# 一、深度学习入门

## 比较好的资源：

Cs231n，cs20sI(or cs20s1),需要英语

网易：吴恩达的课

文档：tensorflow官方文档，

极客学院：官方文档翻译。

## 配置tensorflow环境

CPU版(Win)

pip install --ignore-installed --upgrade <https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.0-cp35-cp35m-win_amd64.whl>

GPU版(Win)

pip install --ignore-installed --upgrade <https://storage.googleapis.com/tensorflow/windows/gpu/tensorflow_gpu-1.0.0-cp35-cp35m-win_amd64.whl>

也可以用清华镜像网站，但是资源更新略慢：https://mirrors.tuna.tsinghua.edu.cn/help/tensorflow/

提示缺少库可以在这里下载，

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

运行pip install fileName即可

Tflearn 可以在这里下载<https://github.com/tflearn/tflearn/archive/master.zip>

python setup.py install 即可构建和安装该程序包。

## Anaconda

列出环境：conda env list

进入环境：activate env

列出环境内包：conda list

退出环境：deactivate

删除环境: conda env remove -n env\_name;

创建环境：conda create -n env\_name list of packages.

也可以指定python的版本：conda create -n py3.6 python=3.6:

在环境内安装包：conda install packagename

首次安装anaconda后，运行prompt，运行命令 conda upgrade –all来更新所有包，避免后面运行时出错。

**安装包**：请在终端中键入 conda install package\_name。例如，要安装 numpy，请键入 conda install numpy。

你还可以同时安装多个包。类似 conda install numpy scipy pandas 的命令会同时安装所有这些包。还可以通过添加版本号（例如 conda install numpy=1.10）来指定所需的包版本。

Conda 还会自动为你安装依赖项。例如，scipy 依赖于 numpy，因为它使用并需要 numpy。如果你只安装 scipy (conda install scipy)，则 conda 还会安装 numpy（如果尚未安装的话）。

**卸载包**

大多数命令都是很直观的。要卸载包，请使用 conda remove package\_name。要更新包，请使用 conda update package\_name。如果想更新环境中的所有包（这样做常常很有用），请使用 conda update --all。

最后，要**列出**已安装的包，请使用前面提过的 conda list。

如果不知道要找的包的确切名称，可以尝试使用 conda search search\_term 进行搜索。例如，我知道我想安装 [**Beautiful Soup**](https://www.crummy.com/software/BeautifulSoup/)，但我不清楚确切的包名称。因此，我尝试执行 conda search beautifulsou

**环境管理**

如前所述，可以使用 conda 创建环境以隔离项目。要创建环境，请在终端中使用 conda create -n env\_name list of packages。在这里，-n env\_name 设置环境的名称（-n 是指名称），而 list of packages 是要安装在环境中的包的列表。例如，要创建名为 my\_env 的环境并在其中安装 numpy，请键入 conda create -n my\_env numpy。

创建环境时，可以指定要安装在环境中的 Python 版本。这在你同时使用 Python 2.x 和 Python 3.x 中的代码时很有用。要创建具有特定 Python 版本的环境，请键入类似于 conda create -n py3 python=3 或 conda create -n py2 python=2 的命令。实际上，我在我的个人计算机上创建了这两个环境。我将它们用作与任何特定项目均无关的通用环境，以处理普通的工作（可轻松使用每个 Python 版本）。这些命令将分别安装 Python 3 和 2 的最新版本。要安装特定版本（例如 Python 3.3），请使用conda create -n py python=3.3。

### 进入环境

创建了环境后，在 OSX/Linux 上使用 source activate my\_env 进入环境。在 Windows 上，请使用 activate my\_env。

### 添加国内源

Conda config –add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/

Conda config –set show\_channel\_urls yes

删除源

Conda config –remove

这些操作可以在界面上实现。

### 保存和加载环境

共享环境这项功能确实很有用，它能让其他人安装你的代码中使用的所有包，并确保这些包的版本正确。可以使用 conda env export > environment.yaml 将包保存为 [**YAML**](http://www.yaml.org/)。第一部分 conda env export 输出环境中的所有包的名称（包括 Python 版本）。

要通过环境文件创建环境，请使用 conda env create -f nvironment.yaml。这会创建一个新环境，而且它具有同样的在environment.yaml 中列出的库。

### 列出环境

如果忘记了环境的名称（我有时会这样），可以使用 conda env list 列出你创建的所有环境。你会看到环境的列表，而且你当前所在环境的旁边会有一个星号。默认的环境（即当你不在环境中时使用的环境）名为 root。

### 删除环境

如果你不再使用某些环境，可以使用 conda env remove -n env\_name 删除指定的环境（在这里名为 env\_name）。

### 配置tensflow 和Tflearn环境

conda create -n tflearn python=3.5

activate tflearn

conda install numpy pandas jupyter notebook matplotlib

conda install scipy h5py

pip install tensorflow

pip install TFLearn

如果你遇到 ImportError: No module named '\_curses' 这一错误，请尝试执行以下步骤：

转到[**此链接**](http://www.lfd.uci.edu/~gohlke/pythonlibs/#curses)，找到与你的 Windows 系统版本对应的 Curses 文件并下载该文件。 下载完毕后，运行 pip install FILENAME，其中 FILENAME 是你刚刚下载的文件的名称。

如果不能解决问题，或者出现其他安装问题，请尝试：

学员 Andreas Wienzek 发现从源文件中安装 TFLearn 解决了问题。如果你已经用 pip 安装了 TFLearn，请用 pip uninstall tflearn 卸载该 TFLearn。然后，你可以[**在此处下载源代码**](https://github.com/tflearn/tflearn/archive/master.zip)并解压。转到相关目录下，然后运行python setup.py install 即可构建和安装该程序包。

### 环境问题

module “importlib.\_bootstrap” has no attribute “SourceFileLoader”

stack overflow给的建议是版本不符：

<https://stackoverflow.com/questions/44761958/using-pip3-module-importlib-bootstrap-has-no-attribute-sourcefileloader>

执行的了命令：python -m ensurepip --upgrade

解决了部分问题

搞完了又报别的错module 'socketio' has no attribute 'Server'，，然后Google，用这里的方法解决了<https://github.com/miguelgrinberg/Flask-SocketIO/issues/161>

即运行以下命令

pip uninstall gevent-socketio,

pip install --upgrade flask-socketio

## 最佳做法

### 使用环境

对我帮助很大的一点是，我的 Python 2 和 Python 3 具有独立的环境。我使用了 conda create -n py2 python=2 和 conda create -n py3 python=3 创建两个独立的环境，即 py2 和 py3。现在，我的每个 Python 版本都有一个通用环境。在所有这些环境中，我都安装了大多数标准的数据科学包（numpy、scipy、pandas 等）。

我还发现，为我从事的每个项目创建环境很有用。这对于与数据不相关的项目（例如使用 Flask 开发的 Web 应用）也很有用。例如，我为我的个人博客（使用 [**Pelican**](http://docs.getpelican.com/en/stable/)）创建了一个环境。

### 共享环境

在 GitHub 上共享代码时，最好同样创建环境文件并将其包括在代码库中。这能让其他人更轻松地安装你的代码的所有依赖项。对于不使用 conda 的人，我通常还会使用 pip freeze（[**在此处了解详情**](https://pip.pypa.io/en/stable/reference/pip_freeze/)）将一个 pip requirements.txt 文件包括在内。

### 了解更多信息

要详细了解 conda 和它如何融入到 Python 生态系统中，请查看这篇由 Jake Vanderplas 撰写的文章：[**Conda myths and misconceptions**](https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/)（有关 conda 的迷思和误解）。此外，有空也可以参考这篇 [**conda 文档**](http://conda.pydata.org/docs/using/index.html)。

我们认为最佳学习方法是了解不同人对同一概念提出的不同见解。因此，我们建议你阅读以下几本图书，进一步了解深度学习。

* [**Grokking Deep Learning**](https://www.manning.com/books/grokking-deep-learning)，作者：Andrew Trask。使用我们的专属折扣码 **traskud17** 可以享受六折优惠。这本书对深度学习进行了基本介绍，更加直观，而不是只介绍抽象理论。
* [**Neural Networks And Deep Learning**](http://neuralnetworksanddeeplearning.com/)，作者：Michael Nielsen。这本书比《Grokking Deep Learning》更严密，包含大量的互动性有趣图表。
* [**The Deep Learning Textbook**](http://www.deeplearningbook.org/)，作者：Ian Goodfellow、Yoshua Bengio 和 Aaron Courville。这本在线图书包含了大量的资料，是三本书中最严密的一本。

## jupyter notebook

Y M可以在markdown和代码模式之间切换

A在上方，B在下方创建空白单元格

MarkDown模式下，#写标题。\* \*之间的字为斜体；\*\* \*\*之间的字体加粗，>符号代表退格并加符号。

代码模式下，L显示行号。

按两次D可以删除单元格。

S保存

Shift+Tab可以查看函数说明

### Magic 关键字

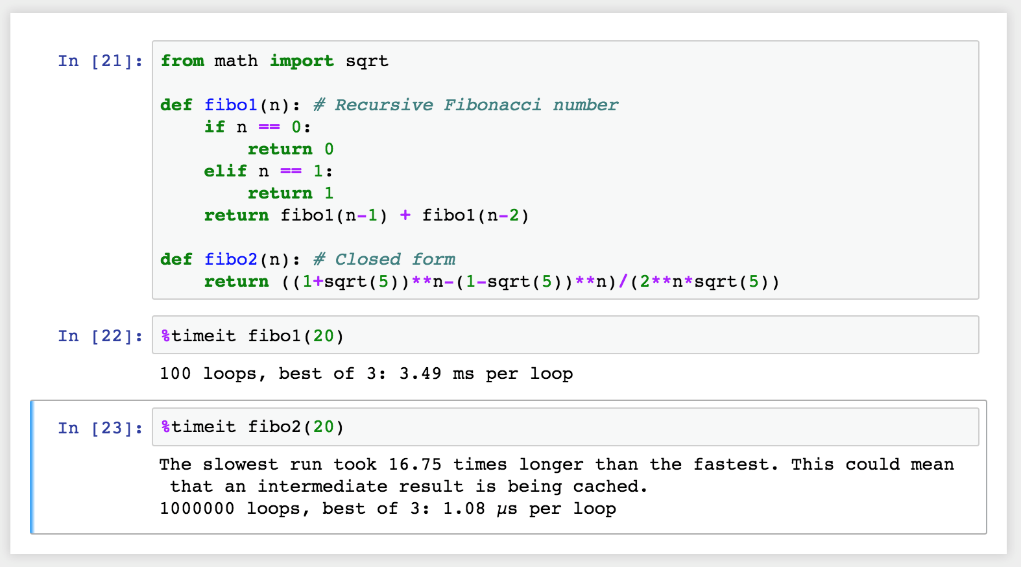
Magic 关键字是可以在单元格中运行的特殊命令，能让你控制 notebook 本身或执行系统调用（例如更改目录）。例如，可以使用%matplotlib 将 matplotlib 设置为以交互方式在 notebook 中工作。

Magic 命令的前面带有一个或两个百分号（% 或 %%），分别对应行 Magic 命令和单元格 Magic 命令。行 Magic 命令仅应用于编写 Magic 命令时所在的行，而单元格 Magic 命令应用于整个单元格。

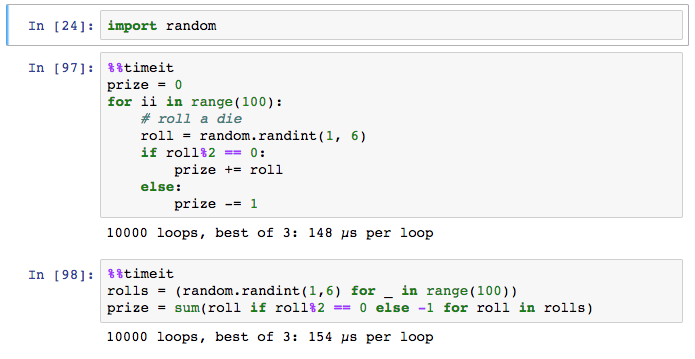
**注意：**这些 Magic 关键字是特定于普通 Python 内核的关键字。如果使用其他内核，这些关键字很有可能无效。

### 代码计时

有时候，你可能要花些精力优化代码，让代码运行得更快。在此优化过程中，必须对代码的运行速度进行计时。可以使用 Magic 命令 timeit 测算函数的运行时间，如下所示：



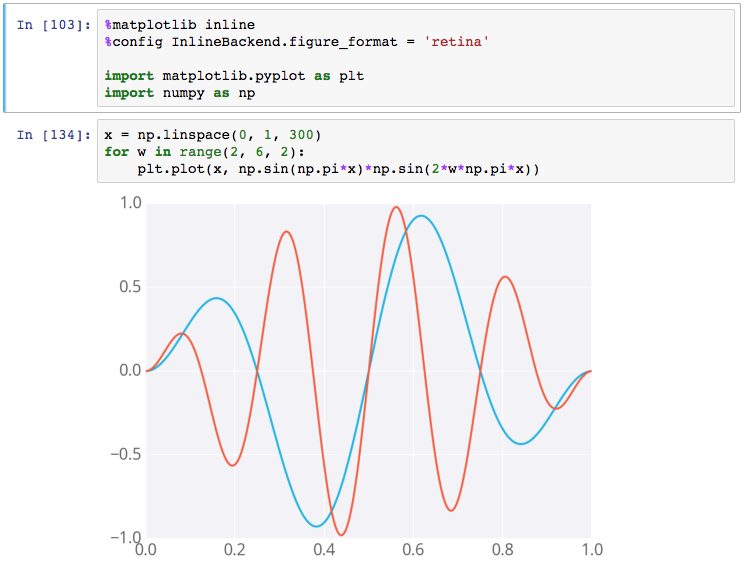
如果要测算整个单元格的运行时间，请使用 %%timeit，如下所示：



### 在 notebook 中嵌入可视化内容

如前所述，notebook 允许你将图像与文本和代码一起嵌入。这在你使用 matplotlib 或其他绘图包创建可视化内容时最为有用。可以使用 %matplotlib 将 matplotlib 设置为以交互方式在 notebook 中工作。默认情况下，图形呈现在各自的窗口中。但是，可以向命令传递参数，以选择特定的[**“后端”**](http://matplotlib.org/faq/usage_faq.html#what-is-a-backend)（呈现图像的软件）。要直接在 notebook 中呈现图形，应将内联后端与命令%matplotlib inline 一起使用。

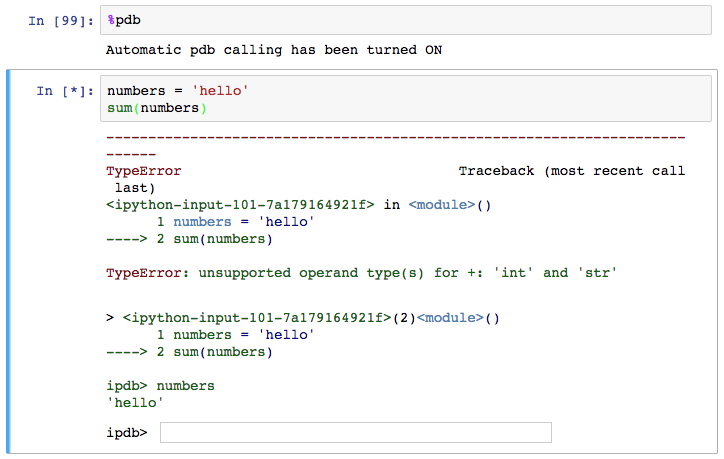
**提示：**在分辨率较高的屏幕（例如 Retina 显示屏）上，notebook 中的默认图像可能会显得模糊。可以在 %matplotlib inline 之后使用 %config InlineBackend.figure\_format = 'retina' 来呈现分辨率较高的图像。



notebook 中的图形示例

### 在 notebook 中进行调试

对于 Python 内核，可以使用 Magic 命令 %pdb 开启交互式调试器。出错时，你能检查当前命名空间中的变量。



在 notebook 中进行调试

在上图中，可以看到我尝试对字符串求和，这造成了错误。调试器指出了该错误，并提示你检查代码。

要详细了解 pdb，请阅读[**此文档**](https://docs.python.org/3/library/pdb.html)。要退出调试器，在提示符中输入 q 即可。

## 格式转换

由于 notebook 是 JSON 文件，因此，可以轻松将其转换为其他格式。Jupyter 附带了一个名为 nbconvert 的实用程序，可将 notebook 转换为 HTML、Markdown、幻灯片等格式。

例如，要将 notebook 转换为 HTML 文件，请在终端中使用

jupyter nbconvert --to html notebook.ipynb

要将 notebook 与不使用 notebook 的其他人共享，转换为 HTML 很有用。而要在博客和其他接受 Markdown 格式化的文本编辑器中显示 notebook，Markdown 很合适。

## 线性回归示例

# TODO: Add import statements

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Assign the dataframe to this variable.

# TODO: Load the data

bmi\_life\_data = pd.read\_csv("bmi\_and\_life\_expectancy.csv")

# Make and fit the linear regression model

#TODO: Fit the model and Assign it to bmi\_life\_model

bmi\_life\_model = LinearRegression()

bmi\_life\_model.fit(bmi\_life\_data[['BMI']],bmi\_life\_data[['Life expectancy']])

# Mak a prediction using the model

# TODO: Predict life expectancy for a BMI value of 21.07931

laos\_life\_exp = bmi\_life\_model.predict(21.07931)

## numpy

矩阵元素之间乘法： \* multiply

矩阵之间相乘：matmul dot

m\_t=m.T可以返回m的转置矩阵

这两个矩阵共享一个数据副本，修改其中一个，另一个也会改变

A =np.array([1,2,3]) 会默认为长度为三的向量。A.shape 会得到一个（3，）的元组tuple。

A.T不会有任何改变。

想要得到一个3\*1的数组。可以这样

B=np.array([[1],[2],[3]]).

想要得到一个1\*3的数组，可以这样

C=np.array([[1,2,3]])

总而言之，要得到一个二维矩阵，就得嵌套中括号

读取数据

input\_array = np.array([inputs])

整个矩阵减去最小值

inputs\_minus\_min = input\_array - np.min(input\_array)

再除以当下的最大值

inputs\_div\_max = inputs\_minus\_min / np.max(inputs\_minus\_min)

看看两个矩阵能否相乘，不用转置，可以交换顺序

if m1.shape[0] != m2.shape[1] and m1.shape[1] != m2.shape[0]:

return False

矩阵相乘

if m1.shape[1] == m2.shape[0]:

return np.matmul(m1, m2)

else:

return np.matmul(m2, m1)

求平均值

return np.mean(values)

## 简单神经网络

and感知器范例：

import pandas as pd

# TODO: Set weight1, weight2, and bias

weight1 = 0.0

weight2 = 0.0

bias = 0.0

# DON'T CHANGE ANYTHING BELOW

# Inputs and outputs

test\_inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]

correct\_outputs = [False, False, False, True]

outputs = []

# Generate and check output

for test\_input, correct\_output in zip(test\_inputs, correct\_outputs):

linear\_combination = weight1 \* test\_input[0] + weight2 \* test\_input[1] + bias

output = int(linear\_combination >= 0)

is\_correct\_string = 'Yes' if output == correct\_output else 'No'

outputs.append([test\_input[0], test\_input[1], linear\_combination, output, is\_correct\_string])

# Print output

num\_wrong = len([output[4] for output in outputs if output[4] == 'No'])

output\_frame = pd.DataFrame(outputs, columns=['Input 1', ' Input 2', ' Linear Combination', ' Activation Output', ' Is Correct'])

if not num\_wrong:

print('Nice! You got it all correct.\n')

else:

print('You got {} wrong. Keep trying!\n'.format(num\_wrong))

print(output\_frame.to\_string(index=False))

### 梯度下降代码实现

之前我们看到一个权重的计算是：

Δ*w*​*i*​​=*ηδx*​*i*​​

这里 error term *δ* 是指

*δ*=(*y*−​*y*​^​​)*f*​′​​(*h*)=(*y*−​*y*​^​​)*f*​′​​(∑*w*​*i*​​*x*​*i*​​)

记住，上面公式中 (*y*−​*y*​^​​) 是输出误差，*f*​′​​(*h*) 是激活函数 *f*(*h*) 的导函数，我们把这个导函数称做输出的梯度。

现在假设只有一个输出，我来把这个写成代码。我们还是用 sigmoid 来作为激活函数 *f*(*h*)。

*# Defining the sigmoid function for activations*

*# 定义 sigmoid 激活函数*

**def** **sigmoid**(x):

**return** 1/(1+np.exp(-x))

*# Derivative of the sigmoid function*

*# 激活函数的导数*

**def** **sigmoid\_prime**(x):

**return** sigmoid(x) \* (1 - sigmoid(x))

*# Input data*

*# 输入数据*

x = np.array([0.1, 0.3])

*# Target*

*# 目标*

y = 0.2

*# Input to output weights*

*# 输入到输出的权重*

weights = np.array([-0.8, 0.5])

*# The learning rate, eta in the weight step equation*

*# 权重更新的学习率*

learnrate = 0.5

*# the linear combination performed by the node (h in f(h) and f'(h))*

*# 输入和权重的线性组合*

h = x[0]\*weights[0] + x[1]\*weights[1]

*# or h = np.dot(x, weights)*

*# The neural network output (y-hat)*

*# 神经网络输出*

nn\_output = sigmoid(h)

*# output error (y - y-hat)*

*# 输出误差*

error = y - nn\_output

*# output gradient (f'(h))*

*# 输出梯度*

output\_grad = sigmoid\_prime(h)

*# error term (lowercase delta)*

error\_term = error \* output\_grad

*# Gradient descent step*

*# 梯度下降一步*

del\_w = [ learnrate \* error\_term \* x[0],

learnrate \* error\_term \* x[1]]

*# or del\_w = learnrate \* error\_term \* x*

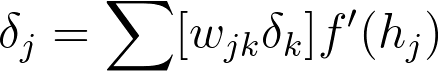
### 正反传播示例

### 实现反向传播

现在我们知道输出层的误差是

*δ*​*k*​​=(*y*​*k*​​−​*y*​^​​​*k*​​)*f*​′​​(*a*​*k*​​)

输入层误差是



现在我们只考虑一个简单神经网络，他只有一层隐藏层和一个输出节点。这是通过反向传播更新权重的算法概述：

* 把每一层权重更新的初始步长设置为 0
  + 输入到隐藏层的权重是 Δ*w*​*ij*​​=0
  + 隐藏层到输出层的权重是 Δ*W*​*j*​​=0
* 对训练数据当中的每一个点
  + 让它正向通过网络，计算输出 ​*y*​^​​
  + 计算输出节点的误差梯度 *δ*​*o*​​=(*y*−​*y*​^​​)*f*​′​​(*z*) 这里 *z*=∑​*j*​​*W*​*j*​​*a*​*j*​​ 输入到输出节点。
  + 误差传播到隐藏层 *δ*​*j*​*h*​​=*δ*​*o*​​*W*​*j*​​*f*​′​​(*h*​*j*​​)
  + 更新权重步长：
    - Δ*W*​*j*​​=Δ*W*​*j*​​+*δ*​*o*​​*a*​*j*​​
    - Δ*w*​*ij*​​=Δ*w*​*ij*​​+*δ*​*j*​*h*​​*a*​*i*​​
* 更新权重, *η* 是学习率，*m* 是数据点的数量：
  + *W*​*j*​​=*W*​*j*​​+*η*Δ*W*​*j*​​/*m*
  + *w*​*ij*​​=*w*​*ij*​​+*η*Δ*w*​*ij*​​/*m*
* 重复这个过程 *e* 代。

完整代码示例：

import numpy as np

from data\_prep import features, targets, features\_test, targets\_test

np.random.seed(21)

def sigmoid(x):

"""

Calculate sigmoid

"""

return 1 / (1 + np.exp(-x))

# Hyperparameters

n\_hidden = 2 # number of hidden units

epochs = 900

learnrate = 0.005

n\_records, n\_features = features.shape

last\_loss = None

# Initialize weights

weights\_input\_hidden = np.random.normal(scale=1 / n\_features \*\* .5,

size=(n\_features, n\_hidden))

weights\_hidden\_output = np.random.normal(scale=1 / n\_features \*\* .5,

size=n\_hidden)

for e in range(epochs):

del\_w\_input\_hidden = np.zeros(weights\_input\_hidden.shape)

del\_w\_hidden\_output = np.zeros(weights\_hidden\_output.shape)

for x, y in zip(features.values, targets):

## Forward pass ##

# TODO: Calculate the output

hidden\_input = np.dot(x, weights\_input\_hidden)

hidden\_output = sigmoid(hidden\_input)

output = sigmoid(np.dot(hidden\_output,

weights\_hidden\_output))

## Backward pass ##

# TODO: Calculate the network's prediction error

error = y - output

# TODO: Calculate error term for the output unit

output\_error\_term = error \* output \* (1 - output)

## propagate errors to hidden layer

# TODO: Calculate the hidden layer's contribution to the error

hidden\_error = np.dot(output\_error\_term, weights\_hidden\_output)

# TODO: Calculate the error term for the hidden layer

hidden\_error\_term = hidden\_error \* hidden\_output \* (1 - hidden\_output)

# TODO: Update the change in weights

del\_w\_hidden\_output += output\_error\_term \* hidden\_output

del\_w\_input\_hidden += hidden\_error\_term \* x[:, None]

# TODO: Update weights

weights\_input\_hidden += learnrate \* del\_w\_input\_hidden / n\_records

weights\_hidden\_output += learnrate \* del\_w\_hidden\_output / n\_records

# Printing out the mean square error on the training set

if e % (epochs / 10) == 0:

hidden\_output = sigmoid(np.dot(x, weights\_input\_hidden))

out = sigmoid(np.dot(hidden\_output,

weights\_hidden\_output))

loss = np.mean((out - targets) \*\* 2)

if last\_loss and last\_loss < loss:

print("Train loss: ", loss, " WARNING - Loss Increasing")

else:

print("Train loss: ", loss)

last\_loss = loss

# Calculate accuracy on test data

hidden = sigmoid(np.dot(features\_test, weights\_input\_hidden))

out = sigmoid(np.dot(hidden, weights\_hidden\_output))

predictions = out > 0.5

accuracy = np.mean(predictions == targets\_test)

print("Prediction accuracy: {:.3f}".format(accuracy))

### 正反向传播算例总结

input layer

hidden layer

output layer

active fuction f(x)

active fuction g(x)

weights\_input\_hidden

weights\_hidden\_output

设输入为X，正确输出为Y，传播过程如下

n\_records 为训练数据集中数据的个数，f(x)和g(x)分别为hidden和out层的激活函数

正向传播过程：

hidden\_layer\_in = np.dot(X,weights\_input\_hidden)

hidden\_layer\_out = f(hidden\_layer\_in)

output\_layer\_in = np.dot(hidden\_layer\_out,weights\_hidden\_output)

output = g(output\_layer\_in)

反向传播如下：

output\_error = Y-output

output\_error\_term = output\_error\*g’(output)

hidden\_error = np.dot(output\_error\_term,weights\_hidden\_output.T)//有必要转置

hidden\_error\_term = hidden\_error\*f’(hidden\_output)

del\_w\_hidden\_output += (output\_error\_term \* hidden\_layer\_out)[:,None]

del\_w\_input\_hidden += hidden\_error\_term \* X[:None]

weights\_input\_hidden += (learnrate \* del\_w\_input\_hidden/n\_reords)

weights\_hidden\_output += learnrate \* del\_w\_hidden\_output/n\_records

红色标记的两个转置视输入数据而定，也可能不需要

关于生成矩阵的两种表示的区别：

np.zeros(5) = np.array([0,0,0,0,0]) = [0,0,0,0,0]

np.zeros((1,5)) = np.array([[0,0,0,0,0]] = [[0,0,0,0,0]]

对于前者，转置只能用[:,None]

对于后者，转置只能用.T

关于反向传播，情感分析那节有另个思路，这样写出来的式子比较清晰：

layer\_2\_error = layer\_2 - self.get\_target\_for\_label(label) # Output layer error is the difference between desired target and actual output.

layer\_2\_delta = layer\_2\_error \* self.sigmoid\_output\_2\_derivative(layer\_2)

# Backpropagated error

layer\_1\_error = layer\_2\_delta.dot(self.weights\_1\_2.T) # errors propagated to the hidden layer

layer\_1\_delta = layer\_1\_error # hidden layer gradients - no nonlinearity so it's the same as the error

# Update the weights

self.weights\_1\_2 -= layer\_1.T.dot(layer\_2\_delta) \* self.learning\_rate # update hidden-to-output weights with gradient descent step

self.weights\_0\_1 -= self.layer\_0.T.dot(layer\_1\_delta) \* self.learning\_rate #

### 超参数调节

learnrate的选择

lr/n\_records的值在0.001-0.01之间就好

神经结点个数

过多的结点会放慢收敛速度，也会导致过拟合

过少的结点，收敛速度会很快，但是会欠拟合

迭代次数

最佳次数是validation开始上升的时候

## 共享单车项目

## 测试驱动开发思路：

一般来说你会在写实际代码前先完成测试代码，确保你写的代码能够完成你想要它完成的工作。这种“测试驱动的开发”的方法在许多领域广泛应用，尤其是当我们需要写一些比较难写或者复杂的代码（比如我们这个例子）。因为在一个复杂度比较高的函数或者类中，你很难去定位一些小错误（比如语法错误，数学表达式出现了错误）。

我的代码审阅反馈：

我们对于模型复杂度的控制，主要体现在隐藏层单元的个数上，一般至少8个才能保证我们的模型不会欠拟合。我选择的是12个

别的参数，学习速率0.15，迭代12000次

# 二、卷积神经网络

## 分离训练和测试数据

from sklearn.model\_selection import train\_test\_split

x代表输入，y代表输出

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25)

上面的命令代表，把数据中的25%用作测试

测试数据不能用于训练

准确度函数

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_true,y\_pred)

### 计算平均绝对误差和均方差

from sklearn.metrics import mean\_absolute\_error(mean\_squared\_error)

from sklearn.linear\_model import LinearRegression

classifier=LinearRegression()

classifier.fit(X,y)

guesses=classifier.predict(X)

error=mean\_absolute\_error(y,guesses)

error=mean\_squared\_error(y,guesses)

### R2分数

用来判定一个模型的好坏

计算方法，1-该模型的均方误差/随机模型的均方误差

from sklearn.metrics import r2\_score

y\_true=[1,2,4]

y\_pred=[1.3,2.5,3.7]

r2\_score(y\_true,y\_pred)

### K折交叉验证

from sklearn.model\_selection import Kfold

kf=Kfold(12,3,shufle=True)

for train\_indices,test\_indices in kf：

print train\_indices,test\_indices

这个意思是，共有12个数据，随机分出3个作为测试数据、验证数据

## MiniFlow梯度下降数学原理

### 到达谷底

现在，我将简单复习下梯度下降概念，使我们能够开始用 MiniFlow 训练我们的网络。注意，我们的目标是通过尽量减小代价，使我们的网络输出与目标值尽量接近。你可以将代价看做一座山，我们想要到达山底。

想象你的模型参数表示为一个停在山顶的球。直观地来说，我们希望将球推下山。这样可以明白，但是我们讨论的是代价函数，如何知道哪条路是下山呢？

幸运的是，梯度下降正好给出了这一信息。

严格来说，梯度实际上指的是上坡，是**最陡上升**方向。但是，如果我们在此值前面加个负号，就得出了**最陡下降**方向，这正是我们需要的。

稍后你将详细了解梯度，但是暂时可以将其看做数字向量。每个数字表示我们应该对照着调整神经网络中相应的权重或偏置的数量。按照梯度值调整所有的权重和偏置降低了网络的代价（或误差）。

听明白了吗？

好的！现在我们知道朝着哪个方向推球了。下一步是考虑用多大的推力，称之为学习速度，该名称比较恰当，因为该值确定了神经网络学习的快慢速度。

你可能希望设置非常大的学习速度，这一网络就能学的非常快，对吧？

要小心！如果该值太大，可能会迭代过度并最终偏离目标。呀！



**收敛**。这是理想的行为。（这里是个收敛的动图）

**发散**。当学习速度过高时会出现这种情况。（这里是个发散的动图）

那么，什么样的学习速度合适呢？

我们只能做出猜测，根据以往经验，0.1 至 0.0001 范围的值效果最好。0.001 至 0.0001 范围的值很常见，因为 0.1 和 0.01 有时候过大。

下面是**梯度下降**的公式（伪代码）：

x = x - learning\_rate \* gradient\_of\_x

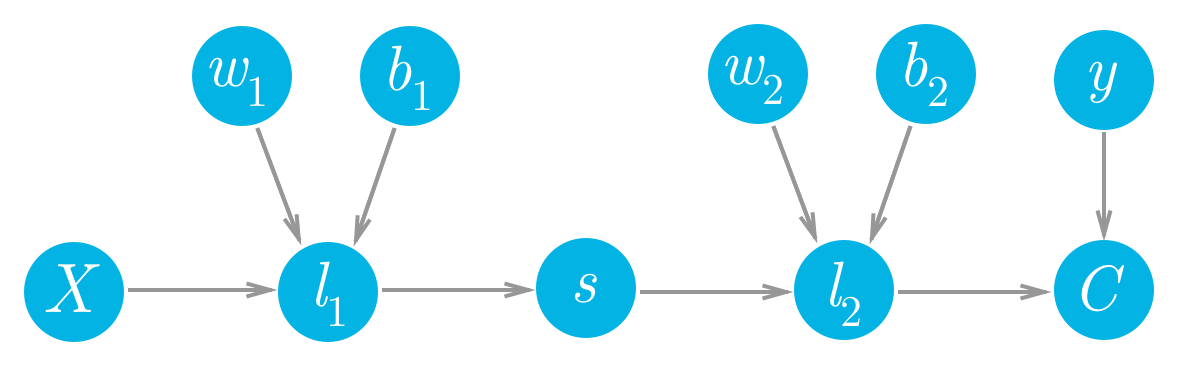
x 是神经网络使用的参数（即单个权重或偏置）。

我们将 gradient\_of\_x（上坡方向）与 learning\_rate（推力）相乘，然后将 x 减去相乘结果，从而向下推动。这就是**梯度下降**

### 反向传播

我们现在知道如何使用梯度更新我们的权重和偏置，我们还需要知道如何计算所有节点的梯度。对于每个节点，我们需要根据梯度更改代价的值（考虑到该节点的值）。这样，我们做出的梯度下降更新最终会实现最低代价。

我们来看一个网络，其中具有一个线性节点 *l*​1​​，一个 S 型节点 *s*，以及另一个线性节点 *l*​2​​，还有一个 MSE 节点，用于计算代价 *C*。



简单的两层网络的前向传递。

用 MiniFlow 编写的话，应该如下所示：

X, y = Input(), Input()

W1, b1 = Input(), Input()

W2, b2 = Input(), Input()

l1 = Linear(X, W1, b1)

s = Sigmoid(l1)

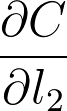
l2 = Linear(s, W2, b2)

cost = MSE(l2, y)

注：MSE为均方误差。

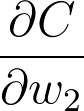
在实际运用中，这里的输出层l2少了个激活函数，不过不影响这里的推导思想。

我们可以看到这些节点的每个值都向前流动，最终生成代价 *C*。例如，第二个线性节点 *l*​2​​ 的值进入代价节点并确定该节点的值。*更改* *l*​2​​ 将导致 *C* 相应地出现更改。我们可以将更改之间的这种关系写成梯度。

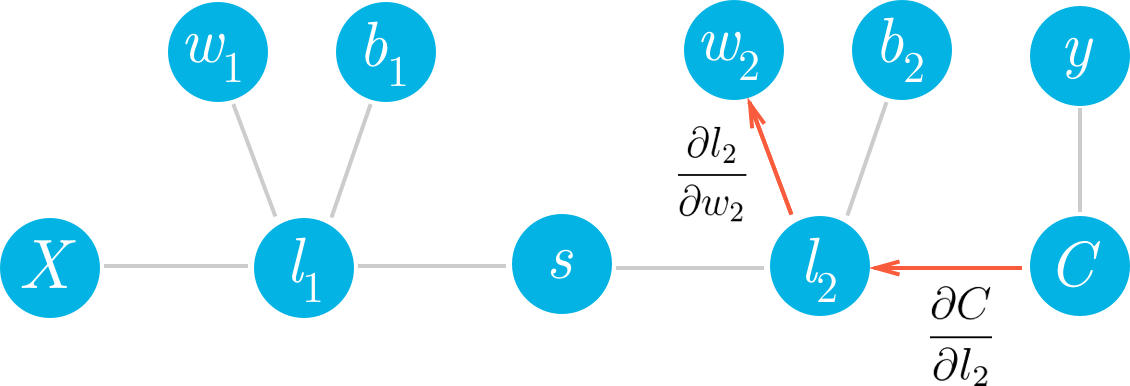


这就是梯度的含义，是一种斜率，表示给出 *l*​2​​ 中的更改，你会对代价 ∂*C* 进行多大幅度的更改。所以，节点的代价梯度更大的话，代价就会改变更大。这样，我们可以对每个节点的代价带来的影响进行分配。节点的梯度越大，就会对最终的代价影响越大。节点的影响越大，我们就会在梯度下降步骤中更新幅度越大。

如果我们想更新某个梯度下降的权重，我们需要知道这些权重对应的代价的梯度。看看我们可以如何使用此框架算出第二层权重 *w*​2​​ 的梯度。我们想要计算 *C* 相对于 *w*​2​​ 的梯度：

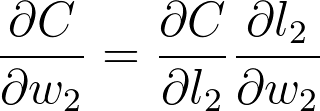


我们可以从图表中看出，*w*​2​​ 与 *l*​2​​ 相关联，所以更改 *w*​2​​ 将导致 *l*​2​​ 出现更改，从而导致 *C* 出现更改。我们可以通过在网络中将代价梯度发送回去，将影响分配给 *w*​2​​。首先，你知道 *l*​2​​ 对 *C* 的影响有多大，然后知道 *w*​2​​ 对 *l*​2​​ 的影响有多大。将这些梯度相乘可以得出归为 *w*​2​​ 的总影响。



### 应用链式法则

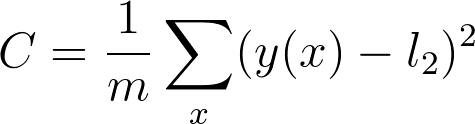
将这些梯度相乘只是链式法则的一种应用：



可以从图表中看出，*w*​2​​、*l*​2​​ 和 *C* 相互链接在一起。如果 *w*​2​​ 发生任何更改，将导致 *l*​2​​ 出现更改，更改幅度由梯度 ∂*l*​2​​/∂*w*​2​​ 决定。因为 *l*​2​​ 更改了，这将导致代价 *C* 出现更改，更改幅度由梯度 ∂*C*/∂*l*​2​​ 决定。你可以将链式法则看做多米诺效应，网络中的某项更改将从网络中传播开来，并一路更改其他节点。

如果你将链式法则看做普通的分数，可以看到分母上的 ∂*l*​2​​ 和分子消掉了，获得 ∂*C*/∂*w*​2​​（虽然和普通分子的计算过程并不完全一样，但是可以帮助你理解）。现在来算算 *w*​2​​ 的梯度。首先，我们需要知道 *l*​2​​ 的梯度。

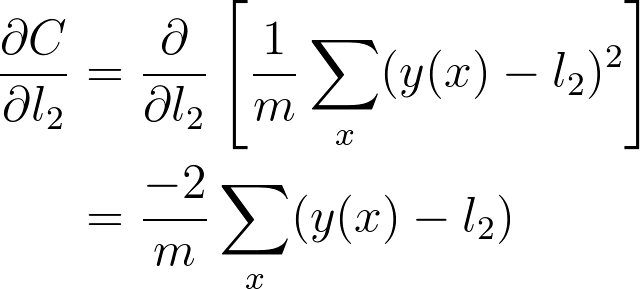
提醒下，代价是：

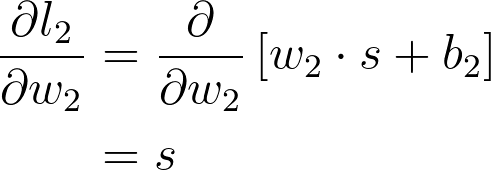


第二个线性节点的值为

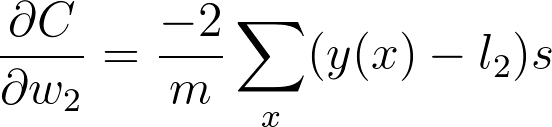
https://d17h27t6h515a5.cloudfront.net/topher/2017/February/589d15a7_l2/l2.png

*w*​2​​、*s*和 *b*​2​​ 都是向量，*w*​2​​⋅*s* 表示 *w*​2​​ 和 *s* 的点积。



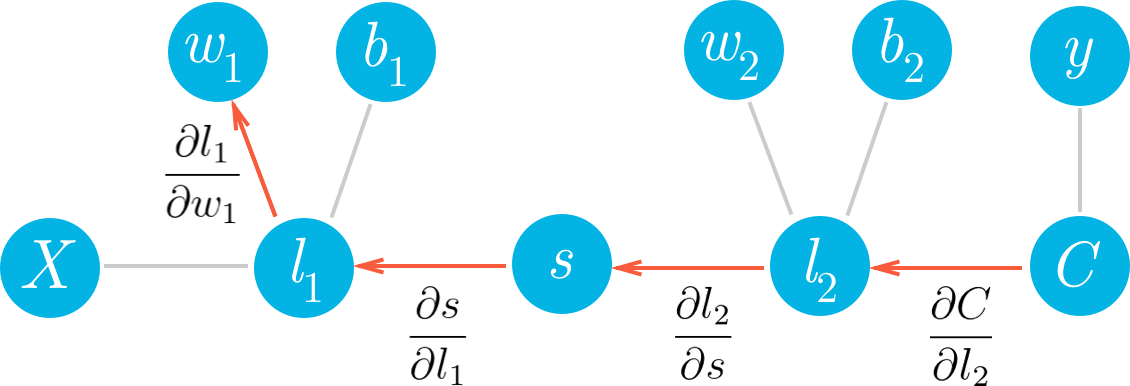


将这些放一起，可以得出 *w*​2​​ 的梯度

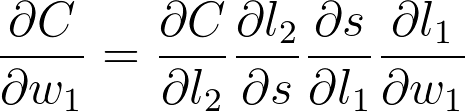


这是你在 *w*​2​​ 的梯度下降更新中用到的梯度。可以看出，我们在图表上一直往回计算，将一路上发现的所有梯度相乘。

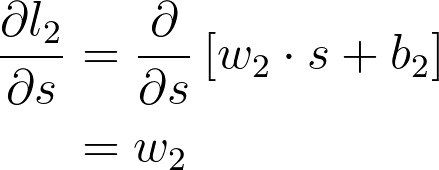
现在，我们再深入一步，计算 *w*​1​​ 的梯度。和之前用到的方法一样，在图表上一直往回计算。



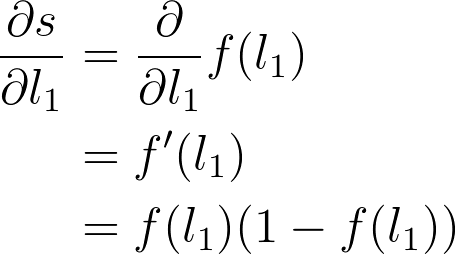
希望现在你能明白如何通过查看图表写出 *w*​1​​ 的梯度。我们将使用链式法则在图表上一直往回计算，得出每个节点的梯度，直到算出 *w*​1​​ 的梯度。

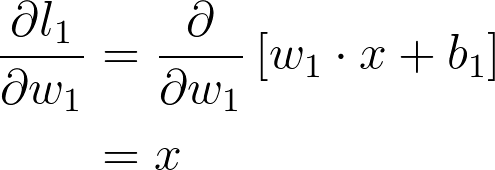


现在我们可以计算此表达式中的每个梯度，以便得出 *w*​1​​ 的梯度

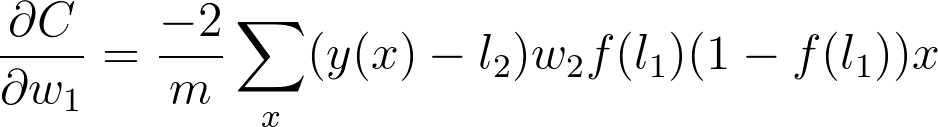


下一步是 S 型函数 *s*=*f*(*l*​1​​) 的梯度。因为这里使用的是逻辑函数，所以导数可以写成 S 型函数本身





将所有这一切放在一起，得出



现在可以看到清晰的规律了。要算出梯度，只需将它前面所有节点（从代价那开始）的梯度相乘。这就是**反向传播**概念。梯度在网络上向后传播，并使用梯度下降来更新权重和偏置。如果某个节点具有多个向外的节点，则直接将每个节点的梯度相加即可。

### 计算节点中的梯度

这部分没明白，但是把代码写下来吧

这是对激活函数为sigmoid的结点的梯度更新：

**def** **backward**(self):

"""

Calculates the gradient using the derivative of

the sigmoid function.

"""

*# Initialize the gradients to 0.*

self.gradients = {n: np.zeros\_like(n.value) **for** n **in** self.inbound\_nodes}

*# Sum the derivative with respect to the input over all the outputs.*

**for** n **in** self.outbound\_nodes:

grad\_cost = n.gradients[self]

sigmoid = self.value

self.gradients[self.inbound\_nodes[0]] += sigmoid \* (1 - sigmoid) \* grad\_cost

下面就是我看不懂的更新了：

**for** n **in** self.outbound\_nodes:

*# Get the partial of the cost with respect to this node.*

grad\_cost = n.gradients[self]

*# Set the partial of the loss with respect to this node's inputs.*

self.gradients[self.inbound\_nodes[0]] += np.dot(grad\_cost, self.inbound\_nodes[1].value.T)

*# Set the partial of the loss with respect to this node's weights.*

self.gradients[self.inbound\_nodes[1]] += np.dot(self.inbound\_nodes[0].value.T, grad\_cost)

*# Set the partial of the loss with respect to this node's bias.*

self.gradients[self.inbound\_nodes[2]] += np.sum(grad\_cost, axis=0, keepdims=**False**)

解释：

|  |  |
| --- | --- |
| 1 | np.dot(grad\_cost, self.inbound\_nodes[1].value.T) |

对于Linear节点来说，有三个输入参数，即inputs, weights, bias分别对应着

|  |  |
| --- | --- |
| 1 | self.inbound\_nodes[0]，self.inbound\_nodes[1]，self.inbound\_nodes[2] |

*So, each node will pass on the cost gradient to its inbound nodes and each node will get the cost gradient from it’s outbound nodes. Then, for each node we’ll need to calculate a gradient that’s the cost gradient times the gradient of that node with respect to its inputs.*

于是Linear对inputs的求导就是weights。所以是grad\_cost\*weights.grad\_cost是Linear输出节点传递进来的变化率。

|  |  |
| --- | --- |
| 1 | np.dot(self.inbound\_nodes[0].value.T, grad\_cost) |

同理可推对weights的求导为inputs，于是gradient=grad\_cost\*inputs

|  |  |
| --- | --- |
| 1 | np.sum(grad\_cost, axis=0, keepdims=False) |

而对于bias，Linear对bias求导恒为1.所以gradient=1\*grad\_cost

**解释二：为何是+=**

因为每一个节点将误差传递给每一个输出节点。于是在Backpropagation时，要求出每一个节点的误差，就要将每一份传递出去给输出节点的误差加起来。于是用+=。  
于是可以理解为什么要for n in self.outbound\_nodes: 目的是为了在每一个节点的输出节点里遍历。  
If a node has multiple outgoing nodes, you just sum up the gradients from each node.

***注意点一：***  
要区分Backpropagation 和Gradient Descent是两个步骤，我通过Backpropagation找到gradient，于是找到了变化方向。再通过Gradient Descent来最小化误差。

*To find the gradient, you just multiply the gradients for all nodes in front of it going backwards from the cost. This is the idea behind backpropagation. The gradients are passed backwards through the network and used with gradient descent to update the weights and biases.*

最终目的是：

http://very9s.b0.upaiyun.com/wp-file/2017/03/change-weight.png

Backpropagation只求了导数部分。Gradient Descent则是整个过程。

### 随机梯度下降

随机梯度下降 (SGD) 是一种梯度下降形式，其中对于每次前向传递，都会从总的数据集中随机选择一批数据。还记得之前讨论的批量大小吗？即批次大小。理想情况下，每次前向传递时，都会将整个数据集提供给神经网络。但是实际操作中，这么做是不现实的，因为内存有限。随机梯度下降是梯度下降的近视值，神经网络处理的批次越多，近视值就越准确。 SGD 的实现包括：

1. 从总的数据集中随机抽样一批数据。
2. 前向和后向运行网络，计算梯度（根据第 (1) 步的数据）。
3. 应用梯度下降更新。
4. 重复第 1-3 步，直到出现收敛情况或者循环被其他机制暂停（即迭代次数）。

如果一切顺利，网络的损失应该通常会下降，表示权重和偏置越来越有用。

到目前为止，MiniFlow 已经可以完成第 2 步。在下面的测验中，第 1 和 4 步已执行。你需要执行第 3 步。

提醒下，下面是梯度下降更新方程，其中 *α* 表示学习速度：

*x*=*x*−*α*∗​∂*x*​​∂*cost*​​

这是我对上个练习的解决方案

**def** **sgd\_update**(trainables, learning\_rate=1e-2):

"""

Updates the value of each trainable with SGD.

Arguments:

`trainables`: A list of `Input` nodes representing weights/biases.

`learning\_rate`: The learning rate.

"""

*# Performs SGD*

*#*

*# Loop over the trainables*

**for** t **in** trainables:

*# Change the trainable's value by subtracting the learning rate*

*# multiplied by the partial of the cost with respect to this*

*# trainable.*

partial = t.gradients[t]

t.value -= learning\_rate \* partial

有两个关键步骤。第一步，Cost (*C*) 关于 trainable t 的偏导 可以如下得到

partial = t.gradients[t]

第二步，trainable 的值根据等式 (12) 进行更新

t.value -= learning\_rate \* partial

对所有 trainable 执行。

等式 (12)

等式 (12)

这样的话，下次在网络中传递时，损失会下降。

## 跟Andrew Trask学情感分析

### 按行读取数据

g = open('reviews.txt','r')

reviews = list(map(lambda x:x[:-1],g.readlines()))

g.close()

g = open('labels.txt','r')

labels = list(map(lambda x:x[:-1].upper(),g.readlines()))

g.close()

### word2index

word2idx ={word : i for i,word in enumerate(vocab)}

## 5，为Siraj的课程做准备

### 修正线性单元（ReLU）：

f(x) = max(x,0)

正如在反向传播资料中提到的，S 型函数的导数最大值为 0.25（如上所示）。这意味着，当你用 S 型函数单元进行反向传播时，网络上每层出现的错误至少减少 75%，如果有很多层，权重更新将很小，这些权重需要很长的训练时间。因此，S 型函数不适合作为隐藏单元上的激活函数。

ReLU 激活函数是你可以使用的最简单非线性激活函数。当输入是整数时，导数是 1，所以没有 S 型函数的反向传播错误导致的消失效果。[**研究表明**](http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf)，对于大型神经网络来说，ReLU 的训练速度要快很多。TensorFlow 和 TFLearn 等大部分框架使你能够轻松地在隐藏层使用 ReLU，你不需要自己去实现这些 ReLU。

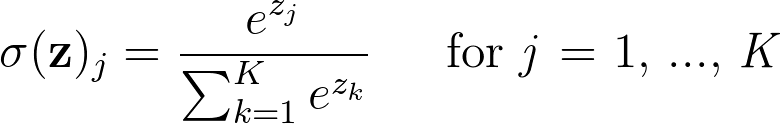
不足

有可能大的梯度设置的权重使 ReLU 单元始终为 0。这些“无效”的单元将始终为 0，很多计算在训练中被浪费了。

### Softmax

softmax 函数将每个单元的输出压缩到 0 和 1 之间,它在输出时会使输出之和等于1. softmax 函数的输出等于分类概率分布，显示了任何类别为真的概率。

softmax 函数的数学表达式如下所示，其中 **z** 是输出层的输入向量（如果你有 10 个输出单元，则 **z** 中有 10 个元素）。同样，*j*表示输出单元的索引。

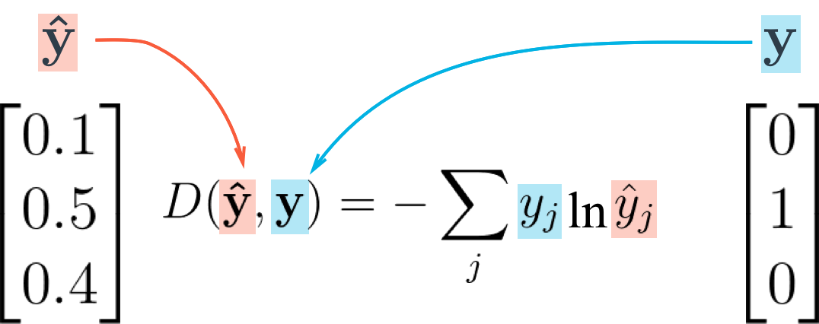


softmax 可以用于任何数量的分类。接下来你将看到，它可以用于预测两种类别的情感（积极和消极）。还可以用于数百个和数千个类别，例如对于物体识别，物体存在数百种可能性。

### 分类交叉熵

但是当你在使用 softmax 时，输出是向量。一个向量是输出单元的概率值。你还可以使用**一位热码编码**将数据标签表示为向量

我们希望误差与这些向量之间的距离成比例。要计算这一距离，我们将使用[**交叉熵**](https://en.wikipedia.org/wiki/Cross_entropy)。我们的神经网络训练目标将为：通过尽可能地减小交叉熵使预测向量与标签向量尽量靠近。交叉熵计算公式如下所示：



Cross entropy calculation

可以从上文中看出，交叉熵等于标签元素的和乘以预测概率的自然对数。注意，该等式并不对称！千万不能交换向量，因为标签向量里有很多 0，对 0 求对数将产生错误。

对标签向量使用一位热码编码的好处是 *y*​*j*​​ 为 0（类别为真的除外）。然后，和中 *y*​*j*​​=1 之外的所有其他项为 0，对于为真的标签来说，交叉熵直接变成 *D*=−ln​*y*​^​​。例如，如果输入图片为数字 4 并且标为 4，那么只有与 4 对应的单元的输出在交叉熵成本函数中会产生影响。

### TFlearn

从一个counter获取索引字典

word2idx ={word : i for i,word in enumerate(vocab)}

### 组件Tflearn的实例

import numpy as np

import tensorflow as tf

import tflearn

import tflearn.datasets.mnist as mnist

# Retrieve the training and test data

trainX, trainY, testX, testY = mnist.load\_data(one\_hot=True)

def build\_model():

# This resets all parameters and variables, leave this here

tf.reset\_default\_graph()

# Inputs

net = tflearn.input\_data([None, trainX.shape[1]])

# Hidden layer(s)

net = tflearn.fully\_connected(net, 128, activation='ReLU')

net = tflearn.fully\_connected(net, 32, activation='ReLU')

# Output layer and training model

net = tflearn.fully\_connected(net, 10, activation='softmax')

net = tflearn.regression(net, optimizer='sgd', learning\_rate=0.01, loss='categorical\_crossentropy')

model = tflearn.DNN(net)

return model

# Build the model

model = build\_model()

# Training

model.fit(trainX, trainY, validation\_set=0.1, show\_metric=True, batch\_size=100, n\_epoch=100)

# Compare the labels that our model predicts with the actual labels

# Find the indices of the most confident prediction for each item. That tells us the predicted digit for that sample.

predictions = np.array(model.predict(testX)).argmax(axis=1)

# Calculate the accuracy, which is the percentage of times the predicated labels matched the actual labels

actual = testY.argmax(axis=1)

test\_accuracy = np.mean(predictions == actual, axis=0)

# Print out the result

print("Test accuracy: ", test\_accuracy)

### 计算单词频率的两种方法

一、

Counter(text.split())

二、

for word in text.split(' '):

count[word] += 1

## TensorFlow入门

* 在 [**tf.Session**](https://www.tensorflow.org/api_docs/python/tf/Session) 里面进行运算。
* 通过 [**tf.constant()**](https://www.tensorflow.org/api_docs/python/tf/constant) 创建常量 tensor。
* 用 [**tf.placeholder()**](https://www.tensorflow.org/api_docs/python/tf/placeholder) 和 feed\_dict 得到输入。
* 应用 [**tf.add()**](https://www.tensorflow.org/api_docs/python/tf/add)、 [**tf.subtract()**](https://www.tensorflow.org/api_docs/python/tf/subtract)、 [**tf.multiply()**](https://www.tensorflow.org/api_docs/python/tf/multiply) 和 [**tf.divide()**](https://www.tensorflow.org/api_docs/python/tf/divide) 函数进行数学运算。
* 学习如何用 [**tf.cast()**](https://www.tensorflow.org/api_docs/python/tf/cast) 进行类型转换。

训练模型用SGD方法时，有两个方法提高性能。一、随着训练降低学习率。二、momentum：追踪梯度实时平均值，用该值代替当前一批数据计算得出的方向。

### 数据形式

在 TensorFlow 中，数据不是以整数，浮点数或者字符串形式存在的。这些值被封装在一个叫做 tensor 的对象中。

hello\_constant = tf.constant('Hello World!') 代码中，hello\_constant是一个 0 维度的字符串 tensor，tensors 还有很多不同大小：

A = tf.constant(1234)

*# B is a 1-dimensional int32 tensor*

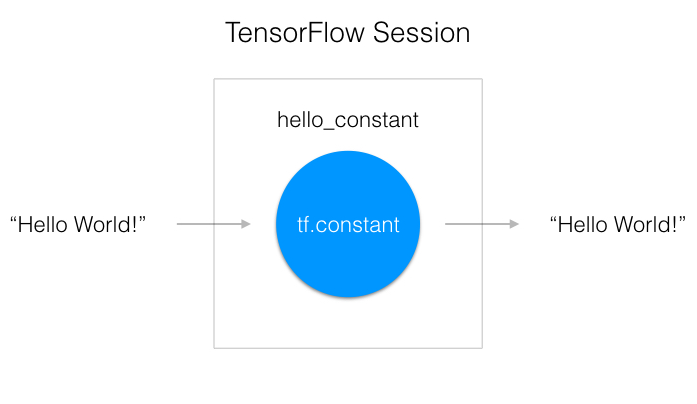
B = tf.constant([123,456,789])

*# C is a 2-dimensional int32 tensor*

C = tf.constant([ [123,456,789], [222,333,444] ])

### Session

TensorFlow 的 api 构建在 computational graph 的概念上，它是一种对数学运算过程进行可视化的一种方法。让我们把你刚才运行的 TensorFlow 的代码变成一个图：



如上图所示，一个 "TensorFlow Session" 是用来运行图的环境。这个 session 负责分配 GPU(s) 和／或 CPU(s) 包括远程计算机的运算。让我们看看怎么使用它：

**with** tf.Session() **as** sess:

output = sess.run(hello\_constant)

这段代码已经从上一行创建了一个 tensor hello\_constant。下一行是在session里对 tensor 求值。

这段代码用 [**tf.Session**](https://www.tensorflow.org/api_docs/python/tf/Session) 创建了一个sess的 session 实例。 [**sess.run()**](https://www.tensorflow.org/api_docs/python/tf/Session#run) 函数对 tensor 求值，并返回结果。

### 输入数据 placeholder() feed\_dict

惯例：用tf.placeholder（）来创建数据类型，在Session.run里调用feed\_dict来赋值：

x = tf.placeholder(tf.string)

**with** tf.Session() **as** sess:

output = sess.run(x, feed\_dict={x: 'Hello World'})

### 四则运算

x = tf.add(5, 2) *# 7*

x = tf.subtract(10, 4) *# 6*

y = tf.multiply(2, 5) *# 10*

y = tf.divide(10,2) #5

### 类型转换

tf.subtract(tf.cast(tf.constant(2.0), tf.int32), tf.constant(1))

### 声明变量

tf.Variable()

x = tf.Variable(5) 返回初始值为5的变量

返回正态分布的二维矩阵

n\_features = 120

n\_labels = 5

weights = tf.Variable(tf.truncated\_normal((n\_features, n\_labels)))

返回一个初始值为0的一维向量

n\_labels = 5

bias = tf.Variable(tf.zeros(n\_labels))

tensor 把它的状态存在 session里，所以你必须手动初始化它的状态。你用[**tf.global\_variables\_initializer()**](https://www.tensorflow.org/api_docs/python/tf/global_variables_initializer) 来初始化所有可变 tensors。

### softmax实例

def run():

output = None

logit\_data = [2.0, 1.0, 0.1] #原始数据

logits = tf.placeholder(tf.float32) #tensor数据格式（空）

# TODO: Calculate the softmax of the logits

softmax1 = tf.nn.softmax(logits) #函数

with tf.Session() as sess: #可视化运行

##softmax1为目标参数，feed\_dict为目标置换合格数据类型

output = sess.run(softmax1,feed\_dict = {logits:logit\_data} )

return output

### tensorflow交叉熵

import tensorflow as tf

softmax\_data = [0.7, 0.2, 0.1]

one\_hot\_data = [1.0, 0.0, 0.0]

softmax = tf.placeholder(tf.float32)

one\_hot = tf.placeholder(tf.float32)

# TODO: Print cross entropy from session

cross\_entropy = -tf.reduce\_sum(tf.multiply(tf.log(softmax),one\_hot))

with tf.Session() as sess:

print(sess.run(cross\_entropy, feed\_dict={softmax: softmax\_data,one\_hot: one\_hot\_data}))

### 数据预处理

把数据处理为零均值和方差齐性

模型预设为：零均值和标准偏差（从小的sigma开始）

### TensorFlow Mini-batching

要使用 mini-batching，你首先要把你的数据集分成 batches。

不幸的是，有时候不可能把数据完全分割成相同数量的 batch。例如有 1000 个数据点，你想每个 batch 有 128 个数据。但是1000 无法被 128 整除。你得到的结果是 7 batch，每个128个数据点，一个 batch 有 104个数据点。(7\*128 + 1\*104 = 1000)

batch里面的数据点数量会不同的情况下，你需要利用 TensorFlow 的[**tf.placeholder()**](https://www.tensorflow.org/api_docs/python/tf/placeholder) 函数来接收这些不同的 batch。

继续上述例子，如果每个样本有n\_input = 784特征，n\_classes = 10个可能的标签，features的维度应该是[None, n\_input]，labels的维度是 [None, n\_classes]。

*# Features and Labels*

features = tf.placeholder(tf.float32, [**None**, n\_input])

labels = tf.placeholder(tf.float32, [**None**, n\_classes])

None在这里做什么用呢？

None 维度在这里是一个 batch size 的占位符。在运行时，TensorFlow 会接收任何大于 0 的 batch size。

回到之前的例子，这个设置可以让你把features 和 labels给到模型。无论 batch 中包含128，还是104个数据点。

### batches函数实现代码

import math

def batches(batch\_size, features, labels):

assert len(features) == len(labels)

# TODO: Implement batching

output\_batches = []

data\_len = len(features)

for start\_i in range(0,data\_len,batch\_size):

end\_i = start\_i + batch\_size

batch = [features[start\_i:end\_i],labels[start\_i:end\_i]]

output\_batches.append(batch)

return output\_batches

### 标准TensorFlow代码

**from** tensorflow.examples.tutorials.mnist **import** input\_data

**import** tensorflow **as** tf

**import** numpy **as** np

**from** helper **import** batches *# Helper function created in Mini-batching section*

**def** **print\_epoch\_stats**(epoch\_i, sess, last\_features, last\_labels):

"""

Print cost and validation accuracy of an epoch

"""

current\_cost = sess.run(

cost,

feed\_dict={features: last\_features, labels: last\_labels})

valid\_accuracy = sess.run(

accuracy,

feed\_dict={features: valid\_features, labels: valid\_labels})

print('Epoch: {:<4} - Cost: {:<8.3} Valid Accuracy: {:<5.3}'.format(

epoch\_i,

current\_cost,

valid\_accuracy))

n\_input = 784 *# MNIST data input (img shape: 28\*28)*

n\_classes = 10 *# MNIST total classes (0-9 digits)*

*# Import MNIST data*

mnist = input\_data.read\_data\_sets('/datasets/ud730/mnist', one\_hot=**True**)

*# The features are already scaled and the data is shuffled*

train\_features = mnist.train.images

valid\_features = mnist.validation.images

test\_features = mnist.test.images

train\_labels = mnist.train.labels.astype(np.float32)

valid\_labels = mnist.validation.labels.astype(np.float32)

test\_labels = mnist.test.labels.astype(np.float32)

*# Features and Labels*

features = tf.placeholder(tf.float32, [**None**, n\_input])

labels = tf.placeholder(tf.float32, [**None**, n\_classes])

*# Weights & bias*

weights = tf.Variable(tf.random\_normal([n\_input, n\_classes]))

bias = tf.Variable(tf.random\_normal([n\_classes]))

logits = tf.add(tf.matmul(features, weights), bias)

*# Define loss and optimizer*

learning\_rate = tf.placeholder(tf.float32)

cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=labels))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate).minimize(cost)

*# Calculate accuracy*

correct\_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

init = tf.global\_variables\_initializer()

batch\_size = 128

epochs = 10

learn\_rate = 0.001

train\_batches = batches(batch\_size, train\_features, train\_labels)

**with** tf.Session() **as** sess:

sess.run(init)

*# Training cycle*

**for** epoch\_i **in** range(epochs):

*# Loop over all batches*

**for** batch\_features, batch\_labels **in** train\_batches:

train\_feed\_dict = {

features: batch\_features,

labels: batch\_labels,

learning\_rate: learn\_rate}

sess.run(optimizer, feed\_dict=train\_feed\_dict)

*# Print cost and validation accuracy of an epoch*

print\_epoch\_stats(epoch\_i, sess, batch\_features, batch\_labels)

*# Calculate accuracy for test dataset*

test\_accuracy = sess.run(

accuracy,

feed\_dict={features: test\_features, labels: test\_labels})

print('Test Accuracy: {}'.format(test\_accuracy))

## 8,亚马逊服务器 amazon web services

## 9.深度神经网络

### 模型存取

**保存一个训练好的模型**

**import** tensorflow **as** tf

*# The file path to save the data*

*# 文件保存路径*

save\_file = './model.ckpt'

*# Two Tensor Variables: weights and bias*

*# 两个 Tensor 变量：权重和偏置项*

weights = tf.Variable(tf.truncated\_normal([2, 3]))

bias = tf.Variable(tf.truncated\_normal([3]))

*# Class used to save and/or restore Tensor Variables*

*# 用来存取 Tensor 变量的类*

saver = tf.train.Saver()

**with** tf.Session() **as** sess:

*# Initialize all the Variables*

*# 初始化所有变量*

sess.run(tf.global\_variables\_initializer())

*# Show the values of weights and bias*

*# 显示变量和权重*

print('Weights:')

print(sess.run(weights))

print('Bias:')

print(sess.run(bias))

*# Save the model*

*# 保存模型*

saver.save(sess, save\_file)

**读取模型**

*# Remove the previous weights and bias*

*# 移除之前的权重和偏置项*

tf.reset\_default\_graph()

*# Two Variables: weights and bias*

*# 两个变量：权重和偏置项*

weights = tf.Variable(tf.truncated\_normal([2, 3]))

bias = tf.Variable(tf.truncated\_normal([3]))

*# Class used to save and/or restore Tensor Variables*

*# 用来存取 Tensor 变量的类*

saver = tf.train.Saver()

**with** tf.Session() **as** sess:

*# Load the weights and bias*

*# 加载权重和偏置项*

saver.restore(sess, save\_file)

*# Show the values of weights and bias*

*# 显示权重和偏置项*

print('Weight:')

print(sess.run(weights))

print('Bias:')

print(sess.run(bias))

### 标准TensorFlow代码

*#从一个模型开始*

*# Remove previous Tensors and Operations*

*# 移除之前的 Tensors 和运算*

tf.reset\_default\_graph()

**from** tensorflow.examples.tutorials.mnist **import** input\_data

**import** numpy **as** np

learning\_rate = 0.001

n\_input = 784 *# MNIST 数据输入 (图片尺寸: 28\*28)*

n\_classes = 10 *# MNIST 总计类别 (数字 0-9)*

*# Import MNIST data*

*# 加载 MNIST 数据*

mnist = input\_data.read\_data\_sets('.', one\_hot=**True**)

*# Features and Labels*

*# 特征和标签*

features = tf.placeholder(tf.float32, [**None**, n\_input])

labels = tf.placeholder(tf.float32, [**None**, n\_classes])

*# Weights & bias*

*# 权重和偏置项*

weights = tf.Variable(tf.random\_normal([n\_input, n\_classes]))

bias = tf.Variable(tf.random\_normal([n\_classes]))

*# Logits - xW + b*

logits = tf.add(tf.matmul(features, weights), bias)

*# Define loss and optimizer*

*# 定义损失函数和优化器*

cost = tf.reduce\_mean(\

tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=labels))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate)\

.minimize(cost)

*# Calculate accuracy*

*# 计算准确率*

correct\_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

**让我们训练模型并保存权重**：

**import** math

save\_file = './train\_model.ckpt'

batch\_size = 128

n\_epochs = 100

saver = tf.train.Saver()

*# Launch the graph*

**with** tf.Session() **as** sess:

sess.run(tf.global\_variables\_initializer())

*# Training cycle*

**for** epoch **in** range(n\_epochs):

total\_batch = math.ceil(mnist.train.num\_examples / batch\_size)

*# Loop over all batches*

**for** i **in** range(total\_batch):

batch\_features, batch\_labels = mnist.train.next\_batch(batch\_size)

sess.run(

optimizer,

feed\_dict={features: batch\_features, labels: batch\_labels})

*# Print status for every 10 epochs*

**if** epoch % 10 == 0:

valid\_accuracy = sess.run(

accuracy,

feed\_dict={

features: mnist.validation.images,

labels: mnist.validation.labels})

print('Epoch {:<3} - Validation Accuracy: {}'.format(

epoch,

valid\_accuracy))

*# Save the model*

saver.save(sess, save\_file)

print('Trained Model Saved.')

**加载训练好的模型**

saver = tf.train.Saver()

*# Launch the graph*

**with** tf.Session() **as** sess:

saver.restore(sess, save\_file)

test\_accuracy = sess.run(

accuracy,

feed\_dict={features: mnist.test.images, labels: mnist.test.labels})

print('Test Accuracy: {}'.format(test\_accuracy))

### 训练模型加载

模型保存时，按照一定顺序保存，（据我观察）与权重与偏置申请顺序一致，而从文件中读取这些参数时，也会按照该顺序读取，并按照此次相关变量的申请顺序赋值，如果两次申请的顺序不一致，系统就会报错。

解决办法：申请变量时，添加名称参数，如下：

weights = tf.Variable(tf.truncated\_normal([2, 3]), name='weights\_0')

bias = tf.Variable(tf.truncated\_normal([3]), name='bias\_0')

### 正则化和dropout防止过拟合

防止过拟合的两种方法，一、观察验证集性能，在准确率下降的时候终止训练，这叫做提前终止。二、正则化，指对网络应用认为约束，隐式减少自由参数的数量，同时使其更难优化。

正则化，new loss = loss + a\*权重的L2范数

dropout，随机斩断神经元之间一半的链接，将剩下的链接强度乘以2

### dropout实例：

[**tf.nn.dropout()**](https://www.tensorflow.org/api_docs/python/tf/nn/dropout)函数有两个参数：

1. hidden\_layer：你要应用 dropout 的 tensor
2. keep\_prob：任何一个给定单位的留存率（**没有**丢弃的）

keep\_prob 可以让你调整 drop 单位的数量。为了补偿被丢弃的单位，[**tf.nn.dropout()**](https://www.tensorflow.org/api_docs/python/tf/nn/dropout) 把所有保留下来的单位（**没有**丢弃的）乘 1/keep\_prob

在训练时，一个好的keep\_prob初始值是0.5。

在计算准确率和应用模型的时候，keep——prob要设置为1

在测试时，把 keep\_prob 值设为1.0 ，这样保留所有的单位，最大化模型的能力。

keep\_prob = tf.placeholder(tf.float32) *# probability to keep units*

hidden\_layer = tf.add(tf.matmul(features, weights[0]), biases[0])

hidden\_layer = tf.nn.relu(hidden\_layer)

hidden\_layer = tf.nn.dropout(hidden\_layer, keep\_prob)

logits = tf.add(tf.matmul(hidden\_layer, weights[1]), biases[1])

## 权重初始化

权重不能都设为相同的数字

均匀分布，权重的分布范围：[-y,y]，，n为输入层神经元数量

均匀分布

tf.random\_uniform()

正态分布

tf.random\_normal()

参数顺序：shape,mean=0,stddev=1,dtype=tf.float32,seed=None,name=None

截断正态分布：

tf.truncated\_normal()，大于两个标准偏差的数值会被舍弃

截断正太分布表现最好

## 卷积神经网络

<https://zhuanlan.zhihu.com/p/22038289?refer=intelligentunit>

cs231n的翻译

统计不变性-权重共享

### 卷积网络术语

窗口大小称为patch/kernel

图像的每个通道，或者说深度的每一层称为feature map

滑动步幅大小：stride

滑块不越界称为有效填充：valid padding

越过边界，以0填充空白处，称为相同填充：same padding

### 计算下一层的体积(有点问题呀)：

H = height, W = width, D = depth

我们有一个输入形状是 32x32x3 (HxWxD)

20个形状为 8x8x3 (HxWxD) 的过滤器

高和宽的stride（步长）都为 2。(S)

padding 大小为1 (P)

计算新的高度和宽度的公式是：

new\_height = (input\_height - filter\_height + 2 \* P)/S + 1

new\_width = (input\_width - filter\_width + 2 \* P)/S + 1

输出为 14\*14\*20

TensorFlow 使用如下等式计算 SAME 、PADDING

**SAME Padding**， 输出的高和宽，计算如下：

out\_height = ceil(float(in\_height) / float(strides1))

out\_width = ceil(float(in\_width) / float(strides[2]))

**VALID Padding**， 输出的高和宽，计算如下：

out\_height = ceil(float(in\_height - filter\_height + 1) / float(strides1))

out\_width = ceil(float(in\_width - filter\_width + 1) / float(strides[2]))

### 参数数量

上面的那个神经网络，没有参数共享的时候，卷积层的参数数量为：

(8\*8\*3+1)\*(14\*14\*20)=756560，这会耗费极大空间

共享的情况下，数量变为：

(8\*8\*3+1)\*20=3860个

### TensorFlow卷积层

*# Output depth*

k\_output = 64

*# Image Properties*

image\_width = 10

image\_height = 10

color\_channels = 3

*# Convolution filter*

filter\_size\_width = 5

filter\_size\_height = 5

*# Input/Image*

input = tf.placeholder(

tf.float32,

shape=[**None**, image\_height, image\_width, color\_channels])

*# Weight and bias*

weight = tf.Variable(tf.truncated\_normal(

[filter\_size\_height, filter\_size\_width, color\_channels, k\_output]))

bias = tf.Variable(tf.zeros(k\_output))

*# Apply Convolution*

*#strides的参数 (batch, height, width, depth)*

conv\_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')

*# Add bias*

conv\_layer = tf.nn.bias\_add(conv\_layer, bias)

*# Apply activation function*

conv\_layer = tf.nn.relu(conv\_layer)

### pooling池化

其实就是最大值或者平均值滤波，个人认为高斯滤波的效果会更好

conv\_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')

conv\_layer = tf.nn.bias\_add(conv\_layer, bias)

conv\_layer = tf.nn.relu(conv\_layer)

*# Apply Max Pooling*

conv\_layer = tf.nn.max\_pool(

conv\_layer,

ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1],

padding='SAME')

[**tf.nn.max\_pool()**](https://www.tensorflow.org/api_docs/python/tf/nn/max_pool) 函数实现最大池化时， ksize参数是滤波器大小，strides参数是步长。2x2 的滤波器配合 2x2 的步长是常用设定。

ksize 和 strides 参数也被构建为四个元素的列表，每个元素对应 input tensor 的一个维度 ([batch, height, width, channels])，对 ksize 和 strides 来说，batch 和 channel 通常都设置成 1。

### TensorFlow卷积和池化

函数参数中：conv\_ksize,conv\_strides,pool\_ksize,pool\_stredes都是一个长度为2的tuple

def conv2d\_maxpool(x\_tensor, conv\_num\_outputs, conv\_ksize, conv\_strides, pool\_ksize, pool\_strides):

# TODO: Implement Function

print(pool\_ksize)

weight=tf.Variable(tf.truncated\_normal([conv\_ksize[1],conv\_ksize[0],x\_tensor.get\_shape().as\_list()[3],conv\_num\_outputs]))

bias = tf.Variable(tf.zeros(conv\_num\_outputs))

conv\_layer = tf.nn.conv2d(x\_tensor,weight,strides = [1,conv\_strides[0],conv\_strides[1],1],padding='SAME')

conv\_layer = tf.nn.bias\_add(conv\_layer,bias)

conv\_layer = tf.nn.relu(conv\_layer)

conv\_layer = tf.nn.max\_pool(conv\_layer,ksize = [1,pool\_ksize[0],pool\_ksize[1],1],\

strides = [1,pool\_strides[0],pool\_strides[1],1],padding = 'SAME')

return conv\_layer

### 1\*1卷积，没看懂

### inception module

卷积层变为一个集合，1\*1+3\*3+5\*5滤波器的输出加上平均池化的输出，可以提高模型的性能

### 卷积神经网络一般结构

conv  
maxpool  
(conv  
maxpool)  
...  
flatten  
fully\_connected  
dropout  
(fully\_connected  
dropout)  
...  
output  
注意：  
卷积层通过卷积，把图像缩小，“厚度”(conv\_num\_outputs)应该逐层增加，这样才能有效保留图像信息。  
全链接层需要把“图像条”归纳到十个类别，所以num\_outputs应该逐层递减。

可以参考这个链接： https://zhuanlan.zhihu.com/p/22038289

### 卷积神经网络完整代码

#### 数据集

你从之前的课程中见过这节课的代码。这里我们导入 MNIST 数据集，用一个方便的函数完成对数据集的 batch，缩放和独热编码。

**from** tensorflow.examples.tutorials.mnist **import** input\_data

mnist = input\_data.read\_data\_sets(".", one\_hot=**True**, reshape=**False**)

**import** tensorflow **as** tf

*# Parameters*

*# 参数*

learning\_rate = 0.00001

epochs = 10

batch\_size = 128

*# Number of samples to calculate validation and accuracy*

*# Decrease this if you're running out of memory to calculate accuracy*

*# 用来验证和计算准确率的样本数*

*# 如果内存不够，可以调小这个数字*

test\_valid\_size = 256

*# Network Parameters*

*# 神经网络参数*

n\_classes = 10 *# MNIST total classes (0-9 digits)*

dropout = 0.75 *# Dropout, probability to keep units*

#### Weights and Biases

在设置每层的权重时候，就要提前设计好自己网络的结构，包括由滤波器不同影响每一层的图像尺寸。

卷积层的4个参数为：滤波器高，滤波器宽，输入图像深度（层数），输出图像深度。非卷积层不用前两个参数

*# Store layers weight & bias*

weights = {

'wc1': tf.Variable(tf.random\_normal([5, 5, 1, 32])),

'wc2': tf.Variable(tf.random\_normal([5, 5, 32, 64])),

'wd1': tf.Variable(tf.random\_normal([7\*7\*64, 1024])),

'out': tf.Variable(tf.random\_normal([1024, n\_classes]))}

biases = {

'bc1': tf.Variable(tf.random\_normal([32])),

'bc2': tf.Variable(tf.random\_normal([64])),

'bd1': tf.Variable(tf.random\_normal([1024])),

'out': tf.Variable(tf.random\_normal([n\_classes]))}

#### 卷积+偏置

**def** **conv2d**(x, W, b, strides=1):

x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')

x = tf.nn.bias\_add(x, b)

**return** tf.nn.relu(x)

**strides**的参数 (batch, height, width, depth)

strides 是一个4个元素的序列；第一个位置指的是对 batch 的stride，最后一个位置指对特征的 stride。最好的移除 batches 和 特征的方法是你直接在数据集中把他们忽略，而不是使用 stride。要使用所有的 batches 和特征，你可以把第一个和最后一个元素设成 1。

#### 最大池化

**def** **maxpool2d**(x, k=2):

**return** tf.nn.max\_pool(

x,

ksize=[1, k, k, 1],

strides=[1, k, k, 1],

padding='SAME')

strides的理解同上，ksize的首末位不确定

#### 模型

如上所述，在设计weight的时候，每一步计算生成图片的尺寸就提前计算好了。

两次卷积，一次全连接

**def** **conv\_net**(x, weights, biases, dropout):

*# Layer 1 - 28\*28\*1 to 14\*14\*32*

conv1 = conv2d(x, weights['wc1'], biases['bc1'])

conv1 = maxpool2d(conv1, k=2)

*# Layer 2 - 14\*14\*32 to 7\*7\*64*

conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])

conv2 = maxpool2d(conv2, k=2)

*# Fully connected layer - 7\*7\*64 to 1024*

fc1 = tf.reshape(conv2, [-1, weights['wd1'].get\_shape().as\_list()[0]])

fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])

fc1 = tf.nn.relu(fc1)

fc1 = tf.nn.dropout(fc1, dropout)

*# Output Layer - class prediction - 1024 to 10*

out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])

**return** out

#### Session

*# tf Graph input*

x = tf.placeholder(tf.float32, [**None**, 28, 28, 1])

y = tf.placeholder(tf.float32, [**None**, n\_classes])

keep\_prob = tf.placeholder(tf.float32)

*# Model*

logits = conv\_net(x, weights, biases, keep\_prob)

*# Define loss and optimizer*

cost = tf.reduce\_mean(\

tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=y))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate)\

.minimize(cost)

*# Accuracy*

correct\_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

*# Initializing the variables*

init = tf. global\_variables \_initializer()

*# Launch the graph*

**with** tf.Session() **as** sess:

sess.run(init)

**for** epoch **in** range(epochs):

**for** batch **in** range(mnist.train.num\_examples//batch\_size):

batch\_x, batch\_y = mnist.train.next\_batch(batch\_size)

sess.run(optimizer, feed\_dict={

x: batch\_x,

y: batch\_y,

keep\_prob: dropout})

*# Calculate batch loss and accuracy*

loss = sess.run(cost, feed\_dict={

x: batch\_x,

y: batch\_y,

keep\_prob: 1.})

valid\_acc = sess.run(accuracy, feed\_dict={

x: mnist.validation.images[:test\_valid\_size],

y: mnist.validation.labels[:test\_valid\_size],

keep\_prob: 1.})

print('Epoch {:>2}, Batch {:>3} -'

'Loss: {:>10.4f} Validation Accuracy: {:.6f}'.format(

epoch + 1,

batch + 1,

loss,

valid\_acc))

*# Calculate Test Accuracy*

test\_acc = sess.run(accuracy, feed\_dict={

x: mnist.test.images[:test\_valid\_size],

y: mnist.test.labels[:test\_valid\_size],

keep\_prob: 1.})

print('Testing Accuracy: {}'.format(test\_acc))

## 使用floyd

1. 在 [floydhub.com](https://www.floydhub.com/) 上创建一个帐户（别忘了确认电子邮件）。你将自动获得 100 个免费 GPU 小时。
2. 在你的计算机上运行 floyd 命令：

pip install -U floyd-cli

即使你之前已经安装了 floyd-cli，也要执行这一步。确保你安装的是最新版本（它的开发速度很快！）。

1. 用此命令绑定你的 Floyd 账号：

floyd login

（系统会在浏览器中打开一个具有身份验证令牌的页面，你需要将该令牌复制到你的终端里）

1. 克隆代码库:

git clone https://github.com/ludwiktrammer/deep-learning.git

注意：这个代码库与优达学城的代码库之间有些许差别。你可以在 [README](https://github.com/ludwiktrammer/deep-learning/tree/master/image-classification#how-is-this-repository-different-from-the-original) 找到详细说明。要使用这份说明一步步操作，我们建议你使用 ludwiktrammer 的代码库。

1. 进入图像分类项目文件夹：

cd image-classification

1. 初始化 Floyd 项目：

floyd init dlnd\_image\_classification

1. 运行项目：

floyd run --gpu --env tensorflow --mode jupyter --data diSgciLH4WA7HpcHNasP9j

这段命令的意思是：它将在有 GPU（--gpu）的机器上运行，使用 Tenserflow 环境（--env tensorflow），使用 Jupyter 记事本（--mode jupyter），且可用 Floyd 的内置的 cifar-10 数据集（--data diSgciLH4WA7HpcHNasP9j）。

等待 Jupyter 记事本准备好，然后复制终端里显示的 URL（见 “path to jupyter notebook”）在浏览器中打开，你将看到该记事本。

8 停止使用

当你没有使用该记事本时，请记得关闭实验（experiment）。只要实验在运行（即使是在后台运行），就会消耗 GPU 时间，而你只有 100 小时的免费时间。你可以在 floyd.com 的“[Experiments](https://www.floydhub.com/experiments)”部分，或使用 floyd stop 命令停止实验：

floyd stop ID

（其中 ID 是当你运行该项目时，在终端里显示的 “RUN ID”。如果你找不到该 ID，可以在 floyd.com 的“[Experiments](https://www.floydhub.com/experiments)”部分找到。）

**重要提醒**：当你运行项目时，它将始终从头开始（即从计算机上的本地状态开始）。如果你在此前的运行中，对服务器上的 Jupiter 记事本上做了修改，这些更改将不会在之后的运行中生效。要永久保留这些更改，你需要将这些更改添加到本地项目文件夹中。运行记事本时，你可以直接从 Jupyter 菜单栏的 - File / Download / Notebook 下载记事本。下载完毕后，将本地的dlnd\_image\_classification.ipynb 文件替换为新下载的文件即可。

如果你已经停止实验，依然可以使用 floyd output 命令下载文件：

floyd output ID

（其中 ID 是当你运行该项目时，在终端里显示的 “RUN ID”。如果你找不到该 ID，可以在 floyd.com 的“[Experiments](https://www.floydhub.com/experiments)”部分找到。）

只需运行上述命令，下载 dlnd\_image\_classification.ipynb，并将本地版本替换为新下载的文件即可。

# 三、循环神经网络

## 1、intro-to-rnns/LSTM

下面这个链接对LSTM 和 RNN解释的非常清楚，可以参考学习下：  
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>，这个文章有空要看一下

循环神经网络不错，但是单独使用会因为梯度消失和梯度爆炸的情况而表现很差，用LSTM来处理梯度消失，用梯度修剪来防止梯度爆炸。

RNN是目前的研究热点，但是一些表现很好的结构并没有明确的解释。

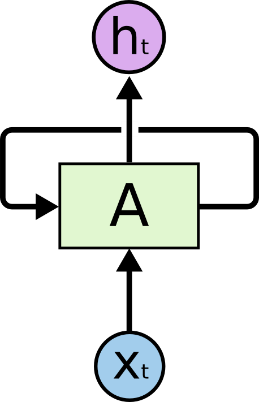
### Understanding LSTM Networks博文翻译

原文地址：<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

人类每秒钟都不会从头开始思考。 在阅读本文时，您可以根据您对以前单词的理解来理解每个单词。 你不要把所有东西都扔掉，再从头开始思考。 你的想法有持久性。

传统的神经网络不能做到这一点，这似乎是一个主要的缺点。 例如，想象一下，您想要分析电影中每一点发生的事件。 传统神经网络如何能够利用其关于电影中以前的事件的推理来通知以后的事情是不清楚的。

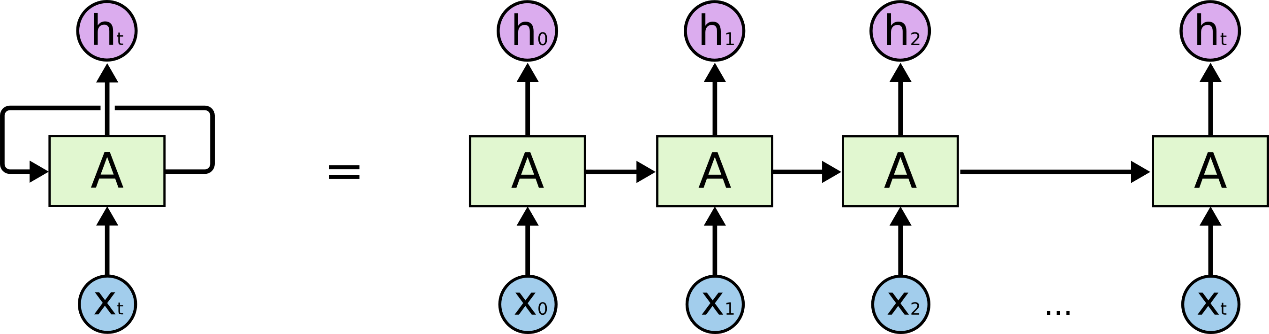
循环神经网络解决了这个问题。 它们是具有循环的网络，允许信息持续存在。



**Recurrent Neural Networks have loops.**

在上图中，一组神经网络A查看某些输入xt并输出一个值ht。 循环允许信息从网络的一个步骤传递到下一个。

这些循环使得循环神经网络似乎是神秘的。 然而，如果你再想一点，结果是它们并不是一般的神经网络的不同。 一个循环神经网络可以被认为是同一个网络的多个副本，每一个都传递一个信息给后继者。 考虑如果我们展开循环会发生什么：



**An unrolled recurrent neural network.**

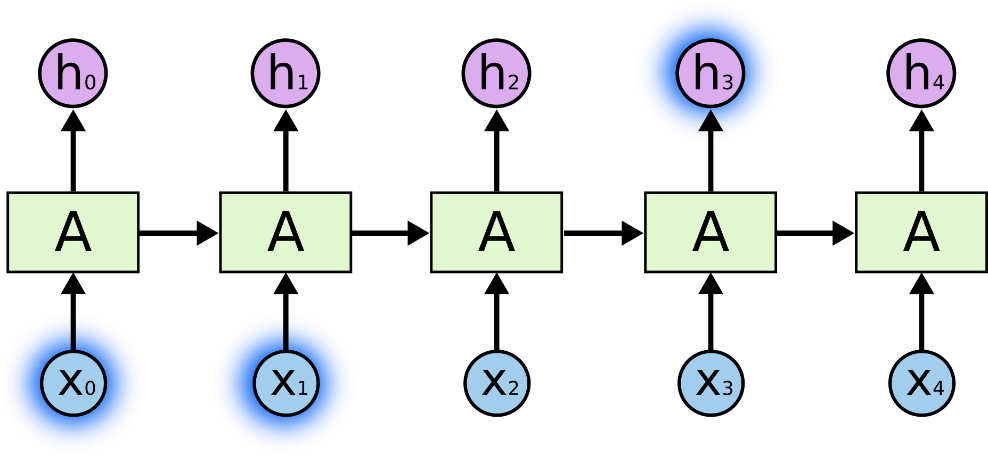
这种链状特征揭示了循环神经网络与序列和列表密切相关。 它们是用于此类数据的神经网络的自然结构。

而且他们肯定是用的！ 在过去几年中，RNN已经成功应用于语音识别，语言建模，翻译，图像字幕等各种问题。 我将讨论可以通过RNN实现的令人惊叹的专长，使Andrej Karpathy的优秀博客“反复神经网络的不合理有效性”成为可能。 但他们真的很神奇。

这些成功的关键在于使用“LSTM”，这是一种非常特殊的循环神经网络，对于许多任务来说，它们比标准版本好得多。 几乎所有令人兴奋的结果基于循环神经网络是通过他们实现的。 这篇文章将探讨这些LSTM。

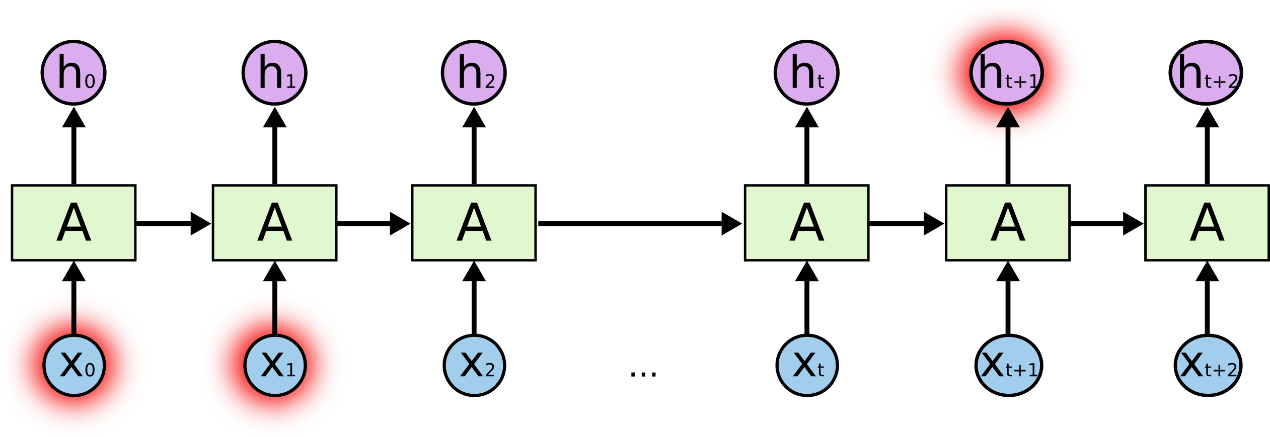
RNN的上诉之一是他们可能能够将先前的信息连接到当前任务的想法，例如使用先前的视频帧可以通知对当前帧的理解。 如果RNN可以做到这一点，他们将非常有用。 但可以吗 这取决于具体情况。

有时，我们只需要查看最近的信息来执行当前的任务。 例如，考虑一种语言模型，尝试基于以前的单词来预测下一个单词。 如果我们试图预测“云在天空中”的最后一个字，我们不需要任何进一步的语境 - 下一个字将是天空很明显。 在这种情况下，如果相关信息与所需地点之间的差距很小，则RNN可以学习使用过去的信息。



但也有需要更多上下文的情况。 考虑尝试预测文本中的最后一个单词“我在法国长大...我说流利的法语”。最近的信息表明下一个单词可能是一种语言的名称，但如果我们想缩小哪种语言，我们需要后文提到的法国。 相关信息之间的差距和需要变得非常大的点是完全可能的。

不幸的是，随着差距的增加，RNN无法学会连接信息。



在理论上，RNN绝对有能力处理这样的“长期依赖”。人们可以仔细挑选参数来解决这种形式的玩具问题。 可悲的是，实际上，RNN似乎无法学习。 Hochreiter（1991）[German]和Bengio等人深入探讨了这个问题。 （1994），他们发现了一些很可能是困难的根本原因。

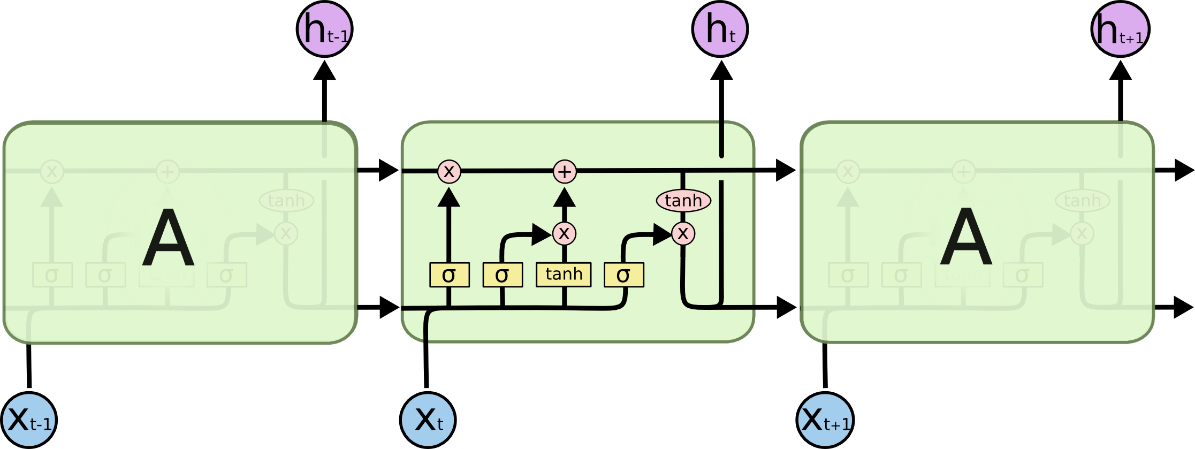
幸运的是，LSTM没有这个问题！

长时间内存网络（通常称为“LSTM”）是一种特殊的RNN，能够学习长期的依赖关系。 他们由Hochreiter＆Schmidhuber（1997）介绍，并被许多人进行了精细化和推广。1他们在各种各样的问题上工作得很好，现在被广泛使用。

LSTM是明确设计的，以避免长期依赖问题。 记住长时间的信息实际上是他们的默认行为，而不是他们难以学习的东西！

所有复发神经网络都具有神经网络重复模块链的形式。 在标准RNN中，该重复模块将具有非常简单的结构，例如单个tanh层。

LSTM也具有这样的链结构，但重复模块具有不同的结构。有四个神经网络层，以非常特殊的方式进行交互。



**The repeating module in an LSTM contains four interacting layers.**

不要担心发生了什么的细节。 我们将逐步介绍LSTM图。 现在，让我们尽量让我们使用我们使用的符号。

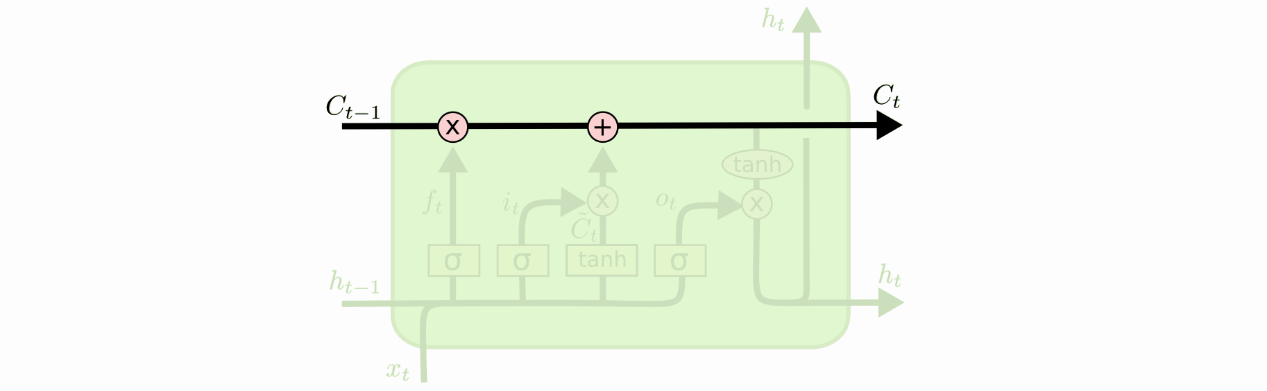


在上图中，每一行都带有一个整个向量，从一个节点的输出到其他节点的输入。 粉色圆圈表示点向运算，如向量加法，而黄色框是学习神经网络层。 行合并表示串联，而行叉表示其内容正在复制，副本将转到不同的位置。

#### The Core Idea Behind LSTMs

LSTM的关键是细胞状态，水平线穿过图的顶部。

Cell状态类似于输送带。 它直接在整个链上运行，只有一些小的线性相互作用。 信息流畅地保持不变。



LSTM确实具有去除或添加信息到细胞状态的能力，由称为门的结构小心调节。

门是一种可选地让信息通过的方式。 它们由sigmoid神经网络层和点向乘法运算组成。

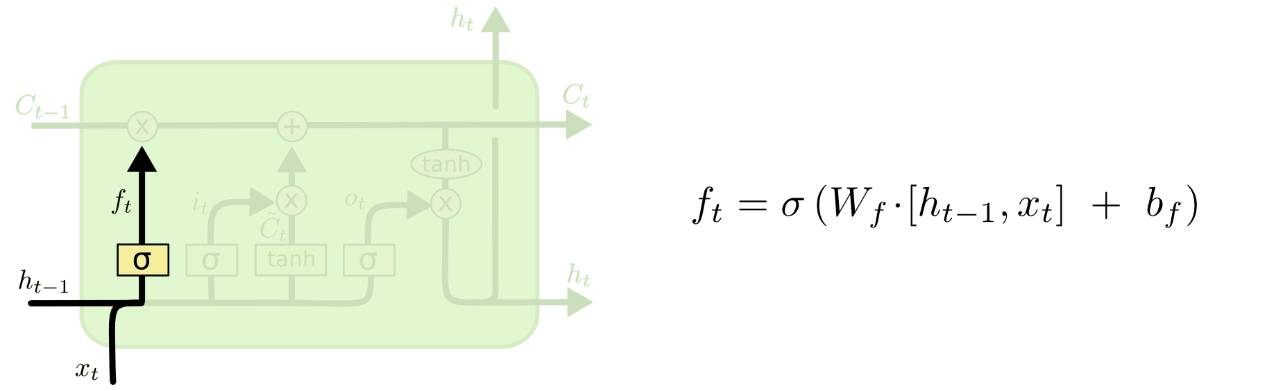


sigmoid结构层输出零和一之间的数字，描述每个组件应该通过多少。 值为零表示“没有任何通过”，而一个值意味着“让一切通过！”

#### Step-by-Step LSTM Walk Through

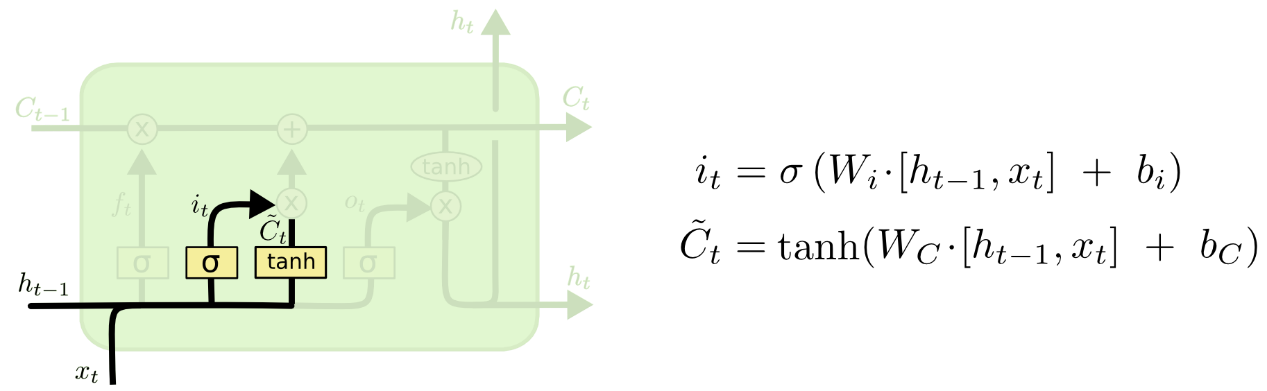
我们的LSTM的第一步是决定我们要从单元格状态中丢弃什么信息。 该决定由称为“忘记门层”的sigmoid层进行。它查看ht-1和xt，并且在单元格状态Ct-1中为每个数字输出0和1之间的数字。 1代表“完全保持这个”，而0代表“完全摆脱”。

我们回到我们的一个语言模型的例子，试图根据所有以前的语料来预测下一个单词。 在这样一个问题中，细胞状态可能包括当前受试者的性别，从而可以使用正确的代词。 当我们看到一个新主题时，我们想要忘记这个老主题的性别。



下一步是决定我们要在单元格状态下存储的新信息。 这有两部分。 首先，称为“输入门层”的S形层决定了我们将更新哪些值。 接下来，tanh层创建可以被添加到状态的新候选值Ct的向量。 在下一步中，我们将结合这两个来创建对状态的更新。

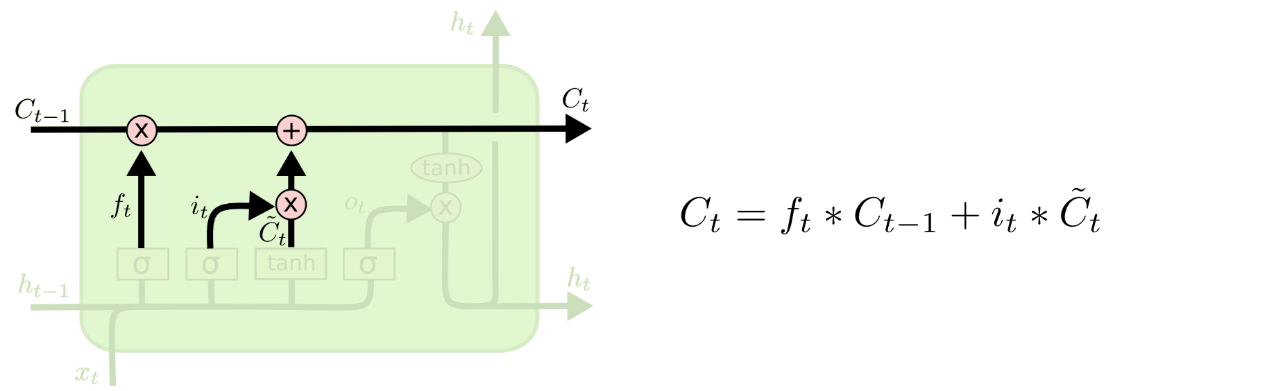
在我们的语言模型的例子中，我们希望将新主题的性别添加到单元格状态，以替换我们忘记的旧对象。



现在是将旧单元状态Ct-1更新为新的单元格状态Ct的时候了。 以前的步骤已经决定了要做什么，我们只需要做到这一点。

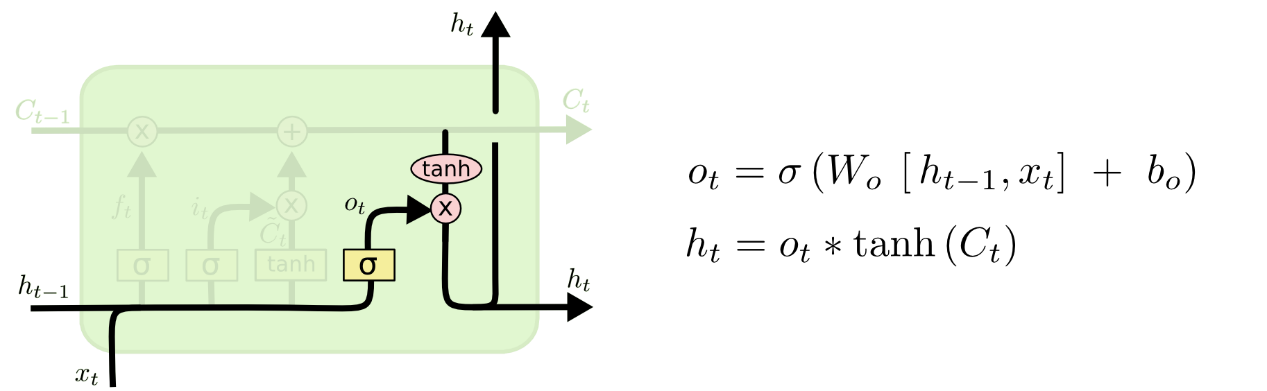
我们将旧状态乘以ft，忘记了我们之前决定忘记的事情。 然后我们添加它\* Ct。 这是新的候选值，按照我们决定更新每个状态值的多少来缩放。

在语言模型的情况下，我们实际上会放弃关于旧主题性别的信息，并添加新信息，就像我们之前的步骤中决定一样。



最后，我们需要决定要输出的内容。 此输出将基于我们的单元格状态，但将是一个过滤版本。 首先，我们运行一个sigmoid层，决定了我们要输出的单元格状态的哪些部分。 然后，我们将单元格状态通过tanhtanh（将值推到-1和1之间），并将其乘以Sigmoid门的输出，以便我们只输出我们决定的部分。

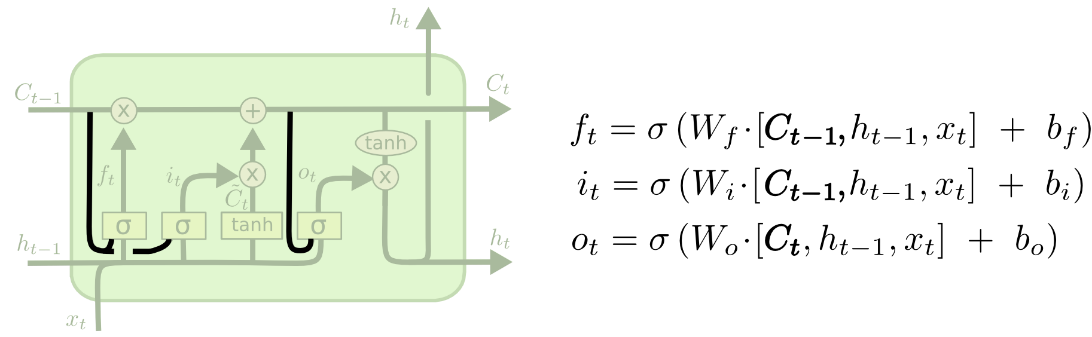
对于语言模型示例，由于它只是看到一个主题，它可能需要输出与动词相关的信息，以防下一步发生。 例如，它可能会输出主题是单数还是复数，以便我们知道动词应该如何组合在一起，如果这是接下来的内容。



#### Variants on Long Short Term Memory

到目前为止，我所描述的是一个很正常的LSTM。 但并不是所有的LSTM都与上述相同。 事实上，似乎几乎每一篇涉及LSTM的论文都使用了一个略有不同的版本。 差异很小，但值得一提的是其中之一。

Gers＆Schmidhuber（2000）介绍的一种受欢迎的LSTM变体正在添加“窥视孔连接”。这意味着我们让门层看着细胞状态。



还有几个变体这里不加了

## 2、word2vec

<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>

博文翻译

我们从高层了解我们将要从哪里开始。 Word2Vec使用您可能在机器学习的其他地方看到的技巧。 我们要训练一个具有单个隐藏层的简单神经网络来执行某个任务，但是我们实际上并没有使用该神经网络来完成我们对其进行的培训。 相反，目标实际上只是为了学习隐藏层的权重 - 我们会看到这些权重实际上是我们试图学习的“字向量”。

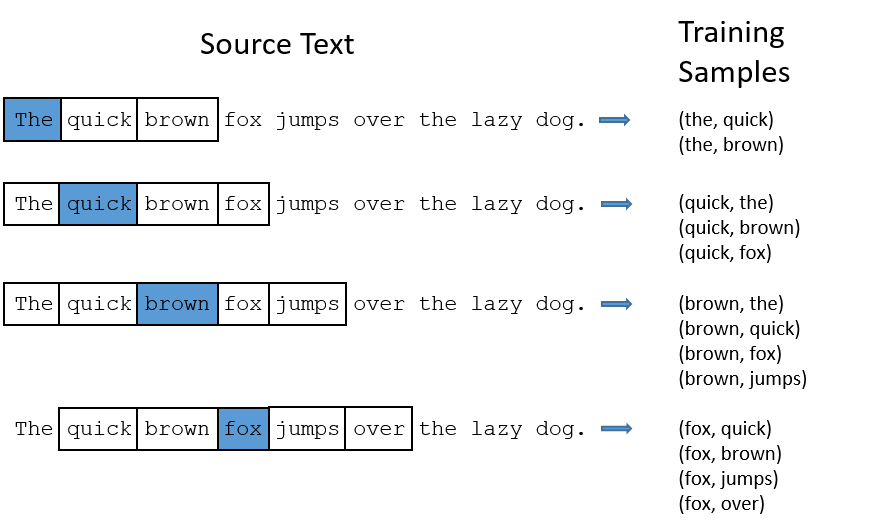
您可能看到这个诀窍的另一个地方是在无监督学习中，您可以在其中训练自动编码器来压缩隐藏层中的输入向量，并将其解压缩回输出层中的原始状态。 训练完毕后，您将剥离输出层（解压缩步骤），只需使用隐藏层 - 这是学习良好图像功能的技巧，而无需标记培训数据。

所以现在我们需要谈论这个我们要建立神经网络来执行的“假的”任务，然后我们稍后再来回顾一下，这是如何间接地给我们这些我们真正在后面的字矢量的。

我们要训练神经网络来做以下操作。 给定一个句子中的一个特定的单词（输入单词），看看附近的单词，随机选择一个。 网络将告诉我们，我们词汇中每个单词作为我们选择的“附近单词”的概率。这里的附近指的是前后窗口，前后距离5以内，即共10个单词

输出概率将与我们的输入字附近找到每个词汇单词的可能性有关。 例如，如果你给训练有素的网络输入了“苏联”字样，输出概率就会比“联盟”和“俄罗斯”这个词比“西瓜”和“袋鼠”这样的无关词更高。

我们将训练神经网络，通过喂养我们的训练文档中找到的单词对来完成。 下面的例子显示了我们从“快速的棕色狐狸跳过懒狗”这个句子的一些训练样本（单词对）。我使用了一个小窗口大小为2的例子。 蓝色突出显示的字是输入字。



网络将从每次配对显示的次数中了解统计信息。 所以，例如，网络可能会比（“苏联”，“萨斯奎奇”）更多地获得（“苏联”，“联盟”）的更多培训样本。 训练结束后，如果以“苏联”一词作为输入，那么“联盟”或“俄罗斯”比“萨斯奎奇”的概率要高得多。

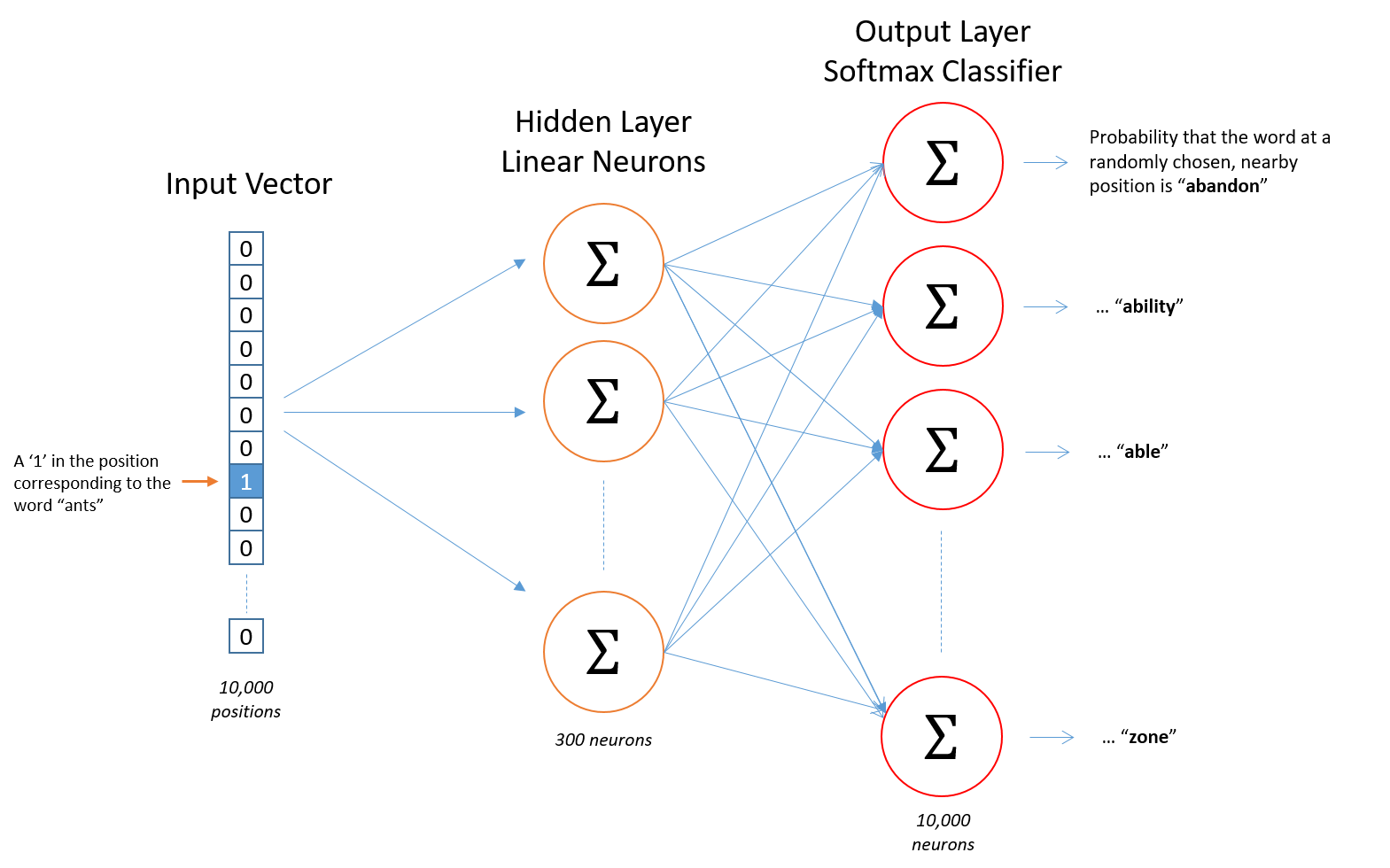
更多细节

首先，你知道你不能像一个文本字符串给一个神经网络一个个单词，所以我们需要一种方式来表示网络的单词。 为了做到这一点，我们首先从我们的培训文件中构建一个单词的词汇 - 假设我们有一个10,000个独特词汇的词汇。

我们将像“蚂蚁”一样输入一个输入单词作为一个独热的向量。 该矢量将有10,000个组件（我们的词汇表中的每个单词一个），我们将在对应于单词“ants”的位置放置一个“1”，在所有其他位置都放置“0”。

网络的输出是单个向量（也包含10,000个组件），对于我们的词汇表中的每个单词，包含随机选择的附近单词是该词汇单词的概率。

这是我们神经网络的架构。



隐层神经元没有激活函数，但输出神经元使用softmax。

当在字对上训练这个网络时，输入是表示输入字的单热矢量，训练输出也是表示输出字的单热矢量。 但是，当您对输入字进行训练网络评估时，输出向量实际上将是概率分布（即，一堆浮点值，而不是一个热向量）。

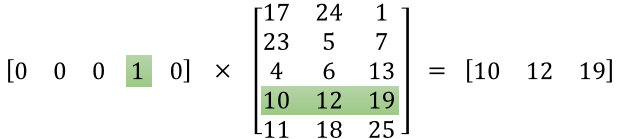
隐藏层

对于我们的例子，我们要说我们正在学习具有300个特征的字矢量。 所以隐含层将由10,000行（一个用于我们的词汇中的每个单词）和300个列（每个隐藏的神经元一个）的权重矩阵表示·。这个300，是谷歌所用的参数

所以所有这一切的最终目标就是要学习这个隐层重量矩阵 -

不过，让我们回头一点，就是要完成我们要训练的这个模型的定义。

现在，你可能会问自己 - “那个一个热的矢量几乎全是零，这有什么影响？”如果你乘以一个1×10,000的一个热矢量，用一个10,000×300的矩阵，它将有效地选择 矩阵行对应于“1”。 这是一个小例子给你一个视觉效果。

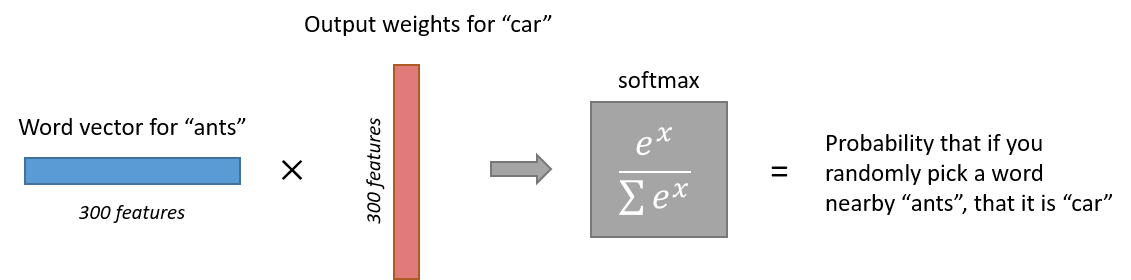


这意味着这个模型的隐藏层真的只是作为一个查找表来操作。 隐藏层的输出只是输入字的“字矢量”。

ants”的1 x 300字矢量然后被馈送到输出层。 输出层是一个softmax回归分类器。 这里有一个关于Softmax回归的深入教程，但其主旨是每个输出神经元（我们的词汇中的每个单词一个）将产生0到1之间的输出，并且所有这些输出值的总和将相加 到1。

具体来说，每个输出神经元具有与隐藏层相乘的字向量的权重向量，然后将函数exp（x）应用于结果。 最后，为了将输出总和为1，我们将结果除以所有10,000个输出节点的结果之和。

以下是计算“car”一词输出神经元输出的图示。



注意，神经网络不知道输出字相对于输入字的偏移量。 它不会在输入与单词之后的单词中学习一组不同的概率。 要理解这个含义，让我们来说，在我们的训练语料库中，单词“约克”的每一个单词都在“New”之前。 也就是说，至少根据培训数据，“纽约”将在“约克”附近有100％的概率。 但是，如果我们在“约克”附近拿10个字，随机选择一个，那么“新”的概率不是100％; 你可能选择了附近的其他单词。

## 超参数

### 学习率

可以每隔几个循环，就让学习率乘一个系数，可以使0.5，也可以是0.1.

Tensorflow有自适应学习率改变

### Minibatch\_Size

过小，训练速度慢，需要较小计算资源，会得到比较好的结果

过大，训练速度快，需要较大计算资源，容易卡在一个局部最小值

建议：lr = 0.01时候，建议batch\_size = 32,64,128,256

或者lr = 0.01\*batch\_size/256

### Epoch

### 隐藏层数量

隐藏层过多时容易过拟合。

第一个隐藏层的node数量多于输入。

卷积神经网络越深越好，别的网络3层就可以了，再深并没有看到更好的效果。

## Get\_batch数据分批

我自己写的垃圾代码：

def get\_batches(int\_text, batch\_size, seq\_length):

"""

Return batches of input and target

:param int\_text: Text with the words replaced by their ids

:param batch\_size: The size of batch

:param seq\_length: The length of sequence

:return: Batches as a Numpy array

"""

# TODO: Implement Function

batch\_length = batch\_size\*seq\_length

batch\_num = len(int\_text)//batch\_length

x = int\_text[:batch\_length\*batch\_num]

if(len(int\_text)>batch\_length\*batch\_num):

y = int\_text[1:batch\_length\*batch\_num+1]

else:

y = x[1:].append(x[0])

final\_batch = []

for i in range(0,len(x),batch\_length):

batch\_text\_x = x[i:i+batch\_length]

batch\_text\_y = y[i:i+batch\_length]

batch\_x = []

batch\_y = []

for ii in range(0,batch\_length,seq\_length):

batch\_x.append(batch\_text\_x[ii:ii+seq\_length])

batch\_y.append(batch\_text\_y[ii:ii+seq\_length])

single\_batch = [batch\_x,batch\_y]

final\_batch.append(single\_batch)

final\_batch = np.array(final\_batch)

return final\_batch

例子：

int\_text = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

a = get\_batches(int\_text,2,3)

a

输出如下：

array([[[[ 1, 2, 3],

[ 4, 5, 6]],

[[ 2, 3, 4],

[ 5, 6, 7]]],

[[[ 7, 8, 9],

[10, 11, 12]],

[[ 8, 9, 10],

[11, 12, 13]]],

[[[13, 14, 15],

[16, 17, 18]],

[[14, 15, 16],

[17, 18, 19]]]])

优美的代码：

n\_batches = int(len(int\_text) / (batch\_size \* seq\_length))

# Drop the last few characters to make only full batches

xdata = np.array(int\_text[: n\_batches \* batch\_size \* seq\_length])

ydata = np.array(int\_text[1: n\_batches \* batch\_size \* seq\_length + 1])

ydata[-1] = xdata[0]

x\_batches = np.split(xdata.reshape(batch\_size, -1), n\_batches, 1)

y\_batches = np.split(ydata.reshape(batch\_size, -1), n\_batches, 1)

return np.array(list(zip(x\_batches, y\_batches)))

还有一个经典的节省内存的方法，在intro-to-rnn里

def get\_batches(arr, n\_seqs, n\_steps):

'''Create a generator that returns batches of size

n\_seqs x n\_steps from arr.

Arguments

---------

arr: Array you want to make batches from

n\_seqs: Batch size, the number of sequences per batch

n\_steps: Number of sequence steps per batch

'''

# Get the number of characters per batch and number of batches we can make

characters\_per\_batch = n\_seqs \* n\_steps

n\_batches = len(arr)//characters\_per\_batch

# Keep only enough characters to make full batches

arr = arr[:n\_batches \* characters\_per\_batch]

# Reshape into n\_seqs rows

arr = arr.reshape((n\_seqs, -1))

for n in range(0, arr.shape[1], n\_steps):

# The features

x = arr[:, n:n+n\_steps]

# The targets, shifted by one

y = np.zeros\_like(x)

y[:, :-1], y[:, -1] = x[:, 1:], x[:, 0]

yield x, y

# 四、数据分析入门课程

## 一、数据分析一般过程：

1、提出问题

2、获取数据、清洗数据

3、寻找模式

4、验证猜测

5、交流

## 二、一般术语

与数据分析相比，以下术语的特点

Data Science：Similar to data analysis, More Focused on building system,Need more experience

数据科学：与数据分析相似，数据分析用关于发论文之类，数据科学侧重于构建系统性工程，相比之下需要更多经验

数据工程：注重数据再加工，更多做数据存储和处理及扩展

## 第一节：数据分析过程

### 打开文件操作：

import csv

with open('enrollments.csv', 'rb') as f:

reader = csv.DictReader(f)

enrollments = list(reader)

rb：read without truncation（不截断，阅读模式）

缩进结束后，文件自动关闭，这样不用主动调用f.close()

### python转换str到时间类

from datetime import datetime as dt

time = dt.strptime(date, '%Y-%m-%d')

### 把list活动记录中的内容按某个key分组（比如名字）

建立一个字典，每个key对应一个list来存放记录

from collections import defaultdict

engagement\_by\_account = defaultdict(list)

for engagement\_record in paid\_engagement\_in\_first\_week:

account\_key = engagement\_record['account\_key']

engagement\_by\_account[account\_key].append(engagement\_record)

### 在这个分组中，统计每个人的活动记录总数

将统计结果放在list中

minutes\_pass = {}

for student,records in pass\_group.items():

minutes = 0

for record in records:

minutes += record['total\_minutes\_visited']

student=records[0]['account\_key']

minutes\_pass[student] = minutes

### 直方图显示数据

%matplotlib inline 在笔记本页面显示图而不会新建标签页

import seaborn as sns 自动美化图表

import matplotlib.pyplot as plt

plt.hist(data) 显示数据

plt.hist(data，bins=7) 设置直方图条数

轴说明和图题

plt.xlabel(‘x label’)

plt.title(‘’)

## 第二节：numpy和pandas处理一维数据

### np返回最大值索引

np.max()返回最大值

np.argmax()返回最大值索引

### print格式化输出

print(‘country {} has employment {}’.format(countrymame,number)

### 标准化数据

(values – values.mean())/values.std()

### 索引数组

a=np.array([1,2,3,4,5])

b=a>2会得到这样一个索引数组：[F,F,T,T,T]

而a[b]的输出是[3,4,5]

这样，就是用一个索引数组来对数组进行操作，也可以直接这样写

a[a>2]

### 原地：数组与内存/原位运算

a=np.array([1,2,3,4])

b=a //此时，a和b共享同一个内存地址

下面的命令会产生不同结果

+= 号称**原位运算**，即在原来地址改变内存

a += np.array([1,1,1,1])，会在原来的地址上加1，a，b都会变成[2,3,4,5]

这个命令会有不同结果:

a = a + np.array([1,1,1,1])

这个命令会重新**申请一块内存**地址给a，所以这时a，b的值是你不同的

### 原地：慎重对待切片

a = np.array([1,2,3,4])

b = a[:3]

此时，b为数组a的切片，但是没有新申请内存，所以更改b的值也会更改a的值

## Pandas series

s.describe() 会输出平均值、标准偏差、中位数，但是numpy没有这个函数

与numpy的相似处：

获取元素或切片： S[0] S[:6]

循环 for x in s

统计函数 s.mean() s.max()

向量运算 s1+s2

### pandas索引

a = pd.Series([74.7,75.,83.4,57.6],

index = [‘A’,’B’,C’,’D’])

a.loc[‘A’]的输出为74.7

a.iloc[0]的输出也为74.7

这两个的术语分别为**索引和位置**

### pandas按照索引相加

s1 = pd.Series([1,2,3,4],index=[‘a’,’b’,’c’,’d’])

s2 = pd.Series([10,20,30,40],index = [‘c’,’d’,’e’,’f’])

result = s1 + s2得到的结果是按索引值相加，结果如下：

a NaN

b Nan

c 13

d 24

e NaN

f NaN

### dropna()

如果不想要这些空值，可以用dropna()函数：

result.dropna()的输出如下：

c 13

d 24

### pd.add()

这个方法把缺省值视为0

s1.add(s2,fill\_value = 0)

得到结果如下：

a 1

b 2

c 13

d 24

e 30

f 40

### pandas的最大值

获取索引

index = pandas.argmax()

根据索引获得值

max\_value = pandas.loc[index]

### 遍历padas内的一维数据

for in

new\_pd = pd.apply(fun)

## 第三节： 二维数组

1，从二维numpy获取元素：a[1,2]

2,mean()，std()作用于整个二维数组，

a[1:3,3:5] 获取a的1-2行，3-4列元素

a[1,:] 获取a的第1行元素

### pandas

示例代码：

ridership\_df = pd.DataFrame(

data=[[ 0, 0, 2, 5, 0],

[1478, 3877, 3674, 2328, 2539],

[1613, 4088, 3991, 6461, 2691],

[1560, 3392, 3826, 4787, 2613],

[1608, 4802, 3932, 4477, 2705],

[1576, 3933, 3909, 4979, 2685],

[ 95, 229, 255, 496, 201],

[ 2, 0, 1, 27, 0],

[1438, 3785, 3589, 4174, 2215],

[1342, 4043, 4009, 4665, 3033]],

index=['05-01-11', '05-02-11', '05-03-11', '05-04-11', '05-05-11',

'05-06-11', '05-07-11', '05-08-11', '05-09-11', '05-10-11'],

columns=['R003', 'R004', 'R005', 'R006', 'R007']

)

print rider\_ship

df\_1 = pd.DataFrame({'A': [0, 1, 2], 'B': [3, 4, 5]})

print df\_1

A B

0 0 3

1 1 4

2 2 5

df\_2 = pd.DataFrame([[0, 1, 2], [3, 4, 5]], columns=['A', 'B', 'C'])

print df\_2

A B C

0 0 1 2

1 3 4 5

numpy的组内数据类型必须统一

pandas可以有不同的数据类型

pandas.loc[‘ ‘]以及pandas.iloc[‘ ‘]可以获取某一行的数据

pandas[‘ ‘]可以获取某一列数据

pandas.values 可以返回array格式的数据全部数据，支持mean等。

pd = pd.read\_csv(‘ ‘)读取csv数据

pd.head()为打印前5行数据

pd.describe()为统计数据指标

### 皮尔逊积矩相关系数

NumPy.corrcoef()计算‘’**皮尔逊积矩相关系数**。计算原理如下

def correlation(x, y):

x\_std = (x-x.mean()) / x.std(ddof = 0)

y\_std = (y-y.mean()) / y.std(ddof = 0)

return (x\_std \* y\_std).mean()

数轴

pd.mean(axis = 0)，返回的结果为每一列的平局值

### pandas相加

pandas + pandas，Series+Series对应位置相加

pandas+Series，把Series视为一行，对应的index的情况下，加在pandas的每一行，形状不符，对应位置为Nan，如果pandas的index和Series的index不符合，则都为Nan

特列，只有一列的pandas想和Series相加，可以用pd.add(s,axis = ‘index’)，这样会把原来为行的Series变为列进行相加。同理，相减可以用sub。

### pandas计算行、列数据的参数

求一列的平均值，用默认函数mean（）

求一行的平均值，用mean（axix = ‘coloum’）

求列与列之间的四则运算直接用符号

求行之间的四则运算，add,sub,div(s,axis = ‘index’)

### pandas.shift()

整体位移数据，pandas.shift(1)就是把数据向下位移一格

### 遍历pandas

pandas.apply(func)：用于一维数据Series遍历元素

pandas.applymap(func)：用于二维及多位数据DataFrame遍历元素

DataFrame中用apply则是调用一次调用一整列作为参数，再返回一整列，或者一个单值

### 给数据按照百分比分段

比如按照百分比给学生分配成绩，pandas方法如下：

pd.qcut(exam\_grades,[0, 0.1, 0.2, 0.5, 0.8, 1],labels=['F', 'D', 'C', 'B', 'A'])

### std函数的不同

默认标准偏差类型在 numpy 的 .std() 和 pandas 的 .std() 函数之间是不同的。默认情况下，numpy 计算的是总体标准偏差，ddof = 0。另一方面，pandas 计算的是样本标准偏差，ddof = 1。如果我们知道所有的分数，那么我们就有了总体——因此，要使用 pandas 进行归一化处理，我们需要将“ddof”设置为 0。

### Dataframe.groupby(‘ ‘)

可以实现数据根据某个字段分组

### 分组后获取某一行的值

data\_bygroup = subway\_data.groupby([‘latitude’,’longtitude’],as\_index = False).mean()

### merge合并DataFrame

data1.merge(data2,on = ‘account\_key’,how = ‘left’)

on的意思是根据’account\_key’字段来合并，left为以左侧表的字段建立总表字段，right为以右侧字段为主，inner为左右共同的字段，outter为所有字段

如果两个数据的表头名称不一致，可以用如下方法：

left\_on=[‘ ‘,’ ‘,],right\_on=[‘ ‘,’ ‘]

### Word2vec

将单词表示为数字很棒，但是神经网络依然需要学习单词的含义。我们用多个数字表示一个单词，而不是用一个数字呢？每个数字可以表示单词的抽象特性。近义词的值将很相近。这种表示方法叫做[**单词嵌入**](https://en.wikipedia.org/wiki/Word_embedding)。我们将多个单词和短语映射为向量。

[**Word2vec**](https://en.wikipedia.org/wiki/Word2vec)是创建此类单词嵌入最热门的方法之一。Word2vec 是一种神经网络模型，根据文字进行训练以创建单词嵌入。

Word2Vec 存在两种类型：Continuous bag-of-words (CBOW) 和 Skip grams。我们将重点介绍 Skip grams，但是涉及的很多内容也适用于 CBOW。Skip grams 模型是一种神经网络，传入一个单词并尝试预测 n 个周围的单词。

# 五、pandas系统学习

## 基础操作

### 读写数据

Pd.read\_csv(‘ ‘ )

写入数据

Pd.to\_csv(‘’,index = false) 参数表示不要索引了，即id列

### 查看数据

Pd.head()

Pd.info()

### 查看某一列

Pd.columns

### 删除某一列

Pd.drop(‘name’,axis=1)

删除名字为name的列

### 依据某一列而排序

Pd.sort\_values(‘names’,ascending=False) 降序排列

### 对某一列进行画图展示

Df[‘births’].plot()

## 统计分析

### 查看统计数据

Pd.describe()

### 某列的唯一值

Pd[‘name’].unique()

### 根据某列的值分组

Name = dataframe.groupby(‘name’)

Name.size() 分组的大小

Name.count() 根据分组的计数数据

Name.sum() 根据分组求和

# 六、matplotlib

两个常用句子：

%matplotlib inline

%config InlineBackend.figure\_format = 'retina'

意思是，内联显示，以及高清模式

# 七、tensorflow教程

## 1、简介

API变化很快，经常查官网文档

tf.contrib.learn:简单接口，可以很方便得从一维线性转换过来，比如scikit-learn

tf.contrib.slim 定义、训练、预测复杂模型的轻量级库。

高层次API： Keras,TFLearn,PrettyTensor

建议先从低级API用起，这样可以有更深的理解，以及更容易扩展自己写的网络。

## 2、设计网络与计算分开

简图------运算；

示例代码：

import tensorflow as tf

config = tf.ConfigProto()

config.gpu\_options.per\_process\_gpu\_memory\_fraction = 0.4

a = tf.constant(5)

b = tf.constant(6)

c = a + b

with tf.Session(config = config) as sess:

print(sess.run(c))

同时计算两张图：

x = 2

y = 3

op1 = tf.add(x,y)

op2 = tf.multiply(x,y)

useless = tf.multiply(x,op1)

op3 = tf.pow(op2,op1)

with tf.Session(config=config) as sess:

print(sess.run([op3,useless]))

TF可以实现分布计算，每个gpu计算图的一部分：

示例：

#建图：

with tf.device('/gpu:2'):

a = tf.constant([1,2,3,4,5,6],name='a')

b = tf.constant([1,2,3,4,5,6],name='b')

c = tf.matmul(a,b)

sess = tf.Session(config=tf.ConfigProto(log\_device\_placement=True))

print sess.run(c)

## 3、多图

如果想要同时建立多个图来计算独立任务，要小心每个session都会争取最大的GPU资源，而且每个图不可能相互传输数据。可以尝试在一个session中运行相互分离的子图。

多图示例：

#创建图

g = tf.Graph()

with g.as\_default():

a = tf.constant(3)

b = tf.constant(5)

x = tf.add(a,b)

#为指定的图创建计算会话--Session

with tf.Session(graph=g) as sess:

print(sess.run(x))

多图示例2：

g1 = tf.get\_default\_graph()

g2 = tf.Graph()

with g1.as\_default():

a = tf.Constant(3)

with g2.as\_default():

b = tf.Constant(5)

最后，还是不建议多图

## 4、为什么会用图来设计：

1.保存计算节点

2.把计算分为小块来帮助---自动差分？？

3.帮助实现分布式计算

## 5、TensorBoard

import tensorflow as tf

config = tf.ConfigProto()

config.gpu\_options.per\_process\_gpu\_memory\_fraction = 0.4

a = tf.constant(2)

b = tf.constant(3)

x = tf.add(a,b)

with tf.Session(config=config) as sess:

#add this line to use TensorBoard

writer = tf.summary.FileWriter('./graphs',sess.graph)

print(sess.run(x))

writer.close()

打开tensorBoard，在命令行进到项目目录，输入以下部分：

tensorboard --logdir="./graphs"

## 6、tf.constant()

tf**.**constant**(**value**,** dtype**=None,** shape**=None,** name**='Const',** verify\_shape**=False)**

为图提供固定值

Veryfh\_shape的意思是检查value的数量跟shape的形状是否相符

# Constant 1-D Tensor populated with value list.

tensor = tf.constant([1, 2, 3, 4, 5, 6, 7]) => [1 2 3 4 5 6 7]

# Constant 2-D tensor populated with scalar value -1.

tensor = tf.constant(-1.0, shape=[2, 3]) => [[-1. -1. -1.]

[-1. -1. -1.]]

## 7、tf.zeros()/tf.zeros\_like(input\_tensor)

tf**.**zeros**(**shape**,** dtype**=**tf**.**float32**,** name**=None)**

tf.zeros([3, 4], tf.int32) # [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

类似的还有1

Tf.ones()

Tf.ones\_like()

## 8\tf.fill(dims, value, name=None)

用值填充一个numpy，或者说矩阵；

fill([2, 3], 9) ==> [[9, 9, 9]

[9, 9, 9]]

# 八、python一些概念记录

## 1、关于复制

复制python中的list，可以用切片方式：qiepian = list[:]。这样相当于创造了副本，而不是在原图上做更改。

## 2、快捷创建列表

比较方便的创建列表的方式：

Lista = [value\*\*2 for value in range(10)]

## 3、字符串路径

python在windows系统下，\符号会被视为转义字符标记，最好的方式是在一个字符串的引号前加上r字符，例如：

with open(r"C:\Users\Parker\Desktop\desk.txt") as file\_ob:

contents = file\_ob.read()

print(contents)

## 4、python保存文件会出现多余空行

python处理文件时候，read()达到文件末尾时候回返回一个空字符串，这个打印下来就是一个空行，要删除末尾的空行，可对那个字符串使用rstrip(),该方法的效果是删除字符串末尾的空白：

with open('pi.txt') as file\_ob:

for line in file\_ob:

print(line.rstrip())

## 5、datetimp的实参

模块 datetime中，日期和时间格式的实参

|  |  |
| --- | --- |
| %A | 星期的名称，如Monday |
| %B | 月份名，如January |
| %m | 数字表示的月份（1-12） |
| %d | 数字表示月份中的一天（1-31） |
| %Y | 四位的年份，如2018 |
| %y | 两位的年份，如18 |
| %H | 24小时制的小时（0-23） |
| %h | 12小时制的小时（1-12） |
| %p | Am或者pm |
| %M | 分钟（0-59） |
| %S | 秒数（0-61） |

## 6、Python对于列表的del, remove, pop操作的区别

首先，remove 是删除首个符合条件的元素。并不是删除特定的索引。如下例：

1. >>> a = [0, 2, 2, 3]
2. >>> a.remove(2)
3. >>> a
4. [0, 2, 3]

而对于 del 来说，它是根据索引（元素所在位置）来删除的，如下例：

1. >>> a = [3, 2, 2, 1]
2. >>> del a[1]
3. [3, 2, 1]

        第1个元素为a[0] －－是以0开始计数的。则a[1]是指第2个元素，即里面的值2.

        del还可以删除指定范围内的值

        a = [3,2,2,1]

        del a[1,3]

        print a

        结果[3]

        del还可以删除整个列表

        del a

pop返回的是你弹出的那个数值。

所以使用时要根据你的具体需求选用合适的方法

1. >>> a = [4, 3, 5]
2. >>> a.pop(1)
3. 3
4. >>> a
5. [4, 5]

## 7、pyghon中的zip作用

zip([1,2,3],['a','b','c'])

结果是

[(1, 'a'), (2, 'b'), (3, 'c')]

就是依次取出每一个数组的元素，然后组合

操作数可以更多

zip([1,2,3],['a','b','c'],[4,5,6])也是可以的

如果元素长度不一致会被砍到一样长

另外zip(\*list)也就是数组前面带个星号，是上述操作的逆操作

zip(\*[(1, 'a'), (2, 'b'), (3, 'c')])的结果是[1,2,3],['a','b','c']

同理，[ ]是把一个东西变为list，[\* ] ，则是拆开；

\*号在上述两个函数中的作用就是，逆函数

## 8、统计画图的经典代码

def draw\_line(x\_data,y\_data,title,y\_legend):

xy\_map = []

for x,y in groupby(sorted(zip(x\_data,y\_data)), key=lambda \_:\_[0]):

y\_list = [v for \_,v in y]

xy\_map.append([x,sum(y\_list)/len(y\_list)])

x\_unique,y\_mean = [\*zip(\*xy\_map)]

line\_chart = pygal.Line()

line\_chart.title = title

line\_chart.x\_labels = x\_unique

line\_chart.add(y\_legend,y\_mean)

line\_chart.render\_to\_file(title + '.svg')

return line\_chart

## 9、python函数之可变参数：

def fn(\*args):

print(args)

函数内形参有\*符号，表示可以传入任意数量的参数，函数内部将args当作一个tuple使用即可。

## 10、batches代码

import math

def batches(batch\_size, features, labels):

"""

Create batches of features and labels

:param batch\_size: The batch size

:param features: List of features

:param labels: List of labels

:return: Batches of (Features, Labels)

"""

assert len(features) == len(labels)

# TODO: Implement batching

output\_batches = []

data\_len = len(features)

for start\_i in range(0,data\_len,batch\_size):

end\_i = start\_i + batch\_size

batch = [features[start\_i:end\_i],labels[start\_i:end\_i]]

output\_batches.append(batch)

return output\_batches

## 11、切片

a = [0,1,2,3,4,5,6,7,8,9]

b = a[i:j] 表示复制a[i]到a[j-1]，以生成新的list对象

b = a[1:3] 那么，b的内容是 [1,2]

当i缺省时，默认为0，即 a[:3]相当于 a[0:3]

当j缺省时，默认为len(alist), 即a[1:]相当于a[1:10]

当i,j都缺省时，a[:]就相当于完整复制一份a了

b = a[i:j:s]这种格式呢，i,j与上面的一样，但s表示步进，缺省为1.

所以a[i:j:1]相当于a[i:j]

当s<0时，i缺省时，默认为-1. j缺省时，默认为-len(a)-1

所以a[::-1]相当于 a[-1:-len(a)-1:-1]，也就是从最后一个元素到第一个元素复制一遍。所以你看到一个倒序的东东。

如果还不理解，把我说的东西测试一遍，你就明白了

## 13、加载名字相似的文件名

images = glob.glob('calibration\_wide/GOPR\*.jpg')

得到一个list

14、array.array参数

array.array(‘d’,(random() for i in range(10)))

会生成10个随机浮点数据，这是一个结构紧凑的数组

参数说明

Type code C Type Minimum size in bytes

'b' signed integer 1

'B' unsigned integer 1

'u' Unicode character 2 (see note)

'h' signed integer 2

'H' unsigned integer 2

'i' signed integer 2

'I' unsigned integer 2

'l' signed integer 4

'L' unsigned integer 4

'q' signed integer 8 (see note)

'Q' unsigned integer 8 (see note)

'f' floating point 4

'd' floating point 8

# 九、迁移学习

从头训练一个网络要经历设计、训练、调整这些步骤，会比较慢，而迁移学习可以很快达到一个比较好的目标。根据新数据量及与原始数据的相似程度，迁移学习面临4种情况。

1、新数据集小，与原始数据分布较像；

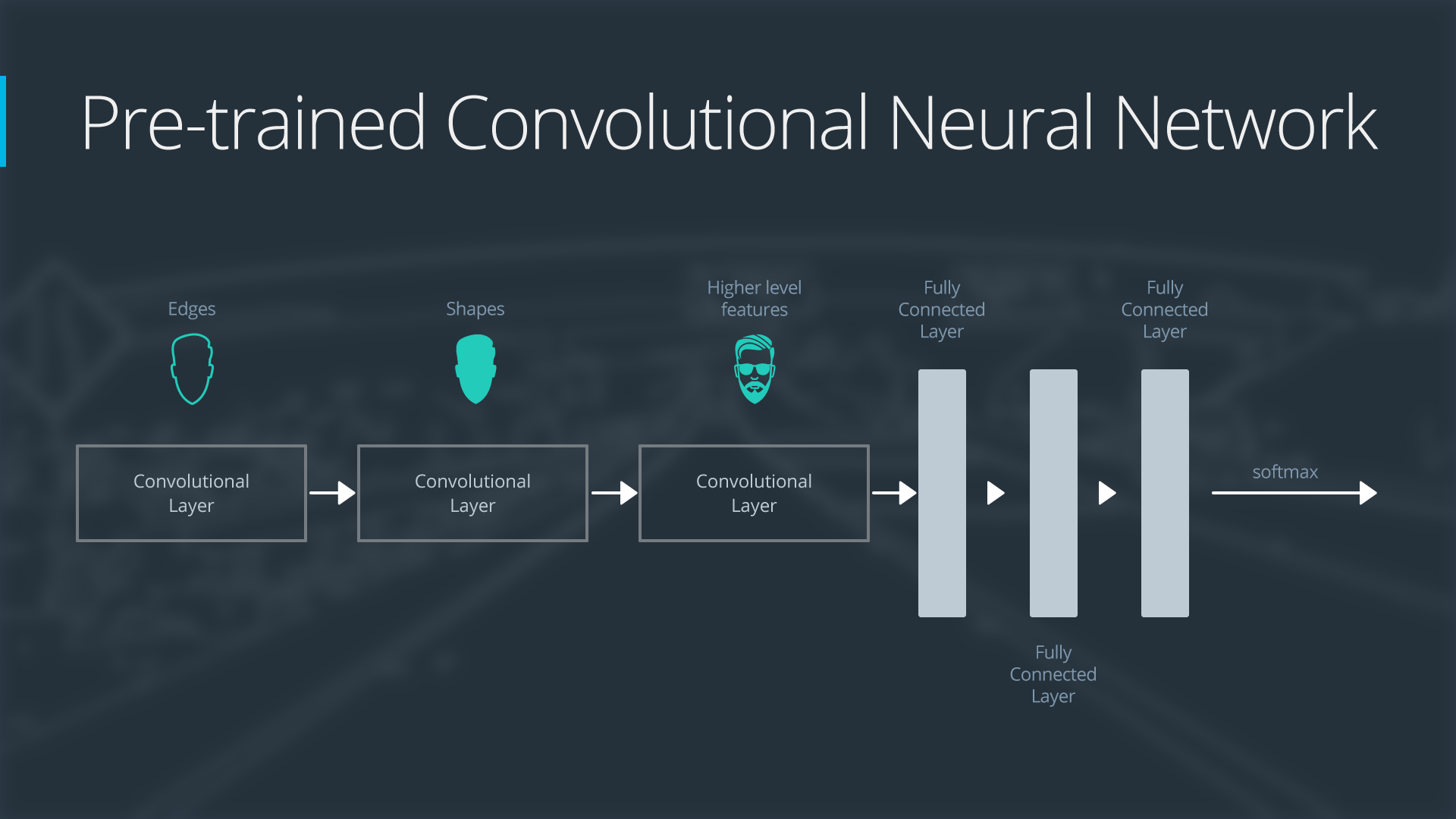
2、数据量小，差异大

3、数据量大，差异小

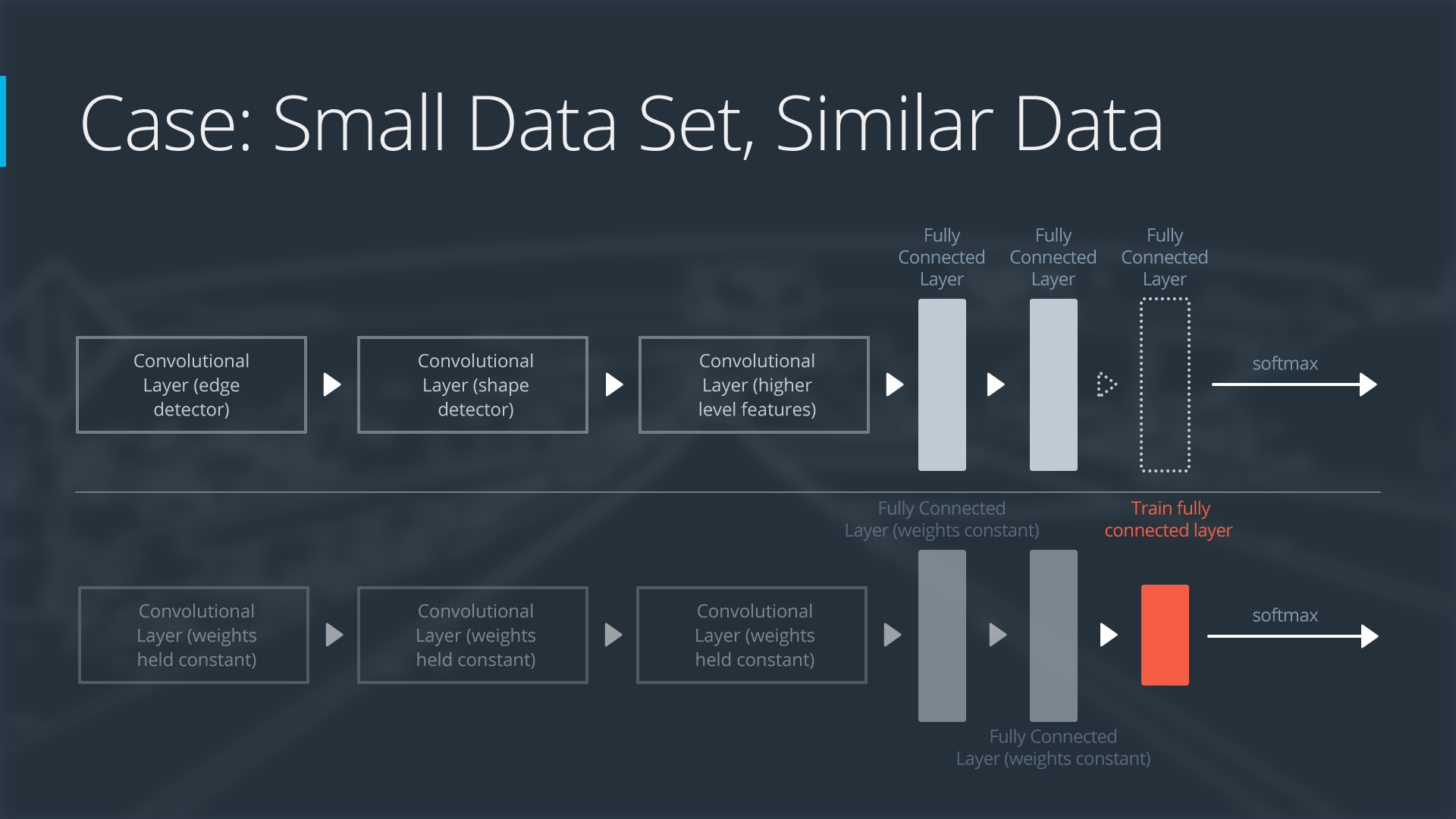
4、数据量大，差异大

下面是yodua的思路：

原始网络的样子：



## 1、small-similar

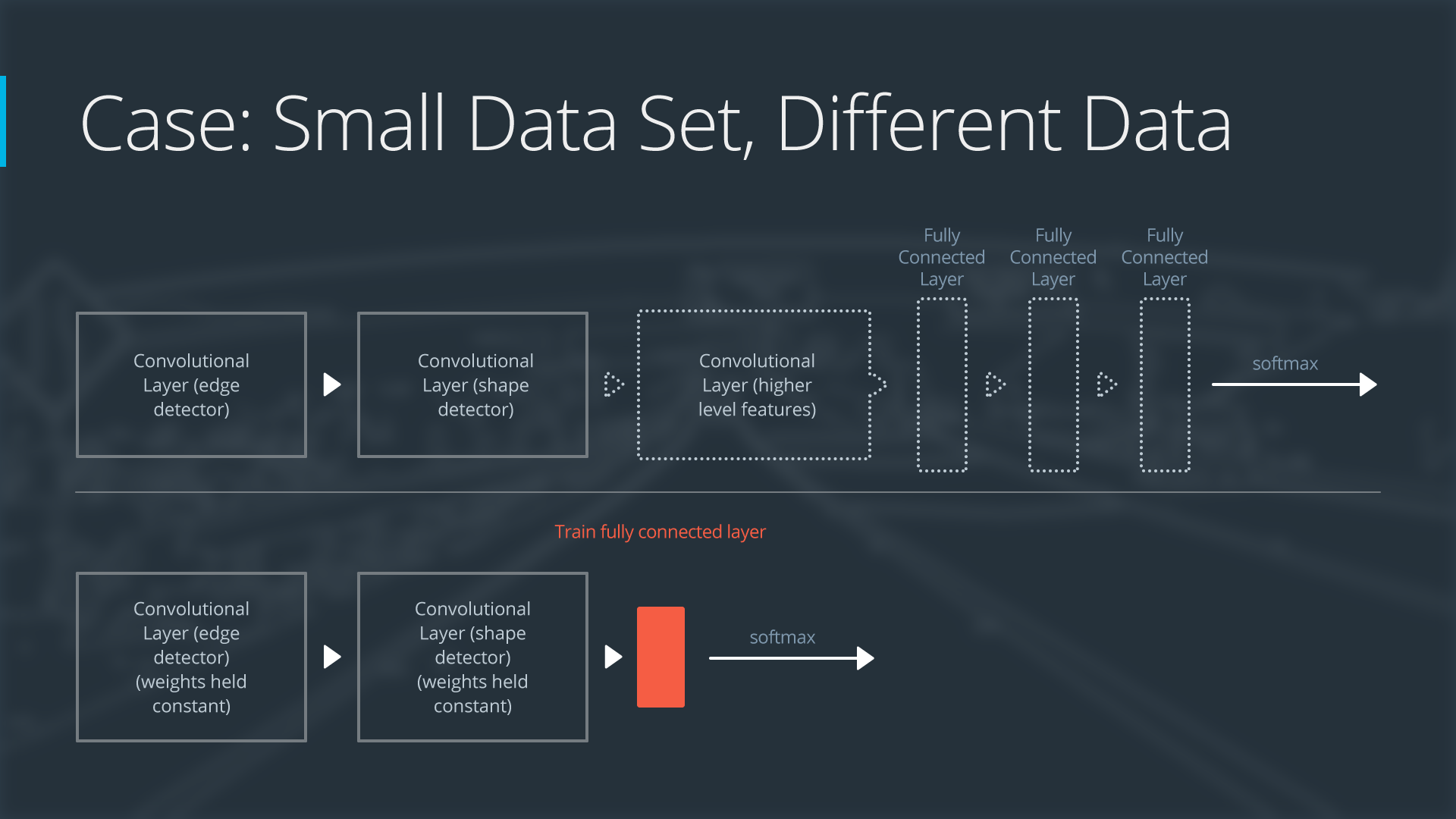


替换掉原始网络的最后一层，并冻结前面的参数，只训练最后一层，原因如下：

为避免小数据集过度拟合，原始网络的权重将保持不变，而不是重新训练权重。

由于数据集相似，每个数据集的图像将具有相似的更高级别的特征。因此，大多数或所有预先训练的神经网络图层都已经包含了有关新数据集的相关信息，因此应予以保留。

## 2、Small-Different



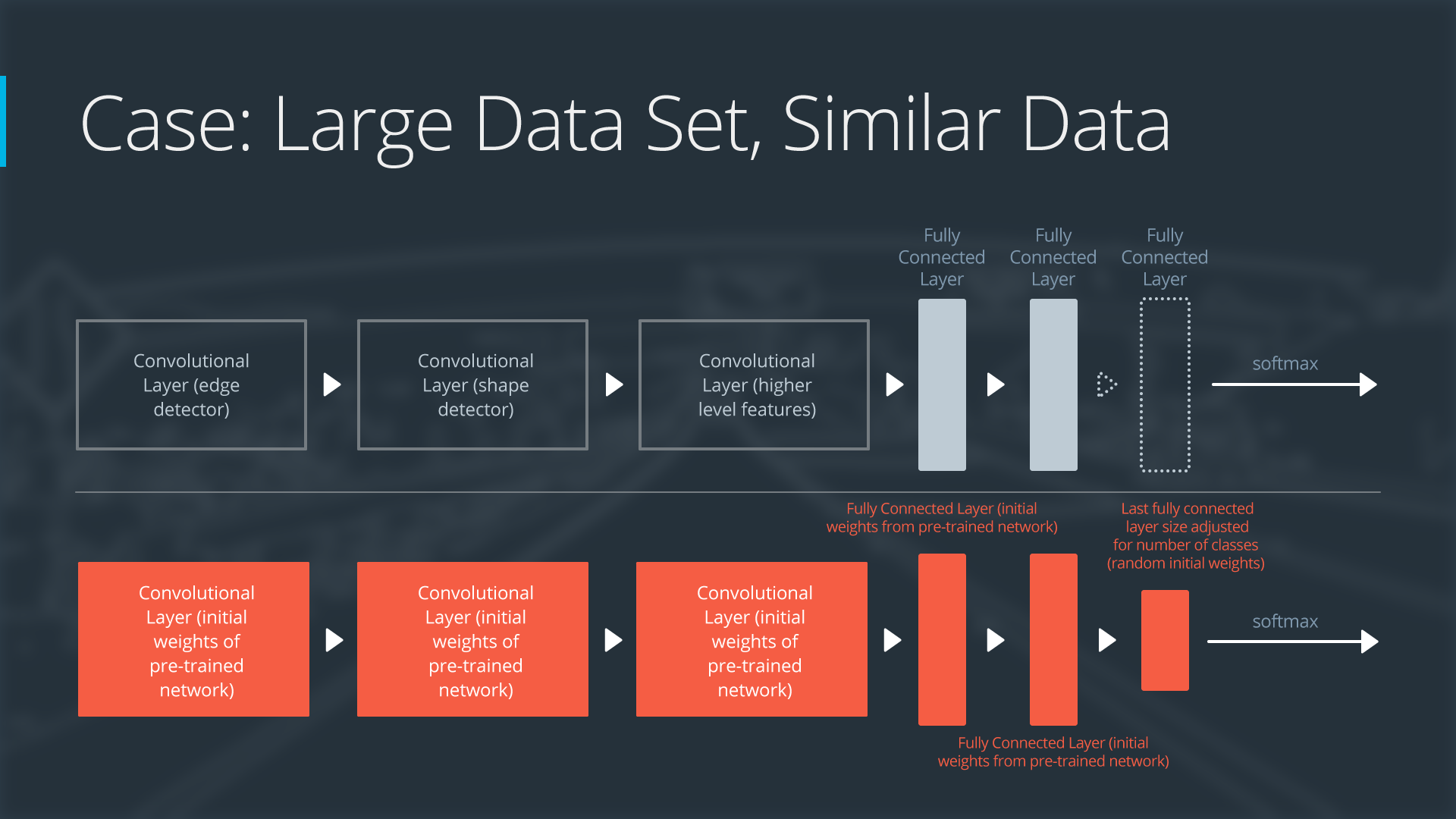
如果新数据集较小并且与原始训练数据不同：

* 切断网络开始附近的大部分预先训练的层
* 向剩余的预先训练的图层添加一个与新数据集中的类数相匹配的全新连接图层
* 随机化新的完全连接层的权重; 冻结来自预先训练的网络的所有权重
* 训练网络以更新新的完全连接层的权重

由于数据集很小，过度拟合仍然是一个问题。为了防止过拟合，原始神经网络的权重将保持不变，就像第一种情况一样。

但原始训练集和新数据集不共享更高级别的功能。在这种情况下，新网络将只使用包含较低级别功能的图层。

## 3、Large-Similar



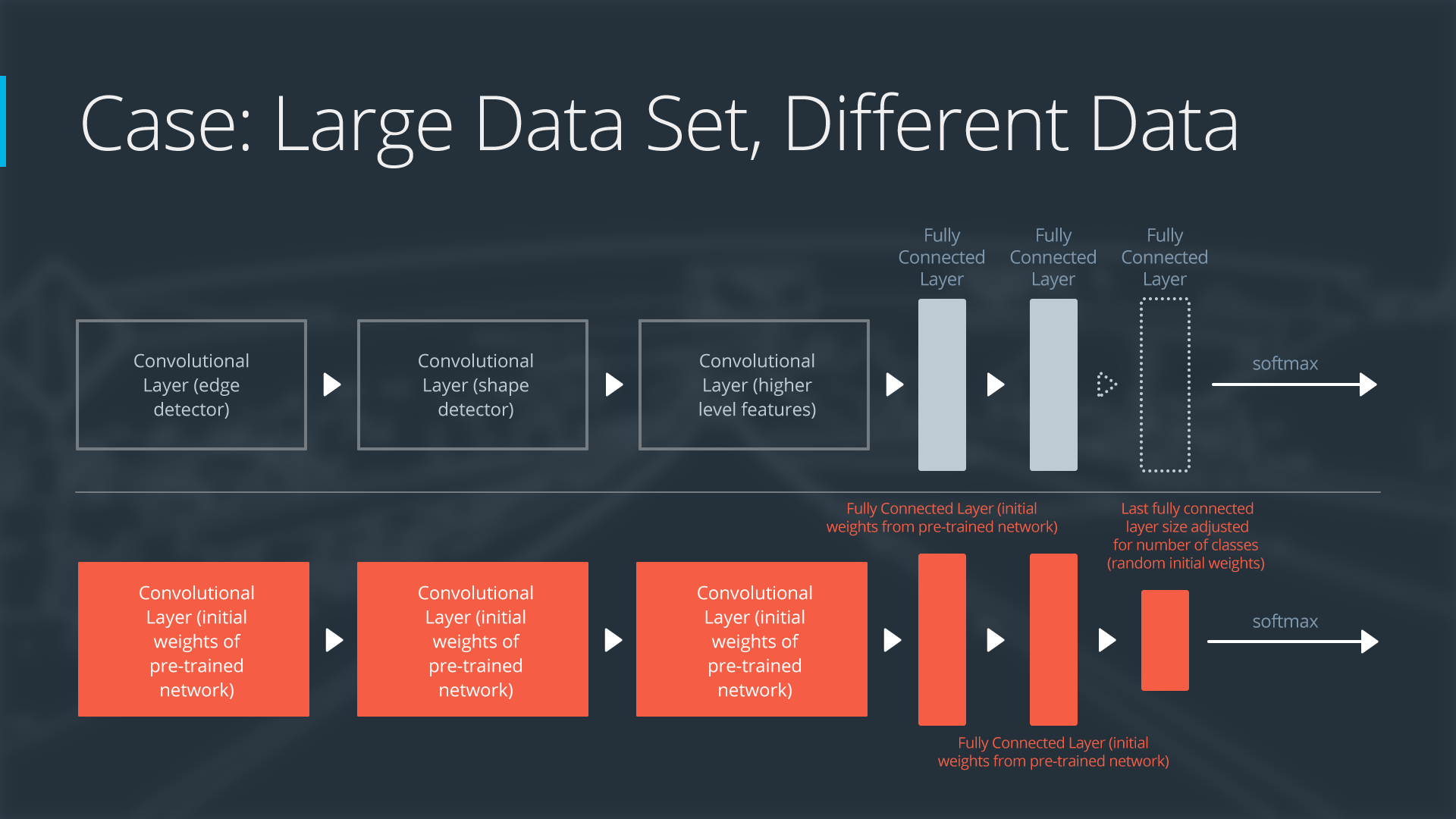
如果新数据集很大并且与原始训练数据相似：

* 删除最后一个完全连接的图层，并替换为与新数据集中的类数匹配的图层
* 随机初始化新完全连接层中的权重
* 使用预先训练的权重初始化其余权重
* 重新训练整个神经网络

在对大型数据集进行培训时，过度拟合并不是那么重要; 因此，你可以重新训练所有的重量。

由于原始训练集和新数据集共享更高级别的特征，因此也使用整个神经网络。

## 4、Large-Different



如果新数据集很大并且与原始训练数据不同：

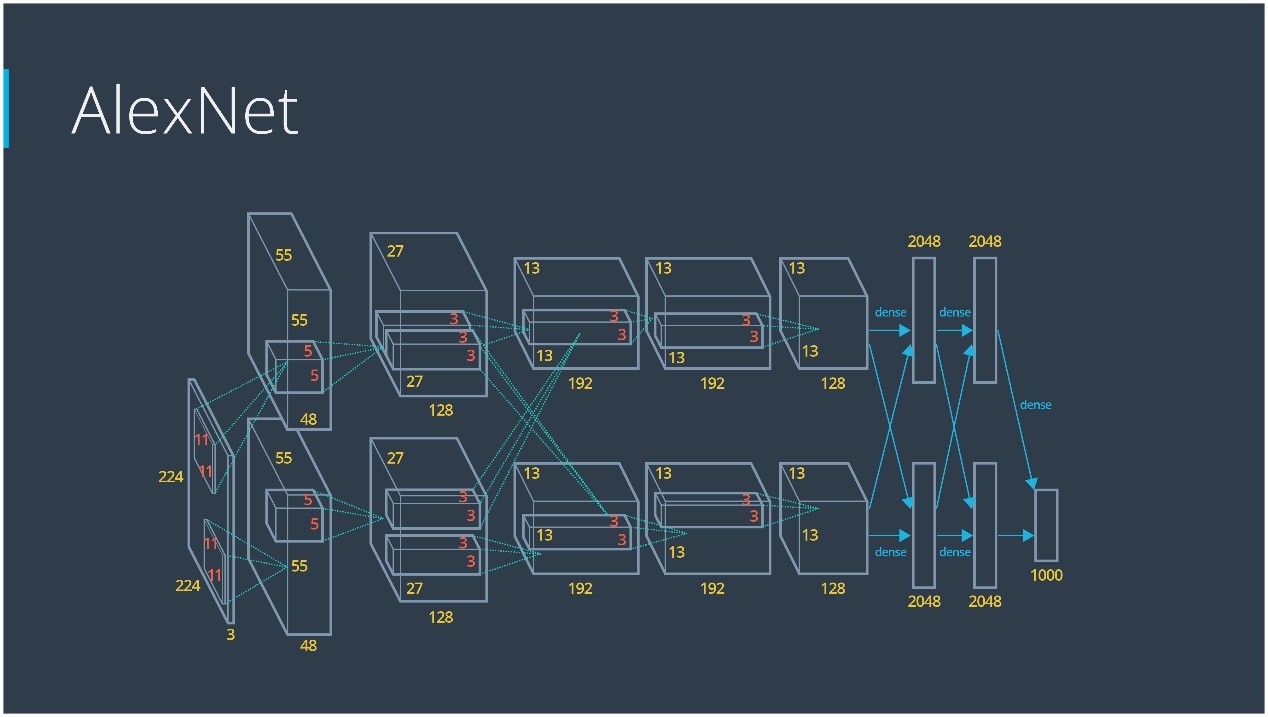
* 删除最后一个完全连接的图层，并替换为与新数据集中的类数匹配的图层
* 从头开始用随机初始化权重重新训练网络
* 或者，您可以使用与“大而相似”的数据案例相同的策略

即使数据集与训练数据不同，从预先训练的网络初始化权重可能会使训练速度更快。所以这种情况与大量类似数据集的情况完全相同。

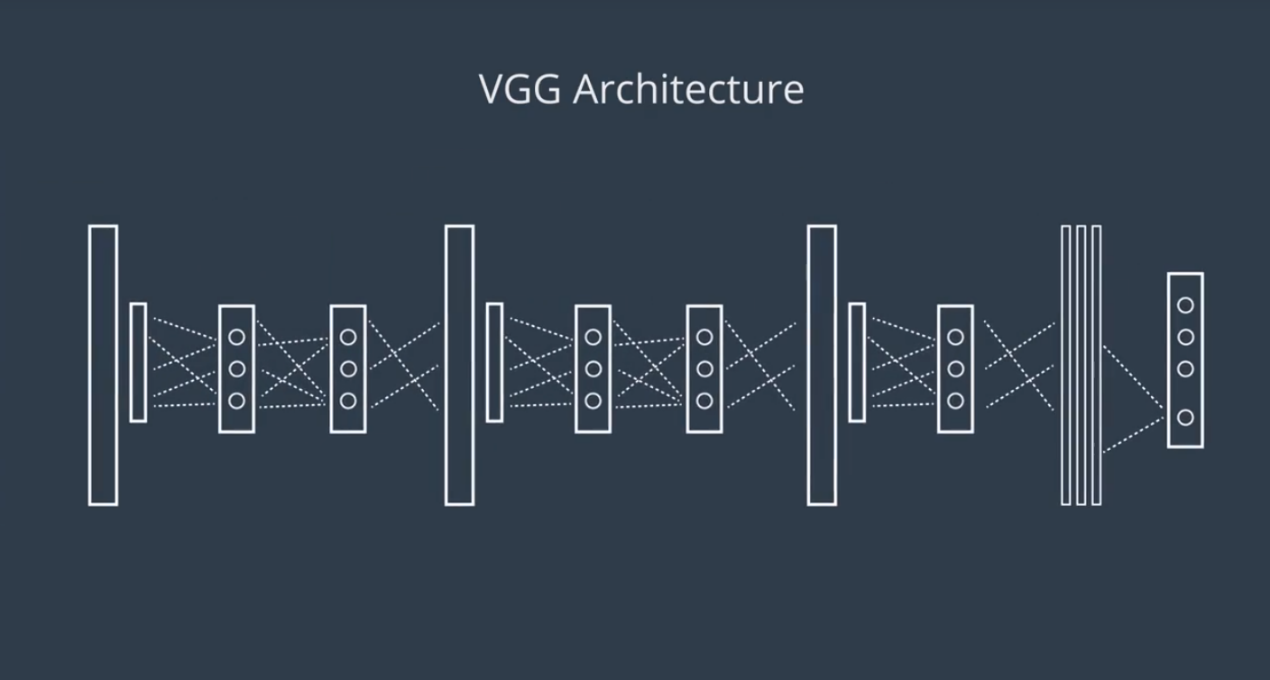
如果使用预先训练的网络作为起点并不能产生成功的模型，另一种选择是随机初始化卷积神经网络权重并从头开始训练网络。

# 十、比较著名的网络

## 1、Alexnet



## 2、VGGNet



组成：3\*3的卷积层，2\*2的池化层

简洁而优雅，可以作为图像分类任务的起始网络

灵活性是其最大的优势之一

## 3、GoogleNet

十一、SelfCar克隆项目审阅者给我的课外资料

1、python代码的注释

Python官网

<https://www.python.org/dev/peps/pep-0257/>

Google风格的python注释

<https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html>

Google guide

<https://google.github.io/styleguide/pyguide.html>

2、激活函数

<https://cs231n.github.io/neural-networks-1/#actfun>

3、卷积网络的设计---来自CS231N

<https://www.youtube.com/watch?v=u6aEYuemt0M>

4、Nuts and Bolts of Applying Deep Learning (Andrew Ng)

<https://www.youtube.com/watch?v=F1ka6a13S9I>

5、关于P3的课外资料

<https://medium.com/udacity/how-udacitys-self-driving-car-students-approach-behavioral-cloning-5ffbfd2979e5>

<https://medium.com/@xslittlegrass/self-driving-car-in-a-simulator-with-a-tiny-neural-network-13d33b871234>

# 十一、关于成像畸变

## 1、畸变种类

radial distortion，径向畸变，光线在通过镜头（一般是凸透镜）时，边缘处光线折射率不一致，导致成像后边缘处形变。

tangential distortion，切向畸变，镜头组与成像平面不完全平行造成的畸变。

Perspective distortion, 透视畸变，物体离观察点越远，看起来就越小；平行的直线在最远处似乎汇聚在一个点上。

## 2、纠正畸变的代码

### 1、纠正径向畸变和切向畸变

import numpy as np

import cv2

import glob

import matplotlib.pyplot as plt

# %matplotlib qt

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)

objp = np.zeros((6\*8,3), np.float32)

objp[:,:2] = np.mgrid[0:8, 0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.

objpoints = [] # 3d points in real world space

imgpoints = [] # 2d points in image plane.

# Make a list of calibration images

images = glob.glob('calibration\_wide/GO\*.jpg')

# Step through the list and search for chessboard corners

for idx, fname in enumerate(images):

img = cv2.imread(fname)

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Find the chessboard corners

ret, corners = cv2.findChessboardCorners(gray, (8,6), None)

# If found, add object points, image points

if ret == True:

objpoints.append(objp)

imgpoints.append(corners)

# Draw and display the corners

cv2.drawChessboardCorners(img, (8,6), corners, ret)

#write\_name = 'corners\_found'+str(idx)+'.jpg'

#cv2.imwrite(write\_name, img)

cv2.imshow('img', img)

cv2.waitKey(500)

cv2.destroyAllWindows()

import pickle

%matplotlib inline

# Test undistortion on an image

img = cv2.imread('calibration\_wide/test\_image.jpg')

img\_size = (img.shape[1], img.shape[0])

# Do camera calibration given object points and image points

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img\_size,None,None)

dst = cv2.undistort(img, mtx, dist, None, mtx)

cv2.imwrite('calibration\_wide/test\_undist.jpg',dst)

# Save the camera calibration result for later use (we won't worry about rvecs / tvecs)

dist\_pickle = {}

dist\_pickle["mtx"] = mtx

dist\_pickle["dist"] = dist

pickle.dump( dist\_pickle, open( "calibration\_wide/wide\_dist\_pickle.p", "wb" ) )

#dst = cv2.cvtColor(dst, cv2.COLOR\_BGR2RGB)

# Visualize undistortion

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))

ax1.imshow(img)

ax1.set\_title('Original Image', fontsize=30)

ax2.imshow(dst)

ax2.set\_title('Undistorted Image', fontsize=30)

### 2、Perspective transform

透视变换和纠正径向畸变及切向畸变思路相似，不同之处在与，纠正径向畸变和切向畸变是在相同视角下纠正，而透视变化是把原始图像点映射到一个新的观察视角下的图像上的对应点。

import pickle

import cv2

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

# Read in the saved camera matrix and distortion coefficients

# These are the arrays you calculated using cv2.calibrateCamera()

dist\_pickle = pickle.load( open("calibration\_wide/wide\_dist\_pickle.p", "rb" ) )

mtx = dist\_pickle["mtx"]

dist = dist\_pickle["dist"]

# Read in an image

img = cv2.imread(images[0])

nx = 8 # the number of inside corners in x

ny = 6 # the number of inside corners in y

# MODIFY THIS FUNCTION TO GENERATE OUTPUT

# THAT LOOKS LIKE THE IMAGE ABOVE

def corners\_unwarp(img, nx, ny, mtx, dist):

# Pass in your image into this function

# Write code to do the following steps

# 1) Undistort using mtx and dist

undist = cv2.undistort(img,mtx,dist, None, mtx)

# 2) Convert to grayscale

gray = cv2.cvtColor(undist, cv2.COLOR\_BGR2GRAY)

# 3) Find the chessboard corners

ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)

# 4) If corners found:

# a) draw corners

# b) define 4 source points src = np.float32([[,],[,],[,],[,]])

#Note: you could pick any four of the detected corners

# as long as those four corners define a rectangle

#One especially smart way to do this would be to use four well-chosen

# corners that were automatically detected during the undistortion steps

#We recommend using the automatic detection of corners in your code

# c) define 4 destination points dst = np.float32([[,],[,],[,],[,]])

# d) use cv2.getPerspectiveTransform() to get M, the transform matrix

# e) use cv2.warpPerspective() to warp your image to a top-down view

#delete the next two lines

if ret == True:

cv2.drawChessboardCorners(undist, (nx, ny), corners, ret)

offset = 100

img\_size = (gray.shape[1], gray.shape[0])

src = np.float32([corners[0],corners[nx-1],corners[-1], corners[-nx]])

dst = np.float32([[offset,offset], [img\_size[0]-offset,offset], [img\_size[0]-offset,img\_size[1]-offset], [offset, img\_size[1]-offset]])

M = cv2.getPerspectiveTransform(src, dst)

warped = cv2.warpPerspective(undist, M, img\_size)

return warped, M

top\_down, perspective\_M = corners\_unwarp(img, nx, ny, mtx, dist)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(img)

ax1.set\_title('Original Image', fontsize=50)

ax2.imshow(top\_down)

ax2.set\_title('Undistorted and Warped Image', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

## 3、相机标定 Camera Calibration

在图像测量过程以及机器视觉应用中，为确定空间物体表面某点的三维几何位置与其在图像中对应点之间的相互关系，必须建立相机成像的几何模型，这些几何模型参数就是相机参数。在大多数条件下这些参数必须通过实验与计算才能得到，这个求解参数的过程就称之为相机标定（或摄像机标定）。无论是在图像测量或者机器视觉应用中，相机参数的标定都是非常关键的环节，其标定结果的精度及算法的稳定性直接影响相机工作产生结果的准确性。因此，做好相机标定是做好后续工作的前提，提高标定精度是科研工作的重点所在。

# 十二、颜色空间

1、HSV

Hue 色相

Saturation 饱和度

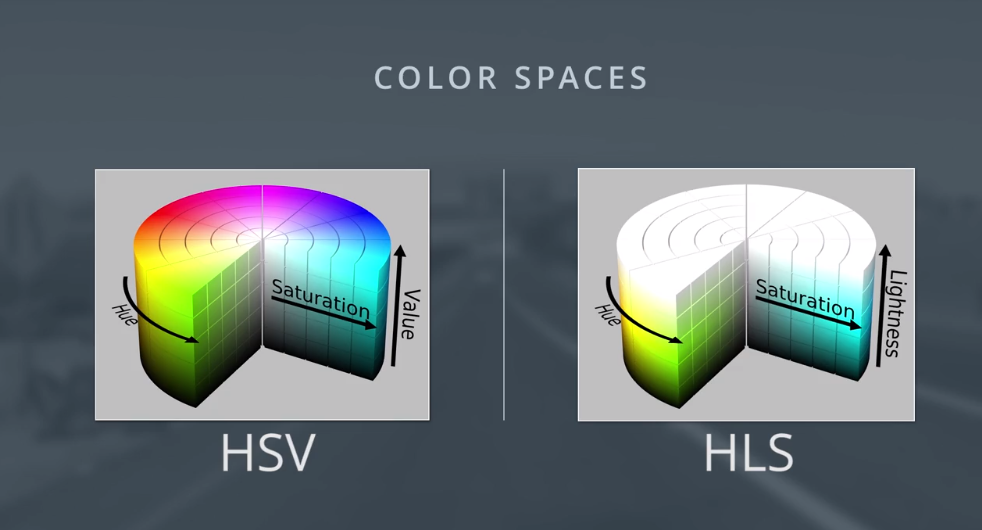
Value 颜色值

2、HLS

Hue 色相

Lightness 亮度

Saturation 饱和度



车道线检测的步骤

