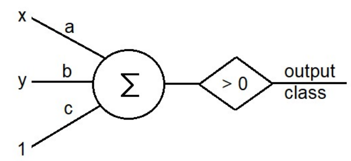
Computer Vision
and
Machine Learning

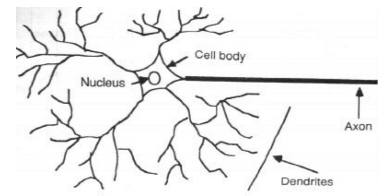
(Neural Network-2)

Bhabatosh Chanda

bchanda57@gmail.com

Linear classifier and neuron



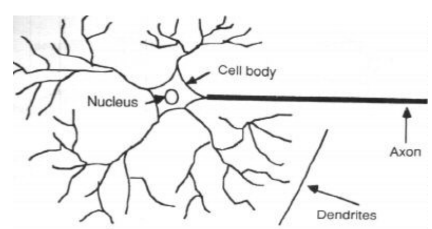


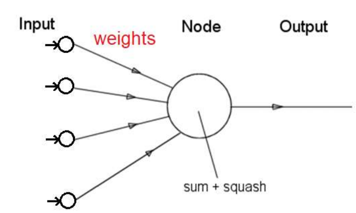
What are Artificial Neural Networks?

- Mimics the function of the brain and nervous system
- Highly parallel
  - Process information much more like the brain than a serial computer
- Learning

- Very simple principles
- Very complex behaviours

Neuron versus Node





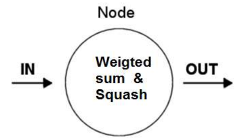
Function of a node

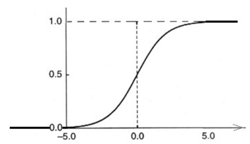
- At node

  Output $O = \sigma(\sum w_i x_i)$

where $\sigma(.)$ is a squashing function.

- Squashing function limits node output.





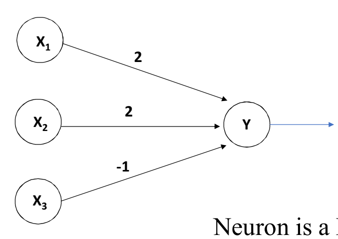
Neural Networks: History

- McCulloch & Pitts (1943) are generally recognised as the designers of the first neural network
- Many of their ideas still used today (e.g. many simple units combine to give increased complexity and the idea of a threshold).
- 1949-First learning rule
- 1969-Minsky & Papert - perceptron limitation - Death of ANN
- 1980's - Re-emergence of ANN - multi-layer networks

Theory of Back Propagation Neural Net (BPNN)

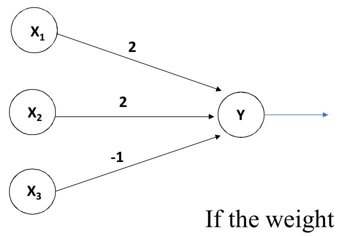
- Use many samples to train the weights (W), so it can be used to classify an unknown input into different classes.
- Will explain
  - How to use it after training: forward pass (classification or recognition of the input).
  - How to train it: how to train the weights and biases (using forward and backward passes).

The First Neural Networks



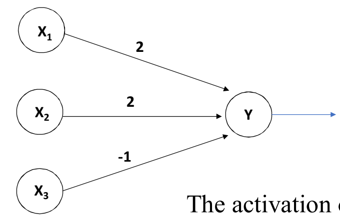
Neuron is a McCulloch-Pitts network are connected by directed, weighted paths.

The First Neural Networks



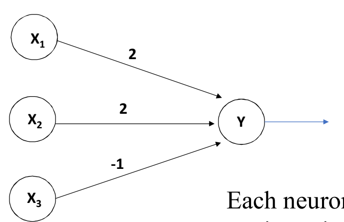
If the weight on a path is positive the path is excitatory, otherwise it is inhibitory.

The First Neural Networks



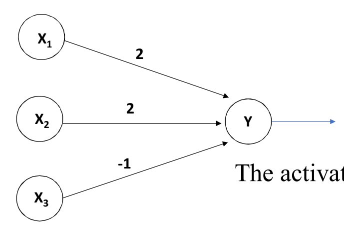
The activation of a neuron is binary. That is, the neuron either fires (activation of one) or does not fire (activation of zero).

The First Neural Networks



Each neuron has a fixed threshold. If the net input into the neuron is greater than the threshold, the neuron fires.

The First Neural Networks

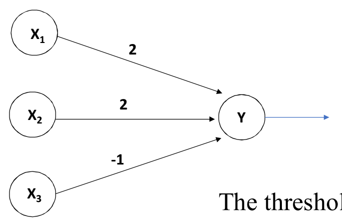


The activation function for unit Y is

$$\sigma(Y) = \begin{cases} 1, & \text{if } Y >= \theta \\ 0, & \text{otherwise} \end{cases}$$

where Y is the total input signal received
θ is the threshold for the node.

The First Neural Networks

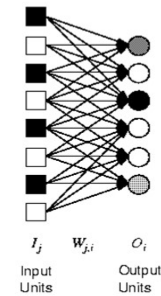


The threshold is set such that any non-zero inhibitory input will prevent the neuron from firing.

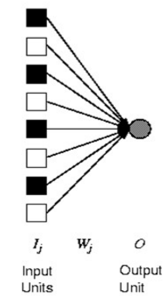
Perceptron network

- Synonym for single layer, feed-forward network capable of learning.
- Output $O = f\left(\sum_j W_j I_j + b_j\right)$

where 'b' is bias, which however, may be included as additional weight.



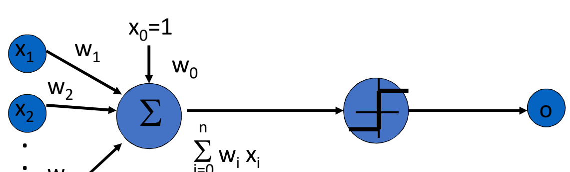
Perceptron Network



Single Perceptron

Perceptron

- Linear treshold unit (LTU)



$$o(x_i)=\begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i I_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Standard activation functions

- The hard-limiting threshold function
  - Corresponds to the biological paradigm
    - either fires or not ***(Perceptron)***
- Sigmoid functions ('S'-shaped curves)
  - The hyperbolic tangent (symmetrical)
  - Both functions have a simple differential
  - Only the shape is important ***(Neuron)***

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

Data

- Input data is presented to the network in the form of activations in the input layer
- Examples
  - Pixel intensity (for pictures)
  - Share prices (for stock market prediction)
- Data usually requires pre-processing
  - Analogous to senses in biology
- How to represent more abstract data, e.g. a label?
  - Choose a pattern, e.g., 0-0-0-0-1-0-0-0-0 for "digit 5"

Loss function or Error or Cost function

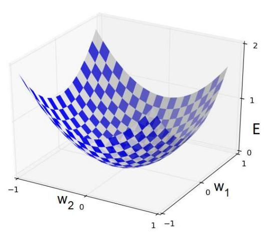
- Training sample is composed of
  - Input data (feature vector) and
  - Actual class label (also known as groundtruth)
- Given the input, feed forward network predicts class label
  - based on current parameters
  - Loss or error or cost is measured as total deviation from groundtruth

<u>Cost</u> or <u>Loss</u> or <u>Error</u>: $E(\boldsymbol{w}) = \sum(\textit{Predicted label} - \textit{Actual label})^2$

where $\boldsymbol{w}$ is parameter vector.

Training the network

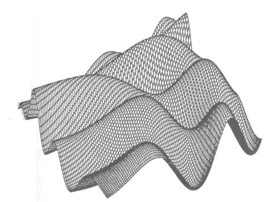
- Means setting correct weights (including bias) or parameters of the network.
- Backpropagation
  - Requires training set (input / output pairs)
  - Starts with small random weights
  - Compute error between predicted label and actual label (groundtruth)
  - Error is used to adjust weights (supervised learning)
  - → Gradient descent on error landscape



$E(\boldsymbol{w})$

Training the network

- Means setting correct weights (including bias) or parameters of the network.
- Backpropagation
  - Requires training set (input / output pairs)
  - Starts with small random weights
  - Compute error between predicted label and actual label (groundtruth)
  - Error is used to adjust weights (supervised learning)
  - → Gradient descent on error landscape



$E(\boldsymbol{w})$

Maths: Weight setting by gradient descent

- Error function: $E(w) = \frac{1}{2n} \sum_x \| y(x,w) - a_x \|^2$
- A small change in error E may be given by

$$\Delta E \approx \frac{\partial E}{\partial w_1} \Delta w_1 + \frac{\partial E}{\partial w_2} \Delta w_2 = \begin{pmatrix} \frac{\partial E}{\partial w_1} & \frac{\partial E}{\partial w_2} \end{pmatrix}^T \cdot (\Delta w_1 \quad \Delta w_2)^T = \nabla E \cdot \Delta w$$

- Let $\Delta w = -\eta \nabla E$ which implies $\Delta E = -\eta \| \nabla E \|^2 \leq 0$
- This suggests updating weights as $w_k^{(t+1)} = w_k^{(t)} - \eta \frac{\partial E}{\partial w_k}$

Time requirement

- Error function: $E(\boldsymbol{w}) = \frac{1}{2n} \sum_{\boldsymbol{x}} ||\hat{y}(\boldsymbol{x}, \boldsymbol{w}) - y(\boldsymbol{x})||^2$
- This suggests updating weights as $w_k^{(t+1)} = w_k^{(t)} - \eta \frac{\partial E}{\partial w_k}$
- Error due to individual input $\boldsymbol{x}$, $E_{\boldsymbol{x}} = \frac{||\hat{y}(\boldsymbol{x},\boldsymbol{w}) - y(\boldsymbol{x})||^2}{2}$ and

$$E(\boldsymbol{w}) = \frac{1}{n} \sum_{\boldsymbol{x}} E_{\boldsymbol{x}}$$

- This implies that error due to all inputs are computed before updating the weights.
  - Huge training time is needed if n is large (which is the case always).

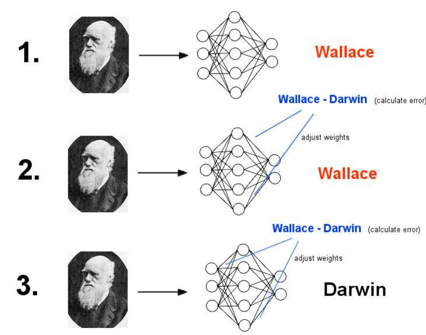
Weight setting by stochastic gradient descent

- To speed up the training process, stochastic gradient descent may be adopted.
  - Randomly m ($\ll n$) samples from the training data set may be picked up.
  - Suppose these samples are $\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \tilde{\boldsymbol{x}}_3, \ldots, \tilde{\boldsymbol{x}}_m$.
- Error due to individual input $\tilde{\boldsymbol{x}}_i$, $E_{\tilde{\boldsymbol{x}}_i} = \frac{||\hat{y}(\tilde{\boldsymbol{x}}_i,\boldsymbol{w}) - y(\tilde{\boldsymbol{x}}_i)||^2}{2}$ and

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{m} \sum_i E_{\tilde{\boldsymbol{x}}_i}$$

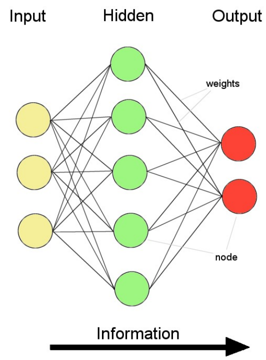
- If m is sufficiently large, $\tilde{E}(\boldsymbol{w}) \approx E(\boldsymbol{w})$

Training the network: Example



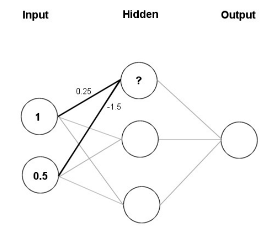
Feed-forward nets

- Information flow is unidirectional
  - Data is presented to *Input layer*
  - Passed on to *Hidden Layer*
  - Passed on to *Output layer*
- Information is distributed
- Information processing is parallel
- True while testing new data



Example: node function

- Feeding data through the net:



(1 × 0.25) + (0.5 × (-1.5)) = 0.25 + (-0.75) = **- 0.5**

Squashing: $\frac{1}{1+e^{0.5}} = 0.3775$

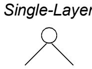
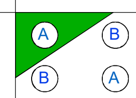
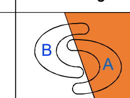
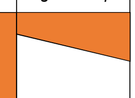
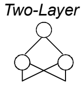
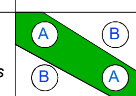
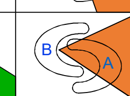
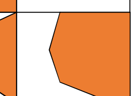
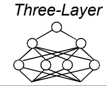
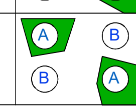
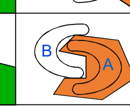
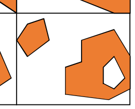
Machine learning network

Machine learning models for classification have followings are common:

- ***Input layer:*** quantitative representation of object features
- ***Hidden layer(s):*** apply transformations with nonlinearity
- ***Output layer:*** Result for classification, regression etc.

- The models are trained through ***supervised learning***.
  - Training data are explicitly labelled (known output).
  - Weights are updated to minimize error between prediction and the groundtruth.

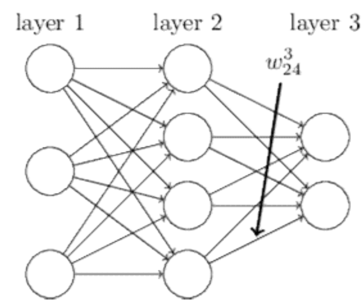
Different Non-Linearly Separable Problems

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Backpropagation

- Algorithm proposed in 1970.
- Became convincingly popular in 1986 due to a paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams.
- At the core of backpropagation is an expression for the partial derivative of Error function with respect to weights, i.e., $\frac{\partial E}{\partial w}$

Weights of neural network



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

$$w^{to\ the\ layer}_{to\ neuron,\ from\ neuron}$$

Output of j-th node at the l-th layer

- Input to l-th layer is coming from $(l-1)$-th layer, i.e., $\boldsymbol{y}^{(l-1)}$.
- Suppose there are K nodes in the $(l-1)$-th layer.
  - $\boldsymbol{y}^{(l-1)} = \left(1,\ y_1^{(i-1)}, y_2^{(l-1)}, y_3^{(l-1)}, \ldots, y_{K-1}^{(l-1)}\right)^T$
- Weight of the connection from k-th node of the $(l-1)$-th layer to the j-th node of the l-th layer is $w_{jk}^{(l)}$.
  - $\boldsymbol{w}_j^{(l)} = \left(w_{j0}^{(l)}, w_{j1}^{(l)}, w_{j2}^{(l)}, \ldots, w_{jK-1}^{(l)}\right)^T$, where $w_{j0}^{(l)}$ is the weight to the bias.

Output of j-th node at the l-th layer

- Output of the j-th node at the l-th layer is

$$y_j^{(l)} = \sigma\left(\left(\boldsymbol{w}_j^{(l)}\right)^T \boldsymbol{y}^{(l-1)}\right)$$

where $l = 1, 2, 3, \ldots, L$ and that means the NN has $L-1$ hidden layers.

- Note that at the input layer, i.e., $\boldsymbol{y}^{(0)} = \boldsymbol{x}$ and output is $\boldsymbol{y}^{(L)} = \widehat{\boldsymbol{y}}$.
- Let us decompose $y_j^{(l)} = \sigma\left(\left(\boldsymbol{w}_j^{(l)}\right)^T \boldsymbol{y}^{(l-1)}\right)$ into

$$y_j^{(l)} = \sigma\left(z_j^{(l)}\right) \text{ where } z_j^{(l)} = \sum_k w_{jk}^{(l)} y_k^{(l-1)}$$

Chain rule to compute

- Considering single output node, rewrite $y_j^{(l)} = \sigma\left(\left(\boldsymbol{w}_j^{(l)}\right)^T \boldsymbol{y}^{(l-1)}\right)$ as

  $y_j^{(l)} = \sigma\left(z_j^{(l)}\right)$ where $z_j^{(l)} = \sum_k w_{jk}^{(l)} y_k^{(l-1)}$
- Following the chain rule:

$$\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = y^{(L)} = \sigma\left(\sum_k w_{jk}^{(L)} y_k^{(L-1)}\right)$$
$$= \sigma\left(\sum_k w_{jk}^{(L)} \sigma\left(z_k^{(L-1)}\right)\right)$$
$$= \sigma\left(\sum_k w_{jk}^{(L)} \sigma\left(\sum_m w_{km}^{(L-1)} y_m^{(L-2)}\right)\right) \ \cdots$$

Derivative of function of functions

- Suppose we have $f(x) = log_e(sin(x^2))$
- Consider $f(x) = f_1(y)$ where $y = sin(x^2)$
- then $y = f_2(z)$ where $z = x^2$
- then $z = f_3(x)$
- Thus $f(x) = f_1(y) \Rightarrow f(x) = f_1(f_2(z)) \Rightarrow f(x) = f_1(f_2(f_3(x)))$
- $\frac{df}{dx} = \frac{df_1}{df_2}\frac{df_2}{df_3}\frac{df_3}{dx}$ OR $\frac{df}{dx} = \frac{df}{dy}\frac{dy}{dz}\frac{dz}{dx}$ OR $\frac{df}{dx} = \frac{1}{sin(x^2)}\cos(x^2)2x$

Backpropagation

- We do not have groundtruth at the output of every layer, except the final layer, change in weight at any layer is related to the change in error ΔE as

$$\Delta E = \frac{\partial E}{\partial w_{jk}^{(l)}} \Delta w_{jk}^{(l)}$$

Recall that

$$\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = y^{(L)} = \sigma\left(\sum_k w_{jk}^{(L)} \sigma\left(\sum_m w_{km}^{(L-1)} y_m^{(L-2)}\right)\right) \ \cdots$$

Backpropagation (contd.)

- However, to compute total change in error ΔE due to change in weights of k-th node of $(l-1)$-th layer connected to the j-th node of l-th layer, it is plausible that we should sum over all possible paths from k-th node of the $(l-1)$-th layer to the final layer, i.e.,

$$\Delta E \approx \sum_{mnp\ldots qr} \frac{\partial E}{\partial y_m^{(L)}} \frac{\partial y_m^{(L)}}{\partial y_n^{(L-1)}} \frac{\partial y_n^{(L-1)}}{\partial y_p^{(L-2)}} \cdots \frac{\partial y_q^{(l+1)}}{\partial y_r^{(l)}} \frac{\partial y_r^{(l)}}{\partial w_{jk}^{(l)}} \Delta w_{jk}^{(l)}$$

Backpropagation (contd.)

- Now combining following two equations:

$$\Delta E = \frac{\partial E}{\partial w_{jk}^{(l)}} \Delta w_{jk}^{(l)} \quad \text{and}$$

$$\Delta E \approx \sum_{mnp\ldots qr} \frac{\partial E}{\partial y_m^{(L)}} \frac{\partial y_m^{(L)}}{\partial y_n^{(L-1)}} \frac{\partial y_n^{(L-1)}}{\partial y_p^{(L-2)}} \cdots \frac{\partial y_q^{(l+1)}}{\partial y_r^{(l)}} \frac{\partial y_r^{(l)}}{\partial w_{jk}^{(l)}} \Delta w_{jk}^{(l)}$$

- We obtain

$$\frac{\partial E}{\partial w_{jk}^{(l)}} = \sum_{mnp\ldots qr} \frac{\partial E}{\partial y_m^{(L)}} \frac{\partial y_m^{(L)}}{\partial y_n^{(L-1)}} \frac{\partial y_n^{(L-1)}}{\partial y_p^{(L-2)}} \cdots \frac{\partial y_q^{(l+1)}}{\partial y_r^{(l)}} \frac{\partial y_r^{(l)}}{\partial w_{jk}^{(l)}}$$

Updating weight

- Error function: $E(\boldsymbol{w}) = \frac{1}{2n} \sum_x ||\hat{y}(\boldsymbol{x}, \boldsymbol{w}) - y(\boldsymbol{x})||^2$

where $\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_k w_{jk}^{(l)} \sigma\left(\sum_m w_{km}^{(l-1)} y_m^{(l-2)}\right)\right) \cdots$

- Earlier we had (for single layer): $w_k^{(t+1)} = w_k^{(t)} - \eta \frac{\partial E}{\partial w_k}$
- Now updating weight from k-th node of $(l-1)$-th layer to j-th node of l-th layer as

$$w_{jk}^{(l)(t+1)} = w_{jk}^{(l)(t)} - \eta \frac{\partial E}{\partial w_{jk}^{(l)(t)}}$$

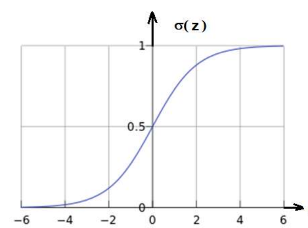
Standard activation functions

- The hard-limiting threshold function
  - Corresponds to the biological paradigm
    - either fires or not ***(Perceptron)***
- Sigmoid functions ('S'-shaped curves)
  - The hyperbolic tangent (symmetrical)
  - Both functions have a simple differential
  - Only the shape is important ***(Neuron)***

$$\sigma(z) = \frac{1}{1 + e^{-az}}$$

Sigmoid function

- Sigmoid function
  - may be expressed as
    $$\sigma(z) = \frac{1}{1 + e^{-\alpha z}}$$
  - is one of the most popular activation function.
  - squashes the input value between 0 and 1.
  - is smooth and differentiable.
  - maximum slope is at $z = 0$



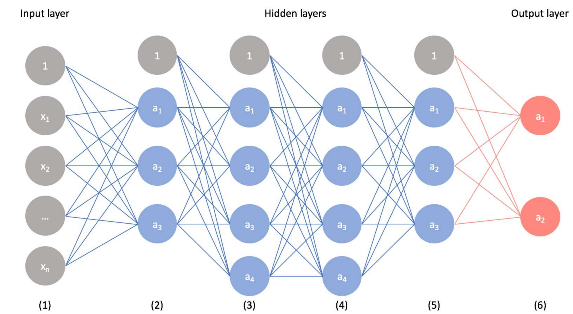
Derivative of sigmoid function

- We have $y = \sigma(z) = \frac{1}{1+e^{-\alpha z}}$
- Derivative of $\sigma(z)$ at $z = 0$ may be written as

$$\frac{d\sigma}{dz}|_{z=0} = \frac{e^{-\alpha}}{(1+e^{-\alpha z})^2}|_{z=0} = \frac{1}{(1+1)^2} = 0.25$$

- This is the maximum value of gradient for any z.

Multilayer neural network: Example



Vanishing gradient problem

- Number of layers are usually approximates the degree of polynomial function it can realize.
- However, more layers means more neurons and consequently more time to train the network.
- Second, since the derivative of the activation function (resulting in output at each layer) ≤ 0.25,

$$\frac{\partial E}{\partial w_{jk}^{(l)}} = \sum_{mn \;\; \ldots qr} \frac{\partial E}{\partial y_m^{(L)}} \frac{\partial y_m^{(L)}}{\partial y_n^{(L-1)}} \frac{\partial y_n^{(L-1)}}{\partial y_p^{(L-2)}} \cdots \frac{\partial y_q^{(l+1)}}{\partial y_r^{(l)}} \frac{\partial y_r^{(l)}}{\partial w_{jk}^{(l)}}$$

- May tend to zero. This is known as *vanishing gradient problem*.
- This problem is more evident as we deeper layers from output input.

Vanishing gradient problem

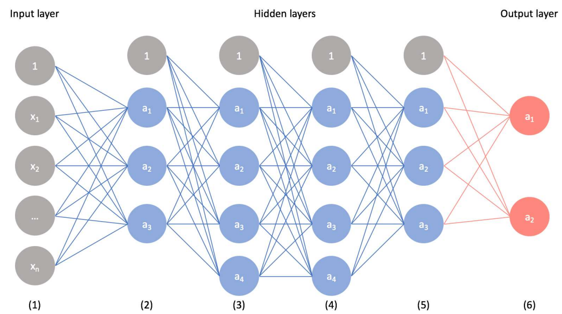
- Recall the weight updating rule

$$w_{jk}^{(l)(t+1)} = w_{jk}^{(l)(t)} - \eta \frac{\partial E}{\partial w_{jk}^{(l)(t)}}$$

- If $\frac{\partial E}{\partial w_{jk}^{(l)(t)}} \to 0$, we have $w_{jk}^{(l)(t+1)} \approx w_{jk}^{(l)(t)}$

- The first layers are supposed to carry most of the information, but we see it gets trained the least.
- Hence, the problem of vanishing gradient eventually leads to the death of the network.

Vanishing gradient problem

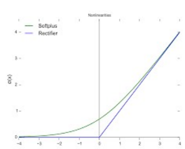


Exploding gradient problem

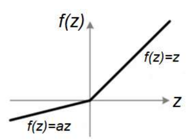
- Suppose vanishing gradient problem does not occur.
- Then $\frac{\partial E}{\partial w_{jk}^{(l)}} = \sum_{mn\ \ldots qr} \frac{\partial E}{\partial y_m^{(L)}} \frac{\partial y_m^{(L)}}{\partial y_n^{(L-1)}} \frac{\partial y_n^{(L-1)}}{\partial y_p^{(L-2)}} \cdots \frac{\partial y_q^{(l+1)}}{\partial y_r^{(l)}} \frac{\partial y_r^{(l)}}{\partial w_{jk}^{(l)}}$ implies that
- $\frac{\partial E}{\partial w_{jk}^{(l)}}$ is a sum of gradient magnitude along $m \times n \times p \times \cdots \times q \times r$ number of paths, where each gradient is greater than 0.
- Thus this sum could be significantly high resulting in *exploding gradient problem*.

Activation function (non-linear)

- Rectified Linear Unit (ReLU): $y = \max(0, x)$
  - Softplus function: $y = \log(1 + e^x)$



- Leaky ReLU:



Some hyperparameters

- ***Epoch:*** Suppose there are n samples in the training set. Passing (or using) all n samples to train the network is known as one epoch.
  - To train the network we need pass the training samples over and over again.
  - As the number of epoch increases network upgrades from underfitting to optimum to overfitting.
- ***Batch:*** If n is large, training set is divided into small batches or groups or sets of training data. *Batch size* is the number of training samples, say m, in each batch.
- ***Iteration:*** The number of batches that are passed through the network to complete one epoch, *i.e.*, n/m.

Batch normalization

- As the training progresses the network encounters (or being feed into) newer data
  - The statistical distribution of the input data keeps changing.
  - The statistical distribution of the input data in different batches are different.
  - This reduces training efficiency.
- The input samples (in every batch) are normalized before feeding it into the network.
  - The mean and variance of all such batches, instead of the entire data, are computed.
  - This is known as *batch normalization*.

Dropout

- This is used to overcome the overfitting problem.
- Often certain nodes in the network are randomly switched off, from some or all the layers of a neural network.
  - Hence, in every iteration, we get a new network.
  - The resulting network (obtained at the end of training) is a combination of all of them.
  - This is an way of implementing the *regularization*.

Thank you!

Any question?