



# Deep Learning with Python and Friends

## Douglas Starnes

## CodeStock 2018

Polyglot ninja

Memphis, TN area

Co-director of MemPy

Data Science/Machine Learning, Mobile Apps,  
wannabe Game Designer

Pluralsight Author

@poweredbyaltnet



@poweredbyaltnet

Artificial Intelligence

Machine Learning

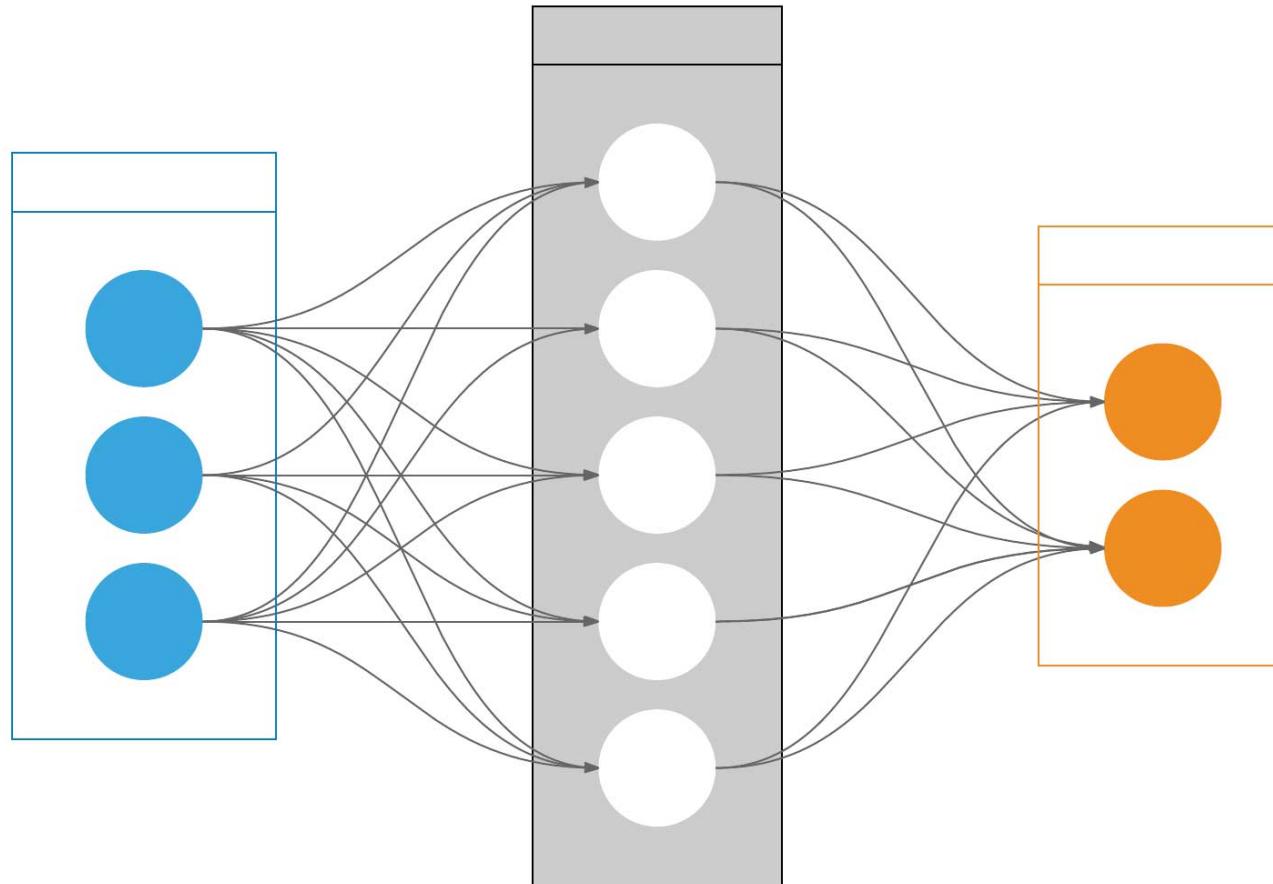
Deep Learning

D  
a  
t  
a  
  
S  
c  
i  
e  
n  
c  
e

@poweredbyaltnet

# NEURAL NETWORK

@poweredbyaltnet



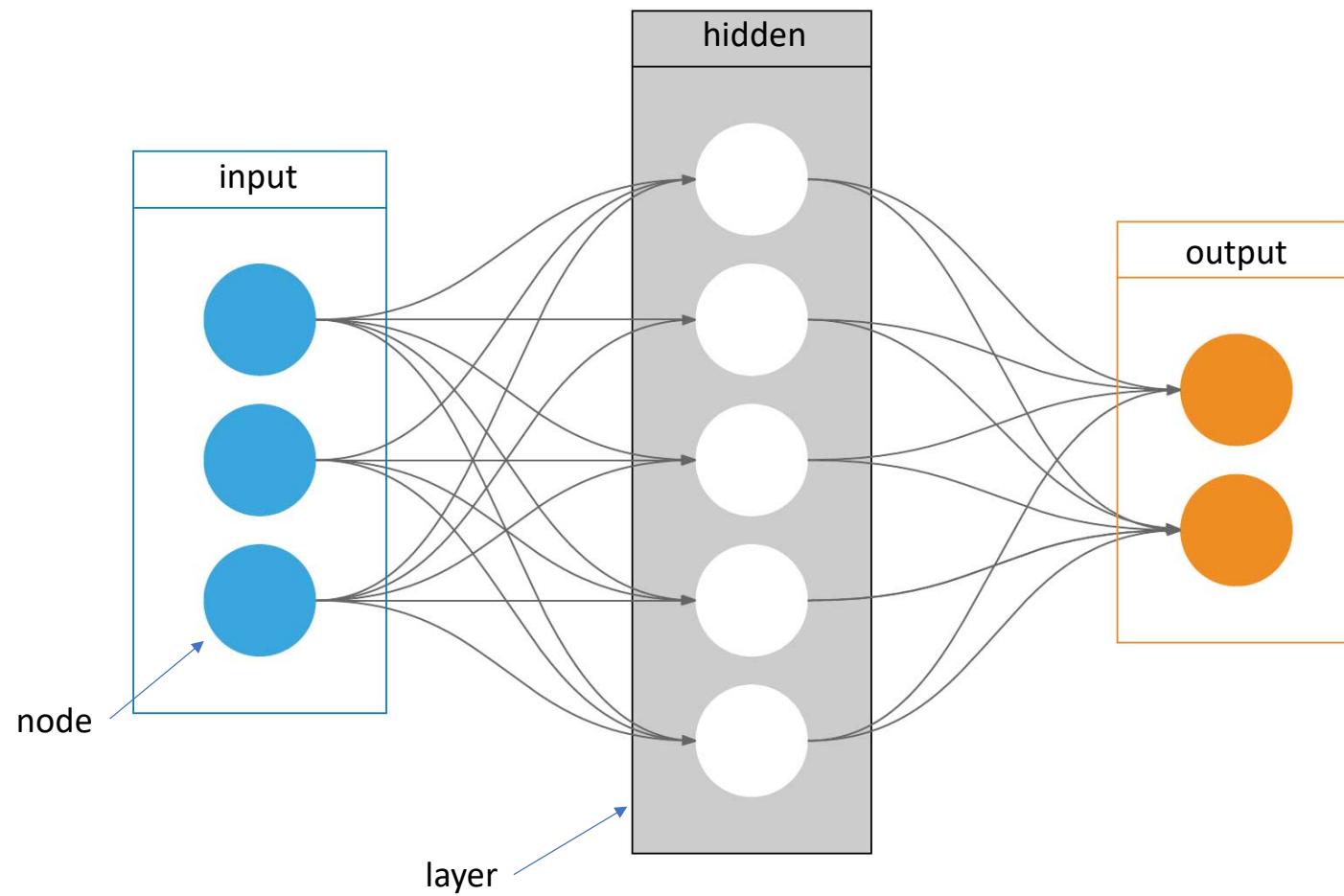
@poweredbyaltnet

Neural networks are loosely inspired by how neuroscientists today think the brain might possibly work.

*(assuming they are on the right track)*

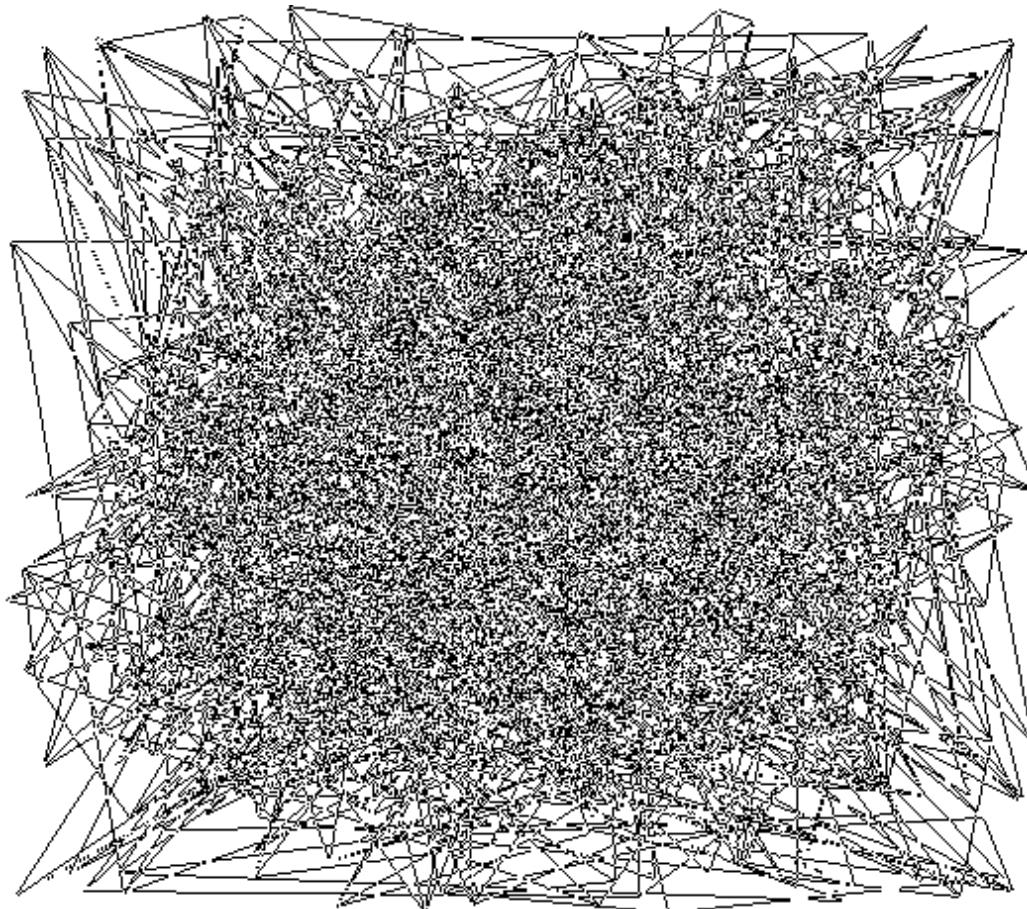


@poweredbyaltnet



@poweredbyaltnet

A neural network with more than two hidden layers is called a **deep** neural network.



@poweredbyaltnet

feature vector				target
Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa
...	...	...	...	...
7.0	3.2	4.7	1.4	versicolor
...	...	...	...	...
5.7	2.8	4.1	1.3	virginica

rows -> observations

columns -> features

feature vector / independent variables -> observed values

target / dependent variable(s) -> what we want to predict

train a **model** that, given a **feature vector**, will (hopefully accurately) assign a **target**

train a **model** that, given values for *Sepal Length*, *Sepal Width*, *Petal Length*, *Petal Width*, will assign one value from the set of *setosa*, *versicolor*, or *virginica*



@poweredbyaltnet



Is this  
prediction  
correct?



@poweredbyaltnet

During training, we are not concerned with making a correct prediction, but rather making a prediction that is as close to correct as possible.



WT\_?

@poweredbyalt.net

Prediction	Actual
.55	1.0
.30	0
.15	0



I think we  
can do  
better!

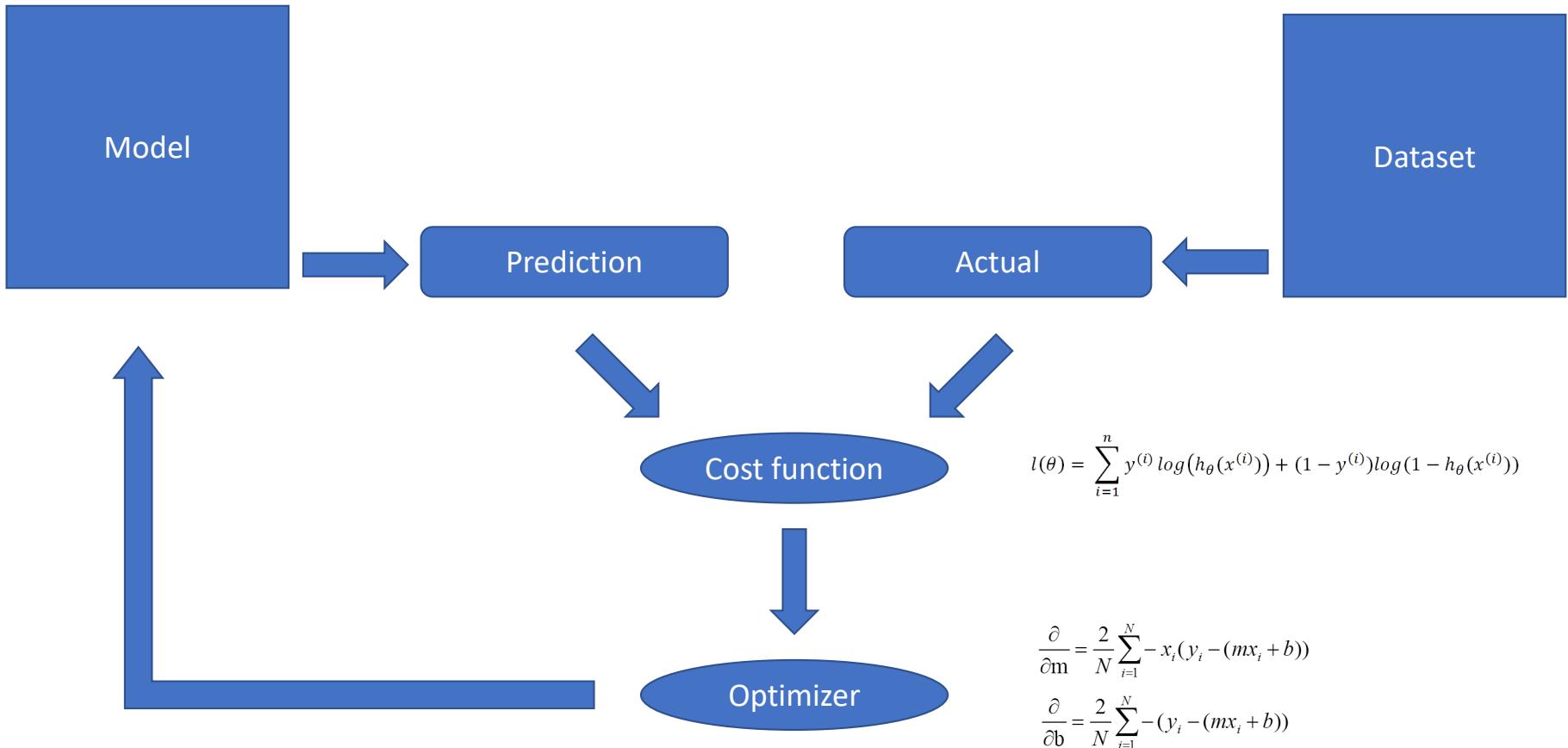
@poweredbyaltnet

Prediction	Actual
.9999999	1.0
.0000001	0
.0000001	0

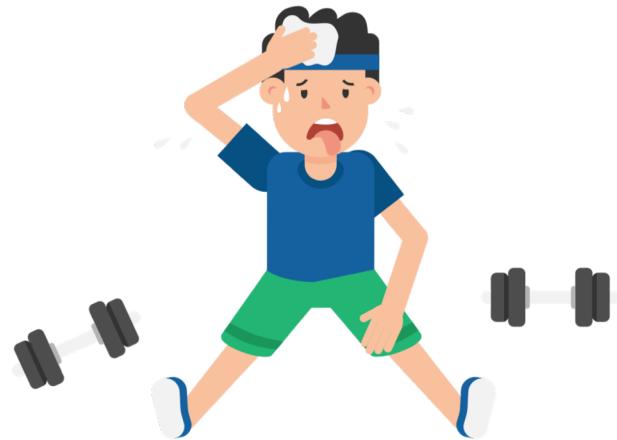


More 9's please!

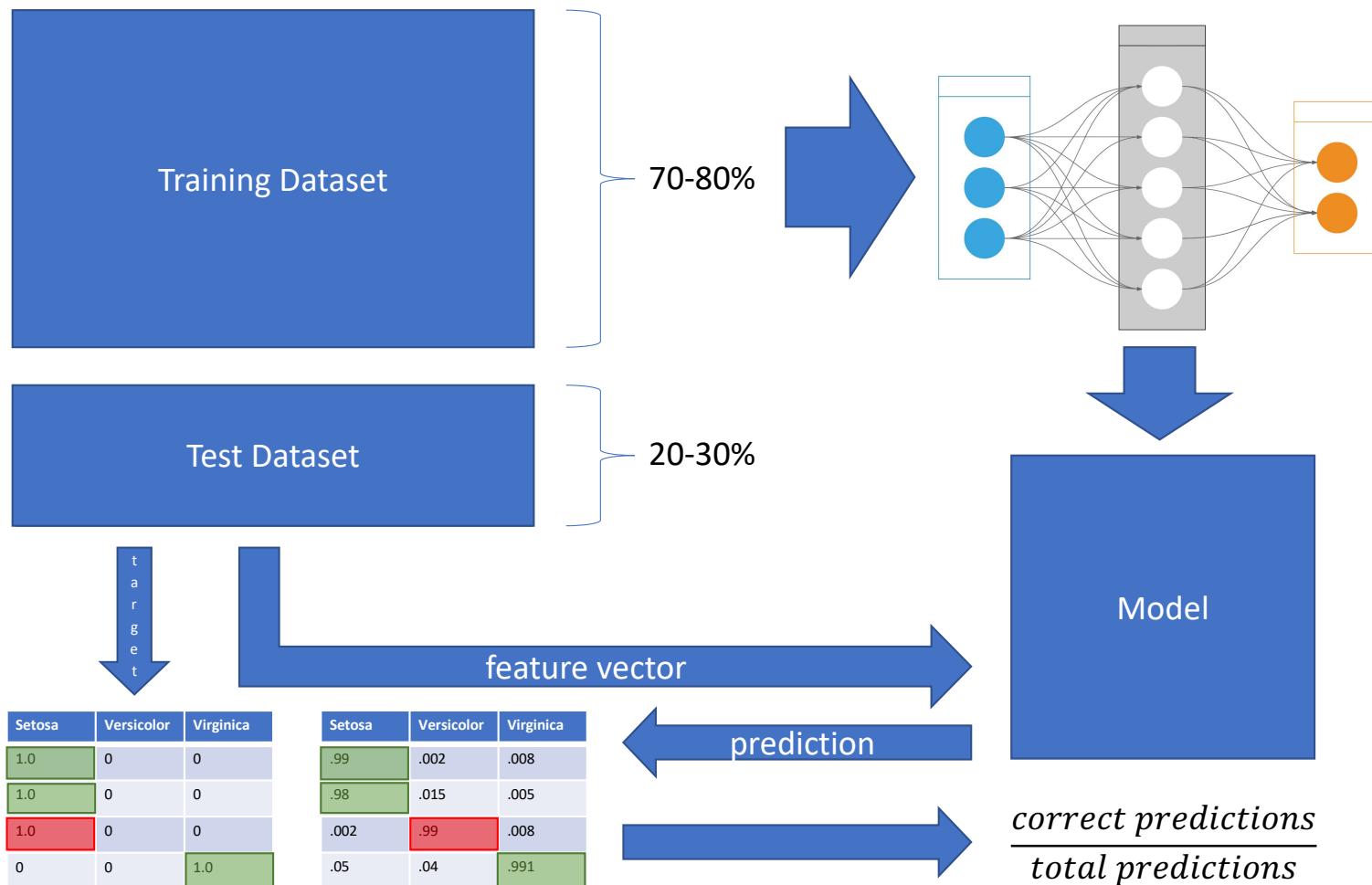
@poweredbyaltnet

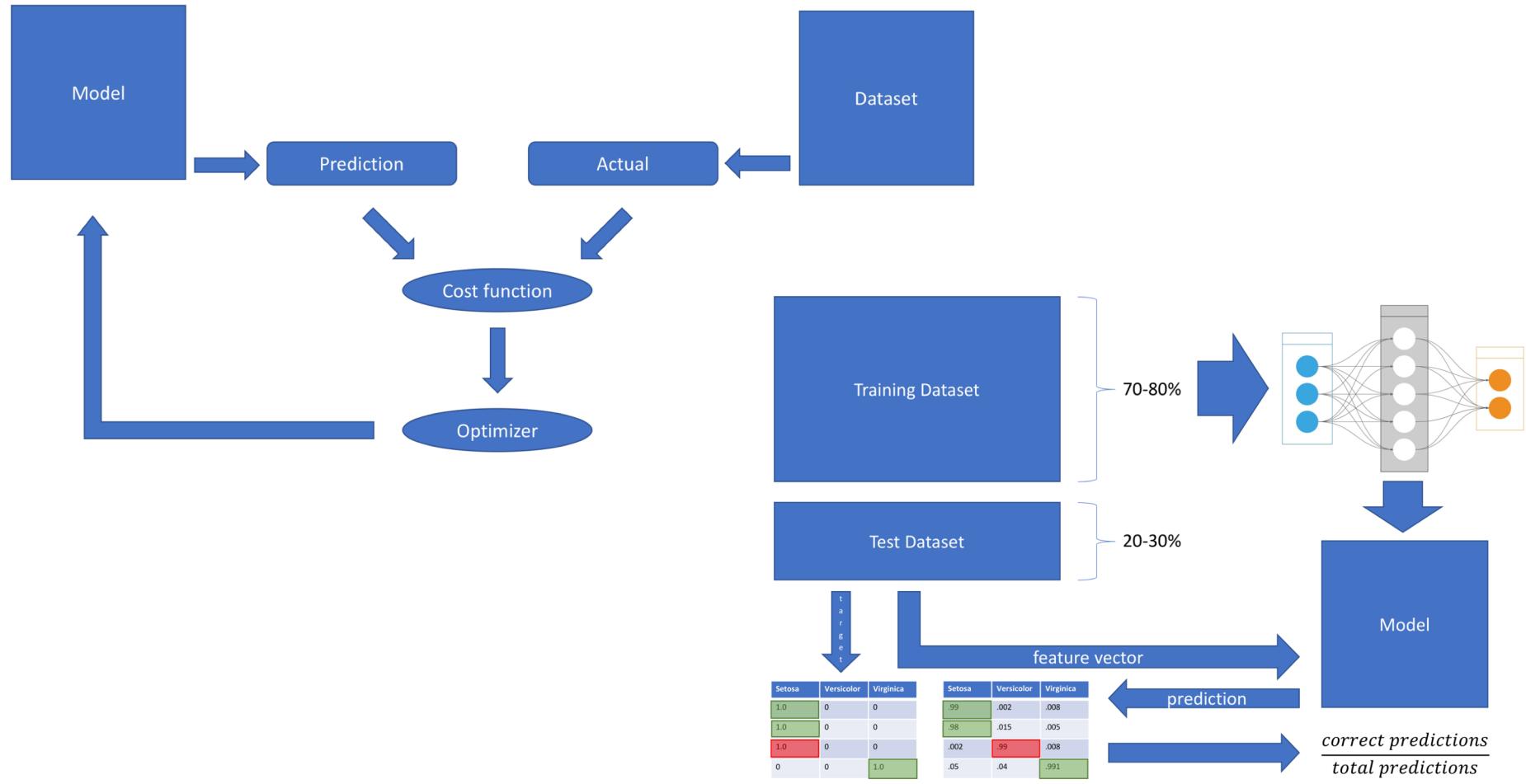


The model is trained



Now what?





Epoch  
000,732Learning rate  
0.03Activation  
ReLURegularization  
NoneRegularization rate  
0Problem type  
Classification

## DATA

Which dataset do you want to use?



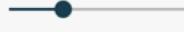
Ratio of training to test data: 50%



Noise: 0



Batch size: 10



REGENERATE

## FEATURES

Which properties do you want to feed in?

- $X_1$  
- $X_2$  
- $X_1^2$  
- $X_2^2$  
- $X_1X_2$  
- $\sin(X_1)$  
- $\sin(X_2)$  

## 4 HIDDEN LAYERS

+ - 5 neurons

+ - 4 neurons

+ - 3 neurons

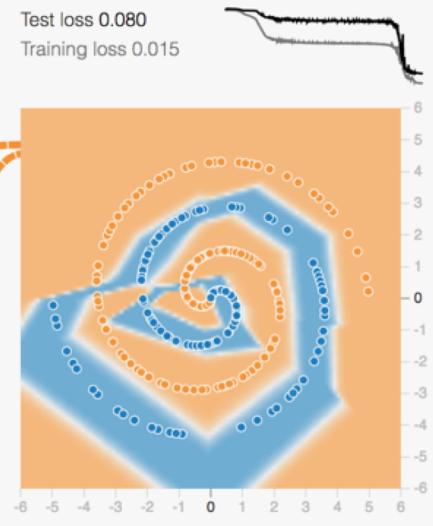
+ - 2 neurons

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

## OUTPUT

Test loss 0.080  
Training loss 0.015



Colors shows data, neuron and weight values.

Show test data

Discretize output

playground.tensorflow.org

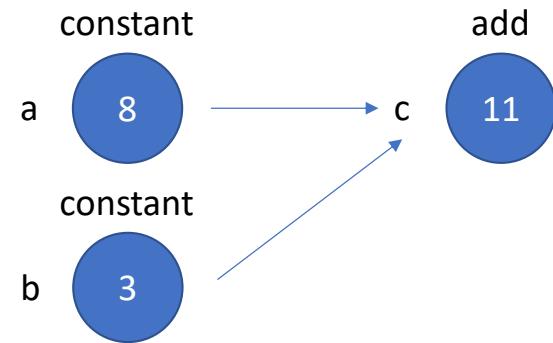
@poweredbyaltnet



# TensorFlow

@poweredbyaltnet

```
import tensorflow as tf  
a = tf.constant(8)  
b = tf.constant(3)  
c = tf.add(a, b) # or a + b, operator is overloaded
```



```
Last login: Tue Apr 10 09:40:34 on ttys001
douglasmbpr:~ douglasstarnes$ ipython
Python 3.6.3 |Anaconda custom (64-bit)| (default, Oct  6 2017, 12:04:38)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]: import tensorflow as tf
```

```
In [2]: a = tf.constant(8)
```

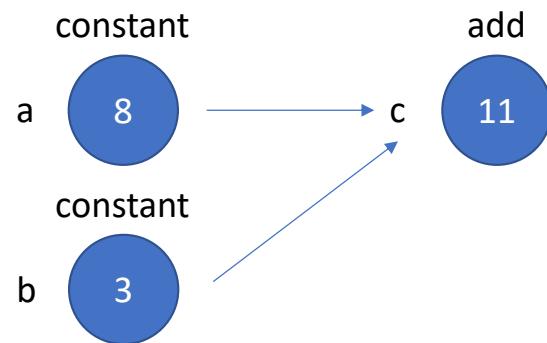
```
In [3]: b = tf.constant(3)
```

```
In [4]: c = tf.add(a, b)
```

```
In [5]: print(c)
Tensor("Add:0", shape=(), dtype=int32)
```

```
In [6]: 
```

```
import tensorflow as tf  
a = tf.constant(8)  
b = tf.constant(3)  
c = tf.add(a, b) # or a + b  
with tf.Session() as s:  
    print(s.run(c))
```



```
In [5]: print(c)
Tensor("Add:0", shape=(), dtype=int32)

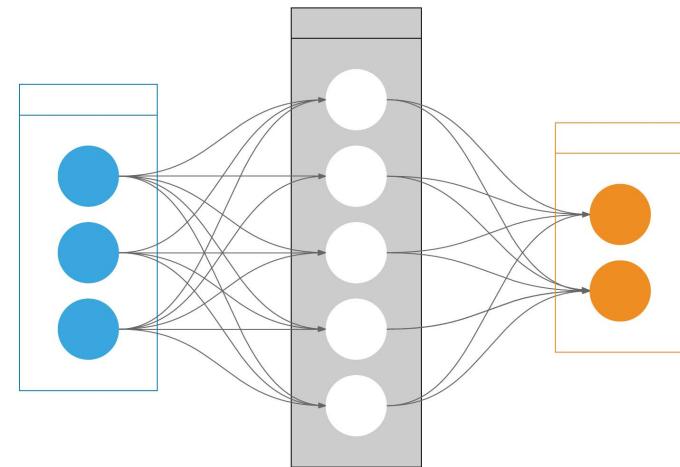
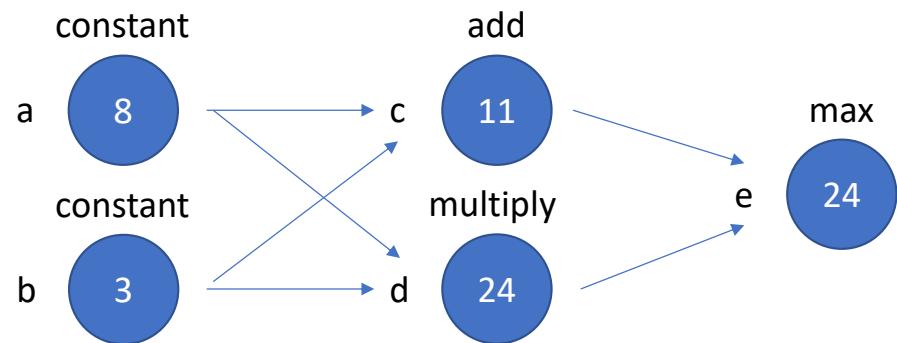
In [6]: with tf.Session() as s:
...:     print(s.run(c))
...:
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, b
ut these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, b
ut these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but
these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but
these are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but
these
11
```

```
In [7]:
```

```

import tensorflow as tf
a = tf.constant(8)
b = tf.constant(3)
c = tf.add(a, b) # or a + b
with tf.Session() as s:
    print(s.run(c))
d = tf.multiply(a, b) # or a * b
e = tf.max(c, d)
with tf.Session() as s:
    print(s.run(e))

```



@poweredbyaltnet

## Meet MNIST

Scans of handwritten digits

Each image is 28x28 pixels

Each pixel value is 0 (black)  
to 255 (white)

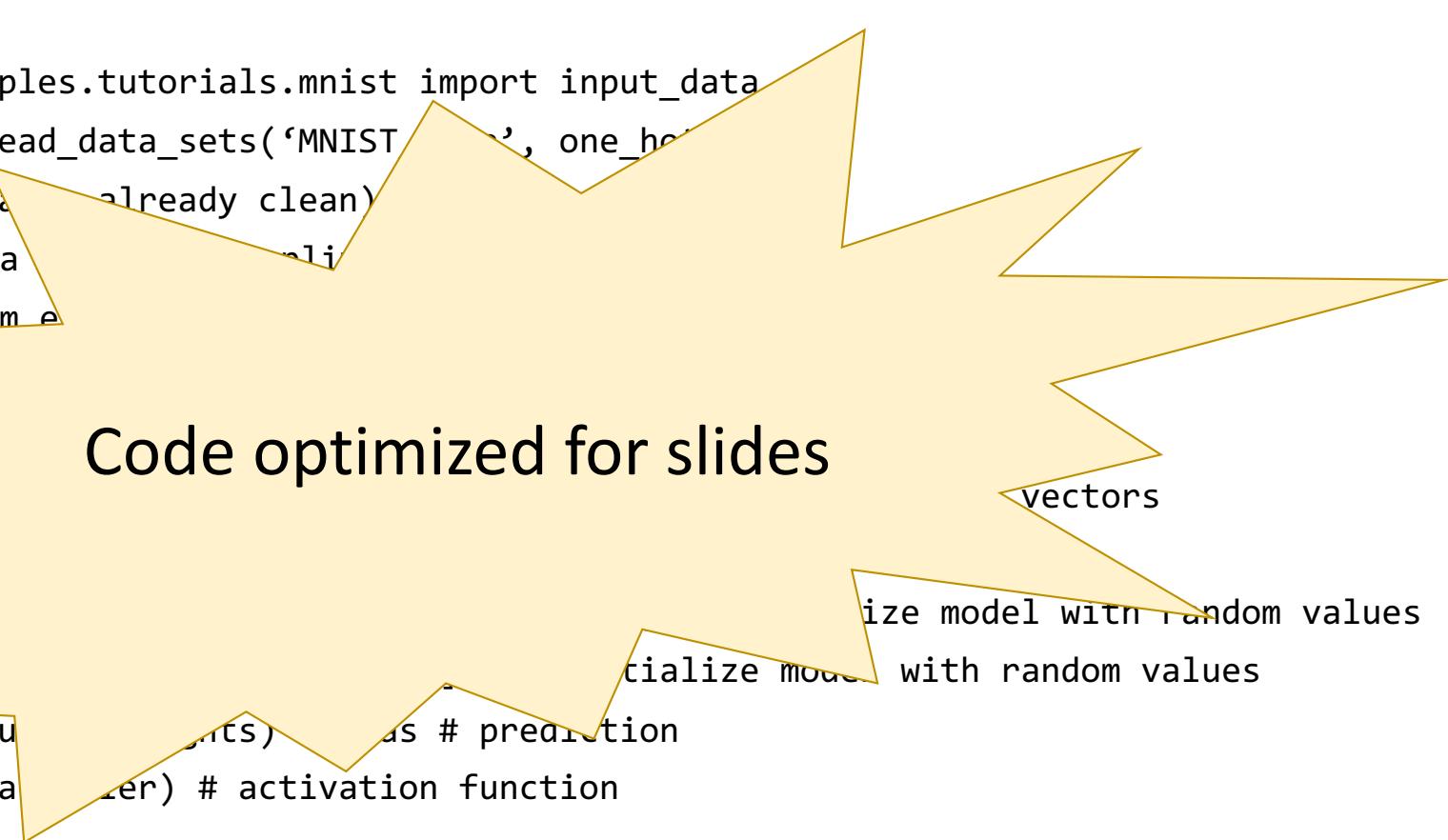
Feature vector is the entire  
image (784 pixels)

The target is the digit  
represented (0-9)

Interesting variations  
(ie. fashion MNIST)



```
# Step 1: Acquire
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
# Step 2: Clean (data is already clean)
# Step 3: Split (data is already split)
print(mnist.train.num_examples)
print(mnist.validation.num_examples)
# Step 4: Train
x = tf.placeholder(tf.float32, [None, 784])
y_hat = tf.placeholder(tf.float32, [None, 10])
weights = tf.Variable(tf.random_normal([784, 10]))
bias = tf.Variable(tf.random_normal([10]))
classifier = tf.matmul(x, weights) + bias # units)
y = tf.nn.softmax(classifier) # activation function
cost_function =
    tf.reduce_mean(-tf.reduce_sum(y_hat * tf.log(y))) # measure of similiarity
```

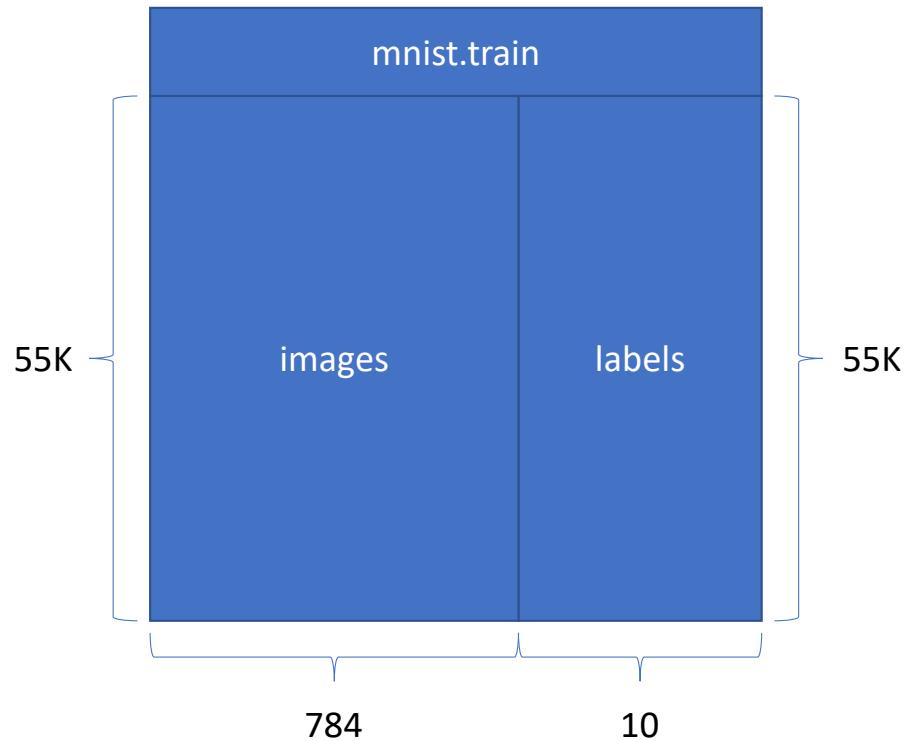


## Code optimized for slides

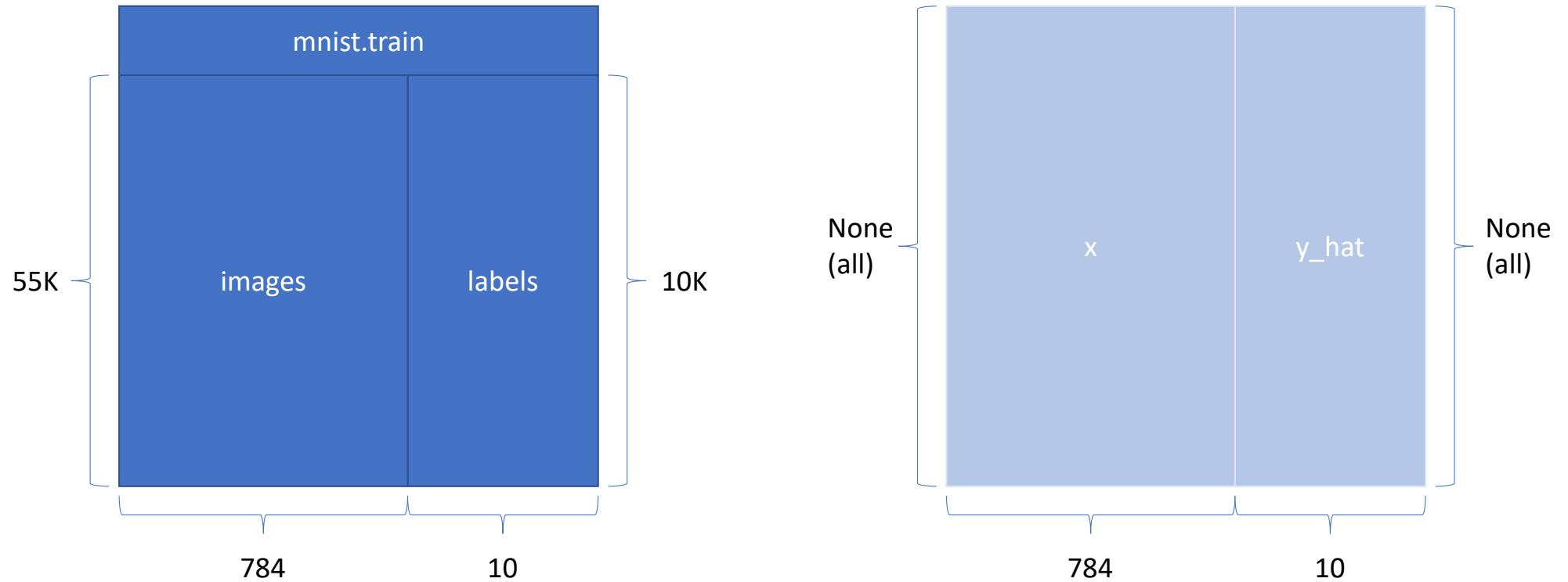
```
optimizer = tf.train.GradientDescentOptimizer(0.5).minimize(cost_function) # optimizer
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer()) # assign random values
    for _ in range(mnist.train.num_examples / 100): # run one epoch
        batch = mnist.train.next_batch(100)
        sess.run(optimizer, feed_dict={x: batch[0], y_hat: batch[1]})

# Step 5: Test
correct_predictions = tf.equal(tf.argmax(y, 1), tf.argmax(y_hat, 1)) # total no. correct
# Step 6: Score
average_correct = tf.reduce_mean(tf.cast(correct_predictions, tf.float32)) # avg. no. correct
with tf.Session() as sess:
    print(
        sess.run(average_correct,
            feed_dict={x: mnist.test.images, y_hat: mnist.test.labels}))

# Step 7: Evaluate - make some pretty pictures
```



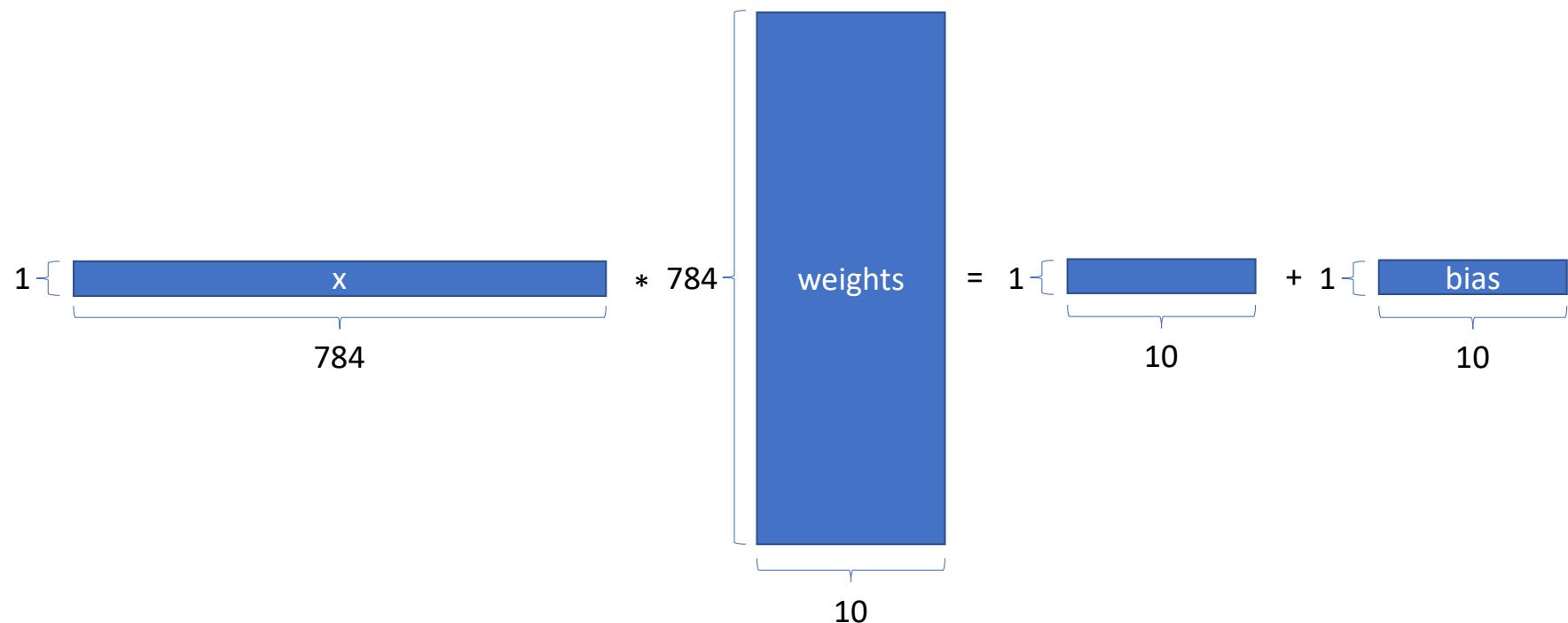
```
# Step 1: Acquire  
from tensorflow.examples.tutorials.mnist import input_data  
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)  
# Step 2: Clean (data is already clean)  
# Step 3: Split (data is already split)  
print(mnist.train.num_examples)  
print(mnist.test.num_examples)
```



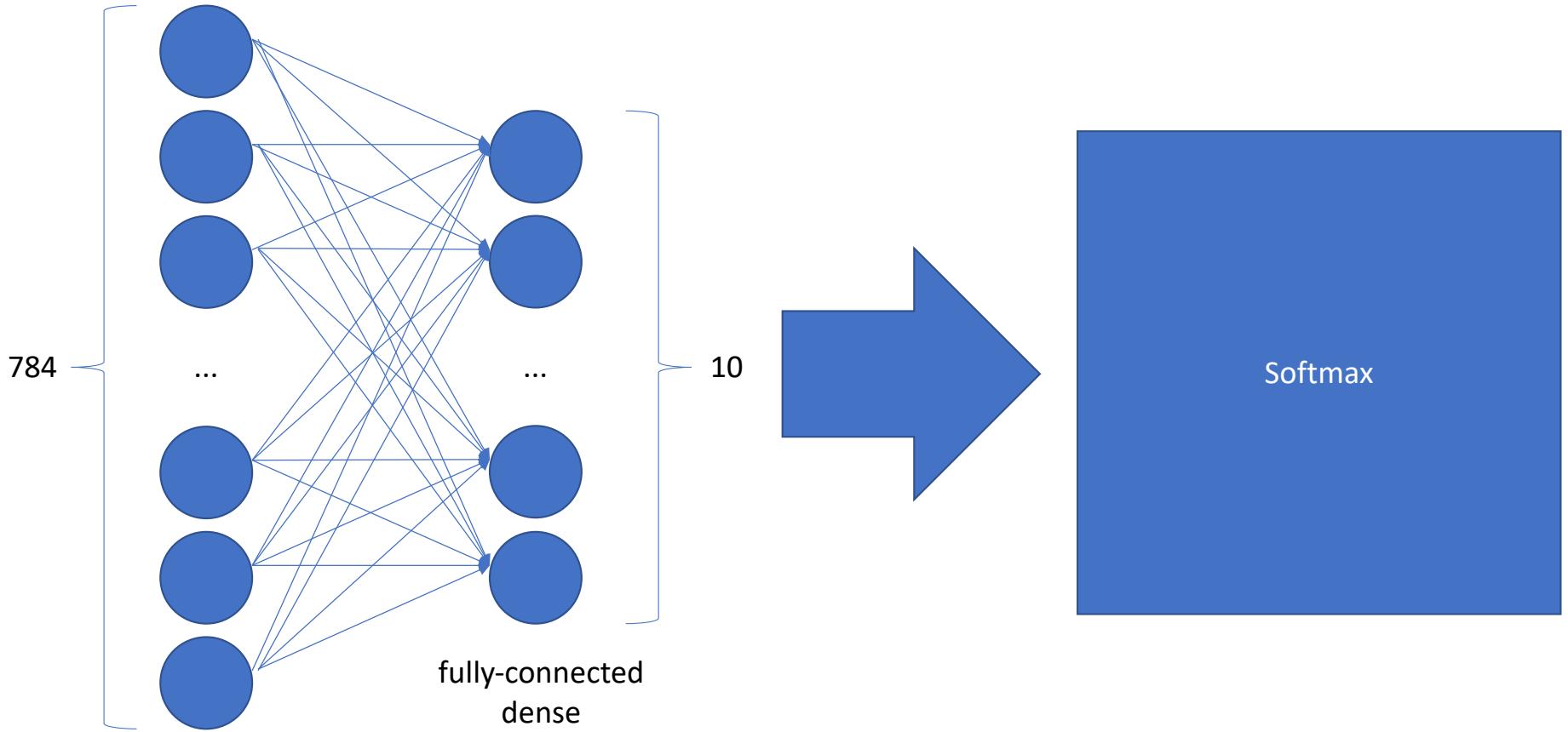
```
# Step 4: Train  
x = tf.placeholder(tf.float32, shape=[None, 784])  
y_hat = tf.placeholder(tf.float32, shape=[None, 10])
```

@poweredbyaltnet

```
weights = tf.Variable(tf.truncated_normal([784,10])  
bias = tf.Variable(tf.truncated_normal([10]))  
classifier = tf.matmul(x, weights) + bias
```



@poweredbyaltnet



@poweredbyaltnet

```
y = tf.nn.softmax(classifier)
```

$$\text{softmax}(i_x) = \frac{e^{i_x}}{\sum_{j=1}^k e^{i_j}}$$

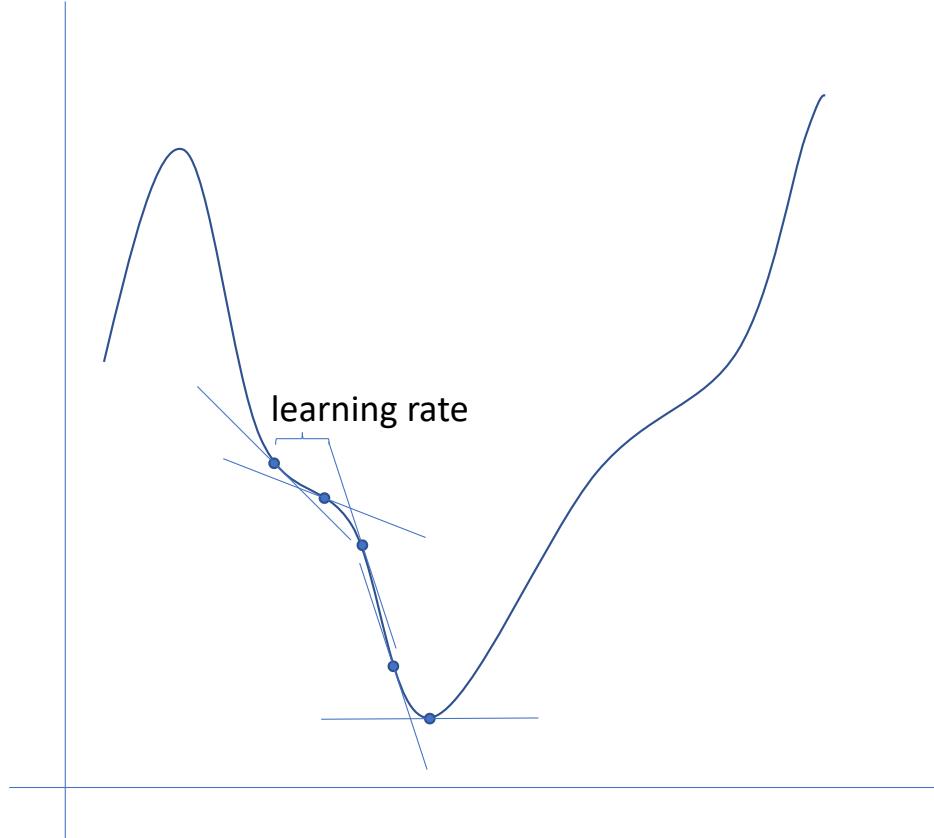
```
from math import exp
i = [100, 101, 102]
def softmax(x):
    return exp(i[x]) / sum([exp(j) for j in i])
print([softmax(a) for a in range(len(i))])
```

```
[0.09003057317038046, 0.24472847105479764, 0.6652409557748219]
```

9%    24%    67%

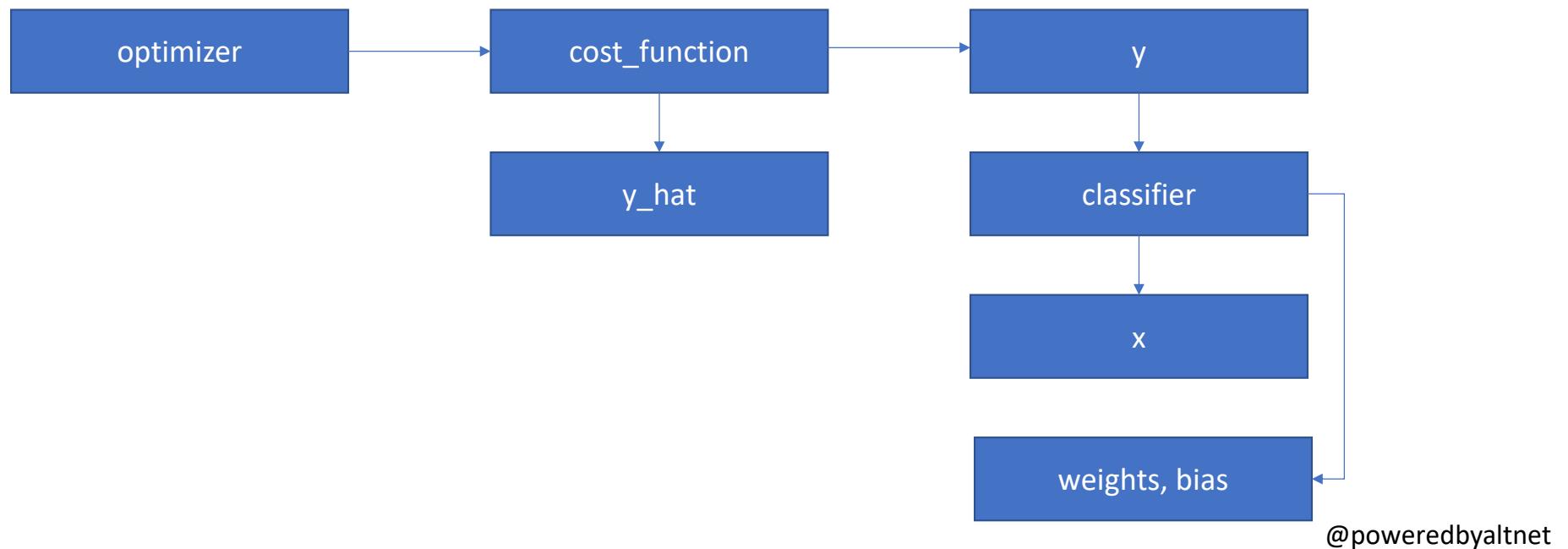
@poweredbyaltnet

```
optimizer = tf.train.GradientDescentOptimizer(0.5).minimize(cost_function)
```



If the learning rate is too large, we will skip over the minimum.  
If the learning rate is too small, we will waste time.

```
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())  
    for _ in mnist.train.num_examples / 100:  
        batch = mnist.train.next_batch(100)  
        sess.run(optimizer, feed_dict={x: batch[0], y_hat: batch[1]})
```



```
correct_predictions = tf.equal(tf.argmax(y, 1), tf.argmax(y_hat, 1))
```

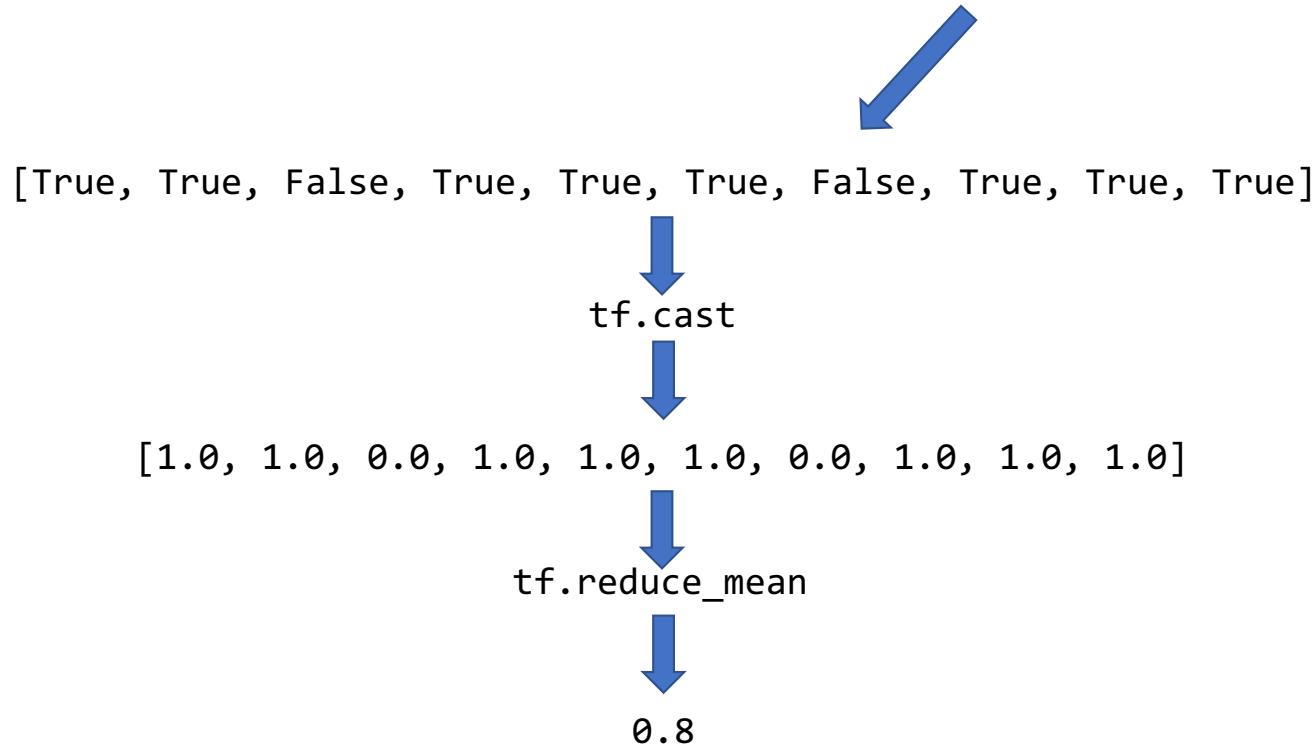
0	1	2	3	4	5	6	7	8	9
.003	.24	.9	.012	.004	0	.031	.001	.013	.008
.01	.019	.012	.003	.9	.02	.012	.014	.005	0

y

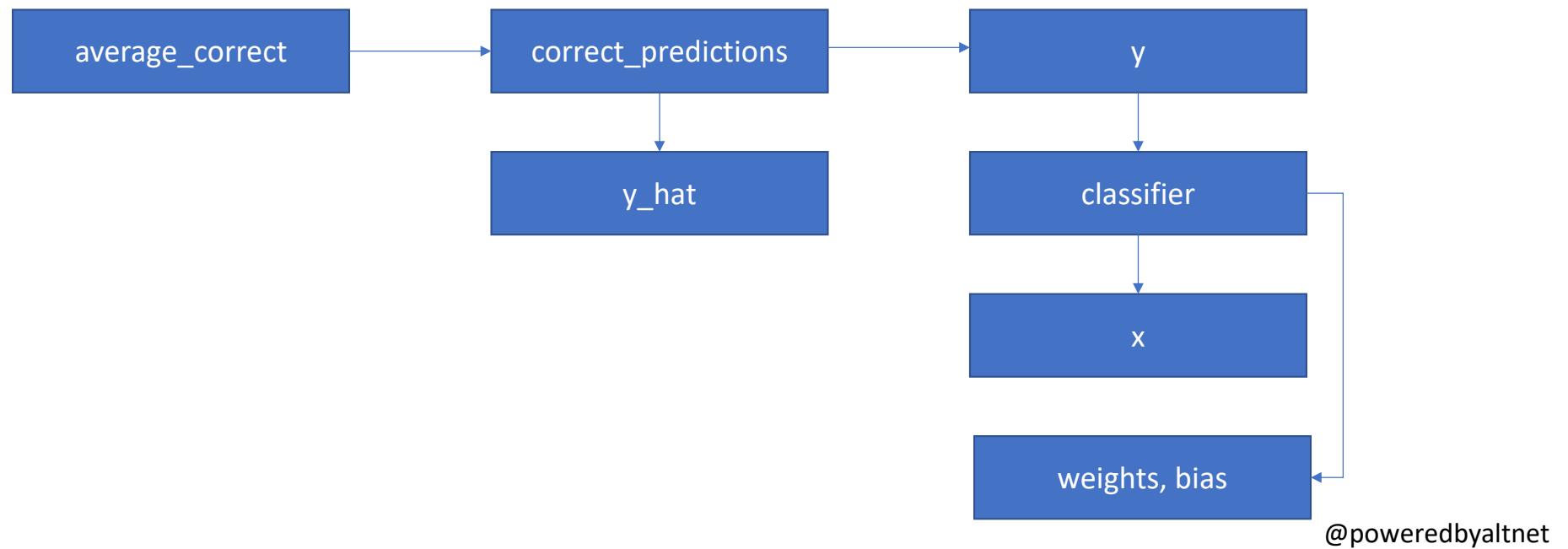
0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0

y\_hat

```
average_correct = tf.reduce_mean(tf.cast(correct_predictions, tf.float32))
```



```
with tf.Session() as sess:  
    sess.run(average_correct, feed_dict={x: mnist.test.images,  
                                         y_hat: mnist.test.labels})
```



**DEMO**

@poweredbyaltnet



That works but surely there  
is an easier method?

@poweredbyalt.net

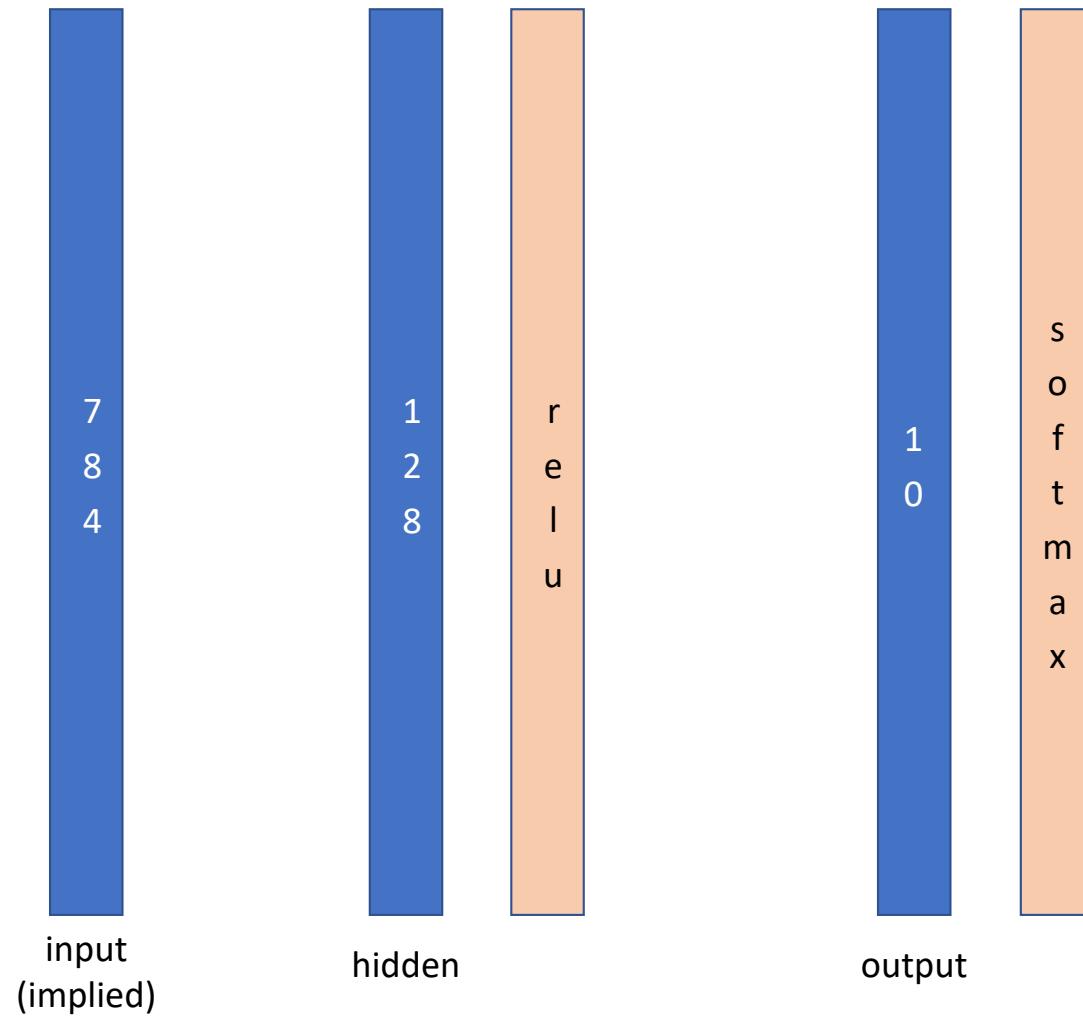


# Keras

@poweredbyaltnet

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import RMSProp
from keras.utils import np_utils

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 784).astype('float32') / 255 # apply to X_test
Y_train = np_utils.to_categorical(y_train, 10) # apply to y_test
model = Sequential()
model.add(Dense(128, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer=RMSProp(), metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=128, epochs=5)
score = model.evaluate(X_test, Y_test)
```



@poweredbyaltnet

**DEMO**

@poweredbyaltnet



That's all there is to  
it?

Surely there is a  
catch.

28px



1  
0

7,840

28px

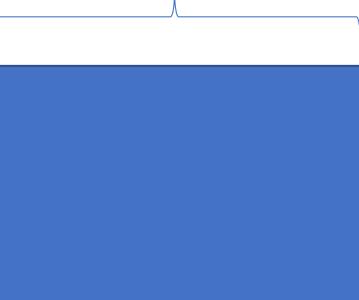


1  
2  
8

1  
0

101,770

640px



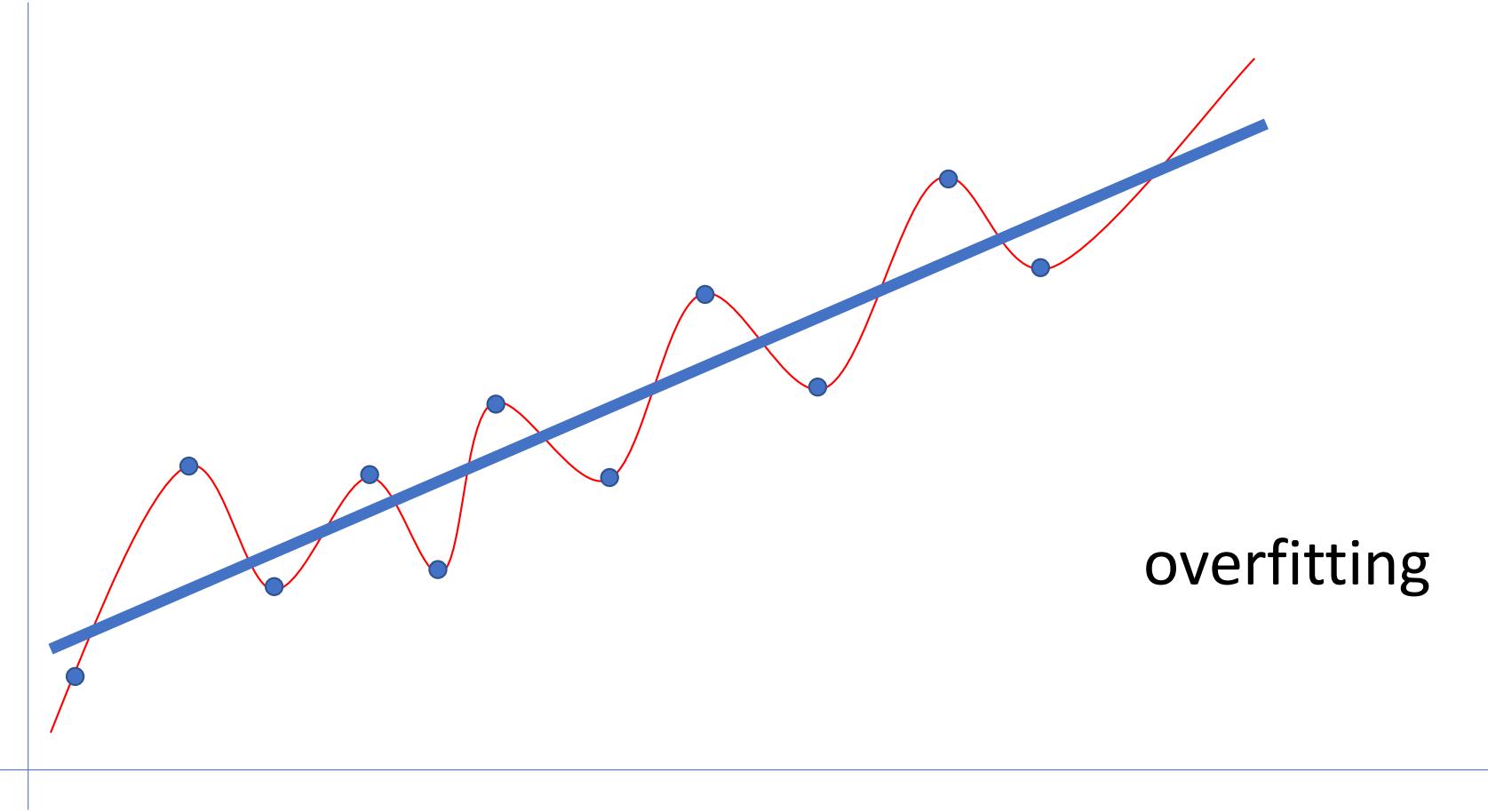
>300,000



@poweredbyaltnet



@poweredbyaltnet



overfitting



@poweredbyalt.net

Goal: consider fewer, more relevant, features

How to determine which features are relevant?

Let the network decide!

Convolutional Neural Network (CNN)

Two new layers

Convolutional layers – detect features

Pooling layers – combine features

Features are then used in dense layers



input



conv2d

1	0	0	1	1	0
1	1	1	0	1	1
0	1	1	1	0	1
1	1	1	0	1	1
0	0	1	1	0	1
0	1	1	0	0	0

1	0	0
1	1	1
0	1	1

filter/kernel

1 1	0 0	0 0	1	1	0
1 1	1 1	1 1	0	1	1
0 0	1 1	1 1	1	0	1
1	1	1	0	1	1
0	0	1	1	0	1
0	1	1	0	0	0

6

1	0 1	0 0	1 0	1	0
1	1 1	1 1	0 1	1	1
0	1 0	1 1	1 1	0	1
1	1	1	0	1	1
0	0	1	1	0	1
0	1	1	0	0	0

6	4
---	---

1	0	0 1	1 0	1 0	0
1	1	1 1	0 1	1 1	1
0	1	1 0	1 1	0 1	1
1	1	1	0	1	1
0	0	1	1	0	1
0	1	1	0	0	0

6	4	3
---	---	---

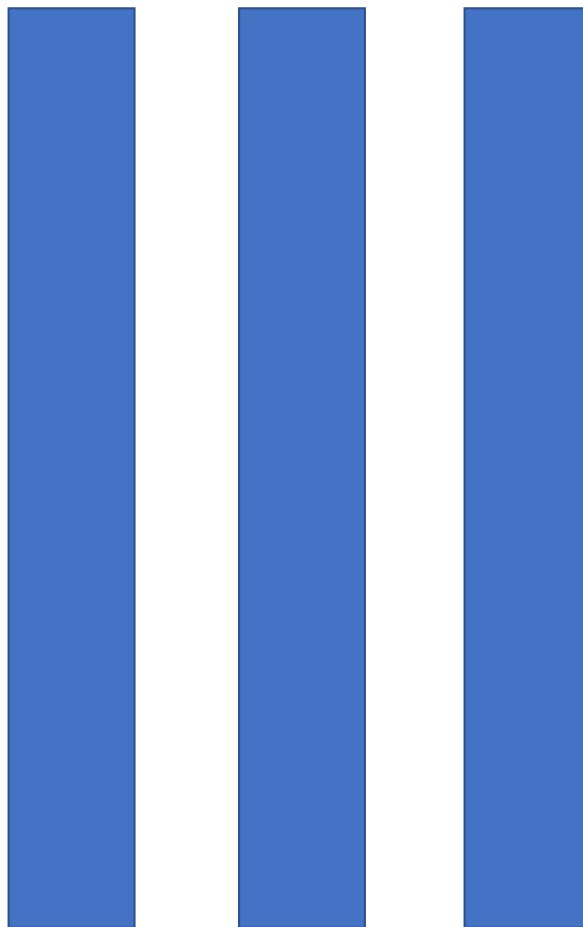
1	0	0	1 1	1 0	0 0
1	1	1	0 1	1 1	1 1
0	1	1	1 0	0 1	1 1
1	1	1	0	1	1
0	0	1	1	0	1
0	1	1	0	0	0

6	4	3	4
---	---	---	---

1	0	0	1	1	0
1	1	1	0	1	1
0	1	1	1	0	1
1	1	1	0 1	1 0	1 0
0	0	1	1 1	0 1	1 1
0	1	1	0 0	0 1	0 1

6	4	3	4
5	5	4	4
4	5	4	4
4	4	3	2

convolution



input

conv2d

pooling

6	4	3	4
5	5	4	4
4	5	4	4
4	4	3	2

6

6	4	3	4
5	5	4	4
4	5	4	4
4	4	3	2

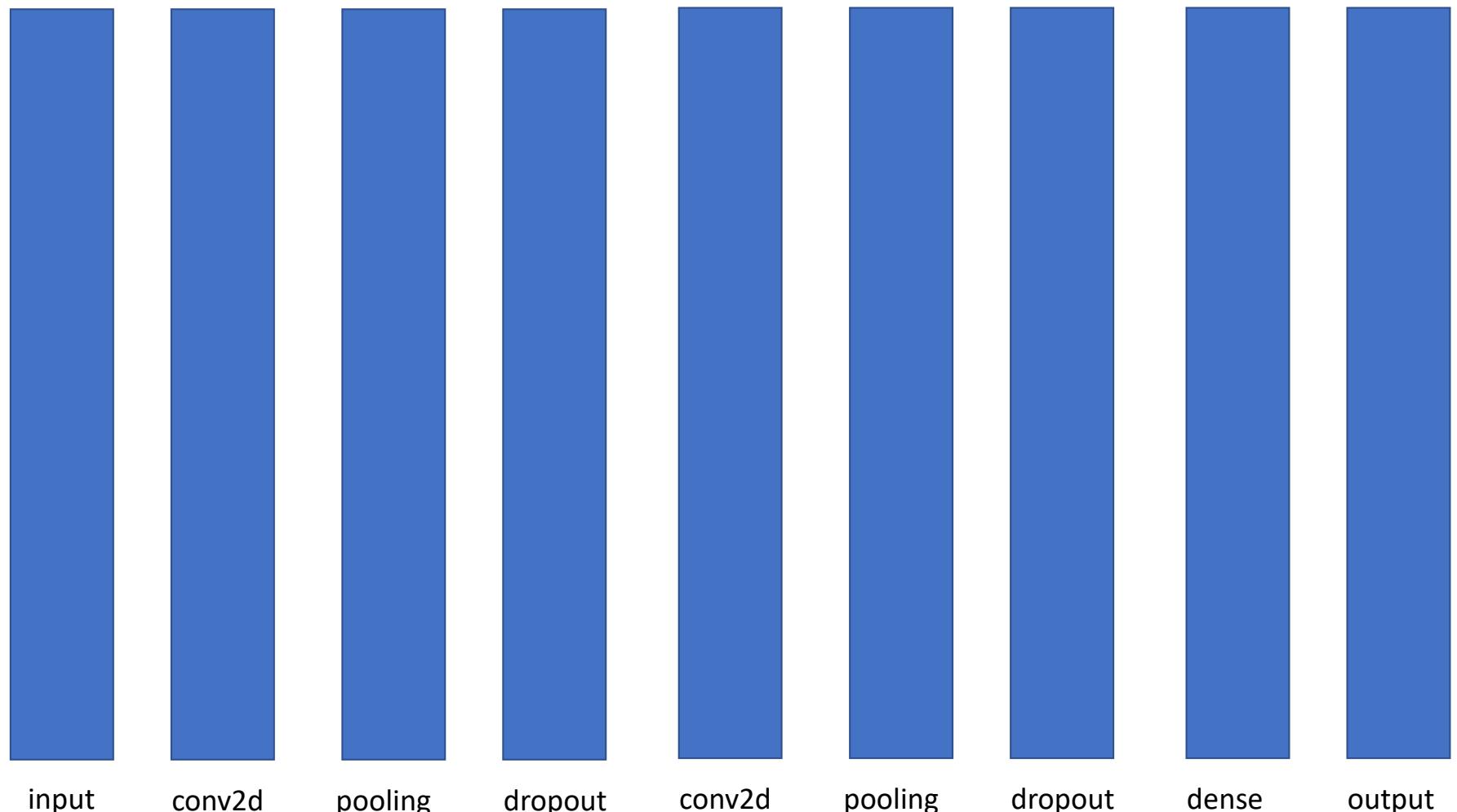
6	4
---	---

6	4	3	4
5	5	4	4
4	5	4	4
4	4	3	2

6	4
5	

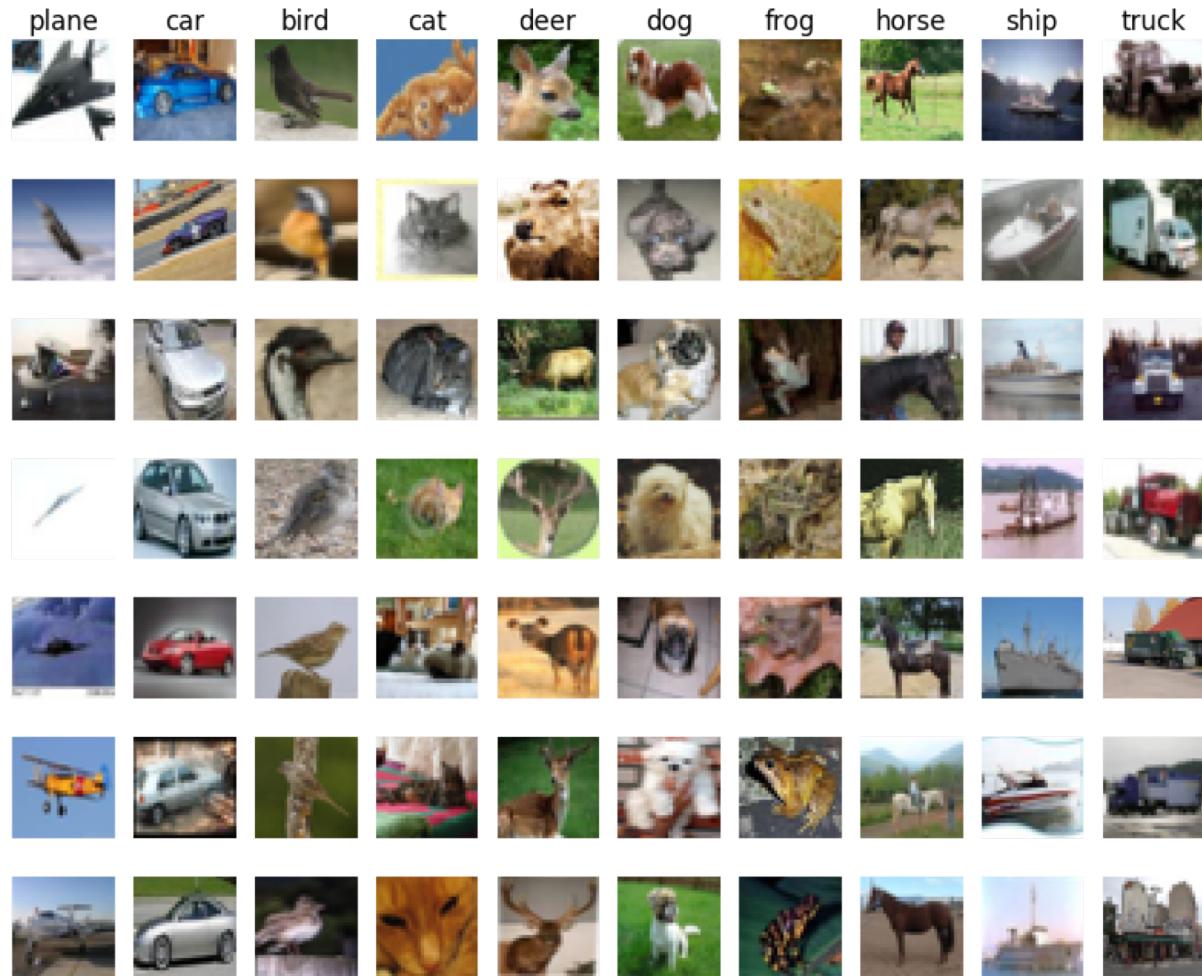
6	4	3	4
5	5	4	4
4	5	4	4
4	4	3	2

6	4
5	4



@poweredbyaltnet

# CIFAR 10



@poweredbyaltnet

```
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=(32, 32, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))
```

@poweredbyaltnet

Each word has a specific meaning based on its position within the sentence.

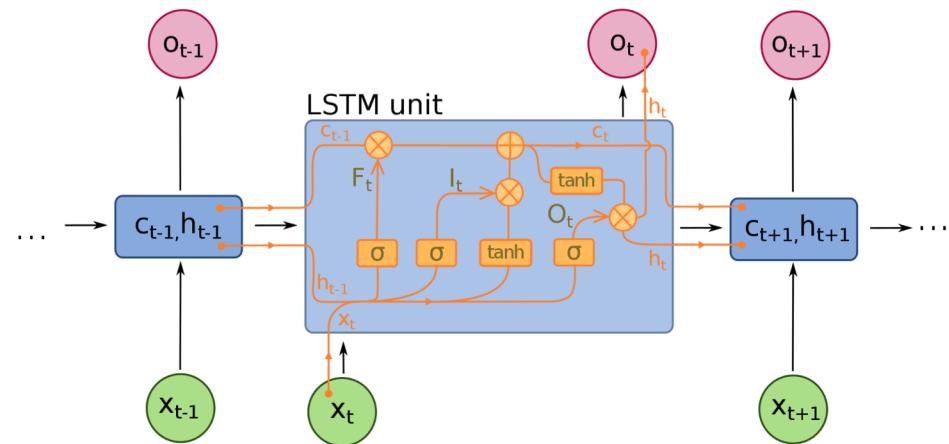
Time-series data

Recurrent neural network

Long Short Term Memory (LSTM) cell

The cell stores a state

State is a combination of the current input and the previous state



```
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(20000, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
```

Microsoft Cognitive Toolkit (CNTK) - Microsoft

PyTorch - Facebook

MXNet – Apache (used by AWS)

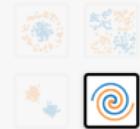
Caffe

Theano (no longer active)

Epoch  
000,732Learning rate  
0.03Activation  
ReLURegularization  
NoneRegularization rate  
0Problem type  
Classification

## DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

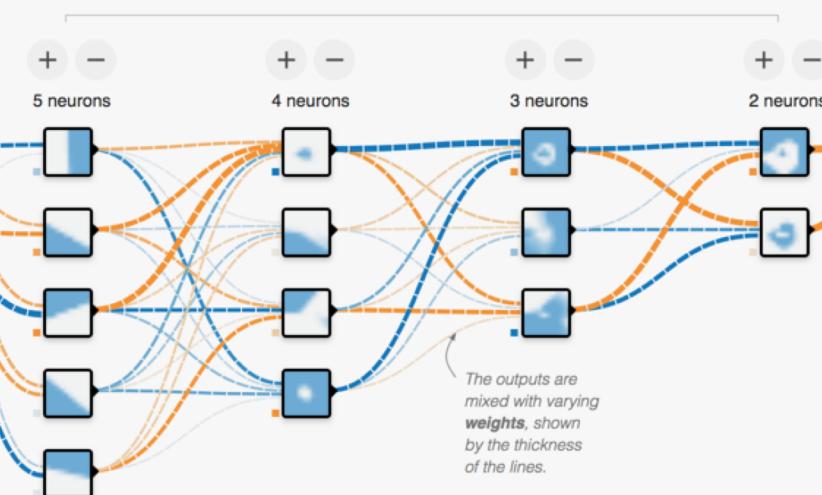
REGENERATE

## FEATURES

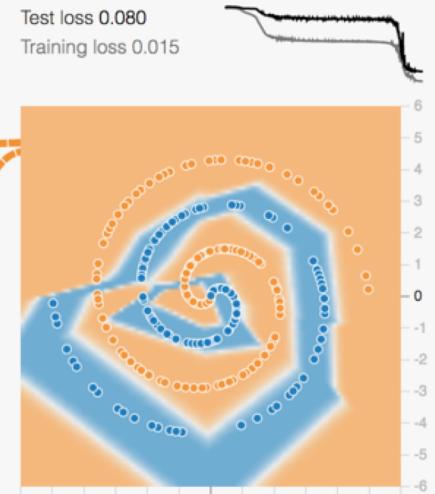
Which properties do you want to feed in?

 $X_1$ ,  $X_2$ ,  $X_1^2$ ,  $X_2^2$ ,  $X_1 X_2$ ,  $\sin(X_1)$ ,  $\sin(X_2)$ 

## 4 HIDDEN LAYERS



## OUTPUT

Test loss 0.080  
Training loss 0.015

Colors shows data, neuron and weight values.  
 Show test data    Discretize output

[playground.tensorflow.org](http://playground.tensorflow.org)

@poweredbyaltnet



# PLURALSIGHT

## Getting Started with Jupyter Notebook and Python



Scan the code or visit the link to get a 10-day FREE trial!



<https://bit.ly/jupyter-notebook>

@poweredbyaltnet

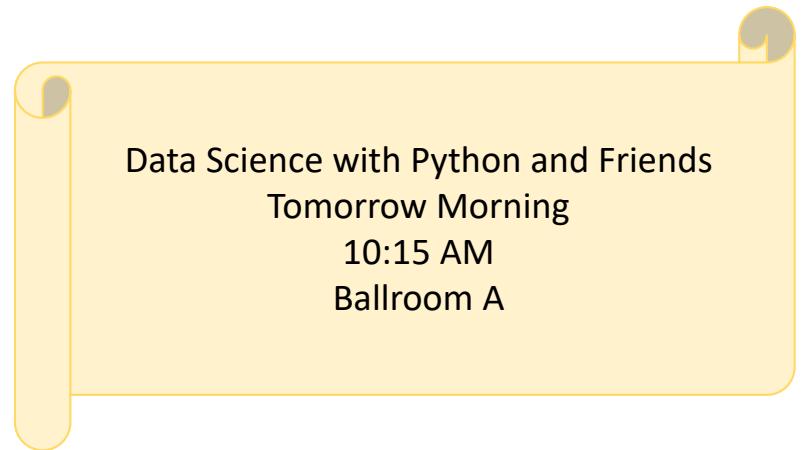
# Thank You!

@poweredbyaltnet

douglas@douglasstarnes.com

<http://douglasstarnes.com>

<https://github.com/douglasstarnes/codestock2018>



@poweredbyaltnet