**(*layer=None, act=<function identity>, shape=[3, 3, 128, 256], output_shape=[1, 256, 256, 128], strides=[1, 2, 2, 1], padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='decnn2d_layer'*)

The *`DeConv2dLayer`* class is deconvolutional 2D layer, see tf.nn.conv2d_transpose.

Parameters **layer** : a *`Layer`* instance

The *Layer* class feeding into this layer.

act : activation function

The function that is applied to the layer activations.

shape : list of shape

shape of the filters, [height, width, output_channels, in_channels], filter's in_channels dimension must match that of value.

output_shape : list of output shape

representing the output shape of the deconvolution op.

strides : a list of ints.

The stride of the sliding window for each dimension of the input tensor.

padding : a string from: "SAME", "VALID".

The type of padding algorithm to use.

W_init : weights initializer

The initializer for initializing the weight matrix.

b_init : biases initializer

The initializer for initializing the bias vector. If None, skip biases.

W_init_args : dictionary

The arguments for the weights initializer.

b_init_args : dictionary

The arguments for the biases initializer.

name : a string or None

An optional name to attach to this layer.

Notes

- shape = [h, w, the number of output channels of this layer, the number of output channel of previous layer]
- output_shape = [batch_size, any, any, the number of output channels of this layer]
- the number of output channel of a layer is its last dimension.

Examples

- A part of the generator in DCGAN example

```
>>> batch_size = 64
>>> inputs = tf.placeholder(tf.float32, [batch_size, 100], name='z_noise')
>>> net_in = tl.layers.InputLayer(inputs, name='g/in')
>>> net_h0 = tl.layers.DenseLayer(net_in, n_units = 8192,
...                             W_init = tf.random_normal_initializer(stddev=0.02),
...                             act = tf.identity, name='g/h0/lin')
>>> print(net_h0.outputs._shape)
... (64, 8192)
>>> net_h0 = tl.layers.ReshapeLayer(net_h0, shape = [-1, 4, 4, 512], name='g/h0/
↪reshape')
>>> net_h0 = tl.layers.BatchNormLayer(net_h0, act=tf.nn.relu, is_train=is_train,␣
↪name='g/h0/batch_norm')
>>> print(net_h0.outputs._shape)
... (64, 4, 4, 512)
>>> net_h1 = tl.layers.DeConv2dLayer(net_h0,
...                             shape = [5, 5, 256, 512],
...                             output_shape = [batch_size, 8, 8, 256],
...                             strides=[1, 2, 2, 1],
...                             act=tf.identity, name='g/h1/decon2d')
>>> net_h1 = tl.layers.BatchNormLayer(net_h1, act=tf.nn.relu, is_train=is_train,␣
↪name='g/h1/batch_norm')
>>> print(net_h1.outputs._shape)
... (64, 8, 8, 256)
```

- U-Net

```
>>> ....
>>> conv10 = tl.layers.Conv2dLayer(conv9, act=tf.nn.relu,
...         shape=[3,3,1024,1024], strides=[1,1,1,1], padding='SAME',
...         W_init=w_init, b_init=b_init, name='conv10')
>>> print(conv10.outputs)
... (batch_size, 32, 32, 1024)
>>> deconv1 = tl.layers.DeConv2dLayer(conv10, act=tf.nn.relu,
...          shape=[3,3,512,1024], strides=[1,2,2,1], output_shape=[batch_size,64,
↪64,512],
...          padding='SAME', W_init=w_init, b_init=b_init, name='devcon1_1')
```

Methods

`count_params()`	Return the number of parameters in the network

Continued on next page

表 2.13 – continued from previous page

print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

3D卷积层

class tensorlayer.layers.**Conv3dLayer**(*layer=None, act=<function identity>, shape=[2, 2, 2, 64, 128], strides=[1, 2, 2, 2, 1], padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='cnn3d_layer'*)

The *Conv3dLayer* class is a 3D CNN layer, see tf.nn.conv3d.

Parameters layer : a *Layer* instance

The *Layer* class feeding into this layer.

act : activation function

The function that is applied to the layer activations.

shape : list of shape

shape of the filters, [filter_depth, filter_height, filter_width, in_channels, out_channels].

strides : a list of ints. 1-D of length 4.

The stride of the sliding window for each dimension of input. Must be in the same order as the dimension specified with format.

padding : a string from: "SAME", "VALID".

The type of padding algorithm to use.

W_init : weights initializer

The initializer for initializing the weight matrix.

b_init : biases initializer

The initializer for initializing the bias vector.

W_init_args : dictionary

The arguments for the weights initializer.

b_init_args : dictionary

The arguments for the biases initializer.

name : a string or None

An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

3D反卷积层

class tensorlayer.layers.**DeConv3dLayer**(*layer=None, act=<function identity>, shape=[2, 2, 2, 128, 256], output_shape=[1, 12, 32, 32, 128], strides=[1, 2, 2, 2, 1], padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='decnn3d_layer'*)

The `DeConv3dLayer` class is deconvolutional 3D layer, see tf.nn.conv3d_transpose.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

act : activation function

The function that is applied to the layer activations.

shape : list of shape

shape of the filters, [depth, height, width, output_channels, in_channels], filter's in_channels dimension must match that of value.

output_shape : list of output shape

representing the output shape of the deconvolution op.

strides : a list of ints.

The stride of the sliding window for each dimension of the input tensor.

padding : a string from: "SAME", "VALID".

The type of padding algorithm to use.

W_init : weights initializer

The initializer for initializing the weight matrix.

b_init : biases initializer

The initializer for initializing the bias vector.

W_init_args : dictionary

The arguments for the weights initializer.

b_init_args : dictionary

The arguments for the biases initializer.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

2D上采样层

class tensorlayer.layers.**UpSampling2dLayer**(*layer=None*, *size=[]*, *is_scale=True*, *method=0*, *align_corners=False*, *name='upsample2d_layer'*)

The *UpSampling2dLayer* class is upSampling 2d layer, see tf.image.resize_images.

Parameters **layer** : a layer class with 4-D Tensor of shape [batch, height, width, channels] or 3-D Tensor of shape [height, width, channels].

size : a tuple of int or float.

(height, width) scale factor or new size of height and width.

is_scale : boolean, if True (default), size is scale factor, otherwise, size is number of pixels of height and width.

method : 0, 1, 2, 3. ResizeMethod. Defaults to ResizeMethod.BILINEAR.

- ResizeMethod.BILINEAR, Bilinear interpolation.
- ResizeMethod.NEAREST_NEIGHBOR, Nearest neighbor interpolation.
- ResizeMethod.BICUBIC, Bicubic interpolation.
- ResizeMethod.AREA, Area interpolation.

align_corners : bool. If true, exactly align all 4 corners of the input and output. Defaults to false.

name : a string or None

An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

2D下采样层

class tensorlayer.layers.**DownSampling2dLayer**(*layer=None*, *size=[]*, *is_scale=True*, *method=0*, *align_corners=False*, *name='downsample2d_layer'*)

The *DownSampling2dLayer* class is downSampling 2d layer, see tf.image.resize_images.

Parameters **layer** : a layer class with 4-D Tensor of shape [batch, height, width, channels] or 3-D Tensor of shape [height, width, channels].

size : a tupe of int or float.

(height, width) scale factor or new size of height and width.

is_scale : boolean, if True (default), size is scale factor, otherwise, size is number of pixels of height and width.

method : 0, 1, 2, 3. ResizeMethod. Defaults to ResizeMethod.BILINEAR.

- ResizeMethod.BILINEAR, Bilinear interpolation.

- ResizeMethod.NEAREST_NEIGHBOR, Nearest neighbor interpolation.
- ResizeMethod.BICUBIC, Bicubic interpolation.
- ResizeMethod.AREA, Area interpolation.

align_corners : bool. If true, exactly align all 4 corners of the input and output. Defaults to false.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

1D多孔卷积层

`tensorlayer.layers.`**`AtrousConv1dLayer`**(*net, n_filter=32, filter_size=2, stride=1, dilation=1, act=None, padding='SAME', use_cudnn_on_gpu=None, data_format='NWC', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='conv1d'*)

Wrapper for `AtrousConv1dLayer`, if you don't understand how to use `Conv1dLayer`, this function may be easier.

Parameters **net** : TensorLayer layer.

n_filter : number of filter.

filter_size : an int.

stride : an int.

dilation : an int, filter dilation size.

act : None or activation function.

others : see `Conv1dLayer`.

2D多孔卷积层

`class` `tensorlayer.layers.`**`AtrousConv2dLayer`**(*layer=None, n_filter=32, filter_size=(3, 3), rate=2, act=None, padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='atrou2d'*)

The `AtrousConv2dLayer` class is Atrous convolution (a.k.a. convolution with holes or dilated convolution) 2D layer, see tf.nn.atrous_conv2d.

Parameters **layer** : a layer class with 4-D Tensor of shape [batch, height, width, channels].

filters : A 4-D Tensor with the same type as value and shape [filter_height, filter_width, in_channels, out_channels]. filters' in_channels dimension must match that of value. Atrous convolution is equivalent to standard convolution with upsampled filters with effective height filter_height + (filter_height - 1) * (rate - 1) and effective width filter_width + (filter_width - 1) * (rate - 1), produced by inserting rate - 1 zeros along consecutive elements across the filters' spatial dimensions.

n_filter : number of filter.

filter_size : tuple (height, width) for filter size.

rate : A positive int32. The stride with which we sample input values across the height and width dimensions. Equivalently, the rate by which we upsample the filter values by inserting zeros across the height and width dimensions. In the literature, the same parameter is sometimes called input stride or dilation.

act : activation function, None for linear.

padding : A string, either 'VALID' or 'SAME'. The padding algorithm.

W_init : weights initializer. The initializer for initializing the weight matrix.

b_init : biases initializer or None. The initializer for initializing the bias vector. If None, skip biases.

W_init_args : dictionary. The arguments for the weights tf.get_variable().

b_init_args : dictionary. The arguments for the biases tf.get_variable().

name : a string or None, an optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2D Separable卷积层

class tensorlayer.layers.**SeparableConv2dLayer**(*layer=None, filters=None, kernel_size=5, strides=(1, 1), padding='valid', data_format='channels_last', dilation_rate=(1, 1), depth_multiplier=1, act=None, use_bias=True, depthwise_initializer=None, pointwise_initializer=None, bias_initializer=<class 'tensorflow.python.ops.init_ops.Zeros'>, depthwise_regularizer=None, pointwise_regularizer=None, bias_regularizer=None, activity_regularizer=None, name='atrou2d'*)

The `SeparableConv2dLayer` class is 2-D convolution with separable filters, see tf.layers.separable_conv2d.

Parameters **layer** : a layer class

filters : integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).

kernel_size : a tuple or list of N positive integers specifying the spatial dimensions of of the filters. Can be a single integer to specify the same value for all spatial dimensions.

strides : a tuple or list of N positive integers 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 : one of "valid" or "same" (case-insensitive).

data_format : A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shapedata_format = 'NWHC' (batch, width, height, channels) while channels_first corresponds to inputs with shape (batch, channels, width, height).

dilation_rate : an integer or tuple/list of 2 integers, 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 : 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.

act (activation) : Activation function. Set it to None to maintain a linear activation.

use_bias : Boolean, whether the layer uses a bias.

depthwise_initializer : An initializer for the depthwise convolution kernel.

pointwise_initializer : An initializer for the pointwise convolution kernel.

bias_initializer : An initializer for the bias vector. If None, no bias will be applied.

depthwise_regularizer : Optional regularizer for the depthwise convolution kernel.

pointwise_regularizer : Optional regularizer for the pointwise convolution kernel.

bias_regularizer : Optional regularizer for the bias vector.

activity_regularizer : Regularizer function for the output.

name : a string or None, an optional name to attach to this layer.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

2.1.14 卷积层(Simplified)

对于不擅长 TensorFlow 的用户，下面的简化的函数使用起来更简单。接下来我们将添加更多简化函数。

1D卷积层

`tensorlayer.layers.Conv1d`(*net*, *n_filter=32*, *filter_size=5*, *stride=1*, *dilation_rate=1*, *act=None*, *padding='SAME'*, *use_cudnn_on_gpu=None*, *data_format='NWC'*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='conv1d'*)

Wrapper for `Conv1dLayer`, if you don't understand how to use `Conv1dLayer`, this function may be easier.

Parameters **net** : TensorLayer layer.

n_filter : number of filter.

filter_size : an int.

stride : an int.

dilation_rate : As it is 1D conv, the default is "NWC".

act : None or activation function.

others : see `Conv1dLayer`.

Examples

```
>>> x = tf.placeholder(tf.float32, [batch_size, width])
>>> y_ = tf.placeholder(tf.int64, shape=[batch_size,])
>>> n = InputLayer(x, name='in')
>>> n = ReshapeLayer(n, [-1, width, 1], name='rs')
>>> n = Conv1d(n, 64, 3, 1, act=tf.nn.relu, name='c1')
>>> n = MaxPool1d(n, 2, 2, padding='valid', name='m1')
>>> n = Conv1d(n, 128, 3, 1, act=tf.nn.relu, name='c2')
>>> n = MaxPool1d(n, 2, 2, padding='valid', name='m2')
>>> n = Conv1d(n, 128, 3, 1, act=tf.nn.relu, name='c3')
>>> n = MaxPool1d(n, 2, 2, padding='valid', name='m3')
>>> n = FlattenLayer(n, name='f')
>>> n = DenseLayer(n, 500, tf.nn.relu, name='d1')
>>> n = DenseLayer(n, 100, tf.nn.relu, name='d2')
>>> n = DenseLayer(n, 2, tf.identity, name='o')
```

2D卷积层

`tensorlayer.layers.Conv2d`(*net*, *n_filter=32*, *filter_size=(3, 3)*, *strides=(1, 1)*, *act=None*, *padding='SAME'*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *use_cudnn_on_gpu=None*, *data_format=None*, *name='conv2d'*)

Wrapper for `Conv2dLayer`, if you don't understand how to use `Conv2dLayer`, this function may be easier.

Parameters **net** : TensorLayer layer.

n_filter : number of filter.

filter_size : tuple (height, width) for filter size.

strides : tuple (height, width) for strides.

act : None or activation function.

others : see `Conv2dLayer`.

Examples

```
>>> w_init = tf.truncated_normal_initializer(stddev=0.01)
>>> b_init = tf.constant_initializer(value=0.0)
>>> inputs = InputLayer(x, name='inputs')
>>> conv1 = Conv2d(inputs, 64, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv1_1')
>>> conv1 = Conv2d(conv1, 64, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv1_2')
>>> pool1 = MaxPool2d(conv1, (2, 2), padding='SAME', name='pool1')
>>> conv2 = Conv2d(pool1, 128, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv2_1')
>>> conv2 = Conv2d(conv2, 128, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv2_2')
>>> pool2 = MaxPool2d(conv2, (2, 2), padding='SAME', name='pool2')
```

2D反卷积层

`tensorlayer.layers.`**`DeConv2d`**(*net, n_out_channel=32, filter_size=(3, 3), out_size=(30, 30), strides=(2, 2), padding='SAME', batch_size=None, act=None, W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='decnn2d'*)

Wrapper for `DeConv2dLayer`, if you don't understand how to use `DeConv2dLayer`, this function may be easier.

Parameters **net** : TensorLayer layer.

n_out_channel : int, number of output channel.

filter_size : tuple of (height, width) for filter size.

out_size : tuple of (height, width) of output.

batch_size : int or None, batch_size. If None, try to find the batch_size from the first dim of net.outputs (you should tell the batch_size when define the input placeholder).

strides : tuple of (height, width) for strides.

act : None or activation function.

others : see `DeConv2dLayer`.

1D Max池化层

`tensorlayer.layers.`**`MaxPool1d`**(*net, filter_size, strides, padding='valid', data_format='channels_last', name=None*)

Wrapper for tf.layers.max_pooling1d .

Parameters **net** : TensorLayer layer, the tensor over which to pool. Must have rank 3.

filter_size (pool_size) : An integer or tuple/list of a single integer, representing the size of the pooling window.

strides : An integer or tuple/list of a single integer, specifying the strides of the pooling operation.

padding : A string. The padding method, either 'valid' or 'same'. Case-insensitive.

data_format : A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, length, channels) while channels_first corresponds to inputs with shape (batch, channels, length).

name : A string, the name of the layer.

Returns

- A `Layer` which the output tensor, of rank 3.

1D Mean池化层

`tensorlayer.layers.`**`MeanPool1d`**(*net*, *filter_size*, *strides*, *padding='valid'*, *data_format='channels_last'*, *name=None*)

Wrapper for tf.layers.average_pooling1d .

Parameters **net** : TensorLayer layer, the tensor over which to pool. Must have rank 3.

filter_size (pool_size) : An integer or tuple/list of a single integer, representing the size of the pooling window.

strides : An integer or tuple/list of a single integer, specifying the strides of the pooling operation.

padding : A string. The padding method, either 'valid' or 'same'. Case-insensitive.

data_format : A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, length, channels) while channels_first corresponds to inputs with shape (batch, channels, length).

name : A string, the name of the layer.

Returns

- A `Layer` which the output tensor, of rank 3.

2D Max池化层

`tensorlayer.layers.`**`MaxPool2d`**(*net*, *filter_size=(2, 2)*, *strides=None*, *padding='SAME'*, *name='maxpool'*)

Wrapper for `PoolLayer`.

Parameters **net** : TensorLayer layer.

filter_size : tuple of (height, width) for filter size.

strides : tuple of (height, width). Default is the same with filter_size.

others : see `PoolLayer`.

2D Mean池化层

`tensorlayer.layers.`**`MeanPool2d`**(*net*, *filter_size=(2, 2)*, *strides=None*, *padding='SAME'*, *name='meanpool'*)

Wrapper for `PoolLayer`.

Parameters **net** : TensorLayer layer.

filter_size : tuple of (height, width) for filter size.

strides : tuple of (height, width). Default is the same with filter_size.

others : see `PoolLayer`.

3D Max池化层

`tensorlayer.layers.`**`MaxPool3d`**(*net*, *filter_size*, *strides*, *padding='valid'*, *data_format='channels_last'*, *name=None*)

Wrapper for tf.layers.max_pooling3d .

Parameters **net** : TensorLayer layer, the tensor over which to pool. Must have rank 5.

filter_size (pool_size) : An integer or tuple/list of 3 integers: (pool_depth, pool_height, pool_width) specifying the size of the pooling window. Can be a single integer to specify the same value for all spatial dimensions.

strides : An integer or tuple/list of 3 integers, specifying the strides of the pooling operation. Can be a single integer to specify the same value for all spatial dimensions.

padding : A string. The padding method, either 'valid' or 'same'. Case-insensitive.

data_format : A string. The ordering of the dimensions in the inputs. channels_last (default) and channels_first are supported. channels_last corresponds to inputs with shape (batch, depth, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, depth, height, width).

name : A string, the name of the layer.

3D Mean池化层

`tensorlayer.layers.`**`MeanPool3d`**(*net*, *filter_size*, *strides*, *padding='valid'*, *data_format='channels_last'*, *name=None*)

Wrapper for tf.layers.average_pooling3d

Parameters **net** : TensorLayer layer, the tensor over which to pool. Must have rank 5.

filter_size (pool_size) : An integer or tuple/list of 3 integers: (pool_depth, pool_height, pool_width) specifying the size of the pooling window. Can be a single integer to specify the same value for all spatial dimensions.

strides : An integer or tuple/list of 3 integers, specifying the strides of the pooling operation. Can be a single integer to specify the same value for all spatial dimensions.

padding : A string. The padding method, either 'valid' or 'same'. Case-insensitive.

data_format : A string. The ordering of the dimensions in the inputs. channels_last (default) and channels_first are supported. channels_last corresponds to inputs with shape (batch, depth, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, depth, height, width).

name : A string, the name of the layer.

2.1.15 Super-Resolution 层

1D 子像素卷积层

`tensorlayer.layers.`**`SubpixelConv1d`**`(`*net*, *scale=2*, *act=<function identity>*, *name='subpixel_conv1d'*`)`

One-dimensional subpixel upsampling layer. Calls a tensorflow function that directly implements this functionality. We assume input has dim (batch, width, r)

Parameters **net** : TensorLayer layer.

scale : int, upscaling ratio, a wrong setting will lead to Dimension size error.

act : activation function.

name : string.

An optional name to attach to this layer.

References

- Audio Super Resolution Implementation.

Examples

```
>>> t_signal = tf.placeholder('float32', [10, 100, 4], name='x')
>>> n = InputLayer(t_signal, name='in')
>>> n = SubpixelConv1d(n, scale=2, name='s')
>>> print(n.outputs.shape)
... (10, 200, 2)
```

2D 子像素卷积层

`tensorlayer.layers.`**`SubpixelConv2d`**`(`*net*, *scale=2*, *n_out_channel=None*, *act=<function identity>*, *name='subpixel_conv2d'*`)`

The *`SubpixelConv2d`* class is a sub-pixel 2d convolutional ayer, usually be used for Super-Resolution applications, example code.

Parameters **net** : TensorLayer layer.

scale : int, upscaling ratio, a wrong setting will lead to Dimension size error.

n_out_channel : int or None, the number of output channels.

Note that, the number of input channels == (scale x scale) x The number of output channels. If None, automatically set n_out_channel == the number of input channels / (scale x scale).

act : activation function.

name : string.

An optional name to attach to this layer.

References

- Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network

Examples

```
>>> # examples here just want to tell you how to set the n_out_channel.
>>> x = np.random.rand(2, 16, 16, 4)
>>> X = tf.placeholder("float32", shape=(2, 16, 16, 4), name="X")
>>> net = InputLayer(X, name='input')
>>> net = SubpixelConv2d(net, scale=2, n_out_channel=1, name='subpixel_conv2d')
>>> y = sess.run(net.outputs, feed_dict={X: x})
>>> print(x.shape, y.shape)
... (2, 16, 16, 4) (2, 32, 32, 1)
>>>
>>> x = np.random.rand(2, 16, 16, 4*10)
>>> X = tf.placeholder("float32", shape=(2, 16, 16, 4*10), name="X")
>>> net = InputLayer(X, name='input2')
>>> net = SubpixelConv2d(net, scale=2, n_out_channel=10, name='subpixel_conv2d2')
>>> y = sess.run(net.outputs, feed_dict={X: x})
>>> print(x.shape, y.shape)
... (2, 16, 16, 40) (2, 32, 32, 10)
>>>
>>> x = np.random.rand(2, 16, 16, 25*10)
>>> X = tf.placeholder("float32", shape=(2, 16, 16, 25*10), name="X")
>>> net = InputLayer(X, name='input3')
>>> net = SubpixelConv2d(net, scale=5, n_out_channel=None, name='subpixel_conv2d3
↪')
>>> y = sess.run(net.outputs, feed_dict={X: x})
>>> print(x.shape, y.shape)
... (2, 16, 16, 250) (2, 80, 80, 10)
```

2.1.16 空间变换网络

2D 仿射变换层

class tensorlayer.layers.**SpatialTransformer2dAffineLayer**(*layer=None, theta_layer=None, out_size=[40, 40], name='sapatial_trans_2d_affine'*)

The *SpatialTransformer2dAffineLayer* class is a Spatial Transformer Layer for 2D Affine Transformation.

Parameters **layer** : a layer class with 4-D Tensor of shape [batch, height, width, channels]

theta_layer : a layer class for the localisation network.

In this layer, we will use a *DenseLayer* to make the theta size to [batch, 6], value range to [0, 1] (via tanh).

out_size : tuple of two ints.

The size of the output of the network (height, width), the feature maps will be resized by this.

References

- Spatial Transformer Networks
- TensorFlow/Models

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2D 仿射变换函数

`tensorlayer.layers.`**`transformer`**(*U*, *theta*, *out_size*, *name='SpatialTransformer2dAffine'*, ***kwargs*)

Spatial Transformer Layer for 2D Affine Transformation , see *`SpatialTransformer2dAffineLayer`* class.

Parameters U : 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 two ints

The size of the output of the network (height, width)

Notes

- To initialize the network to the identity transform init.

```
>>> ``theta`` to
>>> identity = np.array([[1., 0., 0.],
...                      [0., 1., 0.]])
>>> identity = identity.flatten()
>>> theta = tf.Variable(initial_value=identity)
```

References

- Spatial Transformer Networks
- TensorFlow/Models

批 2D 仿射变换函数

`tensorlayer.layers.`**`batch_transformer`**(*U*, *thetas*, *out_size*, *name='BatchSpatialTransformer2dAffine'*)

Batch Spatial Transformer function for 2D Affine Transformation.

Parameters **U** : float

tensor of inputs [batch, height, width, num_channels]

thetas : float

a set of transformations for each input [batch, num_transforms, 6]

out_size : int

the size of the output [out_height, out_width]

Returns: float

Tensor of size [batch * num_transforms, out_height, out_width, num_channels]

2.1.17 池化层

该池化层可以实现各种纬度（1D，2D，3D）以及各种池化方法（Mean，Max）。

class tensorlayer.layers.**PoolLayer**(*layer=None, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', pool=<function max_pool>, name='pool_layer'*)

The `PoolLayer` class is a Pooling layer, you can choose `tf.nn.max_pool` and `tf.nn.avg_pool` for 2D or `tf.nn.max_pool3d` and `tf.nn.avg_pool3d` for 3D.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

ksize : a list of ints that has length >= 4.

The size of the window for each dimension of the input tensor.

strides : a list of ints that has length >= 4.

The stride of the sliding window for each dimension of the input tensor.

padding : a string from: "SAME", "VALID".

The type of padding algorithm to use.

pool : a pooling function

- see TensorFlow pooling APIs
- class `tf.nn.max_pool`
- class `tf.nn.avg_pool`
- class `tf.nn.max_pool3d`
- class `tf.nn.avg_pool3d`

name : a string or None

An optional name to attach to this layer.

Examples

- see `Conv2dLayer`.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.18 填充层

该填充层可以实现任意模式的填充。

class `tensorlayer.layers.`**`PadLayer`**(*layer=None*, *paddings=None*, *mode='CONSTANT'*, *name='pad_layer'*)

The `PadLayer` class is a Padding layer for any modes and dimensions. Please see tf.pad for usage.

Parameters layer : a `Layer` instance

The *Layer* class feeding into this layer.

padding : a Tensor of type int32.

mode : one of "CONSTANT", "REFLECT", or "SYMMETRIC" (case-insensitive)

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.19 规范化层

Local Response Normalization 不包含任何参数，也没有复杂的设置。您也可以在 `network.outputs` 上使用 `tf.nn.lrn()` 来实现之。

Batch Normalization

class `tensorlayer.layers.`**`BatchNormLayer`**(*layer=None*, *decay=0.9*, *epsilon=1e-05*, *act=<function identity>*, *is_train=False*, *beta_init=<class 'tensorflow.python.ops.init_ops.Zeros'>*, *gamma_init=<tensorflow.python.ops.init_ops.RandomNormal object>*, *name='batchnorm_layer'*)

The `BatchNormLayer` class is a normalization layer, see `tf.nn.batch_normalization` and `tf.nn.moments`.

Batch normalization on fully-connected or convolutional maps.

Parameters layer : a `Layer` instance

The *Layer* class feeding into this layer.

decay : float, default is 0.9.

A decay factor for ExponentialMovingAverage, use larger value for large dataset.

epsilon : float

A small float number to avoid dividing by 0.

act : activation function.

is_train : boolean

Whether train or inference.

beta_init : beta initializer

The initializer for initializing beta

gamma_init : gamma initializer

The initializer for initializing gamma

name : a string or None

An optional name to attach to this layer.

References

- Source
- stackoverflow

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

Local Response Normalization

class `tensorlayer.layers.`**`LocalResponseNormLayer`**(*layer=None*, *depth_radius=None*, *bias=None*, *alpha=None*, *beta=None*, *name='lrn_layer'*)

The *`LocalResponseNormLayer`* class 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 treated as 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, squared sum of inputs within depth_radius.

Parameters **layer** : a layer class. Must be one of the following types: float32, half. 4-D.

depth_radius : An optional int. Defaults to 5. 0-D. Half-width of the 1-D normalization window.

bias : An optional float. Defaults to 1. An offset (usually positive to avoid dividing by 0).

alpha : An optional float. Defaults to 1. A scale factor, usually positive.

beta : An optional float. Defaults to 0.5. An exponent.

name : A string or None, an optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

Instance Normalization

class tensorlayer.layers.**InstanceNormLayer**(*layer=None, act=<function identity>, epsilon=1e-05, scale_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, offset_init=<tensorflow.python.ops.init_ops.Constant object>, name='instan_norm'*)

The `InstanceNormLayer` class is a for instance normalization.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

act : activation function.

epsilon : float

A small float number.

scale_init : beta initializer

The initializer for initializing beta

offset_init : gamma initializer

The initializer for initializing gamma

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

Layer Normalization

class tensorlayer.layers.**LayerNormLayer**(*layer=None, center=True, scale=True, act=<function identity>, reuse=None, variables_collections=None, outputs_collections=None, trainable=True, begin_norm_axis=1, begin_params_axis=-1, name='layernorm'*)

The `LayerNormLayer` class is for layer normalization, see tf.contrib.layers.layer_norm.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

act : activation function

The function that is applied to the layer activations.

others : see tf.contrib.layers.layer_norm

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

2.1.20 物体检测

ROI 层

class `tensorlayer.layers.`**`ROIPoolingLayer`**(*layer=None*, *rois=None*, *pool_height=2*, *pool_width=2*, *name='roipooling_layer'*)

The *`ROIPoolingLayer`* class is Region of interest pooling layer.

Parameters **layer** : a *`Layer`* instance

The *Layer* class feeding into this layer, the feature maps on which to perform the pooling operation

rois : list of regions of interest in the format (feature map index, upper left, bottom right)

pool_width : int, size of the pooling sections.

pool_width : int, size of the pooling sections.

Notes

- This implementation is from Deepsense-AI .
- Please install it by the instruction HERE.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

2.1.21 TimeDistributed 包装器

class `tensorlayer.layers.`**`TimeDistributedLayer`**(*layer=None*, *layer_class=None*, *args={}*, *name='time_distributed'*)

The *`TimeDistributedLayer`* class that applies a function to every timestep of the input tensor. For example, if using *`DenseLayer`* as the `layer_class`, inputs [batch_size , length, dim] outputs [batch_size , length, new_dim].

Parameters **layer** : a *`Layer`* instance

The *Layer* class feeding into this layer, [batch_size , length, dim]

layer_class : a *Layer* class

args : dictionary

The arguments for the `layer_class`.

name : a string or None

An optional name to attach to this layer.

Examples

```
>>> batch_size = 32
>>> timestep = 20
>>> input_dim = 100
>>> x = tf.placeholder(dtype=tf.float32, shape=[batch_size, timestep,  input_dim],
↪ name="encode_seqs")
>>> net = InputLayer(x, name='input')
>>> net = TimeDistributedLayer(net, layer_class=DenseLayer, args={'n_units':50,
↪'name':'dense'}, name='time_dense')
... [TL] InputLayer  input: (32, 20, 100)
... [TL] TimeDistributedLayer time_dense: layer_class:DenseLayer
>>> print(net.outputs._shape)
... (32, 20, 50)
>>> net.print_params(False)
... param   0: (100, 50)          time_dense/dense/W:0
... param   1: (50,)              time_dense/dense/b:0
... num of params: 5050
```

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

2.1.22 定长递归层

所有递归层可实现任意 RNN 内核，只需输入不同的 cell 函数。

递归层

class `tensorlayer.layers.`**`RNNLayer`**(*layer=None, cell_fn=None, cell_init_args={}, n_hidden=100, initializer=<tensorflow.python.ops.init_ops.RandomUniform object>, n_steps=5, initial_state=None, return_last=False, return_seq_2d=False, name='rnn_layer'*)

The *RNNLayer* class is a RNN layer, you can implement vanilla RNN, LSTM and GRU with it.

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer.

cell_fn : a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).

- see RNN Cells in TensorFlow

cell_init_args : a dictionary

The arguments for the cell initializer.

n_hidden : an int

The number of hidden units in the layer.

initializer : initializer

The initializer for initializing the parameters.

n_steps : an int

The sequence length.

initial_state : None or RNN State

If None, initial_state is zero_state.

return_last : boolean

- 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 apply one or more RNN(s) on this layer, set to False.

return_seq_2d : boolean

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer after it.
- If False, return 3D Tensor [n_example/n_steps, n_steps, n_hidden], for stacking multiple RNN after it.

name : a string or None

An optional name to attach to this layer.

Notes

Input dimension should be rank 3 : [batch_size, n_steps, n_features], if no, please see `ReshapeLayer`.

References

- Neural Network RNN Cells in TensorFlow
- tensorflow/python/ops/rnn.py
- tensorflow/python/ops/rnn_cell.py
- see TensorFlow tutorial `ptb_word_lm.py`, TensorLayer tutorials `tutorial_ptb_lstm*.py` and `tutorial_generate_text.py`

Examples

- For words

```
>>> input_data = tf.placeholder(tf.int32, [batch_size, num_steps])
>>> net = tl.layers.EmbeddingInputlayer(
...                 inputs = input_data,
...                 vocabulary_size = vocab_size,
...                 embedding_size = hidden_size,
...                 E_init = tf.random_uniform_initializer(-init_scale, init_
↪scale),
...                 name ='embedding_layer')
>>> net = tl.layers.DropoutLayer(net, keep=keep_prob, is_fix=True, is_train=is_
↪train, name='drop1')
>>> net = tl.layers.RNNLayer(net,
...             cell_fn=tf.contrib.rnn.BasicLSTMCell,
...             cell_init_args={'forget_bias': 0.0},# 'state_is_tuple': True},
...             n_hidden=hidden_size,
...             initializer=tf.random_uniform_initializer(-init_scale, init_
↪scale),
...             n_steps=num_steps,
...             return_last=False,
...             name='basic_lstm_layer1')
>>> lstm1 = net
>>> net = tl.layers.DropoutLayer(net, keep=keep_prob, is_fix=True, is_train=is_
↪train, name='drop2')
>>> net = tl.layers.RNNLayer(net,
...             cell_fn=tf.contrib.rnn.BasicLSTMCell,
...             cell_init_args={'forget_bias': 0.0}, # 'state_is_tuple': True},
...             n_hidden=hidden_size,
...             initializer=tf.random_uniform_initializer(-init_scale, init_
↪scale),
...             n_steps=num_steps,
...             return_last=False,
...             return_seq_2d=True,
...             name='basic_lstm_layer2')
>>> lstm2 = net
>>> net = tl.layers.DropoutLayer(net, keep=keep_prob, is_fix=True, is_train=is_
↪train, name='drop3')
>>> net = tl.layers.DenseLayer(net,
...             n_units=vocab_size,
...             W_init=tf.random_uniform_initializer(-init_scale, init_scale),
...             b_init=tf.random_uniform_initializer(-init_scale, init_scale),
...             act = tl.activation.identity, name='output_layer')
```

- For CNN+LSTM

```
>>> x = tf.placeholder(tf.float32, shape=[batch_size, image_size, image_size, 1])
>>> net = tl.layers.InputLayer(x, name='input_layer')
>>> net = tl.layers.Conv2dLayer(net,
...                         act = tf.nn.relu,
...                         shape = [5, 5, 1, 32],  # 32 features for each 5x5␣
↪patch
...                         strides=[1, 2, 2, 1],
...                         padding='SAME',
...                         name ='cnn_layer1')
```

```
>>> net = tl.layers.PoolLayer(net,
...                         ksize=[1, 2, 2, 1],
...                         strides=[1, 2, 2, 1],
...                         padding='SAME',
...                         pool = tf.nn.max_pool,
...                         name ='pool_layer1')
>>> net = tl.layers.Conv2dLayer(net,
...                         act = tf.nn.relu,
...                         shape = [5, 5, 32, 10], # 10 features for each 5x5␣
↪patch
                            strides=[1, 2, 2, 1],
...                         padding='SAME',
...                         name ='cnn_layer2')
>>> net = tl.layers.PoolLayer(net,
...                         ksize=[1, 2, 2, 1],
...                         strides=[1, 2, 2, 1],
...                         padding='SAME',
...                         pool = tf.nn.max_pool,
...                         name ='pool_layer2')
>>> net = tl.layers.FlattenLayer(net, name='flatten_layer')
>>> net = tl.layers.ReshapeLayer(net, shape=[-1, num_steps, int(net.outputs._
↪shape[-1])])
>>> rnn1 = tl.layers.RNNLayer(net,
...                         cell_fn=tf.nn.rnn_cell.LSTMCell,
...                         cell_init_args={},
...                         n_hidden=200,
...                         initializer=tf.random_uniform_initializer(-0.1, 0.1),
...                         n_steps=num_steps,
...                         return_last=False,
...                         return_seq_2d=True,
...                         name='rnn_layer')
>>> net = tl.layers.DenseLayer(rnn1, n_units=3,
...                         act = tl.activation.identity, name='output_layer')
```

Attributes

outputs	(a tensor) The output of this RNN. return_last = False, outputs = all cell_output, which is the hidden state. cell_output.get_shape() = (?, n_hidden)
final_state	(a tensor or StateTuple) When state_is_tuple = False, it is the final hidden and cell states, states.get_shape() = [?, 2 * n_hidden]. When state_is_tuple = True, it stores two elements: (c, h), in that order. You can get the final state after each iteration during training, then feed it to the initial state of next iteration.
initial_state	(a tensor or StateTuple) It is the initial state of this RNN layer, you can use it to initialize your state at the begining of each epoch or iteration according to your training procedure.
batch_size	(int or tensor) Is int, if able to compute the batch_size, otherwise, tensor for `?`.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

双向递归层

class tensorlayer.layers.**BiRNNLayer**(*layer=None, cell_fn=None, cell_init_args={'state_is_tuple': True, 'use_peepholes': True}, n_hidden=100, initializer=<tensorflow.python.ops.init_ops.RandomUniform object>, n_steps=5, fw_initial_state=None, bw_initial_state=None, dropout=None, n_layer=1, return_last=False, return_seq_2d=False, name='birnn_layer'*)

The `BiRNNLayer` class is a Bidirectional RNN layer.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

cell_fn : a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).

- see RNN Cells in TensorFlow

cell_init_args : a dictionary

The arguments for the cell initializer.

n_hidden : an int

The number of hidden units in the layer.

initializer : initializer

The initializer for initializing the parameters.

n_steps : an int

The sequence length.

fw_initial_state : None or forward RNN State

If None, initial_state is zero_state.

bw_initial_state : None or backward RNN State

If None, initial_state is zero_state.

dropout : *tuple* of *float*: (input_keep_prob, output_keep_prob).

The input and output keep probability.

n_layer : an int, default is 1.

The number of RNN layers.

return_last : boolean

- 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 apply one or more RNN(s) on this layer, set to False.

return_seq_2d : boolean

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer after it.
- If False, return 3D Tensor [n_example/n_steps, n_steps, n_hidden], for stacking multiple RNN after it.

name : a string or None

An optional name to attach to this layer.

Notes

- Input dimension should be rank 3 : [batch_size, n_steps, n_features], if no, please see `ReshapeLayer`.
- For predicting, the sequence length has to be the same with the sequence length of training, while, for normal

RNN, we can use sequence length of 1 for predicting.

References

- Source

Attributes

outputs	(a tensor) The output of this RNN. return_last = False, outputs = all cell_output, which is the hidden state. cell_output.get_shape() = (?, n_hidden)
fw(bw)_final_state	(a tensor or StateTuple) When state_is_tuple = False, it is the final hidden and cell states, states.get_shape() = [?, 2 * n_hidden]. When state_is_tuple = True, it stores two elements: (c, h), in that order. You can get the final state after each iteration during training, then feed it to the initial state of next iteration.
fw(bw)_initial_state	(a tensor or StateTuple) It is the initial state of this RNN layer, you can use it to initialize your state at the begining of each epoch or iteration according to your training procedure.
batch_size	(int or tensor) Is int, if able to compute the batch_size, otherwise, tensor for `?`.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network

2.1.23 动态递归的高级 Ops 函数

这些函数通常在使用 Dynamic RNN layer 时使用，他们用以计算不同情况下的 sequence length，以及已知 sequence length 时对 输出进行索引。

输出索引

用以得到 last output。

`tensorlayer.layers.`**`advanced_indexing_op`**(*input*, *index*)

Advanced Indexing for Sequences, returns the outputs by given sequence lengths. When return the last output `DynamicRNNLayer` uses it to get the last outputs with the sequence lengths.

Parameters **input** : tensor for data

[batch_size, n_step(max), n_features]

index : tensor for indexing, i.e. sequence_length in Dynamic RNN.

[batch_size]

References

- Modified from TFlearn (the original code is used for fixed length rnn), references.

Examples

```
>>> batch_size, max_length, n_features = 3, 5, 2
>>> z = np.random.uniform(low=-1, high=1, size=[batch_size, max_length, n_
↪features]).astype(np.float32)
>>> b_z = tf.constant(z)
>>> sl = tf.placeholder(dtype=tf.int32, shape=[batch_size])
>>> o = advanced_indexing_op(b_z, sl)
>>>
>>> sess = tf.InteractiveSession()
>>> tl.layers.initialize_global_variables(sess)
>>>
>>> order = np.asarray([1,1,2])
>>> print("real",z[0][order[0]-1], z[1][order[1]-1], z[2][order[2]-1])
>>> y = sess.run([o], feed_dict={sl:order})
>>> print("given",order)
>>> print("out", y)
... real [-0.93021595  0.53820813] [-0.92548317 -0.77135968] [ 0.89952248  0.
↪19149846]
... given [1 1 2]
... out [array([[-0.93021595,  0.53820813],
...             [-0.92548317, -0.77135968],
...             [ 0.89952248,  0.19149846]], dtype=float32)]
```

计算 Sequence length 方法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.

References

- Borrow from TFlearn.

Examples

```
>>> data = [[[1],[2],[0],[0],[0]],
...          [[1],[2],[3],[0],[0]],
...          [[1],[2],[6],[1],[0]]]
>>> data = np.asarray(data)
>>> print(data.shape)
... (3, 5, 1)
>>> data = tf.constant(data)
>>> sl = retrieve_seq_length_op(data)
>>> sess = tf.InteractiveSession()
>>> tl.layers.initialize_global_variables(sess)
>>> y = sl.eval()
... [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]]]
>>> sl
... [4 3 4]
```

计算 Sequence length 方法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]]
>>> o = retrieve_seq_length_op2(data)
>>> sess = tf.InteractiveSession()
>>> tl.layers.initialize_global_variables(sess)
>>> print(o.eval())
... [2 3 4]
```

2.1.24 动态递归层

class tensorlayer.layers.**DynamicRNNLayer**(*layer=None, cell_fn=None, cell_init_args={'state_is_tuple': True}, n_hidden=256, initializer=<tensorflow.python.ops.init_ops.RandomUniform object>, sequence_length=None, initial_state=None, dropout=None, n_layer=1, return_last=False, return_seq_2d=False, dynamic_rnn_init_args={}, name='dyrnn_layer'*)

The `DynamicRNNLayer` class is a Dynamic RNN layer, see `tf.nn.dynamic_rnn`.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

cell_fn : a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).

- see RNN Cells in TensorFlow

cell_init_args : a dictionary

The arguments for the cell initializer.

n_hidden : an int

The number of hidden units in the layer.

initializer : initializer

The initializer for initializing the parameters.

sequence_length : a tensor, array or None. The sequence length of each row of input data, see `Advanced Ops for Dynamic RNN`.

- If None, it uses `retrieve_seq_length_op` to compute the sequence_length, i.e. when the features of padding (on right hand side) are all zeros.
- If using word embedding, you may need to compute the sequence_length from the ID array (the integer features before word embedding) by using `retrieve_seq_length_op2` or `retrieve_seq_length_op`.
- You can also input an numpy array.
- More details about TensorFlow dynamic_rnn in Wild-ML Blog.

initial_state : None or RNN State

If None, initial_state is zero_state.

dropout : *tuple* of *float*: (input_keep_prob, output_keep_prob).

The input and output keep probability.

n_layer : an int, default is 1.

The number of RNN layers.

return_last : boolean

- 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 apply one or more RNN(s) on this layer, set to False.

return_seq_2d : boolean

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer or computing cost after it.
- If False, return 3D Tensor [n_example/n_steps(max), n_steps(max), n_hidden], for stacking multiple RNN after it.

name : a string or None

An optional name to attach to this layer.

Notes

Input dimension should be rank 3 : [batch_size, n_steps(max), n_features], if no, please see *ReshapeLayer*.

References

- Wild-ML Blog
- dynamic_rnn.ipynb
- tf.nn.dynamic_rnn
- tflearn rnn
- `tutorial_dynamic_rnn.py`

Examples

```
>>> input_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"input_seqs")
>>> net = tl.layers.EmbeddingInputlayer(
...             inputs = input_seqs,
...             vocabulary_size = vocab_size,
...             embedding_size = embedding_size,
...             name = 'seq_embedding')
>>> net = tl.layers.DynamicRNNLayer(net,
...             cell_fn = tf.contrib.rnn.BasicLSTMCell, # for TF0.2 tf.nn.rnn_
↪cell.BasicLSTMCell,
...             n_hidden = embedding_size,
...             dropout = 0.7,
...             sequence_length = tl.layers.retrieve_seq_length_op2(input_seqs),
...             return_seq_2d = True,     # stack denselayer or compute cost␣
↪after it
...             name = 'dynamic_rnn')
... net = tl.layers.DenseLayer(net, n_units=vocab_size,
...             act=tf.identity, name="output")
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network

Continued on next page

表 2.31 – continued from previous page

`print_params([details])`	Print all info of parameters in the network

动态递归层

class `tensorlayer.layers.`**`DynamicRNNLayer`**(*layer=None, cell_fn=None, cell_init_args={'state_is_tuple': True}, n_hidden=256, initializer=<tensorflow.python.ops.init_ops.RandomUniform object>, sequence_length=None, initial_state=None, dropout=None, n_layer=1, return_last=False, return_seq_2d=False, dynamic_rnn_init_args={}, name='dyrnn_layer'*)

The *`DynamicRNNLayer`* class is a Dynamic RNN layer, see `tf.nn.dynamic_rnn`.

Parameters **layer** : a *`Layer`* instance

The *Layer* class feeding into this layer.

cell_fn : a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).

- see RNN Cells in TensorFlow

cell_init_args : a dictionary

The arguments for the cell initializer.

n_hidden : an int

The number of hidden units in the layer.

initializer : initializer

The initializer for initializing the parameters.

sequence_length : a tensor, array or None. The sequence length of each row of input data, see `Advanced Ops for Dynamic RNN`.

- If None, it uses `retrieve_seq_length_op` to compute the sequence_length, i.e. when the features of padding (on right hand side) are all zeros.
- If using word embedding, you may need to compute the sequence_length from the ID array (the integer features before word embedding) by using `retrieve_seq_length_op2` or `retrieve_seq_length_op`.
- You can also input an numpy array.
- More details about TensorFlow dynamic_rnn in Wild-ML Blog.

initial_state : None or RNN State

If None, initial_state is zero_state.

dropout : *tuple* of *float*: (input_keep_prob, output_keep_prob).

The input and output keep probability.

n_layer : an int, default is 1.

The number of RNN layers.

return_last : boolean

- 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 apply one or more RNN(s) on this layer, set to False.

return_seq_2d : boolean

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer or computing cost after it.
- If False, return 3D Tensor [n_example/n_steps(max), n_steps(max), n_hidden], for stacking multiple RNN after it.

name : a string or None

An optional name to attach to this layer.

Notes

Input dimension should be rank 3 : [batch_size, n_steps(max), n_features], if no, please see `ReshapeLayer`.

References

- Wild-ML Blog
- dynamic_rnn.ipynb
- tf.nn.dynamic_rnn
- tflearn rnn
- `tutorial_dynamic_rnn.py`

Examples

```
>>> input_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"input_seqs")
>>> net = tl.layers.EmbeddingInputlayer(
...             inputs = input_seqs,
...             vocabulary_size = vocab_size,
...             embedding_size = embedding_size,
...             name = 'seq_embedding')
>>> net = tl.layers.DynamicRNNLayer(net,
...             cell_fn = tf.contrib.rnn.BasicLSTMCell, # for TF0.2 tf.nn.rnn_
↪cell.BasicLSTMCell,
...             n_hidden = embedding_size,
...             dropout = 0.7,
...             sequence_length = tl.layers.retrieve_seq_length_op2(input_seqs),
...             return_seq_2d = True,     # stack denselayer or compute cost␣
↪after it
...             name = 'dynamic_rnn')
... net = tl.layers.DenseLayer(net, n_units=vocab_size,
...             act=tf.identity, name="output")
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

动态双向递归层

class tensorlayer.layers.**BiDynamicRNNLayer**(*layer=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=256*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *sequence_length=None*, *fw_initial_state=None*, *bw_initial_state=None*, *dropout=None*, *n_layer=1*, *return_last=False*, *return_seq_2d=False*, *dynamic_rnn_init_args={}*, *name='bi_dyrnn_layer'*)

The `BiDynamicRNNLayer` class is a RNN layer, you can implement vanilla RNN, LSTM and GRU with it.

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

cell_fn : a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).

- see RNN Cells in TensorFlow

cell_init_args : a dictionary

The arguments for the cell initializer.

n_hidden : an int

The number of hidden units in the layer.

initializer : initializer

The initializer for initializing the parameters.

sequence_length : a tensor, array or None.

The sequence length of each row of input data, see `Advanced Ops for Dynamic RNN`.

- If None, it uses `retrieve_seq_length_op` to compute the sequence_length, i.e. when the features of padding (on right hand side) are all zeros.
- If using word embedding, you may need to compute the sequence_length from the ID array (the integer features before word embedding) by using `retrieve_seq_length_op2` or `retrieve_seq_length_op`.
- You can also input an numpy array.
- More details about TensorFlow dynamic_rnn in Wild-ML Blog.

fw_initial_state : None or forward RNN State

If None, initial_state is zero_state.

bw_initial_state : None or backward RNN State

If None, initial_state is zero_state.

dropout : *tuple* of *float*: (input_keep_prob, output_keep_prob).

The input and output keep probability.

n_layer : an int, default is 1.

The number of RNN layers.

return_last : boolean

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 apply one or more RNN(s) on this layer, set to False.

return_seq_2d : boolean

- When return_last = False
- If True, return 2D Tensor [n_example, 2 * n_hidden], for stacking DenseLayer or computing cost after it.
- If False, return 3D Tensor [n_example/n_steps(max), n_steps(max), 2 * n_hidden], for stacking multiple RNN after it.

name : a string or None

An optional name to attach to this layer.

Notes

Input dimension should be rank 3 : [batch_size, n_steps(max), n_features], if no, please see `ReshapeLayer`.

References

- Wild-ML Blog
- bidirectional_rnn.ipynb

Attributes

outputs	(a tensor) The output of this RNN. return_last = False, outputs = all cell_output, which is the hidden state. cell_output.get_shape() = (?, 2 * n_hidden)
fw(bw)_final_state	(a tensor or StateTuple) When state_is_tuple = False, it is the final hidden and cell states, states.get_shape() = [?, 2 * n_hidden]. When state_is_tuple = True, it stores two elements: (c, h), in that order. You can get the final state after each iteration during training, then feed it to the initial state of next iteration.
fw(bw)_initial_state	(a tensor or StateTuple) It is the initial state of this RNN layer, you can use it to initialize your state at the begining of each epoch or iteration according to your training procedure.
sequence_length	(a tensor or array, shape = [batch_size]) The sequence lengths computed by Advanced Opt or the given sequence lengths.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.25 序列到序列

Simple Seq2Seq

class `tensorlayer.layers.`**`Seq2Seq`**(*net_encode_in=None*, *net_decode_in=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=256*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *encode_sequence_length=None*, *decode_sequence_length=None*, *initial_state_encode=None*, *initial_state_decode=None*, *dropout=None*, *n_layer=1*, *return_seq_2d=False*, *name='seq2seq'*)

The *`Seq2Seq`* class is a Simple *`DynamicRNNLayer`* based Seq2seq layer without using tl.contrib.seq2seq. See Model and Sequence to Sequence Learning with Neural Networks.

- Please check the example Chatbot in 200 lines of code.
- The Author recommends users to read the source code of *`DynamicRNNLayer`* and *`Seq2Seq`*.

Parameters **net_encode_in** : a *`Layer`* instance

Encode sequences, [batch_size, None, n_features].

net_decode_in : a *`Layer`* instance

Decode sequences, [batch_size, None, n_features].

cell_fn : a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).

- see RNN Cells in TensorFlow

cell_init_args : a dictionary

The arguments for the cell initializer.

n_hidden : an int

The number of hidden units in the layer.

initializer : initializer

The initializer for initializing the parameters.

encode_sequence_length : tensor for encoder sequence length, see *`DynamicRNNLayer`* .

decode_sequence_length : tensor for decoder sequence length, see *`DynamicRNNLayer`* .

initial_state_encode : None or RNN state (from placeholder or other RNN).

If None, initial_state_encode is of zero state.

initial_state_decode : None or RNN state (from placeholder or other RNN).

If None, initial_state_decode is of the final state of the RNN encoder.

dropout : *tuple* of *float*: (input_keep_prob, output_keep_prob).

The input and output keep probability.

n_layer : an int, default is 1.

The number of RNN layers.

return_seq_2d : boolean

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer or computing cost after it.
- If False, return 3D Tensor [n_example/n_steps(max), n_steps(max), n_hidden], for stacking multiple RNN after it.

name : a string or None

An optional name to attach to this layer.

Notes

- How to feed data: Sequence to Sequence Learning with Neural Networks
- input_seqs : `['how', 'are', 'you', '<PAD_ID'>]`
- decode_seqs : `['<START_ID>', 'I', 'am', 'fine', '<PAD_ID'>]`
- target_seqs : `['I', 'am', 'fine', '<END_ID', '<PAD_ID'>]`
- target_mask : `[1, 1, 1, 1, 0]`
- related functions : tl.prepro <pad_sequences, precess_sequences, sequences_add_start_id, sequences_get_mask>

Examples

```
>>> from tensorlayer.layers import *
>>> batch_size = 32
>>> encode_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"encode_seqs")
>>> decode_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"decode_seqs")
>>> target_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"target_seqs")
>>> target_mask = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"target_mask") # tl.prepro.sequences_get_mask()
>>> with tf.variable_scope("model"):
...     # for chatbot, you can use the same embedding layer,
...     # for translation, you may want to use 2 seperated embedding layers
>>>     with tf.variable_scope("embedding") as vs:
>>>         net_encode = EmbeddingInputlayer(
...                 inputs = encode_seqs,
...                 vocabulary_size = 10000,
...                 embedding_size = 200,
...                 name = 'seq_embedding')
>>>         vs.reuse_variables()
>>>         tl.layers.set_name_reuse(True)
>>>         net_decode = EmbeddingInputlayer(
...                 inputs = decode_seqs,
...                 vocabulary_size = 10000,
...                 embedding_size = 200,
...                 name = 'seq_embedding')
```

```
>>>     net = Seq2Seq(net_encode, net_decode,
...             cell_fn = tf.contrib.rnn.BasicLSTMCell,
...             n_hidden = 200,
...             initializer = tf.random_uniform_initializer(-0.1, 0.1),
...             encode_sequence_length = retrieve_seq_length_op2(encode_seqs),
...             decode_sequence_length = retrieve_seq_length_op2(decode_seqs),
...             initial_state_encode = None,
...             dropout = None,
...             n_layer = 1,
...             return_seq_2d = True,
...             name = 'seq2seq')
>>> net_out = DenseLayer(net, n_units=10000, act=tf.identity, name='output')
>>> e_loss = tl.cost.cross_entropy_seq_with_mask(logits=net_out.outputs, target_
↪seqs=target_seqs, input_mask=target_mask, return_details=False, name='cost')
>>> y = tf.nn.softmax(net_out.outputs)
>>> net_out.print_params(False)
```

Attributes

outputs	(a tensor) The output of RNN decoder.
initial_state_encode	(a tensor or StateTuple) Initial state of RNN encoder.
initial_state_decode	(a tensor or StateTuple) Initial state of RNN decoder.
final_state_encode	(a tensor or StateTuple) Final state of RNN encoder.
final_state_decode	(a tensor or StateTuple) Final state of RNN decoder.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

PeekySeq2Seq

class `tensorlayer.layers.`**`PeekySeq2Seq`**(*net_encode_in=None, net_decode_in=None, cell_fn=None, cell_init_args={'state_is_tuple': True}, n_hidden=256, initializer=<tensorflow.python.ops.init_ops.RandomUniform object>, in_sequence_length=None, out_sequence_length=None, initial_state=None, dropout=None, n_layer=1, return_seq_2d=False, name='peeky_seq2seq'*)

Waiting for contribution. The `PeekySeq2Seq` class, see Model and Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation .

Methods

`count_params()`	Return the number of parameters in the network

Continued on next page

表 2.35 – continued from previous page

print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

AttentionSeq2Seq

class tensorlayer.layers.**AttentionSeq2Seq**(*net_encode_in=None, net_decode_in=None, cell_fn=None, cell_init_args={'state_is_tuple': True}, n_hidden=256, initializer=<tensorflow.python.ops.init_ops.RandomUniform object>, in_sequence_length=None, out_sequence_length=None, initial_state=None, dropout=None, n_layer=1, return_seq_2d=False, name='attention_seq2seq'*)

Waiting for contribution. The *AttentionSeq2Seq* class, see Model and Neural Machine Translation by Jointly Learning to Align and Translate .

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

2.1.26 形状修改层

Flatten层

class tensorlayer.layers.**FlattenLayer**(*layer=None, name='flatten_layer'*)

The *FlattenLayer* class is layer which reshape high-dimension input to a vector. Then we can apply DenseLayer, RNNLayer, ConcatLayer and etc on the top of it.

[batch_size, mask_row, mask_col, n_mask] —> [batch_size, mask_row * mask_col * n_mask]

Parameters layer : a *Layer* instance

The *Layer* class feeding into this layer.

name : a string or None

An optional name to attach to this layer.

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 28, 28, 1])
>>> net = tl.layers.InputLayer(x, name='input_layer')
>>> net = tl.layers.Conv2dLayer(net,
...                     act = tf.nn.relu,
...                     shape = [5, 5, 32, 64],
...                     strides=[1, 1, 1, 1],
...                     padding='SAME',
...                     name ='cnn_layer')
>>> net = tl.layers.Pool2dLayer(net,
...                       ksize=[1, 2, 2, 1],
...                       strides=[1, 2, 2, 1],
...                       padding='SAME',
...                       pool = tf.nn.max_pool,
...                       name ='pool_layer',)
>>> net = tl.layers.FlattenLayer(net, name='flatten_layer')
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

Reshape层

class tensorlayer.layers.**ReshapeLayer**(*layer=None*, *shape=[]*, *name='reshape_layer'*)

The *ReshapeLayer* class is layer which reshape the tensor.

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer.

shape : a list

The output shape.

name : a string or None

An optional name to attach to this layer.

Examples

- The core of this layer is `tf.reshape`.
- Use TensorFlow only :

```
>>> x = tf.placeholder(tf.float32, shape=[None, 3])
>>> y = tf.reshape(x, shape=[-1, 3, 3])
>>> sess = tf.InteractiveSession()
>>> print(sess.run(y, feed_dict={x:[[1,1,1],[2,2,2],[3,3,3],[4,4,4],[5,5,5],[6,6,
↪6]]}))
... [[[ 1.  1.  1.]
... [ 2.  2.  2.]
... [ 3.  3.  3.]]
... [[ 4.  4.  4.]
... [ 5.  5.  5.]
... [ 6.  6.  6.]]]
```

Methods

count_params()	Return the number of parameters in the network

Continued on next page

表 2.38 – continued from previous page

print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

Transpose层

class tensorlayer.layers.**TransposeLayer**(*layer=None*, *perm=None*, *name='transpose'*)

The *TransposeLayer* class transpose the dimension of a teneor, see tf.transpose() .

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer.

perm: list, a permutation of the dimensions

Similar with numpy.transpose.

name : a string or None

An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details])	Print all info of parameters in the network

2.1.27 Lambda层

class tensorlayer.layers.**LambdaLayer**(*layer=None*, *fn=None*, *fn_args={}*, *name='lambda_layer'*)

The *LambdaLayer* class is a layer which is able to use the provided function.

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer.

fn : a function

The function that applies to the outputs of previous layer.

fn_args : a dictionary

The arguments for the function (option).

name : a string or None

An optional name to attach to this layer.

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 1], name='x')
>>> net = tl.layers.InputLayer(x, name='input_layer')
>>> net = LambdaLayer(net, lambda x: 2*x, name='lambda_layer')
>>> y = net.outputs
>>> sess = tf.InteractiveSession()
>>> out = sess.run(y, feed_dict={x : [[1],[2]]})
... [[2],[4]]
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.28 合并层

Concat层

class `tensorlayer.layers.`**`ConcatLayer`**(*layer=[]*, *concat_dim=1*, *name='concat_layer'*)

The `ConcatLayer` class is layer which concat (merge) two or more tensor by given axis..

Parameters **layer** : a list of `Layer` instances

The *Layer* class feeding into this layer.

concat_dim : int

Dimension along which to concatenate.

name : a string or None

An optional name to attach to this layer.

Examples

```
>>> sess = tf.InteractiveSession()
>>> x = tf.placeholder(tf.float32, shape=[None, 784])
>>> inputs = tl.layers.InputLayer(x, name='input_layer')
>>> net1 = tl.layers.DenseLayer(inputs, n_units=800, act = tf.nn.relu, name=
↪'relu1_1')
>>> net2 = tl.layers.DenseLayer(inputs, n_units=300, act = tf.nn.relu, name=
↪'relu2_1')
>>> net = tl.layers.ConcatLayer(layer = [net1, net2], name ='concat_layer')
...     [TL] InputLayer input_layer (?, 784)
...     [TL] DenseLayer relu1_1: 800, <function relu at 0x1108e41e0>
...     [TL] DenseLayer relu2_1: 300, <function relu at 0x1108e41e0>
...     [TL] ConcatLayer concat_layer, 1100
...
>>> tl.layers.initialize_global_variables(sess)
>>> net.print_params()
...     param 0: (784, 800) (mean: 0.000021, median: -0.000020 std: 0.035525)
...     param 1: (800,) (mean: 0.000000, median: 0.000000 std: 0.000000)
...     param 2: (784, 300) (mean: 0.000000, median: -0.000048 std: 0.042947)
...     param 3: (300,) (mean: 0.000000, median: 0.000000 std: 0.000000)
...     num of params: 863500
>>> net.print_layers()
...     layer 0: Tensor("Relu:0", shape=(?, 800), dtype=float32)
...     layer 1: Tensor("Relu_1:0", shape=(?, 300), dtype=float32)
...
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

Elementwise层

class `tensorlayer.layers.`**`ElementwiseLayer`**(*layer=[]*, *combine_fn=<function minimum>*, *name='elementwise_layer'*)

The `ElementwiseLayer` class combines multiple `Layer` which have the same output shapes by a given elemwise-wise operation.

Parameters **layer** : a list of `Layer` instances

The *Layer* class feeding into this layer.

combine_fn : a TensorFlow elemwise-merge 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 .

name : a string or None

An optional name to attach to this layer.

Examples

- AND Logic

```
>>> net_0 = tl.layers.DenseLayer(net_0, n_units=500,
...                         act = tf.nn.relu, name='net_0')
>>> net_1 = tl.layers.DenseLayer(net_1, n_units=500,
...                         act = tf.nn.relu, name='net_1')
>>> net_com = tl.layers.ElementwiseLayer(layer = [net_0, net_1],
...                          combine_fn = tf.minimum,
...                          name = 'combine_layer')
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.29 扩充层

Expand 层

class `tensorlayer.layers.`**`ExpandDimsLayer`**(*layer=None*, *axis=None*, *name='expand_dims'*)

The `ExpandDimsLayer` class inserts a dimension of 1 into a tensor's shape, see tf.expand_dims() .

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

axis : int, 0-D (scalar).

Specifies the dimension index at which to expand the shape of input.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

Tile 层

class `tensorlayer.layers.`**`TileLayer`**(*layer=None*, *multiples=None*, *name='tile'*)

The `TileLayer` class constructs a tensor by tiling a given tensor, see tf.tile() .

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

multiples: a list of int

Must be one of the following types: int32, int64. 1-D. Length must be the same as the number of dimensions in input

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.30 堆叠层

Stack 层

class `tensorlayer.layers.`**`StackLayer`**(*layer=[]*, *axis=0*, *name='stack'*)

The `StackLayer` class is layer for stacking a list of rank-R tensors into one rank-(R+1) tensor, see tf.stack().

Parameters **layer** : a list of `Layer` instances

The *Layer* class feeding into this layer.

axis : an int

Dimension along which to concatenate.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

Unstack 层

`tensorlayer.layers.`**`UnStackLayer`**(*layer=None*, *num=None*, *axis=0*, *name='unstack'*)

The *UnStackLayer* is layer for unstacking the given dimension of a rank-R tensor into rank-(R-1) tensors., see tf.unstack().

Parameters **layer** : a list of `Layer` instances

The *Layer* class feeding into this layer.

num : an int

The length of the dimension axis. Automatically inferred if None (the default).

axis : an int

Dimension along which to concatenate.

name : a string or None

An optional name to attach to this layer.

Returns The list of layer objects unstacked from the input.

2.1.31 Estimator 层

`class` `tensorlayer.layers.`**`EstimatorLayer`**(*layer=None*, *model_fn=None*, *args={}*, *name='estimator_layer'*)

The `EstimatorLayer` class accepts `model_fn` that described the model. It is similar with `KerasLayer`, see tutorial_keras.py

Parameters **layer** : a `Layer` instance

The *Layer* class feeding into this layer.

model_fn : a function that described the model.

args : dictionary

The arguments for the model_fn.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network

Continued on next page

表 2.46 – continued from previous page

`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.32 连接 TF-Slim

没错！TF-Slim 可以和 TensorLayer 无缝对接！所有 Google 预训练好的模型都可以直接使用！ 模型请见 Slim-model 。

class tensorlayer.layers.**SlimNetsLayer**(*layer=None*, *slim_layer=None*, *slim_args={}*, *name='tfslim_layer'*)

The *SlimNetsLayer* class can be used to merge all TF-Slim nets into TensorLayer. Model can be found in slim-model , more about slim see slim-git .

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer.

slim_layer : a slim network function

The network you want to stack onto, end with `return net, end_points`.

slim_args : dictionary

The arguments for the slim model.

name : a string or None

An optional name to attach to this layer.

Notes

The due to TF-Slim stores the layers as dictionary, the `all_layers` in this network is not in order ! Fortunately, the `all_params` are in order.

Examples

- see Inception V3 example on Github

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.33 连接 Keras

没错！Keras 可以和 TensorLayer 无缝对接！ 参见 tutorial_keras.py .

class tensorlayer.layers.**KerasLayer**(*layer=None*, *keras_layer=None*, *keras_args={}*, *name='keras_layer'*)

The *KerasLayer* class can be used to merge all Keras layers into TensorLayer. Example can be found here tutorial_keras.py

Parameters **layer** : a *Layer* instance

The *Layer* class feeding into this layer.

keras_layer : a keras network function

keras_args : dictionary

The arguments for the keras model.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.34 带参数的激活函数

class `tensorlayer.layers.`**`PReluLayer`**(*layer=None*, *channel_shared=False*, *a_init=<tensorflow.python.ops.init_ops.Constant object>*, *a_init_args={}*, *name='prelu_layer'*)

The `PReluLayer` class is Parametric Rectified Linear layer.

Parameters **x** : A *Tensor* with type *float*, *double*, *int32*, *int64*, *uint8*,

int16, or *int8*.

channel_shared : *bool*. Single weight is shared by all channels

a_init : alpha initializer, default zero constant.

The initializer for initializing the alphas.

a_init_args : dictionary

The arguments for the weights initializer.

name : A name for this activation op (optional).

References

- Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.35 流控制层

class `tensorlayer.layers.`**`MultiplexerLayer`**(*layer=[]*, *name='mux_layer'*)

The `MultiplexerLayer` selects one of several input and forwards the selected input into the output, see *tutorial_mnist_multiplexer.py*.

Parameters **layer** : a list of *Layer* instances

The *Layer* class feeding into this layer.

name : a string or None

An optional name to attach to this layer.

References

- See `tf.pack()` for TF0.12 or `tf.stack()` for TF1.0 and `tf.gather()` at TensorFlow - Slicing and Joining

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
>>> y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')
>>> # define the network
>>> net_in = tl.layers.InputLayer(x, name='input_layer')
>>> net_in = tl.layers.DropoutLayer(net_in, keep=0.8, name='drop1')
>>> # net 0
>>> net_0 = tl.layers.DenseLayer(net_in, n_units=800,
...                                 act = tf.nn.relu, name='net0/relu1')
>>> net_0 = tl.layers.DropoutLayer(net_0, keep=0.5, name='net0/drop2')
>>> net_0 = tl.layers.DenseLayer(net_0, n_units=800,
...                                 act = tf.nn.relu, name='net0/relu2')
>>> # net 1
>>> net_1 = tl.layers.DenseLayer(net_in, n_units=800,
...                                 act = tf.nn.relu, name='net1/relu1')
>>> net_1 = tl.layers.DropoutLayer(net_1, keep=0.8, name='net1/drop2')
>>> net_1 = tl.layers.DenseLayer(net_1, n_units=800,
...                                 act = tf.nn.relu, name='net1/relu2')
>>> net_1 = tl.layers.DropoutLayer(net_1, keep=0.8, name='net1/drop3')
>>> net_1 = tl.layers.DenseLayer(net_1, n_units=800,
...                                 act = tf.nn.relu, name='net1/relu3')
>>> # multiplexer
>>> net_mux = tl.layers.MultiplexerLayer(layer = [net_0, net_1], name='mux_layer')
>>> network = tl.layers.ReshapeLayer(net_mux, shape=[-1, 800], name='reshape_layer
↪') #
>>> network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
>>> # output layer
>>> network = tl.layers.DenseLayer(network, n_units=10,
...                                 act = tf.identity, name='output_layer')
```

Methods

`count_params()`	Return the number of parameters in the network
`print_layers()`	Print all info of layers in the network
`print_params([details])`	Print all info of parameters in the network

2.1.36 包装器(Wrapper)

嵌入+注意机制+Seq2seq

class tensorlayer.layers.**EmbeddingAttentionSeq2seqWrapper**(*source_vocab_size*, *target_vocab_size*, *buckets*, *size*, *num_layers*, *max_gradient_norm*, *batch_size*, *learning_rate*, *learning_rate_decay_factor*, *use_lstm=False*, *num_samples=512*, *forward_only=False*, *name='wrapper'*)

Sequence-to-sequence model with attention and for multiple buckets (Deprecated after TF0.12).

This example implements a multi-layer recurrent neural network as encoder, and an attention-based decoder. This is the same as the model described in this paper: - Grammar as a Foreign Language please look there for details, or into the seq2seq library for complete model implementation. This example also allows to use GRU cells in addition to LSTM cells, and sampled softmax to handle large output vocabulary size. A single-layer version of this model, but with bi-directional encoder, was presented in - Neural Machine Translation by Jointly Learning to Align and Translate The sampled softmax is described in Section 3 of the following paper. - On Using Very Large Target Vocabulary for Neural Machine Translation

Parameters **source_vocab_size** : size of the source vocabulary.

target_vocab_size : size of the target vocabulary.

buckets : a list of pairs (I, O), where I specifies maximum input length

that will be processed in that bucket, and O specifies maximum output length. Training instances that have inputs longer than I or outputs longer than O will be pushed to the next bucket and padded accordingly. We assume that the list is sorted, e.g., [(2, 4), (8, 16)].

size : number of units in each layer of the model.

num_layers : number of layers in the model.

max_gradient_norm : gradients will be clipped to maximally this norm.

batch_size : the size of the batches used during training;

the model construction is independent of batch_size, so it can be changed after initialization if this is convenient, e.g., for decoding.

learning_rate : learning rate to start with.

learning_rate_decay_factor : decay learning rate by this much when needed.

use_lstm : if true, we use LSTM cells instead of GRU cells.

num_samples : number of samples for sampled softmax.

forward_only : if set, we do not construct the backward pass in the model.

name : a string or None

An optional name to attach to this layer.

Methods

`count_params`()	Return the number of parameters in the network
`get_batch`(data, bucket_id[, PAD_ID, GO_ID, ...])	Get a random batch of data from the specified bucket, prepare for step.
`print_layers`()	Print all info of layers in the network
`print_params`([details])	Print all info of parameters in the network
`step`(session, encoder_inputs, ...)	Run a step of the model feeding the given inputs.

`get_batch` (*data*, *bucket_id*, *PAD_ID=0*, *GO_ID=1*, *EOS_ID=2*, *UNK_ID=3*)
Get a random batch of data from the specified bucket, prepare for step.

To feed data in step(..) it must be a list of batch-major vectors, while data here contains single length-major cases. So the main logic of this function is to re-index data cases to be in the proper format for feeding.

Parameters **data** : a tuple of size len(self.buckets) in which each element contains

lists of pairs of input and output data that we use to create a batch.

bucket_id : integer, which bucket to get the batch for.

PAD_ID : int

Index of Padding in vocabulary

GO_ID : int

Index of GO in vocabulary

EOS_ID : int

Index of End of sentence in vocabulary

UNK_ID : int

Index of Unknown word in vocabulary

Returns The triple (encoder_inputs, decoder_inputs, target_weights) for

the constructed batch that has the proper format to call step(...) later.

`step` (*session*, *encoder_inputs*, *decoder_inputs*, *target_weights*, *bucket_id*, *forward_only*)
Run a step of the model feeding the given inputs.

Parameters **session** : tensorflow session to use.

encoder_inputs : list of numpy int vectors to feed as encoder inputs.

decoder_inputs : list of numpy int vectors to feed as decoder inputs.

target_weights : list of numpy float vectors to feed as target weights.

bucket_id : which bucket of the model to use.

forward_only : whether to do the backward step or only forward.

Returns A triple consisting of gradient norm (or None if we did not do backward),

average perplexity, and the outputs.

Raises **ValueError** : if length of encoder_inputs, decoder_inputs, or

target_weights disagrees with bucket size for the specified bucket_id.

2.1.37 辅助函数

Flatten tensor

`tensorlayer.layers.flatten_reshape(variable, name='')`

Reshapes high-dimension input to a vector. [batch_size, mask_row, mask_col, n_mask] —> [batch_size, mask_row * mask_col * n_mask]

Parameters **variable** : a tensorflow variable

name : a string or None

An optional name to attach to this layer.

Examples

```
>>> W_conv2 = weight_variable([5, 5, 100, 32])   # 64 features for each 5x5 patch
>>> b_conv2 = bias_variable([32])
>>> W_fc1 = weight_variable([7 * 7 * 32, 256])
```

```
>>> h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
>>> h_pool2 = max_pool_2x2(h_conv2)
>>> h_pool2.get_shape()[:].as_list() = [batch_size, 7, 7, 32]
...         [batch_size, mask_row, mask_col, n_mask]
>>> h_pool2_flat = tl.layers.flatten_reshape(h_pool2)
...         [batch_size, mask_row * mask_col * n_mask]
>>> h_pool2_flat_drop = tf.nn.dropout(h_pool2_flat, keep_prob)
...
```

永久删除现有 layer 名字

`tensorlayer.layers.clear_layers_name()`

Clear all layer names in set_keep['_layers_name_list'], enable layer name reuse.

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DenseLayer(network, n_units=800, name='relu1')
...
>>> tl.layers.clear_layers_name()
>>> network2 = tl.layers.InputLayer(x, name='input_layer')
>>> network2 = tl.layers.DenseLayer(network2, n_units=800, name='relu1')
...
```

初始化 RNN state

`tensorlayer.layers.initialize_rnn_state(state)`

Return the initialized RNN state. The input is LSTMStateTuple or State of RNNCells.

Parameters **state** : a RNN state.

去除列表中重复内容

`tensorlayer.layers.list_remove_repeat(l=None)`

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 l : a list

Examples

```
>>> l = [2, 3, 4, 2, 3]
>>> l = list_remove_repeat(l)
... [2, 3, 4]
```

2.2 API - 损失函数

为了尽可能地保持TensorLayer的简洁性，我们最小化损失函数的数量。因此我们鼓励直接使用TensorFlow官方的函数，比如你可以通过 `tf.nn.l2_loss`, `tf.contrib.layers.l1_regularizer`, `tf.contrib.layers.l2_regularizer` and `tf.contrib.layers.sum_regularizer` 来实现L1, L2 和 sum 规则化， 参考 TensorFlow API。

2.2.1 自定义损失函数

TensorLayer提供一个简单的方法来创建您自己的损失函数。下面以多层神经网络(MLP)为例：

```
network = tl.InputLayer(x, name='input_layer')
network = tl.DropoutLayer(network, keep=0.8, name='drop1')
network = tl.DenseLayer(network, n_units=800, act = tf.nn.relu, name='relu1')
network = tl.DropoutLayer(network, keep=0.5, name='drop2')
network = tl.DenseLayer(network, n_units=800, act = tf.nn.relu, name='relu2')
network = tl.DropoutLayer(network, keep=0.5, name='drop3')
network = tl.DenseLayer(network, n_units=10, act = tl.activation.identity, name=
↪'output_layer')
```

那么其模型参数为 `[W1, b1, W2, b2, W_out, b_out]`， 这时，你可以像下面的例子那样实现对前两个weights矩阵的L2规则化。

```
cost = tl.cost.cross_entropy(y, y_, name = 'cost')
cost = cost + tf.contrib.layers.l2_regularizer(0.001)(network.all_params[0]) + tf.
↪contrib.layers.l2_regularizer(0.001)(network.all_params[2])
```

此外，TensorLayer 提供了通过给定名称，很方便地获取参数列表的方法，所以您可以如下对某些参数执行L2规则化。

```
l2 = 0
for w in tl.layers.get_variables_with_name('W_conv2d', train_only=True,␣
↪printable=False):#[-3:]:
    l2 += tf.contrib.layers.l2_regularizer(1e-4)(w)
cost = tl.cost.cross_entropy(y, y_, name = 'cost') + l2
```

权值的正则化

在初始化变量之后，网络参数的信息可以使用 `network.print.params()` 来获得。

```
sess.run(tf.initialize_all_variables())
network.print_params()
```

```
param 0: (784, 800) (mean: -0.000000, median: 0.000004 std: 0.035524)
param 1: (800,) (mean: 0.000000, median: 0.000000 std: 0.000000)
param 2: (800, 800) (mean: 0.000029, median: 0.000031 std: 0.035378)
param 3: (800,) (mean: 0.000000, median: 0.000000 std: 0.000000)
param 4: (800, 10) (mean: 0.000673, median: 0.000763 std: 0.049373)
param 5: (10,) (mean: 0.000000, median: 0.000000 std: 0.000000)
num of params: 1276810
```

网络的输出是 `network.outputs` ，那么交叉熵的可以被如下定义。 另外，要正则化权重， `network.all_params` 要包含网络的所有参数。 在这种情况下根据 `network.print_params()` 所展示的参数 0,1,...,5的值, `network.all_params = [W1, b1, W2, b2, Wout, bout]` 然后对W1和W2的最大范数正则化可以按如下进行：

```
y = network.outputs
# Alternatively, you can use tl.cost.cross_entropy(y, y_, name = 'cost') instead.
cross_entropy = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(y, y_))
cost = cross_entropy
cost = cost + tl.cost.maxnorm_regularizer(1.0)(network.all_params[0]) +
          tl.cost.maxnorm_regularizer(1.0)(network.all_params[2])
```

另外，所有的TensorFlow的正则化函数，像 `tf.contrib.layers.l2_regularizer` 在TensorLayer中也能使用。

激活输出(Activation outputs)的规则化

实例方法 `network.print_layers()` 整齐地打印不同层的所有输出。 为了实现对激活输出的正则化，您可以使用 `network.all_layers` ，它包含了不同层的所有输出。 如果您想对第一层隐藏层的激活输出使用L1惩罚，仅仅需要添加 `tf.contrib.layers.l2_regularizer(lambda_l1)(network.all_layers[1])` 到成本函数中。

```
network.print_layers()
```

```
layer 0: Tensor("dropout/mul_1:0", shape=(?, 784), dtype=float32)
layer 1: Tensor("Relu:0", shape=(?, 800), dtype=float32)
layer 2: Tensor("dropout_1/mul_1:0", shape=(?, 800), dtype=float32)
layer 3: Tensor("Relu_1:0", shape=(?, 800), dtype=float32)
layer 4: Tensor("dropout_2/mul_1:0", shape=(?, 800), dtype=float32)
layer 5: Tensor("add_2:0", shape=(?, 10), dtype=float32)
```

cross_entropy(output, target[, name])	It is a softmax cross-entropy operation, returns the TensorFlow expression of cross-entropy of two distributions, implement softmax internally.
sigmoid_cross_entropy(output, target[, name])	It is a sigmoid cross-entropy operation, see `tf.nn.sigmoid_cross_entropy_with_logits`.
binary_cross_entropy(output, target[, ...])	Computes binary cross entropy given *output*.

Continued on next page

表 2.52 – continued from previous page

mean_squared_error(output, target[, is_mean])	Return the TensorFlow expression of mean-square-error of two distributions.
normalized_mean_square_error(output, target)	Return the TensorFlow expression of normalized mean-square-error of two distributions.
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], wiki.
li_regularizer(scale[, scope])	li regularization removes the neurons of previous layer, *i* represents *inputs*.
lo_regularizer(scale[, scope])	lo regularization removes the neurons of current layer, *o* represents *outputs*
maxnorm_regularizer([scale, scope])	Max-norm regularization returns a function that can be used to apply max-norm regularization to weights.
maxnorm_o_regularizer(scale, scope)	Max-norm output regularization removes the neurons of current layer.
maxnorm_i_regularizer(scale[, scope])	Max-norm input regularization removes the neurons of previous layer.

2.2.2 Softmax cross entropy

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

It is a softmax cross-entropy operation, returns the TensorFlow expression of cross-entropy of two distributions, implement softmax internally. See `tf.nn.sparse_softmax_cross_entropy_with_logits`.

Parameters **output** : Tensorflow variable

A distribution with shape: [batch_size, n_feature].

target : Tensorflow variable

A batch of index with shape: [batch_size,].

name : string

Name of this loss.

References

- About cross-entropy: wiki.
- The code is borrowed from: here.

Examples

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

2.2.3 Sigmoid cross entropy

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

It is a sigmoid cross-entropy operation, see `tf.nn.sigmoid_cross_entropy_with_logits`.

2.2.4 Binary cross entropy

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

Computes binary cross entropy given *output*.

For brevity, let x = *output*, z = *target*. The binary cross entropy loss is

loss(x, z) = - sum_i (x[i] * log(z[i]) + (1 - x[i]) * log(1 - z[i]))

Parameters **output** : tensor of type *float32* or *float64*.

target : tensor of the same type and shape as *output*.

epsilon : float

A small value to avoid output is zero.

name : string

An optional name to attach to this layer.

References

- DRAW

2.2.5 Mean squared error

`tensorlayer.cost.`**`mean_squared_error`**(*output*, *target*, *is_mean=False*)

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

Parameters **output** : 2D or 4D tensor.

target : 2D or 4D tensor.

is_mean : boolean, if True, use `tf.reduce_mean` to compute the loss of one data, otherwise, use `tf.reduce_sum` (default).

References

- Wiki Mean Squared Error

2.2.6 Normalized mean square error

`tensorlayer.cost.`**`normalized_mean_square_error`**(*output*, *target*)

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

Parameters **output** : 2D or 4D tensor.

target : 2D or 4D tensor.

2.2.7 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

A distribution with shape: [batch_size, ...], (any dimensions).

loss_type : string

`jaccard` or `sorensen`, default is `jaccard`.

axis : list of integer

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.

References

- Wiki-Dice

Examples

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

2.2.8 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

A distribution with shape: [batch_size, ...], (any dimensions).

threshold : float

The threshold value to be true.

axis : list 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

2.2.9 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, 1 means totally match.

Parameters **output** : tensor

A distribution with shape: [batch_size, ...], (any dimensions).

target : tensor

A distribution with shape: [batch_size, ...], (any dimensions).

threshold : float

The threshold value to be true.

axis : list 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.

2.2.10 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.

Parameters **logits** : Tensorflow variable

2D tensor, `network.outputs`, [batch_size*n_steps (n_examples), number of output units]

target_seqs : Tensorflow variable

target : 2D tensor [batch_size, n_steps], if the number of step is dynamic, please use `cross_entropy_seq_with_mask` instead.

batch_size : None or int.

If not None, the return cost will be divided by batch_size.

Examples

```
>>> see PTB tutorial for more details
>>> input_data = tf.placeholder(tf.int32, [batch_size, num_steps])
>>> targets = tf.placeholder(tf.int32, [batch_size, num_steps])
>>> cost = tl.cost.cross_entropy_seq(network.outputs, targets)
```

2.2.11 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 outputs.

Parameters **logits** : network identity outputs

2D tensor, `network.outputs`, [batch_size, number of output units].

target_seqs : int of tensor, like word ID.

[batch_size, ?]

input_mask : the mask to compute loss

The same size with target_seqs, normally 0 and 1.

return_details : boolean

- If False (default), only returns the loss.
- If True, returns the loss, losses, weights and targets (reshape to one vetcor).

Examples

- see Image Captioning Example.

2.2.12 Cosine similarity

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

Cosine similarity [-1, 1], wiki.

Parameters **v1, v2** : tensor of [batch_size, n_feature], with the same number of features.

Returns a tensor of [batch_size,]

2.2.13 规则化函数

更 多 `tf.nn.l2_loss`, `tf.contrib.layers.l1_regularizer`, `tf.contrib.layers.l2_regularizer` 与 `tf.contrib.layers.sum_regularizer`, 请见 TensorFlow API.

Maxnorm

`tensorlayer.cost.`**`maxnorm_regularizer`**(*scale=1.0*, *scope=None*)

Max-norm regularization returns a function that can be used to apply max-norm regularization to weights. About max-norm: wiki.

The implementation follows TensorFlow contrib.

Parameters **scale** : float

A scalar multiplier *Tensor*. 0.0 disables the regularizer.

scope: An optional scope name.

Returns A function with signature *mn(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.

Special

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

li regularization removes the neurons of previous layer, *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: An optional scope name for TF12+.

Returns 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*, *scope=None*)

lo regularization removes the neurons of current layer, *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.

scope: An optional scope name for TF12+.

Returns 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*, *scope*)

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.

scope: An optional scope name.

Returns 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*, *scope=None*)

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.

scope: An optional scope name.

Returns 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.

2.3 API - 数据预处理

我们提供大量的数据增强及处理方法，使用 Numpy, Scipy, Threading 和 Queue。 不过，我们建议你直接使用 TensorFlow 提供的 operator，如 `tf.image.central_crop` ，更多关于 TensorFlow 的信息请见 这里 和 `tutorial_cifar10_tfrecord.py`. 这个包的一部分代码来自Keras。

`threading_data`([data, fn, thread_count])	Return a batch of result by given data.
`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,
`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.
`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 deformation of images as described in [Simard2003] .

Continued on next page

表 2.53 – continued from previous page

elastic_transform_multi(x, alpha, sigma[, ...])	Elastic deformation of images as described in [Simard2003].
zoom(x[, zoom_range, is_random, row_index, ...])	Zoom in and out of a single image, randomly or non-randomly.
zoom_multi(x[, zoom_range, is_random, ...])	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.
imresize(x[, size, interp, mode])	Resize an image by given output size and method.
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.
transform_matrix_offset_center(matrix, x, y)	Return transform matrix offset center.
apply_transform(x, transform_matrix[, ...])	Return transformed images by given transform_matrix from `transform_matrix_offset_center`.
projective_transform_by_points(x, src, dst)	Projective transform by given coordinates, usually 4 coordinates.
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 .
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.
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.

Continued on next page

表 2.53 – continued from previous page

sequences_get_mask(sequences[, pad_val])	Return mask for sequences.
distorted_images([images, height, width])	Distort images for generating more training data.
crop_central_whiten_images([images, height, ...])	Crop the central of image, and normailize it for test data.

2.3.1 并行 Threading

`tensorlayer.prepro.`**`threading_data`**(*data=None*, *fn=None*, *thread_count=None*, ***kwargs*)

Return a batch of result by given data. Usually be used for data augmentation.

Parameters data : numpy array, file names and etc, see Examples below.

thread_count : the number of threads to use

fn : the function for data processing.

more args : the args for fn, see Examples below.

References

- python queue
- run with limited queue

Examples

- Single array

```
>>> X --> [batch_size, row, col, 1] greyscale
>>> results = threading_data(X, zoom, zoom_range=[0.5, 1], is_random=True)
... results --> [batch_size, row, col, channel]
>>> tl.visualize.images2d(images=np.asarray(results), second=0.01, saveable=True,␣
↪name='after', dtype=None)
>>> tl.visualize.images2d(images=np.asarray(X), second=0.01, saveable=True, name=
↪'before', dtype=None)
```

- List of array (e.g. functions with `multi`)

```
>>> X, Y --> [batch_size, row, col, 1]  greyscale
>>> data = threading_data([_ for _ in zip(X, Y)], zoom_multi, zoom_range=[0.5, 1],
↪ is_random=True)
... data --> [batch_size, 2, row, col, 1]
>>> X_, Y_ = data.transpose((1,0,2,3,4))
... X_, Y_ --> [batch_size, row, col, 1]
>>> tl.visualize.images2d(images=np.asarray(X_), second=0.01, saveable=True, name=
↪'after', dtype=None)
>>> tl.visualize.images2d(images=np.asarray(Y_), second=0.01, saveable=True, name=
↪'before', dtype=None)
```

- Single array split across `thread_count` threads (e.g. functions with `multi`)

```
>>> X, Y --> [batch_size, row, col, 1]  greyscale
>>> data = threading_data(X, zoom_multi, 8, zoom_range=[0.5, 1], is_random=True)
... data --> [batch_size, 2, row, col, 1]
>>> X_, Y_ = data.transpose((1,0,2,3,4))
... X_, Y_ --> [batch_size, row, col, 1]
>>> tl.visualize.images2d(images=np.asarray(X_), second=0.01, saveable=True, name=
↪'after', dtype=None)
>>> tl.visualize.images2d(images=np.asarray(Y_), second=0.01, saveable=True, name=
↪'before', dtype=None)
```

- Customized function for image segmentation

```
>>> def distort_img(data):
...     x, y = data
...     x, y = flip_axis_multi([x, y], axis=0, is_random=True)
...     x, y = flip_axis_multi([x, y], axis=1, is_random=True)
...     x, y = crop_multi([x, y], 100, 100, is_random=True)
...     return x, y
>>> X, Y --> [batch_size, row, col, channel]
>>> data = threading_data([_ for _ in zip(X, Y)], distort_img)
>>> X_, Y_ = data.transpose((1,0,2,3,4))
```

2.3.2 图像

- 这些函数只对一个图像做处理， 使用 `threading_data` 函数来实现多线程处理，请参考 `tutorial_image_preprocess.py` 。
- 所有函数都有一个 `is_random` 。
- 所有结尾是 *multi* 的函数通常用于图像分隔，因为输入和输出的图像必需是匹配的。

旋转

`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, default False

If True, randomly rotate.

row_index, col_index, channel_index : int

Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode : string

Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'

- scipy ndimage affine_transform

cval : scalar, optional

Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0

order : int, optional

The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

Examples

```
>>> x --> [row, col, 1] greyscale
>>> x = rotation(x, rg=40, is_random=False)
>>> tl.visualize.frame(x[:,:,0], second=0.01, saveable=True, name='temp',cmap=
↪'gray')
```

`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 : see `rotation`.

Examples

```
>>> x, y --> [row, col, 1]  greyscale
>>> x, y = rotation_multi([x, y], rg=90, is_random=False)
>>> tl.visualize.frame(x[:,:,0], second=0.01, saveable=True, name='x',cmap='gray')
>>> tl.visualize.frame(y[:,:,0], second=0.01, saveable=True, name='y',cmap='gray')
```

裁剪

`tensorlayer.prepro.`**`crop`**(*x, wrg, hrg, is_random=False, row_index=0, col_index=1, channel_index=2*)

Randomly or centrally crop an image.

Parameters **x** : numpy array

An image with dimension of [row, col, channel] (default).

wrg : float

Size of weight.

hrg : float

Size of height.

is_random : boolean, default False

If True, randomly crop, else central crop.

row_index, col_index, channel_index : int

Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

`tensorlayer.prepro.`**`crop_multi`**(*x*, *wrg*, *hrg*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*)

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 : see `crop`.

颠倒

`tensorlayer.prepro.`**`flip_axis`**(*x*, *axis*, *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

- 0, flip up and down
- 1, flip left and right
- 2, flip channel

is_random : boolean, default False

If True, randomly flip.

`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 : see `flip_axis`.

位移

`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, default False

If True, randomly shift.

row_index, col_index, channel_index : int

Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode : string

Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'.

- scipy ndimage affine_transform

cval : scalar, optional

Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.

order : int, optional

The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

`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 : see `shift`.

切变

`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, default False

If True, randomly shear.

row_index, col_index, channel_index : int

Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode : string

Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'.

- scipy ndimage affine_transform

cval : scalar, optional

Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.

order : int, optional

The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

`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 : see `shear`.

漩涡

`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 : (row, column) tuple or (2,) ndarray, optional

Center coordinate of transformation.

strength : float, optional

The amount of swirling applied.

radius : float, optional

The extent of the swirl in pixels. The effect dies out rapidly beyond radius.

rotation : float, (degree) optional

Additional rotation applied to the image, usually [0, 360], relates to center.

output_shape : tuple (rows, cols), optional

Shape of the output image generated. 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 : {'constant', 'edge', 'symmetric', 'reflect', 'wrap'}, optional

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, optional

Used in conjunction with mode 'constant', the value outside the image boundaries.

clip : bool, optional

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 : bool, optional

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, default False

If True, random swirl.

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

Examples

```
>>> x --> [row, col, 1] greyscale
>>> x = 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 : see `swirl`.

局部扭曲(Elastic transform)

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

Elastic deformation of images as described in [Simard2003] .

Parameters x : numpy array, a greyscale image.

alpha : scalar factor.

sigma : scalar or sequence of scalars, 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 : default constant, see scipy.ndimage.filters.gaussian_filter.

cval : float, optional. Used in conjunction with mode 'constant', the value outside the image boundaries.

is_random : boolean, default False

References

- Github.
- Kaggle

Examples

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

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

Elastic deformation of images as described in [Simard2003].

Parameters **x** : list of numpy array

others : see `elastic_transform`.

缩放

`tensorlayer.prepro.`**`zoom`**(*x*, *zoom_range=(0.9, 1.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Zoom in and out of a single image, randomly or non-randomly.

Parameters **x** : numpy array

An image with dimension of [row, col, channel] (default).

zoom_range : list or tuple

- If is_random=False, (h, w) are the fixed zoom factor for row and column axies, factor small than one is zoom in.
- If is_random=True, (min zoom out, max zoom out) for x and y with different random zoom in/out factor.

e.g (0.5, 1) zoom in 1~2 times.

is_random : boolean, default False

If True, randomly zoom.

row_index, col_index, channel_index : int

Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode : string

Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'.

- scipy ndimage affine_transform

cval : scalar, optional

Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.

order : int, optional

The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

`tensorlayer.prepro.`**`zoom_multi`**(*x*, *zoom_range=(0.9, 1.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

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 : see `zoom`.

亮度

`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, small than 1 means brighter.

Non negative real number. Default value is 1, smaller 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, default False

- If True, randomly change brightness.

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 numpy array

List of images with dimension of [n_images, row, col, channel] (default).

others : see `brightness`.

调整大小

`tensorlayer.prepro.`**`imresize`**(*x, size=[100, 100], 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 : int, float or tuple (h, w)

- int, Percentage of current size.
- float, Fraction of current size.
- tuple, Size of the output image.

interp : str, optional

Interpolation to use for re-sizing ('nearest', 'lanczos', 'bilinear', 'bicubic' or 'cubic').

mode : str, optional

The PIL image mode ('P', 'L', etc.) to convert arr before resizing.

Returns imresize : ndarray

The resized array of image.

References

- scipy.misc.imresize

正规化

`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 : 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 : set each sample mean to 0.

samplewise_std_normalization : divide each input by its std.

epsilon : small position value for dividing standard deviation.

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.

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
```

`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 : value for subtraction.

std : value for division.

epsilon : small position value for dividing standard deviation.

通道位移

`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, default False

If True, randomly shift.

channel_index : int

Index of channel, default 2.

`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 : see `channel_shift`.

噪声

`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 (0, 1)

The keeping probability, the lower more values will be set to zero.

手动变换

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

Return transform matrix offset center.

Parameters **matrix** : numpy array

Transform matrix

x, y : int

Size of image.

Examples

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

`tensorlayer.prepro.`**`apply_transform`**(*x*, *transform_matrix*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Return transformed images by given transform_matrix from `transform_matrix_offset_center`.

Parameters **x** : numpy array

Batch of images with dimension of 3, [batch_size, row, col, channel].

transform_matrix : numpy array

Transform matrix (offset center), can be generated by `transform_matrix_offset_center`

channel_index : int

Index of channel, default 2.

fill_mode : string

Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'

- scipy ndimage affine_transform

cval : scalar, optional

Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0

order : int, optional

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
- scipy ndimage affine_transform

Examples

- See `rotation`, `shift`, `shear`, `zoom`.

`tensorlayer.prepro.`**`projective_transform_by_points`**(*x*, *src*, *dst*, *map_args={}*, *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 (x, y).

dst : list or numpy

The coordinates after transformation, the number of coordinates is the same with src.

map_args : dict, optional

Keyword arguments passed to inverse_map.

output_shape : tuple (rows, cols), optional

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, optional

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 : {'constant', 'edge', 'symmetric', 'reflect', 'wrap'}, optional

Points outside the boundaries of the input are filled according to the given mode. Modes match the behaviour of numpy.pad.

cval : float, optional

Used in conjunction with mode 'constant', the value outside the image boundaries.

clip : bool, optional

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 : bool, optional

Whether to keep the original range of values. Otherwise, the input image is converted according to the conventions of img_as_float.

References

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

Examples

```
>>> Assume X is an image from CIFAR 10, i.e. shape == (32, 32, 3)
>>> src = [[0,0],[0,32],[32,0],[32,32]]
>>> dst = [[10,10],[0,32],[32,0],[32,32]]
>>> x = projective_transform_by_points(X, src, dst)
```

Numpy 与 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

A image with dimension of 3 and channels of 1 or 3.

dim_ordering : list or tuple of 3 int

Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

scale : boolean, default is True

If True, converts image to [0, 255] from any range of value like [-1, 2].

References

- PIL Image.fromarray

找轮廓

`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, {'low', '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 : 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.

一列点到图

tensorlayer.prepro.**pt2map**(*list_points=[]*, *size=(100, 100)*, *val=1*)
Inputs a list of points, return a 2D image.

Parameters **list_points** : list of [x, y].

size : tuple of (w, h) for output size.

val : float or int for the contour value.

二值膨胀

tensorlayer.prepro.**binary_dilation**(*x*, *radius=3*)
Return fast binary morphological dilation of an image. see skimage.morphology.binary_dilation.

Parameters **x** : 2D array image.

radius : int for the radius of mask.

灰度膨胀

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

Parameters **x** : 2D array image.

radius : int for the radius of mask.

二值腐蚀

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

Parameters **x** : 2D array image.

radius : int for the radius of mask.

灰度腐蚀

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

Parameters **x** : 2D array image.

radius : int for the radius of mask.

2.3.3 序列

更多相关函数，请见 `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 lists where each element is a sequence

maxlen : int, maximum length

dtype : type to cast the resulting sequence.

padding : 'pre' or 'post', pad either before or after each sequence.

truncating : '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** : numpy array with dimensions (number_of_sequences, maxlen)

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.

pad_id : 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** : numpy array or list of list with token IDs.

e.g. [[4,3,5,3,2,2,2,2], [5,3,9,4,9,2,2,3]]

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, default True.

Shorten the sequences.

remain_end_id : boolean, default False.

Keep an end_id in the end.

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.

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.

end_id : 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.

end_id : int.

pad_id : 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.

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]]
```

2.3.4 Tensor Opt

注解： 这几个函数将被弃用， 关于如何使用 Tensor Operator 请参考 tutorial_cifar10_tfrecord.py 。

tensorlayer.prepro.**distorted_images**(*images=None*, *height=24*, *width=24*)
Distort images for generating more training data.

Parameters **images** : 4D Tensor

The tensor or placeholder of images

height : int

The height for random crop.

width : int

The width for random crop.

Returns result : tuple of Tensor

(Tensor for distorted images, Tensor for while loop index)

Notes

- The first image in 'distorted_images' should be removed.

References

- tensorflow.models.image.cifar10.cifar10_input

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,␣
↪32, 32, 3), plotable=False)
>>> sess = tf.InteractiveSession()
>>> batch_size = 128
>>> x = tf.placeholder(tf.float32, shape=[batch_size, 32, 32, 3])
>>> distorted_images_op = tl.preprocess.distorted_images(images=x, height=24,␣
↪width=24)
>>> sess.run(tf.initialize_all_variables())
>>> feed_dict={x: X_train[0:batch_size,:,:,:]}
>>> distorted_images, idx = sess.run(distorted_images_op, feed_dict=feed_dict)
>>> tl.visualize.images2d(X_train[0:9,:,:,:], second=2, saveable=False, name=
↪'cifar10', dtype=np.uint8, fig_idx=20212)
>>> tl.visualize.images2d(distorted_images[1:10,:,:,:], second=10, saveable=False,
↪ name='distorted_images', dtype=None, fig_idx=23012)
```

`tensorlayer.prepro.`**`crop_central_whiten_images`**(*images=None*, *height=24*, *width=24*)

Crop the central of image, and normailize it for test data.

They are cropped to central of height * width pixels.

Whiten (Normalize) the images.

Parameters images : 4D Tensor

The tensor or placeholder of images

height : int

The height for central crop.

width : int

The width for central crop.

Returns result : tuple Tensor

(Tensor for distorted images, Tensor for while loop index)

Notes

The first image in 'central_images' should be removed.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,␣
↪32, 32, 3), plotable=False)
>>> sess = tf.InteractiveSession()
>>> batch_size = 128
>>> x = tf.placeholder(tf.float32, shape=[batch_size, 32, 32, 3])
>>> central_images_op = tl.preprocess.crop_central_whiten_images(images=x,␣
↪height=24, width=24)
>>> sess.run(tf.initialize_all_variables())
>>> feed_dict={x: X_train[0:batch_size,:,:,:]}
>>> central_images, idx = sess.run(central_images_op, feed_dict=feed_dict)
>>> tl.visualize.images2d(X_train[0:9,:,:,:], second=2, saveable=False, name=
↪'cifar10', dtype=np.uint8, fig_idx=20212)
>>> tl.visualize.images2d(central_images[1:10,:,:,:], second=10, saveable=False,␣
↪name='central_images', dtype=None, fig_idx=23012)
```

2.4 API - 迭代函数

数据迭代。

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 batchsize.
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 tutorial.

2.4.1 非时间序列

`tensorlayer.iterate.`**`minibatches`**(*inputs=None*, *targets=None*, *batch_size=None*, *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 batchsize.

Parameters **inputs** : numpy.array

24. The input features, every row is a example.

targets : numpy.array

25. The labels of inputs, every row is a example.

batch_size : int

The batch size.

shuffle : boolean

Indicating whether to use a shuffling queue, shuffle the dataset before return.

Notes

- If you have two inputs, e.g. X1 (1000, 100) and X2 (1000, 80), you can ‘‘np.hstack((X1, X2))

into (1000, 180) and feed into `inputs`, then you can split a batch of X1 and X2.

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]))
```

2.4.2 时间序列

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`, `seq_length = 5`, one return will have `500` rows (examples).

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

- Hint, if the input data are images, you can modify the code as follow.

```
from
data = np.zeros([batch_size, batch_len)
to
data = np.zeros([batch_size, batch_len, inputs.shape[1], inputs.shape[2], inputs.
↪shape[3]])
```

Parameters **inputs** : a list

the context in list format; note that context usually be represented by splitting by space, and then convert to unique word IDs.

targets : 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].

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.]]
```

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 tutorial. Yields (Returns) the source contexts and the target context by the given batch_size and num_steps (sequence_length).

see `PTB tutorial`.

e.g. x = [0, 1, 2] y = [1, 2, 3] , when batch_size = 1, num_steps = 3, raw_data = [i for i in range(100)]

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]]
```

2.5 API - 实用函数

fit(sess, network, train_op, cost, X_train, ...)	Traing a given non time-series network by the given cost function, training data, batch_size, n_epoch etc.
test(sess, network, acc, X_test, y_test, x, ...)	Test a given non time-series network by the given test data and metric.
predict(sess, network, X, x, y_op[, 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, max, 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, use for disable dropout, dropconnect layer and so on.
flatten_list([list_of_list])	Input a list of list, return a list that all items are in a list.

2.5.1 训练、测试及预测

训练

`tensorlayer.utils.`**`fit`**`(`*sess*, *network*, *train_op*, *cost*, *X_train*, *y_train*, *x*, *y_*, *acc=None*, *batch_size=100*, *n_epoch=100*, *print_freq=5*, *X_val=None*, *y_val=None*, *eval_train=True*, *tensorboard=False*, *tensorboard_epoch_freq=5*, *tensorboard_weight_histograms=True*, *tensorboard_graph_vis=True*`)`

Traing a given non time-series network by the given cost function, training data, batch_size, n_epoch etc.

Parameters **sess** : TensorFlow session

sess = tf.InteractiveSession()

network : a TensorLayer layer

the network will be trained

train_op : a TensorFlow optimizer

like tf.train.AdamOptimizer

X_train : numpy array

the input of training data

y_train : numpy array

the target of training data

x : placeholder

for inputs

y_ : placeholder

for targets

acc : the TensorFlow expression of accuracy (or other metric) or None

if None, would not display the metric

batch_size : int

batch size for training and evaluating

n_epoch : int

the number of training epochs

print_freq : int

display the training information every `print_freq` epochs

X_val : numpy array or None

the input of validation data

y_val : numpy array or None

the target of validation data

eval_train : boolean

if X_val and y_val are not None, it refects whether to evaluate the training data

tensorboard : boolean

if True summary data will be stored to the log/ direcory for visualization with tensorboard. See also detailed tensorboard_X settings for specific configurations of features. (default False) Also runs tl.layers.initialize_global_variables(sess) internally in fit() to setup the summary nodes, see Note:

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 visulaization 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)

Notes

If tensorboard=True, the global_variables_initializer will be run inside the fit function in order to initalize the automatically generated summary nodes used for tensorboard visualization, thus tf.global_variables_initializer().run() before the fit() call will be undefined.

Examples

```
>>> see tutorial_mnist_simple.py
>>> tl.utils.fit(sess, network, train_op, cost, X_train, y_train, x, y_,
...             acc=acc, batch_size=500, n_epoch=200, print_freq=5,
...             X_val=X_val, y_val=y_val, eval_train=False)
>>> tl.utils.fit(sess, network, train_op, cost, X_train, y_train, x, y_,
...             acc=acc, batch_size=500, n_epoch=200, print_freq=5,
...             X_val=X_val, y_val=y_val, eval_train=False,
...             tensorboard=True, tensorboard_weight_histograms=True, tensorboard_
↪graph_vis=True)
```

测试

`tensorlayer.utils.test` (*sess*, *network*, *acc*, *X_test*, *y_test*, *x*, *y_*, *batch_size*, *cost=None*)

Test a given non time-series network by the given test data and metric.

Parameters **sess** : TensorFlow session

sess = tf.InteractiveSession()

network : a TensorLayer layer

the network will be trained

acc : the TensorFlow expression of accuracy (or other metric) or None

if None, would not display the metric

X_test : numpy array

the input of test data

y_test : numpy array

the target of test data

x : placeholder

for inputs

y_ : placeholder

for targets

batch_size : int or None

batch size for testing, when dataset is large, we should use minibatche for testing. when dataset is small, we can set it to None.

cost : the TensorFlow expression of cost or None

if None, would not display the cost

Examples

```
>>> see tutorial_mnist_simple.py
>>> tl.utils.test(sess, network, acc, X_test, y_test, x, y_, batch_size=None,␣
↪cost=cost)
```

预测

`tensorlayer.utils.predict` (*sess*, *network*, *X*, *x*, *y_op*, *batch_size=None*)

Return the predict results of given non time-series network.

Parameters **sess** : TensorFlow session

sess = tf.InteractiveSession()

network : a TensorLayer layer

the network will be trained

X : numpy array

the input

x : placeholder

for inputs

y_op : placeholder

the argmax expression of softmax outputs

batch_size : int or None

batch size for prediction, when dataset is large, we should use minibatche for prediction. when dataset is small, we can set it to None.

Examples

```
>>> see tutorial_mnist_simple.py
>>> y = network.outputs
>>> y_op = tf.argmax(tf.nn.softmax(y), 1)
>>> print(tl.utils.predict(sess, network, X_test, x, y_op))
```

2.5.2 评估函数

`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** : numpy.array or list

target results

y_predict : numpy.array or list

predicted results

n_classes : int

number of classes

Examples

```
>>> c_mat, f1, acc, f1_macro = evaluation(y_test, y_predict, n_classes)
```

2.5.3 类平衡函数(class balancing)

`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

Features, each row is an example

y_train : numpy.array

Labels

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:]
```

2.5.4 随机函数

`tensorlayer.utils.get_random_int(min=0, max=10, number=5, seed=None)`

Return a list of random integer by the given range and quantity.

Examples

```
>>> r = get_random_int(min=0, max=10, number=5)
... [10, 2, 3, 3, 7]
```

2.5.5 辅助函数

设字典内容全为一

`tensorlayer.utils.dict_to_one(dp_dict={})`

Input a dictionary, return a dictionary that all items are set to one, use for disable dropout, dropconnect layer and so on.

Parameters dp_dict : dictionary

keeping probabilities

Examples

```
>>> dp_dict = dict_to_one( network.all_drop )
>>> dp_dict = dict_to_one( network.all_drop )
>>> feed_dict.update(dp_dict)
```

拉平列表

`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]
```

2.6 API - 自然语言处理

自然语言处理与词向量。

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.

Continued on next page

表 2.56 – continued from previous page

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, see create_vocab() and tutorial_tfrecord3.py.
process_sentence(sentence[, start_word, ...])	Converts 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, see create_vocab() and tutorial_tfrecord3.py.
simple_read_words([filename])	Read context from file without any preprocessing.
read_words([filename, replace])	File to list format context.
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 for converting 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])	Given a context (words) in list format and the vocabulary, Returns a list of IDs to represent the context.
word_ids_to_words(data, id_to_word)	Given a context (ids) in list format and the vocabulary, Returns a list of words to represent the context.
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.

2.6.1 训练嵌入矩阵的迭代函数

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.

Parameters **data** : a list

To present context.

batch_size : an int

Batch size to return.

num_skips : an int

How many times to reuse an input to generate a label.

skip_window : an int

How many words to consider left and right.

data_index : an int

Index of the context location. without using yield, this code use data_index to instead.

Returns **batch** : a list

Inputs

labels : a list

Labels

data_index : an int

Index of the context location.

References

- TensorFlow word2vec tutorial

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]]
```

2.6.2 抽样方法

简单抽样

`tensorlayer.nlp.`**`sample`**(*a=[]*, *temperature=1.0*)

Sample an index from a probability array.

Parameters **a** : a list

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 to avoid error.

从top k中抽样

`tensorlayer.nlp.`**`sample_top`**`(`*a=[], top_k=10*`)`

Sample from `top_k` probabilities.

Parameters **a** : a list

List of probabilities.

top_k : int

Number of candidates to be considered.

2.6.3 词的向量表示

词汇类 (class)

Simple vocabulary class

`class` `tensorlayer.nlp.`**`SimpleVocabulary`**`(`*vocab, unk_id*`)`

Simple vocabulary wrapper, see create_vocab().

Parameters **vocab** : A dictionary of word to word_id.

unk_id : Id of the special 'unknown' word.

Methods

`word_to_id`(word)	Returns the integer id of a word string.

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** : File containing the vocabulary, 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 : Special word denoting sentence start.

end_word : Special word denoting sentence end.

unk_word : Special word denoting unknown words.

Attributes

vocab	(a dictionary from word to id.)
reverse_vocab	(a list from id to word.)
start_id	(int of start id)
end_id	(int of end id)
unk_id	(int of unk id)
pad_id	(int of padding id)

Methods

`id_to_word`(word_id)	Returns the word string of an integer word id.
`word_to_id`(word)	Returns the integer word id of a word string.

Process sentence

`tensorlayer.nlp.`**`process_sentence`**(*sentence*, *start_word='<S>'*, *end_word='</S>'*)

Converts a sentence string into a list of string words, add start_word and end_word, see `create_vocab()` and `tutorial_tfrecord3.py`.

Returns A list of strings; the processed caption.

Notes

- You have to install the following package.
- Installing NLTK
- Installing NLTK data

Examples

```
>>> c = "how are you?"
>>> c = tl.nlp.process_sentence(c)
>>> print(c)
... ['<S>', 'how', 'are', 'you', '?', '</S>']
```

Create vocabulary

`tensorlayer.nlp.create_vocab`(*sentences*, *word_counts_output_file*, *min_word_count=1*)

Creates the vocabulary of word to word_id, see create_vocab() and `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** : a list of lists of strings.

word_counts_output_file : A string

The file name.

min_word_count : a int

Minimum number of occurrences for a word.

Returns

- tl.nlp.SimpleVocabulary object.

Notes

- See more `tl.nlp.build_vocab()`

Examples

```
>>> 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>']]
```

```
>>> tl.nlp.create_vocab(processed_capts, word_counts_output_file='vocab.txt', min_
↪word_count=1)
...   [TL] Creating vocabulary.
...   Total words: 8
...   Words in vocabulary: 8
...   Wrote vocabulary file: vocab.txt
>>> 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
```

2.6.4 从文件中读取文本

Simple read file

`tensorlayer.nlp.`**`simple_read_words`**(*filename='nietzsche.txt'*)
Read context from file without any preprocessing.

Parameters **filename** : a string

A file path (like .txt file)

Returns The context in a string

Read file

`tensorlayer.nlp.`**`read_words`**(*filename='nietzsche.txt', replace=['\n', '<eos>']*)
File to list format context. Note that, this script can not handle punctuations. For customized read_words method, see `tutorial_generate_text.py`.

Parameters **filename** : a string

A file path (like .txt file)

replace : a list

[original string, target string], to disable replace use [", "]

Returns The context in a list, split by space by default, and use `<eos>` to represent `\n`,

e.g. `[... 'how', 'useful', 'it', "'s" ... ]`.

References

- tensorflow.models.rnn.ptb.reader

2.6.5 从文件中读取类比题目

`tensorlayer.nlp.`**`read_analogies_file`**(*eval_file='questions-words.txt', word2id={}*)
Reads through an analogy question file, return its id format.

Parameters **eval_data** : a string

The file name.

word2id : a dictionary

Mapping words to unique IDs.

Returns **analogy_questions** : a [n, 4] numpy array containing the analogy question's

word ids. questions_skipped: questions skipped due to unknown words.

Examples

```
>>> eval_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
...
```

```
>>> 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]]
```

2.6.6 建立词汇表、文本与ID转换字典及文本ID化

为单词到ID建立字典

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 : a list of string

the context in list format

Returns word_to_id : a dictionary

mapping words to unique IDs. e.g. {'campbell': 2587, 'atlantic': 2247, 'aoun': 6746 .... }

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))
```

为ID到单词建立字典

`tensorlayer.nlp.`**`build_reverse_dictionary`**(*word_to_id*)

Given a dictionary for converting word to integer id. Returns a reverse dictionary for converting a id to word.

Parameters **word_to_id** : dictionary

mapping words to unique ids

Returns **reverse_dictionary** : a dictionary

mapping ids to words

建立字典，统计表等

`tensorlayer.nlp.`**`build_words_dataset`**(*words=[]*, *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** : a list of string 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 : an 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 : a string

Unknown words = unk_key

Returns **data** : a list of integer

The context in a list of ids

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)]

dictionary : a dictionary

word_to_id, mapping words to unique IDs.

reverse_dictionary : a dictionary

id_to_word, mapping id to unique word.

References

- tensorflow/examples/tutorials/word2vec/word2vec_basic.py

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)
```

保存词汇表

`tensorlayer.nlp.`**`save_vocab`**(*count=[], 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
```

2.6.7 文本转ID，ID转本文

These functions can be done by `Vocabulary` class.

单词到ID

`tensorlayer.nlp.`**`words_to_word_ids`**(*data=[], word_to_id={}, unk_key='UNK'*)
Given a context (words) in list format and the vocabulary, Returns a list of IDs to represent the context.

Parameters **data** : a list of string or byte

the context in list format

word_to_id : a dictionary

mapping words to unique IDs.

unk_key : a string

Unknown words = unk_key

Returns A list of IDs to represent the context.

References

- tensorflow.models.rnn.ptb.reader

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']
```

ID到单词

`tensorlayer.nlp.`**`word_ids_to_words`**(*data*, *id_to_word*)
Given a context (ids) in list format and the vocabulary, Returns a list of words to represent the context.

Parameters **data** : a list of integer

the context in list format

id_to_word : a dictionary

mapping id to unique word.

Returns A list of string or byte to represent the context.

Examples

```
>>> see words_to_word_ids
```

2.6.8 机器翻译相关函数

本文ID化

`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.

References

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

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)
>>>       print(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']
```

建立或读取词汇表

`tensorlayer.nlp.`**`create_vocabulary`**(*vocabulary_path, data_path, max_vocabulary_size, tokenizer=None, normalize_digits=True, _DIGIT_RE=re.compile(b'\\d'), _START_VOCAB=[b'_PAD', b'_GO', b'_EOS', b'_UNK']*)

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** : path where the vocabulary will be created.

data_path : data file that will be used to create vocabulary.

max_vocabulary_size : limit on the size of the created vocabulary.

tokenizer : 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 0s.

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:

dog

cat

will result in a vocabulary {"dog": 0, "cat": 1}, and this function will also return the reversed-vocabulary ["dog", "cat"].

Parameters **vocabulary_path** : path to the file containing the vocabulary.

Returns **vocab** : a dictionary

Word to id. A dictionary mapping string to integers.

rev_vocab : a list

Id to word. The reversed vocabulary (a list, which reverses the vocabulary mapping).

Raises **ValueError** : if the provided vocabulary_path does not exist.

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']
```

文本转ID，ID转本文

`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(), data_to_token_ids()

vocabulary : a dictionary mapping tokens to integers.

tokenizer : 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 0s.

Returns A list of integers, the token-ids for the sentence.

`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** : path to the data file in one-sentence-per-line format.

target_path : path where the file with token-ids will be created.

vocabulary_path : path to the vocabulary file.

tokenizer : 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 0s.

References

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

2.6.9 衡量指标

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** : A numpy array of strings where each string is a single example.

references : A numpy array of strings where each string is a single example.

lowercase : If true, pass the "-lc" flag to the multi-bleu script

Returns The BLEU score as a float32 value.

References

- Google/seq2seq/metric/bleu

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)
```

2.7 API - 强化学习

强化学习（增强学习）相关函数。

discount_episode_rewards([rewards, gamma, mode])	Take 1D float array of rewards and compute discounted rewards for an episode.
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.

2.7.1 奖励函数

`tensorlayer.rein.`**`discount_episode_rewards`**(*rewards=[]*, *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** : numpy list

a list of rewards

gamma : float

discounted factor

mode : int

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.

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.        ]
```

2.7.2 损失函数

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/ placeholder

The agent actions.

rewards : tensor/ placeholder

The rewards.

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

2.7.3 采样选择函数

`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** : a list of float.

The probability distribution of all actions.

action_list : None or a list of action in integer, string or others.

If None, returns an integer range between 0 and len(probs)-1.

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
```

2.8 API - 文件

下载基准(benchmark)数据集，保存加载模型和数据。 TensorFlow提供 `.ckpt` 文件格式来保存和加载模型，但为了更好地实现跨平台， 我们建议使用python标准文件格式 `.npz` 来保存和加载模型。

```
# 保存模型为 .ckpt
saver = tf.train.Saver()
save_path = saver.save(sess, "model.ckpt")
# 从 .ckpt 加载模型
saver = tf.train.Saver()
saver.restore(sess, "model.ckpt")

# 保存模型为 .npz
tl.files.save_npz(network.all_params , name='model.npz')

# 从 .npz 加载模型
load_params = tl.files.load_npz(path='', name='model.npz')
tl.files.assign_params(sess, load_params, network)

# 此外，你可以这样加载预训练的参数
# 加载第一个参数
tl.files.assign_params(sess, [load_params[0]], network)
# 加载前三个参数
tl.files.assign_params(sess, load_params[:3], network)
```

load_mnist_dataset([shape, path])	Automatically download MNIST dataset and return the training, validation and test set with 50000, 10000 and 10000 digit images respectively.
load_cifar10_dataset([shape, path, ...])	The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
load_ptb_dataset([path])	Penn TreeBank (PTB) dataset is used in many LANGUAGE MODELING papers, including "Empirical Evaluation and Combination of Advanced Language Modeling Techniques", "Recurrent Neural Network Regularization".
load_matt_mahoney_text8_dataset([path])	Download a text file from Matt Mahoney's website if not present, and make sure it's the right size.
load_imdb_dataset([path, nb_words, ...])	Load IMDB dataset
load_nietzsche_dataset([path])	Load Nietzsche dataset.
load_wmt_en_fr_dataset([path])	It will download English-to-French translation data from the WMT'15 Website (10^9-French-English corpus), and the 2013 news test from the same site as development set.
load_flickr25k_dataset([tag, path, ...])	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.

Continued on next page

表 2.60 – continued from previous page

load_flickr1M_dataset([tag, size, path, ...])	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.
load_cyclegan_dataset([filename, path])	Load image data from CycleGAN's database, see this link.
save_npz([save_list, name, sess])	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_params(sess, params, network)	Assign the given parameters to the TensorLayer network.
load_and_assign_npz([sess, name, network])	Load model from npz and assign to a network.
save_npz_dict([save_list, name, sess])	Input parameters and the file name, save parameters as a dictionary into .npz file.
load_and_assign_npz_dict([name, sess])	Restore the parameters saved by `tl.files.save_npz_dict()`.
save_ckpt([sess, mode_name, save_dir, ...])	Save parameters into ckpt file.
load_ckpt([sess, mode_name, save_dir, ...])	Load parameters from ckpt file.
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.
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,
natural_keys(text)	Sort list of string with number in human order.
npz_to_W_pdf([path, regx])	Convert the first weight matrix of .npz file to .pdf by using tl.visualize.W().

2.8.1 下载数据集

MNIST

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

path : string

The path that the data is downloaded to, defaults is `data/mnist/`.

Examples

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

CIFAR-10

`tensorlayer.files.`**`load_cifar10_dataset`**(*shape=(-1, 32, 32, 3)*, *path='data/cifar10/'*, *plotable=False*, *second=3*)

The CIFAR-10 dataset 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) , (-1, 32, 32, 3) , (-1, 32, 32, 3)

plotable : True, False

Whether to plot some image examples.

second : int

If `plotable` is True, `second` is the display time.

path : string

The path that the data is downloaded to, defaults is `data/cifar10/`.

References

- CIFAR website
- Data download link
- Code references

Examples

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

Penn TreeBank (PTB)

`tensorlayer.files.`**`load_ptb_dataset`**(*path='data/ptb/'*)

Penn TreeBank (PTB) dataset 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** : : string

The path that the data is downloaded to, defaults is `data/ptb/`.

Returns train_data, valid_data, test_data, vocabulary size

References

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

Examples

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

Matt Mahoney's text8

`tensorlayer.files.`**`load_matt_mahoney_text8_dataset`**(*path='data/mm_test8/'*)

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** : : string

The path that the data is downloaded to, defaults is `data/mm_test8/`.

Returns **word_list** : a list

a list of string (word).

e.g. [.... 'their', 'families', 'who', 'were', 'expelled', 'from', 'jerusalem', ...]

Examples

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

IMBD

`tensorlayer.files.`**`load_imdb_dataset`**(*path='data/imdb/', 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** : : string

The path that the data is downloaded to, defaults is `data/imdb/`.

References

- Modified from keras.

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]
```

Nietzsche

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

Load Nietzsche dataset. Returns a string.

Parameters path : string

The path that the data is downloaded to, defaults is `data/nietzsche/`.

Examples

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

WMT'15 Website 的英文译法文数据

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

It will download English-to-French translation 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 : string

The path that the data is downloaded to, defaults is `data/wmt_en_fr/`.

Notes

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

References

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

Flickr25k

`tensorlayer.files.`**`load_flickr25k_dataset`**(*tag='sky'*, *path='data/flickr25k'*, *n_threads=50*, *printable=False*)

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** : string or None

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 : string

The path that the data is downloaded to, defaults is `data/flickr25k/`.

n_threads : int, number of thread to read image.

printable : bool, 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/flickr1M', n_threads=50, printable=False*)

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** : string or None

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 1 to 10.

1 means 100k images ... 5 means 500k images, 10 means all 1 million images. Default is 10.

path : string

The path that the data is downloaded to, defaults is `data/flickr25k/`.

n_threads : int, number of thread to read image.

printable : bool, 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/cyclegan'*)

Load image data from CycleGAN's database, see this link.

Parameters **filename** : string

The dataset you want, see this link.

path : string

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')
```

2.8.2 保存与加载模型

以列表保存模型到 .npz

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

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

Parameters **save_list** : a list

Parameters want to be saved.

name : a string or None

The name of the .npz file.

sess : None or Session

Notes

If you got session issues, you can change the value.eval() to value.eval(session=sess)

References

- Saving dictionary using numpy

Examples

```
>>> tl.files.save_npz(network.all_params, name='model_test.npz', sess=sess)
... File saved to: model_test.npz
>>> load_params = tl.files.load_npz(name='model_test.npz')
... Loading param0, (784, 800)
... Loading param1, (800,)
... Loading param2, (800, 800)
... Loading param3, (800,)
... Loading param4, (800, 10)
... Loading param5, (10,)
>>> put parameters into a TensorLayer network, please see assign_params()
```

从save_npz加载模型参数列表

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

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

Parameters **path** : a string

Folder path to .npz file.

name : a string or None

The name of the .npz file.

Returns **params** : list

A list of parameters in order.

References

- Saving dictionary using numpy

Examples

- See save_npz and assign_params

把模型参数载入模型

`tensorlayer.files.`**`assign_params`**(*sess*, *params*, *network*)

Assign the given parameters to the TensorLayer network.

Parameters **sess** : TensorFlow Session. Automatically run when sess is not None.

params : a list

A list of parameters in order.

network : a `Layer` class

The network to be assigned

Returns **ops** : list

A list of tf ops in order that assign params. Support sess.run(ops) manually.

References

- Assign value to a TensorFlow variable

Examples

```
>>> Save your network as follow:
>>> tl.files.save_npz(network.all_params, name='model_test.npz')
>>> network.print_params()
...
... Next time, load and assign your network as follow:
>>> tl.layers.initialize_global_variables(sess)
>>> load_params = tl.files.load_npz(name='model_test.npz')
>>> tl.files.assign_params(sess, load_params, network)
>>> network.print_params()
```

从.npz中加载参数并导入模型

`tensorlayer.files.`**`load_and_assign_npz`**(*sess=None*, *name=None*, *network=None*)
Load model from npz and assign to a network.

Parameters **sess** : TensorFlow Session

name : string

Model path.

network : a `Layer` class

The network to be assigned

Returns Returns False if faild to model is not exist.

Examples

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

以字典保存模型到 .npz

`tensorlayer.files.`**`save_npz_dict`**(*save_list=[]*, *name='model.npz'*, *sess=None*)
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** : a list to tensor for parameters

Parameters want to be saved.

name : a string

The name of the .npz file.

sess : Session

从save_npz_dict加载模型参数列表

`tensorlayer.files.`**`load_and_assign_npz_dict`**`(`*name='model.npz'*, *sess=None*`)`

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

Parameters **name** : a string

The name of the .npz file.

sess : Session

以列表保存模型到 .ckpt

`tensorlayer.files.`**`save_ckpt`**`(`*sess=None*, *mode_name='model.ckpt'*, *save_dir='checkpoint'*, *var_list=[]*, *global_step=None*, *printable=False*`)`

Save parameters into ckpt file.

Parameters **sess** : Session.

mode_name : string, name of the model, default is `model.ckpt`.

save_dir : string, path / file directory to the ckpt, default is `checkpoint`.

var_list : list of variables, if not given, save all global variables.

global_step : int or None, step number.

printable : bool, if True, print all params info.

Examples

- see `tl.files.load_ckpt()`.

从.ckpt中加载参数并导入模型

`tensorlayer.files.`**`load_ckpt`**`(`*sess=None*, *mode_name='model.ckpt'*, *save_dir='checkpoint'*, *var_list=[]*, *is_latest=True*, *printable=False*`)`

Load parameters from ckpt file.

Parameters **sess** : Session.

mode_name : string, name of the model, default is `model.ckpt`.

Note that if `is_latest` is True, this function will get the `mode_name` automatically.

save_dir : string, path / file directory to the ckpt, default is `checkpoint`.

var_list : list of variables, if not given, save all global variables.

is_latest : bool, if True, load the latest ckpt, if False, load the ckpt with the name of `` `mode_name``.

printable : bool, if True, print all params info.

Examples

- Save all global parameters.

```
>>> tl.files.save_ckpt(sess=sess, mode_name='model.ckpt', save_dir='model',␣
↪printable=True)
- Save specific parameters.
>>> tl.files.save_ckpt(sess=sess, mode_name='model.ckpt', var_list=net.all_params,
↪ save_dir='model', printable=True)
- Load latest ckpt.
>>> tl.files.load_ckpt(sess=sess, var_list=net.all_params, save_dir='model',␣
↪printable=True)
- Load specific ckpt.
>>> tl.files.load_ckpt(sess=sess, mode_name='model.ckpt', var_list=net.all_params,
↪ save_dir='model', is_latest=False, printable=True)
```

2.8.3 保存与加载数据

保持数据到.npy文件

`tensorlayer.files.`**`save_any_to_npy`**(*save_dict={}, name='file.npy'*)
Save variables to .npy file.

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']}
```

从.npy文件加载数据

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

Examples

- see save_any_to_npy()

2.8.4 文件夹/文件相关函数

判断文件存在

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

判断文件夹存在

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

删除文件

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

删除文件夹

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

读取文件

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

Examples

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

从文件夹中读取文件名列表

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

Parameters **path** : a string or None

A folder path.

regx : a string

The regx of file name.

printable : boolean, whether to print the files infomation.

Examples

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

从文件夹中读取文件夹列表

`tensorlayer.files.load_folder_list`(*path=''*)
Return a folder list in a folder by given a folder path.

Parameters **path** : a string or None

A folder path.

查看或建立文件夹

`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** : a string

A folder path.

verbose : boolean

If True, prints results, deaults is True

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

Examples

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

下载或解压

`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** : string

The name of the (to be) dowloaded file.

working_directory : string

A folder path to search for the file in and dowload the file to

url : string

The URL to download the file from

extract : bool, defaults is False

If True, tries to uncompress the dowloaded file is ".tar.gz/.tar.bz2" or ".zip" file

expected_bytes : int/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 filepath to dowloaded (uncompressed) file

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)
```

2.8.5 排序

字符串按数字排序

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

Sort list of string with number in human order.

References

alist.sort(key=natural_keys) sorts in human order http://nedbatchelder.com/blog/200712/human_sorting.html (See Toothy's implementation in the comments)

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']
```

2.8.6 可视化 npz 文件

`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** : a string or None

A folder path to npz files.

regx : a string

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)')
```

2.9 API - 可视化

TensorFlow 提供了可视化模型和激活输出等的工具 TensorBoard。 在这里，我们进一步提供一些可视化模型参数和数据的函数。

`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 one image.
`save_images`(images, size[, image_path])	Save mutiple images into one single image.
`W`([W, second, saveable, shape, name, fig_idx])	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(image).
`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.

2.9.1 读取与保存图片

读取单个图片

`tensorlayer.visualize.`**`read_image`**(*image*, *path=''*)

Read one image.

Parameters **images** : string, file name.

path : string, path.

读取多个图片

`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 string, the image file names.

path : string, image folder path.

n_threads : int, number of thread to read image.

printable : bool, print infomation when reading images, default is True.

保存单个图片

`tensorlayer.visualize.`**`save_image`**(*image*, *image_path=''*)

Save one image.

Parameters **images** : numpy array [w, h, c]

image_path : string.

保存多个图片

`tensorlayer.visualize.`**`save_images`**(*images*, *size*, *image_path=''*)

Save mutiple images into one single image.

Parameters **images** : numpy array [batch, w, h, c]

size : list of two int, row and column number.

number of images should be equal or less than size[0] * size[1]

image_path : string.

Examples

```
>>> images = np.random.rand(64, 100, 100, 3)
>>> tl.visualize.save_images(images, [8, 8], 'temp.png')
```

2.9.2 可视化模型参数

可视化Weight Matrix

`tensorlayer.visualize.`**`W`**(*W=None, second=10, saveable=True, shape=[28, 28], 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

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.W(network.all_params[0].eval(), second=10, saveable=True, name=
↪'weight_of_1st_layer', fig_idx=2012)
```

可视化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 : a string

A name to save the image, if saveable is True.

fig_idx : int

matplotlib figure index.

Examples

```
>>> tl.visualize.CNN2d(network.all_params[0].eval(), second=10, saveable=True,␣
↪name='cnn1_mnist', fig_idx=2012)
```

2.9.3 可视化图像

matplotlib显示单图

`tensorlayer.visualize.`**`frame`**(*I=None*, *second=5*, *saveable=True*, *name='frame'*, *cmap=None*, *fig_idx=12836*)

Display a frame(image). 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 : a string

A name to save the image, if saveable is True.

cmap : None or string

'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)
```

matplotlib显示多图

`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 : a string

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)
```

2.9.4 可视化词嵌入矩阵

`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** : a matrix

The images.

reverse_dictionary : a 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 : a string

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')
```

2.10 API - 操作系统管理

系统操作，更多函数可在 TensorFlow API 中找到。

exit_tf([sess, port])	Close TensorFlow session, TensorBoard and Nvidia-process if available.
open_tb([logdir, port])	Open Tensorboard.
clear_all([printable])	Clears all the placeholder variables of keep prob, including keeping probabilities of all dropout, denoising, dropconnect etc.
set_gpu_fraction([sess, gpu_fraction])	Set the GPU memory fraction for the application.
disable_print()	Disable console output, `suppress_stdout` is recommended.
enable_print()	Enable console output, `suppress_stdout` is recommended.
suppress_stdout()	Temporarily disable console output.
get_site_packages_directory()	Print and return the site-packages directory.
empty_trash()	Empty trash folder.

2.10.1 TensorFlow 操作函数

中断 Nvidia 进程

`tensorlayer.ops.`**`exit_tf`**(*sess=None*, *port=6006*)

Close TensorFlow session, TensorBoard and Nvidia-process if available.

Parameters **sess** : a session instance of TensorFlow

TensorFlow session

tb_port : an integer

TensorBoard port you want to close, 6006 as default.

打开 TensorBoard

`tensorlayer.ops.`**`open_tb`**`(`*logdir='/tmp/tensorflow', port=6006*`)`

Open Tensorboard.

Parameters **logdir** : a string

Directory where your tensorboard logs are saved

port : an integer

TensorBoard port you want to open, 6006 is tensorboard default

删除 placeholder

`tensorlayer.ops.`**`clear_all`**`(`*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.

2.10.2 GPU 配置函数

`tensorlayer.ops.`**`set_gpu_fraction`**`(`*sess=None, gpu_fraction=0.3*`)`

Set the GPU memory fraction for the application.

Parameters **sess** : a session instance of TensorFlow

TensorFlow session

gpu_fraction : a float

Fraction of GPU memory, (0 ~ 1]

References

- TensorFlow using GPU

2.10.3 命令窗口显示

禁止 print

`tensorlayer.ops.`**`disable_print`**`()`

Disable console output, `suppress_stdout` is recommended.

Examples

```
>>> print("You can see me")
>>> tl.ops.disable_print()
>>> print(" You can't see me")
>>> tl.ops.enable_print()
>>> print("You can see me")
```

允许 print

`tensorlayer.ops.`**`enable_print`**`()`
Enable console output, `suppress_stdout` is recommended.

Examples

- see tl.ops.disable_print()

临时禁止 print

`tensorlayer.ops.`**`suppress_stdout`**`()`
Temporarily disable console output.

References

- stackoverflow

Examples

```
>>> print("You can see me")
>>> with tl.ops.suppress_stdout():
>>>     print("You can't see me")
>>> print("You can see me")
```

2.10.4 Site packages 信息

`tensorlayer.ops.`**`get_site_packages_directory`**`()`
Print and return the site-packages directory.

Examples

```
>>> loc = tl.ops.get_site_packages_directory()
```

2.10.5 垃圾管理

`tensorlayer.ops.`**`empty_trash`**`()`
Empty trash folder.

2.11 API - 激活函数

为了尽可能地保持TensorLayer的简洁性，我们最小化激活函数的数量，因此我们鼓励用户直接使用 TensorFlow官方的函数，比如 `tf.nn.relu`, `tf.nn.relu6`, `tf.nn.elu`, `tf.nn.softplus`, `tf.nn.softsign` 等等。更多TensorFlow官方激活函数请看 这里.

2.11.1 自定义激活函数

在TensorLayer中创造自定义激活函数非常简单。

下面的例子实现了把输入乘以2。对于更加复杂的激活函数，你需要用到TensorFlow的API。

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

identity(x[, name])	The identity activation function, Shortcut is `linear`.
ramp([x, v_min, v_max, name])	The ramp activation function.
leaky_relu([x, alpha, name])	The LeakyReLU, Shortcut is `lrelu`.
pixel_wise_softmax(output[, name])	Return the softmax outputs of images, every pixels have multiple label, the sum of a pixel is 1.

2.11.2 Identity

`tensorlayer.activation.`**`identity`**(*x*, *name=None*)

The identity activation function, Shortcut is `linear`.

Parameters **x** : a tensor input

input(s)

Returns A *Tensor* with the same type as *x*.

2.11.3 Ramp

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

The ramp activation function.

Parameters **x** : a tensor input

input(s)

v_min : float

if input(s) smaller than v_min, change inputs to v_min

v_max : float

if input(s) greater than v_max, change inputs to v_max

name : a string or None

An optional name to attach to this activation function.

Returns A *Tensor* with the same type as *x*.

2.11.4 Leaky Relu

`tensorlayer.activation.`**`leaky_relu`**(*x=None*, *alpha=0.1*, *name='LeakyReLU'*)

The LeakyReLU, Shortcut is `lrelu`.

Modified version of ReLU, introducing a nonzero gradient for negative input.

Parameters **x** : A *Tensor* with type *float*, *double*, *int32*, *int64*, *uint8*, *int16*, or *int8*.

alpha : *float*. slope.

name : a string or None

An optional name to attach to this activation function.

References

- Rectifier Nonlinearities Improve Neural Network Acoustic Models, Maas et al. (2013)

Examples

```
>>> network = tl.layers.DenseLayer(network, n_units=100, name = 'dense_lrelu',
...                 act= lambda x : tl.act.lrelu(x, 0.2))
```

2.11.5 Pixel-wise Softmax

tensorlayer.activation.**pixel_wise_softmax**(*output*, *name='pixel_wise_softmax'*)

Return the softmax outputs of images, every pixels have multiple label, the sum of a pixel is 1. Usually be used for image segmentation.

Parameters **output** : tensor

- For 2d image, 4D tensor [batch_size, height, weight, channel], channel >= 2.
- For 3d image, 5D tensor [batch_size, depth, height, weight, channel], channel >= 2.

References

- tf.reverse

Examples

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

2.11.6 带有参数的激活函数

请见神经网络层。

2.12 API - 数据库

Alpha 版本的数据管理系统已经发布，详情请见 英文文档 .

CHAPTER 3

索引与附录

- genindex
- modindex
- search

Python 模块索引

索引