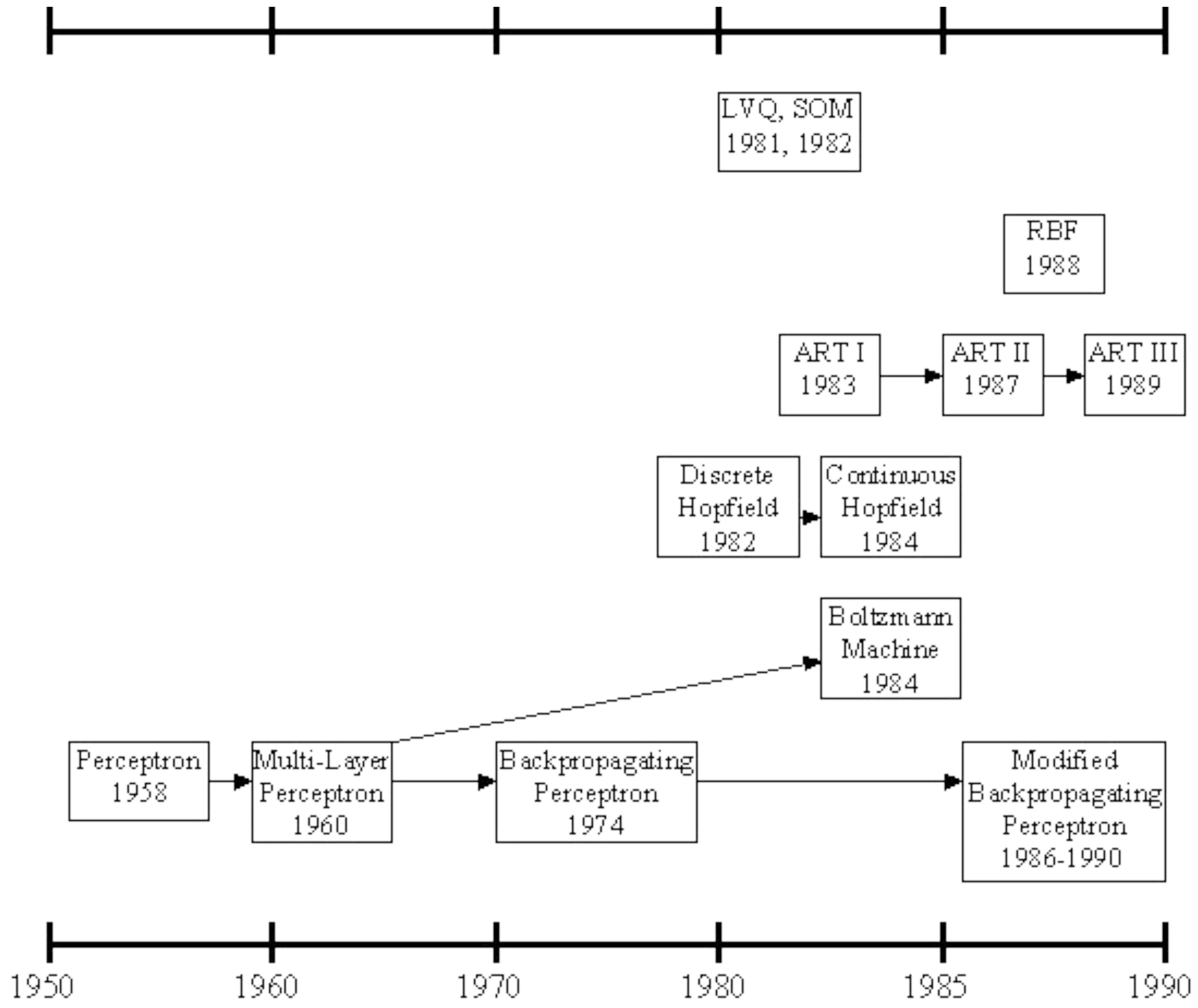
Artificial Neural Networks (ANN)

DR. RAIHAN UL ISLAM
ASSOCIATE PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
ROOM NO# 256
EMAIL: RAIHAN.ISLAM@EWUBD.EDU
MOBILE: +8801992392611

History of the Artificial Neural Networks

- History of the ANNs stems from the 1940s, the decade of the first electronic computer.
- However, the first important step took place in 1957 when Rosenblatt introduced the first concrete neural model, the perceptron. Rosenblatt also took part in constructing the first successful neurocomputer, the Mark I Perceptron. After this, the development of ANNs has proceeded as described in *Figure*.



Artificial Neural Network

An artificial neural network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections.

Why Artificial Neural Networks?

There are two basic reasons why we are interested in building artificial neural networks (ANNs):

Technical viewpoint: Some problems such as character recognition or the prediction of future states of a system require massively parallel and adaptive processing.

Biological viewpoint: ANNs can be used to replicate and simulate components of the human (or animal) brain, thereby giving us insight into natural information processing.

How do our brains work?

The Brain is A massively parallel information processing system.

Our brains are a huge network of processing elements. A typical brain contains a network of 10 billion neurons.

How do our brains work?

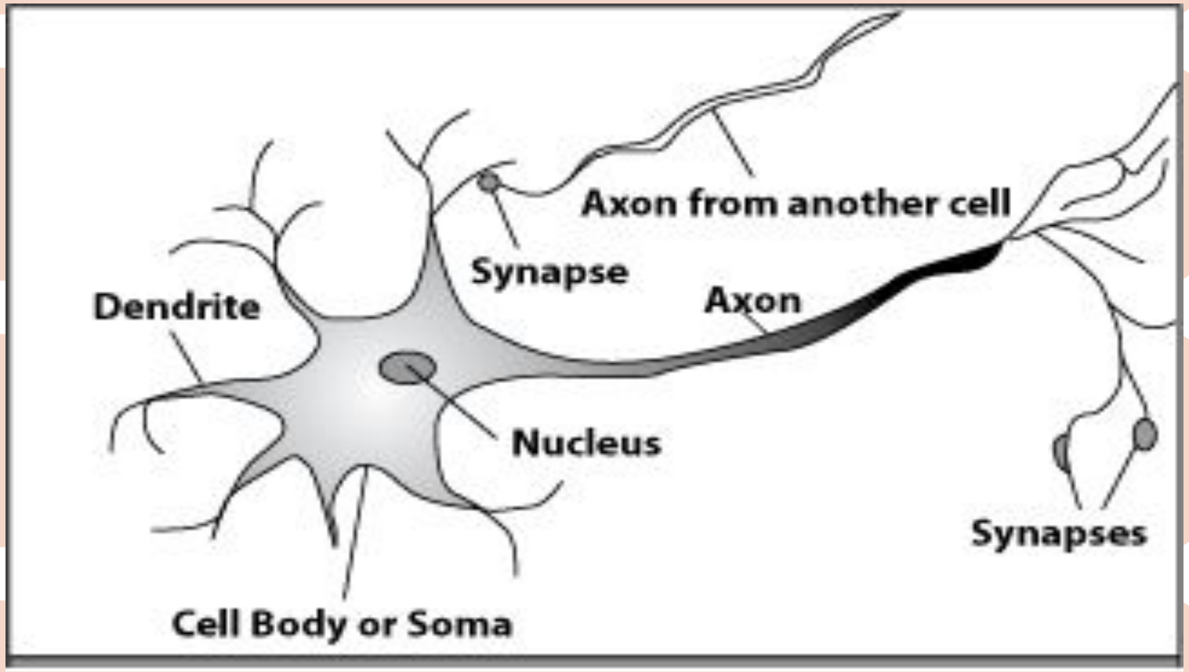
Dendrites: Input

Cell body: Processor

Synaptic: Link

Axon: Output

- A processing element



How do our brains work?

A neuron is connected to other neurons through about *10,000 synapses*

A neuron receives input from other neurons. Inputs are combined.

Once input exceeds a critical level, the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s)

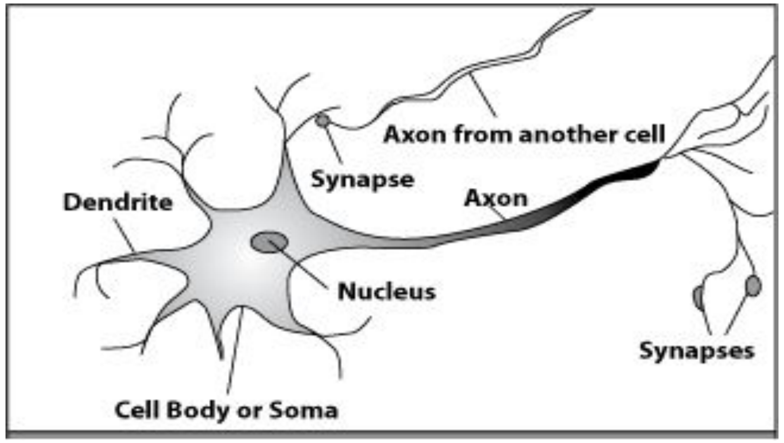
The axon endings almost touch the dendrites or cell body of the next neuron.

Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters.

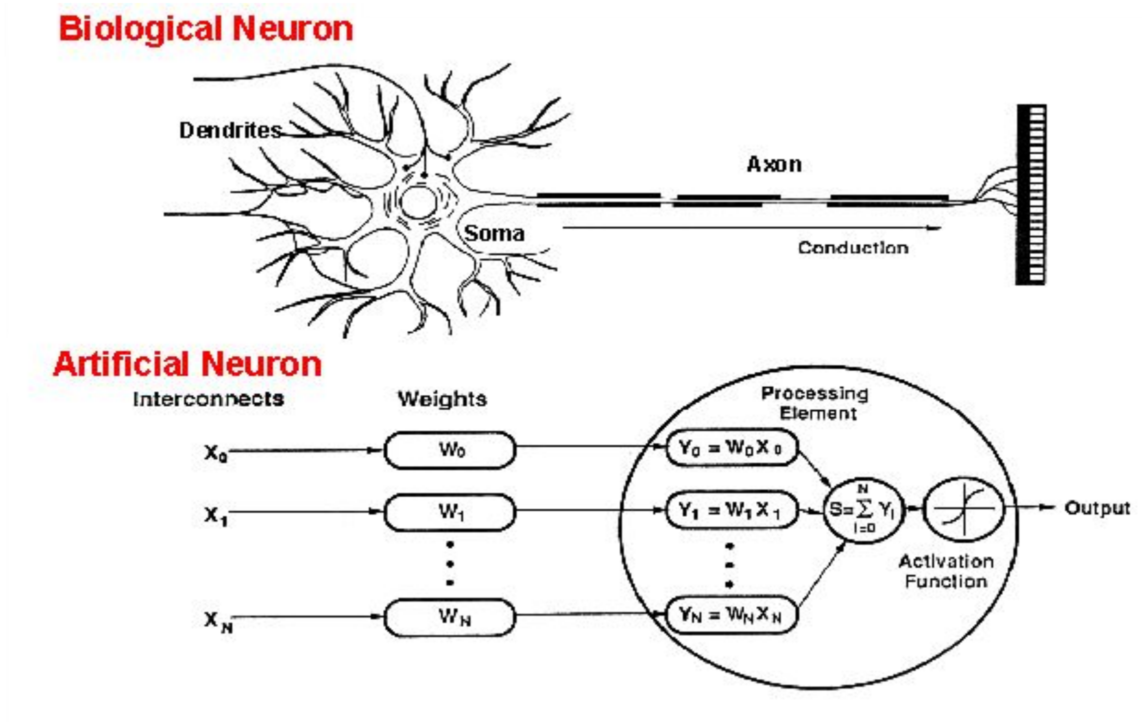
Neurotransmitters are chemicals which are released from the first neuron and which bind to the Second.

This link is called a synapse. The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

- A processing element



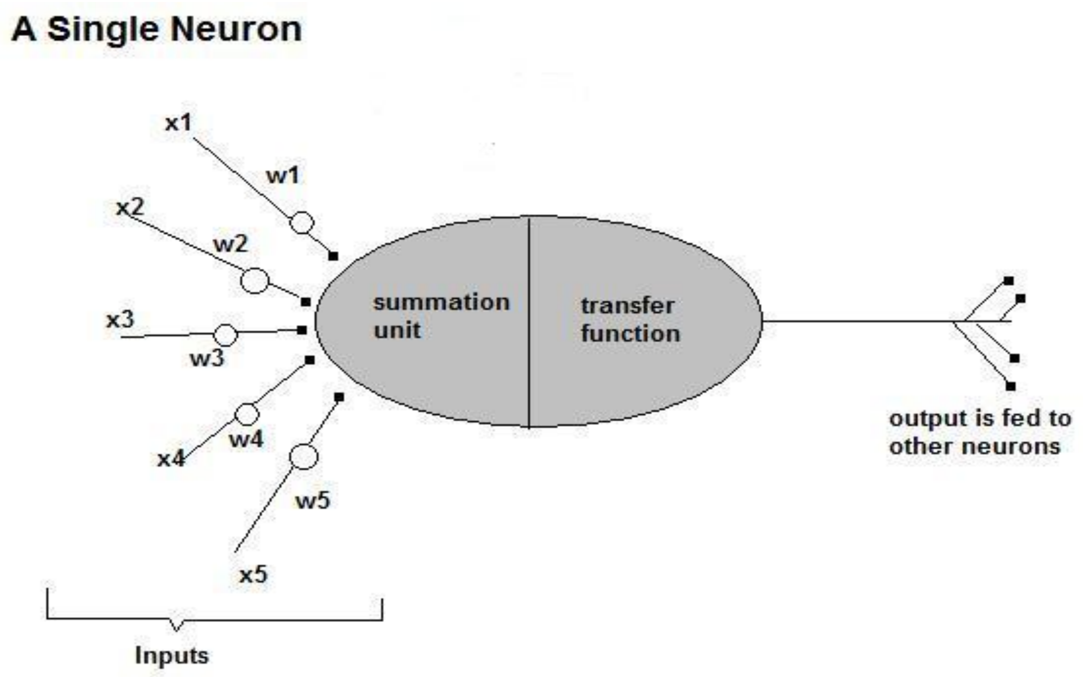
How do ANNs work?



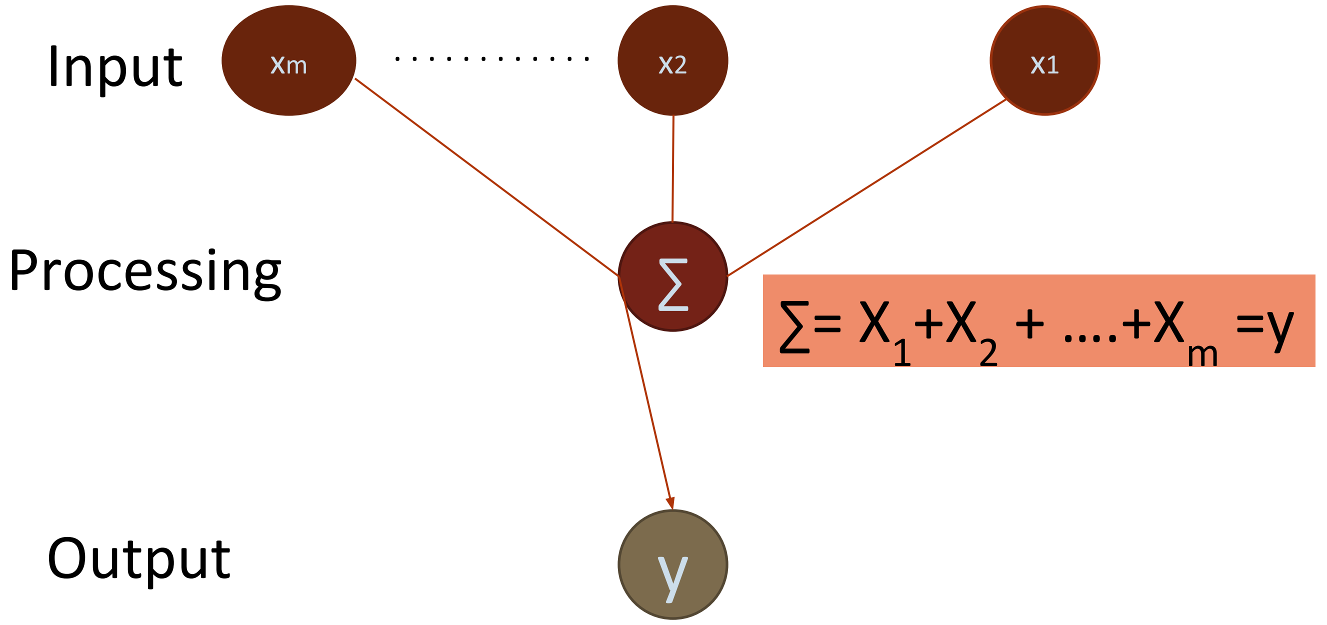
An artificial neuron is an imitation of a human neuron

How do ANNs work?

- Now, let us have a look at the model of an artificial neuron.



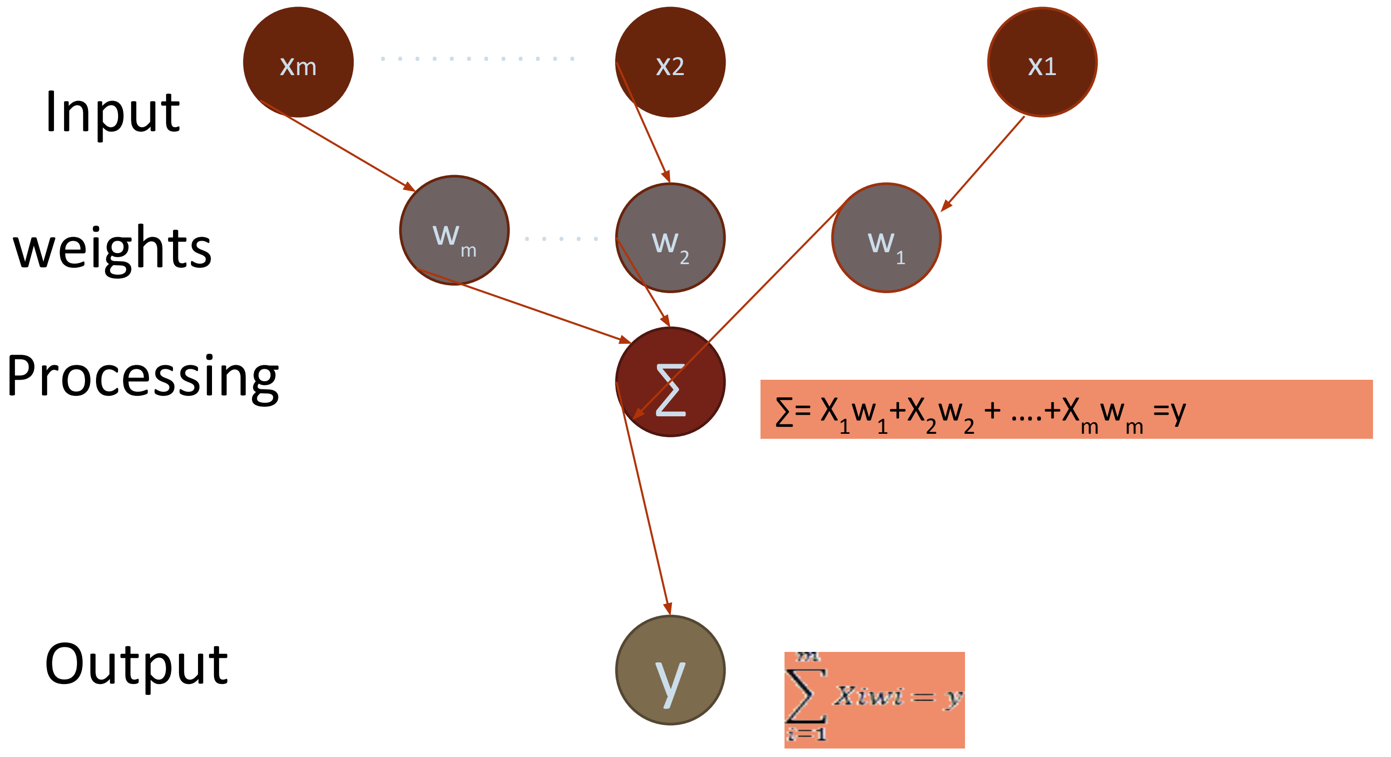
How do ANNs work?



$\sum = X_1 + X_2 + \ldots + X_m = y$

How do ANNs work?

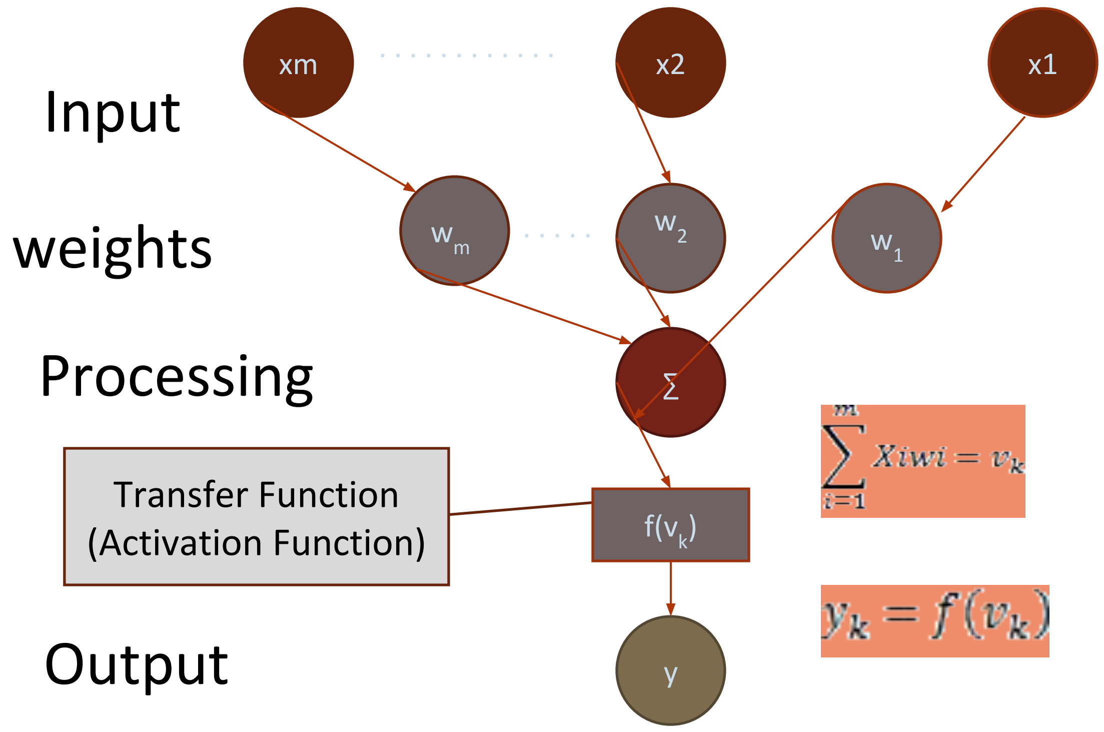
Not all inputs are equal



$$\sum = X_1w_1+X_2w_2 + \ldots + X_mw_m = y$$

$$\sum_{i=1}^{m} Xiwi = y$$

How do ANNs work?

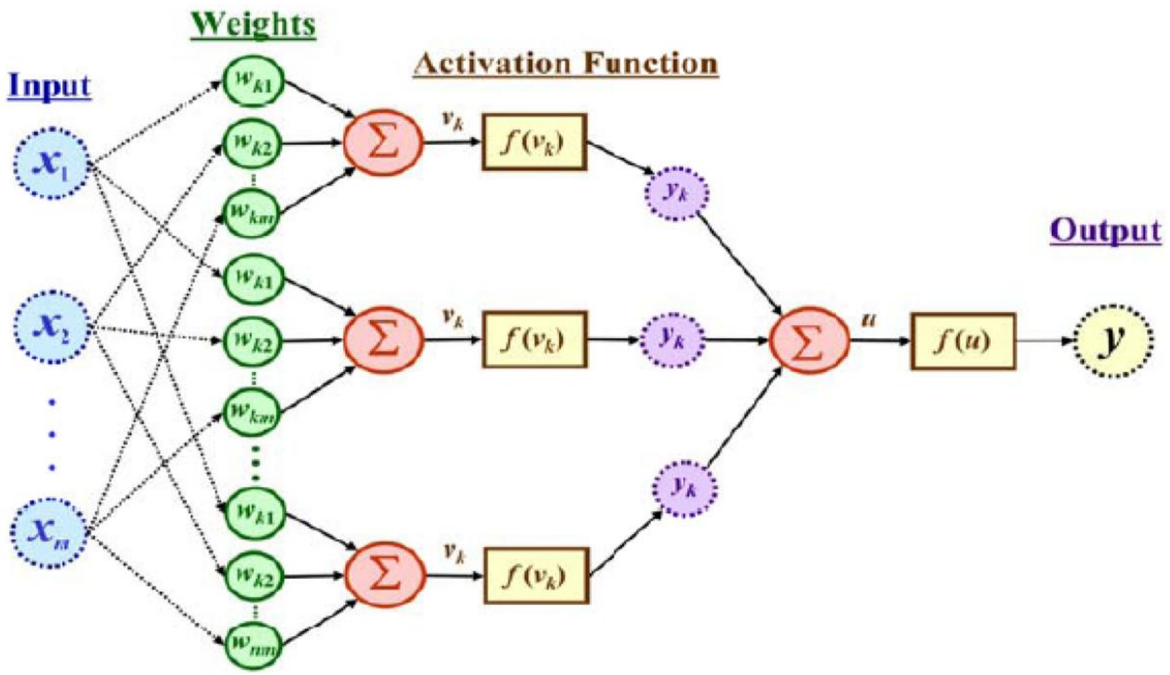


$$\sum_{i=1}^{m} Xiwi = v_k$$

$$y_k = f(v_k)$$

How do ANNs work?

The output is a function of the input, that is affected by the weights, and the transfer functions



Network architectures

Three different classes of network architectures

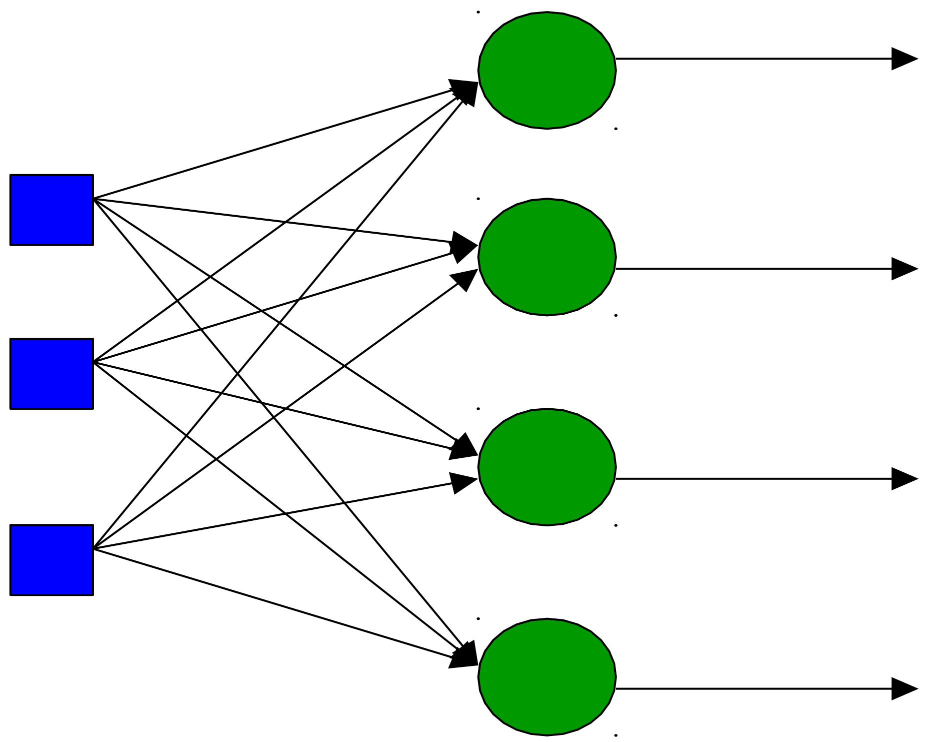
- –single-layer feed-forward
- –multi-layer feed-forward
- –recurrent

} neurons are organized in acyclic layers

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

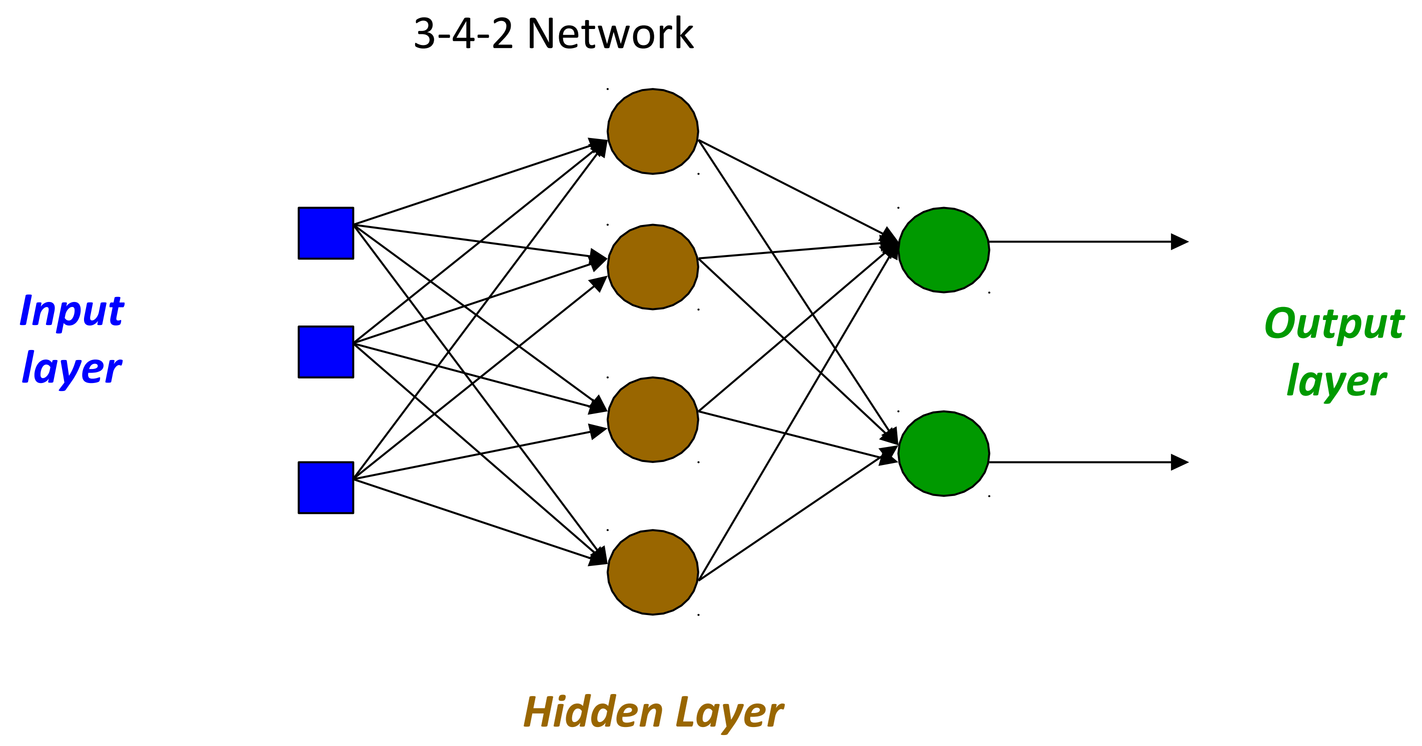
Single Layer Feed-forward

Input layer
of
source nodes



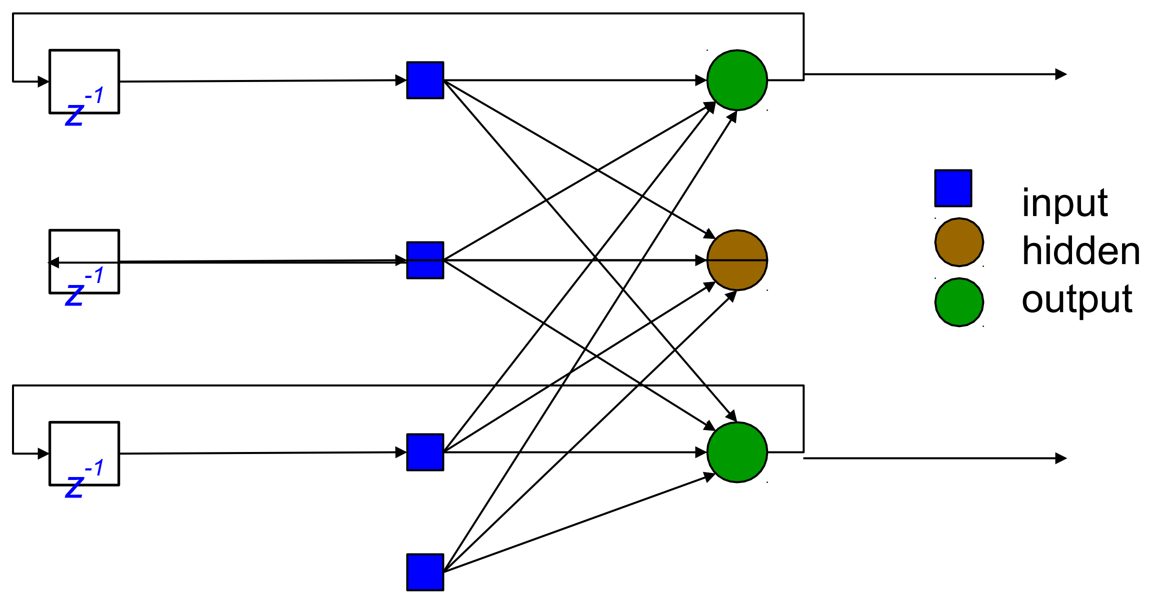
Output layer
of
neurons

Multi layer feed-forward



Recurrent network

Recurrent Network with ***hidden neuron(s)***: unit delay operator z^{-1} implies dynamic system



Dimensions of a Neural Network

Various types of neurons

Various network architectures

Various learning algorithms

Various applications

Training Algorithm

Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.

However, in practice, does converge to low error for many large networks on real data.

Many epochs (thousands) may be required, hours or days of training for large networks.

• To avoid local-minima problems, run several trials starting with different random weights (random restarts).

- Take results of trial with lowest training set error.
- Build a committee of results from multiple trials (possibly weighting votes by training set accuracy).

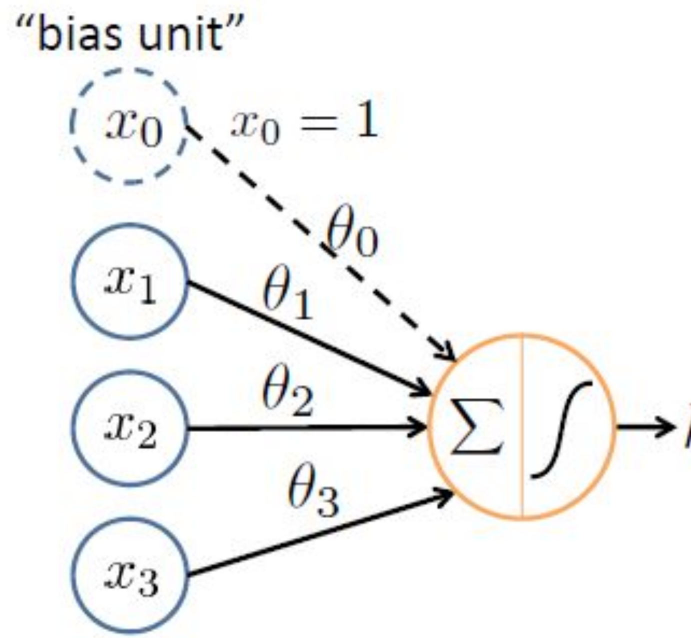
Backpropagation Training Algorithm

Create the 3-layer network with H hidden units with full connectivity between layers. Set weights to small random real values.

Until all training examples produce the correct value (within ε), or mean squared error ceases to decrease, or other termination criteria:

```
Begin epoch
   For each training example, d, do:
        Calculate network output for d's input values
        Compute error between current output and correct output  for d
        Update weights by backpropagating error and using learning rule
End epoch
```

Single Node

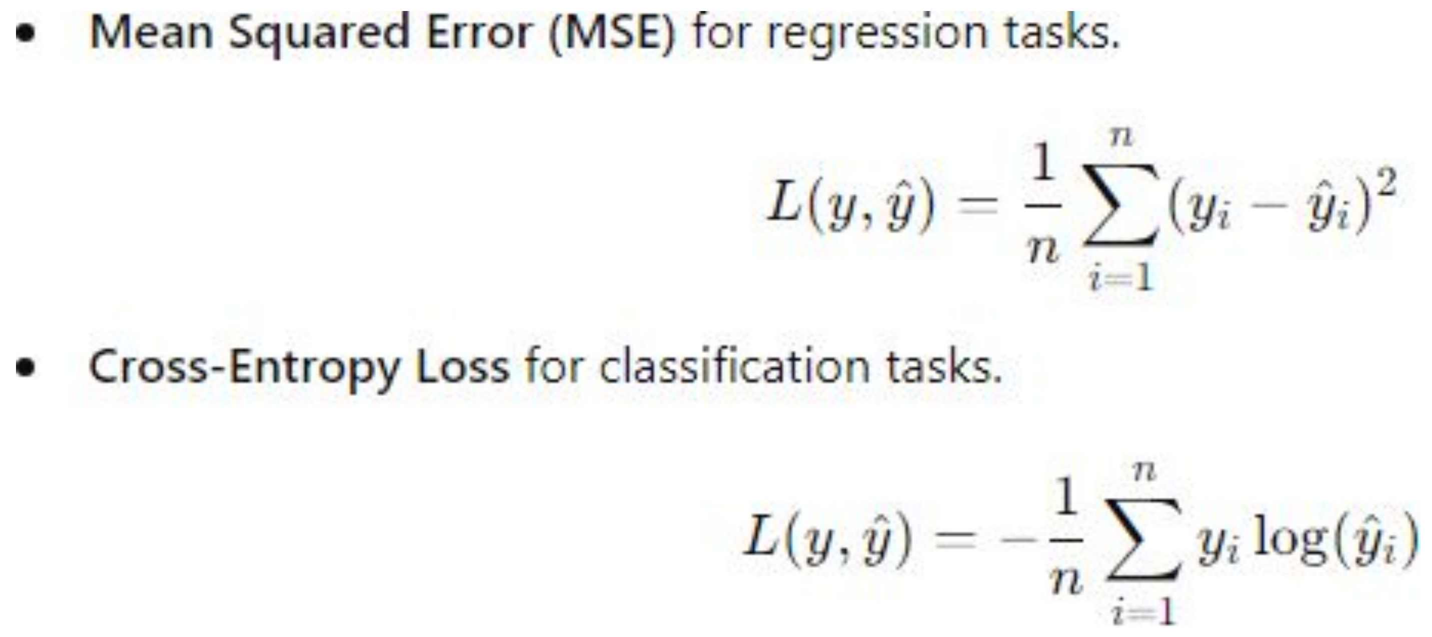


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\intercal \mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\intercal \mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Loss Function: A mathematical function that measures how well the neural network’s predictions match the true labels. Common examples include:



- **Mean Squared Error (MSE)** for regression tasks.

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- **Cross-Entropy Loss** for classification tasks.

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

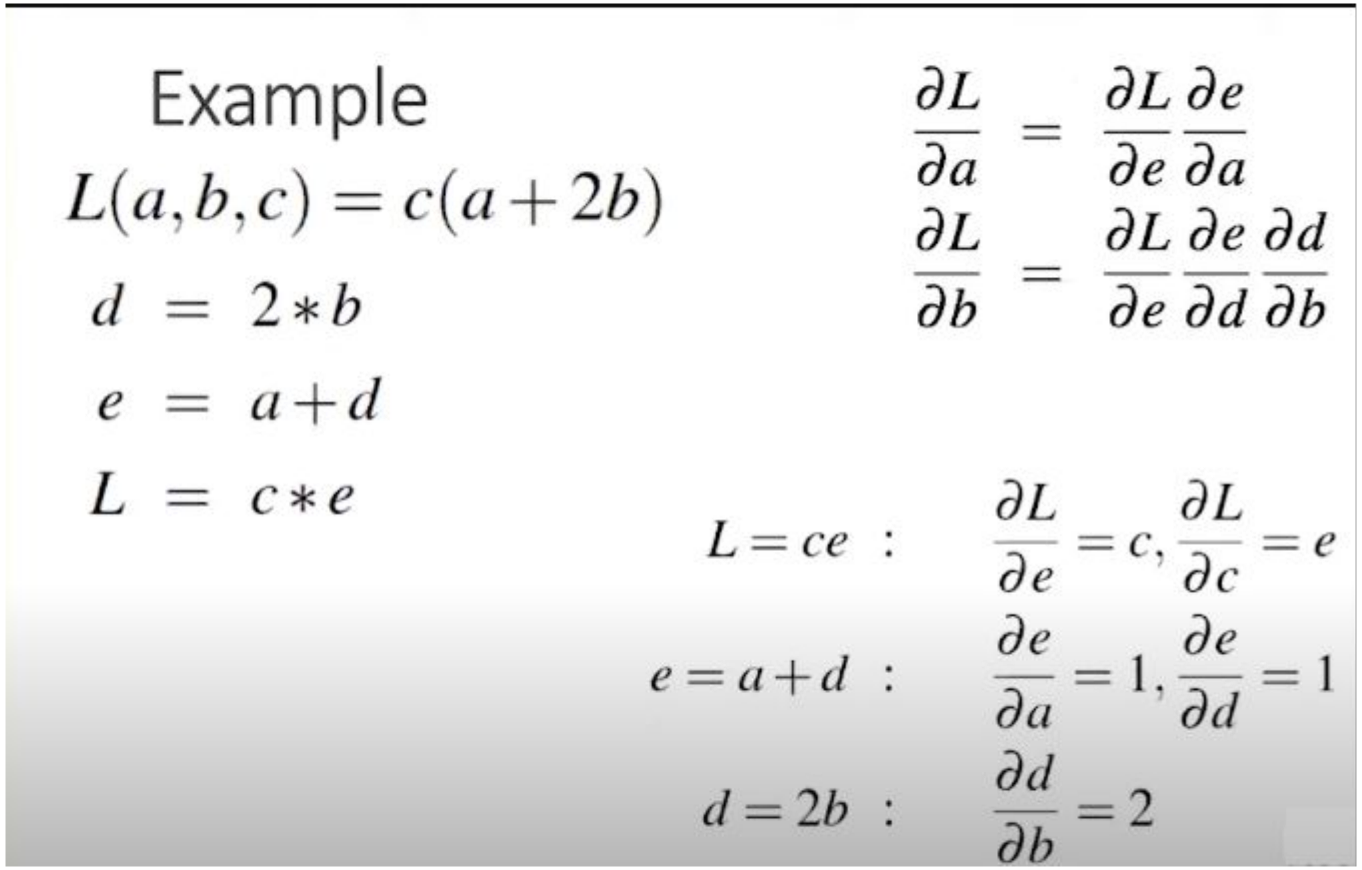
The chain rule

Computing the derivative of a composite function:

$$f(x) = \underline{u(v(x))} \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$$f(x) = \underline{u(v(w(x)))} \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

Computational graph and Chain Rule



Example

$$L(a,b,c) = c(a+2b)$$

$$d = 2*b$$

$$e = a+d$$

$$L = c*e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

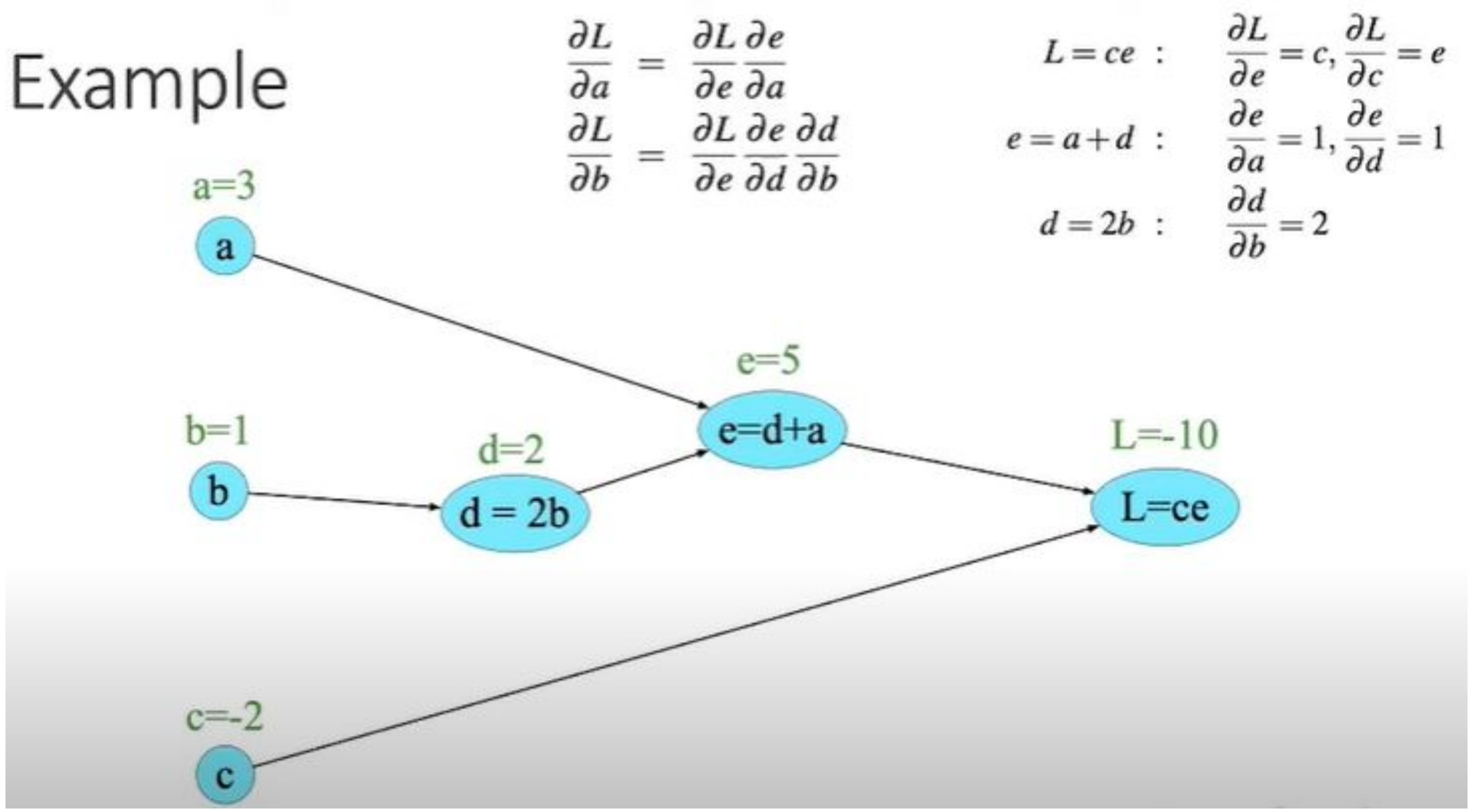
$$L = ce \ : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a+d \ : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d = 2b \ : \quad \frac{\partial d}{\partial b} = 2$$

https://www.youtube.com/watch?v=hM74RH82LyI&t=181s

Computational graph and Chain Rule

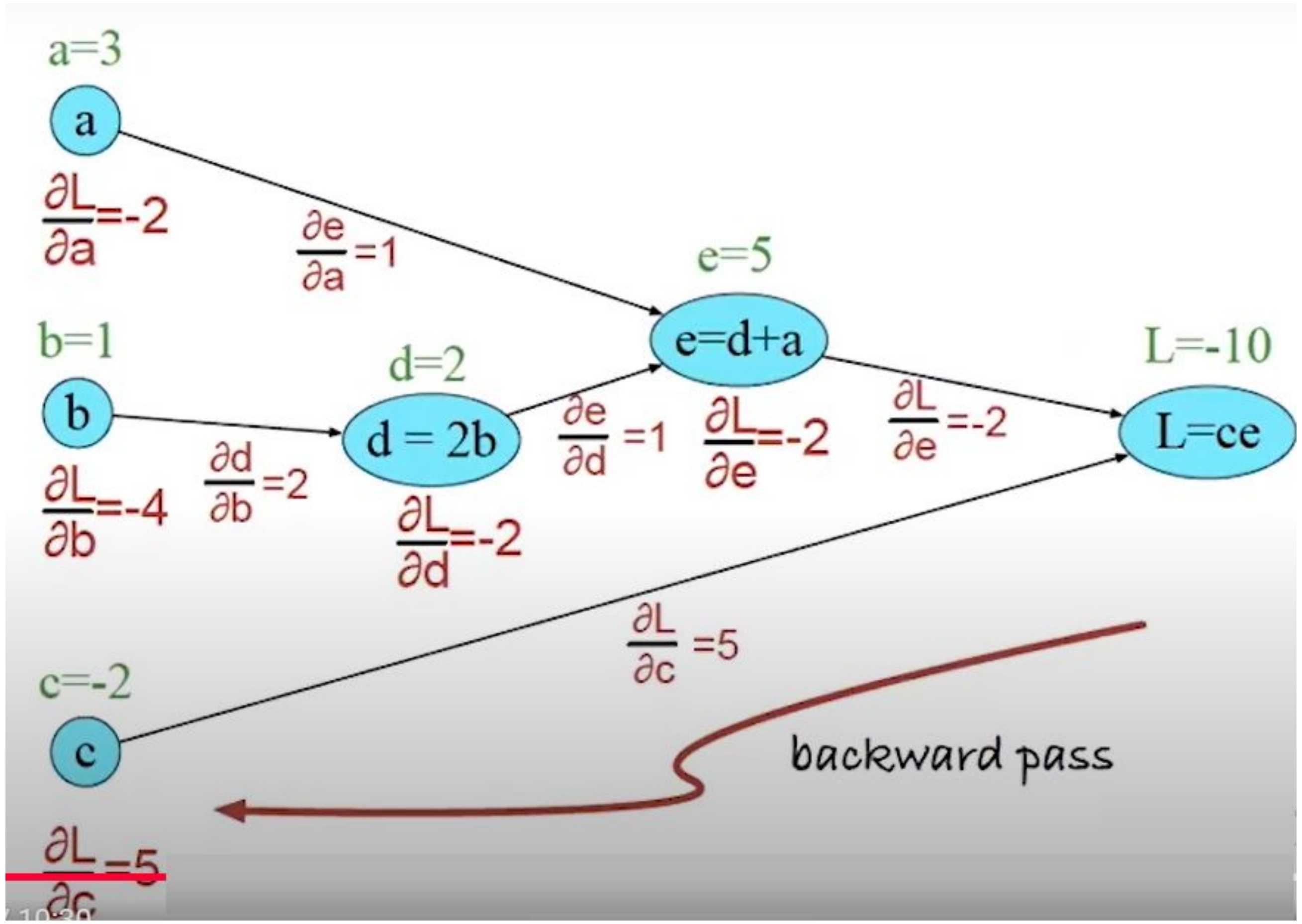


Example

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

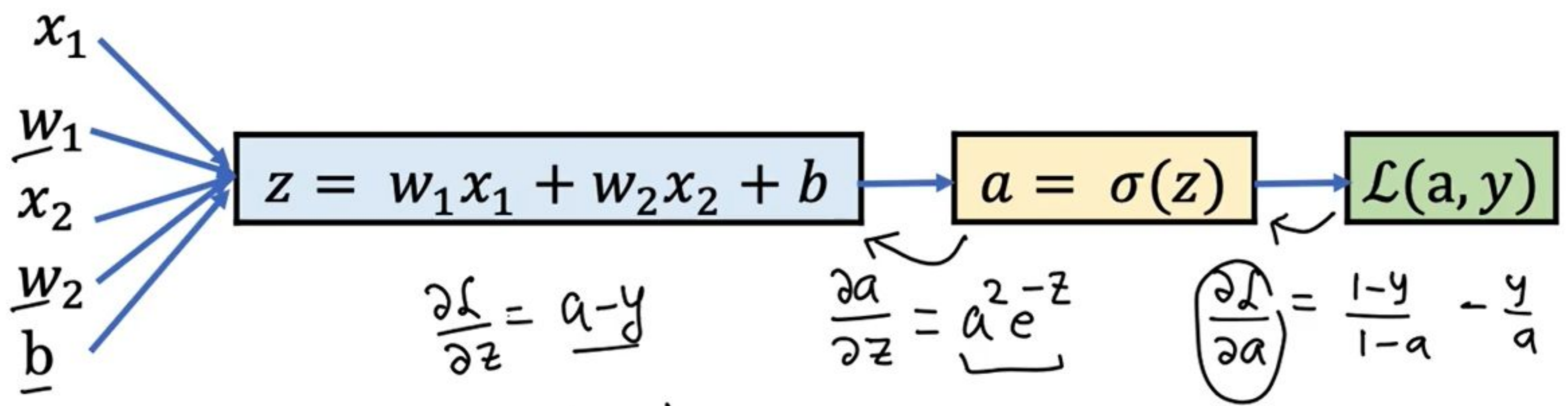
$$L = ce \ : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$
$$e = a + d \ : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$
$$d = 2b \ : \quad \frac{\partial d}{\partial b} = 2$$

Computational graph and Chain Rule



a=3
a
$\frac{\partial L}{\partial a}=-2$
$\frac{\partial e}{\partial a}=1$
e=5
e=d+a
b=1
b
d=2
d = 2b
$\frac{\partial L}{\partial b}=-4$
$\frac{\partial d}{\partial b}=2$
$\frac{\partial e}{\partial d}=1$
$\frac{\partial L}{\partial e}=-2$
$\frac{\partial L}{\partial e}=-2$
L=-10
L=ce
$\frac{\partial L}{\partial d}=-2$
$\frac{\partial L}{\partial c}=5$
c=-2
c
backward pass
$\frac{\partial L}{\partial c}=5$

Computational graph and Chain Rule



$$\frac{\partial \mathcal{L}}{\partial z} = \underline{a-y}$$

$$\frac{\partial a}{\partial z} = \underbrace{a^2 e^{-z}}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial a}} = \frac{1-y}{1-a} - \frac{y}{a}$$

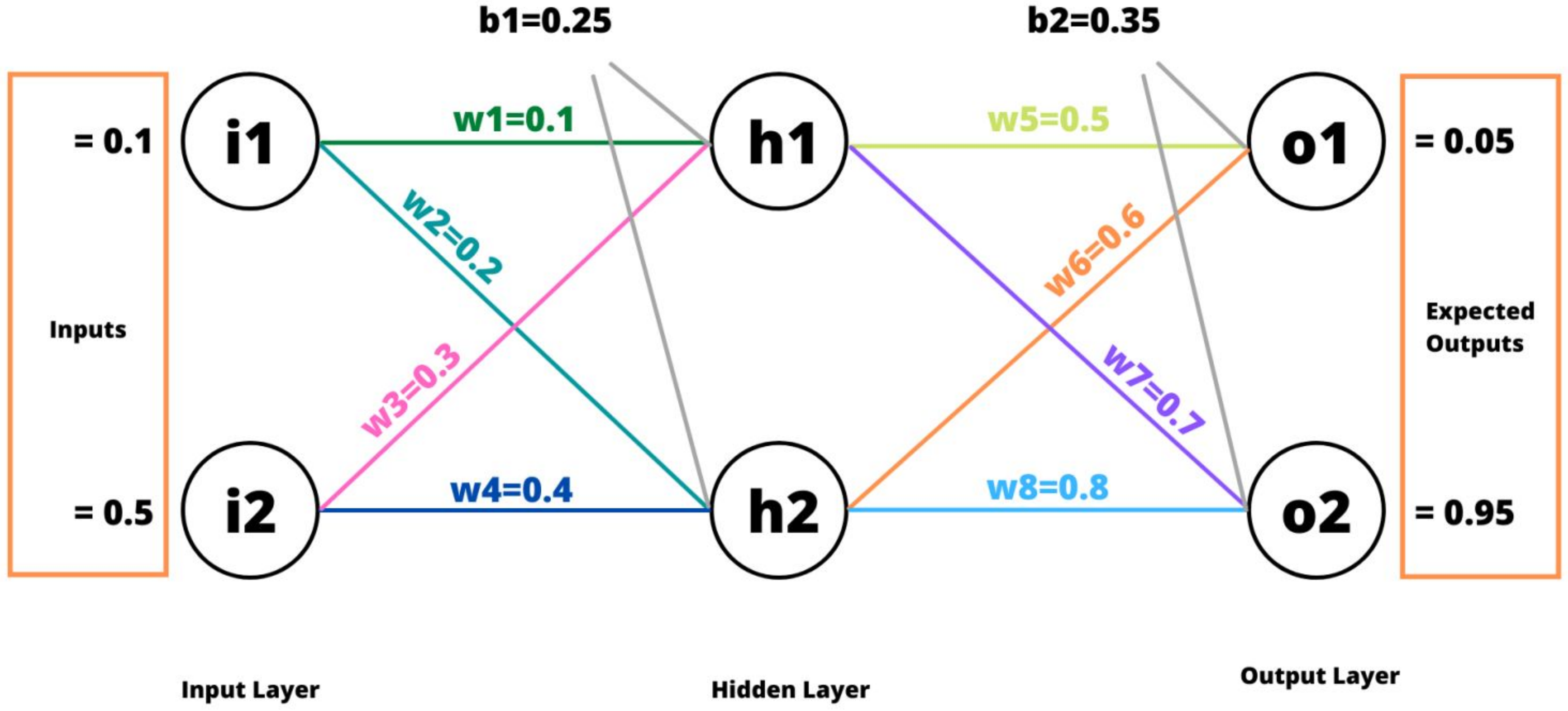
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial z} \times \frac{\partial z}{\partial w_1} = x_1(a-y)$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = x_2(a-y)$$

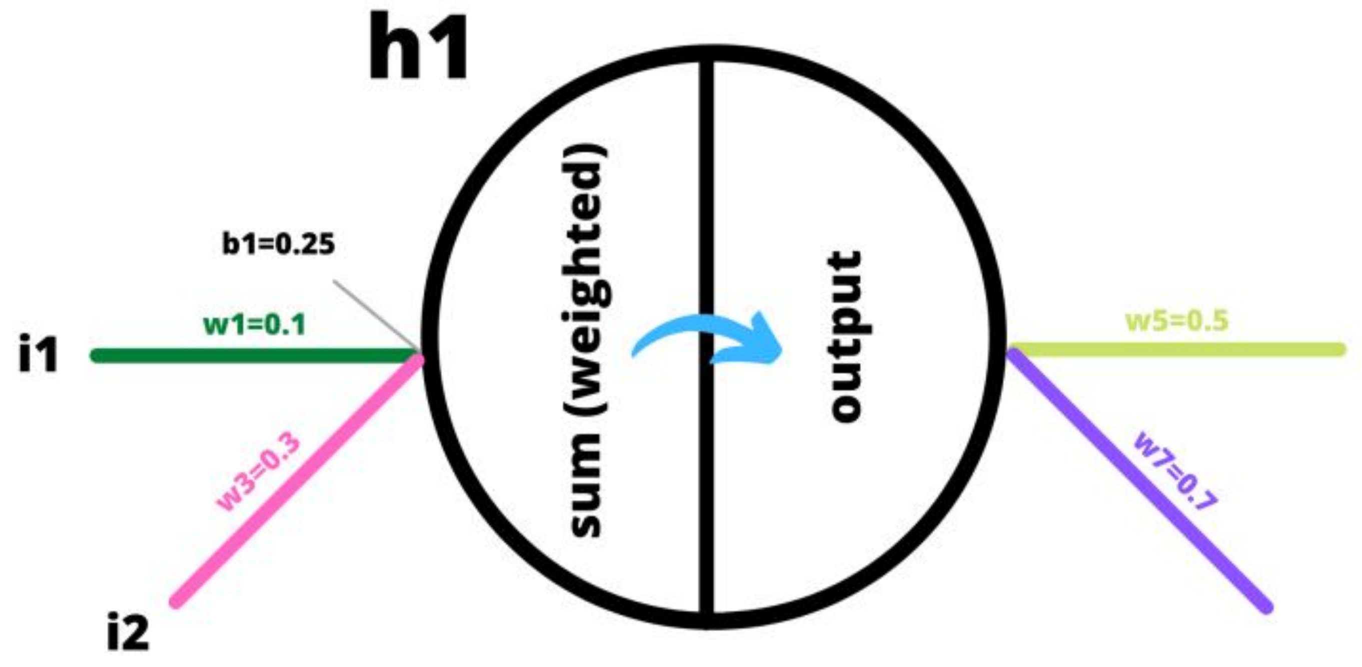
$$\partial \mathcal{L}/\partial b = (a-y)$$

$$\left.\begin{aligned} w_1 &\leftarrow w_1 - \alpha \frac{\partial \mathcal{L}}{\partial w_1} \\ w_2 &\leftarrow w_2 - \alpha\, \partial \mathcal{L}/\partial w_2 \\ b &\leftarrow b - \alpha \frac{\partial \mathcal{L}}{\partial b}. \end{aligned}\right\}$$

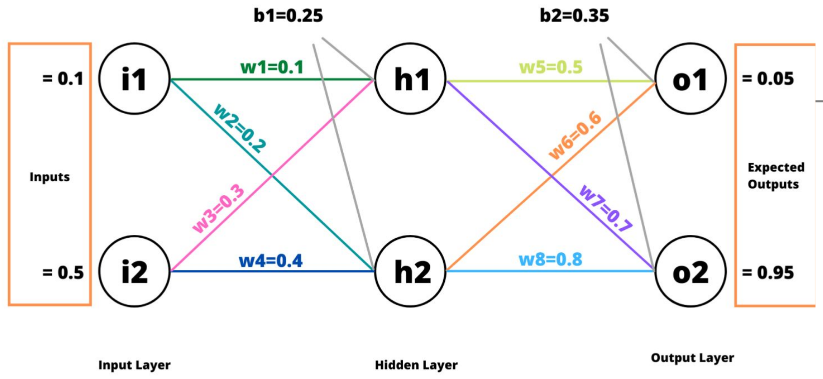
Example Network



Single Neuron



Forward Pass



h$_1$

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1$$

$$sum_{h1} = 0.1 * 0.1 + 0.5 * 0.3 + 0.25 = 0.41$$

$$output_{h1} = \frac{1}{1 + e^{-sum_{h1}}}$$

$$output_{h1} = \frac{1}{1 + e^{-0.41}} = 0.60108$$

h$_2$

$$sum_{h2} = i_1 * w_2 + i_2 * w_4 + b_1 = 0.47$$

$$output_{h2} = \frac{1}{1 + e^{-sum_{h2}}} = 0.61538$$

O$_1$

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2 = 1.01977$$

$$output_{o1} = \frac{1}{1 + e^{-sum_{o1}}} = 0.73492$$

O$_2$

$$sum_{o2} = output_{h1} * w_7 + output_{h2} * w_8 + b_2 = 1.26306$$

$$output_{o2} = \frac{1}{1 + e^{-sum_{o2}}} = 0.77955$$

Computing the total error

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_1 = \frac{1}{2}(target_1 - output_{o1})^2$$

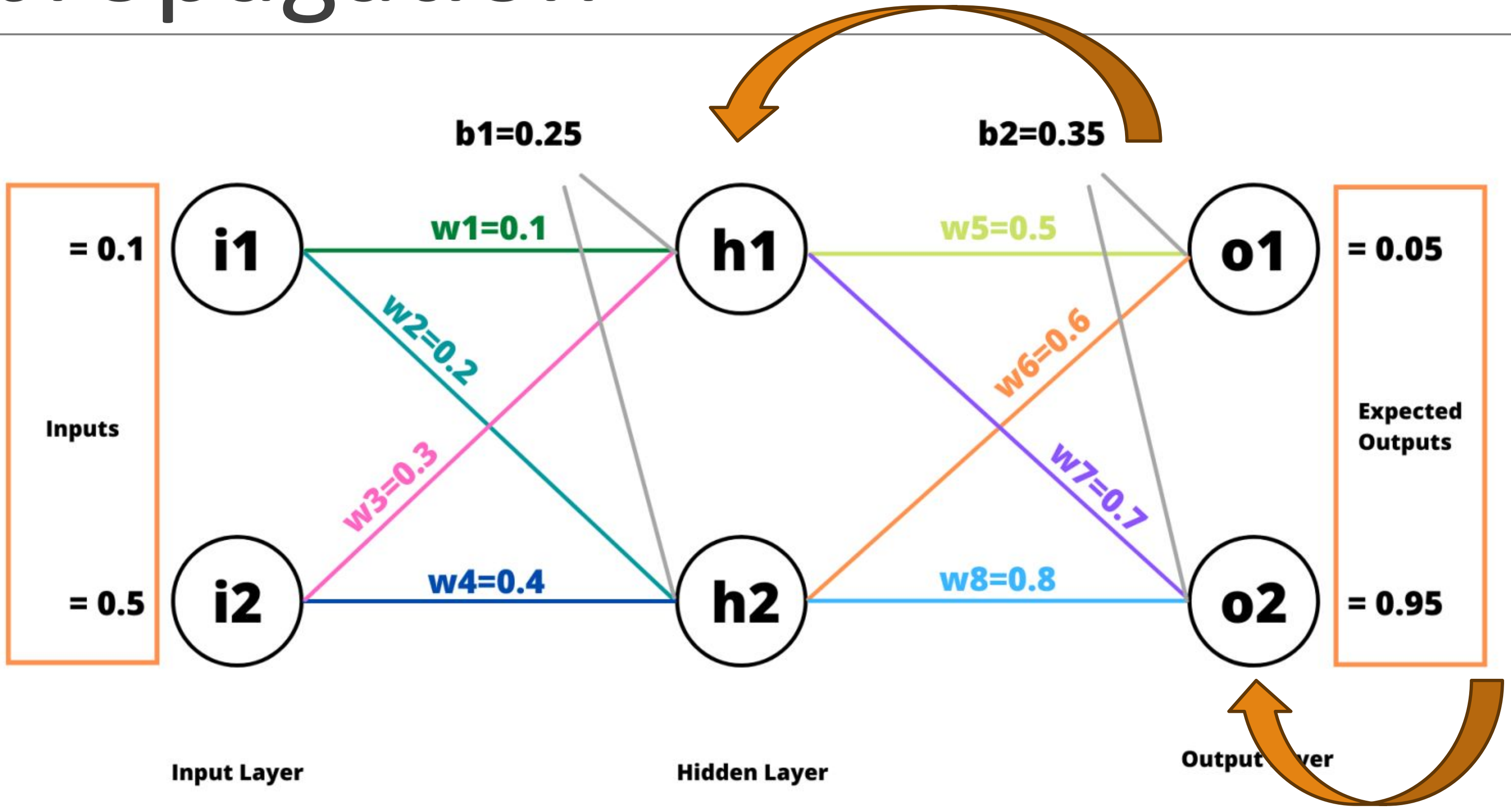
$$E_1 = \frac{1}{2}(0.05 - 0.73492)^2 = 0.23456$$

$$E_2 = \frac{1}{2}(target_2 - output_{o2})^2$$

$$E_2 = \frac{1}{2}(0.95 - 0.77955)^2 = 0.01452$$

$$E_{total} = E_1 + E_2 = 0.24908$$

Back propagation

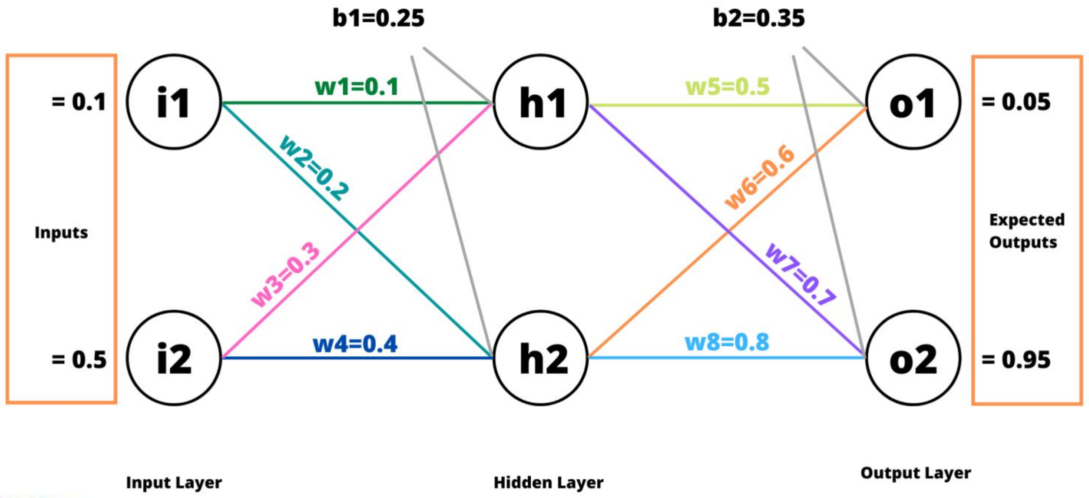


Chain Rule

If we have $y = f(u)$ and $u = g(x)$ then we can write the derivative of y as:

$$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx}$$

For weights in the output layer (w5, w6, w7, w8)



$$\frac{\partial E_{total}}{\partial w5} = \boxed{\frac{\partial E_{total}}{\partial output_{o1}}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

Component 1: partial derivative of Error w.r.t. Output

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{total} = \frac{1}{2}(target_1 - output_{o1})^2 + \frac{1}{2}(target_2 - output_{o2})^2$$

$$\frac{\partial E_{total}}{\partial output_{o1}} = 2 * \frac{1}{2} * (target_1 - output_{o1}) * -1$$

$$= output_{o1} - target_1$$

For weights in the output layer (w5, w6, w7, w8)

Component 2: partial derivative of Output w.r.t. Sum

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \boxed{\frac{\partial output_{o1}}{\partial sum_{o1}}} * \frac{\partial sum_{o1}}{\partial w5}$$

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

$$\frac{\mathrm{d}}{\mathrm{dx}}\sigma(x) = \sigma(x)(1-\sigma(x))$$

$$\frac{\partial output_{o1}}{\partial sum_{o1}} = output_{o1}(1-output_{o1})$$

- **Component 3: partial derivative of Sum w.r.t. Weight**

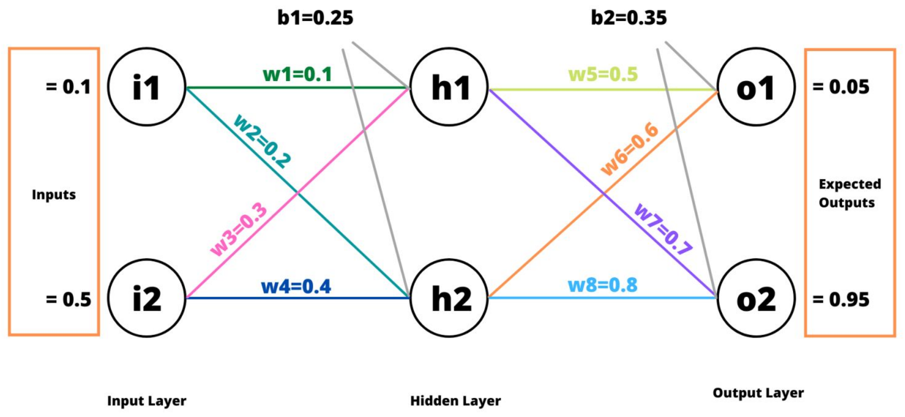
$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \boxed{\frac{\partial sum_{o1}}{\partial w5}}$$

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2$$

Therefore,

$$\frac{\partial sum_{o1}}{\partial w5} = output_{h1}$$

For weights in the output layer (w5, w6, w7, w8)



Putting them together,

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

$$\frac{\partial E_{total}}{\partial w5} = [output_{o1} - target_1] * [output_{o1}(1 - output_{o1})] * [output_{h1}]$$

$$\frac{\partial E_{total}}{\partial w5} = 0.68492 * 0.19480 * 0.60108$$

$$\frac{\partial E_{total}}{\partial w5} = 0.08020$$

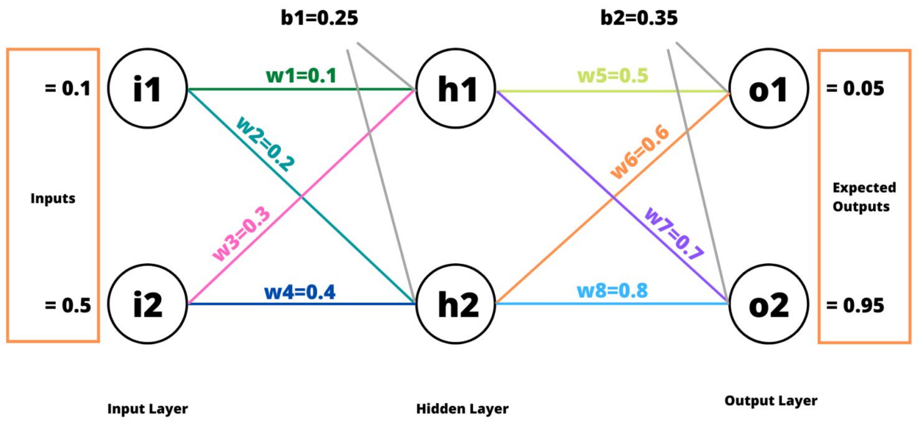
The new_w_5 is,

$new\_w_5 = w5 - n * \frac{\partial E_{total}}{\partial w5}$, where n is learning rate.

$$new\_w_5 = 0.5 - 0.6 * 0.08020$$

$$new\_w_5 = 0.45187$$

For weights in the output layer (w5, w6, w7, w8)



For w6,

$$\frac{\partial E_{total}}{\partial w6} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w6}$$

$$\frac{\partial sum_{o1}}{\partial w6} = output_{h2}.$$

$$\frac{\partial E_{total}}{\partial w6} = 0.68492 * 0.19480 * 0.61538 = 0.08211$$

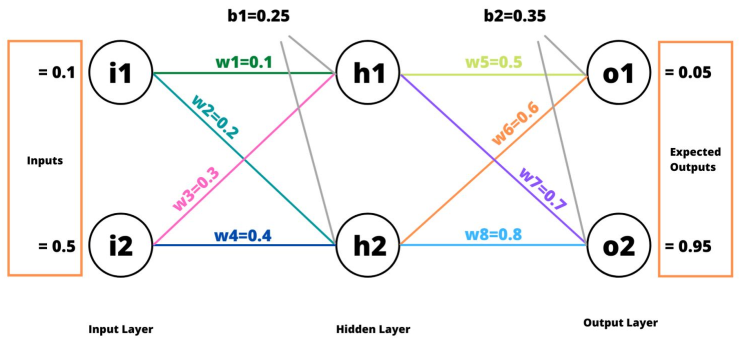
The new_w_6 is,

$$new\_w_6 = w6 - n * \frac{\partial E_{total}}{\partial w6}$$

$$new\_w_6 = 0.6 - 0.6 * 0.08211$$

$$new\_w_6 = 0.55073$$

For weights in the output layer (w5, w6, w7, w8)



For w7,

$$\frac{\partial E_{total}}{\partial w7} = \frac{\partial E_{total}}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial w7}$$

$$\frac{\partial E_{total}}{\partial output_{o2}} = output_{o2} - target_2$$

$$\frac{\partial output_{o2}}{\partial sum_{o2}} = output_{o2}(1 - output_{o2})$$

$$\frac{\partial sum_{o2}}{\partial w7} = output_{h1}$$

$$\frac{\partial E_{total}}{\partial w7} = [output_{o2} - target_2] * [output_{o2}(1 - output_{o2})] * [output_{h1}]$$

$$\frac{\partial E_{total}}{\partial w7} = -0.17044 * 0.17184 * 0.60108$$

$$\frac{\partial E_{total}}{\partial w7} = -0.01760$$

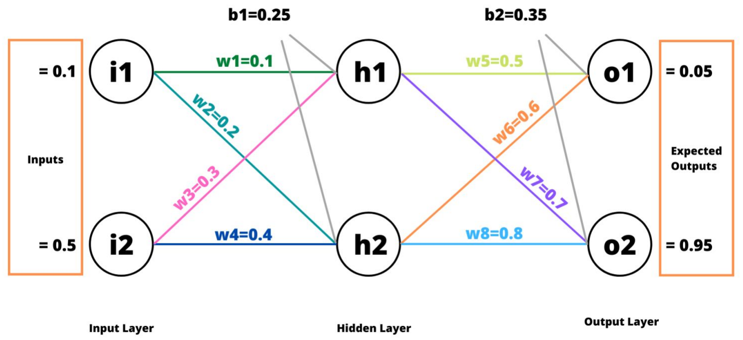
The new_w_7 is,

$$new\_w_7 = w7 - n * \frac{\partial E_{total}}{\partial w7}$$

$$new\_w_7 = 0.7 - 0.6 * -0.01760$$

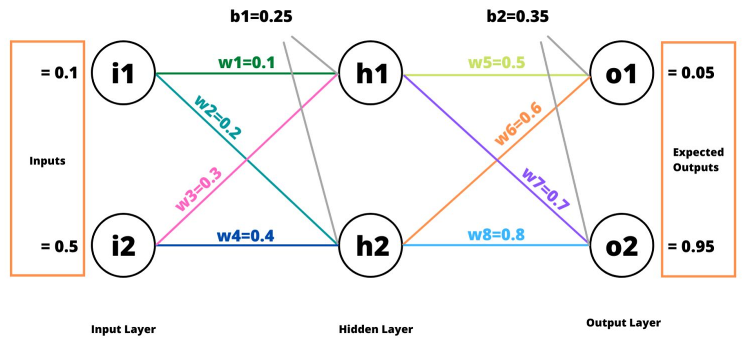
$$new\_w_7 = 0.71056$$

For weights in the output layer (w5, w6, w7, w8)



Proceeding similarly, we get $new\_w_8 = 0.81081$ (with $\frac{\partial E_{total}}{\partial w8} = -0.01802$).

For weights in the output layer (w1, w2, w3, w4)



$$\frac{\partial E_1}{\partial w1} = \frac{\partial E_1}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

Component 1

$$\frac{\partial E_1}{\partial output_{o1}} = output_{o1} - target_1$$

Component 4

$$\frac{\partial output_{h1}}{\partial sum_{h1}} = output_{h1} * (1 - output_{h1})$$

Component 3

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2$$

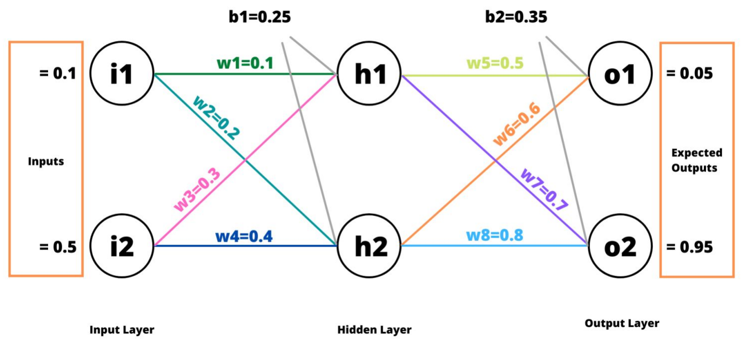
$$\frac{\partial sum_{o1}}{\partial output_{h1}} = w5$$

Component 5

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1$$

$$\frac{\partial sum_{h1}}{\partial w1} = i_1$$

For weights in the output layer (w1, w2, w3, w4)

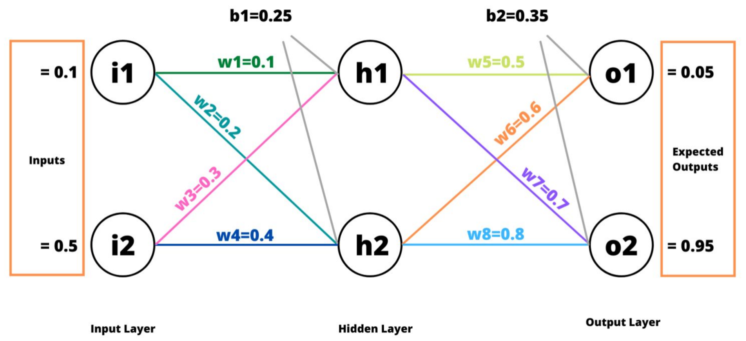


Putting them all together,

$$\frac{\partial E_1}{\partial w1} = \frac{\partial E_1}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_1}{\partial w1} = 0.68492 * 0.19480 * 0.5 * 0.23978 * 0.1 = 0.00159$$

For weights in the output layer (w1, w2, w3, w4)



Similarly, for w1 (with respect to E2),

$$\frac{\partial E_2}{\partial w1} = \frac{\partial E_2}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_2}{\partial output_{o2}} = output_{o2} - target_2$$

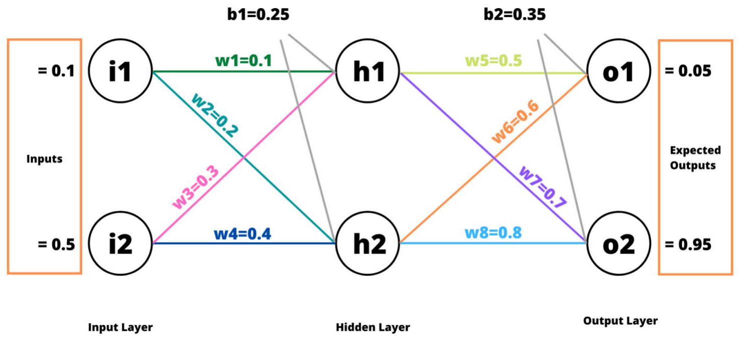
$$sum_{o2} = output_{h1} * w_7 + output_{h2} * w_8 + b_2$$

$$\frac{\partial sum_{o2}}{\partial output_{h1}} = w7$$

$$\frac{\partial E_2}{\partial w1} = \frac{\partial E_2}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_2}{\partial w1} = -0.17044 * 0.17184 * 0.7 * 0.23978 * 0.1 = -0.00049$$

For weights in the output layer (w1, w2, w3, w4)



Now we can compute $\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_1}{\partial w1} + \frac{\partial E_2}{\partial w1}$.

$$\frac{\partial E_{total}}{\partial w1} = 0.00159 + (-0.00049) = 0.00110.$$

The new_w_1 is,

$$new\_w_1 = w1 - n * \frac{\partial E_{total}}{\partial w1}$$

$$new\_w_1 = 0.1 - 0.6 * 0.00110$$

$$new\_w_1 = 0.09933$$

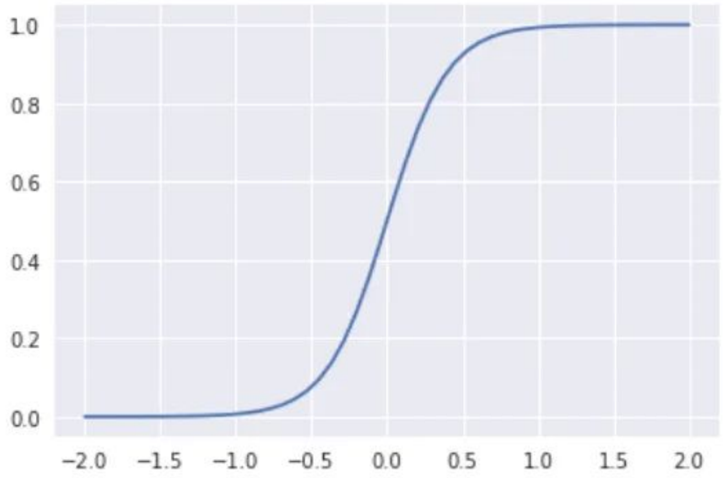
$$new\_w_2 = 0.19919$$

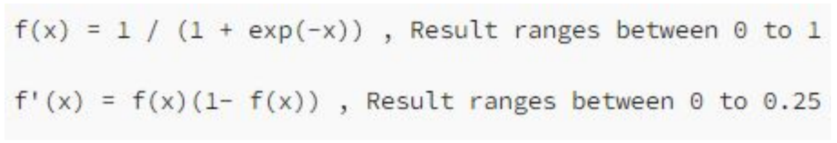
$$new\_w_3 = 0.29667$$

$$new\_w_4 = 0.39597$$

Activation Functions

Sigmoid or Logistic activation function



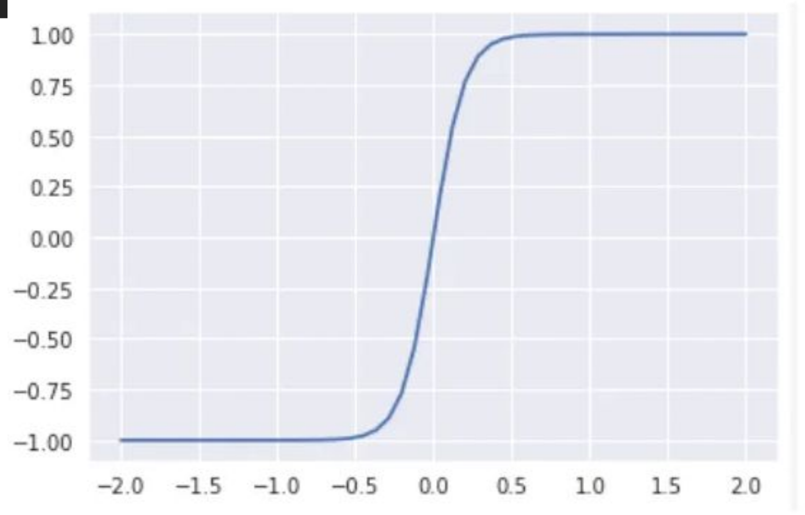


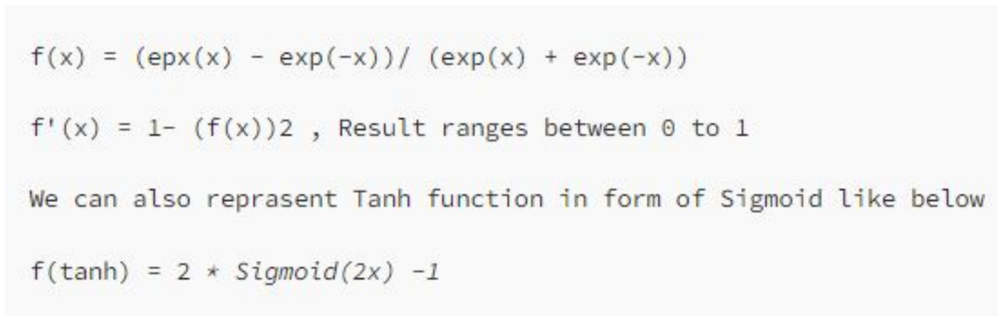
```
f(x) = 1 / (1 + exp(-x)) , Result ranges between 0 to 1

f'(x) = f(x)(1- f(x)) , Result ranges between 0 to 0.25
```

Advantage :-
a) As output of sigmoid function ranges between 0 to 1 hence this **helps to normalize(or scale data) and to achieve non linearity** of the data set.
b) As we **can also perform differentiation** which help the NN to learn during backward propagation.

Disadvantage :-
a) As Sigmoid function **is not a zero centric function** (does not cross origin) so it always gives gradient into +ve direction, which is not good and make optimiser hard to learn/adjust weight of NN.
b) **Vanishing gradient issue**: As the differential value of Sigmoid function ranges between 0 to 1, hence **effect of gradient decreases layer by layer** which cause issue during learning of neural network or **very less impact on first layer of the Neuron.**
c) As Sigmoid function is combination of exponential function (exp(x)), hence it **required lots of computational power** which makes this **more expensive**.

Tanh or Hyper tangent Act. function





```
f(x) = (epx(x) - exp(-x))/ (exp(x) + exp(-x))

f'(x) = 1- (f(x))2 , Result ranges between 0 to 1

We can also reprasent Tanh function in form of Sigmoid like below

f(tanh) = 2 * Sigmoid(2x) -1
```

Advantage :-

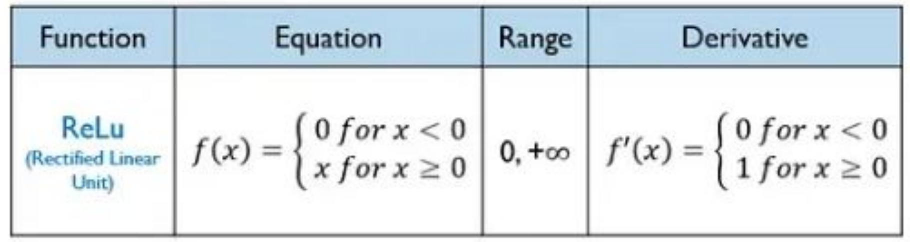
a) Zero centric function, hence this will give gradient in both +ve and -ve direction.

b) Modified version of Sigmoid and comparatively better sigmoid function..

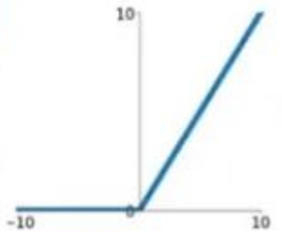
Disadvantage :-

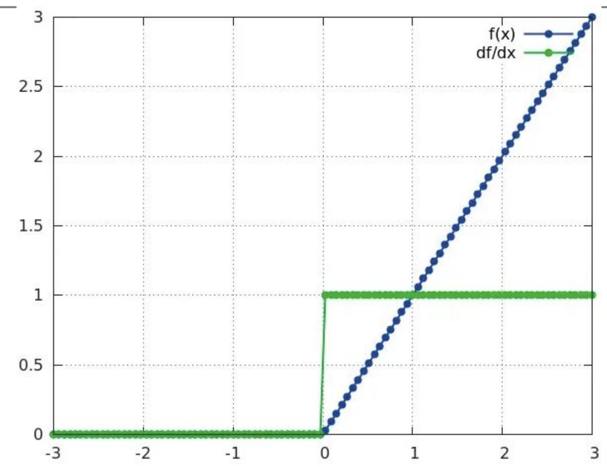
a) For large layers **this function also faces vanishing gradient issue.**
b) As this is modified version of sigmoid hence bit more expensive computational power .

ReLU (Rectified Linear Unit) Act. function



Function	Equation	Range	Derivative
ReLu (Rectified Linear Unit)	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	0, +∞	$f'(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$





Advantage :-
a. This **overcomes the vanishing gradient** issue as differential is going to give the output range between 0 to 1.
b. Main advantage of ReLu, **it does not activate all neurons at a time**. if the output of one neuron is zero then this will activate that neuron.

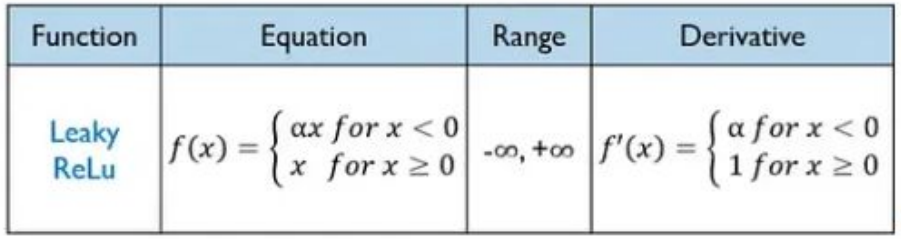
Disadvantage :-

a. This is not zero centric function so it gives only +ve direction gradient

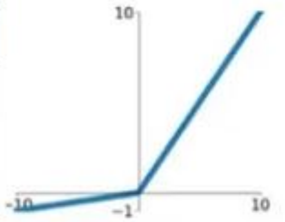
b. For -ve input it does not do anything, which is not good for learning of Neural Network.

c. Dead neuron is the biggest issue of ReLu as it deactivates the neurons for -ve data set & do not provide learning -ve direction.

Leaky ReLU Act. function



Function	Equation	Range	Derivative
Leaky ReLu	$f(x) = \begin{cases} \alpha x \; for \; x < 0 \\ x \;\; for \; x \geq 0 \end{cases}$	-∞, +∞	$f'(x) = \begin{cases} \alpha \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$



$\alpha = 0.01$

Advantage :-
a. **Overcomes the dead neuron** issue by adding constant slope alpha.
b. Less computational expensive compare to sigmoid and tanh function

- Disadvantage : -

a. It does not provide much leaning for -ve dataset/input as alpha value is constant.

Softmax Activation Function

Formula:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Range: (0, 1), where the sum of outputs is equal to 1.

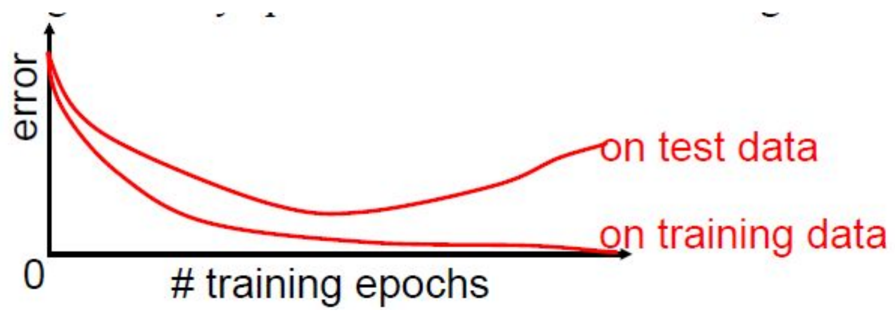
Characteristics:
Converts the raw output into a probability distribution over multiple classes.

Advantages:
Ideal for multi-class classification tasks, providing a probabilistic output.

Disadvantages:
Susceptible to the vanishing gradient problem in deep networks

Over-Training Prevention

Running too many epochs can result in over-fitting.



Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.

Too few hidden units prevents the network from adequately fitting the data.

Too many hidden units can result in over-fitting

Artificial Neural Networks

Advantages

- The output of ANNs can be discrete-valued, real-valued, or a vector of multiple real or discrete-valued characteristics, while the target function can be discrete-valued, real-valued, or a vector of numerous real or discrete-valued attributes.
- Noise in the training data is not a problem for ANN learning techniques. There may be mistakes in the training samples, but they will not affect the final result.

Disadvantages

- Hardware Dependence
- It acts like a black box. Output is not explainable.

Sucessful Applications

Fraud detection

Financial Applications
- HNC (eventually bought by Fair Isaac)

Chemical Plant Control
- Pavillion Technologies

Automated Vehicles

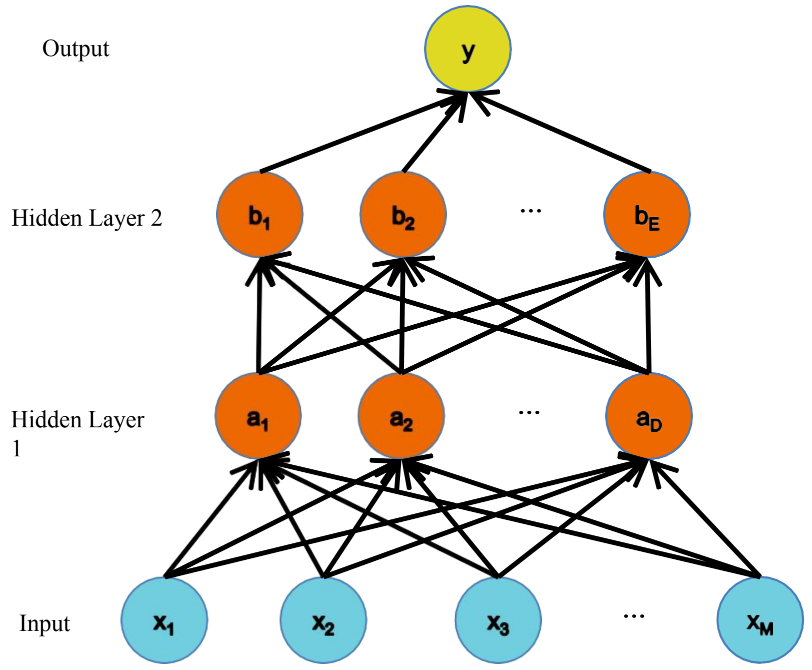
Game Playing
- Neurogammon

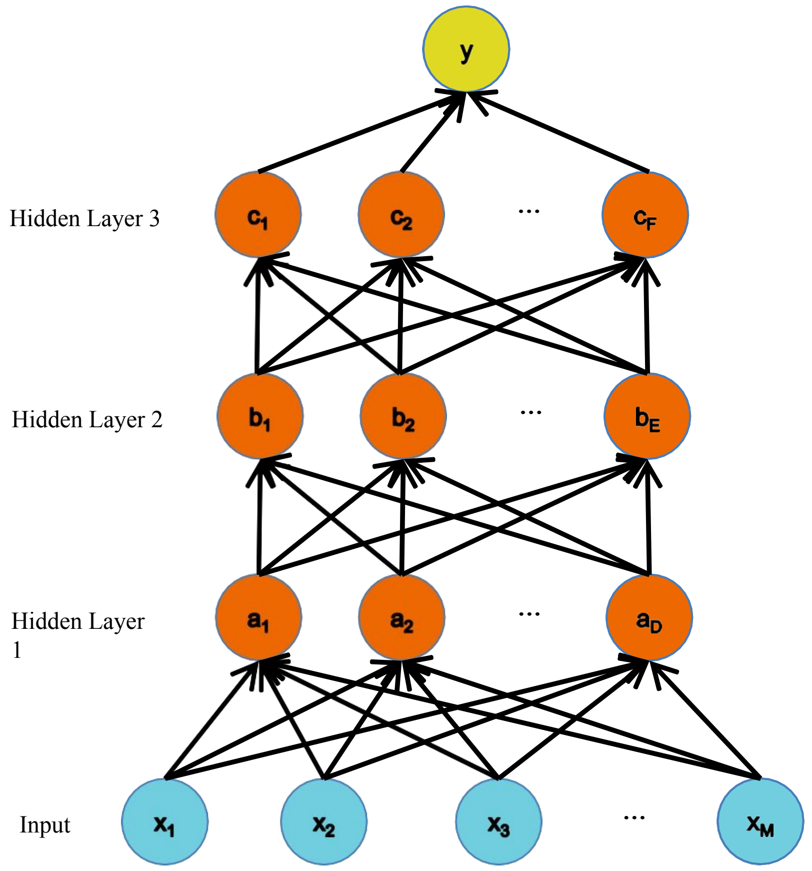
Handwriting recognition

DEEPER NETWORKS

DEEPER NETWORKS

Deeper Networks



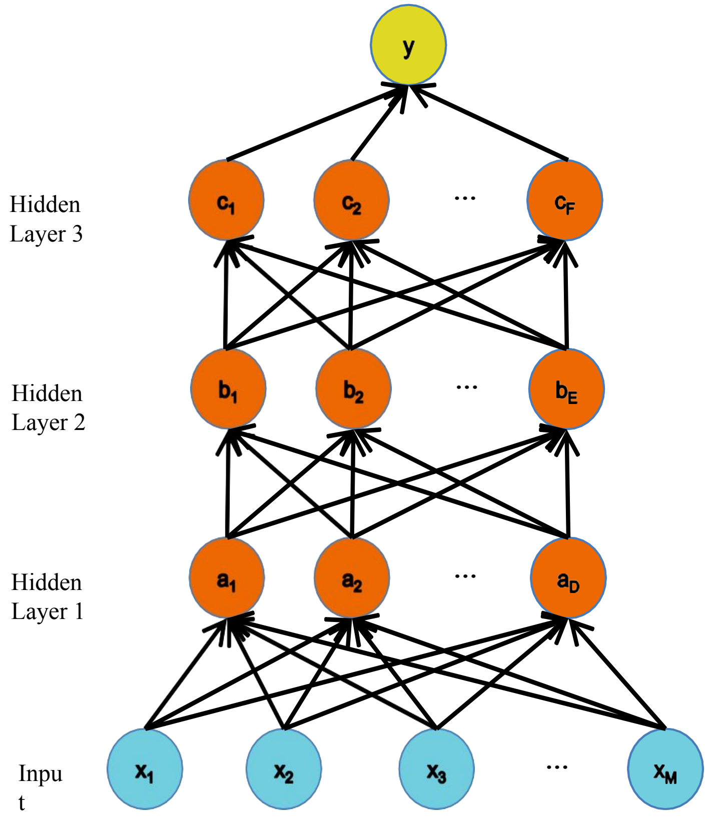


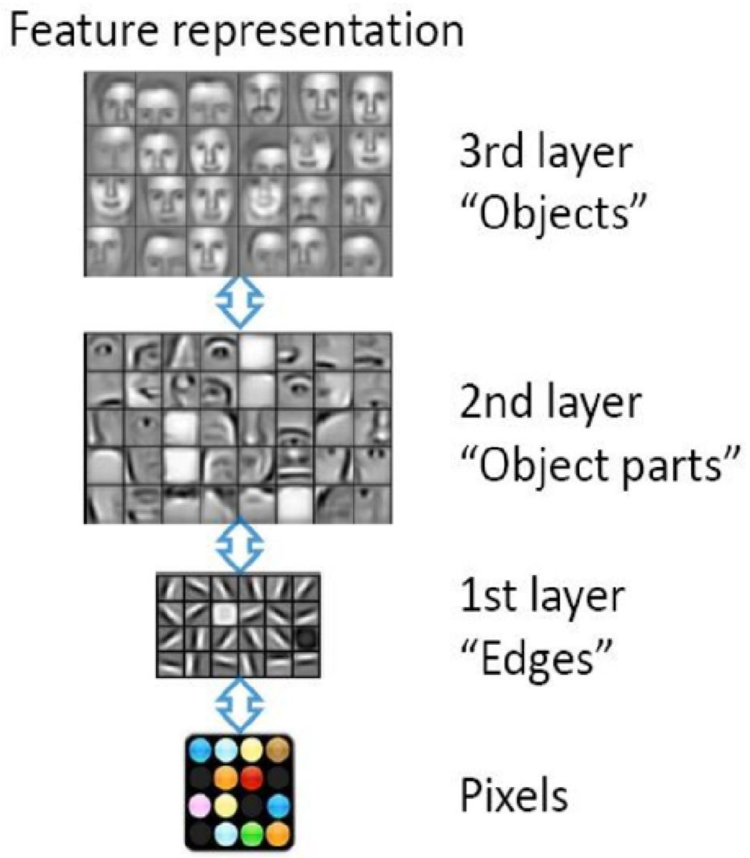
Different Levels of Abstraction

- We don't know the "right" levels of abstraction
- So let the model figure it out!

Face Recognition:

- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions





Example from Honglak Lee (NIPS 2010)

Types of Deep Learning

Feedforward Neural Networks (FNN)

Concept: The simplest type of neural network where information flows in one direction, from the input layer to the output layer, without any loops.

Applications: Basic classification and regression tasks where there are no temporal or spatial dependencies in the data.

Types of Deep Learning

Convolutional Neural Networks (CNNs)

Concept: CNNs are specialized for processing grid-like data, such as images. They consist of convolutional layers that apply filters (kernels) to extract spatial features (e.g., edges, textures) from the data.

Key Components:

- Convolutional Layers: Apply filters to the input to create feature maps that highlight important spatial patterns.
- Pooling Layers: Reduce the spatial dimensions of feature maps to lower computational complexity and make the network more robust to small translations in the input.
- Fully Connected Layers: Flatten the feature maps and connect every neuron to every other neuron to produce the final output.

Applications: Image classification, object detection, semantic segmentation, and other vision-related tasks.

Types of Deep Learning

Transformer Networks

Concept: Transformers use self-attention mechanisms to process sequential data without relying on recurrence. They handle long-range dependencies effectively and process data in parallel, making them efficient for large-scale tasks.

Key Components:

- **Self-Attention:** Calculates attention scores between elements of the sequence, allowing the model to weigh the importance of each element when making predictions.
- **Multi-Head Attention:** Uses multiple attention mechanisms in parallel to capture different aspects of the sequence relationships.
- **Positional Encoding**: Adds positional information to the input to retain the order of elements in the sequence.

Applications: NLP tasks (e.g., translation, summarization), image classification (e.g., Vision Transformers), and multi-modal tasks (e.g., text and image fusion).

Types of Deep Learning

Generative Adversarial Networks (GANs)

Concept: GANs consist of two neural networks, a generator and a discriminator, that compete against each other. The generator tries to create realistic data samples, while the discriminator attempts to distinguish between real and generated samples.

Training Process:

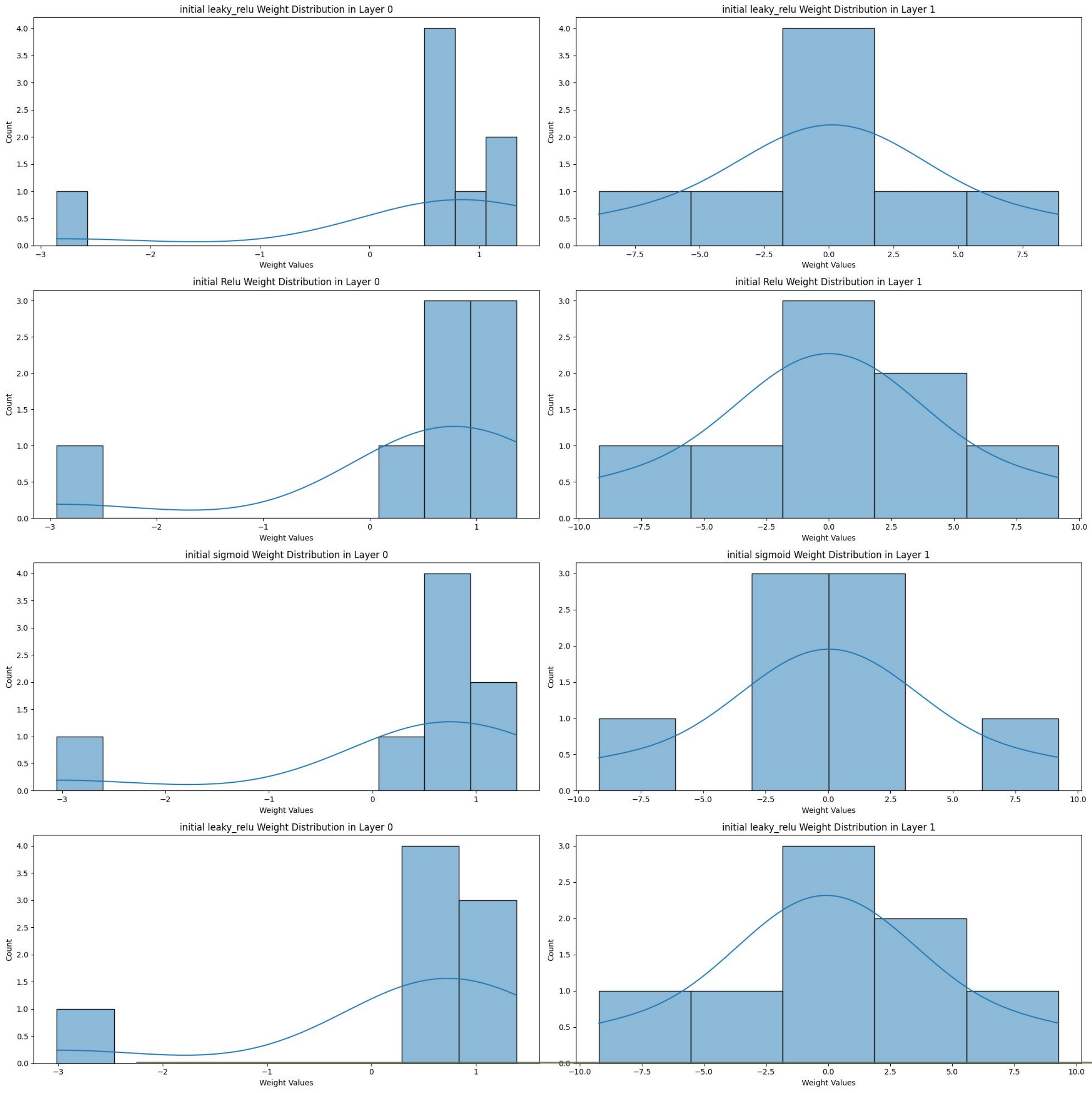
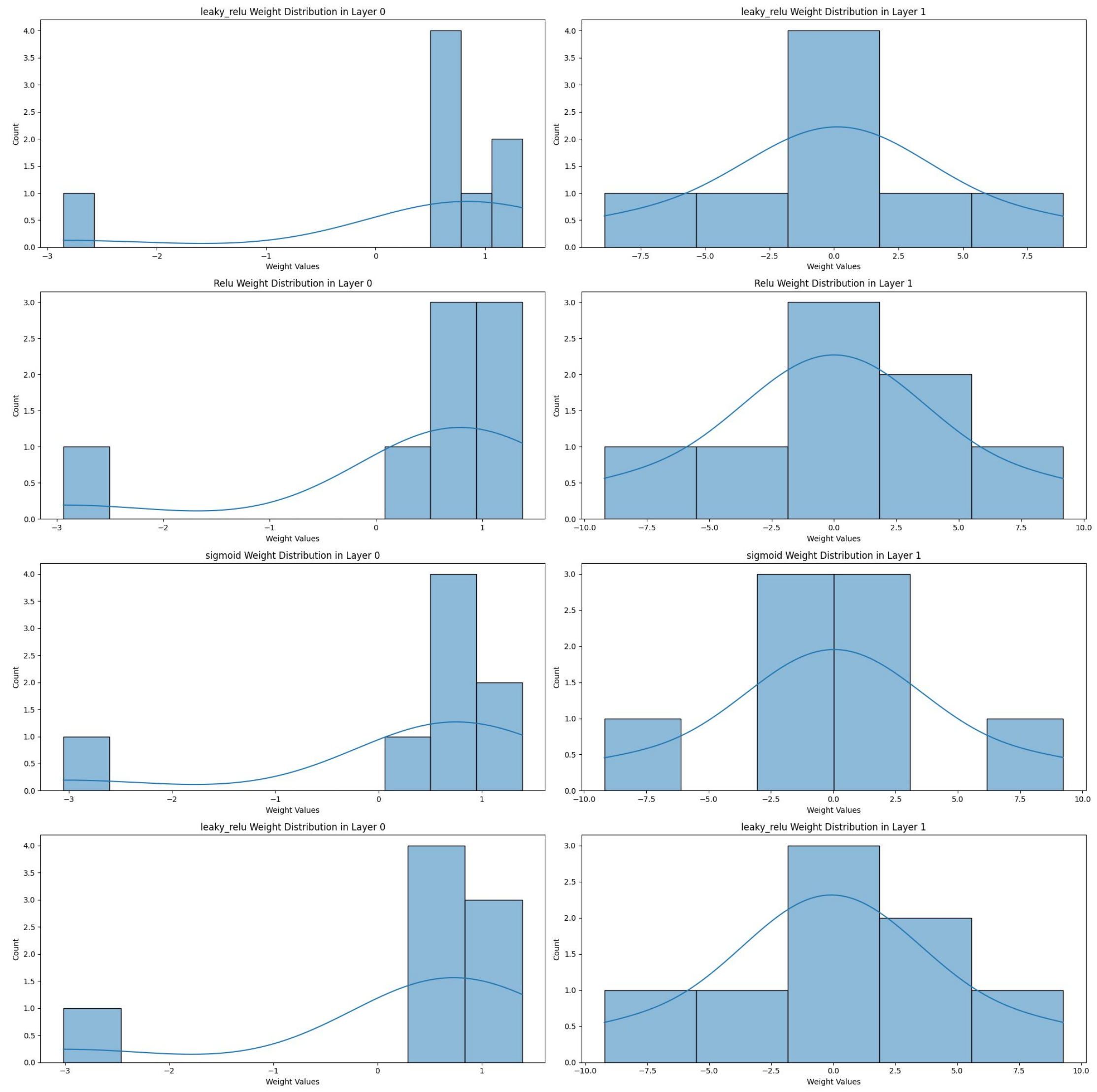
- ❑The generator produces fake samples from random noise.
- ❑The discriminator evaluates both real and generated samples, providing feedback to the generator to improve its output.

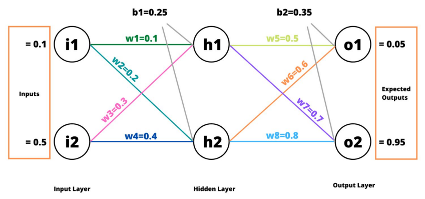
Applications: Image generation, style transfer, super-resolution, and data augmentation.

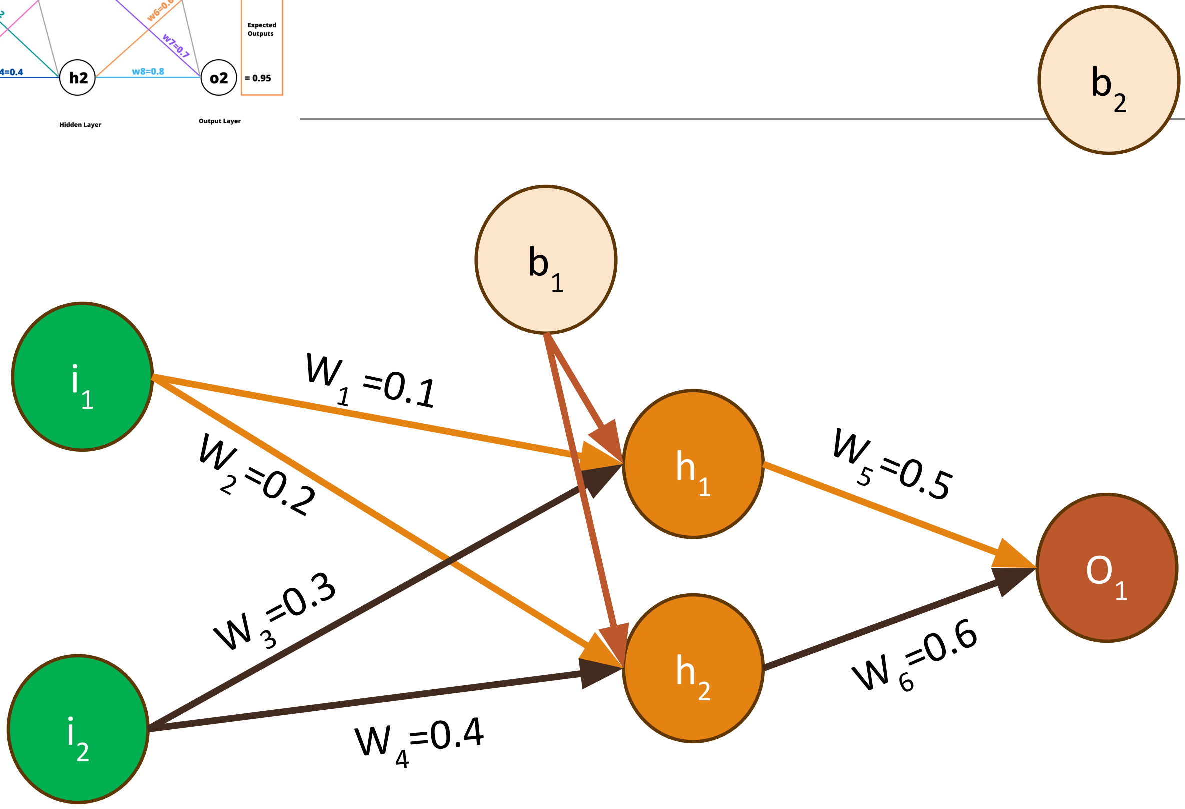
Thank You

Reference

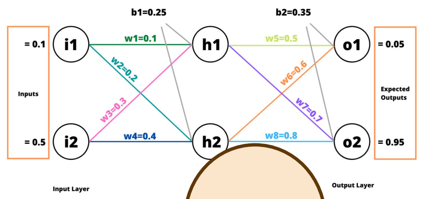
- https://theneuralblog.com/forward-pass-backpropagation-example/
- https://medium.com/@santosh76792/activation-functions-and-their-advantages-disadvantages-d5eaa1717805

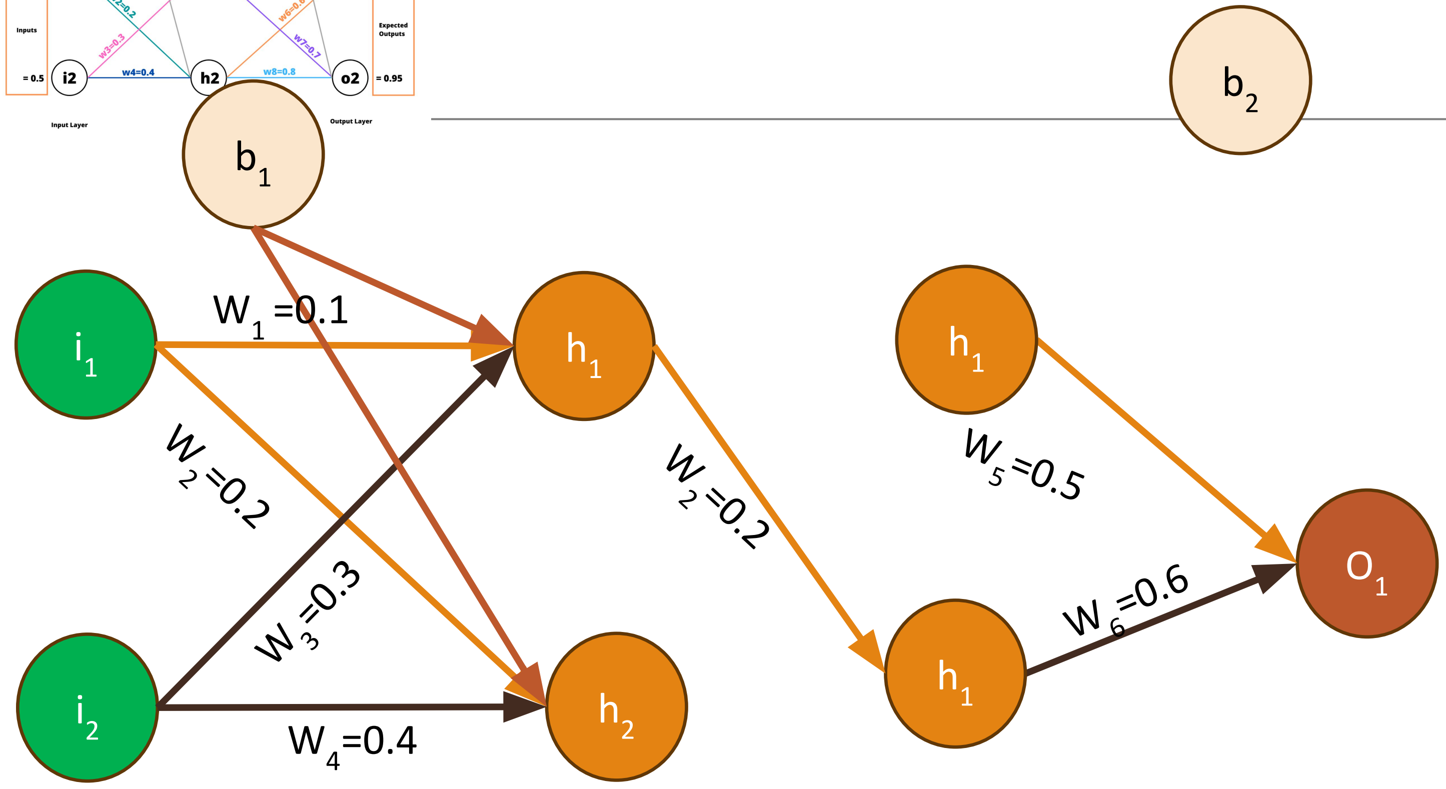

initial leaky_relu Weight Distribution in Layer 0
initial leaky_relu Weight Distribution in Layer 1
initial Relu Weight Distribution in Layer 0
initial Relu Weight Distribution in Layer 1
initial sigmoid Weight Distribution in Layer 0
initial sigmoid Weight Distribution in Layer 1
initial leaky_relu Weight Distribution in Layer 0
initial leaky_relu Weight Distribution in Layer 1
Count
Weight Values

leaky_relu Weight Distribution in Layer 0
leaky_relu Weight Distribution in Layer 1
Relu Weight Distribution in Layer 0
Relu Weight Distribution in Layer 1
sigmoid Weight Distribution in Layer 0
sigmoid Weight Distribution in Layer 1
leaky_relu Weight Distribution in Layer 0
leaky_relu Weight Distribution in Layer 1
Count
Weight Values





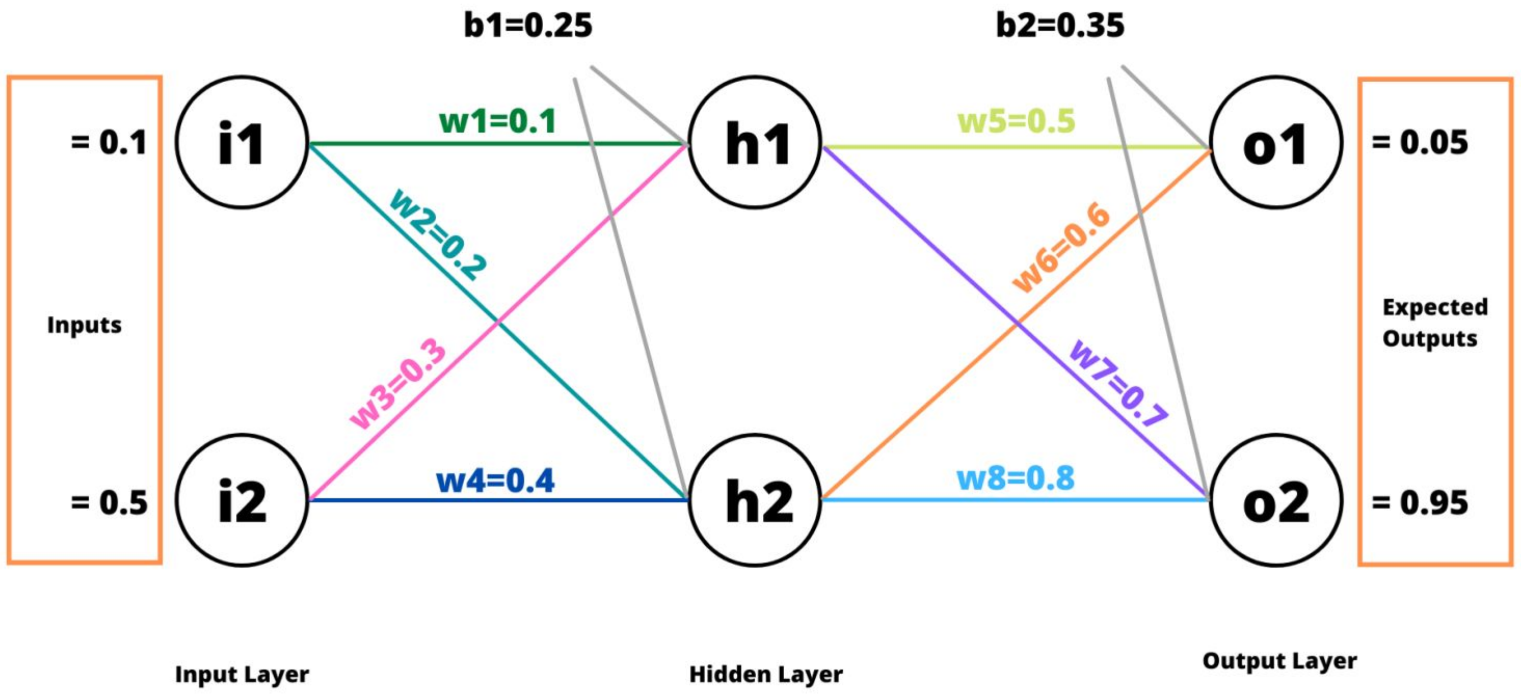
I_1	I_2	o_1
0	0	0
0	1	1
1	0	1
1	1	0





I_1	I_2	o_1
0	0	0
0	1	1
1	0	1
1	1	0

Example Network


b1=0.25
b2=0.35
= 0.1
= 0.5
Inputs
i1
i2
h1
h2
o1
o2
w1=0.1
w2=0.2
w3=0.3
w4=0.4
w5=0.5
w6=0.6
w7=0.7
w8=0.8
= 0.05
= 0.95
Expected Outputs
Input Layer
Hidden Layer
Output Layer