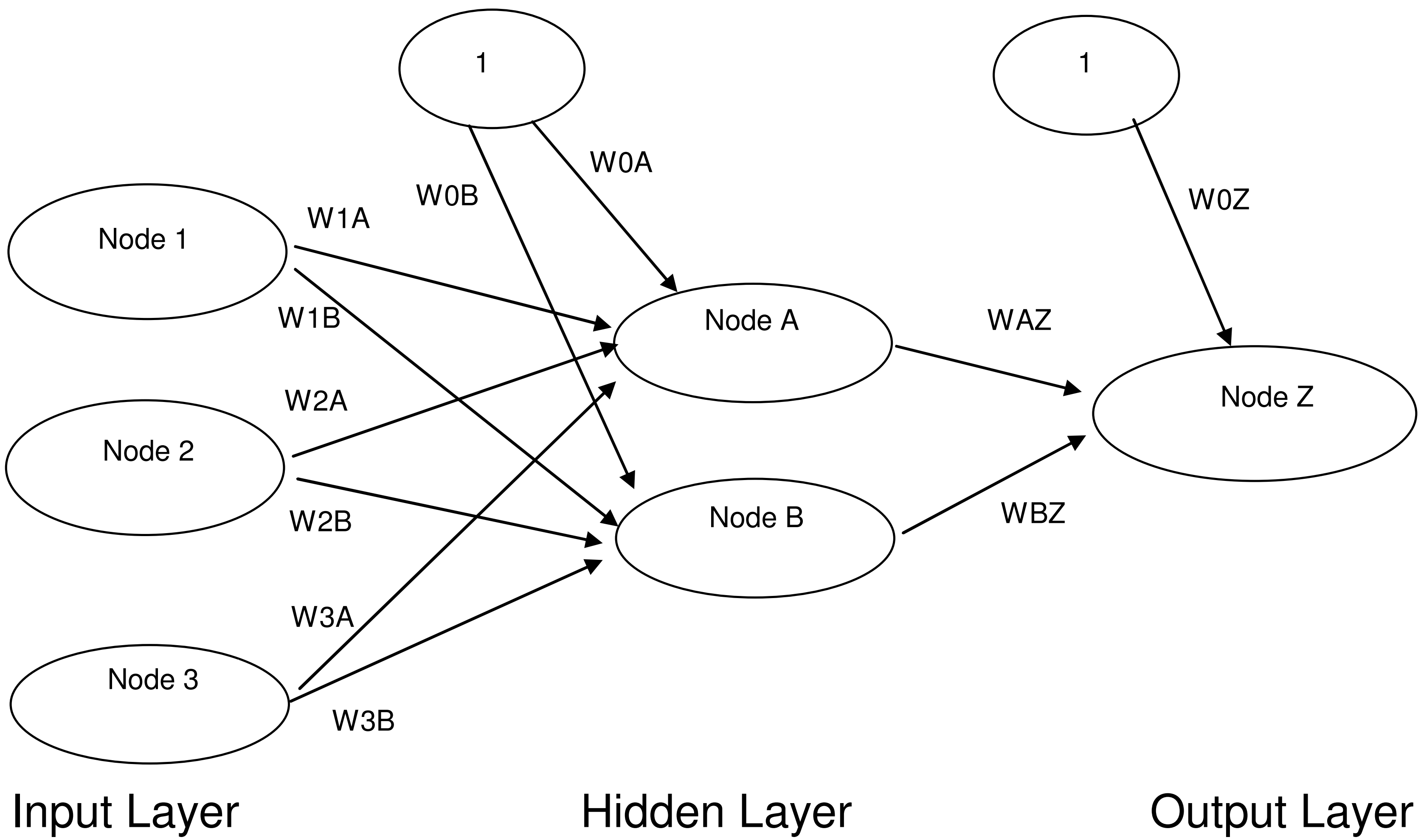
Lecture 11a
Neural Networks

Breitzman 8/8/2018

Brief Intro

- A neural network is a set of connected input/output units in which each connection has a weight associated with it
- There are many different neural network algorithms
- The one we are studying is called backpropagation.
- Invented in 1982, by John Hopfield, backpropagation is by far the most popular method
- Neural Networks are “Black Box” algorithms
- Most people that use them, have no idea how or why they work. Most data mining books only talk about how to prepare the data for using them
- There are whole books and whole courses devoted to Neural Nets, we will barely scratch the surface, but we will learn some of the math behind them

Simple Neural Network



Previous Slide

- Input Layer has as many nodes as attributes
- Attributes must be numeric; should be between 0 and 1
- Can have multiple hidden layers, but usually one is sufficient
- Hidden layer may have any number of nodes. Often as many nodes as attributes, sometimes more. Trial and Error will find best dimension (A rule of thumb is 2/3 the difference between input dimensions and output dimensions-but test between 1/3 and 4/3 the input size to be safe)
- Too many hidden nodes leads to overfitting
- Output layer will have as many nodes as needed for classification. Often only need one (with output of 0 or 1, or <.5 and >=.5)

Big Picture

- For the simple neural net from 2 slides ago inputs are 3 tuples
- Training data will look like a 4 tuple with 3 inputs and output class
- The 3 inputs go to Node 1, 2, 3 and the neural net operates on them to get the Z node using the various weights Wi
- Z node compared to the training output to compute an error
- An algorithm called backpropagation is used to propagate the error back through the nodes and adjust the weights
- After much training, the weights get corrected to model whatever we are modeling
- Often we have no idea how many nodes to have and what the weights should be. We just randomly assign weights between 0 and 1 and the algorithm will eventually find the correct weights
- Training of neural nets can take a long time compared with other algorithms

Example

- Suppose we have the following training tuple (.4, .2, .7, .8)
- So Node 1, 2, and 3 are .4, .2, and .7. Call them x1, x2, x3
- We'll let the initial weights be

W0A	=	0.5	W0B	=	0.7			
W1A	=	0.6	W1B	=	0.9	W0Z	=	0.5
W2A	=	0.8	W2B	=	0.8	WAZ	=	0.9
W3A	=	0.6	W3B	=	0.4	WBZ	=	0.9

$$net_A = \sum_i W_{iA} x_i = W_{0A}(1) + W_{1A}(.4) + W_{2A}(.2) + W_{3A}(.7)$$

$$= .5 + (.6)(.4) + (.8)(.2) + (.6)(.7) = 1.32$$

$$net_B = \sum_i W_{iB} x_i = W_{0B}(1) + W_{1B}(.4) + W_{2B}(.2) + W_{3B}(.7)$$

$$= .7 + (.9)(.4) + (.8)(.2) + (.4)(.7) = 1.5$$

Example (II)

- netA and netB are used as inputs to an activation function
- In biological neurons, signals are sent between neurons and when a combination of inputs to a particular neuron cross a threshold, the neuron "fires."
- This is in general, non-linear behavior. We model this with a non-linear "activation" function.
- A common activation function is

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

- This is known as the sigmoid function and has some useful properties
- It is sometimes called the "squashing function" because it takes any real value and returns a number between 0 and 1.
- Between $-1 < x < 1$ it behaves nearly linearly
- Between [1,5] and [-5,-1] it acts curvilenear
- Outside of 5 and -5 it behaves almost like a constant function
- So depending on weights and inputs we can model almost anything

Example (III)

- We have netA=1.32 and netB=1.5
- It follows that f(netA)= 1/(1+exp(-1.32))=.7892
- f(netB)=1/(1+exp(-1.5))=.8176
- $net_Z = \sum_i W_{iZ} x_{iZ} = W_{0Z}(1) + W_{AZ}(.7892) + W_{BZ}(.8176)$

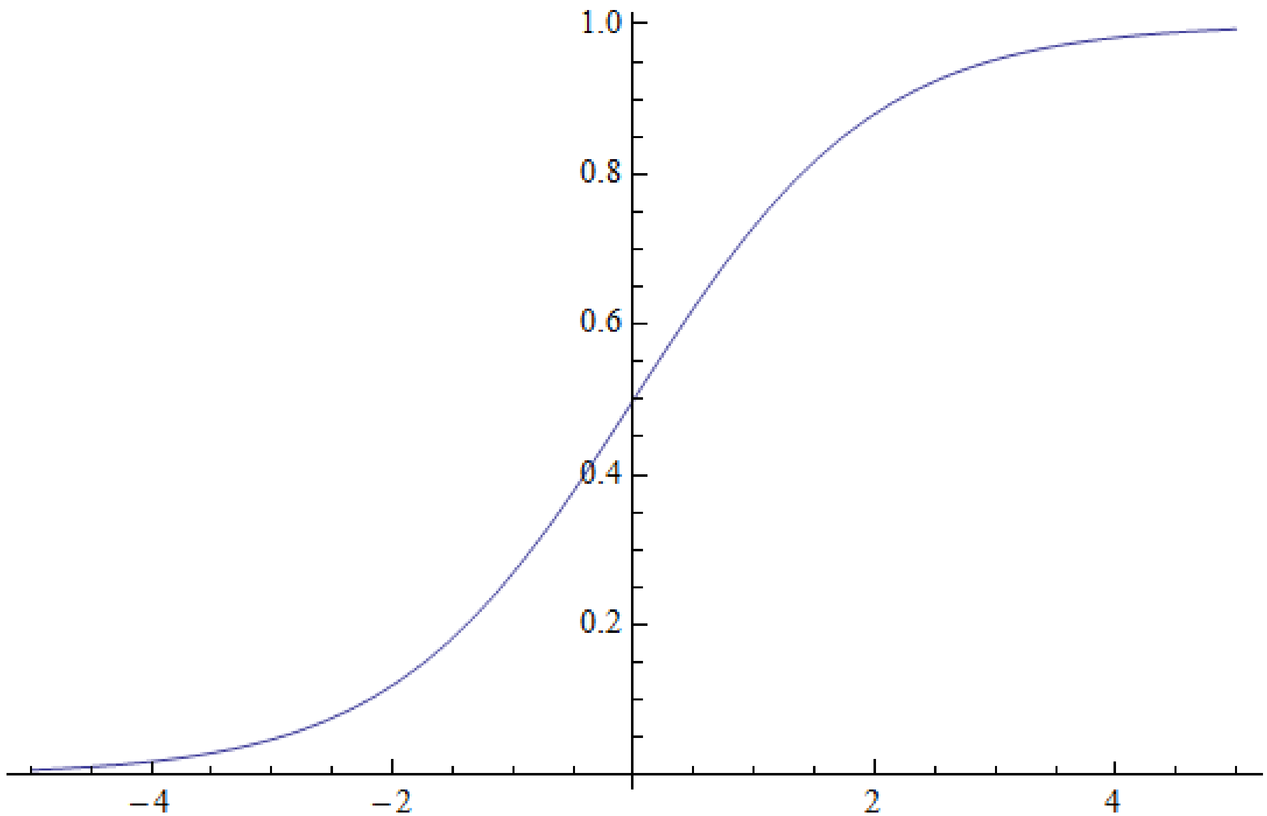
$$= .5 + (.9)(.7892) + (.9)(8176) = 1.9461$$

- F(netZ)=1/(1+exp(-1.9461))=.875
- Expected answer was .8 from the training data so error is (.8 - .875)= -.075

Graph of the Activation Function

```
In[1]:= f[x_] := 1 / (1 + Exp[-1 * x])
        Plot[f[x], {x, -5, 5}]
```

Out[2]=



frame ▾ | axes ▾ | image size ▾ | plot style... | more...

How does the Neural Network Learn?

- We need lot's of training data
- Ultimately we wish to minimize the following

$$SSE = \sum_{records} \left(\sum_{output\ nodes} (actual - output)^2 \right)$$

- This error is analogous to the residuals in regression models
- It's essentially the sum of the squares of all the errors over the entire training set
- The method we use is called the gradient descent method
- Recall from Calc III a gradient is just a vector of partial derivatives
- The derivatives represent slopes of current trajectories, so we can use them to see if we need to increase or decrease a given weight

Back Propagation

- We won’t actually be taking derivatives or computing gradients
- We’ll gloss over the details, but the algorithm is taken from Mitchell, Machine Learning, McGraw Hill, 1997 and Larose, Discovering Knowledge in Data, Wiley, 2005.

$$w_{ij,new} = w_{ij,current} + \Delta w_{ij}$$

$$where \quad \Delta w_{ij} = \eta \delta_j x_{ij} \quad and\ \eta = learning\ \ rate$$

$$and\ \ \delta_j = \left\langle \begin{array}{ll} output_j(1-output_j)(actual_j - output_j) & for\ output\ layer\ nodes \\ output_j(1-output_j) \quad \sum\limits_{downstream} W_{jk}\delta_j & for\ hidden\ layer\ nodes \end{array} \right.$$

Back Propagation (II)

- All of this looks scarier than it actually is
- The idea is to "propagate" our error backwards through the nodes. Creating new weights as we go
- Use a learning rate of 0.1. (We'll talk more about learning rates later)

$$\delta_Z = output_Z(1 - output_Z)(actual_Z - output_Z) = .875(1 - .875)(.8 - .875)$$
$$= -.0082$$
$$\Delta W_{0Z} = \eta\delta_Z(1) = (.1)(-.0082) = -.00082$$
$$W_{0Z,new} = W_{0Z,current} + \Delta W_{0Z} = .5 - .00082 = .49918$$

- Next move upstream to Node A and compute it's weights

- The only downstream node is node Z and we already computed its error (-.0082) and it's associated weight is W_{AZ}=.9

$$\delta_A = output_A(1 - output_A) \sum_{downstream} W_{jk}\delta_j$$

Back Propagation (III)

- I have a feeling I’ve lost everyone, so let’s look at an Excel example and see if I can bring you back

When does the algorithm terminate?

1. If we run out of testing data (this is probably not going to be optimal)

2. Or: Maintain a set of best-so-far weights as we go. If new weights stop improving, or deviate a lot from the ‘best’ weights it’s time to stop.

3. Or: When the error reaches a certain threshold, stop

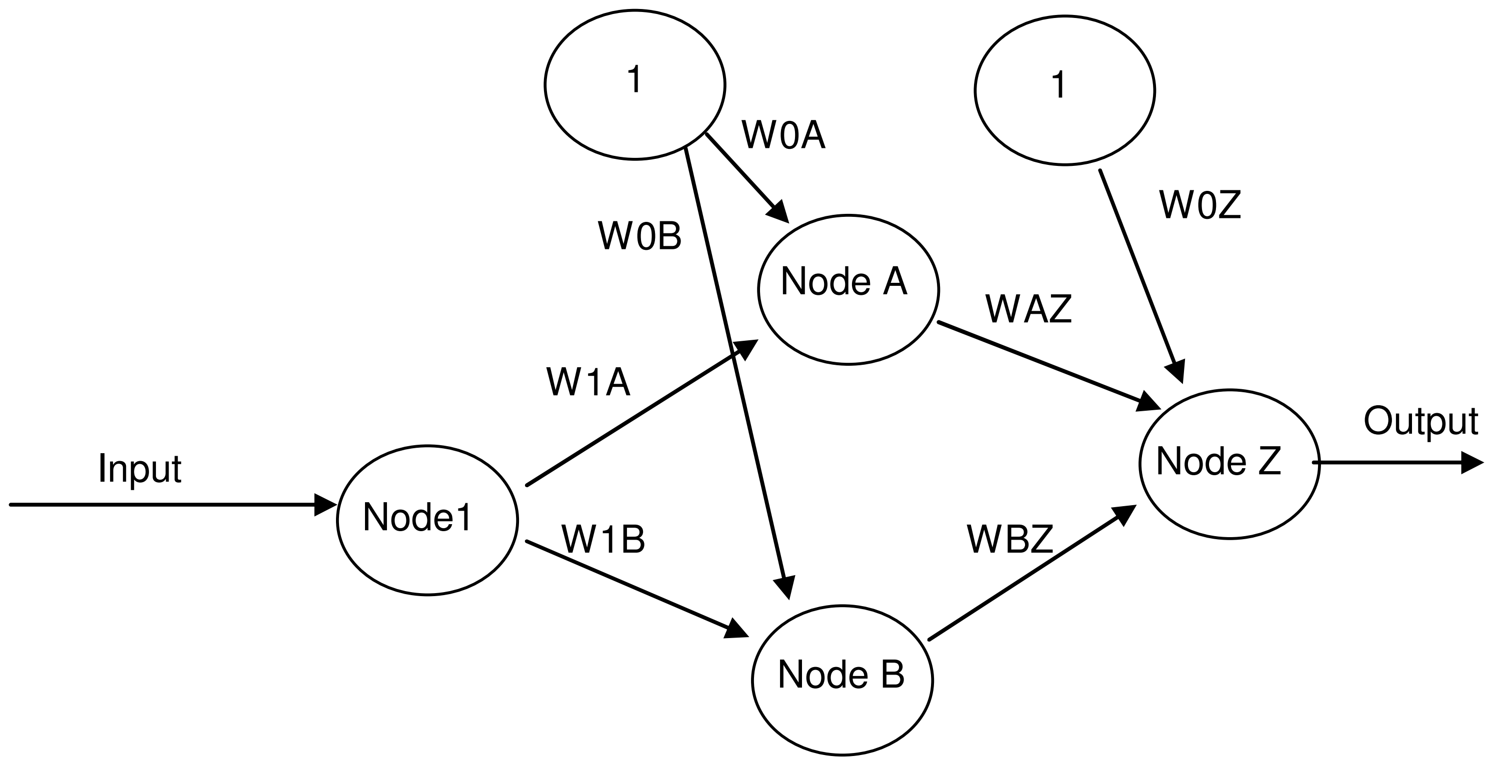
For our next example we will build a Neural Net to Find Square Roots

(I know this is a dumb way to compute square roots)

X	Sqrt(x)	Normalized X	Normalized Sqrt X
5	2.236	0.040	0.137
1	1.000	0.000	0.000
100	10.000	1.000	1.000
48	6.928	0.475	0.659
45	6.708	0.444	0.634
32	5.657	0.313	0.517
11	3.317	0.101	0.257
4	2.000	0.030	0.111
16	4.000	0.152	0.333

- We’ll use min-max normalization to get inputs and outputs between 0 and 1
- We’ll use 2 hidden nodes, although 10 would probably work better

Square Root Neural Net with 2 Hidden Nodes



- Set weights arbitrarily between 0 and 1. (Don't make them the same or we won't get any advantage from the 2 hidden nodes)
- Let W1a=0.4, W1b=0.6, W0a=0.3, W0b=0.4, Waz=0.8, Wbz=0.5, W0z=0.32

Square Root NN

- Go To Excel
- Final Set of Weights

W1a	1.1059
W1b	4.1961
W0a	-0.0597
W0b	-1.5547
Waz	0.4632
Wbz	4.7952
W0z	-2.4443

- Wildly different, from initial weights. Probably not optimal

Results

Test Data	Normalized In	Normalized Sqrt	Guess	Real Square Root	Guess Square Root	Error
47.188	0.467	0.652	0.671	6.869	7.041	2.5%
8.105	0.072	0.205	0.241	2.847	3.172	11.4%
92.173	0.921	0.956	0.905	9.601	9.144	-4.8%
51.505	0.510	0.686	0.716	7.177	7.445	3.7%
50.785	0.503	0.681	0.709	7.126	7.381	3.6%
70.372	0.701	0.821	0.846	8.389	8.613	2.7%
89.880	0.898	0.942	0.901	9.481	9.111	-3.9%
83.303	0.831	0.903	0.888	9.127	8.992	-1.5%
46.195	0.457	0.644	0.660	6.797	6.941	2.1%
95.885	0.958	0.977	0.910	9.792	9.191	-6.1%

- Not awful, but not great
- Could do better by using a variable learning rate, more hidden nodes, or by pairing input nodes (low and high)

Learning Rate

- We used a fixed learning rate 0.1
- That may be too small (it will take forever for algorithm to converge)
- That may be too large (we will keep bouncing back and forth and overshooting an optimal solution)
- Current algorithms use a variable learning rate, that adjusts based on the momentum of the algorithm.
- Without getting too technical, when the gradient is steep, we know we can use a bigger rate.
- If the gradient has switched signs, we have overshot the minimum or maximum
- The Interested reader can see Larose for details

Forbes Article on Google’s Jeff Dean

- http://www.forbes.com/sites/roberthof/2013/05/01/meet-the-guy-who-helped-google-beat-apples-siri/
- Good overview of the way Neural Networks are being improved and used in various applications
- Apparently speech recognition has improved greatly just in the last few years through Neural Nets
- Ed Jung (Former Chief Architect at Microsoft/Cofounder of Intellectual Ventures/Inventor of 700 Patents) back in 2004 (long before Siri) identified speech recognition as the biggest threat/opportunity for Microsoft’s Operating System business

Next

- Repeat Square Root Model in R
- Do Census Example in R
- Sensitivity Analysis