***SVM: (kernel tricks)***

* It is useful for both linearly Separable (hard margin) and Non-linearly Separable (soft margin) data.
* It is effective in high dimensional spaces.

Disadvantages:

* Picking the right kernel and parameters can be computationally intensive.
* It also does not perform very well, when the data set has more noise i.e., target classes are overlapping.

***KNN: (similar neighbors for the new data point.)***

Non-parametric, so makes no assumption about the underlying data pattern.

***Random Forest: (***t operates by constructing a multitude of decision trees at training time and outputting the class that is the **mode** of the **classes (classification)** or **mean prediction (regression)** of the individual trees)

Advantage:

* It has an **effective method for estimating missing data** and maintains accuracy when a large proportion of the data are missing.
* It is one of the most accurate learning algorithms available. For many data sets, it produces a **highly accurate classifier**.

Disadvantages:

For data including categorical variables with different number of levels,**random forests are biased in favor of those attributes with more levels.**

***Decision Tree:***

Advantage:

* Easy to understand and interpret, perfect for visual representation.
* Can work with numerical and categorical features.

Disadvantage:

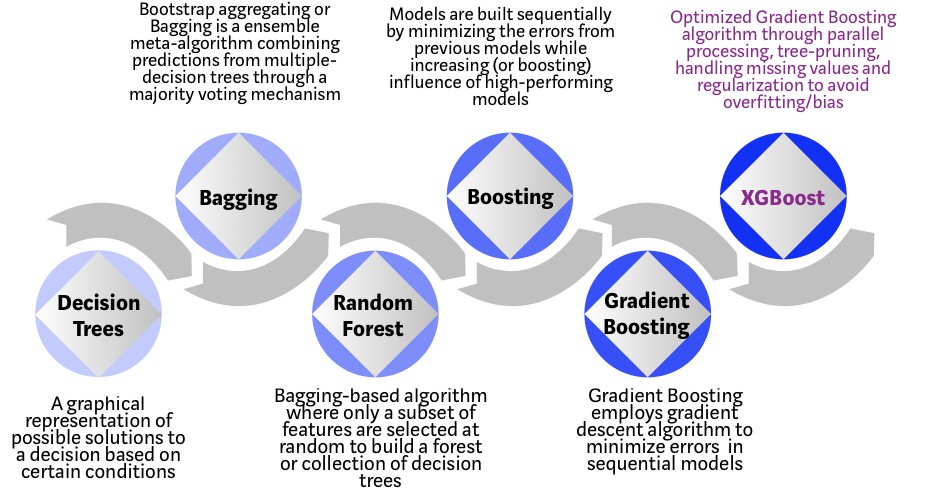
main drawback of Decision Tree is that it generally leads to overfitting of the data.

***Naïve Bayes:*** (Bayes Theorem helps us to find the probability of a hypothesis given our prior knowledge)

It can also be trained on small dataset.

this algorithm makes an assumption as all the variables in the dataset is “Naive” i.e., not correlated to each other.

***XGboots***: [XGBoost](https://xgboost.ai/) is a decision-tree-based ensemble Machine Learning algorithm that uses a [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting) framework. when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.



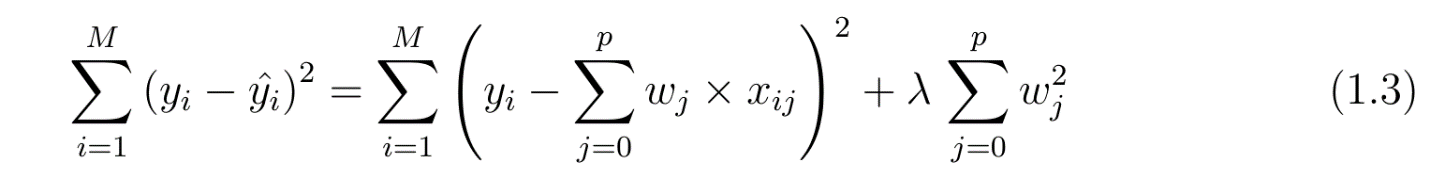
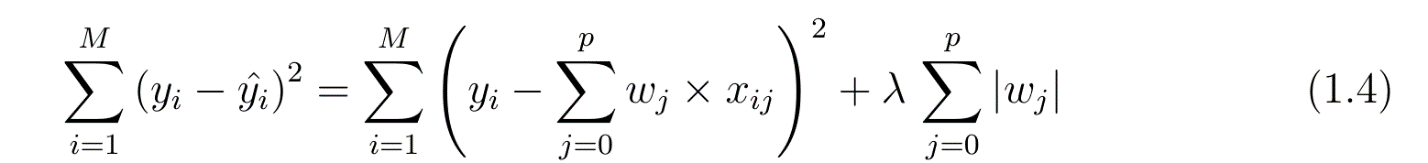
**R2 vs Adjusted R2:**

R-squared or R2 explains the **degree to which your input variables explain the variation of your output / predicted variable**. So, if R-square is 0.8, it means 80% of the variation in the output variable is explained by the input variables. So, in simple terms, higher the R squared, the more variation is explained by your input variables and hence better is your model.

However, the problem with R-squared is that it will either stay the same or increase with addition of more variables, even if they do not have any relationship with the output variables. This is where “Adjusted R square” comes to help. **Adjusted R-square penalizes you for adding variables which do not improve your existing model.**

Hence, if you are building Linear regression on multiple variables, it is always suggested that you use Adjusted R-squared to judge goodness of model. In case you only have one input variable, R-square and Adjusted R squared would be exactly same.

**Ridge and Lasso:** they are some of the simple techniques to reduce model complexity and prevent over-fitting which may result from simple linear regression.

**Ridge Regression:** In ridge regression, the cost function is altered by adding a penalty equivalent to square of **the magnitude of the coefficients**. ( L2 regularization)**Lasso Regression :** The cost function for Lasso (**least absolute** shrinkage and selection operator) regression can be written as (L1 regularization)

[**Parameters vs Hyperparameters:-**](https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/#:~:text=In%20summary%2C%20model%20parameters%20are,be%20set%20manually%20and%20tuned.)

Some examples of model parameters include:

* The weights in an artificial neural network.
* The support vectors in a [support vector machine](https://machinelearningmastery.com/support-vector-machines-for-machine-learning/).
* The coefficients in a linear regression or logistic regression.

Some examples of model hyperparameters include:

* The learning rate for training a neural network.
* The C and sigma hyperparameters for support vector machines.
* The k in k-nearest neighbors.

[**Chose right machine learning Algorithm:**](https://www.kdnuggets.com/2020/05/guide-choose-right-machine-learning-algorithm.html)

1. **Size of the training data:**

If the training data is smaller or the training data has a fewer number of observation and higher number of features.

Chose algorithm with a high bias/low variance like linear regression, Navie Bayes or linear SVM.

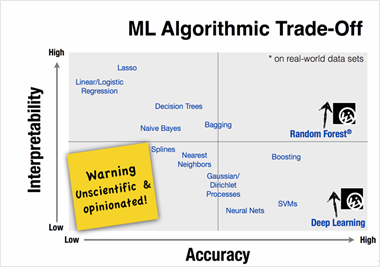
If the training data is larger/sufficient or the training data has a greater number of observations as compared to number of features.

Chose algorithm with a low bias/high variance like KNN, Decision tree or Kernel SVM.

1. ***Accuracy and interpretability of the output:***

Accuracy means that the model predicts the response for a given observation, that is close to the true value of that observation. (high flexibility and low interpretability)

Interpretability means that one can easily understand how the individual predictor is associated with the response. (low flexibility and high interpretability)



Now, to use which algorithm depends on the objective of the **business problem**. If inference is the goal, then restrictive models are better as they are much more interpretable. Flexible models are better if higher accuracy is the goal.

### ***Speed or Training time***

High Accuracy mean higher training time. Also, the algorithms require more time to train on large training data. In real world application, the choice of algorithm is driven by these two factors.

Algorithm like Naïve Bayes, linear Regression and Logistic regression are easy to implement and quick to run. Algorithm like SVM which involve tuning of parameters, Neural network with convergence time and random forest need a lot of time to train on data.

***Linearity***:

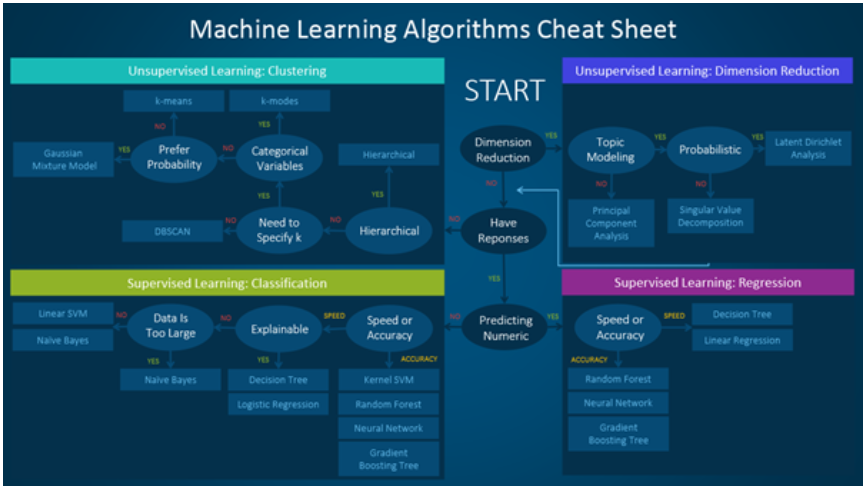
The many algorithms work on the assumption that the data is separated by a straight line (or it analog in higher dimension). This example includes linear regression, Logistic regression and SVM.

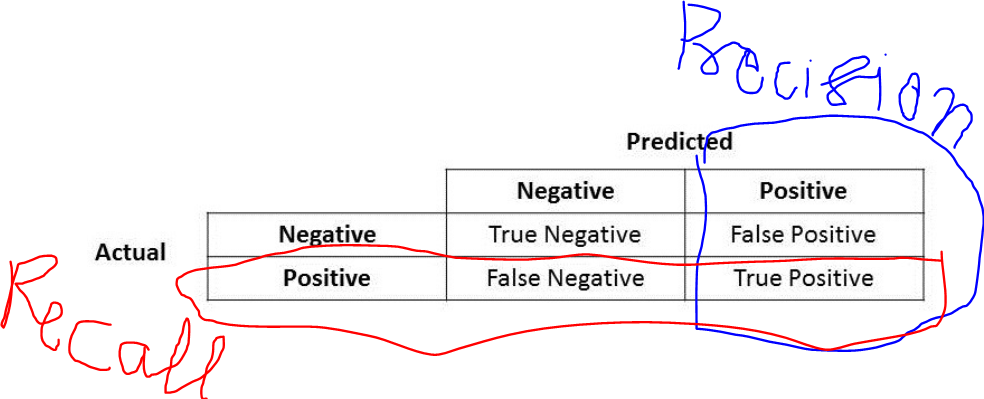
However, not all data is not linear. In that case, we need some algorithm to work in nonlinear and high dimension data. For example, random forest, kernel SVM and neural net.

### ***Number of features:***

The dataset may have many features that may not all be relevant and significant. For a certain type of data, such as genetics or textual, the number of features can be large compared to the number of data points.

Many features can bog down some learning algorithms, making training time unfeasibly long. SVM is better suited in case of data with large feature space and lesser observations. PCA and feature selection techniques should be used to reduce dimensionality and select important features.





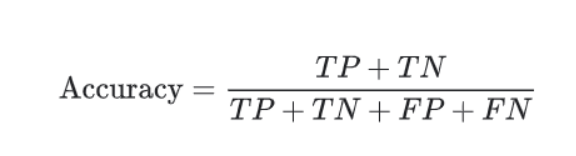
**Precision** is equal to true positive by total predicted positive. It is a good measure when the false positive cost is high.

For example: In case of email spam detection, a email is identified as spam (false positive) that in actuality not a spam email. By the user might can loose valuable email.

Recall is equal to true positive by total actual positive. It is a good measure when the false negative cost is too high.

For example: case of fraud detection and sick patient detection

***Accuracy:***

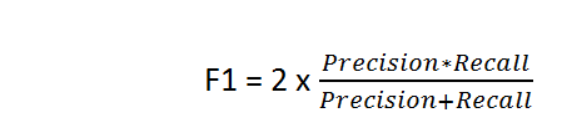


**F1 is used when we seek a balance b/w precision and recall.**

**Difference b/w accuracy and F1 score.**

F1 Score is needed when you want to seek a balance between Precision and Recall. Right…so what is the difference between F1 Score and Accuracy then?

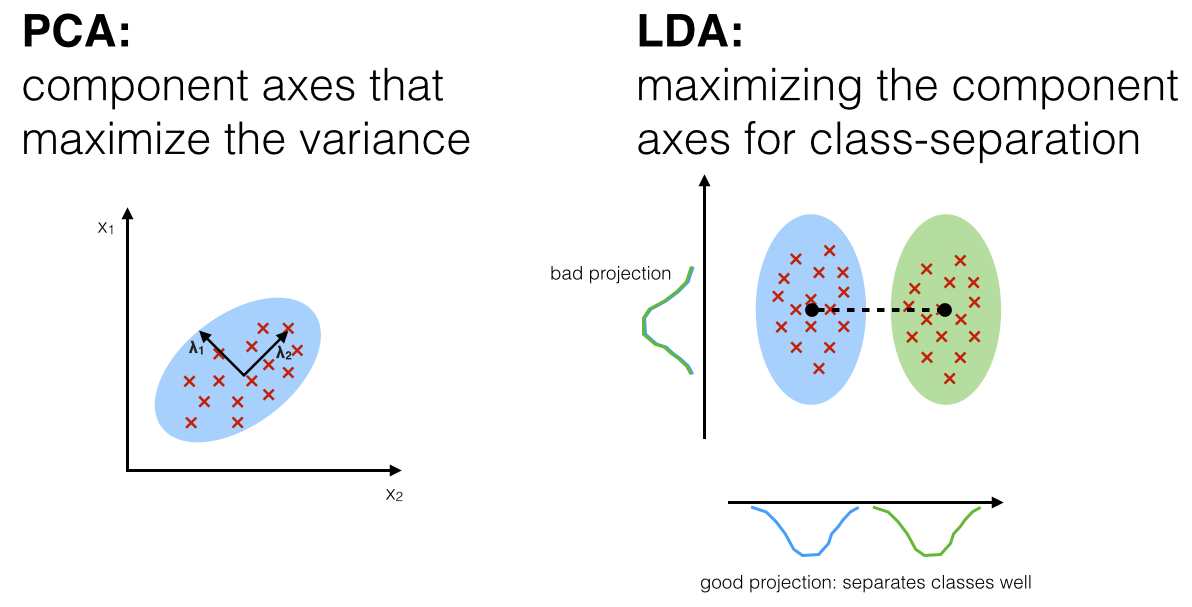
We have previously seen that accuracy can be largely contributed by a large number of True Negatives which in most business circumstances, we do not focus on much whereas False Negative and False Positive usually has business costs (tangible & intangible) thus F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).



**Type I Error**: False positive (rejection of a true null hypothesis)

**Type II Error**: False negative (non-rejection of a false null hypothesis)

***Dimensionality Reduction Techniques***



Principal Component Analysis (Unsupervised):

It reduces the dimension of a d-dimensional dataset by projecting it onto a (k)-dimensional subspace (where k<d)

There is a certain step to perform PCA:

* + 1. Standardize the dataset.
    2. Find the Eigenvalue and Eigenvectors using covariance matrix or correlation matrix.
    3. Sort the Eigenvectors in descending order.
    4. Create a projection matrix w using top K Eigenvectors.
    5. Transform the original dataset x using projection matrix to obtain k-dimensional feature subspace Y.

Linear Discriminant Analysis (Supervised):

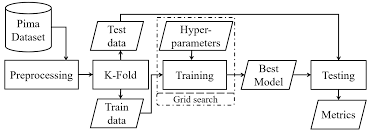
It is used to project a feature space onto a small subspace while maintaining the class discriminatory information.

There is a certain step to perform LDA:

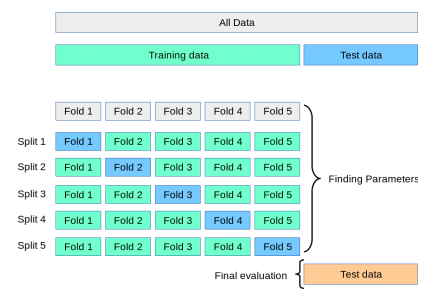
* 1. Compute the d-dimensional mean vectors for the different classes from the dataset.
  2. Compute the scatter matrices (in b/w classes and within class scatter matrices)
  3. Compute Eigenvectors and Eigenvalues from matrices
  4. Sort the Eigenvector and choose k eigenvectors with largest eigenvalues to form a d \* k dimensional matrix (d dataset dimension)
  5. Use this d\*k matrix to transform the samples onto the new subspace.

Note: PCA and LDA both are dimensionality reduction technique. PCA is unsupervised technique where LDA is supervised technique because of the relation to the dependent variable.

**Model Selection**



[***K-Fold cross validation:***](https://towardsdatascience.com/cross-validation-and-hyperparameter-tuning-how-to-optimise-your-machine-learning-model-13f005af9d7d)

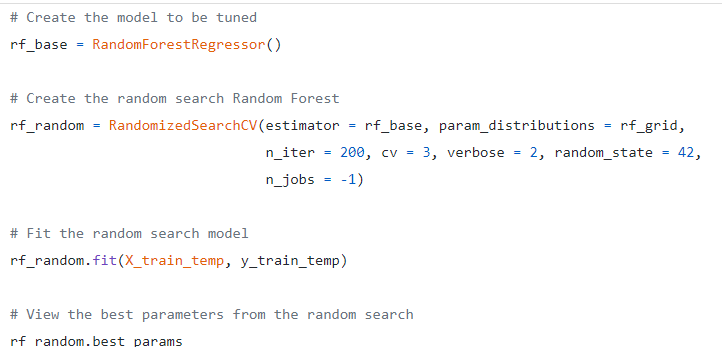


By training and testing the model K number of times on different subsets of the same training data we get a more accurate representation of how well our model might perform on data it has not seen before. In a K-fold CV we score the model after every iteration and compute the average of all scores to get a better representation of how the model performs compared to only using one training and validation set.

***Grid Search:***

One of the most popular approach to tune machine learning hyperparameters is called Grid search (Randomised Grid search cross validation)

In Randomised Grid Search Cross-Validation we start by creating a grid of hyperparameters we want to optimise with values that we want to try out for those hyperparameters.



**Clustering**

**K-mean Clustering**

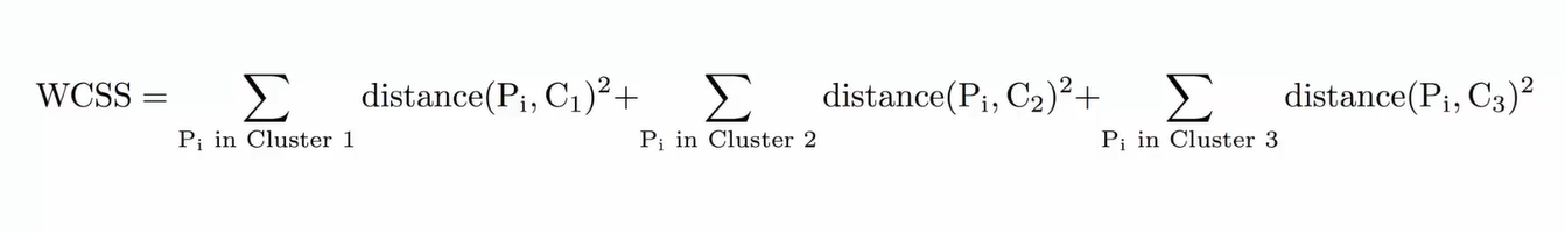
There is a certain step to perform K-mean:

1. Choose the number K of clusters.
2. Select at random K points as centroids.
3. Assign each data point to the nearest Centroid.
4. Computer and place the new centroids of each cluster.
5. Reassign each data point to the new closest centroid. If any reassignment took place, go back to step 4, otherwise go to finish.

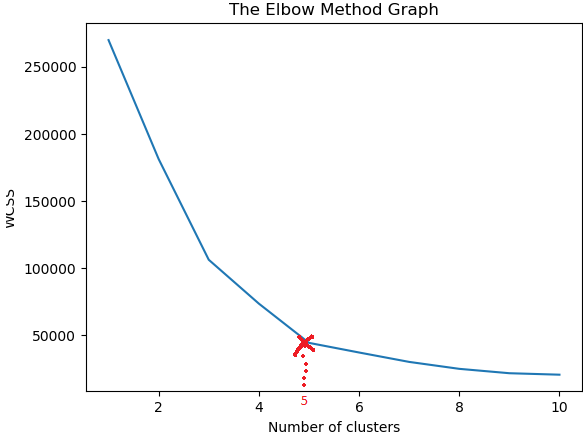
**K-Mean initialization trap:**

Random initialization trap is a problem that occurs in the K-means algorithm. In random initialization trap when the centroids of the clusters to be generated are explicitly defined by the User then inconsistency may be created, and this may sometimes lead to generating wrong clusters in the dataset.

**Choosing the right number of clusters:**



**Once we will compute WCSS:**



**Number of clusters are 5 in the above diagram.**

[***Neural Network:***](https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/)

***Forward Propagate***: We can calculate an output from a neural network by propagating an input signal through all layer till the outputs layer output its value.

There are three step for forward propagation:

* + 1. Neuron activation
    2. Neuron transfer
    3. Forward Propagation

1. Neuron activation:

The first is to calculate activation of one neuron given an input. It is weighted sum of input much like linear regression:

**activation = sum(weight\_i \* input\_i) + bias**

1. Neuron transfer:

Once the neuron is activated we need to transfer the activation to check what is the neuron output is.

**E.g Sigmoid, tanH, Relu, Leaky Relu**

3. Forward Propagation:

We work through each layer of our network calculating the outputs for each neuron. All of the outputs from one layer become inputs to the neurons on the next layer.

### Back Propagate Error:

Error is calculated between the expected outputs and the outputs forward propagated from the network. These errors are then propagated backward through the network from the output layer to the hidden layer, assigning blame for the error and updating weights as they go.

1. Transfer Derivative: Given an output value from a neuron, we need to calculate it’s slope.
2. Error Backpropagation: The first step is to calculate the error for each output neuron. This will give to error signal to propagate backwards through the network.

**error = (expected - output) \* transfer\_derivative(output)**

**Training Network:**

This part is broken down into two sections:

1. Update Weights.
2. Train Network.

**Update Weights:**

Once errors are calculated for each neuron in the network via the back propagation method above, they can be used to update weights.

Network weights are updated as follows:

**weight = weight + learning\_rate \* error \* input**

Where weight is a given weight, learning\_rate is a parameter that you must specify, error is the error calculated by the backpropagation procedure for the neuron and input is the input value that caused the error.

***Training Network:***

The network is updated using stochastic gradient descent.

This involves first looping for a fixed number of epochs and within each epoch updating the network for each row in the training dataset.

Because updates are made for each training pattern, this type of learning is called online learning. If errors were accumulated across an epoch before updating the weights, this is called batch learning or batch gradient descent.

BPTT and Truncated BPTT

***BPTT(Backpropagation through time):***

Backpropagation Through Time, or BPTT, is the application of the Backpropagation training algorithm to recurrent neural network applied to sequence data like a time series.

We can summarize the algorithm as follows:

1. Present a sequence of timesteps of input and output pairs to the network.
2. Unroll the network then calculate and accumulate errors across each timestep.
3. Roll-up the network and update weights.

Repeat.

BPTT can be computationally expensive as the number of timesteps increases.

***Truncated BPTT:***

We can summarize the algorithm as follows:

1. Present a sequence of k1 timesteps of input and output pairs to the network.
2. Unroll the network then calculate and accumulate errors across k2 timesteps.
3. Roll-up the network and update weights.

Repeat

The TBPTT algorithm requires the consideration of two parameters:

1. **k1**: The number of forward-pass timesteps between updates. Generally, this influences how slow or fast training will be, given how often weight updates are performed.
2. **k2**: The number of timesteps to which to apply BPTT. Generally, it should be large enough to capture the temporal structure in the problem for the network to learn. Too large a value results in vanishing gradients.

RNN:

**Return Sequence=true:-** It is possible to access the hidden state output for each input time step.

**Return State=True:-**

1. The LSTM hidden state output for the last time step.
2. The LSTM hidden state output for the last time step (again).
3. The LSTM cell state for the last time step.

**Return States=True and Sequences=True:**

We can access both the sequence of hidden state and the cell states at the same time.

This can be done by configuring the LSTM layer to both return sequences and return states.

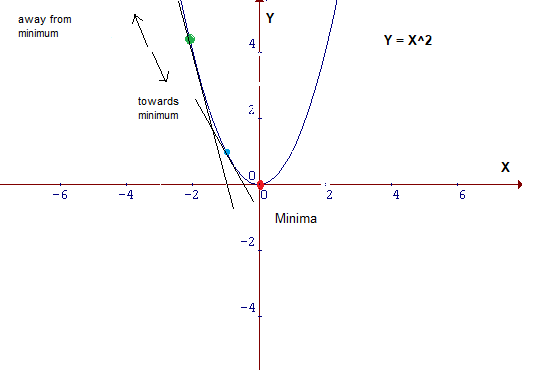
**lstm1, state\_h, state\_c = LSTM(1, return\_sequences=True, return\_state=True)**

|  |  |
| --- | --- |
|  | **Note:** |

* That return sequences return the hidden state output for each input time step.
* That return state returns the hidden state output and cell state for the last input time step.
* That return sequences and return state can be used at the same time.

***Gradient descent*** is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks.

Essentially, there are two things that you should know to reach the minima, i.e. which way to go and how big a step to take. The tangent gives us a sense of the steepness of the slope.



***Learning*** ***rate:***

This size of steps taken to reach the minimum or bottom is called **Learning Rate**



Note:

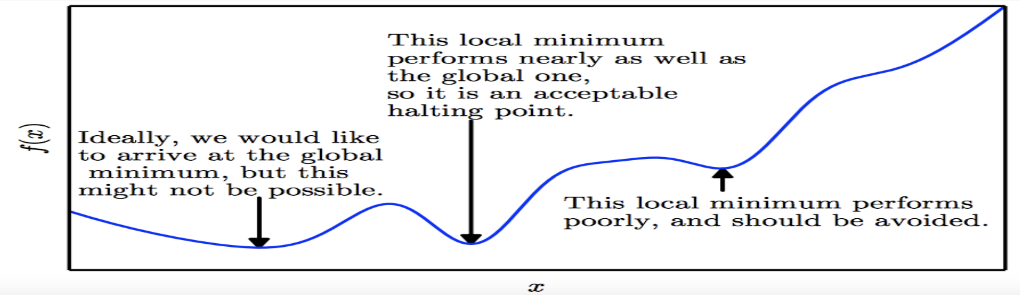
**During the batch gradient descent, we look at the error of all the training examples at once while in the SGD we look at each error at a time.**

**Challenges with Optimization**: Non-convex optimization involves a function which has multiple optima, only one of which is the global optima. Depending on the loss surface, it can be exceedingly difficult to locate the global optima.

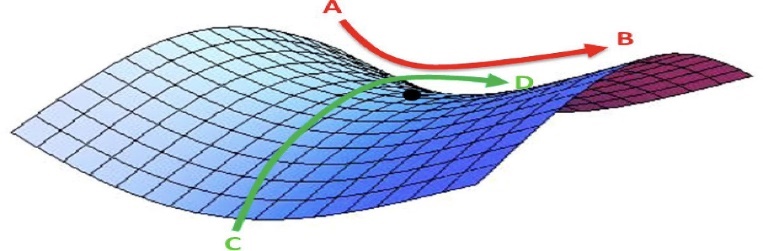
There are certain problems associated with this:

1. What is reasonable learning rate to use?
2. How do we avoid getting stuck in local optima?
3. What if the loss surface morphology changes?

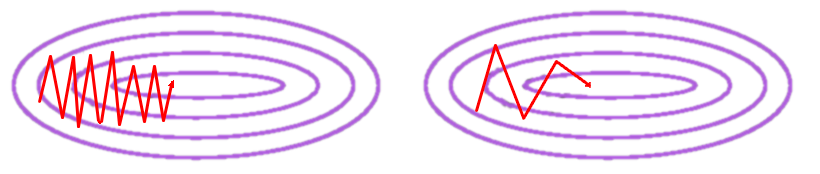
***Local Optima:***Previously, local minima were viewed as a major problem in neural network training. Nowadays, researchers have found that when using sufficiently large neural networks, most local minima incur a low cost, and thus it is not particularly important to find the true global minimum — a local minimum with reasonably low error is acceptable.



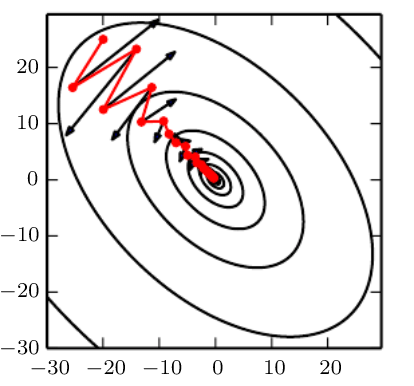
**Saddle Points:** Recent studies indicate that in high dimensions, saddle points are more likely than local minima. Saddle points are also more problematic than local minima because close to a saddle point the gradient can be exceedingly small. Thus, gradient descent will result in negligible updates to the network and hence network training will cease.



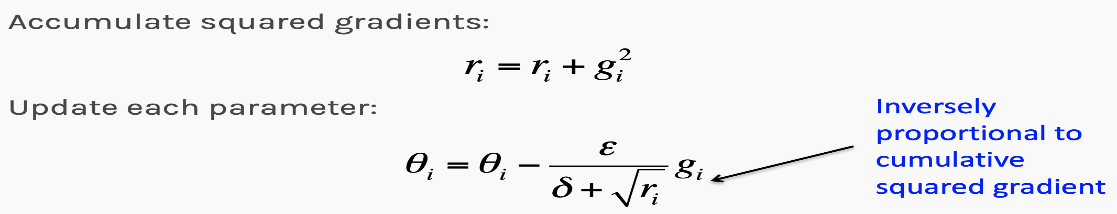
**Momentum:** One problem with stochastic gradient descent (SGD) is the presence of oscillations which result from updates not exploiting curvature information. This results in SGD being slow when there is high curvature.



We can see the effects of adding momentum on an optimization algorithm. The first few updates show no real advantage over vanilla SGD — since we have no previous gradients to utilize for our update. As the number of updates increases our momentum kickstarts and allows faster convergence.



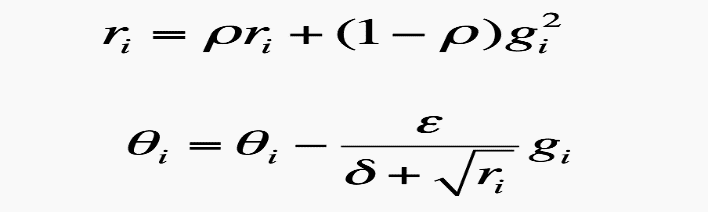
**AdaGrad:** Momentum adds updates to the slope of our error function and speeds up SGD in turn. AdaGrad adapts updates to each **individual parameter to perform larger or smaller updates depending on their importance.**



One of Adagrad’s main benefits is that it eliminates the need to manually tune the learning rate and results in greater progress along gently sloped directions.

AdaGrad’s main weakness is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. This, in turn, causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.

**RMSProp:** For non-convex problems, AdaGrad can prematurely decrease the learning rate. We can use **an exponentially weighted average for gradient accumulation**.

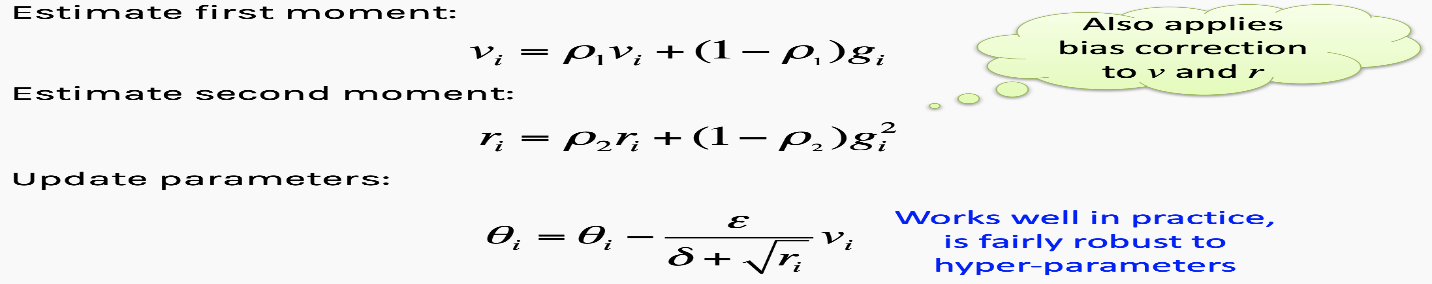


## **Adam**

Adam is a combination of RMSprop and momentum. Adam refers to adaptive moment estimation, and it is the most popular optimizer used for neural networks today.

Adam computes adaptive learning rates for each parameter. In addition to **storing an exponentially decaying average of past squared gradients** like RMSprop,

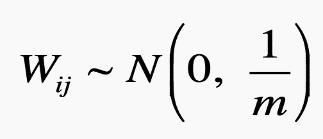
Adam also keeps an **exponentially decaying average of past gradients**, like momentum.



***Parameter Initialization:***

What should be the scale of this initialization? If we choose large values for the weights, this can lead to exploding gradients. On the other hand, small values for weights can lead to vanishing gradients.

Xavier Initialization: We need to initialize the weights in such a way that the variance remains the same for both the input and the output.

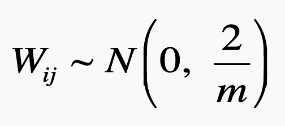


The value m is sometimes called the fan-in: the number of incoming neurons (input units in the weight tensor).

**He Normal Initialization:**

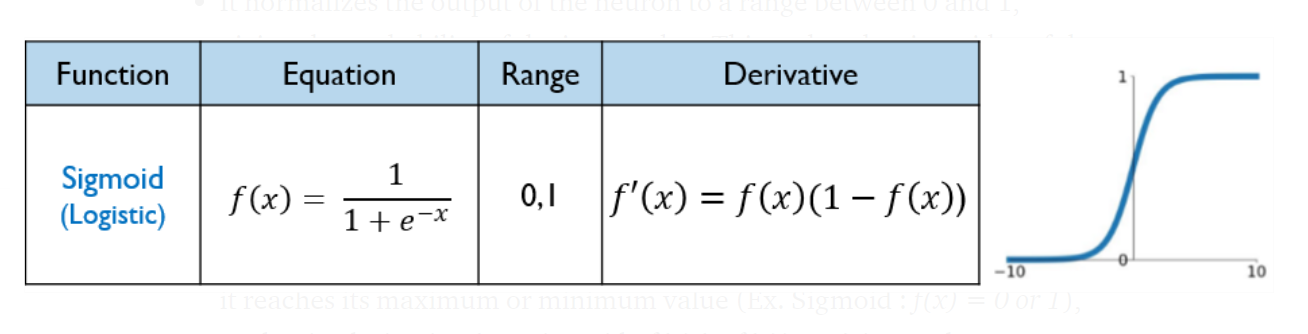
He normal initialization is essentially the same as Xavier initialization, except that the variance is multiplied by a factor of two.

In this method, the weights are initialized keeping in mind the size of the previous layer which helps in attaining a global minimum of the cost function faster and more efficiently. The weights are still random but differ in range depending on the size of the previous layer of neurons. This provides a controlled initialization hence the faster and more efficient gradient descent.



**Nonlinear Activation function**

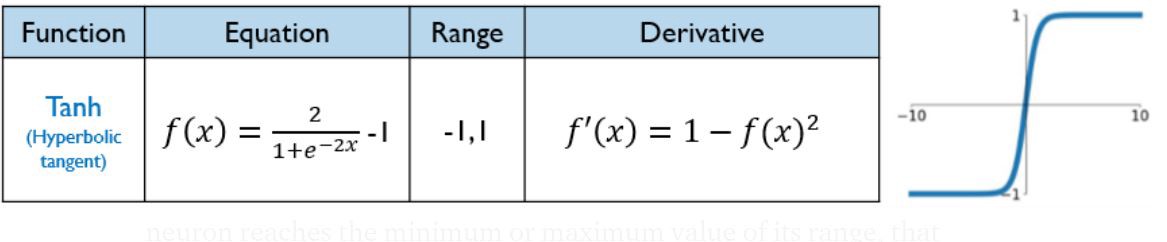
**Sigmoid or logistic Activation Function:**



Cons:

* Derivative of sigmoid function suffers **“Vanishing gradient and Exploding gradient problem”.**
* Sigmoid function in not “zero-centric”. This makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
* Slow convergence- as its computationally heavy.

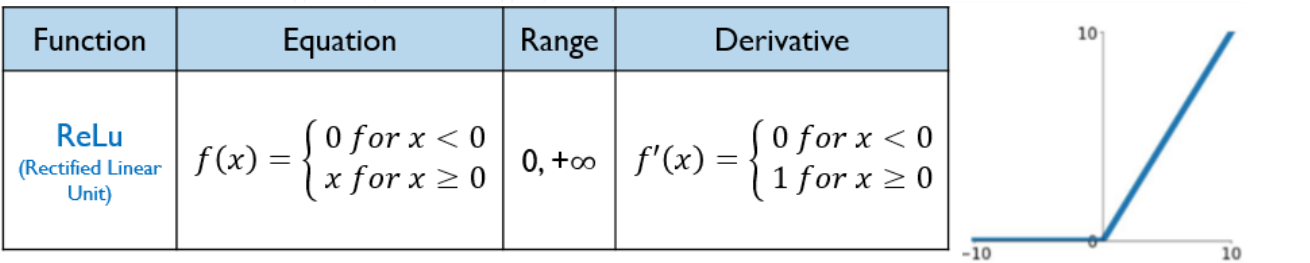
**Tanh Activation Function:**



Cons:

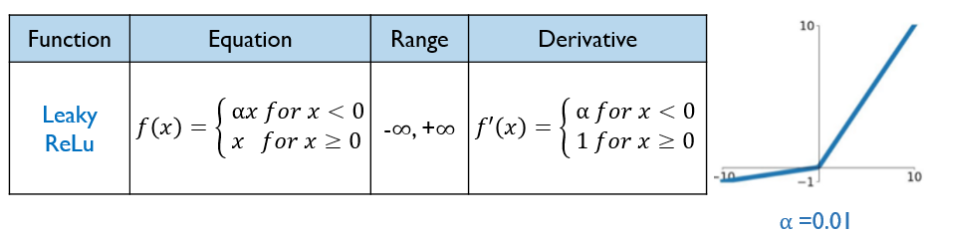
* Derivative of Tanh function suffers “Vanishing gradient and Exploding gradient problem”.
* Slow convergence- as its computationally heavy.

**ReLu Activation Function (ReLU-Rectified Linear units):**



**Problem of Dying neuron/Dead neuron :**As the ReLu derivative f’(x) is not 0 for the positive values of the neuron (f’(x)=1 for x ≥ 0), ReLu does not saturate (exploid) and no dead neurons (Vanishing neuron)are reported. **Saturation and vanishing gradient only occur for negative values that, given to ReLu, are turned into 0- This is called the problem of dying neuron**.”

***Leaky ReLu Activation Function:***



Leaky ReLU is defined to address problem of dying neuron/dead neuron.

## ***NLP basic***

***Term Frequency***: The number of times a term occurs in a document is called its term frequency.

***Document Frequency***: The only difference is that TF is frequency counter for a term t in document d, where as DF is the count of **occurrences** of term t in the document set N.

***Inverse Document Frequency (IDF):***

While computing TF, all terms are considered equally important. However, it is known that certain terms, such as “is”, “of”, and “that”, may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing IDF, an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

[***Topic modeling***](https://en.wikipedia.org/wiki/Topic_model)

**Note\*: Word🡪document🡪 Corpus.**

[***Topic modeling***](https://en.wikipedia.org/wiki/Topic_model)***:***  it is a type of statistical modeling for discovering the abstract “topics” that occur in a collection of documents. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions.

***LSA: Latent Semantic Analysis:***

Decomposing Documents and terms matrix into a separate document-topic matrix and a topic-term matrix.

LSA models typically replace raw counts in the document-term matrix with a **tf-idf score.**

This dimensionality reduction can be performed using **truncated SVD**. SVD, or singular value decomposition, is a technique in linear algebra that factorizes any matrix M into the product of 3 separate matrices: M=U\*S\*V, where S is a diagonal matrix of the [singular values](https://en.wikipedia.org/wiki/Singular_value) of M.

***pLSA, or Probabilistic Latent Semantic Analysis:*** It uses a ***probabilistic method instead of SVD*** to tackle the problem. The core idea is to find a probabilistic model with latent topics that can generate the data we observe in our document-term matrix.

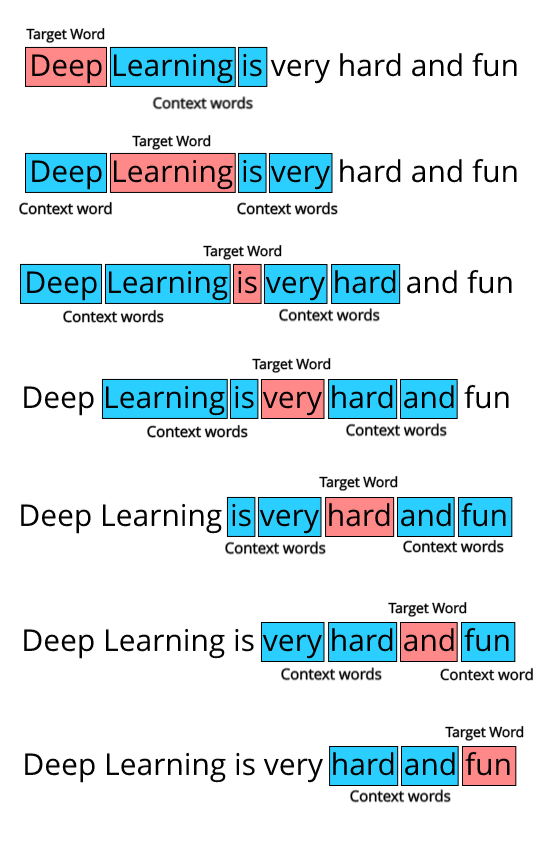
LDA stands for Latent Dirichlet Allocation. **LDA is a Bayesian version of pLSA**. It uses Dirichlet priors for the document-topic and word-topic distributions, lending itself to better generalization.

***Word Embedding***

***Word2Vec:***

***Skipgram model***: we try to predict each context word given a target word.

Consider, “Deep Learning is very hard and fun”. We need to set something known as window size. Let us say 2 in this case.



***CBOW:* the only difference is that we try to predict the target word given the context words.**

***Glove***

Glove is based on **matrix factorization technique on word context matrix**. It first **constructs** a **large matrix** of (words x context) co-occurrence information ie. for each word, you count how frequently we see those word in some context in a large corpus.

In order to understand how GloVe works, we need to understand 2 main methods which GloVe was built on -

**Global matrix factorization:**

In NLP, global matrix factorization is the process of using matrix factorization form linear algebra to reduce large term frequency matrices. These matrices usually represent the occurrences or the absence of words in the document.

**Local context window:**

Local context window methods are **CBOW**and **Skip-Gram**

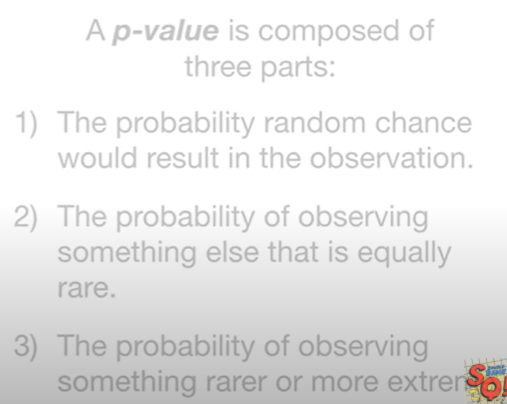
**Statistics**

**What Is a Null Hypothesis?**

A null hypothesis is a type of hypothesis used in [statistics](https://www.investopedia.com/terms/s/statistics.asp) that proposes that there is no difference between certain characteristics of a population (or data-generating process).

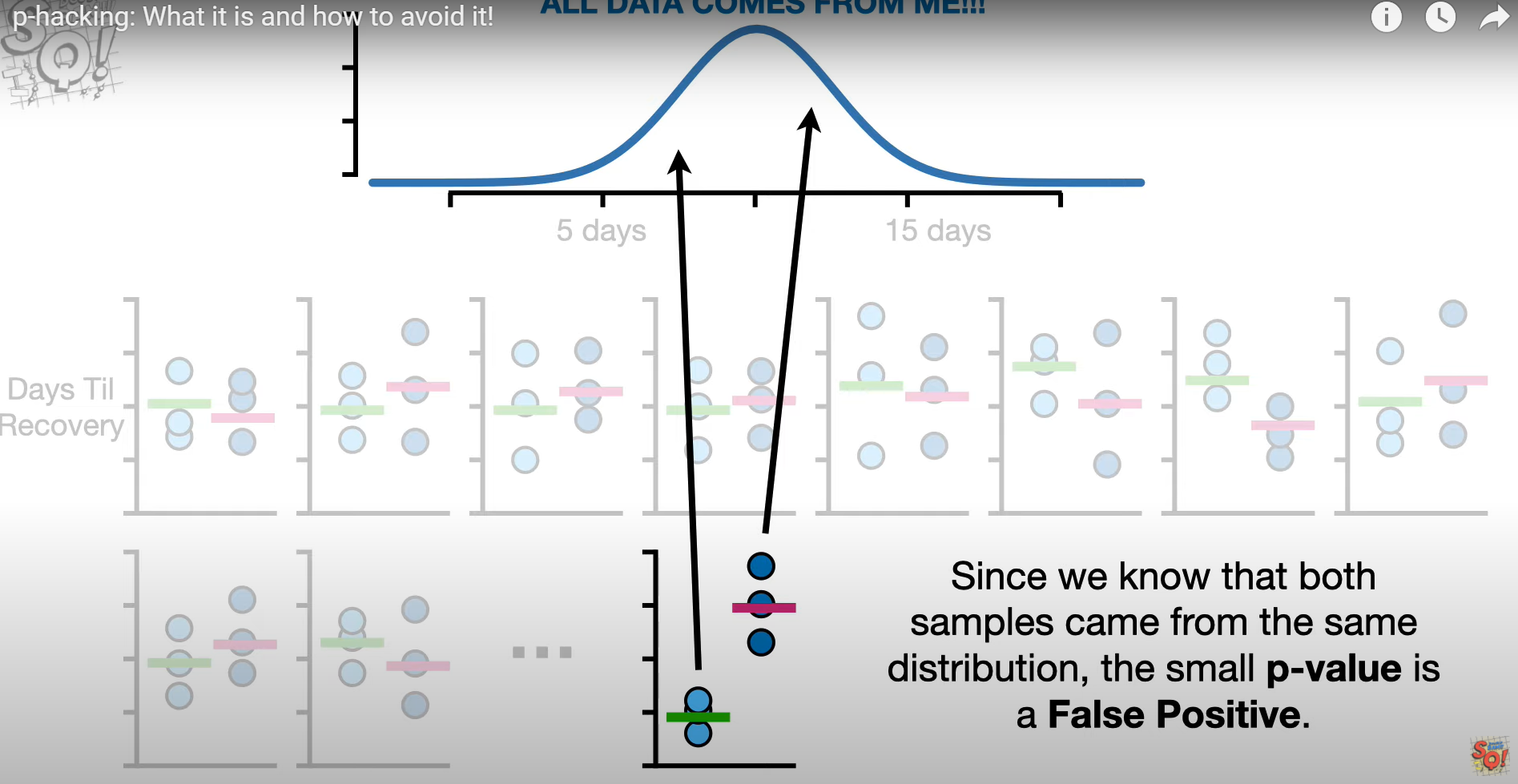
**What is the P-value?**

A p-value, or probability value, is a number describing how likely it is that **your data would have occurred by random chance** (i.e. that the null hypothesis is true).

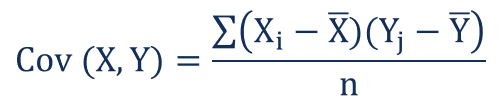


If the P value is smaller, then It tells us that some other distribution would do a better job explaining the data.

[P-hacking:](https://www.youtube.com/watch?v=HDCOUXE3HMM) it refers to the misuse and abuse of analysis techniques and results in being fooled by false positives.

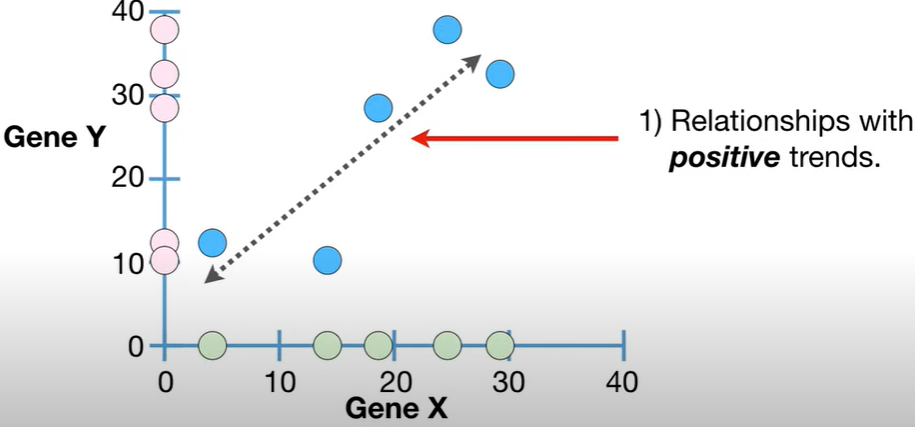


**Covariance** is a measure of the relationship between two random variables. The metric evaluates how much – to what extent – the variables change together.

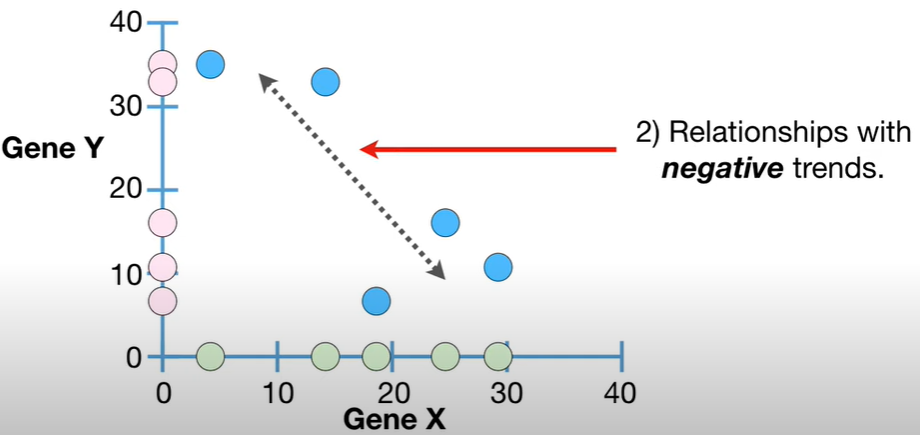


Covariance can classify three types of relationships.

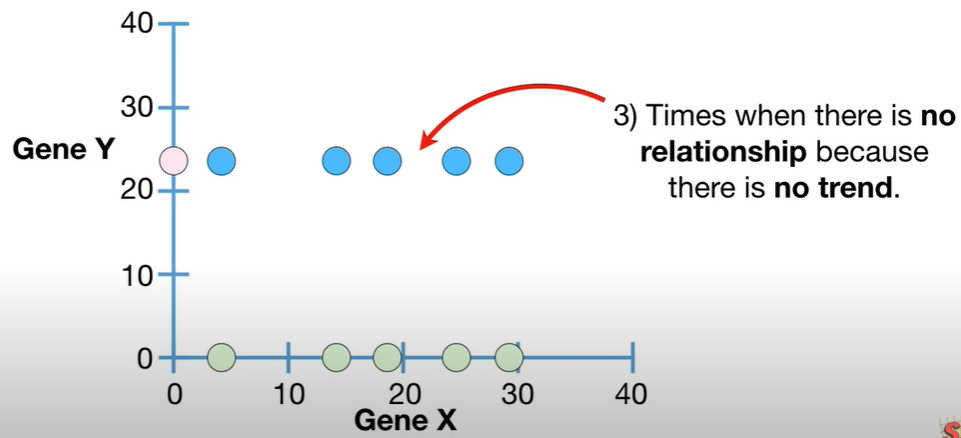
1.Positive trend(slope):



2.Negative Trend:

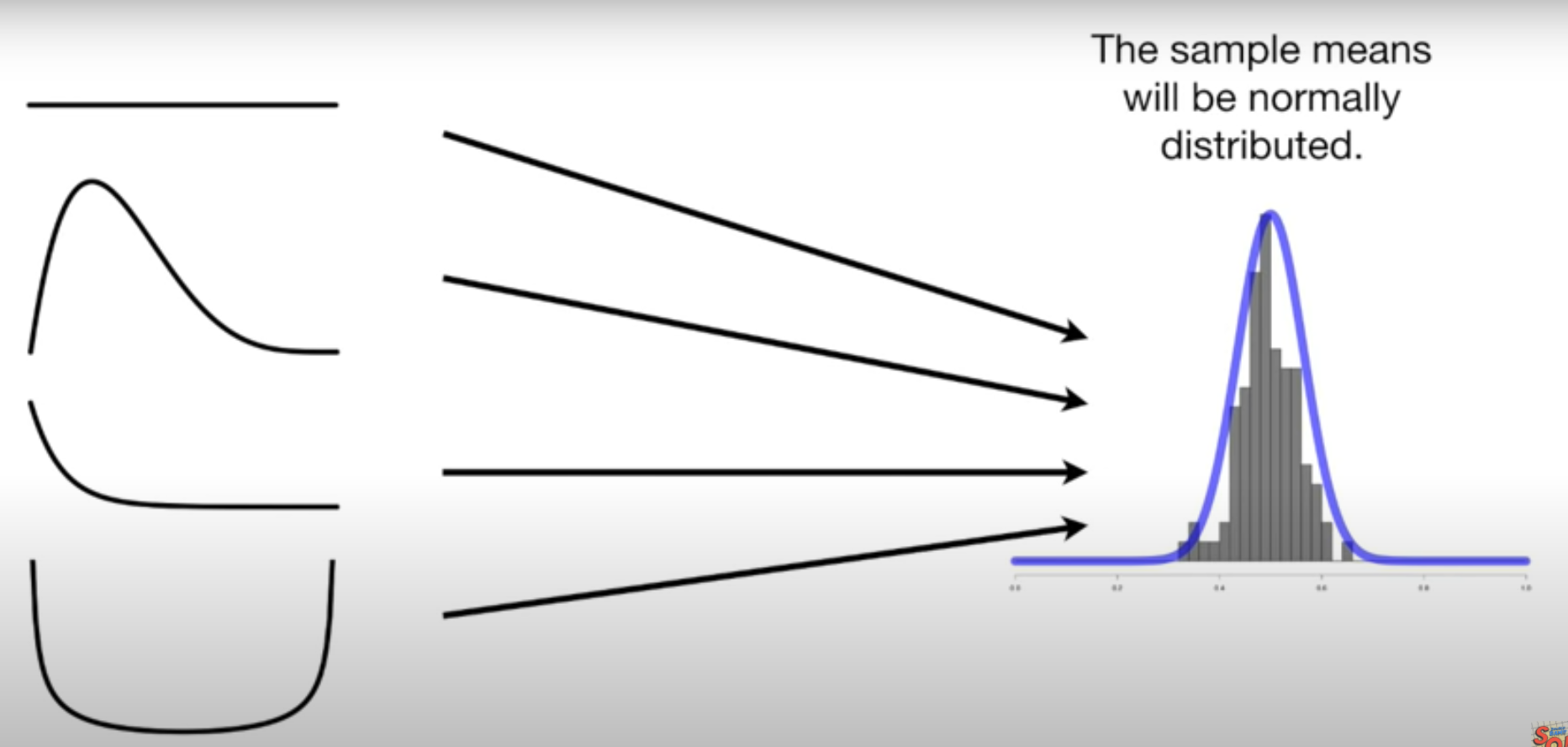


3.No relationship:



**Central limit theorem:**

The Central Limit Theorem states that the [sampling distribution](https://www.statisticshowto.com/sampling-distribution/)**of the**[sample means](https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/sample-mean/) approaches a [normal distribution](https://www.statisticshowto.com/probability-and-statistics/normal-distributions/) as the [sample size](https://www.statisticshowto.com/probability-and-statistics/find-sample-size/) gets larger — no matter what the shape of the [population](https://www.statisticshowto.com/what-is-a-population/) distribution.

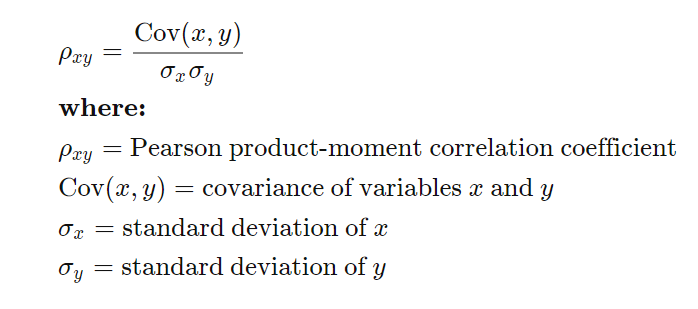


**Note:**

**The Standard deviation of means is called Standard error.**

**Correlation:**

The correlation coefficient is a statistical measure of the strength of the relationship between the relative movements of two variables. The values range between -1.0 and 1.0.



**Interquartile range**

QR = Q3 – Q1

Where, Q3 is the third quartile (75 percentile)

Where, Q1 is the first quartile (25 percentile)

How to find outliers?

Widely used – Any data point that lies outside the 1.5 \* IQR

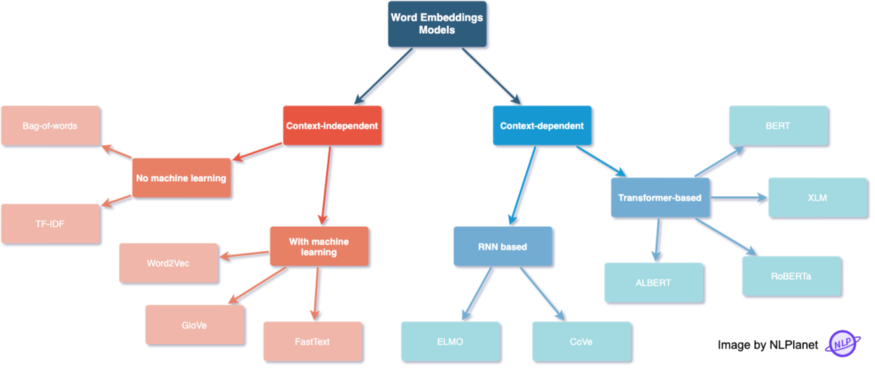
Lower bound = Q1 – (1.5 \* IQR)

Upper bound = Q3 + (1.5 \* IQR)

***Causation*** indicates that one event is the result of the occurrence of the other event, i.e. there is a causal relationship between the two events.

The use of a controlled study is the most effective way of establishing causality between variables. In a controlled study, the sample or population is split in two, with both groups being comparable in almost every way. The two groups then receive different treatments, and the outcomes of each group are assessed.  
  
For example, in medical research, one group may receive a placebo while the other group is given a new type of medication. If the two groups have noticeably different outcomes, the different experiences may have caused the different outcomes.

# Word embeddings models



## Context-independent

It is methods learn same embeddings for the same word based on its context

1. I went to bank to withdraw money
2. I went to bank of river to see fish

Both places word having same meaning.

**Context-independent without machine learning**

### [Bag-of-words](https://en.wikipedia.org/wiki/Bag-of-words_model)

A text, such as a sentence or a document, is represented as the bag of its words, disregarding grammar and even word order but keeping multiplicity.

### [TF-IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf)

Gets this importance score by getting the term’s frequency (TF) and multiplying it by the term inverse document frequency (IDF).

**Context-independent with machine learning**

### Word2Vec

Two model architectures: continuous bag-of-words (CBOW) or continuous skip-gram.

CBOW predicts current using the  surrounding window of context words.

CSG predicts surrounding window of context words using current word.

No, Word2Vec is not a deep learning model, although it is using a neural network with one input layer, one hidden layer and one output layer.  The number of parameters, layers and non-linearlities will be too small to be considered a deep learning model.

### [GloV](https://nlp.stanford.edu/projects/glove/)e(Global Vectors for Word Representation)

Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space

### FastText

Unlike GloVe, it embeds words by treating each word as being composed of character n-grams instead of a word whole. This feature enables it not only to learn rare words but also out-of-vocabulary words.

## Context-dependent

It is methods learn different embeddings for the same word based on its context

1. I went to bank to withdraw money
2. I went to bank of river to see fish

Both places word having different meaning.

**Context-dependent and RNN based**

### ELMO

Embeddings from Language Model: learns contextualized word representations based on a neural language model with a character-based encoding layer and two BiLSTM layers.

### CoVe

Contextualized Word Vectors: uses a deep LSTM encoder from an attentional sequence-to-sequence model trained for machine translation to contextualize word vectors.

**Context-dependent and transformer-based**

### BERT

Bidirectional Encoder Representations from Transformers: transformer-based language representation model trained on a large cross-domain corpus. Applies a masked language model to predict words that are randomly masked in a sequence, and this is followed by a next-sentence-prediction task for learning the associations between sentences.

### XLM

Cross-lingual Language Model: it’s a transformer pretrained using next token prediction, a BERT-like masked language modeling objective, and a translation objective.

### RoBERTa

Robustly Optimized BERT Pretraining Approach: it builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates.

### ALBERT

A Lite BERT for Self-supervised Learning of Language Representations: it presents parameter-reduction techniques to lower memory consumption and increase the training speed of BERT.

<https://medium.com/nlplanet/two-minutes-nlp-11-word-embeddings-models-you-should-know-a0581763b9a9>

# [Hugging Face](https://huggingface.co/)

an open-source library for building, training, and deploying state-of-the-art machine learning models

two main libraries, [transformers](https://github.com/huggingface/transformers) for models and [datasets](https://github.com/huggingface/datasets)

<https://medium.com/nlplanet/bert-finetuning-with-hugging-face-and-training-visualizations-with-tensorboard-46368a57fc97>

<https://medium.com/nlplanet/two-minutes-nlp-spacy-cheat-sheet-21471dac7837>