Computer Vision
and
Machine Learning

(Neural Network-1)

Bhabatosh Chanda
bchanda57@gmail.com

Regression

- Regression is a technique to establish relation between independent variables or primary observations (features) and dependent variables.
- Depending on type of relation between dependent variable (*decision or prediction*) and independent variable(s), we may classify the regression as
  - *Linear regression*
  - *Logistic regression*

Machine learning tasks T

- **Classification:** To decide which of the k classes the given input belongs to. Learning system tries to develop a mapping (function)

$$f: R^n \rightarrow \{1, 2, \cdots, k\}$$

- **Prediction:** To predict a numerical value for the given input. So the task is similar to classification except the representation of output. Thus the mapping (function) is

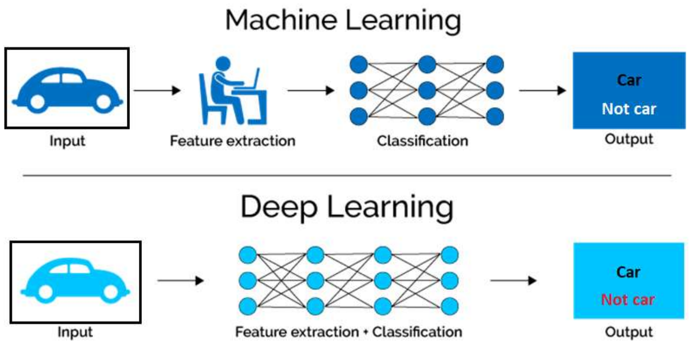
$$f: R^n \rightarrow R$$

Idea of machine learning

- A system (here, machine or computer) is said to have learned
  - to do some ***task*** T
  - from a set of ***examples*** E
  - in terms of a ***performance measure*** P,

if its performance improves
  - as measured by the same P
  - to carry out the same task T
  - by dealing with the example set E.

Machine learning vs. Deep learning



Linear regression

- Task is to build a system to predict a scalar value $y \in R$ as output from the given input $\boldsymbol{x} \in R^n$.
- Suppose $\hat{y}$ is the value predicted by the system, i.e.,

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$$

  where $\boldsymbol{w} \in R^n$ is parameter vector that controls behaviour of system.
- Assume $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$, similarly $\boldsymbol{w} = (w_1, w_2, \cdots, w_n)$

Linear regression (contd.)

- A more general relation between $\boldsymbol{x} \in R^n$ and $y \in R$ may be expressed as

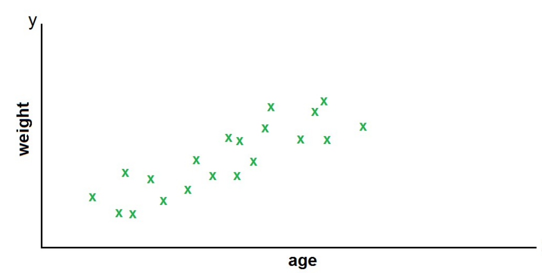
$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$$

- If we append a '1' to $\boldsymbol{x}$ and including 'b' as a weight
  - Relation between y and $\boldsymbol{x}$ becomes affine, but
  - Relation between y and $\boldsymbol{w}$ remains linear.
- Consider $\boldsymbol{x} = (x_0, x_1, x_2, \cdots, x_n)$ and $\boldsymbol{w} = (w_0, w_1, w_2, \cdots, w_n)$

  where $x_0 = 1$ and $w_0 = b$.

Linear regression (contd.)

Example:



Linear regression (contd.)

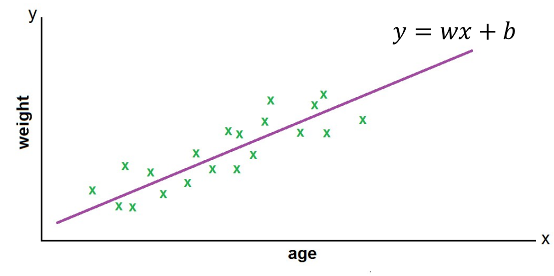
- x → independent variable (e.g., age of a deer, time in quarter, etc.)
- y → dependent variable (resp., weight of a deer, pairs of shoes sold)
- Let us consider relation between x and y may be modeled as a straight line:

$$y = wx + b$$

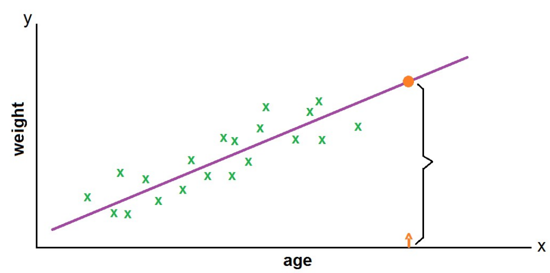
- Exploiting linear regression technique, we estimate

$$w = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sum(x-\bar{x})^2} \qquad b = \bar{y} - w\bar{x}$$

Linear regression (contd.)



Prediction



Prediction: multiple input

- So far we have discussed the cases where input is a single variable.

$$y = f(x)$$

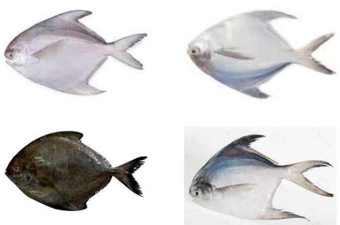
- No. of input variables (independent variables) may be more than 1.

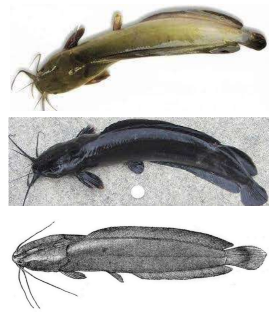
$$y = f(x_1, x_2, x_3, \ldots, x_n)$$

- A contrived example may have following input variables:

Variable	1	2	3	4	5	6	7	8	9	10
x_1	37	42	38	34	41	42	36	40	39	43
x_2	95	93	97	96	98	98	94	97	99	95
y	0	0	0	0	1	1	0	1	1	1

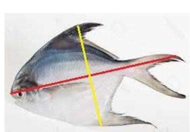
Binary classification: Pomfret and Magur

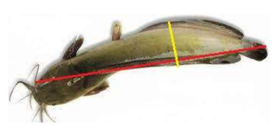




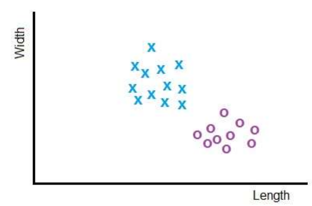
Two class problem: Pomfret and Magur

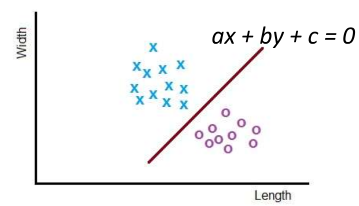
Features: Pomfret and Magur





Two-class problem: Feature space





Boundary function

- Find coefficients a, b and c of equation of a straight line

$$ax + by + c = 0$$

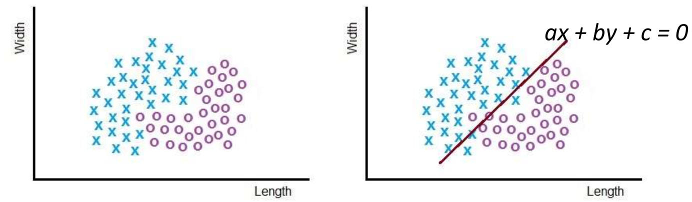
such that for all observation a feature pair (x, y):

$$ax + by + c > 0 \quad \text{if } (x,y) \text{ belongs to } C_1$$

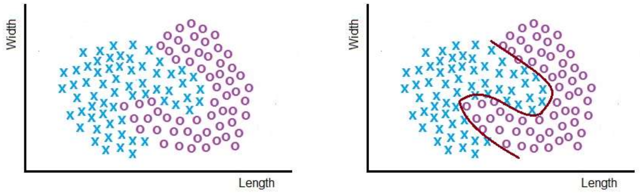
$$ax + by + c < 0 \quad \text{if } (x,y) \text{ belongs to } C_2$$

- If the desired condition is not satisfied for any feature-label pair we call a classification error has occurred.
- In general, decision boundary must be estimated to minimize this error.

Two-class problem



Two-class problem



Generalization

- The ability to perform well on previously unseen data is called ***generalization***.
- The target of machine learning to keep ***generalization error*** or ***test error*** as low as possible.
  - Note that system is built by minimizing the train error.
  - Is there any relation between training error and test error?

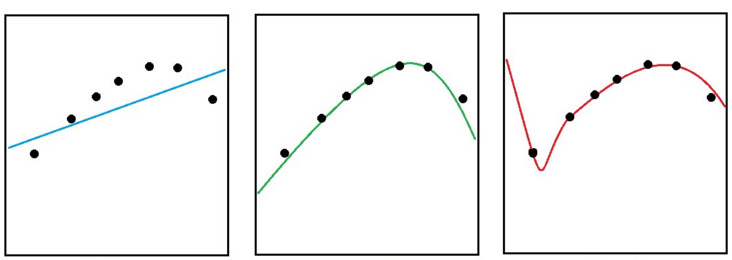
Generalization (contd.)

- Training and test data are accumulated by same data generating process.
  - Each example in training and test datasets are ***independent*** to each other.
  - The training and test datasets are ***identically distributed***.
- The *i.i.d.* assumption allows us to study the relationship between the training error and the test error.
  - Expected training error and the expected test error of a model are equal.

Overfitting and underfitting

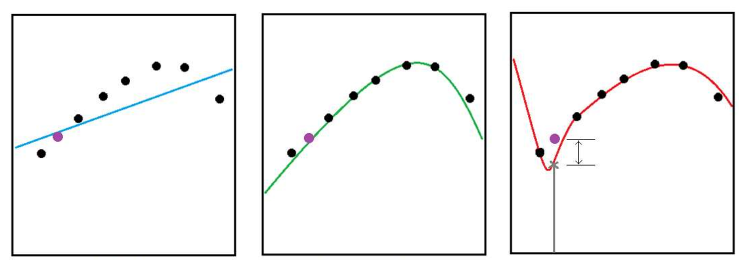
- ***Two criteria*** that determines how well a machine learning algorithm performs are its ability to
  1. make the training error small, and
  2. Make the gap between the training error and the test error small.
- These correspond to two problems: ***overfitting*** and ***underfitting***.
  - If the training error is not small → underfitting
  - If gap between training and test errors is not small → overfitting.

Overfitting and underfitting (contd.)



Underfit **Appropriate** **Overfit**

Overfitting and underfitting (contd.)



Underfit **Appropriate** **Overfit**

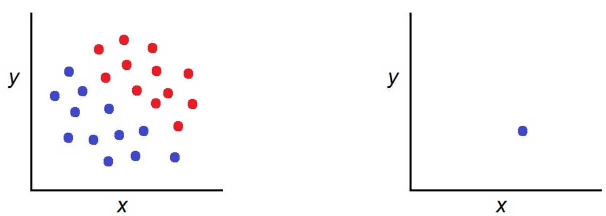
How to set the boundary function

- Based on the training data set.
  - All at a time.
    - Linear discriminant analysis
  - One at a time.
    - Perceptron network, neural network

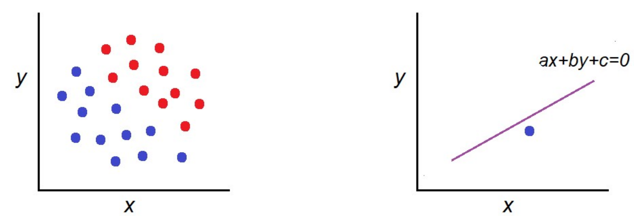
How to set the boundary function

- Based on the training data set.
  - All at a time.
    - Linear discriminant analysis
  - One at a time.
    - Perceptron network, neural network

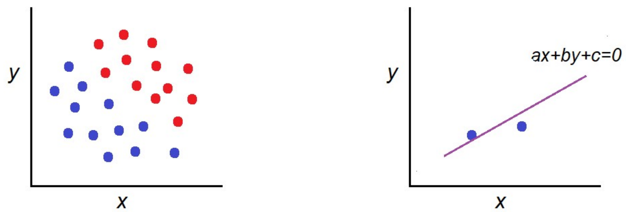
Forming the decision boundary



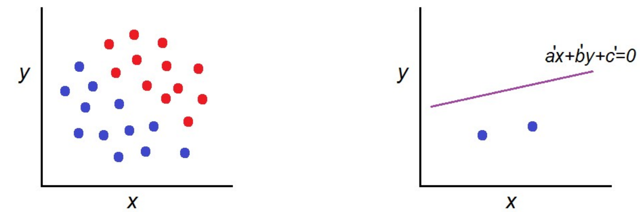
Forming the decision boundary



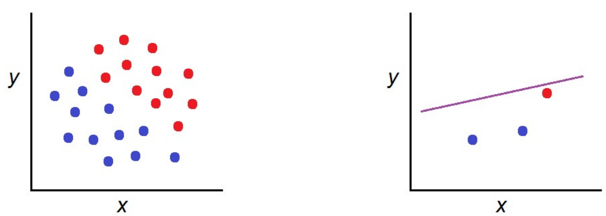
Forming the decision boundary



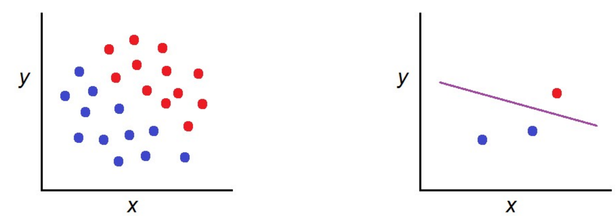
Forming the decision boundary



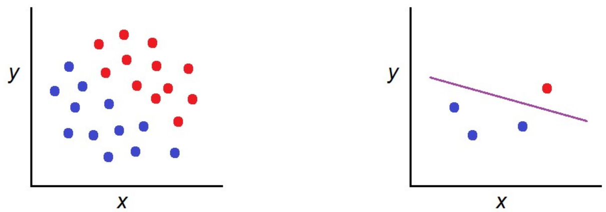
Forming the decision boundary



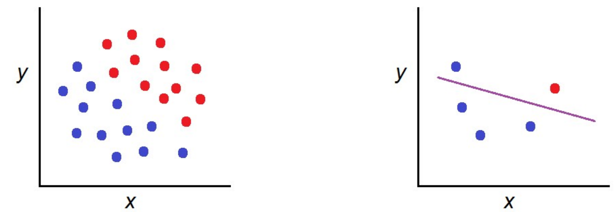
Forming the decision boundary



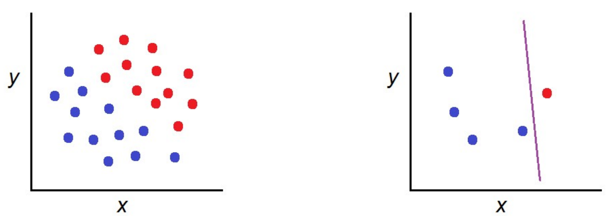
Forming the decision boundary



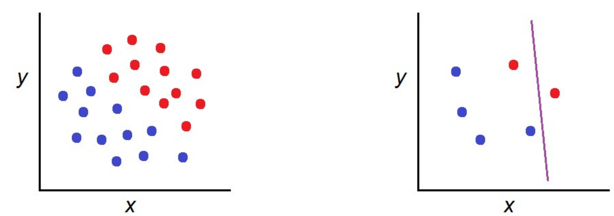
Forming the decision boundary



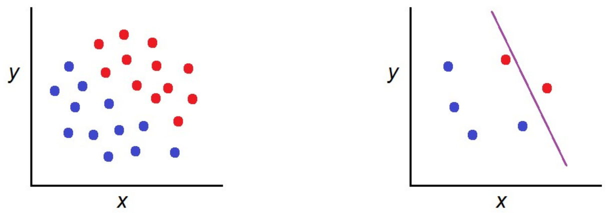
Forming the decision boundary



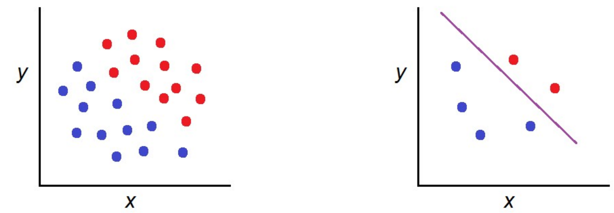
Forming the decision boundary



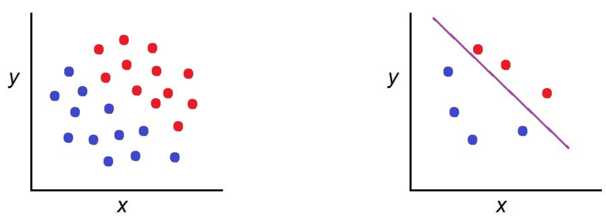
Forming the decision boundary



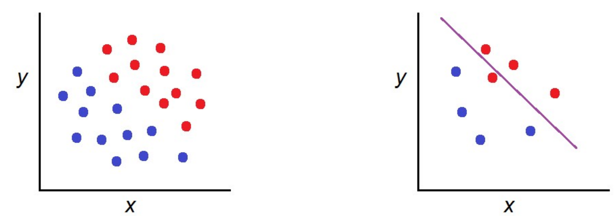
Forming the decision boundary



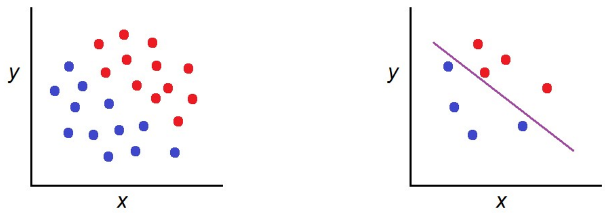
Forming the decision boundary



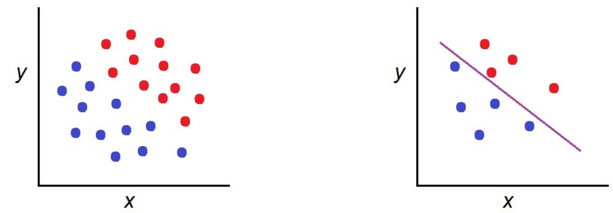
Forming the decision boundary



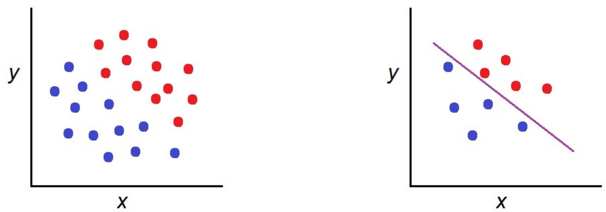
Forming the decision boundary



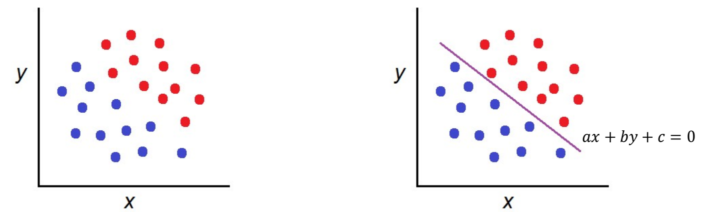
Forming the decision boundary



Forming the decision boundary



Forming the decision boundary



Boundary function

- Find coefficients *a*, *b* and *c* of equation of a straight line

$$ax + by + c = 0$$

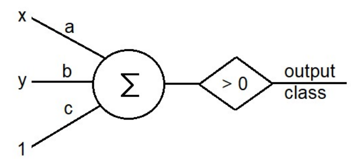
such that for all observation a feature pair (x, y):

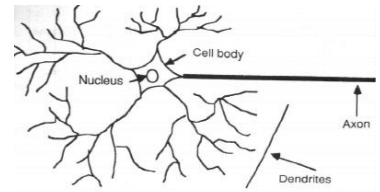
$ax + by + c > 0$ if (x,y) belongs to C_1

$ax + by + c < 0$ if (x,y) belongs to C_2

- If the desired condition is not satisfied for any feature-label pair we call a classification error has occurred.
- In general, decision boundary must be estimated to minimize this error.

Linear classifier and neuron



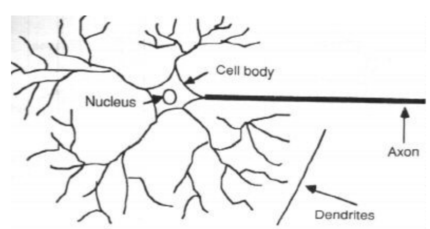


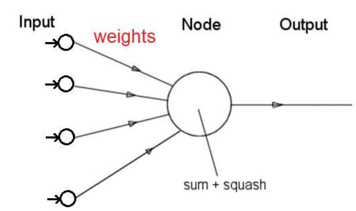
What are Artificial Neural Networks?

- 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

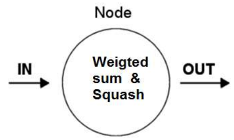


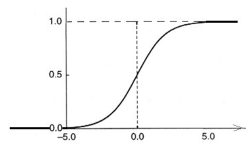


Function of a node

- At node
  Output $O = f(\sum w_i x_i)$
  where $f(.)$ is a squashing function.

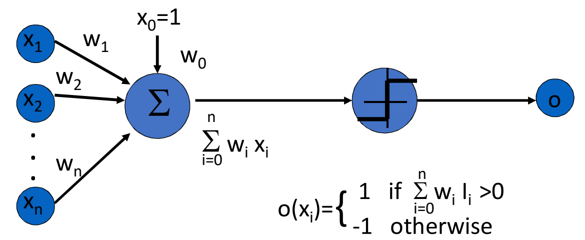
- Squashing function limits node output.





Perceptron

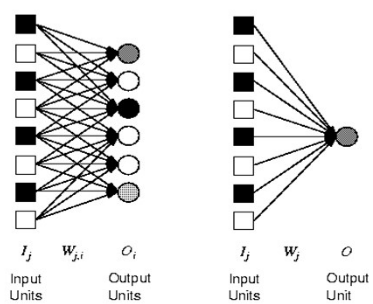
- Linear treshold unit (LTU)



Perceptron network

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

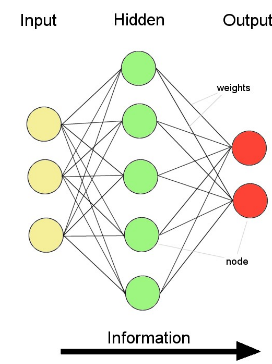
where 'b' is bias, which however, may be included as additional weight.



Perceptron Network **Single Perceptron**

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



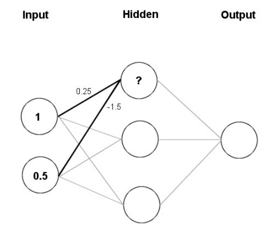
Standard activation functions

- 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)***

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

Example: node function

- Feeding data through the net:



$(1 \times 0.25) + (0.5 \times (-1.5)) = 0.25 + (-0.75) =$ **- 0.5**

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

Loss function or Error or Cost function

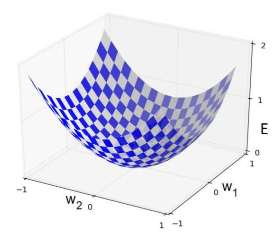
- Training sample is composed of
  - Input data (feature vector) and
  - Actual class label (also known as groundtruth)
- Given the input, feed forward network predicts class label
  - based on current parameters
  - Loss or error or cost is measured as total deviation from groundtruth

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

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(**w**)*

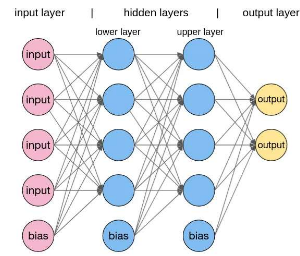
Machine learning network

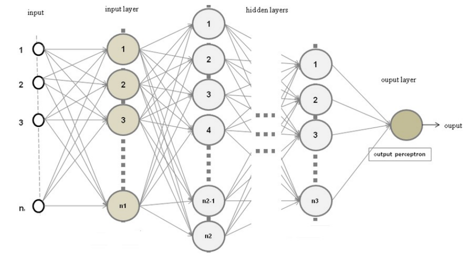
Machine learning models for classification have followings are common:

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

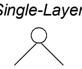
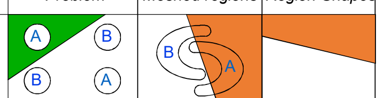
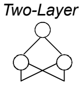
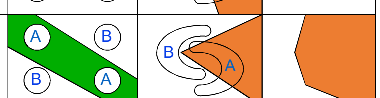
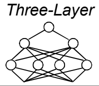
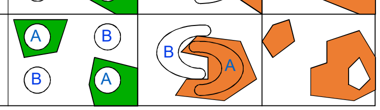
- The models are trained through ***supervised learning***.
  - Training data are explicitly labelled (known output).
  - Weights are updated to minimize error between prediction and the groundtruth.

Multilayer neural networks





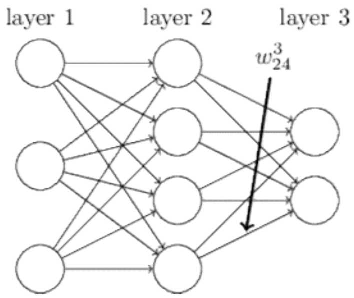
Different Non-Linearly Separable Problems

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_{jk}^{l} 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

$w_{\textit{to neuron}, \textit{from neuron}}^{\textit{to the layer}}$

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.
  - $\boldsymbol{y}^{(l-1)} = \left(1,\ y_1^{(l-1)}, y_2^{(l-1)}, y_3^{(l-1)}, \dots, y_{K-1}^{(l-1)}\right)^T$
- 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 $w_{jk}^{(l)}$.
  - $\boldsymbol{w}_j^{(l)} = \left(w_{j0}^{(l)}, w_{j1}^{(l)}, w_{j2}^{(l)}, \dots, w_{jK-1}^{(l)}\right)^T$, 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 $l = 1, 2, 3, \dots, L$ 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 $\boldsymbol{y}^{(L)} = \hat{\boldsymbol{y}}$.
- 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

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

- Following the chain rule:

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

Derivative of function of functions

- Suppose we have $f(x) = log_e\left(sin(x^2)\right)$
- Consider $f(x) = f_1(y)$ where $y = sin(x^2)$
- then $y = f_2(z)$ where $\mathrm{z} = x^2$
- then $\mathrm{z} = f_3(x)$
- 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.,

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

Backpropagation (contd.)

- Now combining following two equations:

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

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

- We obtain

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

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

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

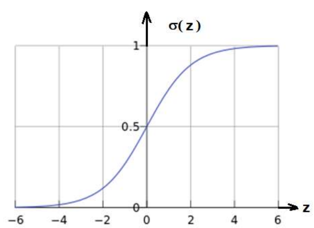
Standard activation functions

- 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 $z = 0$



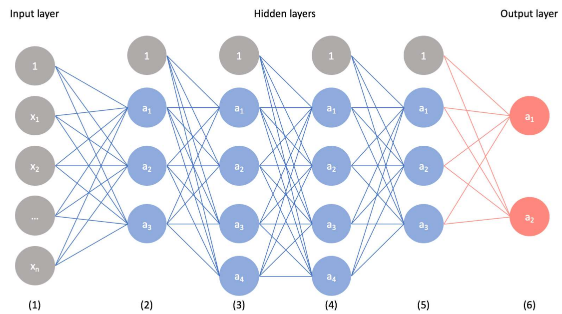
Derivative of sigmoid function

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

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

- 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,

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

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

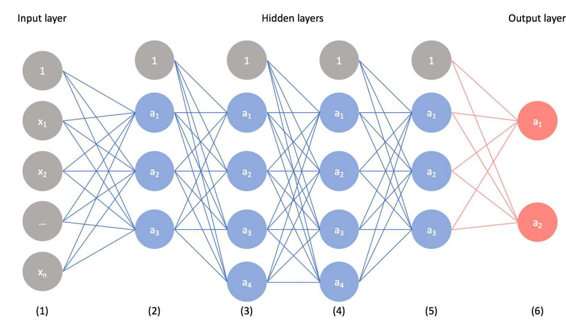
Vanishing gradient problem

- 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

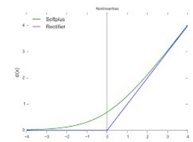


Exploding gradient problem

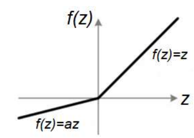
- Suppose vanishing gradient problem does not occur.
- Then $\frac{\partial E}{\partial w_{jk}^{(l)}} = \sum_{mn\ \ldots qr} \frac{\partial E}{\partial y_m^{(L)}} \frac{\partial y_m^{(L)}}{\partial y_n^{(L-1)}} \frac{\partial y_n^{(L-1)}}{\partial y_p^{(L-2)}} \cdots \frac{\partial y_q^{(l+1)}}{\partial y_r^{(l)}} \frac{\partial y_r^{(l)}}{\partial w_{jk}^{(l)}}$ 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 to layer(s) keeps changing.
  - The distribution of the output of each layer in different batches are different.
  - This reduces training efficiency.
- The input samples (in every batch) are normalized before feeding it into the next layer of 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?