Introduction to machine vision with Keras

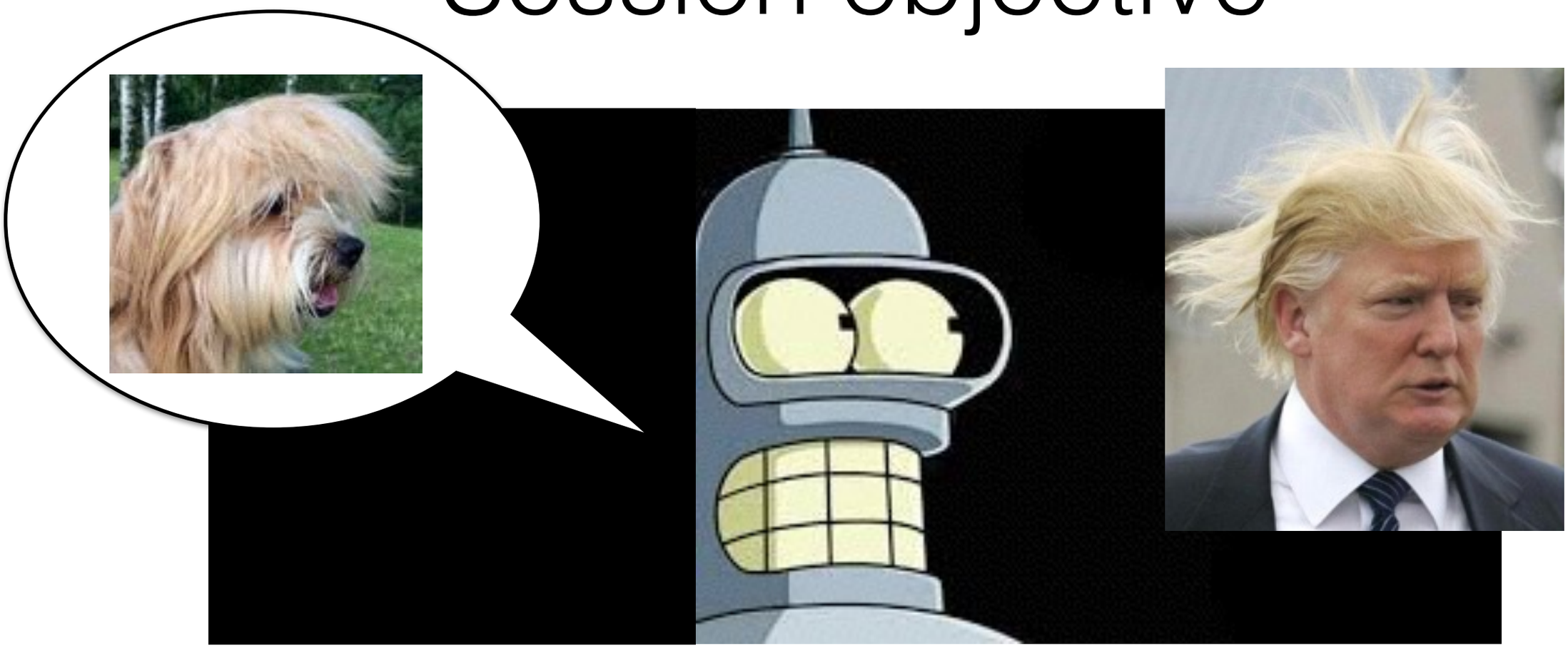
Mike Amy

PYCON
APAC 2016

A little about me

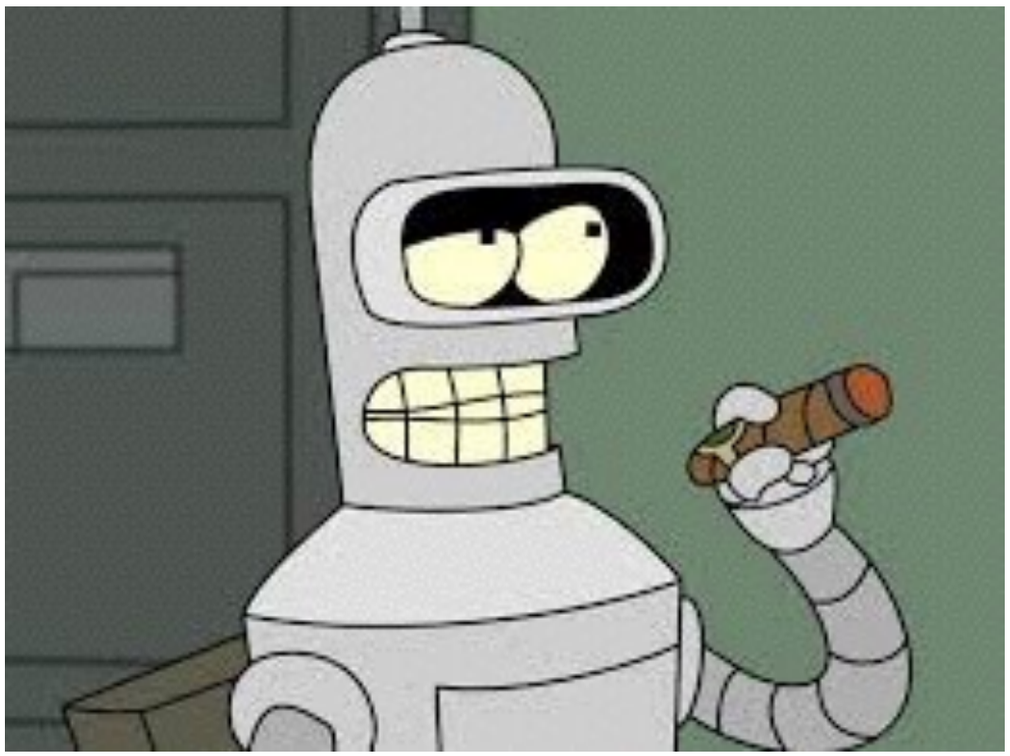
- 8 years working in Python.
- Consulted at UN WFP for the last 4 years. I'm not representing the UN here though.
- Very interested in machine learning.

Session objective



Get a machine to recognize objects in images

I want to show you pitfalls I fell into. Tutorials don't teach you these things, only showing you the right way.



So you want to do some deep learning?

You don't need:

- A PhD.
- A deep understanding of calculus, probability, linear algebra and statistics.

(But having these things won't hurt!)

You will need:

- A computer with a modern CUDA GPU.
- Python experience.
- Understand scipy/numpy basics.
- To enjoy learning and solving problems.
- The right amount of patience.
- A scientific mindset.

SciPy

NumPy

Installation requirements

- Python version 2.7 or 3+.
- SciPy and NumPy.
- Pandas for loading/saving datasets and models.
- Theano and/or TensorFlow.
- Keras.

Recommended

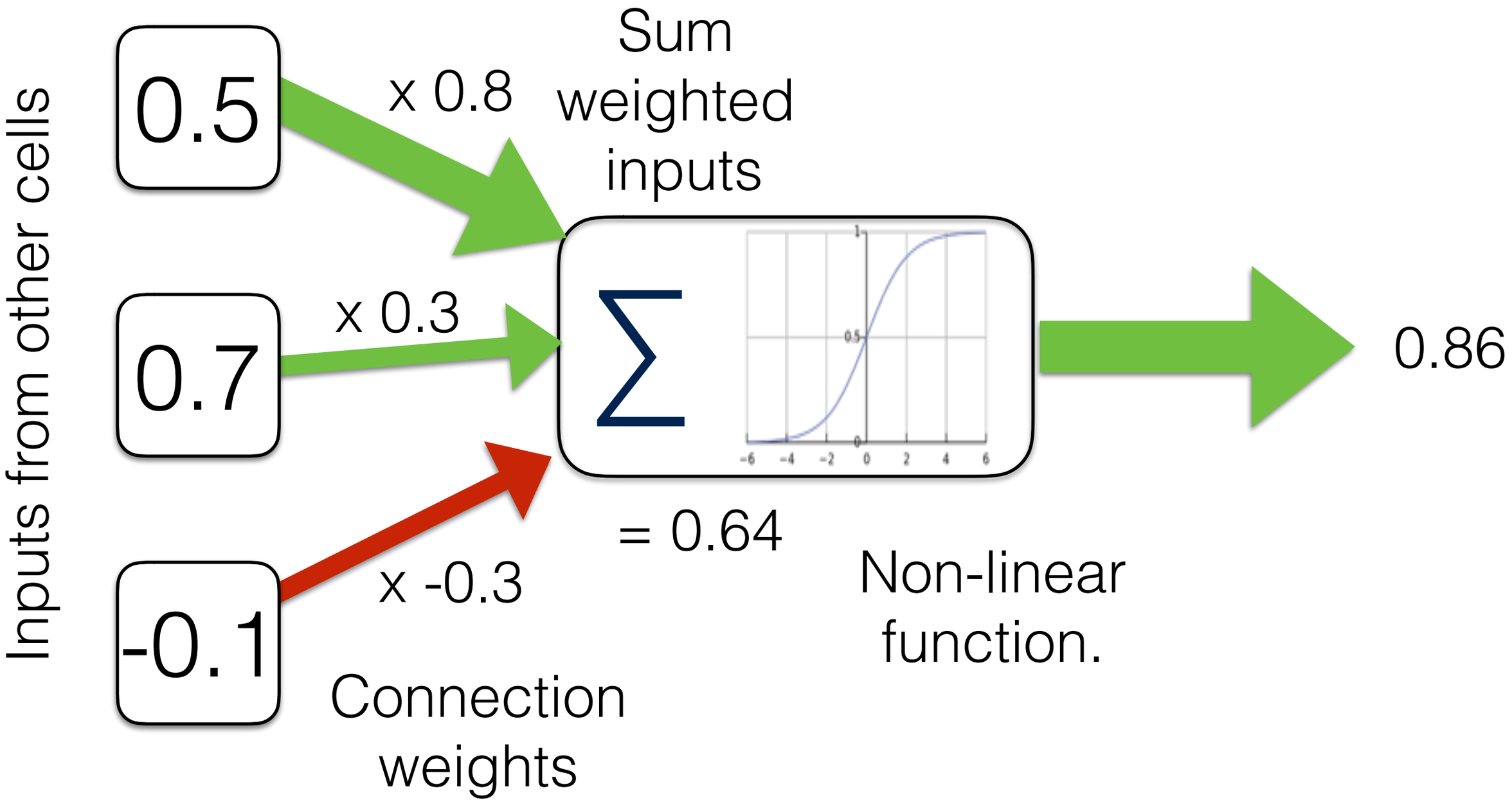
- Jupyter notebooks - better for experimental style of working.

- Matplotlib.

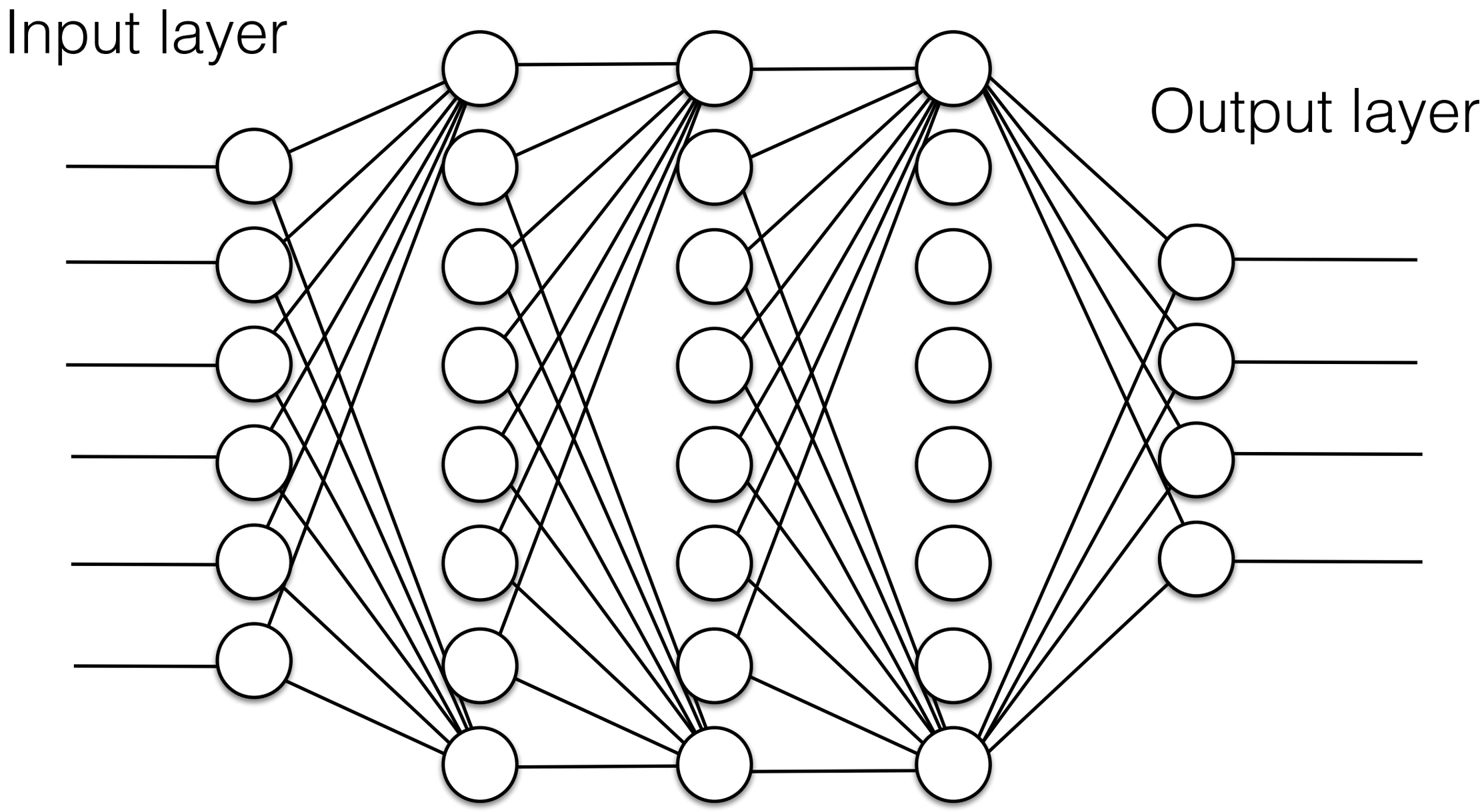
What is Deep Learning?

- One branch of machine learning.
- Features in data are learned by a **deep (i.e. multiple layers)** graph or network of connected layers of nodes.
- Nodes perform **non-linear or linear transformations** on the data.
- Algorithms iteratively train the network by propagating errors back through the network.
- Loosely inspired by brain cells, however, neurons are much more sophisticated.

Basic model of an artificial neuron.



Deep Neural Network



Many hidden layers

Why deep learning?

"Neural networks are the second best way of doing just about anything"

– John S. Denker, AT&T

But it's better to be pretty good at everything than the best at only one thing

“Deep learning” > “Artificial neural networks”?

- More layers.
  - E.g. deep residual network with 156 layers.
- Modern techniques and tools applied, especially use of GPUs, Dropout, regularisation and convolutional networks.

Keras

https://keras.io/

- Minimalist deep learning framework.
- Allows easy construction of deep networks, running and training them, saving them, etc.
- Nice abstraction level. Easily create layers and connect them together. Beginner-friendly.
- MIT license.

Keras Backends

http://deeplearning.net/software/theano/

https://www.tensorflow.org/

- Both frameworks are for fast numerical computation.
- You describe a mathematical operation on **tensors** (n-dimensional blocks of floats).
- The frameworks then compile code to run on GPU (or CPU). Using a GPU is really important if you are training networks.

So why Keras?

Theano and Tensorflow:

- powerful libraries, i.e.
- can certainly be used for deep learning, but…
- are for more general use than just deep learning,
- therefore can be a bit hard to use directly for deep learning

vs.

Why use Keras?

- Abstracts away the complexity of Theano and TensorFlow, lets you focus on the problem.
- Write backend-independent code.
- Friendly, helpful lead developer. François Chollet, a Google engineer. Many contributors. (includes me).
- Modular design.
- All native Python. No XML model files etc.

Keras components

Layers

Dense, Convolutional, Pooling, Recurrent, Embedding, Normalisation, Noise, Activations.

Optimizers

SGD, RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam

Models

Sequential
Functional API

Objectives (loss functions)

Mean squared error
Mean absolute error
Binary crossentropy
Categorical crossentropy
etc.

Weight constraints and activity regularisers

Datasets

Tools for

Preprocessing
Visualising models using graphviz
etc.

Demo time!

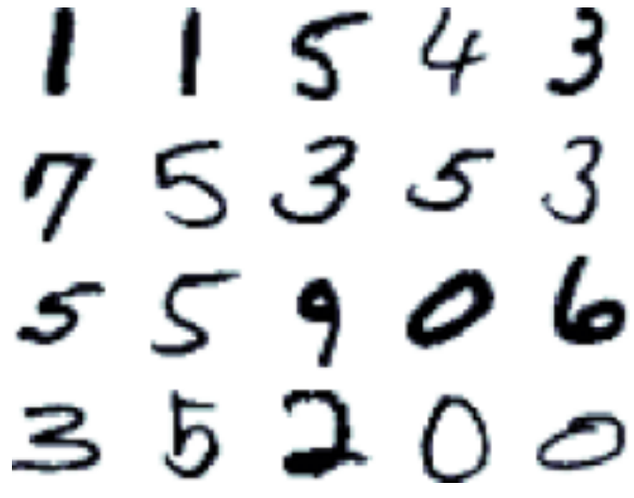
What shall we make it learn?

Standard Machine Learning Problems.

- Allow different machine learning techniques to be compared against each other.
- Some of them are used in competitions.
- You can use them to compare your ideas with the state of the art.
- They can give you an idea of when to stop trying to improve.

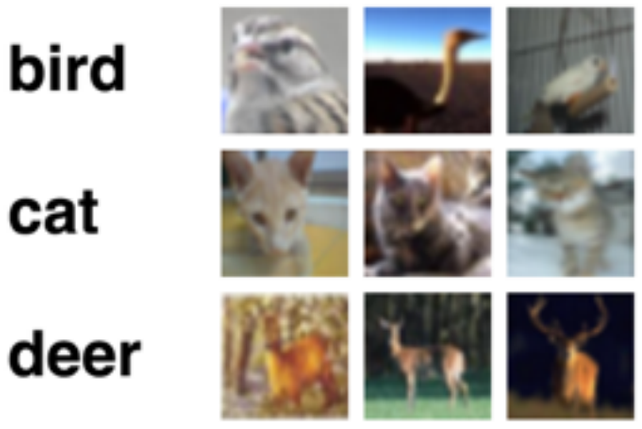
Standard Image Classification Problems.

See http://goo.gl/HRHz2L



- MNIST digits. Recognize hand-written digits, e.g. for reading mail addresses.

- CIFAR-(10/100): classify 32x32 images into 10 classes.



- STL-10: 96x96 images.

- SVHN: Street View House Numbers

- Various others

MNIST example

Dataset is easy to get using Keras:

```
from keras.datasets import mnist
```

```
images, classes = mnist.load_data()
```

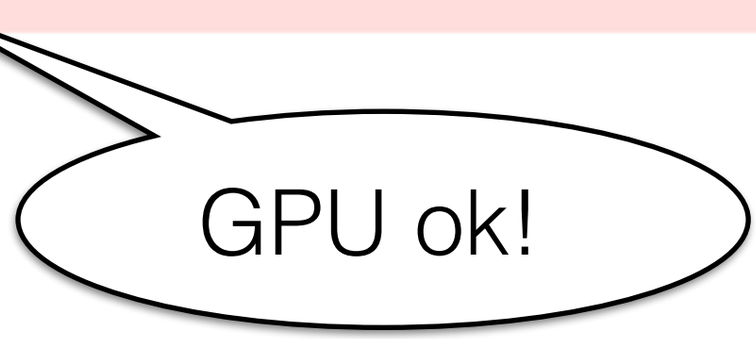
Note that you'll often see example code with x and y, or `x_train, y_train, x_test, y_test` x is input, y is output, (because we are learning a function). I like to use the real names of things for clarity.

MNIST example

Via Jupiter notebook

In [1]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import keras.datasets.mnist
import keras.models
import keras.layers
from keras.utils import np_utils
```

```
Using Theano backend.
Using gpu device 0: Graphics Device (CNMeM is enabled with initial size:
80.0% of memory, cuDNN not available)
```



Pitfall!
Not preparing your data

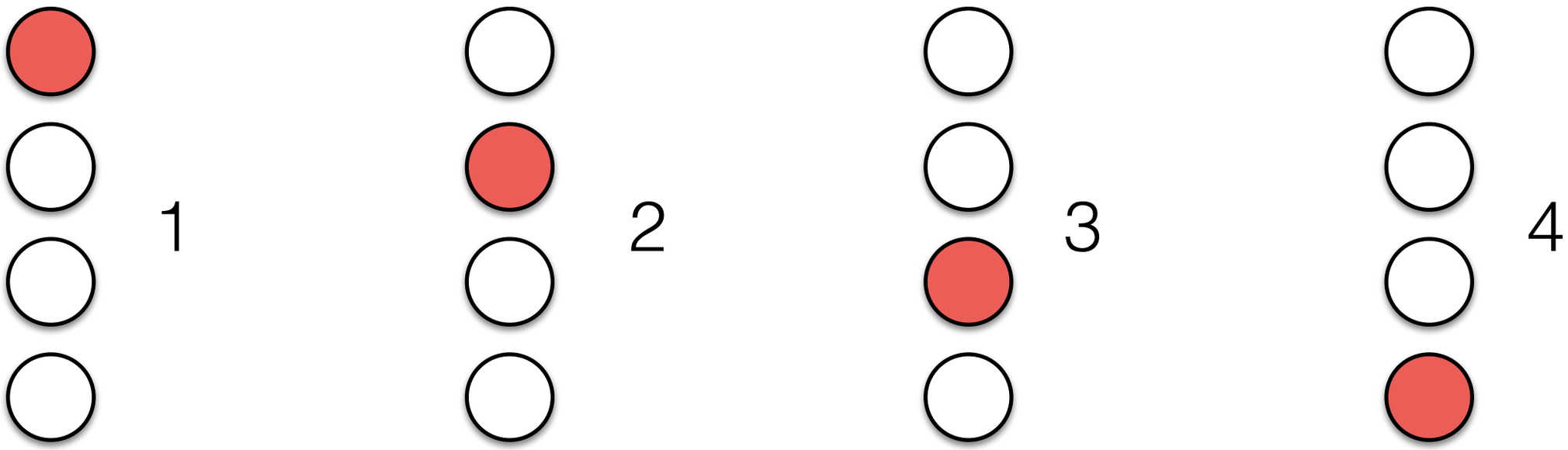
- Normalising values to range 0 - 1.0 or -1.0 to 1.0 really helps things along

- Reshaping data or interpolating it to fit your network.

- Often preprocess it in some way helps. E.g. RGB to HSV. Some channels contain different information.

Understand: One hot encoding

- Represent one output class by one ‘hot’ output amongst other cold ones.
- Simple and easy to deal with. Harder when there are many classes.
- np_utils.to_categorical() and np_utils.categorical_probas_to_classes() encode and decode.



MNIST example

Reshape and normalize

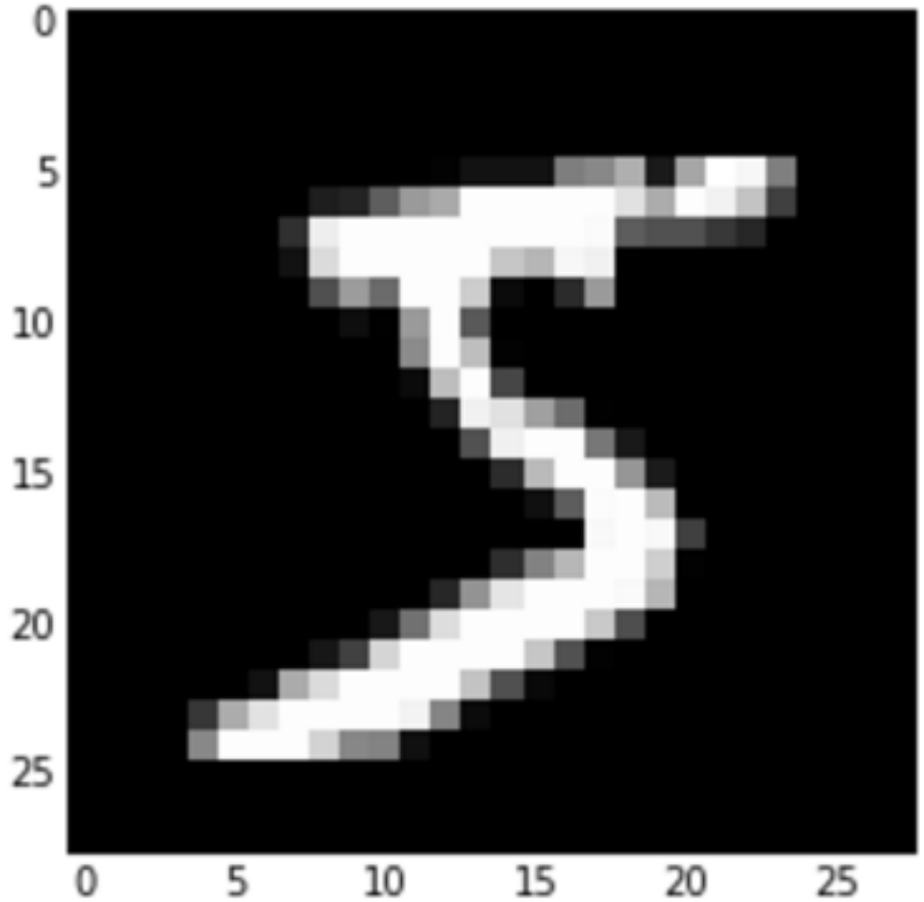
In [2]:
```
(
    (raw_training_images, training_classes),
    (test_images, test_classes)
) = keras.datasets.mnist.load_data()


def prepare_images_and_classes(images, image_classes):
    """Reshape and normalize the input data"""
    # flatten images to a single row of values
    images = images.reshape(
        images.shape[0],  # count of images
        28 * 28  # total pixels
    ).astype('float32')
    # normalize pixel values to be between 0 to 1
    return (
        images / 255.0,
        np_utils.to_categorical(image_classes) # one hot encoding
    )


training_images, training_classes = prepare_images_and_classes(
    raw_training_images,
    training_classes
)
test_images, test_classes = prepare_images_and_classes(
    test_images,
    test_classes
)
```

Tip! Visualise intermediate steps.

- You are creating a long chain of computation.
- Anything going wrong in the middle leads to garbage out.
- pyplot in jupyter is good for this.

MNIST example

In [3]:
```
plt.imshow(
    raw_training_images[0],
    cmap=plt.get_cmap('gray'),
    interpolation='nearest'
)
plt.show()
```



MNIST example

In [4]:
```
model = keras.models.Sequential()

flat_pixel_count = 28*28

model.add(
    keras.layers.Dense(
        28 * 28,
        input_dim=28*28,
        init='normal',
        activation='relu'
    )
)
model.add(
    keras.layers.Dense(
        10,
        init='normal',
        activation='softmax'
    )
)

model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Tip: model.summary()

Displays your model layers so you can see what shape of data they expect. You'll run into this error about mismatched dimensions. model.summary() will help.

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_2 (Dense)              (None, 10)                7850
=================================================================
Total params: 623290
_________________________________________________________________
```

Total params gives you an idea of the network size!
Useful to understand how long it will take to train.

Understand

- Epoch: one training session using all data in the training set.

- Batch: a bunch of samples that are trained together. Weights are updated per batch.

- Validation data: non-training data, that is used to check how well the network is learning to generalise.

MNIST example

```
In [6]: history = model.fit(
            training_images, training_classes,
            validation_data=(test_images, test_classes),
            nb_epoch=10,
            batch_size=200,
            verbose=2
        ).history
```

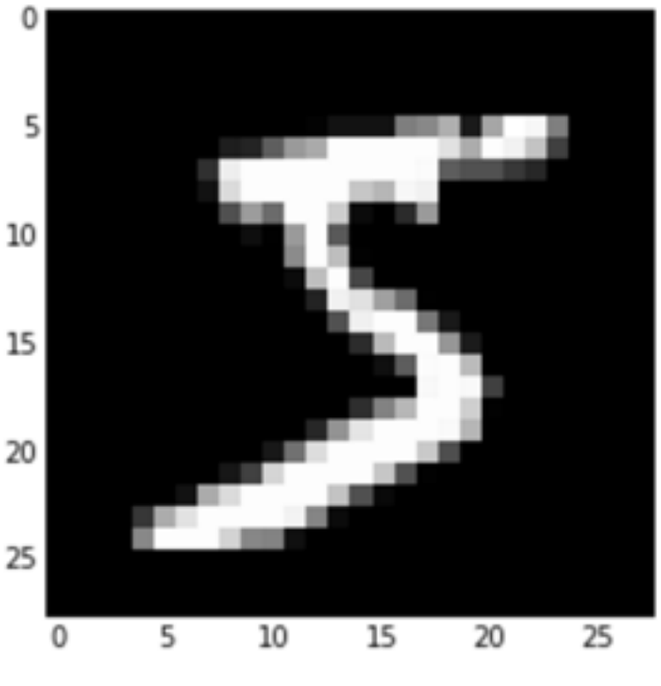
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
0s - loss: 0.2831 - acc: 0.9180 - val_loss: 0.1403 - val_acc: 0.9597
Epoch 2/10
0s - loss: 0.1111 - acc: 0.9675 - val_loss: 0.1000 - val_acc: 0.9708
Epoch 3/10
0s - loss: 0.0729 - acc: 0.9790 - val_loss: 0.0750 - val_acc: 0.9770
Epoch 4/10
0s - loss: 0.0497 - acc: 0.9855 - val_loss: 0.0754 - val_acc: 0.9762
Epoch 5/10
0s - loss: 0.0360 - acc: 0.9900 - val_loss: 0.0608 - val_acc: 0.9813
Epoch 6/10
0s - loss: 0.0258 - acc: 0.9935 - val_loss: 0.0600 - val_acc: 0.9810
Epoch 7/10
0s - loss: 0.0193 - acc: 0.9956 - val_loss: 0.0601 - val_acc: 0.9812
Epoch 8/10
0s - loss: 0.0141 - acc: 0.9968 - val_loss: 0.0554 - val_acc: 0.9813
Epoch 9/10
0s - loss: 0.0103 - acc: 0.9979 - val_loss: 0.0548 - val_acc: 0.9833
Epoch 10/10
0s - loss: 0.0077 - acc: 0.9987 - val_loss: 0.0553 - val_acc: 0.9831
```

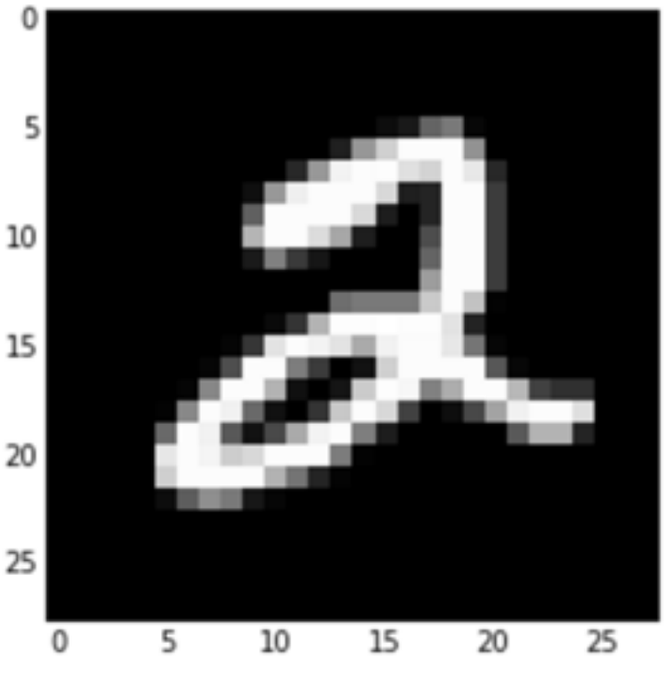
Here the network is being trained!

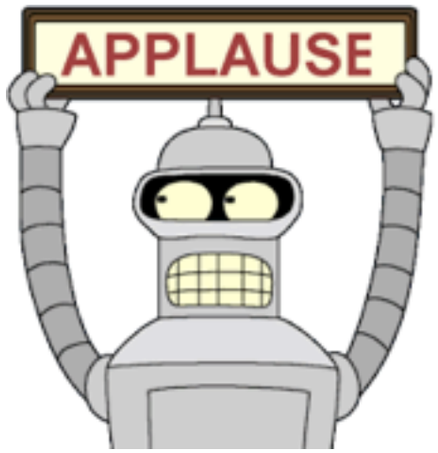
MNIST example

In [7]:
```
# plot the (root of) the prediction probability from 0 to 9
# (I used the root only to show up next best guesses better)
image_number_to_predict = 0
plt.imshow(
    raw_training_images[image_number_to_predict],
    cmap=plt.get_cmap('gray'),
    interpolation='nearest'
)
plt.show()
class_probabilities = model.predict(
    training_images[image_number_to_predict:image_number_to_predict+1]
)[0]
plt.bar(
    np.arange(0,10),
    np.sqrt(class_probabilities)
)
# Convert the probability prediction to an actual number.
predicted_number = np_utils.categorical_probas_to_classes(
    [class_probabilities]
)[0]
print "That looks like a", predicted_number
```

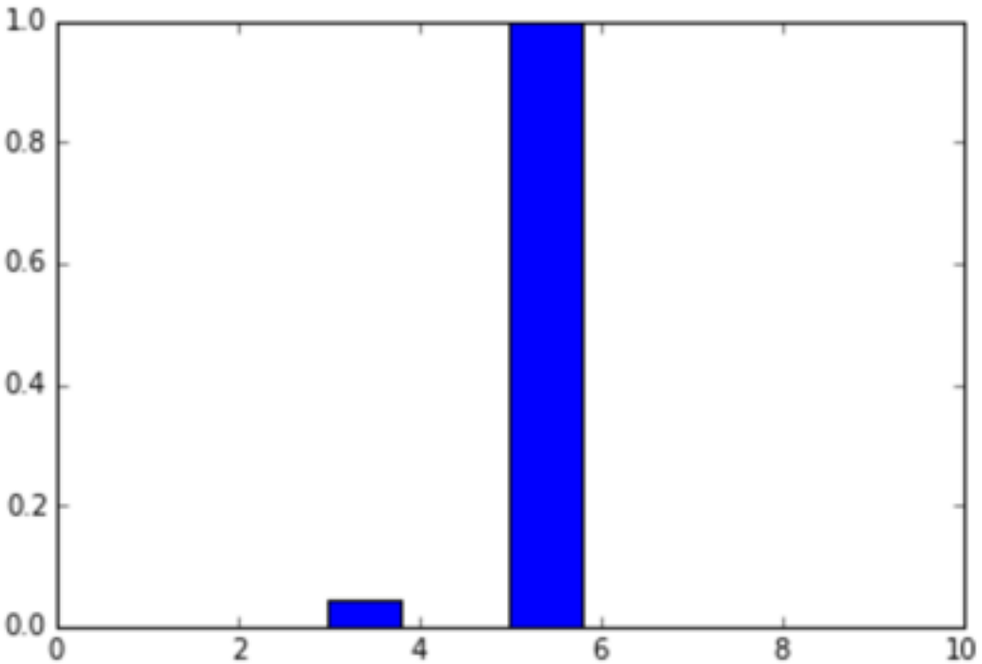
Introduction to machine vision using Keras



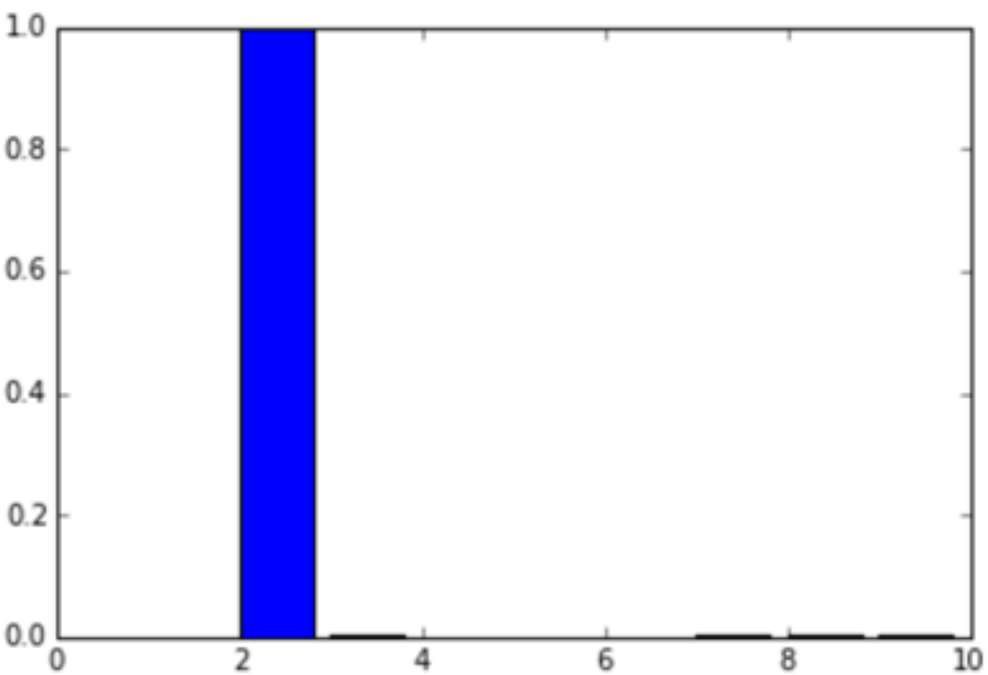




That looks like a 5

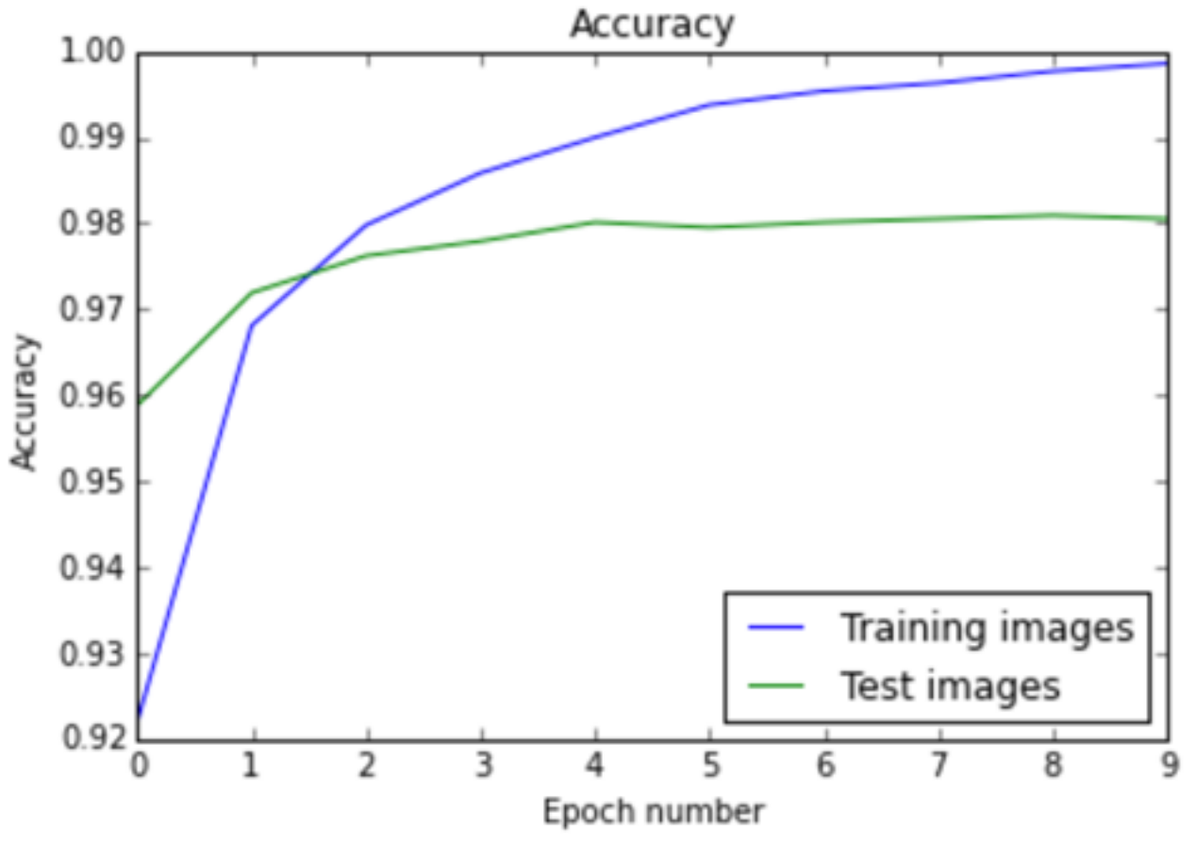


That looks like a 2



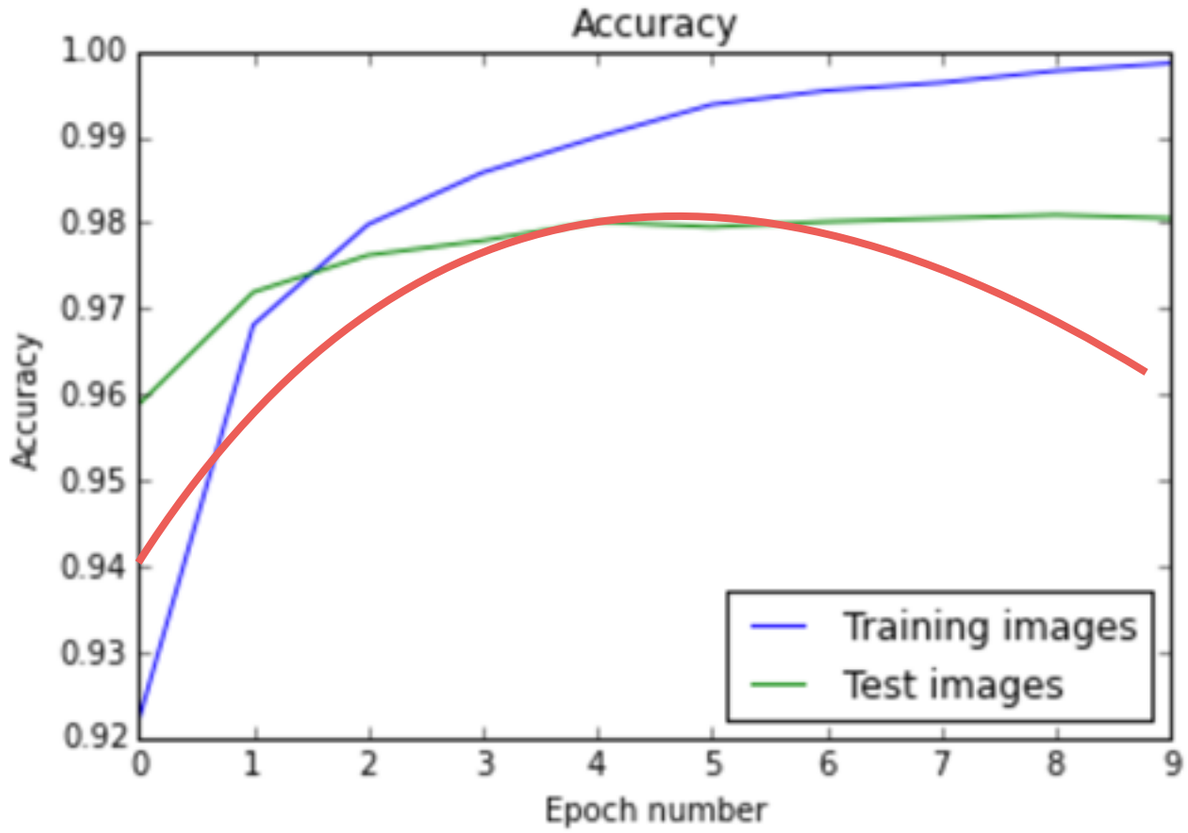
MNIST example

In [10]:
```
plt.plot(history['acc'])
plt.plot(history['val_acc'])
plt.title('Accuracy')
plt.xlabel('Epoch number')
plt.ylabel('Accuracy')
plt.legend(
    ['Training images', 'Test images'],
    loc='lower right'
)
plt.show()
```



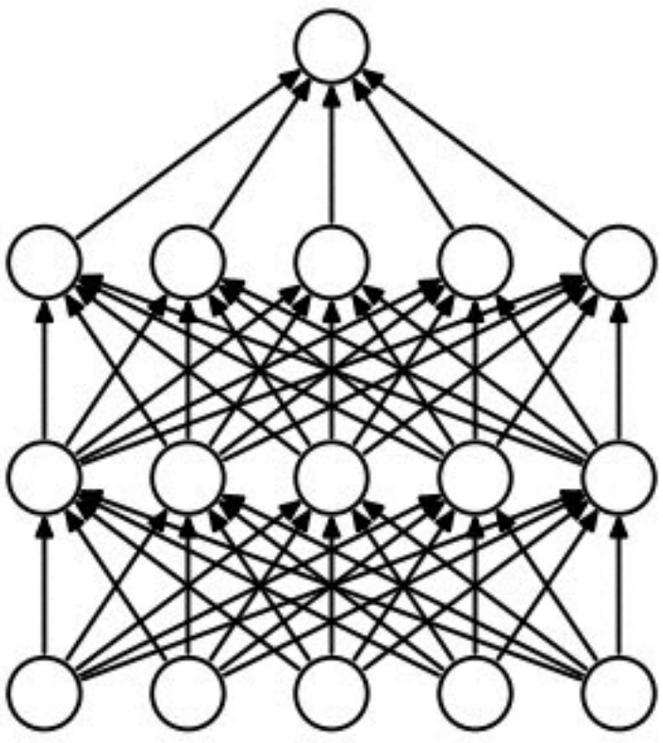
Pitfall: overfitting

- A bad case of over-fitting is shown in red.
- Notice that the training accuracy is higher than the test accuracy.
- This means the network is trained too specifically, over-fitted to the training data.

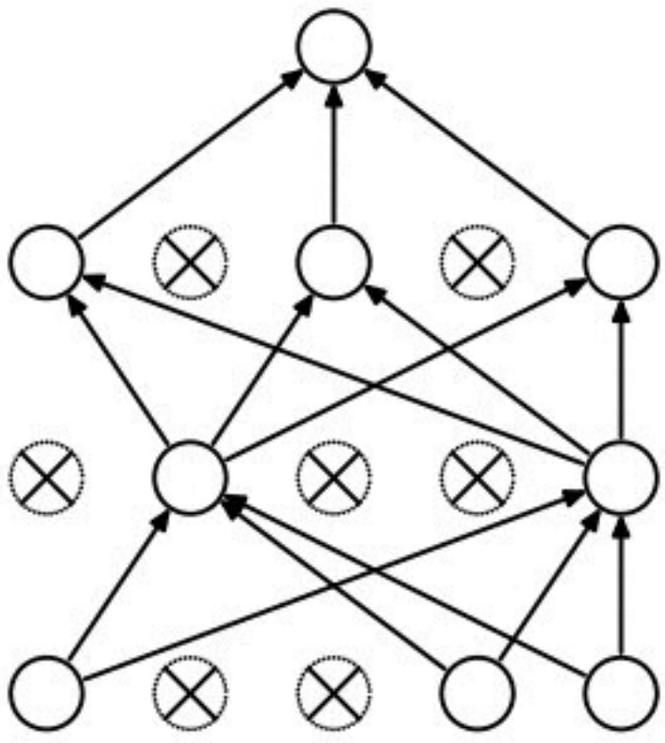


Dropout layer

- Randomly turns off cells in a layer.
- Helps prevent over-fitting.

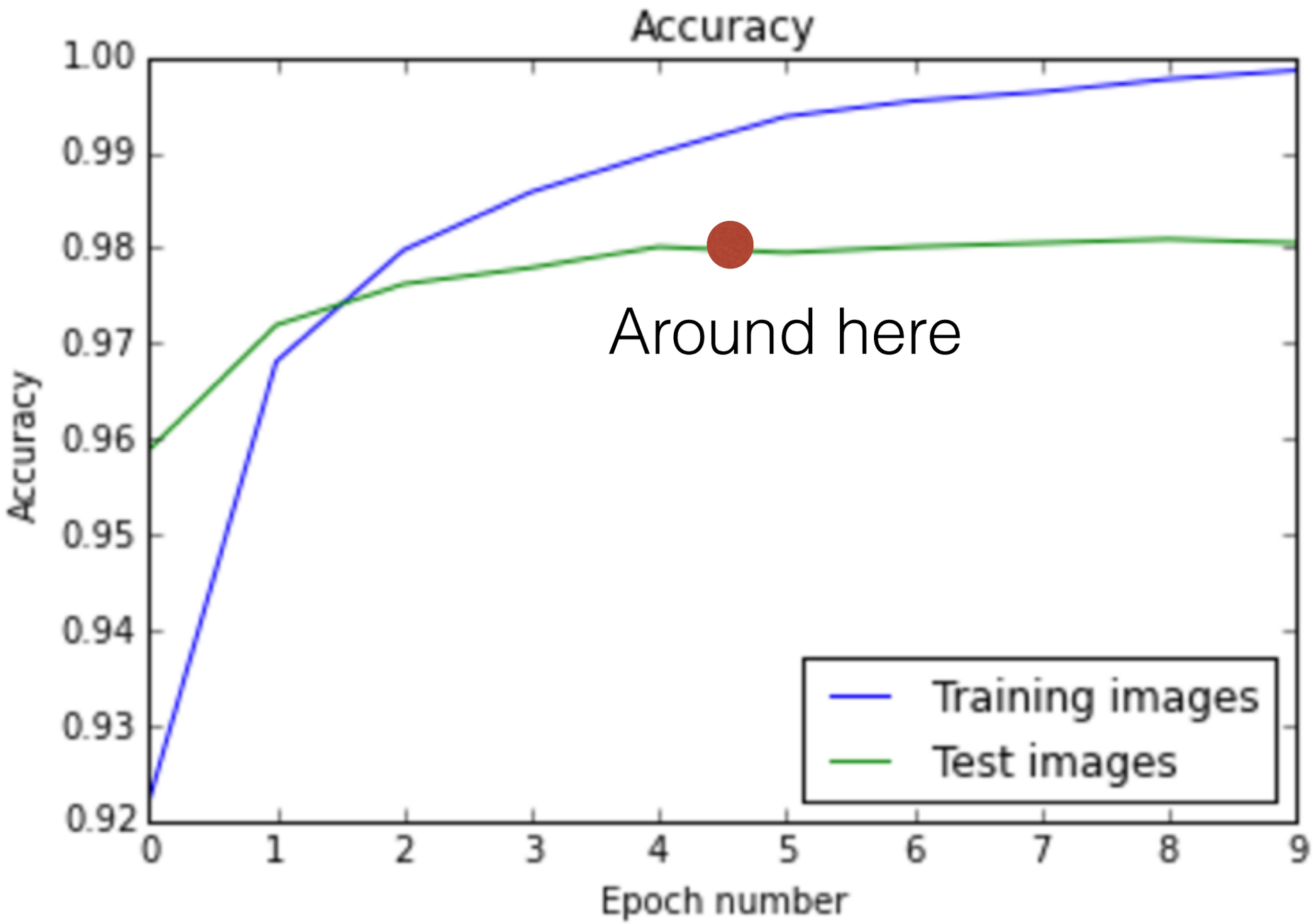


(a) Standard Neural Net



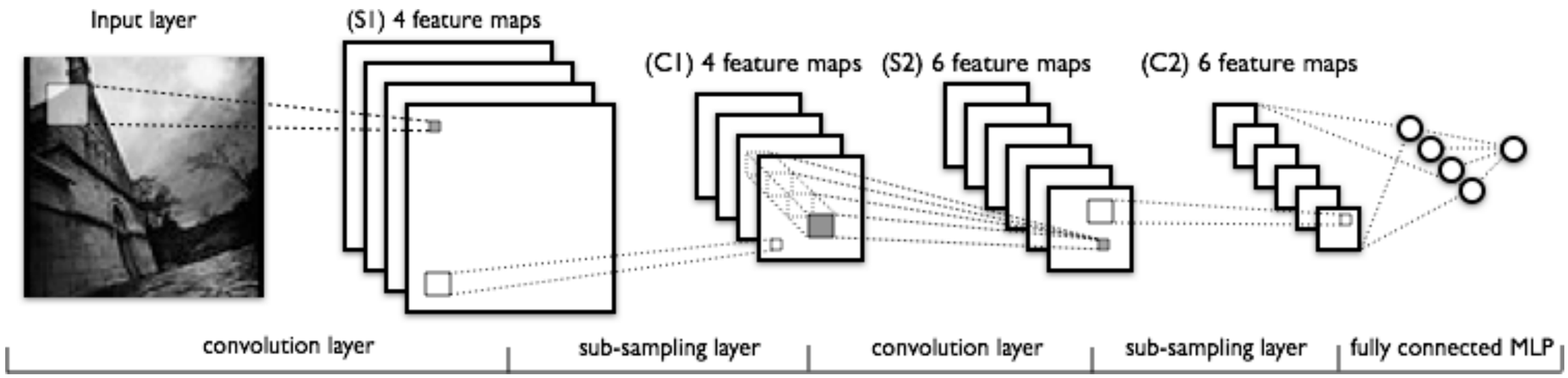
(b) After applying dropout.

When to stop training



Convolutional Layer

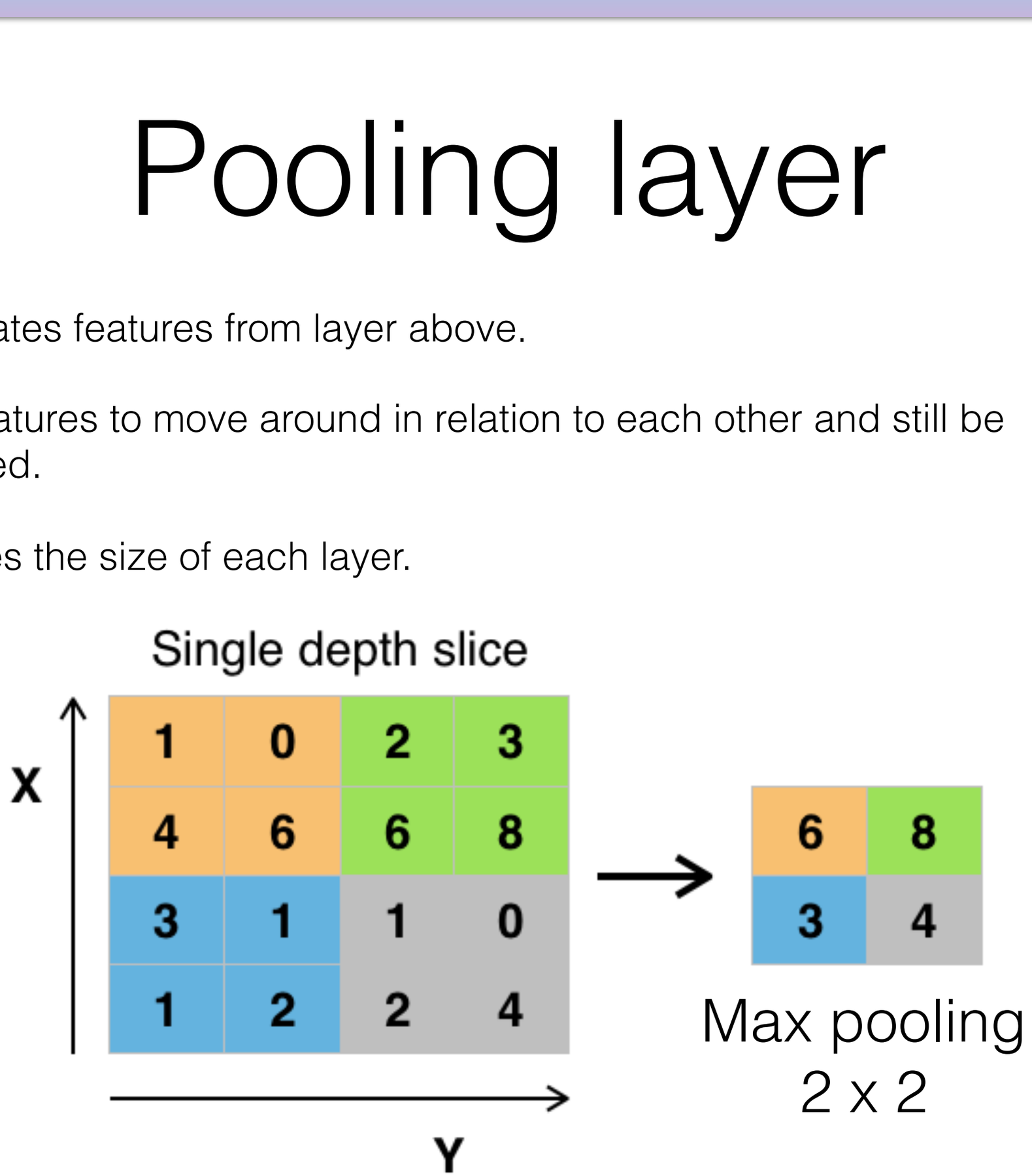
- Developed for object recognition tasks and natural language processing tasks.
- Far fewer parameters than dense neural network. Learns faster.



http://deeplearning.net/tutorial/lenet.html

Pooling layer

- Consolidates features from layer above.
- Allows features to move around in relation to each other and still be recognised.
- Decreases the size of each layer.



Convolutional example

```
model = Sequential()

model.add(Convolution2D(32,  # filters
                        3, 3,  # 3 x 3
                        border_mode='valid',
                        input_shape=(1, img_rows, img_cols)))
model.add(Activation('relu'))
model.add(Convolution2D(32, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

Improvement over dense network.

```
X_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
9s - loss: 0.3955 - acc: 0.8799 - val_loss: 0.1021 - val_acc: 0.9679
Epoch 2/12
9s - loss: 0.1483 - acc: 0.9561 - val_loss: 0.0660 - val_acc: 0.9793
Epoch 3/12
9s - loss: 0.1123 - acc: 0.9666 - val_loss: 0.0521 - val_acc: 0.9831
Epoch 4/12
9s - loss: 0.0944 - acc: 0.9722 - val_loss: 0.0456 - val_acc: 0.9840
Epoch 5/12
9s - loss: 0.0811 - acc: 0.9761 - val_loss: 0.0412 - val_acc: 0.9857
Epoch 6/12
9s - loss: 0.0720 - acc: 0.9784 - val_loss: 0.0404 - val_acc: 0.9866
Epoch 7/12
9s - loss: 0.0667 - acc: 0.9805 - val_loss: 0.0368 - val_acc: 0.9878
Epoch 8/12
9s - loss: 0.0604 - acc: 0.9813 - val_loss: 0.0342 - val_acc: 0.9889
Epoch 9/12
9s - loss: 0.0552 - acc: 0.9833 - val_loss: 0.0312 - val_acc: 0.9906
Epoch 10/12
9s - loss: 0.0537 - acc: 0.9842 - val_loss: 0.0340 - val_acc: 0.9886
Epoch 11/12
9s - loss: 0.0510 - acc: 0.9849 - val_loss: 0.0319 - val_acc: 0.9892
Epoch 12/12
9s - loss: 0.0492 - acc: 0.9861 - val_loss: 0.0373 - val_acc: 0.9883
Test score: 0.0373221943885
Test accuracy: 0.9883
```

Network pitfalls

Too large network or data, want to train quickly	Training can take a long time! Use more efficient GPUs. Leave running for 1 hour or overnight.
Too small network, never gets accurate.	Use a bigger network. It's good to start smallish though.
Not enough regularization / normalization.	You'll see slow/no learning or wild behaviour like un-learning.
Exploding/vanishing gradients problem. (Fundamental problem worth reading about).	Better weight initialisation, momentum/learning rate schedules. Fewer layers. Pre-trained networks

Hardware Issues

No GPU.	You need them for problems with lots of parameters. Can be OK for production.
Not enough GPU	More is usually better.
Wrong type of GPU.	CUDA is more compatible, compute capability >3.
Using EC2 GPU instances.	Good for trying tutorials & production. Not good for training on big datasets.
Not enough RAM.	Easy to use up 32 GB of RAM. Buy lots of cheap RAM. Max out the machine.

Human Pitfalls

- Spending too much time reading research papers.
- Not reading enough.
- Expecting amazing results quickly.
- Listening to Elon Musk.

Further reading

- https://keras.io/
- https://blog.keras.io/
- http://www.deeplearningbook.org/
- http://machinelearningmastery.com/deep-learning-books/
- http://cs231n.github.io/
  - especially http://cs231n.github.io/neural-networks-3/
- For fun: http://lossfunctions.tumblr.com/

Do try out some tutorials!