MACHINE LEARNING WITH TENSOR FLOW

SCOPE: INTRODUCTION TO SOME FEATURES OF TENSOR FLOW TO GET YOU STARTED

Adrian Bevan
a.j.bevan@qmul.ac.uk

https://pprc.qmul.ac.uk/~bevan/statistics/TensorFlow.html

OUTLINE

- What this is not
- What this is mean to be
- Machine learning context
- Resources
- Tensor flow basics
- Example 1: Fractals
- Example 2: Fisher discriminant
- Example 3: Perceptron
- Example 4: Neural network
- Example 5: Using Tensor Board
- Example 6: Convolutional Neural Networks
- Want more data?

WHAT THIS IS NOT

- This is not a formal lecture or tutorial.
- You will not learn about algorithms.

WHAT THIS IS MEAN TO BE

- A relaxed session to explore a machine learning toolkit.

- A collection of resources is provided to get you started with using TensorFlow:
  - Provides you with working examples.
  - Run these to understand what they output.
  - Adapt examples to learn at a deeper level at your own pace.

- If you enjoy this then you may wish to explore the online tutorials further to delve into the toolkit's functionality.

- If you *really* enjoy this then you may wish to find some if your own data (see some suggestions at the end) and apply TensorFlow (or some other toolkit) to that in your own time.

- If you *really really* enjoy this then you may want to try and find a project to work on to take your interest further.

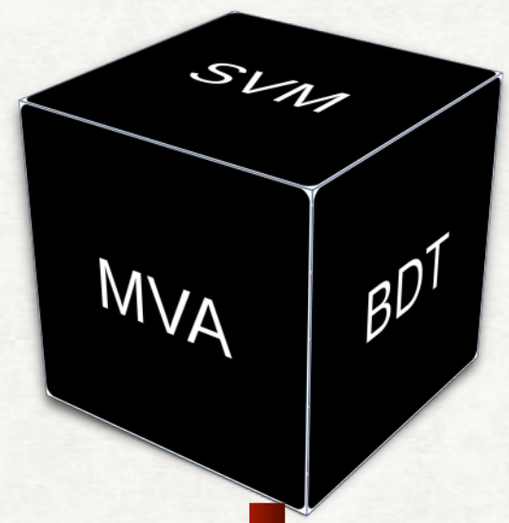
TECHNICAL DETAILS

- TensorFlow V1.0 was released on 15th Feb 2017;
- These scripts are compatible with that version;
- Some optimal code options have not been compiled in - please ignore those warnings when you get them.
- We are using Python 2.7.13 :: Anaconda 4.3.0 (64-bit)

MACHINE LEARNING CONTEXT

- Take features of some data

Invariant mass, transverse momentum, energy flow, jet tagging, missing energy, missing mass, angular separation, …



Separate Higgs, ttbar/QCD/etc background, …

- Do some magical* stuff with it
- Draw some insight seemingly from nowhere

*Machine learning (ML) is only magical if you consider the underlying algorithm as a complicated black box. Taking some time to understand the underlying algorithms and related computer science issues that underpin ML demystifies the magic and can highlight when things will work and when they might go wrong.

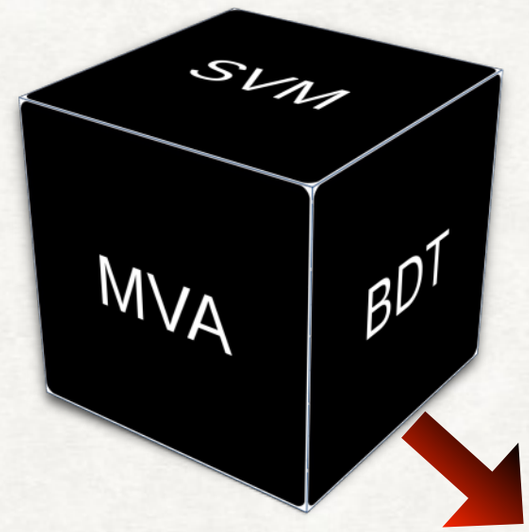
MACHINE LEARNING CONTEXT

- Take features of some data

Invariant mass, transverse momentum, energy flow, jet tagging, missing energy, missing mass, angular separation, …

MNIST: Image data; handwritten 0, 1, 2, 3, … 9 formatted to a fixed size matrix of pixels

- Do some magical* stuff with it



0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Separate Higgs, ttbar/QCD/etc background, …

- Draw some insight seemingly from nowhere

*Machine learning (ML) is only magical if you consider the underlying algorithm as a complicated black box. Taking some time to understand the underlying algorithms and related computer science issues that underpin ML demystifies the magic and can highlight when things will work and when they might go wrong.

RESOURCES

- The example scripts provided are to give you a base to start working with this toolkit.
- Download them, run them, read them, modify them.
- Great tutorials online at: https://www.tensorflow.org.
- If you prefer books, you can also find some online - ask for some suggestions.
- Downloading TensorFlow on your own computer can be complicated (we have experience with MacOSX and Scientific Linux), so defer to the website for that in the first instance… If you run into real problems after having a go then please come and ask; we may be able to help out.

USING PYTHON

- System requirements: see the TensorFlow web page (using API V1.0 for these examples).

- At the time of writing this TensorFlow is being actively maintained and developed; this means the interface, features etc. may change and the examples scripts will rot* with time.

- If you encounter code rot* in a script for a more recent version of TensorFlow please let me know so that I can update the examples.

- We are using Linux for these tutorials; useful commands are handed out in a moment.

*code rot is a commonly used term to indicate this situation; just because code worked on an old system does not mean it will work on a new on. Libraries change, standards change, languages evolve and your old code rots away accordingly.

TENSOR FLOW BASICS

- Typical modules to import for TensorFlow:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

- The first provides TensorFlow's Machine Learning library, the second provides a set of scientific computing tools and the third provides plotting tools.

- See the following for more information on each package:

- tensorflow https://www.tensorflow.org
- numpy http://www.numpy.org
- matplotlib http://matplotlib.org

SOME EXAMPLES

- **0) Simple calculations**
  - A selection of simple tf calculations on 2D matrices (tensors).
- **1) Fractals**
  - Aim: Use TensorFlow constants and variables to compute values in an array and plot the output. Start to get used to the way that you implement computations in TensorFlow.
- **2) Fisher Discriminant**
  - Generate a sample of data (2D) and from this compute fisher coefficients.
- **3) Perceptron**
  - Aim: Use TensorFlow to optimise the hyper-parameters of a perceptron.
- **4) Multilayer perceptron**
  - Aim: go beyond the simple network example above.
- **5) Tensor Board**
  - Aim: introduce you to tensor board as a way to graphically inspect models; using a simple calculation.

EXAMPLE 0: SIMPLE CALCULATIONS

- Mathematical ops on data are documented on the TensorFlow website:
  - https://www.tensorflow.org/api_guides/python/math_ops
- The aim here is to introduce you to a few with a simple example:

$$A = \begin{pmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{pmatrix} \qquad B = \begin{pmatrix} 1.0 & 2.0 \\ 2.0 & 1.0 \end{pmatrix}$$

- Element wise:
  - Addition, subtraction, multiplication, division.
- Scaling.
- Matrix operations: trace, multiplication.

- Run the script Examples.py to explore these operations; we will build on this background with the subsequent examples.

SUGGESTED EXERCISES

- Take a look at the following URL:
  - https://www.tensorflow.org/api_guides/python/math_ops
- to see what functionality is available and extend the script to perform more ops on data.

EXAMPLE 1: FRACTALS

- There are two example scripts provided for you to explore:
  - Mandelbrot.py — Use matplotlib to draw Mandelbrot sets
    - Draw points on the complex plane with the pixel colour related to the number of steps taken to reach the divergence threshold (initially set to 10). $z_0 = C$.

$$z_{n+1} = z_n^2 + C$$

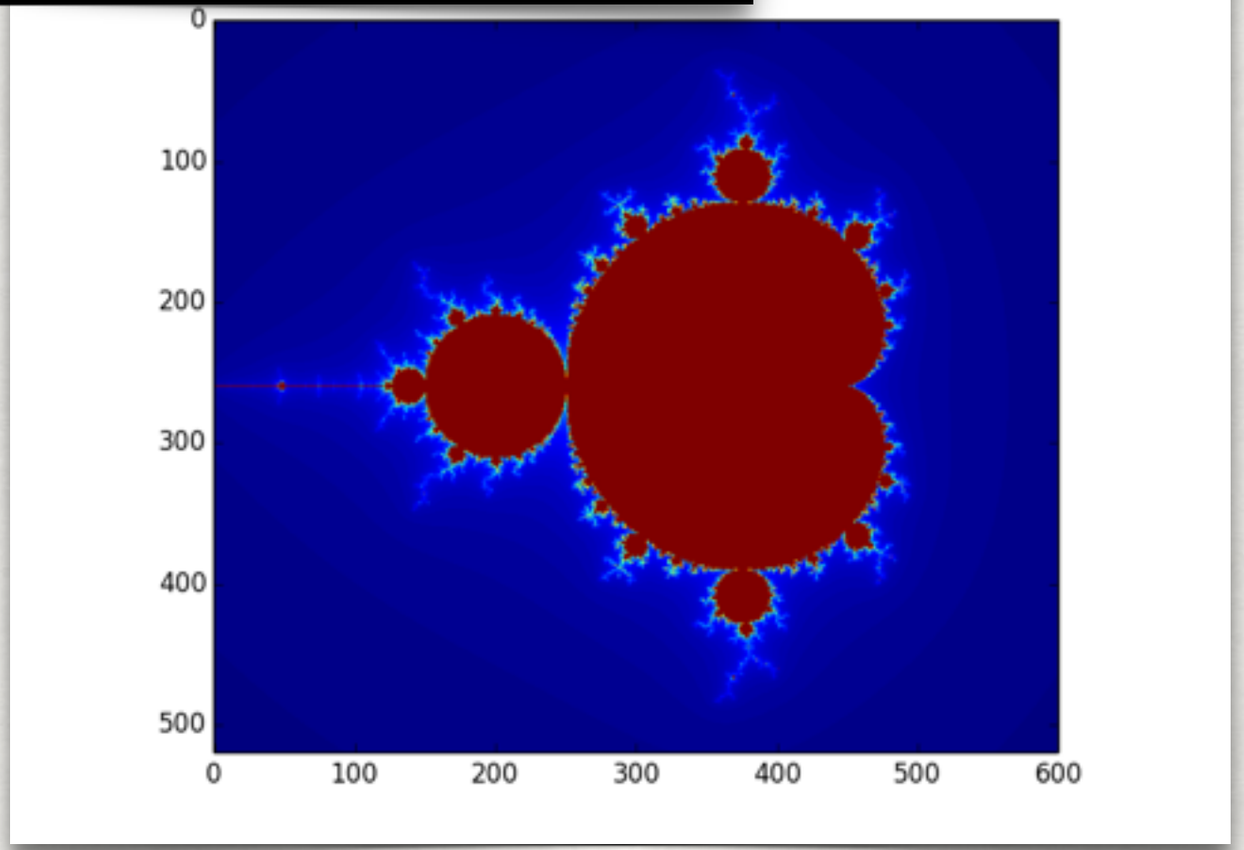
  - Julia.py — Use matplotlib to draw Julia sets
    - Draw points on the complex plane with the pixel colour related to the number of steps taken to reach the divergence threshold (initially set to 4). $z_0 = C$.

$$z_{n+1} = z_n^2 - C$$

EXAMPLE 1: FRACTALS

- np.mgrid creates a meshgrid between the minimum and maximum values using the specified step.
- np.mesh[min:max:step, min:max:step]

```
# set the grid for the MB plot, and define c as complex numbers,
# the corresponding tf object is zs.
Y, X = np.mgrid[ -1.3:1.3:0.005, -2:1:0.005 ]
Z = X+1j*Y
c = tf.constant(Z.astype(np.complex64))
zs = tf.Variable(c)
ns = tf.Variable(tf.zeros_like(c, tf.float32))
```

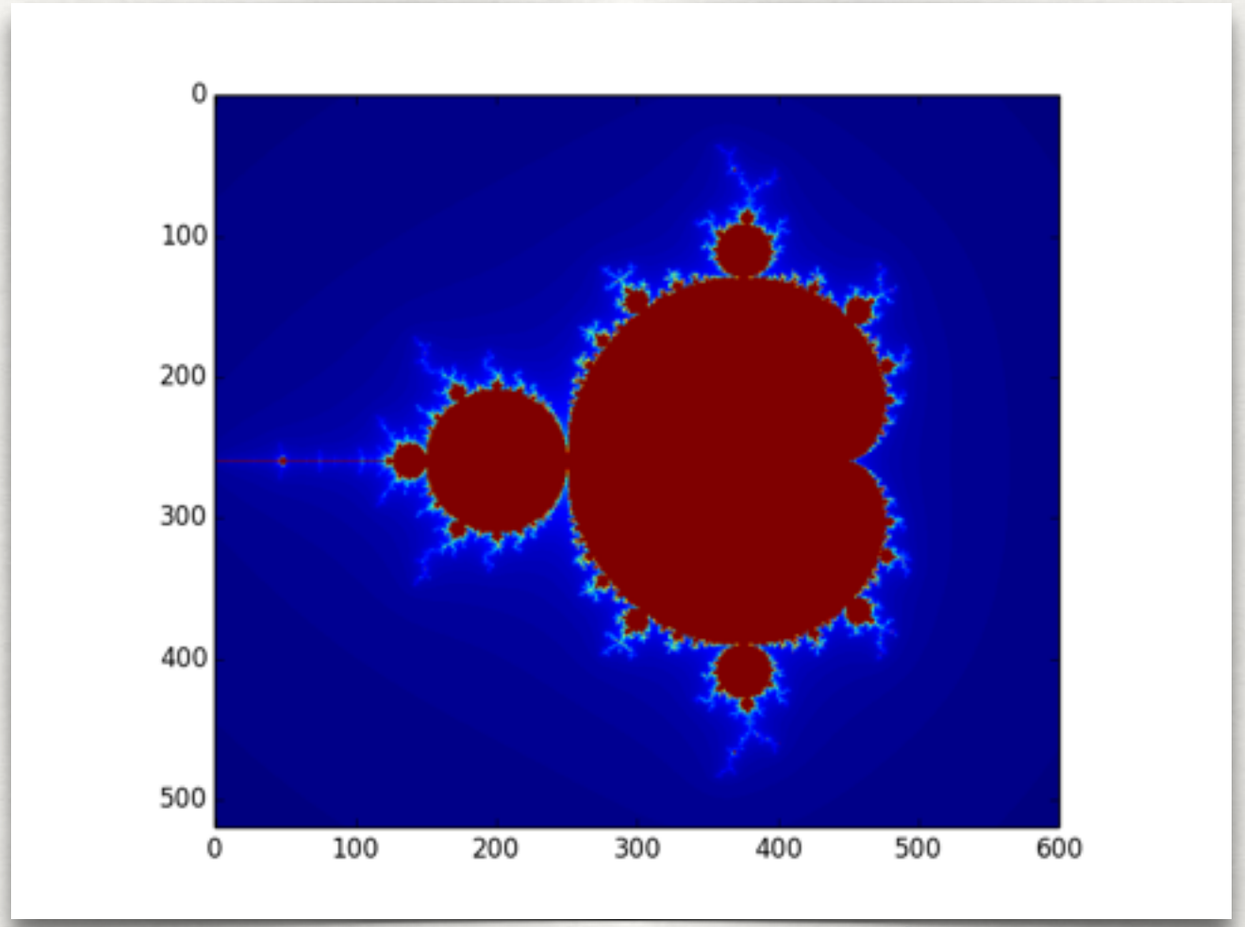


- Z and c are complex.
- zs is the initial value of the set as a TensorFlow variable.
- A tensor of zeros matching the shape of c with type float32.

EXAMPLE 1: FRACTALS

- Start an interactive session and set up the variables for processing.

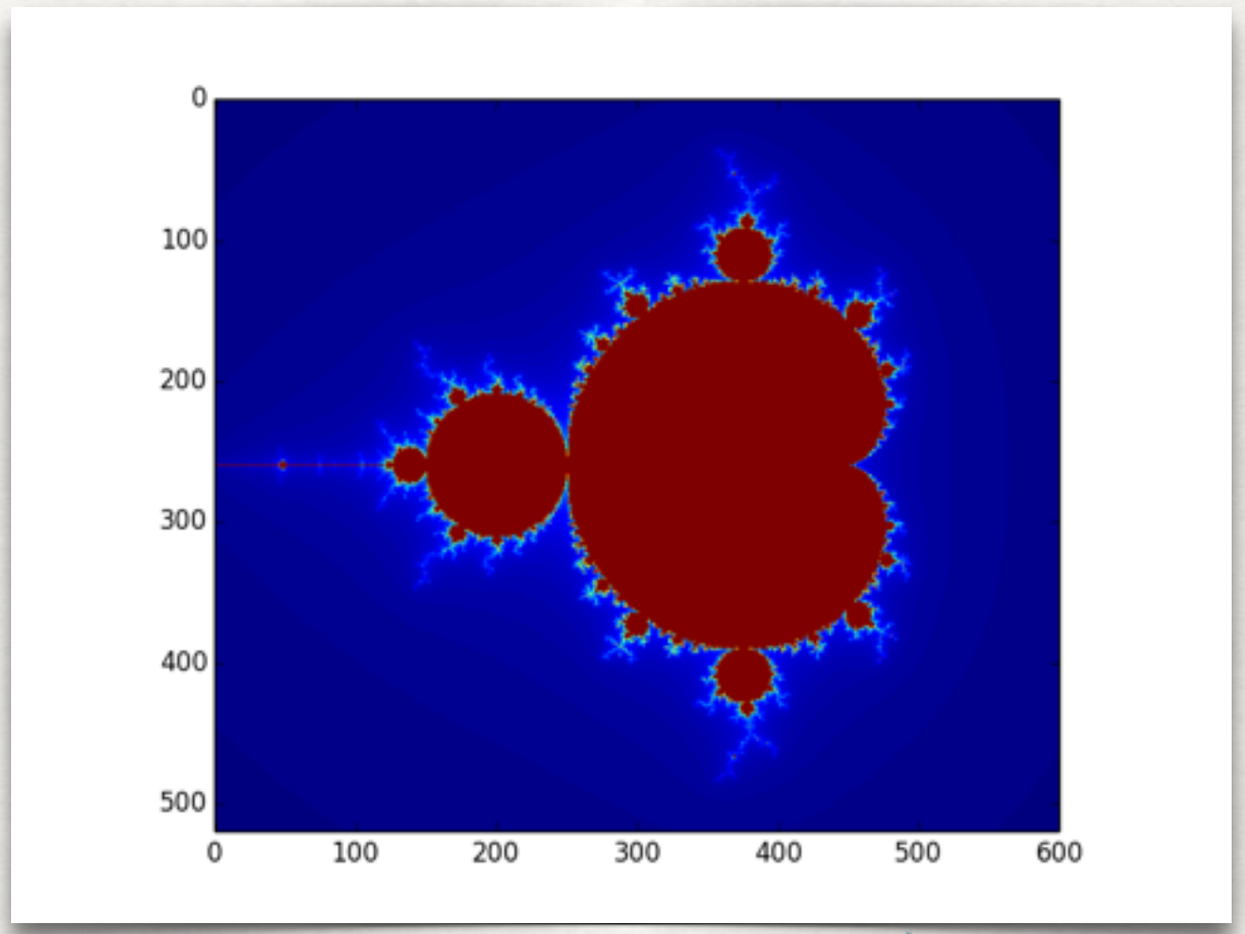
```
sees = tf.InteractiveSession()
tf.global_variables_initializer().run()
```



EXAMPLE 1: FRACTALS

- Compute the Mandelbrot set value using the recursive computation: $z_{n+1} = z_n^2 + C$

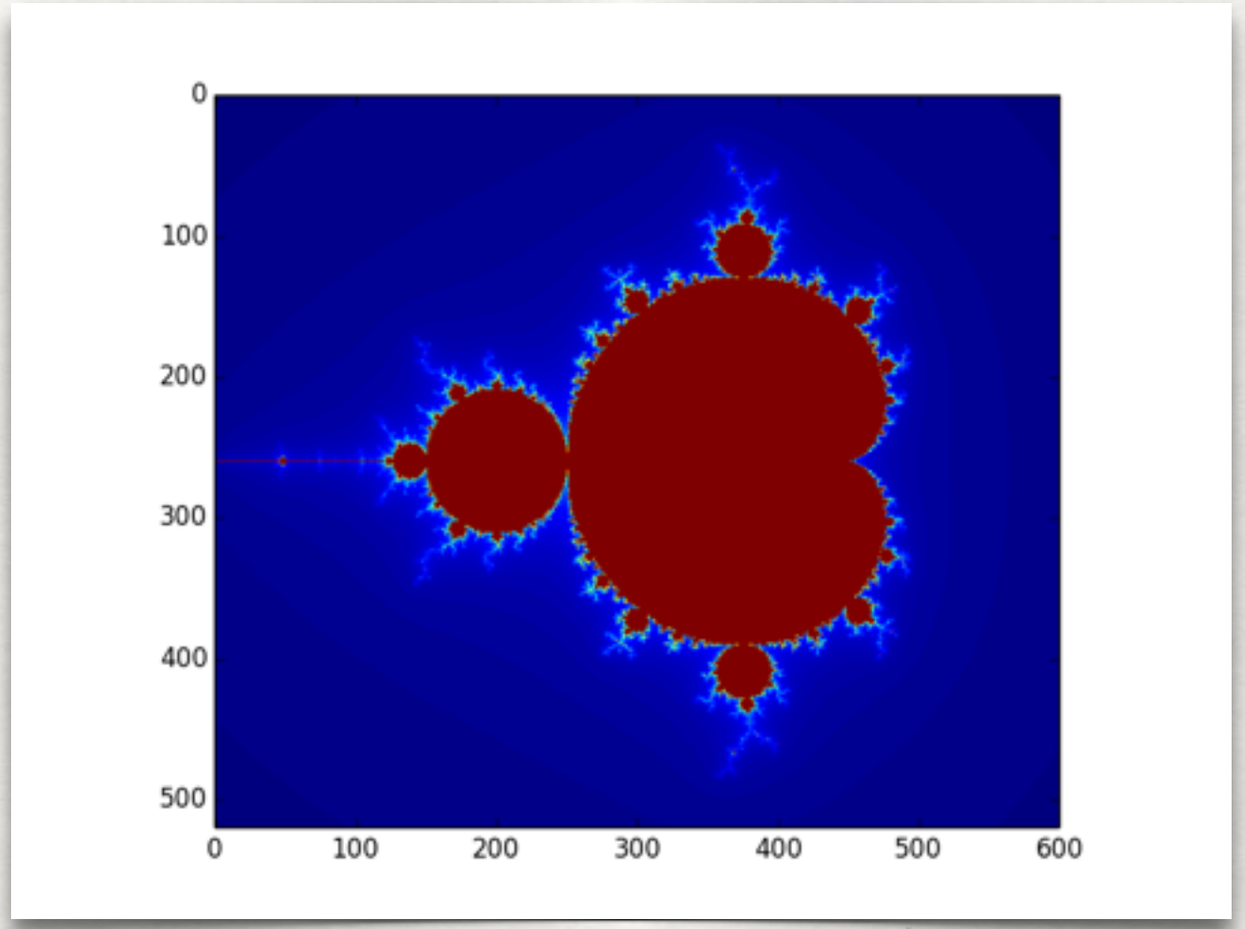
```
zs_ = zs*zs+c
not_diverged = tf.abs(zs_)<10
step = tf.group(zs.assign(zs_), ns.assign_add(tf.cast(not_diverged, tf.float32)))

for i in range(125) : step.run()
```



EXAMPLE 1: FRACTALS

- Evaluate the map and plot the output.

```
plt.imshow(ns.eval())
plt.show()
```

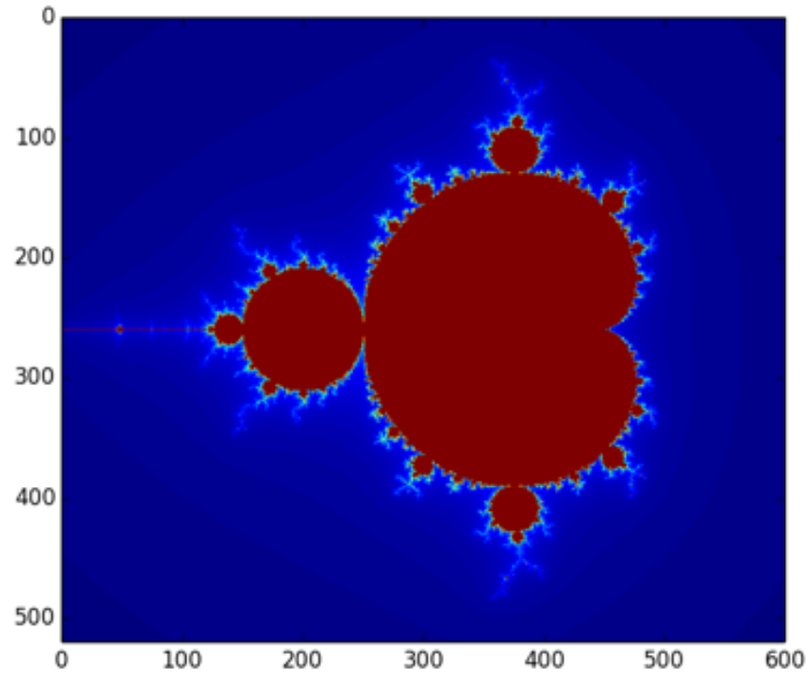


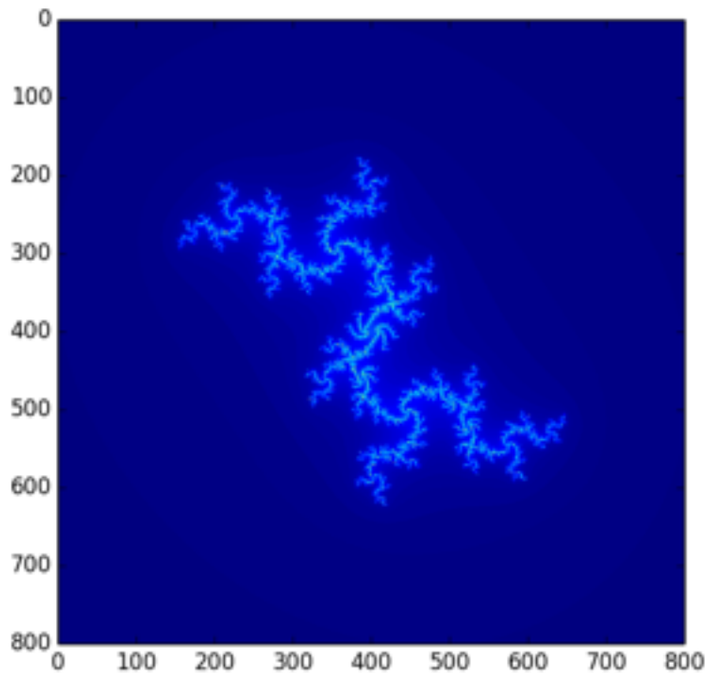
EXAMPLE 1: FRACTALS

- Very similar script, but using the relation.

$$z_{n+1} = z_n^2 - C$$

- Algorithm Logic:
  - Make grid of points
  - Define tensor object of the image map.
  - call InteractiveSession()
  - call initialize_all_variables().run()
  - setup the calculation
  - set up the loop
  - display results





SUGGESTED EXERCISE

- Take a look at the wolfram.com article on fractals:

  http://mathworld.wolfram.com/Fractal.html

- and select a pattern to compute, modifying one of the examples
  - e.g. Hénon Map

EXAMPLE 2: FISHER DISCRIMINANT

- Recall the fisher discriminant structure:

$$\begin{aligned} \mathcal{O} &= \sum_{i=1}^{n} \alpha_i x_i + \beta, \\ &= \underline{\alpha} \cdot \underline{x} + \beta. \end{aligned}$$

- where the α are defined in terms of the means of the class of events A and B, and the sum of covariance matrices W as:

$$\alpha \propto W^{-1}(\underline{\mu}_A - \underline{\mu}_B)$$

EXAMPLE 2: FISHER DISCRIMINANT

- Set up the problem by defining the number of events and the means and variances for x and y:

```
# set a 2D sample of N events to randomly generate
# multinormal distributions in order to compute a Fisher
# discriminant taking the weights as
#
#   alpha = W^-1 (mu[A] - mu[B])
#
# specify the normal means, widths below; assume
# no correlation
ndim = 2
N = 1000
muA = [1.0, 0.7]
muB = [-1.0, -1.0]
sigmaA = [0.6, 0.5]
sigmaB = [0.5, 0.4]

# use random_normal to generate the data
classA = tf.Variable(tf.random_normal([N, ndim], muA, sigmaA))
classB = tf.Variable(tf.random_normal([N, ndim], muB, sigmaB))
```

- tf.random_normal(shape, mean, var) is used to generate the data.
- Here shape is an array of N events (1000) times 2 dimensions (x and y).

EXAMPLE 2: FISHER DISCRIMINANT

- Initialise the variables and start running the session:

```
#initialize the variable
init_op = tf.global_variables_initializer()

print "================================================================="
print "Computing the coefficients of a Fisher discriminant for a toy"
print "example with "
print "muA = ", muA
print "muB = ", muB
print "sigmaA = ", muA
print "sigmaB = ", muB
print "(no correlation between x and y)"
print ""
print "A sample of", N, "examples has been generaated"
print "================================================================="

#run the session
with tf.Session() as sess:
    sess.run(init_op) #execute init_op
```

- After this point we then compute the means and covariances of the data, and then we can compute the fisher coefficients.

EXAMPLE 2: FISHER DISCRIMINANT

- Means and covariances of A (B is similar)

```
# class A
    print "Mean and covariance for class A"
    meanA, varA = tf.nn.moments(classA, axes=[0])
    meanAx, meanAy = sess.run(meanA)
    varAx, varAy = sess.run(varA)
    # define the covariance matrix for the data in class A
    SigmaA = tf.constant([[varAx, 0], [0, varAy]])
    print (sess.run(meanA))
    print (sess.run(SigmaA))
    print ""
```

use `tf.nn.moments` to compute the mean and variance of the data; and construct the covariance matrix for these data (uncorrelated x and y).

To print out some tf constant or variable you need to

`print (sess.run(myVariableOrConstant))`

In order to compute the alphas we can multiply matrices; this only works because Winv is diagonal; *warning: the matrix multiplication only works with tensors of dimension >=2; so one would have to generalise using long hand manipulation of the tensors to introduce correlations*.

- Compute the **α**

```
print "The sum of covariances for classes A and B (W)"
W = tf.add(SigmaA, SigmaB)
print (sess.run(W))
print ""

print "The inverse of W"
Winv = tf.matrix_inverse(W)
print (sess.run(Winv))
print ""

print "The difference between the class means"
DeltaMu = tf.subtract(meanA, meanB)
print (sess.run(DeltaMu))
print ""

print "=============================================================="
print "The fisher coefficients are (up to some arbitrary scaling)"
print "the diagonals of the matrix printed below"
alpha = tf.multiply(Winv, DeltaMu)
print (sess.run(alpha))
print "=============================================================="
```

See the TensorFlow website for details of the functionality:

`tf.matrix_inverse`
`tf.add`
`tf.subbtract`
`tf.multiply`

EXAMPLE 2: FISHER DISCRIMINANT

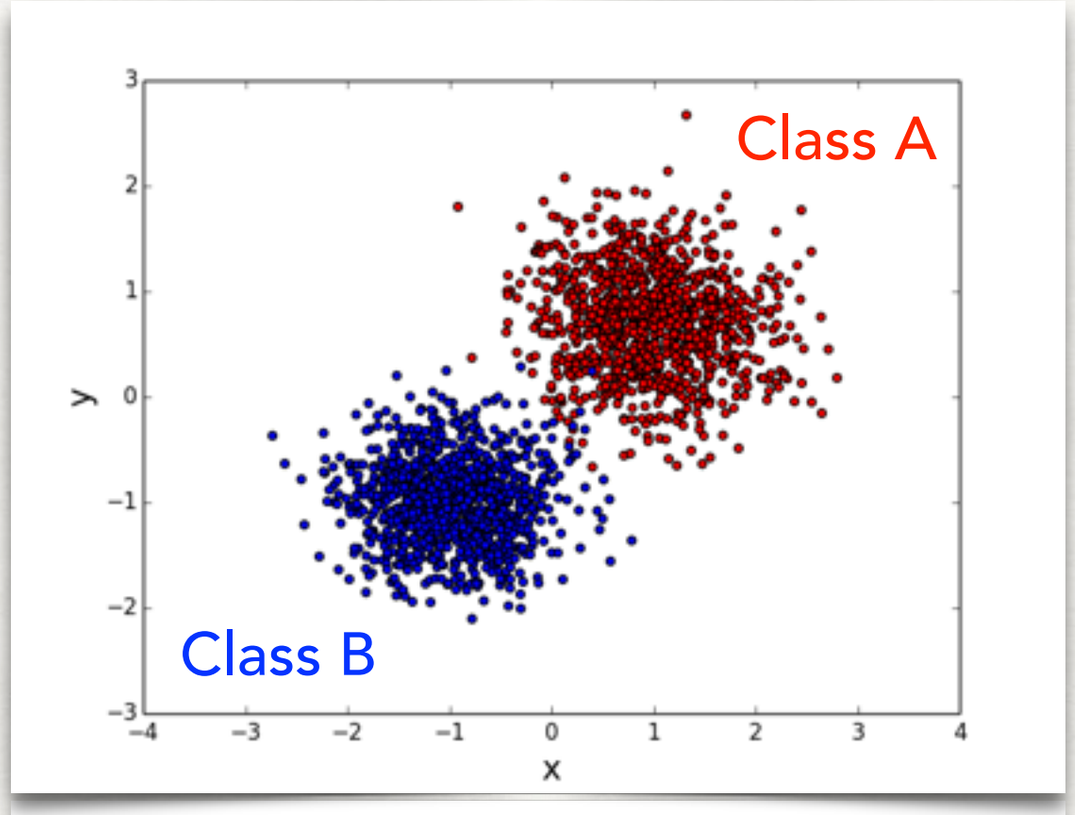
- With the inputs

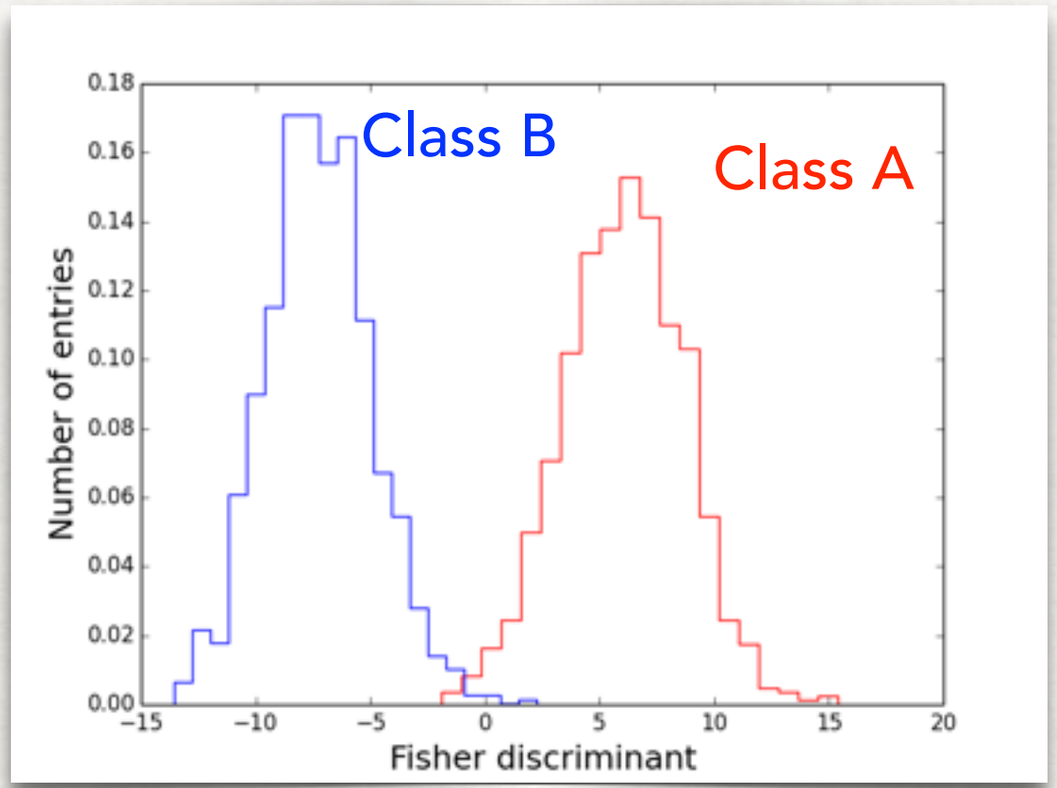
```
ndim = 2
N = 1000
muA = [1.0, 0.7]
muB = [-1.0, -1.0]
sigmaA = [0.6, 0.5]
sigmaB = [0.5, 0.4]
```

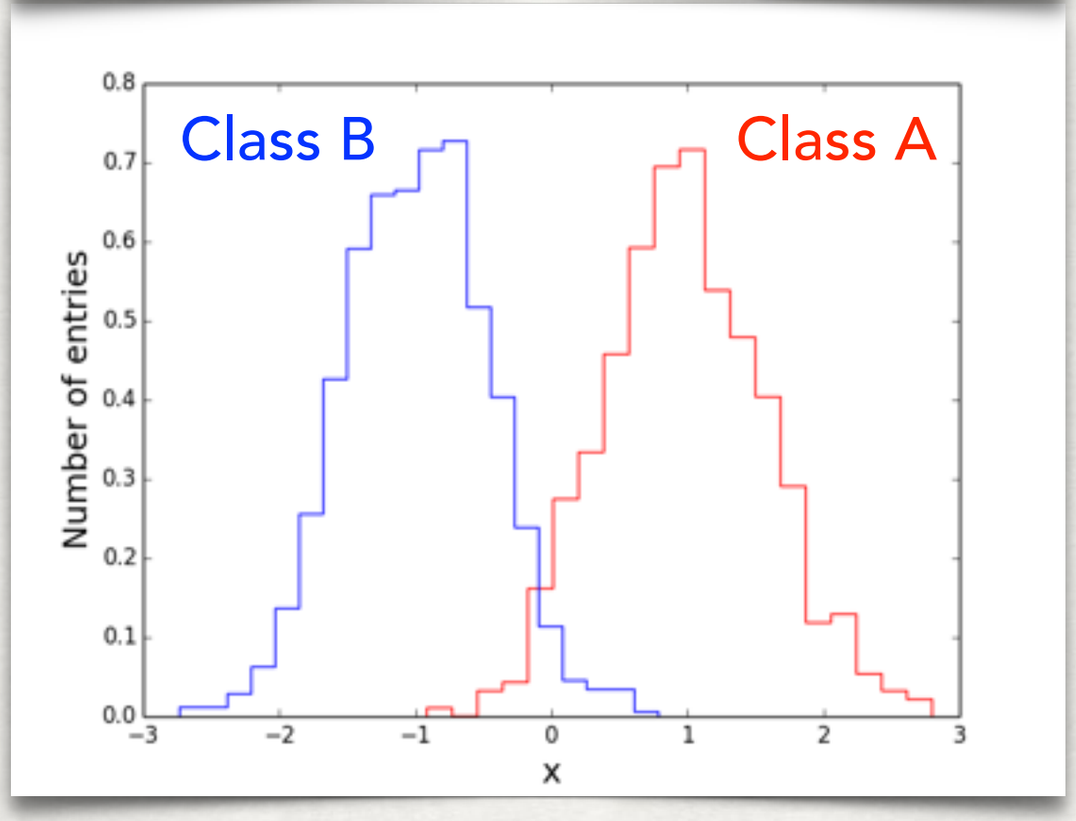
- The coefficients computed from running Fisher.py are
  - **α**=[3.25, 4.00]

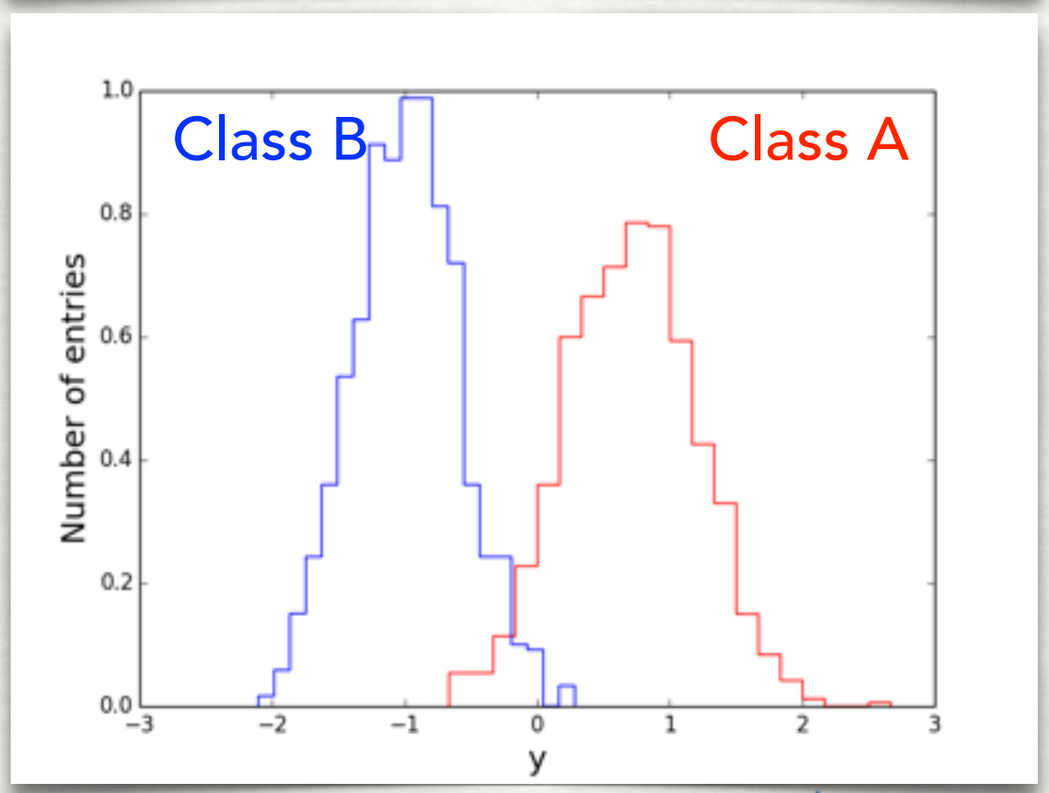
Hint: `print (sess.run(myVar[index]))` will print out the value of the tf variable with that index. If you have an ND tensor then that tensor needs to be accessed with N indices in square brackets.

EXAMPLE 2: FISHER DISCRIMINANT









Queen Mary University of London

SUGGESTED EXERCISE

- Extend the script to plot the data, projections and compute the fisher discriminant corresponding to the generated sample.

  See FisherPlot.py for hints.

- The fisher discriminant example coded up here only works for uncorrelated data. You might wish to consider extending this to the case where the data are correlated in x and y. To do this you will need to consider:
  - Adding a correlation between x and y at the generation stage.
  - Computing the covariance matrices to ensure they are not diagonal.
  - Computing the fisher coefficients not using the `tf.multiply` function as that is an element wise product and not a matrix multiplication. The issue here is that `tf.matmul` does not accept a rank 1 tensor as the second argument; so that will require a different approach.

SUGGESTED EXERCISE

- The fisher discriminant example coded up here only works for uncorrelated data. You might wish to consider extending this to the case where the data are correlated in x and y. To do this you will need to consider:
  - Adding a correlation between x and y at the generation stage.
  - Computing the covariance matrices to ensure they are not diagonal.
  - Computing the fisher coefficients not using the `tf.multiply` function as that is an element wise product and not a matrix multiplication. The issue here is that `tf.matmul` does not accept a rank 1 tensor as the second argument; so that will require a different approach.

EXAMPLE 3: PERCEPTRON

- The data used for this example requires input_data.py

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================

"""Functions for downloading and reading MNIST data."""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import gzip
import os
import tempfile

import numpy
from six.moves import urllib
from six.moves import xrange  # pylint: disable=redefined-builtin
import tensorflow as tf
from tensorflow.contrib.learn.python.learn.datasets.mnist import read_data_sets
```

EXAMPLE 3: PERCEPTRON

- The data used for this example requires input_data.py

- Provides a convenient interface to download MNIST handwriting training examples, prepared in the form of a 28 x 28 pixel image.

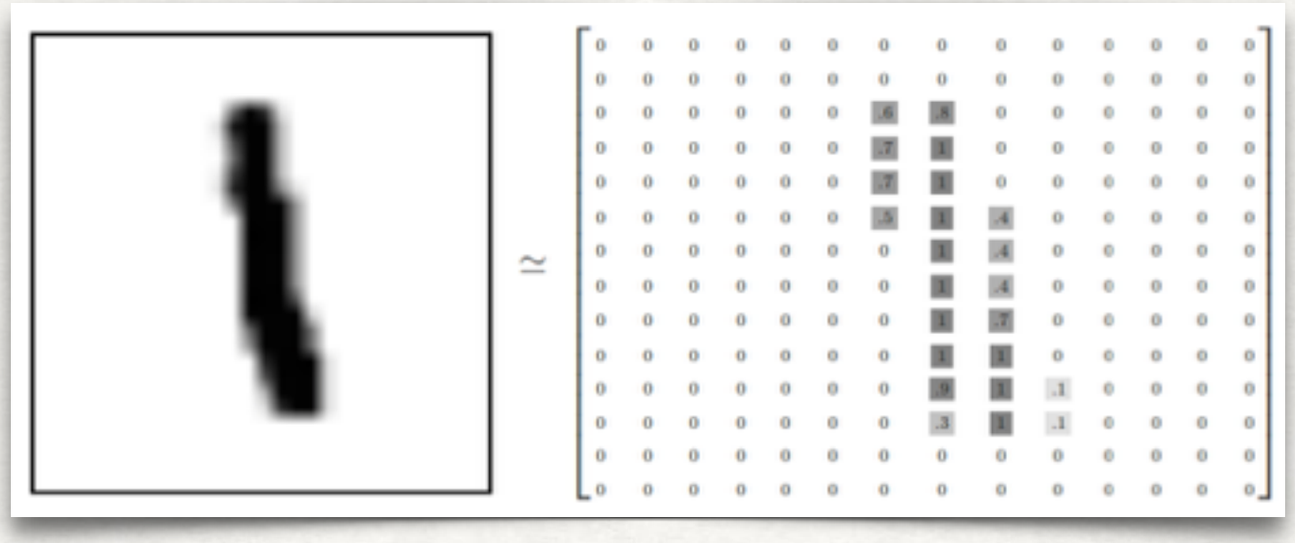


Illustration of an MNIST example represented as a set of features in a 14 x 14 pixel map; from https://www.tensorflow.org/tutorials/mnist/beginners/ ; The MNIST data uses a 28 x 28 pixel representation.

784 dimensional feature space (the pixels) → Perceptron → Output

EXAMPLE 3: PERCEPTRON

- We use a soft max activation function in this example.

$$y_i = \frac{e^{x^T w}}{\sum_{k=1}^{K} e^{x^T w}}$$

(Similar to a sigmoid function in shape)

- The loss function used here is called cross entropy*

$$-\sum_{k=1}^{K} \widehat{P}_{mk} \log \widehat{P}_{mk},$$

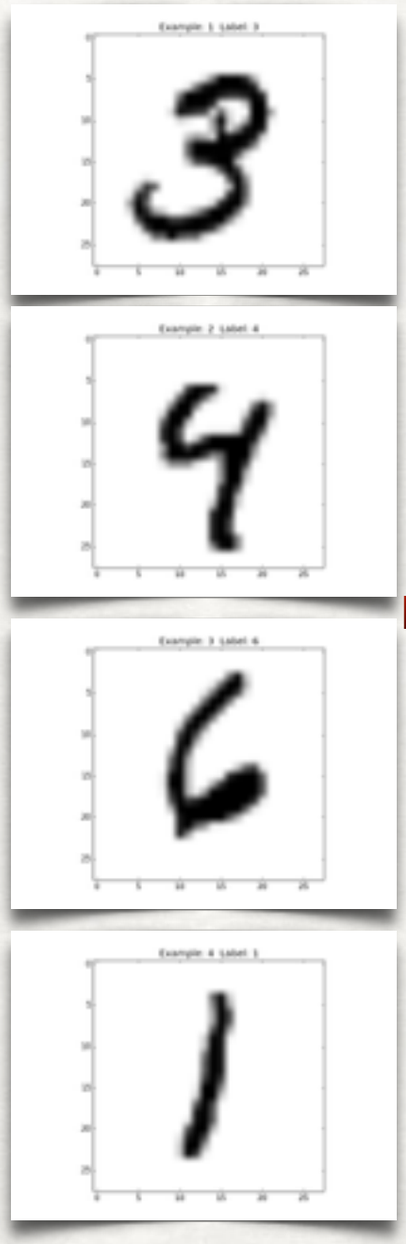
$$\widehat{P}_{mk} = \frac{1}{N} \sum_{x_i \in R} I(y_i = k)$$

k is the class index (K classes in total)
m is the node index
R is a region of data with N observations
p-hat is the proportion of class k observations in node m.

*In the SDA lectures we encounter a loss function of the form $(t_i - y_i)^2/2$; this is the l2_loss function in TensorFlow.

EXAMPLE 3: PERCEPTRON

- Perceptron takes each image as a 784 dimensional feature space and computes the evidence for each of the possible output classes.



$\underline{x}_i=[784]_i$

OUTPUT NODE COMPUTES EVIDENCE FOR EACH OF THE 10 POSSIBLE LABELS

EXAMPLE 3: PERCEPTRON

- As with the previous examples the flow of the script is as follows:
  1. Set up the problem
     - Set up the data
     - Set up the constants and variables required
     - Set up the model:
       - activation function
       - cost = loss function
       - optimiser (minimisation algorithm)
  2. Call `tf.global_variables_initializer()`
  3. "Run the session"
  4. Process the results

EXAMPLE 3: PERCEPTRON

1. Set up the problem

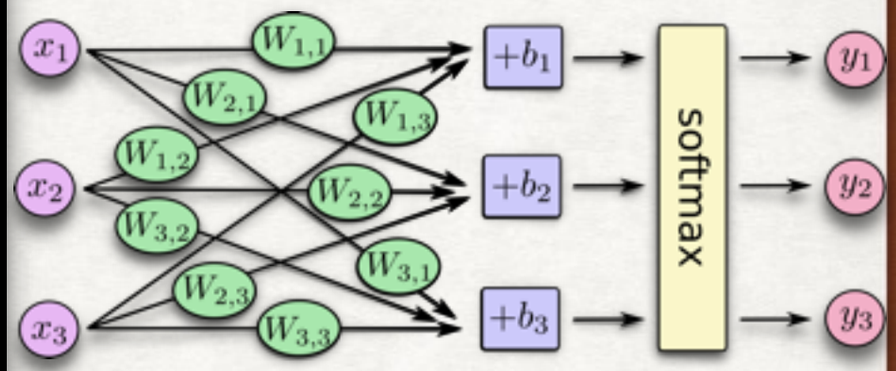
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import input_data

print "Importing the MNIST data"
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

# Network training parameters
learning_rate = 0.01
training_epochs = 25
batch_size = 100
display_step = 1

# the MNIST image data are arrays of 28x28 pixels, and the set
# has 10 target classification types (the numbers zero through
# nine).
img_size_x = 28
img_size_y = 28
img_size = img_size_x*img_size_y
NUM_CLASSES = 10

print "Parameters set for this optimisation"
print "\tLearning Rate     = ", learning_rate
print "\tEpochs            = ", training_epochs
print "\tBatch Size        = ", batch_size
print "\tDisplay Step      = ", display_step
print "\tNumber of classes = ", NUM_CLASSES
```

EXAMPLE 3: PERCEPTRON

1. Set up the problem

```
x = tf.placeholder(tf.float32, [None, img_size])
y = tf.placeholder(tf.float32, [None, NUM_CLASSES])

# weights and bias
W = tf.Variable(tf.zeros([img_size, NUM_CLASSES]))
b = tf.Variable(tf.zeros([NUM_CLASSES]))

print x
print W
print b

print "Setting up the model"
# construct the model; here we use the softmax activation function
activation = tf.nn.softmax(tf.matmul(x, W) + b)

print "Setting up the optimizer"
# minimise error using cross entropy as the loss function
cross_entropy = y*tf.log(activation)
cost = tf.reduce_mean ( -tf.reduce_sum( cross_entropy, reduction_indices=1 ) )

# set the hyper parameter optimisation algorithm. This is where the cost is
# minimised using the specified learning rate.
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Plot variables; store these in order to summarise the information
# once the network is trained.
avg_set = []
epoch_set = []
```

The x_i are the features (pixels), the $w_{i,j}$ are the weights and the b_j are the biases for the outputs of the softmax activation function:



This is equivalent to:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

(Images from www.tensorflow.org)

EXAMPLE 3: PERCEPTRON

2. Call `tf.global_variables_initializer()`

```
# initilise the global variables in order to run the session.
init = tf.global_variables_initializer()
```

This call is required after setting up the variables and constants needed for a calculation.

By executing this function an op (init) is returned that is used to initialise global variables.

EXAMPLE 3: PERCEPTRON

3. “Run the session”

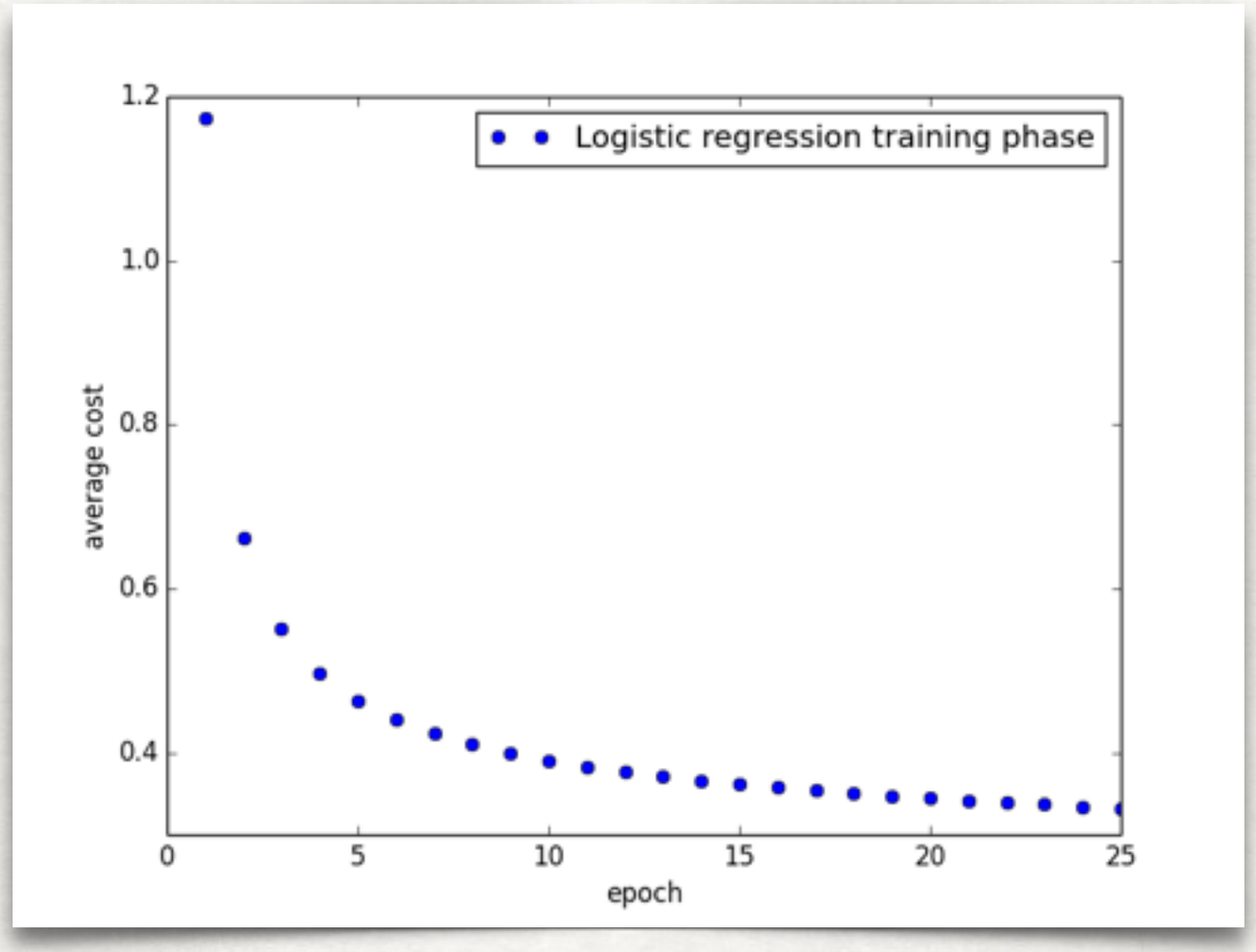
```
print "Running the session"
with tf.Session() as sess:
    sess.run(init)
    for epoch in range( training_epochs ) :
        avg_cost = 0.0
        total_batch = int(mnist.train.num_examples/batch_size)

        # loop over batches
        for i in range (total_batch):
            batch_xs, batch_ys = mnist.train.next_batch( batch_size )
            sess.run( optimizer, feed_dict={x: batch_xs, y: batch_ys} )
            avg_cost += sess.run( cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch

# append information for plotting for every display_step epochs
        if epoch % display_step == 0:
#            print "Epoch: ", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)
            avg_set.append(avg_cost)
            epoch_set.append(epoch+1)

    correct_prediction = tf.equal( tf.argmax(activation, 1), tf.argmax(y,1) )

    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float32"))
    print "Model accuracy:", accuracy.eval( {x: mnist.test.images, y:mnist.test.labels})
```

EXAMPLE 3: PERCEPTRON

4. Process the results

```
print "Training phase finished"
plt.plot(epoch_set, avg_set, 'o', label='Logistic regression training phase')
plt.xlabel('epoch')
plt.ylabel('average cost')
plt.legend()
plt.show()
```

Plot the cost (loss function) as a function of epoch using the data accumulated during training.



SUGGESTED EXERCISE

- Modify the training parameters to see how this affects the output performance and compute time; e.g. accuracy of prediction and loss function evolution with training epoch.
- Note: if you set batch_size to a small number the training may take a while.
- After doing the TensorBoard exercise come back to this and try adding the following just before `sess.run(init)`:

```
merged = tf.merge_all_summaries()
writer = tf.train.SummaryWriter("/tmp/tensorflowlogs", sess.graph)
```

- If you run into problems then take a look at `PerceptronTB.py`.

EXAMPLE 4: MULTILAYER PERCEPTRON

- This example uses a sigmoid activation function:

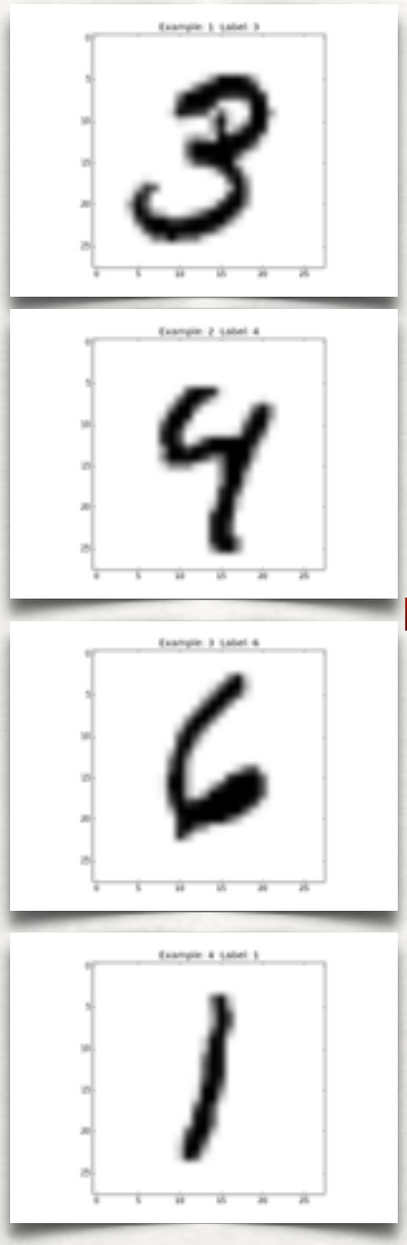
$$y_i = \frac{1}{1 + e^{w^T x + b}}$$

- The hyper parameter optimisation is performed using the AdamOptimizer; this is an adaptive optimisation algorithm that outperforms gradient descent.*
- The "softmax cross entropy" loss function is used (really this is the cross entropy loss function).
- 256 nodes in the first layer.
- 256 nodes in the second layer.
- 10 outputs values, each being the score for the example image for that output class.

* See https://www.tensorflow.org/api_guides/python/train for more details about TensorFlow parameter tuning options.

EXAMPLE 4: MULTILAYER PERCEPTRON

- Break MNIST images down into arrays of 784 features and process these with two layers of 256 perceptrons; collecting the result in an output node with 10 possible classifications:



$\underline{x}_i=[784]_i$

256 NODES IN LAYER 1

256 NODES IN LAYER 2

OUTPUT NODE COMPUTES EVIDENCE FOR EACH OF THE 10 POSSIBLE LABELS

EXAMPLE 4: MULTILAYER

- As with the previous examples the flow of the script is as follows:
  1. Set up the problem
     - Set up the data
     - Set up the constants and variables required
     - Set up the model:
       - activation function
       - cost = loss function
       - optimiser (minimisation algorithm)
  2. Call `tf.global_variables_initializer()`
  3. "Run the session"
  4. Process the results

EXAMPLE 4: MULTILAYER PERCEPTRON

1. Set up the problem

- Start essentially the same as with the perceptron example; but we now need to specifiy the number of nodes in each hidden layer.

```
n_hidden_1 = 256 # 1st layer num features
n_hidden_2 = 256 # 2nd layer num features
```

- Layer 1:

```
# We construct layer 1 from a weight set, a bias set and the activiation function used
# to process the impulse set of features for a given example in order to produce a
# predictive output for that example.
#
#  w_layer_1:    the weights for layer 1.  The first index is the input feature (pixel)
#                and the second index is the node index for the perceptron in the first
#                layer.
#  bias_layer_1: the biases for layer 1.  There is a single bias for each node in the
#                layer.
#  layer_1:      the activation functions for layer 1
#
w_layer_1 = tf.Variable(tf.random_normal([n_input, n_hidden_1]))
bias_layer_1 = tf.Variable(tf.random_normal([n_hidden_1]))
# The mnist beginners example uses a softmax activation function; here we will
# use a sigmoid function instead.
layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x,w_layer_1),bias_layer_1))
```

EXAMPLE 4: MULTILAYER PERCEPTRON

- Layer 2:

```
# We construct layer 2 in an analogous way; now we have a second set of weights,
# biases and activation functions.  These are named similarly to the first set.
# the changes are:
#   1) the input number of features to the second layer, corresponds to the number
#      of nodes in the first
#   2) layer 1 is used in the matrix multiplication when constructing layer 2
# as opposed to the data placeholder x.
#
#  w_layer_2:    the weights for layer 2.  The first index is the input feature (pixel)
#                and the second index is the node index for the perceptron in the first
#                layer.
#  bias_layer_2: the biases for layer 2.  There is a single bias for each node in the
#                layer.
#  layer_2:      the activation functions for layer 2
#
w = tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2]))
bias_layer_2 = tf.Variable(tf.random_normal([n_hidden_2]))
# The mnist beginners example uses a softmax activation function; here we will
# use a sigmoid function instead.
layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1,w),bias_layer_2))
```

- Output:

```
# Similarly we now construct the output of the network, where the output layer
# combines the information down into a space of evidences for the 10 possible
# classes in the problem.
output = tf.Variable(tf.random_normal([n_hidden_2, n_classes]))
bias_output = tf.Variable(tf.random_normal([n_classes]))
output_layer = tf.matmul(layer_2, output) + bias_output
```

EXAMPLE 4: MULTILAYER PERCEPTRON

2. Call `tf.global_variables_initializer()`

```
# initilise the global variables in order to run the session.
init = tf.global_variables_initializer()
```

This call is required after setting up the variables and constants needed for a calculation.

By executing this function an op (init) is returned that is used to initialise global variables.

EXAMPLE 4: MULTILAYER PERCEPTRON

3. "Run the session"

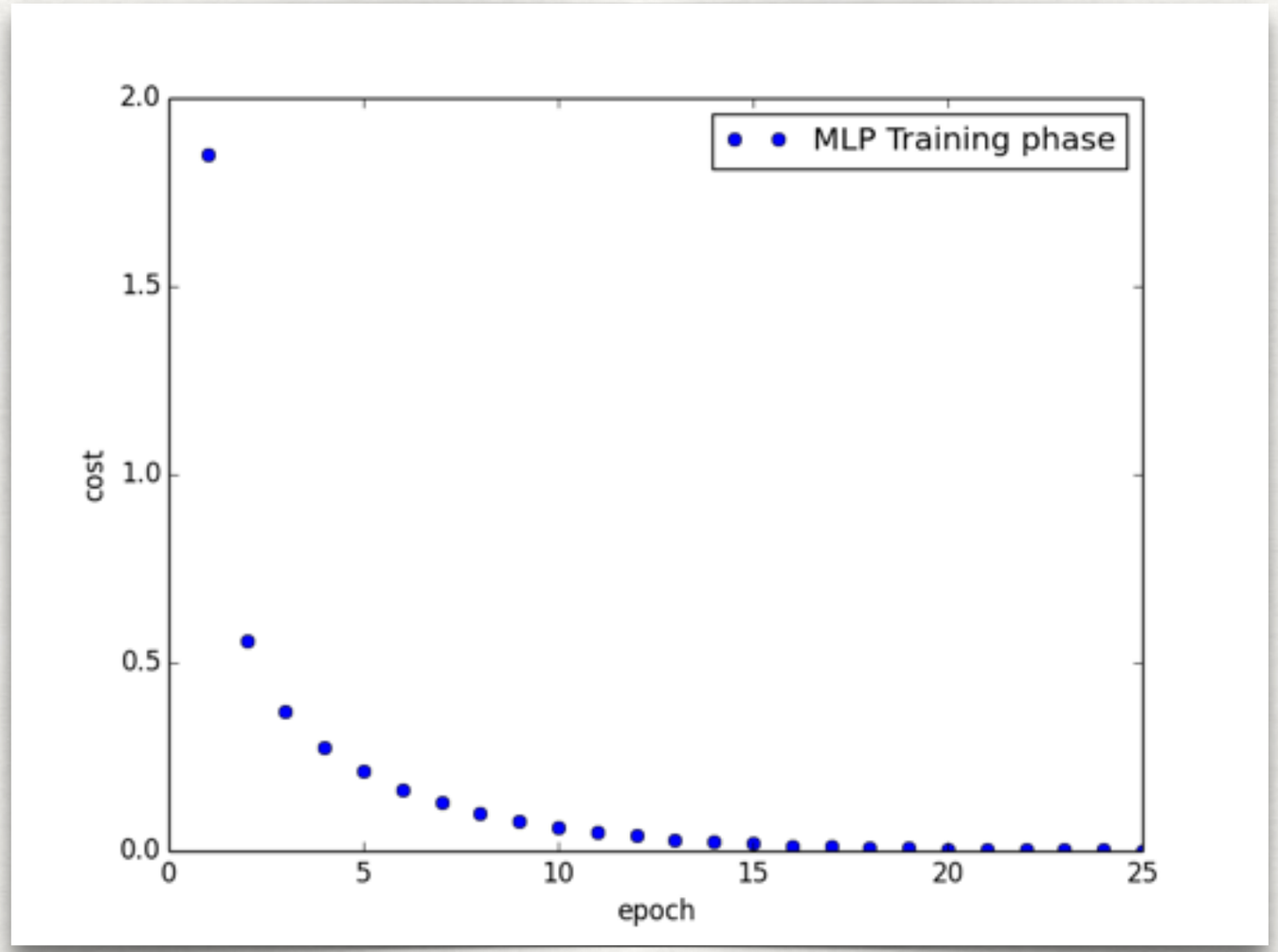
```
# Start the session to embark on the training cycle
with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)

        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch

        # Display progress for each epoch; and store cost per epoch
        if epoch % display_step == 0:
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)
            avg_set.append(avg_cost)
            epoch_set.append(epoch+1)

    # Test model
    correct_prediction = tf.equal(tf.argmax(output_layer, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print "Model Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})
```

EXAMPLE 4: MULTILAYER PERCEPTRON

4. Process the results

```
Epoch: 0001 cost= 1.849095951
Epoch: 0002 cost= 0.557972253
Epoch: 0003 cost= 0.372533757
Epoch: 0004 cost= 0.273199379
Epoch: 0005 cost= 0.210435390
Epoch: 0006 cost= 0.163848224
Epoch: 0007 cost= 0.128521473
Epoch: 0008 cost= 0.101588030
Epoch: 0009 cost= 0.080406694
Epoch: 0010 cost= 0.063262567
Epoch: 0011 cost= 0.050079947
Epoch: 0012 cost= 0.039589912
Epoch: 0013 cost= 0.031024283
Epoch: 0014 cost= 0.024418883
Epoch: 0015 cost= 0.019006889
Epoch: 0016 cost= 0.014566056
Epoch: 0017 cost= 0.011515973
Epoch: 0018 cost= 0.008995460
Epoch: 0019 cost= 0.006957878
Epoch: 0020 cost= 0.005391021
Epoch: 0021 cost= 0.004251376
Epoch: 0022 cost= 0.003413825
Epoch: 0023 cost= 0.002687842
Epoch: 0024 cost= 0.002175575
Epoch: 0025 cost= 0.001699322
Model Accuracy: 0.9492
```

These data are stored in the arrays epoch_set and avg_set in order to create the plot shown.



SUGGESTED EXERCISES

- Take a look at the neural network options on the TensorFlow website: https://www.tensorflow.org/api_docs/python/tf/nn
  - Try changing the activation function type to tanh, relu or softmax and see how the accuracy of the model is affected by the choice of activation function.

    https://www.tensorflow.org/api_guides/python/nn#Activation_Functions

- Try changing the optmisation algorithm from the AdamOptimizer to GradientDescentOptimiser, the AdagradOptimiser, or one of the other available options and study what happens.

    https://www.tensorflow.org/api_guides/python/train#Optimizers

EXAMPLE 5: USING TENSOR BOARD

- Read the start of the script to get an understanding of how to use this tool:

```
#!/usr/bin/python
#======================================================================
# Example using TensorBoard to illustrate models.
#
# First run this script in order to perform the computation a+b*2; then having done that
# run tensorboard on the binary file output, specifying the output log directory
#
#   tensorboard --logdir=/tmp/tensorflowlogs/
#
# This will provide a webpage location to browse the data flow graph for this calculation
#======================================================================
```

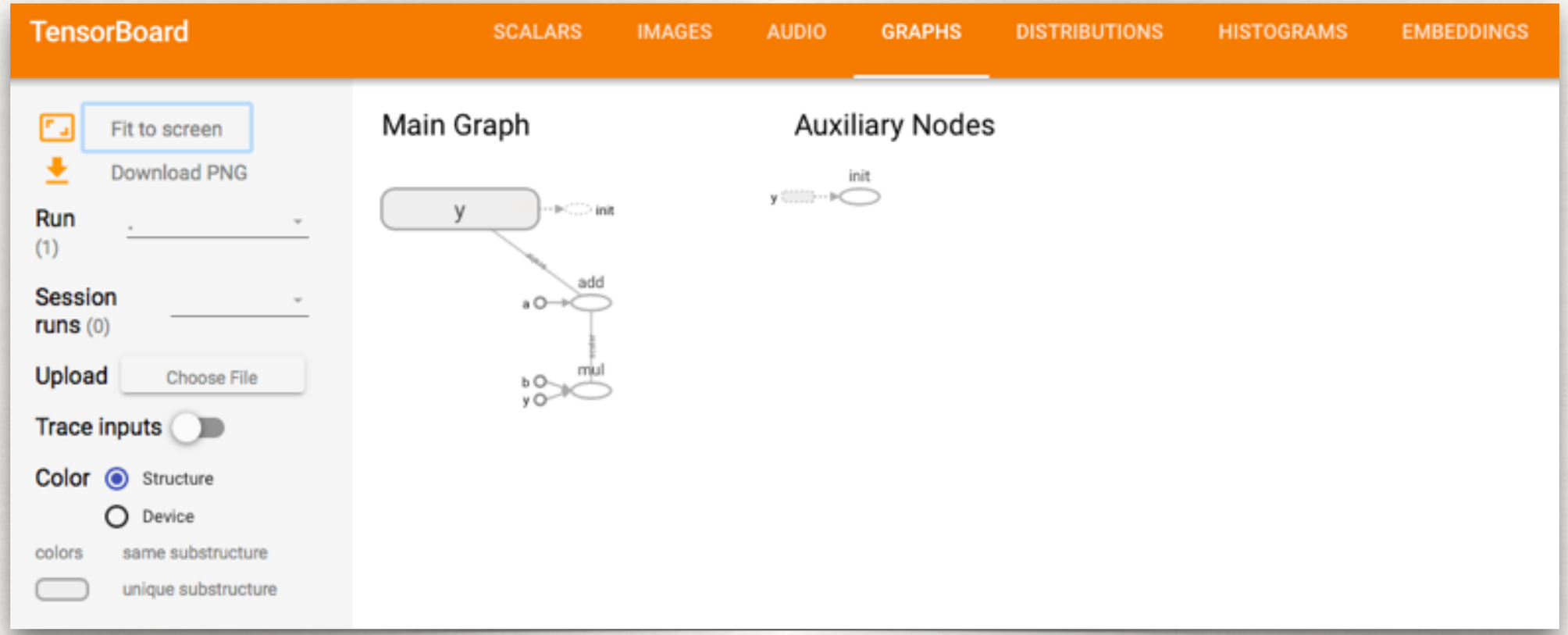
- 1) Run the script `./TensorBoard.py`
- 2) Run tensorboard using:

  `tensorboard --logdir=/tmp/tensorflowlogs/`

- 3) Open the URL given on the terminal

EXAMPLE 5: USING TENSOR BOARD

- For this example only the graphs are interesting:

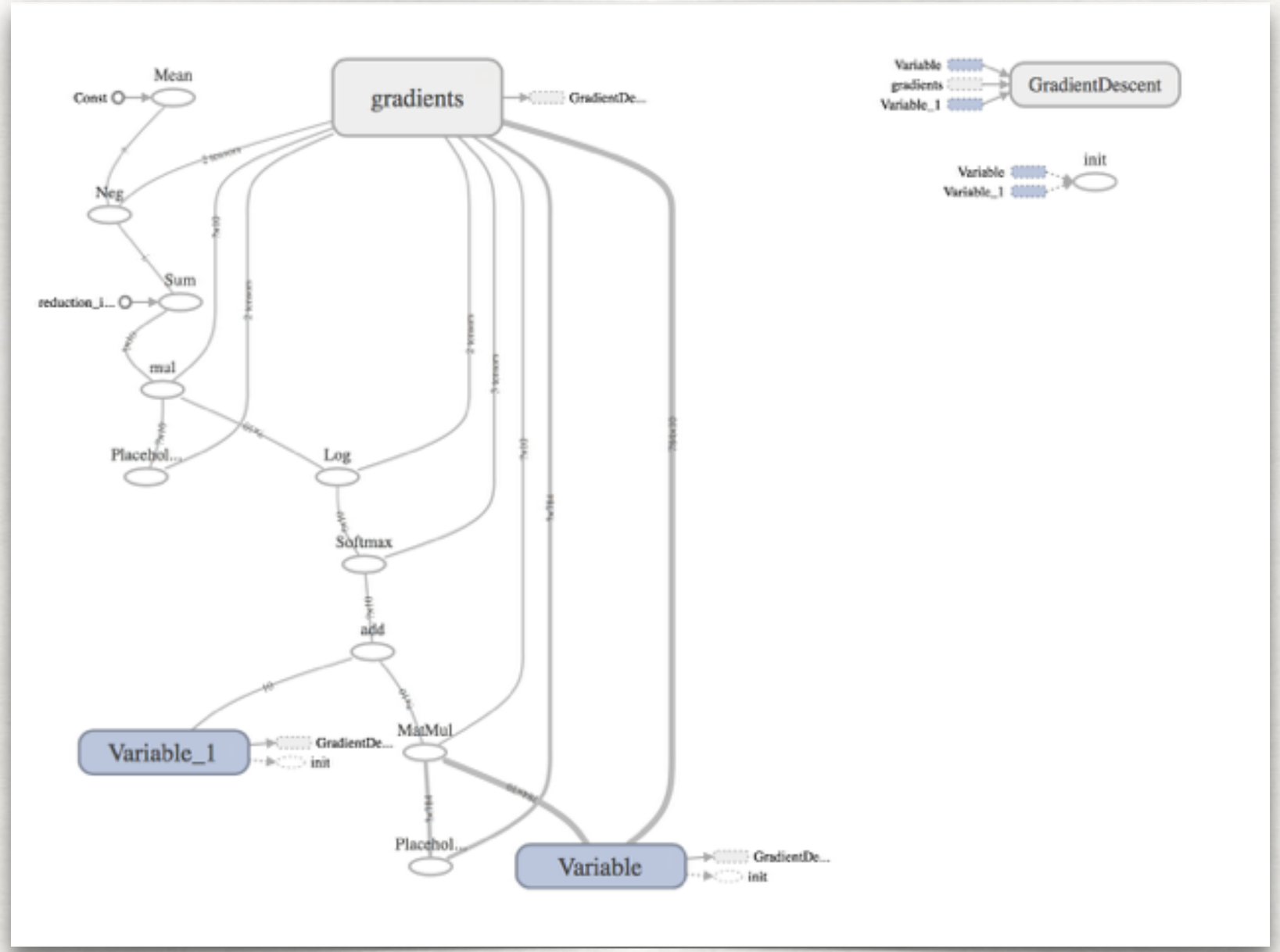


- Compare this output with the constants and variable defined in the example script; and against the output obtained.

- Take a look at https://www.tensorflow.org/how_tos/summaries_and_tensorboard/ for more information on this tool.

EXAMPLE 5: USING TENSOR BOARD

- A more interesting graph to look at is the PerceptronTB.py example.



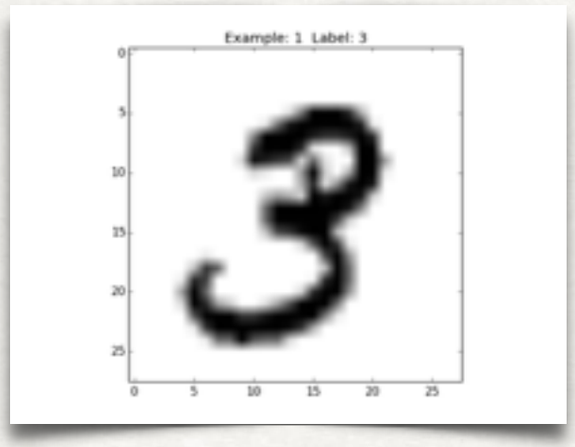
- Try clicking on the different parts of the model.

SUGGESTED EXERCISE

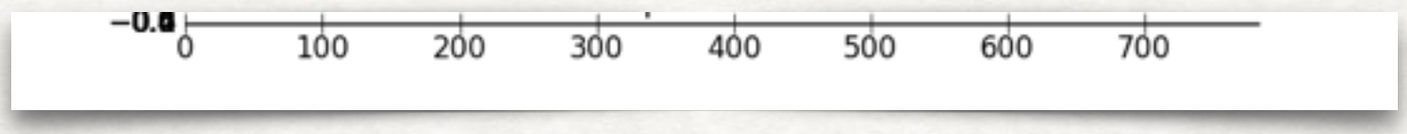
- Adapt some of the other scripts to use TensorBoard to inspect the graphs of other models.
- Take a look at the tensorboard webpage: https://www.tensorflow.org/how_tos/summaries_and_tensorboard/ for an example of using this tool with a neural network; and run through that example.

Example 6: Convolutional Neural Networks

- The previous examples of classifying MNIST data flatten the image structure into a 784 dimensional feature space.
  - This results in loss of information associated between different parts of the image.



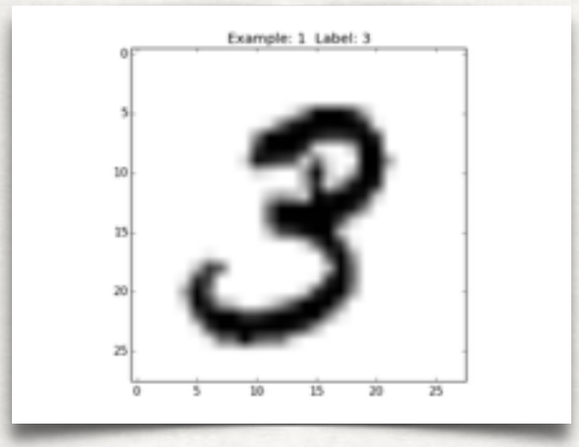
$\underline{x}_i=[784]_i$

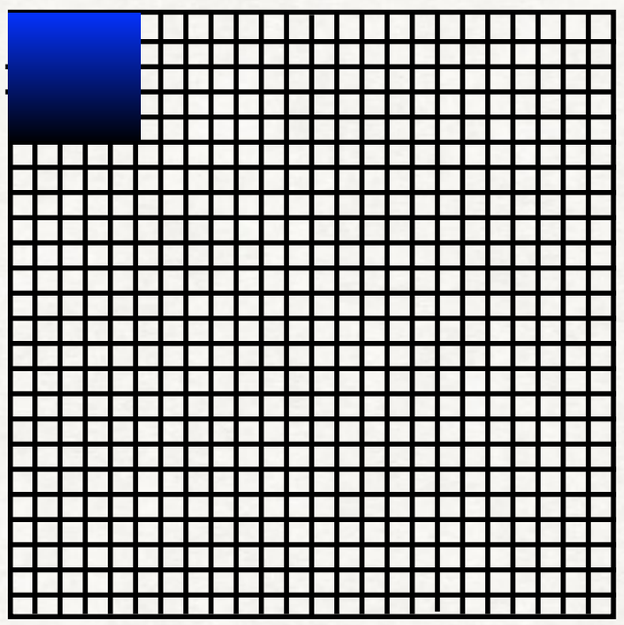


- A Convolutional Neural Network (CNN) uses the spatial correlations in the image array as well as the content of each pixel to compute evidence for a given outcome.
- This is a minor adaptation of the example found at:
  - https://www.tensorflow.org/get_started/mnist/pros

Example 6: Convolutional Neural Networks

- The previous examples of classifying MNIST data flatten the image structure into a 784 dimensional feature space.
  - This results in loss of information associated between different parts of the image.





Analyse the image using a filter that processes a finite region of space; e.g. a 5x5 pixel sample.

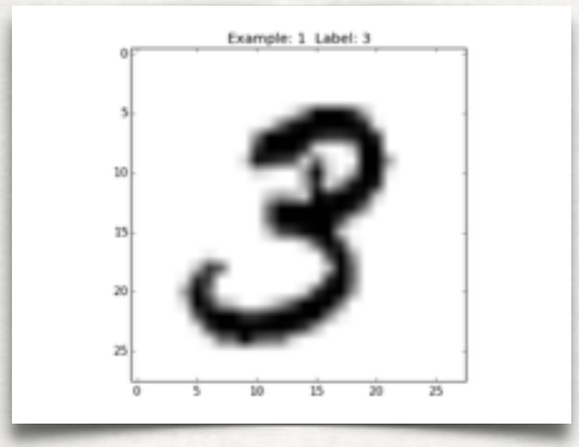
This is the convolutional part of the network.

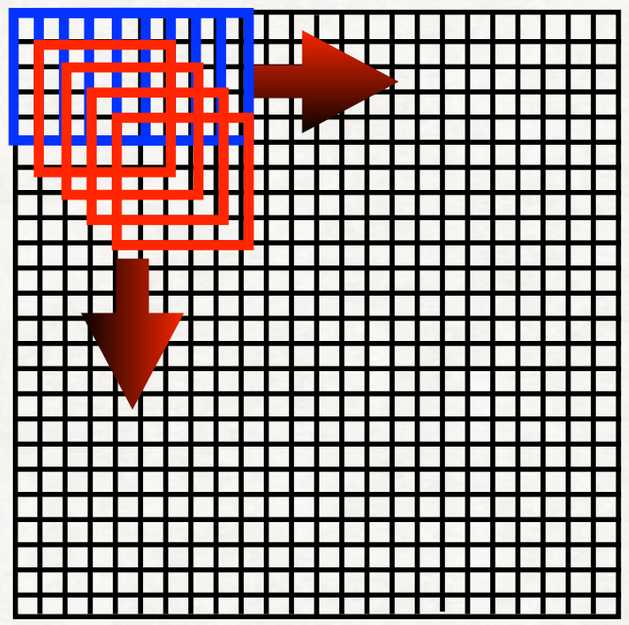
Move the filter over the image.

- A Convolutional Neural Network (CNN) uses the spatial correlations in the image array as well as the content of each pixel to compute evidence for a given outcome.

Example 6: Convolutional Neural Networks

- The previous examples of classifying MNIST data flatten the image structure into a 784 dimensional feature space.
  - This results in loss of information associated between different parts of the image.





Analyse the image using a filter that processes a finite region of space; e.g. a 5x5 pixel sample.

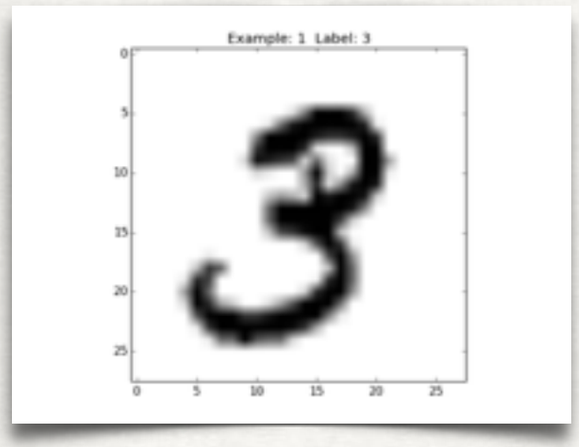
This is the convolutional part of the network.

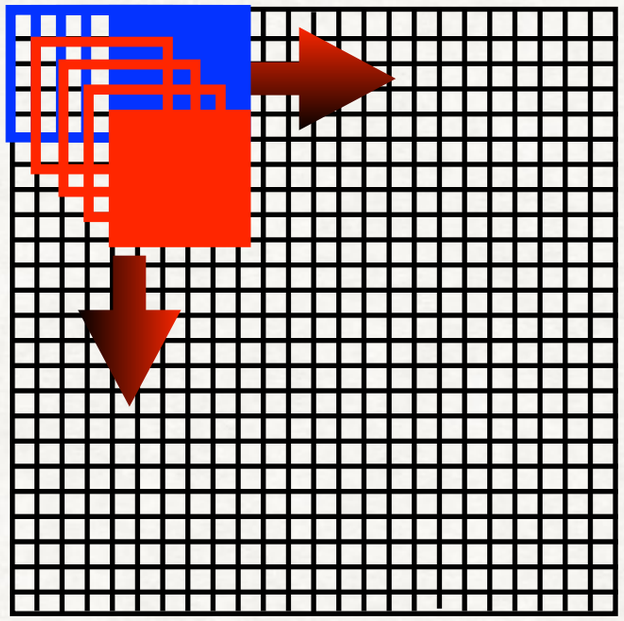
Move the filter over the image.

- A Convolutional Neural Network (CNN) uses the spatial correlations in the image array as well as the content of each pixel to compute evidence for a given outcome.

Example 6: Convolutional Neural Networks

- The previous examples of classifying MNIST data flatten the image structure into a 784 dimensional feature space.
  - This results in loss of information associated between different parts of the image.





Analyse the image using a filter that processes a finite region of space; e.g. a 5x5 pixel sample.

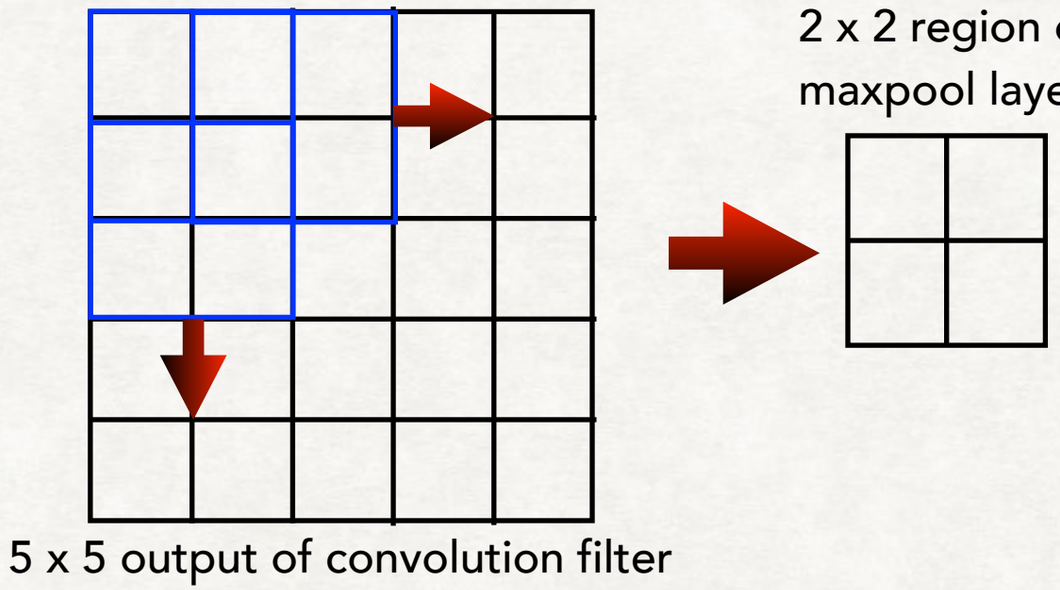
This is the convolutional part of the network.

Move the filter over the image.

- For a 28 pixel wide image; a 5x5 filter can be applied 24=(28-5+1) times in a unique position across, or down one side.
- 576 ways to apply this sized filter to the MNIST image data.

Example 6: Convolutional Neural Networks

- The previous examples of classifying MNIST data flatten the image structure into a 784 dimensional feature space.
  - This results in loss of information associated between different parts of the image.



Take the output of a convolutional layer node and maxpool this with a 2 x 2 filter.

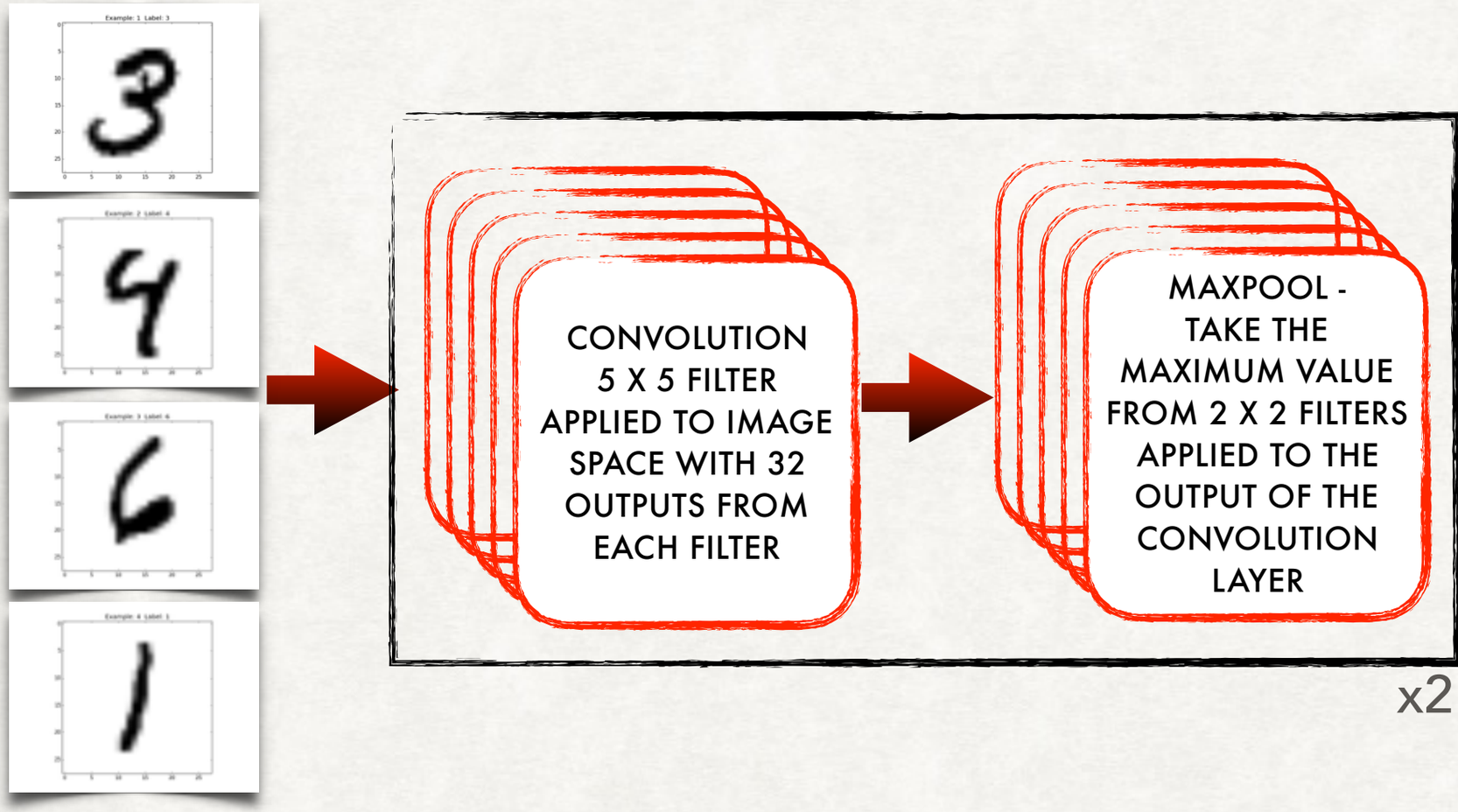
This means take the maximum value in that given array of pixels.

Move the filter over the image.

- For a 28 pixel wide image; a 5x5 filter can be applied 24=(28-5+1) times in a unique position across, or down one side.
- 576 ways to apply this sized filter to the MNIST image data.

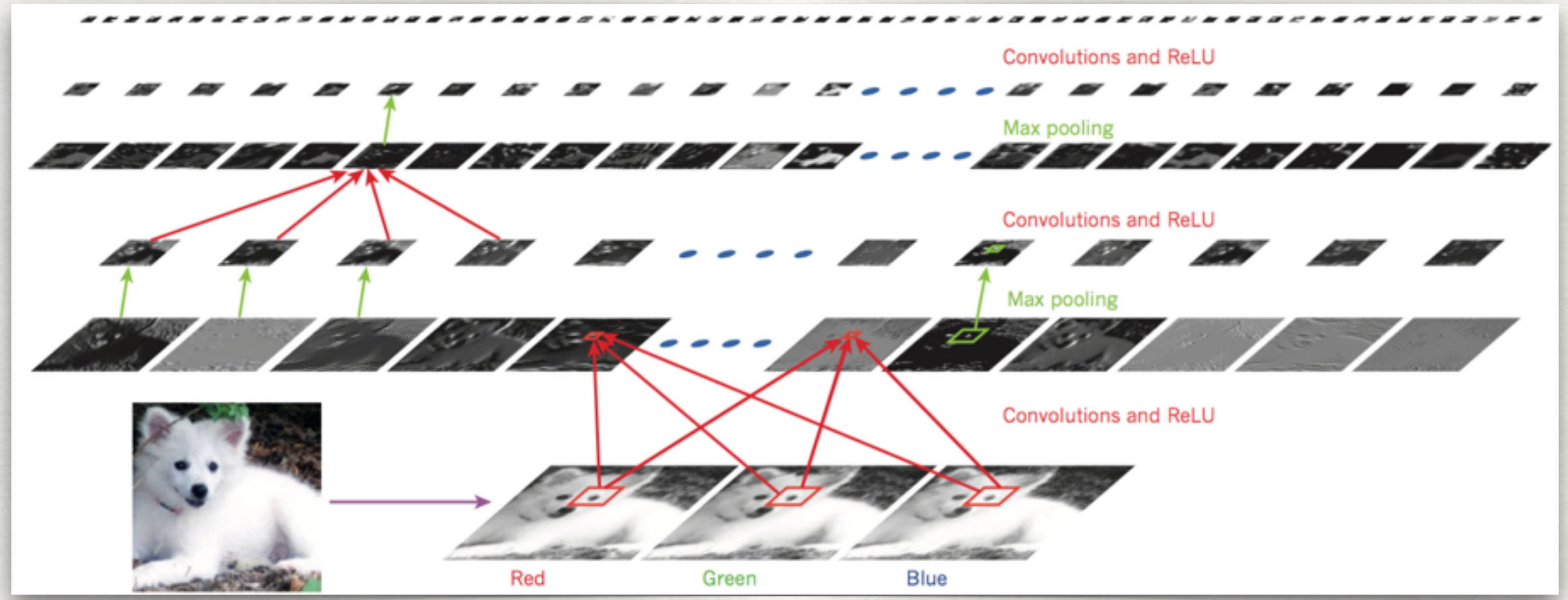
Example 6: Convolutional Neural Networks

- Each filtered image is replicated in order to be analysed in conjunction with the neighbouring images.



Example 6: Convolutional Neural Networks

- Conceptully the MNIST images are passed through different convolutional layers and maxpool layers to determine the score for a given output of the network.



- See for example LeCun et al. doi:10.1038/nature14539 and references therein.

Example 6: Convolutional Neural Networks

- We initialise the weights and biases to be non-zero and wrap their creation in functions with:

```
# function to create weights with a randomly chosen initialisation.
# this is important for the large number of hyperparameters we have in
# the CNN; and in particular for the relu activation function.
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

# function to create biases with a randomly chosen initialisation.
# this is important for the large number of hyperparameters we have in
# the CNN; and in particular for the relu activation function.
def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
```

- The truncated normal is a Gaussian distribution with tails removed (nothing beyond 2 sigma). The weights are generated according to this core Gaussian distribution.
- The biases are set to 0.1 as a starting value (small positive number).

Example 6: Convolutional Neural Networks

- The conv2d function is described at:
  - https://www.tensorflow.org/api_docs/python/tf/nn/conv2d

```
# Create a 2D convolutional layer
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

- Use this to create the convolutional layer.

Example 6: Convolutional Neural Networks

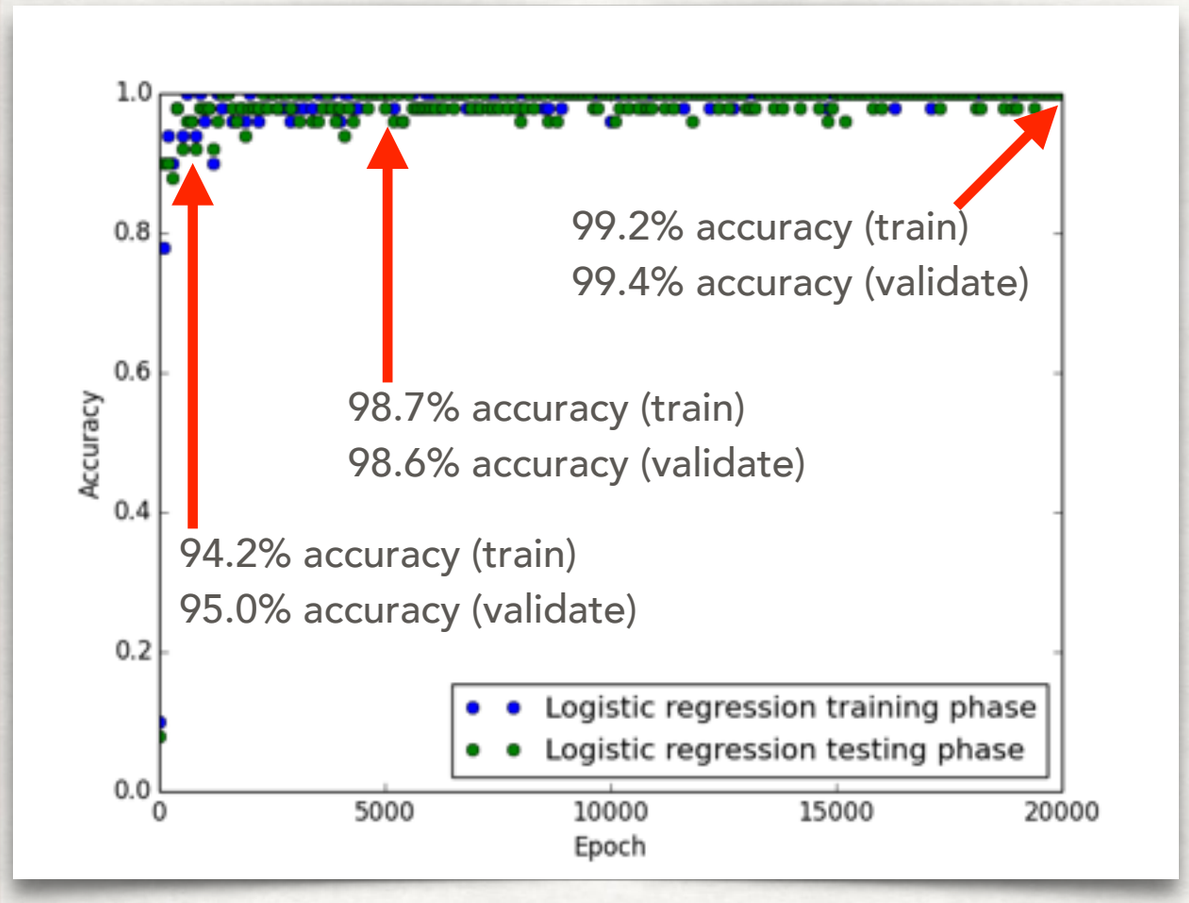
- Maxpooling:
  - https://www.tensorflow.org/api_guides/python/nn#Pooling

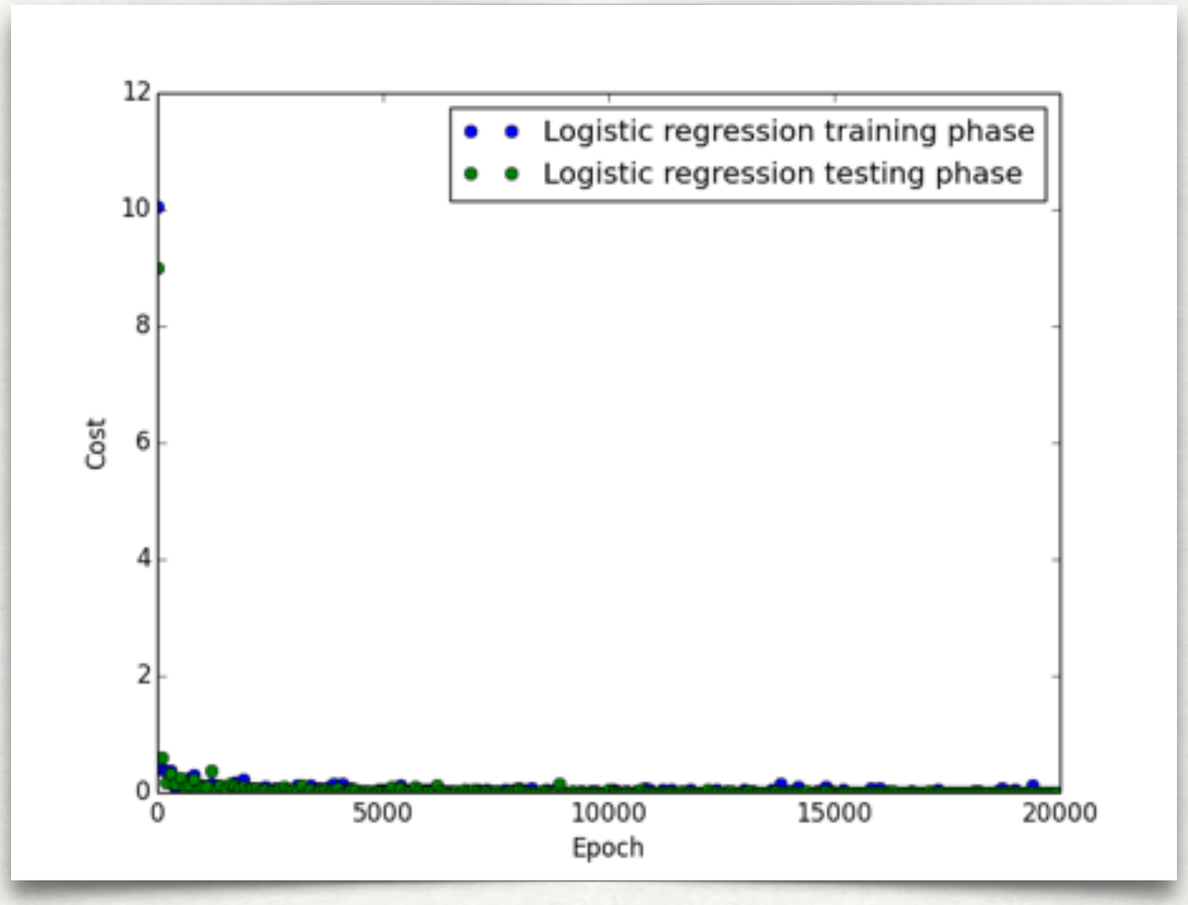
```
# create a max pooling kernel: 2 x 2
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

- Apply a small filter to reduce the number of hyperparameters in a convolutional network. Aim is to reduce tendency for the network to overfit.
- Examples of how to extract information from the network with smaller lost input can be found:
  - https://arxiv.org/abs/1412.6071
  - https://arxiv.org/abs/1412.6806

Example 6: Convolutional Neural Networks

- The CNN hyper parameters quickly get trained to a sufficient level to outperform an MLP on this data; and over time reach >99% accuracy.





WANT MORE DATA?

- The MNIST data is a particular sample that is useful to explore image pattern recognition problems.

- scikit learn has data sets to use (see http://scikit-learn.org/stable/) e.g. Fisher's iris data: http://scikit-learn.org/stable/tutorial/basic/tutorial.html

- For particle physics related problems this is not so relevant (although the experience you get from image pattern recognition is useful).

- You can obtain ATLAS data for the decay channel $H \to \tau^+\tau^-$ from: https://www.kaggle.com/c/higgs-boson

- This provides a rich feature space with data in a csv file format for you to analyse. More information on the problem of searching for this decay with that data can be found at: https://higgsml.lal.in2p3.fr/documentation/

WANT MORE INFORMATION?

- In addition to the various links provided in this file, there are books on the use of tensor flow available… and on order for the library; still reading through those so not in a position to judge..

- However lots of good information online to build on the examples provided here.

- If you want a more formal background in machine learning to follow up on at your own pace then good starting points are:
  - Bishop “Pattern Recognition and Machine Learning”, Springer.
  - Hastie, Tibshirani, Friedman “The Elements of Statistical Learning”, Springer.