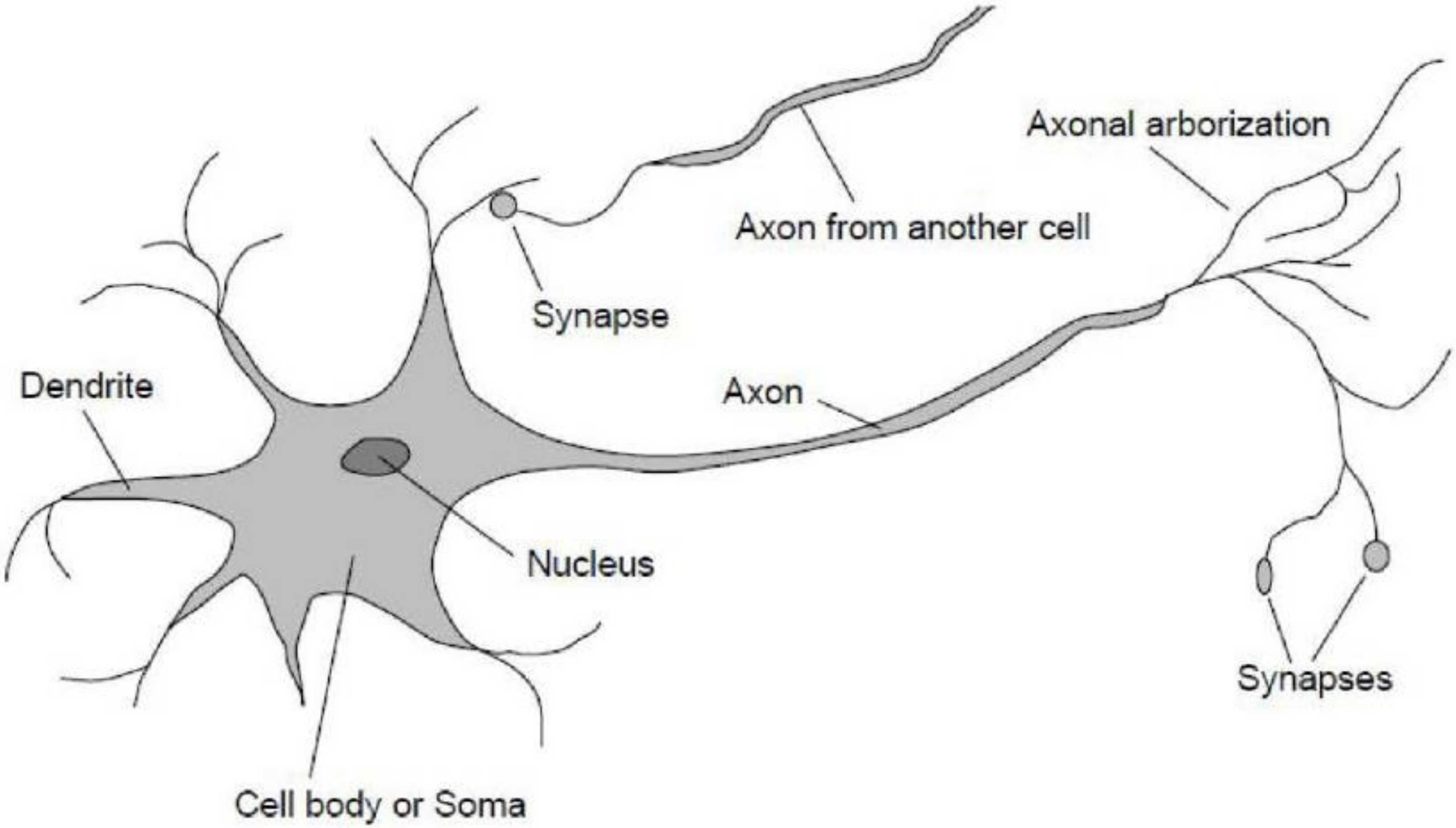
Artificial Neuro Networks (ANN) Classifier

WEEK 3

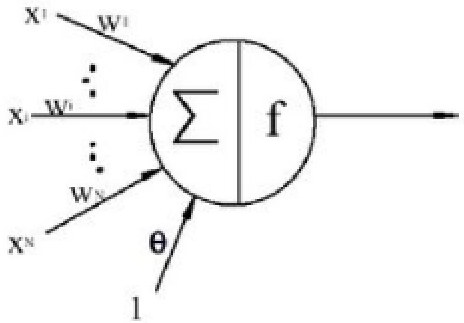
Neuron



Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Perceptron

- A classifier that maps input vector to a single binary output.
- Example Illustration:

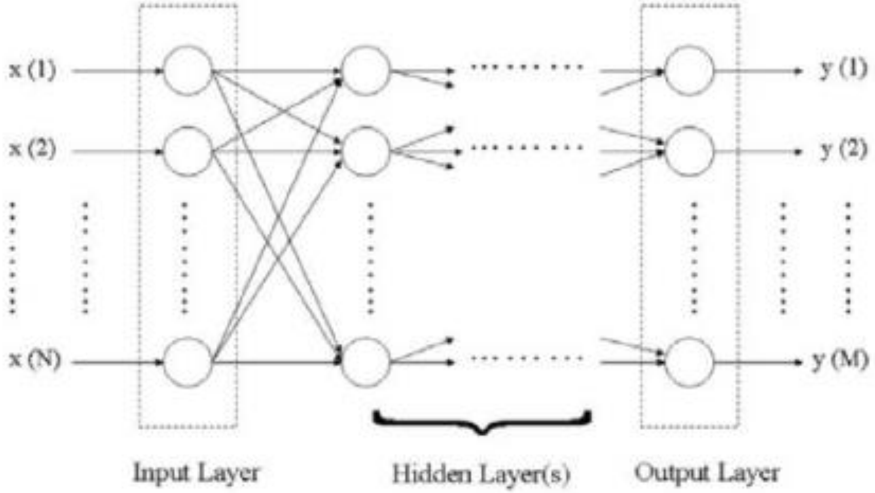


- Commonly used example: A sign function of a linear combination of the input:

$$y = \mathcal{F}\left(\sum_{i=1}^{N} w_i x_i + \theta\right), \quad \mathcal{F}(s) = \begin{cases} 1 & \text{if } s > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Feedforward Neural Network & Multi-layer Perceptron

- Feedforward Neural Network:
  - Layered Structure.
  - Each node receives inputs from nodes directly below, and outputs to the nodes above them.
  - No connection within a layer.

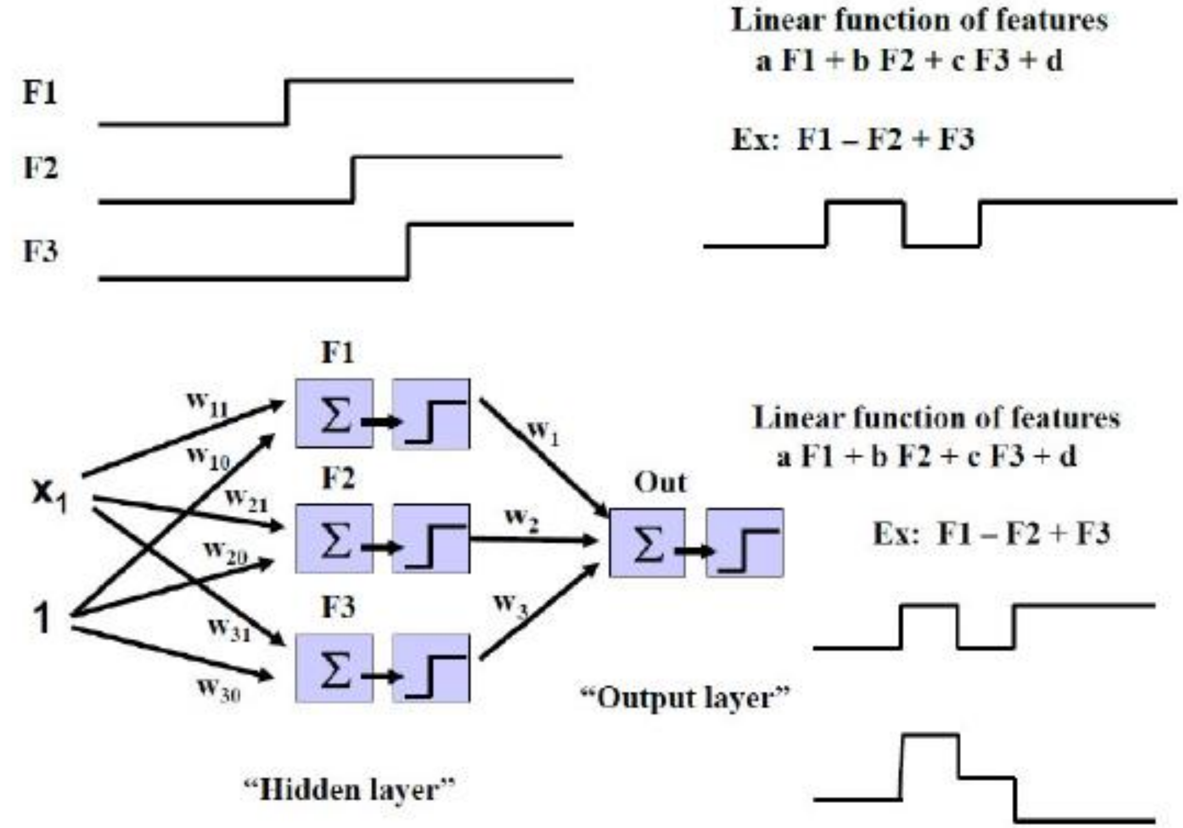


- Multi-layer perceptron is a common practice to implement feedforward neural network.

- In the slides we focus on feedforward neural networks.

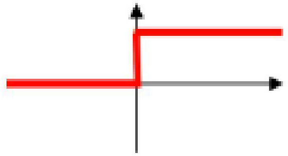
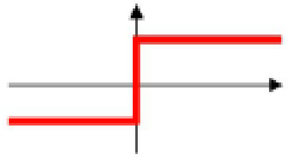
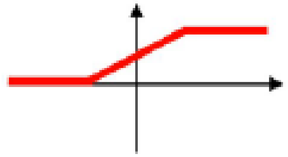
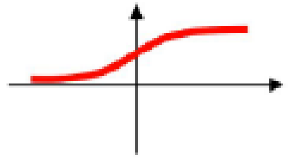
Multi-layer Perceptron and Neural Network

- Intuition: we can approximate almost any 1-D functions with a combination of step functions.



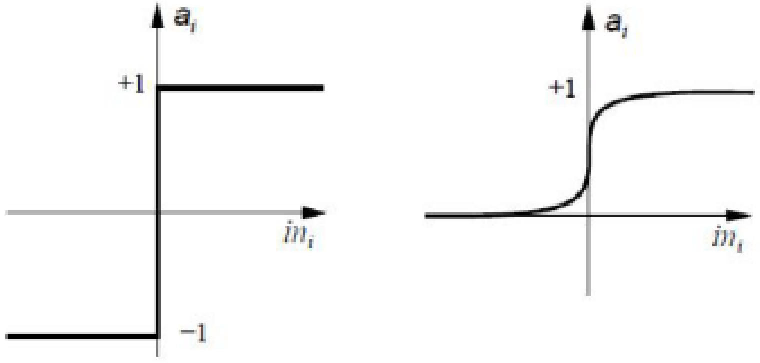
Perceptron

Example Activation Function:

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	

Neural Network

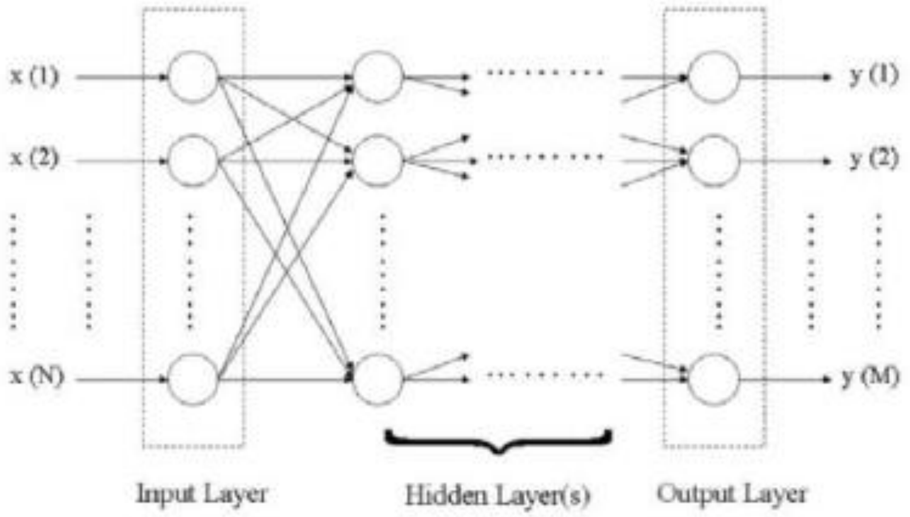
- Choice of Activation Function
  - Sign function, step function, piecewise linear function are hard to deal with.
    - Either non-continuous, or their derivatives are not continuous.
  - Commonly use a **"sigmoid"**
    - A logistic function.
    - $\sigma(z) = \dfrac{1}{1+e^{-z}}$
    - First order derivative: $\dfrac{d\sigma}{dz} = (1-\sigma)\sigma$



(b) Sign function (c) Sigmoid function

Neural Network

- Nice properties by using multiple output nodes in a neural network:
    - **Regression**
      - Multi-dimensional regression
    - **Multi-class Classification:**
      - Use one-hot encoding in class labels
      - Eg. Class 1: [1,0,0,0,...,0]
        Class 2: [0,1,0,0,...,0]
    - **Joint Binary Predictions:**
      - Image tagging



Notations

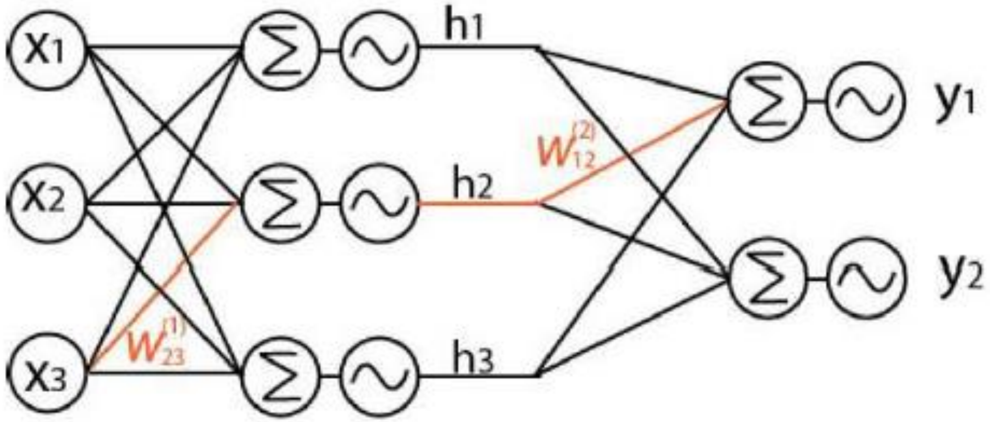
x_i : feature i of input variable x.

$\hat{y}_j$:j-th output variable.

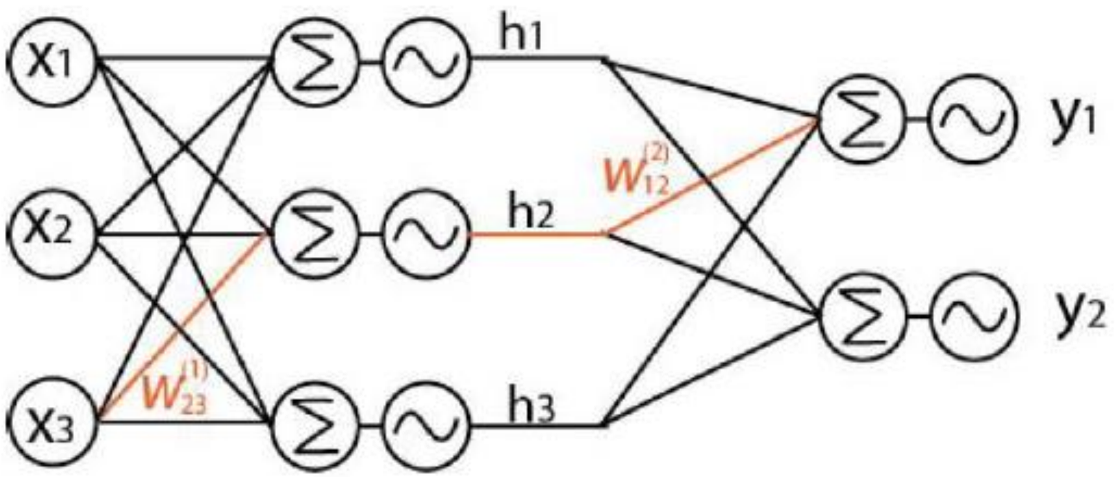
y_j :j-th true output lable.

$w_{ij}^{(k)}$: weight of the output of node j in layer k to node i in layer $k+1$.

$h_i^{(k)}$: output of node i of hidden layer k.



Forward Propagation and Backpropagation



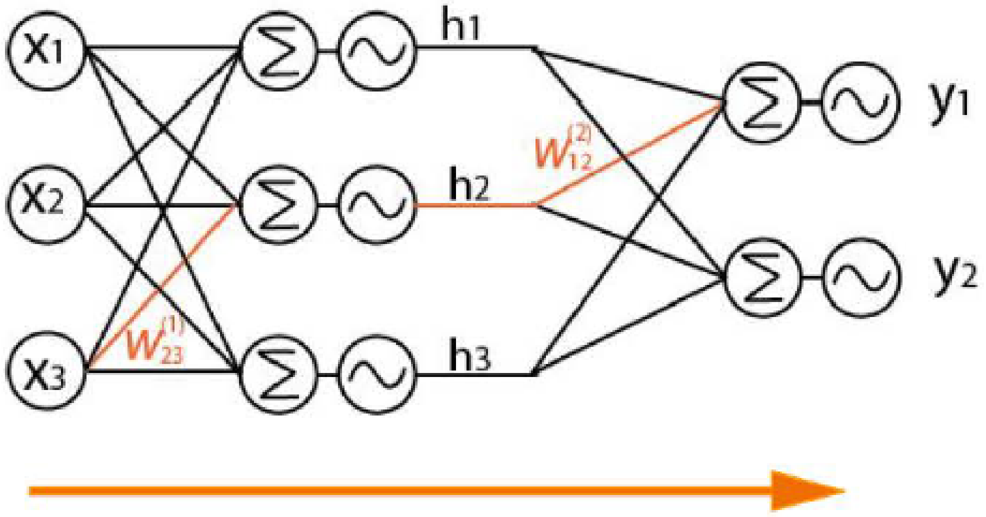
- Forward pathing to make predictions and obtain prediction error.
- Backward pathing to learn the model parameters using the prediction error.

Forward Propagation

- Forward pathing to make predictions

$$h_j = \sigma(\sum_i x_i w_{ji}^{(1)})$$

$$\hat{y}_k = \sigma(\sum_j h_j w_{kj}^{(2)})$$



- Then calculate the error, or the cost function:

$$\text{cost function: } J(w,x,y) = \frac{1}{2}\sum_k (\hat{y}_k - y_k)^2$$

Backpropagation

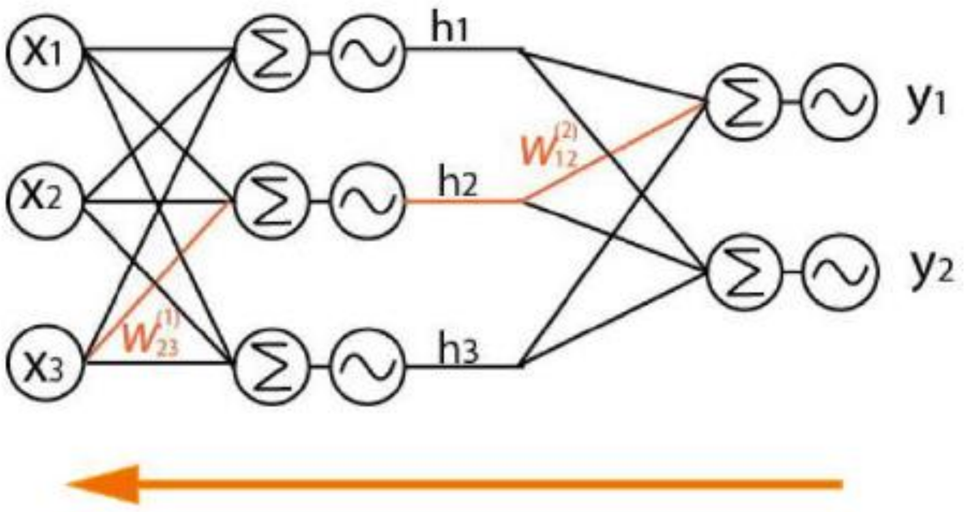
- Backpropagation is short for "backward propagation of errors".
- Commonly referred to the **gradient descent** method to find the optimal weights.
- Recall the general gradient descent method:

Repeat until convergence:

{

$$w_i \leftarrow w_i - \alpha \frac{\partial J(w)}{\partial w_i}, \text{ for every } i$$

}



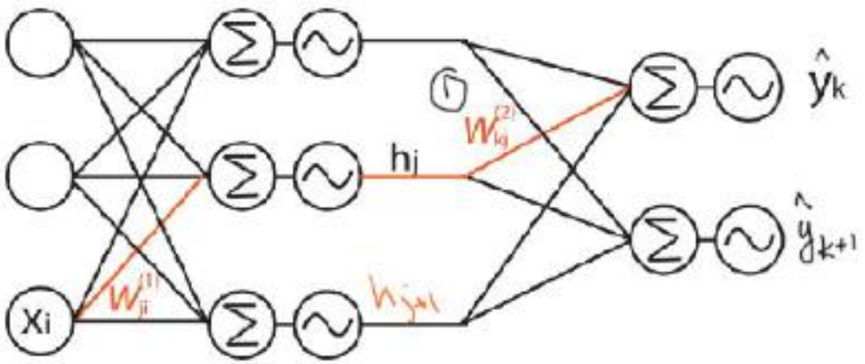
Backpropagation

Recall: Chain rule for partial derivatives:

Let $x = x(u,v)$ and $y = y(u,v)$ have first-order partial derivatives at the point (u,v) and suppose that $z = f(x,y)$ is differentiable at the point $(x(u,v), y(u,v))$. Then $f(x(u,v), y(u,v))$ has first-order partial derivatives at (u,v) given by

$$\begin{aligned}\frac{\partial z}{\partial u} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}\\ \frac{\partial z}{\partial v} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}.\end{aligned}$$

Backpropagation



$$h_j = \sigma(\sum_i x_i w_{ji}^{(1)})$$

$$\hat{y}_k = \sigma(\sum_j h_j w_{kj}^{(2)})$$

$$J(w, x, y) = \frac{1}{2}\sum_k (\hat{y}_k - y_k)^2$$

Calculate the gradient

$$\frac{\partial J(w)}{\partial w_{kj}^{(2)}} = \frac{\partial \frac{1}{2}\sum_k (\hat{y}_k - y_k)^2}{\partial w_{kj}^{(2)}}$$

$$= \frac{\partial \frac{1}{2}\sum_k (\hat{y}_k - y_k)^2}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial w_{kj}^{(2)}}$$

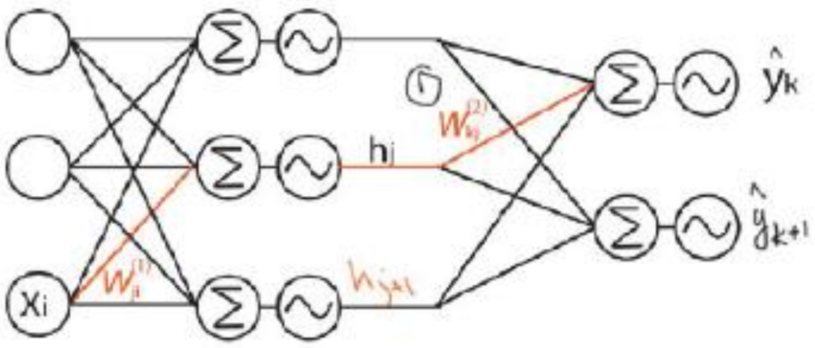
$$= (\hat{y}_k - y_k)\frac{\partial \sigma\left(\sum_j h_j w_{kj}^{(2)}\right)}{\partial w_{kj}^{(2)}}$$

$$= (\hat{y}_k - y_k)\sigma'\left(\sum_j h_j w_{kj}^{(2)}\right) h_j$$

Where $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Similar gradient can be calculated for $\frac{\partial J(w)}{\partial w_{ji}^{(1)}}$.

Backpropagation (Cont'd)

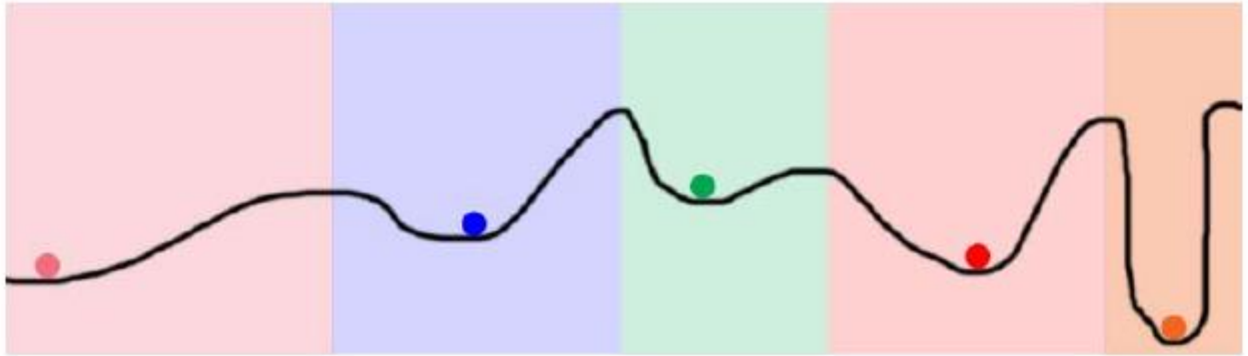


$$\frac{\partial J(w)}{\partial w_{kj}^{(2)}} = (\hat{y}_k - y_k)\sigma'\left(\sum_j h_j w_{kj}^{(2)}\right) h_j$$

$$\frac{\partial J(w)}{\partial w_{ji}^{(1)}} = \sum_k (\hat{y}_k - y_k)\sigma'\left(\sum_j h_j w_{kj}^{(2)}\right) w_{kj}^{(2)} \cdot \sigma'\left(\sum_i w_{ji}^{(1)} x_i\right) x_i$$

Computational cost is linear with feedforward NN depth!

Dropout



First Example

Inputs			Output
0	0	1	0
1	1	1	1
1	0	1	1
0	1	1	0

Denotation

Variable	Definition
X	Input dataset matrix where each row is a training example
y	Output dataset matrix where each row is a training example
l0	First Layer of the Network, specified by the input data
l1	Second Layer of the Network, otherwise known as the hidden layer
syn0	First layer of weights, Synapse 0, connecting l0 to l1.
*	Elementwise multiplication, so two vectors of equal size are multiplying corresponding values 1-to-1 to generate a final vector of identical size.
-	Elementwise subtraction, so two vectors of equal size are subtracting corresponding values 1-to-1 to generate a final vector of identical size.
x.dot(y)	If x and y are vectors, this is a dot product. If both are matrices, it's a matrix-matrix multiplication. If only one is a matrix, then it's vector matrix multiplication.

Second Example

Inputs			Output
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

Denotation

Variable	Definition
X	Input dataset matrix where each row is a training example
y	Output dataset matrix where each row is a training example
l0	First Layer of the Network, specified by the input data
l1	Second Layer of the Network, otherwise known as the hidden layer
l2	Final Layer of the Network, which is our hypothesis, and should approximate the correct answer as we train.
syn0	First layer of weights, Synapse 0, connecting l0 to l1.
syn1	Second layer of weights, Synapse 1 connecting l1 to l2.
l2_error	This is the amount that the neural network "missed".
l2_delta	This is the error of the network scaled by the confidence. It's almost identical to the error except that very confident errors are muted.
l1_error	Weighting l2_delta by the weights in syn1, we can calculate the error in the middle/hidden layer.
l1_delta	This is the l1 error of the network scaled by the confidence. Again, it's almost identical to the l1_error except that confident errors are muted.