Introduction to TensorFlow2

20210413

problem: Installed package won't import in notebook

Solution :

1. Identify the execution environment and path issues under python and jupyter notebook

```
>>> import sys
>>> sys.executable
'C:\\Users\\aisndgo\\AppData\\Local\\conda\\conda\\envs\\t\\python.exe'
>>>sys.path
['',
 'C:\\Users\\aisndgo\\AppData\\Local\\conda\\conda\\envs\\t\\python35.zip',
 'C:\\Users\\aisndgo\\AppData\\Local\\conda\\conda\\envs\\t\\DLLs',
 'C:\\Users\\aisndgo\\AppData\\Local\\conda\\conda\\envs\\t\\lib',
 'C:\\Users\\aisndgo\\AppData\\Local\\conda\\conda\\envs\\t',
 'C:\\Users\\aisndgo\\AppData\\Local\\conda\\conda\\envs\\t\\lib\\site-packages']
```

```
In [4]: import sys
        sys.executable
Out[4]: 'C:\\ProgramData\\Anaconda3\\python.exe'

In [5]: sys.path
Out[5]: ['',
         'c:\\ncku\\test',
         'C:\\ProgramData\\Anaconda3\\python37.zip',
         'C:\\ProgramData\\Anaconda3\\DLLs',
         'C:\\ProgramData\\Anaconda3\\lib',
         'C:\\ProgramData\\Anaconda3',
         'C:\\Users\\aisndgo\\AppData\\Roaming\\Python\\Python37\\site-packages',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32\\lib',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\Pythonwin',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
         'C:\\Users\\aisndgo\\.ipython']
```

2.<refence 1:> http://takluyver.github.io/posts/i-cant-import-it.html

```
系統管理員: 命令提示字元
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation. 著作權所有，並保留一切權利。

C:\WINDOWS\system32>cd C:\ProgramData\Anaconda3

C:\ProgramData\Anaconda3>python -m pip install tensorflow
Collecting tensorflow
  Downloading https://files.pythonhosted.org/packages/7b/14/e4538c2bc3ae9f4ce6f6ce7ef1180da05ab
/tensorflow-1.13.1-cp37-cp37m-win_amd64.whl (63.1MB)
    100% |████████████████████████████████| 63.1MB 429kB/s
Collecting tensorboard<1.14.0,>=1.13.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/0f/39/bdd75b08a6fba41f098b6cb091b9e8c7a80
/tensorboard-1.13.1-py3-none-any.whl (3.2MB)
    100% |████████████████████████████████| 3.2MB 4.0MB/s
Collecting termcolor>=1.1.0 (from tensorflow)
Collecting astor>=0.6.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/35/6b/11530768cac581a12952a2aad00e1526b8
f/astor-0.7.1-py2.py3-none-any.whl
Requirement already satisfied: wheel>=0.26 in c:\programdata\anaconda3\lib\site-packages (from
Requirement already satisfied: six>=1.10.0 in c:\programdata\anaconda3\lib\site-packages (from
Requirement already satisfied: numpy>=1.13.3 in c:\programdata\anaconda3\lib\site-packages (fro
Collecting protobuf>=3.6.1 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/34/ef/f020691889031a8e1d8cb20711daa43cfe9
/protobuf-3.7.1-cp37-cp37m-win_amd64.whl (986kB)
    100% |████████████████████████████████| 993kB 5.7MB/s
Collecting gast>=0.2.0 (from tensorflow)
```

2.<refence 2:> **Trouble with TensorFlow in Jupyter Notebook**

To use tensorflow with Ipython and/or Jupyter notebook, simply install them into the tensorflow environment:

```
source activate tensorflow #activate tensorflow env
conda install ipython
conda install jupyter
jupyter notebook #open jupyter notebook
```

```
import sys
sys.executable

'C:\\Users\\aisndgo\\AppData\\Local\\conda\\conda\\envs\\t\\python.exe'
```

Install TensorFlow on Windows

https://www.tensorflow.org/install/install_windows

```
(ncku) C:\Users\FUDEV>conda install tensorflow
Fetching package metadata ...........
Solving package specifications: ..........

Package plan for installation in environment C:\Users\FUDEV\Anaconda3\envs\ncku:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    libprotobuf-3.2.0          |           vc14_0         9.1 MB
    markdown-2.6.9             |           py35_0         101 KB
    werkzeug-0.12.2            |           py35_0         439 KB
    backports.weakref-1.0rc1   |           py35_0           8 KB
    protobuf-3.2.0             |           py35_0         463 KB
    tensorflow-1.2.1           |           py35_0        21.0 MB

                                           Total:        31.1 MB

The following NEW packages will be INSTALLED:

    backports:         1.0-py35_0
    backports.weakref: 1.0rc1-py35_0
    libprotobuf:       3.2.0-vc14_0  [vc14]
    markdown:          2.6.9-py35_0
    protobuf:          3.2.0-py35_0
    tensorflow:        1.2.1-py35_0
    werkzeug:          0.12.2-py35_0

Proceed ([y]/n)? y

Fetching packages ...
libprotobuf-3. 100% |###############################| Time: 0:00:00  10.91 MB/s
```

```
(ncku) C:\Users\FUDEV>conda install tensorflow-gpu
Fetching package metadata ...........
Solving package specifications: ..........

Package plan for installation in environment C:\Users\FUDEV\Anaconda3\envs\ncku:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    cudatoolkit-8.0            |                3       319.9 MB
    cudnn-6.0                  |                0        95.1 MB
    numpy-1.12.1               |           py35_0         3.5 MB
    tensorflow-gpu-1.1.0       |      np112py35_0        41.7 MB
    matplotlib-2.0.2           |      np112py35_0         8.5 MB
    ------------------------------------------------------------
                                           Total:       468.7 MB

The following NEW packages will be INSTALLED:

    cudatoolkit:    8.0-3
    cudnn:          6.0-0
    tensorflow-gpu: 1.1.0-np112py35_0

The following packages will be UPDATED:

    matplotlib:     2.0.2-np113py35_0 --> 2.0.2-np112py35_0

The following packages will be DOWNGRADED due to dependency conflicts:

    numpy:          1.13.1-py35_0     --> 1.12.1-py35_0

Proceed ([y]/n)?
```

```
(ncku) C:\Users\aisndgo>conda install -c conda-forge tensorflow

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    tensorboard-1.5.1          |           py36_1         3.0 MB  conda-forge
    tensorflow-1.5.0           |           py36_0        27.1 MB  conda-forge
    absl-py-0.1.10             |             py_0          72 KB  conda-forge
    libprotobuf-3.5.2          |           vc14_0        10.2 MB  conda-forge
    protobuf-3.5.2             |      py36_vc14_0         513 KB  conda-forge
    ------------------------------------------------------------
                                           Total:        40.9 MB

The following NEW packages will be INSTALLED:

    absl-py:     0.1.10-py_0          conda-forge
    tensorboard: 1.5.1-py36_1         conda-forge

The following packages will be UPDATED:

    certifi:     2018.1.18-py36_0            --> 2018.1.18-py36_0  conda-forge
    libprotobuf: 3.5.2-he0781b1_0            --> 3.5.2-vc14_0      conda-forge [vc14]
    protobuf:    3.5.2-py36h6538335_0        --> 3.5.2-py36_vc14_0 conda-forge [vc14]
    tensorflow:  1.2.1-py36_0                --> 1.5.0-py36_0      conda-forge

Proceed ([y]/n)? y
```

Test:

- "native" pip
- Anaconda

2. Create a conda environment named `tensorflow` by invoking the following command:

```
C:> conda create -n tensorflow pip python=3.5
```

DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality

3. Activate the conda environment by issuing the following command:

```
C:> activate tensorflow
 (tensorflow)C:>  # Your prompt should change
```

4. Issue the appropriate command to install TensorFlow inside your conda environ
To install the CPU-only version of TensorFlow, enter the following command:

```
(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow
```

To install the GPU version of TensorFlow, enter the following command (on a single line):

```
(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow-gpu
```

```
(ncku) C:\Users\FUDEV>pip install --ignore-installed --upgrade tensorflow-gpu
Collecting tensorflow-gpu
  Downloading tensorflow_gpu-1.3.0-cp35-cp35m-win_amd64.whl (60.0MB)
    100% |████████████████████████████████| 60.0MB 19kB/s
```

```python
import tensorflow as tf
tf.__version__
```

'1.5.0'

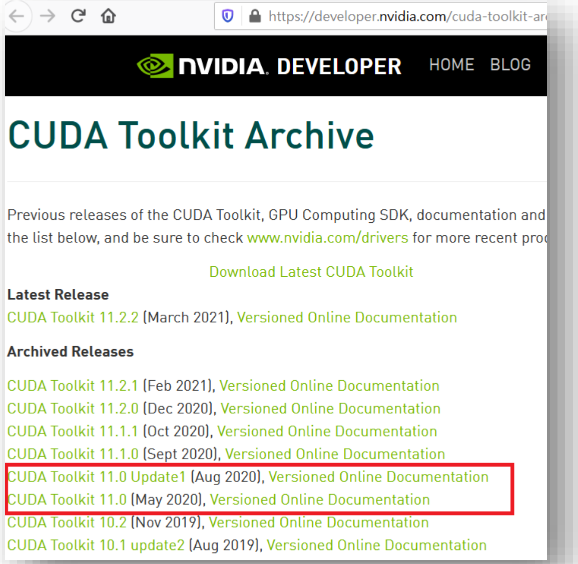
Install Tensorflow-GPU version with Jupyter (Windows 10)

1) Check if your GPU is supported

>>wmic path win32_VideoController get name

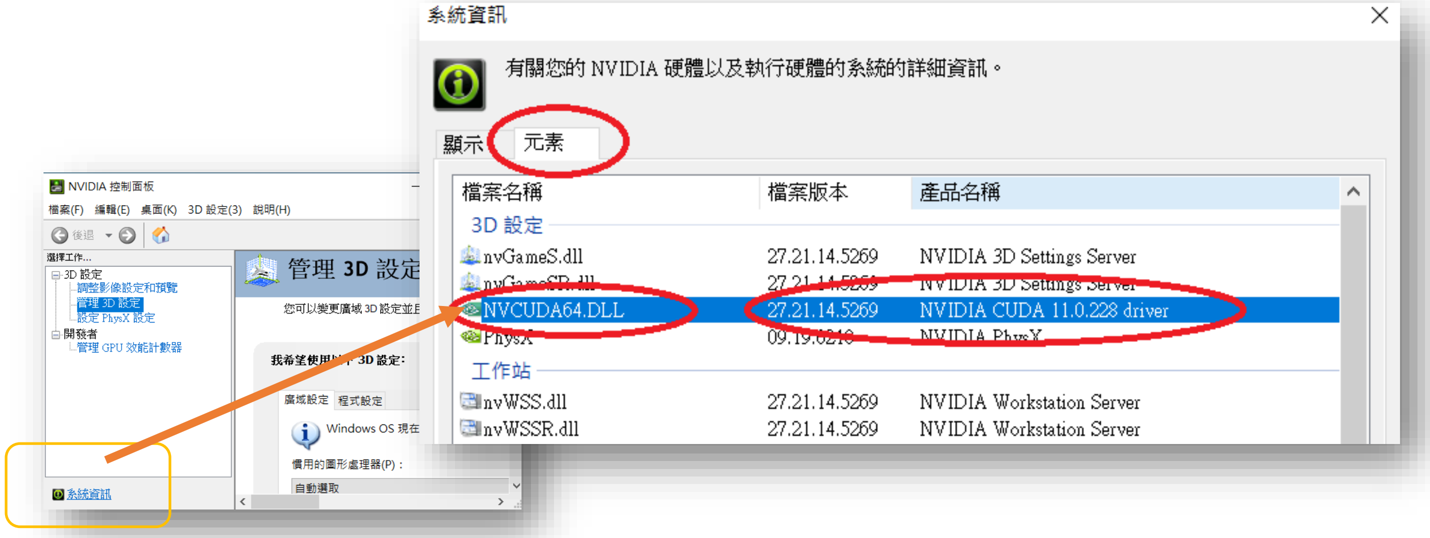
```
C:\Users\fu>wmic path win32_VideoController get name
Name
NVIDIA GeForce MX450
Intel(R) Iris(R) Xe Graphics
```

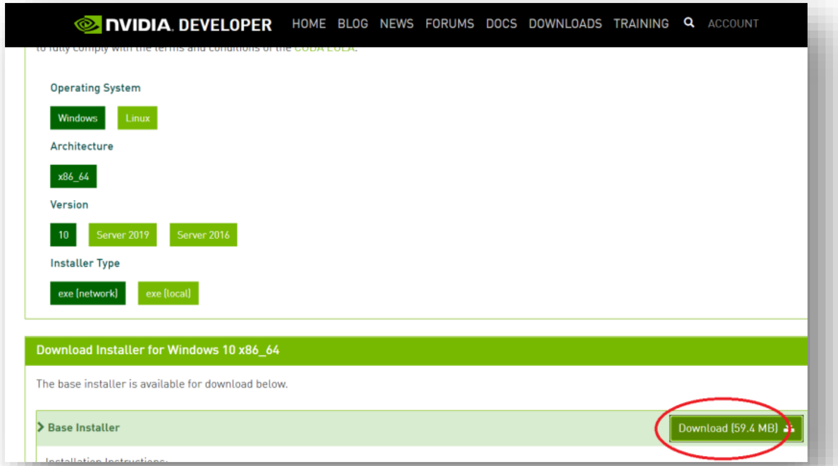
CUDA Toolkit Download the correct version



2) Install Anaconda

3)Set up your Nvidia GPU





Install Tensorflow-GPU version with Jupyter (Windows 10)

If VS is not installed, uncheck VS, and then continue to install in the next step

Download [cuDNN](#) from the website, you must register an account and agree to the terms to download, select the corresponding version and operating system

Install Tensorflow-GPU version with Jupyter (Windows 10)

Copy the following files into the CUDA Toolkit directory.

a. Copy `<installpath>\cuda\bin \cudnn*.dll` to `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v x.x \bin`.

b. Copy `<installpath>\cuda\include\cudnn*.h` to `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\include`.

c. Copy `<installpath>\cuda\lib\x64\cudnn*.lib` to `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\lib\x64`.

Input “nvidia-smi” command in the terminal to check whether the installation is successful, as shown in the figure below.

```
C:\Users\fu>nvidia-smi
Wed Mar 31 18:34:01 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 452.69       Driver Version: 452.69       CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce MX450       WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   52C    P8    N/A /  N/A |    119MiB /  2048MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

You can install TensorFlow using conda, but it is recommended to install Tensorflow using the ***pip.*** (Note you should [specify the version of python based on the version of TensorFlow]() you need)

Starting from TensorFlow 2.1, installing TensorFlow via pip also includes GPU support, and there is no need to install the GPU version via specific pip tensorflow-gpu. If you are sensitive to the file size of the pip installation, you can use tensorflow-cpu to install a version of TensorFlow that only supports CPU.

Windows setup

- Add the CUDA®, CUPTI, and cuDNN installation directories to the %PATH% environmental variable. For example, if the CUDA® Toolkit is installed to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.0 and cuDNN to C:\tools\cuda, update your %PATH% to match:

  - SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.2\bin;%PATH%
  - SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.2\extras\CUPTI\lib64;%PATH%
  - SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.2\include;%PATH%
  - SET PATH=C:\tools\cuda\bin;%PATH%

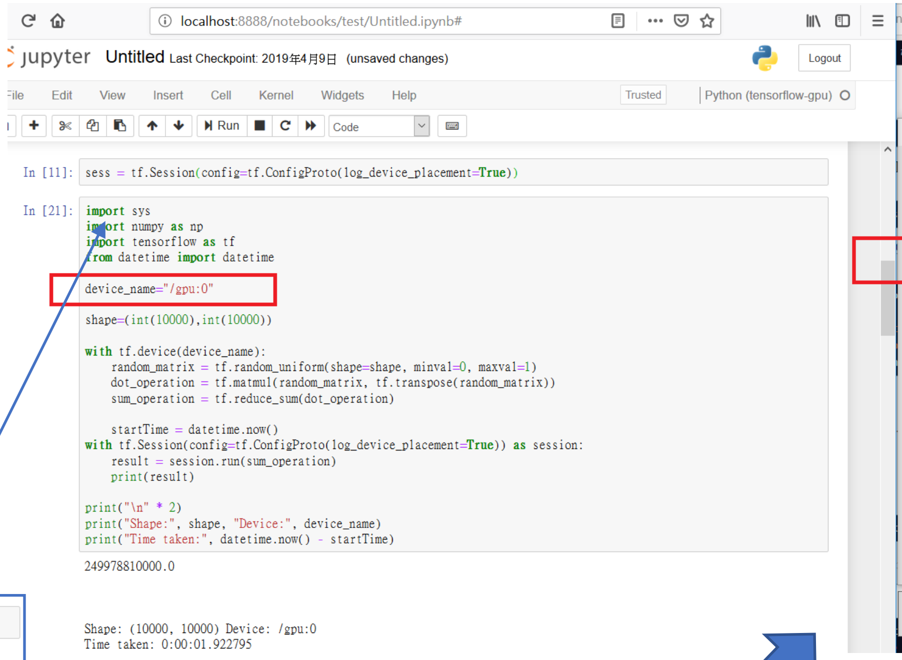
https://www.tensorflow.org/install/gpu

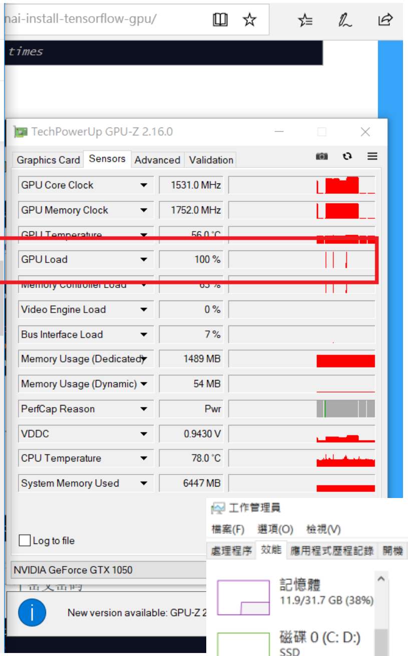
How to ensure tensorflow is using the GPU

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 10992383333532265700
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 1420898713
locality {
  bus_id: 1
  links {
  }
}
incarnation: 5464375807778728951
physical_device_desc: "device: 0, name: GeForce GTX 1050, pci bus id: 0000:01:00.0, compute capability: 6.1"
]
```

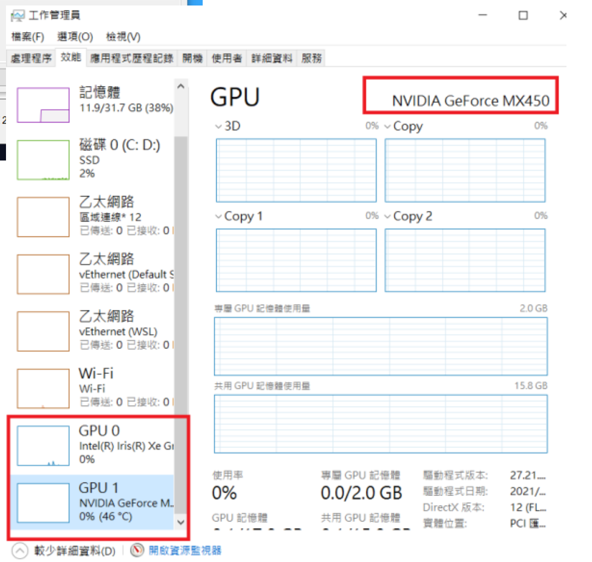




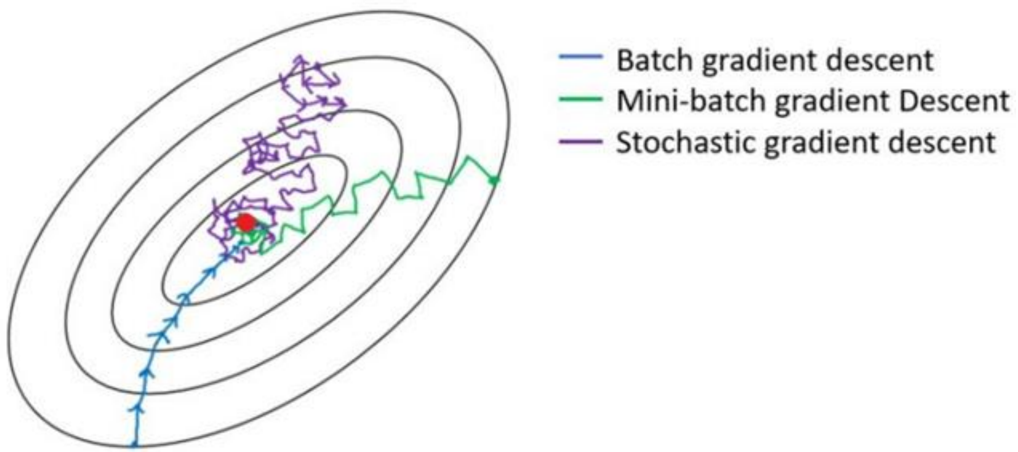

```
sess=tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(log_device_placement=True))

Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: GeForce MX130, pci bus id: 0000:01:0
0.0, compute capability: 5.0
```

Tensorfllow 2.x

[GPU-Z Graphics Card GPU Information Utility (techpowerup.com)]



Practical Aspects of Learning



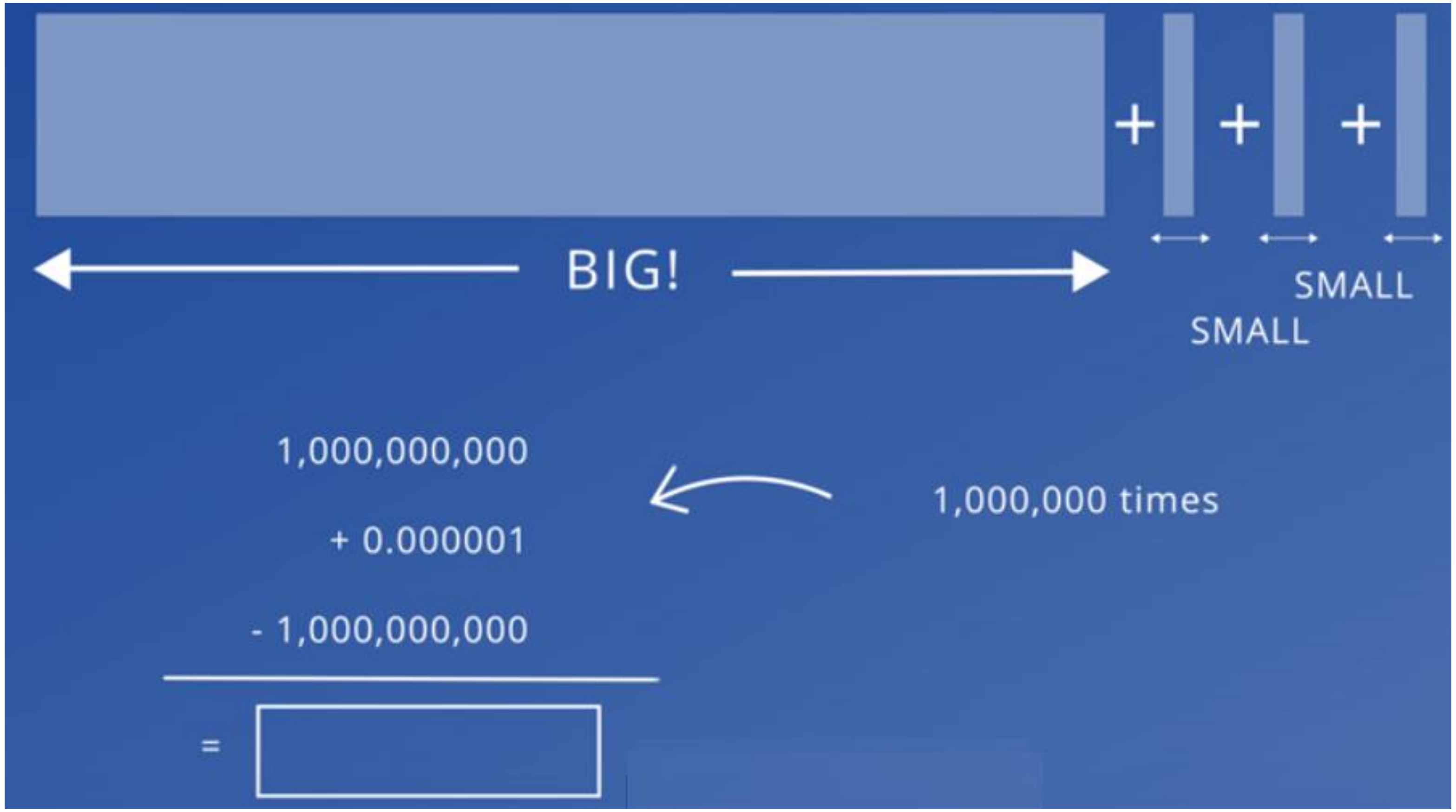
- How do you fill image pixels to this classifier?

- Where do you initialize the optimization?

Data normalization

- In theory, regression is insensitive to standardization since any linear transformation of input data can be counteracted by adjusting model parameters.
- Standardization improves the numerical stability of your model
- Standardization may speed up the training process
- Standardization isn't always great. It can harm the performance of distance-based clustering algorithms by assuming equal importance of features. If there are inherent importance differences between features, it's generally not a good idea to do standardization.

Quiz: Numerical Stability

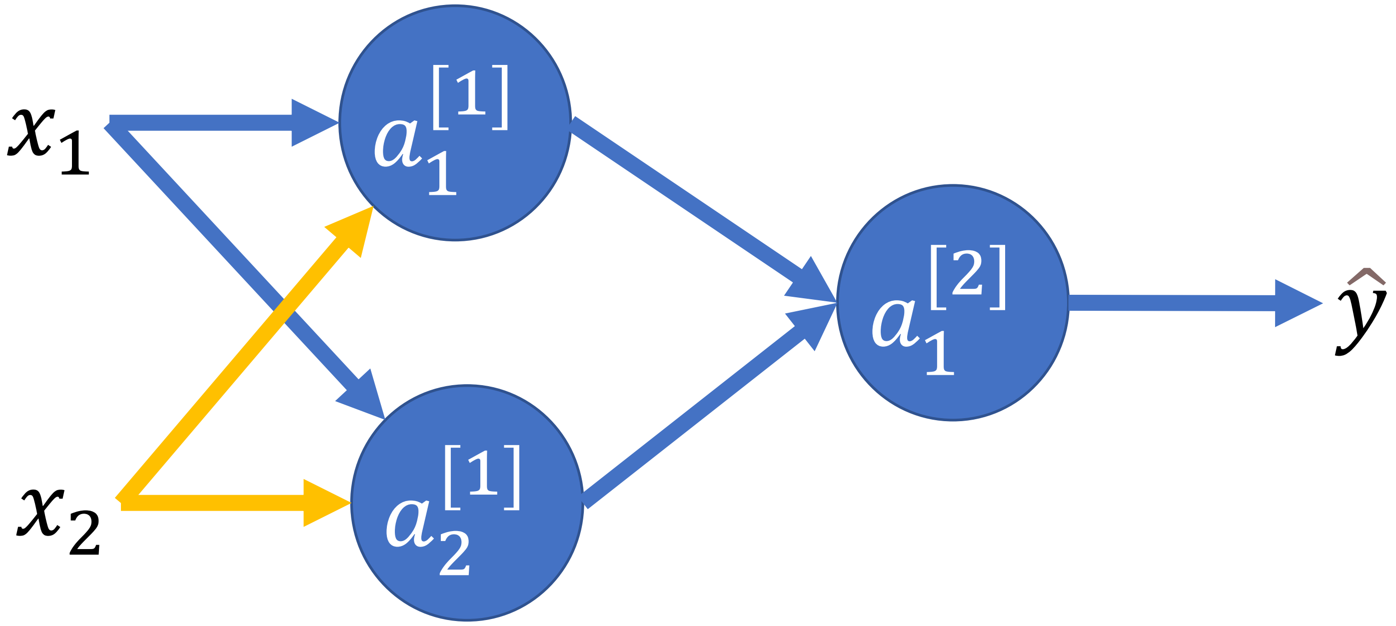



```
a = 1000000000
for i in range(1000000):
    a = a + 1e-6
print(a - 1000000000)
```

Replace the one billion with just one?

Code: [07Code.txt]()

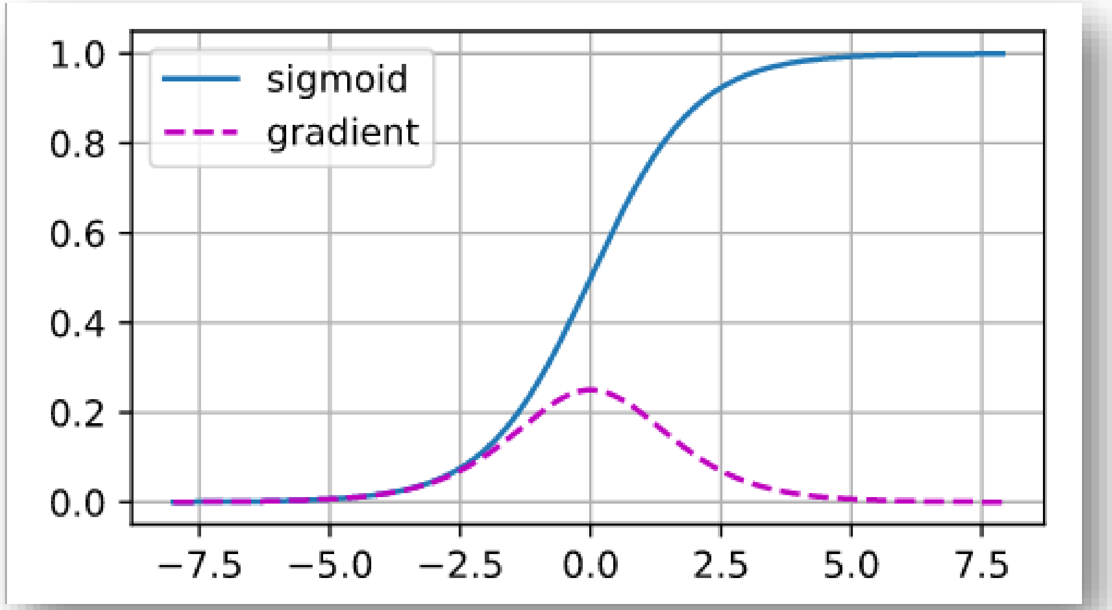
Breaking the Symmetry



$w^{[1]}=\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $w^{[2]}=[0 \;\; 0]$

Vanishing Gradients & Exploding Gradients

```
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from d2l import tensorflow as d2l

x = tf.Variable(tf.range(-8.0, 8.0, 0.1))
with tf.GradientTape() as t:
    y = tf.nn.sigmoid(x)
d2l.plot(x.numpy(), [y.numpy(), t.gradient(y, x).numpy()],
         legend=['sigmoid', 'gradient'], figsize=(4.5, 2.5))
```



Normalized Inputs and Initial Weights

Care about the calculation of “lost function”

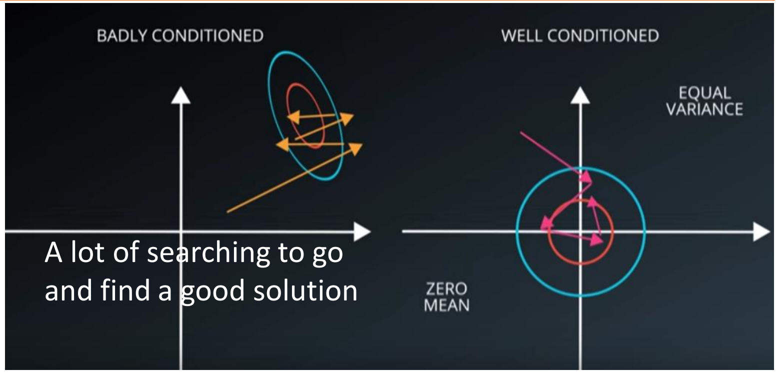
Xavier initialization :2010 Xavier Glorot , Yoshua Bengio《Understanding the difficulty of training deep feedforward neural networks》

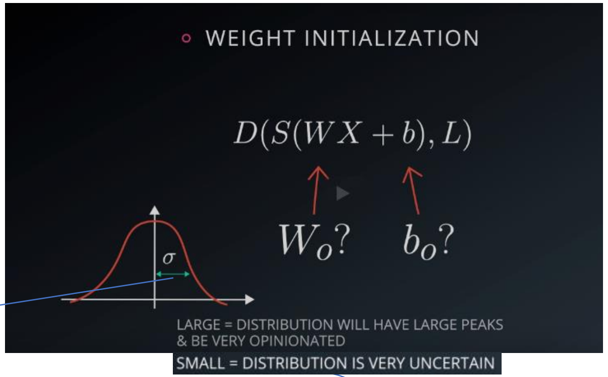
MEAN

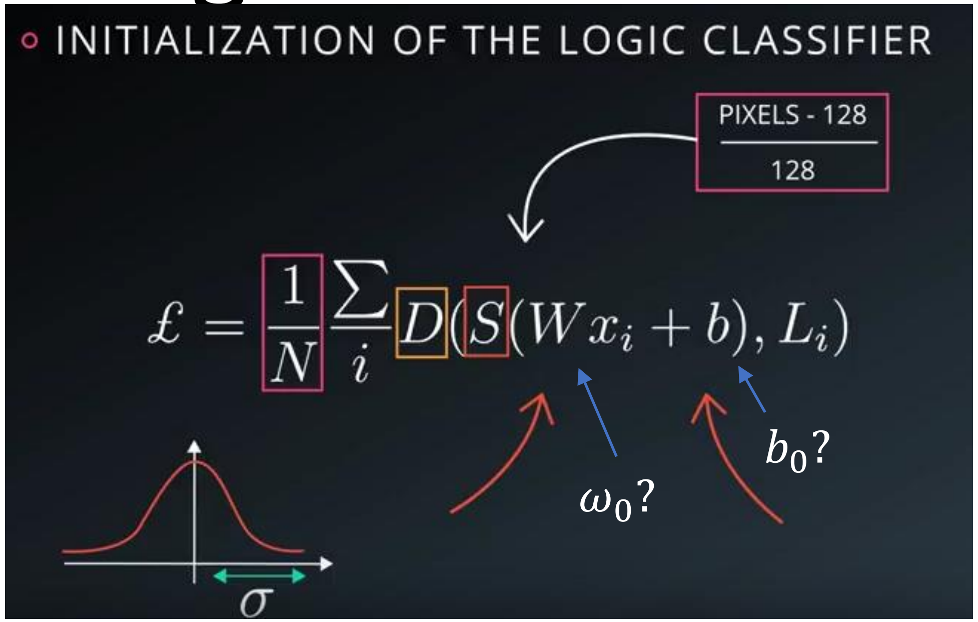
$$\mu(X_i) = 0$$

VARIANCE

$$\sigma(X_i) = \sigma(X_j)$$







Sigma determines order of magnitude of the output at the start

output = WX + b

output = [0.01, 1.0, 0.001]

Softmax determines peakiness

softmax_output = S(WX + b)

softmax_output = [0.214, 0.574, 0.212]

Large sigma = Opinionated model

OPTIMIZATION

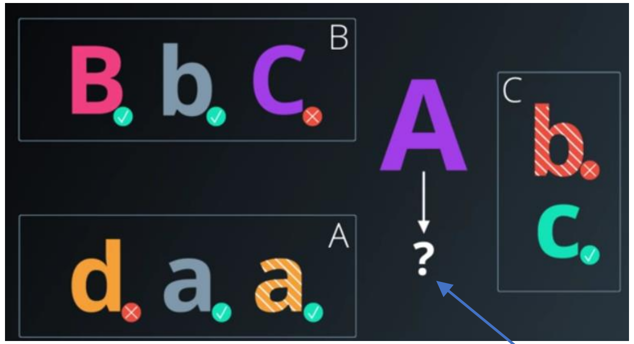
$$w \leftarrow w - \alpha \Delta_w \pounds$$

$$b \leftarrow b - \alpha \Delta_b \pounds$$

Measuring Performance

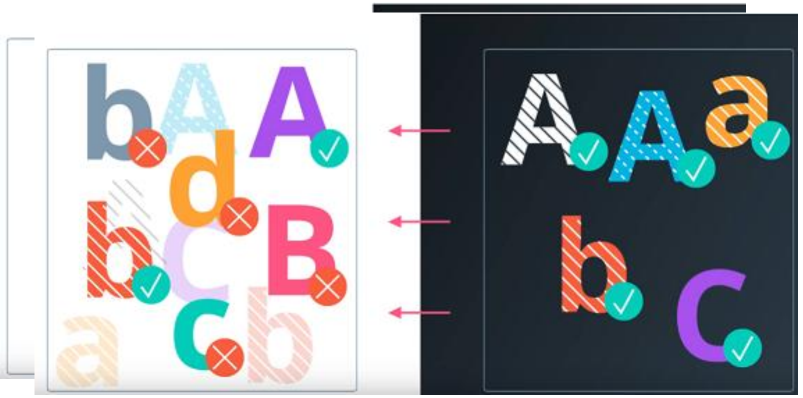
Measuring Performance

The problem is that your classifier has memorized the training set and it fails to generalize to new examples.



Generalization?

Deploy system in a real production environment.



Training set: A set of examples used for learning, which is to fit the parameters [i.e., weights] of the classifier.
Validation set: A set of examples used to tune the parameters [i.e., architecture, not weights] of a classifier, for example to choose the number of hidden units in a neural network.
Test set: A set of examples used only to assess the performance [generalization] of a fully specified classifier.

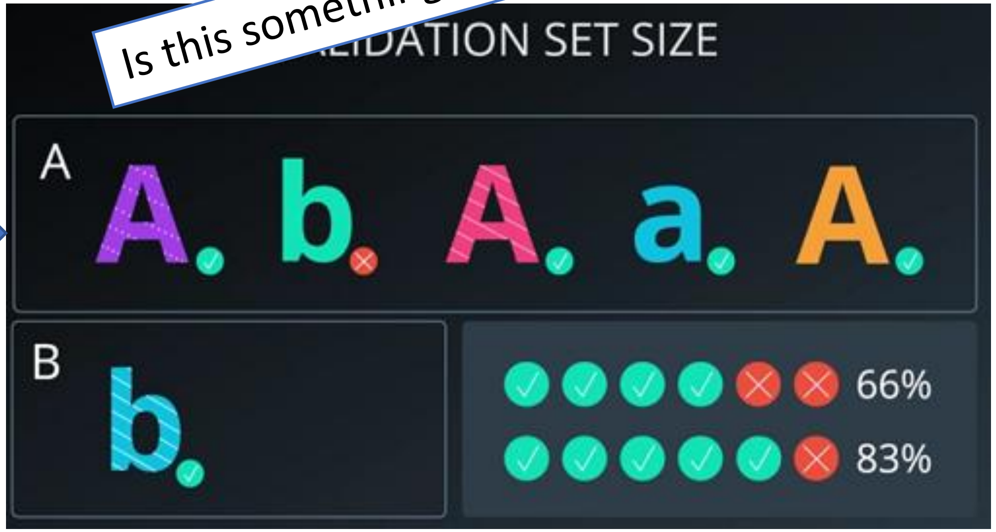
Never look at it until you have made your final decision.

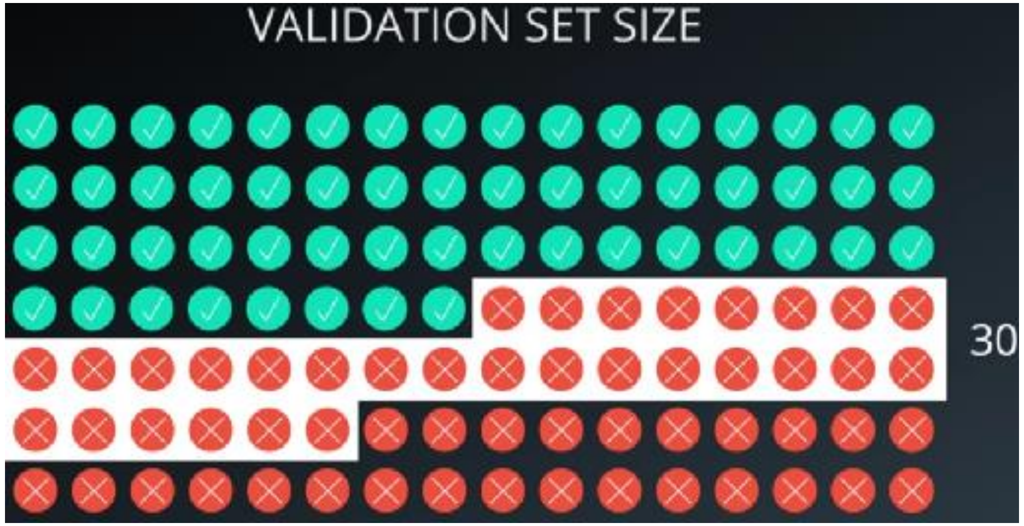
Validation and Test Set Size

Imagine that your validation set have six examples whit an accuracy of 66%.

tweak model

Is this something you can trust?





central limit theorem
(Statistically significant)
sample size>30

Validation Set Size

Quiz: Validation Set Size

Rule of '30'

	Yes	NO	
3000 EXAMPLES	☐	☐	80%→81%
	☐	☐	80%→80.5%
	☐	☐	80%→80.1%

Validation Test Set Size Continued

OR LOOK INTO CROSS-VALIDATION

Validation Set Size

> 30000 EXAMPLES

CHANGES > 0.1% in ACCURACY

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Machine Learning Fundamentals: Cross Validation
https://www.youtube.com/watch?v=fSytzGwwBVw

Optimizing a Logistic Classifier

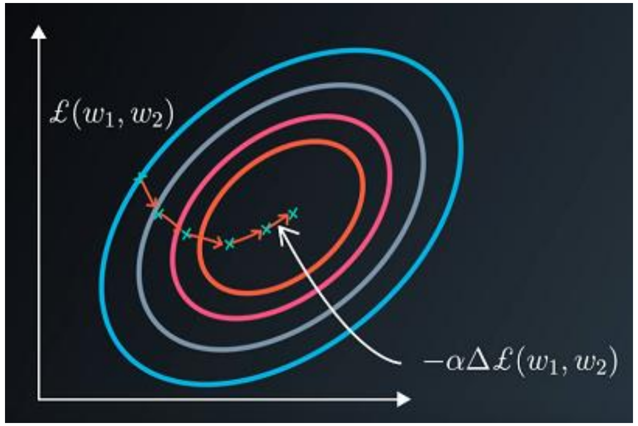
Design the right lost function to optimize.

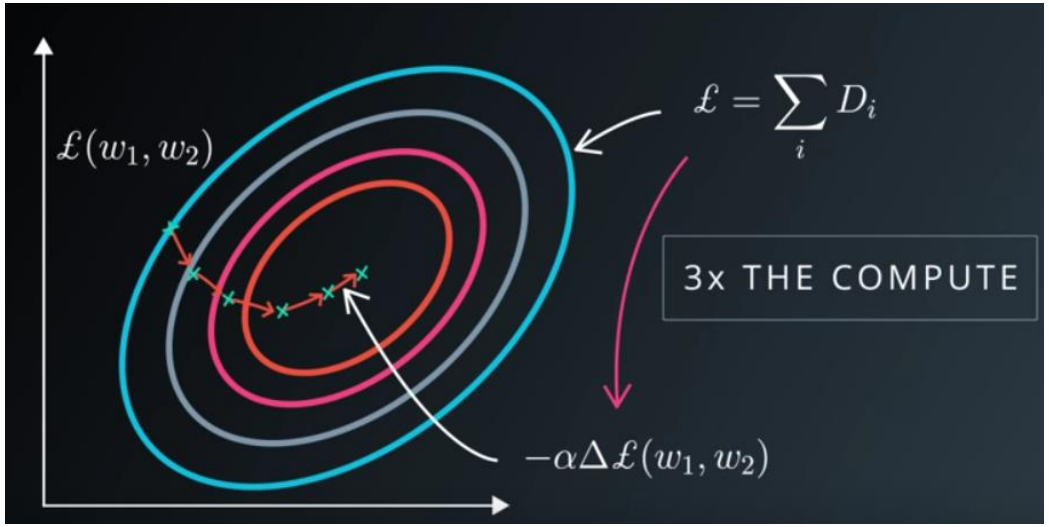
Training logistic regression

- OPTIMIZES ERROR MEASURE
- SCALING ISSUES

The biggest one is that it's very difficult to scale.

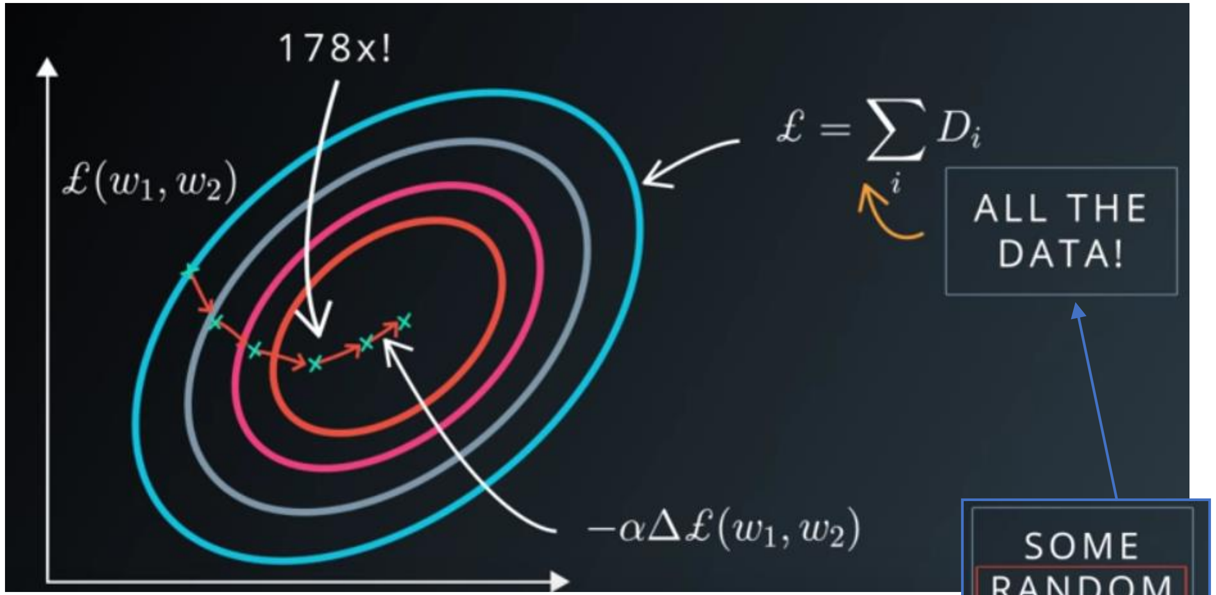
Stochastic Gradient Descent

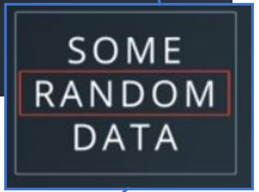




If computing loss takes n floating point operations, computing its gradient takes about 3 times compute.

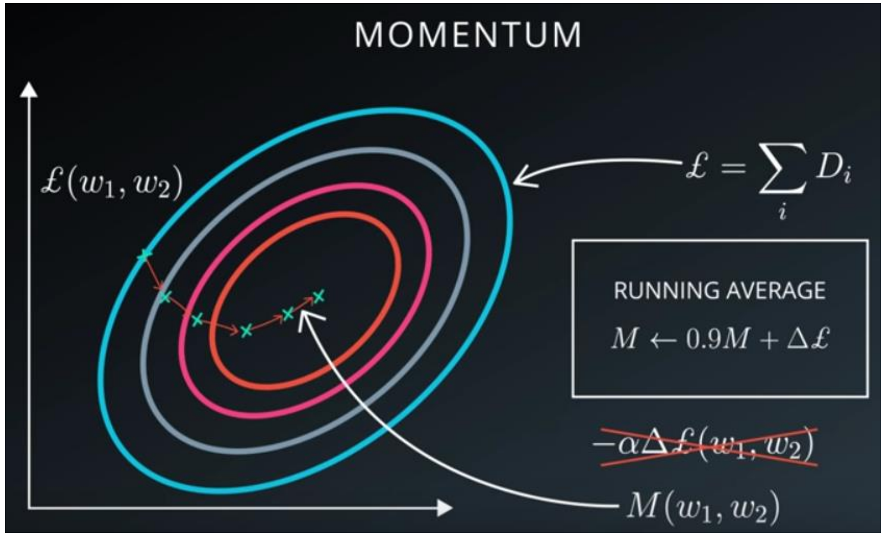
Gradient Descent(iterative)

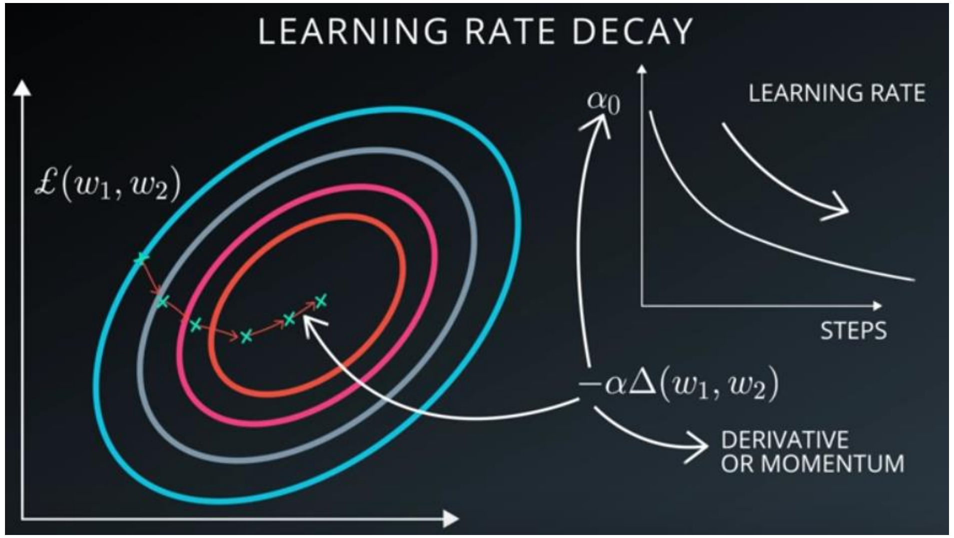




bad optimizer but fast

Momentum and Learning Rate Decay

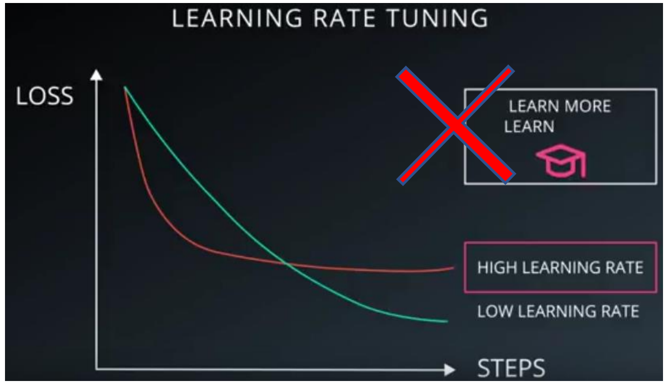


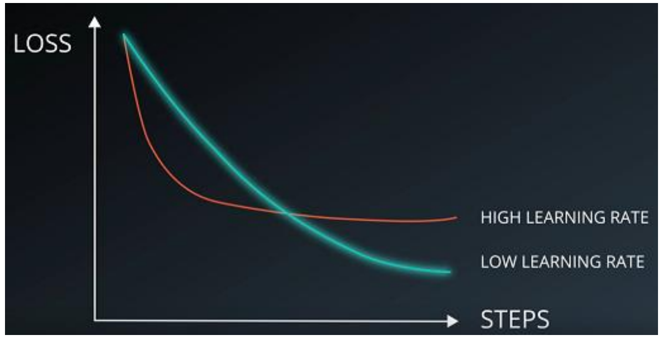


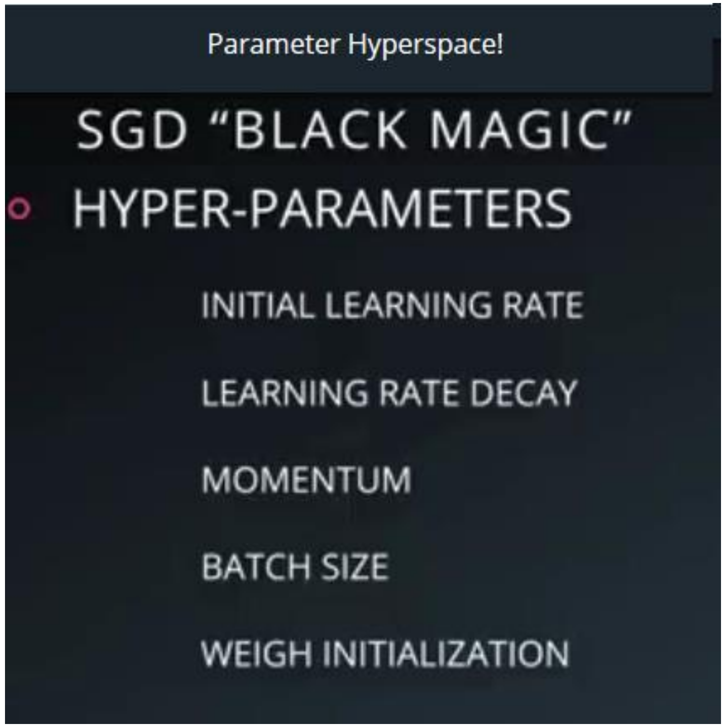
Momentum: We can take advantage of the knowledge that we've accumulated from previous steps about where we should be headed.

Learning Rate Decay: Apply an exponential decay to their learning rate.

Learning RATE TUNING & **Parameter Hyperspace**







Quiz: Mini-batch

Mini-batching

In this section, you'll go over what mini-batching is and how to apply it in TensorFlow. Mini-batching is a technique for training on subsets of the dataset instead of all the data at one time. This provides the ability to train a model, even if a computer lacks the memory to store the entire dataset.

Mini-batching is computationally inefficient, since you can't calculate the loss simultaneously across all samples. However, this is a small price to pay in order to be able to run the model at all. It's also quite useful combined with SGD. The idea is to randomly shuffle the data at the start of each epoch, then create the mini-batches. For each mini-batch, you train the network weights with gradient descent. Since these batches are random, you're performing SGD with each batch.

Let's look at the MNIST dataset with weights and a bias to see if your machine can handle it.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
import numpy as np
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
test_features = mnist.test.images
train_labels = mnist.train.labels.astype(np.float32)
test_labels = mnist.test.labels.astype(np.float32)
# Weights & bias
weights = tf.Variable(tf.random_normal([n_input, n_classes]))
bias = tf.Variable(tf.random_normal([n_classes]))
```

Quiz: Mini-batch

Question 1

Calculate the memory size of train_features, train_labels, weights, and bias in bytes. Ignore memory for overhead, just calculate the memory required for the stored data.

You may have to look up how much memory a float32 requires, using this link.

train_features Shape: (55000, 784) Type: float32

train_labels Shape: (55000, 10) Type: float32

weights Shape: (784, 10) Type: float32

bias Shape: (10,) Type: float32

How many bytes of memory does train_features need?

55000*784*4 byte(float 浮點數，佔 4 Bytes)

How many bytes of memory does train_labels need?

55000*10*4 byte

How many bytes of memory does weights need?

784*10*4 byte

How many bytes of memory does bias need?

10*4 byte

The total memory space required for the inputs, weights and bias is around 174 megabytes, which isn't that much memory. You could train this whole dataset on most CPUs and GPUs.

But larger datasets that you'll use in the future measured in gigabytes or more. It's possible to purchase more memory, but it's expensive. A Titan X GPU with 12 GB of memory costs over $1,000.

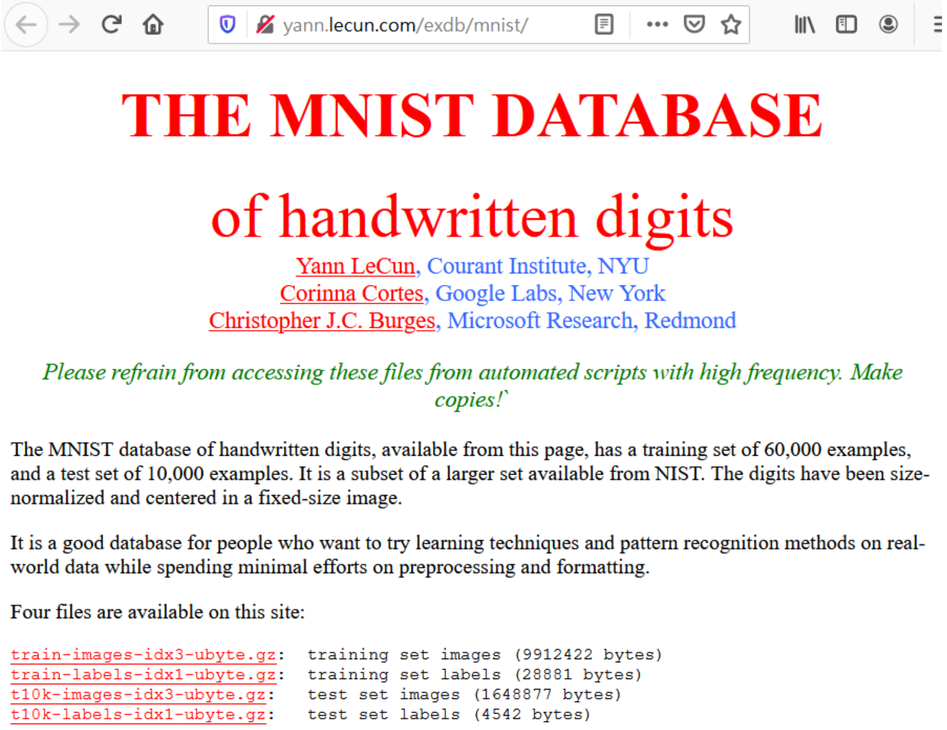
Instead, in order to run large models on your machine, you'll learn how to use mini-batching.

Let's look at how you implement mini-batching in TensorFlow.

The MNIST data

http://yann.lecun.com/exdb/mnist/

The MNIST data is split into three parts: 55,000 data points of training data (mnist.train), 10,000 points of test data (mnist.test), and 5,000 points of validation data (mnist.validation).


yann.lecun.com/exdb/mnist/

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

Please refrain from accessing these files from automated scripts with high frequency. Make copies!

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

Quiz: Mini-batch

TensorFlow Mini-batching

In order to use mini-batching, you must first divide your data into batches.

Unfortunately, it's sometimes impossible to divide the data into batches of exactly equal size. For example, imagine you'd like to create batches of 128 samples each from a dataset of 1000 samples. Since 128 does not evenly divide into 1000, you'd wind up with 7 batches of 128 samples, and 1 batch of 104 samples. (7*128 + 1*104 = 1000)

In that case, the size of the batches would vary, so you need to take advantage of TensorFlow's tf.placeholder() function to receive the varying batch sizes.

Continuing the example, if each sample had n_input = 784 features and n_classes = 10 possible labels, the dimensions for features would be [None, n_input] and labels would be [None, n_classes].

```
# Features and Labels
features = tf.placeholder(tf.float32, [None, n_input])
labels = tf.placeholder(tf.float32, [None, n_classes])
```

What does None do here?
The None dimension is a placeholder for the batch size. At runtime, TensorFlow will accept any batch size greater than 0. Going back to our earlier example, this setup allows you to feed features and labels into the model as either the batches of 128 samples or the single batch of 104 samples.

Question 2
Use the parameters below, how many batches are there, and what is the last batch size?
features is (50000, 400)
labels is (50000, 10)
batch_size is 128

Quiz: Mini-batch

How many batches are there?

50000/128≅391

What is the last batch size?

50000 mod 128=80

Now that you know the basics, let's learn how to implement mini-batching.

Question 3

Implement the batches function to batch features and labels. The function should return each batch with a maximum size of batch_size. To help you with the quiz, look at the following example output of a working batches function.

```
# 4 Samples of features
example_features = [
    ['F11','F12','F13','F14'],
    ['F21','F22','F23','F24'],
    ['F31','F32','F33','F34'],
    ['F41','F42','F43','F44']]
# 4 Samples of labels
example_labels = [
    ['L11','L12'],
    ['L21','L22'],
    ['L31','L32'],
    ['L41','L42']]

example_batches = batches(3, example_features, example_labels)
```

Quiz: Mini-batch

The example_batches variable would be the following:

```
[
  # 2 batches:  #   First is a batch of size 3.  #   Second is a batch of size 1
  [
    # First Batch is size 3
    [
      # 3 samples of features.   # There are 4 features per sample.
      ['F11', 'F12', 'F13', 'F14'],
      ['F21', 'F22', 'F23', 'F24'],
      ['F31', 'F32', 'F33', 'F34']
    ], [
      # 3 samples of labels.  # There are 2 labels per sample.
      ['L11', 'L12'],
      ['L21', 'L22'],
      ['L31', 'L32']
    ]
 ], [
    # Second Batch is size 1.    # Since batch size is 3, there is only one sample left from the 4 samples.
    [
      # 1 sample of features.
      ['F41', 'F42', 'F43', 'F44']
    ], [
      # 1 sample of labels.
      ['L41', 'L42']
    ]
  ]
]
```

Implement the batches function in the "quiz.py" file below.

Code: 08sandbox_py.txt

Code: 08quiz_py.txt

Code: 08quiz_solution_py.txt

Let's use mini-batching to feed batches of MNIST features and labels into a linear model.
Set the batch size and run the optimizer over all the batches with the batches function. The recommended batch size is 128. If you have memory restrictions, feel free to make it smaller.

Code: 09quiz_py.txt Code: 09helper_py.txt

Code: 09quiz_solution_py.txt

The accuracy is low, but you probably know that you could train on the dataset more than once. You can train a model using the dataset multiple times. You'll go over this subject in the next section where we talk about "epochs".

Epochs

Epochs

An epoch is a single forward and backward pass of the whole dataset. This is used to increase the accuracy of the model without requiring more data. This section will cover epochs in TensorFlow and how to choose the right number of epochs.

The following TensorFlow code trains a model using 10 epochs.

```
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-batc


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
```

Running the code will output the following:

```
Epoch: 0    - Cost: 11.0    Valid Accuracy: 0.204
Epoch: 1    - Cost: 9.95    Valid Accuracy: 0.229
Epoch: 2    - Cost: 9.18    Valid Accuracy: 0.246
Epoch: 3    - Cost: 8.59    Valid Accuracy: 0.264
Epoch: 4    - Cost: 8.13    Valid Accuracy: 0.283
Epoch: 5    - Cost: 7.77    Valid Accuracy: 0.301
Epoch: 6    - Cost: 7.47    Valid Accuracy: 0.316
Epoch: 7    - Cost: 7.2     Valid Accuracy: 0.328
Epoch: 8    - Cost: 6.96    Valid Accuracy: 0.342
Epoch: 9    - Cost: 6.73    Valid Accuracy: 0.36
Test Accuracy: 0.3801000118255615
```

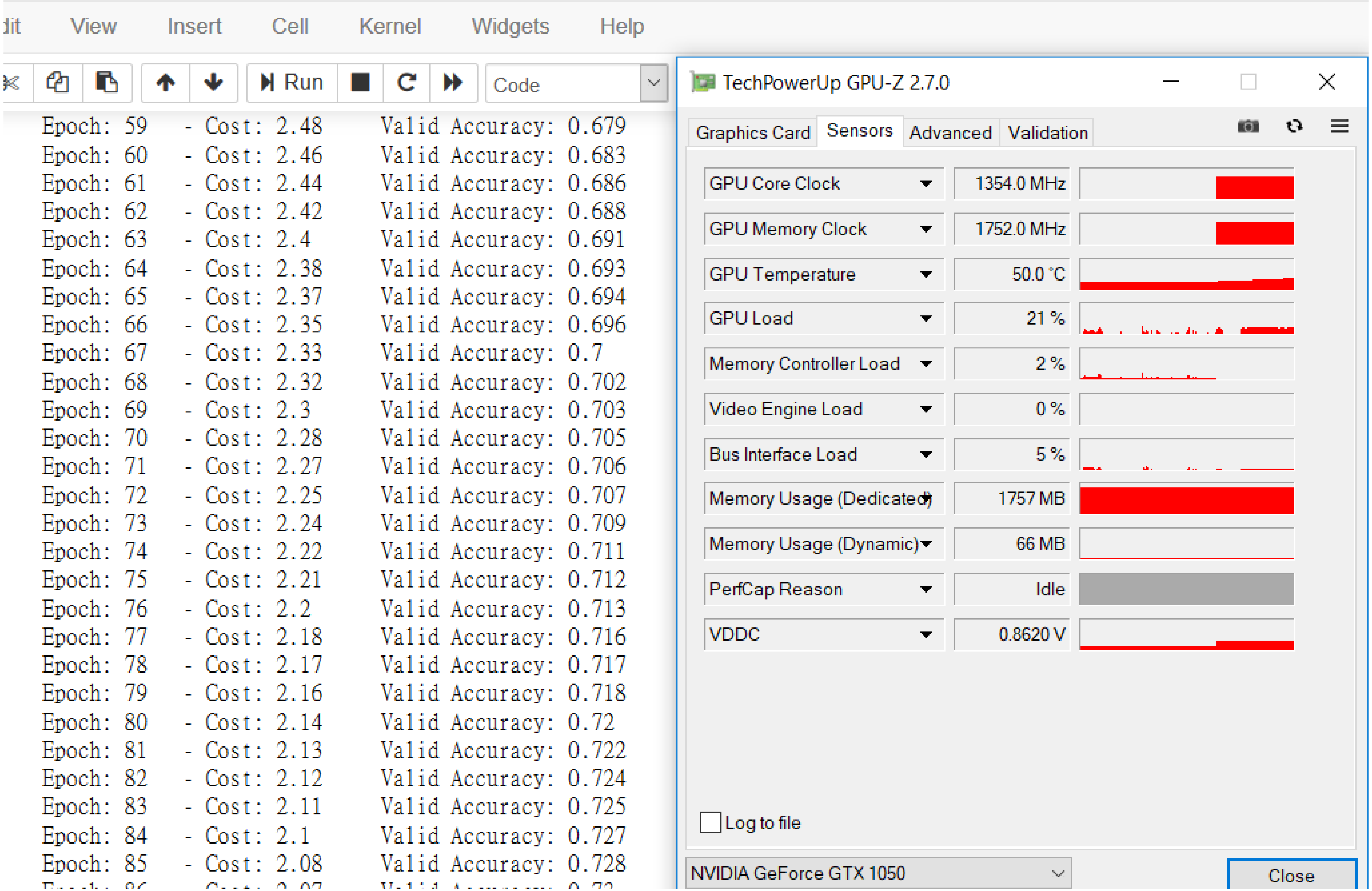
Epochs

Each epoch attempts to move to a lower cost, leading to better accuracy.
This model continues to improve accuracy up to Epoch 9. Let's increase the number of epochs to 100.

```
...
Epoch: 79   - Cost: 0.111    Valid Accuracy: 0.86
Epoch: 80   - Cost: 0.11    Valid Accuracy: 0.869
Epoch: 81   - Cost: 0.109    Valid Accuracy: 0.869
....
Epoch: 85   - Cost: 0.107    Valid Accuracy: 0.869
Epoch: 86   - Cost: 0.107    Valid Accuracy: 0.869
Epoch: 87   - Cost: 0.106    Valid Accuracy: 0.869
Epoch: 88   - Cost: 0.106    Valid Accuracy: 0.869
Epoch: 89   - Cost: 0.105    Valid Accuracy: 0.869
Epoch: 90   - Cost: 0.105    Valid Accuracy: 0.869
Epoch: 91   - Cost: 0.104    Valid Accuracy: 0.869
Epoch: 92   - Cost: 0.103    Valid Accuracy: 0.869
Epoch: 93   - Cost: 0.103    Valid Accuracy: 0.869
Epoch: 94   - Cost: 0.102    Valid Accuracy: 0.869
Epoch: 95   - Cost: 0.102    Valid Accuracy: 0.869
Epoch: 96   - Cost: 0.101    Valid Accuracy: 0.869
Epoch: 97   - Cost: 0.101    Valid Accuracy: 0.869
Epoch: 98   - Cost: 0.1      Valid Accuracy: 0.869
Epoch: 99   - Cost: 0.1      Valid Accuracy: 0.869
Test Accuracy: 0.8696000006198883
```

From looking at the output above, you can see the model doesn‘t increase the validation accuracy after epoch 80. Let’s see what happens when we increase the learning rate. *learn_rate = 0.1*

```
Epoch: 76   - Cost: 0.214    Valid Accuracy: 0.752
Epoch: 77   - Cost: 0.21     Valid Accuracy: 0.756
Epoch: 78   - Cost: 0.21     Valid Accuracy: 0.756
...
Epoch: 85   - Cost: 0.207    Valid Accuracy: 0.756
Epoch: 86   - Cost: 0.209    Valid Accuracy: 0.756
Epoch: 87   - Cost: 0.205    Valid Accuracy: 0.756
Epoch: 88   - Cost: 0.208    Valid Accuracy: 0.756
Epoch: 89   - Cost: 0.205    Valid Accuracy: 0.756
Epoch: 90   - Cost: 0.202    Valid Accuracy: 0.756
Epoch: 91   - Cost: 0.207    Valid Accuracy: 0.756
Epoch: 92   - Cost: 0.204    Valid Accuracy: 0.756
Epoch: 93   - Cost: 0.206    Valid Accuracy: 0.756
Epoch: 94   - Cost: 0.202    Valid Accuracy: 0.756
Epoch: 95   - Cost: 0.2974   Valid Accuracy: 
0.756
Epoch: 96   - Cost: 0.202    Valid Accuracy: 0.756
Epoch: 97   - Cost: 0.2996   Valid Accuracy: 
0.756
Epoch: 98   - Cost: 0.203    Valid Accuracy: 0.756
Epoch: 99   - Cost: 0.2987   Valid Accuracy: 
0.756
Test Accuracy: 0.7556000053882599
```



dit View Insert Cell Kernel Widgets Help
Run
Code
Epoch: 59 - Cost: 2.48 Valid Accuracy: 0.679
Epoch: 60 - Cost: 2.46 Valid Accuracy: 0.683
Epoch: 61 - Cost: 2.44 Valid Accuracy: 0.686
Epoch: 62 - Cost: 2.42 Valid Accuracy: 0.688
Epoch: 63 - Cost: 2.4 Valid Accuracy: 0.691
Epoch: 64 - Cost: 2.38 Valid Accuracy: 0.693
Epoch: 65 - Cost: 2.37 Valid Accuracy: 0.694
Epoch: 66 - Cost: 2.35 Valid Accuracy: 0.696
Epoch: 67 - Cost: 2.33 Valid Accuracy: 0.7
Epoch: 68 - Cost: 2.32 Valid Accuracy: 0.702
Epoch: 69 - Cost: 2.3 Valid Accuracy: 0.703
Epoch: 70 - Cost: 2.28 Valid Accuracy: 0.705
Epoch: 71 - Cost: 2.27 Valid Accuracy: 0.706
Epoch: 72 - Cost: 2.25 Valid Accuracy: 0.707
Epoch: 73 - Cost: 2.24 Valid Accuracy: 0.709
Epoch: 74 - Cost: 2.22 Valid Accuracy: 0.711
Epoch: 75 - Cost: 2.21 Valid Accuracy: 0.712
Epoch: 76 - Cost: 2.2 Valid Accuracy: 0.713
Epoch: 77 - Cost: 2.18 Valid Accuracy: 0.716
Epoch: 78 - Cost: 2.17 Valid Accuracy: 0.717
Epoch: 79 - Cost: 2.16 Valid Accuracy: 0.718
Epoch: 80 - Cost: 2.14 Valid Accuracy: 0.72
Epoch: 81 - Cost: 2.13 Valid Accuracy: 0.722
Epoch: 82 - Cost: 2.12 Valid Accuracy: 0.724
Epoch: 83 - Cost: 2.11 Valid Accuracy: 0.725
Epoch: 84 - Cost: 2.1 Valid Accuracy: 0.727
Epoch: 85 - Cost: 2.08 Valid Accuracy: 0.728
TechPowerUp GPU-Z 2.7.0
Graphics Card Sensors Advanced Validation
GPU Core Clock 1354.0 MHz
GPU Memory Clock 1752.0 MHz
GPU Temperature 50.0 °C
GPU Load 21 %
Memory Controller Load 2 %
Video Engine Load 0 %
Bus Interface Load 5 %
Memory Usage (Dedicated) 1757 MB
Memory Usage (Dynamic) 66 MB
PerfCap Reason Idle
VDDC 0.8620 V
Log to file
NVIDIA GeForce GTX 1050
Close

complement

- In general, an epoch in deep learning sense means we are passing through the whole training dataset, traversing through all the example, for one time, during the training process.
- This is used to increase the accuracy of the model without requiring more data.
- In neural networks generally, **an epoch is a single pass through the full training set**. You don’t just run through the training set once, it can take thousands of epochs for your backpropagation algorithm to converge on a combination of weights with an acceptable level of accuracy. Remember gradient descent only changes the weights by a small amount in the direction of improvement, so backpropagation can’t get there by running through the training examples just once.

complement

In the neural network terminology:

- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

>>Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

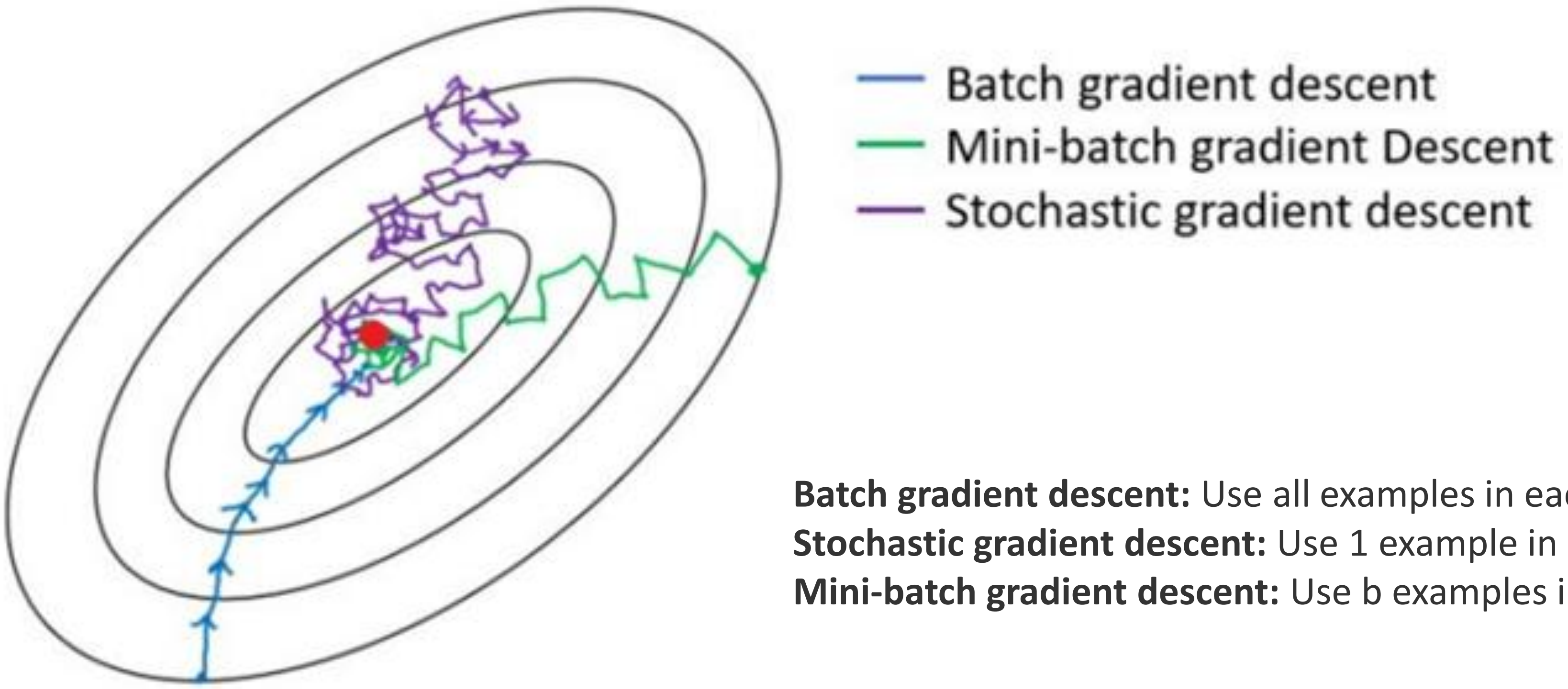
Epochs

Looks like the learning rate was increased too much. The final accuracy was lower, and it stopped improving earlier. Let's stick with the previous learning rate, but change the number of epochs to 80.

```
Epoch: 65   - Cost: 0.122    Valid Accuracy: 0.868
Epoch: 66   - Cost: 0.121    Valid Accuracy: 0.868
Epoch: 67   - Cost: 0.12     Valid Accuracy: 0.868
Epoch: 68   - Cost: 0.119    Valid Accuracy: 0.868
Epoch: 69   - Cost: 0.118    Valid Accuracy: 0.868
Epoch: 70   - Cost: 0.118    Valid Accuracy: 0.868
Epoch: 71   - Cost: 0.117    Valid Accuracy: 0.868
Epoch: 72   - Cost: 0.116    Valid Accuracy: 0.868
Epoch: 73   - Cost: 0.115    Valid Accuracy: 0.868
Epoch: 74   - Cost: 0.115    Valid Accuracy: 0.868
Epoch: 75   - Cost: 0.114    Valid Accuracy: 0.868
Epoch: 76   - Cost: 0.113    Valid Accuracy: 0.868
Epoch: 77   - Cost: 0.113    Valid Accuracy: 0.868
Epoch: 78   - Cost: 0.112    Valid Accuracy: 0.868
Epoch: 79   - Cost: 0.111    Valid Accuracy: 0.868
Epoch: 80   - Cost: 0.111    Valid Accuracy: 0.869
Test Accuracy: 0.8690999418258667
```

The accuracy only reached 0.86, but that could be because the learning rate was too high. Lowering the learning rate would require more epochs, but could ultimately achieve better accuracy. In the upcoming TensorFLow Lab, you'll get the opportunity to choose your own learning rate, epoch count, and batch size to improve the model's accuracy.

complement



Batch gradient descent: Use all examples in each iteration；
Stochastic gradient descent: Use 1 example in each iteration；
Mini-batch gradient descent: Use b examples in each iteration.

AWS GPU Instances

[AWS GPU Instances]

Lab: TensorFlow Neural Network

TensorFlow Neural Network Lab

We've prepared a Jupyter notebook that will guide you through the process of creating a single layer neural network in TensorFlow.

Setup

Run the commands below to clone the Lab Repository and then run the notebook:

```
git clone https://github.com/udacity/CarND-TensorFlow-Lab.git
# Make sure the starter kit environment is activated!
jupyter notebook
```

View The Notebook

Open a browser window and go here. This is the notebook you'll be working on. The notebook has 3 problems for you to solve:

- Problem 1: Normalize the features
- Problem 2: Use TensorFlow operations to create features, labels, weight, and biases tensors
- Problem 3: Tune the learning rate, number of steps, and batch size for the best accuracy

This is a self-assessed lab. Compare your answers to the solutions here.
If you have any difficulty completing the lab, Udacity provides a few services to answer any questions you might have.