

Unit-I

Syllabus

Historical trends in deep learning – Machine Learning basics, Learning algorithms – Supervised and Unsupervised Training, Linear Algebra for machine learning, Testing - Cross Validation, Dimensionality Reduction, Over fitting /Under Fitting, Hyper parameters and validation sets Estimators – Bias – Variance, Loss Function—Regularization, Biological Neuron – Idea of Computational units, McCulloch-Pitts units and Thresholding logic, Linear Perceptron, Perceptron Learning Algorithm, Convergence theorem for Perceptron Learning Algorithm, Linear Separability Multilayer perceptron –The first example of network with Keras code, Backpropagation

1. Machine Learning Basics

Definition: - “Machine learning is a branch of artificial intelligence that seeks to build computer systems that can learn from data without human intervention”.

These powerful techniques rely on the creation of sophisticated analytical models that are “trained” to recognize patterns within a specific dataset before being unleashed to apply these patterns to more and more data, steadily improving performance without further guidance.

For example, machine learning is making increasingly accurate image recognition algorithms possible. Human programmers provide a relatively small set of images that are labeled as “cars” or “not cars,” for instance, and then expose the algorithms to vastly larger numbers of images to learn from. While the iterative algorithms typically used in machine learning aren’t new, the power of today’s computing systems have enabled this method of data analysis to become more effective more rapidly than ever.

1.1 Supervised(Predictive) Learning-the goal is to learn a mapping from inputs \mathbf{x} to outputs y , given a labeled set of input-output pairs $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$

Where $i=1$ to N

Here D is called the **training set**, and N is the number of training examples.

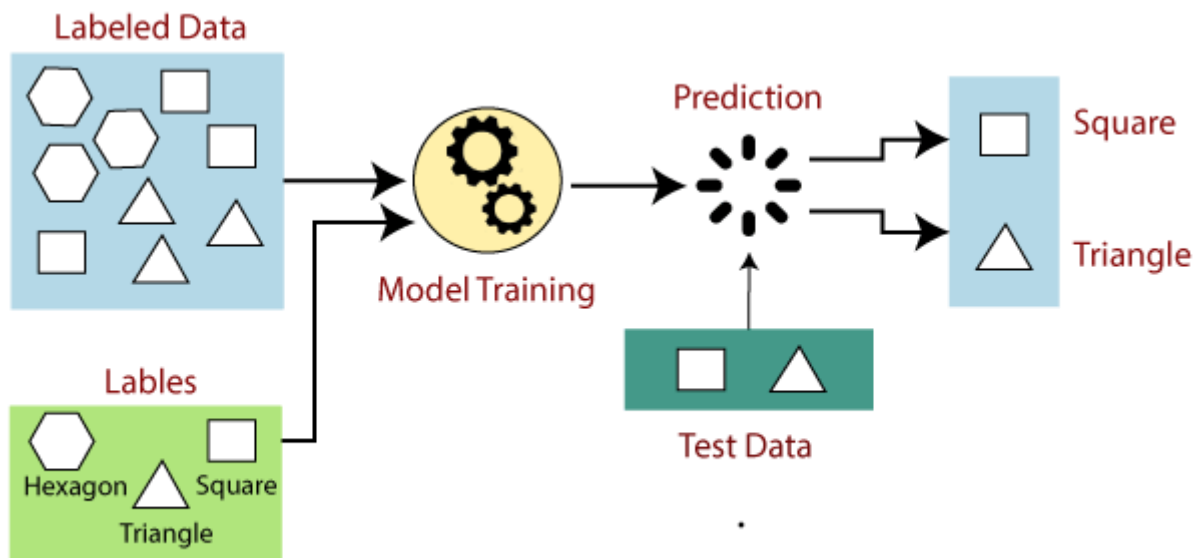
How it works

Figure: - flow of supervised learning

- Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.
- If the given shape has four sides, and all the sides are equal, then it will be labelled as a Square.
- If the given shape has three sides, then it will be labelled as a triangle.
- If the given shape has six equal sides then it will be labelled as hexagon.
- Now, after training, we test our model using the test set, and the task of the model is to identify the shape.
- The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

Example of supervised Learning

- you get bunch of photos with information what is on them and you train a model to recognize new photos
- predicting stock market price
- an email is spam or not
- predicting house/property price
- a patient has disease or not
- Face detection and recognition

- Image classification and handwriting recognition
- Predict the age of a viewer watching a given video on YouTube.
- Predict the temperature at any location inside a building using weather data, time, door

Supervised learning algorithms

- Linear regression.
- Logistic regression.
- Decision tree.
- SVM algorithm.
- Naive Bayes algorithm.
- KNN algorithm.
- K-means.

Random forest algorithm

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as fraud detection, spam filtering, etc.

1.2 Unsupervised(Descriptive) Learning

Unsupervised(Descriptive) Learning- the goal is to find “interesting patterns” in the data. This is sometimes called knowledge discovery. $D = \{\mathbf{x}_i\}N$ where $N=1$, and This is a much less well-defined problem, since we are not told what kinds of patterns to look for, and there is no obvious error metric to us your mother says “that’s a dog”, but that’s very little information.

Example:-

customers by purchasing behavior

Why use unsupervised learning

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.

- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of unsupervised learning

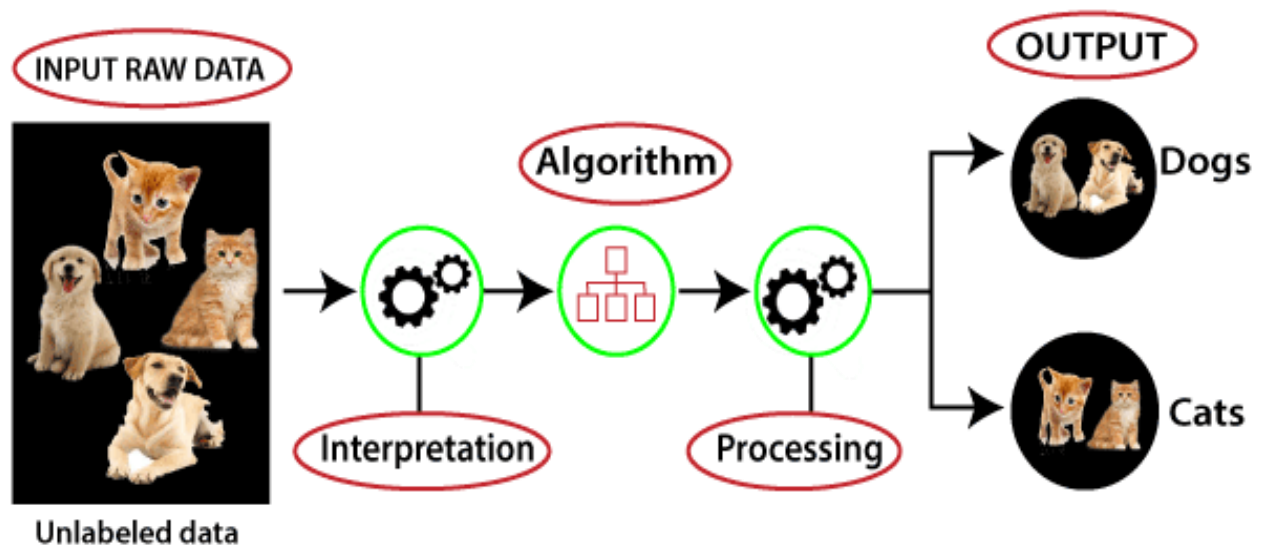


Figure: - flow of unsupervised learning

Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects

Unsupervised Learning algorithm

- Hierarchical clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis

Advantages

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to

labeled data.

Disadvantages

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.

The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance

2. What is Linear Algebra

Linear Algebra is a branch of mathematics that lets you concisely describe coordinates and interactions of planes in higher dimensions and perform operations on them.

Think of it as an extension of algebra (dealing with unknowns) into an arbitrary number of dimensions. Linear Algebra is about working on linear systems of equations (linear regression is an example: $y = Ax$). Rather than working with scalars, we start working with matrices and vectors (vectors are really just a special type of matrix).

Vectors A column vector is a list of numbers stacked on top of each other, e.g. $a = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$

A row vector is a list of numbers written one after the other, e.g. $b = (2, 1, 3)$ In both cases, the list is ordered, i.e. $(2, 1, 3) \neq (1, 2, 3)$.

We make the following convention:

- In what follows all vectors are column vectors unless otherwise stated.
- However, writing column vectors takes up more space than row vectors.

Therefore, we shall frequently write vectors as row vectors, but with the understanding that it really is a column vector.

A general n -vector has the form

3. Cross-Validation

Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

The three steps involved in cross-validation are as follows:

1. Reserve some portion of sample data-set.
2. Using the rest data-set train the model.
3. Test the model using the reserve portion of the data-set.

Methods of Cross Validation

Validation

In this method, we perform training on the 50% of the given data-set and rest 50% is used for the testing purpose. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e higher bias.

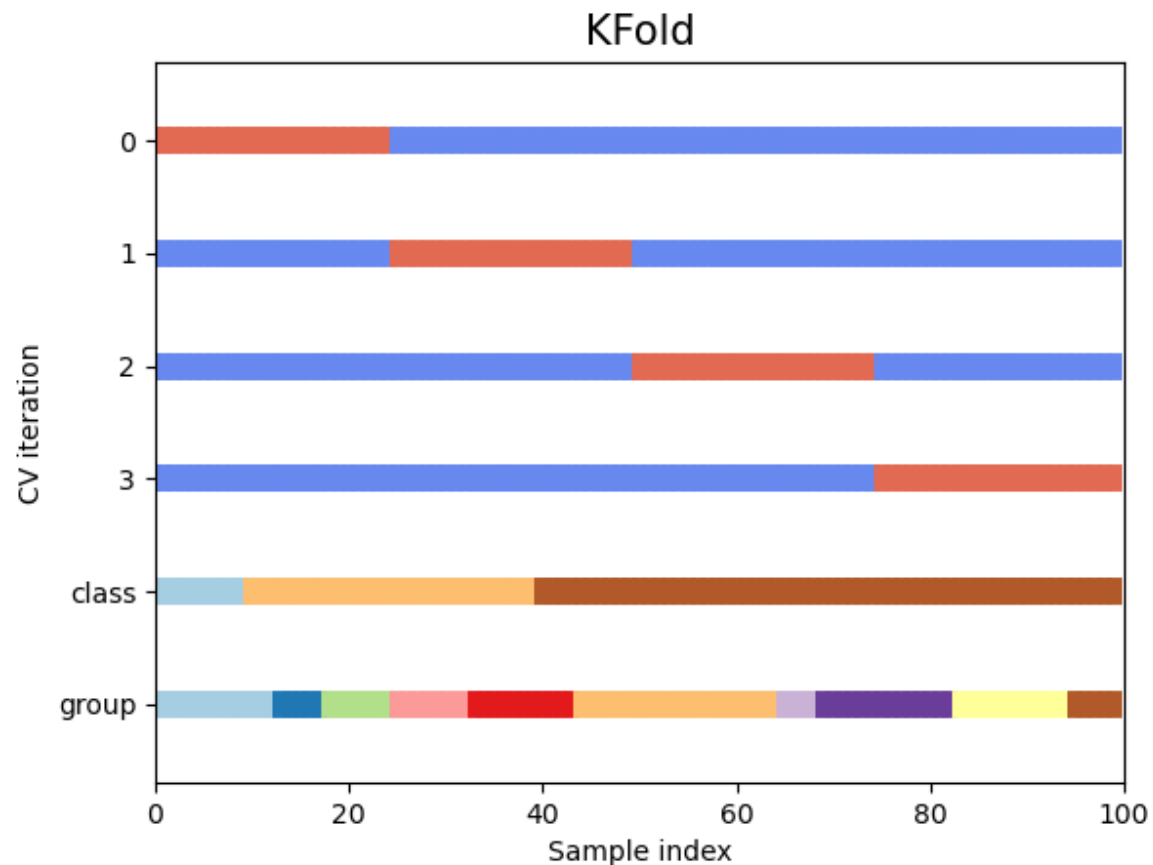
3.1 LOOCV (Leave One Out Cross Validation)

In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point. It has some advantages as well as disadvantages

An advantage of using this method is that we make use of all data points and hence it is low bias. The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points' times.

3.2 K-Fold Cross Validation

In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.



Advantages of train/test split:

1. This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times.
2. Simpler to examine the detailed results of the testing process.

Advantages of cross-validation:

1. More accurate estimate of out-of-sample accuracy.
2. More “efficient” use of data as every observation is used for both training and testing.

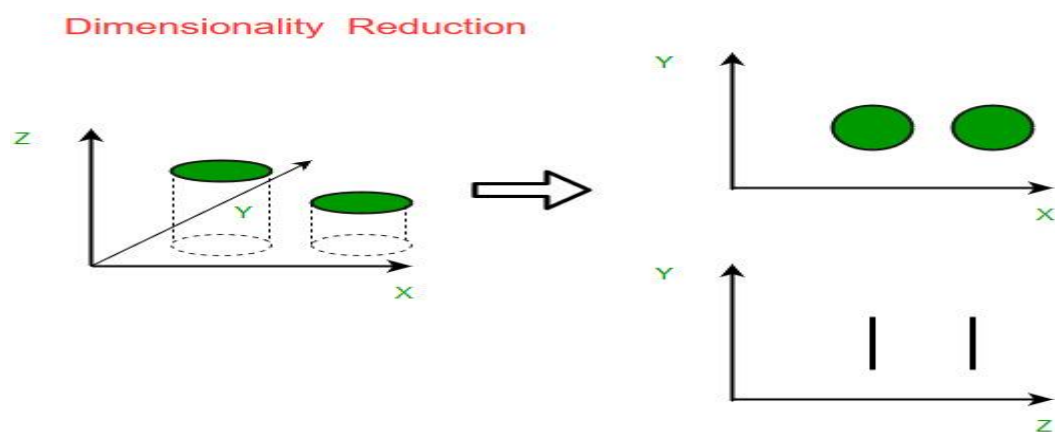
4. What is Dimensionality Reduction?

Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it.

Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play.

. Example: - An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 1-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



Components of Dimensionality Reduction

There are two components of dimensionality reduction:

- **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
 1. Filter
 2. Wrapper
 3. Embedded
- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Methods of Dimensionality Reduction

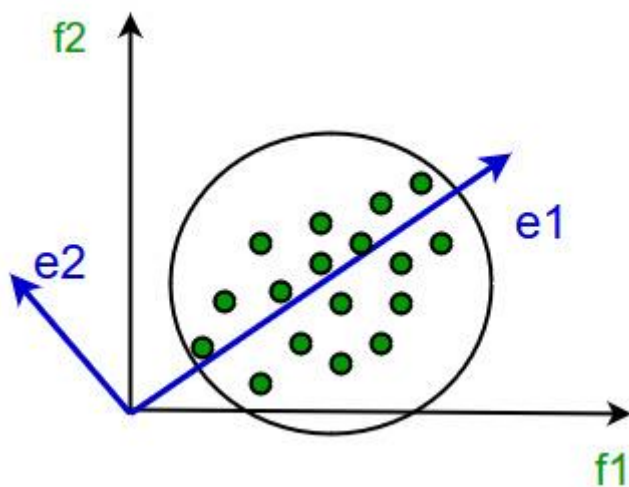
The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

Principal Component Analysis

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data. Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.

- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.

5. validation sets Estimators – Bias – Variance

Let us consider that we are designing a machine learning model. A model is said to be a good machine learning model if it generalizes any new input data from the problem domain in a proper way. This helps us to make predictions in the future data, that the data model has never seen. Now, suppose we want to check how well our machine learning model learns and generalizes to the new data. For that, we have overfitting and under fitting, which are majorly responsible for the poor performances of the machine learning algorithms.

- **Bias:** Assumptions made by a model to make a function easier to learn.
- **Variance:** If you train your data on training data and obtain a very low error, upon changing the data and then training the same previous model you experience a high error, this is variance.

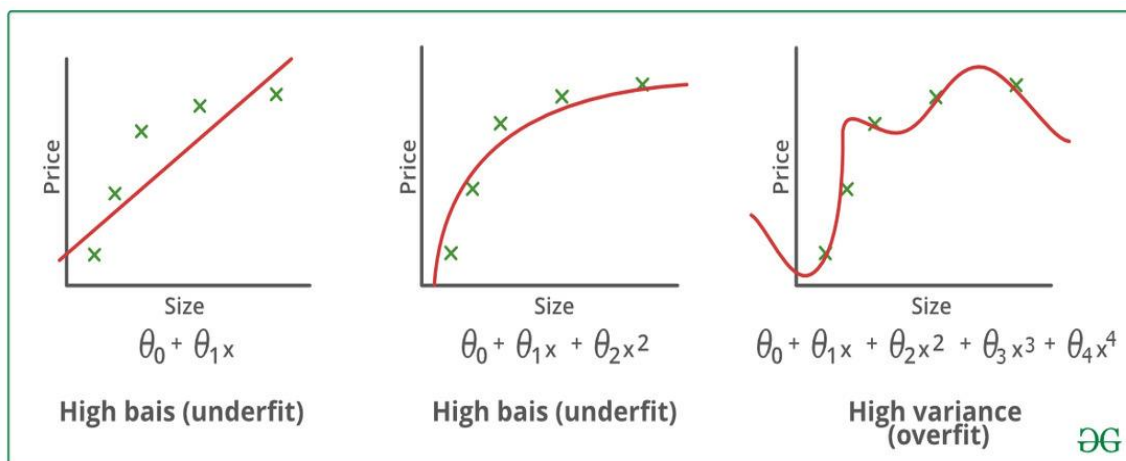


Figure – High bias & High variance

Under-fitting:

A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data.

Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough.

It usually happens when we have fewer data to build an accurate model and also when we try to build a linear model with fewer non-linear data. In such cases, the rules of the machine learning model are too easy and flexible to be applied on such minimal data and therefore the model will probably make a lot of wrong predictions.

Under-fitting can be avoided by using more data and also reducing the features by feature selection.

Under-fitting – High bias and low variance

Techniques to reduce under-fitting:

1. Increase model complexity
2. Increase the number of features, performing feature engineering
3. Remove noise from the data.
4. Increase the number of epochs or increase the duration of training to get better results.

Overfitting:

A statistical model is said to be over fitted when we train it with a lot of data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

Overfitting – High variance and low bias

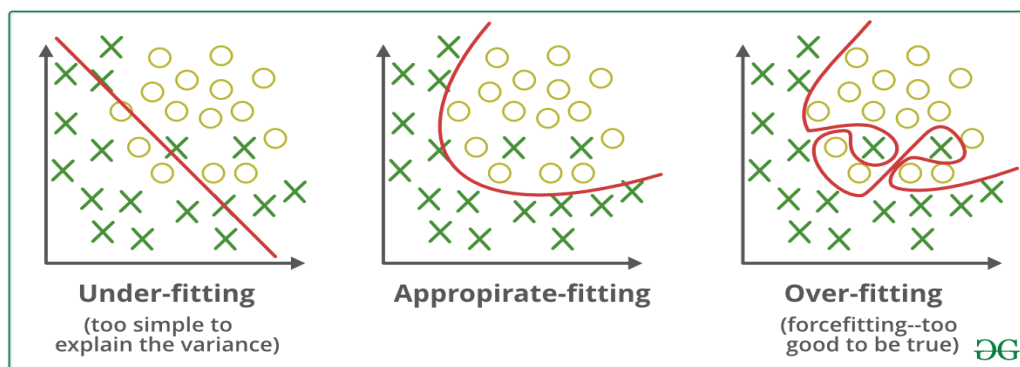


Figure:- under fitting and overfitting

Techniques to reduce overfitting:

1. Increase training data.
2. Reduce model complexity.

3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization and Lasso Regularization
5. Use dropout for neural networks to tackle overfitting.

Good Fit in a Statistical Model:

Ideally, the case when the model makes the predictions with 0 error, is said to have a *good fit* on the data. This situation is achievable at a spot between overfitting and under fitting. In order to understand it, we will have to look at the performance of our model with the passage of time, while it is learning from training dataset.

With the passage of time, our model will keep on learning and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details. Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point, the model is said to have good skills on training datasets as well as our unseen testing dataset.

6. Hyper parameters

A hyper parameter is a parameter that is set before the learning process begins. These parameters are tunable and can directly affect how well a model trains. Some examples of hyper parameters in machine learning:

1. Learning Rate
2. Number of Epochs
3. Momentum
4. Regularization constant
5. Number of branches in a decision tree
6. Number of clusters in a clustering algorithm (like k-means)

Optimizing Hyper parameters

Hyper parameters can have a direct impact on the training of machine learning algorithms. Thus, in order to achieve maximal performance, it is important to understand how to optimize them. Here are some common strategies for optimizing hyper parameters:

1. Grid Search: Search a set of manually predefined hyper parameters for the best performing hyper parameter. Use that value. (This is the traditional method)

2. Random Search: Similar to grid search, but replaces the exhaustive search with random search. This can outperform grid search when only a small number of hyper parameters are needed to actually optimize the algorithm.

3. Bayesian Optimization: Builds a probabilistic model of the function mapping from hyper parameter values to the target evaluated on a validation set.

4. Gradient-Based Optimization: Compute gradient using hyperparameters and then optimize hyper parameters using gradient descent.

5. Evolutionary Optimization

: Uses evolutionary algorithms (e.g. genetic functions) to search the space of possible hyper parameters.

7. Loss Function

Loss functions measure how far an estimated value is from its true value. A loss function maps decisions to their associated costs. Loss functions are not fixed, they change depending on the task in hand and the goal to be met.

7.1 Mean Absolute Error (MAE)

Mean Absolute Error (also called L1 loss) is one of the most simple yet robust loss functions used for regression models.

Regression problems may have variables that are not strictly Gaussian in nature due to the presence of outliers (values that are very different from the rest of the data). Mean Absolute Error would be an ideal option in such cases because it does not take into account the direction of the outliers (unrealistically high positive or negative values).

As the name suggests, MAE takes the average sum of the absolute differences between the actual and the predicted values. For a data point x_i and its predicted value y_i , n being the total number of data points in the dataset, the mean absolute error is defined as:

$$\mathbf{MAE} = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$$

7.2 Mean Squared Error (MSE)

Mean Squared Error (also called L2 loss) is almost every data scientist's preference when it comes to loss functions for regression. This is because most variables can be modeled into a Gaussian distribution.

Mean Squared Error is the average of the squared differences between the actual and the predicted values. For a data point Y_i and its predicted value \hat{Y}_i , where n is the total number of data points in the dataset, the mean squared error is defined as:

$$\mathbf{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

7.3 Mean Bias Error (MBE)

Mean Bias Error is used to calculate the average bias in the model. Bias, in a nutshell, is overestimating or underestimating a parameter. Corrective measures can be taken to reduce the bias post-evaluating the model using MBE.

Mean Bias Error takes the actual difference between the target and the predicted value, and not the absolute difference. One has to be cautious as the positive and the negative errors could cancel each other out, which is why it is one of the lesser-used loss functions.

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

Where y_i is the true value, \hat{y}_i is the predicted value and 'n' is the total number of data points in the dataset.

7.4 Mean Squared Logarithmic Error (MSLE)

Sometimes, one may not want to penalize the model too much for predicting unscaled quantities directly. Relaxing the penalty on huge differences can be done with the help of Mean Squared Logarithmic Error.

Calculating the Mean Squared Logarithmic Error is the same as Mean Squared Error, except the natural logarithm of the predicted values is used rather than the actual values.

$$MSLE = \frac{1}{n} \sum_{i=1}^n (\log(Y_i) - \log(\hat{Y}_i))^2$$

Where y_i is the true value, \hat{y}_i is the predicted value and 'n' is the total number of data points in the dataset.

7.5 Huber Loss

A comparison between L1 and L2 loss yields the following results:

1. L1 loss is more robust than its counterpart.

On taking a closer look at the formulas, one can observe that if the difference between the predicted and the actual value is high, L2 loss magnifies the effect when compared to L1. Since L2 succumbs to outliers, L1 loss function is the more robust loss function.

2. L1 loss is less stable than L2 loss.

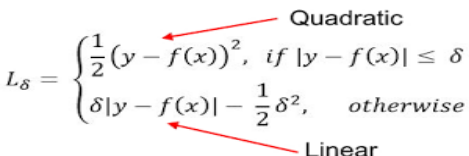
Since L1 loss deals with the difference in distances, a small horizontal change can lead to the regression line jumping a large amount. Such an effect taking place across multiple iterations would lead to a significant change in the slope between iterations.

On the other hand, MSE ensures the regression line moves lightly for a small adjustment in the data point.

Huber Loss combines the robustness of L1 with the stability of L2, essentially the best of L1 and L2 losses. For huge errors, it is linear and for small errors, it is quadratic in nature.

Huber Loss is characterized by the parameter delta (δ). For a prediction $f(x)$ of the data point y , with the characterizing parameter δ , Huber Loss is formulated as:

$$L_{\delta} = \begin{cases} \frac{1}{2}(y - f(x))^2, & \text{if } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$



Loss functions for classification

Classification problems involve predicting a discrete class output. It involves dividing the dataset into different and unique classes based on different parameters so that a new and unseen record can be put into one of the classes.

A mail can be classified as a spam or not a spam and a person's dietary preferences can be put in one of three categories - vegetarian, non-vegetarian and vegan. Let's take a look at loss functions that can be used for classification problems.

Binary Cross Entropy Loss

This is the most common loss function used for classification problems that have two classes. The word “entropy”, seemingly out-of-place, has a statistical interpretation.

Entropy is the measure of randomness in the information being processed, and cross entropy is a measure of the difference of the randomness between two random variables.

If the divergence of the predicted probability from the actual label increases, the cross-entropy loss increases. Going by this, predicting a probability of .011 when the actual observation label is 1 would result in a high loss value. In an ideal situation, a “perfect” model would have a log loss of 0. Looking at the loss function would make things even clearer -

$$J = - \sum_{i=1}^N y_i \log(h_{\theta}(x_i)) + (1 - y_i) \log(1 - h_{\theta}(x_i))$$

Where y_i is the true label and $h_{\theta}(x_i)$ is the predicted value post hypothesis.

Since binary classification means the classes take either 0 or 1, if $y_i = 0$, that term ceases to exist and if $y_i = 1$, the $(1-y_i)$ term becomes 0.

Categorical Cross Entropy Loss

Categorical Cross Entropy loss is essentially Binary Cross Entropy Loss expanded to multiple classes. One requirement when categorical cross entropy loss function is used is that the labels should be one-hot encoded.

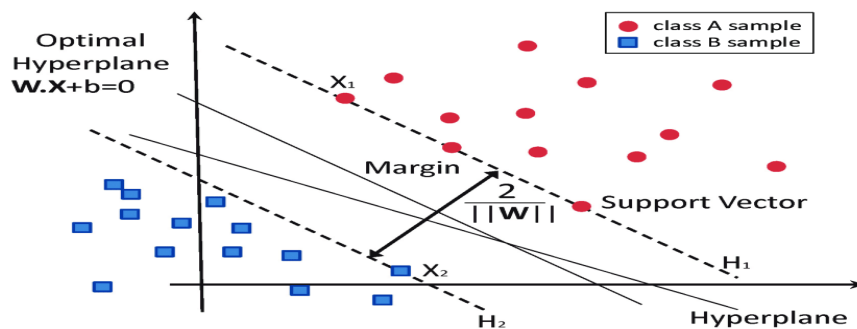
This way, only one element will be non-zero as other elements in the vector would be multiplied by zero. This property is extended to an activation function called softmax,

Hinge Loss

Another commonly used loss function for classification is the hinge loss. Hinge loss is primarily developed for support vector machines for calculating the maximum margin from the hyperplane to the classes.

Loss functions penalize wrong predictions and does not do so for the right predictions. So, the score of the target label should be greater than the sum of all the incorrect labels by a margin of (at the least) one.

This margin is the maximum margin from the hyperplane to the data points, which is why hinge loss is preferred for SVMs. The following image clears the air on what a hyperplane and maximum margin is:



The mathematical formulation of hinge loss is as follows:

$$SVM Loss = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Where s_j is the true value and s_{y_i} is the predicted value.

Hinge Loss is also extended to Squared Hinge Loss Error and Categorical Hinge Loss Error.

8.REGULARIZATION

Regularization are techniques used to reduce the error by fitting function appropriately on the given training set and avoid overfitting

Consider the training dataset comprising of independent variables $X=(x_1, x_2, \dots, x_n)$ and the corresponding target variables $t=(t_1, t_2, \dots, t_n)$. X are random variables lying uniformly between $[0,1]$. The target dataset 't' is obtained by substituting the value of X into the function $\sin(2\pi x)$ and then adding some Gaussian noise into it.

Now, our goal is to find patterns in this underlying dataset and generalize it to predict the corresponding target value for some new values of 'x'. The problem here is, our target dataset is inflicted with some random noise. So, it will be difficult to find the underlying function $\sin(2\pi x)$ in the training data. So, how do we solve it?

Let's try fitting a polynomial on the given data.

$$Y(x, w) = w_0 + w_1x + w_2x^2 + \dots + w_nx^n. \quad - (Eq 1.1)$$

It should be noted that the given polynomial function is a non-linear function of 'x' but a linear function of 'w'. We train our data on this function to determine the values of w that will make the function to minimize the error in predicting target values.

The error function used in this case is mean squared error.

$$E(w) = \left(\frac{1}{2}\right) * \sum (y(w, x) - t)^2 \quad \text{- (Eq 1.2)}$$

In order to minimize the error, calculus is used. The derivative of E(w) is equated with 0 to get the value of w which will result at the minimum value of error function. E(w) is a quadratic equation, but it's derivative will be a linear equation and hence will result in only a single value of w. Let that be denoted by w*.

So now, we will get the correct value of w but the issue is what degree of polynomial (given in Eq 1.1) to choose? All degree of polynomials can be used to fit on the training data, but how to decide the best choice with minimum complexity?

9. Biological Neurons

It is an unusual-looking cell mostly found in animal cerebral cortexes (e.g., your brain), composed of a cell *body* containing the nucleus and most of the cell's complex components, and many branching extensions called dendrites, plus one very long extension called the axon. The axon's length may be just a few times longer than the cell body, or up to tens of thousands of times longer. Near its extremity the axon splits off into many branches called telodendria, and at the tip of these branches are minuscule structures called synaptic terminals (or simply synapses), which are connected to the dendrites (or directly to the cell body) of other neurons. Biological neurons receive short electrical impulses called signals from other neurons via these synapses. When a neuron receives a sufficient number of signals from other neurons within a few milliseconds, it fires its own signals.

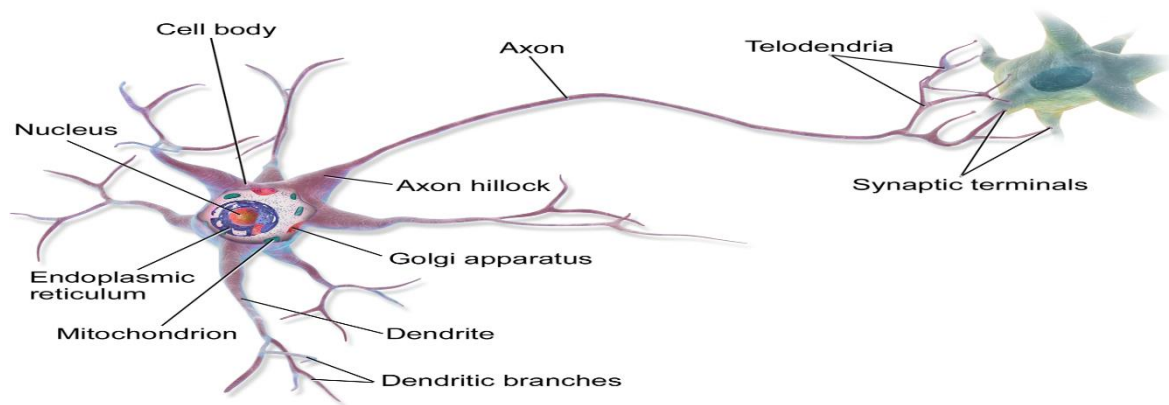


Figure – Biological Neuron

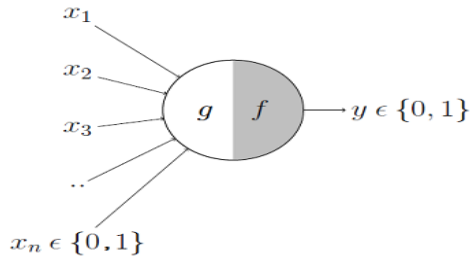
Thus, individual biological neurons seem to behave in a rather simple way, but they are organized in a vast network of billions of neurons, each neuron typically connected to thousands of other neurons. Highly complex computations can be performed by a vast network of fairly simple neurons, much like a complex anthill can emerge from the combined efforts of simple ants. The architecture of biological neural networks (BNN) it seems that neurons are often organized in consecutive layers, as shown in Figure



Figure – multiple layers in biological neural network

10. McCulloch-Pitts Neuron

The first computational model of a neuron was proposed by Warren McCulloch (neuroscientist) and Walter Pitts (logician) in 1943.



It may be divided into 2 parts. The first part, g takes an input (ahem dendrite ahem), performs an aggregation and based on the aggregated value the second part, f makes a decision.

Lets suppose that I want to predict my own decision, whether to watch a random football game or not on TV. The inputs are all boolean i.e., $\{0,1\}$ and my output variable is also boolean $\{0: \text{Will watch it}, 1: \text{Won't watch it}\}$.

1. x_1 could be is Premier League On (I like Premier League more)
2. x_2 could be is It A Friendly Game (I tend to care less about the friendlies)
3. x_3 could be is Not Home (Can't watch it when I'm running errands. Can I?)
4. x_4 could be is Man United Playing (I am a big Man United fan. GGMU!) and so on.

These inputs can either be *excitatory* or *inhibitory*. Inhibitory inputs are those that have maximum effect on the decision making irrespective of other inputs i.e., if x_3 is 1 (not home) then my output will always be 0 i.e., the neuron will never fire, so x_3 is an inhibitory input. Excitatory inputs are NOT the ones that will make the neuron fire on their own but they might fire it when combined together. Formally, this is what is going on:

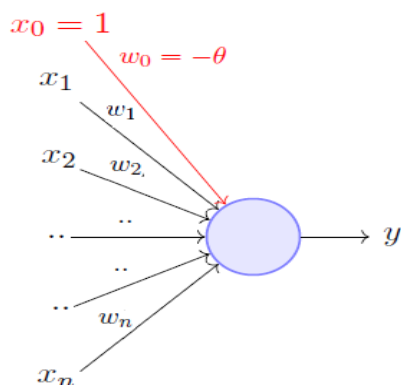
$$g(x_1, x_2, x_3, \dots, x_n) = g(\mathbf{x}) = \sum_{i=1}^n x_i$$

$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{if } g(\mathbf{x}) \geq \theta \\ 0 & \text{if } g(\mathbf{x}) < \theta \end{cases}$$

We can see that $g(\mathbf{x})$ is just doing a sum of the inputs — a simple aggregation. And *theta* here is called thresholding parameter. For example, if I always watch the game when the sum turns out to be 2 or more, the *theta* is 2 here. This is called the Thresholding Logic.

11. Perceptron

The perceptron model is a more general computational model than McCulloch-Pitts neuron. It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0. Rewriting the threshold as shown above and making it a constant input with a variable weight, we would end up with something like the following



A more accepted convention,

$$y = 1 \quad \text{if} \quad \sum_{i=0}^n w_i * x_i \geq 0$$

$$= 0 \quad \text{if} \quad \sum_{i=0}^n w_i * x_i < 0$$

where, $x_0 = 1$ and $w_0 = -\theta$

:

A single perceptron can only be used to implement **linearly separable** functions. It takes both real and boolean inputs and associates a set of **weights** to them, along with a **bias** (the threshold thing I mentioned above). We learn the weights, we get the function. Let's use a perceptron to learn an OR function.

12. Perceptron Learning Algorithm

Our goal is to find the \mathbf{w} vector that can perfectly classify positive inputs and negative inputs in our data. I will get straight to the algorithm. Here goes:

Algorithm: Perceptron Learning Algorithm

```
P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;
while !convergence do
    Pick random  $\mathbf{x} \in P \cup N$  ;
    if  $\mathbf{x} \in P$  and  $\mathbf{w} \cdot \mathbf{x} < 0$  then
        |  $\mathbf{w} = \mathbf{w} + \mathbf{x}$  ;
    end
    if  $\mathbf{x} \in N$  and  $\mathbf{w} \cdot \mathbf{x} \geq 0$  then
        |  $\mathbf{w} = \mathbf{w} - \mathbf{x}$  ;
    end
end
//the algorithm converges when all the
inputs are classified correctly
```

We initialize **w** with some random vector. We then iterate over all the examples in the data, ($P \cup N$) both positive and negative examples. Now if an input \mathbf{x} belongs to P , ideally what should the dot product $\mathbf{w} \cdot \mathbf{x}$ be? I'd say greater than or equal to 0 because that's the only thing what our perceptron wants at the end of the day so let's give it that. And if \mathbf{x} belongs to N , the dot product MUST be less than 0. So if you look at the if conditions in the while loop:

```
while !convergence do
    Pick random  $\mathbf{x} \in P \cup N$  ;
    if  $\mathbf{x} \in P$  and  $\mathbf{w} \cdot \mathbf{x} < 0$  then
         $\mathbf{w} = \mathbf{w} + \mathbf{x}$  ;
    end
    if  $\mathbf{x} \in N$  and  $\mathbf{w} \cdot \mathbf{x} \geq 0$  then
         $\mathbf{w} = \mathbf{w} - \mathbf{x}$  ;
    end
end
```

Case 1: When \mathbf{x} belongs to P and its dot product $\mathbf{w} \cdot \mathbf{x} < 0$

Case 2: When \mathbf{x} belongs to N and its dot product $\mathbf{w} \cdot \mathbf{x} \geq 0$

Only for these cases, we are updating our randomly initialized **w**. Otherwise, we don't touch **w** at all because Case 1 and Case 2 are violating the very rule of a perceptron. So we are adding \mathbf{x} to **w** (ahem vector addition ahem) in Case 1 and subtracting \mathbf{x} from **w** in Case 2.

Multilayer perceptron

A **multilayer perceptron (MLP)** is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to mean *any* feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons (with threshold activation); see § Terminology. Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable

13. The First example of Neural Network with Keras code

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning **models**.

a) Steps: -

1. Load Data.
2. Define Keras Model.
3. Compile Keras Model.
4. Fit Keras Model.
5. Evaluate Keras Model.
6. Tie It All Together.
7. Make Predictions

b) Requirements

1. You have Python 2 or 3 installed and configured.
2. You have SciPy (including NumPy) installed and configured.
3. You have Keras and a backend (Theano or TensorFlow) installed and configured.
- 4.

Load Data

The first step is to define the functions and classes

We will use the NumPy library to load our dataset and we will use two classes from the Keras library to define our model.

first neural network with keras

The imports required are listed below.

```
from numpy import loadtxt
```

```
from keras.models import Sequential
```

```
from keras.layers import Dense
```

load the dataset

```
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
```

split into input (X) and output (y) variables

```
X = dataset[:,0:8]
```

```
y = dataset[:,8]
```

define the keras model

```
model = Sequential()
```

```
model.add(Dense(12, input_dim=8, activation='relu'))
```

```
model.add(Dense(8, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
```

compile the keras model

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

fit the keras model on the dataset

```
model.fit(X, y, epochs=150, batch_size=10)
```

evaluate the keras model

```
_, accuracy = model.evaluate(X, y)
```

```
print('Accuracy: %.2f' % (accuracy*100))
```

14. Backpropagation

Backpropagation is an algorithm used in artificial intelligence (AI) to fine-tune mathematical weight functions and improve the accuracy of an artificial neural network's outputs. A neural network can be thought of as a group of connected input/output (I/O) nodes. The level of accuracy each node produces is expressed as a loss function (error rate). Backpropagation calculates the mathematical gradient of a loss function with respect to the other weights in the neural network.

The calculations are then used to give artificial network nodes with high error rates less weight than nodes with lower error rates.

Backpropagation uses a methodology called chain rule to improve outputs. Basically, after each forward pass through a network, the algorithm performs a backward pass to adjust the model's weights.

An important goal of backpropagation is to give data scientists insight into how changing a weight function will change loss functions and the overall behaviour of the neural network. The term is sometimes used as a synonym for "error correction."