INTRO TO DATA SCIENCE
LECTURE 22:NEURAL NETWORKS

KEY CONCEPTS - WHAT DO WE NEED NEURAL NETWORKS

- Very good a creating complex non-linear decision boundaries in very high dimensional feature space

Key Concepts - History and Background

- Biologically motivated and developed in the 1970s
- Very widely used in the 1980s, and 90s
- Late 1990s popularity waned, and the Non-linear SVM became the more popular technique
- Relatively recent resurgence as more understanding of previous limitations became understood and solvable

Key Concepts - History and Background

- One thing that was not apparent until recently was that neural networks are computationally intensive, and to work at scale requires significant computational power

- In the 1990s this was under-appreciated and the computational power was simply not available

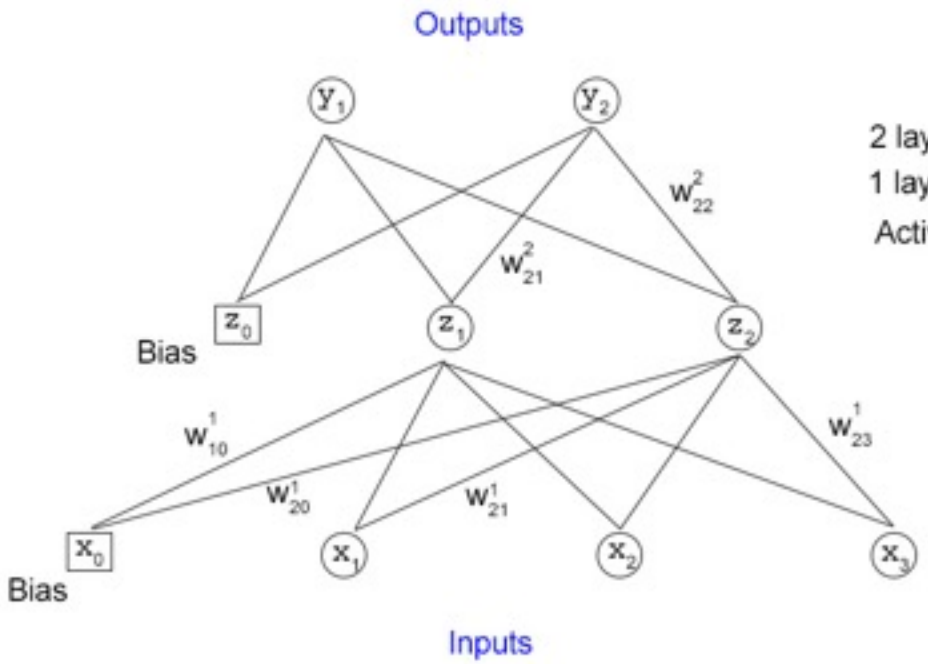
- Focus on one particular type of network - the multi-layered perceptron (MLP)

KEY CONCEPTS - HISTORY AND BACKGROUND - MLP

Output Layer

Hidden Layer

Input Layer



2 layers of adaptive weights

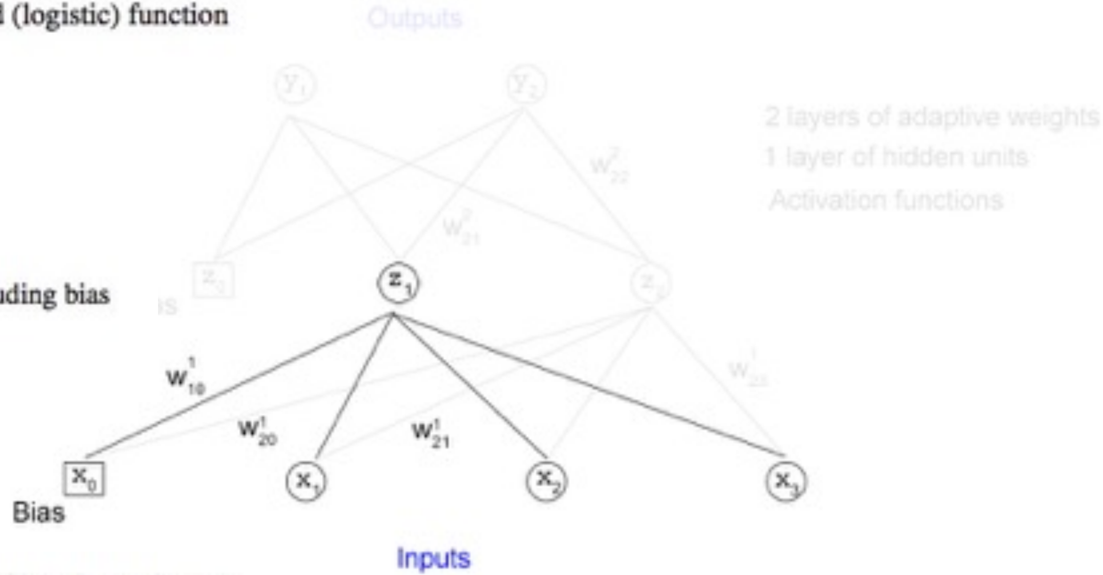
1 layer of hidden units

Activation functions

$z_1 = g(a_1)$ where $g()$ is the activation function. Usually the sigmoid (logistic) function

$$g(x) = \frac{1.0}{1.0 + e^{-x}}$$

$$a_1 = \sum_{i=0}^{d} w_{1i}^1 x_i$$ where d is the number of inputs in the input layer, including bias

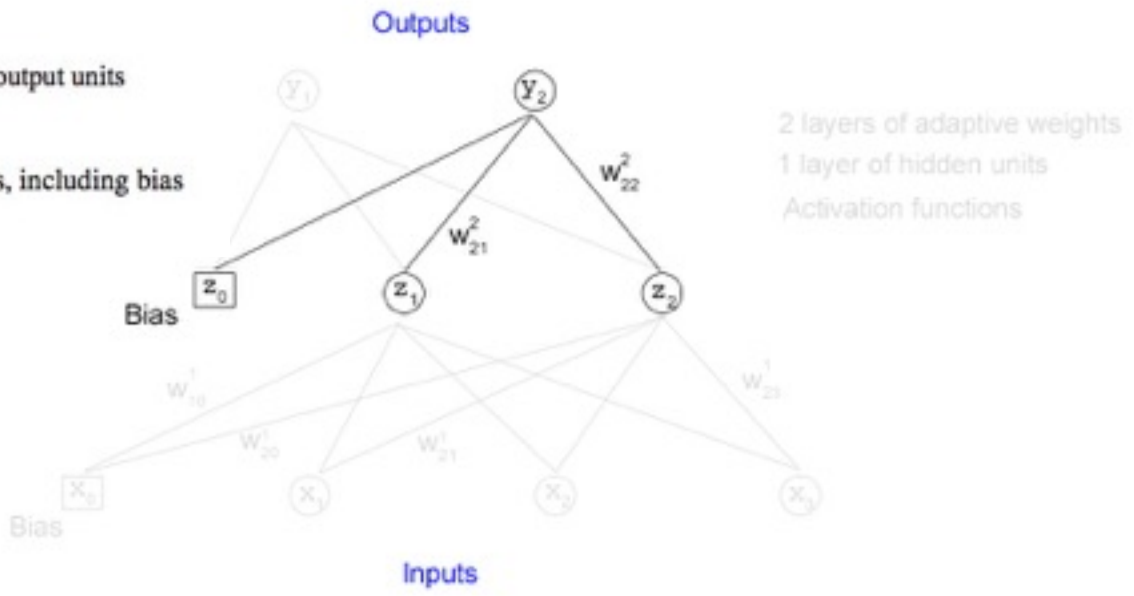


$x_0 = 1$, All bias units are always 1

KEY CONCEPTS - HISTORY AND BACKGROUND - MLP

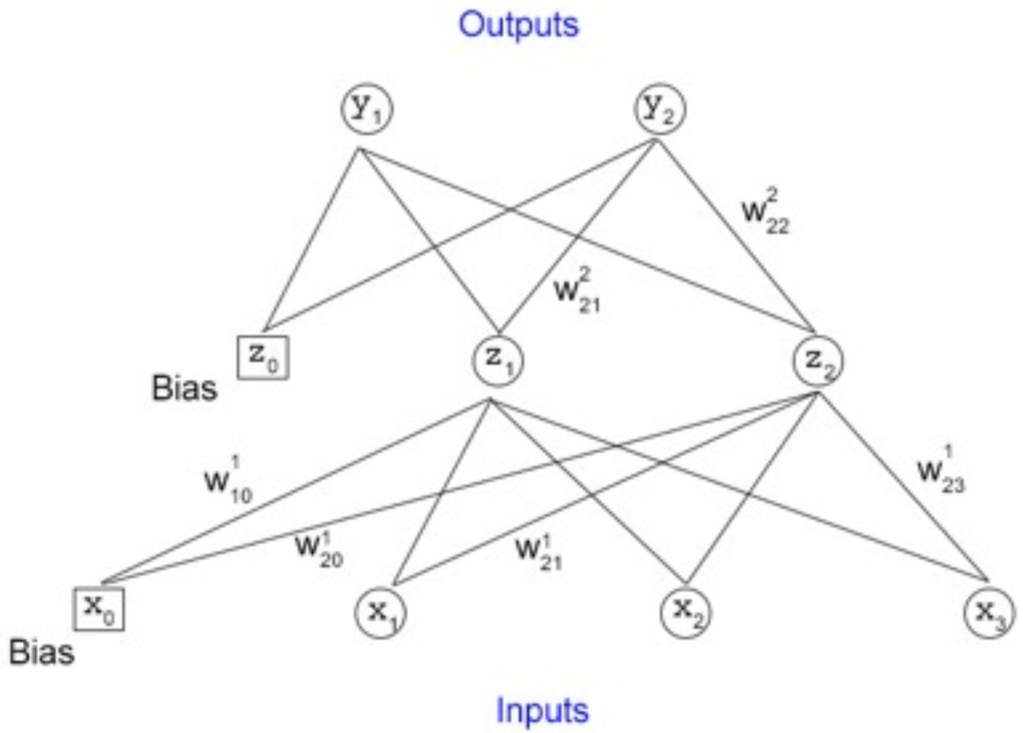
$y_2 = g^{'}(a_2)$ where $g^{'}()$ is the activation function of the output units

$$a_2 = \sum_{j=0}^{M} w_{2j}^2 z_j$$ where M is the number of hidden units, including bias



KEY CONCEPTS - HISTORY AND BACKGROUND - MLP

$$y_k = g'\left[\sum_{j=0}^{M} w_{kj}^{L2} \times g\left(\sum_{i=0}^{d} w_{ji}^{L1} x_i\right)\right]$$



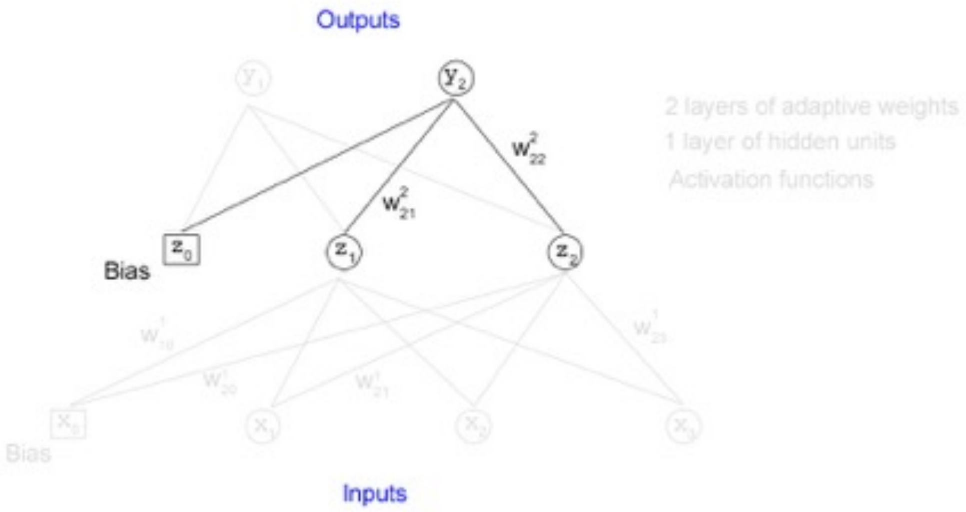
Key Concepts - History and Background - MLP

- The output activation function:

- Is usually linear, and not the logistic function

- y2 = g'(a2) = a2

$$a_2 = \sum_{j=0}^{M} w_{2j}^{2} z_j$$ where M is the number of hidden units, including bias



KEY CONCEPTS - HISTORY AND BACKGROUND - MLP

- But we want the outputs of the network to be probabilities over the mutually exclusive classes

- For example, if you have 3 classes, class A, class B, and class C, and output y1 = 0.95; then this means that there is a 95% chance of the input belonging to class A

- p(class A) + p(class B) + p(class C) = 1.0

- This can be achieved using the softmax function

- Softmax is a generalization of the logistic activation function, and is sometimes called the normalized exponential

$$y_k = \frac{e^{a_k}}{\sum_{k'} e^{a_{k'}}}$$

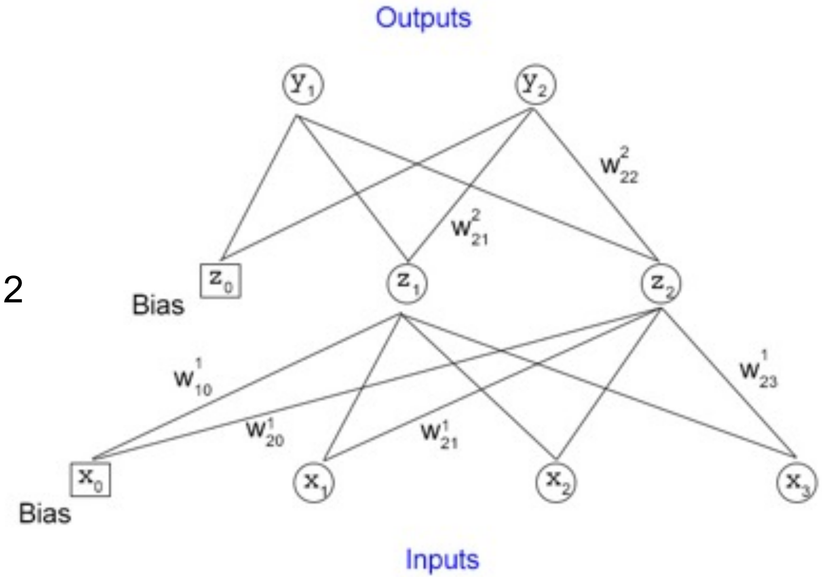
KEY CONCEPTS - HISTORY AND BACKGROUND - MLP

The calculation of an output given an input is called forward propagation

Output Layer

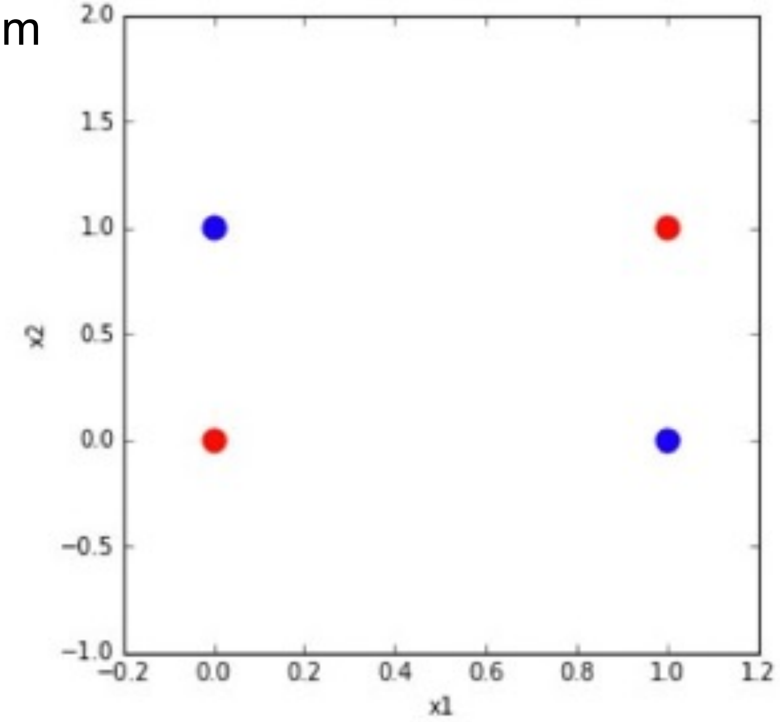
Hidden Layer 2

Input Layer



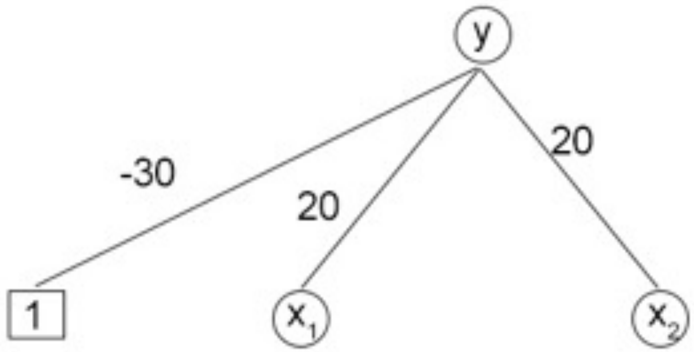
Key Concepts - Neural Networks Create Non-linear Decision Boundaries - Example

The XOR problem



x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

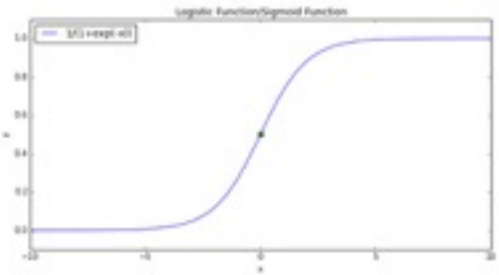
Key Concepts - Computing the Logical 'AND' Function



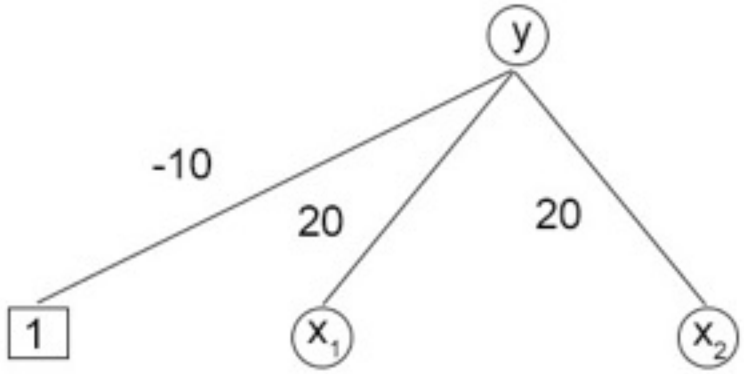
$$y = g(-30 + 20x_1 + 20x_2)$$

x1	x2	y
0	0	g(-30) ~ 0
0	1	g(-10) ~ 0
1	0	g(-10) ~ 0
1	1	g(10) ~ 1

g() is the sigmoid or logistic function

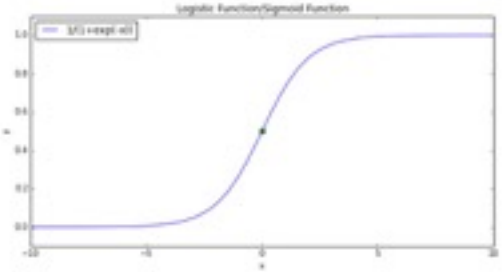


KEY CONCEPTS - COMPUTING THE LOGICAL 'OR' FUNCTION

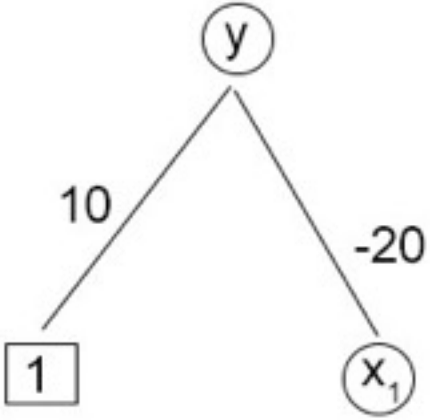


x1	*x2*	*y*
0	*0*	*g(-10) ~ 0*
0	*1*	*g(10) ~ 1*
1	*0*	*g(10) ~ 1*
1	*1*	*g(30) ~ 1*

$$y = g(-10 + 20x_1 + 20x_2)$$

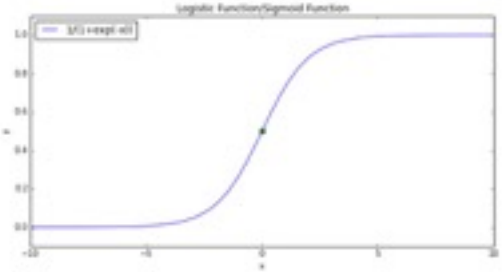


Key Concepts - Computing the Logical 'NOT' Function

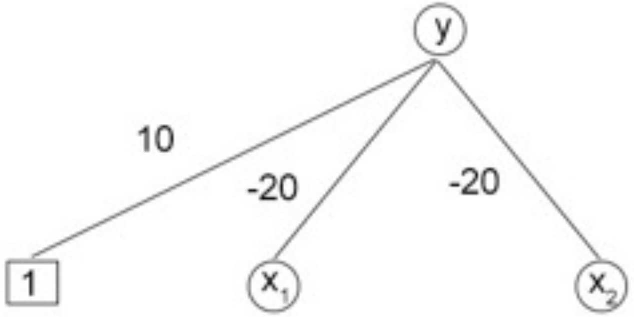


x1	y
0	g(10) ~ 1
1	g(-10) ~ 0

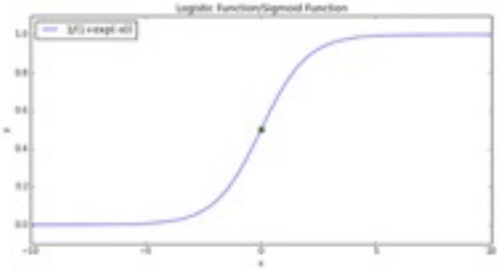
$$y = g(10 - 20x_1)$$



Key Concepts - Computing the Logical '(NOT x1) AND (NOT x2)' Function

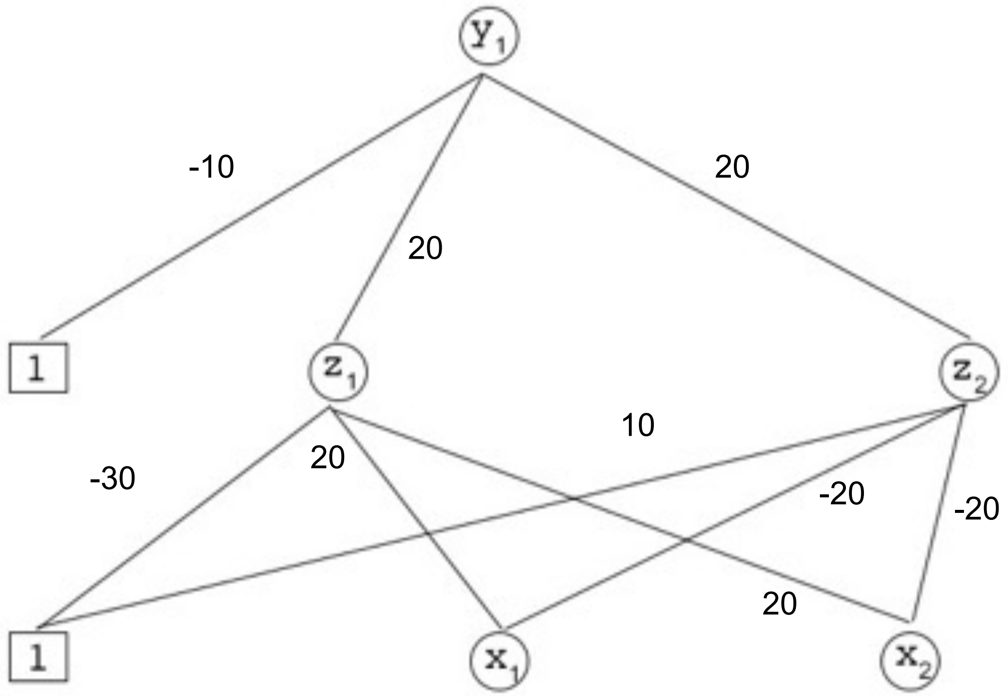


$$y = g(10 + -20x_1 + -20x_2)$$



x1	*x2*	*y*
0	*0*	*g(10) ~ 1*
0	*1*	*g(-10) ~ 0*
1	*0*	*g(-10) ~ 0*
1	*1*	*g(-30) ~ 0*

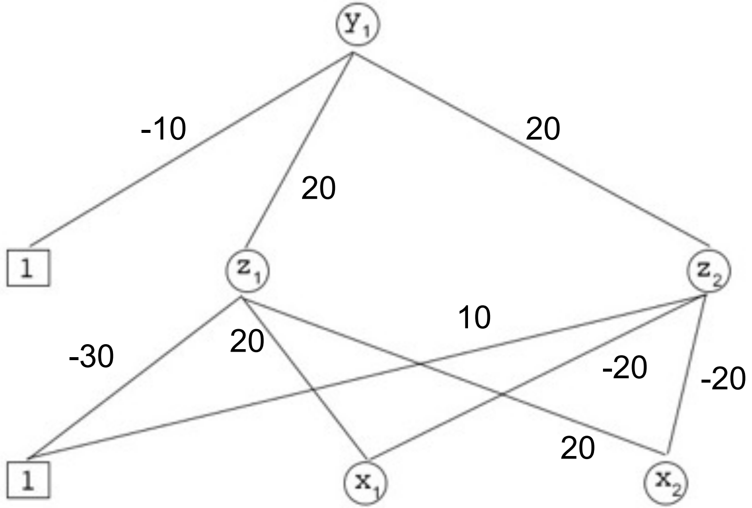
KEY CONCEPTS - COMPUTING THE LOGICAL 'XOR' FUNCTION



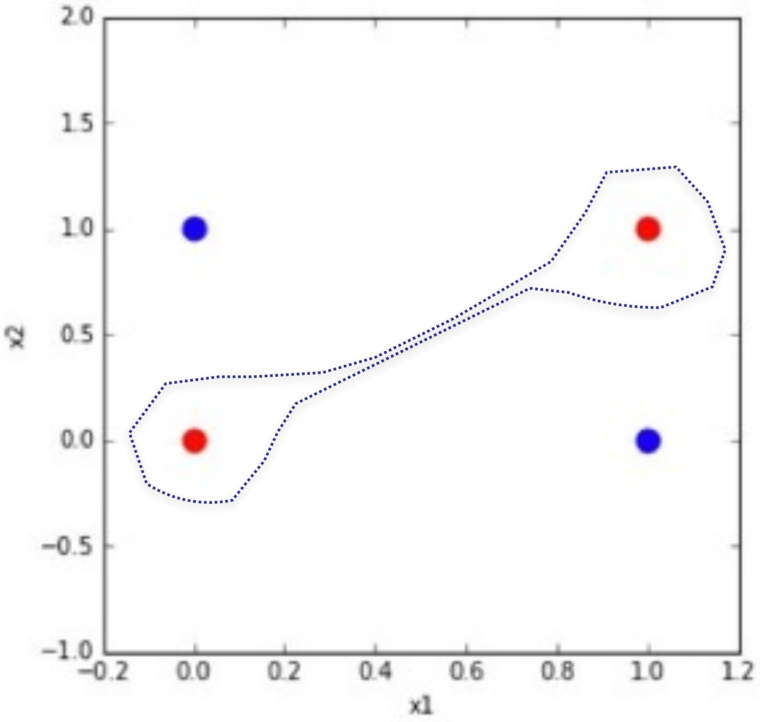
KEY CONCEPTS - COMPUTING THE LOGICAL 'XOR' FUNCTION

```
for x1 in range(0, 2):
    for x2 in range(0, 2):
        z1 = 1.0/(1.0 + np.exp(-(-30 + 20*x1 + 20*x2)))
        z2 = 1.0/(1.0 + np.exp(-(10 - 20*x1 -20*x2)))
        y = 1.0/(1.0 + np.exp(-(-10 + 20*z1 + 20*z2)))
        print "x1 = {:1.0f},  x2 = {:1.0f},  z1 = {:1.0f},  z2 = {:1.0f},  y = {:1.0f}".format(x1, x2, z1, z2, y)
```

```
x1 = 0,  x2 = 0,  z1 = 0,  z2 = 1,  y = 1
x1 = 0,  x2 = 1,  z1 = 0,  z2 = 0,  y = 0
x1 = 1,  x2 = 0,  z1 = 0,  z2 = 0,  y = 0
x1 = 1,  x2 = 1,  z1 = 1,  z2 = 0,  y = 1
```

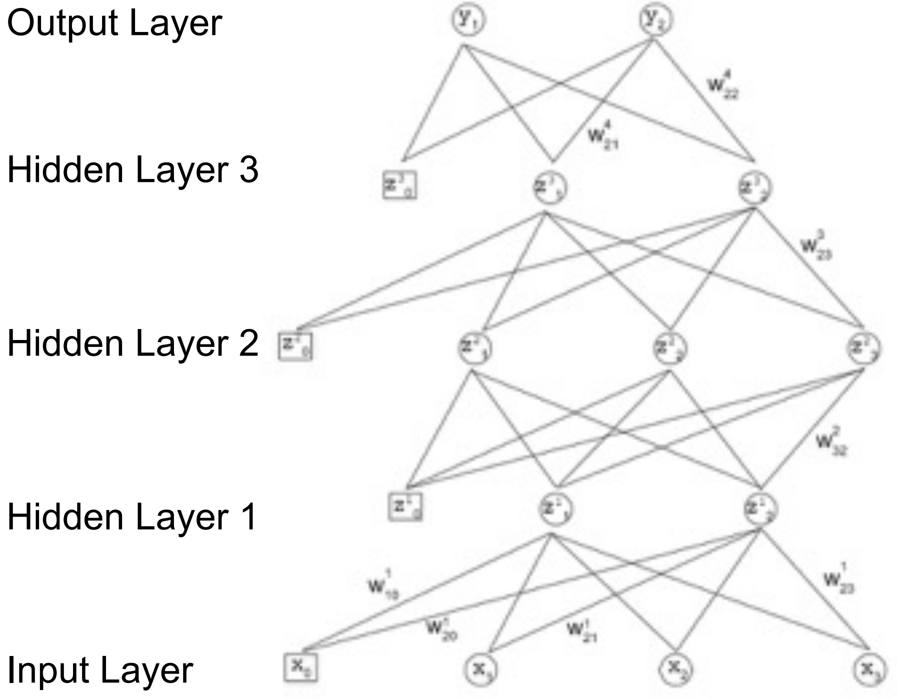


KEY CONCEPTS - COMPUTING THE LOGICAL 'XOR' FUNCTION



Neural Networks can have different network architectures

- Additional hidden layers can compute more complex functions
- More than a single layer of hidden units implies a 'deep' architecture



KEY CONCEPTS - MULTI-CLASS CLASSIFICATION

- Binary classification (y = 0, or y = 1): single output unit
- Multi-class classification (K classes): k output units
- if K = 3, then class 1 = [1, 0 , 0], class 2 = [0, 1, 0], and class 3 = [0, 0, 1]

KEY CONCEPTS - TRAINING A NEURAL NETWORK - THE ERROR BACK-PROPAGATION ALGORITHM

https://class.coursera.org/neuralnets-2012-001/lecture/39

- If an output unit produces an incorrect response we need to determine how, and by how much to adjust the weights of the network
- This is soluble if the network has differentiable activation functions
- At every unit in the network we can compute an error which is a differentiable function of the weights

Key Concepts - Training a Neural Network - The Error Back-propagation Algorithm

- By calculating the derivatives of the error with respect to the weights we can then adjust the weights in order to minimize the error function using gradient descent

- The algorithm for evaluating the derivatives of the error function is known as back-propagation, since we shall propagate errors back through the network

Key Concepts - Training a Neural Network - The Error Back-propagation Algorithm

2 stage process:

1. propagate errors backwards through the network to determine the derivatives

2. using the derivative information the weights of the network are adjusted

In practical terms, there are 4 steps

1. Apply an input vector x^n to the network and forward propagate
2. Evaluate all the output unit deltas (deltas)
3. Back-propagate the delta to obtain the deltas ($delta_j$) for each hidden unit
4. Evaluate the required derivatives and adjust the values of the weights

A delta is the error for any given node in any given layer

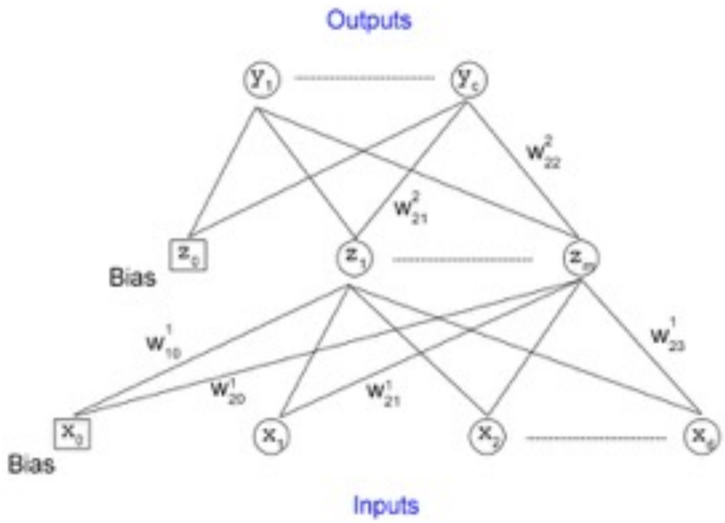
Key Concepts - A Simple Example

Choose this example for simplicity

Use a sum-of-squares error function

Output unit activation function is softmax

Hidden unit activation function is sigmoid



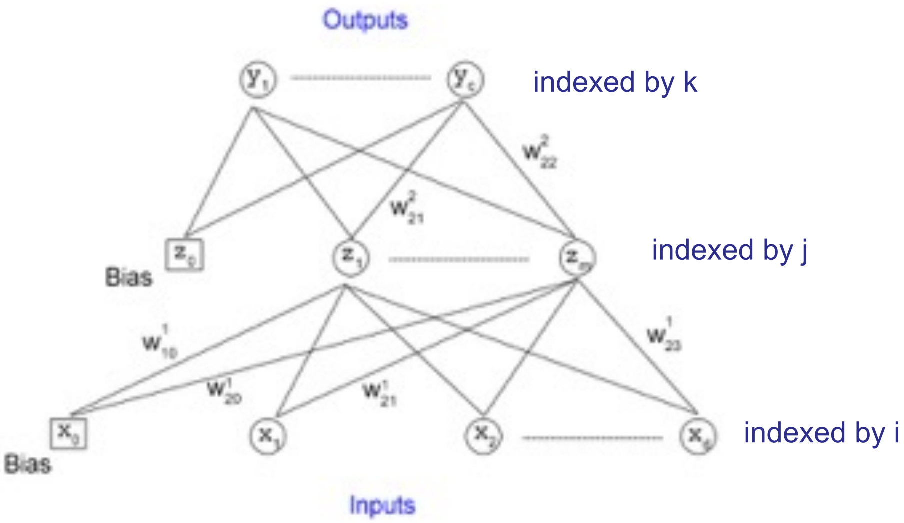
KEY CONCEPTS - A SIMPLE EXAMPLE - STEPS 1 AND 2

The output layer deltas are calculated by:

$$\delta_k = y_k - t_k$$

where y is the output from the network and t is the know target from the training set

we get y by propagating the feature vector through the network in a forwards fashion



KEY CONCEPTS - A SIMPLE EXAMPLE - STEP 3

The hidden layer deltas are calculated by:

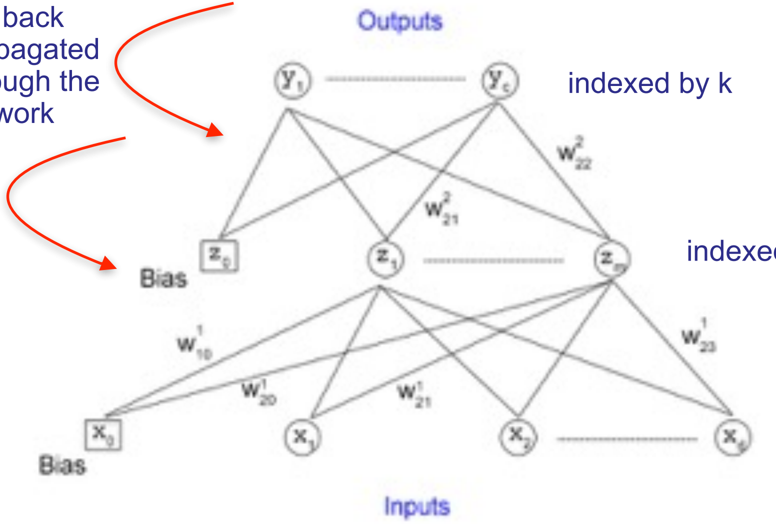
$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k$$

where g prime is the derivative of the activation function, in this case, the sigmoid

$$g(a) = \frac{1}{(1 + exp(-a))}$$

$$g'(a) = g(a)(1 - g(a))$$

the deltas are back propagated through the network



indexed by k

indexed by j

indexed by i

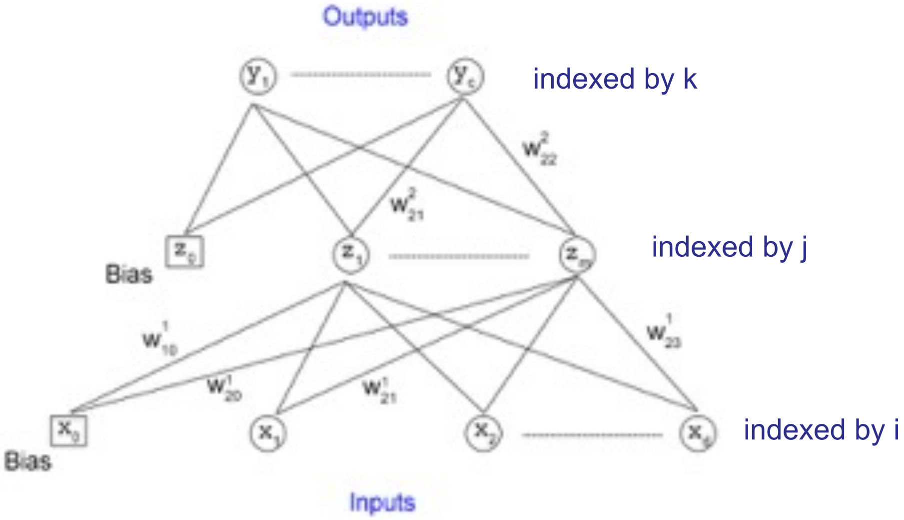
KEY CONCEPTS - A SIMPLE EXAMPLE - STEP 4

The partial derivatives are then given by:

$$\frac{\partial E^n}{\partial w_{kj}} = \delta_k z_j$$

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j x_i$$

which allows for the update to the weights



KEY CONCEPTS - A SIMPLE EXAMPLE - STEP 4

- Decide on the architecture
- Number input units = number of features
- Number output units = number of classes
- Generally start with a single hidden layer, but if you have multiple hidden layers then the number of hidden units might be the same
- Hidden units are generally more in number than the number of features

KEY CONCEPTS - A SIMPLE EXAMPLE - STEP 4

- MLP is just one type of neural network
- Restricted Boltzman Machine (RBM) (there is an implementation is sklearn) is a unsupervised neural network that can perform feature discovery
- Convolutional Neural Network, usually used in image analysis, uses layers that convolve filters over the image, and layers that pool the outputs of the convolutional layers
- Recurrent neural networks, where the outputs of the network feed back in as inputs to the network

KEY CONCEPTS - A SIMPLE EXAMPLE - STEP 4

- In general neural networks can be difficult to train
- To get the results that are now published require networks where the number of free parameters in the networks (weights) is in the millions
- Training takes a considerable time (sometime weeks)
- The networks will only train in this time with hardware specific implementation - Cuda, or OpenCL
- i.e. matrix multiplication is done on GPUs, which were designed for gaming

KEY CONCEPTS - A SIMPLE EXAMPLE - STEP 4

- There are a number of dominant implementations for deep learning architectures:

1. Theano: http://deeplearning.net/software/theano/
    - Python library

2. Caffe: http://caffe.berkeleyvision.org/
    - C++

3. Torch: http://torch.ch/
    - Lua - fast low level language
    - iTorch runs in an iPython notebook