

Unit – 4

Neural Networks and Deep Learning (Artificial Intelligence)

Ms. Zeenal Patel
IT/CS Department
CGPIT, UTU



Topics to be covered

- Network Overview
 - ✓ Biological Neuron
 - ✓ Artificial Neuron
- Perceptron Learning
- Neural Network Representation
- Need for Non-Linear Activation Functions, Cost Function
- Back propagation
- Training & Validation
- Need for Deep representations
- CNN
- RNN

Network Overview

Biological Neuron

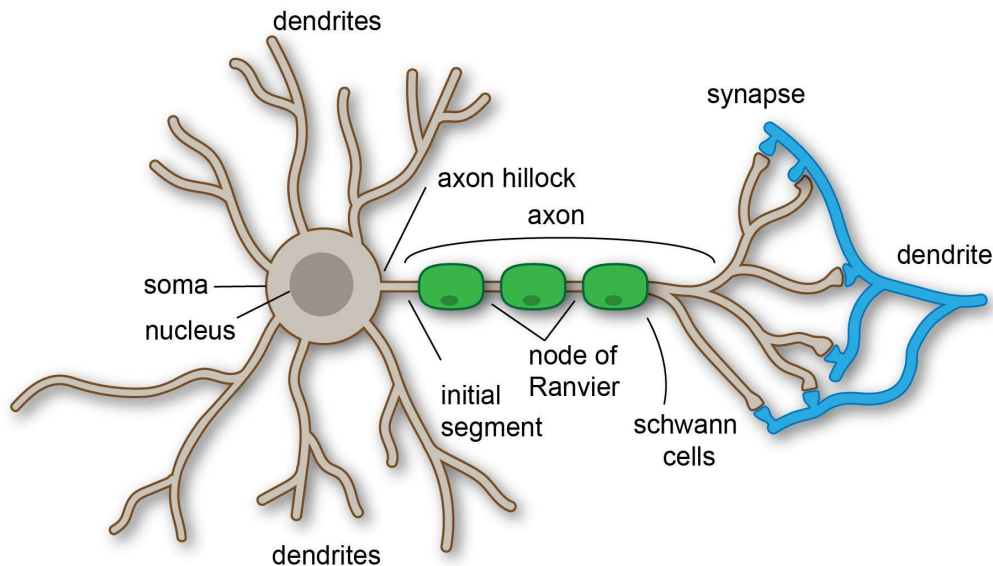


Figure 1.1: Structure of Biological Neuron

- The human nervous system has two main parts –
 - the central nervous system (CNS) consisting of the brain and spinal cord
 - the peripheral nervous system consisting of nerves and ganglia outside the brain and spinal cord.
- The CNS integrates all information, in the form of signals, from the different parts of the body. The peripheral nervous system, on the other hand, connects the CNS with the limbs and organs.
- Neurons are basic structural units of the CNS. A neuron is able to receive, process, and transmit information in the form of chemical and electrical signals.

Biological Neuron

- ▶ It has three main parts to carry out its primary functionality of receiving and transmitting information:
 1. Dendrites – to receive signals from neighbouring neurons.
 2. Soma – main body of the neuron which accumulates the signals coming from the different dendrites. It 'fires' when a sufficient amount of signal is accumulated.
 3. Axon – last part of the neuron which receives signal from soma, once the neuron 'fires', and passes it on to the neighbouring neurons through the axon terminals (to the adjacent dendrite of the neighbouring neurons).
- ▶ There is a very small gap between the axon terminal of one neuron and the adjacent dendrite of the neighbouring neuron. This small gap is known as synapse. The signals transmitted through synapse may be excitatory or inhibitory.

The Artificial Neuron

- ▶ The biological neural network has been modelled in the form of ANN with artificial neurons simulating the function of biological neurons.
- ▶ input signal x_i (x_1, x_2, \dots, x_n) comes to an artificial neuron. Each neuron has three major components:
 - ➔ A set of 'i' synapses having weight w_i . A signal x_i forms the input to the i-th synapse having weight w_i . The value of weight w_i may be positive or negative. A positive weight has an excitatory effect, while a negative weight has an inhibitory effect on the output of the summation junction, y_{sum}
 - ➔ A summation junction for the input signals is weighted by the respective synaptic weight. Because it is a linear combiner or adder of the weighted input signals, the output of the summation junction, y , can be expressed as follows:

$$y_{\text{sum}} = \sum_{i=1}^n w_i x_i$$

The Artificial Neuron

- A threshold activation function (or simply activation function, also called squashing function) results in an output signal only when an input signal exceeding a specific threshold value comes as an input. It is similar in behaviour to the biological neuron which transmits the signal only when the total input signal meets the firing threshold.
- Output of the activation function, y , can be expressed as follows:

$$y_{\text{out}} = f(y_{\text{sum}})$$

The Artificial Neuron

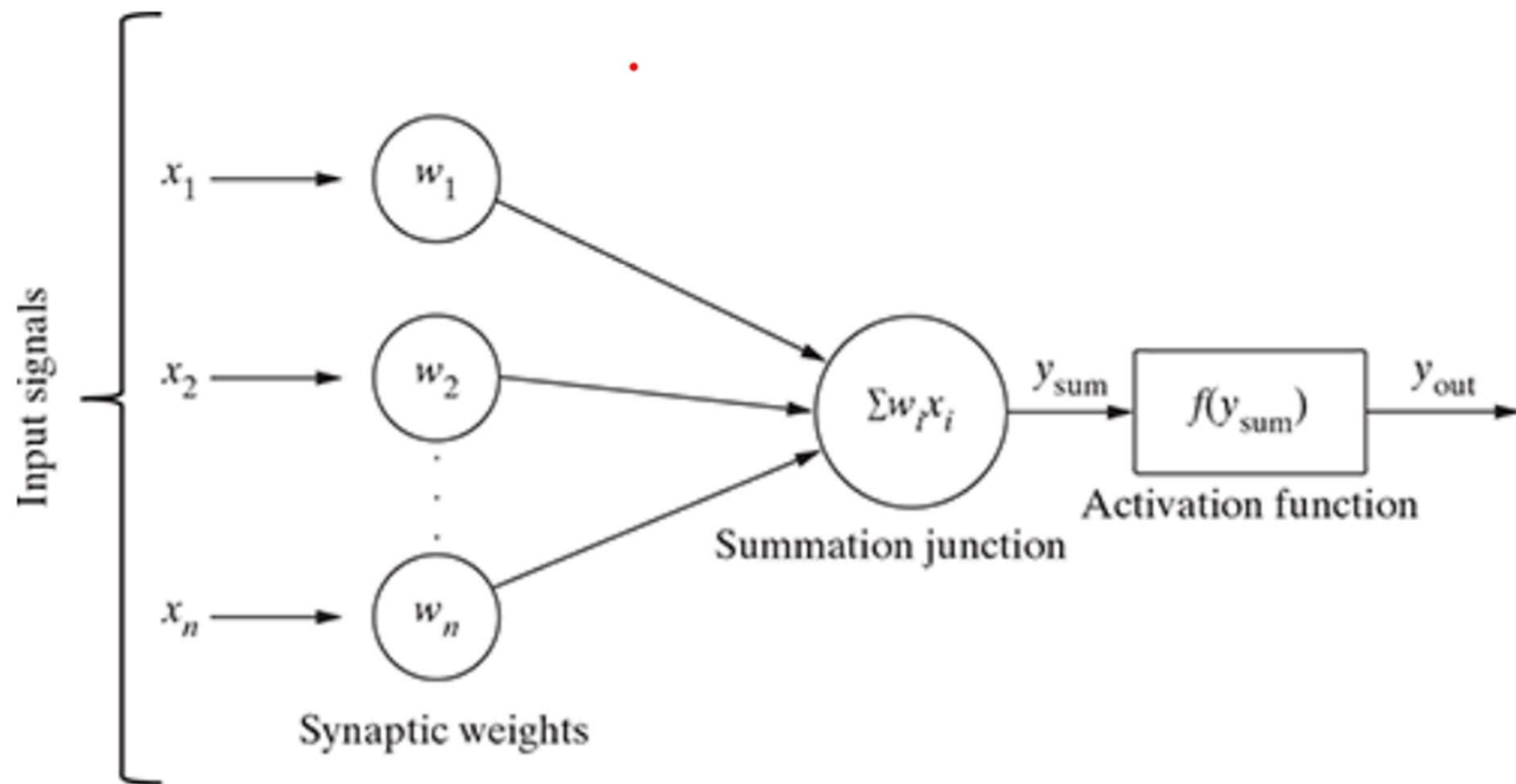


Figure 1.2: Structure of Artificial Neuron

ZP0

csdc
ZEENAL PATEL, 2025-01-24T17:03:41

Perceptron Learning

What is Perceptron?

- ▶ A perceptron is a basic unit in neural networks, mimicking brain neurons.
- ▶ It processes inputs with weighted connections and a bias, producing binary outputs through an activation function.
- ▶ Primarily used in binary classification tasks, perceptrons were instrumental in advancing neural network development and deep learning techniques.
- ▶ The Perceptron model in machine learning is characterized by the following key points:
- ▶ **Binary Linear Classifier:** The Perceptron is a type of binary classifier that assigns input data points to one of two possible categories.
- ▶ **Input Processing:** It takes multiple input signals and processes them, each multiplied by a corresponding weight. The inputs are aggregated, and the model produces a single output.

What is Perceptron?

- ▶ **Activation Function:** The Perceptron uses an activation function, typically a step function, to determine the output based on the aggregated inputs and weights. If the result exceeds a certain threshold, the output is one category; otherwise, it's the other.
- ▶ **Training Process:** During training, the model adjusts its weights based on the error in its predictions compared to the actual outcomes. This adjustment helps improve the model's accuracy over time.
- ▶ **Single-Layer Model:** The Perceptron is a single-layer neural network since it has only one layer of output after processing the inputs.

What is Perceptron?

- ▶ **Rosenblatt's perceptron is basically a binary classifier. The perceptron consists of 3 main parts:**
- ▶ **Activation function:** The activation function determines whether the neuron will fire or not. At its simplest, the activation function is a step function, but based on the scenario, different activation functions can be used.
- ▶ **Input nodes or input layer:** The input layer takes the initial data into the system for further processing. Each input node is associated with a numerical value. It can take any real value.
- ▶ **Weights and bias:** Weight parameters represent the strength of the connection between units. Higher is the weight, stronger is the influence of the associated input neuron to decide the output. Bias plays the same as the intercept in a linear equation.

Working of Perceptron

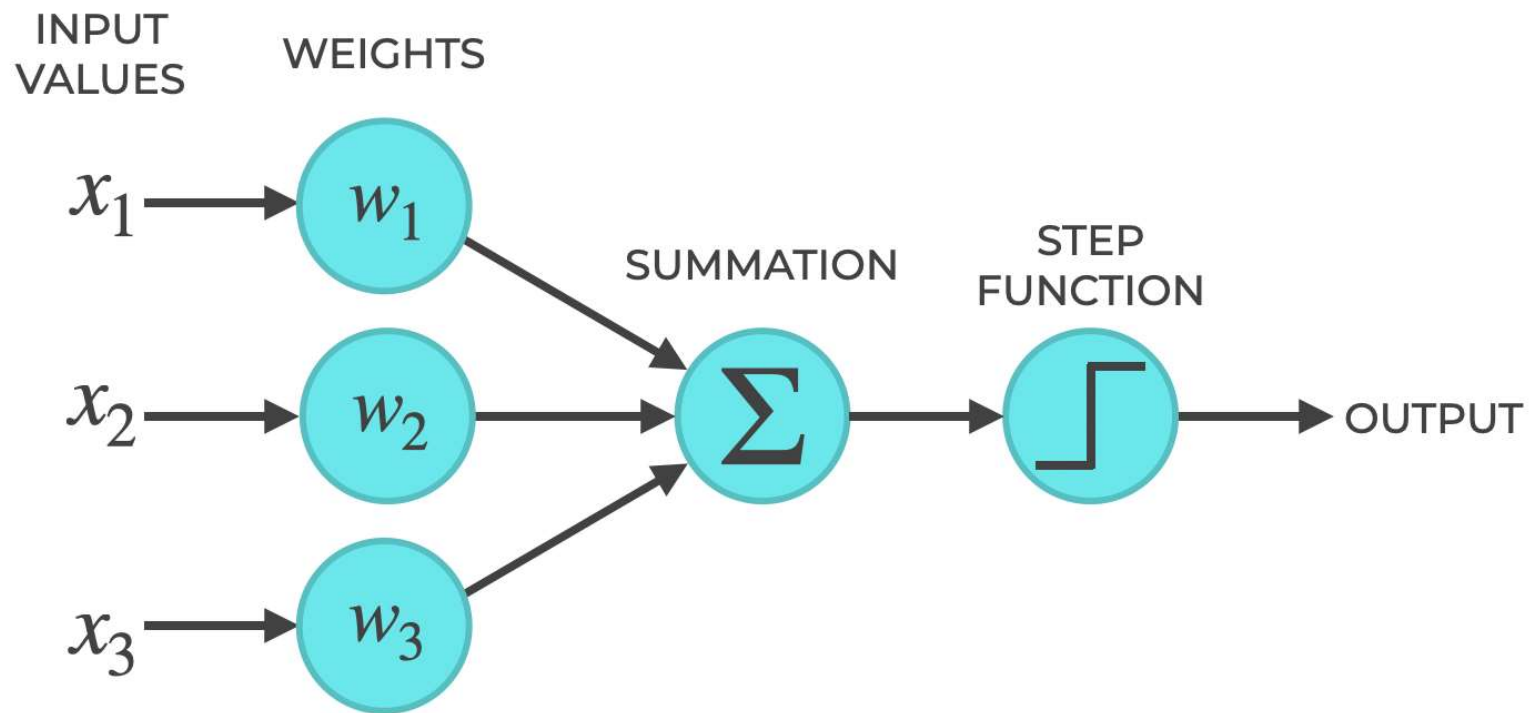


Figure 1.3: Working of Perceptron

Working of Perceptron

- ▶ In the first step, all the input values are multiplied with their respective weights and added together.
- ▶ The result obtained is called weighted sum $\sum w_i * x_i$, or stated differently, $x_1 * w_1 + x_2 * w_2 + \dots w_n * x_n$.
- ▶ This sum gives an appropriate representation of the inputs based on their importance. Additionally, a bias term b is added to this sum $\sum w_i * x_i + b$.
- ▶ Bias serves as another model parameter (in addition to weights) that can be tuned to improve the model's performance.
- ▶ In the second step, an activation function f is applied over the above sum $\sum w_i * x_i + b$ to obtain output $Y = f(\sum w_i * x_i + b)$.
- ▶ Depending upon the scenario and the activation function used, the Output is either binary $\{1, 0\}$ or a continuous value.

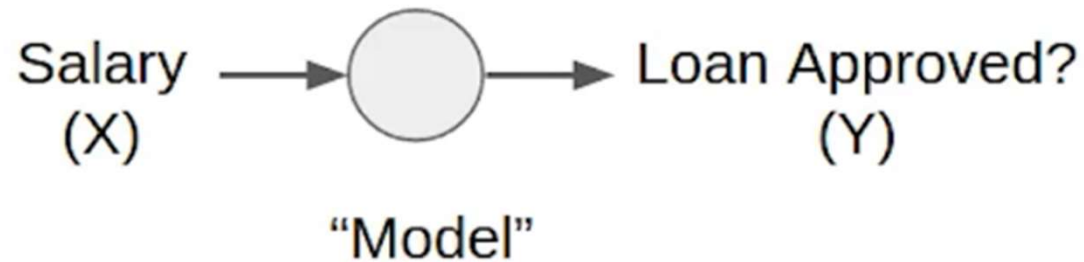
Example of Preceptron

- ▶ Our aim here is to predict whether the loan should be approved or not, depending on the salary of a person.



Example of Preceptron

- ▶ In order to do that will need to build a model that takes the salary of the person as an input and predicts whether the loan should be approved for the person or not.



Example of Preceptron

- Suppose your bank wants to reduce the risk of loan default and hence decides to roll out loans to only such individuals who have a monthly salary of 50000 and above.

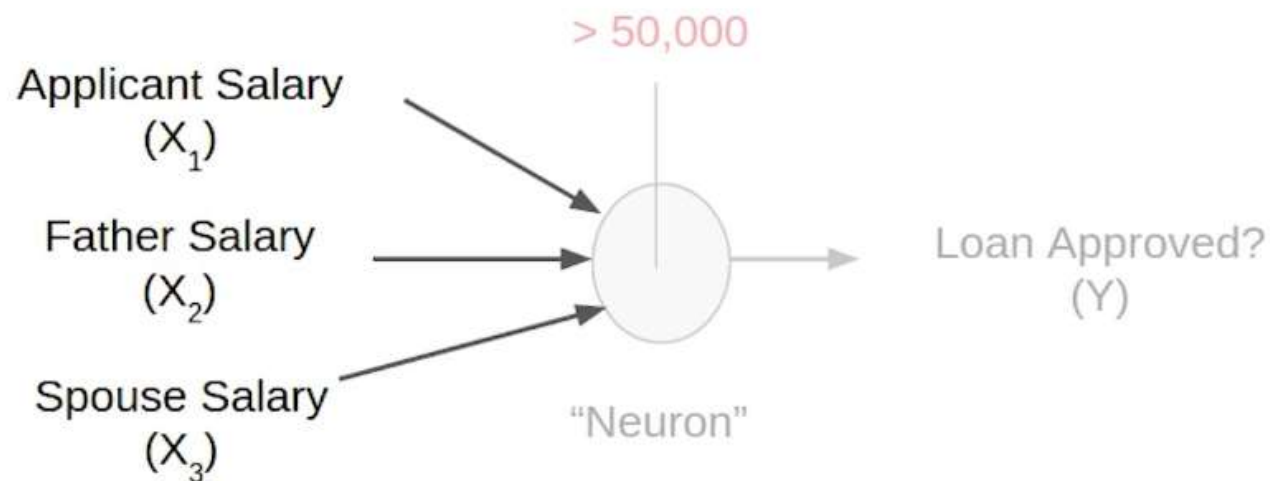


Example of Preceptron

- ▶ In this case, we want our model to learn to check whether the salary input which is represented as X here, is greater than 50000 or not. Here are the tasks we want our model to perform-
- ▶ The first thing is it should take in the salary as input.
- ▶ Next, it has to check whether the given salaries are greater than 50,000 or not.
- ▶ And if the salary's more than 50,000 only then give output as "Yes".
- ▶ Effectively, this model takes in some input, processes it, and generates an output. This is similar to what happens in a biological neuron:
- ▶ We have a single input which is salary but in general, we can have multiple features just like the applicant's salary, his/her father's salary, spouse's salary that can be deciding factors to approve the Loan.
- ▶ And our neurons will take in all of these features as input in order to make decisions.

Example of Preceptron

- ▶ This is to say that the neuron will have multiple inputs. This is similar to the multiple dendrites that we saw in the biological neuron.
- ▶ that there are multiple salary features about the person we'll take all of them into account as they represent the total income of the household. We can sum them up and check if the total income of the household crosses the threshold or not.
- ▶ So, the Total Income = Applicant Salary(X_1) + Father Salary(X_2) + Spouse Salary(X_3)



Example of Preceptron

- ▶ We need to compare this Total Income with the Threshold. Here is the equation representing the same:

$$X_1 + X_2 + X_3 > \text{threshold}$$

- ▶ We have X_1 , X_2 , and X_3 as input features and we want to check if their sum crosses the particular threshold, which is 50000 in our case. Now if you bring this threshold to the left side of the equation it will become something like this:

$$X_1 + X_2 + X_3 - \text{threshold} > 0$$

- ▶ and if we represent this whole quantity which is “- threshold” with a new term **Bias**, the updated equation would look something like:

$$X_1 + X_2 + X_3 + \text{bias} > 0$$

- ▶ this will have the sum of four quantities which are X_1 , X_2 , and X_3 , and note that the bias is actually “- threshold”.

Example of Preceptron

- ▶ Now this quantity which is Bias, although we have selected it arbitrarily here, it is actually something that neuron learn from the underlined data. If the input exceeds the magnitude of the bias, we want the neuron to give the output as “YES”. That means the loan can be approved by this person. This event is known as the firing of a neuron. If want to write this relationship using equations, we can use the following equations:

if $X_1 + X_2 + X_3 + \text{bias} > 0$ then output should be 1

if $X_1 + X_2 + X_3 + \text{bias} \leq 0$ then output should be 0

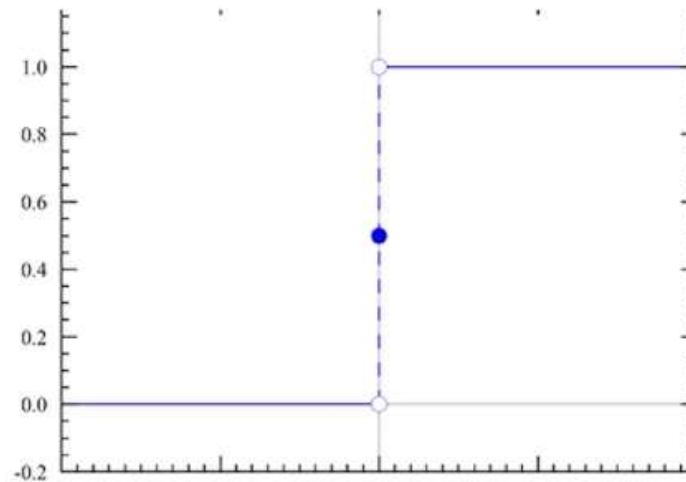
- ▶ We will say that output should be 1 when this equation is true and output should be 0 in all other cases. These two equations can be represented in the form of a function. Let's see how:

$$Z = X_1 + X_2 + X_3 + \text{bias}$$

Output = will be 1 if $(Z = X_1 + X_2 + X_3 + \text{bias}) > 0$, it will be 0 otherwise.

Example of Preceptron

- ▶ So here we have the sum of the features X_1, X_2, X_3 , and bias represented as Z and we want our output to be 1 if the Z is greater than 0, otherwise 0.
- ▶ So we can use a **Step Function** here and this is the graph of the step function:



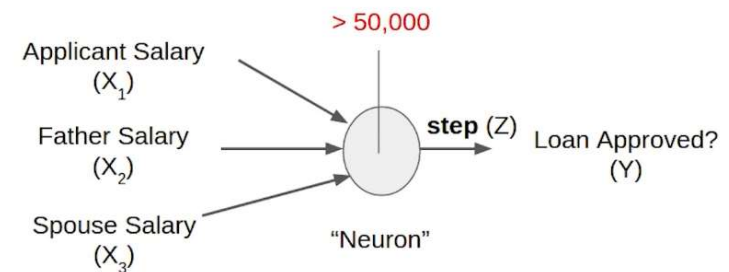
$$\text{Output} = \begin{cases} 1, & Z > 0 \\ 0, & Z \leq 0 \end{cases}$$

Example of Preceptron

- It basically gives us the output 1 for any value greater than zero and gives an output 0 for any value less than zero. So in order to find output, we will apply the step function on Z here. We have denoted this step function as following;

Output = step function (Z) or Output = **step** (Z)

- This step function in this case is used to scale the output of the neuron and in Deep Learning we have an option of choosing such functions to apply to the output of the neurons. They are known as **Activation Functions**. So when we use the step function as the activation function for a neuron it is called a **Perceptron**.



Perceptron Model

Types of Preceptron

- ▶ Based on the layers, Perceptron models are divided into two types. These are as follows:
 1. Single-layer Perceptron Model
 2. Multi-layer Perceptron model

Single-layer Perceptron Model

- ▶ The single-layer perceptron consists of an input layer and an output layer without any hidden layers. Each input feature is assigned a weight, and the weighted sum is passed through an activation function to produce an output.
- ▶ It is ideal for solving linearly separable problems, where the data can be split into two categories with a straight line (or hyperplane).
- ▶ **Strengths:**
 - ↳ Simple to understand and easy to implement.
 - ↳ Efficient for problems with straightforward, linearly separable data.
- ▶ **Limitations:**
 - ↳ Cannot handle non-linear problems or complex data patterns.
 - ↳ Limited ability to learn and model intricate relationships in the data.

Diagram

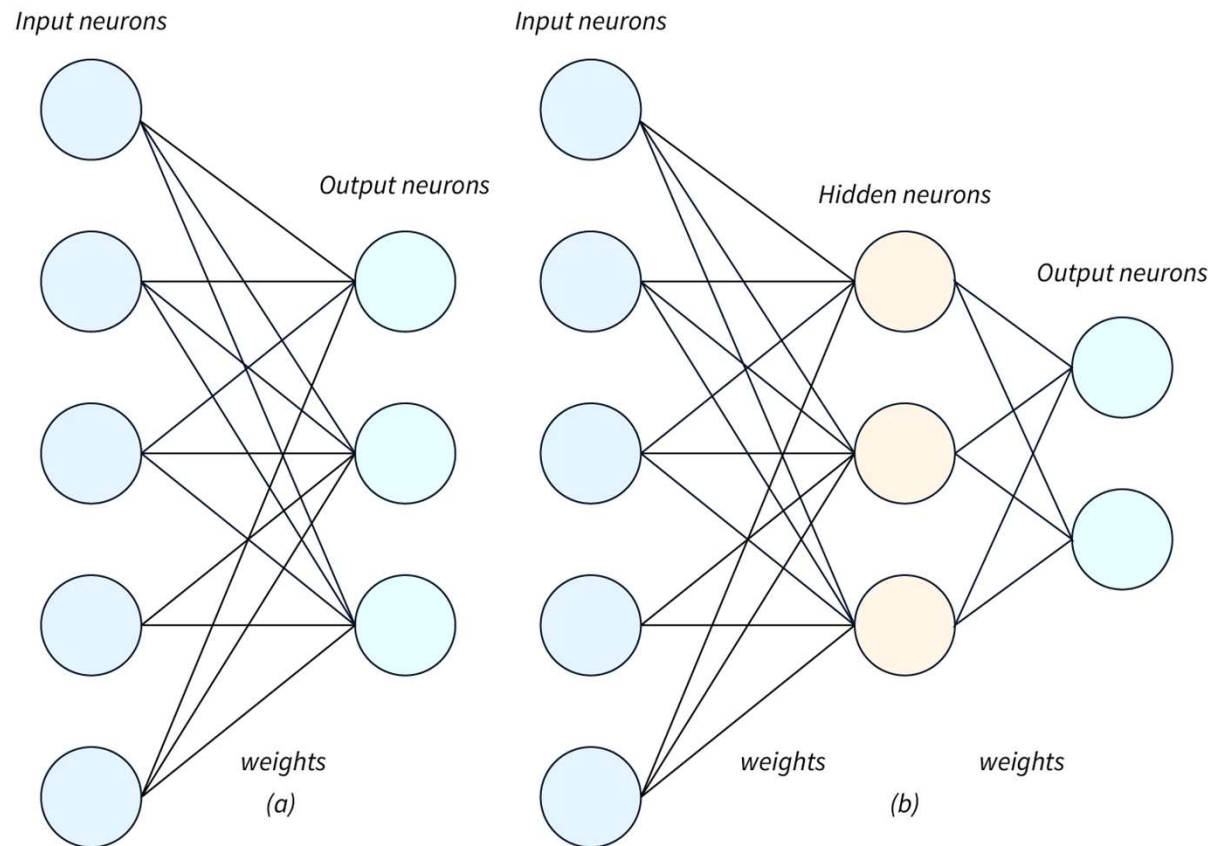


Figure 1.4: a) Single layer Perceptron b) Multi-layer Perceptron (MLP)

Multi-Layer Perceptron (MLP) Model

- ▶ The multi-layer perceptron is an extension of the single-layer perceptron and includes one or more hidden layers between the input and output layers. Each hidden layer allows the network to transform the input data through various stages, enabling it to learn complex patterns.
- ▶ MLPs can address non-linearly separable problems by using multiple layers and non-linear activation functions (e.g., sigmoid, ReLU) to create decision boundaries that can curve and adapt to the data's structure.
- ▶ **Strengths:**
 - ↳ Can solve complex problems by learning deeper and more abstract representations of data.
 - ↳ Capable of handling intricate relationships and patterns that a single-layer perceptron cannot.

Multi-Layer Perceptron (MLP) Model

► Limitations:

- ➔ More computationally intensive, requiring greater data and processing power for training.
- ➔ Training can be more complex due to the need for backpropagation and optimization algorithms.

► The hidden layers in the multi-layer perception model are crucial for enabling the network to process non-linear data. They transform the input data through various weights and activation functions, allowing the network to recognize intricate patterns.

► Here are a few applications of the multi-layer perception model:

► **Image Recognition:** In tasks such as identifying objects in images, MLPs learn to detect and classify features at different levels of complexity.

► **Medical Diagnosis:** MLPs can analyze patient data and identify non-linear relationships to predict conditions such as diabetes, cancer, or heart disease.

Activation Functions

What is Activation Function?

- ▶ Activation functions determine whether or not a neuron should be activated based on its input to the network.
- ▶ These functions use mathematical operations to decide if the input is important for prediction. If an input is deemed important, the function “activates” the neuron.
- ▶ An activation function produces an output using a set of input values given to a node or layer. A node in a neural network is similar to a neuron in the human brain, receiving input signals (external stimuli) and reacting to them.
- ▶ The brain processes input signals and decides whether or not to activate a neuron based on pathways that have been built up over time and the intensity with which the neuron fires.
- ▶ Activation functions in deep learning perform a similar role. The main purpose of an activation function is to transform the summed weighted input from a node into an output value that is passed on to the next hidden layer or used as the final output.

Why are Activation Functions Important?

- ▶ Most activation functions are non-linear. This allows neural networks to "learn" features about a dataset (i.e. how different pixels make up a feature in an image).
- ▶ Without non-linear activation functions, neural networks would only be able to learn linear and affine functions.
- ▶ Why is this an issue? It is an issue because the linear and affine functions cannot capture the complex and non-linear patterns that often exist in real-world data.
- ▶ Activation functions also help map an input stream of unknown distribution and scale to a known one.
- ▶ This stabilizes the training process and maps values to the desired output for non-regression tasks like classification, generative modeling or reinforcement learning.
- ▶ Non-linear activation functions introduce complexity into neural networks and enable them to perform more advanced tasks.

Types of Activation Functions

► the most popular types of neural network activation functions to solidify our knowledge of activation functions in practice. The three most popular functions are:

1. Binary step
2. Linear activation
3. Non-linear activation

Binary Step Activation Functions

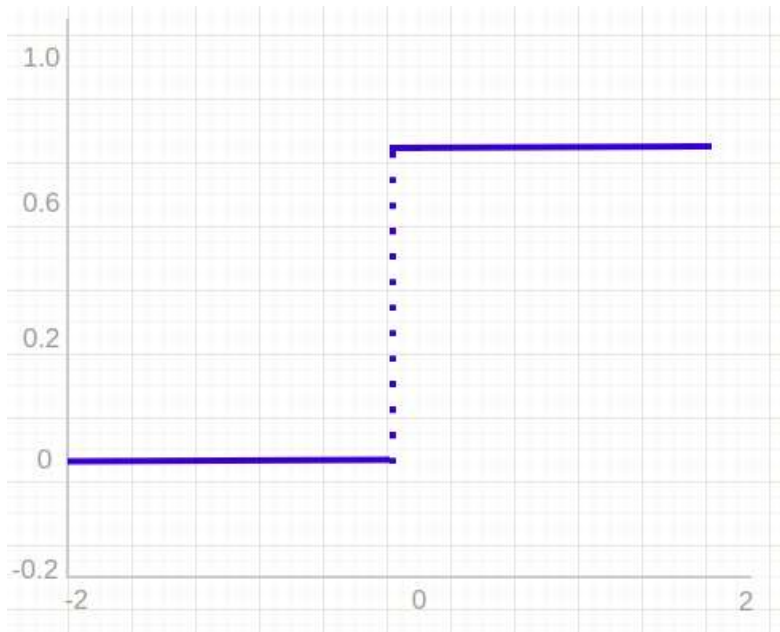


Figure 1.5: Binary Step Function

- ▶ The binary step function uses a threshold value to determine whether or not a neuron should be activated.
- ▶ The input received by the activation function is compared to the threshold. If the input is greater than the threshold, the neuron is activated and its output is passed on to the next hidden layer.
- ▶ If the input is less than the threshold, the neuron is deactivated and its output is not passed on.
- ▶ The binary step function cannot provide multi-value outputs. This means that it is unsuitable for solving multi-class classification problems.
- ▶ Moreover, it is not differentiable, which means that it cannot be used with gradient-based optimization algorithms and this leads to a difficult training.

Linear Activation Function

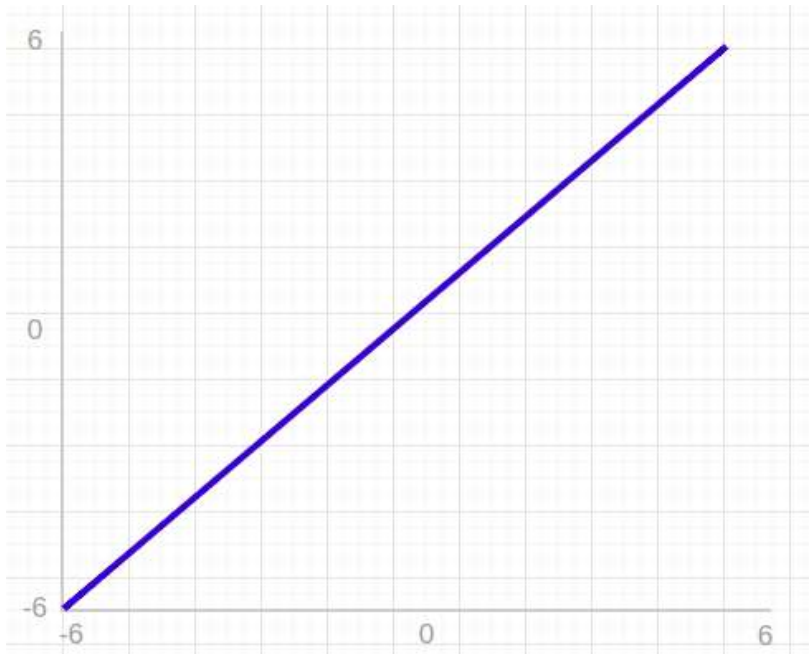


Figure 1.6: Linear Activation Function

- ▶ The linear activation function, also referred to as "no activation" or "identity function," is a function where the activation is directly proportional to the input.
- ▶ This function does not modify the weighted sum of the input and simply returns the value it was given. You may be familiar with identity functions from lambda calculus.
- ▶ The linear activation function is equivalent to a linear regression model. This is because the linear activation function simply outputs the input that it receives, without applying any transformation.

Linear Activation Function

- ▶ In a neural network, the output of a neuron is computed using the following equation:

$$output = activation(inputs \times weights + bias)$$

- ▶ If the activation function is a linear function, then the output will be a linear combination of the inputs and this would be the same equation that is used in linear regression to model the relationships between the input and output variables.
- ▶ **Limitations:**
 - ➔ this function cannot be used with backpropagation because the derivative of the function is a constant with no relation to the input, and it causes all layers of the neural network to collapse into one.
 - ➔ no matter how many layers are present, the last layer will be a linear function of the first layer when a linear activation function is used. This effectively reduces the neural network to a single layer.

Non-Linear Activation Functions

- ▶ non-linear activation functions address the limitations of linear activation functions by enabling backpropagation and the stacking of multiple layers of neurons.
- ▶ This is a standard procedure in many state-of-the-art computer vision and advanced machine learning algorithms.
- ▶ Non-linear functions allow the derivative function to be related to the input, which allows a model to adjust weights in the input neurons to improve prediction.
- ▶ Non-linear activation functions also allow the output to be a non-linear combination of inputs passed through multiple layers. This enables neural networks to model non-linear relationships in the data.
- ▶ Non-linear functions are one of the most useful things in machine learning and computer vision

Sigmoid Function

- ▶ The sigmoid activation function maps an input stream to the range $(0, 1)$. Unlike the step function, which only outputs 0 or 1, sigmoid outputs a range of values between 0 and 1, excluding 0 and 1 themselves
- ▶ The sigmoid function is useful in tasks like binary classification.
- ▶ It has significant drawbacks when it comes to backpropagation. In particular, it suffers from both the **vanishing gradient or exploding gradient**.
- ▶ The vanishing gradient problem happens when the gradients get smaller and smaller as the layers get deeper. This can make it difficult for the neural network to learn as the optimizer updates to the weights will be very small and this will lead to the network to stop learning.

Sigmoid Function

- ▶ The exploding gradient problem happens when the gradients get larger and larger as the layers get deeper. This can cause the weights to become very large, which can lead to numerical instability and cause the model to perform poorly.
- ▶ The output range of $[0, 1]$ is not centered at 0, which can also cause issues during backpropagation.
- ▶ Finally, the use of exponential functions can be computationally expensive and slow down the network.
- ▶ There are two types of sigmoid function:
 - ↳ Binary sigmoid function.
 - ↳ Bipolar sigmoid function

Sigmoid Function

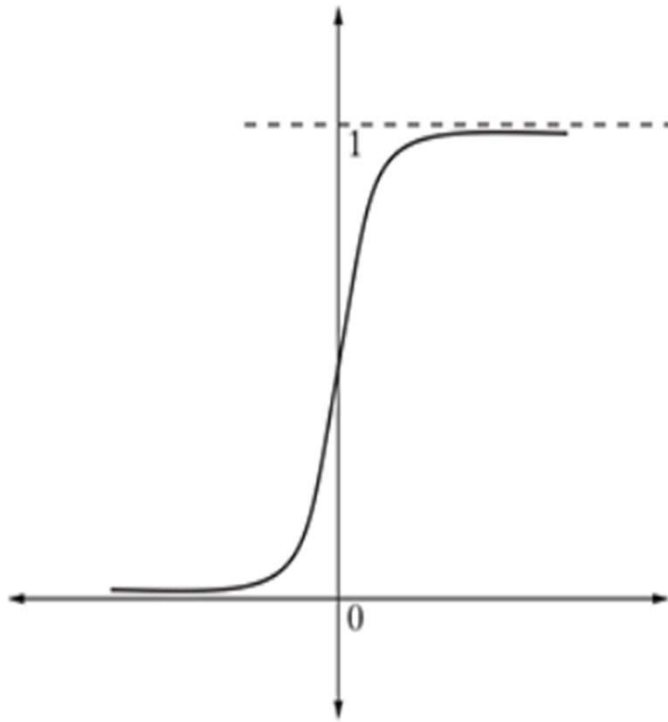


Figure 1.7: Binary Sigmoid Function

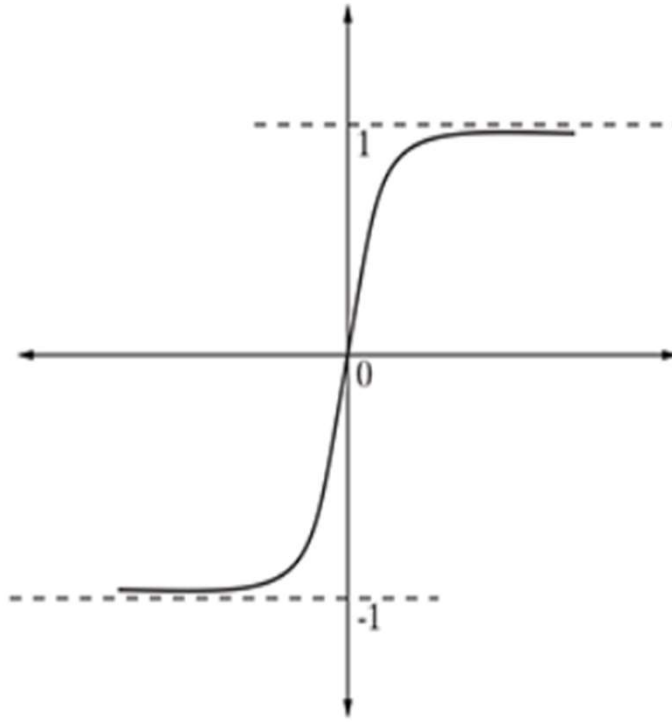


Figure 1.8: Bipolar Sigmoid Function

1. Binary Sigmoid Function

- ▶ A binary sigmoid function is of the form

$$Y_{out} = f(x) = \frac{1}{1 + e^{-kx}}$$

- ▶ where k = steepness or slope parameter of the sigmoid function. By varying the value of k , sigmoid functions with different slopes can be obtained. It has range of $(0, 1)$.

2. Bipolar Sigmoid Function

- ▶ A bipolar sigmoid function is of the form

$$Y_{out} = f(x) = \frac{1 - e^{-kx}}{1 + e^{-kx}}$$

- ▶ The range of values of sigmoid functions can be varied depending on the application. However, the range of $(-1, +1)$ is most commonly adopted.

Tanh

- ▶ The tanh activation function is similar to the sigmoid in that it maps input values to an s-shaped curve. But, in the tanh function, the range is $(-1, 1)$ and is centered at 0.
- ▶ This addresses one of the issues with the sigmoid function.
- ▶ Tanh stands for the hyperbolic tangent, which is the hyperbolic sine divided by the hyperbolic cosine, similar to the regular tangent.
- ▶ Although tanh can be more effective than sigmoid, it still suffers from the same issues as sigmoid when it comes to backpropagation with large or small input values, and it is also an exponential function.

$$Y_{out} = f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Tanh

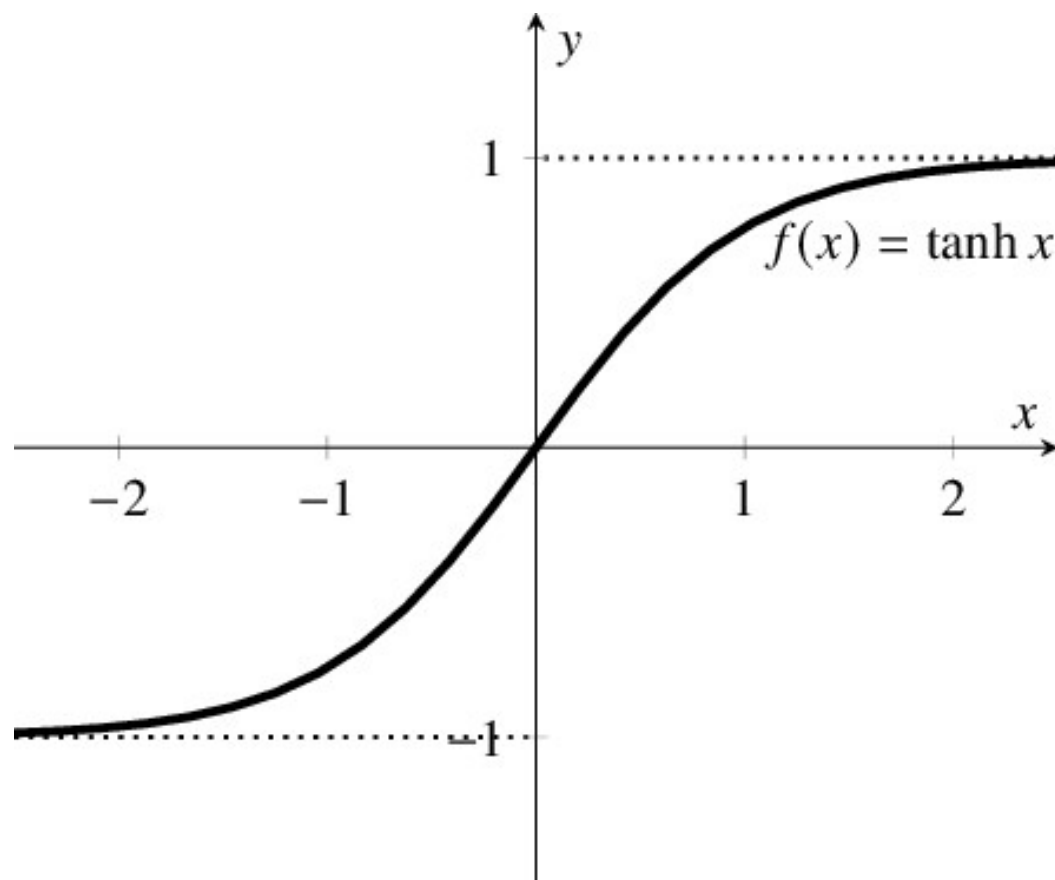


Figure 1.9: Tanh Function

ReLU (Rectified Linear Unit) Function

- ▶ ReLU (Rectified Linear Unit) is a more modern and widely used activation function.
- ▶ ReLU is a simple activation function that replaces negative values with 0 and leaves positive values unchanged, which helps avoid issues with gradients during backpropagation and is faster computationally.
- ▶ Networks using ReLU tend to converge about six times faster than those with sigmoid or tanh. However, ReLU is not centered at 0 and does not handle negative inputs well. This can cause problems during training.
- ▶ Leaky ReLU is an extension of ReLU that addresses these issues.

$$\sigma(x) = \begin{cases} \max(0, x) & , x \geq 0 \\ 0 & , x < 0 \end{cases}$$

ReLU (Rectified Linear Unit) Function

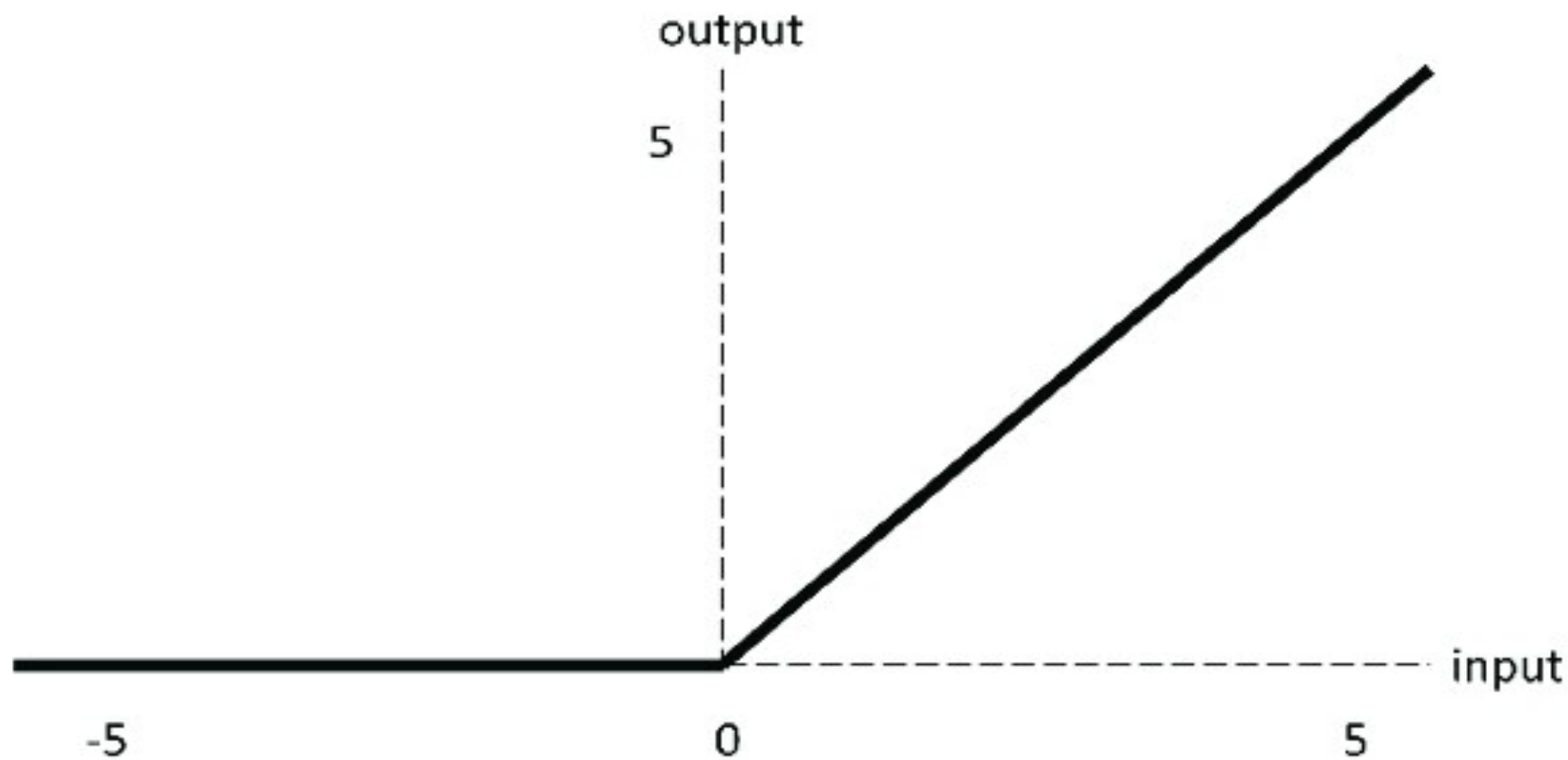


Figure 1.10: ReLU (Rectified Linear Unit) Function

Leaky ReLU Activation Function

- ▶ Leaky ReLU is an extension of the ReLU activation function that aims to address the issue of negative inputs being replaced with zero by the original ReLU function.
- ▶ Instead of replacing negative inputs with zero, Leaky ReLU multiplies them by a small, user-defined value between 0 and 1, allowing some of the information contained in the negative inputs to be retained in the model.

Leaky ReLU Activation Function

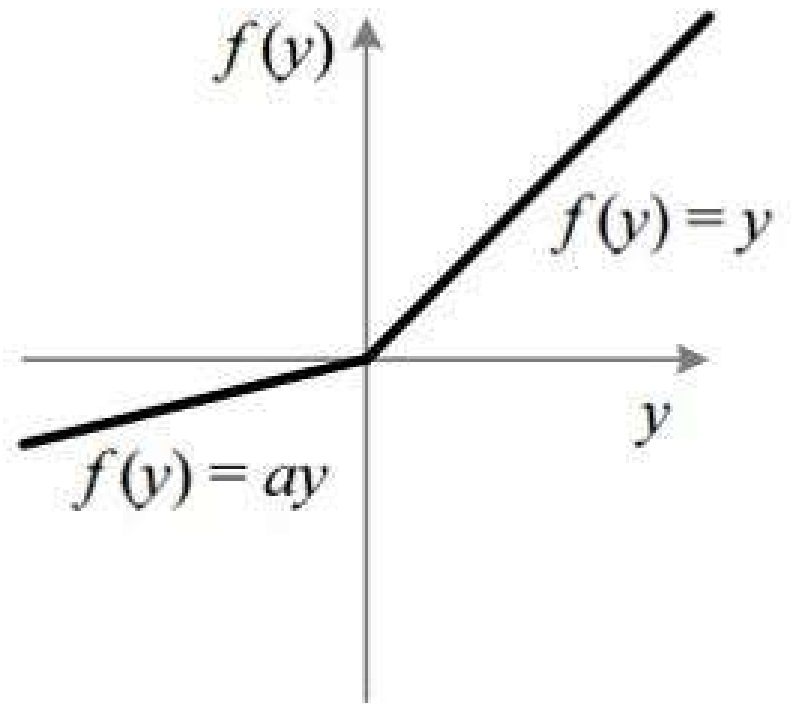
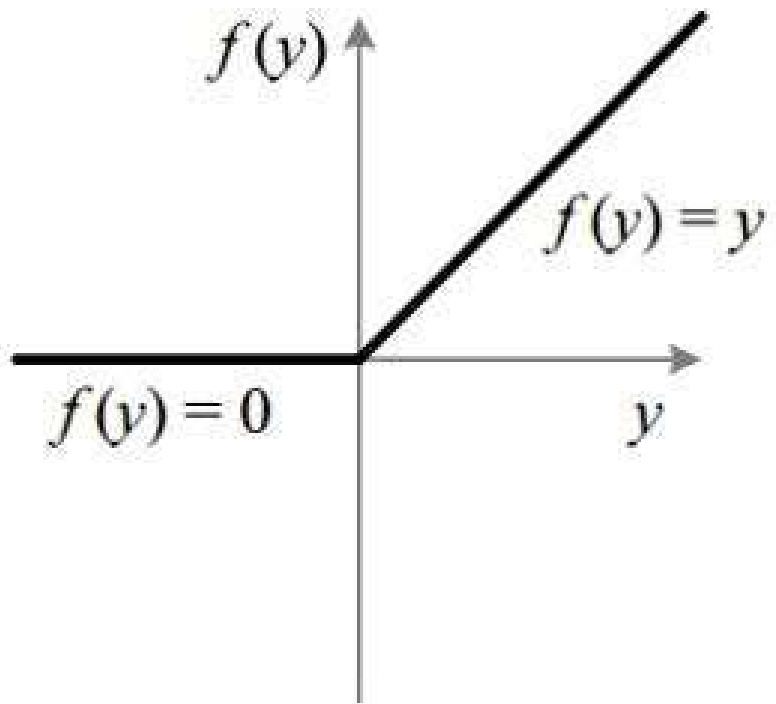
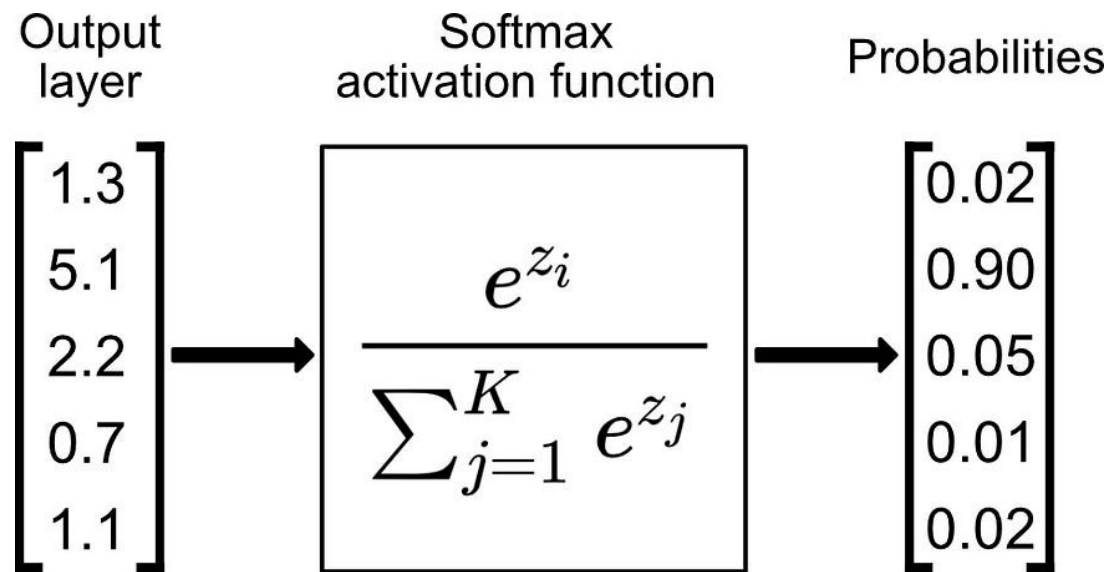


Figure 1.11: Leaky ReLU Activation Function

Softmax Function

- ▶ Softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels.
- ▶ For an arbitrary real vector of length K , Softmax can compress it into a real vector of length K with a value in the range $(0, 1)$, and the sum of the elements in the vector is 1.

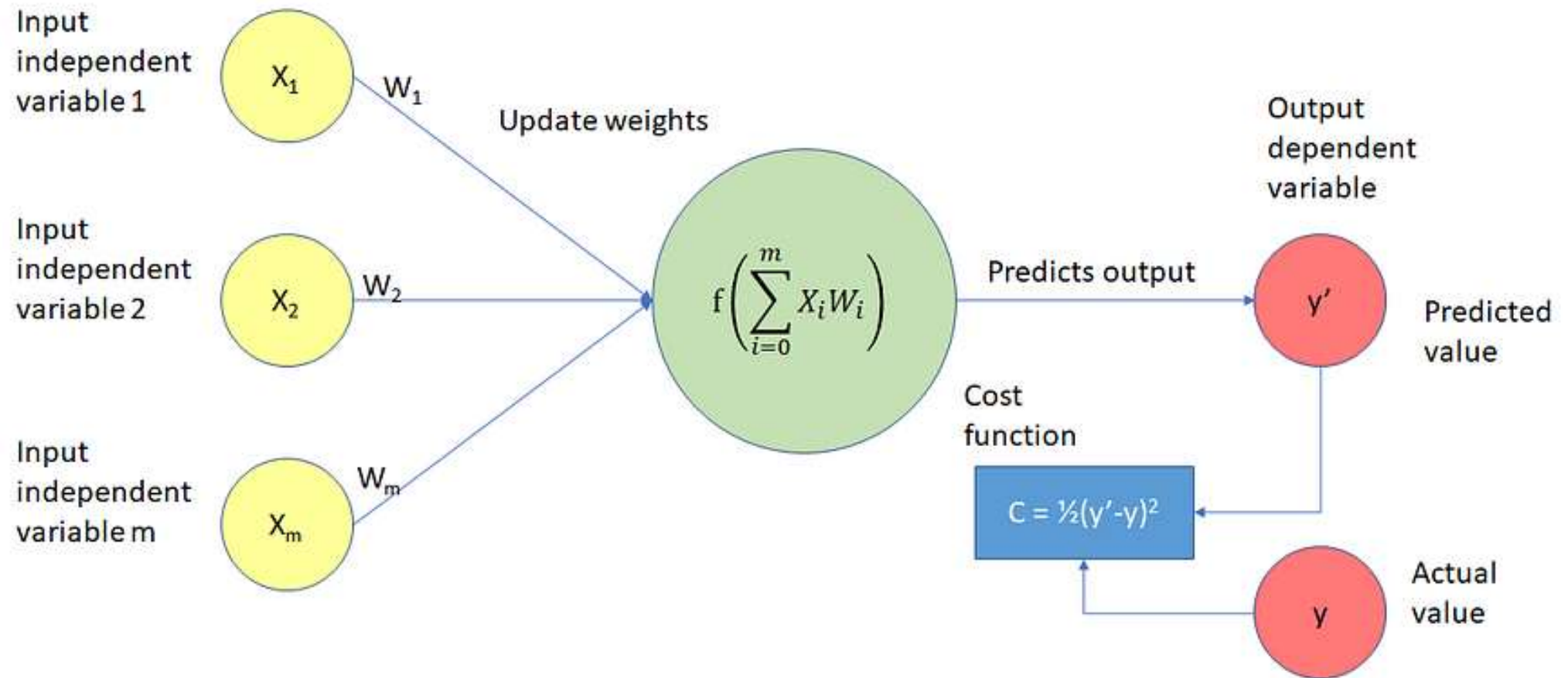


Softmax Function

- ▶ Softmax is different from the normal max function: the max function only outputs the largest value, and Softmax ensures that smaller values have a smaller probability and will not be discarded directly.
- ▶ It is a “max” that is “soft”; it can be thought to be a probabilistic or “softer” version of the argmax function.
- ▶ The denominator of the Softmax function combines all factors of the original output value, which means that the different probabilities obtained by the Softmax function are related to each other.
- ▶ The major drawback in the softmax activation function is that it is -
 - ↳ Non-differentiable at zero and ReLU is unbounded.
 - ↳ The gradients for negative input are zero, which means for activations in that region, the weights are not updated during backpropagation. This can create dead neurons that never get activated.

Cost Functions

What is Cost Function?



What is Cost Function?

- ▶ The loss error is computed for a single training example. If we have 'm' number of examples then the average of the loss function of the entire training set is called 'Cost function'.

$$\text{Cost function (J)} = 1/m (\text{Sum of Loss error for 'm' examples})$$

- ▶ It is a function that measures the performance of a Machine Learning model for given data.
- ▶ Cost Function quantifies the error between predicted values and expected values and presents it in the form of a single real number.

Types of Loss Function?

1. Regression Loss Function
 - ↳ Mean Squared Error Loss
 - ↳ Mean Absolute Error Loss
2. Binary Classification Loss Functions
 - ↳ Binary Cross-Entropy
 - ↳ Hinge Loss
 - ↳ Squared Hinge Loss
3. Multi-Class Classification Loss Functions
 - ↳ Multi-Class Cross-Entropy Loss
 - ↳ Sparse Multiclass Cross-Entropy Loss
 - ↳ Kullback Leibler Divergence Loss

Mean Absolute Error Loss

- ▶ Regression metric which measures the **average magnitude of errors in a group of predictions**, without considering their directions.
- ▶ In other words, it's a **mean of absolute differences among predictions and expected results where all individual deviations have even importance**.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

test setpredicted valueactual value

- ▶ where:
 - ↳ i – index of sample,
 - ↳ \hat{y} – predicted value,
 - ↳ y – expected value,
 - ↳ n – number of samples in dataset.

Mean Squared Error Loss

- ▶ Average squared difference between the predictions and expected results.
- ▶ In other words, an alteration of MAE where instead of taking the absolute value of differences, they are squared.
- ▶ In MAE, the partial error values were equal to the distances between points in the coordinate system.
- ▶ Regarding MSE, each partial error is equivalent to the area of the square created out of the geometrical distance between the measured points. All region areas are summed up and averaged.
- ▶ The MSE formula can be written like this:

$$\text{MSE} = \frac{1}{\underbrace{n}_{\text{test set}}} \sum_{i=1}^n (\underbrace{y_i}_{\text{predicted value}} - \underbrace{\hat{y}_i}_{\text{actual value}})^2$$

Root Mean Square error (RMSE)

- ▶ Root Mean Square error is the extension of MSE – measured as the average of square root of sum of squared differences between predictions and actual observations.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (\text{Predicted}_i - \text{Actual}_i)^2}{N}}$$

Binary Cross Entropy Loss Function

- ▶ Binary cross entropy measures how far away from the true value (which is either 0 or 1) the prediction is for each of the classes and then averages these class-wise errors to obtain the final loss.
- ▶ We can define cross entropy as the difference between two probability distributions p and q , where p is our true output and q is our estimate of this true output.
- ▶ This difference is now applied to our neural networks, where it is extremely effective because of their strong usage of probability.

$$H(x) = \sum_{i=1}^N \overset{\text{true label}}{p}(x) \log \underset{\text{estimate}}{q}(x)$$

Binary Cross Entropy Loss Function



- ▶ We can see above that p is compared to $\log-q(x)$ which will find the distance between the two.
- ▶ Cross entropy will work best when the data is normalized (forced between 0 and 1) as this will represent it as a probability. This normalization property is common in most cost functions.

Categorical cross-entropy

- ▶ It is a loss function that is used for single label categorization.
- ▶ This is when only one category is applicable for each data point. In other words, an example can belong to one class only.
- ▶ Use categorical cross entropy in classification problems where only one result can be correct.

$$L(y, \hat{y}) = - \sum_{j=0}^M \sum_{i=0}^N (y_{ij} * \log(\hat{y}_{ij}))$$

Multi-class Cross Entropy Loss:

	Multi-Class	Multi-Label
$C = 3$	<p>Samples</p>  <p>Labels (t)</p> <p>[0 0 1] [1 0 0] [0 1 0]</p>	<p>Samples</p>  <p>Labels (t)</p> <p>[1 0 1] [0 1 0] [1 1 1]</p>

- ▶ Most time, Multi-class Cross Entropy Loss is used as a loss function.
- ▶ The generalized form of cross entropy loss is the multi-class cross entropy loss.

Multi-class Cross Entropy Loss:

$$-\sum_{c=1}^M y_{o,c} \log(p_{o,c})$$

- ▶ M – No of classes
- ▶ y – binary indicator (0 or 1) if class label c is the correct classification for input o
- ▶ p – predicted probability score of input o belonging to class c .

Forward and backward Propagation

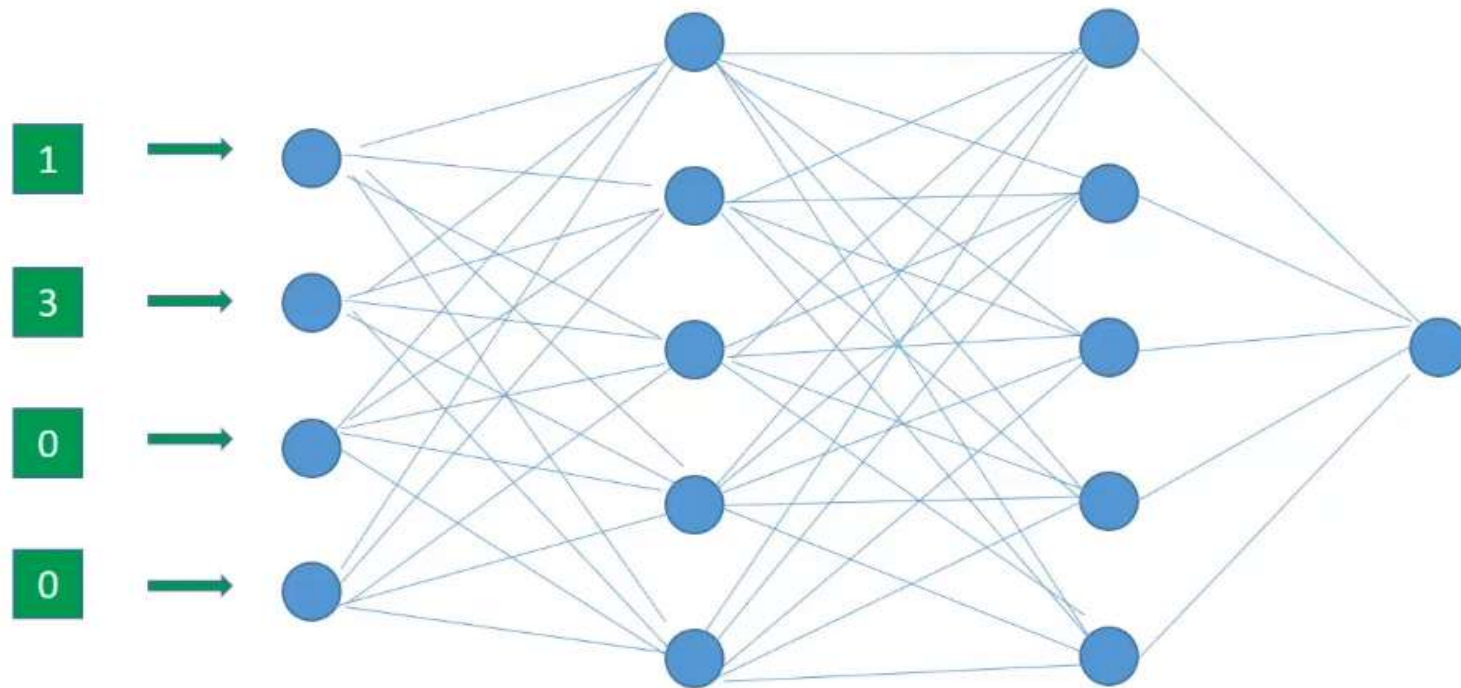
Forward Propagation

gender	age	hypertension	does_smoke	stroke
1	3	0	0	0
1	58	1	1	1
0	8	0	0	0
0	70	0	1	1
1	14	0	0	0
0	47	0	0	0
0	52	0	1	1
0	75	0	0	0
0	32	0	1	0

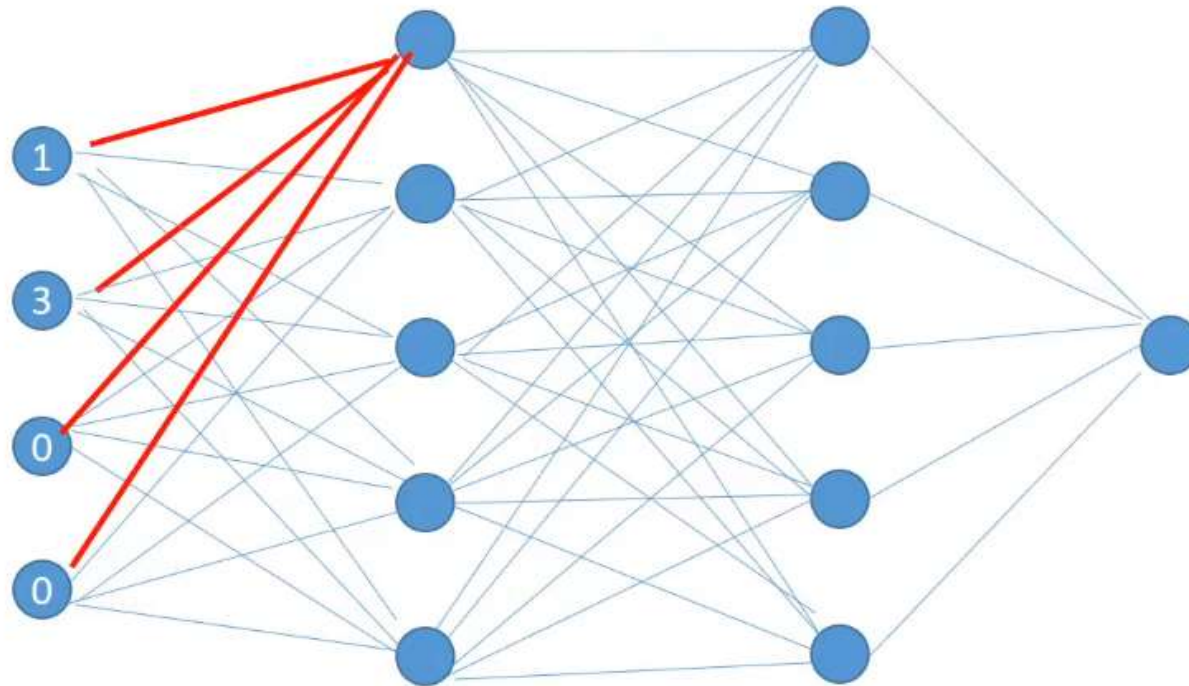
Forward Propagation

- ▶ This data set has four independent features represented by green and one dependent feature represented by red.
- ▶ For the sake of simplicity let's focus on the first row and try to understand forward propagation with this single row input.
- ▶ Each of the independent features in the first row will be passed to the input layer of the neural network to act as input.
- ▶ Now we start off the forward propagation by randomly initializing the weights of all neurons.
- ▶ These weights are depicted by the edges connecting two neurons.
- ▶ Hence the weights of a neuron can be more appropriately thought of as weights between two layers since edges connect two layers.

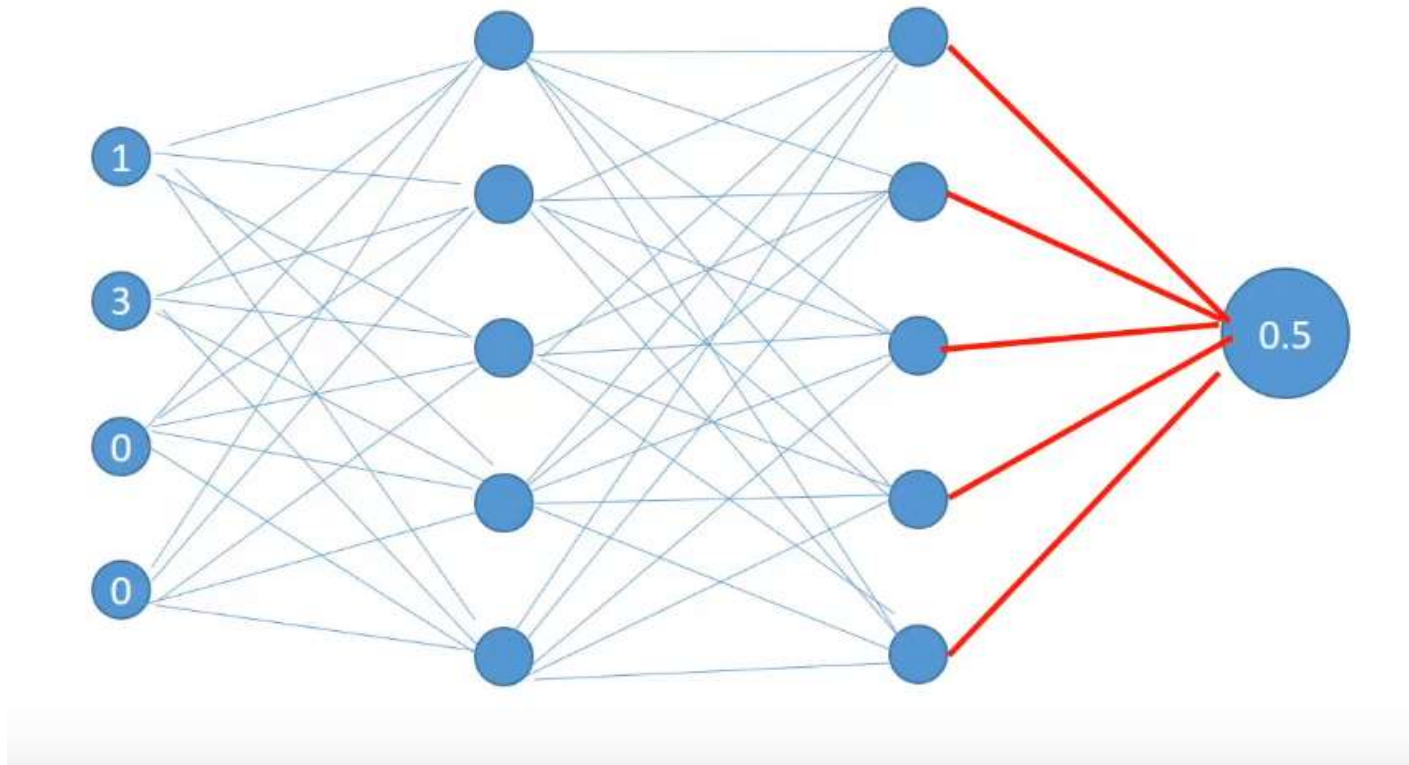
Forward Propagation



Forward Propagation



Forward Propagation



Forward Propagation

- ▶ This data set has four independent features represented by green and one dependent feature represented by red.
- ▶ These calculations of the first row features are 0.5 and the predicted value becomes 0.5 against the actual value of 0.

Actual	Predicted
0	0.5

- ▶ Now if you notice that carefully we calculated the hidden activations for each of the neurons sequentially, that is one after the other.
- ▶ In fact, one optimization that we can make to the calculation is to calculate them in parallel. Since the calculation of neurons is independent of each other they can be computed in parallel easily.

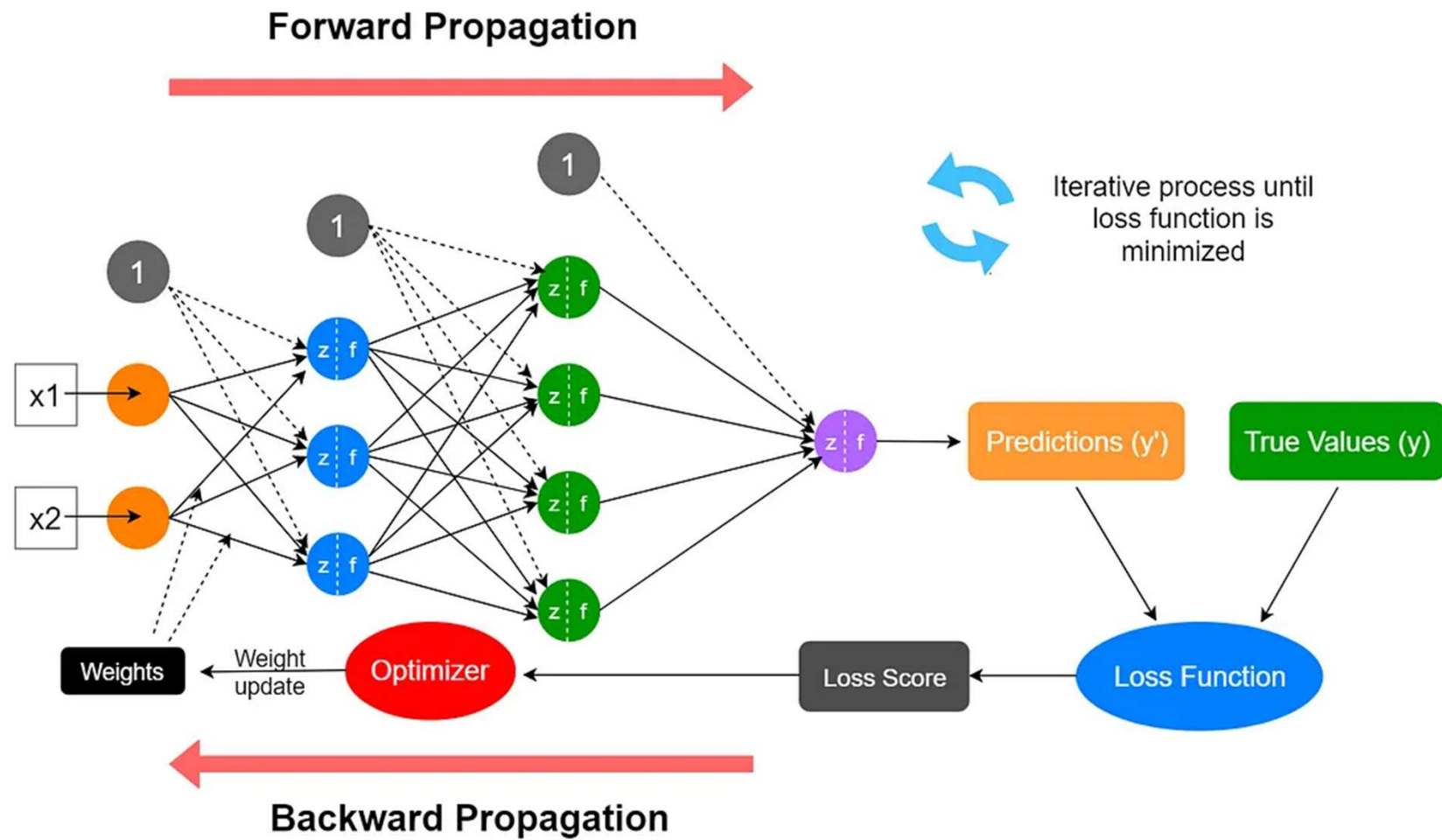
How to reduce Errors?

- ▶ Interestingly what we can do is we can use a smart strategy to adjust the weights and reduce the errors. Specifically what we are interested is in two things-
 - First is the amount of change in error on changing the weight by a small amount
 - And second is the direction of that change.

Backpropagation

- ▶ Backpropagation is a process involved in training a neural network. It takes the error rate of a forward propagation and feeds this loss backward through the neural network layers to fine-tune the weights.
- ▶ Backpropagation is a powerful algorithm in deep learning, primarily used to train artificial neural networks, particularly feed-forward networks. It works iteratively, minimizing the cost function by adjusting weights and biases.
- ▶ In each epoch, the model adapts these parameters, reducing loss by following the error gradient.
- ▶ Backpropagation often utilizes optimization algorithms like gradient descent or stochastic gradient descent.
- ▶ The algorithm computes the gradient using the chain rule from calculus, allowing it to effectively navigate complex layers in the neural network to minimize the cost function.

Backpropagation



Backpropagation Working

- ▶ In the backward pass, the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases.
- ▶ One common method for error calculation is the Mean Squared Error (MSE), given by:
$$\text{MSE} = \frac{(\text{Predicted Output} - \text{Actual Output})^2}{2}$$
- ▶ Once the error is calculated, the network adjusts weights using gradients, which are computed with the chain rule.
- ▶ These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration.
- ▶ The backward pass continues layer by layer, ensuring that the network learns and improves its performance.
- ▶ The activation function, through its derivative, plays a crucial role in computing these gradients during backpropagation.

DNN (Deep Neural Network Architecture)

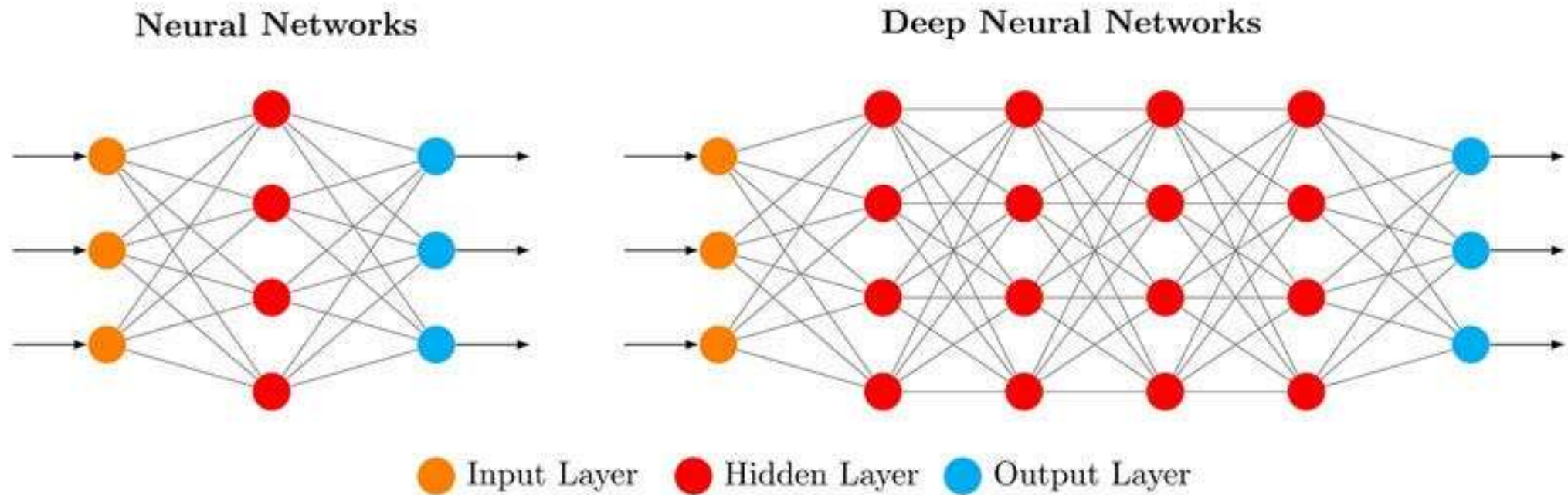


Figure 1.1: Deep Neural Network Architecture

DNN

- ▶ A deep neural network is a type of artificial neural network (ANN) with multiple layers between its input and output layers.
- ▶ Each layer consists of multiple nodes that perform computations on input data. Another common name for a DNN is a deep net.
- ▶ The “deep” in deep nets refers to the presence of multiple hidden layers that enable the network to learn complex representations from input data.
- ▶ These hidden layers enable DNNs to solve complex ML tasks more “shallow” artificial networks cannot handle.
- ▶ Hidden layers in a DNN are dense (fully connected) layers.
- ▶ Each neuron in a dense layer is connected to every neuron in the previous and subsequent layer, which makes DNNs highly suitable for learning complex relationships in data.

DNN

- ▶ Hidden layers within a deep neural network are not simple copies of each other. Instead, a DNN has a complex layer structure with a variety of different layer types, such as:
 - ↳ Convolutional layers that apply a series of learnable filters to input data.
 - ↳ Long short-term memory (LSTM) layers that capture dependencies in sequential data.
 - ↳ Gated recurrent unit (GRU) layers that use gating to control the flow of network info.
 - ↳ Attention layers that focus on specific parts of the input sequence when making predictions.
 - ↳ Normalization layers that stabilize and accelerate the training of the deep neural network.
- ▶ The more hidden layers a deep net has, the better it becomes at learning from and processing input data.

Deep Neural Network Working

- ▶ Deep nets are composed of multiple layers of interconnected nodes called neurons or units. There are three types of layers in a DNN:
- ▶ **Input layer.** The input layer receives raw data. Each node in this layer represents a feature or attribute of the input data. For example, to interpret 28x28 pixel images, the input layer requires 784 nodes, each representing pixel intensity (ranging from 0 to 255).
- ▶ **Hidden layers.** DNNs have two or more hidden layers sandwiched between input and output layers. Each hidden layer comprises multiple neurons, which are connected to neurons in both adjacent layers. However, neurons within the same layer are not connected to each other.
- ▶ **Output layer.** The output layer produces the network's final predictions. The number of neurons in the output layer depends on what tasks the network is performing.
- ▶ Once you provide raw data, the input data is fed forward through the network layer by layer. At each layer, neurons apply assigned activation functions and pass the processed output to the next layer.

What is a Deep Neural Network used for?

► DNNs have become a powerful tool across various domains due to their ability to learn complex patterns from large amounts of data. Here are a few use cases that commonly involve the use of a deep neural network:

1. Image recognition
2. Natural language processing
3. Speech synthesis
4. Recommender systems
5. Healthcare use cases
6. Finance and trading use cases
7. Retail use cases
8. Autonomous vehicles
9. Environmental monitoring

CNN (Convolutional Neural Network Architecture)

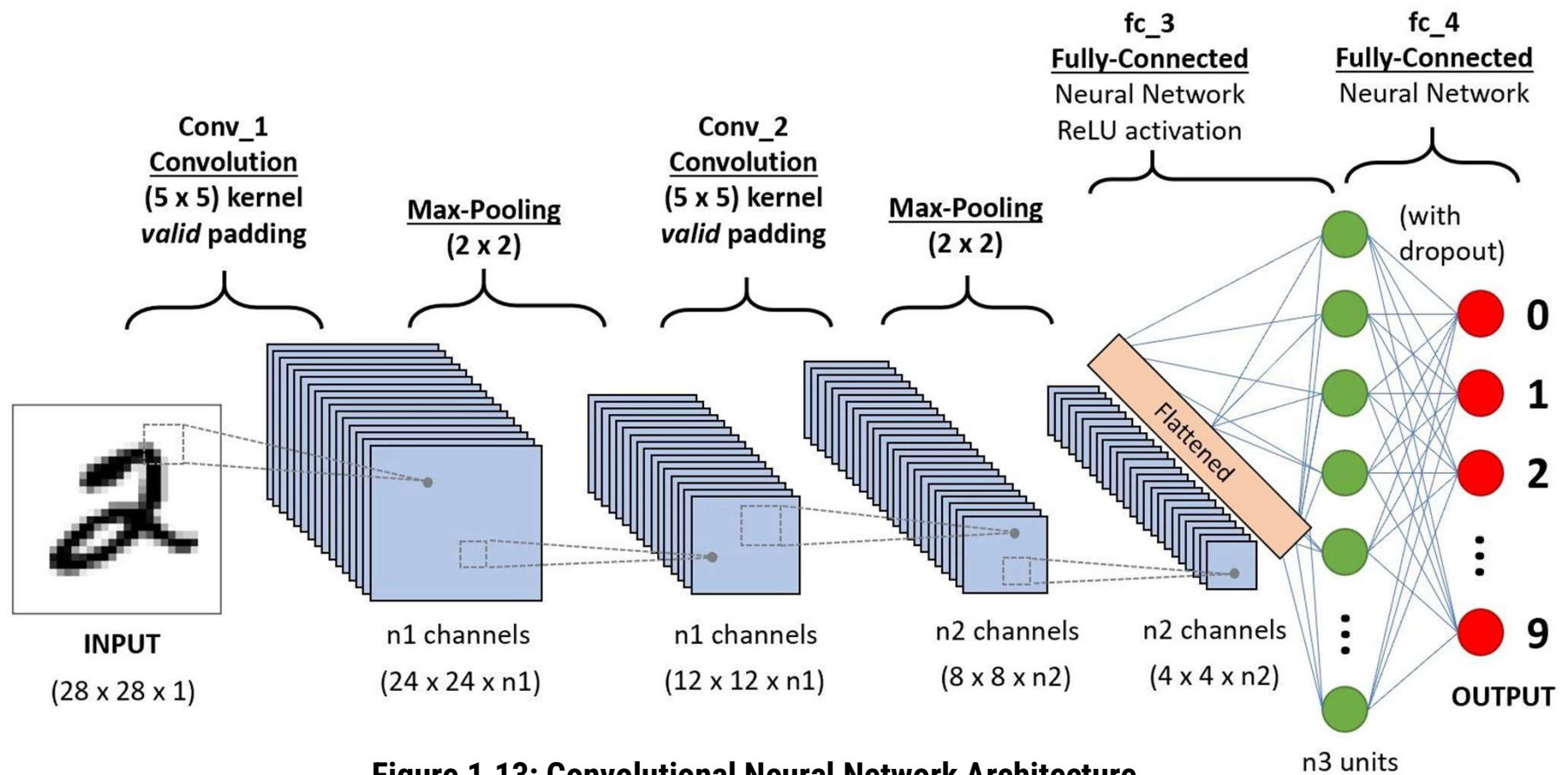


Figure 1.13: Convolutional Neural Network Architecture

CNN

- ▶ In today's world, CNNs are widely used for purposes such as video recognition (e.g., facial recognition), medical imaging (e.g., detecting cancerous tumors), self-driving cars (e.g., identifying road signs), and natural language processing (e.g., text classification).
- ▶ The CNN architecture can be divided into five components. Here's an overview of the five layers.
- ▶ **Feature Extraction through Convolutional Layers**
 - ↳ Convolutional layers scan the input data using filters (kernels) to detect patterns like edges, textures, or shapes.
- ▶ **Pooling Layers**
 - ↳ Pooling layers preserve key features while reducing computational complexity. This helps reduce multiple dimensions.

CNN

▶ **Activation Layers**

- Activation layers apply non-linear functions like ReLU to introduce non-linearity. This enables the network to learn complex patterns.

▶ **Flattening and Fully Connected Layers**

- Once the feature has been extracted, the data is converted into a vector and passed through fully connected layers for classification.

▶ **Output Layer**

- The output layer provides the final prediction using a Softmax function for classification tasks.

- ▶ Among all CNN operations, convolution is the core operation that allows the model to extract meaningful features from the input data. It applies filters to the input to detect patterns like edges, textures, and shapes in an image.

CNN-Striding

- Suppose we have an input matrix of 5×5 and a filter matrix of 3×3 . For those unfamiliar with filters, they are sets of weights in a matrix applied to an image or another matrix to extract specific features.

2	4	9	1	4
2	1	4	4	6
1	1	2	9	2
7	3	5	1	3
2	3	4	8	5

Image

\times

1	2	3
-4	7	4
2	-5	1

Filter /
Kernel

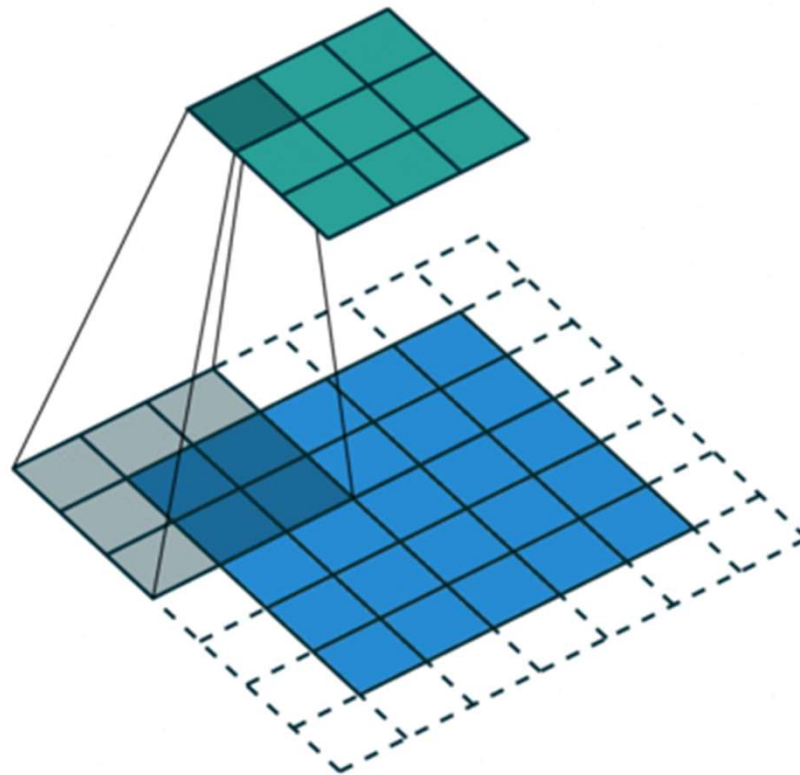
$=$

51		

Feature

CNN-Striding

- ▶ Some times we do not want to capture all the data or information available so we skip some neighboring cells let us visualize it,



CNN-Padding

- ▶ While applying convolutions we will not obtain the output dimensions the same as input we will lose data over borders so we append a border of zeros and recalculate the convolution covering all the input values.

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel		
0	-1	0
-1	5	-1
0	-1	0

114				

CNN-Pooling

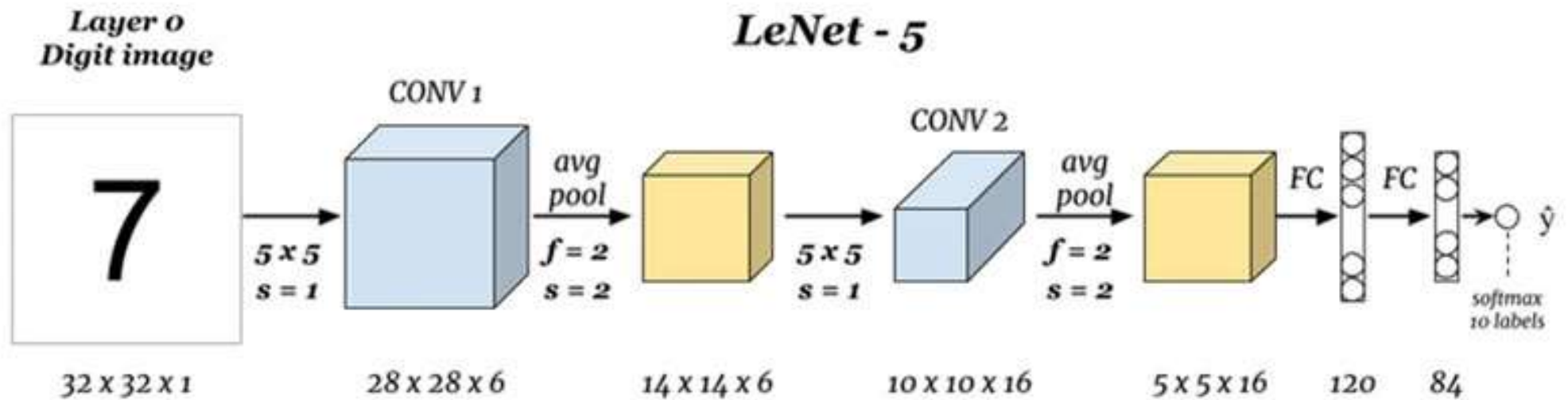
- ▶ In general terms pooling refers to a small portion, so here we take a small portion of the input and try to take the average value referred to as average pooling or take a maximum value termed as max pooling

2	2	7	3
9	4	6	1
8	5	2	4
3	1	2	6

Max Pool
→
Filter - (2 x 2)
Stride - (2, 2)

9	7
8	6

CNN



Thank You