Neural Network Note

Muhamad Ridzuan

March 24, 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

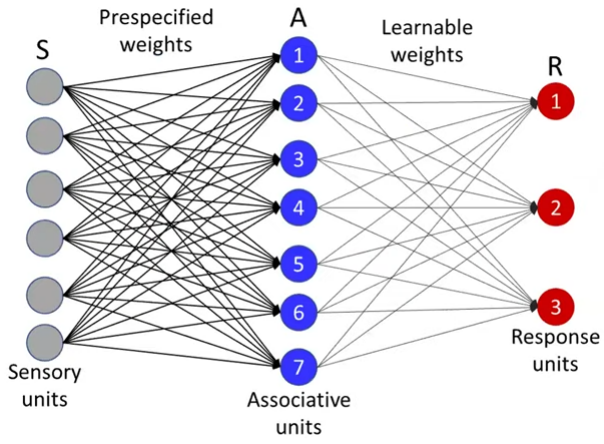
Frank Rosenblatt

- 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.

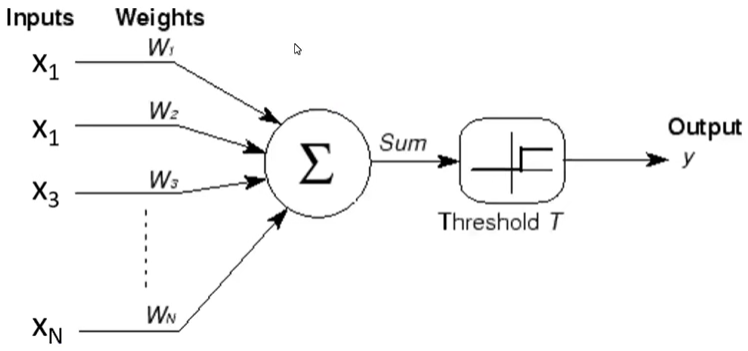
Rosenblatt's perception model can then be further simplified:



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:

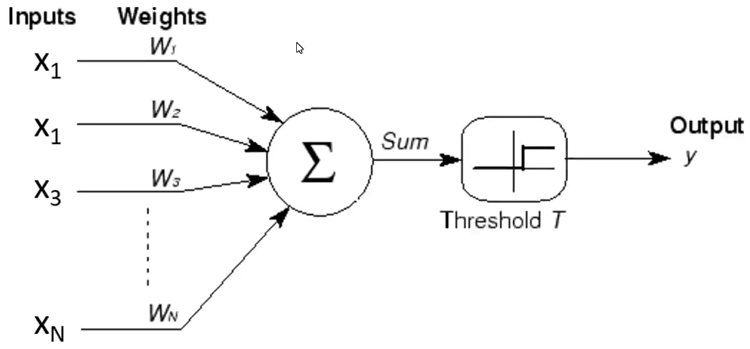
- $d(\mathbf{x})$ is the desired output in response to input $\mathbf{x}$.
- $y(\mathbf{x})$ is the actual output in response to $\mathbf{x}$.
- Weight parameter is changed when the actual response of the neuron does not match the desired response of the neuron.
- If the actual response of the neuron 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.

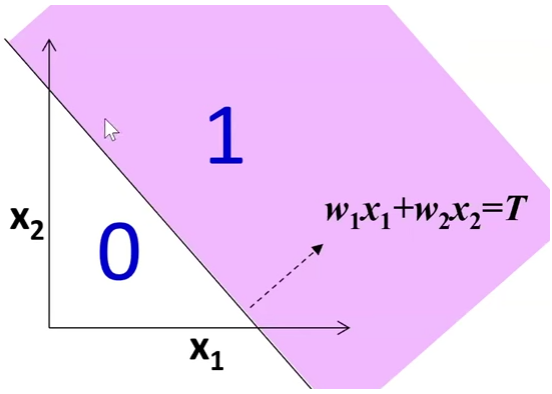
That being said, this provides an alternative view:

- 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.

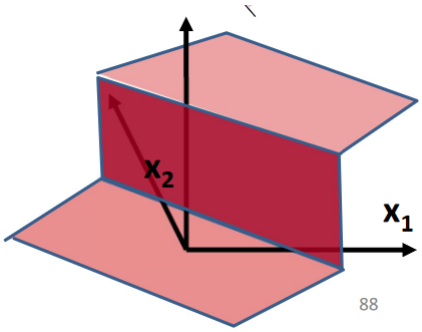
$$y = \theta\left(\sum_i w_i x_i + b\right)$$

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

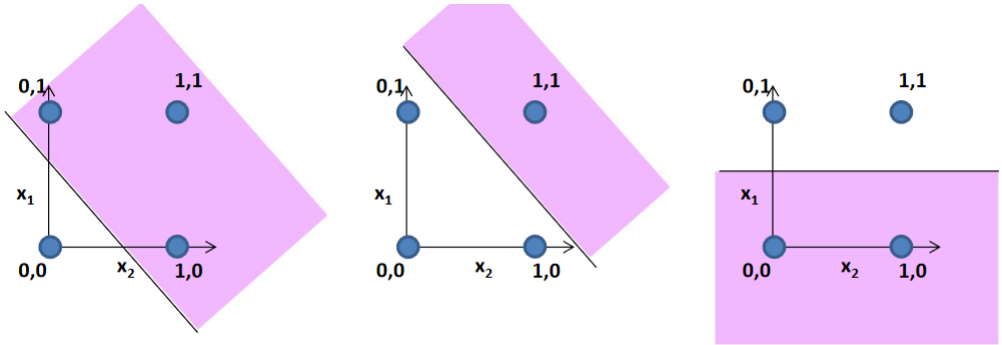
On real numbers, this equation produce a hyperplane such with line separation of $w_1x_1+w_2x_2+...+w_nx_n = T$ where one side of the plane is 1 and the other side of the plane is 0. This would means that a perceptron that operates on real-valued vectors is a type of linear classifier.



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.

1.6 Perceptron as basic logic gates.

AND Gate From our knowledge of logic fates, we know that an AND logic table is given by the diagram below.

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

In order to interpret weights and bias to construct AND perceptron, we first need to understand that the output of an AND gate is 1 only if both inputs (in this case x_1 and x_2) are 1. Following the steps listed above:

From

$$w_1 \times x_1 + w_2 \times x_2 + b \tag{2}$$

Initializing w_1, w_2, as 1 and b as -1, we get:

$$x_1(1) + x_2(1) - 1 \tag{3}$$

Passing the first row of the AND logic table, $(x_1 = 0, x_2 = 0)$, we get:

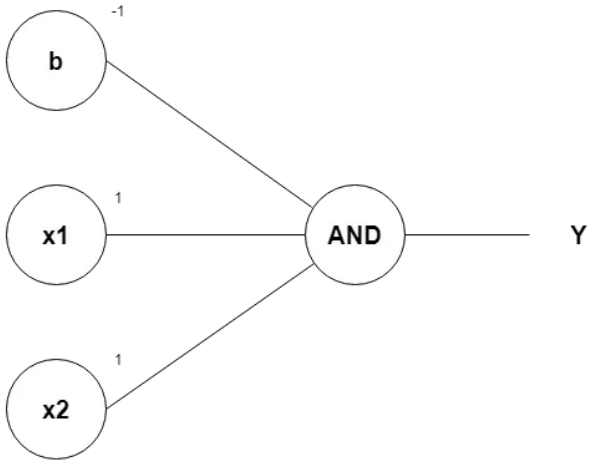
$$0 + 0 - 1 = -1 \tag{4}$$

From the perceptron rule, if $\mathbf{w}x + b \leq 0$, then $y = 0$, therefore this row is correct. Repeating the same process for each of the row:

$$\begin{aligned} 0 + 0 - 1 &= -1 \\ 0 + 1 - 1 &= 0 \\ 1 + 0 - 1 &= 0 \\ 1 + 1 - 1 &= 1 \end{aligned}$$

Therefore, we can conclude that in order for the perceptron to be able to model AND gate, the perceptron algorithm is:

$$x_1 + x_2 - 1 \tag{5}$$



OR Gate

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

From

$$w_1 \times x_1 + w_2 \times x_2 + b \tag{6}$$

Initializing w_1, w_2, as 2 and b as -1, we get:

$$x_1(2) + x_2(2) - 1 \tag{7}$$

Passing the first row of the OR logic table, $(x_1 = 0, x_2 = 0)$, we get:

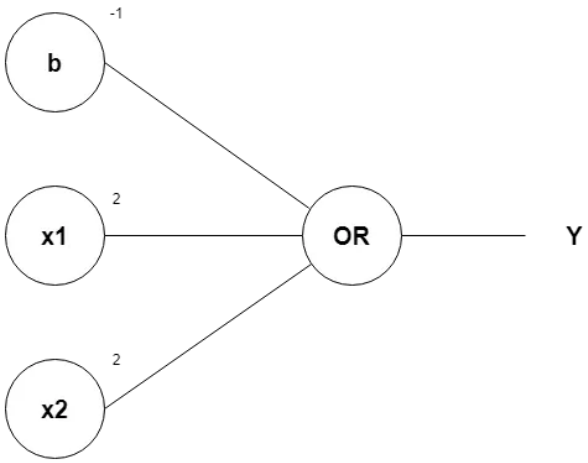
$$0 + 0 - 1 = -1 \tag{8}$$

From the perceptron rule, if $\mathbf{w}x + b \leq 0$, then $y = 0$, therefore this row is correct. Repeating the same process for each of the row:

$$0 + 0 - 1 = -1$$
$$0 + 2 - 1 = 1$$
$$1 + 2 - 1 = 2$$
$$2 + 2 - 1 = 3$$

Therefore, we can conclude that in order for the perceptron to be able to model OR gate, the perceptron algorithm is:

$$2x_1 + 2x_2 - 1 \tag{9}$$



NOT Gate

w	y
0	1
1	0

From the diagram, the output of NOT gate is the inverse of the input. Since w_2 does not exists, the remaining equation would be:

$$w_1 \times x_1 + b \tag{10}$$

Initializing w_1, as -1 and b as 1, we get:

$$x_1(-1) + 1 \tag{11}$$

Passing the first row of the NOT logic table, $(x_1 = 0)$, we get:

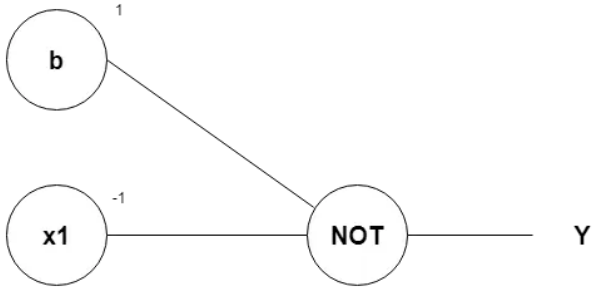
$$0 + 1 = 1 \tag{12}$$

Passing the second row of the NOT logic table, $(x_1 = 1)$, we get:

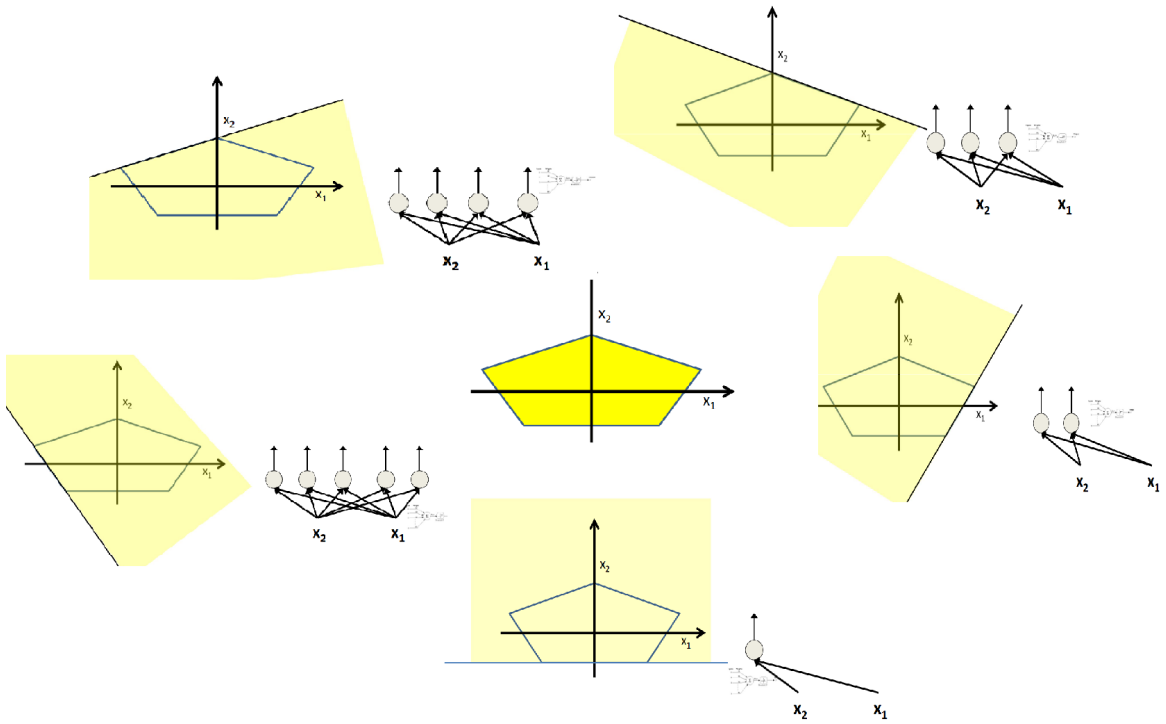
$$-1 + 1 = 0 \tag{13}$$

Therefore, we can conclude that in order for the perceptron to be able to model NOT gate, the perceptron algorithm is:

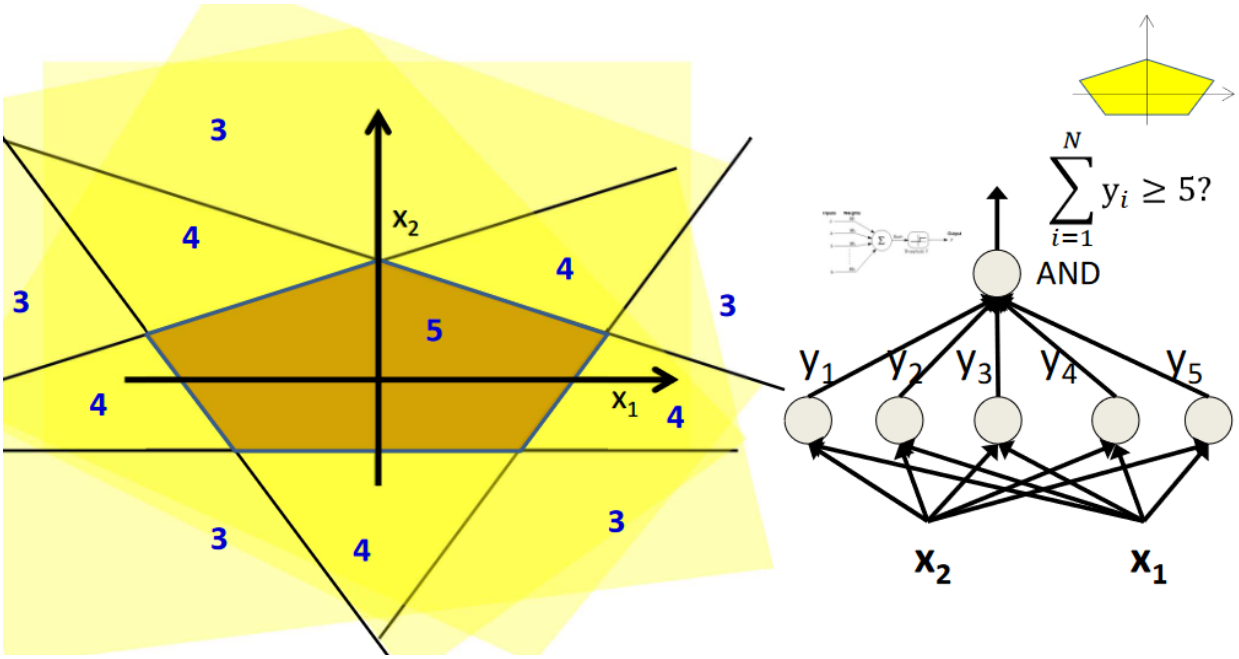
$$-x_1 + 1 \tag{14}$$



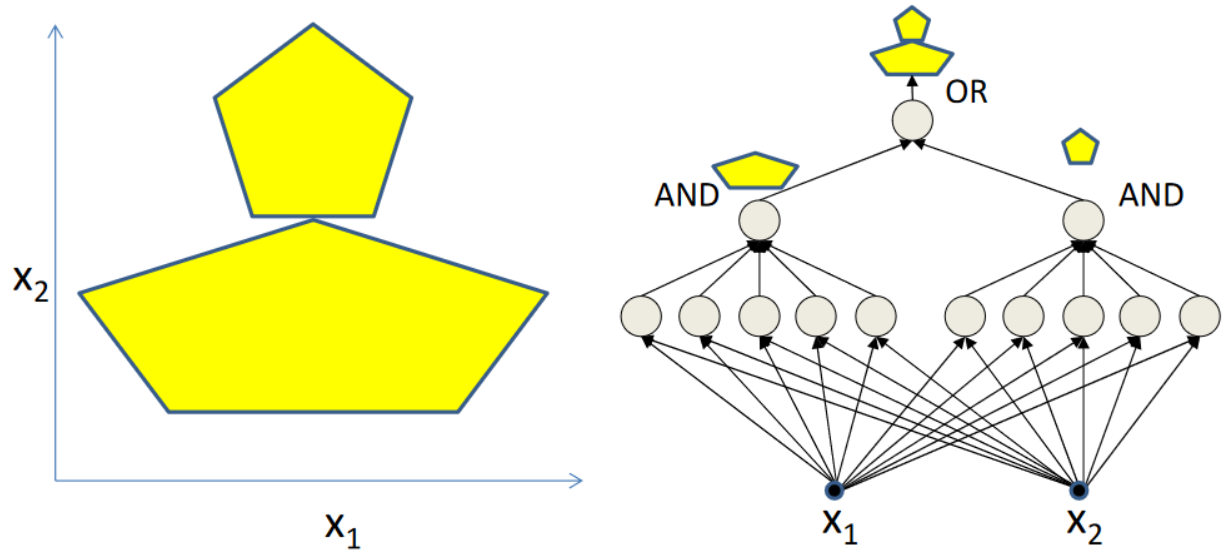
We are able compose complicated decision boundaries by combining multiple linear perceptron by building a network of units with single output that firees if the input is in the desired regions. For example, we can use 5 perceptron model to compose 5 boundary condition of a pentagon. For each perceptron,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

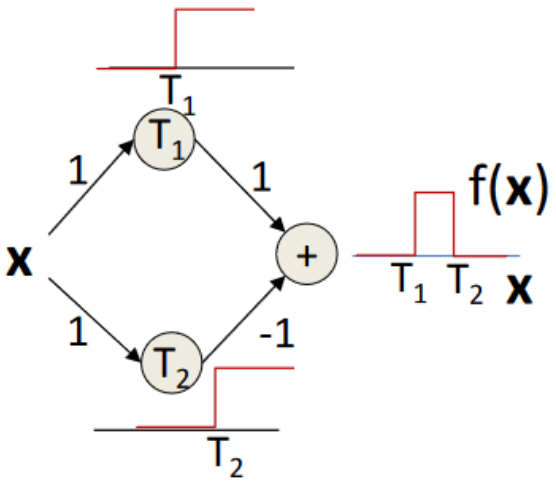
- Classification problems - finding decision boundaries in high-dimensional space.

- 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.

1.7 Continuous Valued Outputs

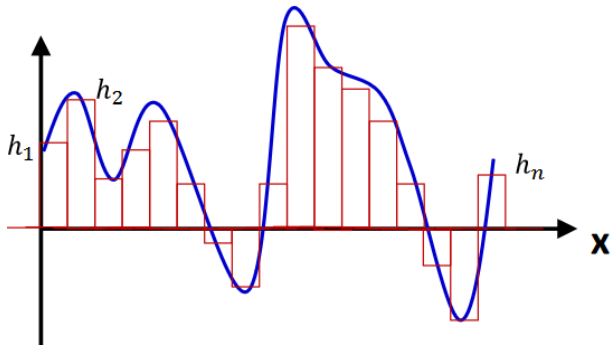
So far we have covered networks which have Boolean functions where the inputs were Boolean and the output was Boolean. We also discussed functions that took continuous valueed inputs that gave Boolean or categorical output. This section covers the possibility of the networks having real-valued inputs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



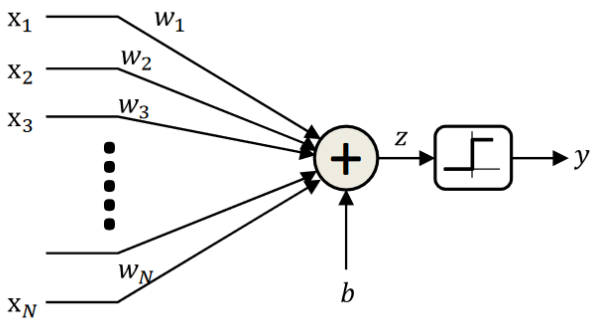
We can partition the input space into lots of little buckets (similar how ADC works), so that we can have one little subnet for each of the bucket. Then we can scale the outputs by the height of the function average within that range.

- 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.

1.8 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.

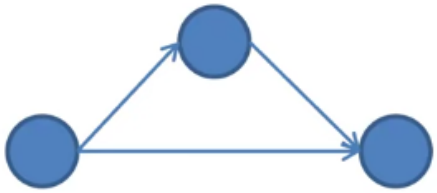
2.1 Multi-layer Perceptron

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

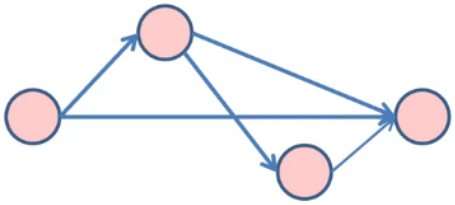
Deep Structures In any directed graph with input source nodes and output sink nodes, "depth" is the length of the longest path from a source to a sink.

- 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.



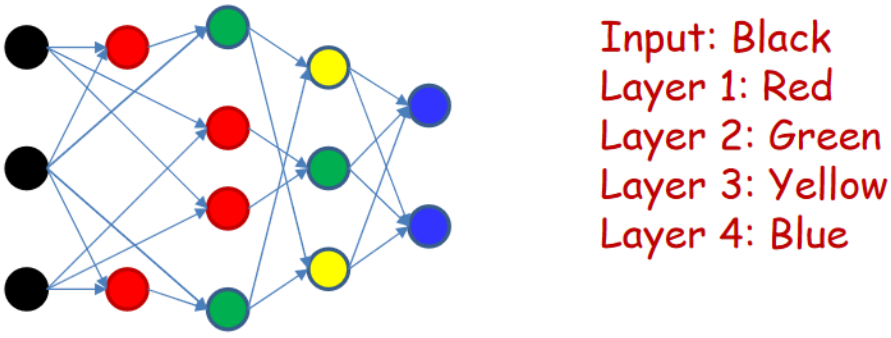
This is an example of depth 3 graph.



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.

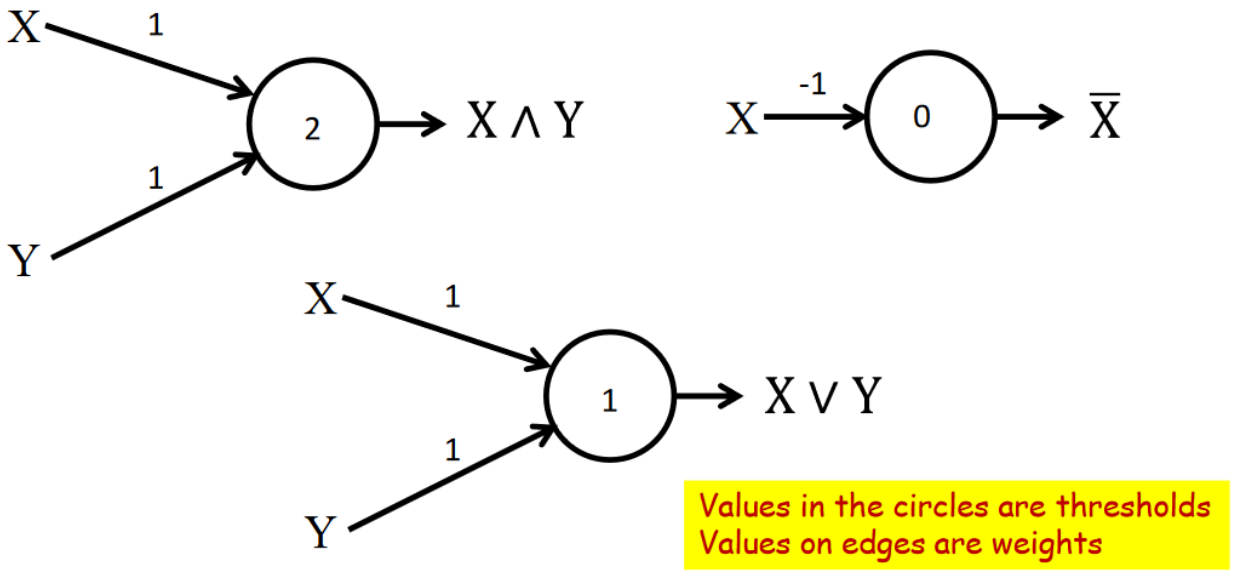


In multi-layered perceptron:

- 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.

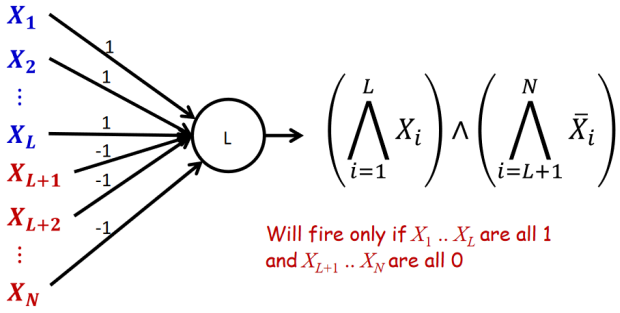
2.3 MLP as universal Boolean functions.

We are already aware with perceptron capability as a Boolean gate as it can model any simple binary Boolean gate (OR, AND & NOT).



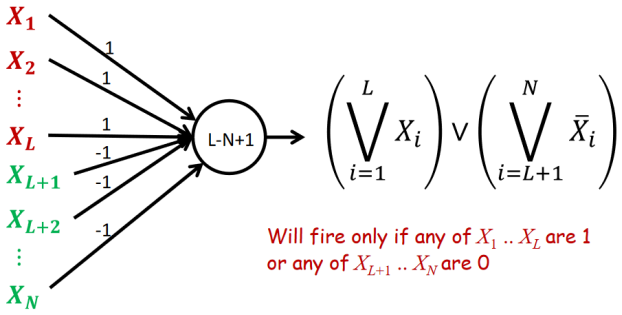
- For the AND gate, the boundary condition is $X + Y \geq 2$
- For the OR gate, the boundary condition is $X + Y \geq 1$
- For the NOT gate, the boundary condition is $-X \geq 0$

It also enables us to compose a much more complex network, for example a universal AND gate where the network is only fired if every input fed to the network is true.



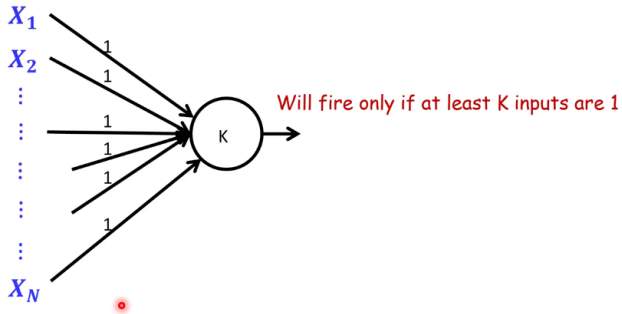
Universal AND Gate Suppose that $X_1, X_2, ..., X_L$ has weight value of 1 and $X_{L+1}, X_{L+2}, ..., X_N$ has weight value of -1, the condition for the perceptron to fire has to be:

$$(X_1 + X_2 + ... + X_L) - (X_{L+1}, X_{L+2}, ..., X_N) \geq L \tag{15}$$



Universal OR Gate Suppose that $X_1, X_2, ..., X_L$ has weight value of 1 and $X_{L+1}, X_{L+2}, ..., X_N$ has weight value of -1, the condition for the perceptron to fire has to be if any value of the first set is 1 or any value from the second set is 0:

$$(X_1 + X_2 + ... + X_L) - (X_{L+1}, X_{L+2}, ..., X_N) \geq L - N + 1 \tag{16}$$



Generalized Majority Gate

Due to perceptrons being able to model Boolean functions, they able to model complex things that are made of Boolean functions. This also means that for any odd Boolean function, wee can construct a multi-layerr perceptron that can compute this Boolean function. This is what is meant by MLPs being a universal Boolean functions. For any Boolean functions, regardless of their complexity, we can compose an MLP that will compute said function. This lead to a new question, how many layers will they need to be sufficient at modeling said Boolean function? What is the smallest number of layers will be required?

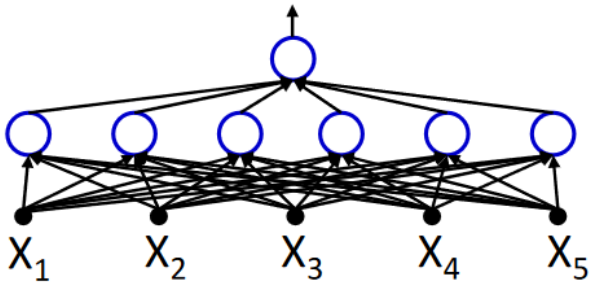
Any Booleans function can be represented by a truth table. For example:

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

In order to get a Boolean function that represent this table, we only have to list the portion of the table that corresponds to the output 1. We can write it as disjunctive normal formula for this table.

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + \tag{17}$$
$$X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5 \tag{18}$$

Once we have obtained the Boolean function in this manner, we can create an MLP where each of the perceptron is an Universal AND gate. These perceptrons is then combined as Universal OR gate.



This would means that MLP with 1 layer is sufficient enough to model any Boolean Function. This lead to a new question, what is the largest number of perceptrons required in the single hidden layer for an N-input-variable function?

2.4 Karnaugh Map

Boolean function can be reduced using Karnaugh Map. It represent a truth table as a grid. Adjacent boxes can be "grouped" to reduce the DNF formula for the table.

WX \ YZ	00	01	11	10
00	1	0	0	1
01	1	1	0	0
11	1	0	0	0
10	1	0	0	1

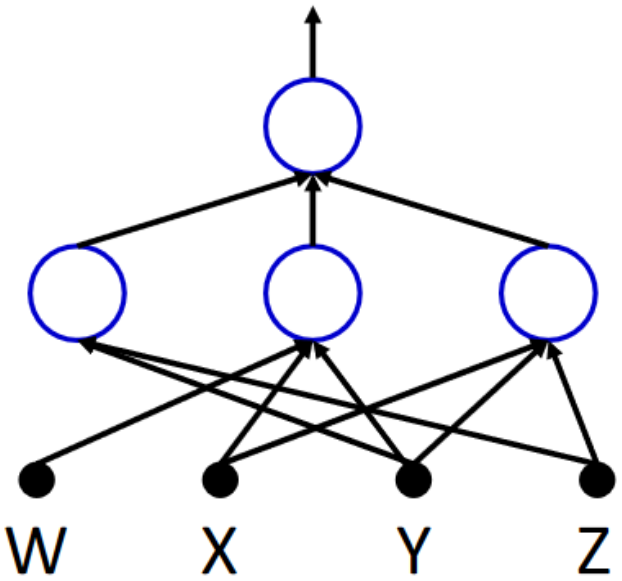
We can group these 3 neighbours together:

$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z} \tag{19}$$

- Regardless value of W and X, Y and Z is always 0.
- The function going to fire when W is always 0 while X can be anything, while Y has to always be 0.

- The other condition for it to fire is when Y is always 1 while Z is always 0 and X is always 0.

Although there are 8 conditions when the network will fire, only 3 perceptions is needed to construct the MLP. The general idea is, when we want to construct the DNF formula, we want to construct it with thee minimum number of clauses.



On irreducible DNF, the size of required perceptrons in the hidden layer can get exponentially large (2^{n-1}).

WX \ YZ	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

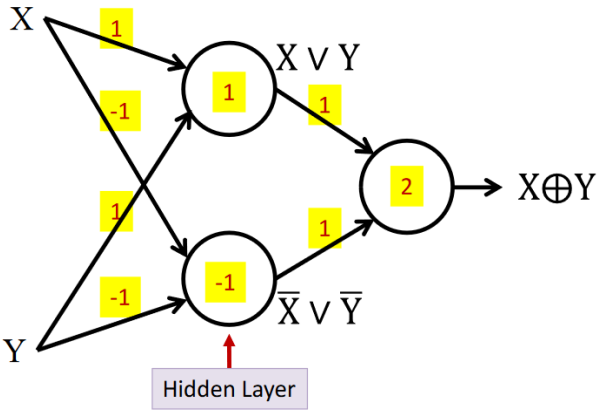
While the DNF cannot be reduced, this type of Boolean function can still be generalized. Is it possible to reduce the number of units if we use multiple hidden layers?

This checkered pattern is a form of XOR Boolean function which become TRUE only if:

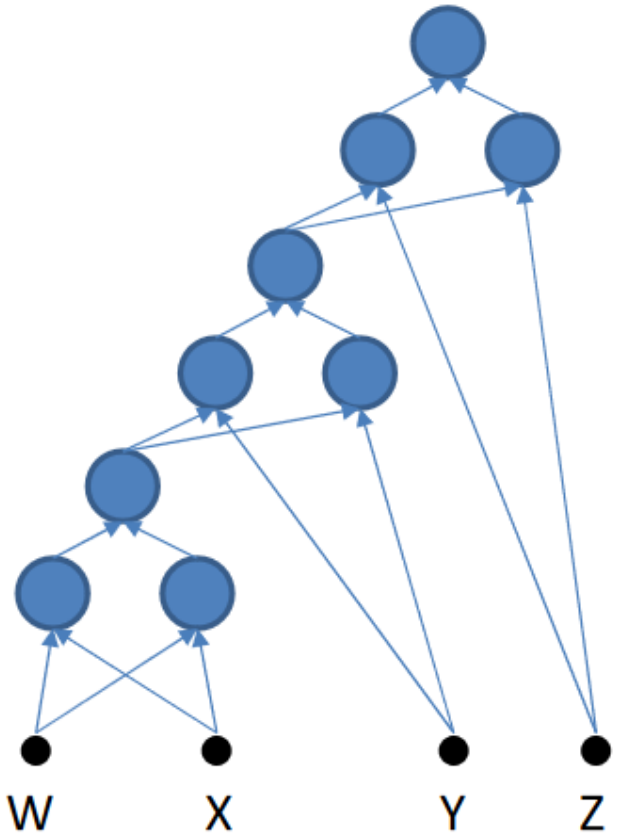
- 1 flag is FALSE while the rest of the flags is TRUE.
- 1 flag is TRUE while the rest of the flags is FALSE.

$$O = W \oplus X \oplus Y \oplus Z \tag{20}$$

Since we need 1 hidden layer consisting of 2 perceptrons to construct XOR gate:



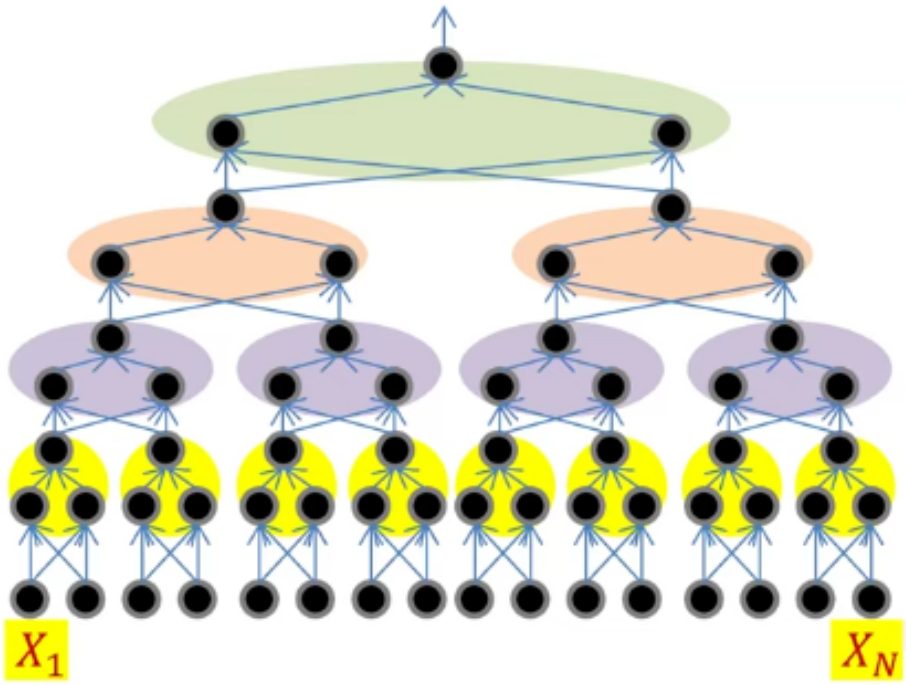
We can cascade the XOR gates to construct the solution for the MLP.



For cascading XOR gates with 4 parameter, 9 perceptrons would be required to construct the Boolean function MLP. The solution of the cascading XOR gates would be $3 \times (N-1)$ number of perceptrons.

These can be arranged in only $2\log_2(N)$ layers. Suppose that wee have $X_1, X_2, ..., X_N$ number of perception. We can keep pairing terms such as:

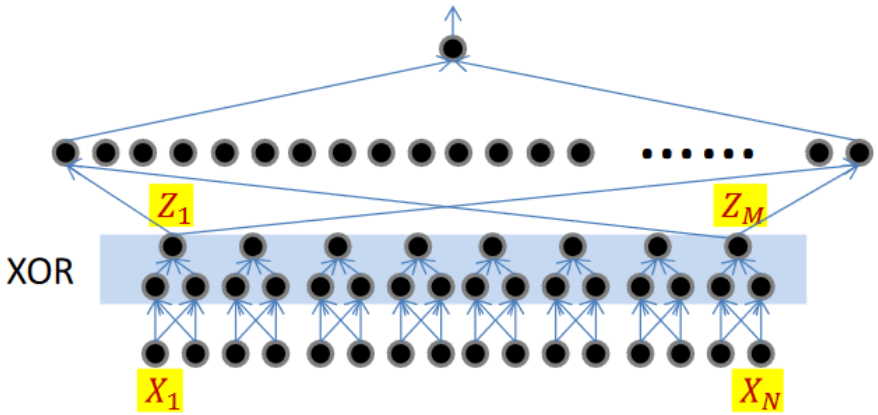
$$O = X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5 \oplus X_6 \oplus X_7 \oplus X_8 \tag{21}$$
$$O = ((X_1 \oplus X_2) \oplus (X_3 \oplus X_4)) \oplus ((X_5 \oplus X_6) \oplus (X_7 \oplus X_8)) \tag{22}$$



2.5 Implementation of MLP on K-layer.

Using only K-hidden layers will require 2^{CN} neurons in the K-th layer, where

$$C = 2^{-\frac{K-1}{2}} \tag{23}$$



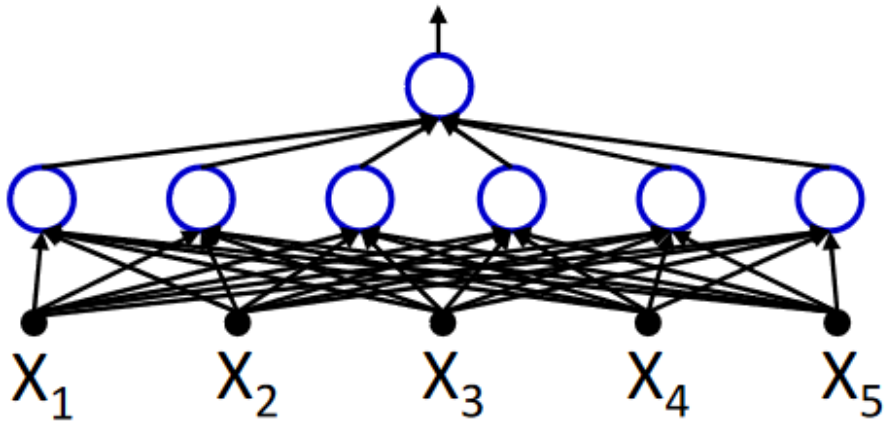
Going through the layer until the K-th layer, we keep reducing the number of neutrons by factor of 2.

However, we still have to XOR whatever number of neutrons left on the K-th layer, exponentially increasing the number of neutrons.

- Because the output is the XOR for all the $\frac{N}{2^{\frac{K-1}{2}}}$ values output by the K-1-th hidden layer.
- In other word, reducing the numbers of layers below the minimum will result in an exponentially sized network to express the function fully.
- A network with fewer than minimum required number of neutrons cannot model the function.

2.6 Network Parameters.

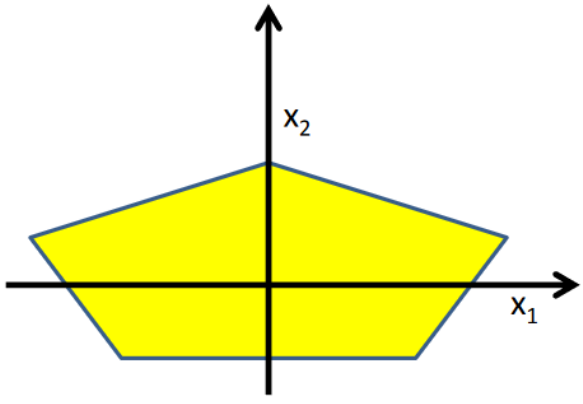
The actual number of parameters in a network is the number of connections. For example, for a network with 1 hidden layer consisting of 5 neutrons, the number of parameters in a network is 30.



This is the number that really matters in software or hardware implementation of the neural networks. Networks that require an exponential number of neurons will require an exponential number of weights.

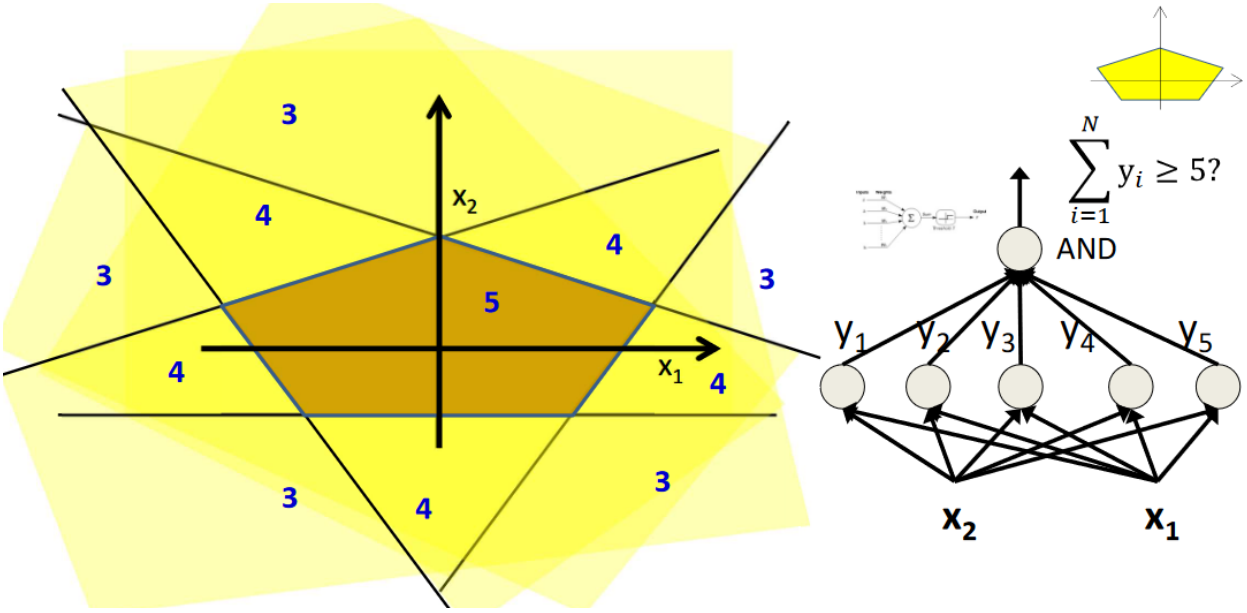
2.7 Composing complicated "decision" boundary.

Once we can computes a linear boundary, we can compose more complex decision boundary as shown below.

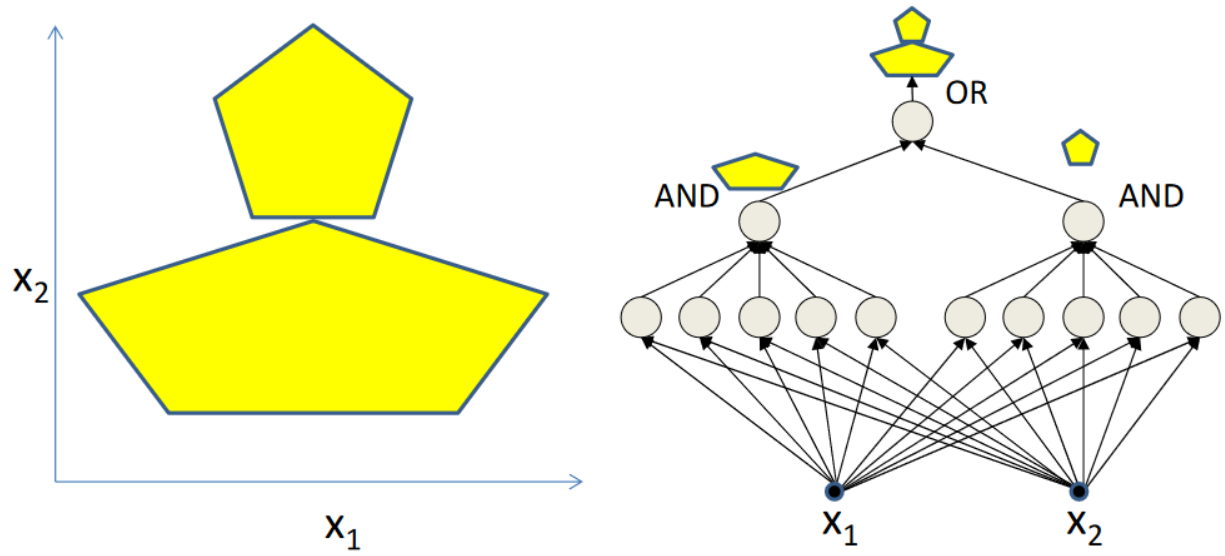


Recall that should we have pentagonal boundary, that would means that we have boundaries for each sides of the pentagon and the network will fire if the input value exceeds the boundary threshold value. One thing

to notice that, only within the pentagon are all five perceptrons going to output 1. This would imply that the network will fire if **sum of all of the output** exceeds the **sum of the threshold boundary value**.



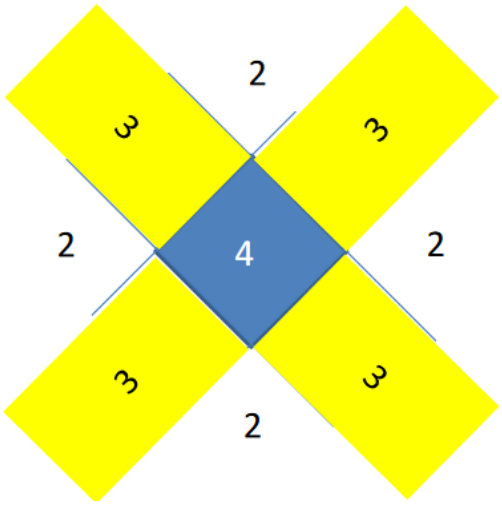
And if there is more than one closed boundaries, we have to OR them together to get the complex decision boundary. This would means that a third layer would be required.

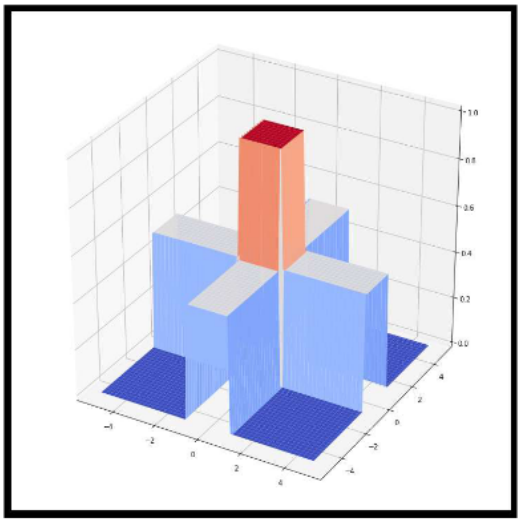


The question is, is it possible to compose these decision boundaries with only one hidden layer?

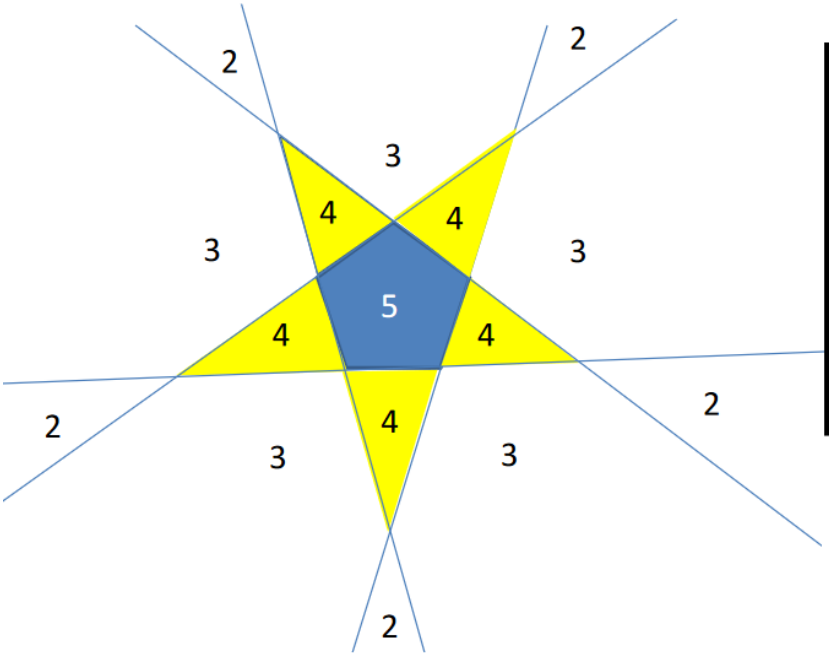
Case: Square boundary

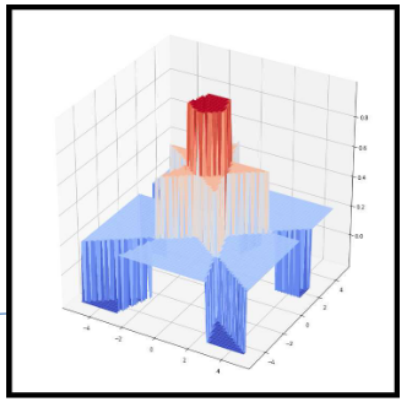
The sum of the output of the inner boundary is 4. Instead of having output layer of AND-ing all of the output of the previous layer perceptron, we just set the threshold value to be the sum of the threshold value of the individual perceptron. The last output neuron is performing SUM instead of AND.



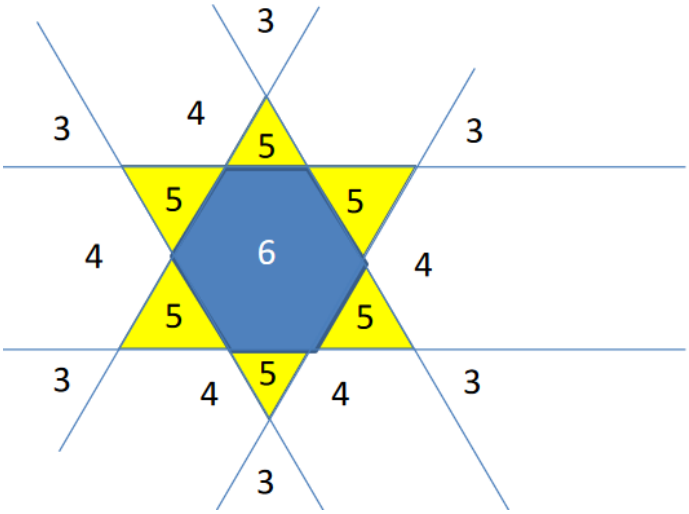


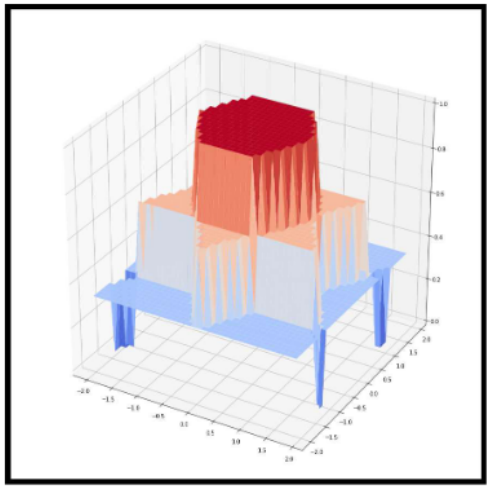
Case: Pentagon boundary



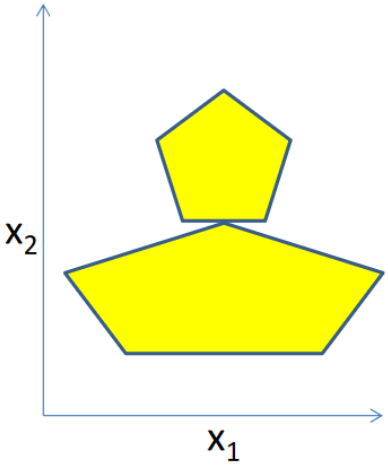


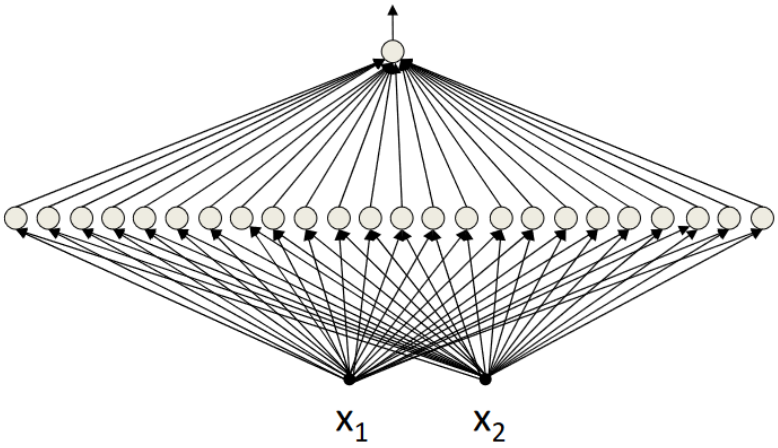
Case: Hexagon boundary





What if we want to compose composite decision boundary with single hidden layer?





If we could just somehow finagle these boundaries, it would not be possible because some of the sides of the lower pentagon will intercept with some of the edges of the upper pentagon. So any threshold that can capture lower pentagon will accidentally capture the upper pentagon region that is outside of the region of the lower pentagon.

Consider a square polygon. We would need a network with four neuron in a hidden layer to be able to describe the decision boundary. The threshold value on the side area of the polygon would be 3 and the rest of the area will have threshold value of 2. The same principle will apply to higher dimension polygon where the threshold value would be s, $s-1$, $s-2$ ... .

We can observe that the first outer layer that is adjacent with the decision boundary would have $s-1$ threshold value. Another observation that could be noticed is that the sum of the area that is outside the decision boundary will always bigger than the sum inside poligon.

Case: Pentagon

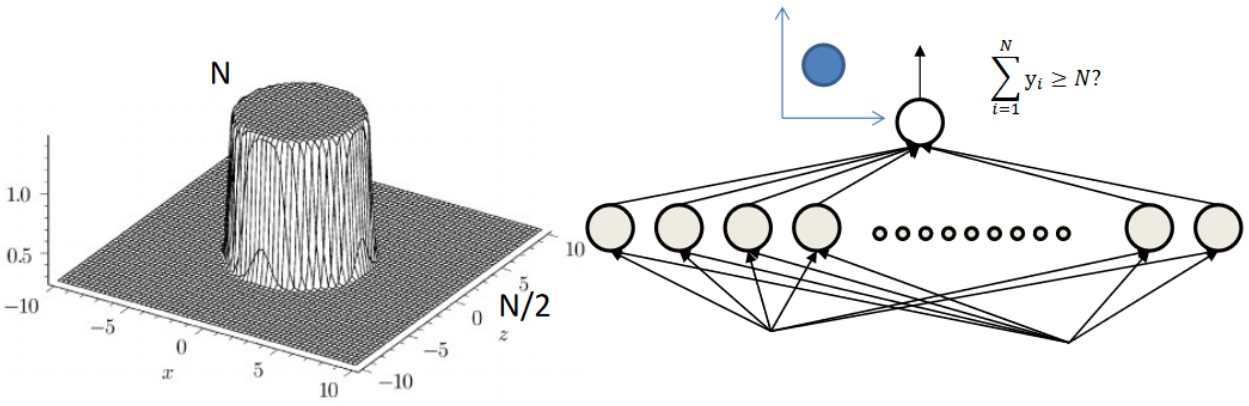
$$
\begin{aligned}
\text{Inner threshold value} &= 5 \\
\text{Sum of the threshold value of the first outer layer} &= 4 \times 5 \\
&= 20 \\
\text{Sum of the threshold value of the second outer layer} &= 3 \times 5 \\
&= 15 \\
\text{Sum of the threshold value of the last outer layer} &= 2 \times 5 \\
&= 10
\end{aligned}
$$

Another observation that could be made is that as the number of the decision boundary increases, the size of the first outer layer will be decreasing. Looking back at the possibility of creating single layer MLP for composite boundary, say if we want to compose a single hidden layer MLP for these outer regions, it would be possible only if the region is really small and does not overlap with the first outer region of another boundary. That being said, constructing MLP for composite boundaries would be possible if these regions are so small that it would not overlap with each other.

Since for any polygons, the size of the first outer layer will be smaller as the number of sides / decision boundary increases, this would means a circle would have the smallest area of first outer layer since the number of edges $\to \infty$. Increasing the number of sides will reduces the area outside the polygon that have $\frac{N}{2} < \sum_i y_i < N$.

Hypothetically speaking, this would means for a single layer MLP for decision boundary of circle:

- The number of neurons will become infinitely large.
- Sum of N inside the circle, $\frac{N}{2}$ outside almost everywhere.

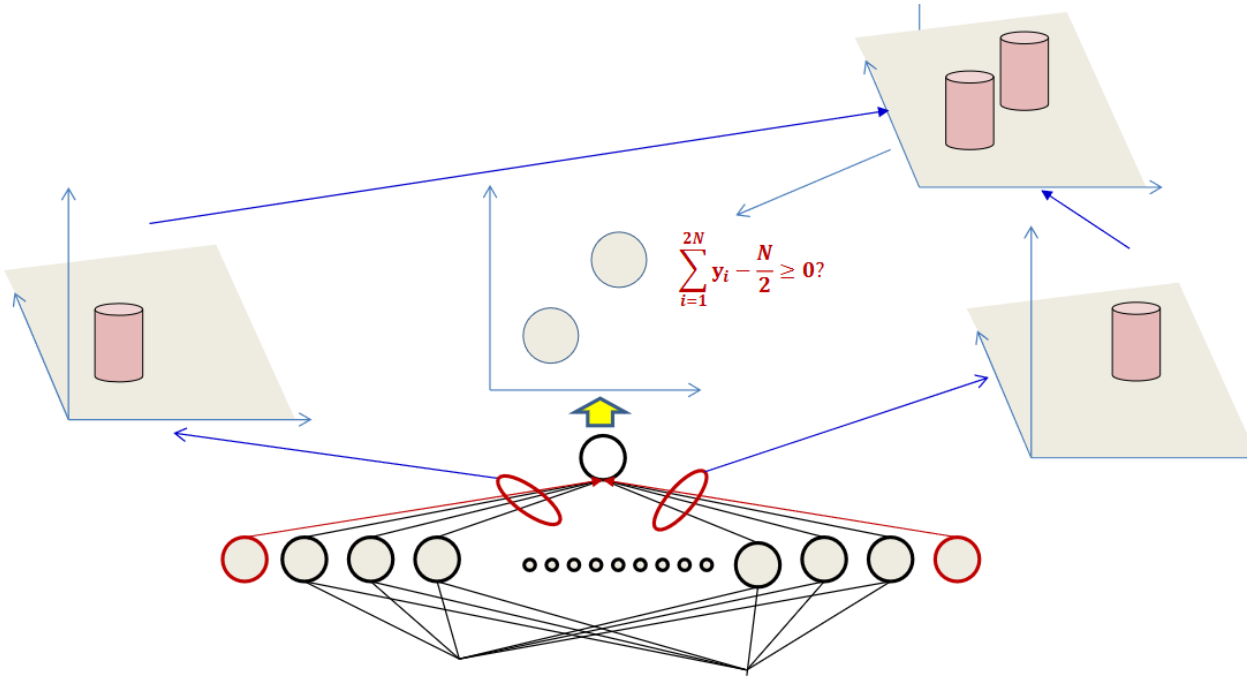


Based on the MLP constructed, if we sum all of these outputs and use a threshold:

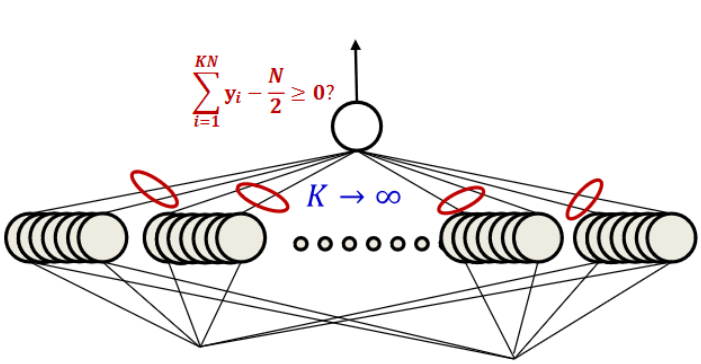
$$\sum_{i=1}^{N} \mathbf{y}_i - \frac{N}{2} \geq 0 \tag{24}$$

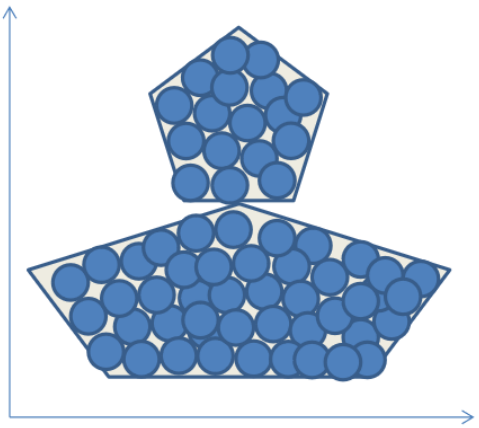
A decision boundary for a perfect circle can be obtained. We also can take a 2 different circle subnets which

compose of circles in two different position and still use the same threshold value on the output perceptron. This is because the "sum" of two circles of the individual subnets is exactly $\frac{N}{2}$ inside either circle and almost 0 everywhere outside of the circles.

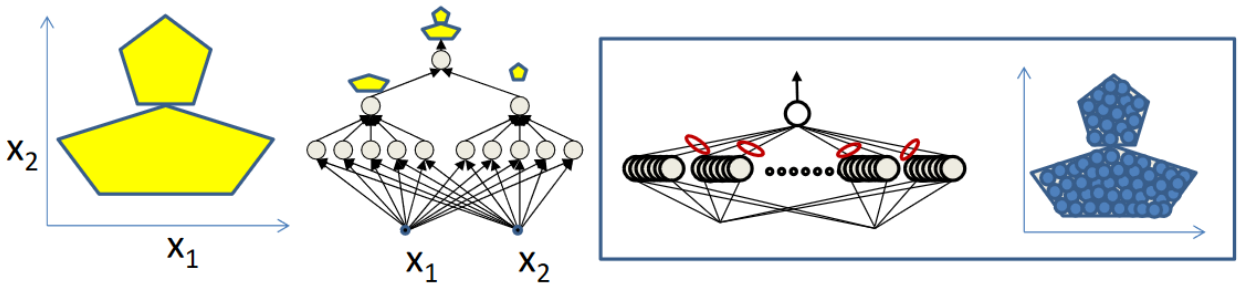


This would also means that it is possible to use subnets of circle decision boundary to compose any arbitrary figure. We just have have to fill in the composite shapes with enough circle decision boundary. An accurate approximation could be achieved with greater number of smaller circles. This demonstrate the capabilities of one hidden layer MLP being able to model any classification boundary with enough number of neurons.

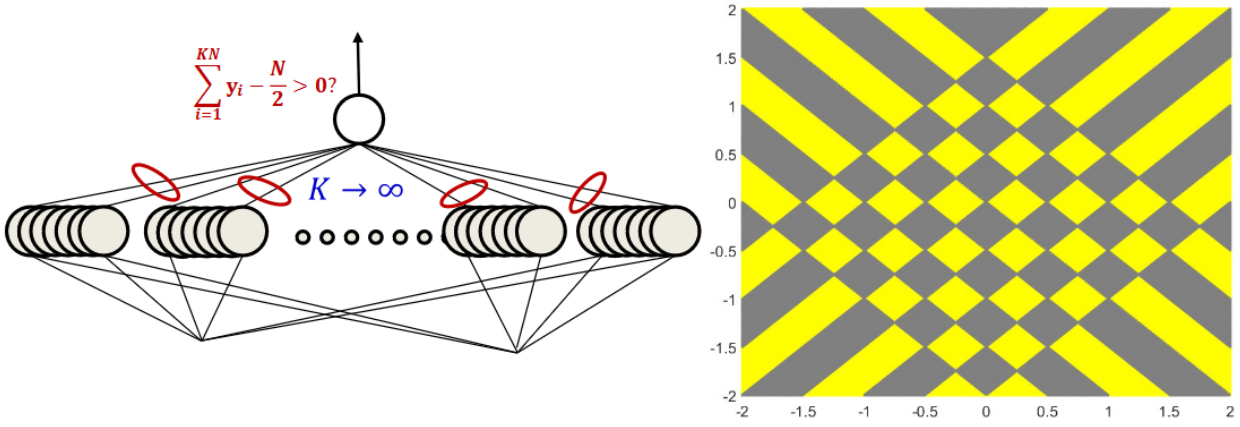




2.8 Performance comparison between deep network and single layer network.

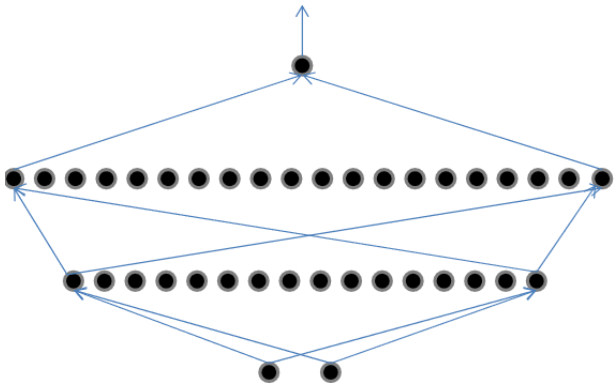


Deep networks require far fewer neutrons to achieve the same result. Depending on the compleexity of the decision boundary, a naive one-hidden-layer neural network will require infinite hidden neurons. For example, consider an image below, a single layer MLP would require infinite amount of neurons as sum of subset of the circles that is going to be filled into the yellow region.



Howerver, if observed carefully, this image contains 16 lines and 40 segments forming the yellow regions. This would means that we can compose a 2 hidden layer MLP where every individual neurons on the same layer are XOR-ed together such that

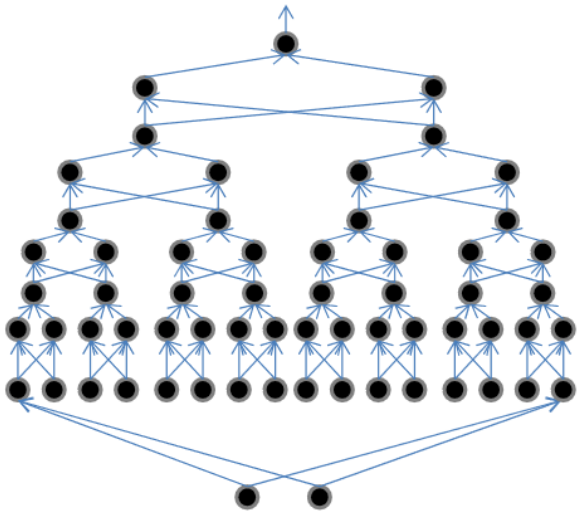
- 16 neurons on the first hidden layer.
- 40 neurons on the second hidden layer.
- 1 neurons on the output layer.



Alternatively, we could observe that the output of the first layer is every 16 individual lines of the decision boundary Y, it form a pattern such that for a single neuron that corresponds to each of the line to fire, it is XOR-ed with the rest of the neurons of every other lines. This would mean that the output of the network is:

$$Y_1 \oplus Y_2 \oplus ... \oplus Y_{16} \tag{25}$$

This would also means that we can compose the first layer of the network as standard XOR network:

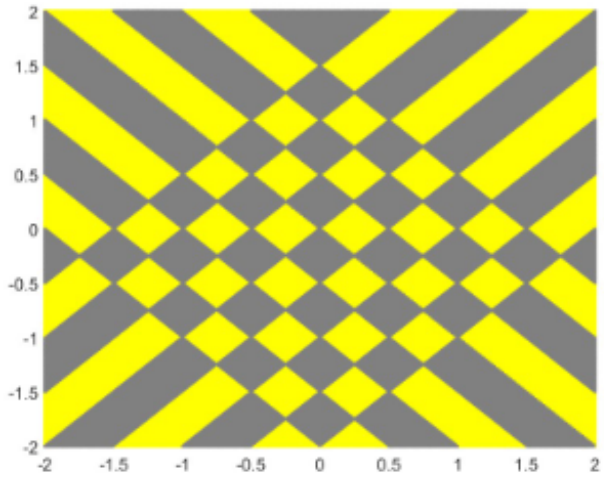


The typical XOR network will require $16 + 155 \times 3 = 61$ neurons. It is also possible to achieve the network with 46 neurons if we are using two-neurons XOR model. We can summarized that the number of neurons required in shallow network is potentially exponential in the dimensionality of the input. Deeper networks may require far fewer neurons than shallower networks to express the same function, making them much more expressive and smaller than their single layer counterparts.

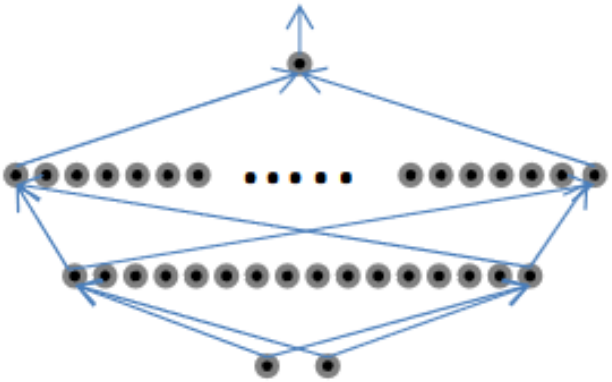
2.9 Sufficiency of Architecture.

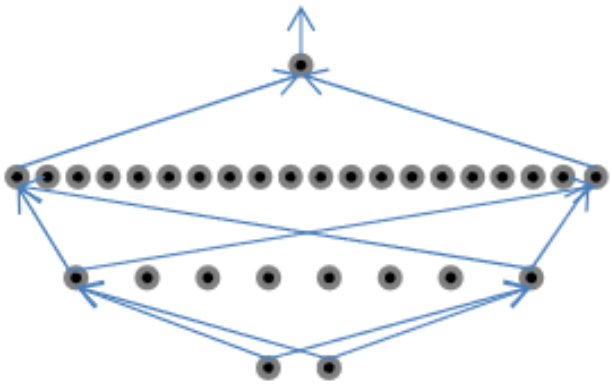
A neural network can represent any function provided that it is sufficiently abroad and deep to represent the function, however not all architectures can represent any function. Consider a boundary condition shown

below:



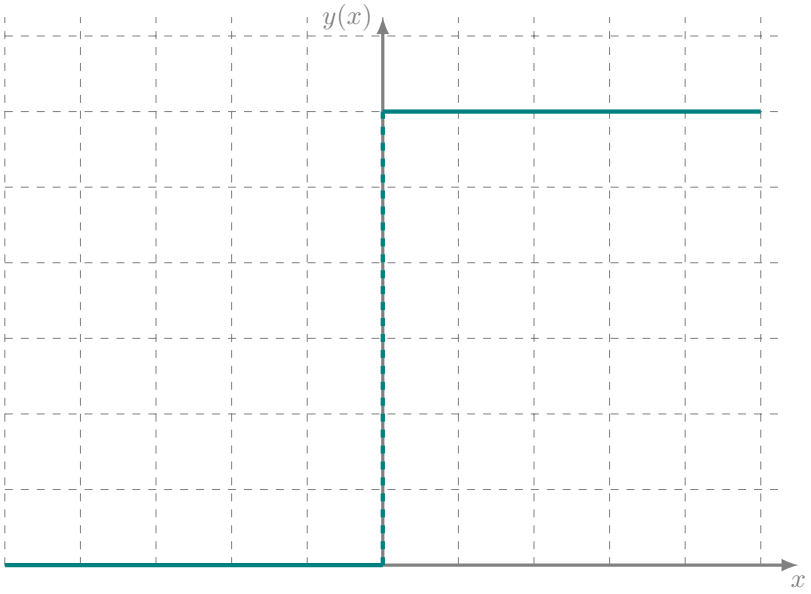
A network with 16 or more neurons in the first layer is capable of representing the boundary condition perfectly. However, a network that is having less than 16 neurons in the first layer would not be able to represent this pattern exactly.



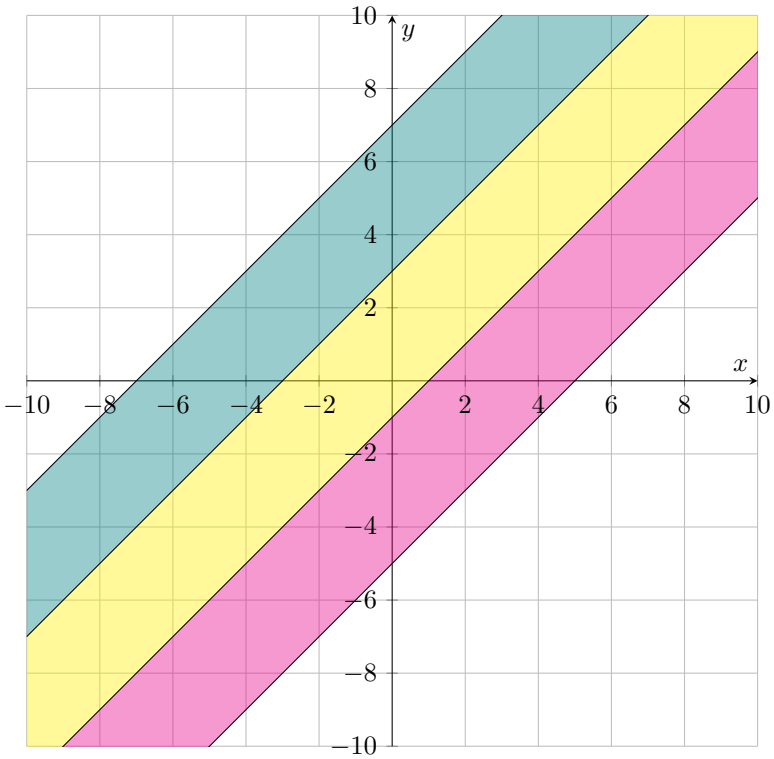


This is because a network with x threshold neurons in the first layer may only capture x number of boundaries. This gives information about which boundary where the threshold is applied to, but not where the boundary is. Similarly restriction is also applied to the higher layers. A 2-layer network with 16 neurons in the first layer alone are not sufficient to represent the checkered region of the pattern if the second layer is having less than 40 neurons inside it. Regardless of depth, every layer must be sufficiently wide in order to express the function accurately.

This is because we are using the threshold activation value. It gates information in the input from the later layer since the output is either 0 or 1. This will lead to a pattern of outputs within any region is identical while the subsequent layers do not obtain enough information to partition them. In order to maintain the position information such as the distance from a boundary, a continuous activation function such as sigmoid function has to be used.



This activation function caused a well defined boundary limit which lead to the following pattern.



A continuous activation functions result in graded output at the layer. The gradation provides information to subsequent layers, to capture information "missed" by the lower layer.