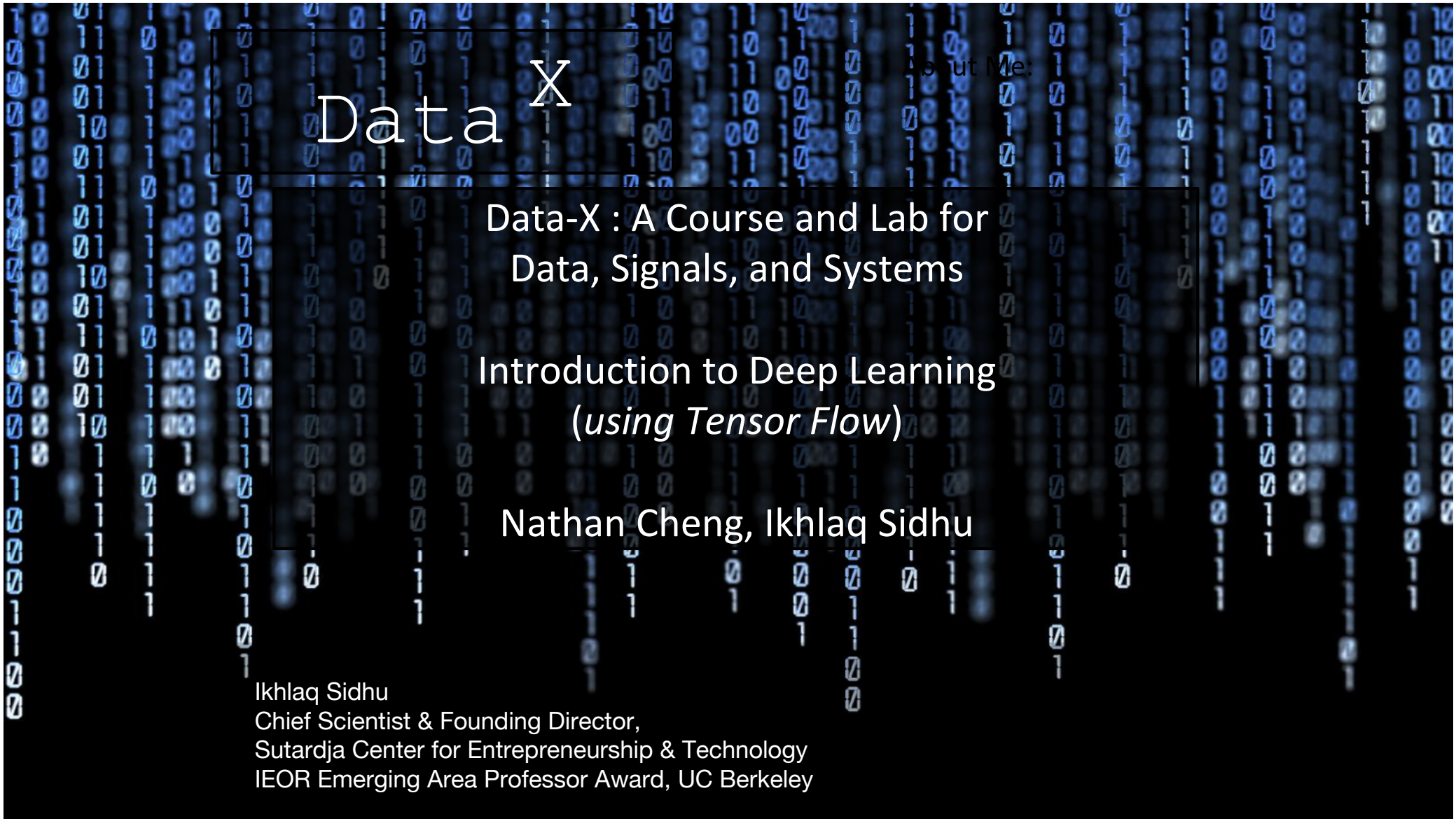
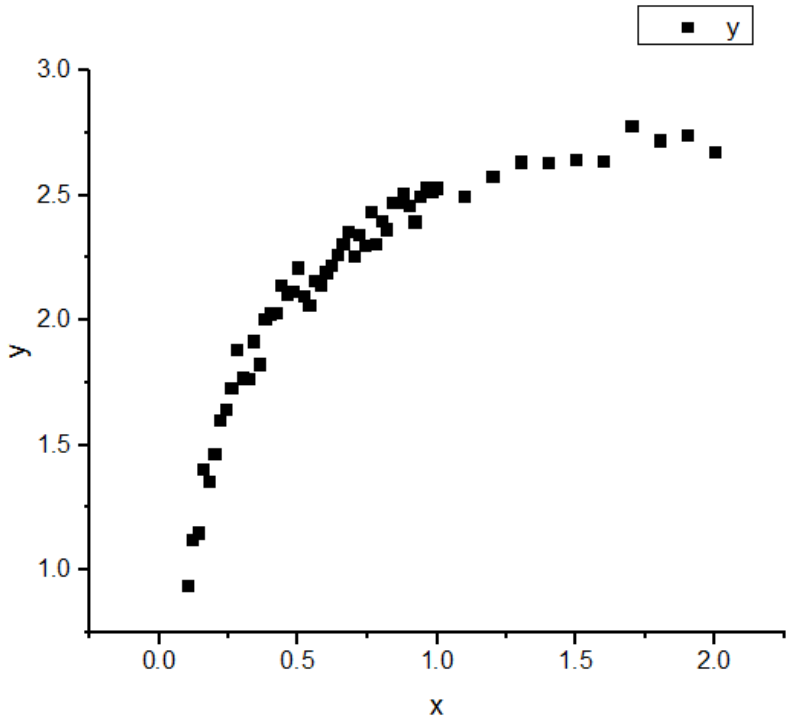

Data X
About Me:
Data-X : A Course and Lab for
Data, Signals, and Systems
Introduction to Deep Learning
(*using Tensor Flow*)
Nathan Cheng, Ikhlaq Sidhu
Ikhlaq Sidhu
Chief Scientist & Founding Director,
Sutardja Center for Entrepreneurship & Technology
IEOR Emerging Area Professor Award, UC Berkeley

What about nonlinear modeling?

- Given x, predict y
- Clearly, a straight line will fall short



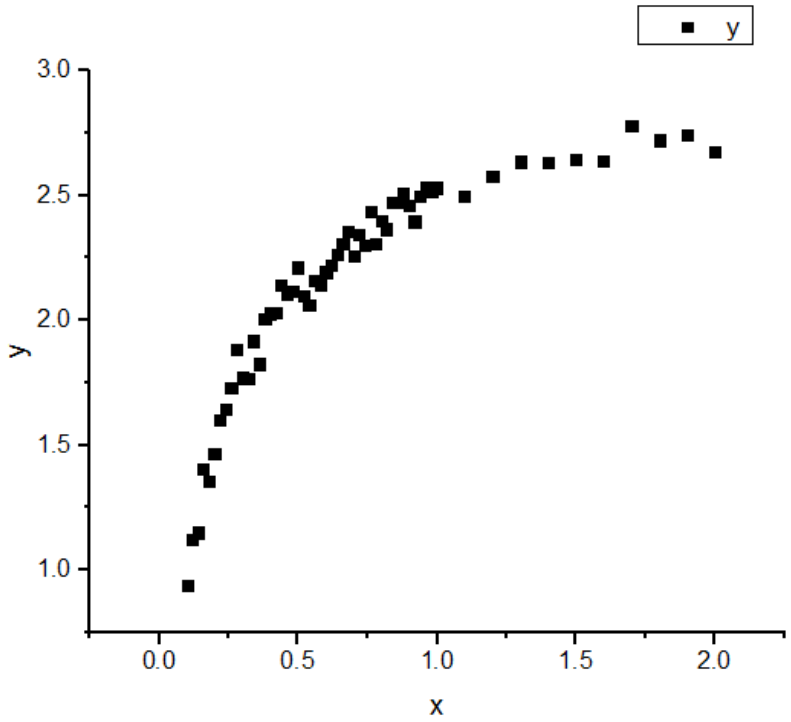
http://www.originlab.com/doc%5Cen/Tutorial/images/Linear_Fit_For_Kinetic_Models/Linear_Fit_Kinetic_Model_10.png

What about nonlinear modeling?

One Solution

- Increase number of parameters

$$f(x) = a_0 + a_1 x + a_2 x^2$$



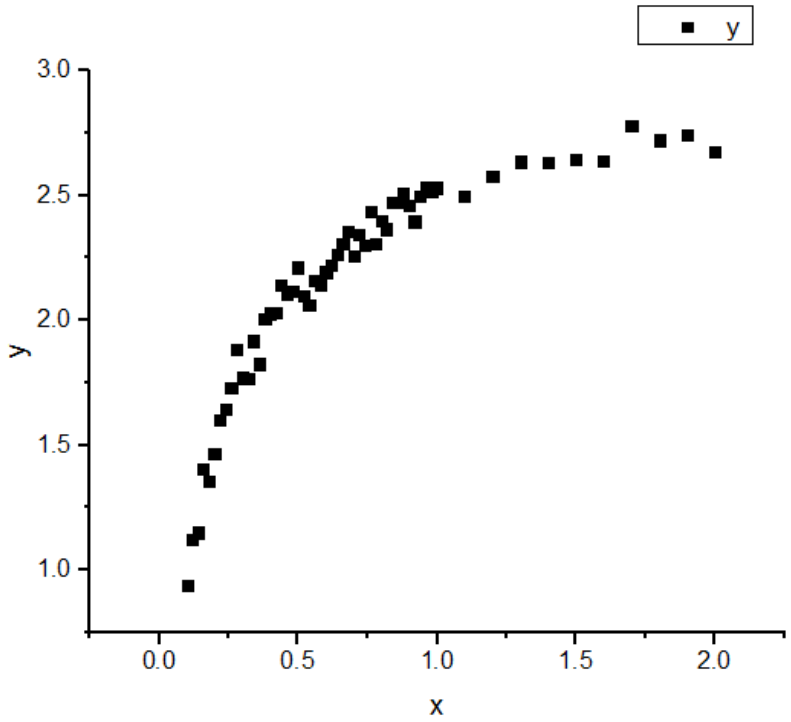
http://www.originlab.com/doc%5Cen/Tutorial/images/Linear_Fit_For_Kinetic_Models/Linear_Fit_Kinetic_Model_10.png

What about nonlinear modeling?

$$f(x) = a_0 + a_1 x + a_2 x^2$$

Problems

- Have to guess which parameters are useful to the model
- High risk of overfitting (if too many parameters are chosen, model may not generalize well)
- Becomes far too computationally expensive given a lot of features and higher degree parameters

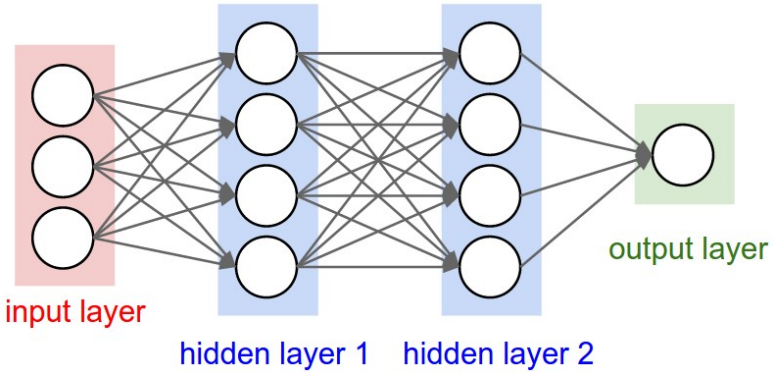


http://www.originlab.com/doc%5Cen/Tutorial/images/Linear_Fit_For_Kinetic_Models/Linear_Fit_Kinetic_Model_10.png

Deep learning: layering of linear and nonlinear classifiers

Neural Networks

Learning and information modeling inspired by the way neurons work in the brain



http://cs231n.github.io/assets/nn1/neural_net2.jpeg

Each circle is a neuron—takes inputs

- The neurons in the hidden layers are equipped with an activation function (will see an example later)

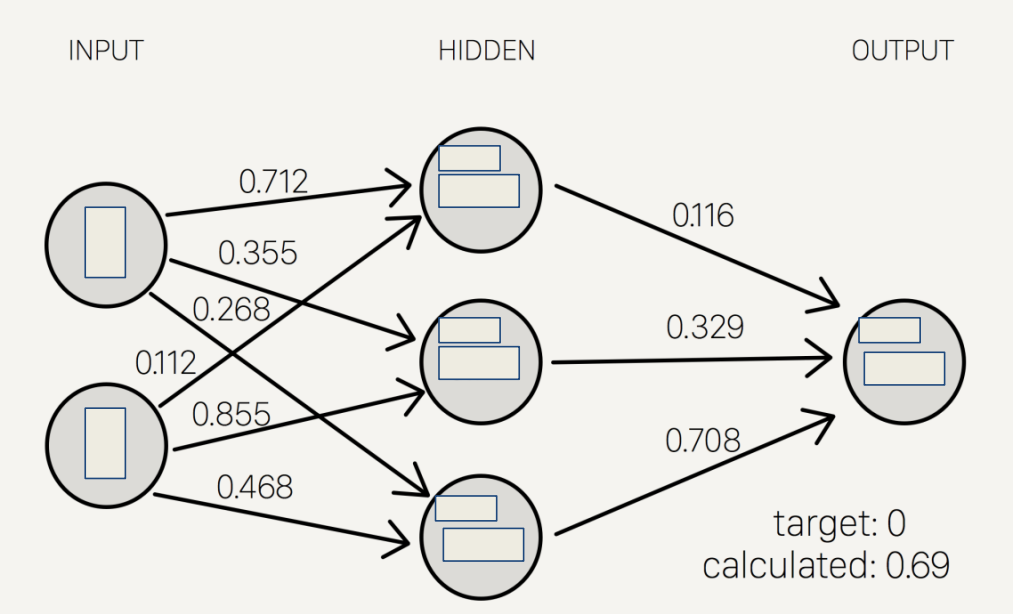
The arrows represent axons—weighs inputs

- Takes weighted sum over neurons

Example Network and Forward Propagation

Activation Function—
Logistic Sigmoid
Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



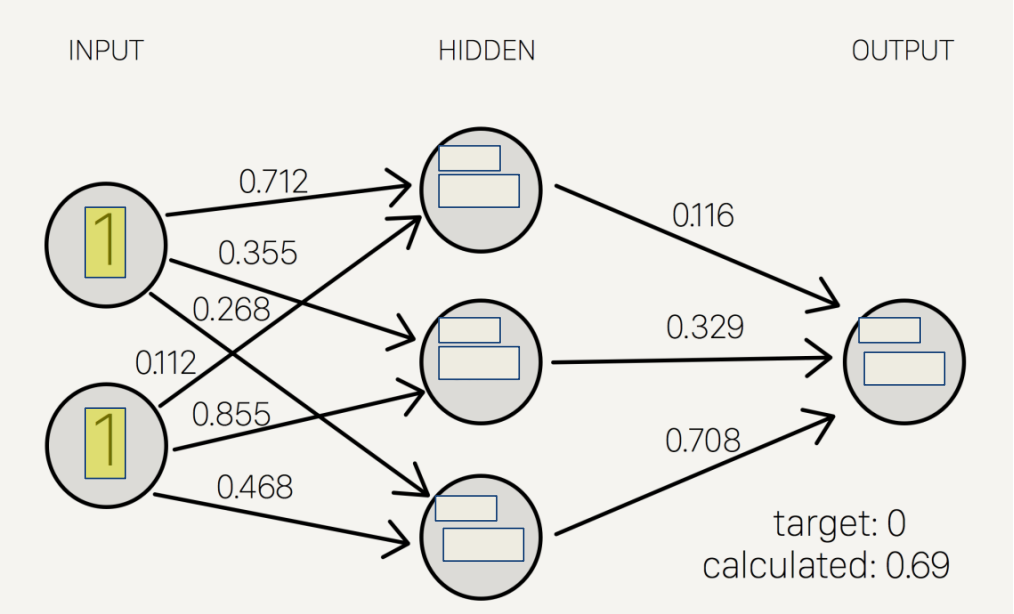
http://i.imgur.com/UNIffE1.png

This is the architecture of a neural network. In order to make a prediction, we do what is called forward propagation.

Example Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



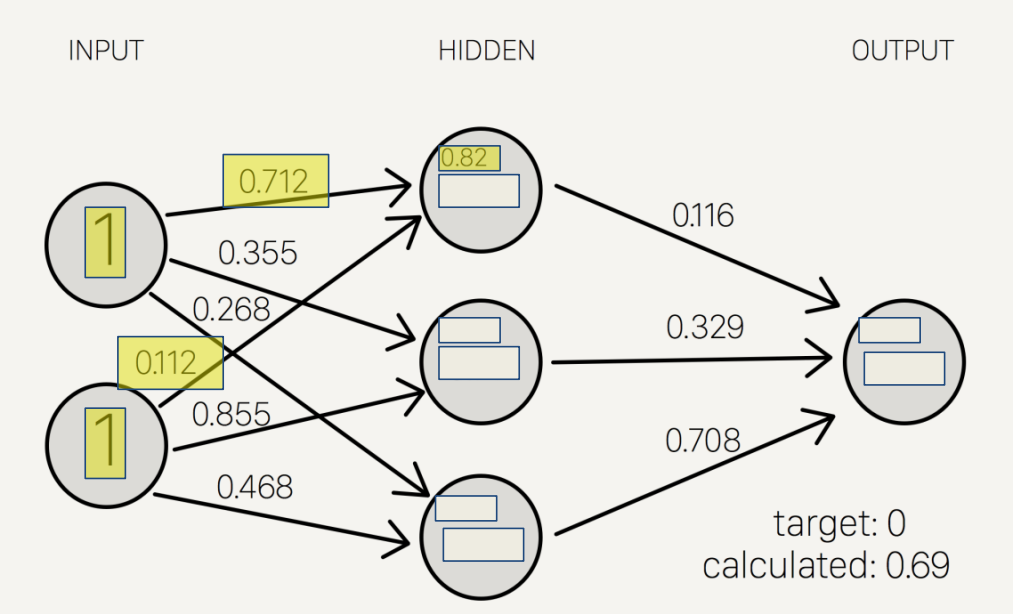
http://i.imgur.com/UNIffE1.png

First, we feed in our inputs into the input layer. In this case, we have two features, each with a value of 1.

Example Network and Forward Propagation

Activation Function—
Logistic Sigmoid
Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



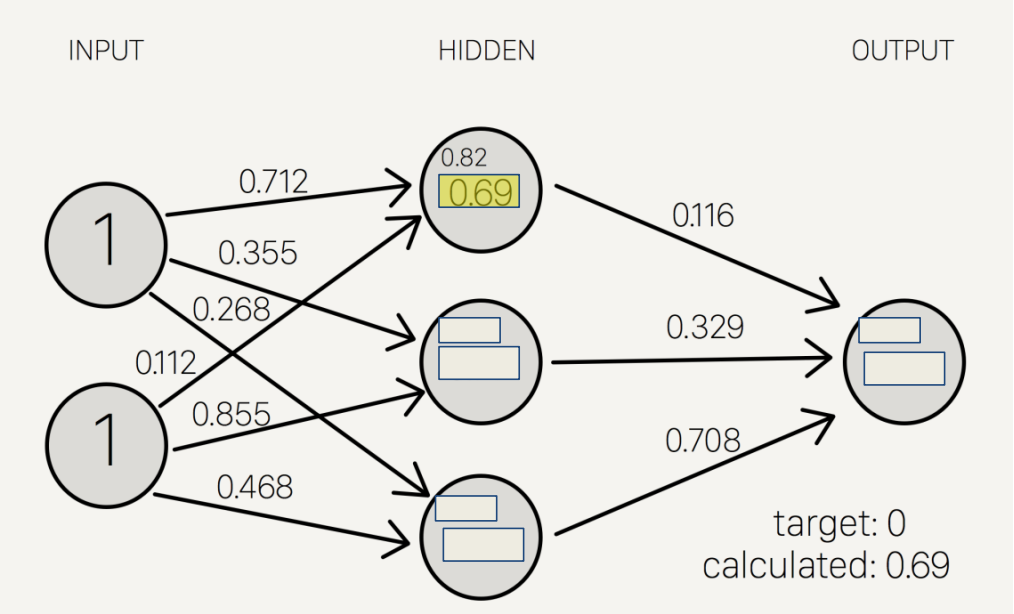
http://i.imgur.com/UNIffE1.png

The relevant axon weights will then take weighted sums and send them to the next layer. In this case, the first hidden neuron gets a value of 1*(0.712) + 1*(0.112)

Example Network and Forward Propagation

Activation Function—
Logistic Sigmoid
Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



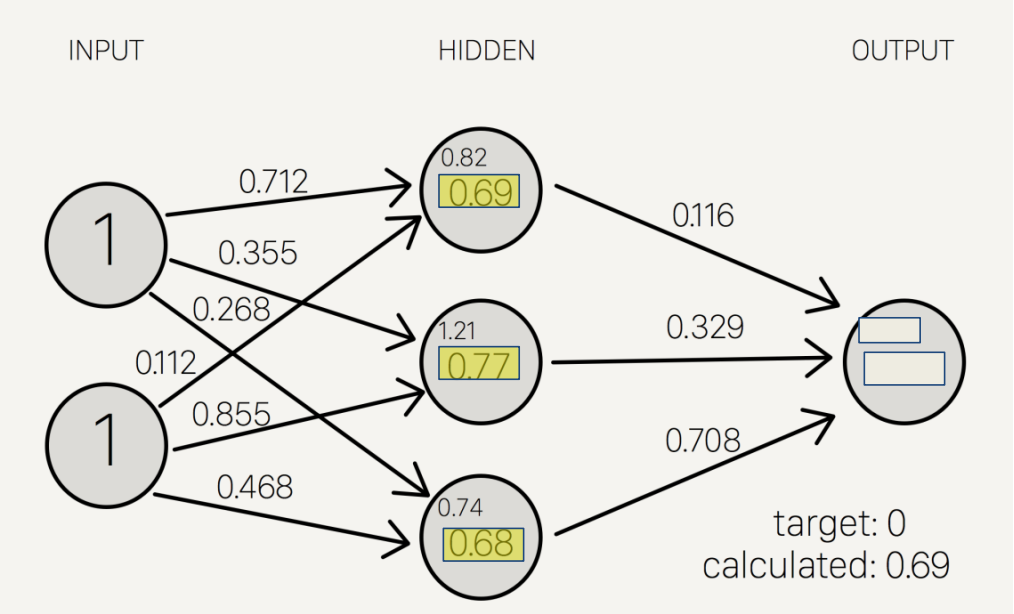
http://i.imgur.com/UNIffE1.png

But 0.82 is only the pre-activation value. We need to then apply our activation function to 0.82, giving us f(0.82) = 0.69. This is the value for the neuron.

Example Network and Forward Propagation

Activation Function—
Logistic Sigmoid
Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



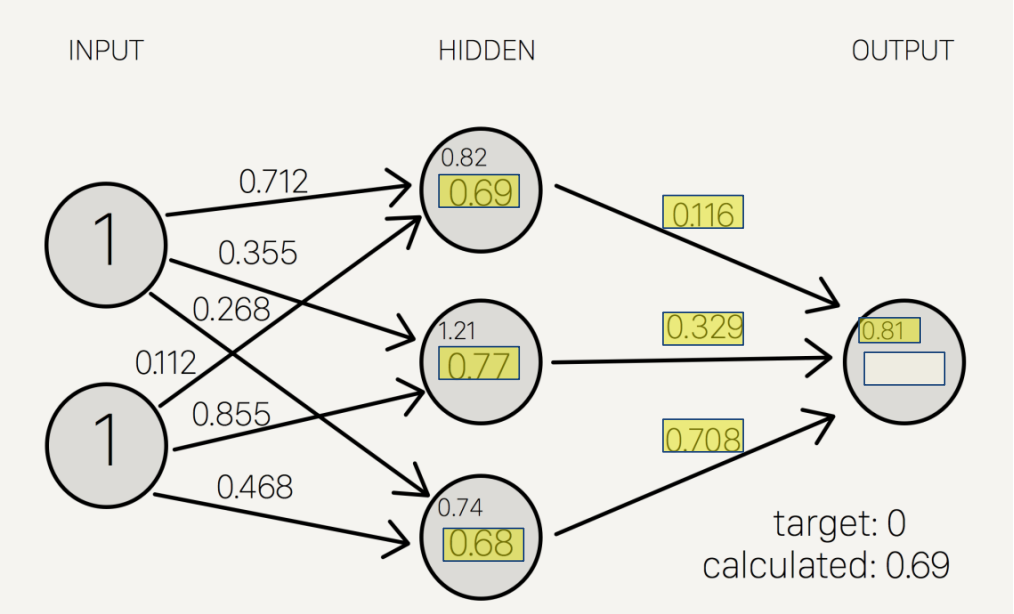
http://i.imgur.com/UNIffE1.png

The same process is applied to each neuron.

Example Network and Forward Propagation

Activation Function—
Logistic Sigmoid
Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



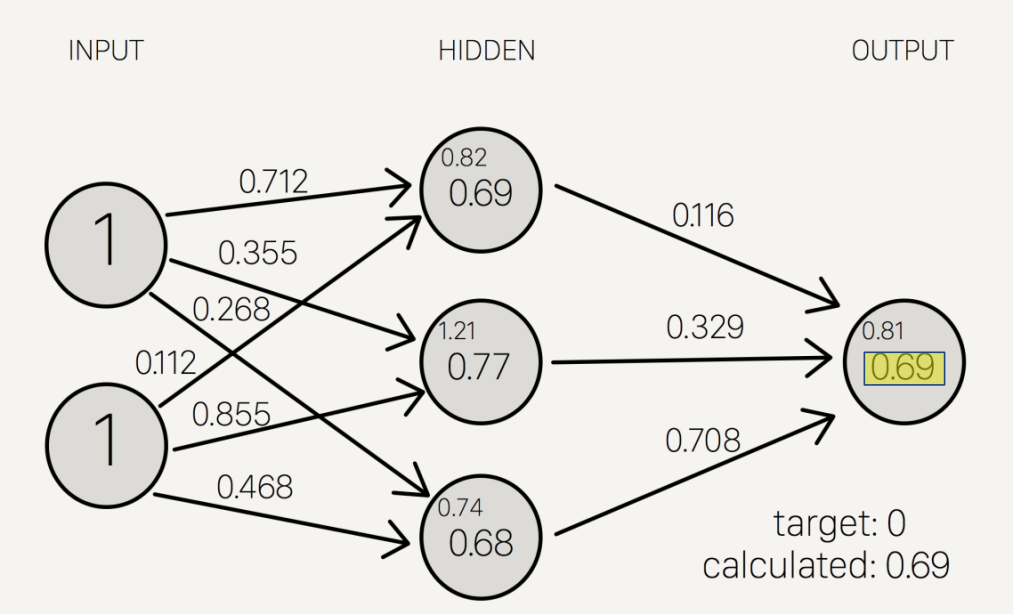
http://i.imgur.com/UNIffE1.png

0.69*(0.116) + 0.77*(0.329) + 0.68*(0.708) = 0.81

Example Network and Forward Propagation

Activation Function—
Logistic Sigmoid
Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



http://i.imgur.com/UNIffE1.png

f(0.81) = 0.69. This is the output, or prediction, or our model.

Mathematical Representation

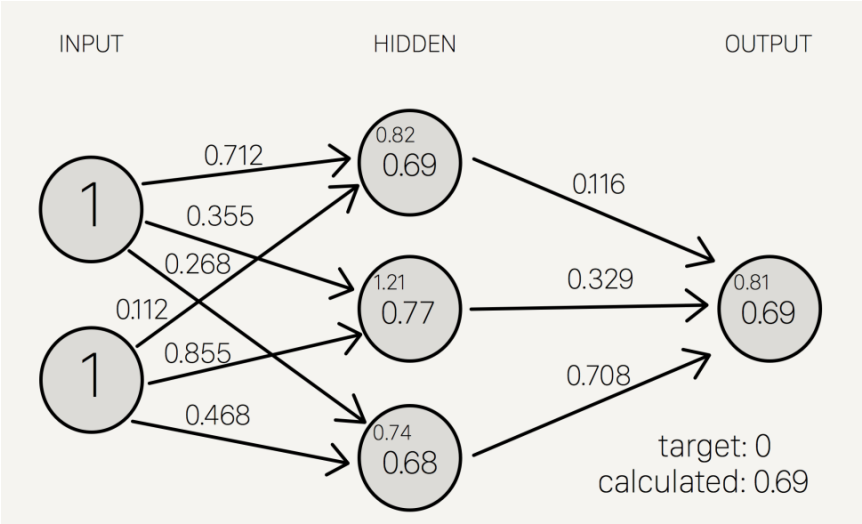
We can represent this model using matrices, for example:

X gets input values:

$$X = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

W gets axon weights:

$$W = \begin{bmatrix} 0.712 & 0.355 & 0.268 \\ 0.112 & 0.855 & 0.468 \end{bmatrix}$$



http://i.imgur.com/UNIffE1.png

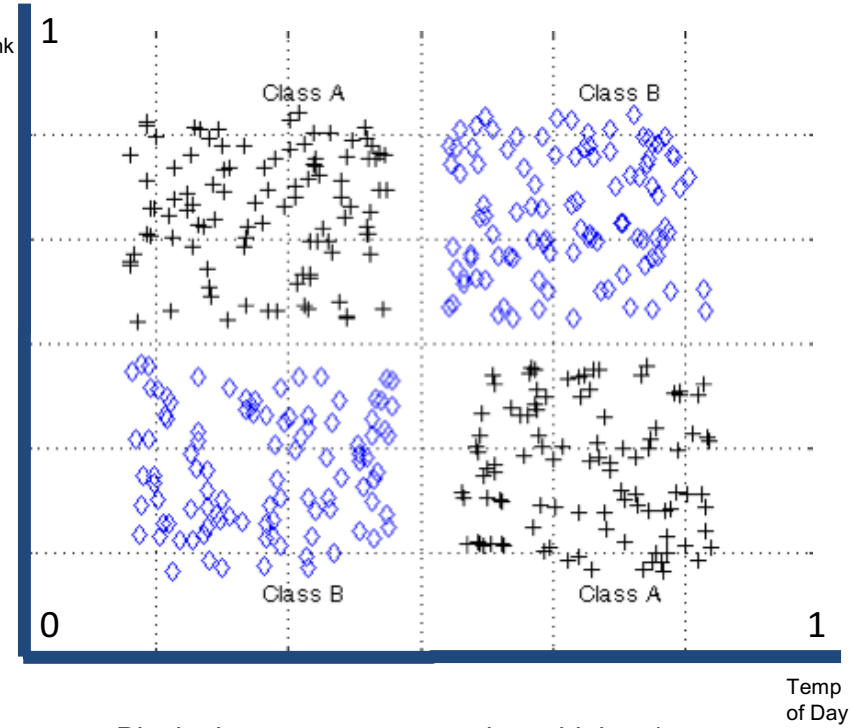
One step of forward propagation looks like:

$$XW = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0.712 & 0.355 & 0.268 \\ 0.112 & 0.855 & 0.468 \end{bmatrix} = \begin{bmatrix} 0.82 & 1.21 & 0.74 \end{bmatrix}$$

$$f(\begin{bmatrix} 0.82 & 1.21 & 0.74 \end{bmatrix}) = \begin{bmatrix} 0.69 & 0.77 & 0.68 \end{bmatrix}$$

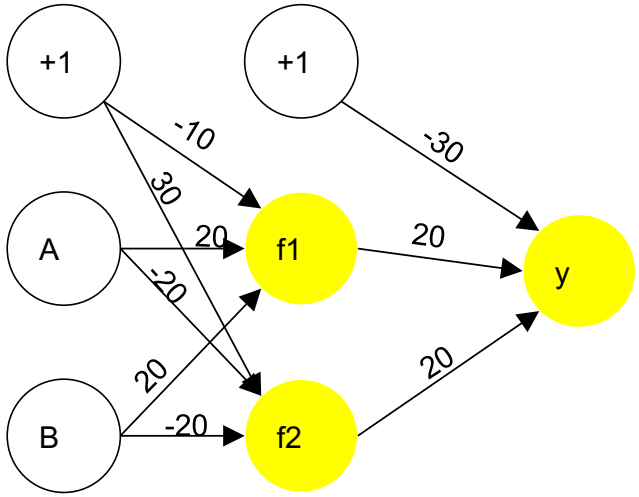
A Toy Problem: XOR

- XOR(A,B) evaluates to "true" if either A or B are "true" but not both.
- Potential manifestation of this property:
  - Predicting whether a customer will enjoy a drink given A=temperature of the day and B=temperature of the drink
- Clearly, this dataset is not linearly separable



- Black pluses = customer enjoys drink = 1
- Blue diamonds = customer does not enjoy drink = 0
- (0=really cold, 1=really hot)

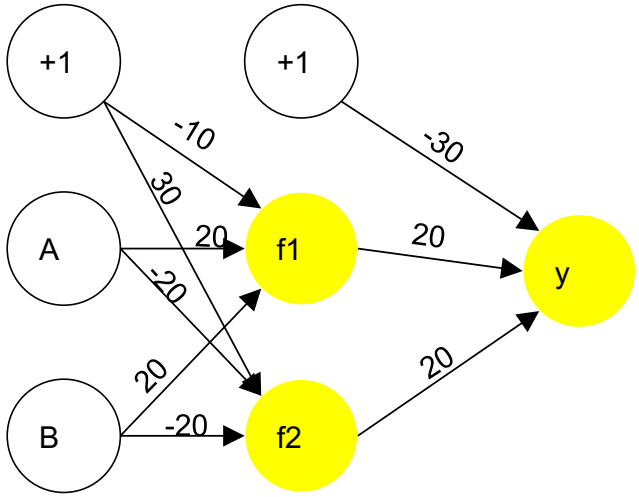
http://lab.fs.uni-lj.si/lasin/wp/IMIT_files/neural/nn06_rbfn_xor/html/nn06_rbfn_xor_2_newrb_01.png



Yellow neurons are equipped with the logistic sigmoid activation function f(t) = 1 / (1+e^-t)

- f(>10) approx = 1
- f(<-10) approx = 0
- y represents level of enjoyment

A	B	f1	f2	y
0	0	f(-10) = 0	f(30) = 1	f(-10) = 0
0	1	f(10) = 1	f(10) = 1	f(10) = 1
1	0	f(10) = 1	f(10) = 1	f(10) = 1
1	1	f(30) = 1	f(-10) = 0	f(-10) = 0



Yellow neurons are equipped with the logistic sigmoid activation function f(t) = 1 / (1+e^-t)

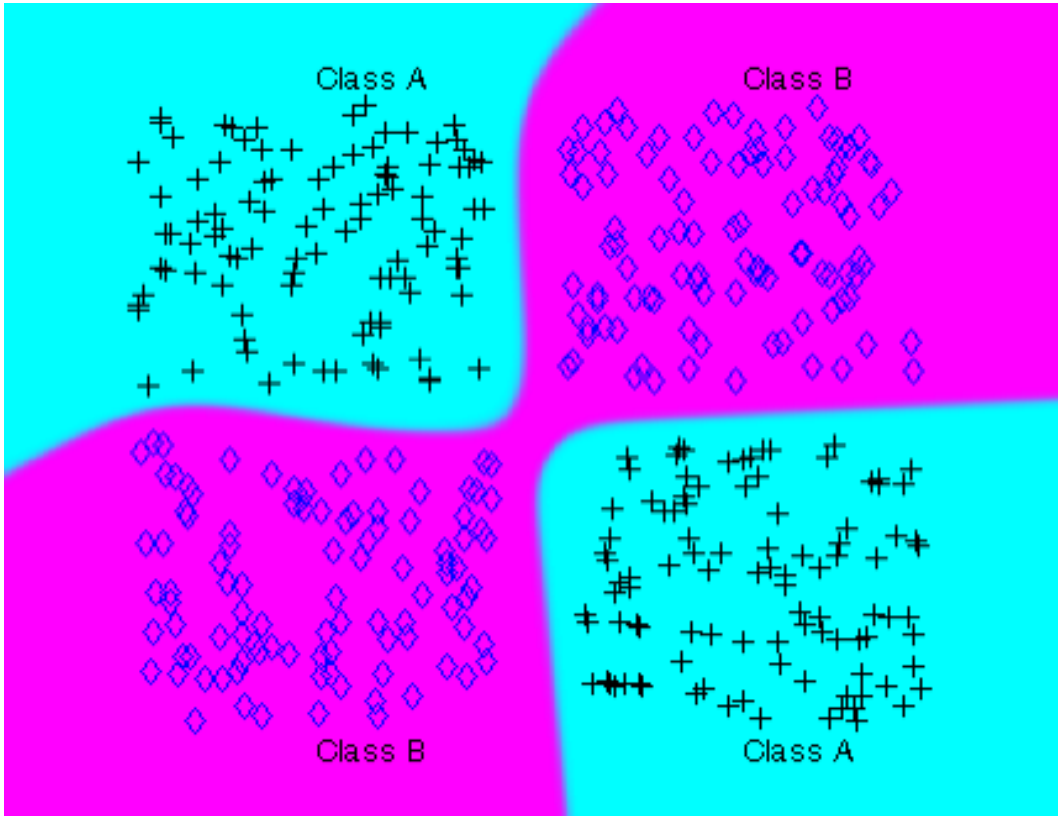
- f(>10) approx = 1
- f(<-10) approx = 0
- y represents level of enjoyment

A	B	f1	f2	y
0	0	f(-10) = 0	f(30) = 1	f(-10) = 0
0	1	f(10) = 1	f(10) = 1	f(10) = 1
1	0	f(10) = 1	f(10) = 1	f(10) = 1
1	1	f(30) = 1	f(-10) = 0	f(-10) = 0

NOTE: XOR(A,B) = (A or B) and (not (A and B))

- f1 = A or B
- f2 = not (A and B)
- y = f1 and f2

Insight: we can see neural networks as a way of representing complex logical decisions



http://lab.fs.uni-lj.si/lasin/wp/IMIT_files/neural/nn04_mlp_xor/nn04_mlp_xor_04.png

Training Our Network—Backpropagation

- How do we train our model?
  - The goal of training a neural network is to find the optimal axon weights such that the average error (defined by some loss function) between predicted outputs and actual outputs in a dataset is minimized
  - Recall that axon weights can be represented by weight matrices
- An algorithm known as backpropagation (backward propagation of errors) was invented to train neural networks
  - Backpropagation is mathematically very rigorous, specific details are probably not too important
  - Conceptually: per training example set (input, correct output):
    - Calculate the error between the prediction and the actual output using some loss function
    - Do the equivalent of forward propagation using the error, but starting at the end of the model and propagating towards the input layers, keeping track of the changing error values
    - The weighted error values at each layer tells us, in a way, how much the layer is contributing to the overall error. They can also be used to calculate the gradient of the network
    - This is key, as we can now perform gradient descent or some other optimization algorithm
  - This sounds like a pain. Fortunately, Tensor Flow takes care of most of the difficult stuff behind the scenes

Loss Functions

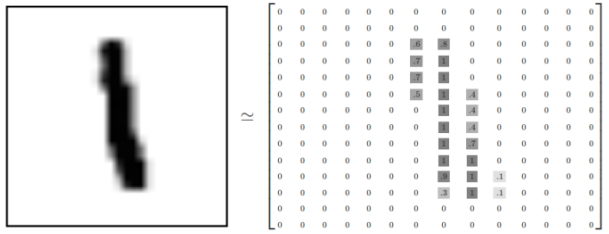
- In order to represent error, we need to choose a loss function
  - Loss function takes in two values, the actual input and the prediction, and returns the "error" between them
  - Rules of thumb:
    - Loss functions in neural networks should be differentiable
    - If the actual input and prediction are equal then the loss should be 0
- There are countless loss functions to choose from. Here are some popular ones:

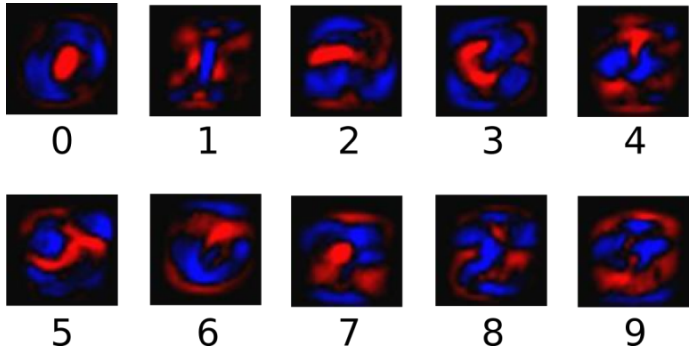
- Cross Entropy. Good for classification. Source: Wikipedia
  - Error = $-y \log \hat{y} - (1-y) \log(1-\hat{y})$
  - where y = target value (either 0 or 1) and y^ = the predicted value
  - this can be extended to vectors
- Mean Squared Error. Good for regression. Source: Wikipedia

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

- Loss functions are important—they determine what your network learns

Code and Tensor Flow





*We will first demonstrate Tensor Flow using a linear model. The following example is technically not “deep learning” yet, but it is easy to see how to generalize this example to more complex network architectures

**The following example and images are taken from https://www.tensorflow.org/versions/r0.9/tutorials/mnist/beginners/index.html

Import data and setup network graph

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import tensorflow as tf

x = tf.placeholder(tf.float32, [None, 784])

W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

y = tf.nn.softmax(tf.matmul(x, W) + b)

y_ = tf.placeholder(tf.float32, [None, 10])
```

Setup loss and training functions

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Evaluating accuracy

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Initialize network and train

```
init = tf.initialize_all_variables()

sess = tf.Session()
sess.run(init)

for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

This model achieves about 92% accuracy

End of Section