Neural Network Note

Muhamad Ridzuan

March 9, 2024

1 Chapter 1

1.1 Donald Hebb

Organization of behaviour - 1949 learning mechanism:

- When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as on of the cells firing B, is increased.
- As A repeatedly excites B, its ability to excite B improves.
- Neuron that fire together wire together.

1.2 Hebbian Learning

- If neuron x repeatedly triggers neuron y, the synaptic knob connecting x to y get larger.
- In mathematical model:

$$w_{xy} = w_{xy} + \eta xy$$

- Weight of the connection from input neuron x to output neuron y.
- This simple formula is actually the basis of many learning algorithms in machine learning.

This idea however is fundamentally unstable:

- Stringer connections will enforce themselves.
- No notion of "competition".
- No *reduction* in weights.
- Learning is unbounded.p

People came up with all kinds of modifications for it to try to make it more stable:

- Allowing for weight normalization.
- Forgetting

This lead to the Generalized Hebbian learning, aka Sanger's rule where the contribution of input is *incrementally distributed* over multiple outputs.

$$w_{ij} = w_{ij} + \eta y_j \left(x_i - \sum_{k=1}^{j} w_{ik} y_k \right)$$

1.3 A better model

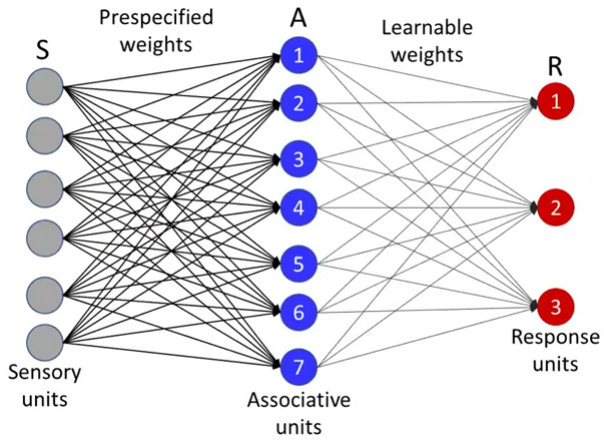
Frank Rosenblatt

- Psychologist, Logician
- Inventor of the solution to everything, aka Perceptron

Original perceptron model Consider the eye structure:

- Groups of sensors on retina combine into cells in association in the **projection area**.
- Groups of **projection area** combine into Association cells in **association area**.
- Signals from **association area** cells combine into response cell **R**.
- All connections may be excitatory or inhibitory.

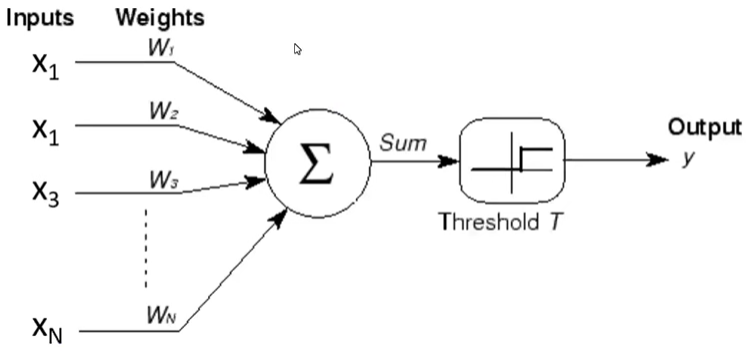
Rosenblatt's perception model can then be further simplified:



Simplified perceptron model

- Association units commbine sensory input with fixed weights.
- Response units combine associative units with learnable weights.

Each of the units can be perceived as shown below:



- Number of inputs combine linearly.
- Threshold logic is applied at the output of the unit: It will only fire if the weighted sum of the input exceeds threshold.

$$Y = \begin{cases} 1, & \text{if } \sum_i w_i x_i - T \geq 0 \\ 0, & \text{else} \end{cases}$$

1.4 The Universal Model

Originally assumed could represent any Boolean circuit and perform any logic. However this was not true and this cause the research in neural networks died down because of the overhype. However, Rosenblatt was right, he not only gave us basic model, he also gives us the learning algorithm.

$$\mathbf{w} = \mathbf{w} + \eta(d(\mathbf{x}) - y(\mathbf{x}))\mathbf{x}$$

Sequential learning:

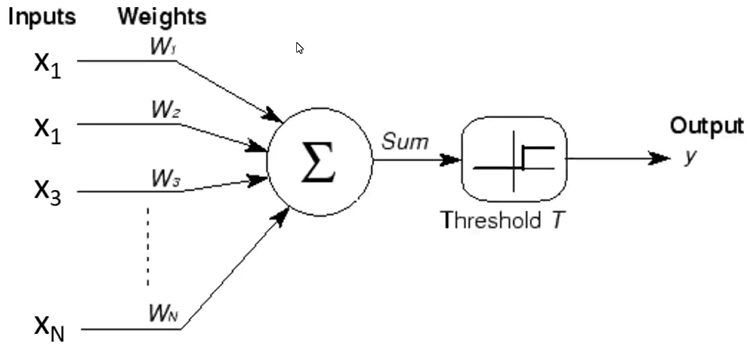
- $d(\mathbf{x})$ is the desired output in response to input $\mathbf{x}$.
- $y(\mathbf{x})$ is the actual output in response to $\mathbf{x}$.
- Weight parameter is changed when the actual response of the neuron does not match the desired response of the neuron.
- If the actual response of the neuron exceeds the desired response of the neeuron, then the weight will decrease.
- If the actual response is lesser than the desired response, the weight will increase.

Single perceptron is not enough While this model able to mimic boolean logic such as AND, OR and NOT, it is not capable to mimic XOR gate. This imply that individual elements are weak computational elements and networked elements are required. However, with multilayered perceptron, XOR gate can be constructed:

- Start with two inputs, A and B.
- Use AND gate to compute A and NOT B.
- Use another AND gate to compute NOT A AND B.
- Use an OR gate to combine the outputs of thee two AND gate.

This would lead to construction of perception network to represent a much more generic model:

- Perceptrons can be multi-layered.
- Can compose arbitrarily complicated boolean functions.
- In cognitive terms, it should be capable to compute arbitrary boolean functions over sensory input.



1.5 Difference between brain and neural networks

- Brains have real inputs.
- We are capable to make non-Boolean inferences and prediction.

We can however try to implement **real inputs** on the perceptron model:

- $x_1, x_2, ..., x_N$ are real-valued.
- $w_1, w_2, ..., w_N$ are real-valued.
- Unit is only fired if weighted sum of input matches or exceeds the threshold values T.

$$Y = \begin{cases} 1, & \text{if } \sum_i w_i x_i - T \geq 0 \\ 0, & \text{else} \end{cases} \tag{1}$$

This create a boundary condition such that $\sum w_i x_i = T$ where every sum of inputs that's greater than T will fire and everything that is less than T would not fire.

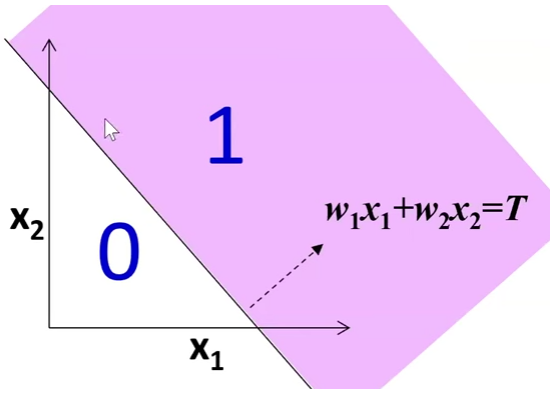
That being said, this provides an alternative view:

- The neuron has a threshold **activation function** $\theta(z)$ operates on the weighted sum of inputs plus a bias. (Affine function of the inputs).
- The boundary b, is the value such that the affine term is equal to zero.
- Linear function is an affine function that passed through origin.
- $\theta(z)$ outputs a 1 if z is non-negative 0 otherwise.
- Unit fires if weighted input matches or exceed threshold.

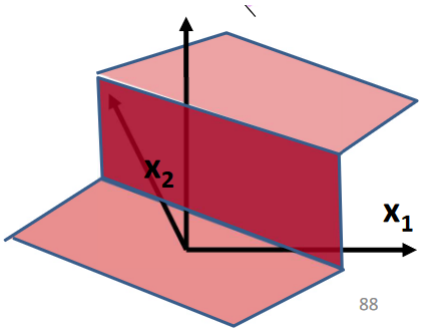
$$y = \theta\left(\sum_i w_i x_i + b\right)$$

$$\theta(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{else} \end{cases}$$

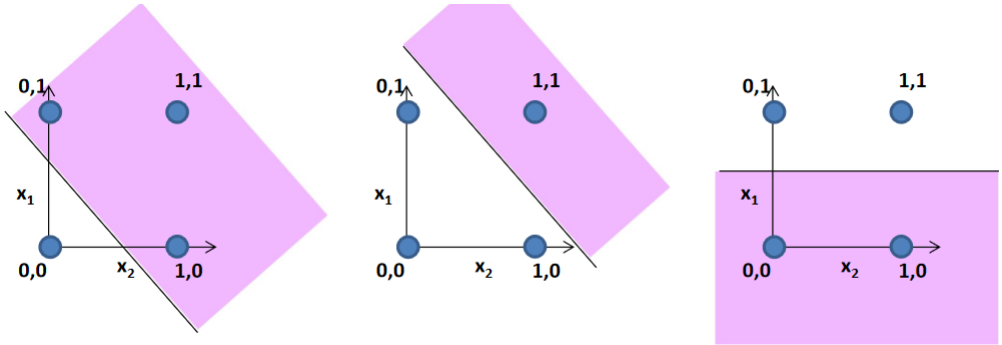
On real numbers, this equation produce a hyperplane such with line separation of $w_1x_1+w_2x_2+...+w_nx_n = T$ where one side of the plane is 1 and the other side of the plane is 0. This would means that a perceptron that operates on real-valued vectors is a type of linear classifier.



Similarly, for two-dimensional inputs, if we ever to plot the function, the function is going to looks like a step function. On one side, the output is 1 and on the other side, the output is 0.



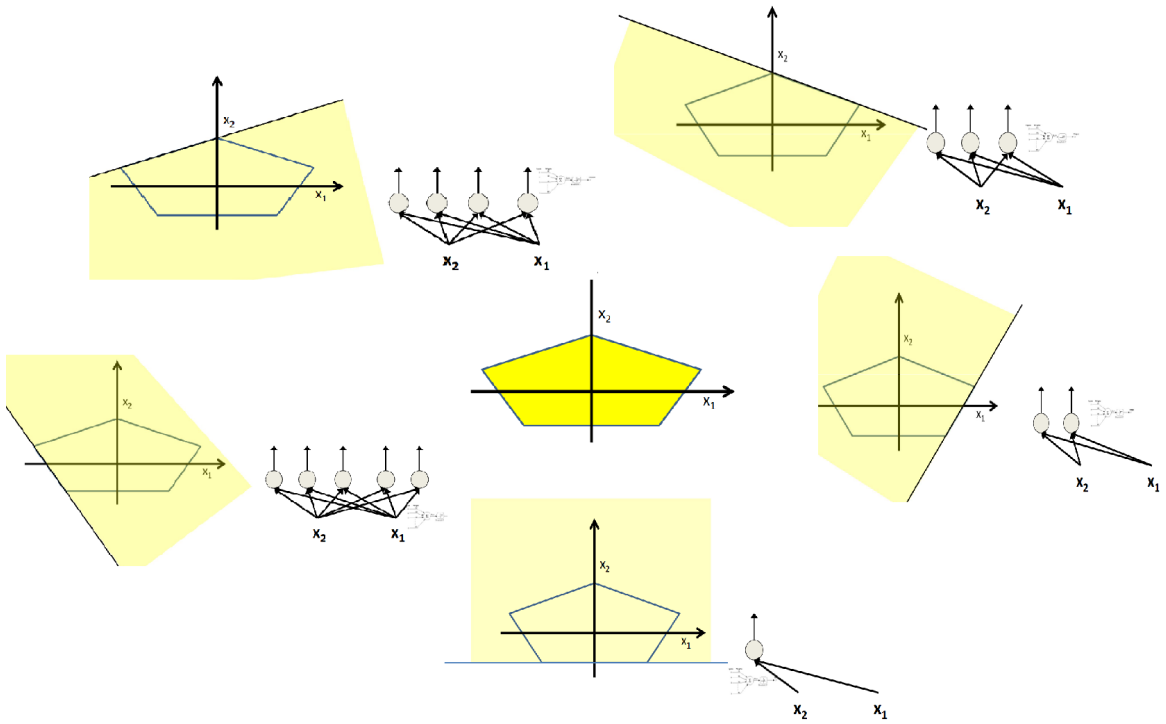
Once we can see how the perceptron model works, we can finally see how it performs Boolean's functions.



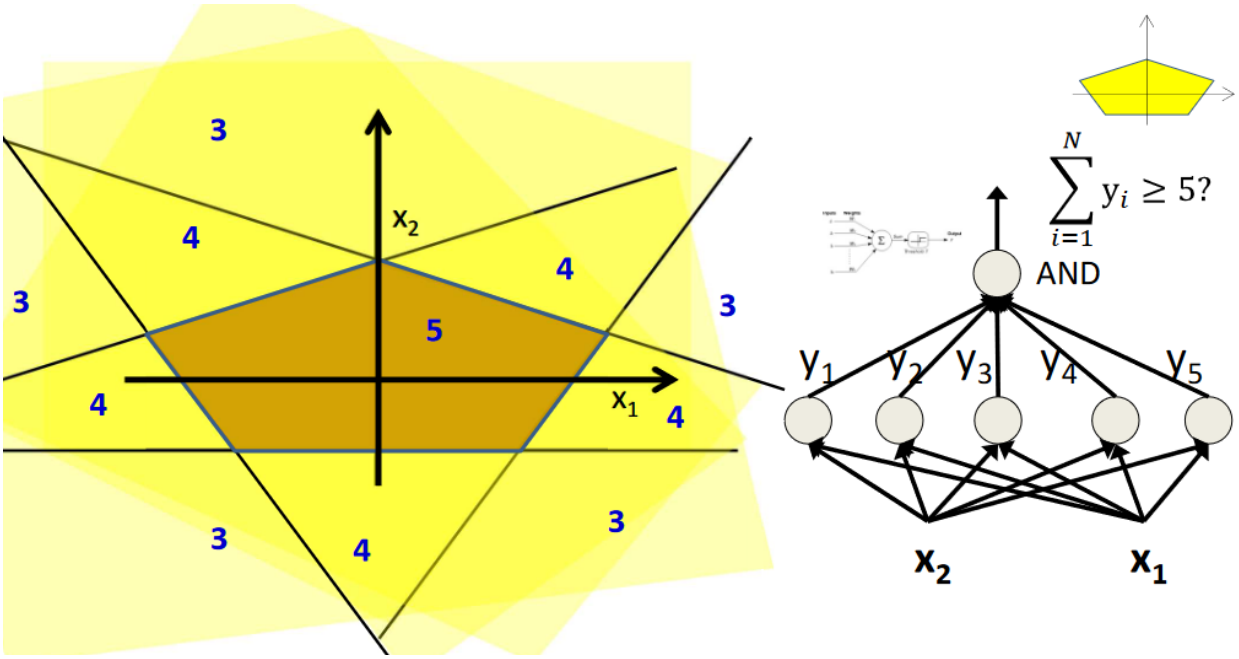
That being said, Boolean's perceptron is also linear classifiers:

- Purple regions have output of 1 in the figures.
- The left figure showcase OR-function.
- The middle figure showcase AND-function.
- The right figure showcase NOT-function.
- This is also the reason why a simple perceptron model is not enough to compose XOR-function.

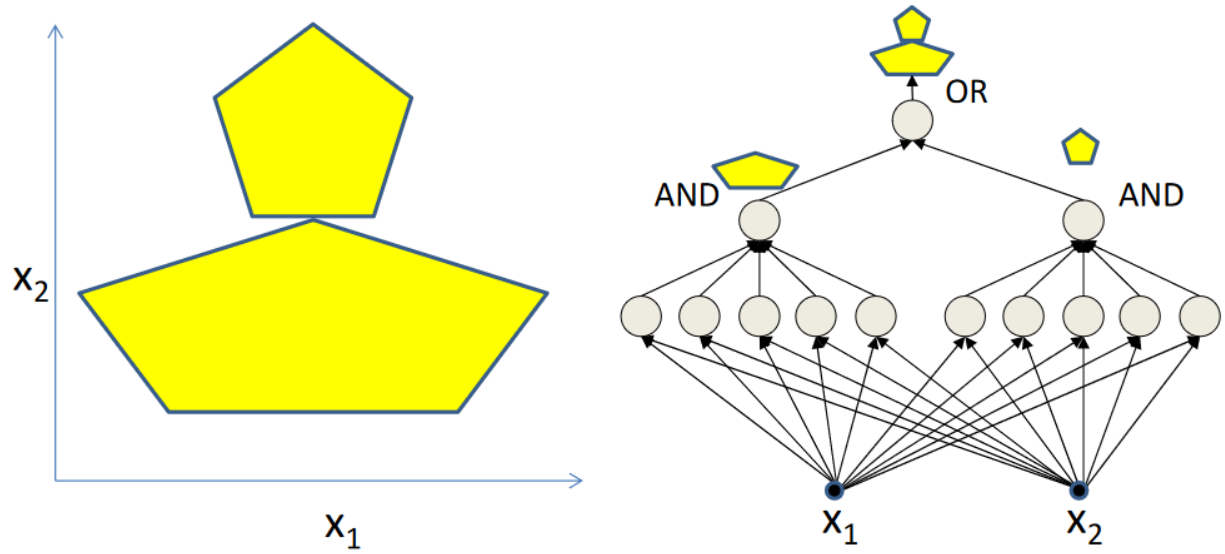
We can however, compose complicated decision boundaries by combining multiple linear perceptron by building a network of units with single output that firees if the input is in the desired regions. For example, we can use 5 perceptron model to compose 5 boundary condition of a pentagon. For each perceptron, the network will fire if the input is in the yellow area.



Given this perceptrons, we can AND them together capture the pentagon decision boundary.



This also imply that much more complex boundary condition can be solved as well.



The network will fire if the input is in the yellow area:

- "OR" two polygons
- This would means that a third layer of perceptron that behaves as "OR" gate is required.

As the multi-layered perceptron is capable to solve complex decision boundaries, it can be used to solve:

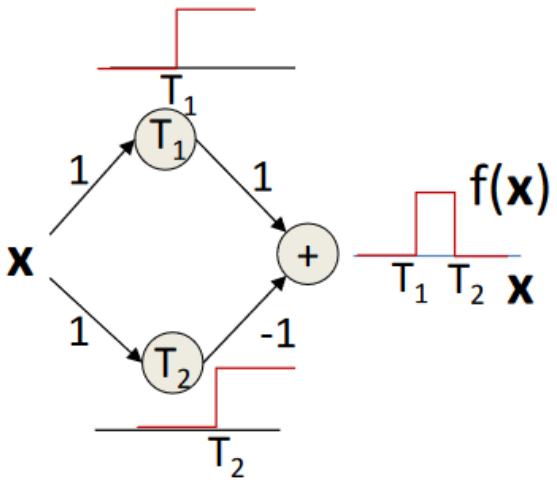
- Classification problems - finding decision boundaries in high-dimensional space.

- For example, the MNIST dataset every image as at least 784 dimensions boundary that need to be solved.
- MLP can be considered as universal classifiers as for any decision boundary, we can construct MLP that captures it with arbitrary precision.

1.6 Continuous Valued Outputs

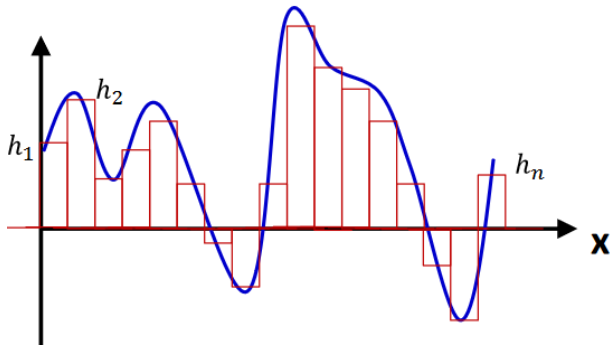
So far we have covered networks which have Boolean functions where the inputs were Boolean and the output was Boolean. We also discussed functions that took continuous valueed inputs that gave Boolean or categorical output. This section covers the possibility of the networks having real-valued inputs with continuous valued output.

Consider a simple three units MLP:



- The input is a single scalar $\mathbf{x}$.
- The first neuron fires, meaning that the output becomes 1 anytime the input $\mathbf{x}$ exceed the threshold value T_1.
- The second neuron also fires, meaning that the output becomes 1 anytime the input $\mathbf{x}$ exceed the threshold value T_2.
- Their outputs are summed with values 1 and -1 respectively.
- As $\mathbf{x}$ goes from large negative value to large positive value, assuming without loss of generality $T_1 < T_2$, eventually it will cross T_1 (At this point it has not yet crossed T_2).
- So at the first perceptron, the output is 1 while it is still 0 on the second perceptron.
- As we continue to increase $\mathbf{x}$, it will eventually crosses T_2 and the output of these 2 neurons will cancel out.
- The output of the overall circuit goes back to zero after passing T_2.
- This network will output 1 if the input is between T_1 and T_2, and zero elsewhere.

Suppose that we are given arbirary function as shown below:



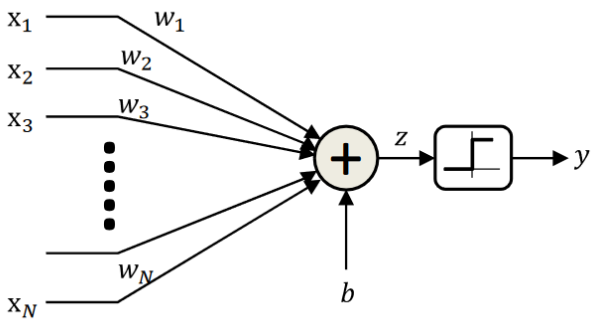
We can partition the input space into lots of little buckets (similar how ADC works), so that we can have one little subnet for each of the bucket. Then we can scale the outputs by the height of the function average within that range.

- An MLP with many units can model an arbirary function over an input.
- We can get an arbirary precision by making the individual buckets smaller.
- This generalizes to functions of any number of inputs.

1.7 Summary

- MLPs are connectionist computational models.
  - Individual perceptrons are computational equivalent of neurons.
  - The MLP is a layered composition of many perceptrons.
- MLPs can model any Booleans function.
  - Individual perceptrons can act as Boolean gates.
  - Networks of perceptrons are Boolean gates.
  - MLPs are universal Boolean functions.
- MLPs can models any decision boundary.
  - Individual perceptrons capture linear boundaries.
  - Complex boundaries can be composed from the linear boundaries.
  - MLPs can represent arbitrary decision boundaries.
  - They can be used to classify data.
  - MLPs are universal classifiers.

2 Chapter 2



Each individual neurons can be represented as a threshold unit.

- It fires if the affine function of inputs is positive.
- The bias value is the negative of threshold T.

$$z = \sum_i w_i x_i + b$$
$$y = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{else} \end{cases}$$

Once we can represent the neuron in this manner, we can modify the activation threshold function to something different such as:

- We define **activation function** as the function that acts on the weighted combination of inputs and bias.

Soft Perceptron (Logistic) A squashing function instead of a threshold at the output. This way the output goes rather smooth from 0 to 1.

$$z = \sum_i w_i x_i + b$$
$$y = \frac{1}{1 + \exp(-z)}$$

We can also replace the activation function with other mathematical function such as tanh, softplus and rectifier.

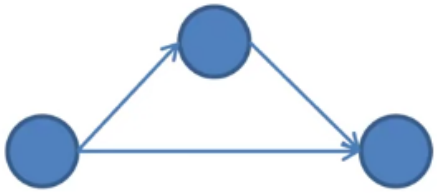
2.1 Multi-layer Perceptron

MLP is a network of perceptrons as the perceptrons of current layer are fed to other perceptrons on the next layer.

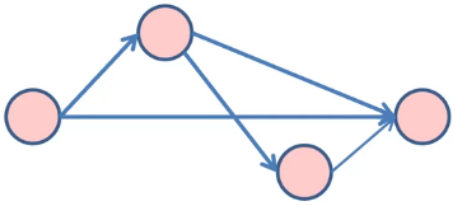
Deep Structures In any directed graph with input source nodes and output sink nodes, "depth" is the length of the longest path from a source to a sink.

- A "source" node in a directed graph is a node that has only outgoing edges.
- A "sink" node is a node that has only incoming edges.

This is an example of depth 2 graph.



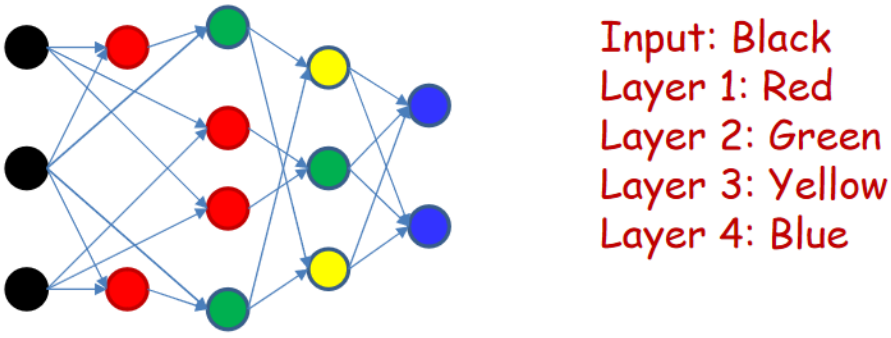
This is an example of depth 3 graph.



In multilayer perceptron, a network is considered "deep" if the depth of the output neurons is greater than 2.

2.2 Layer

Layer is a set of neurons that are all at the same depth with respect to the input (sink). This would imply that the "depth" of the layer is the depth of the neurons in the layer with respect to the input.



In multi-layered perceptron:

- Inputs are real or Boolean stimuli.
- Outputs are real or Boolean values.
- It can compose both Boolean and real-valued functions.
- We can have multiple outputs for a single input.