ANALYTIXLABS

Artificial Neural Networks(ANN)

Artificial Neural Networks

Neural Networks

- ✓ Why do we need neural networks?
  - Complex Non Linear Hypothesis
  - Good way to build classifiers when N is large

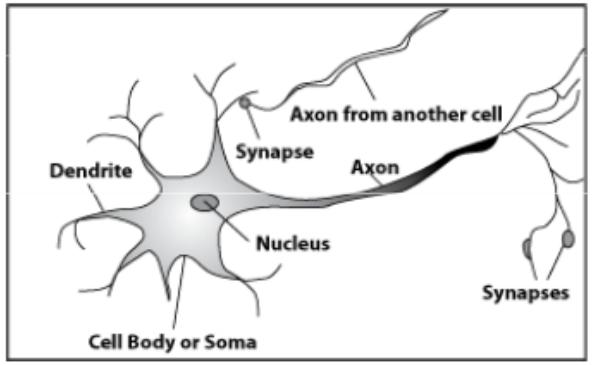
- ✓ **Neural networks (NNs)** were originally motivated by looking at machines which replicate the brain's functionality looked at here as a machine learning technique

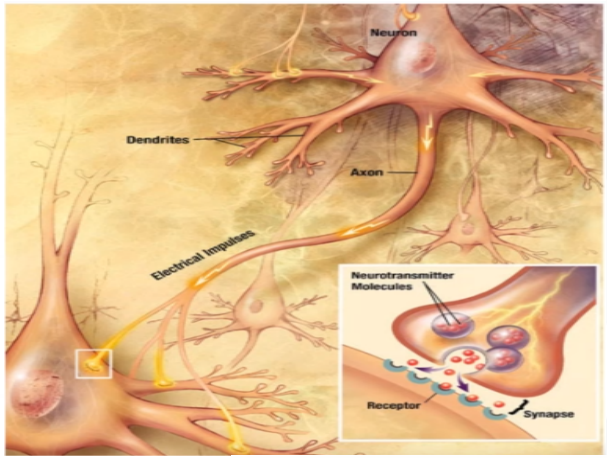
- ✓ **Origins**
  - To build learning systems, why not mimic the brain?
  - Used a lot in the 80s and 90s
  - Popularity diminished in late 90s

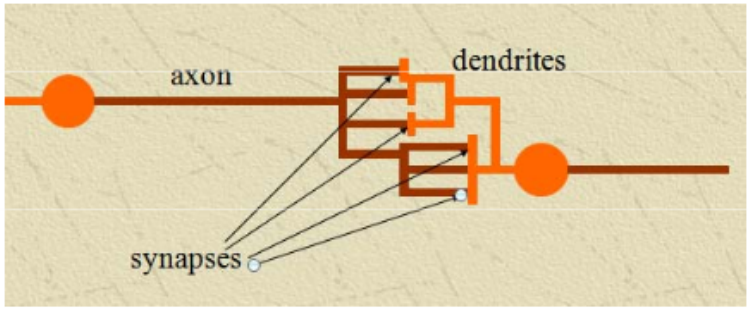
- ✓ **Recent major resurgence**
  - NNs are computationally expensive, so only recently large scale neural networks became computationally feasible

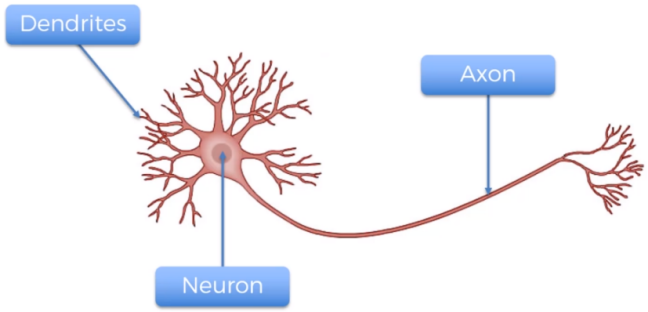
ANN - Introduction

Biological Neural Network

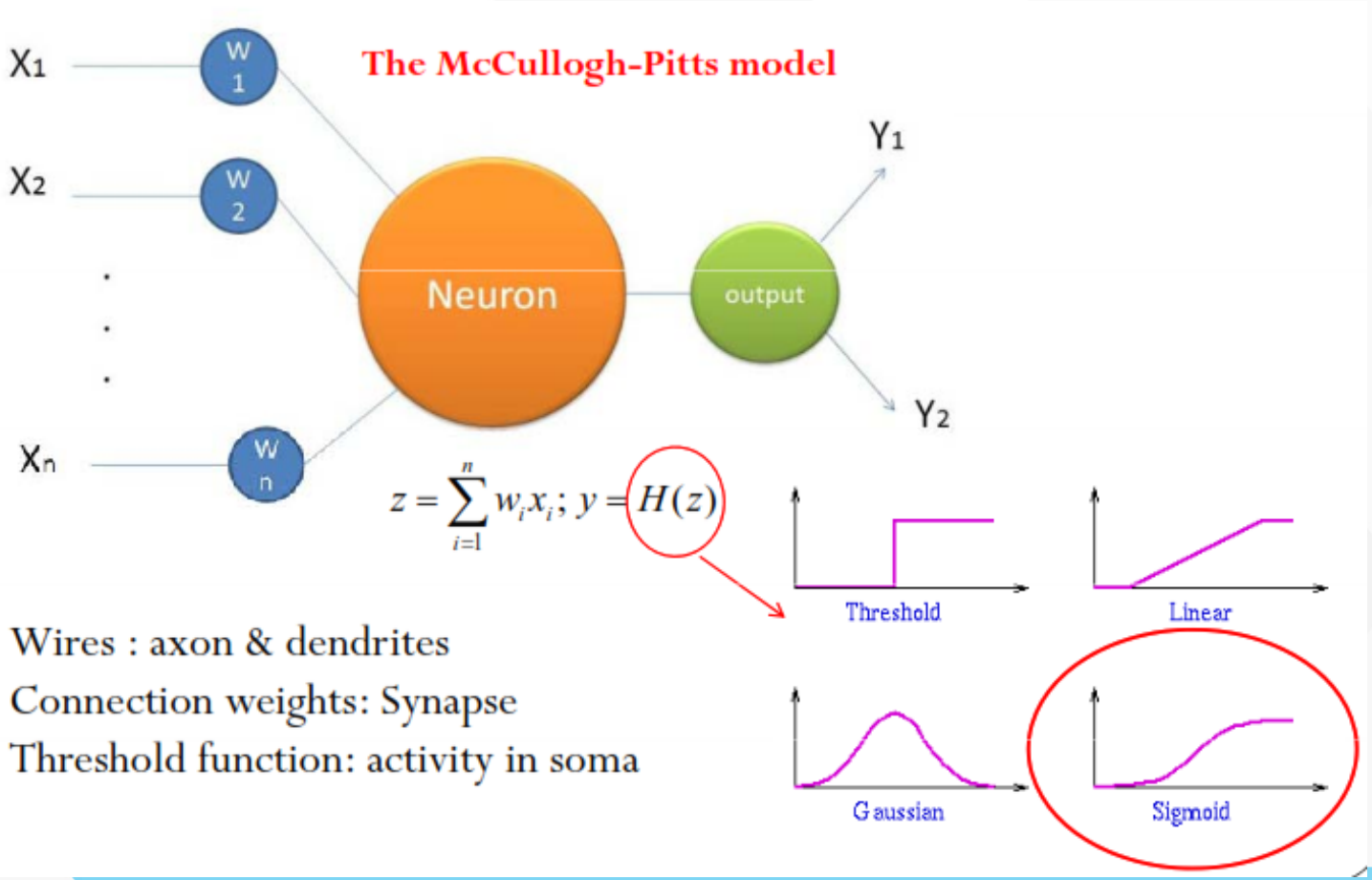








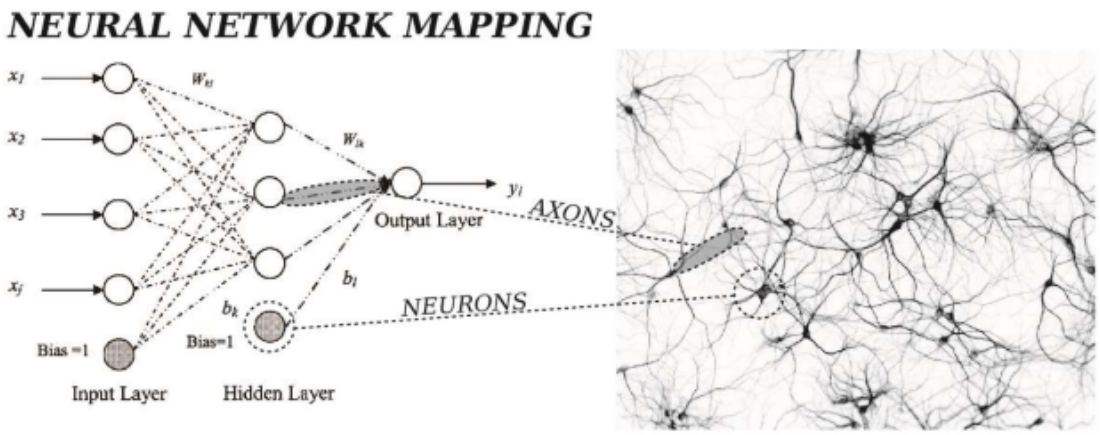
ANN - Introduction

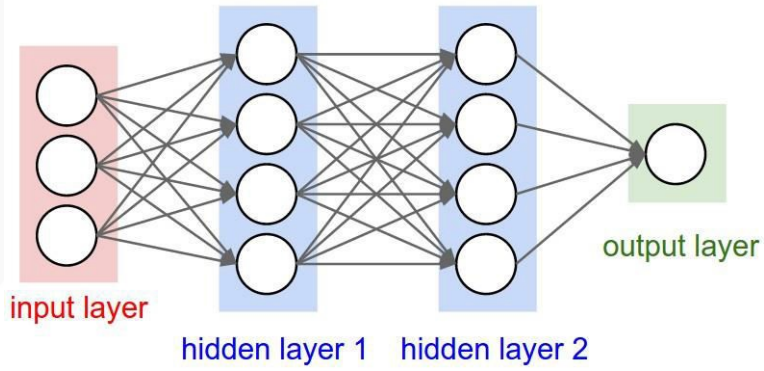


ANN – Deep Learning

- Generally speaking, **neural networks** are nonlinear machine learning models.
  - They can be used for **supervised** or **unsupervised** learning.

- **Deep learning** refers to training neural nets with multiple layers.
  - They are more powerful but only if you have lots of data to train them on.





BNN vs. ANN

Biological Neural Network

- Soma
- Dendrites
- Synapse
- Axon

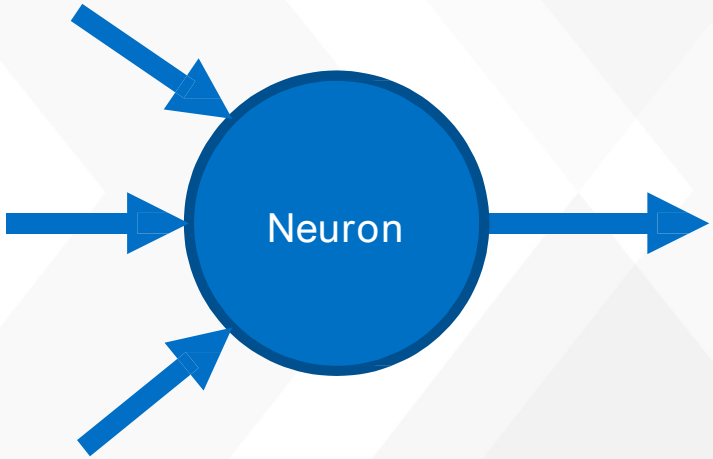
Artificial Neural Network

- Node
- Input
- Weights or Interconnections
- Output

Neuron

There are many models that represent the problem and make decisions in different ways, each with their own advantages and disadvantages.

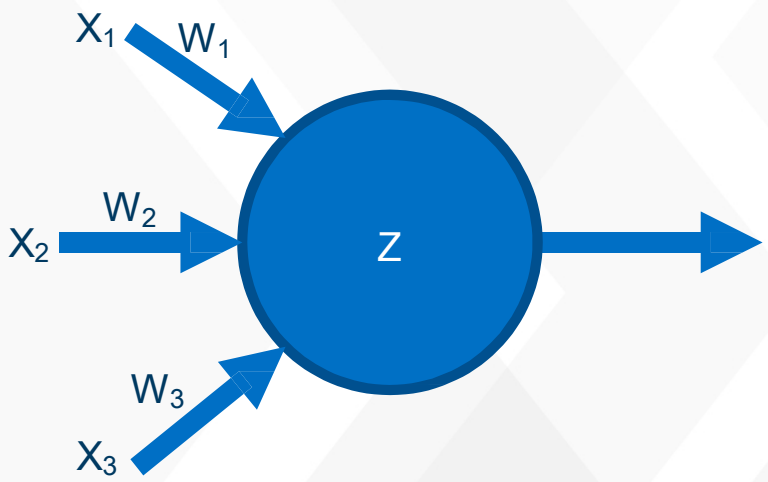
- DL models are biologically inspired.
- The main building block is a **neuron**.



Neuron

A neuron multiplies each feature by a **weight** and then adds these values together.

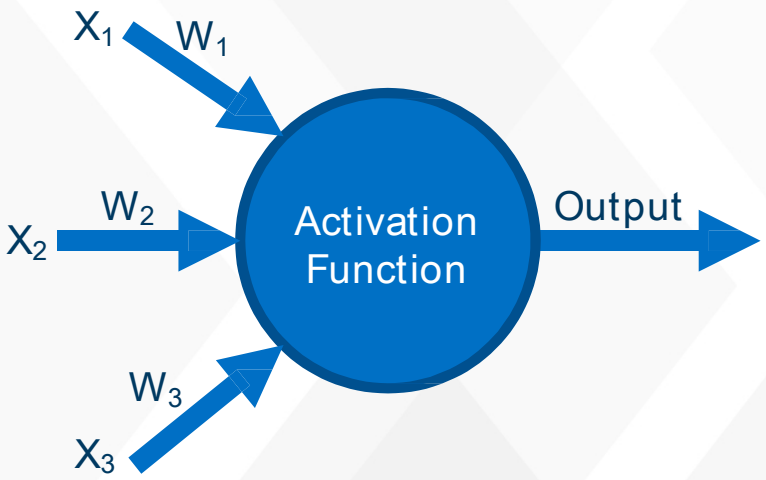
- $Z = X_1W_1 + X_2W_2 + X_3W_3$



Neuron

This value is then put through a function called the **activation function.**

- There are several activation functions that can be used.
- The output of the neuron is the output of the activation function.



In vector notation

$$z = b + \sum_{i=1}^{m} x_i w_i$$

$$z = b + x^T w$$

$$a = f(z)$$

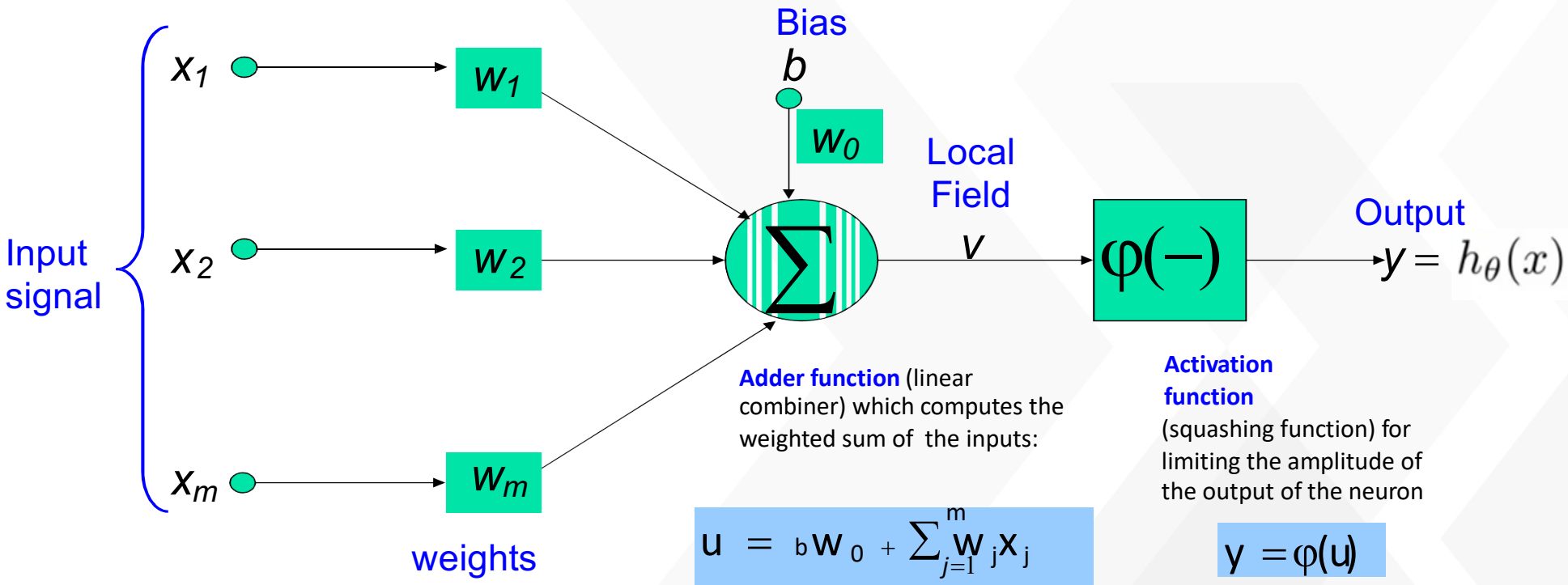
z = "net input"

b = "bias term"

f = activation function

a = output to next layer

Neuron - Snapshot



Adder function (linear combiner) which computes the weighted sum of the inputs:

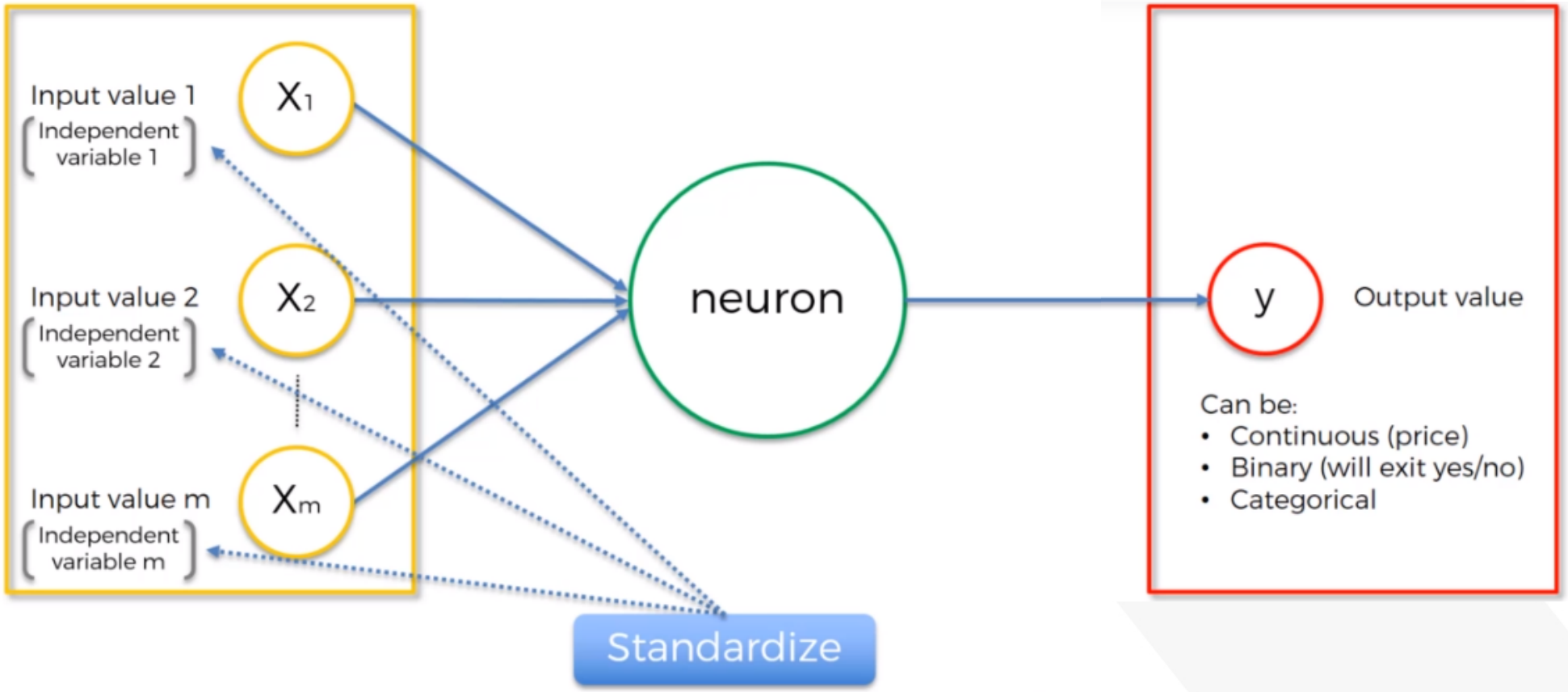
$$u = {}_b w_0 + \sum_{j=1}^{m} w_j x_j$$

Activation function (squashing function) for limiting the amplitude of the output of the neuron

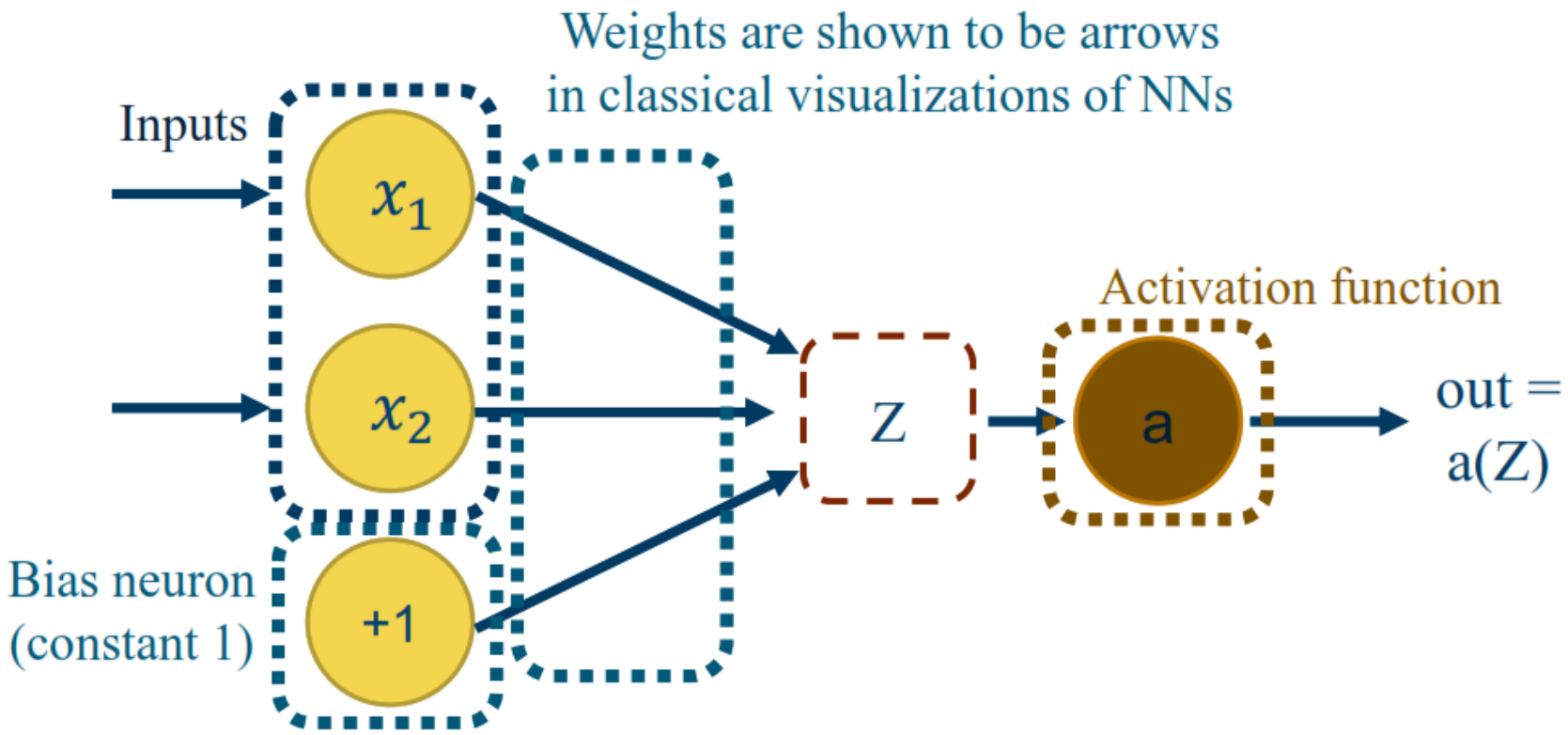
$$y = \varphi(u)$$

Bias: serves to vary the activity of the unit

Neuron


Input value 1
Independent variable 1
X1
Input value 2
Independent variable 2
X2
Input value m
Independent variable m
Xm
neuron
y
Output value
Can be:
• Continuous (price)
• Binary (will exit yes/no)
• Categorical
Standardize

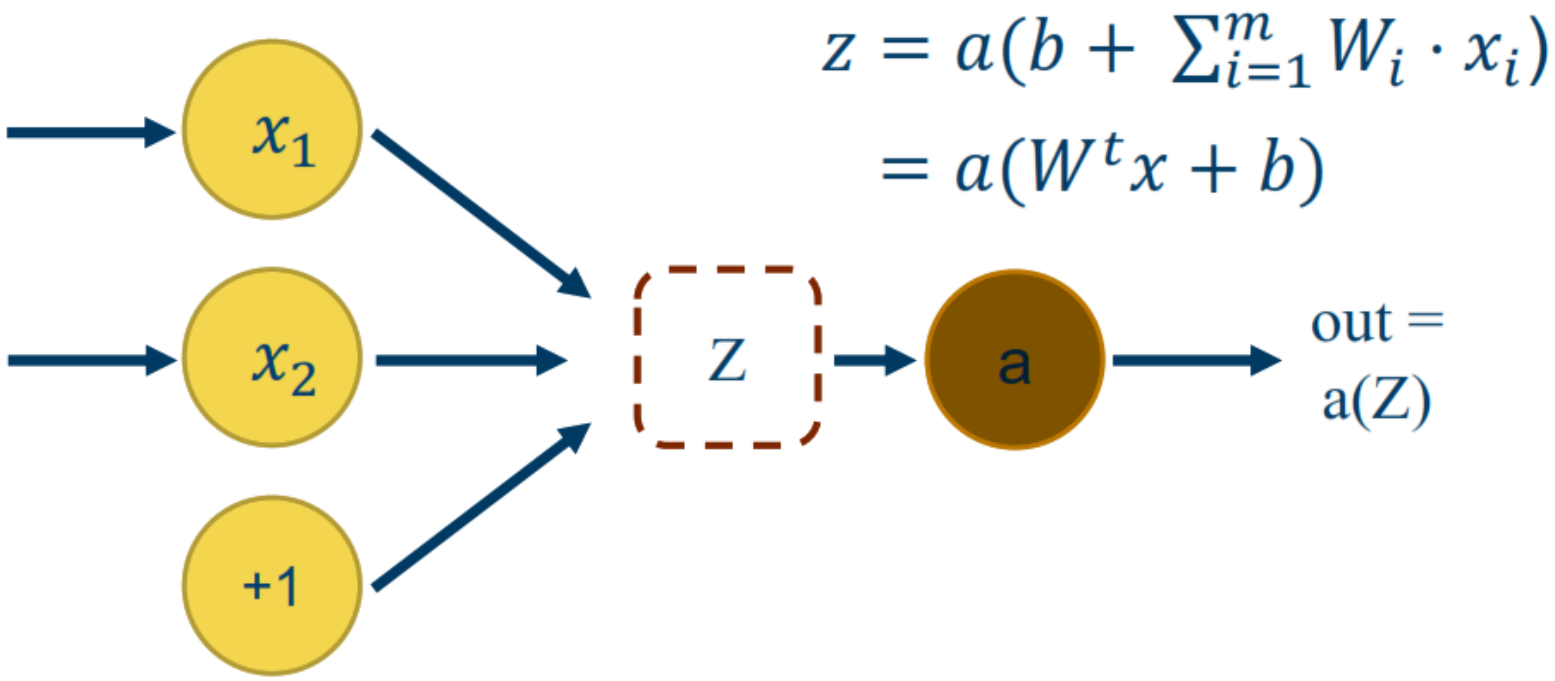
Classic Visualization of Neurons



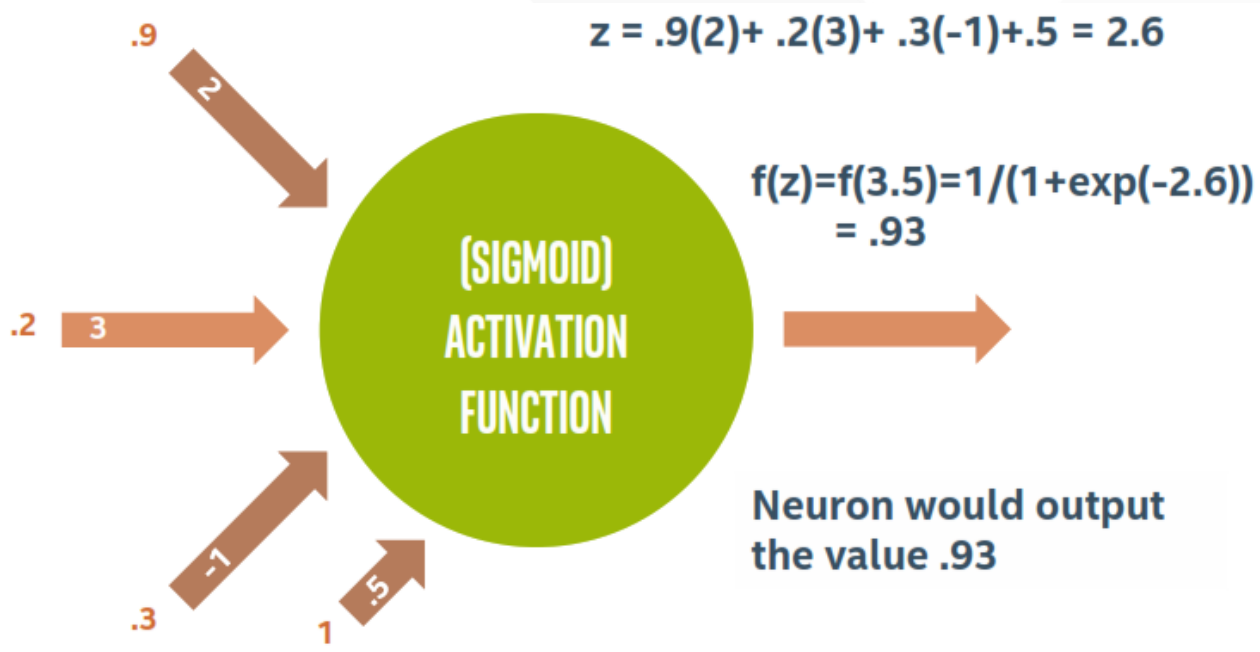
Nodes

Nodes are the primitive elements.

out = activation(f(in) + bias)



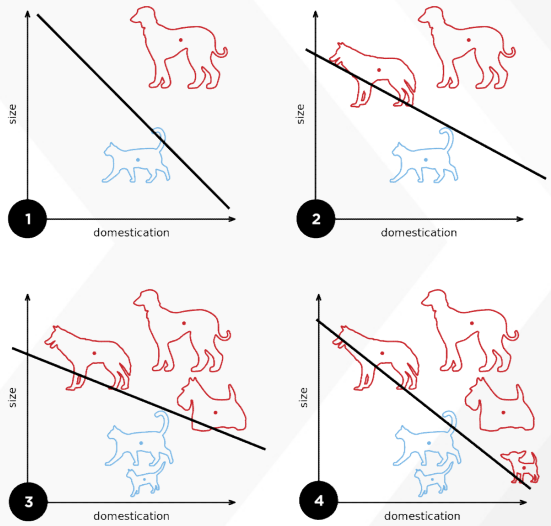
Example Neuron computation



Perceptron

A neuron with a simple activation function can solve **linearly separable** problems.

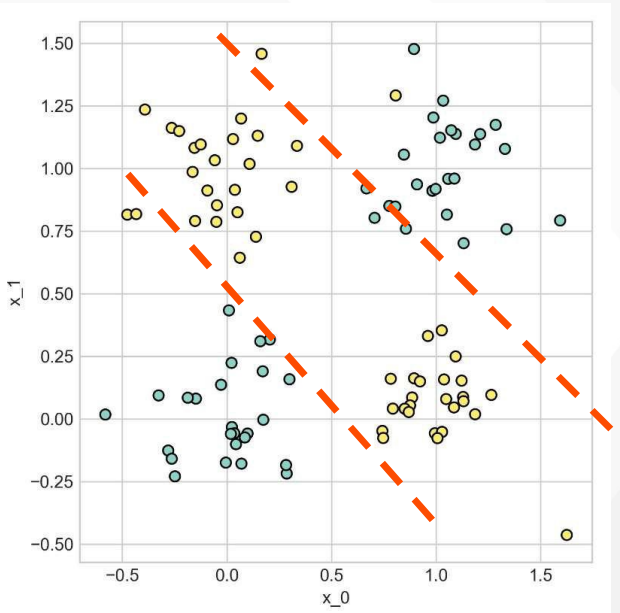
- These are problems where the different classes can be separated by a line.
- **The Perceptron**: one of the earliest neural network models that used neurons with simple activation functions.



Perceptron

Problems where the labels cannot be separated by a single line are not solvable by a single neuron.

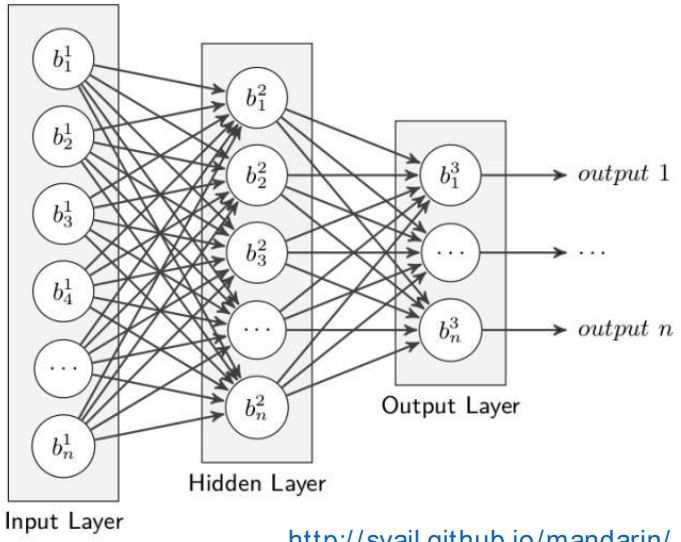
- This is a major limitation, and one of the reasons for the first AI winter, that was discussed in lesson 1.



Fully Connected Network

More complicated problems can be solved by connecting multiple neurons together and using more complicated activation functions.

- Organized into **layers** of neurons.
- Each neuron is connected to every neuron in the previous layer.
- Each layer transforms the output of the previous layer and then passes it on to the next.
- Every connection has a separate weight.

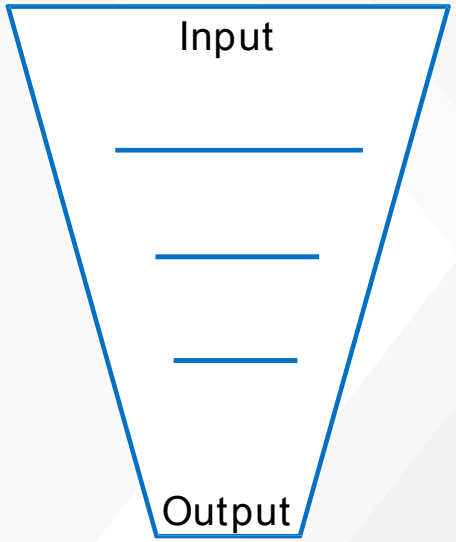


http://svail.github.io/mandarin/

Deep Learning

Deep Learning refers to when many layers are used to build **deep networks**.

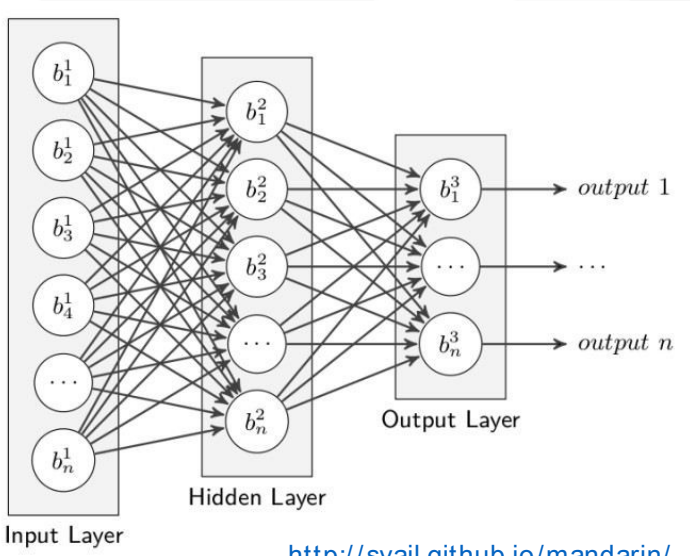
- State-of-the-art models use hundreds of layers.
- Deep layers tend to decrease in width.
- Successive layers transform inputs with two effects:
- **Compression**: each layer is asked to summarize the input in a way that best serves the task.
- **Extraction**: the model succeeds when each layer extracts task-relevant information.



Steps in Building a Fully Connected Network

To build a fully connected network a user needs to:

- Define the network architecture.
- How many layers and neurons?
- Define what activation function to use for each neuron.
- Define an evaluation metric.
- The values for the weights are learned during model training.



http://svail.github.io/mandarin/

Evaluating Metric

The metric used will depend on the problem being solved. Some examples include:

- Regression
  - Mean Squared Error
- Classification
  - Categorical Cross-Entropy
- Multi-Label classification
  - Binary Cross-Entropy

Fully Connected Network Problems

Not optimal for detecting features.

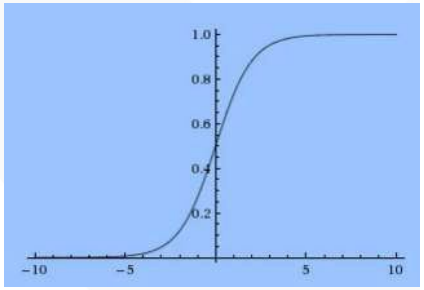
- Computationally intensive – heavy memory usage.

Basics - Activation Functions

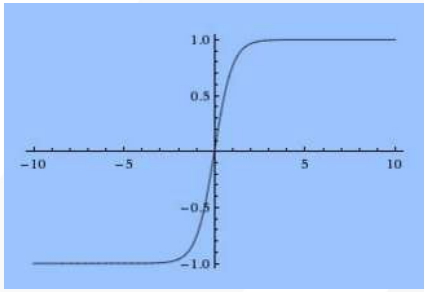
- **Activation Functions:** Popularly used activation functions
  - Sigmoid
  - Tanh
  - Relu
  - Linear

- **Sigmoid:** Takes real valued number and squashes it into range between 0 and 1.

- **Tanh:** Tanh squashes the real valued number to the range [-1,1]. Output is zero centered.



Tanh Activation Function

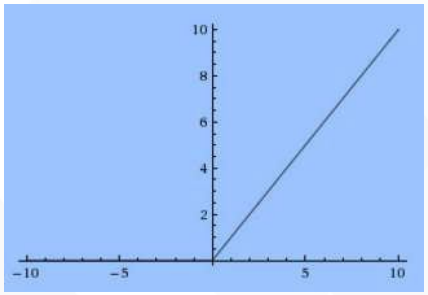


Function	Definition
Identity	x
Logistic	$\frac{1}{1-e^{-1}}$
Hyperbolic	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Exponential	e^{-x}
Softmax	$\frac{e^{x}}{\sum_{i} e^{x_i}}$
Square root	$\sqrt{x}$
Sine	$\sin(x)$
Ramp	$\begin{cases} -1 & x \le -1 \\ x & -1 < x < +1 \\ +1 & x \ge +1 \end{cases}$
Step	$\begin{cases} 0 & x < 0 \\ +1 & x \ge 0 \end{cases}$
Logarithmic	$\ln(x+1) \quad x \ge 0$ $-\ln(-x+1) \quad x < 0$
Inverse Tangent	$\arctan(x)$

Basics - Activations Functions

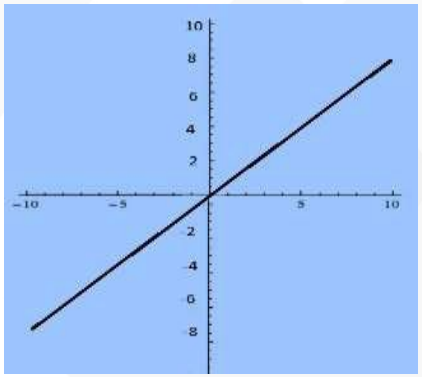
- **ReLU (Rectified Linear Unit):** Very popular now. Activation is simply thresholds at zero

ReLU Activation Function



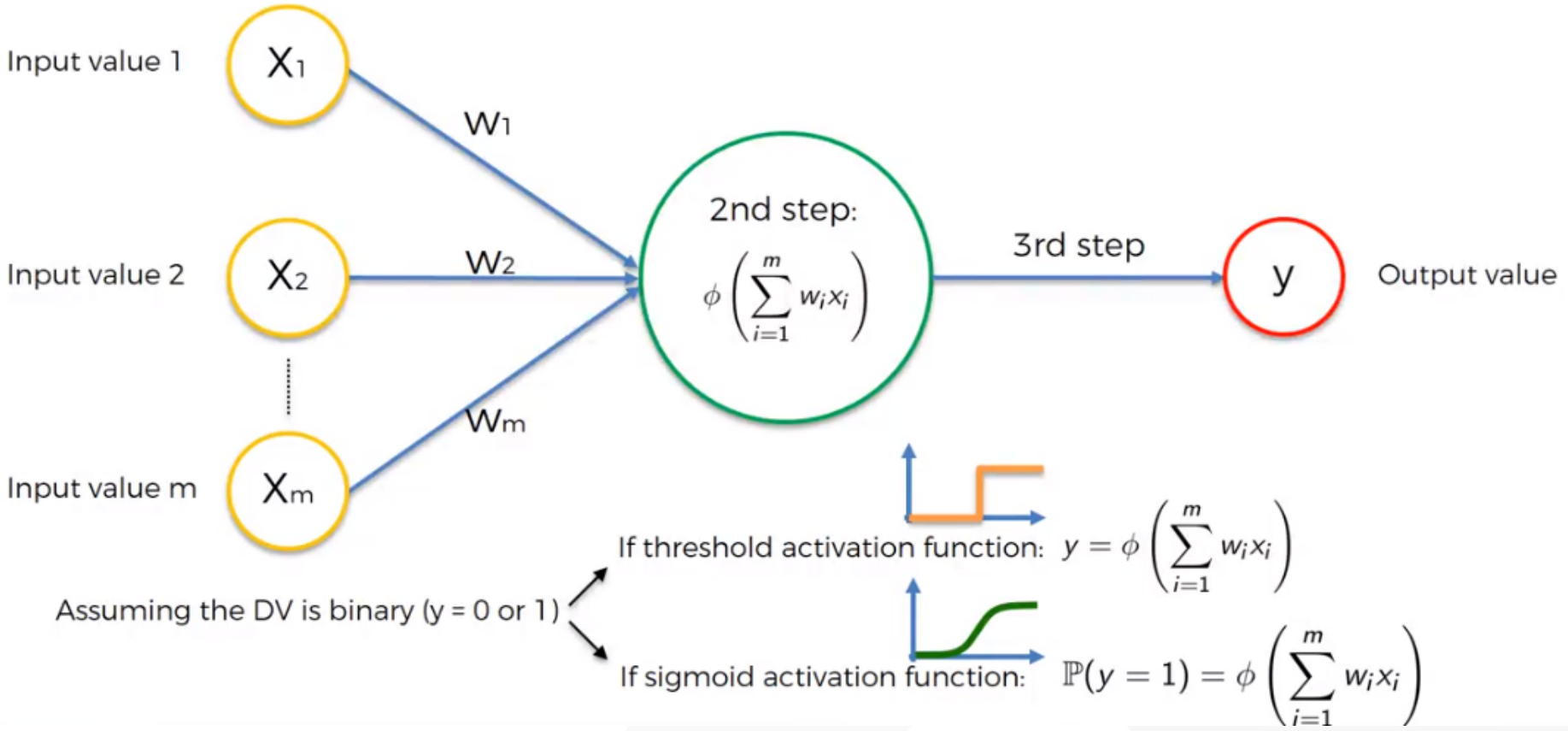
- **Linear:** Linear activation function is also used

Linear Activation Function

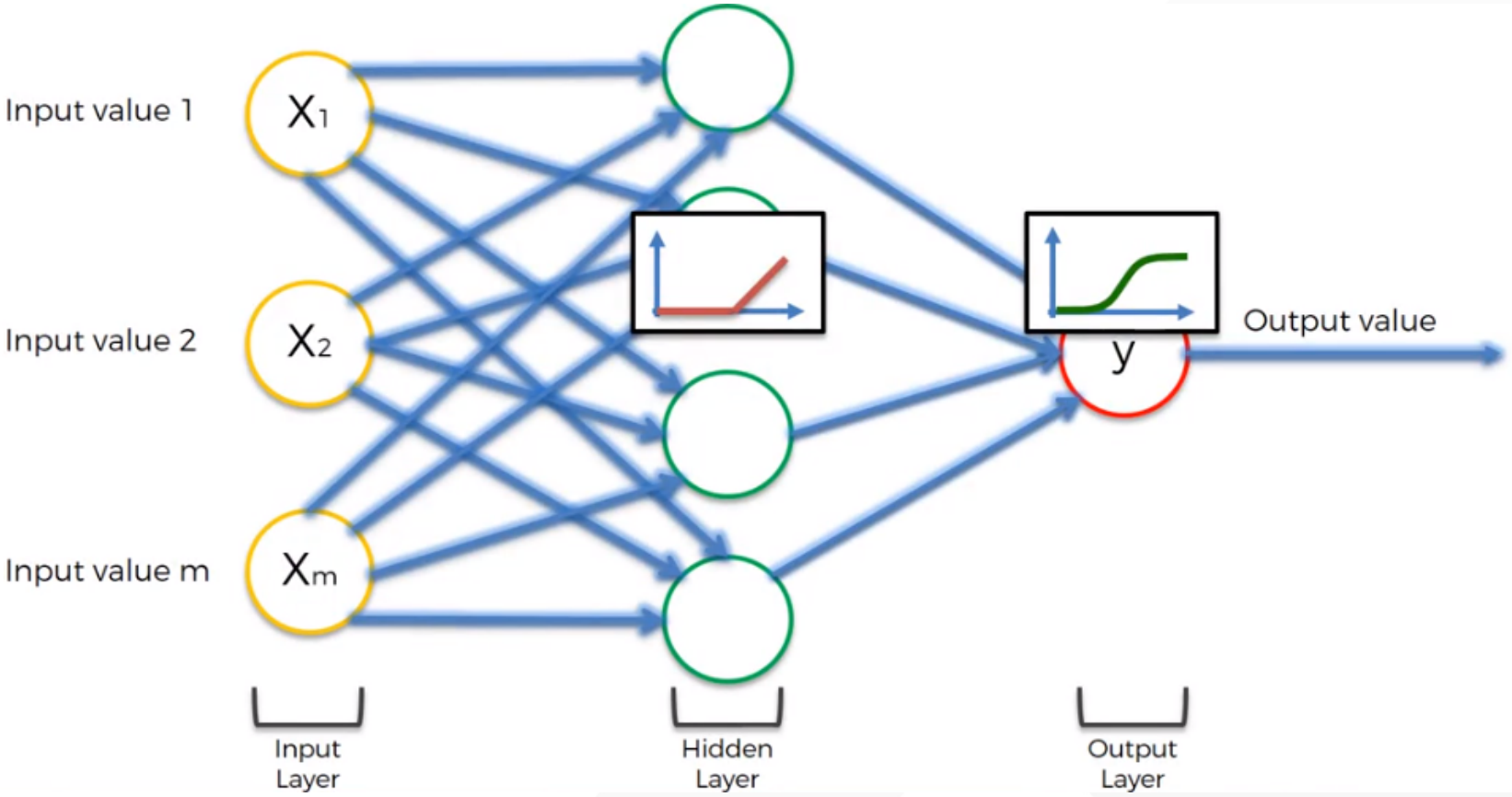


Relu is now popularly being used in place of **Sigmoid** or **Tanh** due to its better property of convergence

Example – Usage of Activation function



Typical Activation functions for solving Binary Classification problem



Hidden Layers: Rectifier function (RELU)

Output Layers: Sigmoid Function (Logistic)

Learning of ANN

- **What is the learning process in ANN?**
  - updating network architecture and connection weights so that network can efficiently perform a task
- **What is the source of learning for ANN?**
  - Available training patterns
  - The ability of ANN to automatically learn from examples or input-out put relations
- **How to design a Learning process?**
  - Knowing about available information
  - Having a model from environment: Learning Paradigm
  - Figuring out the update process of weights: Learning rules
  - Identifying a procedure to adjust weights by learning rules: Learning algorithm

How do the Neural Networks learn?

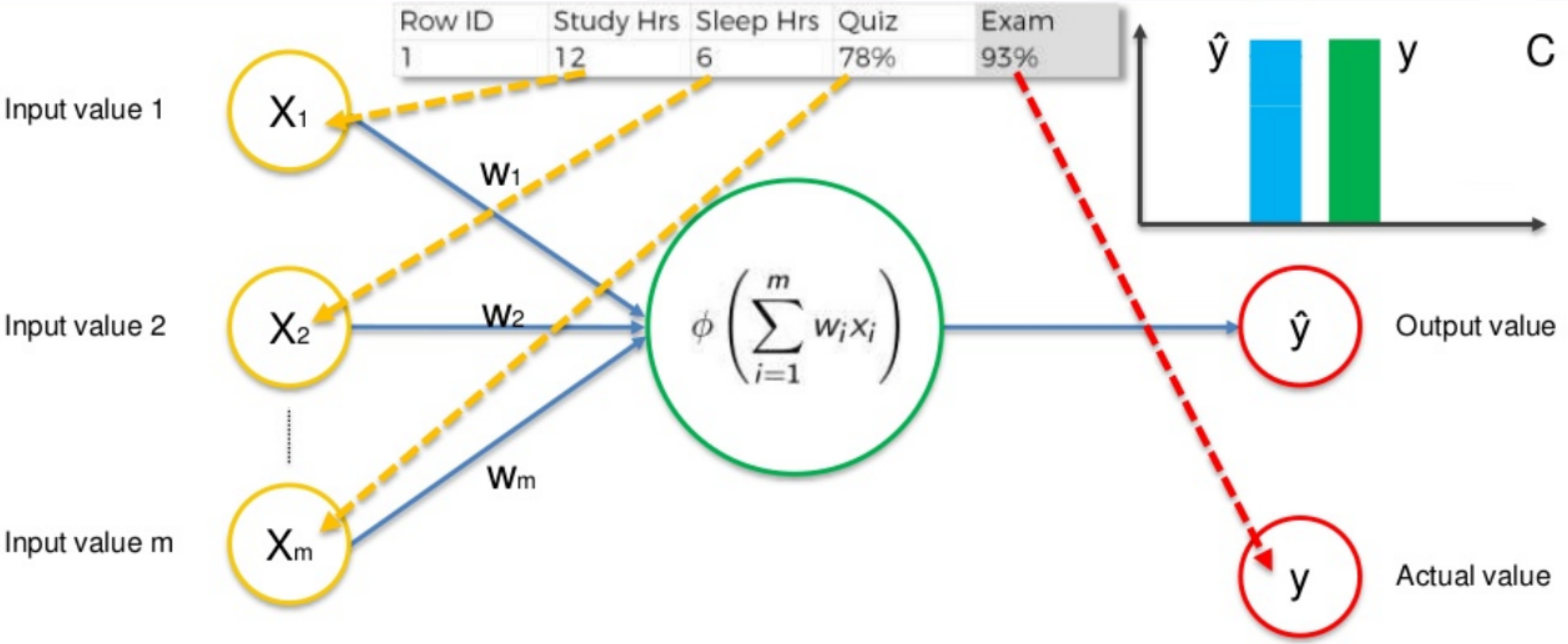
- Initially
  - choose small random weights (w_i)
  - Set threshold = 1 (step function)
  - Choose small *learning rate* (r)

- Apply each member of the *training set* to the neural net model using a *training rule* to adjust the weights
  - For each unit
    - Compute the net input to the unit as a linear combination of all the inputs to the unit
    - Compute the output value using the activation function
    - Compute the error
    - Update the weights and the bias

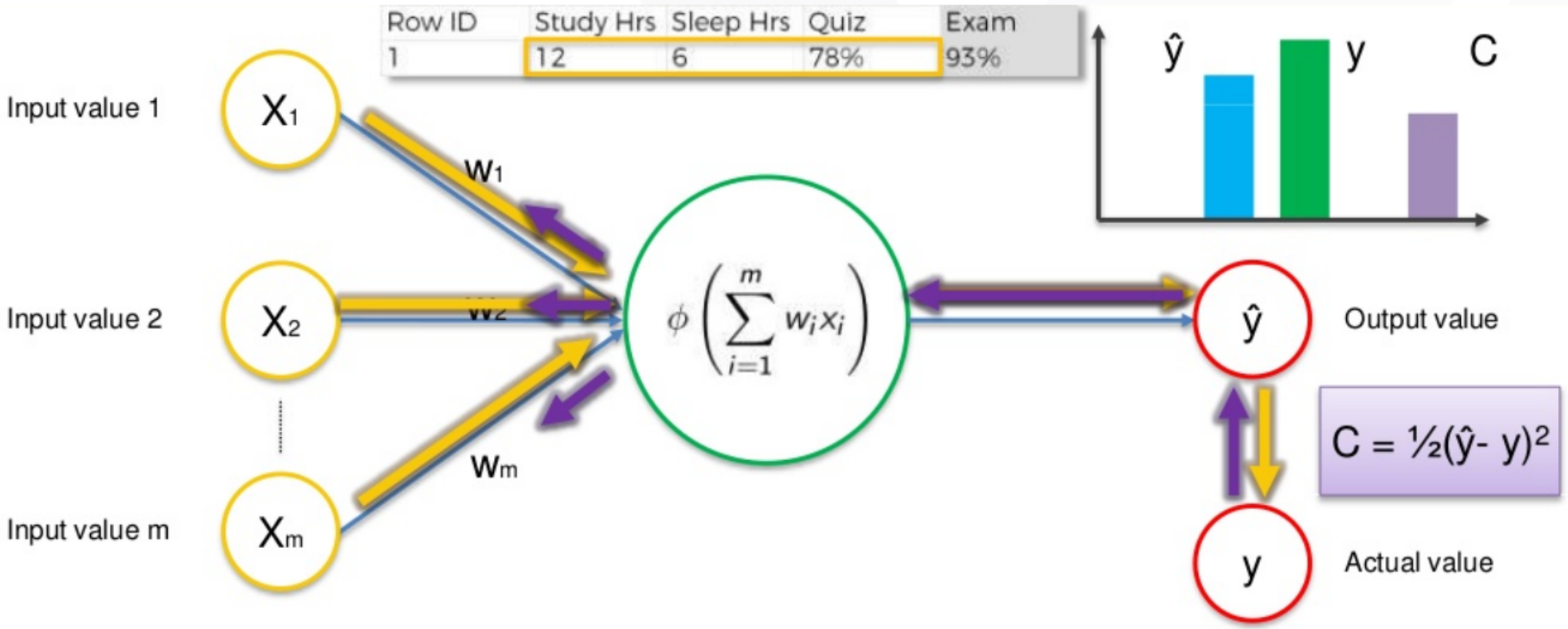
How do the Neural Networks learn?

- Let’s take an example of regression problem
- Problem: Want to predict how much student will score in exam based on different attributes like study hours, sleep hours, quiz marks etc
- Y = Final score
- X1 = Study hours
- X2 = Sleep hours
- X3 = Quiz marks

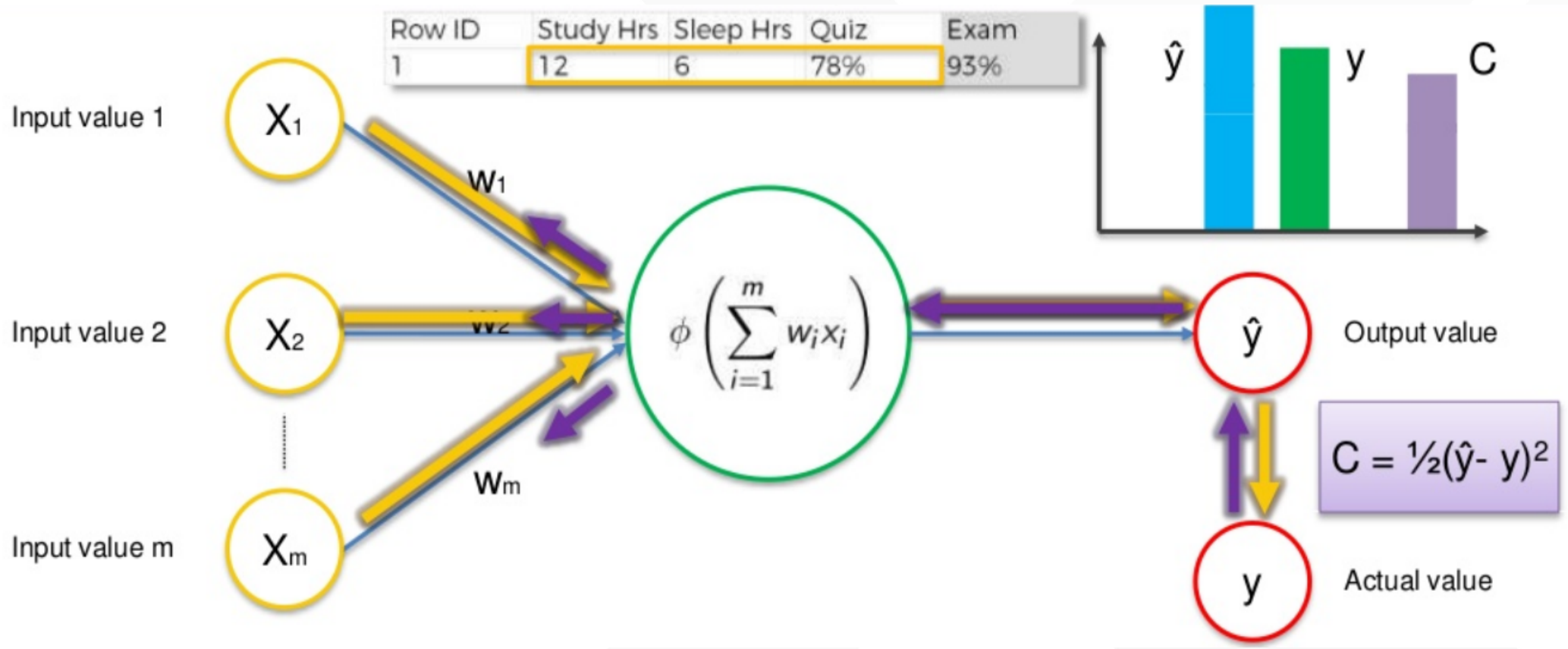
How do the Neural Networks learn?



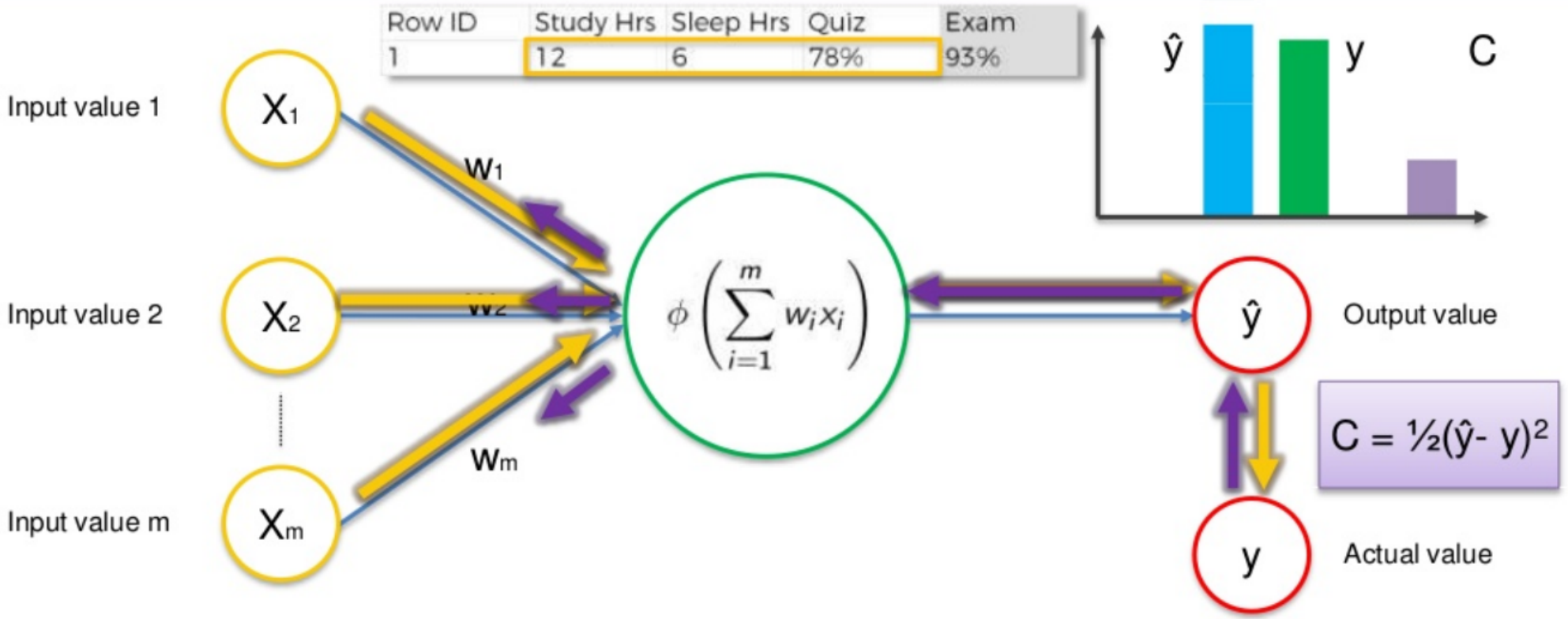
How do the Neural Networks learn?



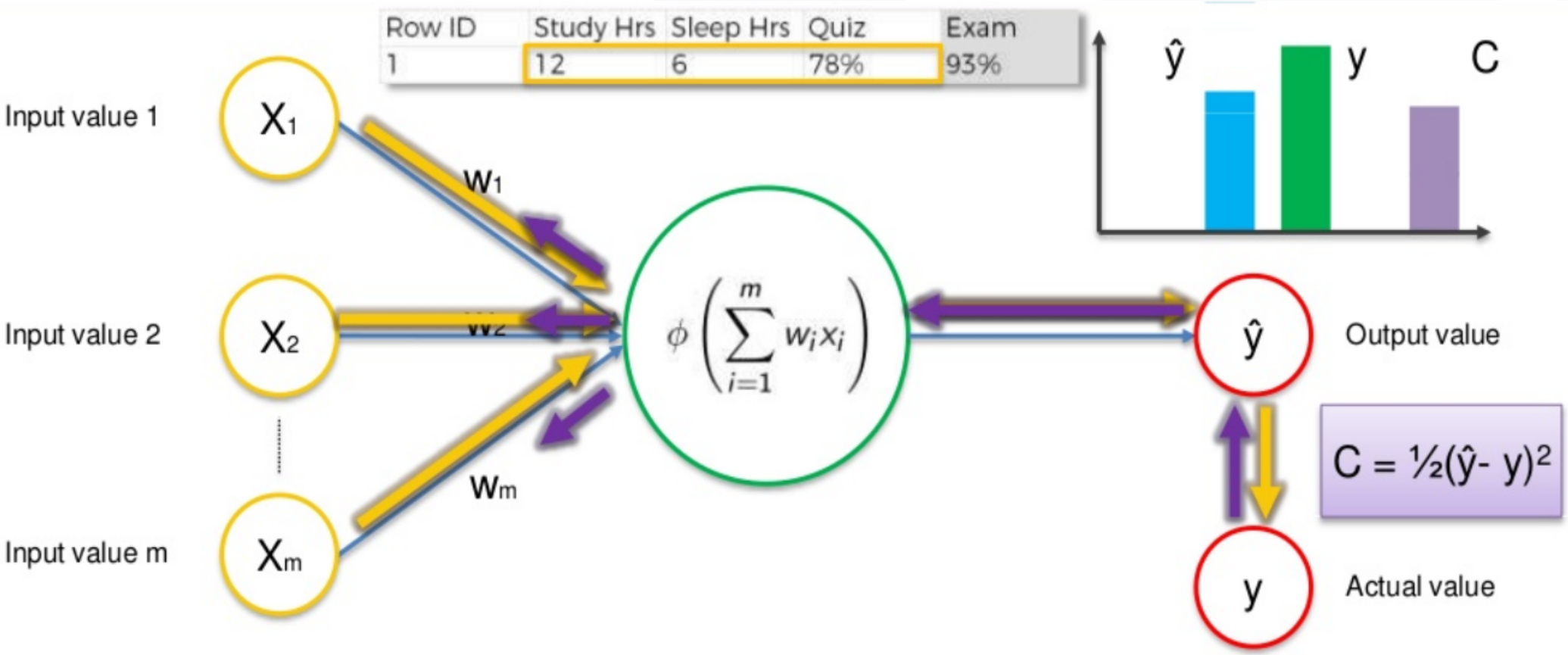
How do the Neural Networks learn?



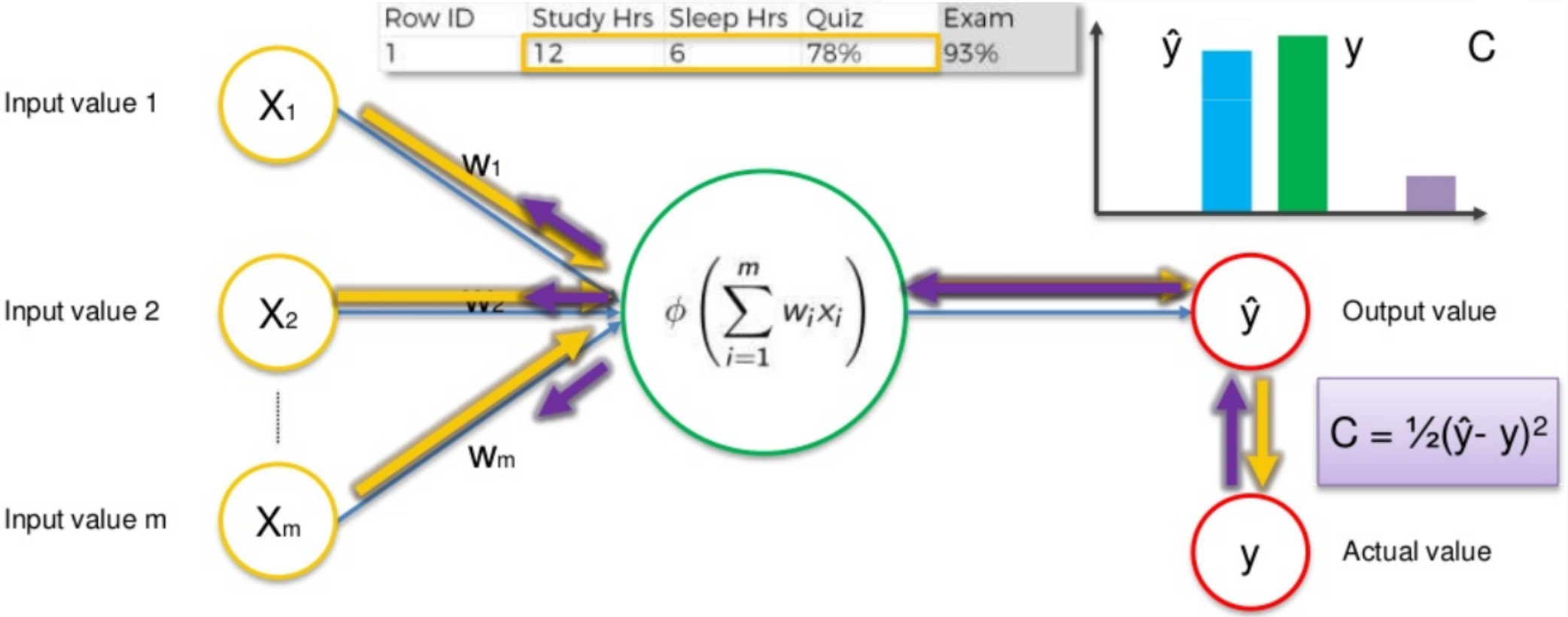
How do the Neural Networks learn?



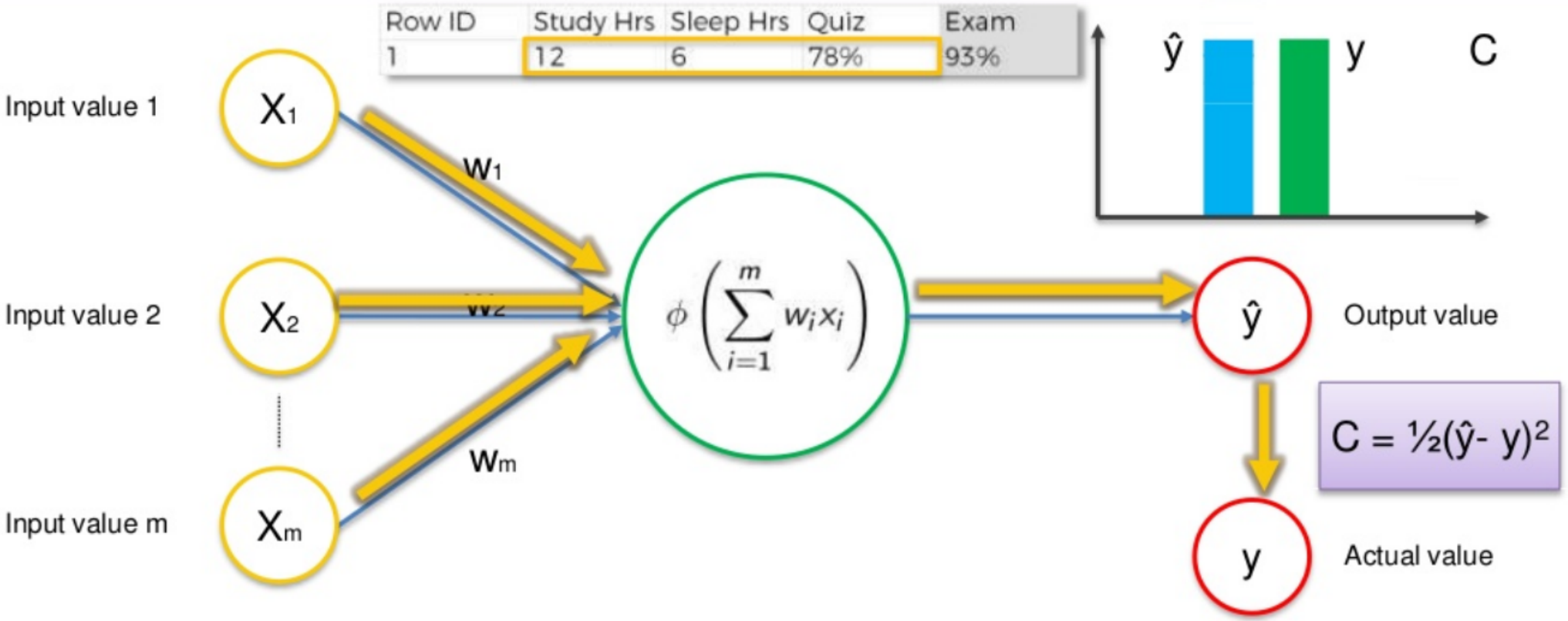
How do the Neural Networks learn?



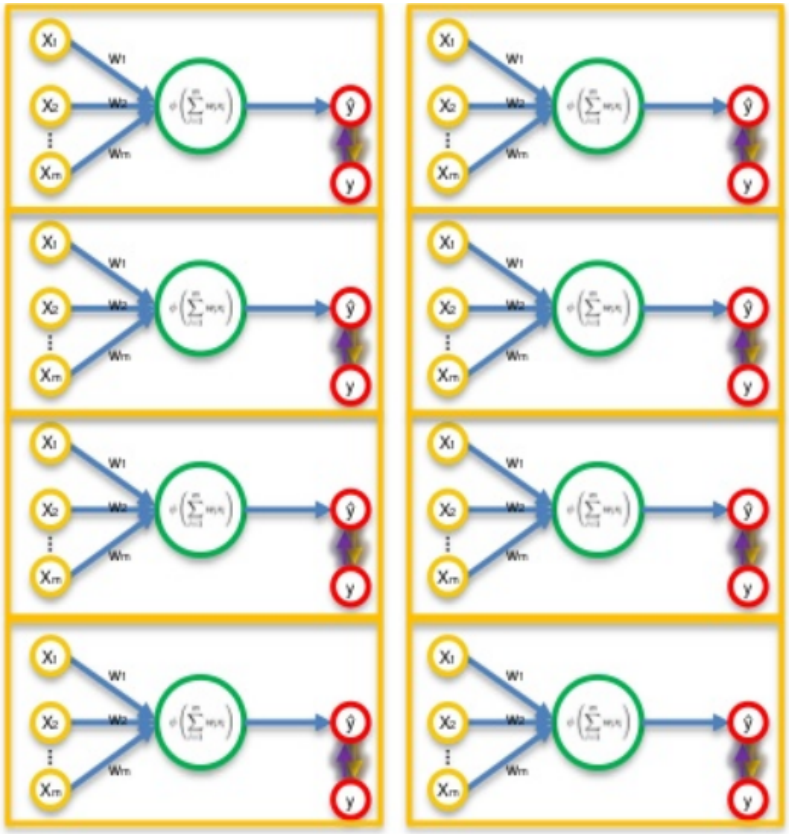
How do the Neural Networks learn?



How do the Neural Networks learn?

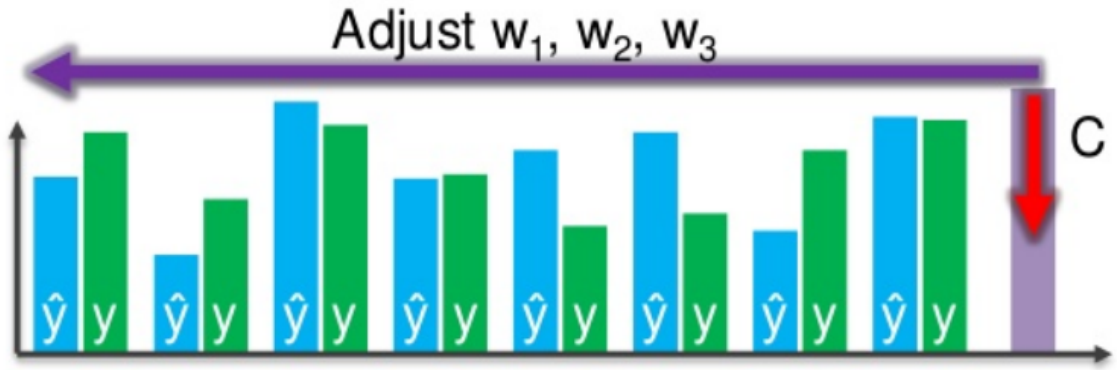


How do the Neural Networks learn?

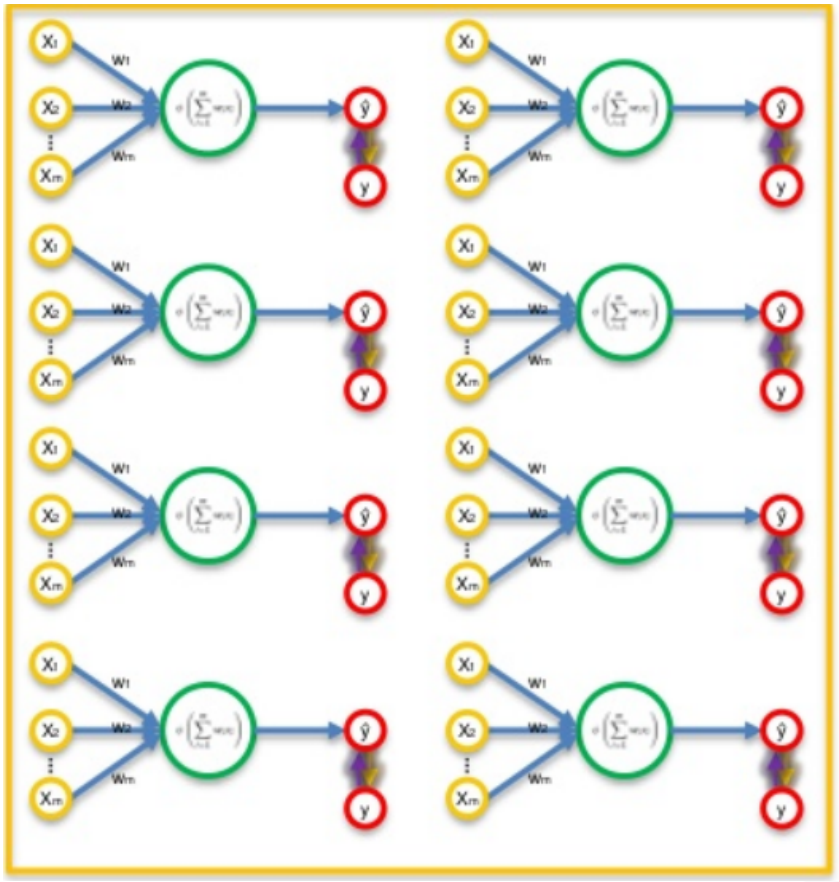


Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$

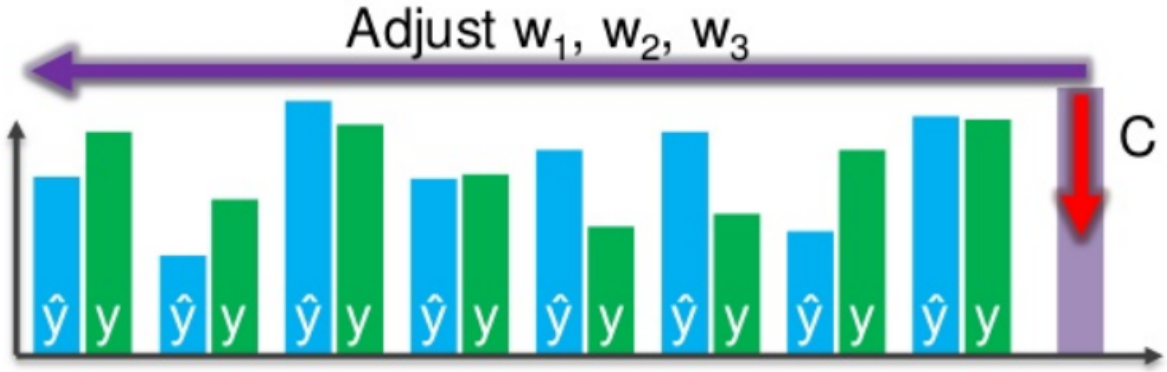


Gradient Descent

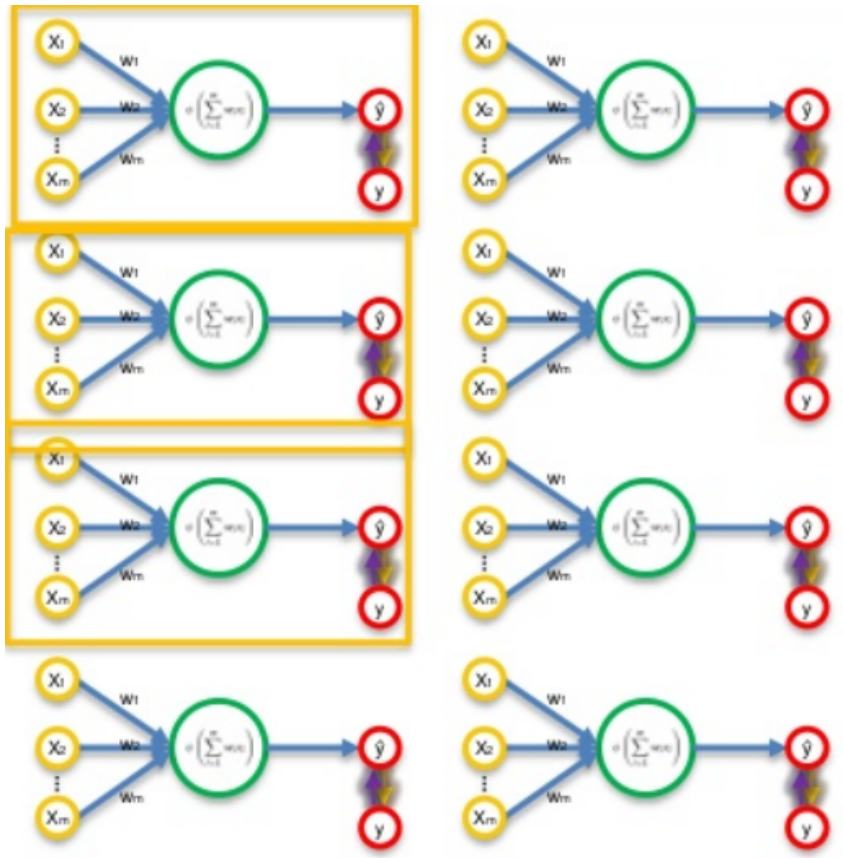


Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$



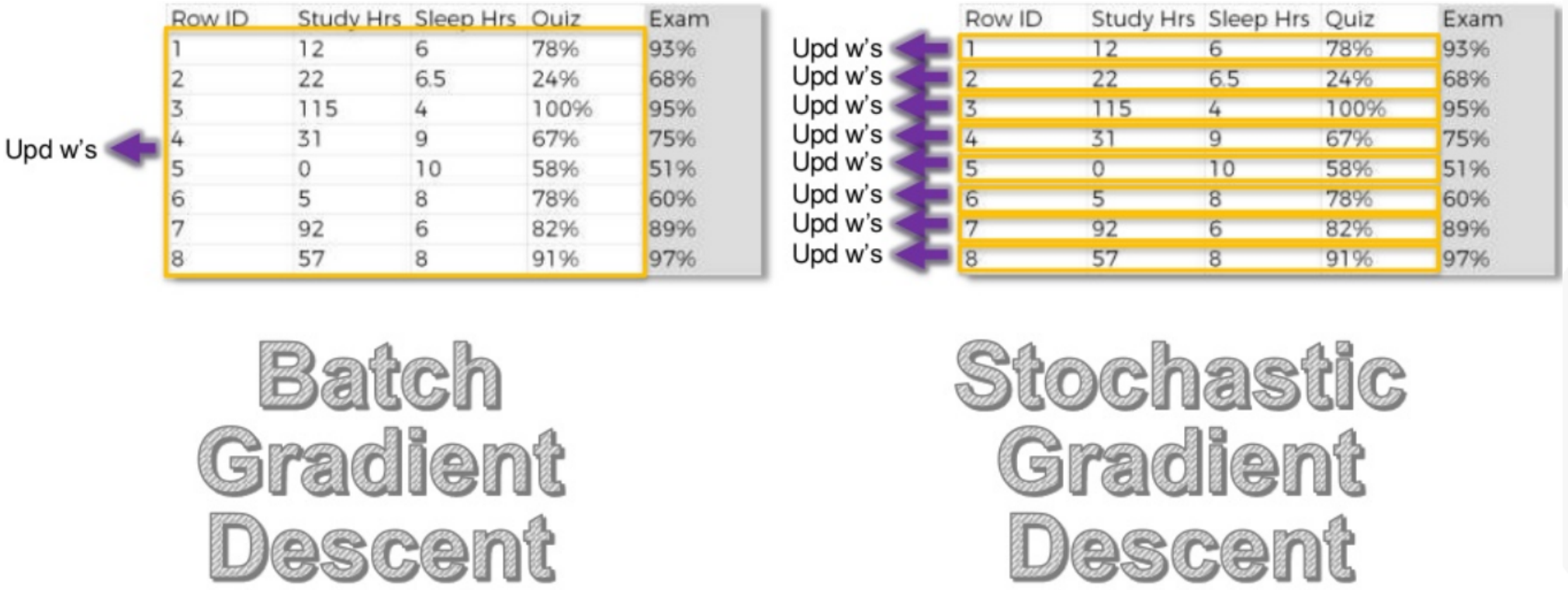
Stochastic Gradient Descent



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

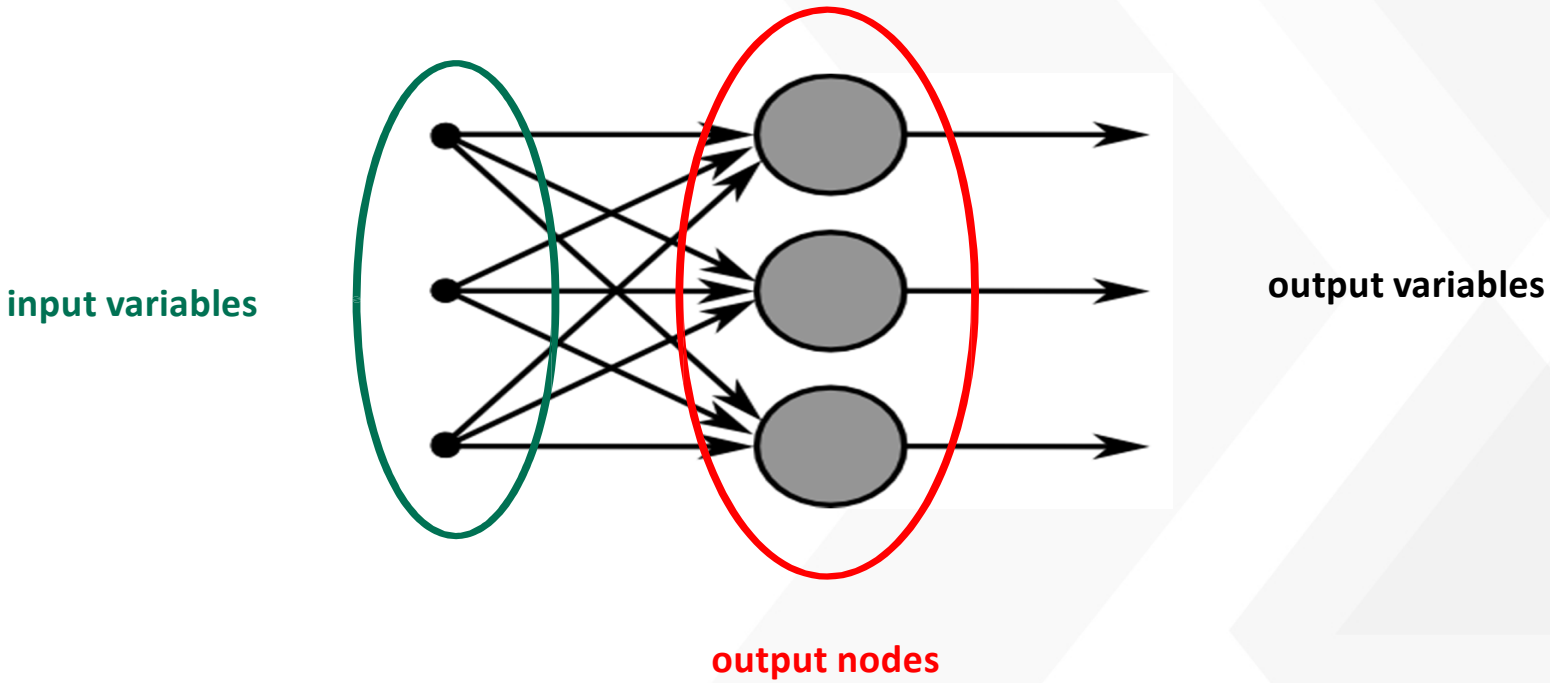
$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$

Gradient Descent vs. Stochastic Gradient Descent



Single Layer Perceptron

Are the simplest form of neural networks



Single layer perceptron: training rule

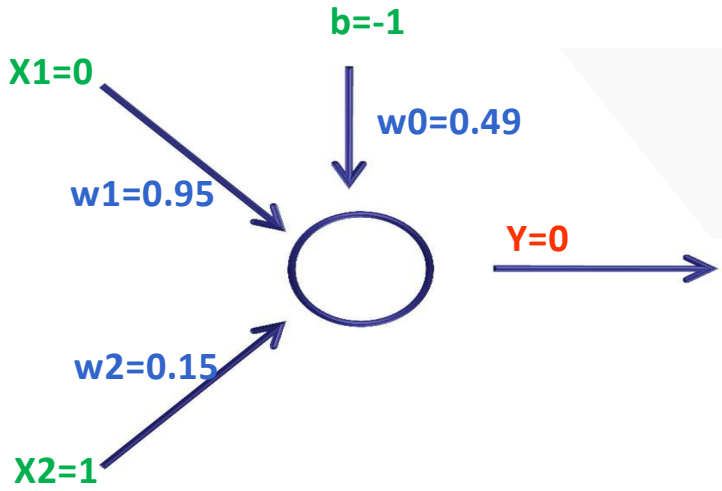
- Modify the *weights* (w_i) according to the *Training Rule:*

$$w_i = w_i + r \cdot (t - a) \cdot x_i$$

- r is the *learning rate* (eg. 0.2)
- t = target output
- a = actual output
- x_i = i-th input value

Learning rate: if too small learning occurs at a small pace, if too large it may stuck in local minimum in the decision space

Example



x1	x2	Y
0	0	0
1	0	1
0	1	1
1	1	1

threshold = 0.5
r=0.05

Compute output for the input

u = -1 x 0.49 + 0 x 0.95 + 1 x 0.15=-0.34 < t
thus, y=0

Compute the error

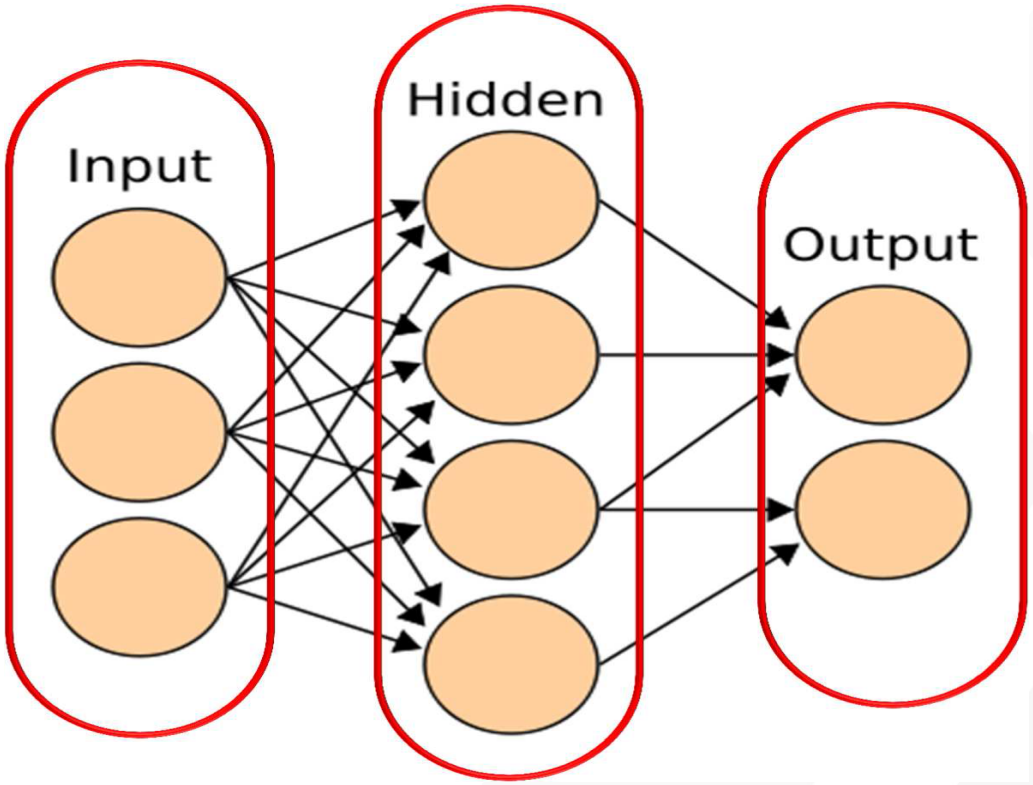
target output = 1
actual output (y) = 0
error = (1-0) = 1
correction factor = error x r = 0.05

Compute the new weights

w0 = 0.49 + 0.05 x (1-0) x (-1) = 0.44
w1 = 0.95 + 0.05 x (1-0) x 0 = 0.95
w2 = 0.15 + 0.05 x (1-0) x 1 = 0.20

Repeat the process with the new weigths for a given number of iterations

Multi layer network



input layer

hidden layer
(one or more)

output layer

Training multi layer networks

STEP 1: Randomly initialise the weights to small numbers close to 0 (but not 0).

STEP 2: Input the first observation of your dataset in the input layer, each feature in one input node.

STEP 3: Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result y.

STEP 4: Compare the predicted result to the actual result. Measure the generated error.

STEP 5: Back-Propagation: from right to left, the error is back-propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.

STEP 6: Repeat Steps 1 to 5 and update the weights after each observation (Reinforcement Learning). Or:
Repeat Steps 1 to 5 but update the weights only after a batch of observations (Batch Learning).

STEP 7: When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.

Multi-Layer network of sigmoid units

Problem: what is the desired output for a hidden node? => Backpropagation algorithm

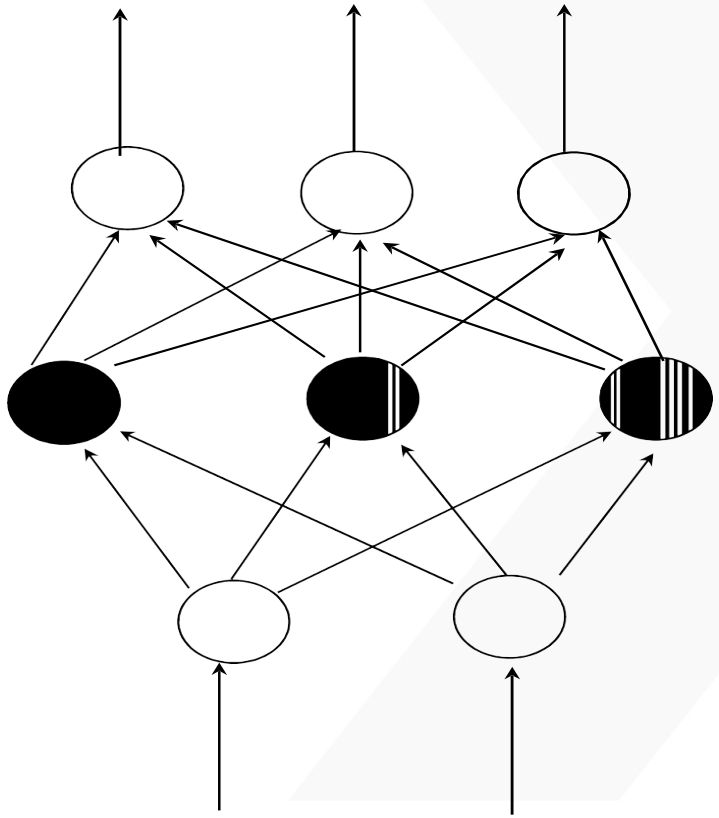
Output vector

Output nodes

Hidden nodes

Input nodes

Input vector: x_i



$$\theta_j = \theta_j + (r)Err_j$$

to update the bias

$$w_{ij} = w_{ij} + (r)Err_j O_i$$

to update the weights

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

error for a node in the hidden layer

$$Err_j = O_j(1-O_j)(T_j - O_j)$$

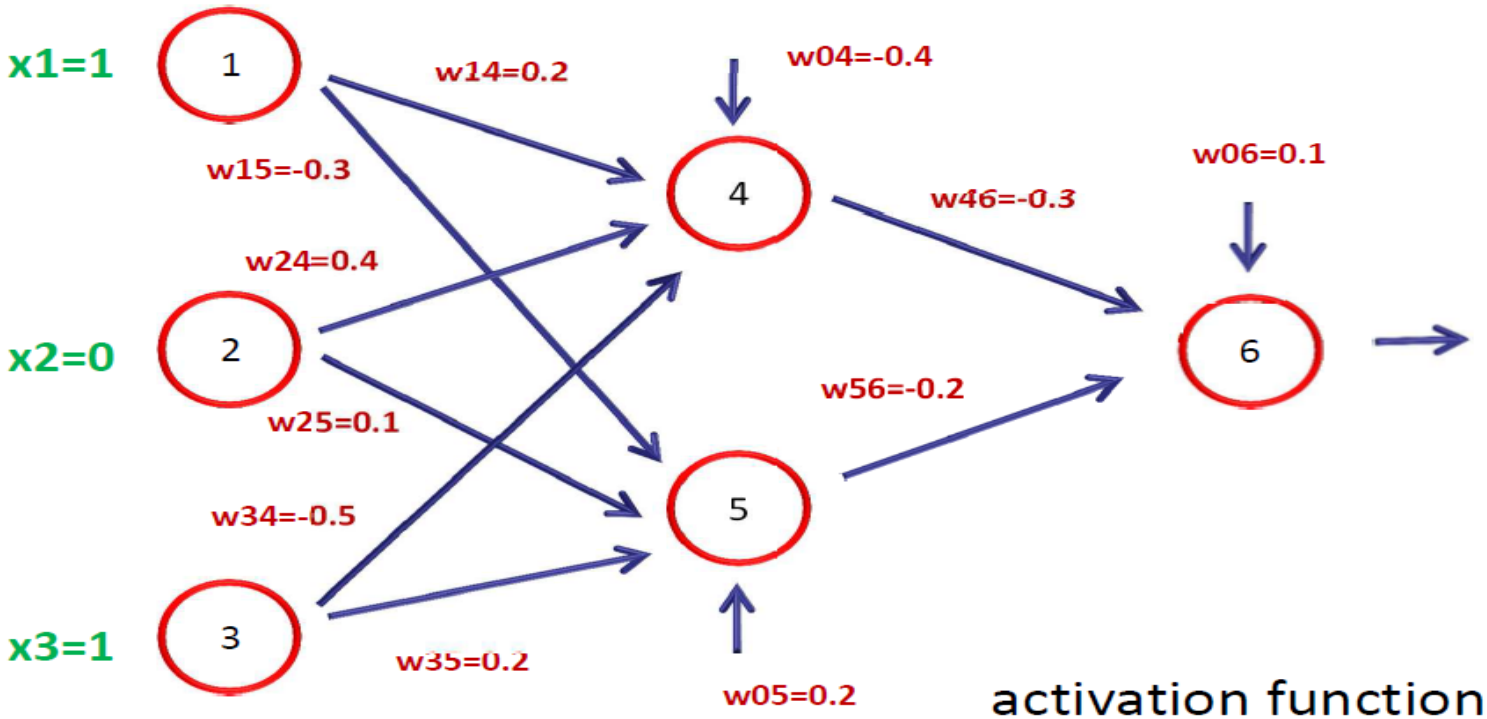
error for a node in the output layer

$$O_j = \frac{1}{1+e^{-I_j}}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Example1 – Forward propagation

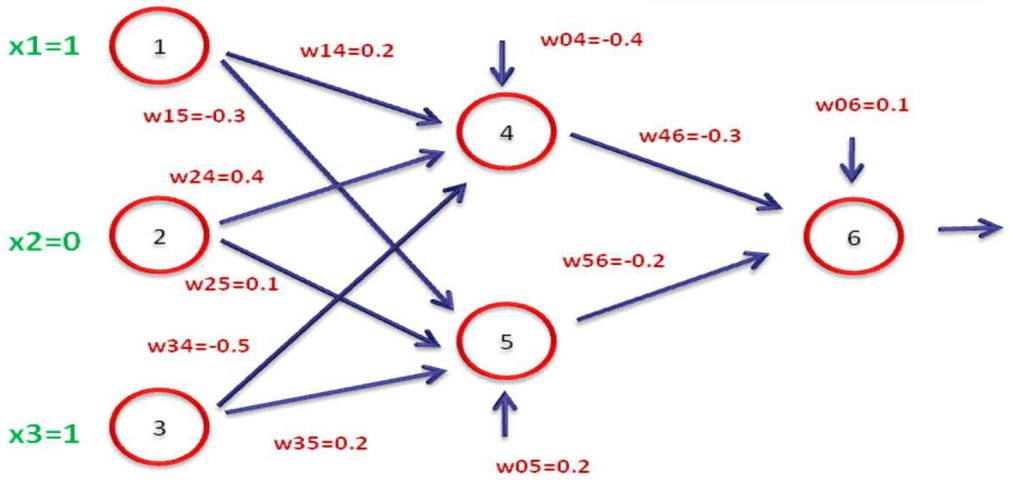
Example



xi – input variables (1,0,1) whose class is 1
wij – randomly assigned weights

activation function
$Oj = 1 / (1+e^{-Ij})$
and
learning rate = 0.9

Another Example - Propagation

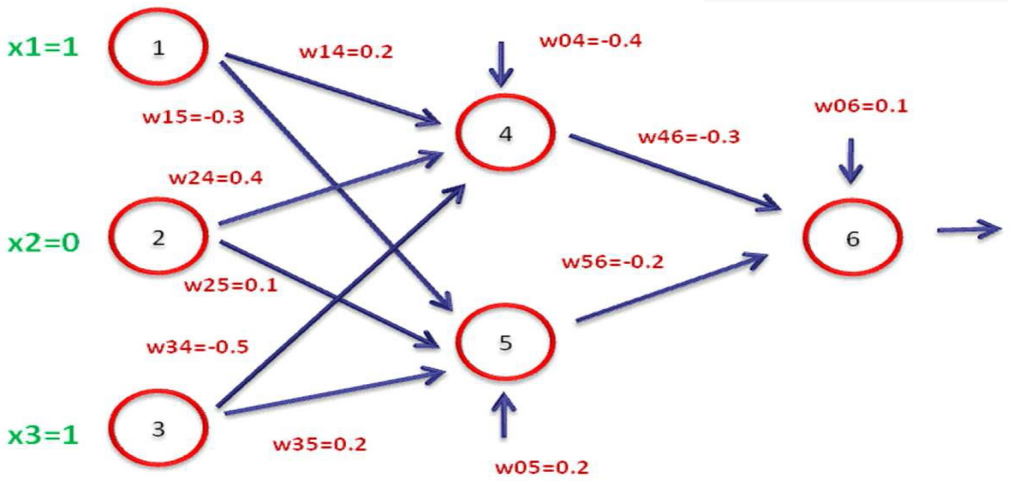


$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

neuron	input	output
4	0.2x1+0.4x0-0.5x1-0.4=-0.7	$1/(1+e^{0.7})=0.332$
5	-0.3x1+0.1x0+0.2x1+0.2=0.1	$1/(1+e^{-0.1})=0.525$
6	-0.3x0.332-0.2x0.525+0.1=-0.105	$1/(1+e^{0.105})=0.474$

Example - Calculation of the neuron



error for a node in the output layer

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

error for a node in the hidden layer

$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$$

neuron	output
4	0.332
5	0.525
6	0.474

neuron	error
6	0.474 x (1 - 0.474) x (1 - 0.474) = 0.1311
5	0.525 x (1 - 0.525) x (-0.2) x 0.1311 = -0.0065
4	0.332 x (1 – 0.332) x (-0.3) x 0.1311 = -0.0087

Example - Updating weights

$\theta_j = \theta_j + (r)Err_j$

to update the bias

$w_{ij} = w_{ij} + (r)Err_jO_i$

to update the weights

$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$

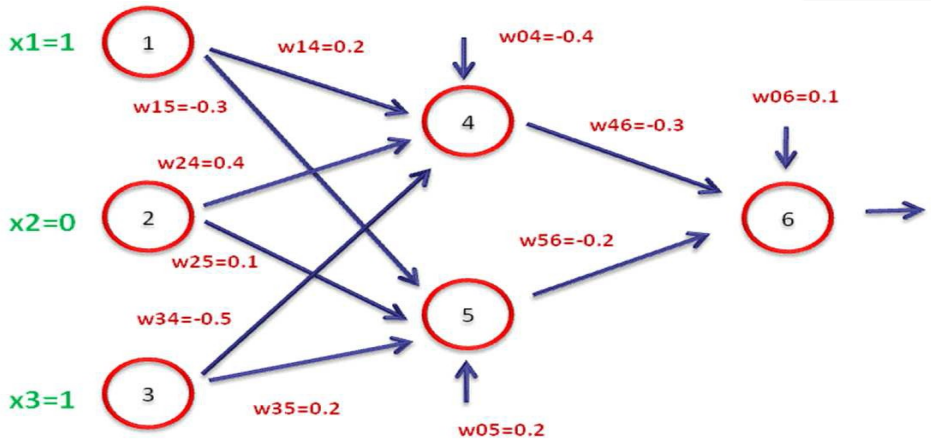
error for a node in the hidden layer

$Err_j = O_j(1-O_j)(T_j - O_j)$

error for a node in the output layer

$O_j = \frac{1}{1+e^{-I_j}}$

$I_j = \sum_i w_{ij}O_i + \theta_j$



to update the weights

$w_{ij} = w_{ij} + (r)Err_jO_i$

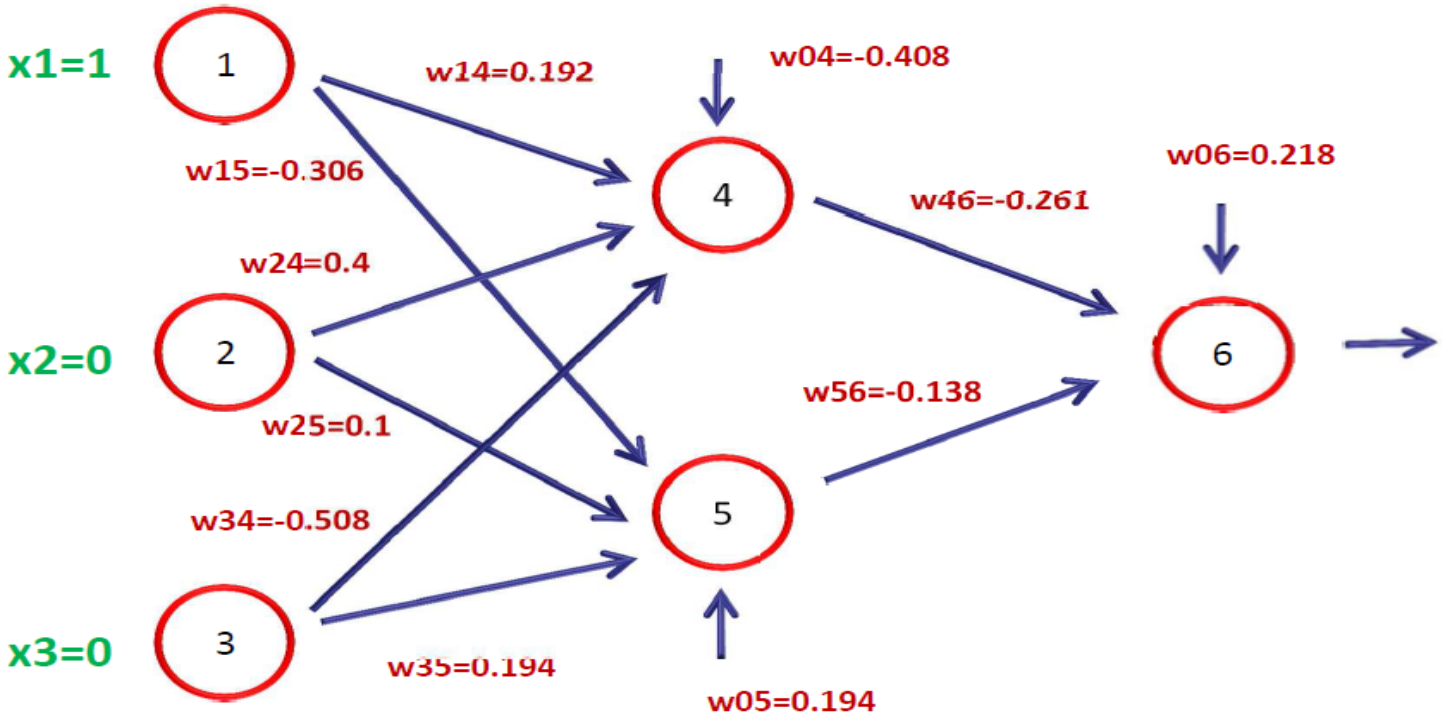
to update the bias

$\theta_j = \theta_j + (r)Err_j$

neuron	output	error
4	0.332	-0.0087
5	0.525	-0.0065
6	0.474	0.1311

weight	New value
w46	-0.3 + 0.9 x 0.1311 x 0.332 = -0.261
w56	-0.2 + 0.9 x 0.1311 x 0.525 = -0.138
w14	0.2 + 0.9 x -0.0087 x 1 = 0.192
w15	-0.3 + 0.9 x -0.0065 x 1 = -0.306
w24	0.4 + 0.9 x -0.0087 x 0 = 0.4
w25	0.1 + 0.9 x -0.0065 x 0 = 0.1
w34	-0.5 + 0.9 x -0.0087 x 1 = -0.508
w35	0.2 + 0.9 x -0.0065 x 1 = 0.194
w06	0.1 + 0.9 x 0.1311 = 0.218
w05	0.2 + 0.9 x -0.0065 = 0.194
w04	-0.4 + 0.9 x -0.0087 = -0.408

Example after 1st iteration

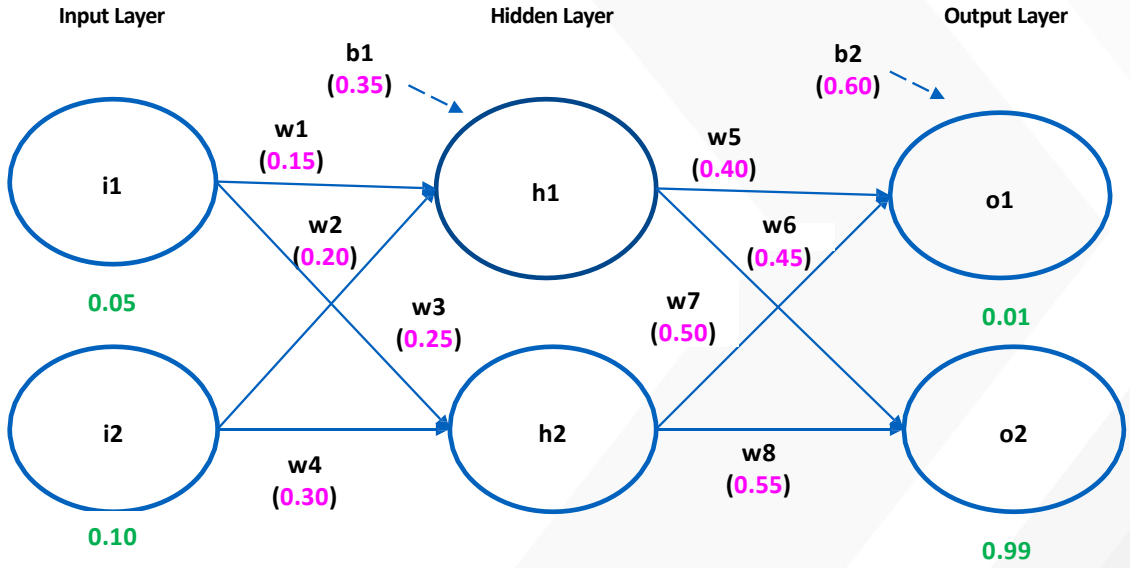


This is the resulting network after the first iteration. We now have to process another training example until the overall error is low or we run out of examples.

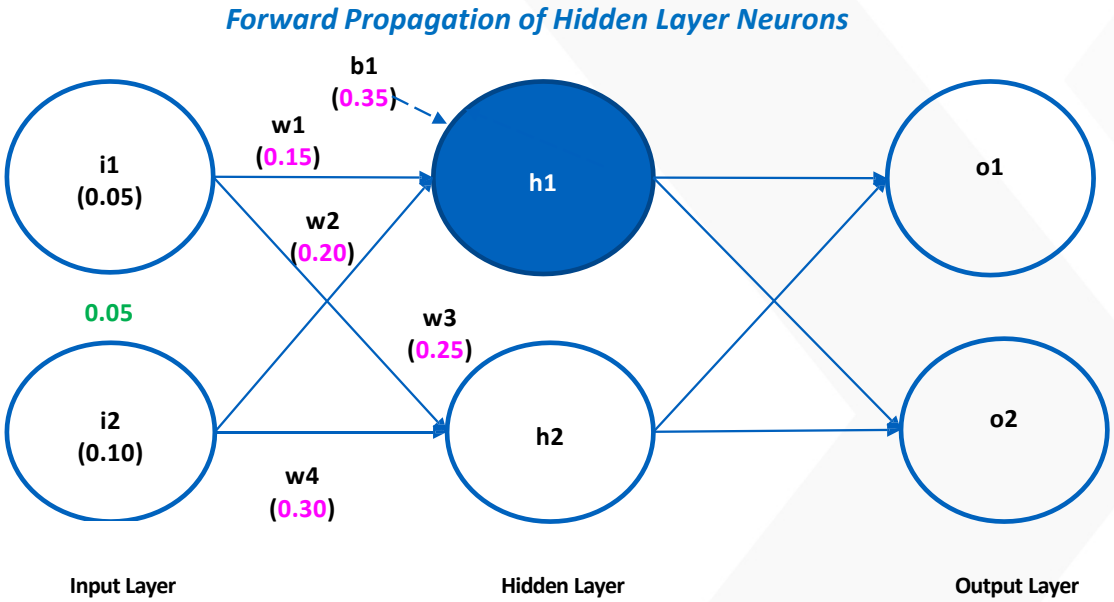
Example2 – Backward propagation

Basics - Forward propagation

Features are inputs to the network and feed through the subsequent layers to produce the outputs.



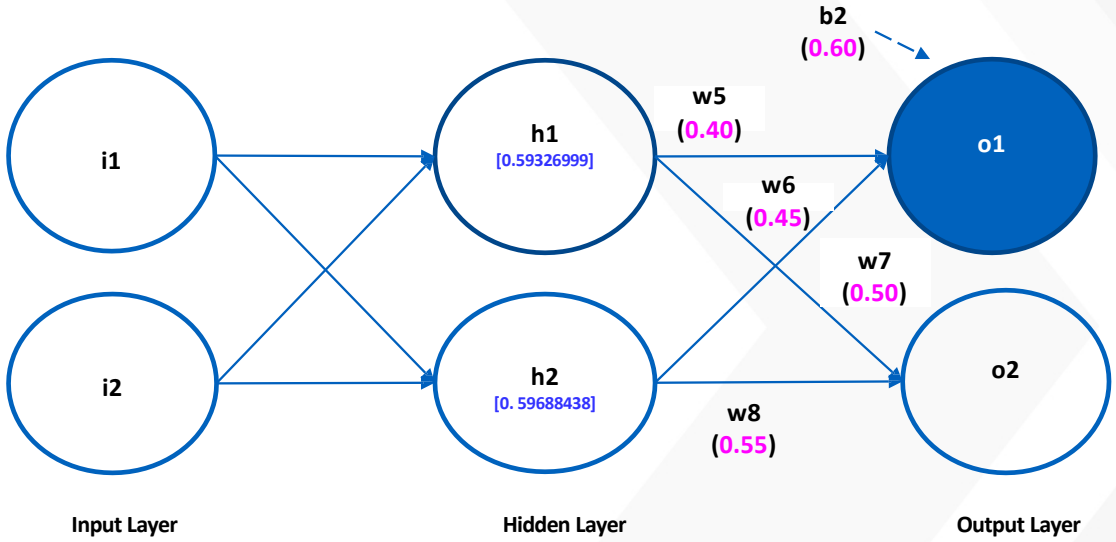
Basics - Forward propagation



Net input for h1: net h1 = (w1 * i1 + w2 * i2 + b1 * 1)
Net input for h1: net h1 = (0.15 * 0.05 + 0.20 * 0.10 + 0.35 * 1) = 0.3775
Output for h1 after activation (say, sigmoid) function: out h1 = 1 / (1 + e^-0.3775) = 0.59326999
Carrying out the same process for h2 we get : out h2 = 0. 59688438

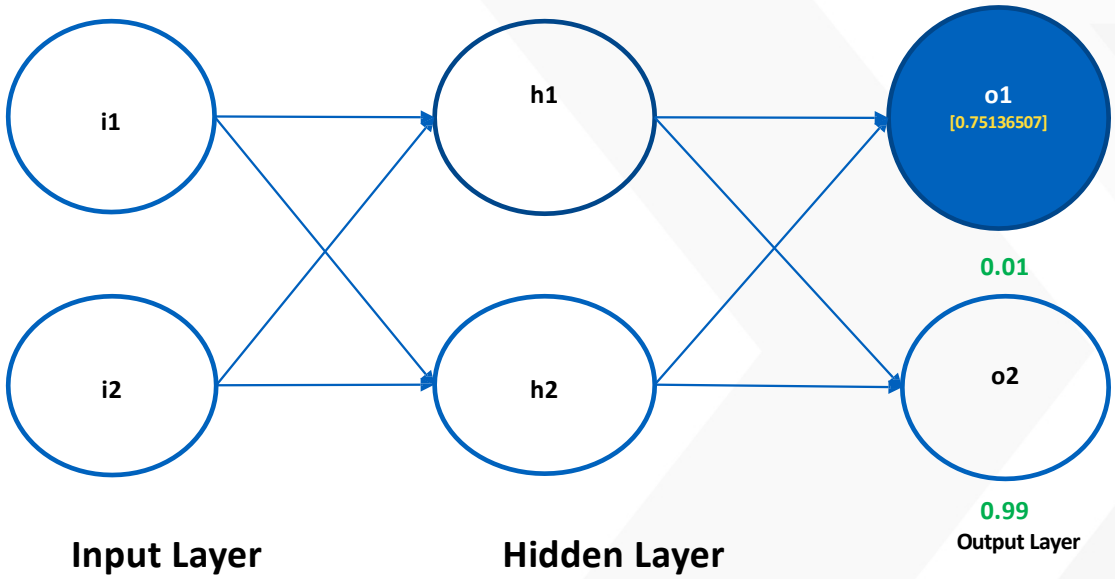
Basics - Forward propagation

Forward Propagation of Output Layer Neurons



Output for o1: net o1 = (w5 * out h1 + w6 * out h2 + b2 * 1)
net o1 = (0.40 * 0.59326999 + 0.45 * 0.596884378 + 0.60 *1) = 1.105905967
Output for o1 after activation (sigmoid) function: out o1 = 1 / (1+e^-1.105905967) = 0.75136507
Carrying out the same process for o2, we get out o2 : 0. 772928465

Basics - Error Calculation



Error for each output neuron by using, say, squared error function
Eo1 = ½ [target o1 – out o1]^2 = ½ [0.01 - 0.75136507]^2 = 0.274811083
Similarly, Eo2 = 0.23560026
So total error E = Eo1 + Eo2 = 0.274811083 + 0.023560026 = 0.298371109

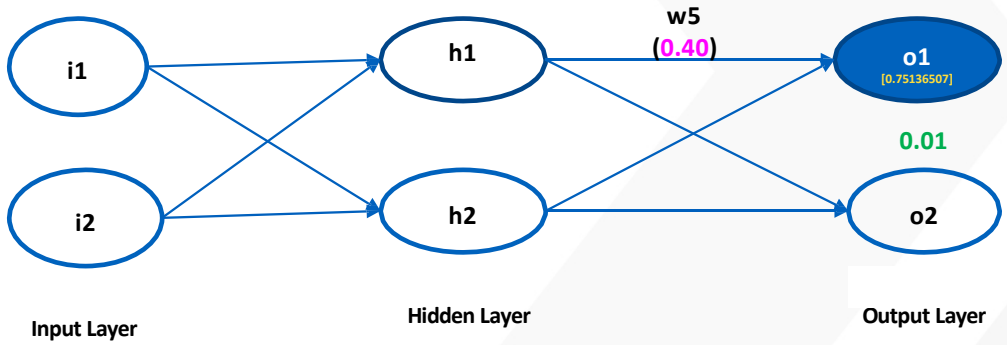
Basics - Back Propagation

To update the weights to optimal, we must propagate the network's errors backwards through its layers

Consider w5, we want to know how much a change in w5 affects the total error E = dE/dw5
By applying chain rule, dE/dw5 = (dE/dout o1) * (dout o1/dnet o1) * (dnet o1/dw5)

Backpropagation of Output Layer Neurons

How much does the total error E change w.r.t output out o1 = dE/dout o1

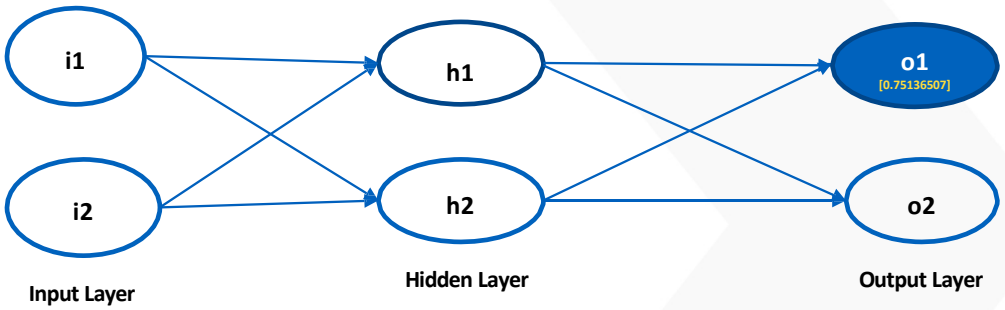


E = ½ [target o1 – out o1]^2 + ½ [target o2 – out o2]^2
So dE/dout o1 = 2 * ½ [target o1 – out o1] * (-1) = - (0.01-0.75136507) = 0.74136507

Basics - Back Propagation

Backpropagation of Output Layer Neurons

How much does the output out o1 change w.r.t its total net input net o1 = dout o1/dnet o1

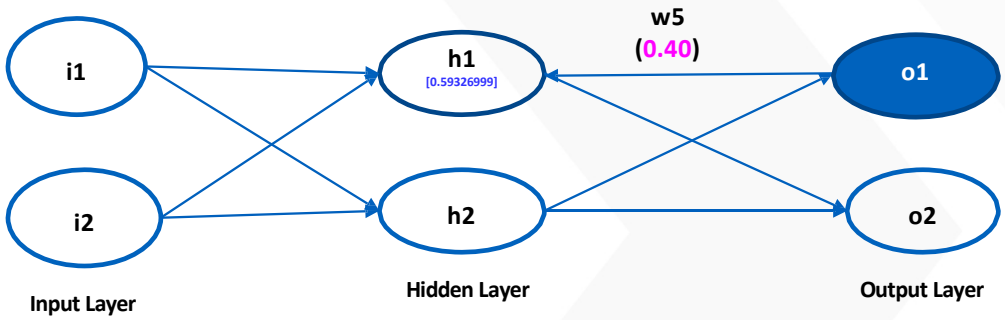


out o1 = 1 / (1 + e^-net o1)
So dout o1/dnet o1 = out o1*(1 - out o1) = 0.75136507 (1 - 0.75136507) = 0.186815602

Basics - Back Propagation

Backpropagation of Output Layer Neurons

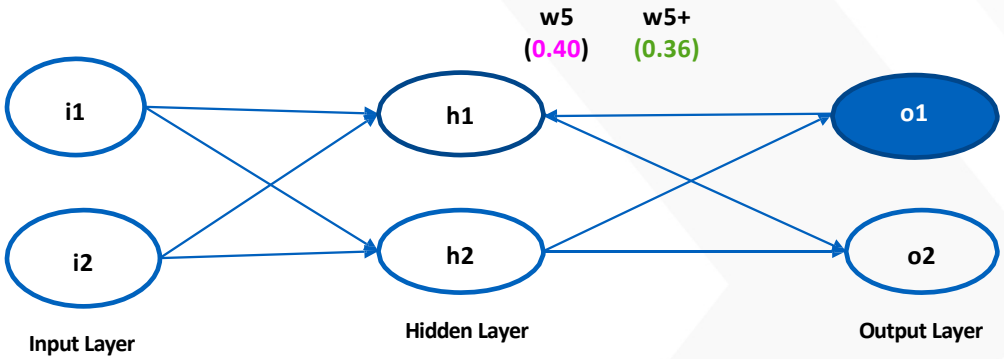
How much does the total net input net o1 change w.r.t w5 = dnet o1/dw5



net o1 = (w5 * out h1 + w6 * out h2 + b2 *1)
So dnet o1/dw5 = out h1 = 0.593269992

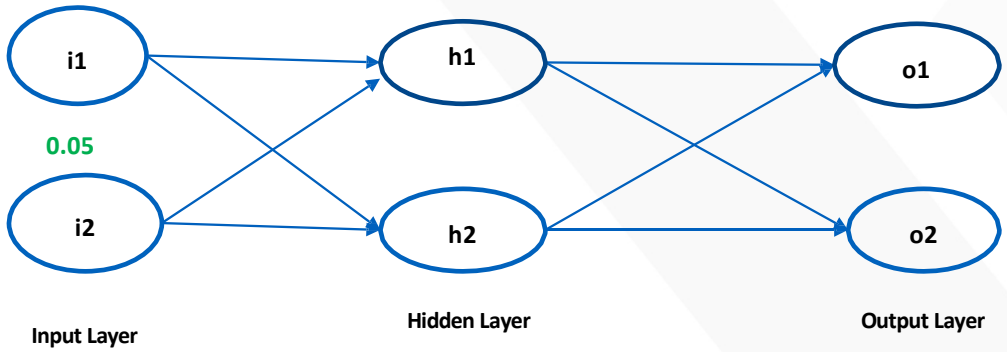
Basics - Weights Updation

Updating weights of Output Layer Neurons



Therefore, dE/dw5 = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041
Now to decrease the error we then subtract this value from the current weight
w5+ = w5 – (learning rate) * dE/dw5 = 0.4 – (0.5) * 0.082167041 = 0.35891648

Basics – After 1st Round of Back propagation



After repeating this process thousand times, the error comes to 0.0000351085. The two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

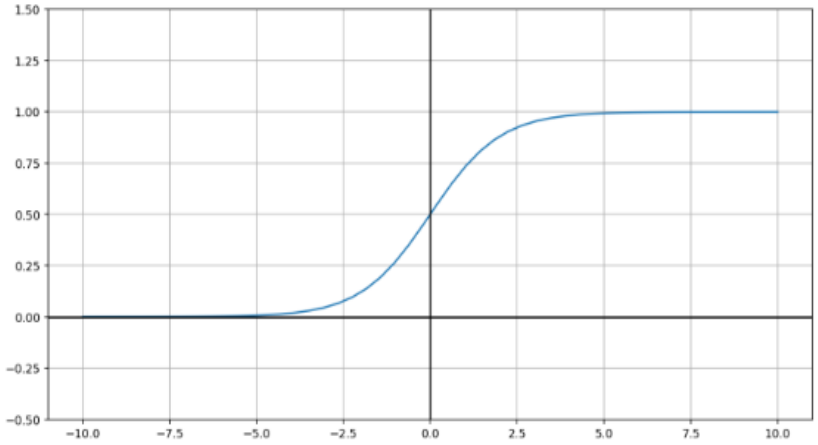
Optimization

Relation to Logistic Regression

When we choose: $f(z) = \frac{1}{1+e^{-z}}$

This is called the "sigmoid" function: $\sigma(z) = \frac{1}{1+e^{-z}}$

$$z = b + \sum_{i=1}^{m} x_i w_i = x_1 w_1 + x_2 w_2 + \cdots + x_m w_m + b$$



Then a neuron is simply a "unit" of logistic regression!

weights ⇔ coefficients inputs ⇔ variables

bias term ⇔ constant term

Nice property of sigmoid function

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

Quotient rule

$$\frac{d}{dx}\cdot\frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

$$\sigma'(z) = \frac{0-(-e^{-z})}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$ **This will be helpful!**

Neural Networks (Cost Function)

The (regularized) logistic regression cost function is as follows;

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

For neural networks our cost function is a generalization of this equation above, so instead of one output we generate *k* outputs

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Minimize Neural Network's Cost Function (Back Propagation)

Back propagation

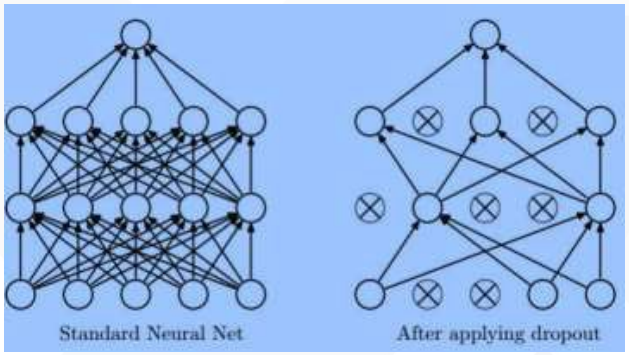
- ✓ Back propagation basically takes the output you got from your network, compares it to the real value (y) and calculates how wrong the network was (i.e. how wrong the parameters were)
- ✓ It then, using the error you've just calculated, back-calculates the error associated with each unit from the preceding layer (i.e. layer *L* - 1)
- ✓ This goes on until you reach the input layer (where obviously there is no error, as the activation is the input)
- ✓ These "error" measurements for each unit can be used to calculate the **partial derivatives**
- ✓ We use the partial derivatives with gradient descent to try minimize the cost function and update all the Θ values
- ✓ This repeats until gradient descent reports convergence

Basics – Dropout & Optimization

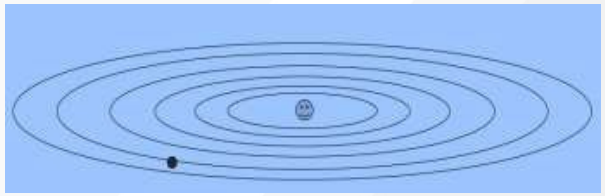
- **Dropout:** Regularization in Neural networks to avoid over fitting the data.

- **Optimization:** Various techniques used to optimize the weights including
  - SGD (Stochastic Gradient Descent)
  - Momentum
  - Nag (Nesterov Accelerated Gradient)
  - Adagrad (Adaptive gradient)
  - Adadelta
  - Rmsprop
  - Adam (Adaptive moment estimation)
  - Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)

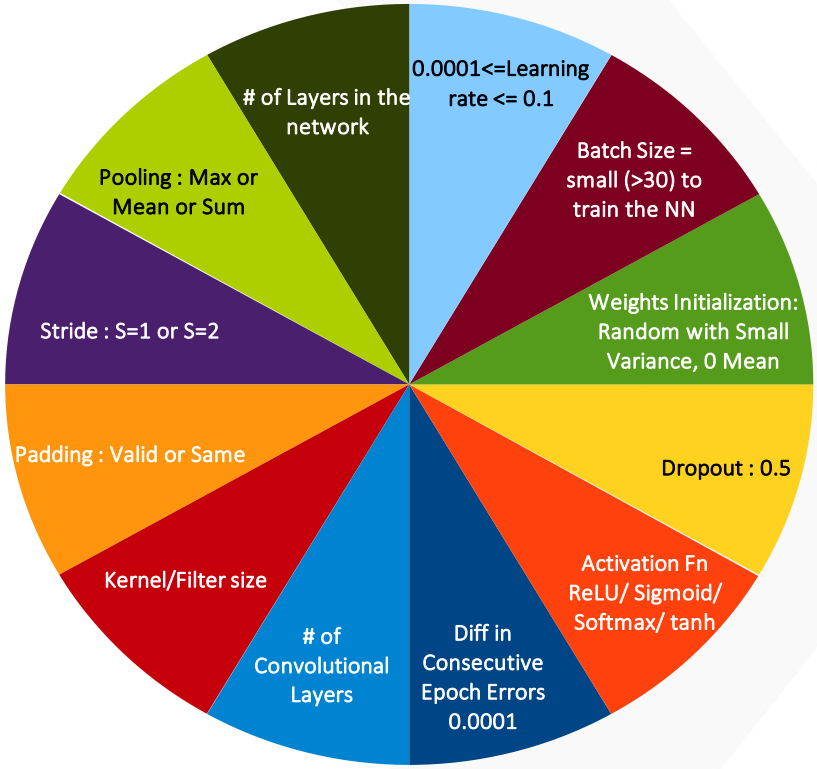
Application of Dropout in Neural network



Optimization of Error Surface

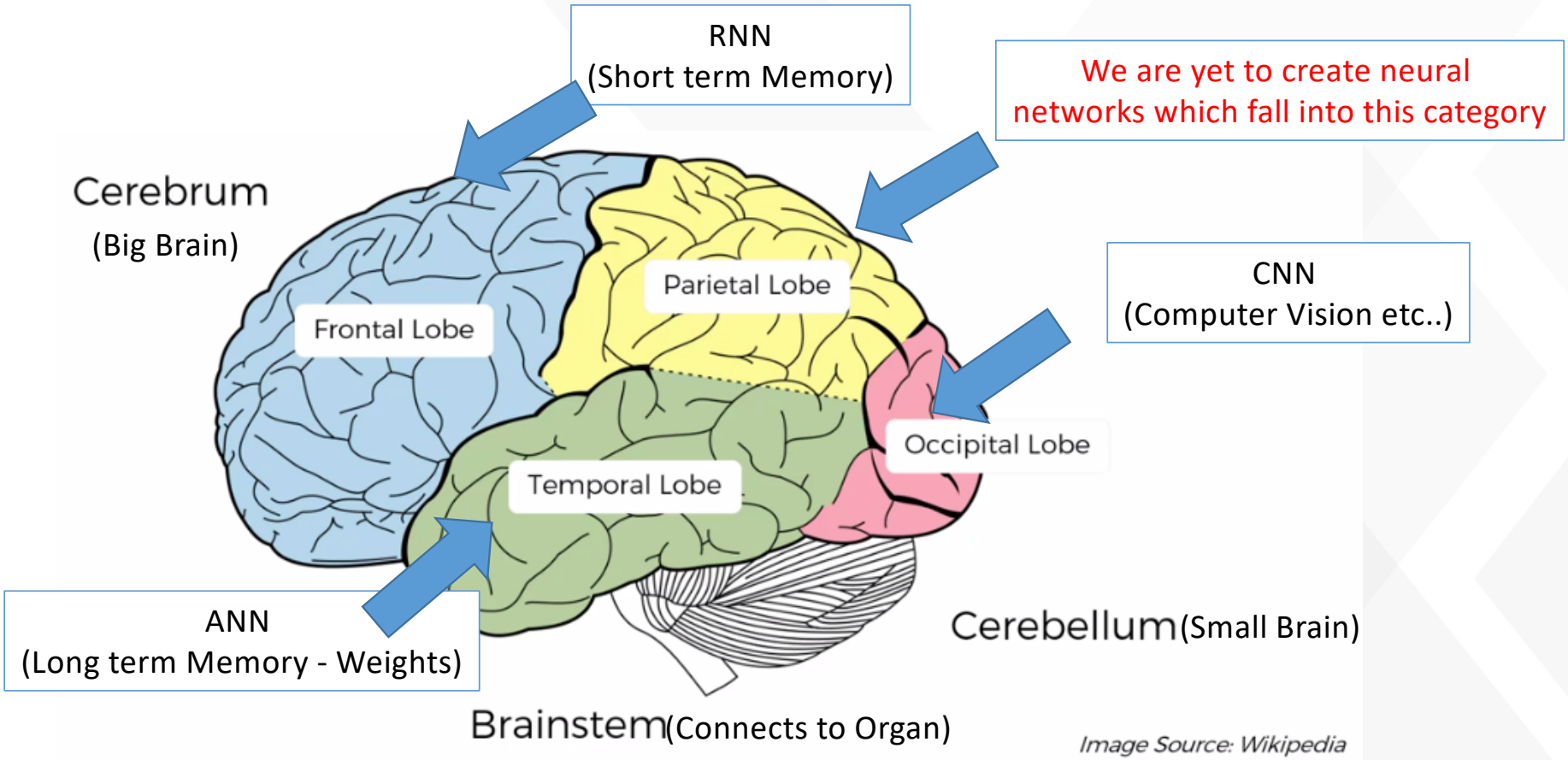


ANN/DL – Key parameters / Hyper parameters



Network Architectures

Human Brain Functions vs. Deep Learning Algorithms

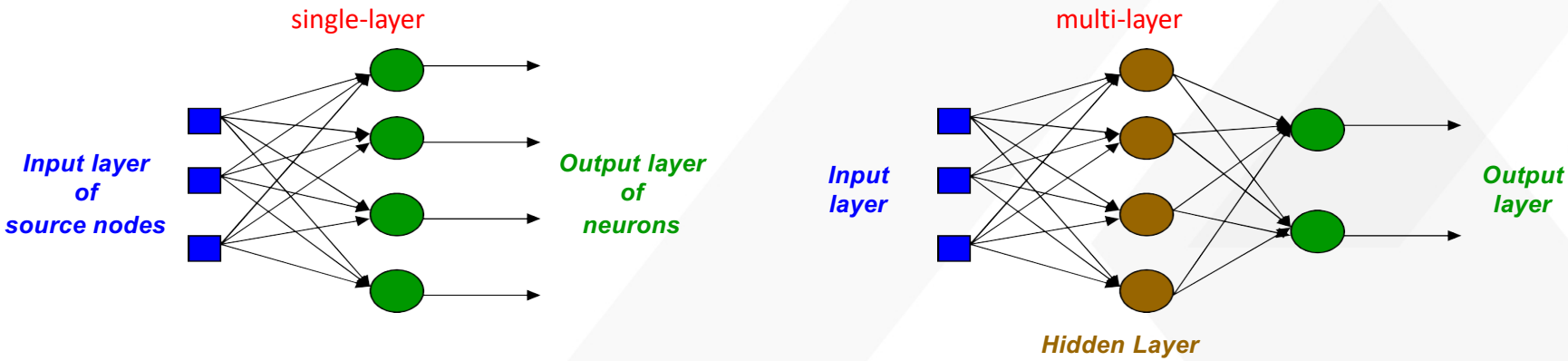


Network Architectures

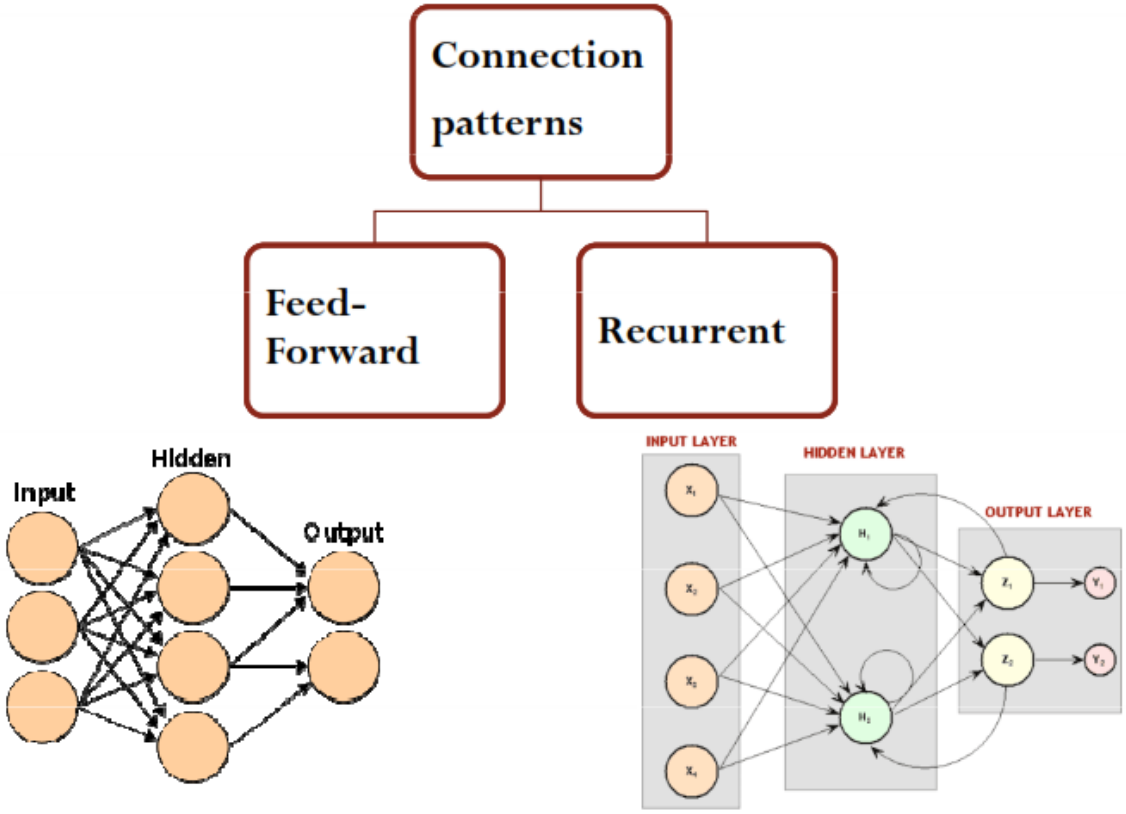
- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent

} *neurons are organized in acyclic layers*

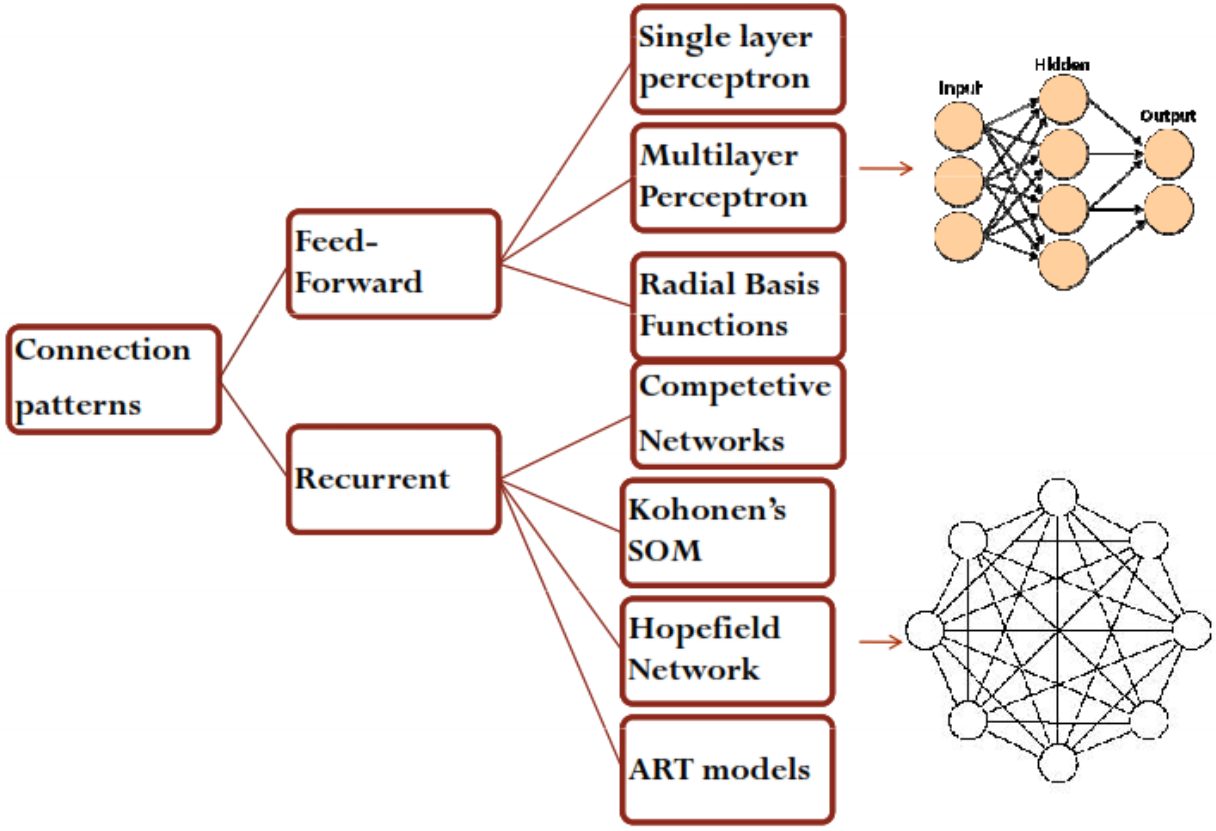
- The architecture of a neural network is linked with the learning algorithm used to train



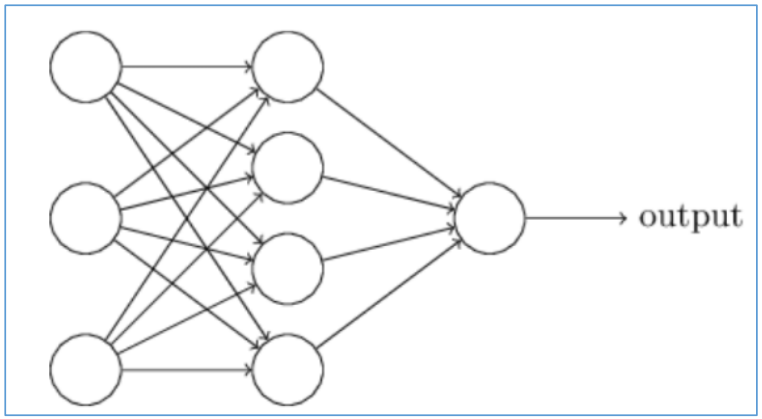
Network Architectures

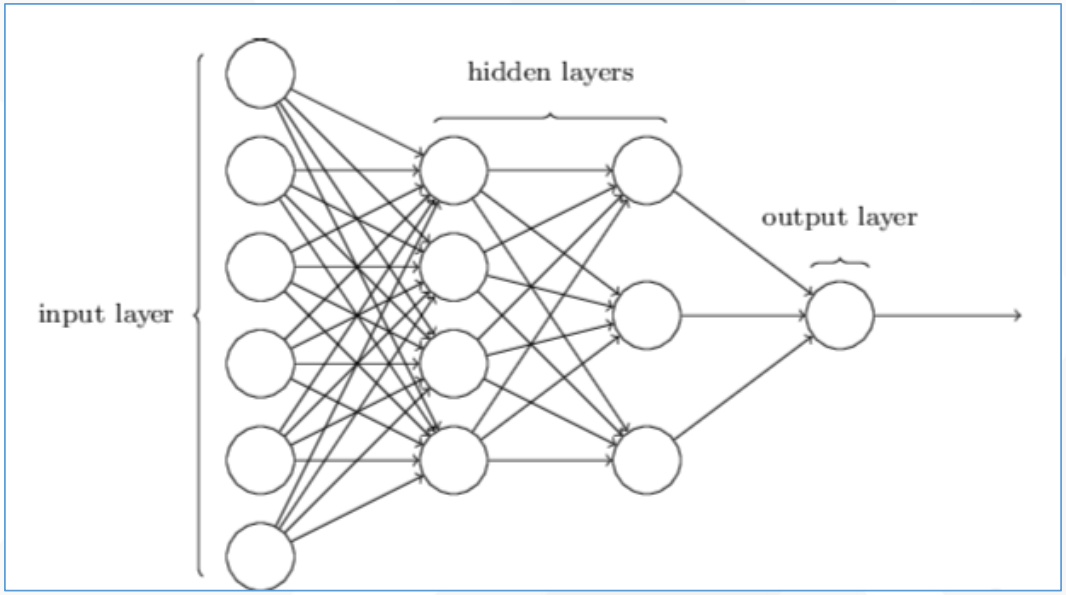


Network Architectures

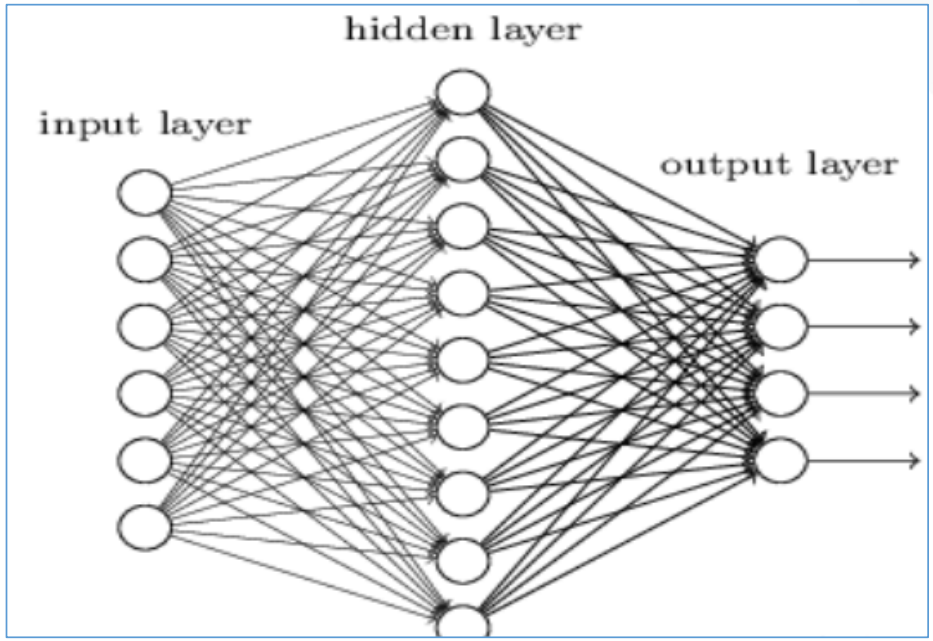


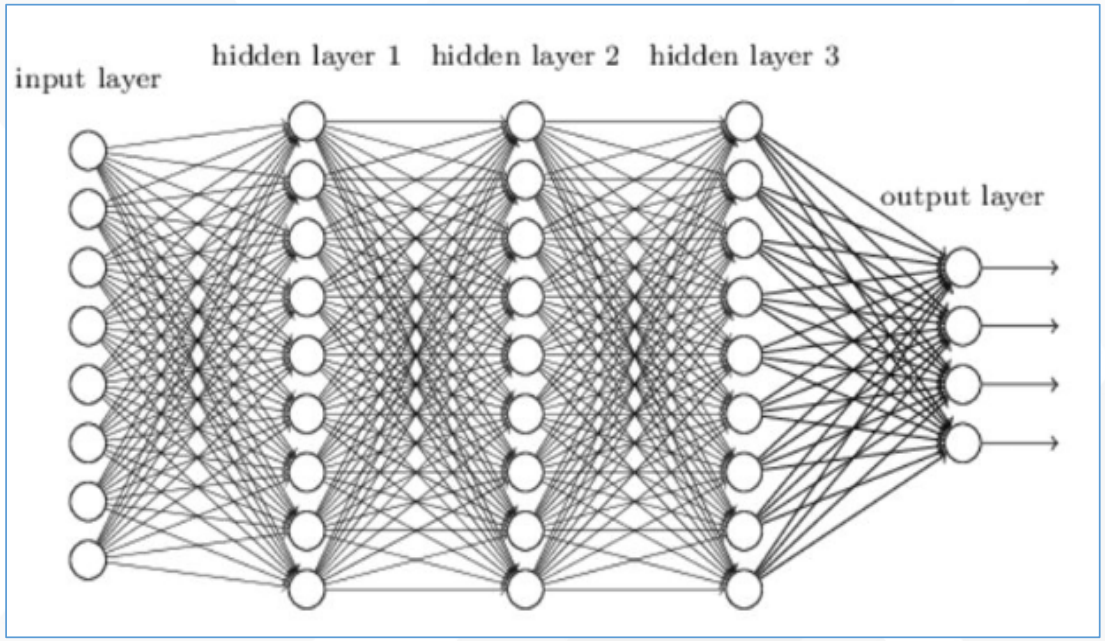
ANN – Sample Architectures



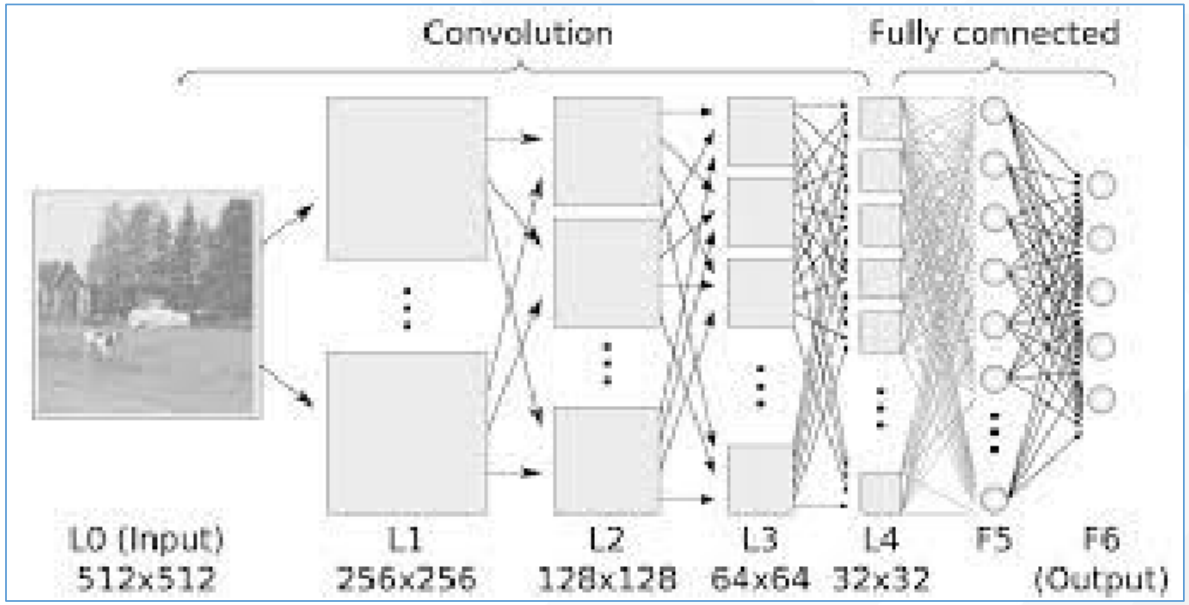


ANN – Sample Architectures

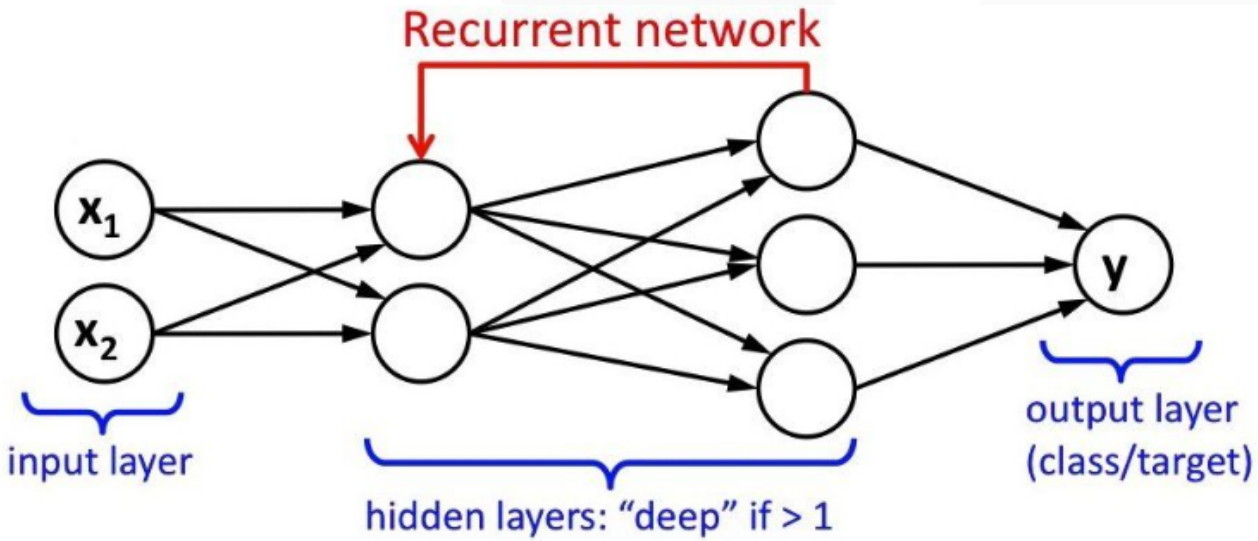




ANN – Sample Architectures



ANN – Sample Architectures



Contact Us

Visit us on: http://www.analytixlabs.in/

For more information, please contact us: http://www.analytixlabs.co.in/contact-us/

Or email: info@analytixlabs.co.in

Call us we would love to speak with you: (+91) 88021-73069

Join us on:

Twitter - http://twitter.com/#!/AnalytixLabs

Facebook - http://www.facebook.com/analytixlabs

LinkedIn - http://www.linkedin.com/in/analytixlabs

Blog - http://www.analytixlabs.co.in/category/blog/