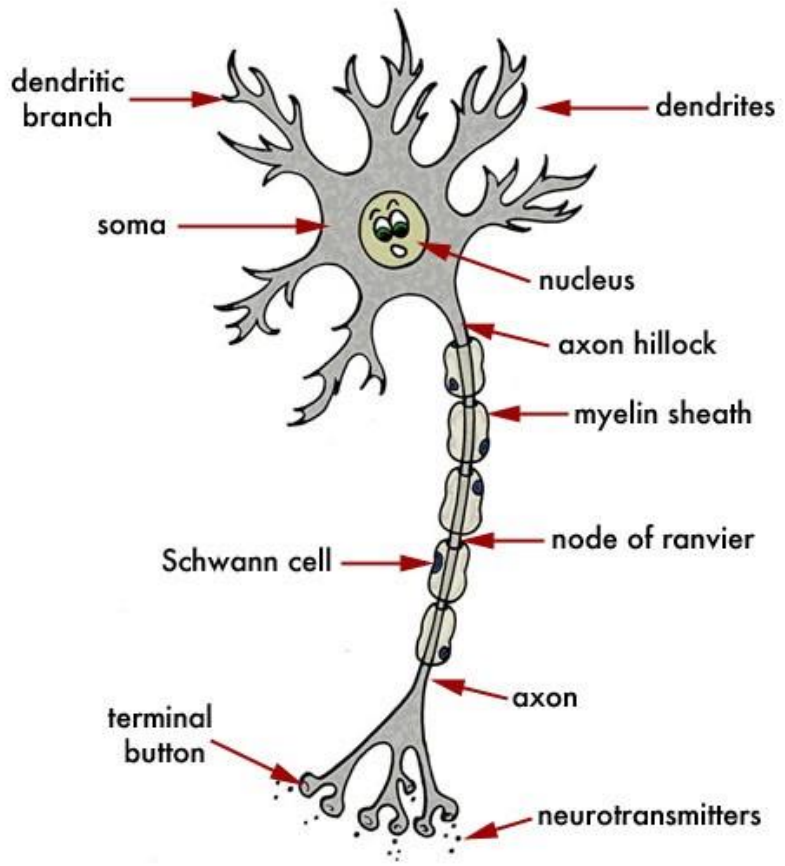
Module 5
Artificial Neural Network

Dr. Kuppusamy .P
Associate Professor / SCOPE

Artificial Neural Networks — Mapping the Human Brain

- The human brain consists of neurons or nerve cells which transmit and process the information received from our senses like ear, nose, etc.
- Many such nerve cells are arranged together in the brain to form a network of nerves.
- These nerves pass electrical impulses i.e., the excitation from one neuron to the other.
- The dendrites receive the impulse from the terminal button or synapse of an adjoining neuron. **(input)**
- Dendrites carry the impulse to the nucleus of the nerve cell (soma).
- Here, the electrical impulse is processed and then passed on to the axon. (**output)**
- The axon is longer branch among the dendrites which carries the impulse from the soma to the synapse.
- The synapse then, passes the impulse to dendrites of the second neuron.
- Thus, a complex network of neurons is created in the human brain.

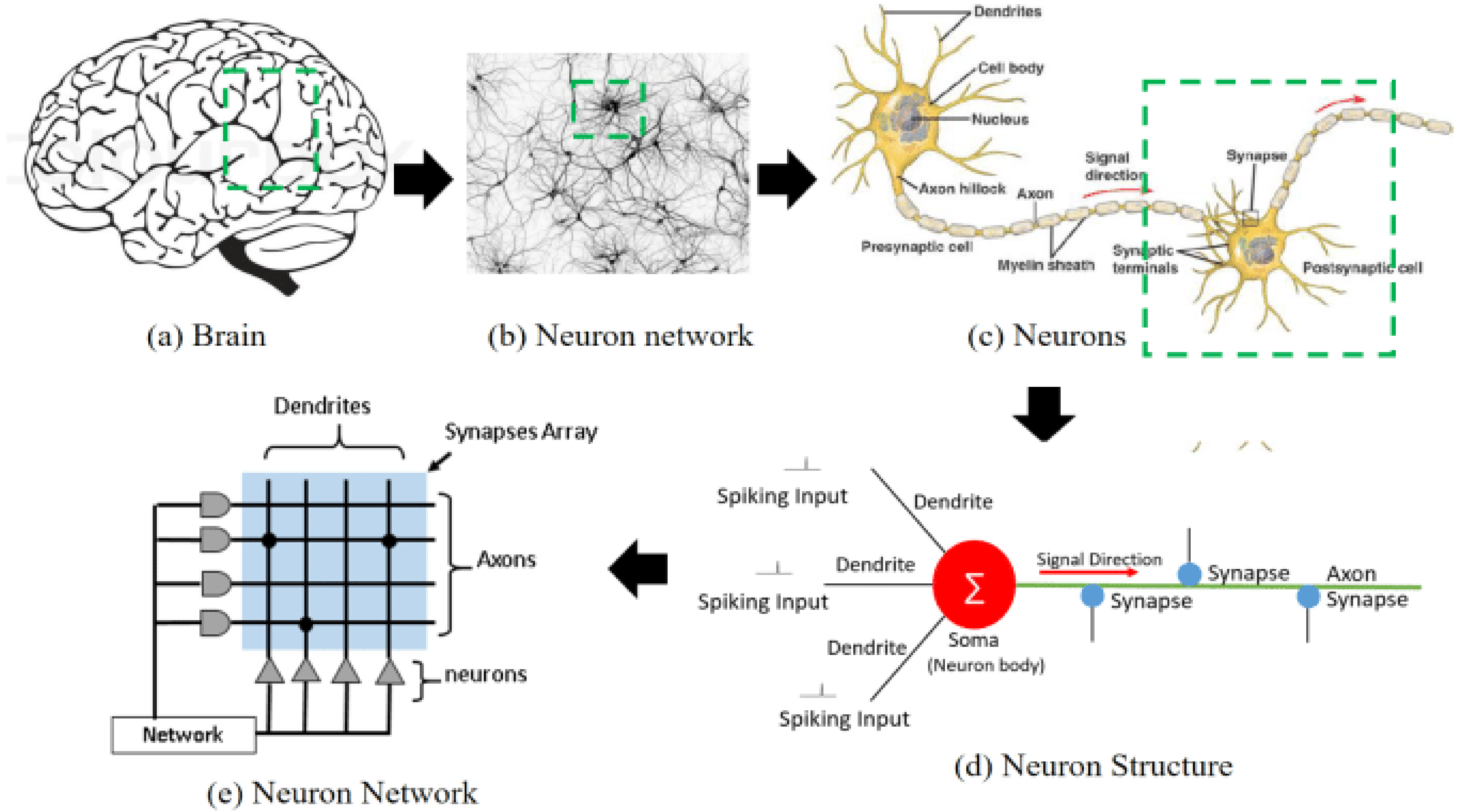


Artificial Neural Networks

- Neural Networks imitate the behavior of the human brain. NN allows computer programs to recognize patterns and solve the problems in the AI, machine learning, and deep learning domains.
- Artificial neural networks (ANNs) contains a node layers such as an input layer, one or more hidden layers, and an output layer.

Artificial Neuron Construction

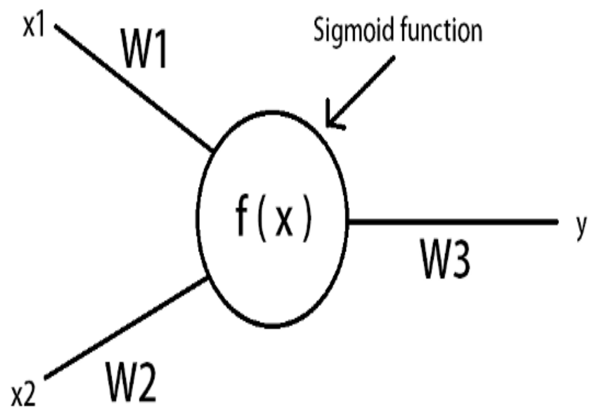
- Neurons are created artificially on a computer. Connecting many such artificial neurons creates an artificial neural network.
- Artificial neuron works like a neuron present in human brain.
- The data in the network flows through each neuron by a connection.
- Every connection has a specific **weight** by which the **flow of data** is **regulated**.
- If the output of any individual node is greater than a specified threshold value, that node is activated, sending data to the next layer of the network.
- Otherwise, the node is not activated, and data is not passed to the next layer.

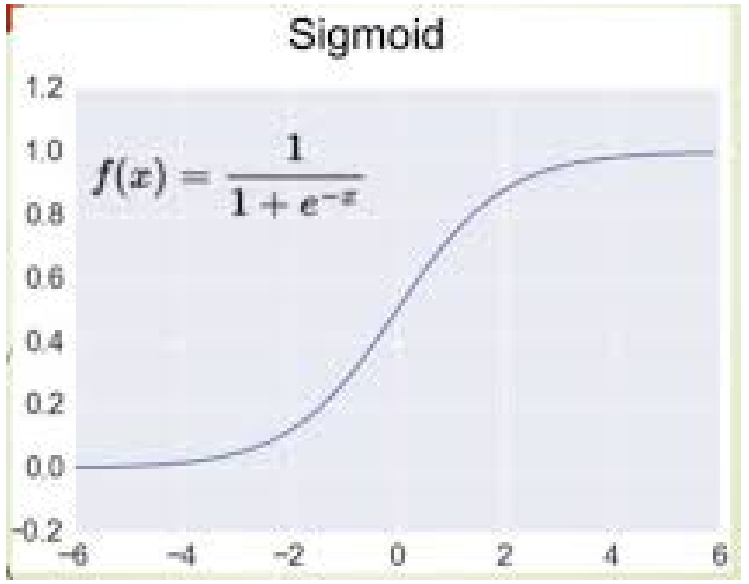

(a) Brain
(b) Neuron network
Dendrites
Cell body
Nucleus
Axon hillock
Axon
Signal direction
Synapse
Presynaptic cell
Myelin sheath
Synaptic terminals
Postsynaptic cell
(c) Neurons
Spiking Input
Dendrite
Dendrite
Spiking Input
Dendrite
Spiking Input
Σ
Soma
(Neuron body)
Signal Direction
Synapse
Synapse
Axon
Synapse
(d) Neuron Structure
Dendrites
Synapses Array
Axons
neurons
Network
(e) Neuron Network

Artificial Neuron Working Principle

- Perceptron is an Artificial Neuron or neural network unit that performs certain computations to recognize features or business intelligence in the input data.
- Perceptron model is a supervised learning algorithm of binary classifiers. It contains a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.
- Let's consider x1 and x2 are the two inputs. They could be an integer, float etc. Assume them as 1 and 0 respectively.
- When these inputs pass through the connections (through W1 and W2), they are adjusted depending on the connection weights.
- Let's assume, W1 = 0.5, W2 = 0.6 and W3 = 0.2 , then adjusting the weights using

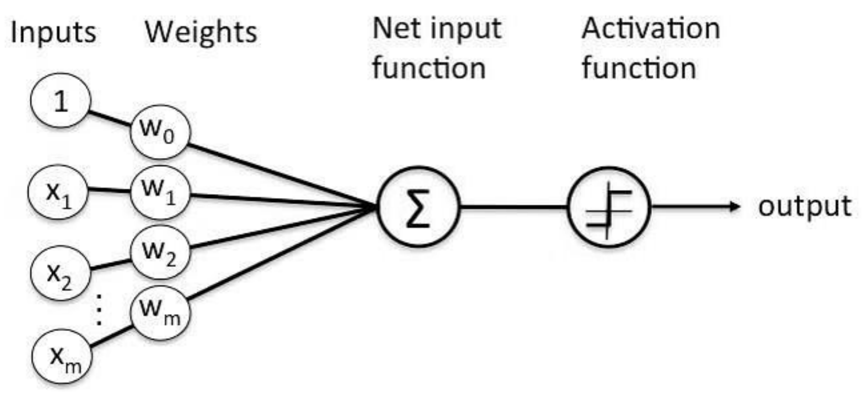
  z = x1 * W1 + x2 * W2 = 1 * 0.5 + 0 * 0.6 = 0.5
- Hence , 0.5 is the value adjusted by the weights of the connection.
- The connections are assumed to be the dendrites in an artificial neuron.
- To process the information, need an activation function (soma) .





Artificial Neuron Working Principle

- Here, use the simple sigmoid function: *f(x)* $= \frac{1}{1+e^{-z}}$
- f (x) = 0.6224
- The above value is the output of the neuron (axon).
- This value needs to be multiplied by W3 .
- 0.6224 * W3 = 0.6224 * 0.2 = 0.12448
- Now, Again apply the activation function to the above value ,
- y = f (0.12448) = 0.5310
- Hence, y (final prediction) = 0.5310
- *In this example the weights were randomly generated.*



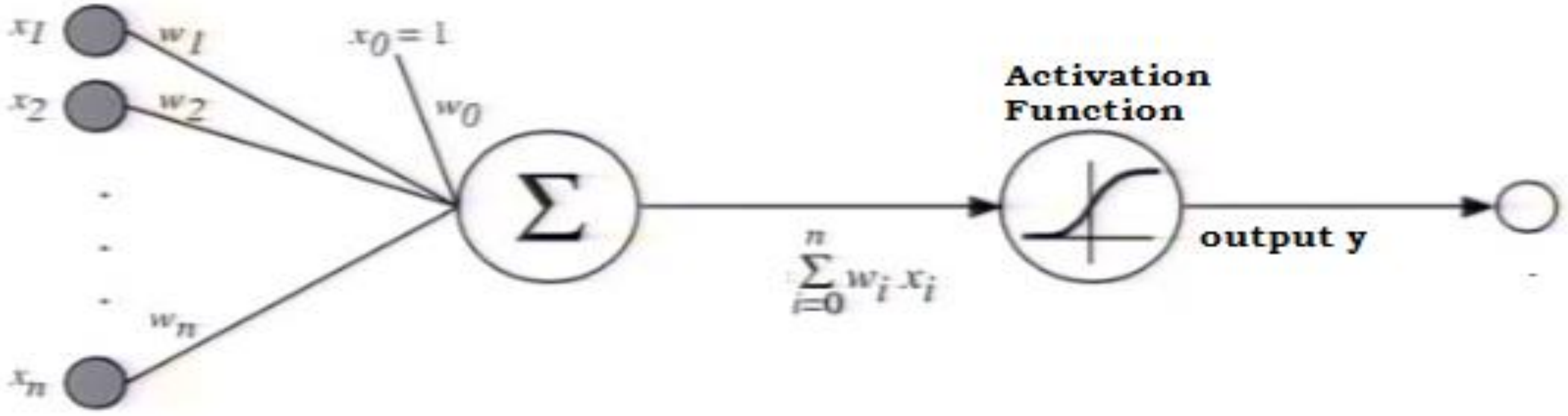
Two types of Perceptron:

Single layer and Multilayer.

- Single layer - Single layer perceptron learns only linearly separable patterns.
- Multilayer - Multilayer perceptron or feedforward NN with two or more layers (more processing power).
- The Perceptron algorithm learns the weights for the input data to draw a linear decision boundary.
- It distinguishes the two linearly separable classes +1 and -1.

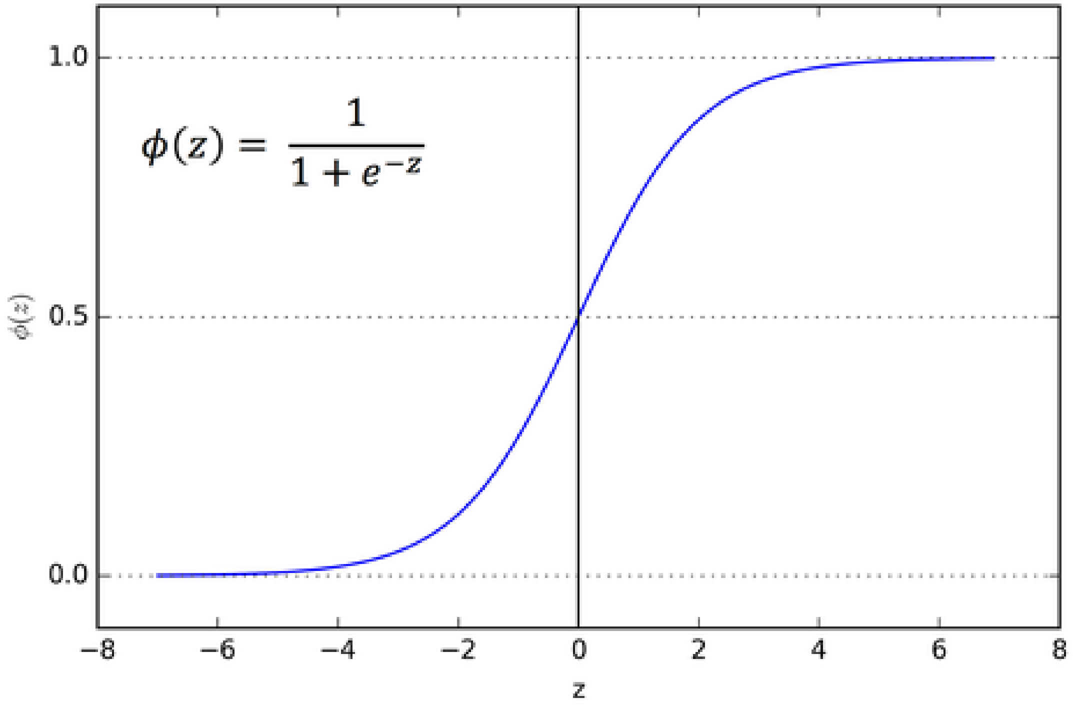
Activation function

- Activation function defines the output of given input or set of inputs or in other terms defines node of the output of node that is given in inputs.
- They will deactivate or activate neurons to get the desired output.
- It also performs a nonlinear transformation on the input to get better results on a complex **neural network**.
- Activation function also helps to normalize the output of any input in the range between **1 to -1**.
- Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points.
- Activation function decides in any neural network that given input or receiving information is relevant or it is irrelevant



1. Sigmoid Activation Function

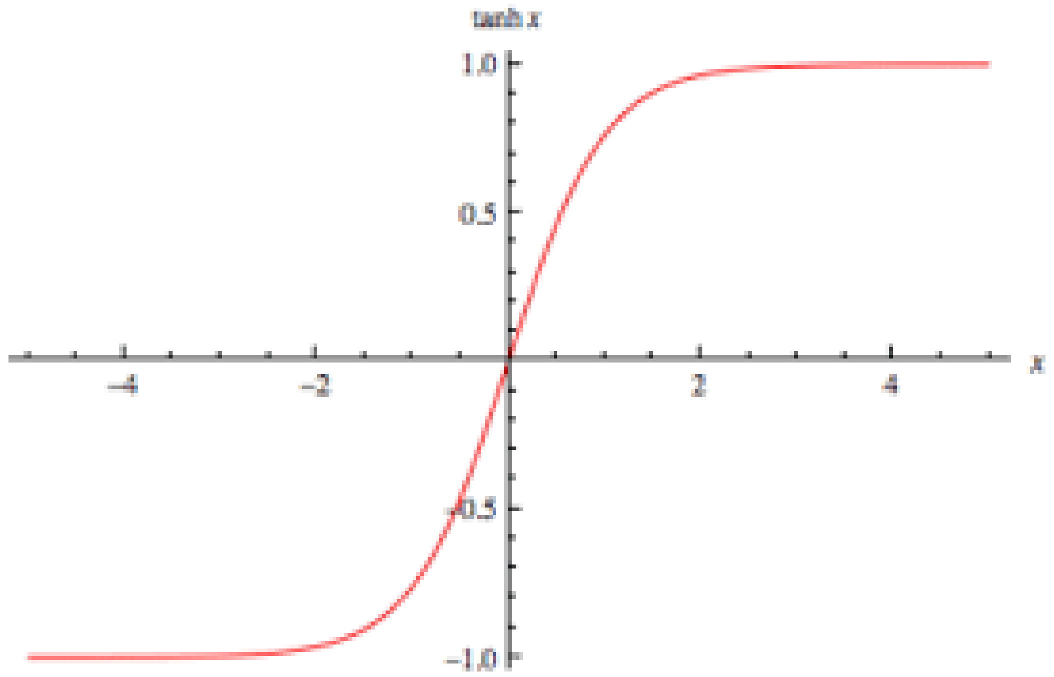
- The sigmoid activation function is a probabilistic approach towards decision making and output range is between **0 to 1** i.e. normalizing the each output of neuron**.**
- Since the range is the minimum, prediction (decision) would be more accurate.
- The equation for the sigmoid function is ***f(x)*** $= \frac{1}{1+e^{-z}}$
- **Disadvantage: Vanishing gradient problem** occurs because it convert large input in between the range of 0 to 1 and therefore their derivatives become much smaller which does not give satisfactory output.
- To solve this problem, ReLU is used which do not have a small derivative problem.



2. Hyperbolic Tangent Activation Function(Tanh)

- This activation function is slightly better than the sigmoid function, like the **sigmoid function.**
- **Zero centered**— Making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
- Range is in between **-1 to 1.**

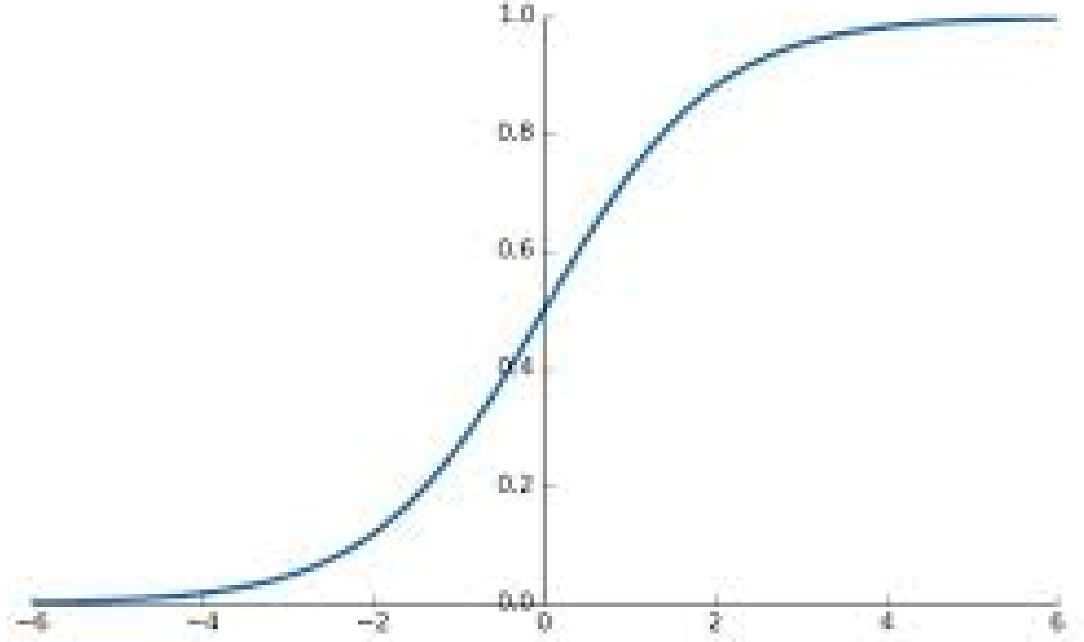
$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



3. Softmax Activation Function

- Softmax is used at the last layer i.e., output to classify inputs into multiple classes. Range is 0 to 1.
- The softmax basically gives value to the input variable according to their weight and the **sum o**f these weights (proportional to each other) is one i.e., sum of all the probabilities will be equal to one.
- For multi-class classification problem, use softmax and cross-entropy along with it.
- Negative inputs will be converted into nonnegative values using exponential function.

$$\phi(z) = \frac{e^{i}}{\sum_{j=0}^{k} e^{j}} \quad \text{where } i=0,1,\ldots.k$$



4. ReLU(Rectified Linear unit) Activation function

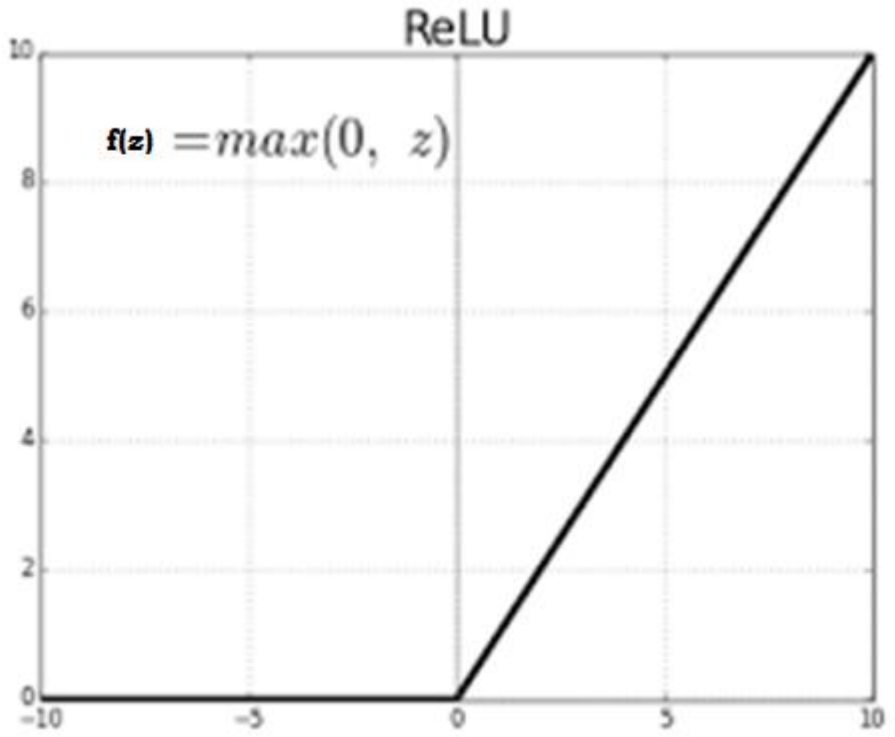
- Rectified linear unit or ReLU is most widely used activation function. Range is from **0 to infinity**.
- It is converging quickly. So, Fast Computation.
- **Disadvantage:**
- ***Dying ReLU*' problem:** when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.
- To avoid this unfitting, use Leaky ReLU function instead of ReLU, in Leaky ReLU range is expanded which enhances the performance.

$$f(x) = \max(0, x),$$

- Where,

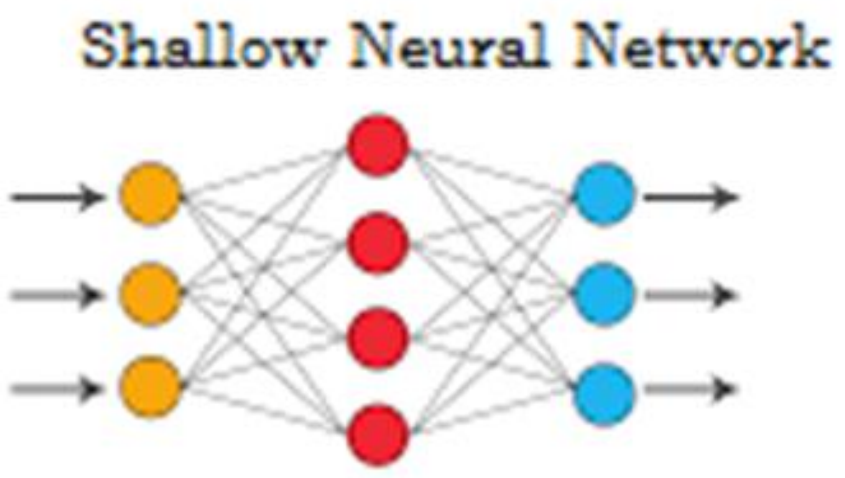
  x -> input to a neuron

  - This function also Ramp function
  - Analogous to half-wave rectifier



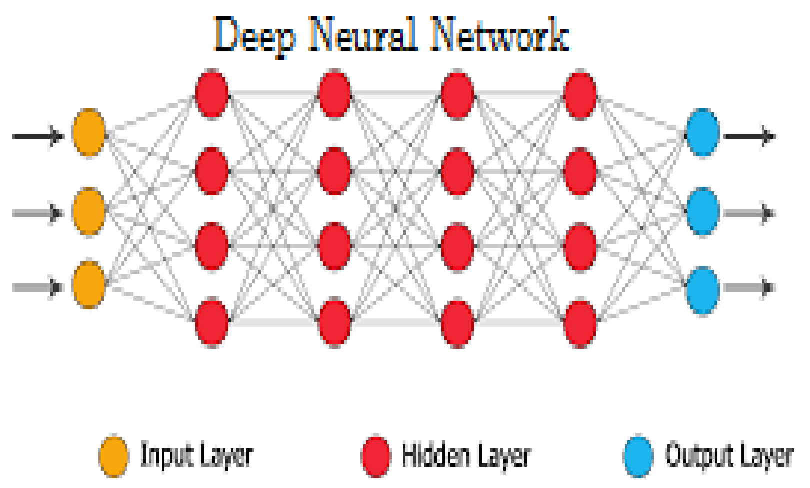
Architecture of Neural Network

- A neural network contains different layers.
- Input layer - The first layer is the, it picks up the input signals and passes them to the next layer.
- Hidden layer - The next layer does all kinds of calculations and feature extractions.
- Often, possibility of more than one hidden layer.
- Output layer - Provides the final result.



Classification of Neural Networks

1. **Shallow neural network**: The Shallow neural network has only **one hidden layer** between the input and output.
2. **Deep neural network**: Deep neural networks have **more than one hidden layer**s. For instance, Google LeNet model for image recognition counts 22 layers.



Building Neural Network Model

1. Weight initialization
2. Forward propagation
3. Loss (or cost) computation
4. Backpropagation
5. Weight update using optimization algorithm.

Weight initialization

1. Zero Initialization (Symmetry Problem)

- When Initialize all weights of a neural network as zero, neural network will become a linear model i.e., all the partial derivatives in backpropagation will be the same for every neuron. So, Parameters are not updated. Hence, should avoid to initialize all weights to zero

2.Weight initialized with too-small values (Vanishing Gradient Problem)

- Model will be converged earlier. It indicated as model performance improves very slowly during training and it stops very early.
- If the initial weights of a neuron are too small relative to the inputs, the gradient of the hidden layers will diminish exponentially in back propagation i.e., vanishing through the layers.
- Use ReLu or leaky ReLu to avoid vanishing gradient.

3. **Too-large** Initialization (Exploding Gradient)

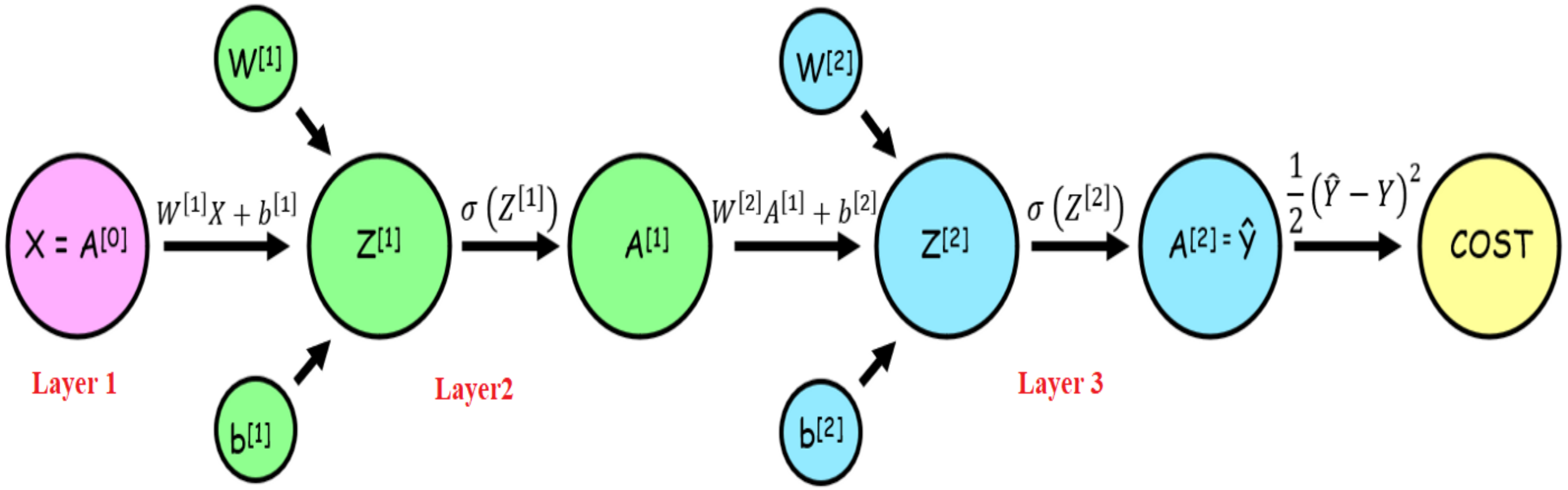
- Loss value will oscillate around minima but unable to converge
- Model involves much change in its loss value on each update due to its instability, may reach infinity sometimes.
- Use ReLu or leaky ReLu to avoid Exploding gradient.

Working of Neural Network

The complete **training process** of a neural network involves two steps.

1. Forward Propagation - Receive input data, process the information, and generate output

- Data are fed into the input layer in the form of numbers. If image is an input, these numerical values denote the intensity of pixels.
- The neurons in the hidden layers apply a few mathematical operations on these values to learn the features.
- To perform these mathematical operations, there are certain parameter values that are randomly initialized.
- Pass these mathematical operations at the hidden layer.
- $z^1 = w^1 \mathrm{x} + b^1$
- $A^1 = \sigma(z^1) = \frac{\mathbf{1}}{\mathbf{1+e^{-z}}}$
- $z^2 = w^2 A^1 + b^2$
- $A^2 = \sigma(z^2) = \frac{\mathbf{1}}{\mathbf{1+e^{-z}}}$
- $Predicted\ output\ (\hat{Y}) = A^2$



Working of Neural Network

3. Loss (or cost) computation

- Once the output is generated, the compare the predicted output with the actual (Ground truth/label) value **y**.
- The optimal value for error is **zero** (no error) i.e., both desired (actual) and predicted results are identical.
- In this context, apply Mean Squared Error (MSE) function:

$$\text{Error (cost) } J = \frac{1}{2}(Actual\ output - \ Predicted\ Output\ )^2$$

- E.g., Error (cost) $J = \frac{1}{2}(0.03\ - 0.0874322143)^2 = 0.356465271$

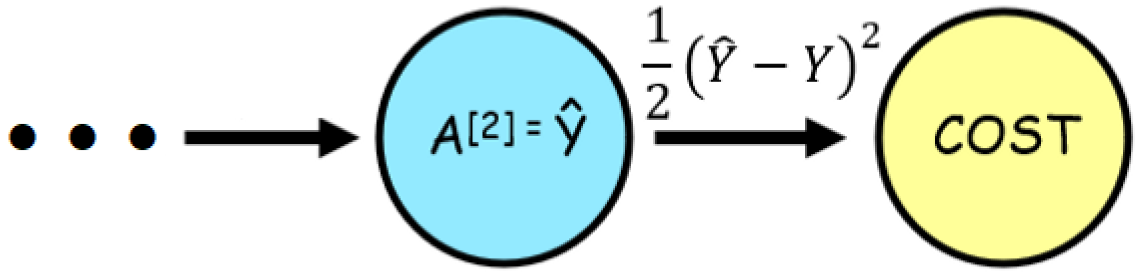


Image:Towardsdatascience.com

Working of Neural Network

3. Backward Propagation

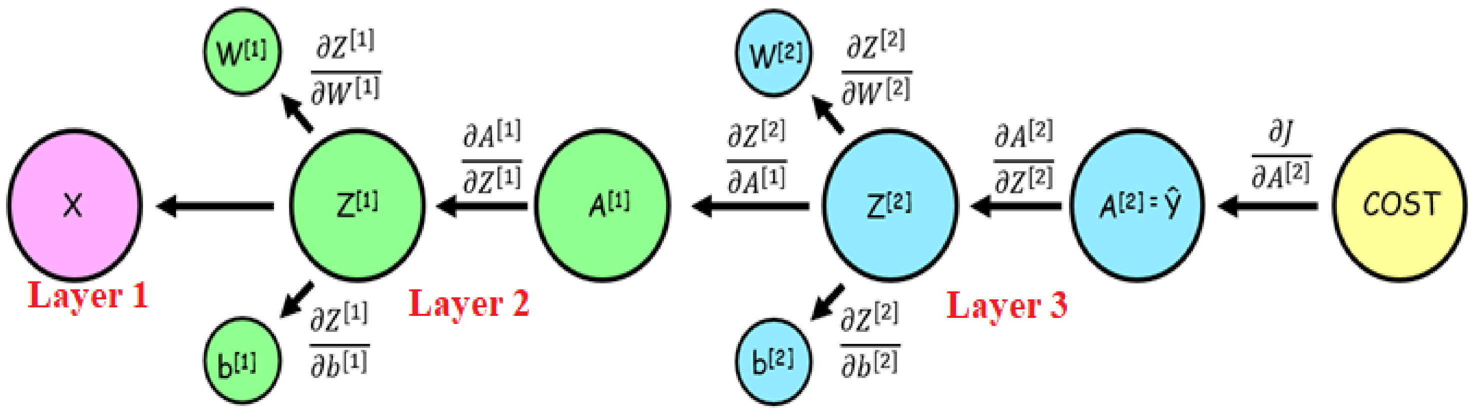
- Calculate the derivatives (gradients) and Update the parameters after each iteration to **reduce** the error.

Calculate the derivatives (gradients)

$$\frac{\partial J}{\partial W^{[2]}}, \quad \frac{\partial J}{\partial b^{[2]}}, \quad \frac{\partial J}{\partial W^{[1]}}, \quad \frac{\partial J}{\partial b^{[1]}}$$

$$z^1 = w^1 \mathrm{x} + b^1; A^1 = \sigma(z^1) = \frac{1}{1+e^{-z^1}}$$

$$z^2 = w^2 A^1 + b^2 ; A^2 = \sigma(z^2) = \frac{1}{1+e^{-z^2}}$$



X
Layer 1
W[1]
$\frac{\partial Z^{[1]}}{\partial W^{[1]}}$
Z[1]
b[1]
$\frac{\partial Z^{[1]}}{\partial b^{[1]}}$
$\frac{\partial A^{[1]}}{\partial Z^{[1]}}$
Layer 2
A[1]
$\frac{\partial Z^{[2]}}{\partial A^{[1]}}$
W[2]
$\frac{\partial Z^{[2]}}{\partial W^{[2]}}$
Z[2]
b[2]
$\frac{\partial Z^{[2]}}{\partial b^{[2]}}$
Layer 3
$\frac{\partial A^{[2]}}{\partial Z^{[2]}}$
A[2] = ŷ
$\frac{\partial J}{\partial A^{[2]}}$
COST

- Based on chain rule, compute the product of the derivatives along all paths connecting the variables.
- Let's consider weights of the third layer w^2 and b^2 gradients:

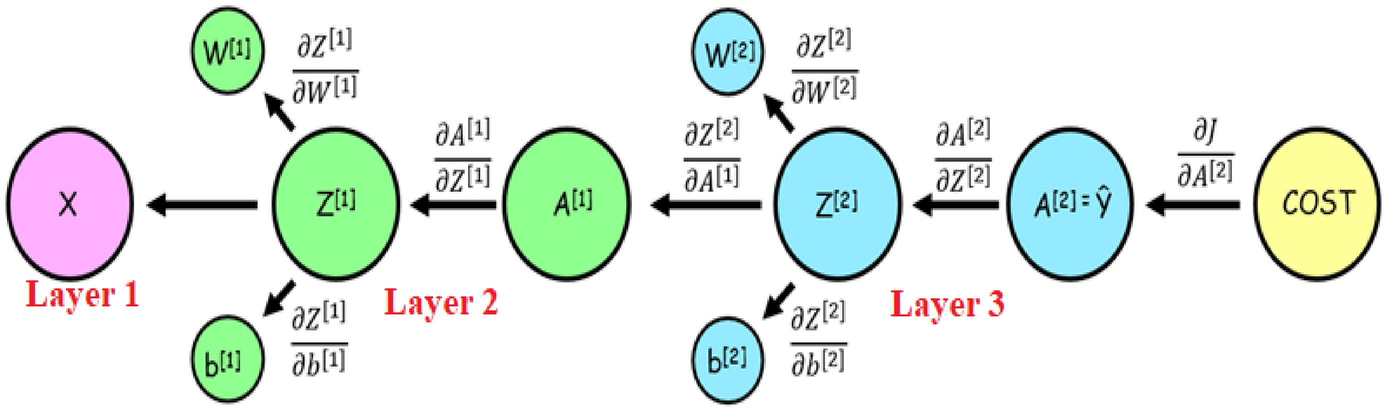
$$\frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial W^{[2]}} = (A^{[2]} - Y) \bullet A^{[2]}(1 - A^{[2]}) \bullet A^{[1]}$$

$$\frac{\partial J}{\partial b^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial b^{[2]}} = (A^{[2]} - Y) \bullet A^{[2]}(1 - A^{[2]}) \bullet 1$$

Image:Towardsdatascience.com

Backward Propagation

- Next, compute the w^1 and b^1 gradients for Layer 2 :
- $z^1 = w^1 x + b^1$
- $A^1 = \sigma(z^1) = \frac{1}{1+e^{-z}}$



$$\frac{\partial J}{\partial W^{[1]}} = \underbrace{\frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}}}_{\text{Previously calculated}} \frac{\partial Z^{[2]}}{\partial A^{[1]}} \frac{\partial A^{[1]}}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial W^{[1]}} = (A^{[2]} - Y) \bullet A^{[2]}(1 - A^{[2]}) \bullet W^{[2]} \bullet A^{[1]}(1 - A^{[1]}) \bullet X$$

$$\frac{\partial J}{\partial b^{[1]}} = \underbrace{\frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}}}_{\text{Previously calculated}} \frac{\partial Z^{[2]}}{\partial A^{[1]}} \frac{\partial A^{[1]}}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial b^{[1]}} = (A^{[2]} - Y) \bullet A^{[2]}(1 - A^{[2]}) \bullet W^{[2]} \bullet A^{[1]}(1 - A^{[1]}) \bullet 1$$

Image:Towardsdatascience.com

Working of Neural Network

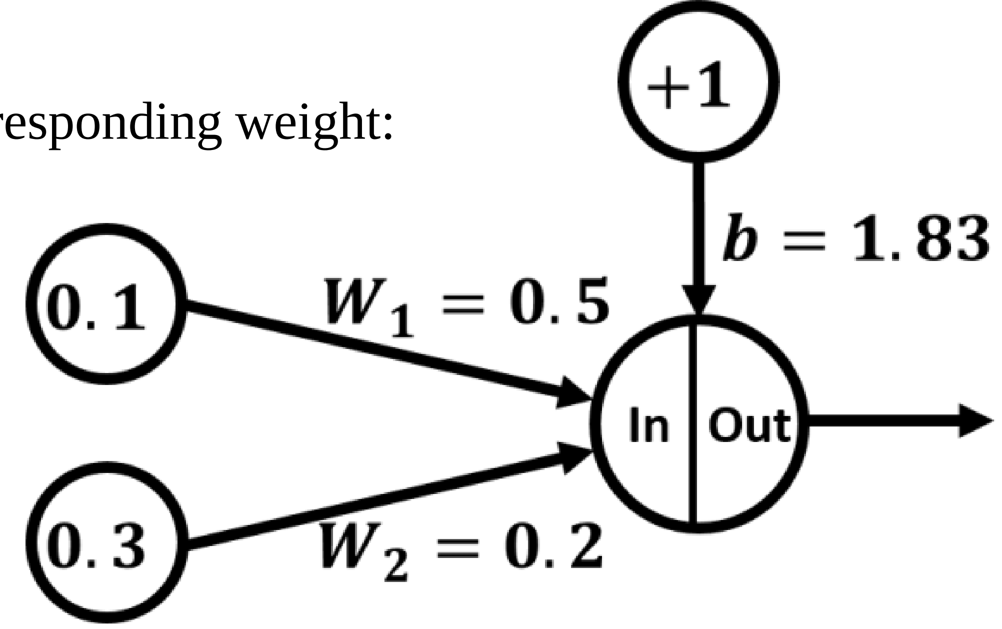
- **5. Update the weights**
- After gradients computed, update the new parameters for the model, and **iterate** (i.e., apply forward propagation) all over again until the model converges.
- Alpha is the learning rate that will determines the convergence rate.

$$W_{new}^{[1]} = W_{old}^{[1]} - \alpha \frac{dJ}{dW^{[1]}} \quad | \quad b_{new}^{[1]} = b_{old}^{[1]} - \alpha \frac{dJ}{db^{[1]}}$$

$$W_{new}^{[2]} = W_{old}^{[2]} - \alpha \frac{dJ}{dW^{[2]}} \quad | \quad b_{new}^{[2]} = b_{old}^{[2]} - \alpha \frac{dJ}{db^{[2]}}$$

Forward Propagation - Example

- Z_1 is the sum of products (SOP) between each input and its corresponding weight:
- $Z_1 = W_1 * X_1 + W_2 * X_2 + b * 1$



X1	X2	Desired Output
0.1	0.3	0.03

Assume that the initial values for both weights and bias are like in the next table.

W1	W2	b
0.5	0.2	1.83

- $Z_1 = W_1 * X_1 + W_2 * X_2 + b * 1$
- $Z_1 = 0.1 * 0.5 + 0.3 * 0.2 + 1.83$
- $Z_1 = 1.94$
- $A_1 = \sigma(Z_1) = 0.874352143.$

$$f(s) = \frac{1}{1 + e^{-s}}$$

$$f(s) = \frac{1}{1 + e^{-1.94}}$$

$$f(s) = \frac{1}{1 + 0.143703949}$$

$$f(s) = \frac{1}{1.143703949}$$

$$f(s) = 0.874352143$$

Backward Propagation- Example

- training steps = n i.e., (0, 1, 2….) ; w_i- Parameter at ith step ; Alpha – learning rate ; Y^n - actual output =0.03;
- $\hat{Y}^n$ or A_1 - predicted output ; n =0 ;b=1.83, w_1 = 0.5. w_2 = 0.2 ; Alpha =0.01; ; Y^n =0.874352143;
- **Inputs:** x_0 or Bias input=+1, x_1 =0.1, x_2 =0.3.
- $Z_1 = W_1 * X_1 + W_2 * X_2 + b * 1 = 0.5* 0.1 + 0.2*0.3+1.83*1 =1.94$
- $Predicted\ Output = A_1 = \sigma(Z_1) = \frac{1}{1+e^{-z^1}} = 0.874352143.$
- $Error\ J = 0.356465271$
- Compute $\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \frac{\partial J}{\partial b}$
- $\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} * \frac{\partial Z_1}{\partial w_1}$
- $\frac{\partial J}{\partial A_1} = \frac{\partial}{\partial A_1}(\frac{1}{2}(Y - A_1)^2) = \frac{1}{2} * 2\ (Y - A_1) * (-1)$ i.e., -1 from differentiate by ($-A_1$)

  $= - (Y - A_1) = - (0.03 - 0.874352143) = 0.844352143$
- $\frac{\partial A_1}{\partial Z_1} = A_1 * (1 - A_1) = 0.874352143 * (1- 0.874352143) = 0.013803732$
- $\frac{\partial Z_1}{\partial w_1} = \frac{\partial(W_1 * X_1 + W_2 * X_2 + b * 1)}{\partial w_1} = X_1 + 0 + 0 = 0.1$
- $\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} * \frac{\partial Z_1}{\partial w_1} = 0.844352143 * 0.013803732 * 0.1 = 0.001165$

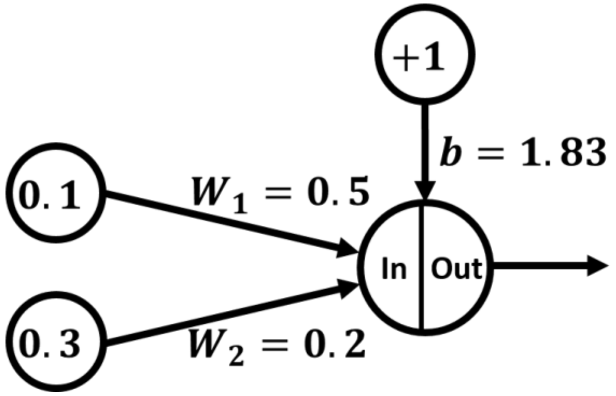


Image:Towardsdatascience.com

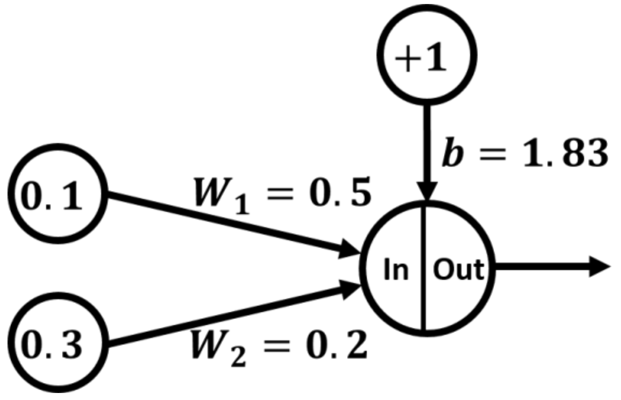
Parameter updation - Example

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} * \frac{\partial Z_1}{\partial w_2}$$

- $\frac{\partial J}{\partial A_1} = 0.844352143$; $\frac{\partial A_1}{\partial Z_1} = 0.013803732$
- $\frac{\partial Z_1}{\partial w_2} = \frac{\partial(W_1 * X_1 + W_2 * X_2 + b * 1)}{\partial w_1} = 0 + X_2 + 0 = 0.3$
- $\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} * \frac{\partial Z_1}{\partial w_2} = 0.844352143 * 0.013803732 * 0.3 = 0.00349656$
- $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} * \frac{\partial Z_1}{\partial b}$

$$\frac{\partial Z_1}{\partial b} = \frac{\partial(W_1 * X_1 + W_2 * X_2 + b * 1)}{\partial b} = 0+0+1 = 1$$

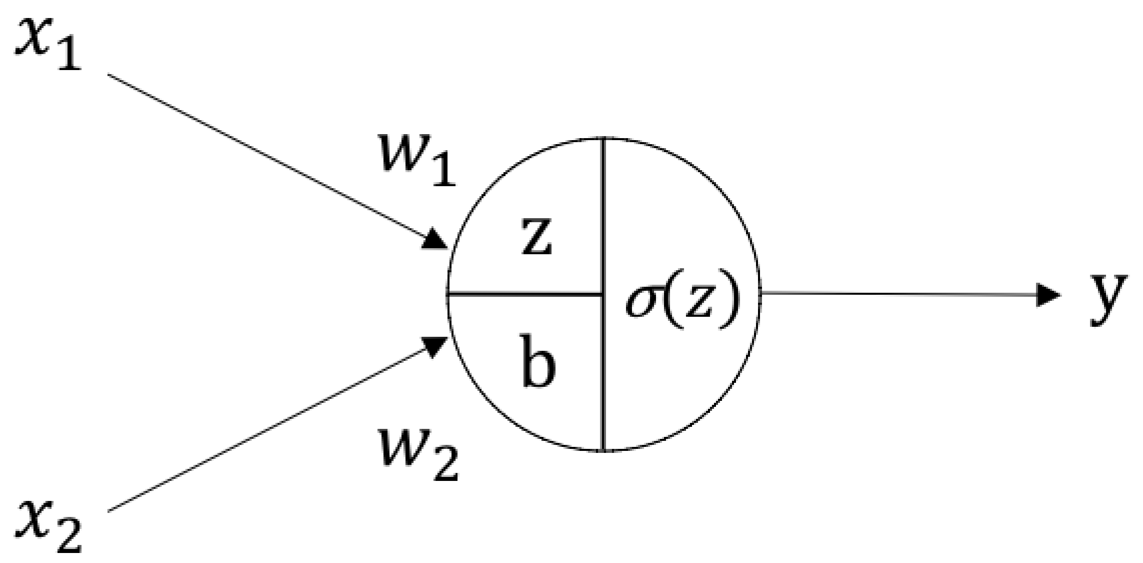
$$\frac{\partial J}{\partial b} = 0.844352143 * 0.013803732 * 1 = 0.013803732$$



Parameter updation:

- $w_1 = w_1 - \alpha\left(\frac{\partial J}{\partial w_1}\right)$ =0.5 – 0.01(0.001165) = 0.498835 ; $w_2 = w_2 - \alpha(\frac{\partial J}{\partial w_2})$= 0.2 – 0.01(0.003496) = 0.19965;
  b = b - $\alpha(\frac{\partial J}{\partial b})$ =1.83 – 0.01(0.01165521) = 1.82988
- Apply these updated new parameters with input and proceed the forward propagation for 2nd iteration until convergence i.e., three consecutive iterations getting same error value.

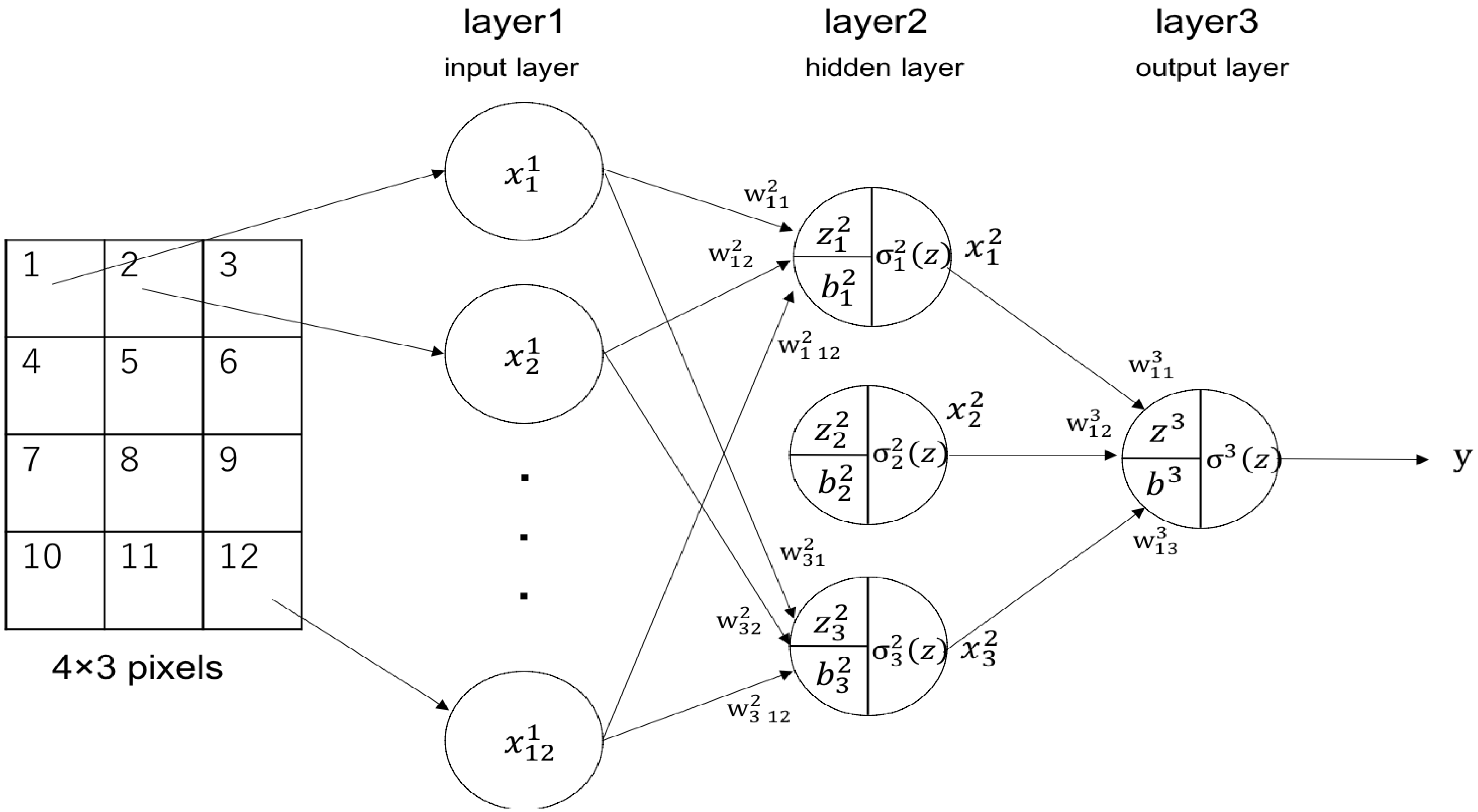
Forward Propagation – Example 2



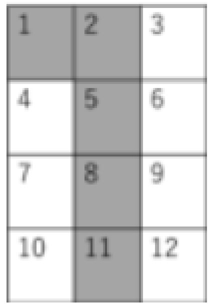
$$z = x_1 w_1 + x_2 w_2 + b$$
$$y = \sigma(z) = \frac{1}{1+\exp(-z)}$$

x1	x2	w1	w2	weighted input z	sigmoid	y
0.1	0.3	0.6	0.2	$z = 0.1 * 0.6 + 0.3 * 0.2 + 0$	$\sigma(z) = \frac{1}{1+\exp(-0.12)} = 0.530$	0.530
0.7	0.2	0.8	0.1	$z = 0.7 * 0.8 + 0.2 * 0.1 + 0$	$\sigma(z) = \frac{1}{1+\exp(-0.58)} = 0.641$	0.641
0.2	0.4	0.9	0.5	$z = 0.2 * 0.9 + 0.4 * 0.5 + 0$	$\sigma(z) = \frac{1}{1+\exp(-0.38)} = 0.594$	0.594

Forward Propagation – Digit Image Example 3



Forward Propagation – Digit Image Example 3

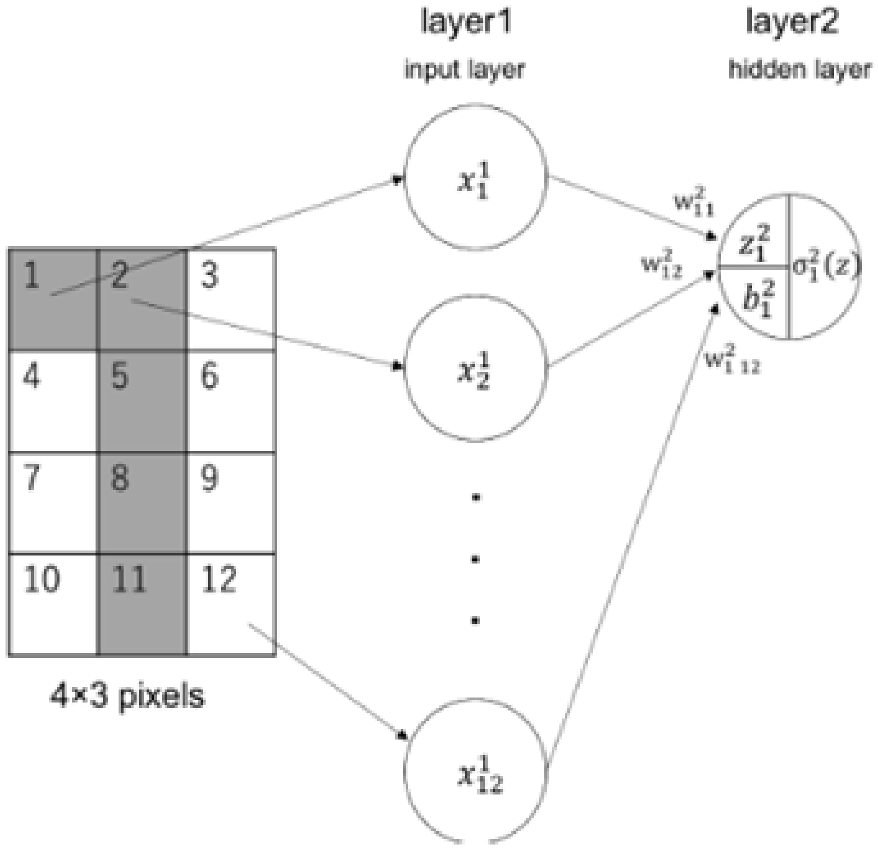


$$x^1_1 = 1,\quad x^1_2 = 1,\quad x^1_3 = 0$$
$$x^1_4 = 0,\quad x^1_5 = 1,\quad x^1_6 = 0$$
$$x^1_7 = 0,\quad x^1_8 = 1,\quad x^1_9 = 0$$
$$x^1_{10} = 0,\quad x^1_{11} = 1,\quad x^1_{12} = 0$$

$$w^2_{11} = 0.48,\ w^2_{12} = 0.29,\ w^2_{13} = 0.58,\ w^2_{14} = 0.61,\ w^2_{15} = 0.47,\ w^2_{16} = 0.18,$$
$$w^2_{17} = 0.02,\ w^2_{18} = 0.49,\ w^2_{19} = 0.67,\ w^2_{1\,10} = 0.62,\ w^2_{1\,11} = 0.017,\ w^2_{1\,12} = 0.28$$
$$b^2_1 = 0$$

$$\begin{aligned}
z^2_1 &= x^1_1 w^2_{11} + x^1_2 w^2_{12} + x^1_3 w^2_{13} + x^1_4 w^2_{14} + x^1_5 w^2_{15} + x^1_6 w^2_{16} + x^1_7 w^2_{17} \\
&\quad + x^1_8 w^2_{18} + x^1_9 w^2_{19} + x^1_{10} w^2_{1\,10} + x^1_{11} w^2_{1\,11} + x^1_{12} w^2_{1\,12} + b^2_1 \\
&= 1 \times w^2_{11} + 1 \times w^2_{12} + 0 \times w^2_{13} + 0 \times w^2_{14} + 1 \times w^2_{15} + 0 \times w^2_{16} + 0 \times w^2_{17} \\
&\quad + 1 \times w^2_{18} + 0 \times w^2_{19} + 0 \times w^2_{1\,10} + 1 \times w^2_{1\,11} + 0 \times w^2_{1\,12} + b^2_1 \\
&= 1 \times 0.48 + 1 \times 0.29 + 0 \times 0.58 + 0 \times 0.61 + 1 \times 0.47 + 0 \times 0.18 + 0 \times 0.02 \\
&\quad + 1 \times 0.49 + 0 \times 0.67 + 0 \times 0.62 + 1 \times 0.017 + 0 \times 0.28 + 0 \\
&= 1 \times 0.48 + 1 \times 0.29 + 1 \times 0.47 + 1 \times 0.49 + 1 \times 0.017 + 0 \\
&= 1.277
\end{aligned}$$

$$\begin{aligned}
x^2_1 = \sigma^2_1(z^2_1) &= \frac{1}{1+\exp(-z^2_1)} \\
&= \frac{1}{1+\exp(-(1.277))} \\
&= 0.782
\end{aligned}$$



Problems in neural network -

Overfitting

- Optimize a model requires to find the best parameters that minimize the loss of the training set.
- Generalization - how the model behaves for unseen data.
- The best method is to have a balanced dataset with sufficient amount of data. Reduce overfitting by regularization.

Network size

- A neural network with too many layers and hidden units are be highly sophisticated (expensive). So, reduce the complexity of the model by reduce its size. There is no best practice to define the number of layers. Start with a small amount of layer and increases its size until find the model overfit.

Weight Regularization

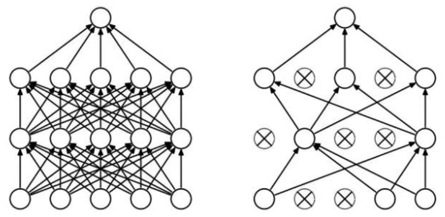
- Prevent overfitting by add constraints to the weights. The constraint forces the network to be small. The constraint is added to the loss function. Two kinds of regularization: L1: Lasso: Cost is proportional to the absolute value of the weight coefficients. L2: Ridge: Cost is proportional to the square of the value of the weight coefficients. $J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$

Early-stopping

- Use validation error to decide when to stop training
- Stop when monitored quantity has not improved after n subsequent epochs

Dropout

Dropout makes some weights will be randomly set to zero. E.g., weights= [0.1, 1.7, 0.7, -0.9]. After apply dropout, it becomes [0.1, 0, 0, -0.9] with randomly distributed 0. The dropout rate parameter controls how many weights to be set to zeroes. Having a rate between 0.2 and 0.5 is common.



Parameter

Learning Rate

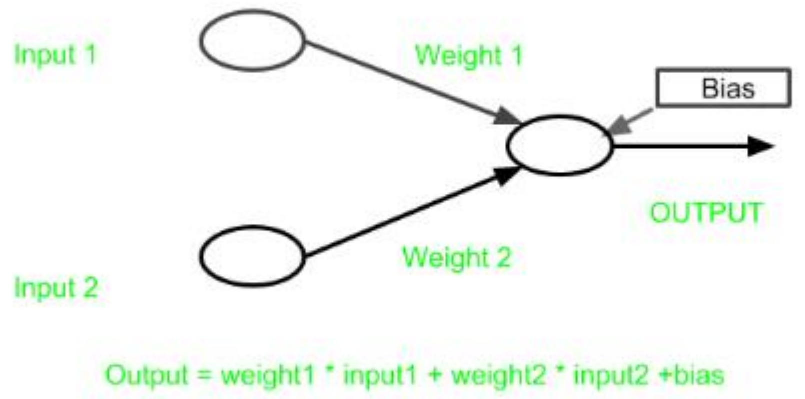
- Learning rate (step size) is hyper-parameter and range between 0.0 and 1.0
- The amount that the weights are updated during training.
- The learning rate controls how quickly the model is adapted to the problem.
- Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.

Bias

- The bias is a node that is always 'on'. i.e. its value is set to 1.
- Bias is like the intercept in a linear equation. It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron.

$$\text{Output} = \text{sum}\ (W * X) + \text{bias}$$

- Therefore, Bias is a constant which helps the model in a way that it can **fit best** for the given data.
- Weight decide how fast the activation function will trigger whereas bias is used to delay the triggering of the activation function.



References

1. Tom M. Mitchell, Machine Learning, McGraw Hill , 2017.
2. EthemAlpaydin, Introduction to Machine Learning (Adaptive Computation and Machine Learning), The MIT Press, 2017.
3. Wikipedia
4. https://www.indowhiz.com/articles/en/the-simple-concept-of-expectation-maximization-em-algorithm/
5. www.towardsdatascience.com