



# Indian Institute of Technology Bombay

Summer Of Science, 2023

Final Term Report On

---

## Fundamentals of Machine Learning and Neural Networks

---

*Author:*

Chaitanya Katti

*Mentor:*

Risbud Advait Parag

Aug 3, 2023

## **Abstract**

This mid-term report presents an overview of the fundamental concepts and techniques in neural networks and deep learning, emphasizing the knowledge acquired during the initial phase of my study. The report outlines the main objectives, methodology, and key findings, providing a solid foundation for further exploration in this rapidly evolving field.

The study begins by elucidating the core principles of machine learning. We delve into the mathematical foundations of cost functions and gradient descent and implement standard machine learning algorithms, such as logistic regression and decision trees.

We then explore deep learning, which represents a subset of machine learning methods that rely on neural networks with multiple hidden layers. The concept of deep learning is introduced, highlighting its ability to automatically extract meaningful features from raw data. We investigate various architectures, including feed forward neural networks and convolutional neural networks (CNNs). Furthermore, this report investigates optimization algorithms (e.g., SGD, Adam).

A Python code implementation of techniques used in this report are available at Github repository [1].

# Contents

<b>1</b>	<b>What is Machine Learning?</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Types of Machine Learning . . . . .	5
<b>2</b>	<b>Supervised Learning</b>	<b>6</b>
2.1	Linear Regression . . . . .	6
2.2	Logistic Regression . . . . .	9
2.3	Decision Trees . . . . .	10
2.4	Random Forests . . . . .	12
2.5	Feature Engineering . . . . .	13
<b>3</b>	<b>Neural Networks and Deep Learning</b>	<b>14</b>
3.1	Inspiration From Biology . . . . .	14
3.2	Feed Forward Neural Networks . . . . .	15
3.3	Training Neural Networks . . . . .	16
<b>4</b>	<b>Convolutional Neural Networks</b>	<b>20</b>
4.1	Convolution operation . . . . .	20
4.2	Convolutional Layer . . . . .	22
4.3	Max Pooling . . . . .	23
4.4	CNN Architecture . . . . .	23
4.5	Parameter Sharing . . . . .	24
<b>5</b>	<b>Sequence Models</b>	<b>26</b>
5.1	RNN . . . . .	26

## List of Figures

1	Linear regression . . . . .	6
2	Convergence of losses . . . . .	8
3	Gradient descent on the loss surface . . . . .	8
4	Binary class visualization . . . . .	9
5	Logistic regression curve . . . . .	10
6	Layout of a decision tree . . . . .	10
7	Plot of Shannon's entropy . . . . .	11
8	Structure of a random forest ensemble . . . . .	12
9	An example of feature engineering . . . . .	13
10	Multi layered perceptron . . . . .	14
11	Neuron . . . . .	14
12	Feed Forward Network . . . . .	16
13	Mini-batch gradient descent . . . . .	18

14	Convolution operation . . . . .	20
15	Edge detection using convolution . . . . .	21
16	CNN layer . . . . .	23
17	Example of max pooling . . . . .	23
18	Architecture of VGG-16 . . . . .	24
19	Architecture of RNN . . . . .	26

# 1 What is Machine Learning?

## 1.1 Introduction

Machine learning and artificial intelligence are two of the most popular buzzwords in past few years. Mostly because of their insane capabilities to mimic human like behaviours. But also because of recent developments in technology that made computation so fast.

Machine learning is a subset of Artificial Intelligence. Learning in ML refers to a machine's ability to learn based on data and an ML algorithm's ability to train a model, evaluate its performance or accuracy, and then make predictions. Over the course of training, the model improves its accuracy in predicting the correct output.

Not every problem requires machine learning algorithms. If one were tasked to make a calculator then a simple C program would do the job. Even ChatGPT fails at doing simple arithmetic. Machine learning algorithms are useful when simple code starts turning into a complex list of rules and becomes hard to maintain.

## 1.2 Types of Machine Learning

ML is broadly classified into 3 categories based on availability of training data.

1. **Supervised Learning:** In supervised learning, the model is trained on labelled data, where each data point is associated with a corresponding target or output value. The goal is for the model to learn the mapping between input features and their corresponding labels, enabling it to make accurate predictions on new, unseen data. Common supervised learning algorithms include linear regression, logistic regression, decision trees, random forests, and support vector machines.
2. **Unsupervised Learning:** Unsupervised learning deals with unlabelled data, where the model learns patterns, structures, or relationships within the data without any explicit target variable. The objective is to uncover hidden patterns, group similar data points, or reduce the dimensionality of the dataset. Clustering and dimensionality reduction techniques such as k-means clustering, hierarchical clustering, principal component analysis (PCA), and t-SNE are commonly used in unsupervised learning.
3. **Reinforcement Learning:** Reinforcement learning involves an agent learning how to interact with an environment to maximize a reward signal. The agent learns through trial and error, taking actions in the environment and receiving feedback in the form of rewards or penalties. The goal is to learn a policy or strategy that optimizes the cumulative reward over time. Reinforcement learning has applications in areas such as robotics, game playing, and autonomous systems.

## 2 Supervised Learning

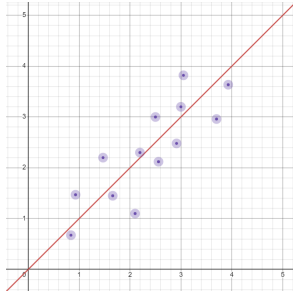
In supervised learning, each data point is labelled or associated with a category or value of interest. An example of a categorical label is assigning an image as either a ‘cat’ or a ‘dog’. An example of a value label is the sale price associated with a used car. The goal of supervised learning is to study many labelled examples like these, and then to be able to make predictions about future data points. For example, identifying new photos with the correct animal or assigning accurate sale prices to other used cars. This is a popular and useful type of machine learning.

### 2.1 Linear Regression

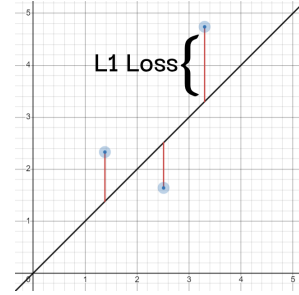
Linear regression at its core it just curve fitting a straight line/plane/hyperplane to best represent a set of points in some n-dimensional space. Here, the training data is the coordinates of the set of points.

Consider the equation of a line  $f_{w,b}(x) = y = wx + b$ . This line has two parameters  $w$  and  $b$ .

Let’s define something called loss function that can describe the if line fits the data or not. A simple way to do this is called the L1-Loss. Here is a graphical representation.



(a) A line fit over set of point.



(b) The loss values for a point

Figure 1: Figures showing linear regression and L1 loss

$$\text{L1 Loss} = |f_{w,b}(x_i) - y_i| = |(w \cdot x_i + b) - y_i| \quad (1)$$

where  $x$  and  $y$  represent the coordinates of a point on the grid and  $w$ ,  $b$  are the parameters of the line. Note that loss is zero when the line passes through the point. To best represent the data the parameters must be chosen in such a way that loss is minimized for every point.

Of course, we can’t make the line pass through every point to make all losses minimum, but we can come up with a compromise that minimizes the sum of all these losses.

$$\text{Cost} = \sum_{i=1}^m |f_{w,b}(x_i) - y_i| = \sum_{i=1}^m |(w \cdot x_i + b) - y_i| \quad (2)$$

There is an even more widely used function to quantify the cost  $J(w, b)$ , it uses the Mean Square Error loss function.

$$\text{MSE Loss} = (f_{w,b}(x_i) - y_i)^2 \quad J(w, b) = \frac{1}{2m} \sum_{i=1}^m ((w \cdot x_i + b) - y_i)^2 \quad (3)$$

In the field of linear algebra there are techniques to directly solve for the values of  $w$  and  $b$  that minimize the cost function but let's explore another algorithm commonly used in machine learning called the Gradient Descent.

---

**Algorithm 1** Gradient Descent for Linear Regression

---

**Require:** Training data:  $(X, Y)$ , Learning rate:  $\alpha$ , Number of iterations:  $N$

1: Initialize parameters:  $w$  and  $b$  randomly

2: **for**  $t = 1$  **to**  $N$  **do**

3:   Compute predictions:  $\hat{y} = f_{w,b}(X)$

4:   Compute gradient:

$$\partial_w J(w, b) = \frac{1}{m} \sum_{i=1}^m x_i (\hat{y} - y_i)$$

$$\partial_b J(w, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y} - y_i)$$

5:   Update parameters:

$$w \leftarrow w - \alpha \cdot \partial_w J(w, b)$$

$$b \leftarrow b - \alpha \cdot \partial_b J(w, b)$$

6: **end for**

7: **Return** Learned parameters:  $w, b$

---

Here training data  $X, Y$  is the list of all  $x$  and  $y$  coordinates respectively. The algorithm works because the parameters are updated in the direction of the steepest descent. The terms  $w$  and  $b$  are also weight and bias.

### Parameter Initialization

It is guaranteed that the model will converge to optimal solution with any initialization. But this might not be case for more complex models, hence a random initialization of weight and biases is always a good approach.

### Effect of learning rate $\alpha$

The learning affects the rate of convergence. It can be thought of the step size when making parameter update. But if the learning rate is too high then the algorithm will

not converge. The below graphical representation gives us a good idea.

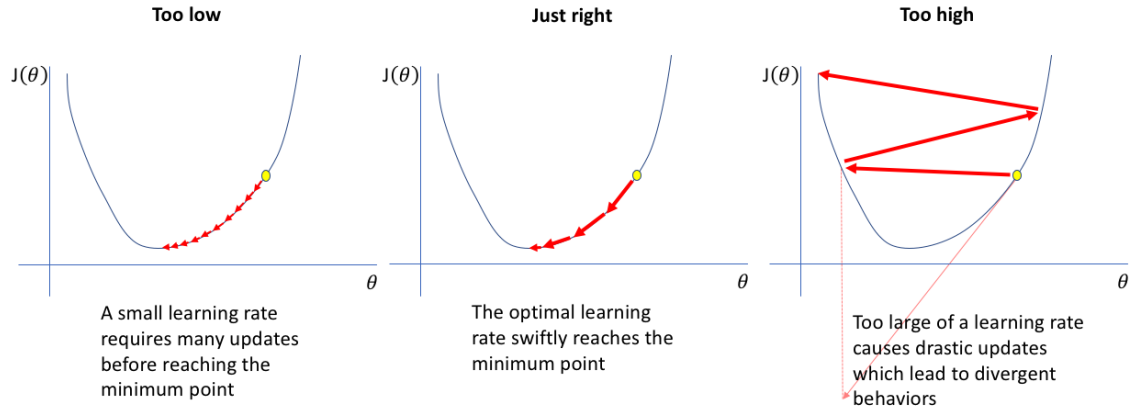


Figure 2: A plot of loss values over training steps with different learning rates. Figure adapted from [10]

## Training Steps $N$

The algorithm runs for a  $N$  steps. After sufficient steps, the loss values will converge. If learning rate is too high then even after the large amount of steps the model won't converge. On the other hand, if learning rate is too little, many steps might be required as step size is small.

## Loss Landscape

The loss function can be thought of a 3D surface plot with x and y-axis representing the weight and bias values. It can be proven mathematically that this surface is convex for a linear regression model and hence a global minimum solution always exists and is achievable.

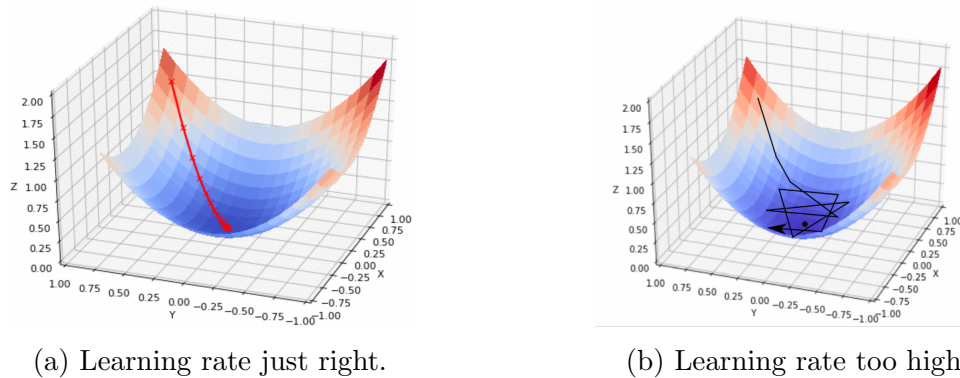


Figure 3: Gradient descent on the loss surface which is convex paraboloid. Figure adapted from [11]



## 2.2 Logistic Regression

Logistic Regression is used for binary classification. It is based on the same ideas of linear regression but uses the logistic function to modify the output. Logistic equation also called Sigmoid is given by

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (4)$$

### Training Data

Suppose we are given a sequence of number along with their class (represented by either 0 or 1). We can visualize this data on a simple plot.

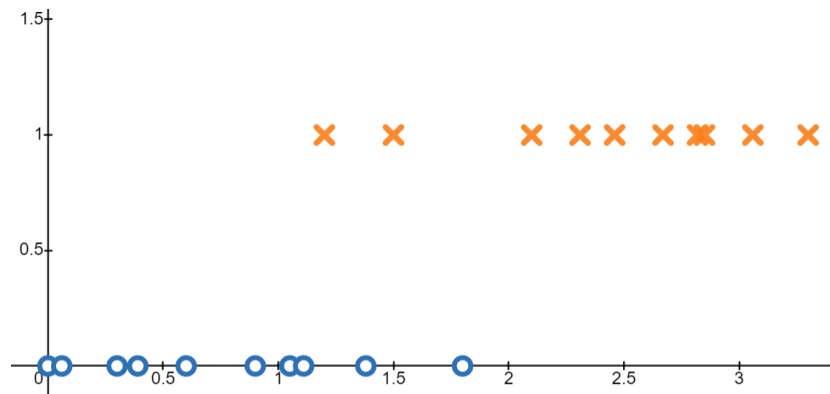


Figure 4: A plot of training data with binary classes

### Regression Model

The model is again parameterized a weight and a bias. The function that maps input  $x$  to a binary class is given by

$$\hat{y} = f_{w,b}(x) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} \quad (5)$$

### Loss and Cost Function

The most common loss function used for classification tasks is the Binary Cross-Entropy loss

$$\text{BCE Loss} = -y_i \cdot \log(\hat{y}) - (1 - y_i) \cdot \log(1 - \hat{y}) \quad (6)$$

where  $x_i$  and  $y_i$  are the training feature and labels. We can now define the cost function which is just the average of all losses.

$$H(w, b) = -\frac{1}{m} \sum_{i=1}^m y_i \cdot \log(\hat{y}) + (1 - y_i) \cdot \log(1 - \hat{y}) \quad (7)$$

The BCE loss is maximum when the values of  $x_i$  and  $y_i$  don't match. Now we can define our algorithm to train the model. The math in computing the gradient is same for logistic and linear regression. After training, we obtain the parameters  $w$ ,  $b$ . Now we can define a threshold and get distinct outputs from the model

$$\text{Prediction} = \begin{cases} 1, & \text{if } \hat{y} > 0.5 \\ 0, & \text{if } \hat{y} < 0.5 \end{cases} \quad (8)$$

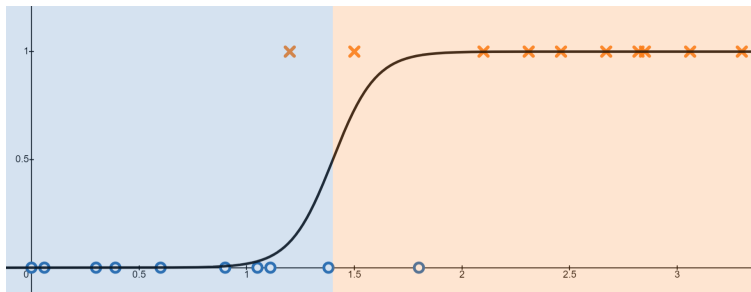


Figure 5: A plot of the sigmoid curve with colors highlighting binary classification

## 2.3 Decision Trees

It is a flowchart-like model that makes decisions based on features of the input data. Decision trees are widely used because they are easy to understand and interpret. They can be used to handle both numerical and categorical data, and can capture complex decision-making processes.

The decision tree model is built in a recursive manner. It starts with a root node that represents the entire dataset. The algorithm selects the best feature to split the data based on certain criteria, such as maximizing the purity or reducing the impurity of the resulting subsets. This splitting process continues until a stopping criterion is met, such as reaching a maximum depth or having a minimum number of examples in a node.

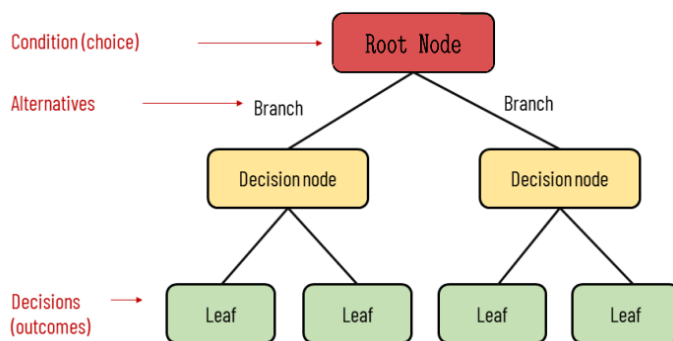


Figure 6: Decision tree highlighting root, decision and leaf nodes

## Entropy

Entropy is the measure of disorder of information. Take for example a set of red and blue balls. If all the balls were red then the disorder would be minimum. Similarly, if we had all blue balls, then entropy would be minimum. However, if we had a 50/50 split then the entropy is high.

There is a beautiful equation called the Shannon's Entropy that is used to quantify this measure of information.

$$h(p) = -p \cdot \log(p)$$

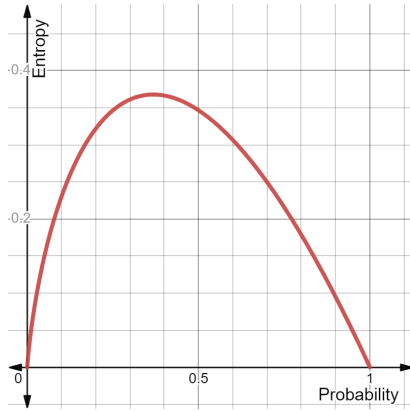
Where  $p$  is the probability of the balls being red and  $h(p)$  is the entropy of class red. In the case of binary classifications, we can add the two entropies of each class to get the total entropy of the set.

$$p = 1 - q$$

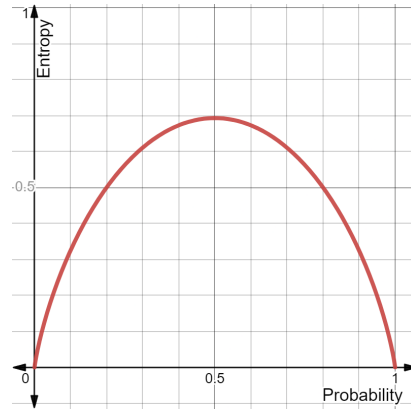
$$E = h(p) + h(q) = -p \cdot \log(p) - q \cdot \log(q)$$

$$E = -p \cdot \log(p) + (1 - p) \cdot \log(1 - p) \quad (9)$$

where  $E$  is the total entropy of the set.



(a) Entropy for 1 class:  $h(p)$



(b) Total entropy of both classes:  $E$

Figure 7: Plot of Shannon's entropy

## Purity

Purity is the opposite of entropy. The decision to split at each node is made according to the metric called purity. A node is 100% impure when a node is split evenly 50/50 and 100% pure when all of its data belongs to a single class.

## Information Gain

At every decision node, the we split the dataset into two branches based on a feature of the data. E.g - Suppose we had a set of balls of different sizes having either red and blue color. A decision could be to spit dataset based on color, or if the size is greater than a particular value.

We split the dataset in such a way that the information gain is maximized.

$$\text{Information Gain} = \text{Entropy}_{\text{Parent}} - \text{Entropy}_{\text{Children}}$$

$$\text{Entropy}_{\text{Children}} = \frac{N_1}{N}E + \frac{N_2}{N}E$$

where  $E_1$  and  $E_2$  are total entropies of respective subset after splitting the dataset.

Recursive splitting at each node until a stopping criterion such at max recursion depth or minimum information gain is met will give the decision tree.

## 2.4 Random Forests

Decision trees are greedy algorithms that try to maximize information gain at the current step and don't take into account the further consequences. As depth of the tree increases, it is prone to overftting the data. Hence, its is not a realiable method.

A random forest consists of multiple decision trees, each trained on a random subset of the training data and using a random subset of features. An ensemble of these trees vote on the final output and the majority vote is used to classify the input data.

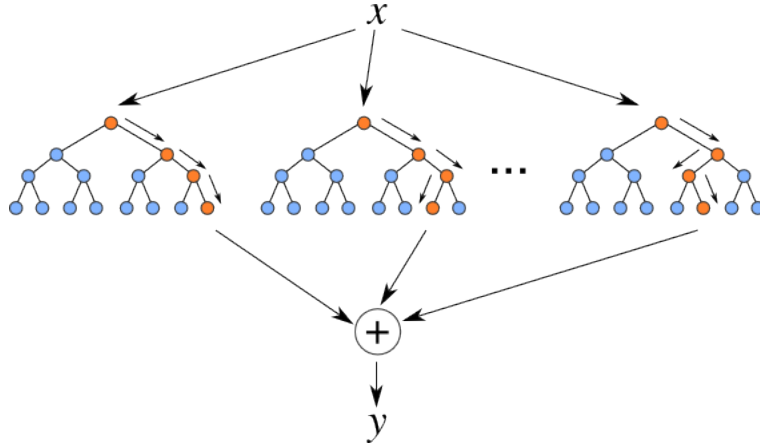


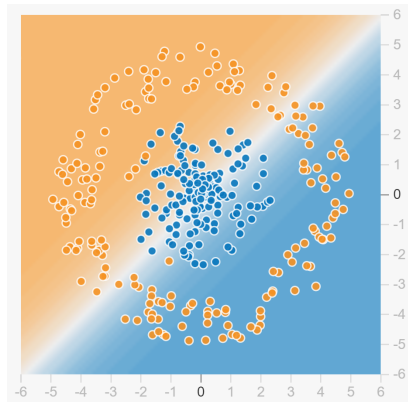
Figure 8: Structure of a random forest ensemble

## 2.5 Feature Engineering

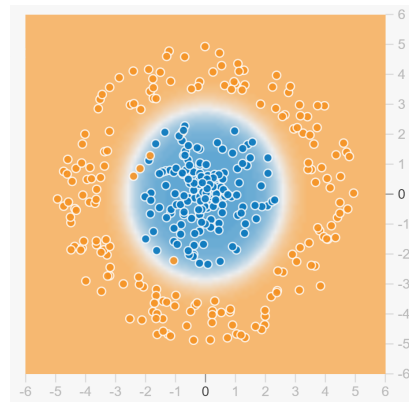
Dataset contains various feature (e.g. In dataset of cat, images, the pixels values and labels are called features). The term feature engineering refers to choosing such features to train a machine learning model that will result in better performance.

Let's say we plan to make machine learning model that can predict sales revenue. We take a dataset and one of the features in this dataset might be 'date'. Date by itself doesn't matter much, but if we include the the day of the week as another input then the model can understand more patterns in the data.

Another simple example could be that when using logistic regression to classify non-linear data distribution, we can simple include non-linear inputs to our model to train. Suppose our data is a set of circular distributed points, and we wish to perform logistic regression. Then linear input features  $x$  and  $y$  will fail. But if we perform logistic regression on the input features  $x^2$  and  $y^2$  then it results in much better prediction.



(a) Classification based on linear inputs  $x$  and  $y$



(b) Classification based on non-linear inputs  $x^2$  and  $y^2$

Figure 9: Comparison between linear and non-linear features for classification

Simply squaring the input features brought enough non-linearity. It is easy to see that a simple equation like  $x^2 + y^2 = 1$  makes a circle and hence will do a better job at classifying the dataset.

## 3 Neural Networks and Deep Learning

### 3.1 Inspiration From Biology

Artificial neural networks are inspired by the structure and function of biological neural networks in the human brain. The artificial neurons are organized into layers, with weighted connections transmitting and processing information. Through a learning process called backpropagation, the network adjusts these weights to improve its performance on specific tasks. While artificial neural networks may simplify the brain's complexity, they excel at pattern recognition. Inspired from biology, the first idea of multi layered perceptron was proposed in 1958 by Frank Rosenblatt.

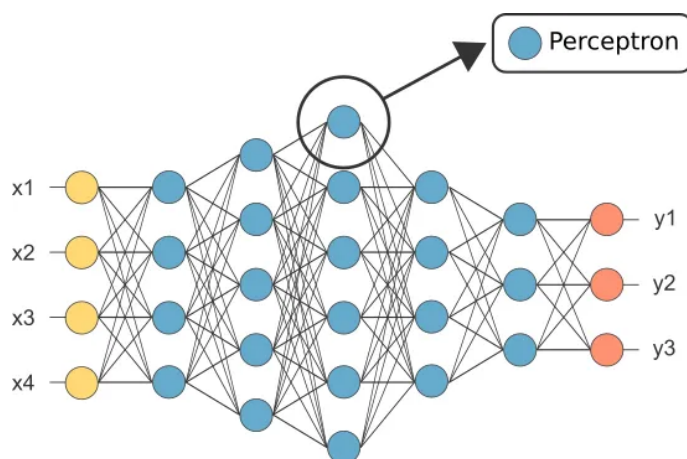


Figure 10: Network of interconnected perceptrons. Figure adapted from [12]

#### Neuron

An Artificial Neural Network (ANN) is made of many interconnected neurons. Each neuron takes in some numbers and multiplies them by some other numbers known as **weights** and then add a number called **bias**. The weights act as a mechanism to focus on, or ignore. The weights get summed together with the bias.

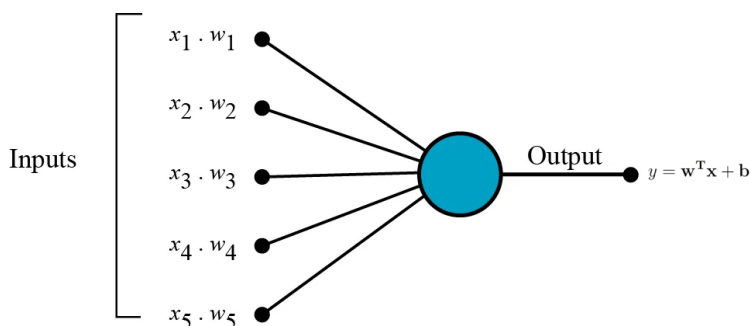
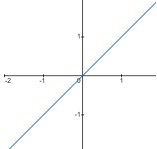
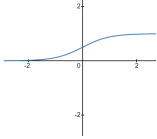
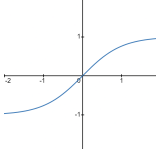
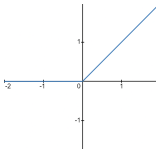


Figure 11: Ouput of a neuron as a function of weights & bias. Figure adapted from[7]

## Activation Function

The summed values are now transformed by the **activation function**  $y = f(x)$ . Activation functions bring the necessary non-linearity between each layer of neurons. Without activation functions the entire neural network could be simplified into a single neuron. Some activation functions are mentioned below.

Activation Function	Plots	Equation
Linear		$y = x$
Sigmoid		$y = \frac{1}{1+e^{-x}}$
Tanh		$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
ReLU		$y = \max(x, 0)$

## 3.2 Feed Forward Neural Networks

Feed Forward neural networks(FFNs) have the simplest network architecture. They consist of fully connected layer where each neuron from the previous layer connects to every neuron in the next layer.

1. **Input Layer:** This layer takes the values of input given to the networks. The inputs are number. Any medium (e.g. Images, Audio) can be converted into a list of number and feed into the network
2. **Hidden Layers:** These layers make up the bulk of the network and are responsible for the pattern recognition.
3. **Output Layer:** This layer is the final output of the neuron and can have various representation. E.g., prediction of a single value, probabilities in multi class classification, pixel values in an image or text embedding in latent space.

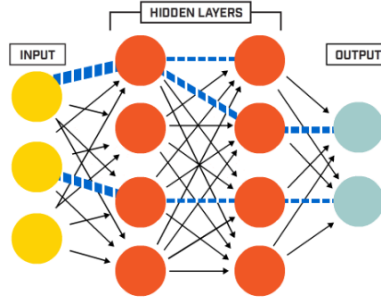


Figure 12: A feed forward network made of input, hidden and output layers

The weights and bias of every neuron at a given layer  $l$  are stored in the weight matrix  $\mathbf{W}^l$  and the bias vector  $\mathbf{b}^l$ .

### 3.3 Training Neural Networks

Every weight and bias in the network affects the neuron activations and hence changes the output behaviour of the network. The core idea of machine learning lies in training these complex networks automatically based on training data.

To train a network we devise algorithms called backpropagation and gradient descent that update the parameters every iteration.

#### Forward Propagation

This algorithm computes the neuron activations are every level step-by-step.

---

#### Algorithm 2 Forward Propagation in Neural Networks

---

**Require:** Input data  $X$ , weights  $W$ , biases  $b$

- 1: Initialize input layer with  $X$
- 2: **for**  $l$  in 1 to number of hidden layers **do**
- 3:     Calculate weighted sum:

$$\mathbf{z}^l = \mathbf{W}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l$$

- 4:     Apply activation function:

$$\mathbf{a}^l = \sigma(\mathbf{z}^l)$$

- 5: **end for**

- 6: Calculate output layer:  $\mathbf{z}^L = \mathbf{W}^L \cdot \mathbf{a}^L + \mathbf{b}^L$
  - 7: Apply final activation function:  $\hat{\mathbf{y}} = \mathbf{a}^L = \sigma(\mathbf{z}^L)$
  - 8: **return**  $\hat{\mathbf{y}}$
-



## Backward Propagation

The backward propagation involves first calculating the error in the final output and then determining the change in the parameter needed to minimize this error. Derivatives of parameters with respect to activation values are calculated at each layer step-by-step by using chain-rule for differentiation. The word 'backwards' is used because the derivatives are calculated in the reverse order to forward propagation.

---

**Algorithm 3** Stochastic Gradient Descent

---

- 1: Initialize network weights and biases
- 2: Set learning rate  $\alpha$
- 3: Set number of epochs  $N$
- 4: **for**  $i = 1$  to  $N$  **do**
- 5:     Randomly select a training sample  $(\mathbf{x}, \mathbf{y})$
- 6:     **Forward propagation:**
- 7:         Compute the activation of each neuron in the network
- 8:     **Backward propagation:**
- 9:         Compute the derivative of the loss function with respect to the output layer activations:

$$\delta^{(L)} = \frac{\partial L}{\partial \mathbf{a}^{(L)}}$$

- 10:     Compute the derivative of the loss function with respect to the weights and biases:
- 11:     **for** each layer **from**  $L - 1$  **to** 2:
- 12:         Compute the derivative of the loss function with respect to the pre-activation:

$$\delta^{(l)} = \frac{\partial L}{\partial \mathbf{z}^{(l)}} = \delta^{(l+1)} \frac{\partial \mathbf{a}^{(l+1)}}{\partial \mathbf{z}^{(l)}} = ((\mathbf{W}^{l+1})^T \times \delta^{(l+1)}) \odot \sigma'(\mathbf{z}^{(l)})$$

- 13:     Compute the derivative of the loss function with respect to the weights and biases:

$$\frac{\partial L}{\partial \mathbf{w}^{(l)}} = \delta^{(l)} \times (\mathbf{a}^{(l-1)})^T \qquad \frac{\partial L}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

- 14:     Update weights and biases using gradient descent:

$$\mathbf{w}^{(l)} \leftarrow \mathbf{w}^{(l)} - \alpha \frac{\partial L}{\partial \mathbf{w}^{(l)}} \qquad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \frac{\partial L}{\partial \mathbf{b}^{(l)}}$$

- 15:     **end for**
  - 16: **end for**
-

## Batch Gradient Descent

Stochastic gradient descent involves updating the parameters after processing a random training sample. This is not the best in practice as a single training sample doesn't generalize to the entire training data. Hence, we should rather process a batch of training samples and then add the losses. When gradient descent is performed on a batch of samples, the losses converge faster in comparison to SGD.

## Loss Function

As already discussed in Supervised Learning, the commonly used loss functions is the Categorical Cross Entropy loss

$$L = - \sum_{i=1}^C y_i \log(\hat{y}_i) \quad (10)$$

It calculates the entropy between two multi-dimensional vectors.

## Optimizers

The step where the parameters are updated based on learning rate and gradients is performed by the optimizer. Several optimizers use different techniques to calculate the updated parameters for better convergence.

1. **Gradient Descent:** Both SGD and mini-batch gradient descent apply similar strategies.

**For each parameter  $\theta_i$ :**

$$\theta_i \leftarrow \theta_i - \eta \cdot \frac{\partial \mathcal{L}}{\partial \theta_i}$$

SGD does not offer strong convergence guarantees while batch gradient descent is computationally slow. Mini batch gradient descent is a compromise between these two. All of these methods are very sensitive to learning rate and are prone to get stuck in a local minima.

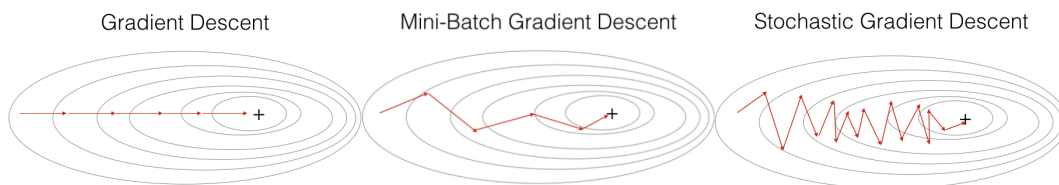


Figure 13: A comparison between gradient descent, mini-batch and stochastic gradient descent

2. **Adaptive Gradient Descent(AdaGrad):** AdaGrad uses adaptive learning rate for each of the weights. It performs smaller updates for parameters associated with frequently occurring features.

The gradient  $g_{t,i}$  at time step  $t$  is then the partial derivative of the loss function w.r.t. to the parameter  $\theta_{t,i}$  at time step  $t$ ,  $\alpha$  is the learning rate.

$$g_{t,i} = \nabla_{\theta} J(\theta_{t,i}) \quad \theta_{t+1,i} = \theta_{t,i} - \Delta\theta_{t,i}$$

$$\Delta\theta_{t,i} = \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

where  $G_{t,i}$  is the sum of squares of past gradients of  $\theta_i$ ,  $\epsilon$  is a small number added for numerical stability.

$$G_{t,i} = \sum_{\tau=1}^t g_{\tau,i}^2$$

The advantage of AdaGrad in that there is no need to do tune the hyperparameter  $\alpha$ . However, AdaGrad suffer from vanishing gradients after long training steps due to accumulations of squared gradients  $G_{t,i}$ .

3. **RMSProp:** This algorithm computes an exponentially decaying average of gradients. RMSProp is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients.

The running exponential average is given by

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

Usually  $\gamma = 0.9$ . The parameter updates take the form

$$\Delta\theta_{t,i} = \frac{\alpha}{\sqrt{E[g^2]_{t,i} + \epsilon}} g_{t,i}$$

4. **Adam:** Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction.

Hyper-parameters  $\beta_1, \beta_2 \in [0, 1)$  control the exponential decay rates of these moving averages. We compute the decaying averages of past and past squared gradients  $m_t$  and  $v_t$  respectively as follows:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) g_t$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) g_t^2$$

The parameter updates take the form

$$\Delta\theta_{t,i} = \frac{\alpha}{\sqrt{v_{t,i} + \epsilon}} m_{t,i}$$

Adam is considered the best amongst all the algorithms. Any reasonable values of learning rate will yield good results as Adam algorithm is very robust.

## 4 Convolutional Neural Networks

In recent years, Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision and become the cornerstone of many state-of-the-art image and video analysis applications. Unlike traditional neural networks, CNNs are specifically designed to exploit the spatial structure and hierarchical patterns present in visual data. Inspired by the organization of the visual cortex in living organisms, CNNs have demonstrated exceptional performance in tasks such as image classification, object detection, and semantic segmentation.

### 4.1 Convolution operation

The key idea behind CNN's is the convolution operation that extracts features from a given data. E.g.- Consider a  $n \times n$  2D grayscale image. It can be represented using a square matrix of size  $n$ . A convolution operation performs a dot product between the kernel and source matrix by sliding the kernel over the source layer.

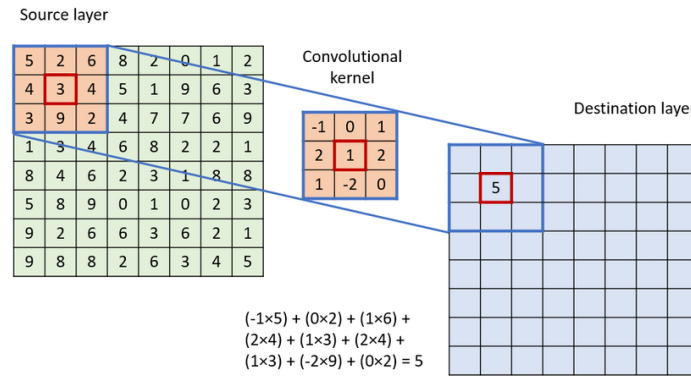


Figure 14: Convolution operation involving kernel and source layer

This process yields another matrix of smaller size with only the essential features of the source matrix highlighted.

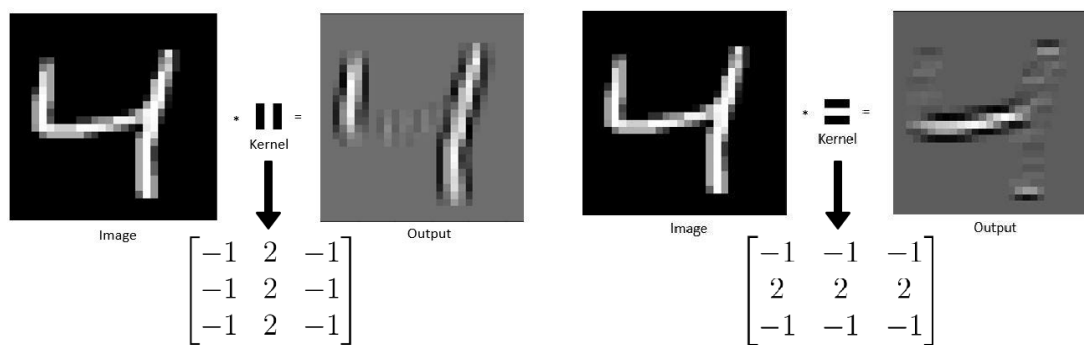


Figure 15: Example of vertical and horizontal edge filter

There are certain hyperparameters that can be controlled in this operation

1. **Kernel Size:** The size of the kernel. Most commonly the kernel is a square matrix of size 3.
2. **Stride:** Stride in convolution refers to the step size or the number of pixels by which the convolutional filter slides over the input image during the convolution operation, affecting the output size. It is an integer greater than or equal to 1.
3. **Padding:** Padding refers to the technique of adding additional rows and columns of zeros around the input image, enabling better preservation of spatial dimensions and information at the edges during the convolution operation.

Each of these parameters control the output dimension.

$$\text{Output Size} = \left\lfloor \frac{\text{Input Size} + 2 \times \text{Padding} - \text{Kernel Size}}{\text{Stride}} + 1 \right\rfloor$$

## 4.2 Convolutional Layer

The input layer contains 3 channels, corresponding to the red, green and blue color channels. The convolutional layers are the foundation of CNN, as they contain the learned kernels (weights), which extract features that distinguish different images from one another. Each link represents a unique kernel, which is used for the convolution operation to produce the current convolutional neuron's output or activation map.

The convolutional neuron performs an element wise dot product with a unique kernel and the output of the previous layer's corresponding neuron. This will yield as many intermediate results as there are unique kernels. The convolutional neuron is the result of all of the intermediate results summed together with the learned bias.

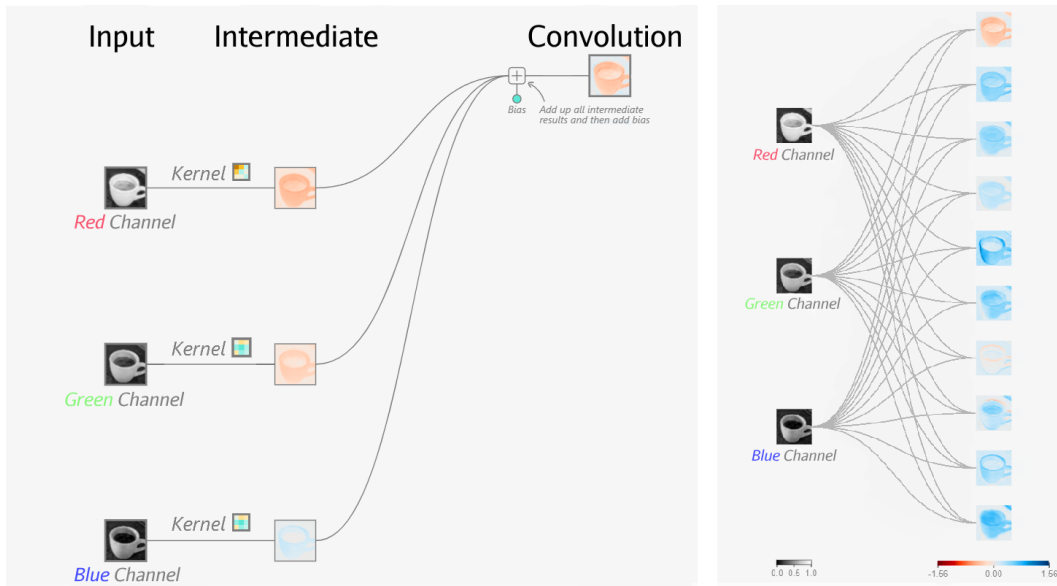


Figure 16: Input and second layer of a CNN. Figure adapted from [9]

### 4.3 Max Pooling

Max pooling is a down sampling technique, where the input feature map is divided into non-overlapping regions, and the maximum value within each region is selected as the representative value, resulting in a reduced spatial dimensionality while preserving important features.

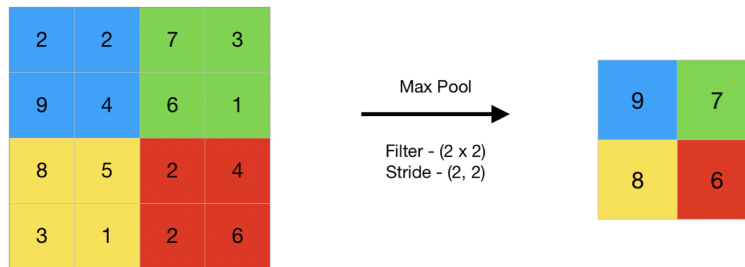


Figure 17: Example of max pooling

There another type of pooling called average pooling which perform an average over the kernel. It is not used as often as max pooling due adding computational complexity.

### 4.4 CNN Architecture

The architecture of CNNs bears similarities to that of a Feed Forward network. However, there are distinct components and operations unique to CNNs that make them highly effective in processing visual data.

- In a CNN, each neuron in the network outputs a matrix instead of a single number, representing a feature map capturing local patterns in the input.
- To reduce the dimensionality of these intermediate feature maps and introduce non-linearity, CNNs incorporate Max pooling layers. This down sampling operation helps in discarding irrelevant information and preserving important features.
- Additionally, activation layers are employed to introduce non-linearity in the network. Common activation functions such as ReLU (Rectified Linear Unit) are applied element-wise to the feature maps.
- A flatten layer is utilized to convert the multidimensional feature maps into a high-dimensional vector. This vectorized representation forms the basis of the classifier part of the network, which aims to make predictions or classifications based on the learned features. The classifier typically comprises a few fully connected layers, where each neuron is connected to all the neurons in the previous layer, followed by a final layer that outputs the desired predictions.

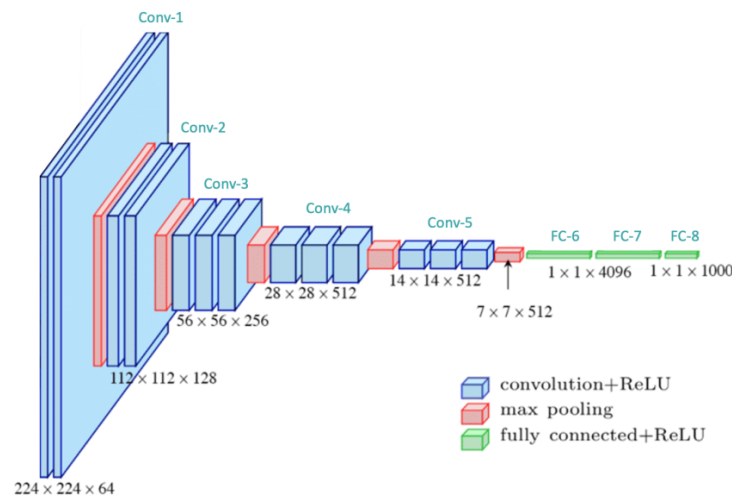


Figure 18: Architecture of VGG-16. Figure adapted from [13]

E.g. VGG-16 developed in 2014 consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 maintains a stack of convolutional and pooling layers, gradually increasing the depth as the network progresses. The total parameter count for VGG is approximately 138 million.

## 4.5 Parameter Sharing

Weight sharing is the key factor that sets CNNs apart from fully connected networks. In a convolutional layer, each neuron shares its weights with multiple neurons in the



previous layer, corresponding to different spatial locations. By using the same set of weights across different locations, CNNs exploit the spatial regularities present in the data, enabling them to learn and generalize from local patterns. This parameter sharing significantly reduces the number of independent parameters that need to be learned, making CNNs more efficient.

Furthermore, parameter sharing plays a crucial role in the translation equivariance property of CNNs. Translation equivariance refers to the ability of CNNs to recognize patterns regardless of their specific location in the input. Because the shared weights are applied across different spatial positions, the network learns to detect the same features irrespective of their position.

## 5 Sequence Models

Sequence models are a fundamental class of neural networks designed to process and analyze sequential data, where the order of elements matters. Unlike Dense and Convolutional Neural Networks (CNNs) that handle fixed-size inputs, sequence models can handle variable-length sequences like sentences, audio, and time-series data.

One of the most crucial applications of sequence models is natural language processing (NLP), where they can be used for tasks like language translation, sentiment analysis, and text generation. Sequence models, such as Recurrent Neural Networks (RNNs) and their variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), excel at capturing long-term dependencies in sequential data.

Additionally, sequence models are essential in time-series forecasting, speech recognition, and music generation, among others. Their ability to process sequential data and consider context over time makes them invaluable in a wide range of real-world applications, advancing artificial intelligence and making computers more capable of understanding and generating human-like sequences.

### 5.1 RNN

#### Basic Architecture and Working Principles

Simple Recurrent Neural Networks (RNNs) are a class of neural networks designed to process sequential data by maintaining hidden states that capture information from previous time steps. They have a fundamental architecture that allows them to handle sequential information effectively.

Figure 19: Architecture of RNN. Figure adapted from [10]

#### Recurrent Connections and Feedback Loops

The RNN architecture consists of a single hidden layer that maintains a hidden state vector  $\mathbf{h}_t$  at each time step  $t$ . The hidden state vector  $\mathbf{h}_t$  is computed based on the current input  $\mathbf{x}_t$  and the previous hidden state

$\mathbf{h}_{t-1}$ , as shown below:

$$\mathbf{h}_t = \sigma(\mathbf{W}^{(hh)}\mathbf{h}_{t-1} + \mathbf{W}^{(hx)}\mathbf{x}_t + \mathbf{b}^{(h)})$$

where  $\mathbf{W}^{(hh)}$  and  $\mathbf{W}^{(hx)}$  are the weight matrices for the recurrent and input connections, respectively, and  $\mathbf{b}^{(h)}$  is the bias vector. The hidden state vector  $\mathbf{h}_t$  is then used to compute the output  $\mathbf{y}_t$  at time step  $t$ :

$$\mathbf{y}_t = \sigma(\mathbf{W}^{(yh)}\mathbf{h}_t + \mathbf{b}^{(y)})$$

where  $\mathbf{W}^{(yh)}$  and  $\mathbf{b}^{(y)}$  are the weight matrix and bias vector for the output layer, respectively. The output  $\mathbf{y}_t$  is then used to compute the loss  $\mathcal{L}_t$  at time step  $t$ :

## Vanishing and Exploding Gradient Problems

During training, RNNs can suffer from vanishing gradients, where the gradients of the loss function with respect to the model's parameters become extremely small. This occurs when long sequences are processed, and the gradients are multiplied repeatedly during backpropagation, leading to values close to zero. As a result, the model becomes insensitive to long-range dependencies, limiting its ability to learn from distant information in the sequence.

Another issue that can arise during training is the exploding gradient problem, where the gradients become extremely large. This can occur when the gradients are multiplied repeatedly during backpropagation, leading to values that grow exponentially. As a result, the model becomes unstable and fails to learn effectively.

To address these gradient problem, various techniques can be employed, such as

- **Gradient Clipping:** Gradient clipping involves capping the gradients to a specific threshold during backpropagation, preventing them from becoming too large. This ensures that the gradients do not explode and allows the model to learn from long-range dependencies more effectively.
- **Weight Initialization:** Weight initialization techniques can be used to ensure that the gradients do not vanish or explode during training. For example, the Xavier initialization technique initializes the weights to random values sampled from a normal distribution with a mean of 0 and a variance of  $\frac{1}{n}$ , where  $n$  is the number of inputs to the neuron. This ensures that the variance of the outputs remains the same across different layers, preventing the gradients from vanishing or exploding.
- **Activations Function:** Rectified Linear Unit (ReLU) activation function is commonly used in RNNs to address the vanishing gradient problem because they saturated in only one direction.

## References

- [1] Chaitanya Katti. “Summer-Of-Science-2023” Github Repo, Available online: <https://github.com/ChaitanyaKatti/Summer-Of-Science-2023>
- [2] Andrew NG. “Machine Learning Specialization.” Coursera, Available online: <https://coursera.org/specializations/machine-learning-introduction>.
- [3] Andrew NG. “Deep Learning Specialization.” Coursera, Available online: <https://www.coursera.org/specializations/deep-learning>.
- [4] Aurélien, Géron (2019) *Hands On Machine Learning With Scikit-Learn, Keras, and TensorFlow*, O’Reilly.
- [5] freeCodeCamp. “Artificial Intelligence and Machine Learning.” Available online: <https://www.freecodecamp.org/news/ai-vs-ml-whats-the-difference/>.
- [6] Ayush Pant, Medium.com “Introduction to Machine Learning for Beginners.” Available at: <https://towardsdatascience.com/introduction-to-machine-learning>.
- [7] Mohammed Terry-Jack, Medium.com “Deep Learning: Feed Forward Neural Networks (FFNNs)” Available at: <https://medium.com/@b.terryjack/introduction-to-deep-learning>
- [8] Nagesh Singh Chauhan, KDnuggets. “Optimization Algorithms in Neural Networks” Available at: <https://kdnuggets.com/2020/12/optimization-algorithms-neural-networks>
- [9] “CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization,”. **Paper:** <https://arxiv.org/abs/2004.15004>  
**Website:** <https://poloclub.github.io/cnn-explainer/>
- [10] Jeremy Jordan “Setting the learning rate of your neural network.” Available at: <https://www.jeremyjordan.me/nn-learning-rate/>
- [11] Nadeem Medium.com “Gradient Descent Optimization Machine Learning” <https://nadeemm.medium.com/gradient-descent-optimization-machine-learning>
- [12] Wu, Junhao, and Zhaocai Wang. 2022. “A Hybrid Model for Water Quality Prediction Based on an Artificial Neural Network, Wavelet Transform, and Long Short-Term Memory” *Water* 14, no. 4: 610. <https://doi.org/10.3390/w14040610>
- [13] Vaibhav Khandelwal, Medium.com “The Architecture and Implementation of VGG-16” Available online: <https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b>