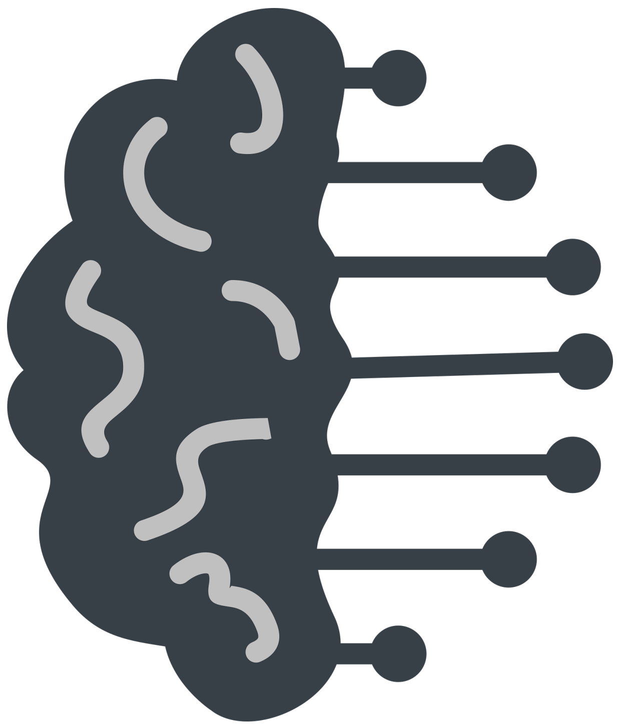


ENDTERM REPORT
Machine Learning and Artificial Learning

Siddhant Mulkikar (23B0956)
Mentor: Sabyasachi Samantaray

July 26, 2024

Contents

1 Objective

This midterm report goes through the basics of Machine Learning, followed a dive into some popular ML algorithms.

2 Introduction

Machine Learning is the study of building mathematical models and algorithms to help understand data. It is often seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions.

3 Categories in Machine Learning

Machine learning can be broadly categorized into two main types: supervised learning and unsupervised learning.

3.1 Supervised Learning

Supervised learning involves training the model on a labelled dataset, which means that each training example is paired with the correct target value. The model aims to find a relationship or map from the input features to the labelled value, such that it can get the closely accurate target value for any prediction we ask from it. It is further subdivided into two categories, regression and classification. Regression has continuous labels, like a real number in a range, while classification has discrete labels i.e. categories.

3.2 Unsupervised Learning

Unsupervised learning involves modelling the underlying structure or distribution in the data in order to learn more about the data. The model is trained on an unlabelled dataset, which means that the model is not provided with the correct target value. These models involve algorithms like clustering, anomaly detection and dimensionality reduction. Clustering involves grouping similar data points together, while dimensionality reduction involves reducing the number of input variables in the dataset, representating the data more clearly and anomaly reduction involves detecting outliers in the data.

4 Model Selection

4.1 Hyperparameters

A hyperparameter is a parameter, such as the learning rate or batch size, which specifies details of the learning process. These are set before the learning process and are not interfered with while fitting the model to the training set. Hyperparamters usually tend to decide the efficiency of the model, and therefore, need to be tuned and optimized to get the most accurate predictions from the model.

4.2 Model validation

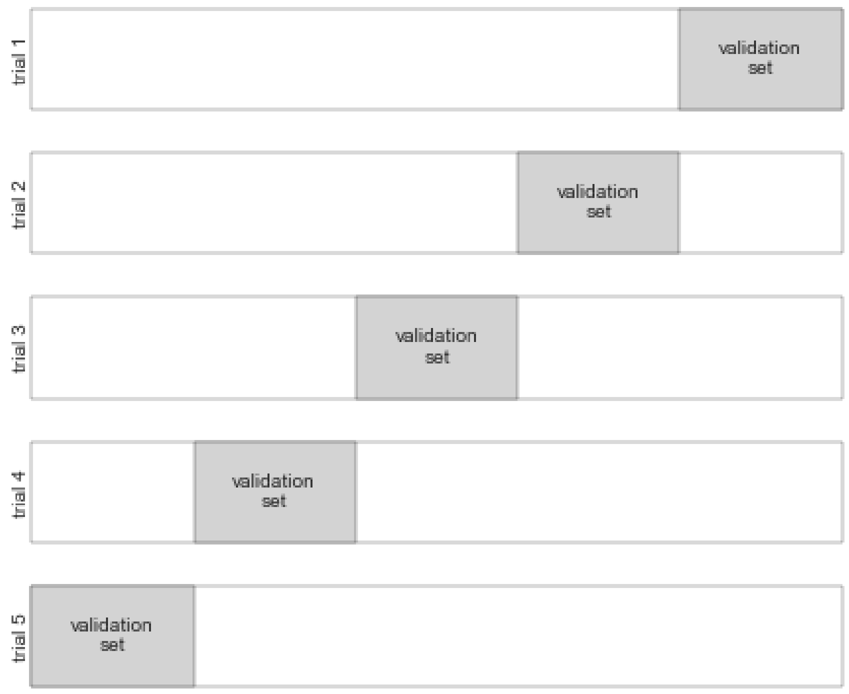
Model validation is important to check the effectiveness of our model, and this is done after choosing the model and the hyperparameters associated with it. However the limitaion we have is that our training set is limited, which forces us to develop ways to validate based on the limited data we have.

Holdout sets

In this method, a subset of data is withheld from the training set, and then this data is used to check the model performance by predicted the target variables for this holdout set and comparing it with its labels. The problem with this is we are reducing the already limited size of training examples even more. This is where cross validation comes in.

Cross Validation

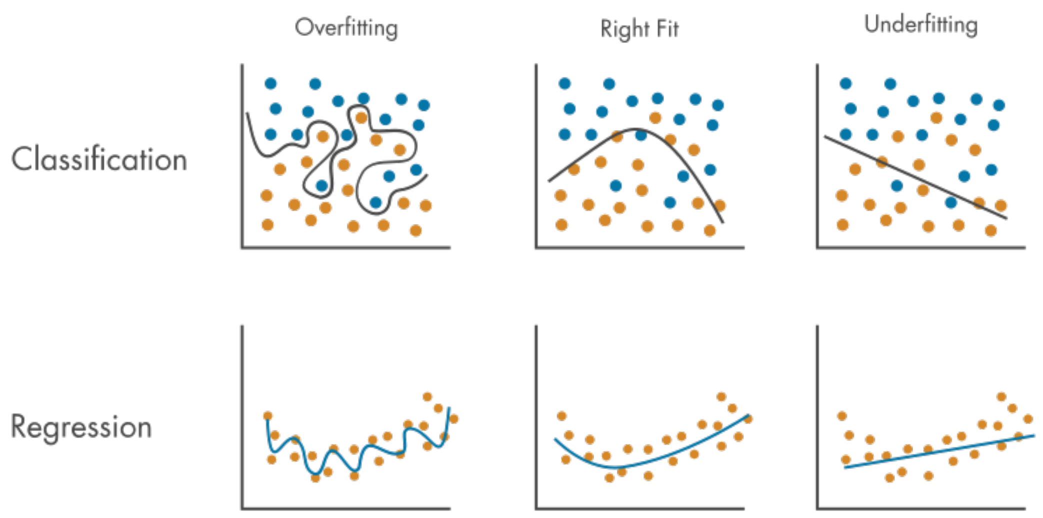
In cross validation, the training set is divided into subsets of equal size, and the model is trained on all of them individually, while the remaining set is used to validate the model. This is repeated for all the subsets and the final validation score is gotten by combining all the validation scores from each subset, which can be done by taking the mean or some other way. This method is more effective than holdout sets, because none of the training examples are lost in the process.



Five-fold cross validation

Bias-Variance Tradeoff

Bias is defined as the error between the predicted value and the actual value, which we are trying to predict. Variance is defined as the variability of the model prediction for a given data point. Simply put, bias is how much does the model simplify the relation between the input and output labels of the training set, which variance is how much the model takes into account the complexity or noise in the training set.
Models with high bias, makes the model more flexible, however it might increase the error in the model prediction. Such a model is said to be underfitting the data. Whereas, models with high variance, take into account even minute changes in the training data, capturing a lot of noise in the process. While they decrease the error in the training set drastically, they might not perform well on sets other than the training set, as they are trained too specific to the training data and are said to overfit the data.



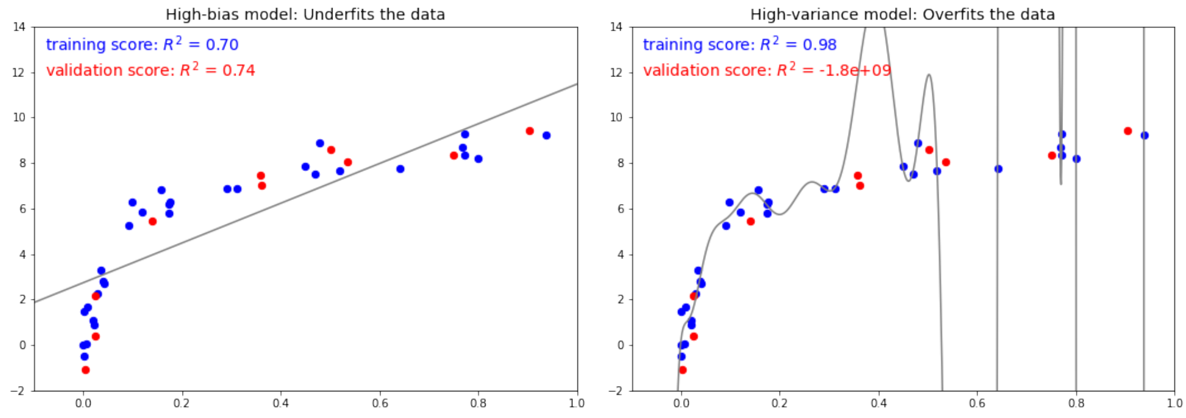
The age-old problem in machine learning is to create the best model is to find the sweet spot between bias and variance, such that the model performs well on all datasets, as models with high bias often have low variance and vice versa. This is known as the **bias-variance tradeoff**. To solve this problem, we need to first quantify what the "best model" should be like, and we can use the R^2 score for this.

R^2 score or coefficient of determination is a statistical measure of how well a model performs relative to a simple mean of the target values. $R^2 = 1$ is a perfect score, whereas $R^2 = 0$ means the model is as good as the mean of the target values. A negative R^2 score means the model is worse than that. It is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{1}$$

Here, y_i is the actual target value, $\hat{y}_i$ is the predicted target value and $\bar{y}$ is the mean of the target values.

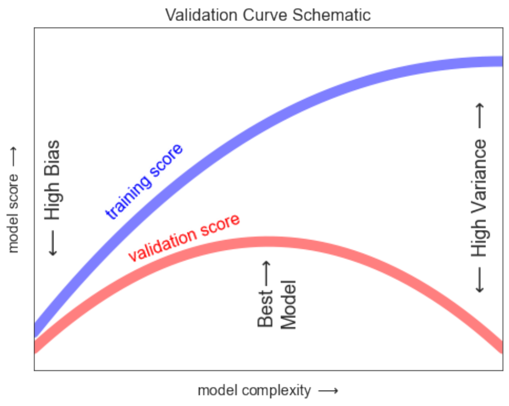
The model is run on the training set and the validation set, and the R^2 score is calculated for both.



As you can see the above figure, for the high-bias model, the performances of the model on the validation set and training set are similar. Whereas for the high-variance model, the performance on the training set is much better than the validation set.

Validation Curve

Validation curve is the plot of the model score vs the model complexity. Lets understand it through an illustration.



From the above diagram, we can observe that:

- The training score is always higher than the validation score, as the model will give better predictions on the data it was trained on, than the data which it encounters for the first time.
- While the training score increases with model complexity, validation score increases and then decreases, like a bell curve. This is because, as the model complexity increases, the model starts to overfit the data, and hence the validation score decreases and at low model complexity, the model is too simple to give accurate predictions, hence both training and validation scores are low.
- At an intermidiate model complexity, the validation score is the greatest, which indicates the suitable level of model complexity for the bias-variance tradeoff, to get the best predictions.

5 Supervised Learning Algorithms

5.1 Linear Regression

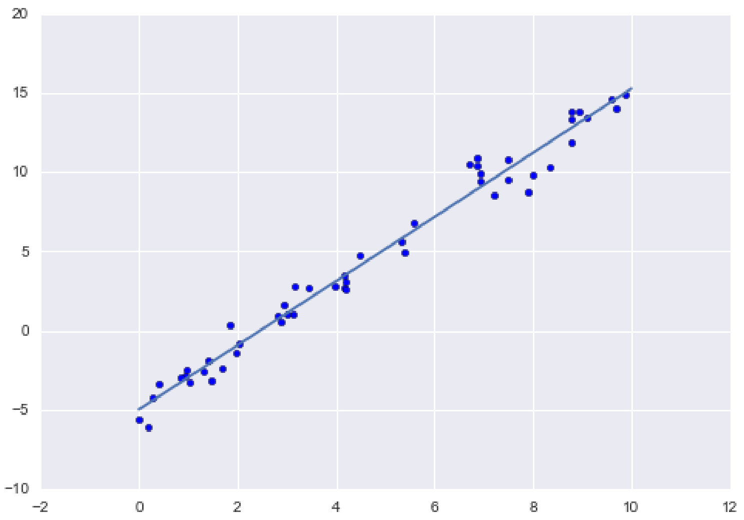
Linear regression is a good starting point for regression models. The simplest linear regression model is fitting a straight line to a dataset, but it can be extended to model more complicated data behaviour. The simplest linear regression model is a straight line fit, which is a model of the form :

$$f_{w,b}(x) = \langle w, x \rangle + b \tag{2}$$

The ultimate goal of the model is to find a weight vector w and bias b, such that the predicted value of the target variable is as close as possible to the actual value over all training examples. Given a input feature vector $x =< x_1, x_2, \dots, x_m >$, the model predicts the output $\hat{y}$, which is called the target variable, as:

$$\hat{y} = f_{w,b}(x) = \langle w, x \rangle + b \tag{3}$$

where, $w =< w_1, w_2, \dots, w_m >$ is a vector of weights, b is the bias term, both of which are learned by the model during the learning process.



Straight line fit after linear regression

In order to train the model, a cost function is defined, which measures the accuracy of the model's prediction. One of the most popular cost functions for linear regression is Mean sqaured error (MSE), which measures the average sqaure deviation between the actual and predicted target, values which is defined as:

$$MSE = \frac{1}{m} \sum_{i=1}^{n} (y^i - \hat{y^i})^2 = L(w, b) \tag{4}$$

The model is trained by updating the weight vector w such that the cost function is minimized, which is done by using another algorithm like gradient descent.

Gradient Descent

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. The algorithm updates the weights vector and bias on each iteration, till it reaches a minima for the cost function.

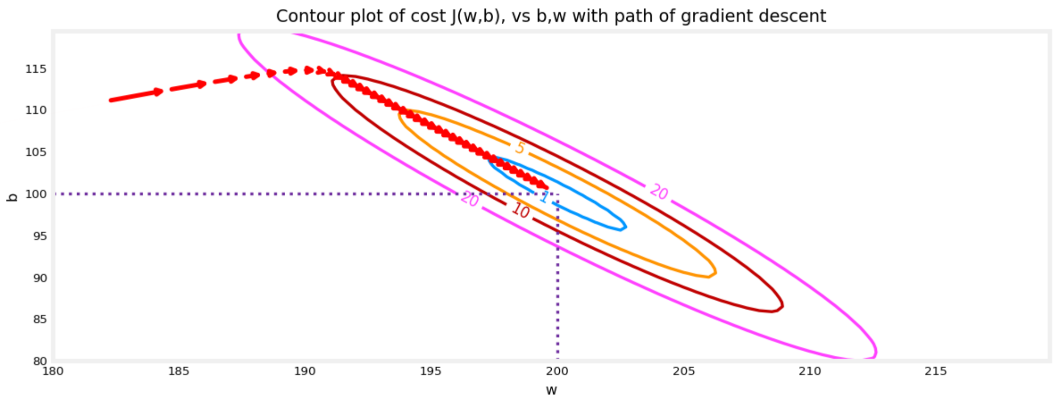
$$w' = w - \alpha \frac{\partial L(w, b)}{\partial w} \tag{5}$$

$$b' = b - \alpha \frac{\partial L(w,b)}{\partial b} \tag{6}$$

where α is the learning rate, which determines the size of the step taken in the direction of the gradient, and can be tuned to get the best predictions. If the number of training examples are m and the number of features are n, then the gradient of the cost function with respect to the bias and $j^{th} weight$ is given by:

$$\frac{\partial L(w,b)}{\partial b} = \frac{2}{m} \sum_{i=1}^{m} (\hat{y^i} - y^i) \tag{7}$$

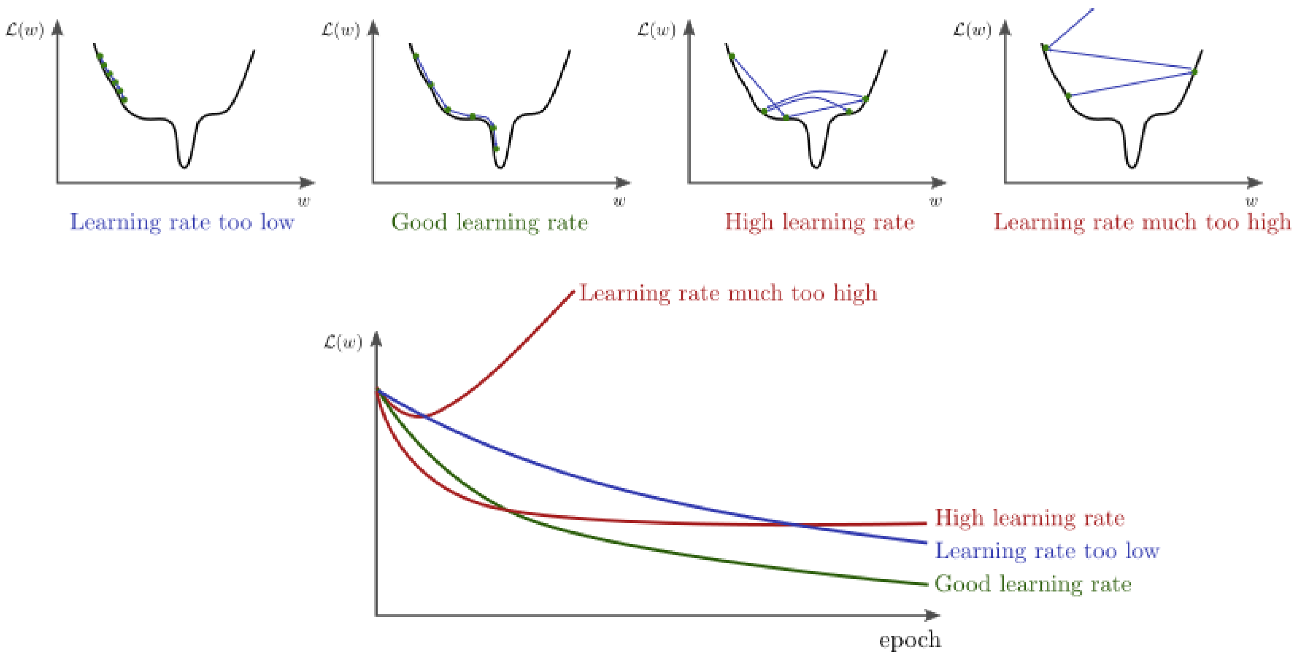
$$\frac{\partial L(w_j,b)}{\partial w_j} = \frac{2}{m} \sum_{i=1}^{m} {x_j}^i (\hat{y^i} - y^i) \tag{8}$$



However, gradient descent can give inaccurate predictions if the algorithm doesn't reach the minima, i.e. it diverges from the minima. The convergence of gradient descent into a minima is determined by the learning rate α.

Learning rate (α)

Learning rate is a hyperparameter that determines the size of the step taken in each iteration of gradient descent.



$L(w)$ vs w for different learning rates α

Lets go through all the cases to see how the learning rate affects the convergence of the gradient descent:

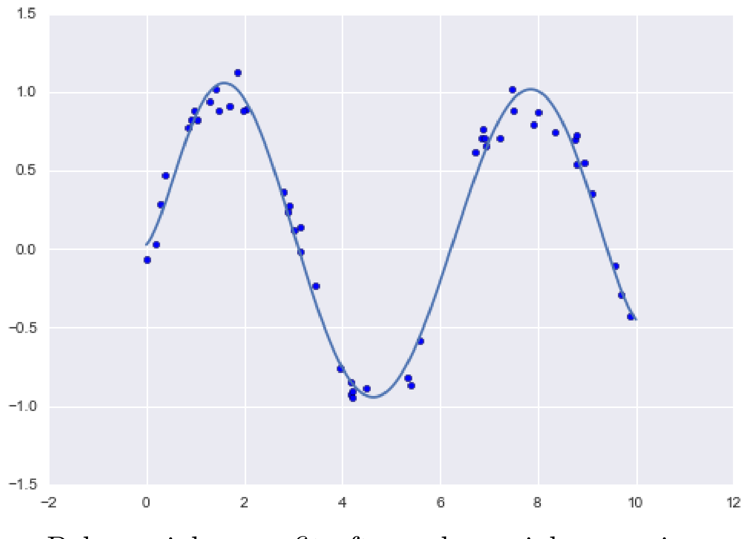
- **Small learning rate:** If α is too low, the algorithm might take a lot of time to converge to the minimum, as the size of step is too small.
- **Optimal learning rate:** If α is chosen properly, gradient descent converges to the minima in a decent number of epochs, which is desired.
- **Large learning rate:** If α is high, it takes a big step during each iteration, such that it might overstep the minima and end up on the other side of the minima. This might result in a delayed convergence or no convergence at all.
- **Too High learning rate:** If α is too high, the cost function would diverg, which might end up increasing the cost function, instead of minimizing it in each iteration.

5.2 Polynomial Regression

Conceptually, polynomial regression is pretty similar to linear regression. The difference is instead of fitting a straight line, a polynomial curve is fitted to the data.
Lets consider an univariate polynomial regression model ($n = 1$), for simplicity, of degree d, with the weights being $w =< w_1, w_2, \ldots, w_d >$. The model is this case would be:

$$f_{w,b}(x) = w_1 x + w_2 x^2 + \cdots + w_d x^d + b \tag{9}$$

This can be thought of a linear regression model of d features, where the features are not independent. Thus, the input feature vector would be $\vec{x} =< x, x^2, \ldots, x^d >$ and the model would be $f_{w,b}(x) = \langle w, \vec{x} \rangle + b$.



Polynomial curve fit after polynomial regression

5.3 Logistic Regression

Despite the name suggesting regression, logistic regression is usually used as a binary classification model, i.e. it has two target variable values - 1 and 0, or True and False, also known as classes. Given an input feature vector $x =< x_1, x_2, \ldots, x_m >$, the model gives the probability of the target variable being 1, as:

$$f_{w,b}(x) = \sigma(\langle w, x \rangle + b) = P(\hat{y} = 1|x) \tag{10}$$

where σ is the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{11}$$

The sigmoid function maps any real number to the range (0,1), which is useful for binary classification, as it can be interpreted as the probability of the target variable being 1.

If the input vectors and their labels are represented as points in the vector space, the hyperplane $z = w \cdot x + b$, divides the vector space into two regions, i.e. two classes, henceforth classifying the dataset.
While one might think of using the MSE cost function for logistic regression, it turns out that the loss function is not convex, which means that gradient descent might not converge to the minima. Hence a different cost function known as the logistic loss function is used, which is defined as:

$$L(f_{w,b}(x), y) = \frac{1}{m} \sum (-y \log(f_{w,b}(x)) - (1-y) \log(1 - f_{w,b}(x))) \tag{12}$$

Gradient descent can used here to train the model in the similar way as linear regression. The gradients for the j^{th} weight and the bias are given by:

$$\frac{\partial L(w,b)}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x) - y^i) \tag{13}$$

$$\frac{\partial L(w_j,b)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} x_j (f_{w,b}(x) - y^i) \tag{14}$$

5.4 Naive Bayes

Naive Bayes models are a set of supervised learning algorithms based on the Bayes theorem, which is:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{15}$$

Lets consider a dataset with n discrete features at first. The probability of the desired class y for a single input feature x_i, is given by:

$$P(y|x_i) = \frac{P(x_i|y)P(y)}{P(x_i)} \tag{16}$$

Considering the dataset has n features, the model calculates the probability of the desired class y for the input feature vector $x =< x_1, x_2, \ldots, x_n >$, as:

$$P(y|x_1, x_2, \ldots, x_n) = \frac{P(x_1, x_2, \ldots, x_n|y)P(y)}{P(x_1, x_2, \ldots, x_n)}$$

$$P(y|x_1, x_2, \ldots, x_n) = \frac{P(y|x_1)P(y|x_2)\ldots P(y|x_n)P(y)}{P(x_1)P(x_2)\ldots P(x_n)}$$

$$P(y|x_1, x_2, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y) \tag{17}$$

If the problem is of binary classifcation, with the classes being y_1 and y_2, we can decide between the two classes by taking the ratio of both the probabilities i.e. $\frac{P(y_1|x_1,x_2,\ldots,x_n)}{P(y_2|x_1,x_2,\ldots,x_n)}$ and check if this ratio is greater or lesser than 1.
More generally for multi-class classification, the class with the highest probability $\hat{y}$ is chosen as the predicted class by:

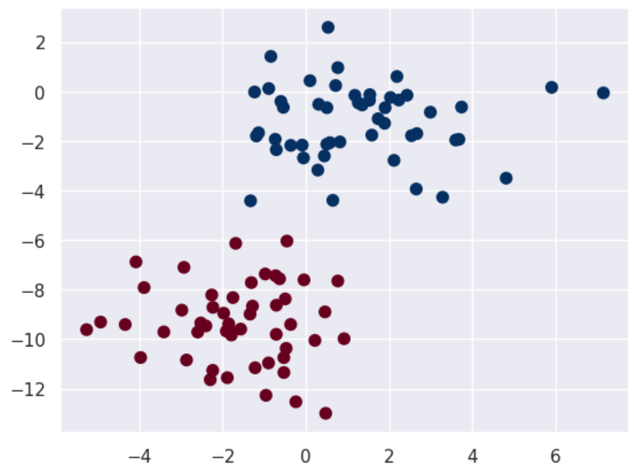
$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y) \tag{18}$$

Gaussian Naive Bayes

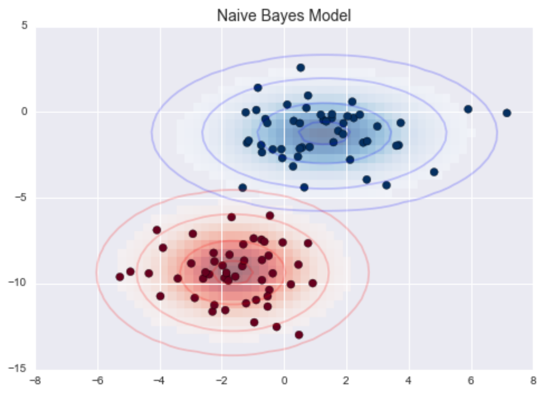
If instead of discrete features, our model has continuous features, the Gaussian Naive Bayes model can be used. The only difference is the way in which the probabilities are calculated. While it is quite self-evident in the case of discrete features, it is not that obvious in the case of continuous features.
The model assumes that the dataset has a Gaussian distribution for each of the classes with no covariance between features (dimensions) i.e. every feature is independent from the other and then calculates the standard deviation σ_y and μ_y for the desired class y, after which the probability for each feature is calculated as:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \tag{19}$$

The steps after this are the same as in the discrete features case.



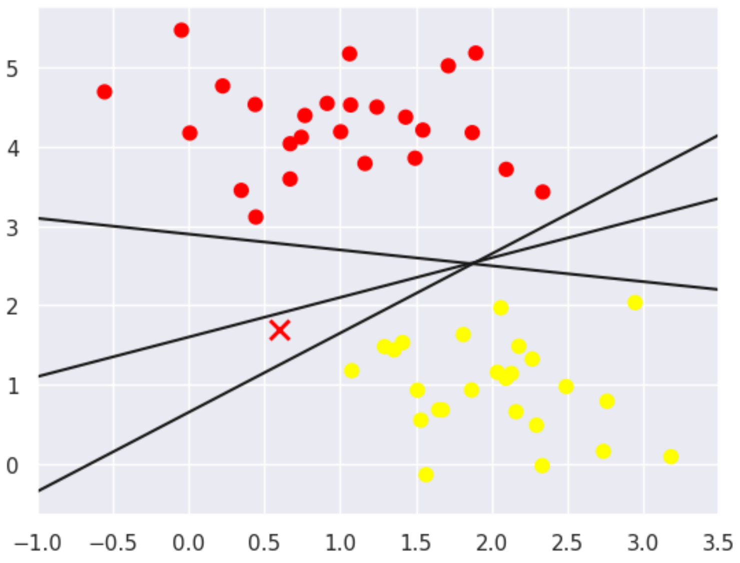
(a) Representation of data points



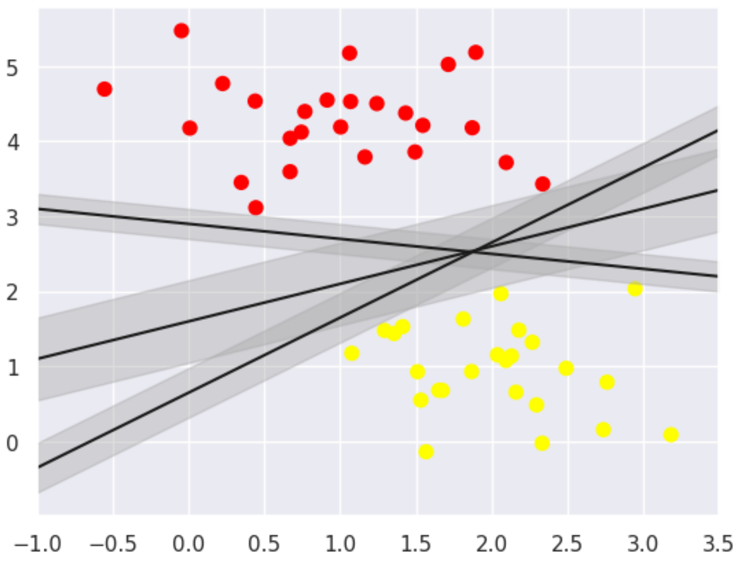
(b) After Gaussian Naive Bayes

5.5 Support Vector Machines

A support vector machine is an supervised learning algorithm usually used for classfication problems. The model is similar to logistic regression, in the sense, that it fits a hyperplane, also known as decision boundary, separating the classes in two or higher dimensions. However, the difference lies in choosing the hyperplane.
Lets consider a binary classification problem, with two classes y_1 and y_2. As we can see in the figure below, there are multiple hyperplanes which can be separators, and the choice of a hyperplane would affect the class which would be assigned to a new input data point, i.e. the red cross in the plot.



SVMs solve this problem by fixing the choice of the hyperplanes. This is done by using a line of non-zero width as the hyperplane, instead of a zero-width line.
The original hyperplane is called the **maximum margin hyperplane**, which we will refer to as MMH hereafter for convenience. The data points which are the closest to the MMH are called the **support vectors**. Two parallel hyperplanes to the MMH which pass through these support vectors are called **marginal planes**. The width between the marginal planes is called the **margin**.
SVMs aim to maximise this margin to get the optimal hyperplane for separation.



Let a hyperplane be defined as:

$$\langle w, x \rangle + b = 0$$

Let the classes be linearly seperable, i.e. they can be separated by a single line between them. If $y_i = 1$, let the point belong to y_1, and if $y_i = -1$, let the point belong to y_2. Taking the margin $2M$, we get For the hyperplane to be the MMH, the following conditions must be satisfied for all $i = 1, 2 \ldots n$, where n is the number of training examples, i.e. the number of data points:

(I) $y_i(\langle w, x_i \rangle + b) \geq 1$

(II) $\max_{w,b} M$

Let's choose a scale for the axes of the data such that $\langle w, x \rangle + b = 1$ at the closest data point x_a to the hyperplane. Let the orthogonal projection of x_a on the hyperplane be x_a'. By definition, it should satisfy the eqaution of the hyperplane.

$$\langle w, x'_a \rangle + b = 0 \tag{20}$$

$\because x_a = x'_a + M\frac{w}{\|w\|}$, as $\frac{w}{\|w\|}$ is the unit vector in the direction of w. Substituting this in (19), we get:

$$w \cdot (x_a - M\frac{w}{\|w\|}) + b = 0$$

$$(w \cdot x_a + b) - M\|w\| = 0$$

Since, $w \cdot x_a + b = 1$, we get:

$$M = \frac{1}{\|w\|} \tag{21}$$

Hence, the second condition can be written as:

$$\min_{w,b} \|w\| \tag{22}$$

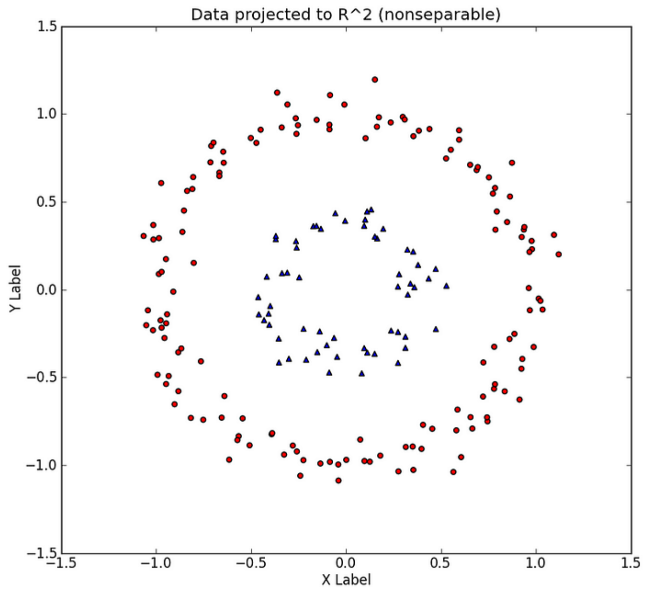
In case of hard margin SVM classifiers, no misclassifcations can be allowed, i.e. the first condition must be satisfied for all $i = 1, 2 \ldots n$. For this classifer, the cost function can be defined as:

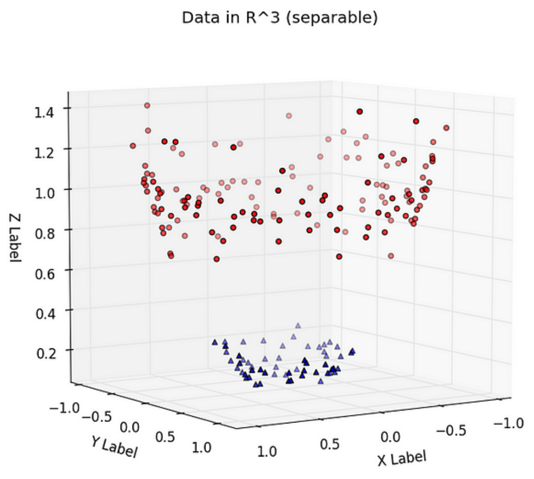
$$L(w, b) = \frac{1}{2} \min_{w,b} \|w\|^2 \tag{23}$$

However, there can be overlaps between classes, for which soft margin SVM classifiers are used, which also allowed for misclassifcations and integrates this into the cost function. The model defines a slack variable ζ_i for i_{th} point, where $\zeta_i = \max(0, 1 - y_i(w \cdot x_i + b))$. The cost function therefore can be defined as:

$$L(w, b) = \frac{1}{2} \min_{w,b} \|w\|^2 + C \sum_{i=1}^{n} \zeta_i \tag{24}$$

The value of C is a hyperparameter, which decides the hardness of the margin. A large C means a hard margin, i.e. it does not allow for points to lie in the margin, while a model with a small C does allow for encompassing points.
In the case of non-linearly seperable data, kernels are used which project the data to a higher-dimensional space defined by the kernel function.





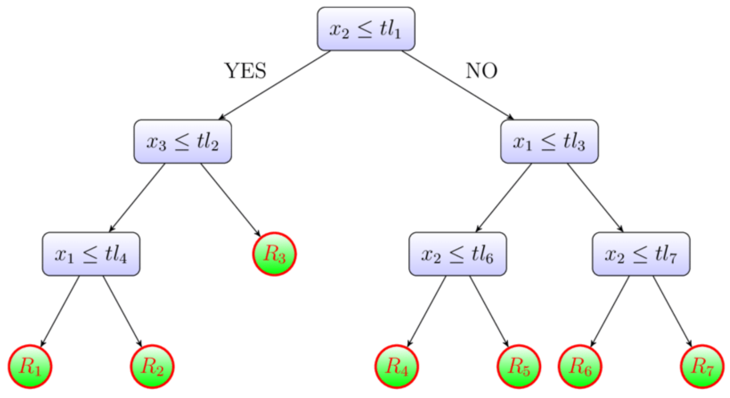
Taking an example of a non-linearly separable dataset in R^2 in the above figure, the data can be projected to R^3 by using the radial basis function (RBF) kernel, which is defined as:

$$K(x, x') = exp(-\gamma \|x - x'\|^2) \tag{25}$$

Now, a hyperplane can be easily constructed in R^3, separating the datasets in two classes. This results in an almost circular (non-linear) decision boundary in R^2.

5.6 Decision Trees

Decision trees is a non-parametric supervised learning algorithm used for classifcation and regression tasks. The model is structured as a rooted tree, consisting of nodes, which are essentially conditions. Each parent node splits into child nodes based on the condition in the node. The tree ends with the leaf nodes, i.e. the stage where the classification or regression is deemed to be completed.



Lets consider we have some points in n-dimensions, marked red and blue. Taking a binary decision tree for this classification task, we start with the root node and compare the value of x_2 with the threshold value tl_1. Based on this condition, the whole n-dimensional space, splits into two regions, and with it the points are also seperated. This process is repeated for each node, where the region passed down from its parent node is splitted further based on the condition of the current node, till the leaf nodes are reached.

Considering the above example, there are 7 leaf nodes, each indicates a non-overlapping region in the n-dimensional space. Splitting at each node can be thought of putting a hyperplane in the n-dimensional space, basically a mini-decision boundary.
The optimal decision tree divides the n-dimensional space into regions, such that each region has only one class of points, which depends on the condition on the nodes. While, the model is said to be non-parametric, the conditions can be thought of parameters, which the model has to learn. This is done by maximising the **information gain** at each split, which is defined as:

$$IG = E(parent) - \sum E(child_i) \tag{26}$$

where IG is the information gain, $E(Data)$ is the entropy of the data before the split and $E(Data|Split)$ is the entropy of the data after the split. The entropy of the data is defined as:

$$E(Data) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

where $p_i = n_i/n$ such that n_i is the number of points of the i_{th} class in the region of the node. From this we can see that decision trees are greedy algorithms as the prioritise the decision which gives the most reward at the current node, without considering any future nodes.

Decision trees can be prone to overfitting and can take a lot of time to train, if not stopped from growning. This can be controlled by setting a maximum depth for the tree, or a minimum number of samples required to split a node, which are essentially hyperparameters set by the user.

5.7 Random Forests

Random forests, simply put, can be thought of as a collection of decision trees, which are trained parallely and then the final prediction is made by merging the outputs of all the individual trees. Hence, it is an ensemble learning method, which means utilising multiple learning algorithms to obtain the prediction.
Ensemble primarily can be of two types, Bagging and Boosting. Bagging is when the models are trained parallely, while Boosting is when the models are trained sequentially, with the output of one model being the input of the next model.
Random forests work on the principle of Bagging.

Bagging

Bagging stands for Bootstrap Aggregation. The steps involved in bagging are:

1. **Bootstrap:** Random subsets of training data is selected with replacement, allowing for multiple instances of the same data point. Each of these subsets is to be trained on a differene decision tree.

2. **Feature selection:** For each sample, a random subset of features is selected, such that the decision tree can split based on only these selected features. The size of this random subset is a hyperparameter, which can be tuned. However, the general consensus by researchers is that the log or square root of the total number of features works best.

3. **Training:** A decision tree is trained on each of the bootstrap samples, with the selected features.

4. **Aggregation:** To get the final prediction, the input data point is passed through all decision trees, and the majority output is assigned as the class of the input data point, in the case of a classification problem. For regression, usually, the average of all the outputs is taken.

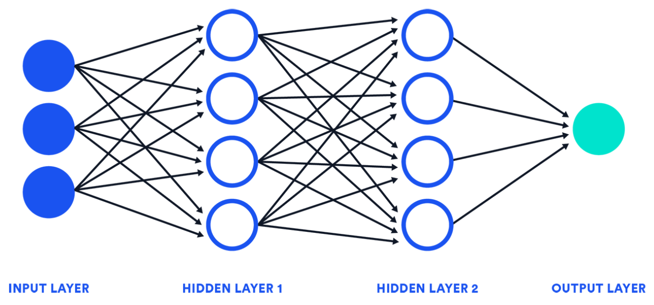
Random forests, while slower, usually perform better than singular decision trees, as they are more robust and thus less prone to overfitting. Decision trees have a high variance and can change a lot with minor changes to the training data, whereas random forests are more resistant to such changes. They also help in feature selection by highlighting important features.

6 Deep Learning

Deep learning is a subset of machine learning, which is based on neural networks. But before we get into deep learning, lets understand what a neural network is and how it works.

6.1 Neural Networks

Neural networks is a machine learning model, closelt inspired by the actual human brain and nervous system. It consists of many nodes, called neurons, arranged sequentially. Each layer is connected to the next layer. The first layer is called the input layer and the last layer is called the output layer. The layers in between are called hidden layers.



As evident from the name, the input layer takes the input data, which is then passed through all the hidden layers, finally ending at the output layer, which gives the prediction. Each neuron takes an input and applies a function or mutltiple functions before passing the transformed input to the next layer.

Neuron

A neuron is the basic unit of a neuron network. It takes a input vector $x =< x_1, x_2, \ldots, x_n >$ from the previous layer. Each neuron has a weight vector $w =< w_1, w_2, \ldots, w_n >$ and bias b associated with it. The neuron first applies a linear function on the input:

$$z = \langle w^T, x \rangle + b$$

Then, an activation function a is applied on z:

$$a(z) = g(z) \tag{27}$$

Now lets consider the L^{th} layer. For any term Q, let $Q^{[L]}$ denote the term in the L^{th} layer. Hence, the number of neurons in this layer be $n^{[L]}$. The weight vector for the j^{th} neuron would be $w_j^{[L]}$ and the weight matrix for the entire layer would be $w^{[L]}$ and the bias matrix to be $b^{[L]}$. Let the input to this layer be $X^{[L]}$.

$$z^{[L]} = w^{[L]}X^{[L]} + b^{[L]} \tag{28}$$

$$z^{[L]} = \begin{bmatrix} w_1^{[L]} \\ w_2^{[L]} \\ \vdots \\ w_{n^{[L]}}^{[L]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1^{[L]} \\ b_2^{[L]} \\ \vdots \\ b_{n^{[L]}}^{[L]} \end{bmatrix} = \begin{bmatrix} \langle w_1^{[L]}, x\rangle + b_1^{[L]} \\ \langle w_2^{[L]}, x\rangle + b_2^{[L]} \\ \vdots \\ \langle w_{n^{[L]}}^{[L]}, x\rangle + b_{n^{[L]}}^{[L]} \end{bmatrix} \tag{29}$$

$$a^{[L]} = g(z^{[L]}) \tag{30}$$

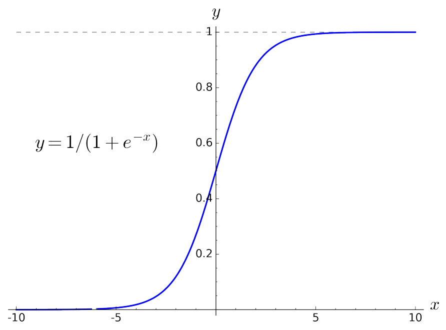
Since, for a hidden layer, the input is the output of the previous layer, i.e. $a^{[L-1]} = X^{[L]}$, and the input layer can be written as $a^{[0]}$. The equation can hence be written as:

$$a^{[L]} = g(w^{[L]}a^{[L-1]} + b^{[L]}) \tag{31}$$

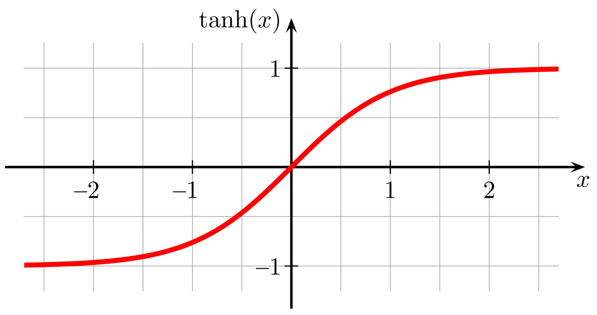
Activation function

An activation function plays an important role in a neural network by introducing non-linearity in it. Without it, the entire network would be linear and can be representated by a single layer, simplifying the NN to a linear regression model, which we don't want as complexity is desired for better predictions.
There are many choices of activation functions available. Some of the popular ones are:

- **Sigmoid:** $g(z) = \frac{1}{1+e^{-z}} = a$ and $a' = a(1-a)$



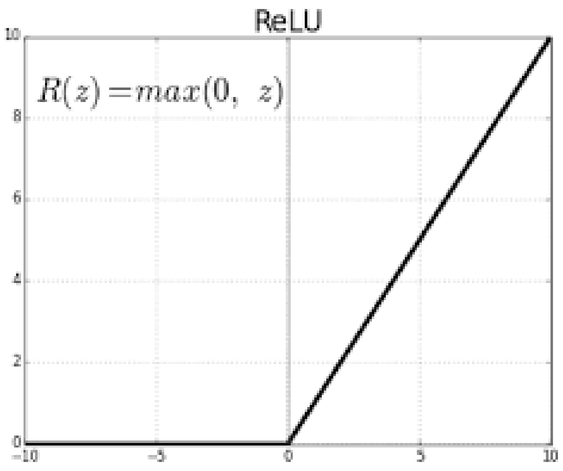
- **Tanh:** $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = a$ and $a' = 1 - a^2$
  It is a shifted sigmoid function, used to mean the data to 0. Usually it is prefered to sigmoid, except the last layer, where sigmoid is used.



- **ReLU:** $g(z) = \max(0, z) = a$

$$a' = \begin{cases} 1 & \text{for x} > 0 \\ 0 & \text{for x} < 0 \end{cases}$$

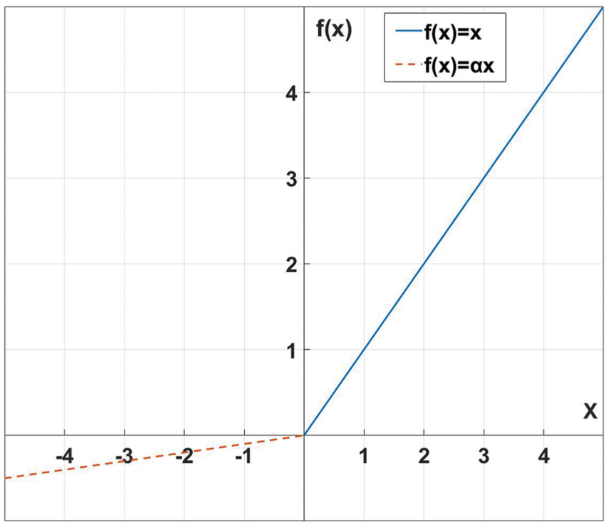
ReLU stands for Rectified Linear Unit. It is used universally, as it is considered to be better than tanh and sigmoid. This because as large z values, the gradient of sigmoid and tan h functions become almost zero. Hence, gradient descent takes a lot of time to converge. ReLU does not have this problem, as the gradient is 1 for $z > 0$.



- **Leaky ReLU:** $g(z) = \max(\alpha z, z)$

$$a' = \begin{cases} 1 & \text{for x} > 0 \\ \alpha & \text{for x} < 0 \end{cases}$$

Leaky ReLU is a slight variation of the ReLU function, where the function is not zero for $z < 0$, but a value αz, where α is a small positive number. This helps solve the Dying ReLU problem, where a neuron with a negative bias would never activate, as the gradient would be zero.



Dimensions

For a single training examples, the dimensions of various quantities are:

- $X^{[0]} = (n^{[0]}, 1)$
- $w^{[L]} = (n^{[L]}, n^{[L-1]})$
- $b^{[L]} = (n^{[L]}, 1)$
- $z^{[L]} = (n^{[L]}, 1)$
- $a^{[L]} = (n^{[L-1]}, 1)$

For m training examples, the dimensions are:

- $X^{[0]} = (n^{[0]}, m)$
- $W^{[L]} = (n^{[L]}, n^{[L-1]})$
- $B^{[L]} = (n^{[L]}, 1)$
- $Z^{[L]} = (n^{[L]}, m)$
- $A^{[L]} = (n^{[L-1]}, m)$

Note: Note that all the quantities are capitalised in the second case, to differentiate them from the single training example case.

6.2 Deep Neural Networks

Deep neural networks simply have more hidden layers than the normal neural networks, which is also known as shallow neural networks. Circuit theory says that the functions which you can compute with a deep L-layer NN, require exponentially more hidden units in a shallow L-layer NN. Hence, deep neural networks are preferred over shallow neural networks for complex tasks. To accurately predict the output, the weights and biases of the neural network need to be trained. This is done by the process of forward and backward propagation.

6.2.1 Initialization of parameters

The weight and bias matrices are initialised randomly with the appropriate shapes, as zero initialisation would symmetrise the whole NN. This would simplify the whole NN to just a linear model, which is not desired.

6.2.2 Forward Propagation

Forward propagation is when the input is passed through the NN, layer by layer, till it reaches the final or output layer. This step comprises of calculating the $A^{[L]}$ and $Z^{[L]}$ for each layer, and $\hat{Y}$, which is the output at the end of forward prop. These values are also stored in cache, as they would be required to calculate the gradients later in backward propagation. This would help to not waste time or computational power, calculating these terms again.

$$Z^{[L]} = W^{[L]}A^{[L-1]} + B^{[L]} \tag{32}$$

$$A^{[L]} = g(Z^{[L]}) \tag{33}$$

$$\hat{Y} = A^{[L]} \tag{34}$$

6.2.3 Cost function

If the label of the training example is Y, and the output of forward propagation is $\hat{Y}$, then the cost function J is defined as:

$$J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{Y^{(i)}}, Y^{(i)}) \tag{35}$$

Here, L can be the mean squared error or the cross-entropy loss. Mean squared error is usually used for regression tasks, while cross-entropy loss is used for classification tasks.

6.2.4 Backward Propagation

Backward propagation is then process of updating the weight and bias matrices of each layer, such that the cost function is minimised.
First the gradients of the cost function with respect to the weight and bias matrices need to be calculated. This is done by going backwards from the output layer towards the input layer and essentially applying chain rule to calculate the deriatives.
Let $dQ^{[L]} = \frac{\partial J}{\partial Q^{[L]}}$, where $Q^{[[L]]}$ is any quantity in the L^{th} layer. Then, the gradients can be calculated as:

$$dZ^{[L]} = dA^{[L]} * (g^{[L]})'(Z^{[L]}) \tag{36}$$

$$dW^{[L]} = \frac{1}{m} dZ^{[L]} * (A^{[L-1]})^T \tag{37}$$

$$dB^{[L]} = \frac{1}{m}\sum_{i=1}^{m} dZ^{[L](i)} \tag{38}$$

$$dA^{[L-1]} = (W^{[L]})^T * dZ^{[L]} \tag{39}$$

Note: Note that all the multiplication is element-wise multiplication.

This is where the cache we stored in the forward pass comes into play as we can see in the above equations.
Next, we update the weight and bias matrices as:

$$W^{[L]} = W^{[L]} - \alpha * dW^{[L]} \tag{40}$$

$$B^{[L]} = B^{[L]} - \alpha * dB^{[L]} \tag{41}$$

6.2.5 Prediction

Once the sufficient number of iterations of forward and backward passes are done, such that the cost function is minimised, the model can be tested on the dev set. The dev set is the set which is used to test the performance of the model, before testing it on the test set. The test input is passed through the NN and the output is compared with the actual label to get the accuracy of the model.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{42}$$

7 Overfitting and Underfitting

Now that, we have the base model of the neural network, we can work towards improving it, so that it can work better on sets other than the training set too.

As covered in section 4.2, overfitting is usually caused by high variance and underfitting is caused by high bias.
One of method of identifying if the model is overfitting or underfitting is to check its performance on the dev test and compare it with the train set.

Train set error	Dev set error	Model
Low(1%)	High(11%)	High variance
High(15%)	High(16%)	High bias
High(15%)	Very High(30%)	High bias and high variance
Low(0.5%)	Low(1%)	Low bias and low variance

As we can see from the table, high bias is based on the train set error, while high variance is based on the dev set error. Higher bias can be solved by a bigger network, or by training the model for a longer time or using a different NN architecture all together. High variance can be solved by using more data, regularisation, or by using a simpler model.
Lets go over some techniques to reduce overfitting.

7.1 Regularisation

Regularisation is a technique where the cost function of the model is modified such that it minimises the weights.
For example, in the case of logistic regression, the modified cost function would be:

$$J(w,b) = \frac{1}{m}\sum(L(\hat{y}^{(i)}, y^{(i)})) + \frac{\lambda}{2m}\|w\|_2^2 \tag{43}$$

where $\lambda > 0$ is the regularisation parameter and $\|w\|_2^2 = \sum_{j=1}^{n} w_j^2 = w^T w$, is called the Euclidean norm of the weight vector. This regularisation is also known as L2 regularisation.
Now, the model in the process of reducing J, would also reduce the regularisation term, $\frac{\lambda}{2m}\|w\|_2^2$, thus reducing each individual weight. One can also use L1 regularisation, where the regularisation term would be $\frac{\lambda}{2m}\|w\|_1 = \sum_{j=1}^{n} |w_j|$, though L2 regularisation is more commonly used.

Note: The bias regularisation term is usually ignored as it is the intercept term and does not actively contribute to the overfitting of the model. Morever, penalising the bias term can lead to shifting of the model, which is not desired.

For Neural Networks, the cost function would usually be:

$$J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{Y^{(i)}}, Y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left\|W^{[l]}\right\|_F^2 \tag{44}$$

Here, $\left\|W^{[l]}\right\|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(W_{ij}^{[l]})^2$, is the Frobenius norm of the weight matrix.
For back propagation, $dW^{[l]} = dW_{bp}^{[l]} + \frac{\lambda}{m}W^{[l]}$ is the modified gradient, where $dW_{bp}^{[l]}$ is the gradient calculated in the back propagation step.

After updating the weight matrix, the new matrix would be:

$$\begin{aligned} W^{[l]} &= W^{[l]} - \alpha dW^{[l]} \\ &= W^{[l]} - \alpha dW^{[l]}_{bp} - \frac{\lambda}{m}W^{[l]} \\ &= (1 - \frac{\alpha\lambda}{m})W^{[l]} - \alpha dW^{[l]}_{bp} \end{aligned} \tag{45}$$

As we can observe, the weight matrix is multiplied by a factor $(1 - \frac{\alpha\lambda}{m})$, which is less than 1, thus reducing the weight matrix at each iteration. This is known as **weight decay**.
Therefore λ is increased, $W^{[l]}$ would be reduced more. Since $Z^{[l]} = W^{[l]}A^{[l-1]} + B^{[l]}$, $Z^{[l]}$ would also be reduced. If we consider the tanh or sigmoid activation function, it is linear at low values of $Z^{[l]}$. A linear function is would make the model simpler and thus reduce the variance and overfitting.

7.2 Dropout

Dropout is another regularisation technique, where a fixed number or proportion of neurons, which is hyperparameter, are randomly dropped out or removed dring training. This ensures that the model does not rely more on any one neuron or feature, thus shrinking the weight matrix.
Inverted Dropout is a variation of dropout. Let `keep_prob` be the probability of keeping a neuron. Then, in inverted dropout, the remaining neurons after removing the dropout neurons are scaled down by a factor of `keep_prob`. This is done to ensure so that the expected value of the output remains same.
`keep_prob` is usually set to 0.8 for hidden layers and 1 for the input layer. However it can be set differently for different layers. If a layer has a bigger weight matrix, then a smaller `keep_prob` can be set to apply dropout.

Note: Dropout is not applied for test set

7.3 Data Augmentation

Data augmentation is another technique by which we can increase the training set by applying transformations to the existing training data.
Some common transformations for images are rotation, inversion, crop or distortion. This basically adds noise to the training data, which helps in generalising the model.

7.4 Early Stopping

While theoritically, the learning curve for dev test should decrease with time, however in practice, it does tend to increase after a certain number of iterations, due to overfitting. Early stopping is a technique where the training is stopped when the dev set error starts to increase.

7.5 Normalization

Normalization is a method where all the features in the input data are scaled to the same order. This helps in giving equal importance to all features and reducing overfitting.
For input $x = [x_1, x_2, \dots, x_n]$, the normalized input $x' =< x'_1, x'_2, \dots, x'_n >$ is given by:

$$x'_i = \frac{x_i - \mu_i}{\sigma_i} \tag{46}$$

where $\mu_i = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ and $\sigma_i = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_i)^2}$
After normalization, the variance of every feature becomes same and the mean becomes zero. This also helps in making the gradient descent faster.

7.6 Weight initialisation

Weight initialisation in a certain way can also help in reducing vanishing/exploding gradients and in turn overfitting.

Vanishing and Exploding gradients

Since, the gradients in back propagation are calculated by chain rule, and thus multiplied at every layer. In a very deep NN, the gradients can explode (become very large) or vanish(tend to zero). To prevent this, weights can be initialised randomly first and then scaled by a factor of $\sqrt{\frac{2}{n^{[l-1]}}}$, where $n^{[l-1]}$ is the number of neurons in the previous layer. This is known as **He initialisation**. It is used when the activation function used is ReLU.
For tanh activation function, the scaling factor is $\sqrt{\frac{1}{n^{[l-1]}}}$, which is known as **Xavier initialisation**.
He/Xavier initialisation usually performs better than random initialisation and zero initialisation.

8 Optimization Algorithms

8.1 Mini-Batch Gradient Descent

Mini-batch gradient descent is a variation of gradient descent, where the training set is divided into smaller batches. In normal gradient descent, the weight and bias matrices are updated after one epoch i.e. one pass through the training set. However, in mini-batch gradient descent, these matrices are updated after each mini-batch is passed through the NN, i.e. one epoch per mini-batch.
Mini-batch size is a hyperparameter. When mini-batch size is equal to m, it is known as batch gradient descent, which is the normal gradient descent which have read before and is used for smaller training sets. When mini-batch size is equal to 1, it is called stochastic gradient descent. As the mini-batch size decreases, the learning curve becomes more noisy, but the convergence is faster. However, if the size becomes too small, the model can infact be slower and might never converge.
Typical mini-batch sizes are usually powers of 2 like 64, 128, 256, 512.

8.2 Gradient Descent with Momentum

Gradient descent with momentum, is a technique where the updation depends not only the current weight and bias matrices, but also on the previous gradients.
The first step is to understand the concept of EMAs or Exponential Moving Averages.

Exponential Moving Averages

Let θ_t be the value of a term at t and $1 \geq \beta \geq 0$ be a constant. Then the EMA of θ_t for the period of p days is given by:

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t \tag{47}$$

Approximately, $p = \frac{1}{1-\beta}$, it is averaging over the last $\frac{1}{1-\beta}$ days.

Bias Correction: At the initial stage, i.e. close to $t = 0$, the EMA tends to be biased towards zero, even when the value of θ_t is high. To correct this bias, the EMA is scaled down by a factor of $\frac{1}{1-\beta^t}$. Hence, at small t, the EMAs are inflated negating the earlier effect. At large t, the denominator tends to 1, which is the normal EMA.

Momentum

On each iteration t, the dW and dB are calculated for each mini-batch. The weight and bias matrices are then updated as:

$$V_{dW} = \beta V_{dW} + (1-\beta)dW \tag{48}$$

$$V_{dB} = \beta V_{dB} + (1-\beta)dB \tag{49}$$

$$W = W - \alpha V_{dW} \tag{50}$$

$$B = B - \alpha V_{dB} \tag{51}$$

Momentum smoothens out the oscillations in GD by carrying the "momentum" from the previous gradient, thus making gradient descent faster.

8.3 RMSprop

RMSprop is another optimization algorithm, where the weight and bias matrices are updated differently than the normal GD.
At iteration t, the updation after calculating dW and db is as follows:

$$S_{dW} = \beta S_{dW} + (1-\beta)dW^2 \tag{52}$$

$$S_{dB} = \beta S_{dB} + (1-\beta)dB^2 \tag{53}$$

$$W = W - \alpha \frac{dW}{\sqrt{S_{dW}}} \tag{54}$$

$$B = B - \alpha \frac{dB}{\sqrt{S_{dB}}} \tag{55}$$

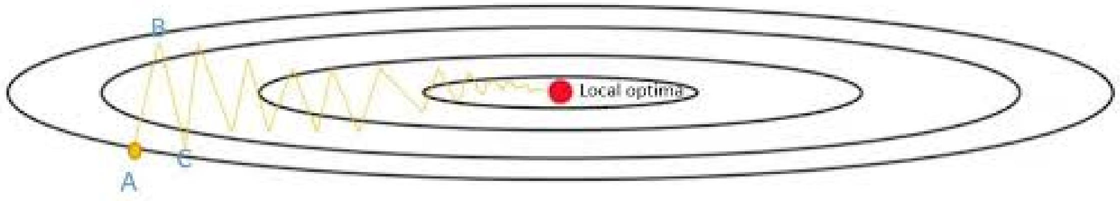


Figure 2: Oscillations in gradient descent

RMSprop essentially aims to restrict the oscillations in the vertical direction. This would increase the learnign rate and the algorithm cna take larger steps in the horizontal direction, thus converging much faster than normal gradient descent.

9 Updated POA

- **Week 1:** Got familiar with various python libraries like numpy, matplotlib and pandas
- **Week 2:** Got started with Andrew Ng's course on Supervised Machine Learning: Regression and Classification
- **Week 3:** Completed the course and got an overview of the mathematics behind linear, polynomial and logistic regression from Trevor Hastie
- **Week 4:** Read about classification algorithms like Naive Bayes and SVMs from Jake-Vandeplas, Trevor Hastie and Deisenroth, and completed the midterm report submission
- **Week 5:** Get started with the Advanced Learning Algorithms course by Andrew Ng and read about the remaining learning algorithms like Random Forests, K-Means, from JakeVandeplas.
- **Week 6:** Delve into deep learning and Read about RNN's and CNN's from Eli Stevens and Andrew Ng's course on Deep Learning
- **Week 7:** Check/Try out implementations of these deep learning models
- **Week 8:** Go over some of the mathematics behind these ML algorithms from Deisenroth and complete the final report submission. Also read about special concepts like Recommender Systems, Anomaly detection and bias-variance tradeoff.

10 References

[1] An introduction to Statistical Learning, Hastie et al.

[2] Mathematics for Machine Learning, Deinsenroth et al.

[3] Python Data Science Handbook, Jake VanderPlas `https://jakevdp.github.io/PythonDataScienceHandbook/`

[4] Supervised Machine Learning: Regression and Classification, Andrew Ng `https://www.coursera.org/specializations/machine-learning-introduction`

[5] Statquest with Josh Starmer: `https://www.youtube.com/@statquest`

[6] Understanding Machine Learning: From Theory to Algorithms, Shai Shalev-Shwartz and Shai Ben-David