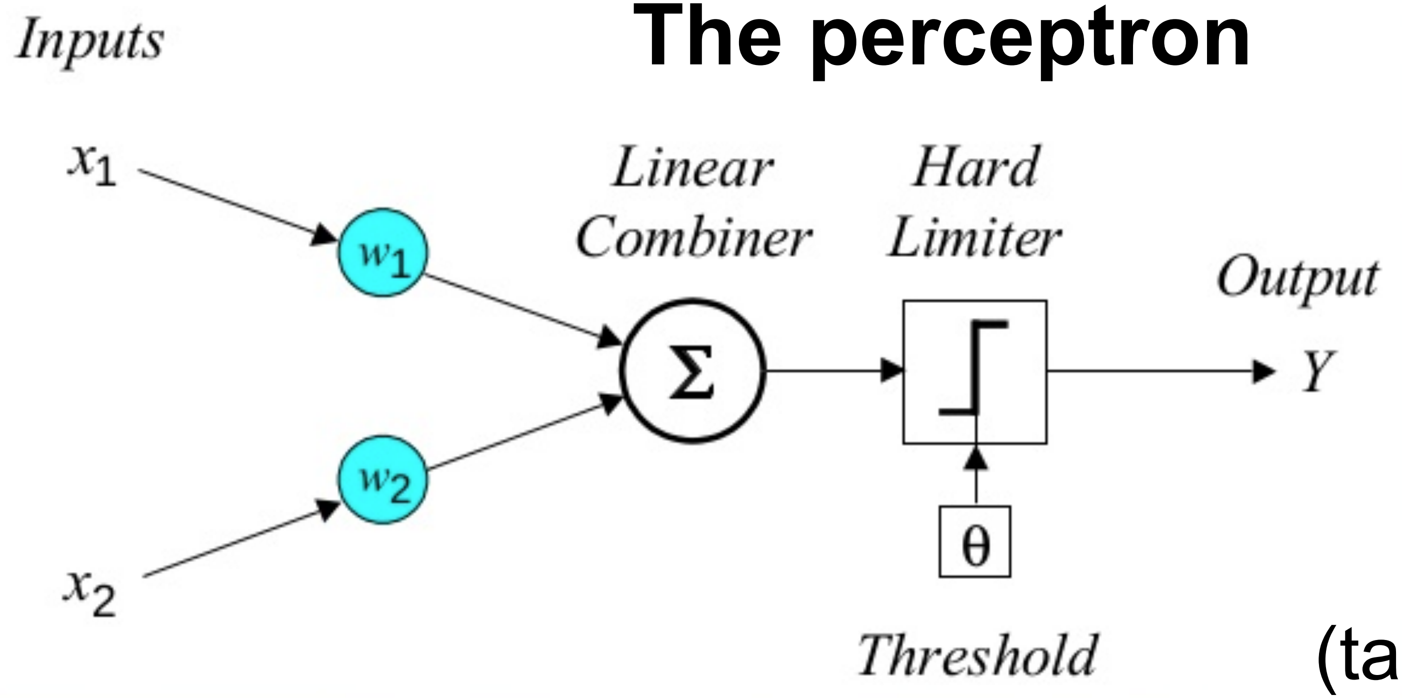
AINT351: Machine Learning

Lecture 12

The perceptron

The perceptron



Output y
(target output t)

- Bias can be augmented into weight vector
- Also add corresponding unity value in augmented data vector

$$\bar{x} = [\ x^1 \quad x^2 \quad 1\ ] \qquad \tilde{w} = [\ w^1 \quad w^2 \quad \theta\ ]$$

$$y = \sum_{i=1}^{n} w^i x^i = \bar{w} \bullet \bar{x}$$

$$if(y > 0) \Rightarrow z = 1$$

$$if(y < 0) \Rightarrow z = 0$$

Perceptron learning rule

- Weight update given by

$$\Delta w_i = \eta\left(t - z\right).x_i$$

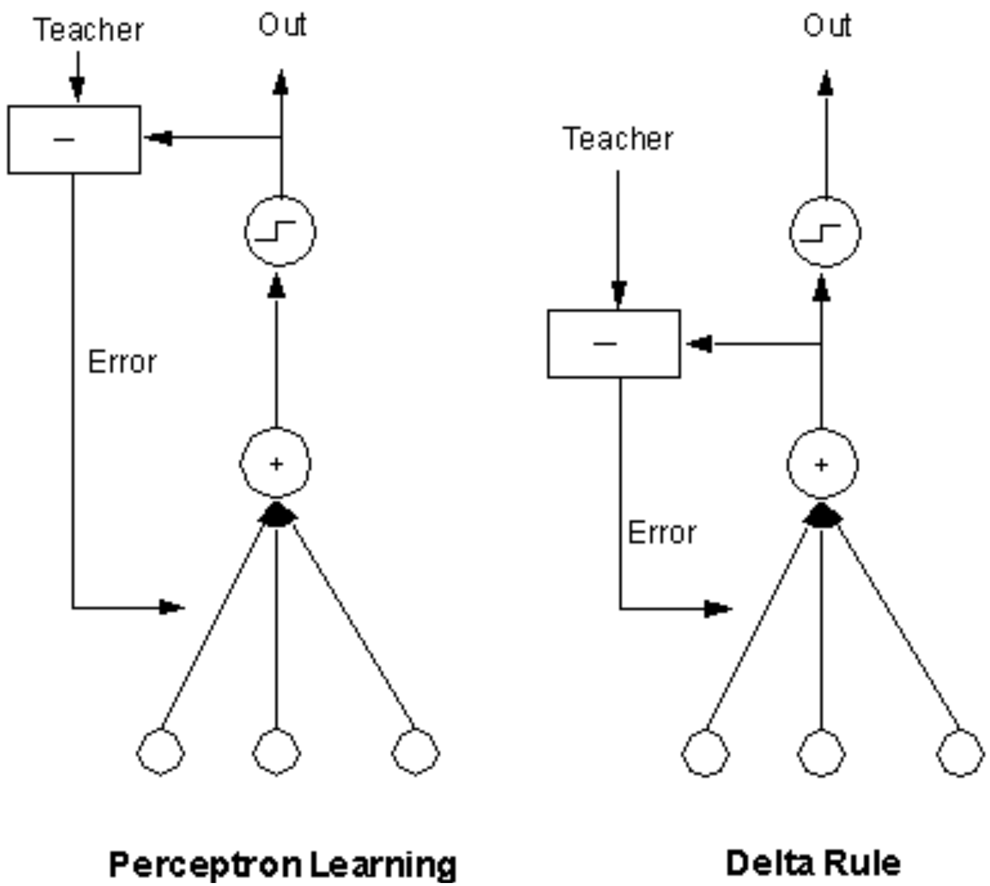
$$\Rightarrow w_i\left(t+1\right) = w_i\left(t\right) + \eta\left(t - z\right).x_i$$

Where: w_i is the weight from input *i* to perceptron node
η is the learning rate
t_i is the target for the current instance (0 or 1)
z is the current output
x_i is i^{th} input

- Guaranteed to converge if the problem is separable
- May be unstable if the problem is not separable

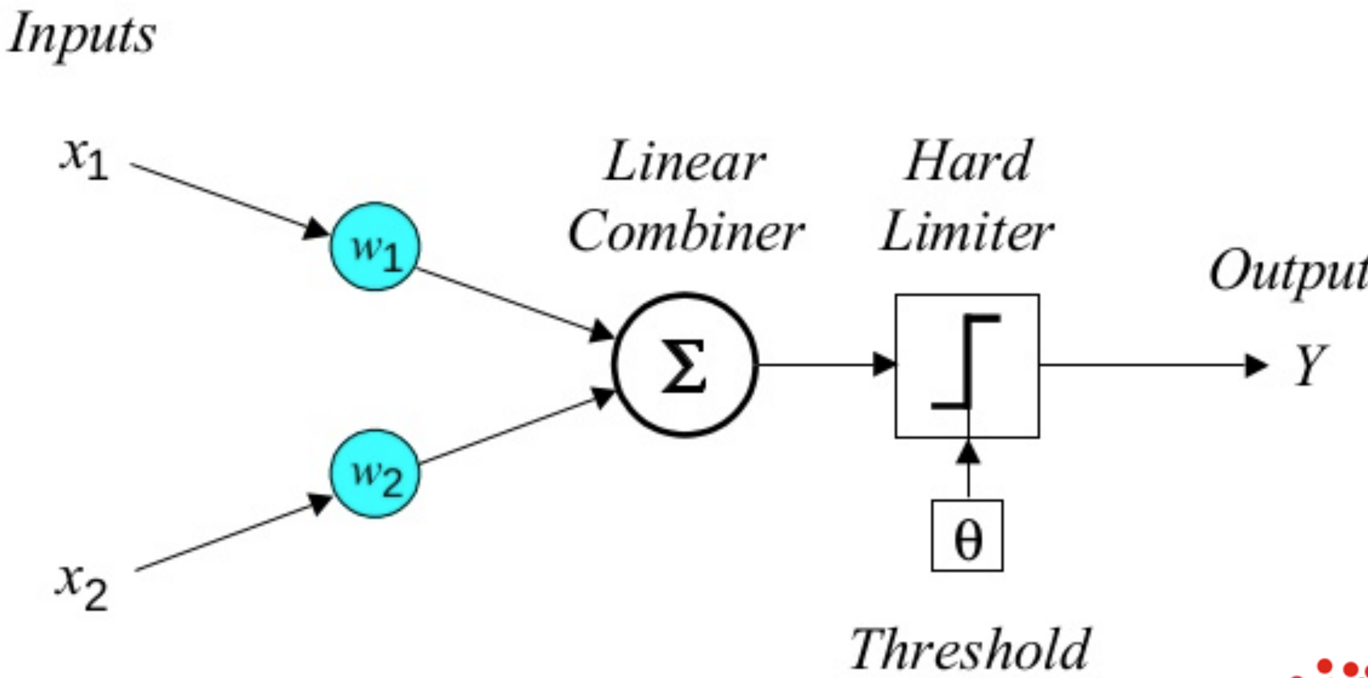
Perceptron rule versus delta rule

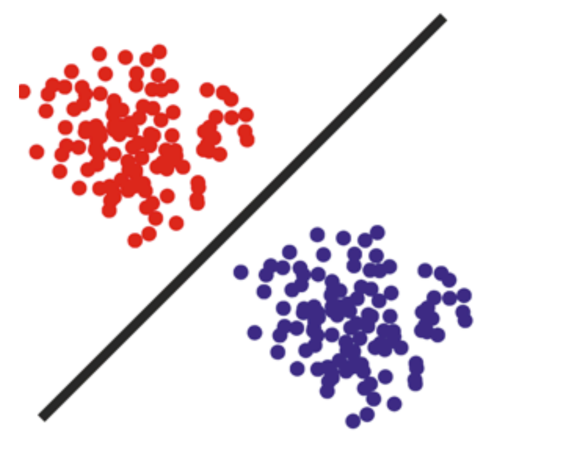
- Perceptron provides learning feedback after a threshold non-linearity



- Instead can provide feedback before threshold
- Replace step function with a continuous differentiable function
- Can be linear – if so, problem similar to that of linear regression
- Can still use threshold for final classification
- This has the advantage we can use gradient descent to find weights

Single layer network

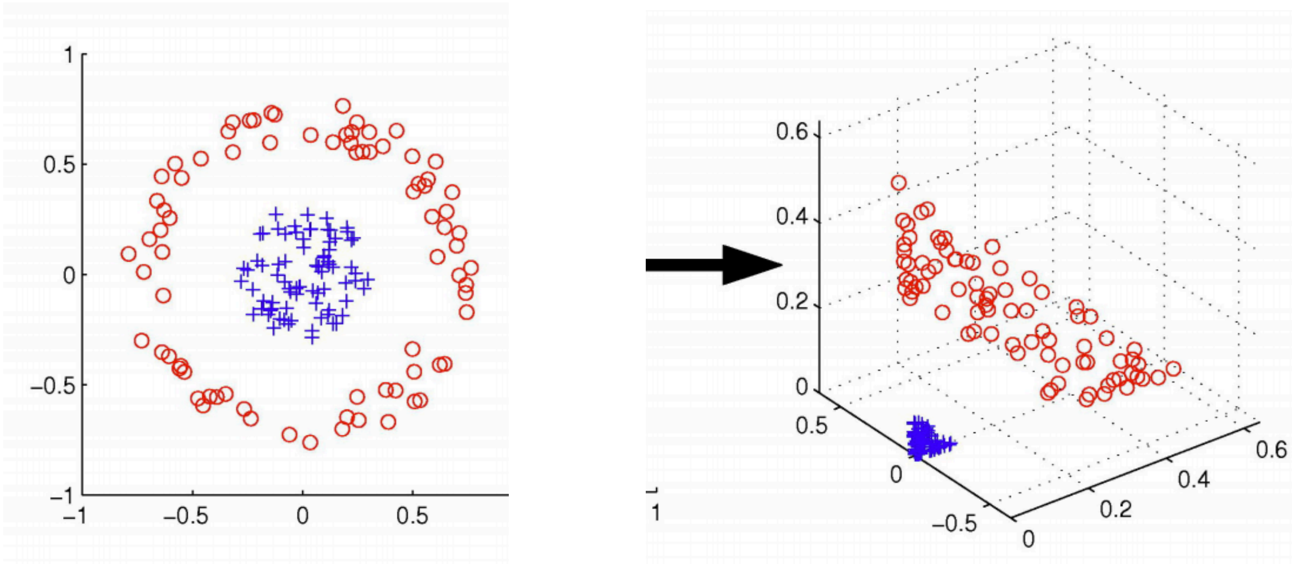




Single layer network implements
linear decision boundary

Kernel trick

- Transforming some datasets can make it linear separable
- E.g. below changing from Cartesian to polar coordinate



- More rigorously this can be done using a kernel
- A kernel is basically a mapping function that transforms one given space into another
- Such transformation of data that leads to easier to solve problems is sometimes known as the kernel trick

AINT351: Machine Learning

Lecture 12

Calculus revision

Chain rule for single variable differentiation

- Consider the one variable equation corresponding to a function of a function

$$z = f(g(x))$$

- Writing

$$y = g(x)$$

- Chain rule gives

$$\frac{dz}{dx} = \frac{dz}{dy}.\frac{dy}{dx}$$

- For example if

$$z = e^{x^2}$$

- In this case let

$$y = x^2 \quad \Rightarrow \frac{dy}{dx} = 2x$$

$$z = e^y \quad \Rightarrow \frac{dz}{dy} = e^y$$

$$\Rightarrow \frac{dz}{dx} = e^y.2x \quad \boxed{= 2x.e^{x^2}}$$

Ordinary derivatives

- With a function of a single variable

$$z = f(x)$$

- The rate of change of y wrt x is given by the full derivative

$$\frac{dz}{dx} = f'(x)$$

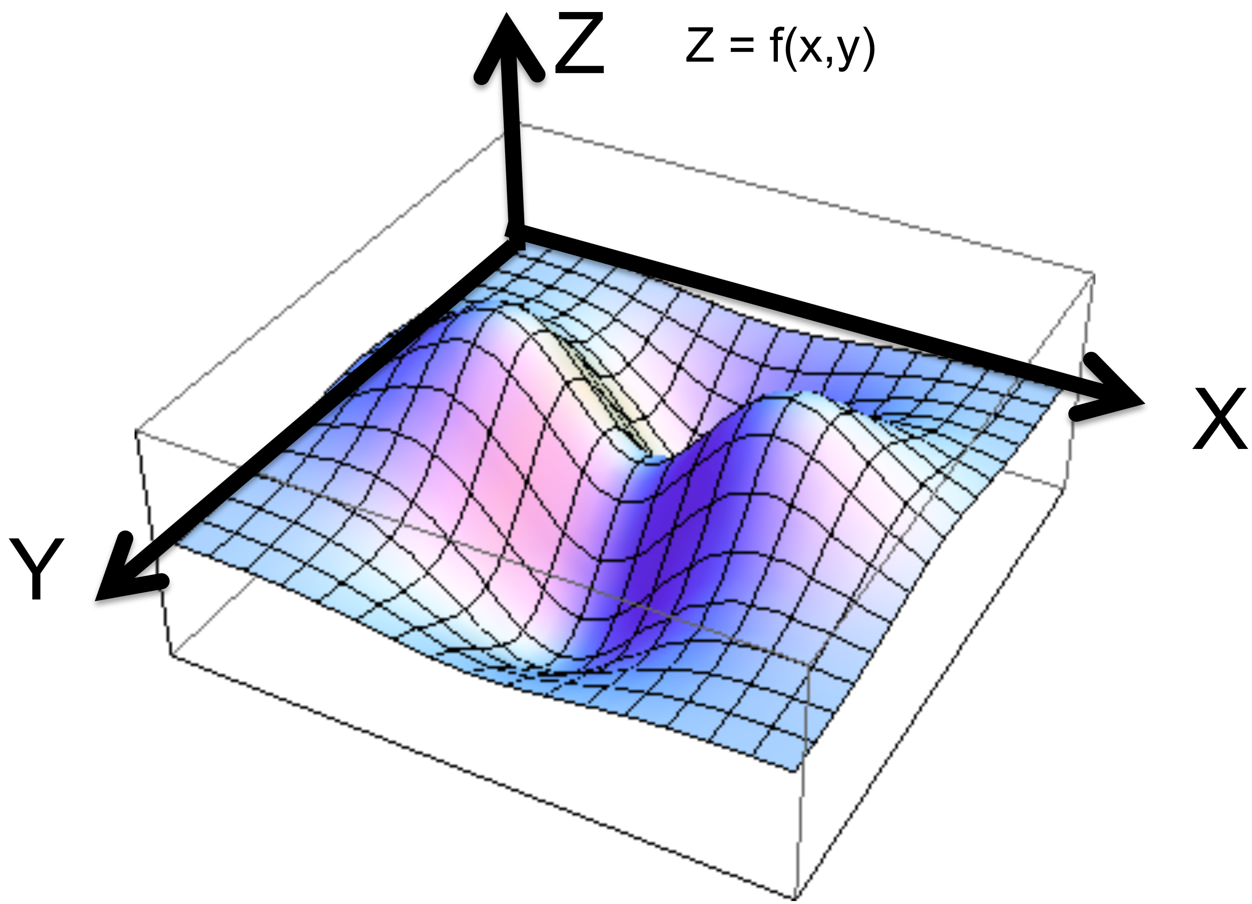
- Straight d symbols used to denote total differential were all variables (x) are allowed to vary

- E.g. given the equation where k is a constant

$$z = (k - x)^2$$

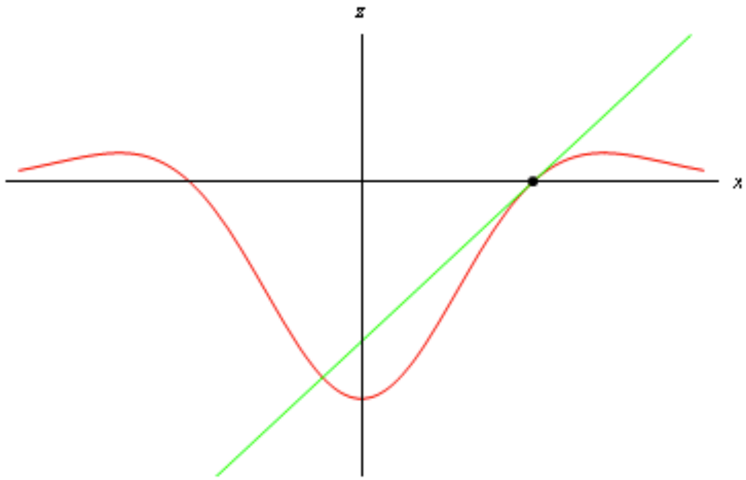
$$\frac{dz}{dx} = -2(k - x)$$

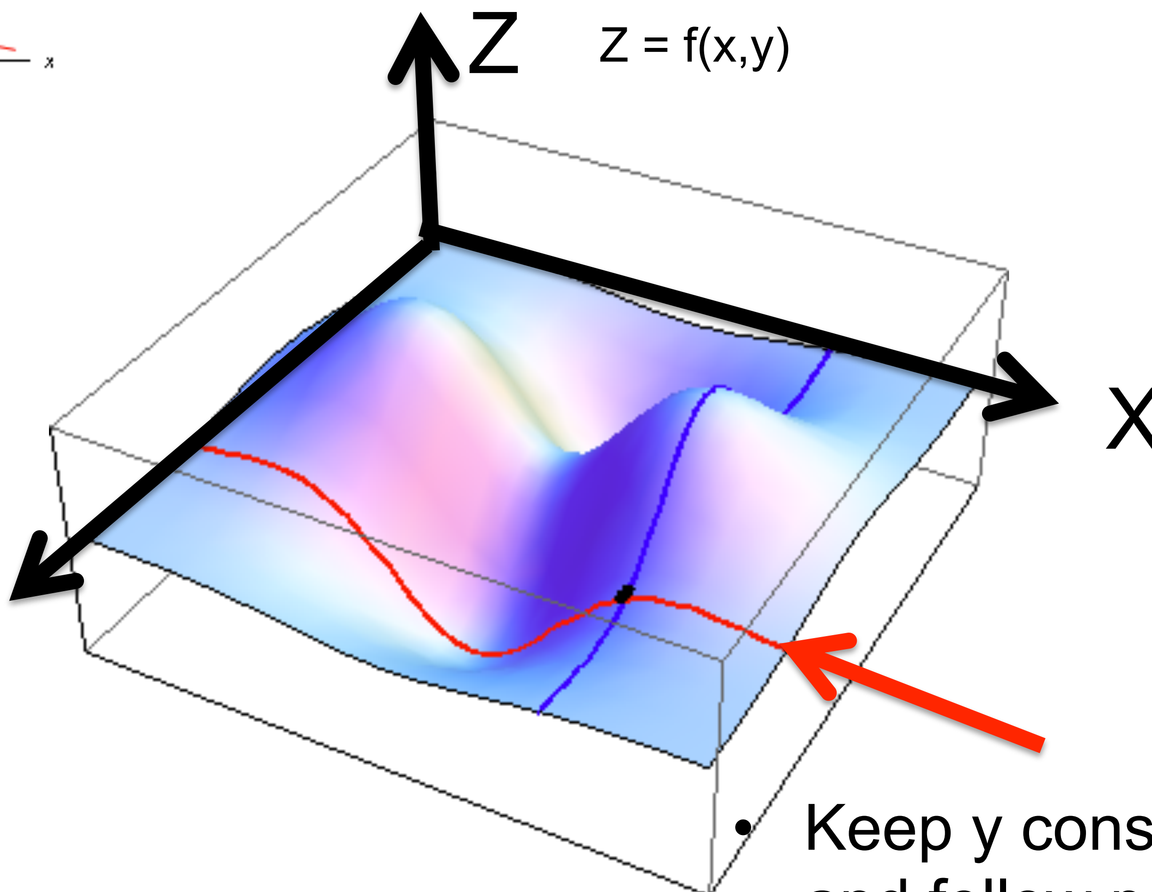
Functions of multiple variables



- Consider function of two valuables
- Z = f(x,y)

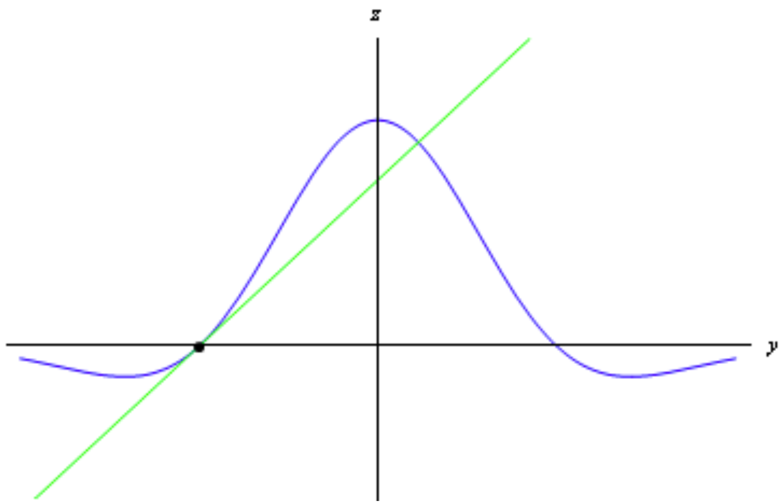
Gradient along two dimensions



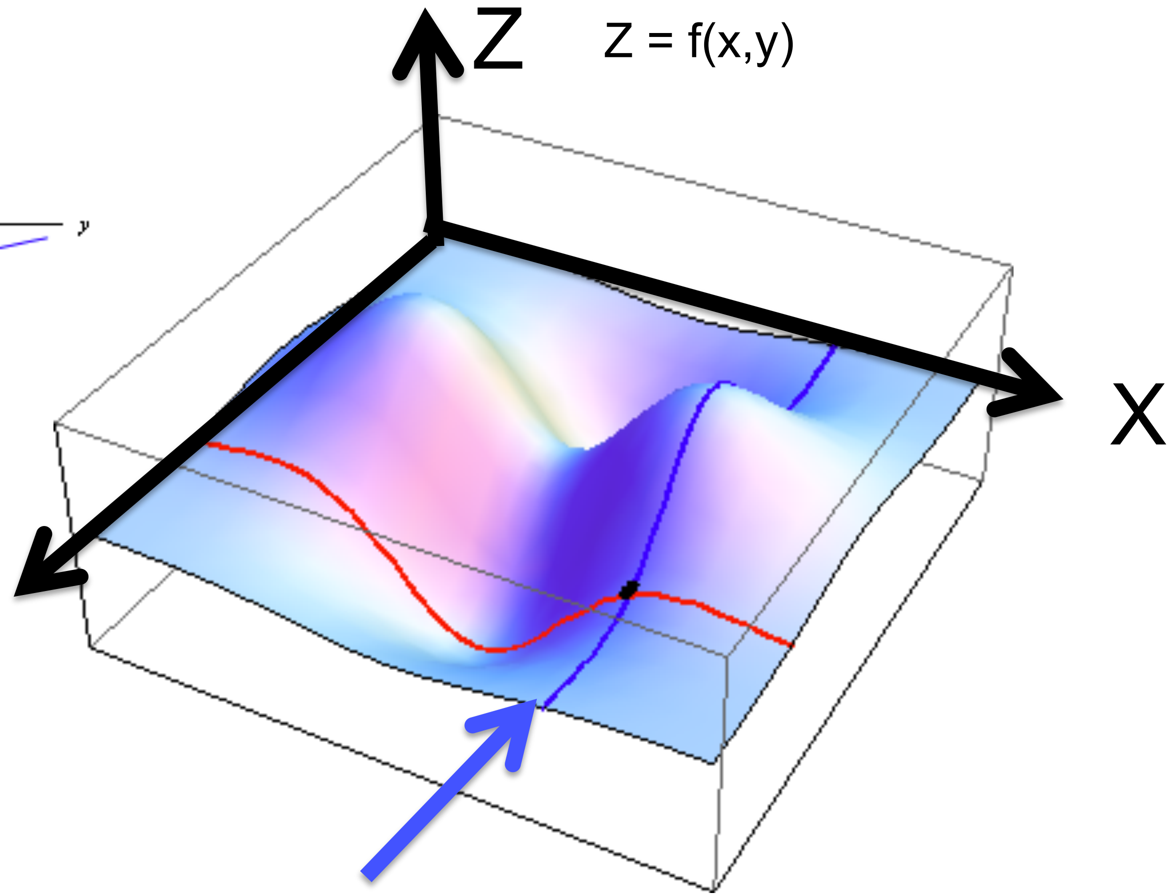


- Can find gradient by differentiating curve wrt x and then evaluating at a given point
- This partial differential is denoted by $$\frac{\partial z}{\partial x}$$
- Keep y constant and follow path along x-axis

Gradient along two dimensions



- Can find gradient by differentiating curve wrt y and then evaluating at a given point



- This is denoted by $\frac{\partial y}{\partial x}$

- Keep x constant and follow path along y-axis

Partial derivatives

- In the case when a function depends on 2 variables

$$z = f(x,y)$$

- The partial derivative of z wrt x is the differential wrt while y is held constant and written as

$$\frac{\partial z}{\partial x}$$

- Curly ∂ symbols used to denote partial differential wrt valuable x

- E.g. given the equation that is a function of x and y

$$z = 3x^2 + 2xy + y^2$$

$$\Rightarrow \frac{\partial z}{\partial x} = 6x + 2y$$

Similarly

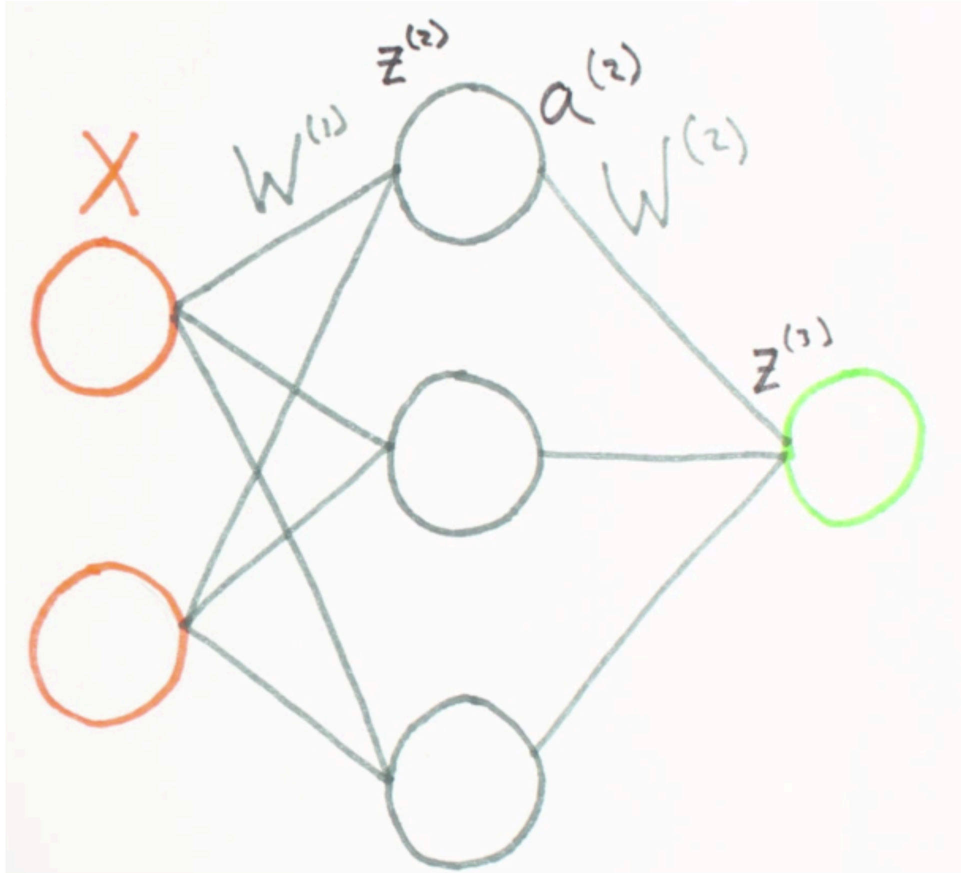
$$\Rightarrow \frac{\partial z}{\partial y} = 2x + 2y$$

AINT351: Machine Learning

Lecture 12

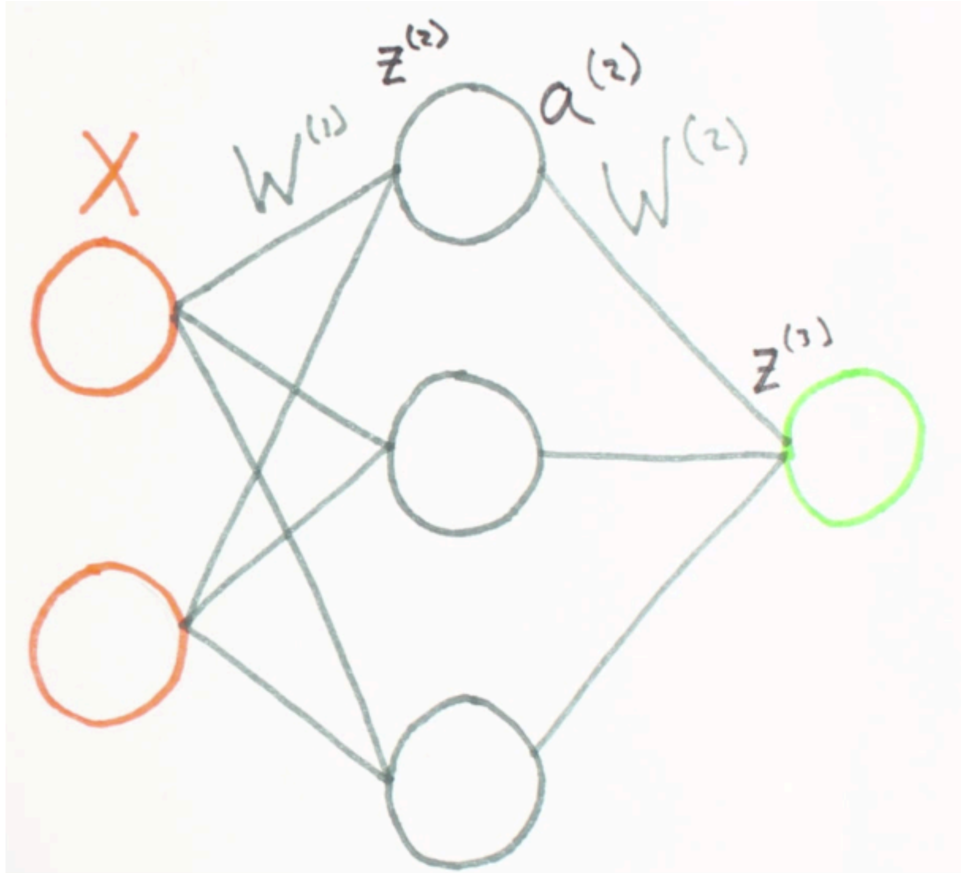
Quick overview of delta rule

Multi layer perceptron



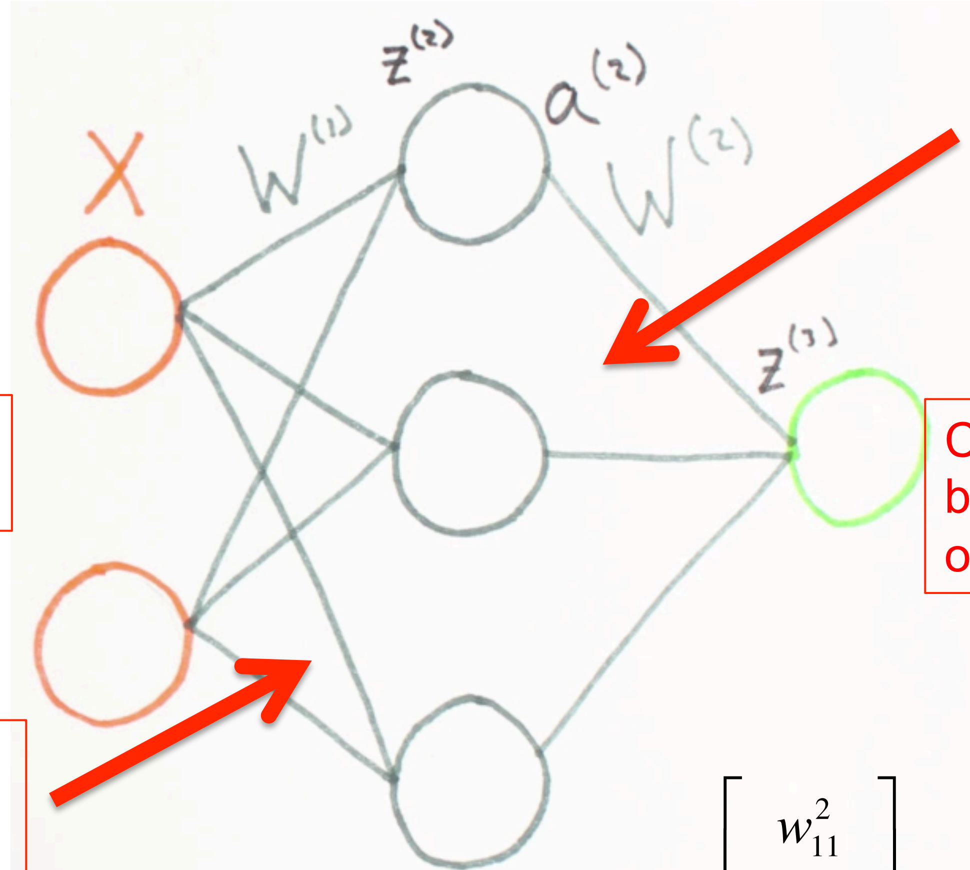
- Consider multi-input/output network
- Feed forward connections
- Fully connected weights
- Multiple inputs and outputs
- Hidden layer of units

Multi layer perceptron



- Training involves minimizing error of calculated output
- Adjust weights by performing gradient descent
- Procedure involves
    - Forward phase
    - Back propagation of errors
- Carry out procedure for each sample or multiple epochs

Training in a nutshell



Adjust weights W2 in layer 2 to improved performance

Apply input and output patterns during training

Calculate error E between target t and output

Adjust weights W1 in layer 1 to improved performance

$$w1 = \begin{bmatrix} w_{11}^{1} & w_{12}^{1} & w_{13}^{1} \\ w_{21}^{1} & w_{22}^{1} & w_{23}^{1} \end{bmatrix}$$

Here we have
2 inputs
3 hidden units

$$w2 = \begin{bmatrix} w_{11}^{2} \\ w_{21}^{2} \\ w_{31}^{2} \end{bmatrix} \Rightarrow$$

Here we have
1 output
3 hidden units

For gradient decent in W1 need to compute $$\frac{\partial E}{\partial W^1}$$

For gradient decent in W2 need to compute $$\frac{\partial E}{\partial W^2}$$

Training in a nutshell

End up with terms for each weight W2

Easy to compute because error only depends on weights in this layer L2

$$\frac{\partial E}{\partial W^2} = \begin{bmatrix} \dfrac{\partial E}{w_{11}^2} \\ \dfrac{\partial E}{w_{31}^2} \\ \dfrac{\partial E}{w_{31}^2} \end{bmatrix}$$

End up with terms for each weight W1

More difficult to compute because error depends on weights in this layerL1 and also 2nd layer L2

$$\frac{\partial E}{\partial W^1} = \begin{bmatrix} \dfrac{\partial E}{\partial w_{11}^1} & \dfrac{\partial E}{\partial w_{11}^1} & \dfrac{\partial E}{\partial w_{13}^1} \\ \dfrac{\partial E}{\partial w_{21}^1} & \dfrac{\partial E}{\partial w_{21}^1} & \dfrac{\partial E}{\partial w_{23}^1} \end{bmatrix}$$

- In a nutshell, training multilayer networks needs repetitive application of the chain rule of partial differentiation
- Once gradient for each weight are found then they can be used to update the weights using iterative gradient decent

Training output layer W2

$$\frac{\partial E}{\partial W^2} = \begin{bmatrix} \frac{\partial E}{w_{11}^2} \\ \frac{\partial E}{w_{31}^2} \\ \frac{\partial E}{w_{31}^2} \end{bmatrix}$$

$$E = \frac{1}{2}\sum_j \left(t_j - o_j\right)^2$$

Compute error gradient wrt weights in 2nd layer W2

$$\Rightarrow \frac{\partial E}{\partial W^2} = \frac{\partial}{\partial W^2}\left[\frac{1}{2}\sum_j \left(t_j - o_j\right)^2\right]$$

Chain rule gives

$$\frac{\partial E}{\partial W^2} = -\sum_j \left(t_j - o_j\right) \cdot \frac{\partial o_j}{\partial W^2}$$

Nonlinear output function

$$o_j = f\left(z^3\right)$$

Split up

Chain rule again gives

$$\frac{\partial E}{\partial W^2} = -\sum_j \left(t_j - o_j\right) \frac{\partial o_j}{\partial z^3} \cdot \frac{\partial z^3}{\partial W^2}$$

Training output layer W2

$$\frac{\partial E}{\partial W^2} = -\sum_j \left(t_j - o_j\right) \frac{\partial o_j}{\partial z^3} \cdot \frac{\partial z^3}{\partial W^2}$$

For nonlinearity

$$o_j = \frac{1}{1+e^{-z^3}}$$

Here represent derivative of nonlinearity as $f_j^{'}\left(z^3\right)$

$$\Rightarrow \frac{\partial E}{\partial W^2} = -\sum_j \left(t_j - o_j\right) \cdot f_j^{'}\left(z^3\right) \cdot \frac{z^3}{\partial W^2}$$

Training output layer W2

But

$$z^3{}_j = \sum_i w^2{}_{ji} a_i \quad \Rightarrow \frac{\partial z^3}{\partial W^2} = a_i$$

Only depends on input activation to layer

$$\Rightarrow \frac{\partial E}{\partial W^2} = -\sum_j \left(t_j - o_j\right) \cdot f_j^{'}\left(z^3\right) \cdot a_i$$

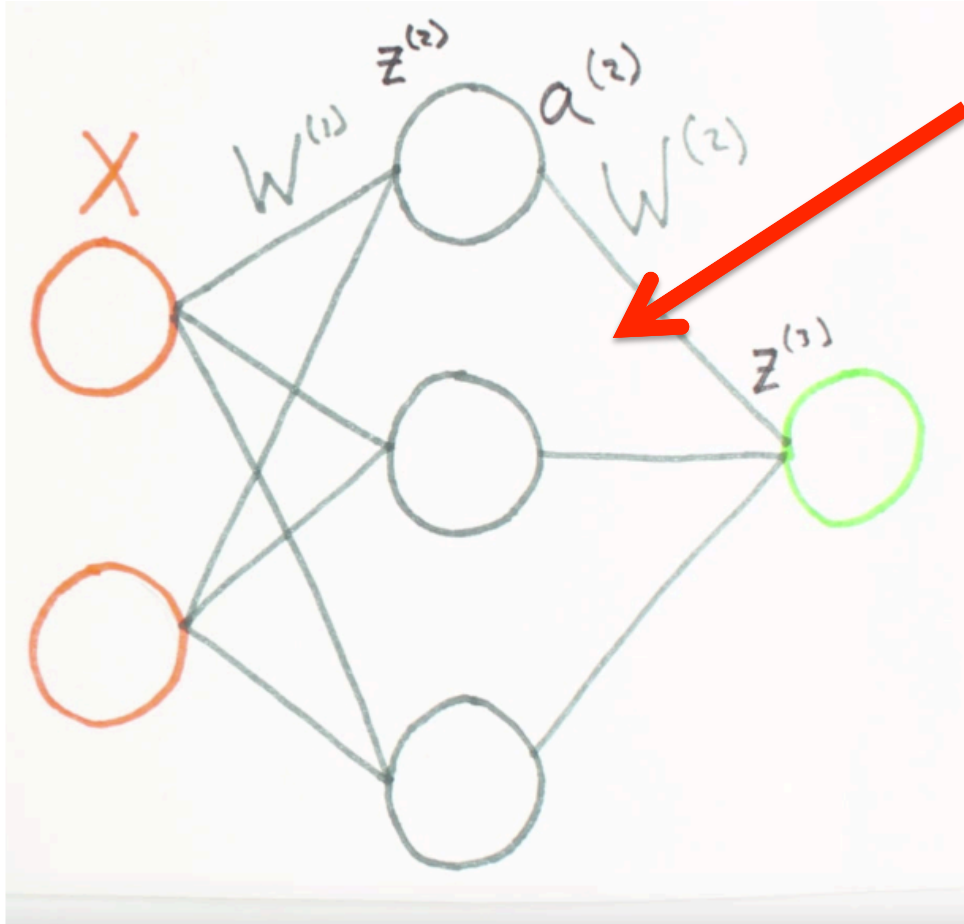
$$\Rightarrow \frac{\partial E}{\partial W^2} = \partial(3) \cdot a_i$$

This gradient expression can be used to update W2

Where

$$\partial(3) = -\sum_j \left(t_j - o_j\right) \cdot f_j^{'}\left(z^3\right)$$

Training output layer W2



Error back propagated to change weight

Depends on activation

Biggest effect on nodes with largest activations

Use vector of gradients to update weights
After the presentation of a single pattern

$$\frac{\partial E}{\partial W^2} = \begin{bmatrix} \dfrac{\partial E}{w_{11}^2} \\ \dfrac{\partial E}{w_{31}^2} \\ \dfrac{\partial E}{w_{31}^2} \end{bmatrix}$$

Training hidden layer W1

The error is the same as before

$$E = \frac{1}{2}\sum_j \left(t_j - o_j\right)^2$$

Now compute error gradient across synapses in different layers!
This is now for weights W1 in layer 1

Now want gradient wrt to first layer weights

$$\Rightarrow \frac{\partial E}{\partial W^1} = \frac{\partial}{\partial W^1}\left[\frac{1}{2}\sum_j \left(t_j - o_j\right)^2\right]$$

Chain rule gives

$$\frac{\partial E}{\partial W^1} = -\sum_j \left(t_j - o_j\right) \cdot \frac{\partial o_j}{\partial W^1}$$

Chain rule again gives

$$\frac{\partial E}{\partial W^1} = -\sum_j \left(t_j - o_j\right) \frac{\partial o_j}{\partial z^3} \cdot \frac{\partial z^3}{\partial W^1}$$

Training hidden layer W1

For nonlinearity as before

$$o_j = f\left(z^3\right)$$

$$\Rightarrow \frac{\partial E}{\partial W^1} = -\sum_j \left(t_j - o_j\right) \cdot f_j^{'}\left(z^3\right) \cdot \frac{\partial z^3}{\partial W^1}$$

$$\Rightarrow \frac{\partial E}{\partial W^1} = \partial(3) \cdot \frac{\partial z^3}{\partial W^1}$$

Again where

$$\partial(3) = -\sum_j \left(t_j - o_j\right) \cdot f_j^{'}\left(z^3\right)$$

Training hidden layer W1

$$\frac{\partial E}{\partial W^1} = \partial(3) \cdot \frac{\partial z^3}{\partial W^1}$$

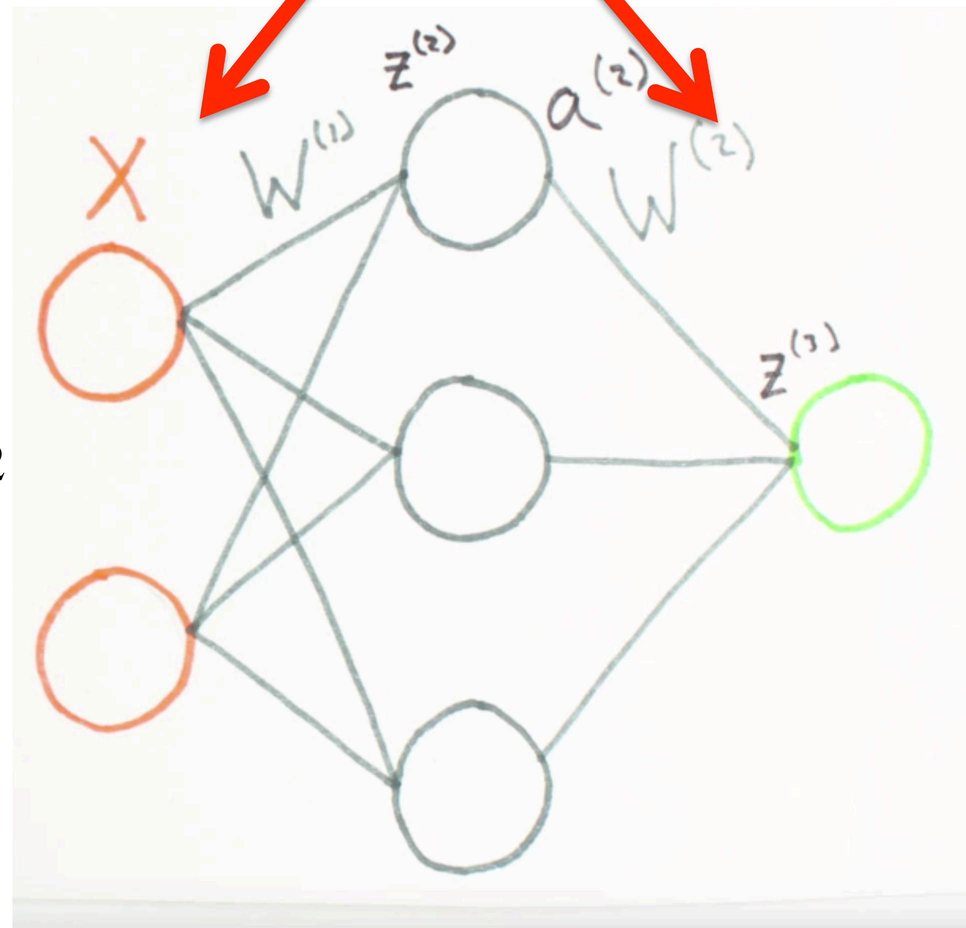
This involves differentiation across synapses

$$\frac{\partial E}{\partial W^1} = \partial(3) \cdot \frac{\partial z^3}{\partial a^2} \cdot \frac{\partial a^2}{\partial W^1}$$

Now are about how z3 changes wrt a2

$$z^3{}_j = \sum_i w^2{}_{ji} a_i^2 \quad \Rightarrow \frac{\partial z^3}{\partial a^2} = W^2$$

$$\frac{\partial E}{\partial W^1} = \partial(3) \cdot \left(W^2\right)^T \cdot \frac{\partial a^2}{\partial W^1}$$



Training hidden layer W1

$$\frac{\partial E}{\partial W^1} = \partial(3) \cdot \left(W^2\right)^T \cdot \frac{\partial a^2}{\partial W^1}$$

Chain rule again

$$\frac{\partial E}{\partial W^1} = \partial(3) \cdot \left(W^2\right)^T \cdot \frac{\partial a^2}{\partial z^2} \cdot \frac{\partial z^2}{\partial W^1}$$

Put in term for nonlinearity

$$\frac{\partial E}{\partial W^1} = \partial(3) \cdot \left(W^2\right)^T \cdot f_j^{'}\left(z^2\right) \cdot \frac{\partial z^2}{\partial W^1}$$

$$\Rightarrow \frac{\partial E}{\partial W^1} = X^T \partial(3) \cdot \left(W^2\right)^T \cdot f_j^{'}\left(z^2\right)$$

Since

$$z_j^2 = \sum_i w_{ji}^1 x_i$$

$$\frac{\partial z^2}{\partial W^1} = X$$

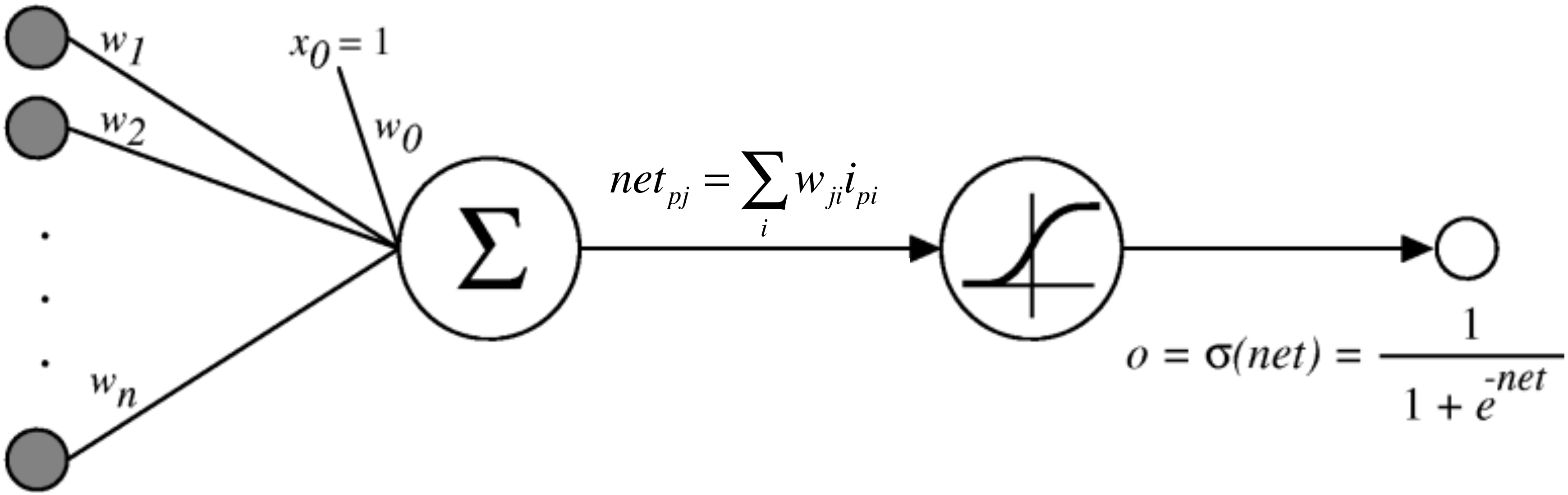
This gradient expression can be used to update W1

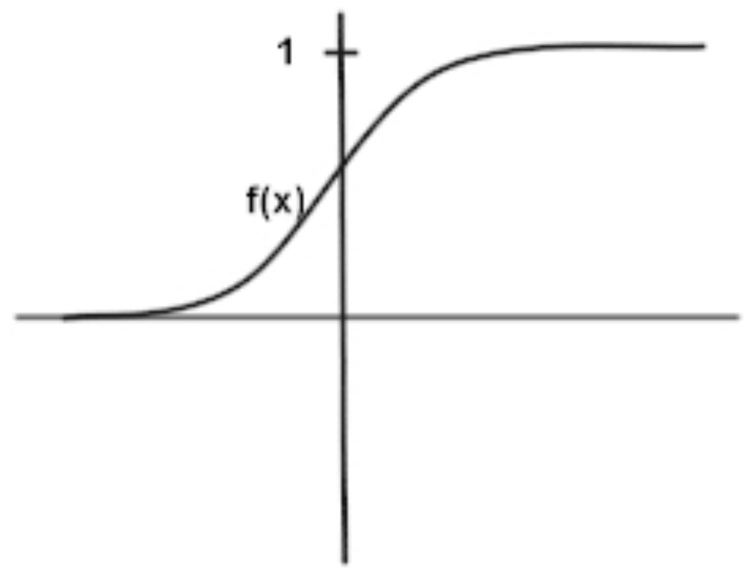
AINT351: Machine Learning

Lecture 12

Sigmoid nonlinearity

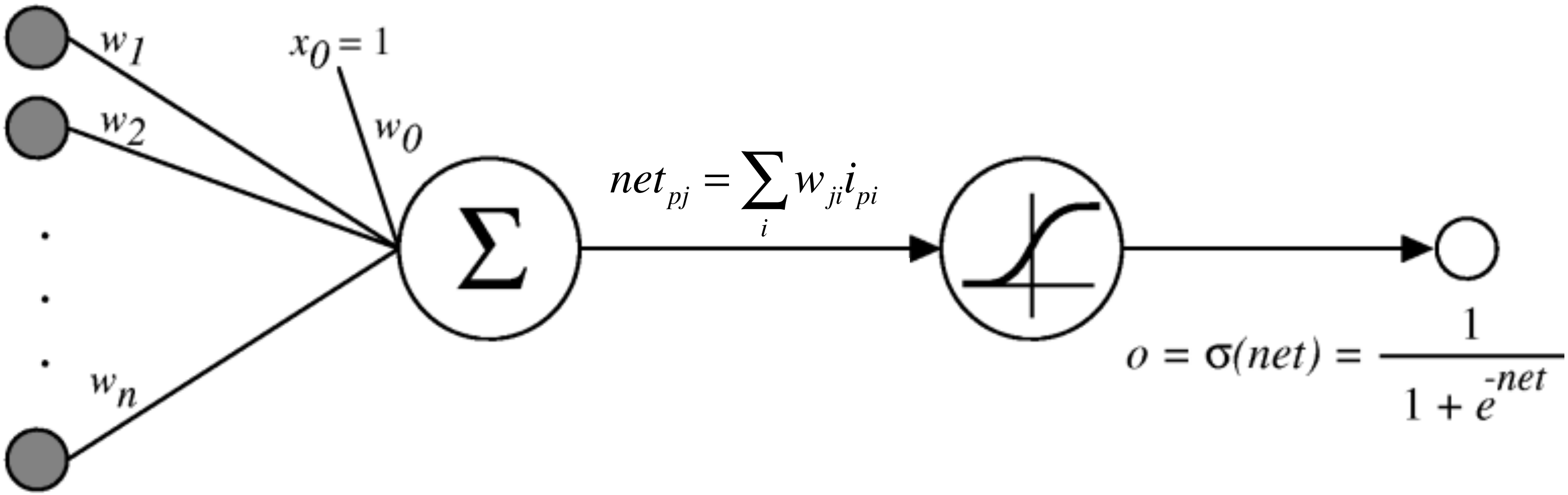
Sigmoid output non-linearity





- Multilayer layer network need nonlinear units
- Otherwise overall network will remain a linear network
- One choice on non-linearity is a sigmoid as semi linear and differentiable

Sigmoid output non-linearity



The net total input to the semi-linear function is given by a weighted sum of its inputs

$$net_{pj} = \sum_i w_{ji} i_{pi}$$

Where i_{pi} is the input i

Output after non-linearity is given by

$$o_{pj} = f_j\left(net_{pj}\right) \qquad = f_j\left(\sum_i w_{ji} i_{pi}\right)$$

A function of a function- so need the chain rule to differentiate

Need to use a semi-linear activation function so that it is differentiable

Differential of sigmoid non-linearity

- For the non-linearity

$$f_j\left(net_{pj}\right) = \frac{1}{1+e^{-net_{pj}}} \qquad \text{or} \qquad o_{pj} = \frac{1}{1+e^{-net_{pj}}}$$

- Can write as

$$f_j\left(net_{pj}\right) = y^{-1} \qquad \text{where} \qquad y = 1+e^{-net_{pj}}$$

- Chain rule gives

$$\Rightarrow f_j^{'}\left(net_{pj}\right) = \frac{\partial}{\partial net_{pj}}\left[f_j\left(net_{pj}\right)\right] = \frac{\partial o_{pj}}{\partial net_{pj}} = \frac{\partial o_{pj}}{\partial y}\frac{\partial y}{\partial net_{pj}}$$

Differential of sigmoid non-linearity

- 1st term gives $$\frac{\partial o_{pj}}{\partial y} = \frac{\partial}{\partial y}\left(y^{-1}\right) = -\frac{1}{y^2} = \frac{-1}{\left(1+e^{-net_{pj}}\right)^2}$$
- 2nd term gives $$\frac{\partial y}{\partial net_{pj}} = \frac{\partial}{\partial net_{pj}}\left(1+e^{-net_{pj}}\right) = -e^{-net_{pj}}$$
- Substituting into equation

$$f_j'\left(net_{pj}\right) = \frac{\partial o_{pj}}{\partial y}\frac{\partial y}{\partial net_{pj}} = \frac{-1}{\left(1+e^{-net_{pj}}\right)^2}\left(-e^{-net_{pj}}\right)$$

$$= \frac{e^{-net_{pj}}}{\left(1+e^{-net_{pj}}\right)^2}$$

Differential of sigmoid non-linearity

- Remember that

$$o_{pj} = \frac{1}{1+e^{-net_{pj}}} = \frac{1+e^{-net_{pj}}}{\left(1+e^{-net_{pj}}\right)^2}$$

$$\left(o_{pj}\right)^2 = \frac{1}{\left(1+e^{-net_{pj}}\right)^2}$$

- Therefore

$$o_{pj} - \left(o_{pj}\right)^2 = \frac{1+e^{-net_{pj}}}{\left(1+e^{-net_{pj}}\right)^2} - \frac{1}{\left(1+e^{-net_{pj}}\right)^2} = \frac{e^{-net_{pj}}}{\left(1+e^{-net_{pj}}\right)^2}$$

$$\Rightarrow f_j'\left(net_{pj}\right) = \frac{e^{-net_{pj}}}{\left(1+e^{-net_{pj}}\right)^2} = o_{pj} - \left(o_{pj}\right)^2 = o_{pj}\left(1-o_{pj}\right)$$

Gradient descent

- From the result that

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} i_{pi}$$

- This implies in order to perform gradient descent we should carry out a weight update from the presentation of pattern p by adding on the quantity

$$\Delta_p w_{ji} = \eta \delta_{pj} i_{pi}$$

- Where η is the learning rate
- This will implement an online pattern by pattern update of the weights

Batch update

- Since the total overall error E across all patterns is given as

$$E = \sum_p E_p$$

- Then we get

$$\frac{\partial E}{\partial w_{ji}} = \sum_p \frac{\partial E_p}{\partial w_{ji}}$$

- Using this relation the error gradient can be calculated as the sum of the individual contributions from all the patterns in the dataset
- This can be used to implement a batch update of the weights

Main results are

Change weights proportional to product of error signal and output of unit sending activation

$$\Delta w_{ji} = \eta \delta_{pi} o_{pi}$$

If at output unit error signal given by

$$\delta_{pi} = \left(t_{pi} - o_{pi}\right) f_j'\left(net_{pj}\right) \qquad = \left(t_{pi} - o_{pi}\right) o_{pj}\left(1 - o_{pj}\right)$$

If at hidden unit error signal operates across synapse and is determined recursively from error signals from connecting units and their weights

$$\delta_{pi} = f_j'\left(net_{pj}\right) \sum_k \delta_{pk} w_{ki} \qquad = o_{pj}\left(1 - o_{pj}\right) \sum_k \delta_{pk} w_{ki}$$

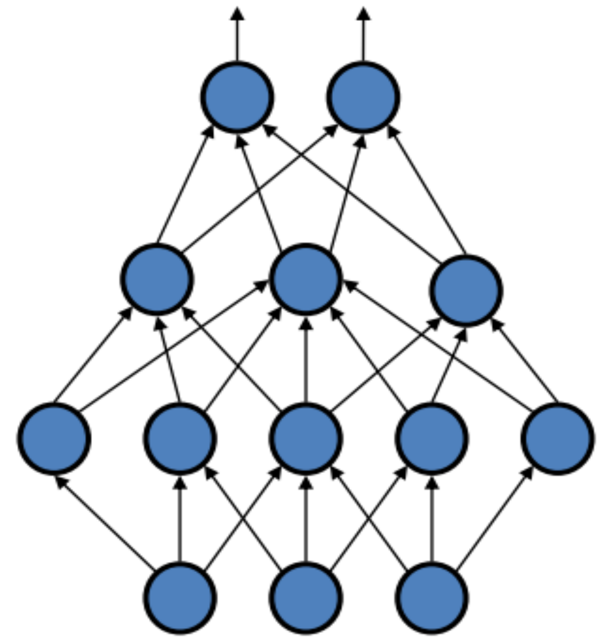
Learning rate and momentum terms

More practical weight update equation

$$\Delta w_{ji}(n+1) = \eta\left(\delta_{pi} o_{pi}\right) + \alpha \Delta w_{ji}(n)$$

η is the learning rate
α is the momentum term

Limitation of backpropagation



- Multiple hidden Layers
- Get stuck in local optima
  - start weights from random positions
- Slow convergence to optimum
  - large training set needed
- Only use labeled data
  - most data is unlabeled
- Error attenuation with deep nets
- Can now training with GPUs and special hardware
- However, can still be time intensive for large networks!

Acknowledgements

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation (No. ICS-8506). CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE.

Neural Networks Demystified [Part 4: Backpropagation]
https://youtu.be/GlcnxUlrtek