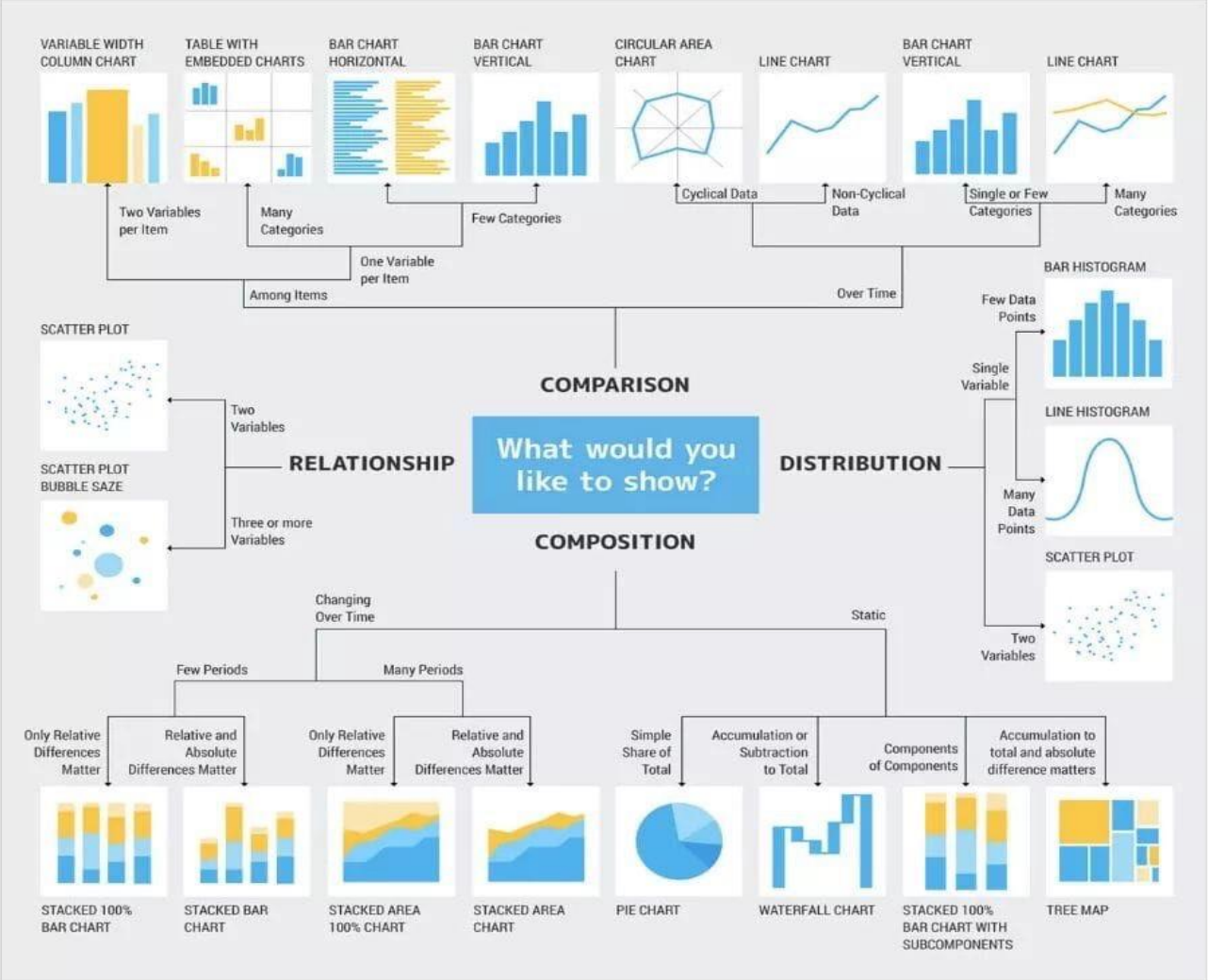
Machine Learning theoretical concepts

By: Vikram Pal

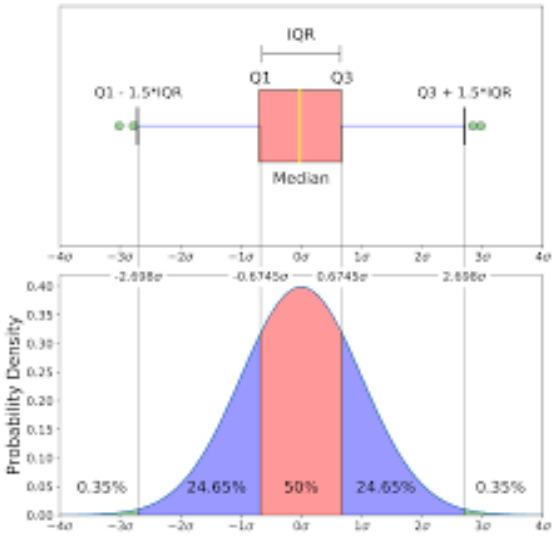
Data cleaning and reducing bias & variance :



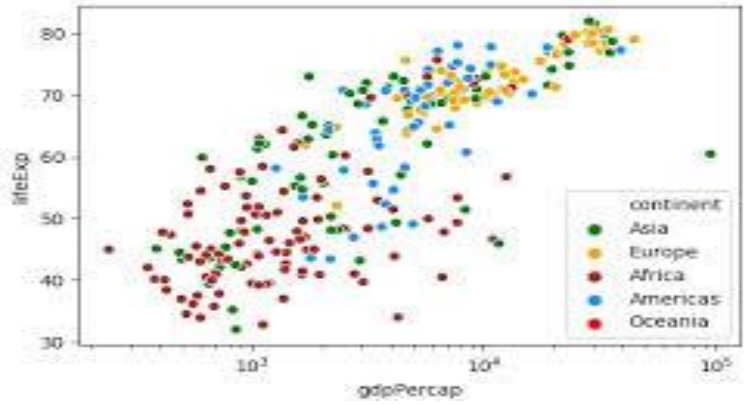
1. Inconsistent column:
    a. pandas.DataFrame.drop
2. Missing data:
    1. Leave as it is, 2. Filling the missing values, 3. Drop them
3. Outliers:
    a. For detecting the outliers, we can use:
        1. Box Plot

Machine Learning theoretical concepts

By: Vikram Pal



2. 2. Scatter plot

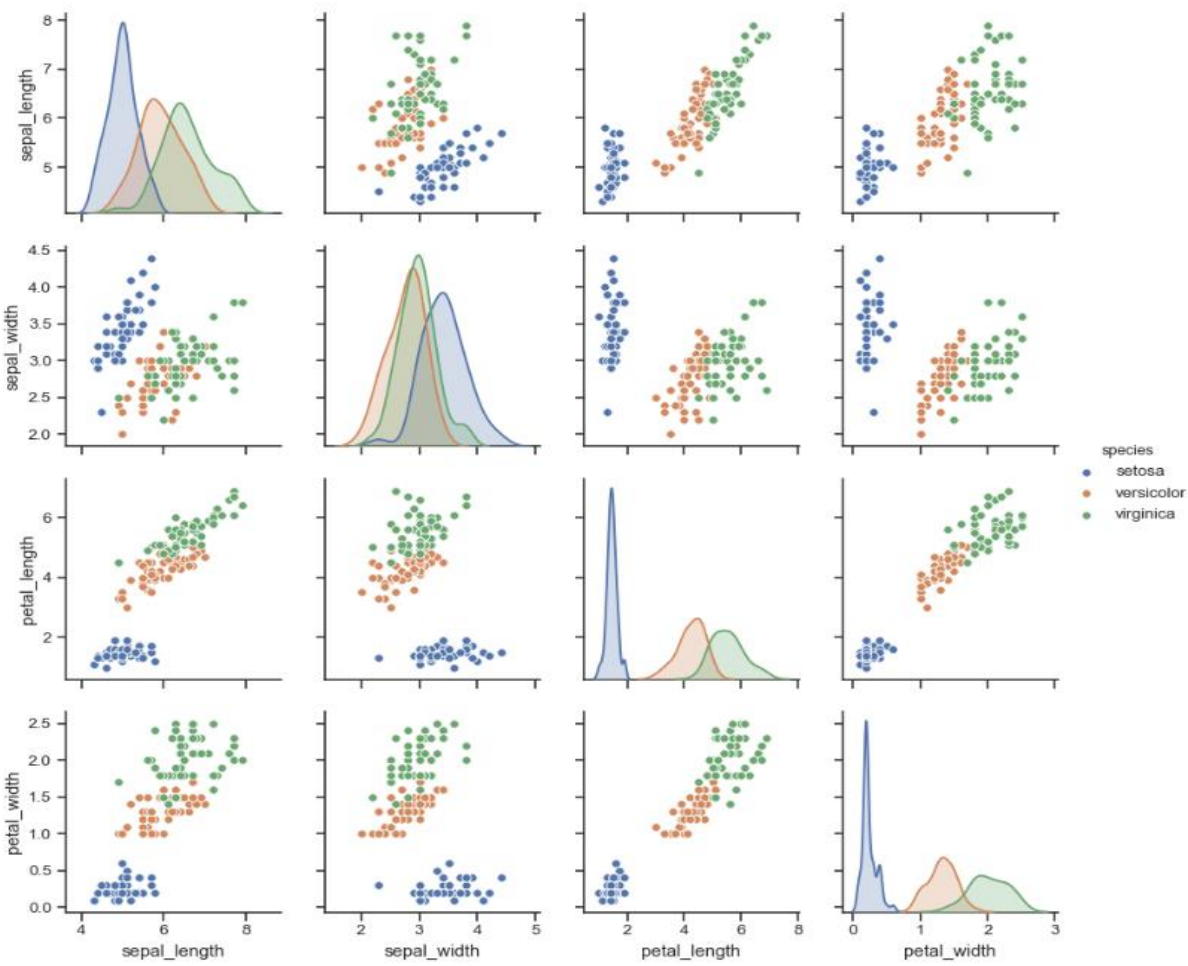


2. Z-score etc.

4. Duplicate rows:
    i. dataset_name.drop_duplicates()
5. Tidy Data set:
    a. pandas.melt.
6. Converting data types:
    a. In Data Frame data can be of many types. As example: 1. Categorical data, 2. Object data, 3. Numeric data, 4. Boolean data
7. String Manipulation
8. Data concatenation
9. Correlation: it is used for checking linear relation b/w two variable.
10. Pair plot: multiple pairwise bivariate distribution

Machine Learning theoretical concepts

By: Vikram Pal



Feature Selection Methods:

1. Univariate Selection: scikit-learn library provides the SelectKBest class
2. Feature importance: it gives you a score for each feature of your data, the higher the score more important or relevant is the feature towards your output variable.
3.Correlation Matrix with Heatmap: Correlation states how the features are related to each other or the target variable.

Must read this:

https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226

Regression key points:

Variance Inflation Factor:

Variance inflation factor (VIF) is a measure of the amount of **multicollinearity in a set of multiple regression variables.**

Machine Learning theoretical concepts

By: Vikram Pal

VIF>5 means that variables have strong multicollinearity.

The Bayesian information criterion (BIC) and the Akaike information criterion (AIC):
measures of model fit to compare different models or to select best model.

AIC and BIC penalize models with more parameters, so they can be used to identify models that are overfitting the data.

results = model.fit()

vif["VIF"] = [variance_inflation_factor(predictors.values, i) for i in range(predictors.shape[1])]

print("BIC:", results.bic)
print("AIC:", results.aic)

Regression Analysis: Femoral Neck versus %Fat S, Weight S, Activity S

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	4	0.55578	0.138946	27.95	0.000
%Fat S	1	0.04786	0.047863	9.63	0.003
Weight S	1	0.30473	0.304728	61.29	0.000
Activity S	1	0.04703	0.047027	9.46	0.003
%Fat S*Weight S	1	0.04175	0.041745	8.40	0.005
Error	87	0.43256	0.004972		
Total	91	0.98834			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
0.0705118	56.23%	54.22%	50.48%

Coefficients

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	0.82161	0.00973	84.40	0.000	
%Fat S	-0.00598	0.00193	-3.10	0.003	3.32
Weight S	0.00835	0.00107	7.83	0.000	4.75
Activity S	0.000022	0.000007	3.08	0.003	1.05
%Fat S*Weight S	-0.000214	0.000074	-2.90	0.005	1.99

Tidy(t.value)

The t-value check evidence against the null hypothesis.

fit <- lm(y ~ x1 + x2 + x3, data = df)
# Extract the estimates and other statistical measures for the model
tidy(fit)

output
term: the term or predictor variable being estimated
estimate: the estimate or coefficient for the term
std.error: the standard error of the estimate
t.value: estimate/ std.error
p.value: probability that the true value of the coefficient is zero

Machine Learning theoretical concepts

By: Vikram Pal

Anova(F-value)

F-value is a measure of the overall fit of the model.

F-value is ratio of the mean square error (MSE) for the model to the MSE for the residuals.

We can get the F-value for a linear regression model by using the anova function.

```
# Fit a linear regression model
fit <- lm(y ~ x1 + x2 + x3, data = df)

# Calculate the F-value for the model
anova(fit)
```

output

Df: the degrees of freedom for the term
Sum Sq: the sum of squares for the term
Mean Sq: sum of squares divided by the degrees of freedom
F value: the F-value for the term
Pr(>F): the p-value for the F-value, which is the probability that the true value of the F-value is zero

T-test

T-test is a univariate hypothesis test, that is applied when standard deviation is not known and the sample size is small.

- size is small (e.g., less than 30).
- used to compare the mean between two groups.
- assumes that the data are normally distributed but does not assume equal variances.
- used to test the null hypothesis that the means between two groups are equal.

F-test

F-test is a statistical test, that determines the equality of the variances of the two normal populations.

- F-test sample size is large greater than 30,
- An F-test is used to compare the variance between two groups,
- An F-test assumes that the data are normally distributed and have equal variances,
- An F-test is used to test the null hypothesis that the variances between two groups are equal,

Machine Learning theoretical concepts

By: Vikram Pal

Standard Error:

$$\mathbf{SE} = \frac{\sigma}{\sqrt{n}}$$

$\mathbf{SE}$ = standard error of the sample

σ = sample standard deviation

n = number of samples

SE indicates how different the population mean is likely to be from a sample mean.

T-Value:

The t statistic is **the coefficient divided by its standard error**. The t-value is a way to quantify the difference between the population means.

How do we interpret T in regression?

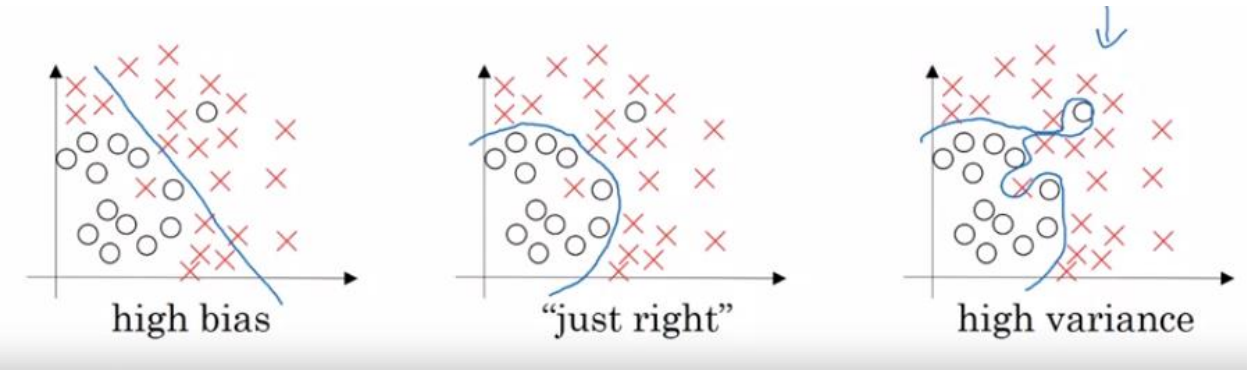
If the p-value that corresponds to t is less than some threshold (e.g. α = . 05) then we reject the null hypothesis and conclude that there is a statistically significant relationship between the predictor variable and the response variable

SMOTE:

is an **oversampling technique** where the synthetic samples are generated for the minority class. This algorithm helps to overcome **the overfitting problem posed by random oversampling**

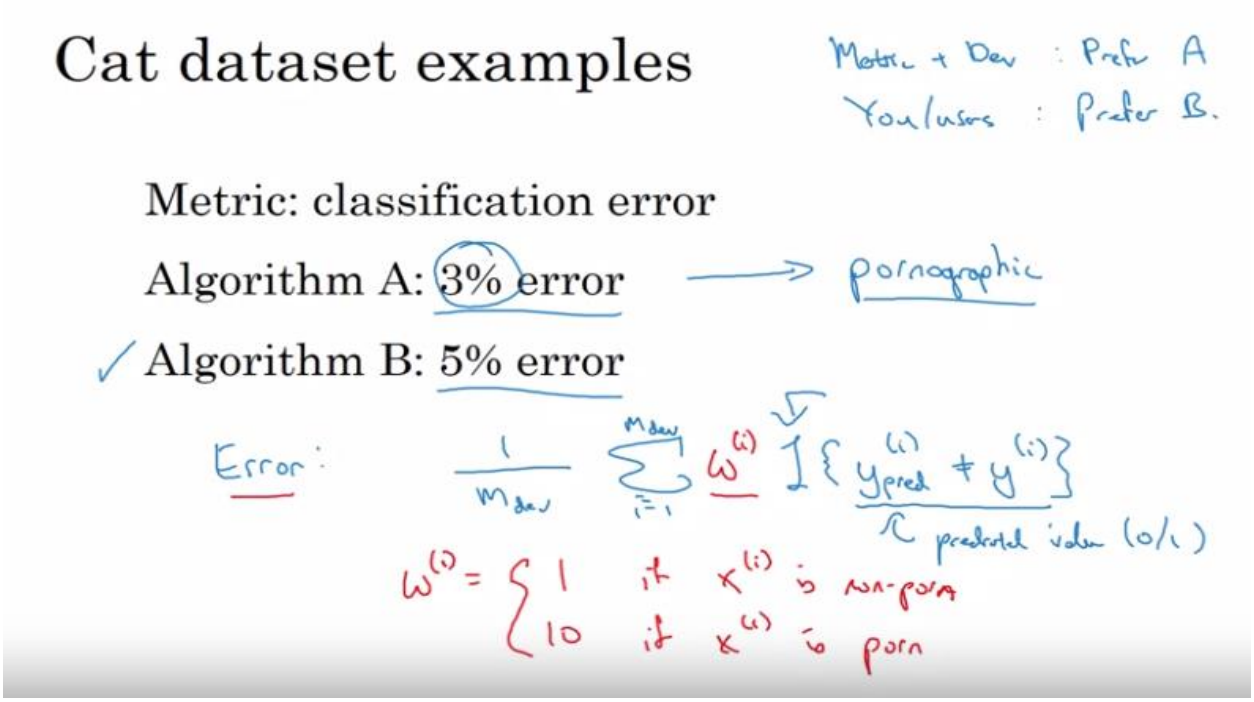
When to Change Dev/Test Sets and Metrics?

https://www.coursera.org/learn/machine-learning-projects/lecture/Ux3wB/when-to-change-dev-test-sets-and-metrics



Machine Learning theoretical concepts

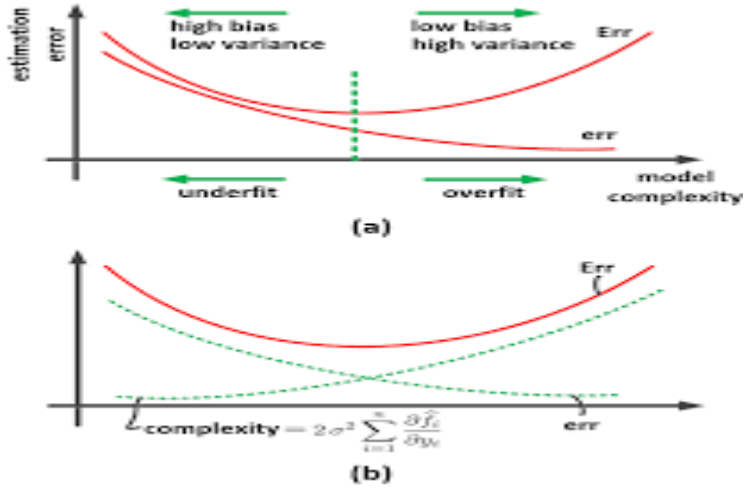
By: Vikram Pal



How do you calculate bias-variance trade-off?

K-fold cross validation + Grid-Search == compare the score across the different tuning options.

You can measure the bias-variance trade-off using k-fold cross validation and applying Grid-Search on the parameters. This way you can compare the score across the different tuning options that you specified and choose the model that achieve the higher test score.



Cook's distance:

It is used to identify influential observations in a regression model.
The formula for Cook's distance is:

$\mathbf{D}_i = (r_i^2 / p*MSE) * (h_{ii} / (1-h_{ii})^2)$

where:

Machine Learning theoretical concepts

By: Vikram Pal

- **r_i** is the i^{th} residual
- **p** is the number of coefficients in the regression model
- **MSE** is the mean squared error
- **h_{ii}** is the **i^{th} leverage value**

Essentially Cook's distance measures how much all of the fitted values in the model change when the i^{th} observation is deleted.

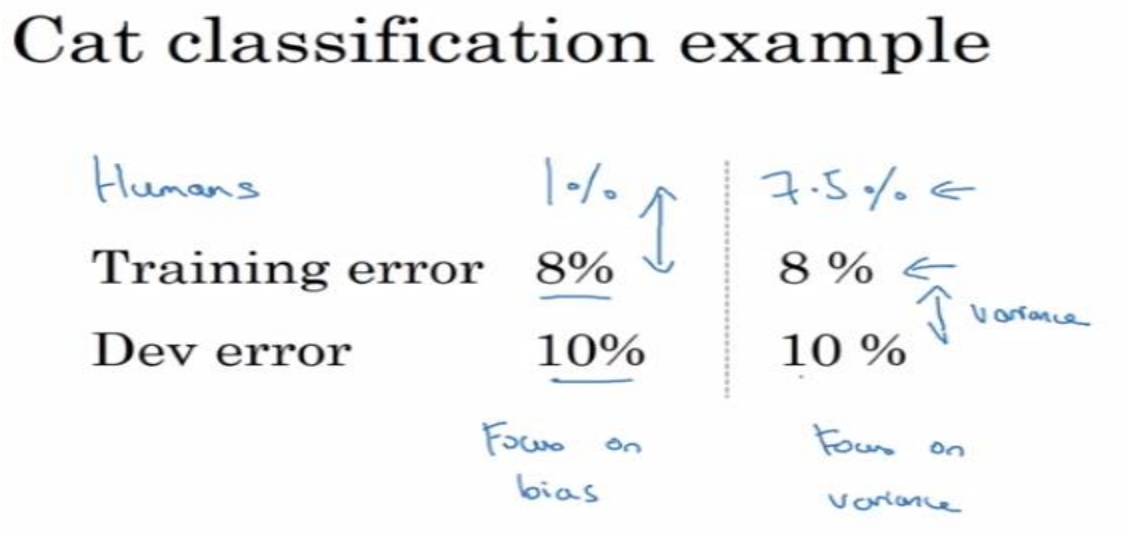
*** **Cook's distance** of more than 3 times the mean were considered outliers

Key Things to remember building in a ML model:

- **Cook's distance** of more than **3 times** the mean were **considered outliers**.
- **Values for** VIF (Variance Inflation Factor) **exceeding 10** were regarded as indicating **multicollinearity**.
- One hot encoding (or label encoding) was used to handle categorical data.
- **Numerical data was** scaled/normalized **to ensure all features are on the same scale.**
- An 80:20 train test split was done to ensure there is no data leakage.
- **Usage of the** t-SNE plot **to see a visible separation between classes**

Avoidable Bias:

https://www.coursera.org/learn/machine-learning-projects/lecture/4IPD6/improving-your-model-performance



To avoid bias:

Machine Learning theoretical concepts

By: Vikram Pal

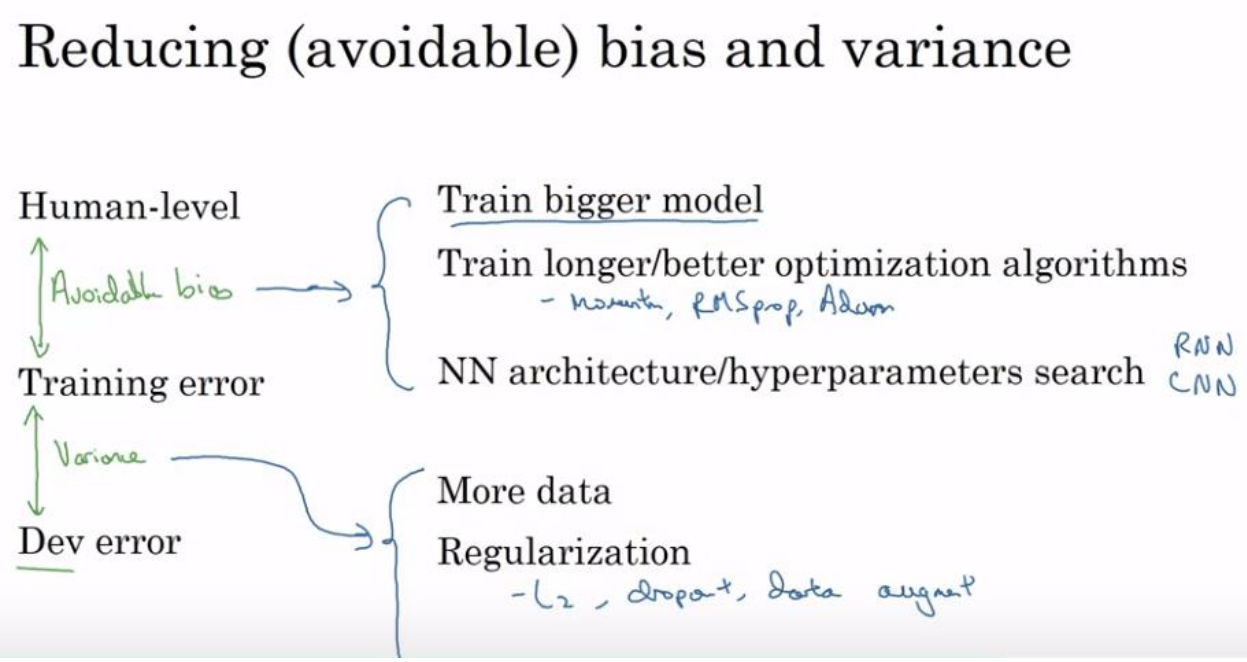
1. Train bigger model

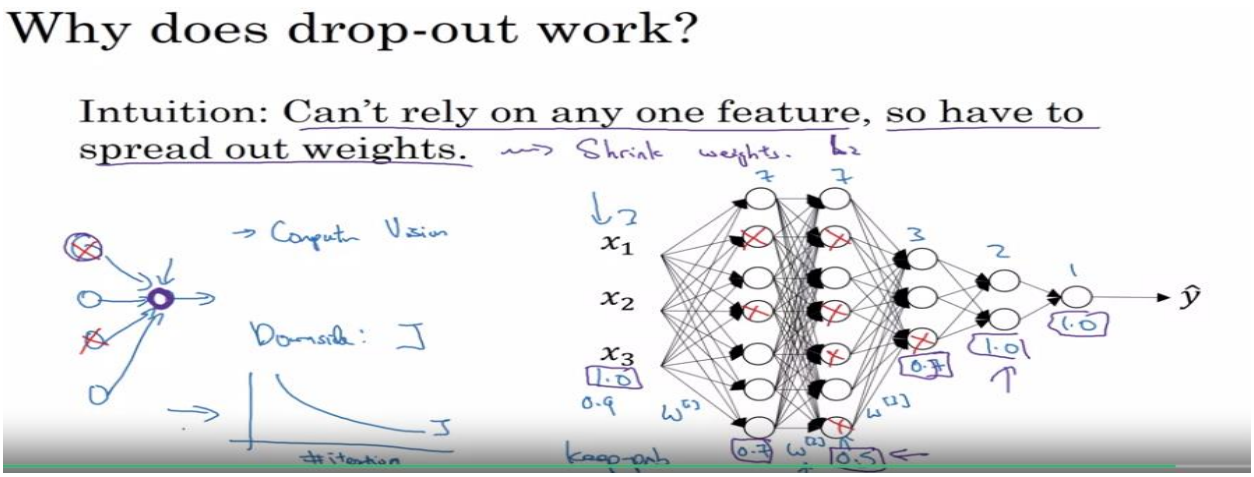
2. Train longer/better optimization algorithms(Momentum, RMSprop and Adam)

3. NN architecture/ Hyperparameters Search (RNN and CNN)

To avoid Variance:

1. More data 2. Regularization (l2) 3. Dropout 4. Data augment





Machine Learning theoretical concepts

By: Vikram Pal

Working with sparse data sets:

Scipy sparse matrices:

Scipy package offers several types of sparse matrices for efficient storage. Sklearn and other machine learning packages such as imblearn accept sparse matrices as input. Therefore, when working with large sparse data sets, it is highly recommended to **convert our panda's data frame into a sparse matrix before passing it to sklearn**.

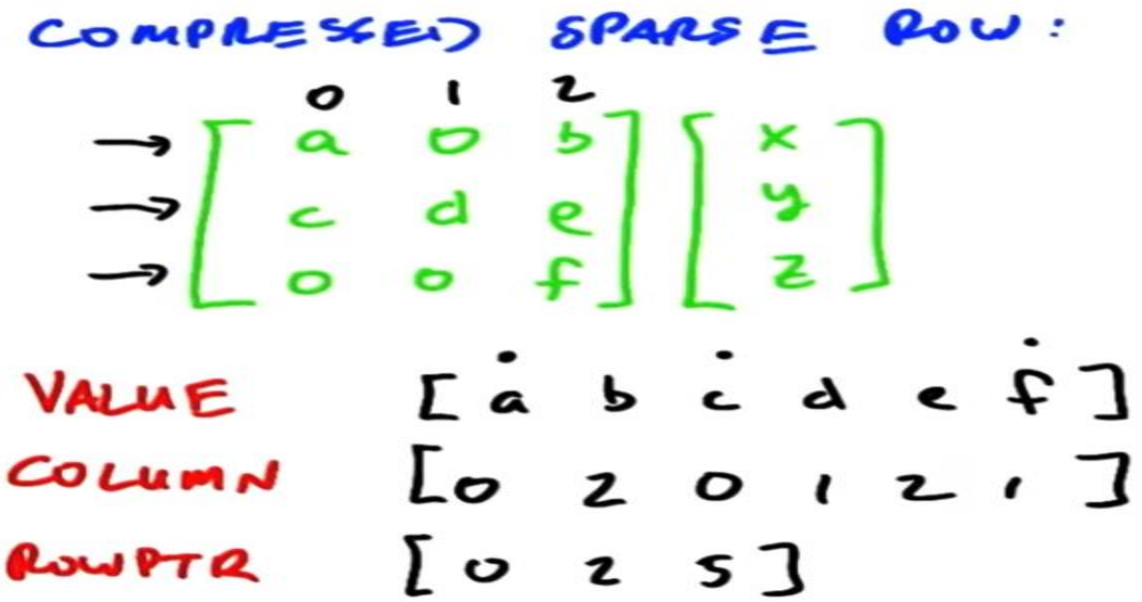
Sklearn sparse matrix type:

`scipy.sparse.csr_matrix`: compressed sparse row matrix, three separate array

`scipy.sparse.csc_matrix` : compressed sparse column matrix, three separate array

`scipy.sparse.coo_matrix` : coordinate matrix, with three separate array

`scipy.sparse.dok_matrix` : This is a dictionary of keys matrix



Methods for dealing with sparse features

1. Removing features from the model:

Sparse features can introduce noise, which the model picks up and **increase the memory** needs of the model. To remedy this, they can be dropped from the model.

2. Make the features dense:

Principal component analysis (PCA) - PCA methods can be used to project the features into the directions of the principal components and select from the most important components.

Feature hashing - In feature hashing, sparse features can be binned into the desired number of output features using a hash function.

Machine Learning theoretical concepts

By: Vikram Pal

3. Using models that are robust to sparse features

Some versions of machine learning models are robust towards sparse data and may be used instead of changing the dimensionality of the data. For example, **the entropy-weighted k-means algorithm is better suited to this problem than the regular k-means algorithm**.

For example, linear models with L1 regularization (such as Lasso regression) are particularly good at automatically selecting the most important features from a large number of sparse features, and can be more efficient and accurate than models that do not take sparsity into account.

On the other hand, decision trees and random forests are generally good at handling sparse data, but may be less efficient when dealing with large datasets.

There are several types of machine learning models that are known to be robust to sparse features, including:

- Linear models with L1 regularization, such as Lasso regression
- Decision trees and random forests
- k-nearest neighbors (k-NN)
- Support vector machines (SVMs)
- Entropy-weighted k-means clustering

Continuous numeric features:

The problem of working with raw, continuous numeric features is that often **the distribution of values in these features will be skewed**. This signifies that some values will occur quite frequently while some will be quite rare. Besides this, there is also another **problem of the varying range of values in any of these features.**

Using appropriate models

Some machine learning models are better suited to handling continuous numeric features than others. For example, linear models (such as linear regression) and tree-based models (such as decision trees and random forests) are often good at dealing with continuous numeric features, while models that rely on distance measures (such as k-nearest neighbors) may be less effective

Binning:

It is also known as quantization is used for transforming **continuous numeric features into discrete ones (categories).** These discrete values or numbers can be thought of as categories or bins into which the raw, continuous numeric values are binned or grouped into.

Fixed-Width Binning

Just like the name indicates, in fixed-width binning, we have specific fixed widths for each of the bins which are usually pre-defined by the user analyzing the data. Each bin has a pre-fixed range of values which should be assigned to that bin on the basis of some domain knowledge, rules or constraints.

Machine Learning theoretical concepts

By: Vikram Pal

```python
# First, we'll import the necessary libraries
import pandas as pd

# Next, we'll create a sample dataset with a continuous numeric feature
df = pd.DataFrame({'value': [1.5, 2.3, 3.7, 4.8, 5.2, 6.1, 7.9, 8.2, 9.5]})

# Now, we'll define the bins and assign the values to the appropriate bin
bins = [1, 3, 6, 9]
df['bin'] = pd.cut(df['value'], bins)
print(df)

# Output:
#   value    bin
# 0   1.5  (1, 3]
# 1   2.3  (1, 3]
# 2   3.7  (3, 6]
# 3   4.8  (3, 6]
# 4   5.2  (3, 6]
# 5   6.1  (6, 9]
# 6   7.9  (6, 9]
# 7   8.2  (6, 9]
# 8   9.5  (6, 9]
```

```
Age Range: Bin
---------------
 0 -  9   : 0
10 - 19   : 1
20 - 29   : 2
30 - 39   : 3
40 - 49   : 4
50 - 59   : 5
60 - 69   : 6
  ... and so on
```

Adaptive Binning

The drawback in using fixed-width binning is that due to us manually deciding the bin ranges, we can end up with **irregular bins which are not uniform based on the number of data points** or values which fall in each bin. Some of the bins might be densely populated and some of them might be sparsely populated or even empty! Adaptive binning is a safer strategy in these scenarios where we let the data speak for itself! That's right, we use the data distribution itself to decide our bin range.

Adaptive binning

- Dynamic binning
- Optimal binning

```
# We'll use the `qcut()` function from the `pandas` library to perform dynamic
binning
# We'll specify the number of bins and the method for determining the bin widths
df['bin'] = pd.qcut(df['value'], q=4, duplicates='drop', precision=0)
print(df)

# Output:
#   value          bin
# 0   1.5  (1.499, 3.0]
# 1   2.3  (1.499, 3.0]
# 2   3.7  (3.0, 5.0]
# 3   4.8  (5.0, 8.0]
# 4   5.2  (5.0, 8.0]
# 5   6.1  (8.0, 9.5]
# 6   7.9  (8.0, 9.5]
# 7   8.2  (8.0, 9.5]
# 8   9.5  (8.0, 9.5]
```

Regenerate response

Quantile based binning is a good strategy to use for adaptive binning. Quantiles are specific values or cut points which help in **partitioning the continuous valued distribution of a specific numeric field into discrete contiguous bins or intervals.**

```
Output
------
0.00      6000.0
0.25     20000.0
0.50     37000.0
0.75     60000.0
1.00    200000.0
Name: Income, dtype: float64
```

Statistical Transformations:

We talked about the adverse effects of skewed data distributions briefly earlier. Let's look at a different strategy of feature engineering now by making use of statistical or mathematical transformations. We will look at the **Log transform as well as the Box-Cox transform**. Both of these transform functions belong to the Power Transform family of functions, typically used to create monotonic data transformations. **Their main significance is that they help in stabilizing variance**, adhering closely to **the normal distribution and making the data independent of the mean based on its distribution**

Log Transform

The log transform belongs to the power transform family of functions. This function can be mathematically represented as

$$y = \log_b(x)$$

Machine Learning theoretical concepts

By: Vikram Pal

Box-Cox Transform

The Box-Cox transform is another popular function belonging to the power transform family of functions. This function has a pre-requisite that the numeric values to be transformed must be positive (similar to what log transform expects). In case they are negative, shifting using a constant value helps. Mathematically, the Box-Cox transform function can be denoted as follows.

$$y = f(x,\lambda) = x^{\lambda} = \begin{cases} \dfrac{x^{\lambda}-1}{\lambda} & for\ \lambda > 0 \\ \log_e(x) & for\ \lambda = 0 \end{cases}$$

Different Types of Distance Metrics used in Machine Learning:

- Minkowski distance

  $$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

  Some common values of 'p' are:-

  p = 1, Manhattan Distance

  p = 2, Euclidean Distance

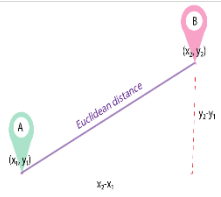
  p = infinity, Chebychev Distance

- Manhattan distance

  $$d = \sum_{i=1}^{n} |x_i - y_i|$$

  Thus, Manhattan Distance is preferred over the Euclidean distance metric as the dimension of the data increases. This occurs due to something known as the 'curse of dimensionality'.

- Euclidean distance

  $$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Hamming distance

  Hamming distance is used to measure the distance between categorical variables.
  Xor Operation:

  Suppose there are two strings 11011001 and 10011101.

11011001 ⊕ 10011101 = 01000100. Since this contains two 1s, the Hamming distance, d(11011001, 10011101) = 2.

- Cosine distance: find similarities between two data points

Regression & Classification:

R^2 coefficient of determination

Residual Sum of Squares (RSS):

The residual sum of squares (RSS) is a statistical technique used to measure the **amount of variance in a data set that is not explained by a regression model itself**. Instead, it estimates the variance in the residuals, or error term.

Multiple Regression/Polynomial Regression:

R2 vs Adjusted R2:

R-squared:

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.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

Y^ predicted
y- mean of predictions

$$\mathrm{R}^2 = 1 - \frac{\text{Unexplained Variation}}{\text{Total Variation}}$$

R^2 Coefficient of determination

In statistics, the coefficient of determination, denoted R^2 or r^2 and pronounced "R squared", is the

proportion of the variation in the dependent variable that is predictable from the independent

$$R^2 = 1 - \frac{RSS}{TSS}$$

RSS = sum of squares of residuals

Machine Learning theoretical concepts

By: Vikram Pal

TSS = total sum of squares

Adjusted R2:

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.

$$R_a^2 = 1 - \left[\left(\frac{n-1}{n-k-1}\right) \times (1 - R^2)\right]$$

where:
n = number of observations
k = number of independent variables
R_a^2 = adjusted R^2

Lasso(L1) [sparsity]and Ridge(L2)[simplicity]:

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

Lasso Regression(L1):

The cost function for Lasso (**least absolute** shrinkage and selection operator) regression can be written as (L1 regularization)

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} |w_j| \qquad (1.4)$$

It is also called **regularization for sparsity**. As the name suggests, it is used to handle sparse vectors which consist of mostly zeroes. **Sparse vectors typically result in very high-dimensional feature vector space. Thus, the model becomes very difficult to handle**.

L1 regularization forces the weights of uninformative features to be zero by subtracting a small amount from the weight at each iteration and thus making the weight zero, eventually.

L1 regularization penalizes |weight|.

Ridge Regression(L2):

In ridge regression, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients. (L2 regularization)

Machine Learning theoretical concepts

By: Vikram Pal

$$\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j \times x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} w_j^2 \qquad (1.3)$$

It is also called **regularization for simplicity**. If we take the model complexity as a function of weights, the **complexity of a feature is proportional to the absolute value of its weight**.

$$y = w_1 x_1 + w_1 x_1 + \ldots + w_n x_n$$

$$L2\ regularization\ terms = w_1^2 + w_2^2 + \ldots + w_n^2$$

L2 regularization forces weights toward zero but it does not make them exactly zero. L2 regularization acts like a force that removes a small percentage of weights at each iteration. Therefore, weights will never be equal to zero.

L2 regularization penalizes $(weight)^2$

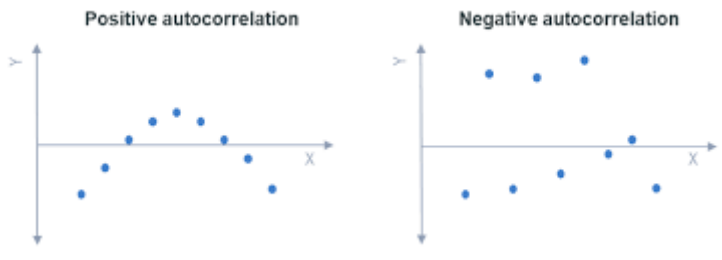
There is an additional parameter to tune the L2 regularization term which is called **regularization rate** (lambda). Regularization rate is a scalar and multiplied by L2 regularization term.

Linear Regression:

The regression has five key assumptions:

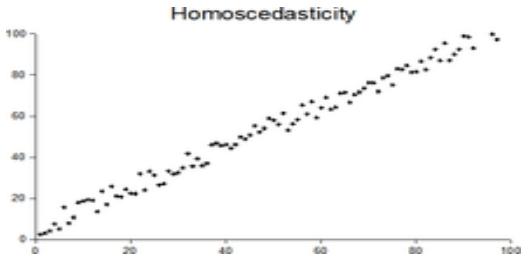
a. Linear relationship b. Multivariate normality c. No or little multicollinearity(**residuals are normally distributed**) d. No-autocorrelation e. Homoscedasticity (**error term is the same across all values)**

- **Linear relationship**: linear relationship between the features and target
- **Multivariate normality:** Multiple regression assumes that the **residuals are normally distributed**.
- **No or little multicollinearity:** Multicollinearity is a state of **very high inter-correlations** or inter-associations among the independent variables
  ➔ Pair Plot and heatmap for correlation.
- **No autocorrelation**: No autocorrelation refers to a situation in which **no identifiable relationship exists between the values of the error term**



-

- **Homoscedasticity:** Homoscedasticity describes a situation in which the **error term is the same across all values** of the independent variables**.**



Plot with random data showing homoscedasticity: at each value of *x*, the *y*-value of the dots has about the same variance.

Evaluation Matrices for a regression model:

- Mean Absolute Error(MAE)
- Mean Squared Error(MSE)
- Root Mean Squared Error(RMSE)
- R-Squared (Coefficient of Determination)

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

- Adjusted R-Squared

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(n-1)}{n-k-1}\right]$$

K= number of independent variables
N= number of observations

MSE is a **differentiable function that makes it easy to perform mathematical operations in comparison to a non-differentiable function** like MAE.

Which evaluation metric should you prefer to use for a dataset having a lot of outliers in it?

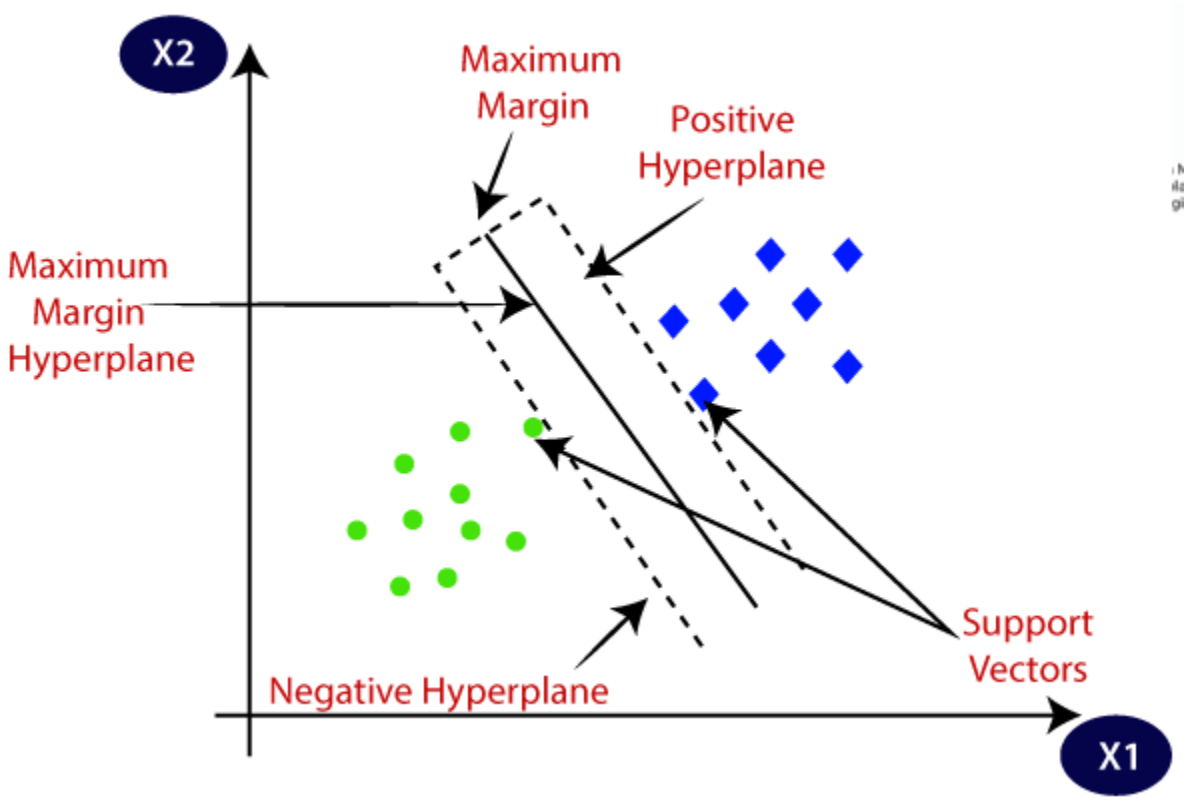
Mean Absolute Error(MAE) is preferred when we have too many outliers present in the dataset because MAE is robust to outliers whereas MSE and RMSE are very susceptible to outliers and this start penalizing the outliers by squaring the error terms, commonly known as residuals.

$$\mathbf{MAE} = \frac{1}{n}\sum_{i=1}^{n} |x_i - x| \quad \mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

Machine Learning theoretical concepts

By: Vikram Pal

SVM: (kernel tricks): -



- Gamma: kernel coefficient small gamma gives less complexity and large gamma give more complexity
- C parameter: regularization parameter, it tells us how much you want to penalize the misclassified error
- Trade off: **width of the margin vs no of error** committed by the linear decision boundary.
- **Linearly Separable (soft margin) [underfitting] and Non-linearly Separable** (hard margin) [overfitting] data.
- Number of dimensions are greater than the number of samples

C is the cost of misclassification as correctly stated by Dima.

A large C gives you low bias and high variance. Low bias because you penalize the cost of missclasification a lot.
A small C gives you higher bias and lower variance.

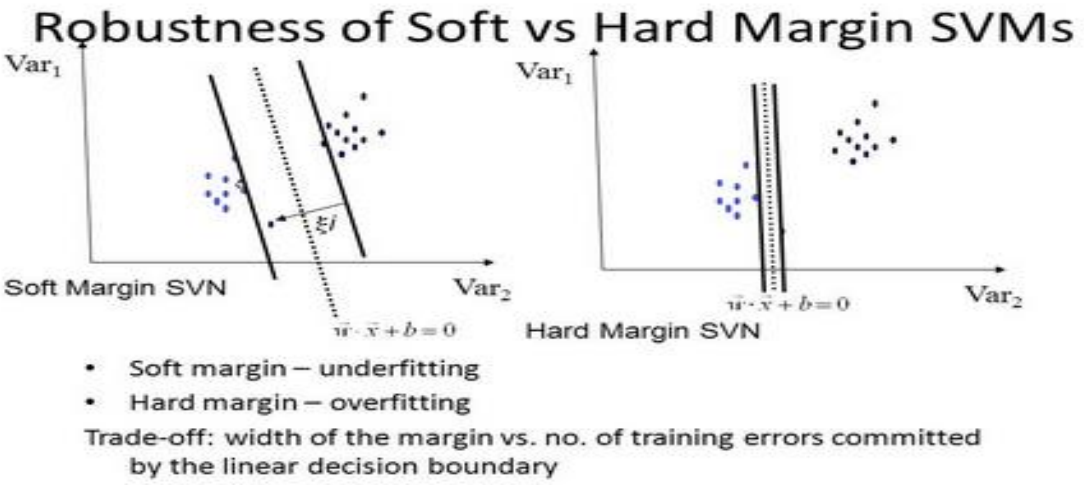
Gamma is the parameter of a Gaussian Kernel (to handle non-linear classification). Check this points:

They are not linearly separable in 2D so you want to transform them to a higher dimension where they will be linearly sepparable. Imagine "raising" the green points, then you can sepparate them from the red points with a plane (hyperplane)

To "raise" the points you use the RBF kernel, gamma controls the shape of the "peaks" where you raise the points. A small gamma gives you a pointed bump in the higher dimensions, a large gamma gives you a softer, broader bump.

So a small gamma will give you low bias and high variance while a large gamma will give you higher bias and low variance.

You usually find the best C and Gamma hyper-parameters using Grid-Search.



To meet the soft margin objective, we need to introduce a slack variable ε>=0 for each sample; it measures how much any instance is allowed to violate the margin.

Kernel: The function used to map a **lower dimensional data into a higher dimensional data**.
Kernel — It specifies the kernel type to be used. There are different kernel options such as linear, radial basis function (RBF), polynomial and sigmoid. Here “rbf” and “poly” are useful for non-linear hyper-plane.

SVM Regression:
it tries to fit as many instances as possible between the margin while limiting the margin violations.
SVM Classification:
fit the largest possible street between two classes.

Machine Learning theoretical concepts

By: Vikram Pal

Loss function

We'll use the Hinge loss. This is a loss function used for training classifiers. **The hinge loss is used for "maximum-margin" classification,** most notably for support vector machines (SVMs).

$$\mathcal{L}(Y, \dot{Y}) = \frac{1}{N} \sum_{n=1}^{N} \max(0, 1 - \dot{y}_n \cdot y_n)$$

c is the loss function, x the sample, y is the true label, f(x) the predicted label.

This means the following: $c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$

Objective Function

$$\underbrace{min \ \|w\|^2}_{\text{regularization term}} + \underbrace{C \sum_{i=1}^{l} max\left(0, 1 - y_i f(x_i)\right)}_{\text{loss function term}}$$

https://stackoverflow.com/questions/53244095/hinge-loss-function-gradient-w-r-t-input-prediction

As you can see, our objective of a SVM consists of two terms. **The first term is a regularizer, the heart of the SVM**, the **second term the loss**. **The regularizer balances between margin maximization and loss**. We want to find the decision surface that is maximally far away from any data points.

How do we minimize our loss/optimize for our objective (i.e learn)?

We have to derive our objective function to get the gradients! Gradient descent ftw. As we have two terms, we will derive them separately using the sum rule in differentiation.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

This means, **if we have a misclassified sample, we update the weight vector w using the gradients of both terms**, else if **classified correctly, we just update w by the gradient of the regularizer.**

Misclassification condition

$$y_i \langle x_i , w \rangle < 1$$

Update rule for our weights (misclassified)

$$w = w + \eta(y_i x_i - 2\lambda w)$$

including the learning rate η and the regularizer λ the learning rate is the length of the steps the algorithm makes down the gradient on the error curve.

- Learning rate too high? The algorithm might overshoot the optimal point.
- Learning rate too low? Could take too long to converge. Or never converge.

The regularizer controls the trade off between the achieving a low training error and a low testing error that is the **ability to generalize your classifier to unseen data.** As a regulizing parameter we choose 1/epochs, so this parameter will decrease, as the number of epochs increases.

- Regularizer too high? overfit (large testing error)
- Regularizer too low? underfit (large training error)

Update rule for our weights (correctly classified)

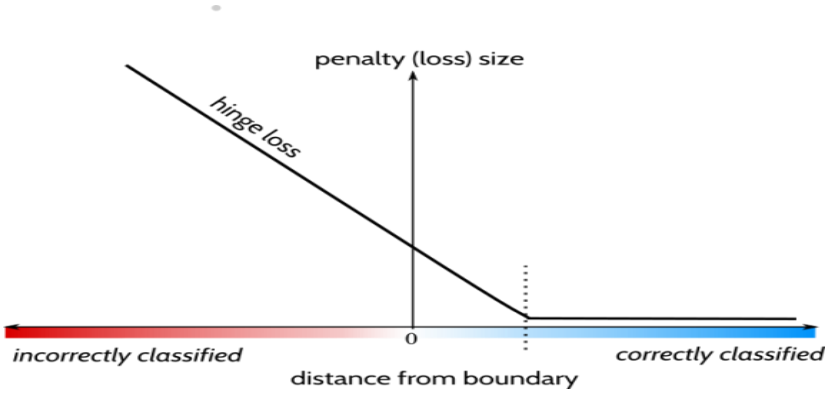
$$w = w + \eta(-2\lambda w)$$

Gamma — It is the **kernel coefficient** for the **'rbf', 'poly' and 'sigmoid'**. Small Gamma (less variance) gives less complexity and larger gamma(more variance) gives more complexity.

C — It is the **regularization parameter**. It allowed you to decide how much you want to penalize the misclassified points

Machine Learning theoretical concepts

By: Vikram Pal



Gamma vs C parameter

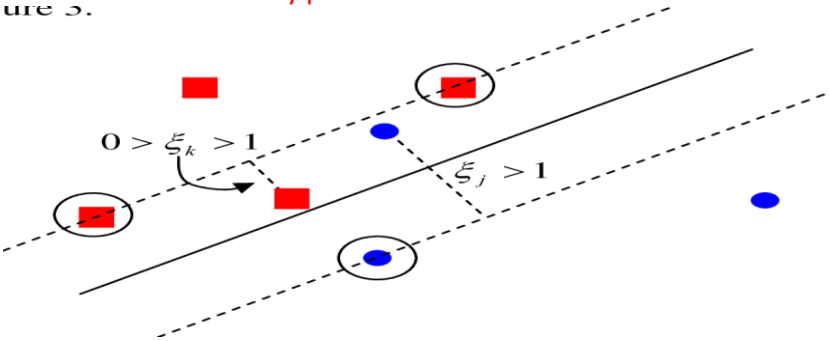
For a linear kernel, we just need to optimize the c parameter. However, if we want to use an RBF kernel, both c and gamma parameters need to optimize simultaneously. If gamma is large, the effect of c becomes negligible. If gamma is small, c affects the model just like how it affects a linear model. Typical values for c and gamma are as follows. However, specific optimal values may exist depending on the application:

For RBF:
If gamma large, effect of c become negligible
If gamma small, c affects the model just like how it affects a linear model

0.0001 < gamma < 10
0.1 < c < 100

Slack Variable:

To meet the soft margin objective, we need to introduce a slack variable ε>=0 for each sample; it measures how much any particular instance is allowed to violate the margin.



Popular kernel

1. Gaussian Redial basis function

It is one of the most popular kernels used in SVM. It is used when there is no prior knowledge about data.

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Gaussians redial basis function

2.Gaussian function

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)}$$

Machine Learning theoretical concepts

By: Vikram Pal

3.Polynomial Kernel Function
It is written as,

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + a)^b$$

4.Linear Kernel

It is just the normal dot product,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$:

Small comparison of all the above kernel,

***time of SVM learning**: linear < poly < rbf*
ability to fit any data: linear < poly < rbf
risk of overfitting: linear < poly < rbf
risk of underfitting: rbf < poly < linear
number of hyperparameters: linear (0) < rbf (2) < poly (3)

Pros:

- It is useful for both **linearly Separable (hard margin) and Non-linearly Separable** (soft margin) data.
- It is effective **in high dimensional spaces**.
- It is effective in cases where **several dimensions are greater than the number of samples**.
- It uses a **subset of training points** in the decision function (called support vectors), so it is also memory efficient.
- **Outliers do not impact the SVM function.**

Cons:

- **Picking the right kernel** and parameters can be computationally intensive.
- It also doesn't perform very well, when the **data set has more noise** i.e., target classes are overlapping
- SVM doesn't directly provide probability estimates, these are calculated using an **expensive five-fold cross-validation.**

KNN:(similar neighbors for the new data point.) classification(count) and Regression(Average) K larger underfitting and K small overfitting

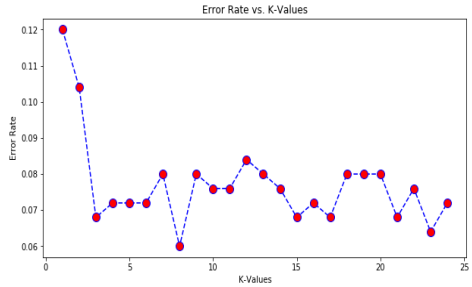
(Hyperparameters: {**n_neighbors*int, default=5},{*weights*{'uniform', 'distance'} or callable, default='uniform'},algorithm{'auto', {'ball_tree', 'kd_tree', 'brute'}, default='auto'})**

K is a **number used to identify similar neighbors** for the new data point.
How to choose the value of K?

Machine Learning theoretical concepts

By: Vikram Pal

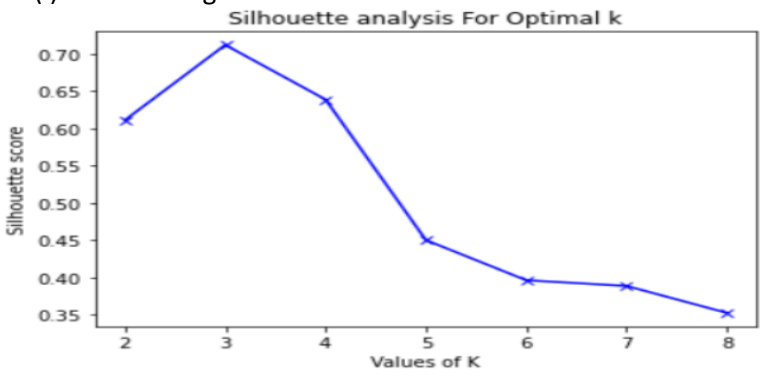
Derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate.



If there too many elbows, then It would be hard to pick a optimal number of K. In that situation, we use Silhouette analysis.
The silhouette coefficient is a measure of how similar a data point is within-cluster (cohesion) compared to other clusters (separation).
The equation for calculating the silhouette coefficient for a particular data point:

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

a. S(i) is the silhouette coefficient of the data point i.
b. a(i) is the average distance between i and all the other data points in **the cluster to which i belongs.**
c. b(i) is the average distance from i to all clusters to **which i does not belong**.



Line plot between K and Silhouette score

Supervised machine learning algorithm as target variable is known.

Having small K value causes overfitting. Because small k value causes noise to have higher influence on Result.

Having large K value leads to underfitting. Because having k large value, the model won't be able to generalize.

KNN can be used for value imputation in both Categorical and Continuous categories **of data. It is the only algorithm that can achieve this.**

The KNN algorithm doesn't learn anything from the training data, but rather it just store the training data at the time of training.

- Nonparametric as **it does not make an assumption** about the underlying data distribution pattern.

- **Lazy algorithm as KNN does not have a training step**. All data points will be used only at the time of prediction. With no training step, prediction step is costly. An eager learner algorithm eagerly learns during the training step.
- Used for both Classification and Regression.
- Uses **feature similarity to predict** the cluster that the new point will fall into.

Note: - How to pick K value: **by plotting accuracy rate or F1 score against different values of K**.

For classification(Count), **count the number of data points in each category** among the k neighbors. **New data point will belong to class that has the most neighbors**.

For regression(average), value for the **new data point will be the average of the k neighbors**.

- **Euclidean distance**
- **Manhattan distance**
- **Hamming Distance**
- **Minkowski Distance**

Pros of K Nearest Neighbors

- Simple algorithm and hence easy to interpret the prediction.
- Nonparametric, so makes no assumption about the underlying data pattern.
- Used for both classification and Regression.
- Training step is **much faster for nearest neighbor** compared to another machine learning algorithms

Cons of K Nearest Neighbors

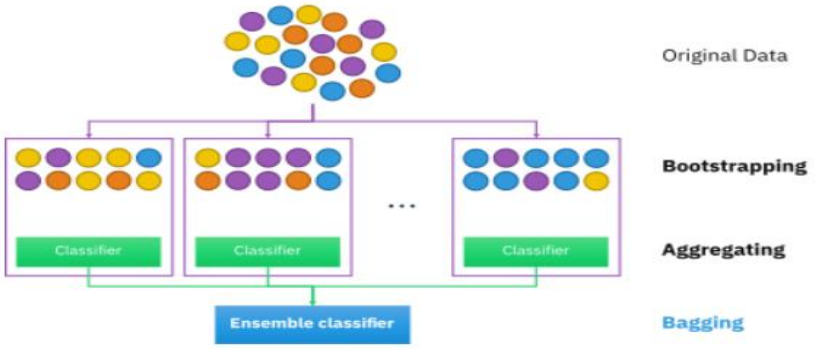
- KNN is computationally expensive as it searches the nearest neighbors for the new point at the prediction stage.
- High memory requirement as KNN must store all the data points.
- Prediction stage is very costly.
- **Sensitive to outliers, accuracy is impacted by noise or irrelevant data. Cons of K Nearest Neighbors**

Random Forest:

(Hyperparameters: {**n_estimatorsint, default=100}, {criterion{"gini", "entropy"}, default="gini"},{ max_features{"auto", "sqrt", "log2"}, int or float, default="auto"**})
Random forest works on the **Bagging principle.**

Machine Learning theoretical concepts

By: Vikram Pal



It 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)

- **Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents overfitting.**

Feature and Advantages of Random Forest:

- It is one of the most **accurate learning algorithms** available. For many data sets, it produces a highly accurate classifier.
- It generates an **internal unbiased estimate of the generalization error** as the forest building progresses.
- It has an **effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.**

Disadvantages of Random Forest:

- Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
- **For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels**. Therefore, the variable importance scores from random forest are not reliable for this type of data.

Proximities: are calculated **for each pair of cases/observations/sample points**. If **two cases occupy the same terminal node through one tree, their proximity is increased by one**. At the end of the run of all trees, the proximities are normalized by dividing by the number of trees. Proximities are used in replacing **missing data, locating outliers, and producing illuminating low-dimensional views of the data**.

Decision Tree:

(Hyperparameters: criterion{“Gini”, “entropy”}, splitter{“best”, “random”}, max_features int, float or {“auto”, “sqrt”, “log2”})

Regression: Impurity metric that is suitable for continuous variables, so we define the **impurity measure using the weighted mean squared error (MSE)** of the children’s nodes instead

Classification: - In classification, **entropy is the most common impurity measure or splitting criteria.**

Machine Learning theoretical concepts

By: Vikram Pal

What causes overfitting in decision tree?

In decision trees, over-fitting occurs when the tree is designed to perfectly **fit all samples in the training data set.** Thus, it ends up with branches with **strict rules of sparse data**. Thus, this effects the accuracy when predicting samples that are not part of the training set.

How do you fix overfitting in decision tree?

Pre-pruning that stops growing the tree earlier, before it perfectly classifies the training set.
Post-pruning that allows the tree to perfectly classify the training set, and then post prune the tree.

Advantage:

- Easy to understand and interpret, perfect for visual representation.
- Can work with numerical and categorical features.
- Decision Trees **are not sensitive to noisy data or outliers** since, extreme values or outliers, never cause much reduction in Residual Sum of Squares(RSS), because they are never involved in the split.

Disadvantage:

- main drawback of Decision Tree is that it generally leads to overfitting of the data.
- Information gain is defined as the reduction in entropy due to the selection of a particular attribute. Information gain biases the Decision Tree against considering attributes with **a large number of distinct values which might lead to overfitting**.

Gini Impurity:

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

Entropy:

$$Entropy(D_1) = -\sum_{i=1}^{m} p_i \log_2 p_i$$

Information Gain:

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{Entropy\ (before) - \sum_{j=1}^{K} Entropy(j, after)}{\sum_{j=1}^{K} w_j \log_2 w_j}$$

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 assumes as all the variables in the dataset is “Naive” i.e., not correlated to each other.
What are the Pros and Cons of Naive Bayes?
Pros:

- It is **easy and fast to predict class of test data set**.

- It also performs well in **multi class prediction When assumption of independence holds**, a Naive Bayes classifier performs better compared to other models like logistic regression and you need less training data.
- It performs well in **case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).**

Cons:

- **If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0** (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, **we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.**
- On the other side naive Bayes is also **known as a bad estimator, so the probability outputs** from predict_proba are not to be taken too seriously.
- Another limitation of **Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible** that we get a set of predictors which are completely independent.

Boosting:

https://www.youtube.com/watch?v=kho6oANGu_A

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

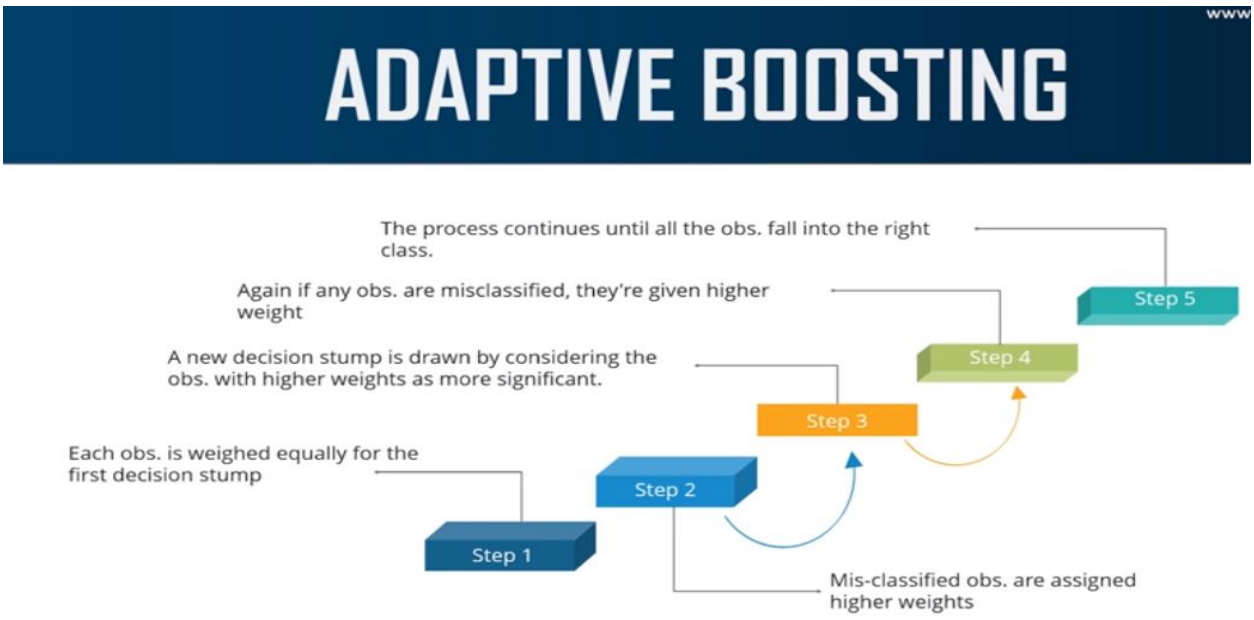
Machine Learning theoretical concepts

By: Vikram Pal

Unlike CatBoost or LGBM, XGBoost **cannot handle categorical features by itself**, it only accepts numerical values similar to Random Forest. Therefore, one must perform various encodings like label encoding, mean encoding or one-hot encoding before supplying categorical data to XGBoost.

Catboost offers a new technique called **Minimal Variance Sampling (MVS), which is a weighted sampling version of Stochastic Gradient Boosting**. ... XGboost is not using any weighted sampling techniques, which makes its splitting process slower compared to GOSS and MVS

AdaBoost	GradientBoost
Both AdaBoost and Gradient Boost use a base weak learner and they try to boost the performance of a weak learner by iteratively shifting the focus towards problematic observations that were difficult to predict. At the end, a strong learner is formed by addition (or weighted addition) of the weak learners.	
In AdaBoost, shift is done by up-weighting observations that were misclassified before.	Gradient boost identifies difficult observations by large residuals computed in the previous iterations.
In AdaBoost "shortcomings" are identified by high-weight data points.	In Gradientboost "shortcomings" are identified by gradients.
Exponential loss of AdaBoost gives more weights for those samples fitted worse.	Gradient boost further dissect error components to bring in more explanation.
AdaBoost is considered as a special case of Gradient boost in terms of loss function, in which exponential losses.	Concepts of gradients are more general in nature.



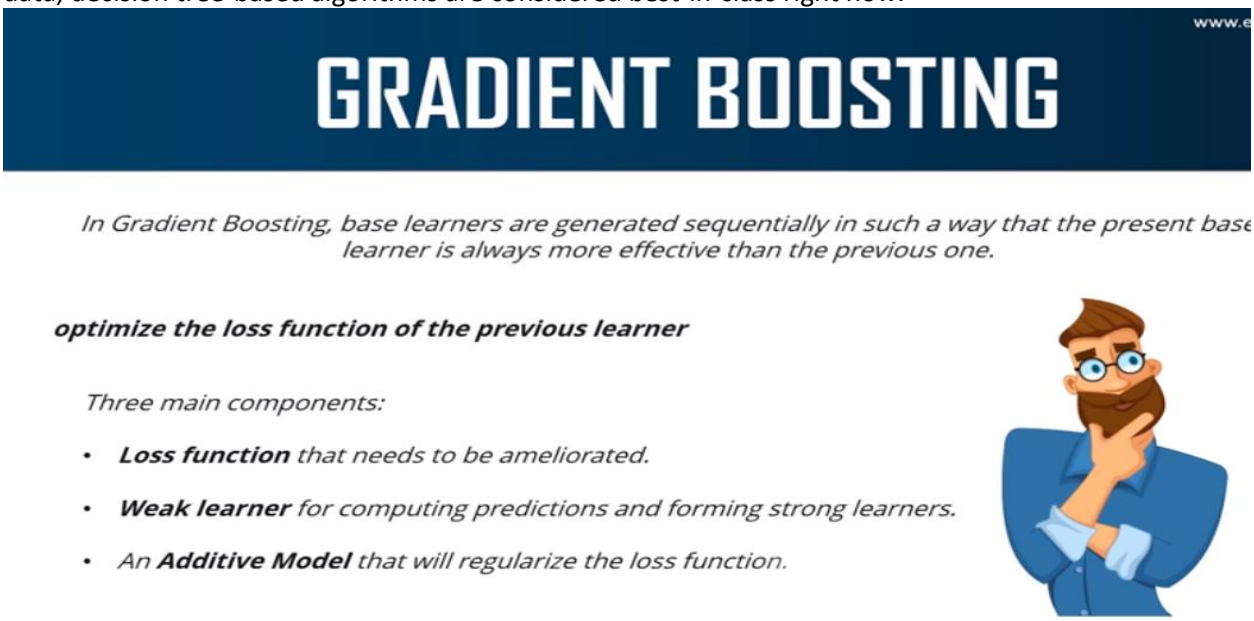
Machine Learning theoretical concepts

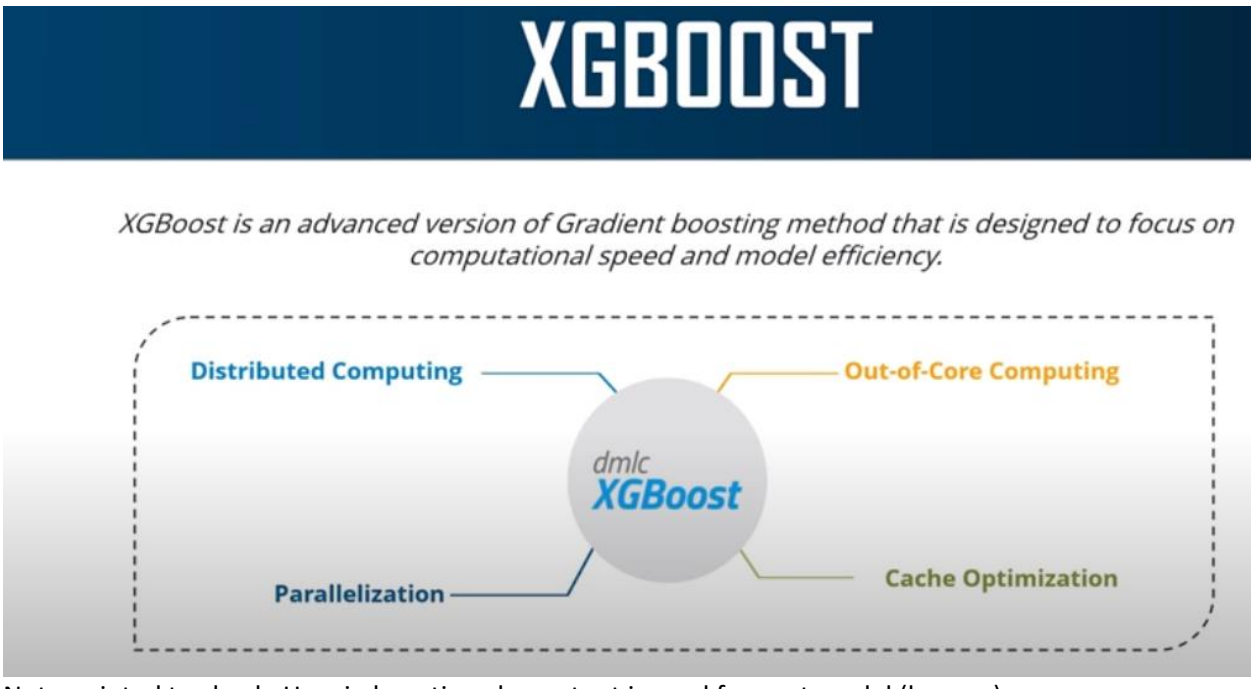
By: Vikram Pal

XGboots:

https://www.youtube.com/watch?v=TyvYZ26alZs

XGBoost is a **decision-tree-based ensemble Machine Learning algorithm** that uses a **gradient boosting framework** (Bagging In Random Forest). when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.





Note pointed to check: How in boosting algo output is used for next model (learner)

We add second tree to first tree such a way that it decreases the loss compared to first alone.

Machine Learning theoretical concepts

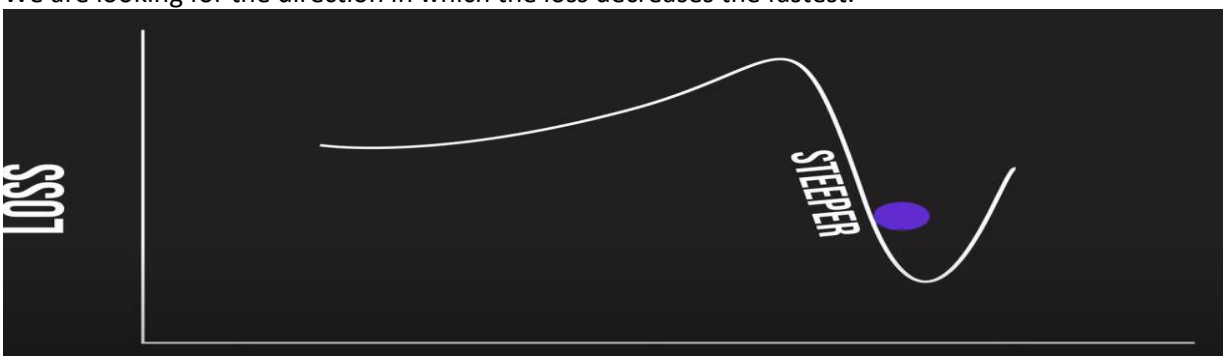
By: Vikram Pal

$$\text{Boosted Ensemble} = \text{First Tree} + \eta * \text{Second Tree}$$

$$\text{Loss(Boosted Ensemble)} < \text{Loss(First Tree)}$$

η = *Learning Rate*

We are looking for the direction in which the loss decreases the fastest.



That is mathematically written as

$$-\frac{\partial \text{ Loss Function}}{\partial \text{ Previous}}$$

$$F(2) = F(1) + \eta * \text{Second Tree}$$

$$\text{Second Tree} = -\frac{\partial L}{\partial F(1)} = -\frac{\partial \text{ Loss Function}}{\partial \text{ Previous Model's Output}}$$

Machine Learning theoretical concepts

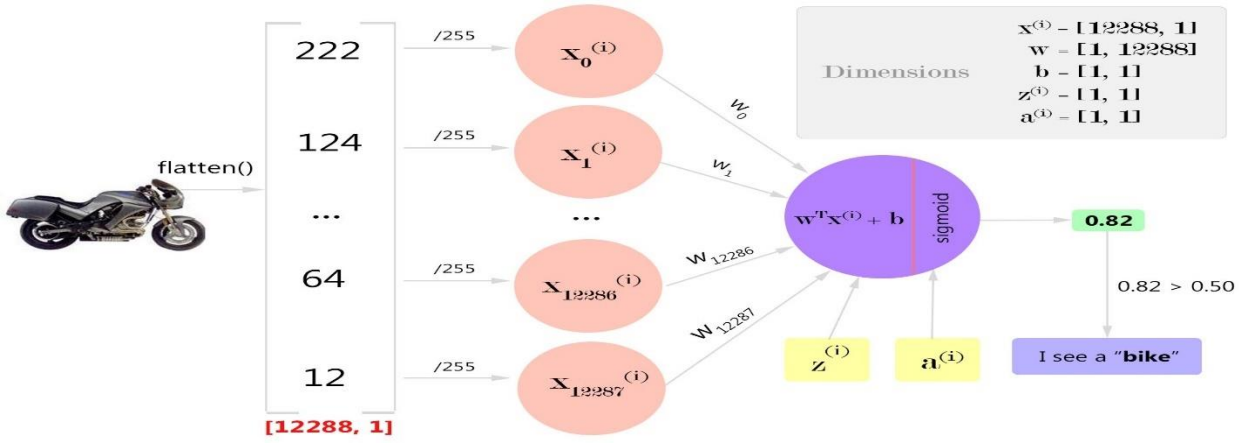
By: Vikram Pal

$$F(m) = F(m-1) + \eta * - \frac{\partial L}{\partial F(m-1)}$$

Logistic Regression:

It's a classification algorithm that is used where the **target variable is of categorical nature**.

The main objective behind Logistic Regression is to determine the **relationship between features and the probability of a particular outcome.**



Giradient

$$z = w_1 x_1 + w_2 x_2 + b \rightarrow \hat{y} = a = \sigma(z) \rightarrow L(\hat{y}, y)$$

$$\Leftrightarrow a = \hat{y}$$

$$w_1 \Rightarrow \frac{\partial(L)}{\partial w_1} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial(z)}{\partial w_1}$$

$$\frac{\partial L}{\partial a} = \frac{\partial}{\partial a}(-y \log a - (1-y)\log(1-a))$$

$$= -y\left(\frac{1}{a}\right) - (-1)\frac{(1-y)}{(1-a)}$$

$$\frac{\partial L}{\partial a} = \left(\frac{-y}{a}\right) + \left(\frac{1-y}{1-a}\right)$$

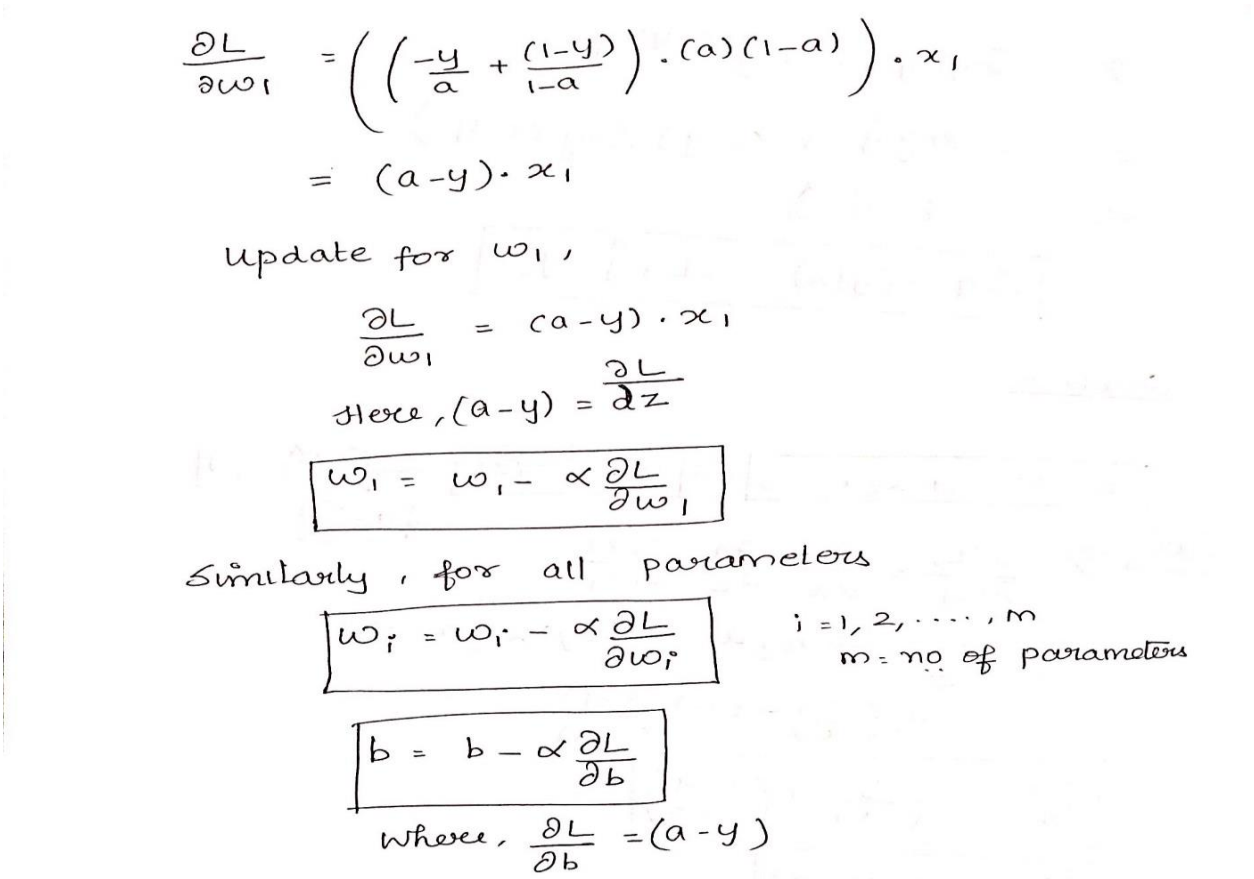
$$\frac{\partial a}{\partial z} = a(1-a)$$

$$\frac{\partial z}{\partial w_1} = x_1$$

Machine Learning theoretical concepts

By: Vikram Pal

In Logistic Regression, we log loss function. Because it gives us non-convex function.



Some of the assumptions of Logistic Regression are as follows:

1. It assumes that there is minimal or **no multicollinearity** among the independent variables i.e, predictors are not correlated.
2. There should be a **linear relationship between the logit of the outcome and each predictor variable**. The logit function is described as **logit(p) = log(p/(1-p))**, where p is the probability of the target outcome.
3. Sometimes to predict properly, it usually requires a **large sample size**.
4. The Logistic Regression which has **binary classification** i.e, two classes assume that the target variable is binary, and ordered Logistic Regression requires the target variable to be ordered.
For example, Too Little, About Right, Too Much.
5. It assumes there is **no dependency** between the observations.

Multiclass classification using Logistic Regression:

Yes, in order to deal with multiclass classification using Logistic Regression, the most famous method is known as the **one-vs-all approach**. In this approach, a number of models are trained, which is equal to the number of classes. These models work in a specific way.

For Example, the first model classifies the datapoint depending on whether it belongs to class 1 or some other class(not class 1); the second model classifies the datapoint into class 2 or some other class(not class 2) and so-on for all other classes.
So, in this manner, each data point can be checked over all the classes.

Linear Regressions cannot be used in the case of binary classification due to the following reasons:

1. **Distribution of error terms:** It assumes that error terms are normally distributed. But this assumption does not hold true in the case of binary classification.

2. **Model output:** In Linear Regression, the output is continuous(or numeric) while in the case of binary classification, an output of a continuous value does not make sense. For binary classification problems, Linear Regression may predict values that can go beyond the range between 0 and 1

3. **The variance of Residual errors:** Linear Regression assumes that the variance of random errors is constant. This assumption is also not held in the case of Logistic Regression

```
def weightInitialization(n_features):
    w = np.zeros((1,n_features))
    b = 0
    return w,b

def sigmoid_activation(result):
    final_result = 1/(1+np.exp(-result))
    return final_result


def model_optimize(w, b, X, Y):
    m = X.shape[0]

    #Prediction
    final_result = sigmoid_activation(np.dot(w,X.T)+b)
    Y_T = Y.T
    cost = (-1/m)*(np.sum((Y_T*np.log(final_result)) + ((1-Y_T)*
(np.log(1-final_result)))))
    #

    #Gradient calculation
    dw = (1/m)*(np.dot(X.T, (final_result-Y.T).T))
    db = (1/m)*(np.sum(final_result-Y.T))

    grads = {"dw": dw, "db": db}

    return grads, cost

def model_predict(w, b, X, Y, learning_rate, no_iterations):
    costs = []
    for i in range(no_iterations):
        #
        grads, cost = model_optimize(w,b,X,Y)
        #
        dw = grads["dw"]
        db = grads["db"]
        #weight update
        w = w - (learning_rate * (dw.T))
        b = b - (learning_rate * db)
        #

        if (i % 100 == 0):
            costs.append(cost)
            #print("Cost after %i iteration is %f" %(i, cost))

    #final parameters
    coeff = {"w": w, "b": b}
    gradient = {"dw": dw, "db": db}

    return coeff, gradient, costs

def predict(final_pred, m):
    y_pred = np.zeros((1,m))
    for i in range(final_pred.shape[1]):
        if final_pred[0][i] > 0.5:
            y_pred[0][i] = 1
    return y_pred
```

Machine Learning theoretical concepts

By: Vikram Pal

Chose right machine learning Algorithm:

- Size of the training data
- Accuracy and interpretability of the output
- Speed or Training time
- Linearity
- Number of features

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.

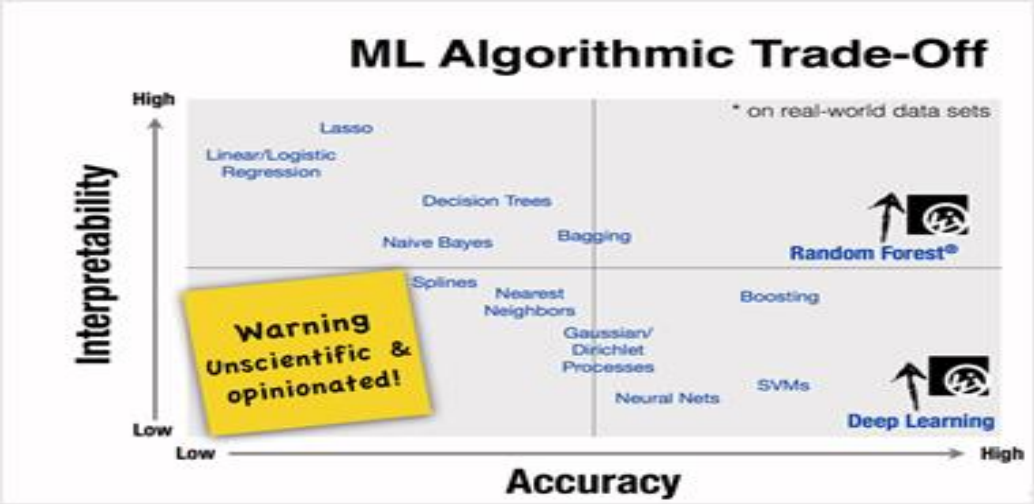
Chose algorithm with a high bias/low variance like linear regression, Naive Bayes or linear SVM.
If the training data is larger/sufficient or the training data has a greater number of observations as compared to number of features.

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

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.

Machine Learning theoretical concepts

By: Vikram Pal

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.

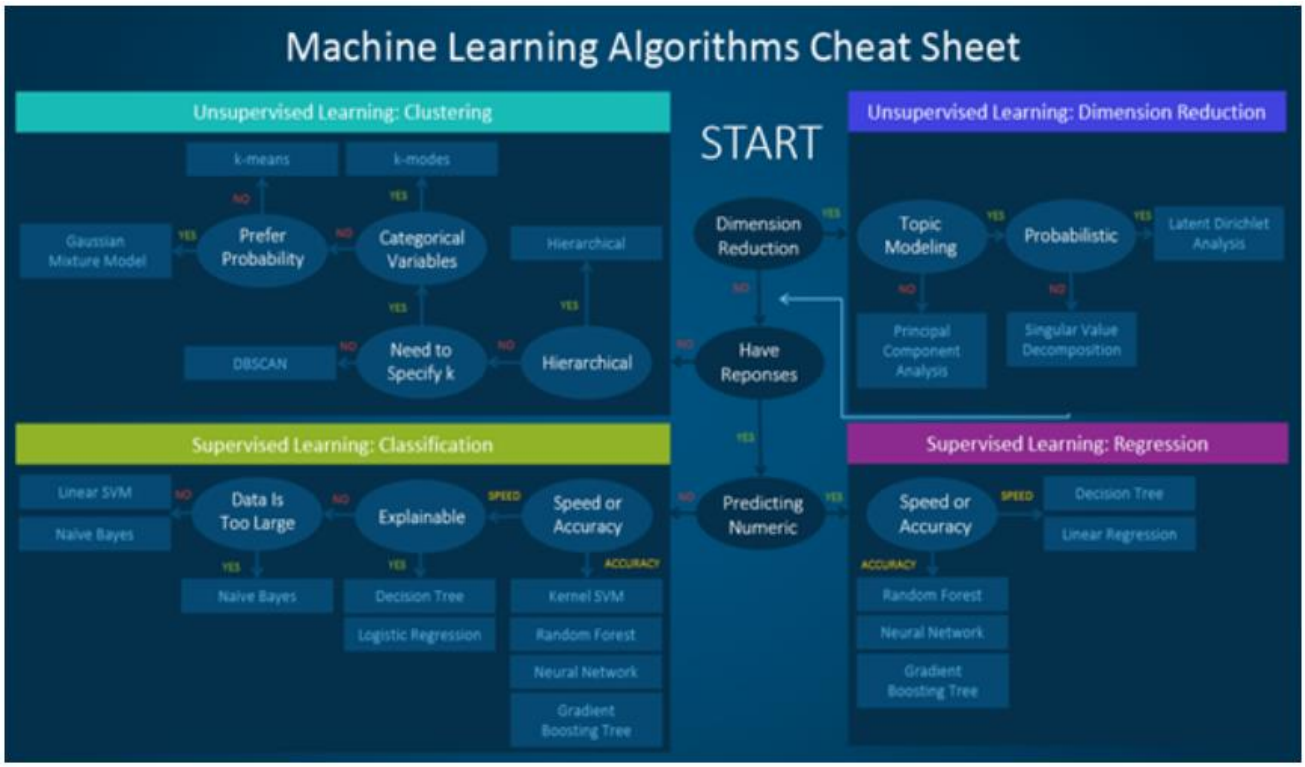
However, not all data is not linear. In that case, **we need some algorithm to work in nonlinear and high dimension data.** For example, random forest, kernel SVM and neural net.

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.



Machine Learning theoretical concepts

By: Vikram Pal

Model Evaluation Metrics:

Precision, Recall, and F1 Score:

		Predicted Class: Positive	**Predicted Class: Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

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, an 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:

Machine Learning theoretical concepts

By: Vikram Pal

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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).

$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Micro-Averaged and Weighted Average:

Micro-averaged: all samples equally contribute to the final averaged metric. Macro-averaged: all classes equally contribute to the final averaged metric.

Weighted-averaged: each classes' contribution to the average is weighted by its size.

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

AUC-ROC Curve:

https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/

Sensitivity / True Positive Rate / Recall:

Sensitivity tells us what proportion of the positive class got correctly classified.

$$Sensitivity = \frac{TP}{TP + FN}$$

Machine Learning theoretical concepts

By: Vikram Pal

A simple example would be to determine what proportion of the actual sick people were correctly detected by the model.

False Negative Rate:

False Negative Rate (FNR) tells us what proportion of the positive class got incorrectly classified by the classifier.

$$FNR = \frac{FN}{TP + FN}$$

A higher TPR and a lower FNR is desirable since we want to correctly classify the positive class.

Specificity / True Negative Rate:

Specificity tells us what proportion of the negative class got correctly classified.

$$Specificity = \frac{TN}{TN + FP}$$

Taking the same example as in Sensitivity, Specificity would mean determining the proportion of healthy people who were correctly identified by the model.

False Positive Rate:

FPR tells us what proportion of the negative class got incorrectly classified by the classifier.

$$FPR = \frac{FP}{TN + FP} = 1 - Specificity$$

A higher TNR and a lower FPR is desirable since we want to correctly classify the negative class.

Precision vs Recall Usage Example:

A healthcare data scientist will have to ensure his (or her) model has **fewer false negatives** as it could cost a patient his life if an **incorrect cancer diagnosis is performed**.

On the other hand, **a system installed in a mall to detect shoplifters** has to worry about **too many false positives** as it would mean causing a **huge deal of embarrassment to otherwise innocent shoppers**.

Probability of Predictions:

A machine learning classification model can be used:

a. predict the actual class of the data point directly
b. predict its probability of belonging to different classes. ### auc roc

Setting different thresholds for classifying positive class for data points will change the Sensitivity and Specificity of the model.

And one of these thresholds will probably give a better result than the others, depending on whether we are aiming to lower the number of False Negatives or False Positives.

ID	Actual	Prediction Probability	>0.6	>0.7	> 0.8	Metric
1	0	0.98	1	1	1	
2	1	0.67	1	0	0	
3	1	0.58	0	0	0	
4	0	0.78	1	1	0	
5	1	0.85	1	1	1	
6	0	0.86	1	1	1	
7	0	0.79	1	1	0	
8	0	0.89	1	1	1	
9	1	0.82	1	1	1	
10	0	0.86	1	1	1	
			0.75	0.5	0.5	TPR
			1	1	0.66	FPR
			0	0	0.33	TNR
			0.25	0.5	0.5	FNR

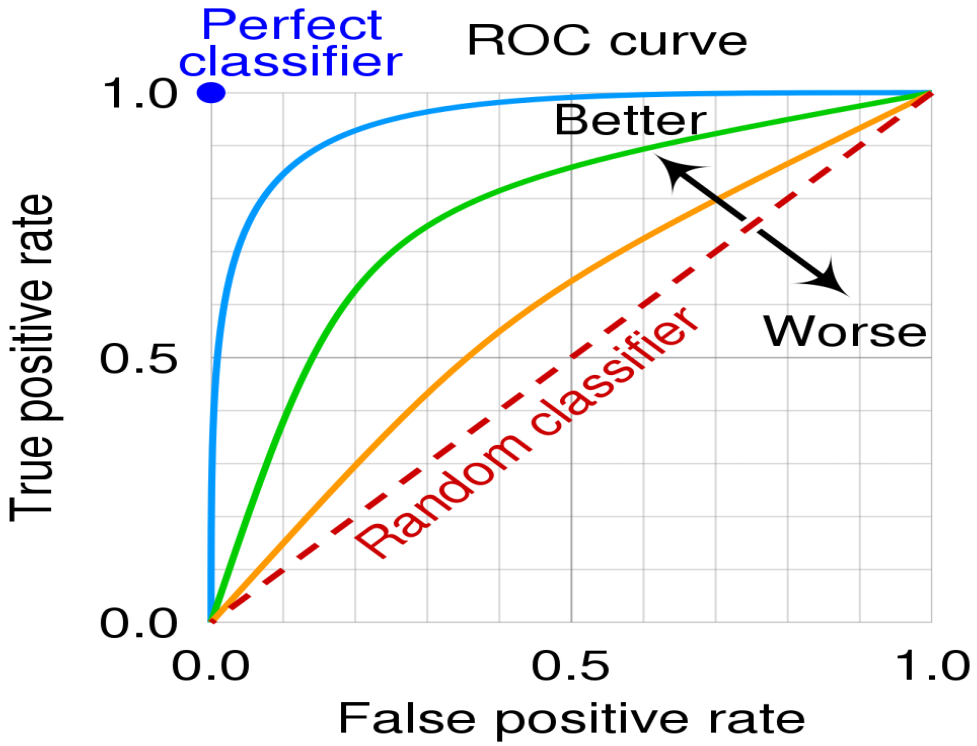
Machine Learning theoretical concepts

By: Vikram Pal

What is the AUC-ROC curve?

The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise'.**

The Area Under the Curve (AUC) is the measure of the **ability of a classifier to distinguish between classes and is used as a summary of the ROC curve**.



a. AUC=1 perfectly distinguish between all the Positive and the Negative class points correctly.
b. AUC=0 the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.
c. 0.5<AUC<1 there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values.
d. AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points.

Parameters vs Hyperparameters:-

Some examples of model parameters include: (model building)

- The weights in an artificial neural network.
- The support vectors in a support vector machine.
- The coefficients in a linear regression or logistic regression.

Some examples of model hyperparameters include: (fine tuning of model)

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

Machine Learning theoretical concepts

By: Vikram Pal

Dimensionality Reduction:

https://towardsdatascience.com/11-dimensionality-reduction-techniques-you-should-know-in-2021-dcb9500d388b

- **Principal Component Analysis (PCA), Factor Analysis (FA), Linear Discriminant Analysis (LDA) and Truncated Singular Value Decomposition (SVD)** are examples of **linear dimensionality reduction** methods.
- **Kernel PCA, t-distributed Stochastic Neighbor Embedding (t-SNE), Multidimensional Scaling (MDS) and Isometric mapping** (Isomap) are examples of **non-linear dimensionality reduction** methods

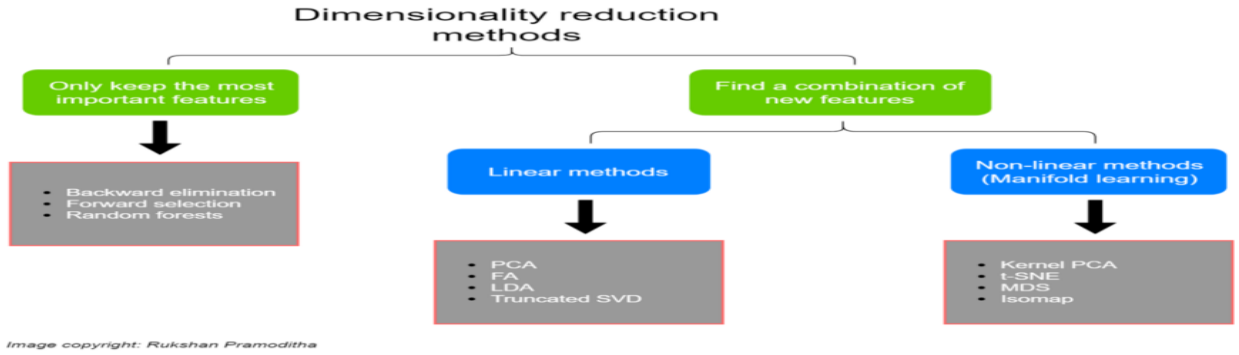


Image copyright: Rukshan Pramoditha

- LDA finds a **linear combination of input features** that **optimizes class separability**
- PCA attempts to find a set of **uncorrelated components of maximum variance** in a dataset.

- As we know, PCA tries to **find directions of maximum variance**. PCA projects data onto new axis in such a way they explain the **maximum variance without taking class labels into consideration**.
- LDA on the other hand, **creates new axis** in such a way that when **we project data on this axis**, there is a **maximum separation between two class categories**. LDA tries to **separate classes as much as feasible on the new axis**.

How does LDA achieves this?

LDA creates new axis based on two criteria:

- Distance between means of classes
- Variation within each category

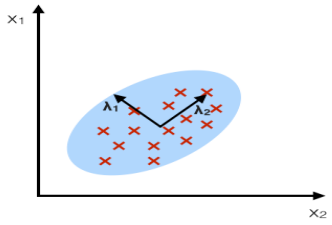
The maximum number of components that LDA can find is **the number of classes minus 1**. If there are only 3 class labels in your dataset, LDA can find only **2 (3–1)** components in dimensionality reduction. It is not needed to perform feature scaling to apply LDA.

Question: how to decide classes or labels when you are not given?

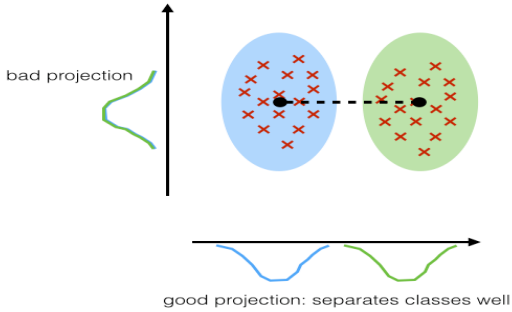
Machine Learning theoretical concepts

By: Vikram Pal

PCA:
component axes that maximize the variance



LDA:
maximizing the component axes for class-separation



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:

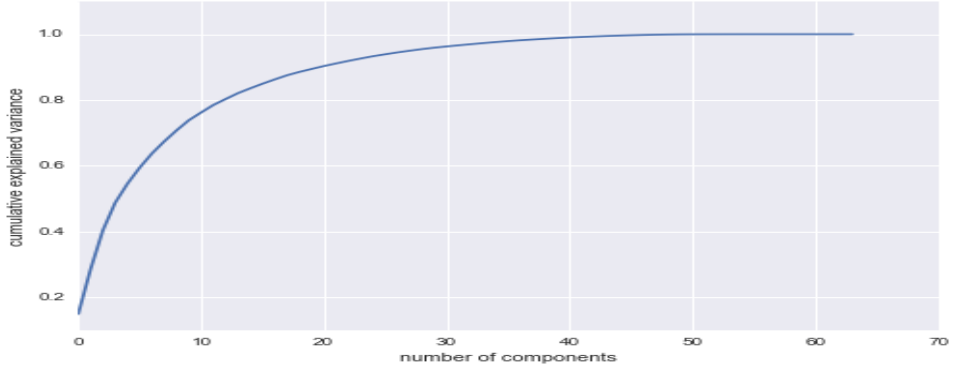
1. Standardize the dataset.
2. Find the Eigenvalue and Eigenvectors using **covariance matrix** or correlation matrix.
3. Sort the Eigenvectors in descending order.
4. Create a projection matrix w using top K Eigenvectors.
5. Transform the original dataset x using projection matrix to obtain k-dimensional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

Picking Principal Components Using the Explained Variance:

we want to see how much of the variance in data is explained by each one of these components. It is a convention to use 95% explained variance.

Determining how many components:

This can be determined by looking at the **cumulative *explained variance ratio*** as a function of the number of components:



This curve quantifies how much of the total, 64-dimensional variance is contained within the first NN components. For example, we see that with the digits the first 10 components contain

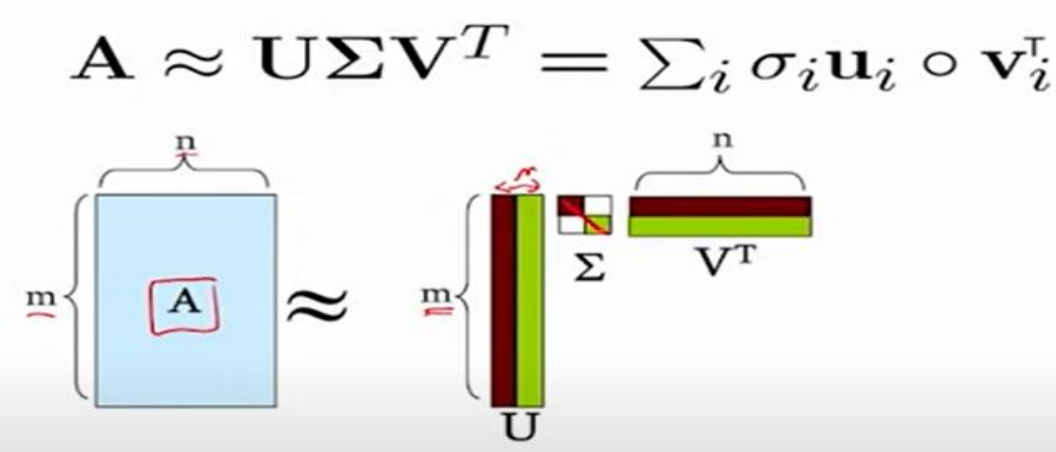
approximately 75% of the variance, while you need around 50 components to describe close to 100% of the variance.

Linear Discriminant Analysis (Supervised): (d-dimensional mean vectors for the different classes from the dataset)

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.

Truncated Singular Value Decomposition (SVD):



SVD Properties:

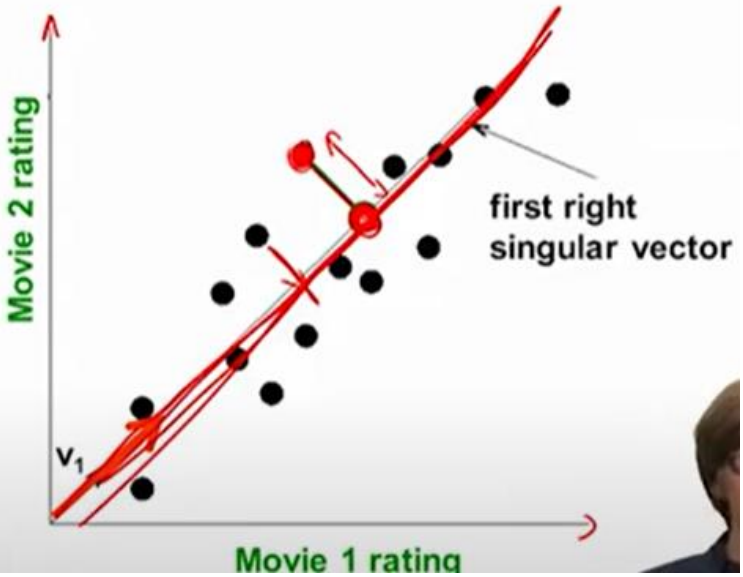
Machine Learning theoretical concepts

By: Vikram Pal

It is **always** possible to decompose a real matrix A into $A = U \Sigma V^T$, where

- U, Σ, V: unique
- U, V: column orthonormal
  - $U^T U = I;\ V^T V = I$ (I: identity matrix)
  - (Columns are orthogonal unit vectors)
- Σ: diagonal
  - Entries (**singular values**) are positive, and sorted in decreasing order ($\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$)

- **SVD gives 'best' axis to project on:**
  - '**best**' = min sum of squares of projection errors
- **In other words, minimum reconstruction error**

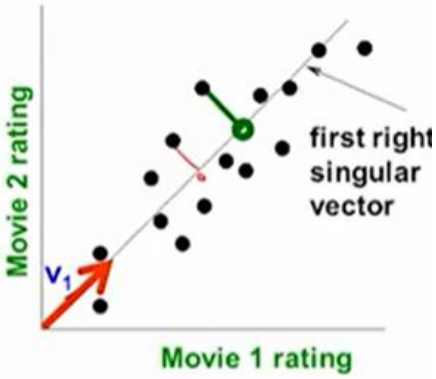


Machine Learning theoretical concepts

By: Vikram Pal

$A = U \Sigma V^T$ - example:

- $U \cdot \Sigma$: Gives the coordinates of the points in the projection axis

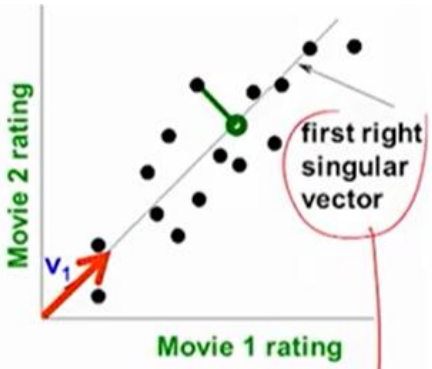


$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix}$$

Projection of users on the "Sci-Fi" axis ($(U \Sigma)^T$):

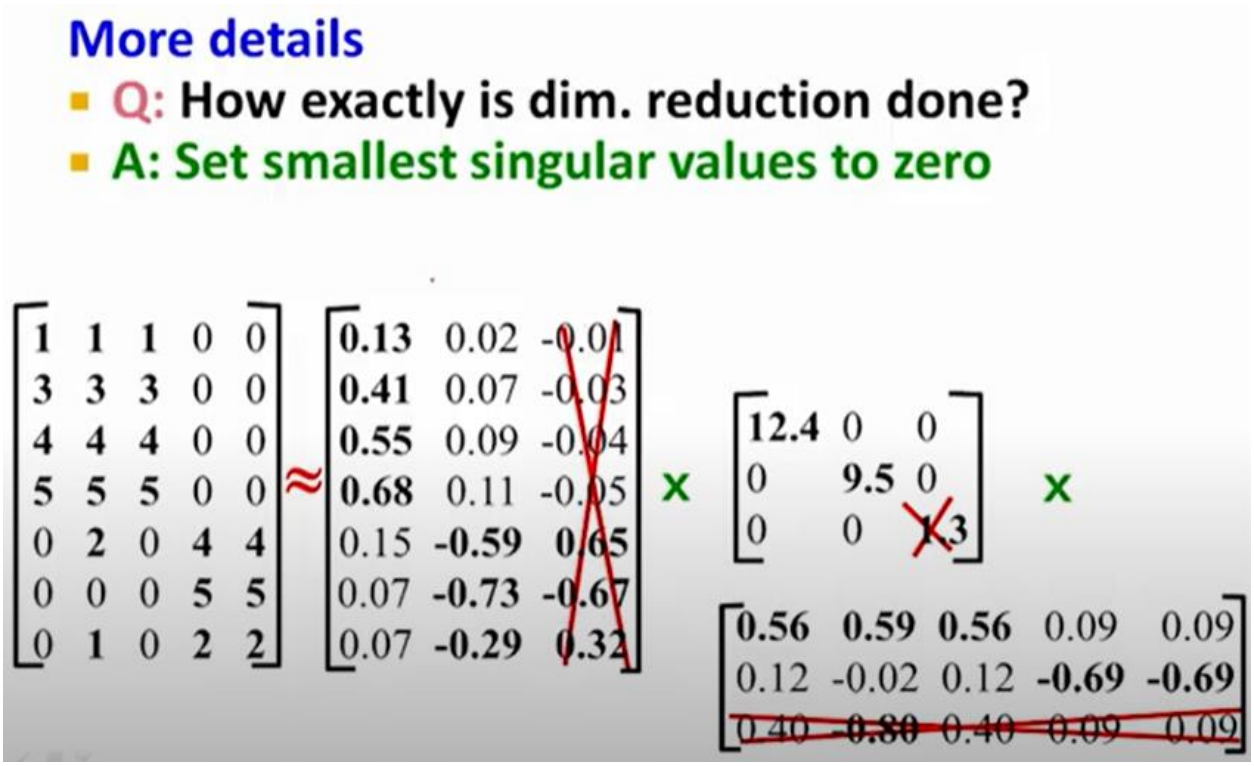
$$\begin{bmatrix} 1.61 & 0.19 & -0.01 \\ 5.08 & 0.66 & -0.03 \\ 6.82 & 0.85 & -0.05 \\ 8.43 & 1.04 & -0.06 \\ 1.86 & -5.60 & 0.84 \\ 0.86 & -6.93 & -0.87 \\ 0.86 & -2.75 & 0.41 \end{bmatrix}$$

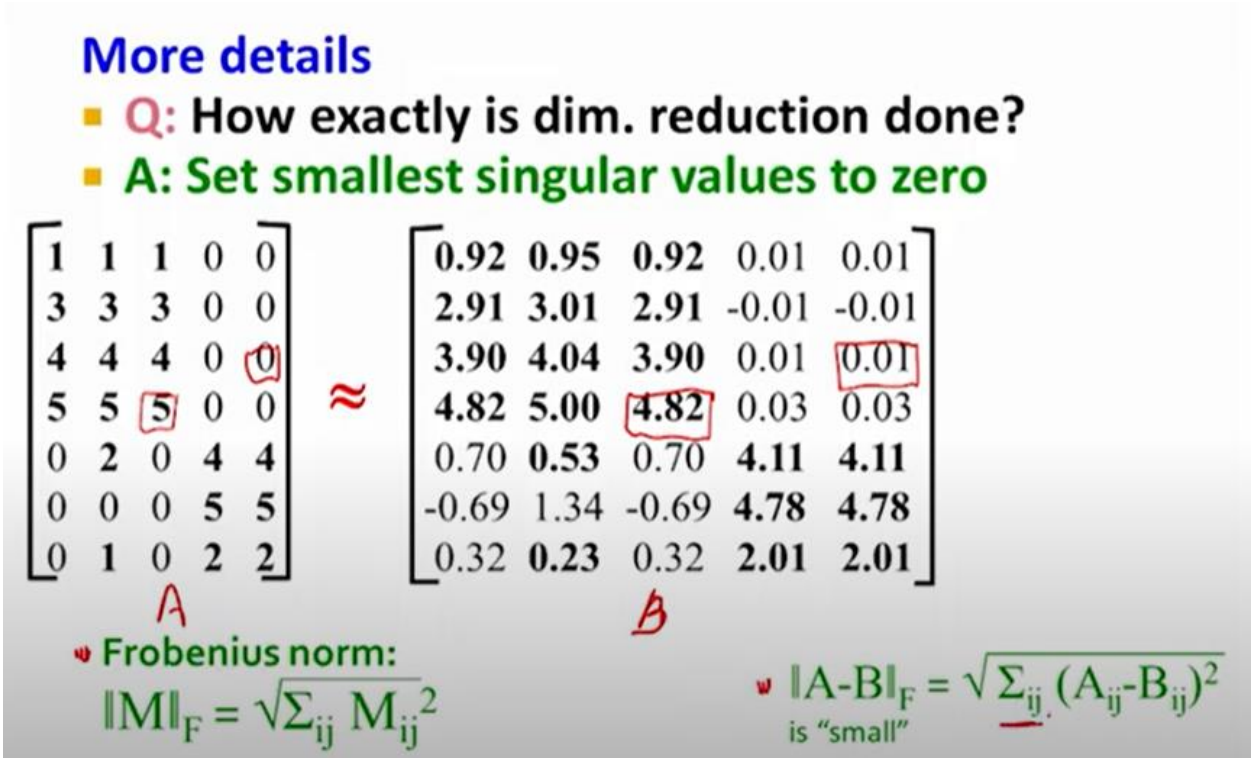
$A = U \Sigma V^T$ - example:



variance ('spread') on the v_1 axis

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} \mathbf{0.13} & 0.02 & -0.01 \\ \mathbf{0.41} & 0.07 & -0.03 \\ \mathbf{0.55} & 0.09 & -0.04 \\ \mathbf{0.68} & 0.11 & -0.05 \\ 0.15 & \mathbf{-0.59} & \mathbf{0.65} \\ 0.07 & \mathbf{-0.73} & \mathbf{-0.67} \\ 0.07 & \mathbf{-0.29} & \mathbf{0.32} \end{bmatrix} \times \begin{bmatrix} \mathbf{12.4} & 0 & 0 \\ 0 & \mathbf{9.5} & 0 \\ 0 & 0 & \mathbf{1.3} \end{bmatrix} \times \begin{bmatrix} \mathbf{0.56} & \mathbf{0.59} & \mathbf{0.56} & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & \mathbf{-0.69} & \mathbf{-0.69} \\ 0.40 & \mathbf{-0.80} & 0.40 & 0.09 & 0.09 \end{bmatrix}$$





It works well with **sparse data in which many of the row values are zero**. In contrast, PCA works well with **dense data**. Truncated SVD can also be used with dense data

Another key difference between truncated SVD and PCA is that **factorization for SVD is done on the data matrix** while factorization **for PCA is done on the covariance matrix**.

Kernal PCA:

Kernel PCA is a **non-linear dimensionality reduction technique that uses *kernels***. It can also be considered as **the non-linear form of normal PCA**. Kernel PCA works well with non-linear datasets where normal PCA cannot be used efficiently.

Steps:

Data is first run through a kernel function → Temporarily projects them into a new higher-dimensional feature space where the classes become linearly separable (classes can be divided by drawing a straight line) → Normal PCA to project the data back onto a lower-dimensional space.

In this way, Kernel PCA transforms non-linear data into a lower-dimensional space of data which can be used with linear classifiers.

In the Kernel PCA, we need to specify 3 important hyperparameters — the **number of components** we want to keep, **the type of kernel** and **the kernel coefficient** (also known as the *gamma*). For the type of kernel, we can use ***'linear', 'poly', 'rbf', 'sigmoid', 'cosine'***. The ***rbf kernel*** which is known as the **radial basis function kernel** is the most popular one.

Now, we are going to implement an RBF kernel PCA to non-linear data which can be generated by using **the Scikit-learn make_moons() function**.

It requires implementing a hyperparameter tuning technique such as Grid Search to find an optimal value for the $\boldsymbol{gamma}$.

t-SNE (t-distributed stochastic neighbor embedding):

. probability distribution:- a. High dimensional object→ similar objects assigned higher probability and dissimilar object assigned lower

. minimize KL divergence

The t-SNE algorithm comprises two main stages.

First, t-SNE constructs a **probability distribution over pairs of high-dimensional objects** in such a way that similar objects are assigned a higher probability while dissimilar points are assigned a lower probability.

Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence (KL divergence) between the two distributions with respect to the locations of the points in the map. While the original algorithm uses the Euclidean distance between objects as the base of its similarity metric, this can be changed as appropriate.

KL divergence: is a measure of how a probability distribution differs from another probability distribution.

Machine Learning theoretical concepts

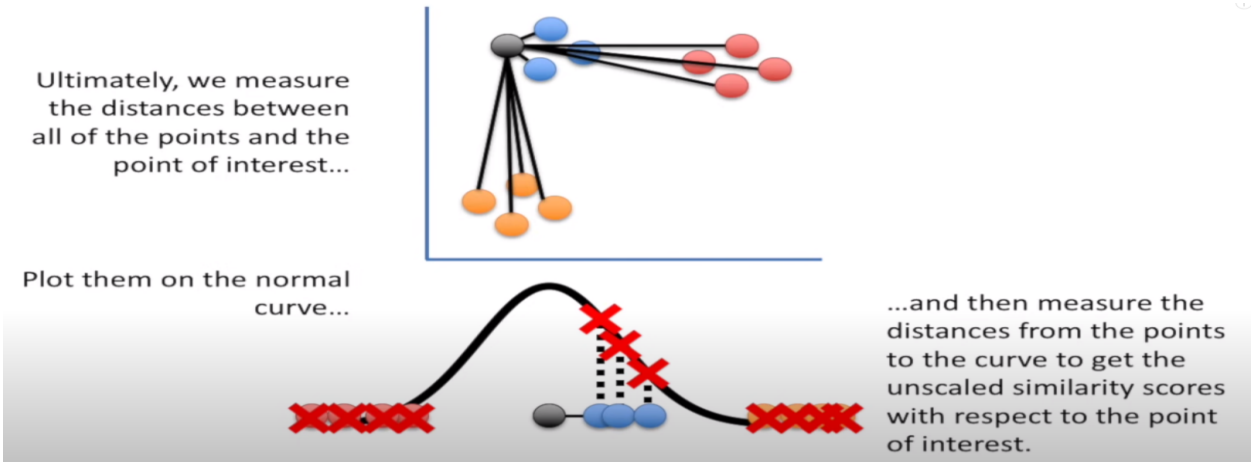
By: Vikram Pal

PCA vs T-SNE:

PCA: it preserves global shape or state of data.

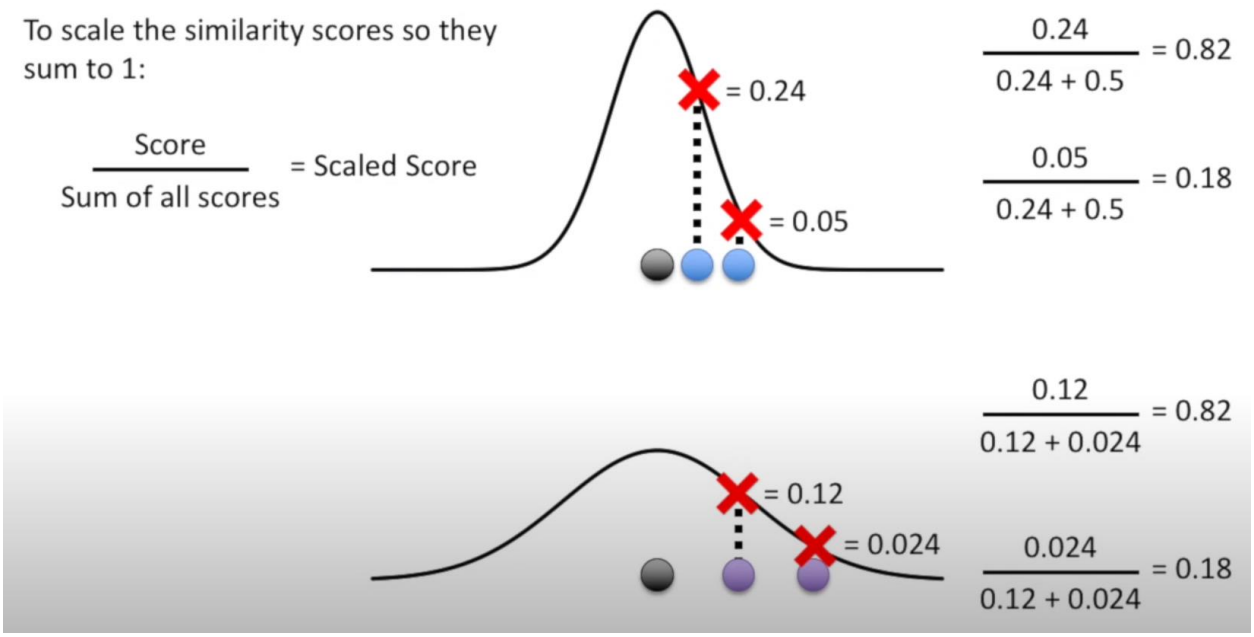
T-SNE: it preserves local shape or state of data.

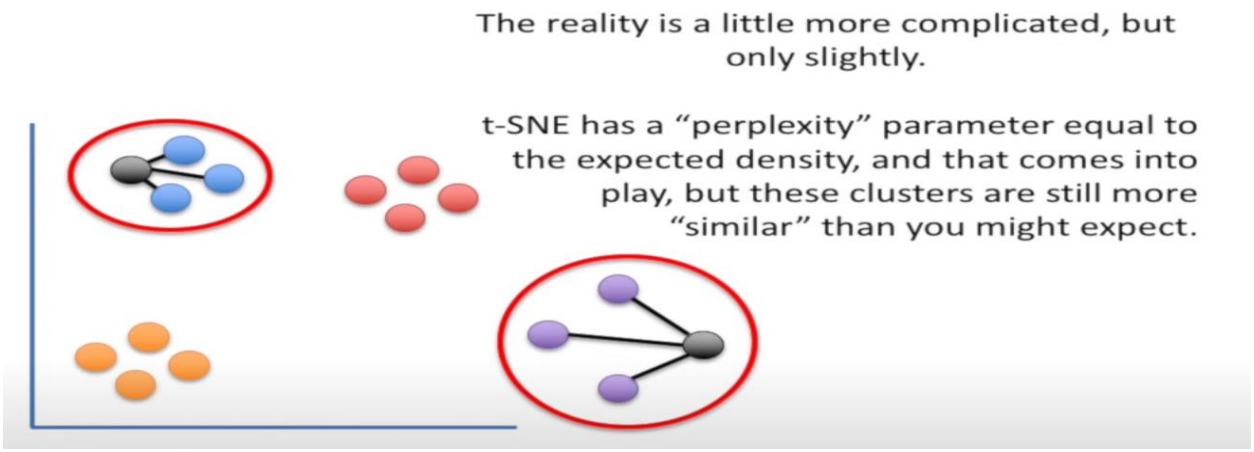
t-distribution: T-SNE uses t-distribution, because without it the clusters would all clump up in the middle and be harder to see.

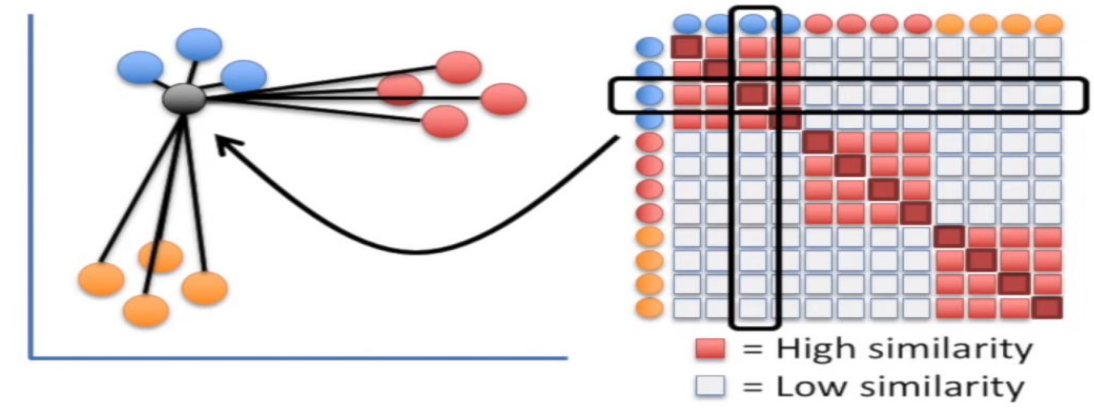


Machine Learning theoretical concepts

By: Vikram Pal

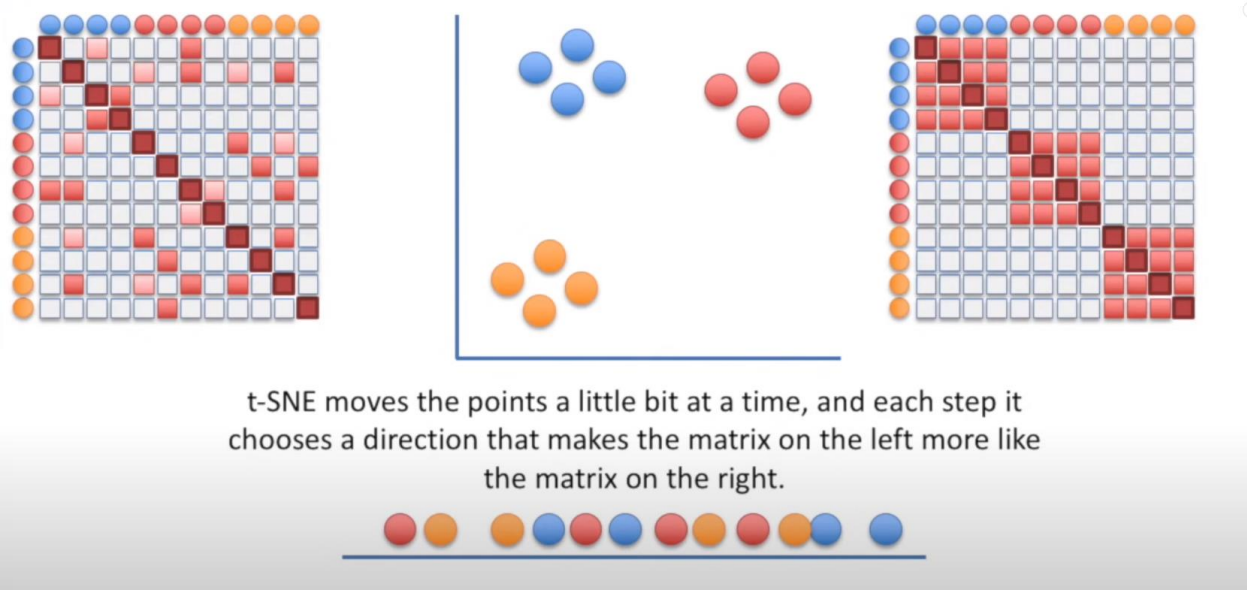






Machine Learning theoretical concepts

By: Vikram Pal



Multidimensional Scaling:

MDA is another non-linear dimensionality reduction technique **that tries to preserve the distances between instances while reducing the dimensionality of non-linear data**.

There are two types of MDS algorithms: **Metric and Non-metric.**

The MDS() class in the Scikit-learn implements both by setting the metric **hyperparameter to True (for Metric type) or False (for Non-metric type).**

Isometric mapping (Isomap)

This method performs non-linear dimensionality reduction **through Isometric mapping**. It is an **extension of MDS or Kernel PCA.** It connects **each instance by calculating the curved or geodesic distance to its nearest neighbors and reduces dimensionality**.

The number of neighbors to consider for each point can be specified through the n_neighbors hyperparameter of the Isomap() class which implements the Isomap algorithm in the Scikit-learn.

Backward Elimination:

This method eliminates (removes) features from a dataset through a **recursive feature elimination** (RFE) process.

The algorithm **first attempts to train the model on the initial set of features in the dataset and calculates the performance of the model (usually, accuracy score for a classification model and RMSE for a regression model).** Then, the algorithm drops one feature (variable) at a time, trains the model on the remaining features and calculates the performance scores. The algorithm repeats eliminating features until it detects a small (or no) change in the performance score of the model and stops there!

Forward Selection

This method can be considered as the opposite process of backward elimination.

Instead of eliminating features recursively, the algorithm attempts to train **the model on a single feature in the dataset and calculates the performance of the model** (usually, accuracy score for a classification model and RMSE for a regression model).

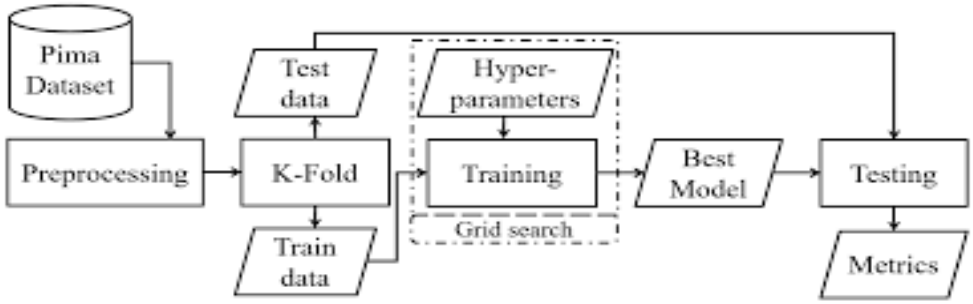
Then, the algorithm **adds (selects) one feature (variable) at a time, trains the model on those features and calculates the performance scores**. The algorithm repeats adding features until it detects a small (or no) change in the performance score of the model and stops there!

Random forests:

Random forests is a tree-based model which is widely used for **regression and classification tasks on non-linear data**.

It can also be used for feature selection with its built-in **feature_importances_** attribute which calculates feature importance scores for each **feature based on the** 'gini' **criterion** (a measure of the quality of a split of internal nodes) while training the model.

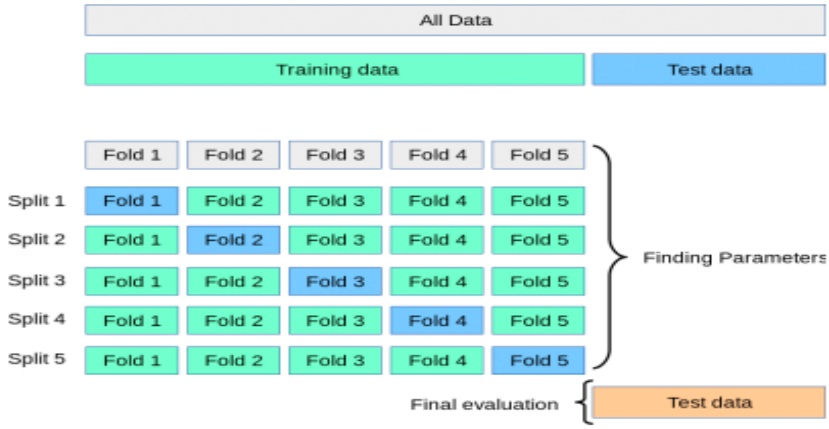
Model Selection:



Machine Learning theoretical concepts

By: Vikram Pal

K-Fold cross validation:



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.

```
# Create the model to be tuned
rf_base = RandomForestRegressor()

# Create the random search Random Forest
rf_random = RandomizedSearchCV(estimator = rf_base, param_distributions = rf_grid,
                               n_iter = 200, cv = 3, verbose = 2, random_state = 42,
                               n_jobs = -1)

# Fit the random search model
rf_random.fit(X_train_temp, y_train_temp)

# View the best parameters from the random search
rf_random.best_params
```

Parametric vs non-parametric model

Parametric models:

To summarise, parametric methods in Machine Learning usually take a model-based approach where we make an assumption with respect to form of the function to be estimated and then we select a suitable model based on this assumption in order to **estimate the set of parameters**.

Some examples of parametric methods in Machine Learning include Linear Discriminant Analysis, Naive Bayes and Perceptron.

Non-Parametric Methods:

Machine Learning theoretical concepts

By: Vikram Pal

On the other hand, non-parametric methods refer to a set of algorithms that do not make any underlying assumptions with respect to the form of the function to be estimated. And since no assumption is being made, such methods are capable of estimating the unknown function f that could be of any form.

Some examples of non-parametric methods in Machine Learning include Support Vector Machines and K-Nearest Neighbours.

Clustering:

K-mean Clustering:

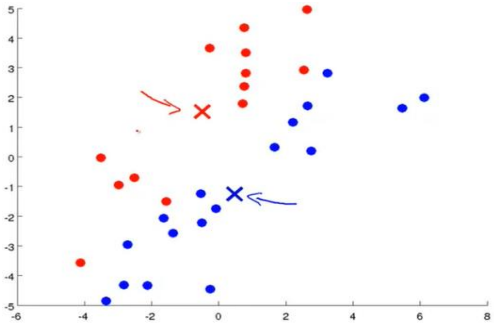


Fig.1 random centroids

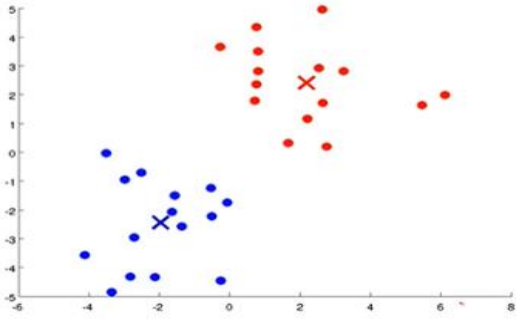


Fig.2 Average of all red and blue datapoints

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.

(We must take an average of all the red dots that are assigned to the red cluster centroid and move the red cluster centroid to that average. We need to do the same for the blue cluster 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**.

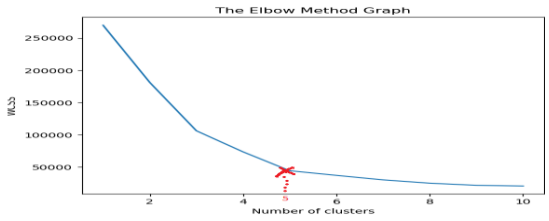
Choosing the right number of clusters:

$$\text{WCSS} = \sum_{P_i \text{ in Cluster 1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} \text{distance}(P_i, C_3)^2$$

Once we will compute WCSS:

Machine Learning theoretical concepts

By: Vikram Pal



Number of clusters are 5 in the above diagram.

Hierarchical Clustering:

It is two types: **agglomerative(top-bottom approach) and Divisive(bottom-up approach).**

a. Make each data point a single point cluster --- > That form N clusters

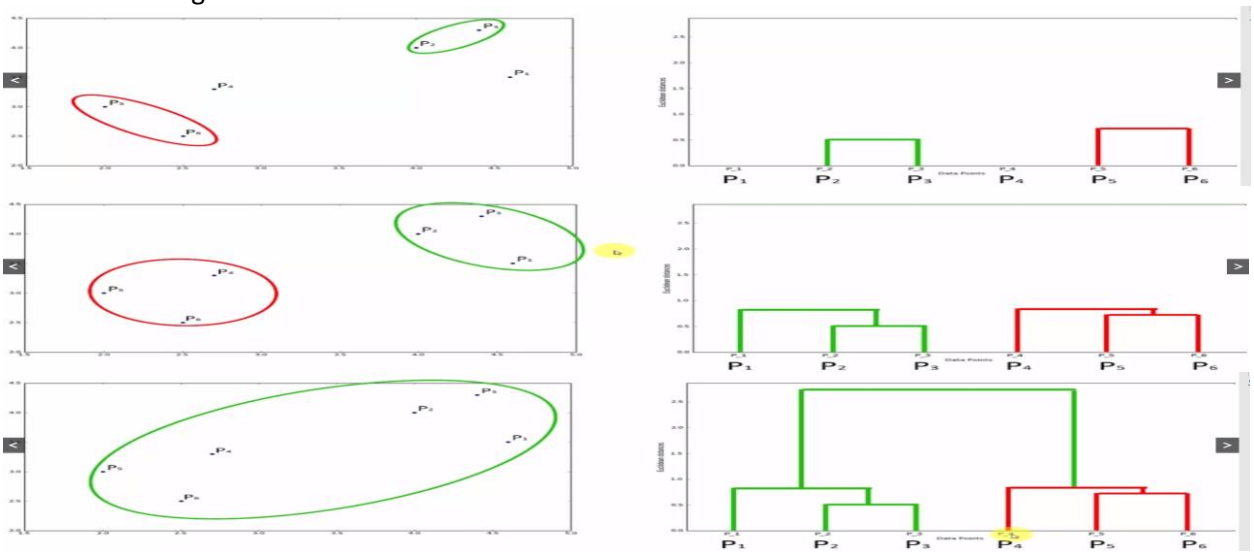
b. Take the two closest data points and make them one cluster --- > that forms N-1 clusters.

c. Take the two closest data points and make them one cluster --- > that forms N-2 clusters.

d. Repeat Step 3 until there is only one cluster.

How to measure distance b/w two clusters?

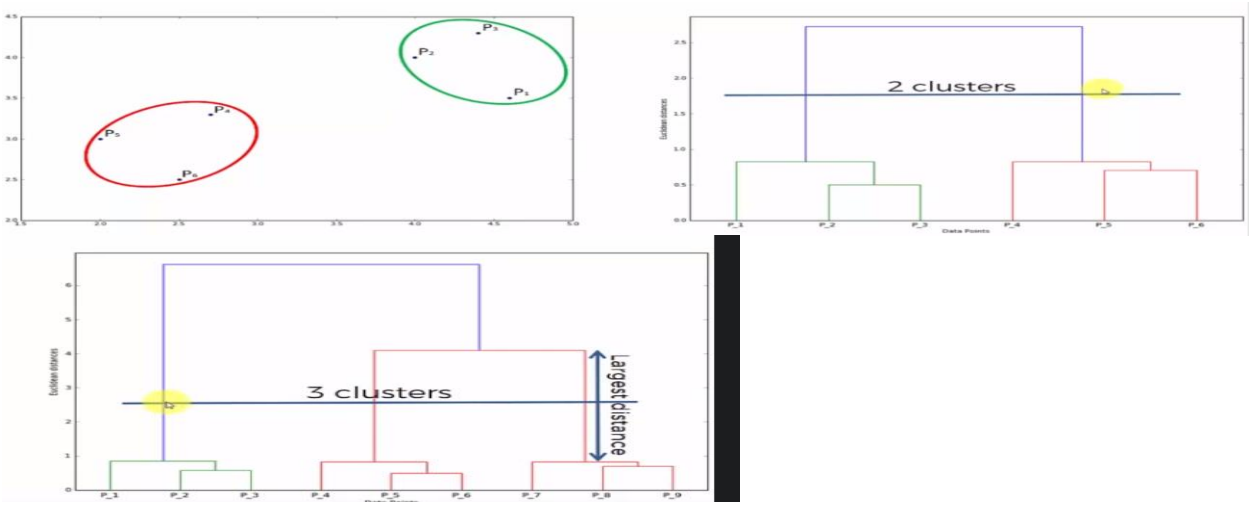
How do Dendrograms work?



We will setup a threshold of Euclidean Distance that will give number of clusters (Number of lines cut with threshold point or largest distance)

Machine Learning theoretical concepts

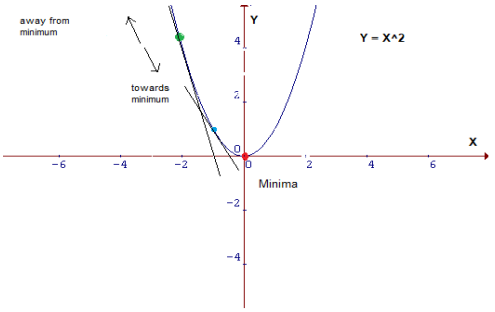
By: Vikram Pal



Gradient descent (Which way to go and how big a step to take {Tangent gives us a sense of steepness of the slope}:

Gradient: **a gradient is a partial derivative with respect to its inputs.** A gradient measure how much the output of a function changes if you change the inputs a little bit. **You can also think of a gradient as the slope of a function**. Higher the gradient, steeper the slope and the faster a model can learn. If the slope is almost zero, the model stops to learn. A gradient simply measures the change in all weights with regard to the change in error.
It 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**.



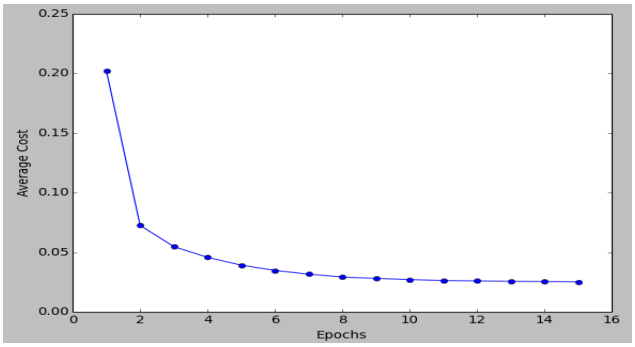
Type of Gradient Descents:

a. Batch Gradient Descent:

All training data is taken into consideration to take a single step.

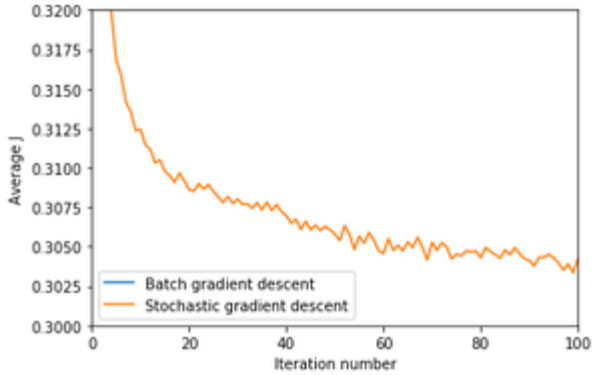
Machine Learning theoretical concepts

By: Vikram Pal



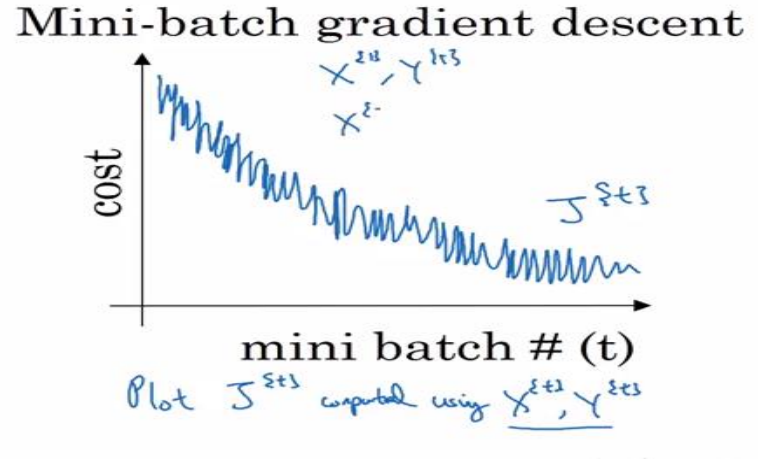
b. Stochastic Gradient Descent:

we consider **one example at a time** to take a single step.



c. Mini Batch GD:

it is mixture of batch and Stochastic gradient descent.



Learning rate:

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

Machine Learning theoretical concepts

By: Vikram Pal

Source

Different types of setting up a learning rate:

1. Learning Rate Schedules:

a. Constant Learning Rate

b. Time Based Decay

```
lr *= (1. / (1. + self.decay * self.iterations))
```

c. Step Decay:

Step decay schedule drops the learning rate by a factor every few epochs. The mathematical form of step decay is :

$$lr = lr0 * drop^{floor(epoch / epochs\_drop)}$$

d. Exponential decay

`lr = lr0 * e^(-kt)`,

where `lr`, `k` are hyperparameters and `t` is the iteration number

2. Adaptive learning rate methods:

a. Momentum:

It is based on **exponential weighted average**.

An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points. The exponential moving average is also referred to as the exponentially weighted moving average.

How does Momentum work?

The basic idea is to compute an exponentially weighted average of your gradients, and then use that gradient to update your weights instead

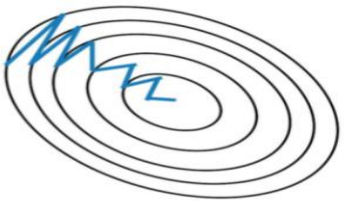
For instance, if we have a momentum of 0.90, we will take 90% of the previous direction plus 10% of the new direction and adjust weights accordingly -- multiplying that direction vector by the learning rate.

Machine Learning theoretical concepts

By: Vikram Pal

Momentum is a technique to **prevent sensitive movement**. When the gradient gets computed every iteration, it can have totally different direction and the steps make a zigzag path, which makes training very slow. Something like this.

Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

On iteration t:

Compute dW, db on the current mini-batch

$v_{dW} = \beta v_{dW} + (1 - \beta) dW$

$v_{db} = \beta v_{db} + (1 - \beta) db$

$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$

Hyperparameters: α, β $\beta = 0.9$

b. RmsProp:

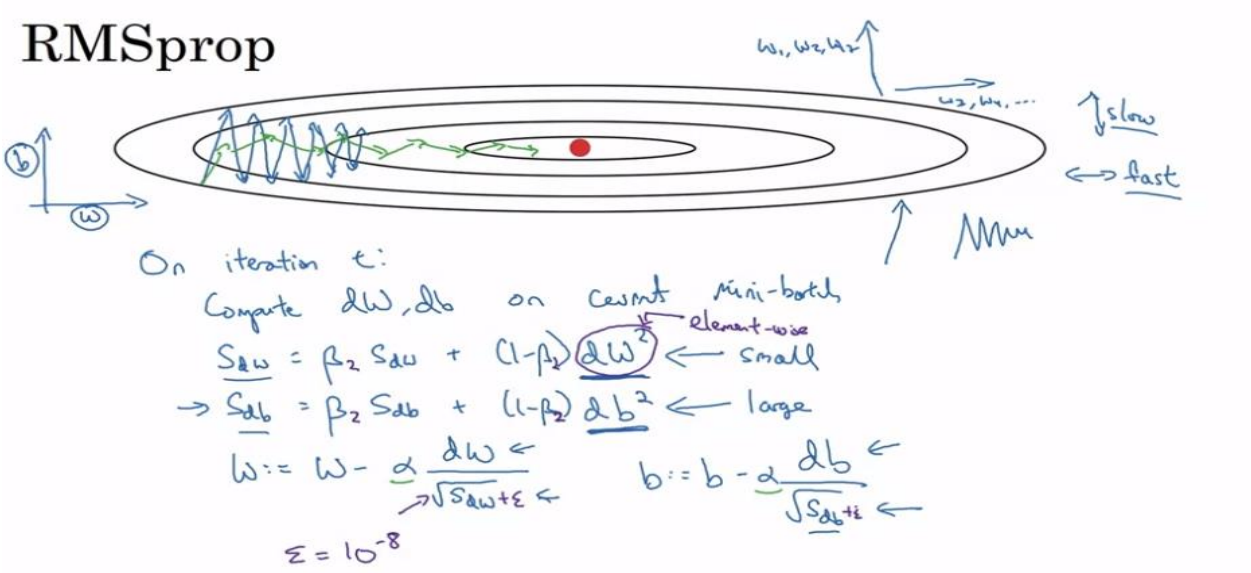
when vanish or explode as the data propagates through the function →(moving average of squared gradients to normalize the gradient)

RMSprop is a gradient-based optimization technique used in training neural networks. Gradients of very complex functions like neural networks have a tendency to either **vanish or explode as the data propagates through the function** (*refer to vanishing gradients problem). Rmsprop was developed as a stochastic technique for mini-batch learning.

RMSprop deals with the above issue by using a **moving average of squared gradients to normalize the gradient**. This normalization balances the step size (momentum), **decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing.**

Machine Learning theoretical concepts

By: Vikram Pal

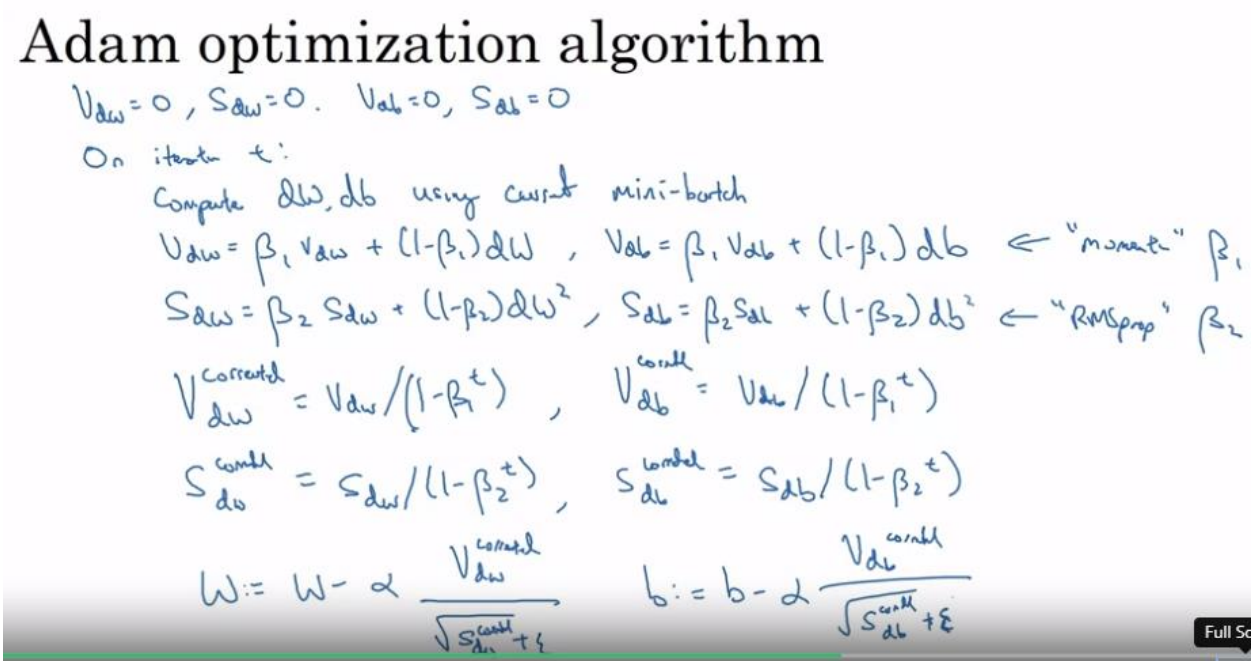


c. Adam:

It can be considered as combination of RMSProp and Stochastic Gradient Descent.

→It uses **Squared Gradient to scale the learning rate** like RMSprop.

→It takes advantage of **momentum by using moving average of the gradient** instead of gradient itself.



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.

Machine Learning theoretical concepts

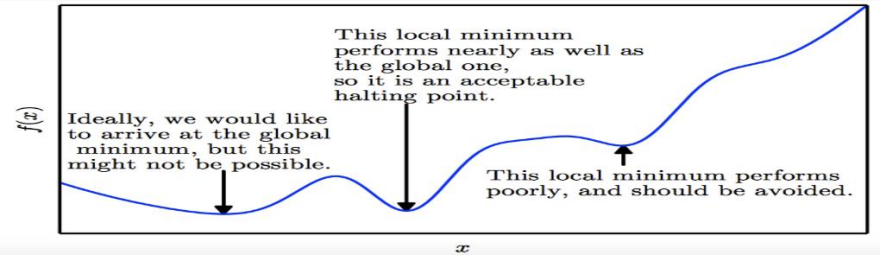
By: Vikram Pal

Challenges with Optimization:

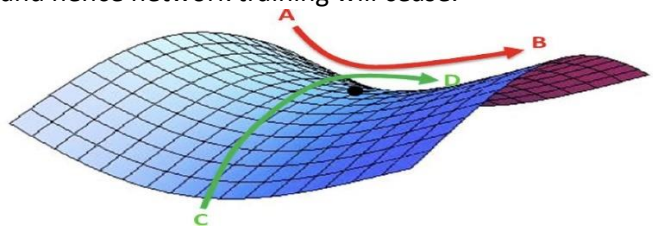
Non-convex optimization involves a function which has multiple optima, only one of which is the global optima. Depending on the loss surface, it can be exceedingly difficult to locate the global optima.
There are certain problems associated with this:

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.



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**.**

$$W_{ij} \sim N\left(0, \frac{1}{m}\right)$$

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.

$$W_{ij} \sim N\left(0, \frac{2}{m}\right)$$

Quantization:

Uniform quantization is a technique for reducing the precision of numerical values in a dataset or model by mapping them to a fixed set of discrete values. It is a common method for reducing the size and complexity of machine learning models, and is often used in conjunction with other techniques such as weight pruning and model compression.

In uniform quantization, the range of possible values is divided into a set of equal intervals, and each interval is assigned a discrete value. For example, if the range of possible values is [0, 255] and we want to quantize the values into 8 intervals, we can divide the range into 8 equal intervals of 32, and assign the value 0 to the interval [0, 31], the value 1 to the interval [32, 63], and so on.

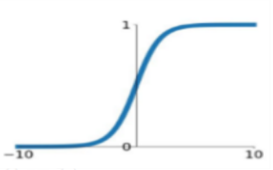
Uniform quantization is often used in deep learning models to reduce the precision of weights and activations, which can significantly reduce the size and complexity of the model. It can also be used to reduce the precision of other types of data, such as images or audio signals.

Overall, uniform quantization is a useful technique for reducing the size and complexity of machine learning models, and can be applied to a wide range of applications. However, it is important to carefully consider the trade-off between model size and performance, as reducing the precision of the values can affect the accuracy and performance of the model.

Nonlinear Activation function

Sigmoid or logistic Activation Function:

Function	Equation	Range	Derivative
Sigmoid (Logistic)	$f(x) = \frac{1}{1+e^{-x}}$	0,1	$f'(x) = f(x)(1-f(x))$



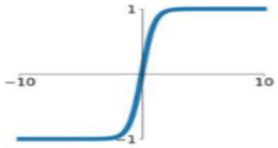
Cons:

- **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:

Function	Equation	Range	Derivative
Tanh (Hyperbolic tangent)	$f(x) = \frac{2}{1+e^{-2x}} - 1$	-1,1	$f'(x) = 1 - f(x)^2$



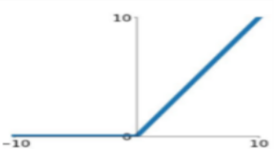
neuron reaches the minimum or maximum value of its range. that

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):

Function	Equation	Range	Derivative
ReLu (Rectified Linear Unit)	$f(x) = \begin{cases} 0 \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$	$0, +\infty$	$f'(x) = \begin{cases} 0 \ for \ x < 0 \\ 1 \ for \ x \geq 0 \end{cases}$

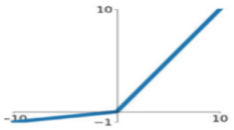


Problem of Dying neuron/Dead neuron : As the ReLu derivative f'(x) is not 0 for the positive values of the neuron (f'(x)=1 for x ≥ 0),

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:

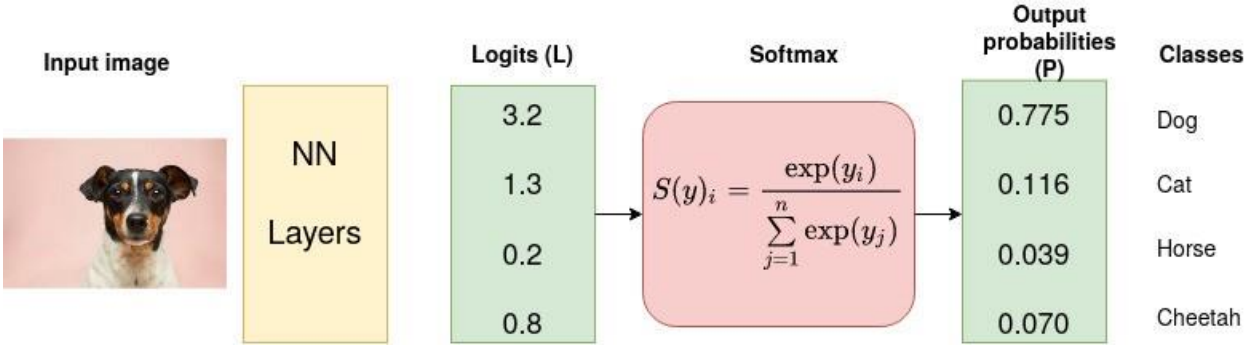
Function	Equation	Range	Derivative
Leaky ReLu	$f(x) = \begin{cases} \alpha x \ for \ x < 0 \\ x \ \ for \ x \geq 0 \end{cases}$	$-\infty, +\infty$	$f'(x) = \begin{cases} \alpha \ for \ x < 0 \\ 1 \ for \ x \geq 0 \end{cases}$



α =0.01

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

SoftMax Activation Function:



$$\exp(3.2) = 24.5325$$
$$\exp(1.3) = 3.6693$$
$$\exp(0.2) = 1.2214$$
$$\exp(0.8) = 2.2255$$

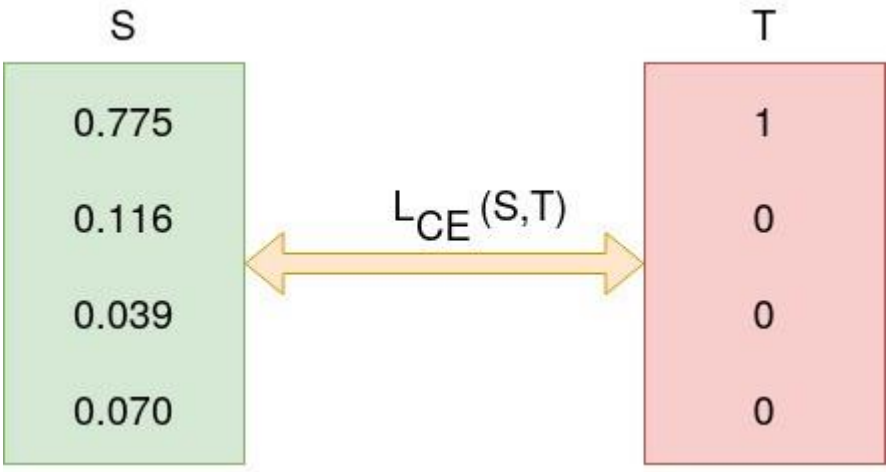
and therefore,

$$S(3.2) = \frac{\exp(3.2)}{\exp(3.2) + \exp(1.3) + \exp(0.2) + \exp(0.8)}$$
$$= \frac{24.5325}{24.5325 + 3.6693 + 1.2214 + 2.2255}$$
$$= 0.775$$

Loss Functions:

Cross-Entropy Loss Function:

The purpose of the Cross-Entropy is to take the output probabilities (P) and measure the distance from the truth values.

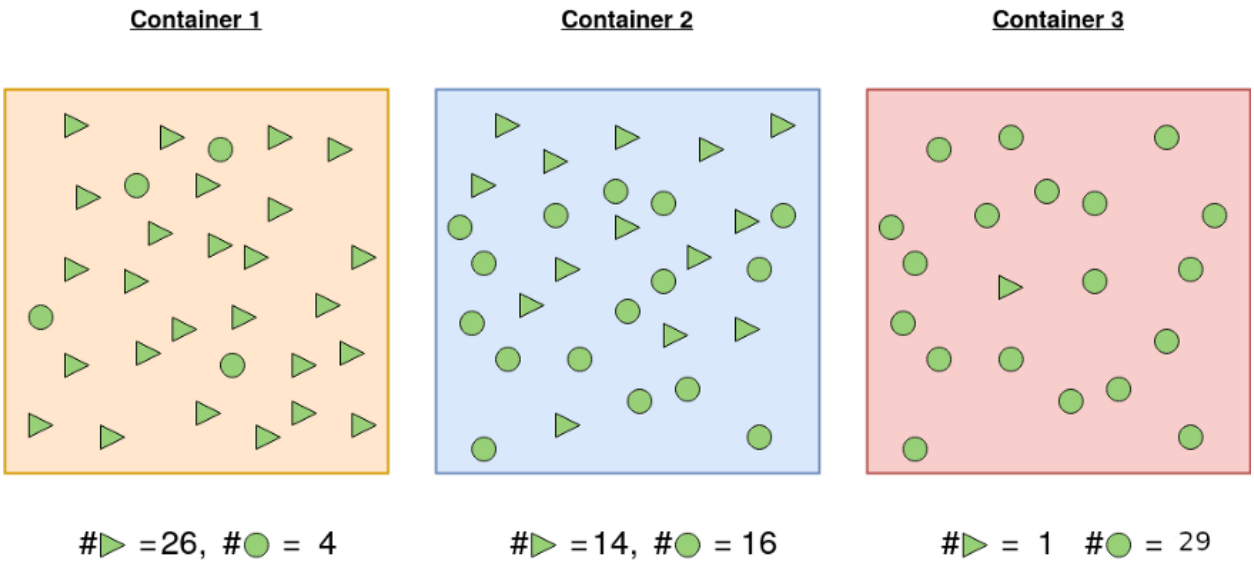


Machine Learning theoretical concepts

By: Vikram Pal

$$H(X) = \begin{cases} -\int_x p(x) \log p(x), & \text{if } X \text{ is continous} \\ -\sum_x p(x) \log p(x), & \text{if } X \text{ is discrete} \end{cases}$$

For Example:



Container 1: The probability of picking a triangle is 26/30 and the probability of picking a circle is 4/30. For this reason, the probability of picking one shape and/or not picking another is more certain.

Container 2: Probability of picking a triangular shape is 14/30 and 16/30 otherwise. There is almost 50–50 chance of picking any particular shape. Less certainty of picking a given shape than in 1.

Container 3: A shape picked from container 3 is highly likely to be a circle. Probability of picking a circle is 29/30 and the probability of picking a triangle is 1/30. It is highly certain than the shape picked will be circle.

Let us calculate the entropy so that we ascertain our assertions about the certainty of picking a given shape.

Entropy for container 1:

$$\begin{aligned} H(X) &= -\sum_{x} p(x)\log(p(x)) \\ &= -[p(x_1)\log_2(p(x_1)) + p(x_2)\log_2(p(x_2))] \\ &= -\left[\frac{26}{30}\log_2\left(\frac{26}{30}\right) + \frac{4}{30}\log_2\left(\frac{4}{30}\right)\right] \\ &= 0.5665 \end{aligned}$$

Entropy for container 2:

$$\begin{aligned} H(X) &= -\sum_{x} p(x)\log(p(x)) \\ &= -[p(x_1)\log_2(p(x_1)) + p(x_2)\log_2(p(x_2))] \\ &= -\left[\frac{14}{30}\log_2\left(\frac{14}{30}\right) + \frac{16}{30}\log_2\left(\frac{16}{30}\right)\right] \\ &= 0.9968 \end{aligned}$$

Entropy for container 3:

$$\begin{aligned} H(X) &= -\sum_{x} p(x)\log(p(x)) \\ &= -[p(x_1)\log_2(p(x_1)) + p(x_2)\log_2(p(x_2))] \\ &= -\left[\frac{1}{30}\log_2\left(\frac{1}{30}\right) + \frac{29}{30}\log_2\left(\frac{29}{30}\right)\right] \\ &= 0.2108 \end{aligned}$$

Binary Cross-Entropy Loss:

$$\begin{aligned} L &= -\sum_{i=1}^{2} t_i \log(p_i) \\ &= -\left[t\log(p) + (1-t)\log(1-p)\right] \end{aligned}$$

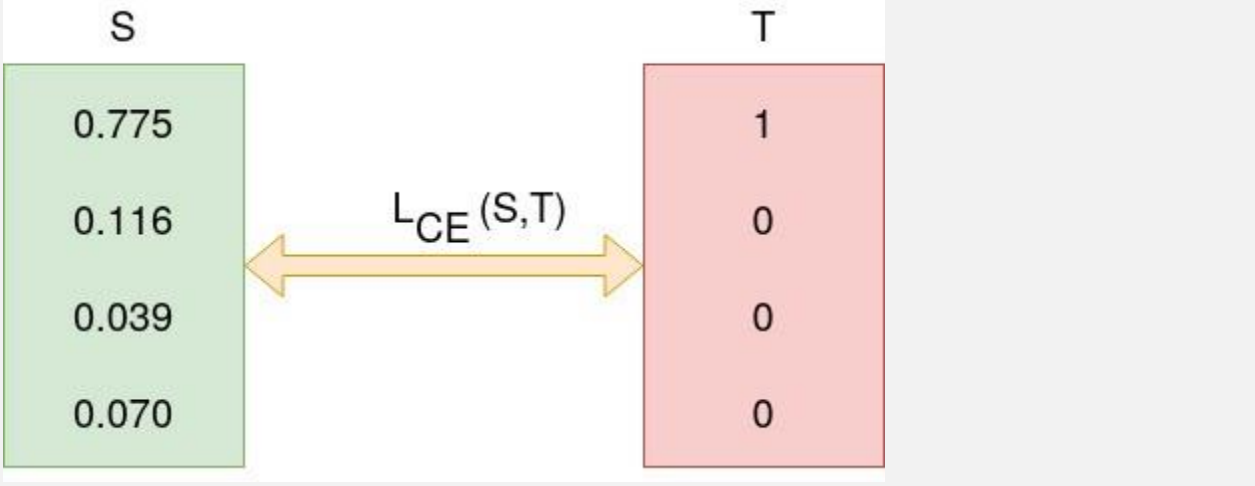
where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} class.

Example:

Machine Learning theoretical concepts

By: Vikram Pal

Consider the classification problem with the following SoftMax probabilities (S) and the labels (T). The objective is to calculate for cross-entropy loss given this information.



Logits(S) and one-hot encoded truth label(T) with Categorical Cross-Entropy loss function used to measure the 'distance' between the predicted probabilities and the truth labels. (Source: Author)

The categorical cross-entropy is computed as follows

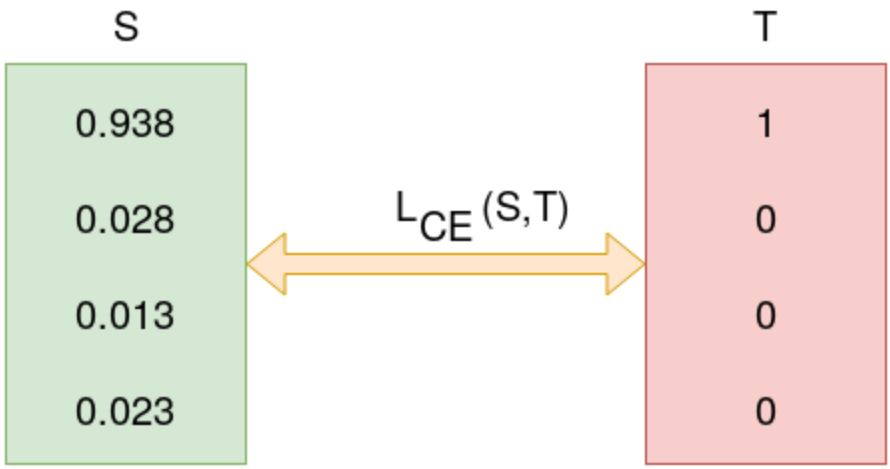
$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -\left[1 \log_2(0.775) + 0 \log_2(0.126) + 0 \log_2(0.039) + 0 \log_2(0.070)\right] \\
&= -\log_2(0.775) \\
&= 0.3677
\end{aligned}
$$

SoftMax is continuously differentiable function. This makes it possible to calculate the derivative of the loss function with respect to every weight in the neural network.
This property allows the model to adjust the weights accordingly to minimize the loss function (model output close to the true values).

Assume that after some iterations of model training the model outputs the following vector of logits

Machine Learning theoretical concepts

By: Vikram Pal



$$L_{CE} = -1\log_2(0.936) + 0 + 0 + 0$$
$$= 0.095$$

0.095 is less than previous loss, that is, 0.3677 implying that the model is learning. The process of optimization (adjusting weights so that the output is close to true values) continues until training is over.

Keras provides the following cross-entropy loss functions: binary, categorical, sparse categorical cross-entropy loss functions.

Categorical Cross-Entropy and Sparse Categorical Cross-Entropy:

Both categorical cross entropy and sparse categorical cross-entropy have the same loss function as defined in Equation 2. The only difference between the two is on how truth labels are defined.

- Categorical cross-entropy is used when true labels are one-hot encoded, for example, we have the following true values for 3-class classification problem [1,0,0], [0,1,0] and [0,0,1].
- In sparse categorical cross-entropy , truth labels are integer encoded, for example, [1], [2] and [3] for 3-class problem.

Computer Vision:

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/?utm_source=blog&utm_medium=computer-vision-implementing-mask-r-cnn-image-segmentation

Image Data Augmentation:

Deep learning models perform well when we have a large amount of data. There are quite a few domains where getting enough data is a problem. In such cases, we use data augmentation to generate training data from the available data. Some of the common augmentation methods are:

- Mirroring: Here we take the mirror image. The class of the image will not change in this case
- Random Cropping
- Rotating
- Shearing
- Color Shifting: We change the RGB scale of the image randomly.

Image Pre-Processing:

There are 4 different types of Image Pre-Processing techniques, and they are listed below.

- Pixel brightness transformations/ Brightness corrections
- Geometric Transformations
- Image Filtering and Segmentation
- Fourier transform and Image restauration

Pixel brightness transformations(PBT):

Brightness transformations modify pixel brightness, and the transformation depends on the properties of a pixel itself.

There are two types of Brightness transformations, and they are below.

- Brightness corrections
- Gray scale transformation

The most common Pixel brightness transforms operations are

- Gamma correction or Power Law Transform: **A non-linear adjustment to individual pixel values.** Gamma correction carries out a non-linear operation on the source image pixels, and can cause saturation of the image being altered
- Sigmoid stretching: a **continuous nonlinear activation function**. Parameters 'c' and threshold value it is possible to tailor the amount of lightening and darkening to control the overall contrast enhancement.
- Histogram equalization: Contrast of an image by altering that image such that its intensity histogram has the desired shape.

Geometric Transformations:

Geometric transforms permit the elimination of geometric distortion that occurs when an image is captured.

It is basically two types:

1. Spatial transformation:

the physical rearrangement of pixels in the image

Transformations :

Geometric transforms permit the elimination of geometric distortion that occurs when an image is captured.

1. Scaling : Scaling is just resizing of the image

2. Translation : Translation is the shifting of object's location

3. Rotation : Just rotating an object with theta degrees

4. Shearing : Shifting the pixels horizontally

2. Grey level interpolation:

which assigns grey levels to the transformed image

Different types of Interpolation methods are

1.Nearest neighbor interpolation is the simplest technique that re samples the pixel values present in the input vector or a matrix

2. Linear interpolation explores four points neighboring the point (x,y), and assumes that the brightness function is linear in this neighborhood.

Image Filtering and Segmentation:

The goal of using filters is to modify or enhance image properties and/or to extract valuable information from the pictures such as edges, corners, and blobs

Some of the basic filtering techniques are

- Low Pass Filtering (Smoothing)
- High pass filters (Edge Detection, Sharpening)
- Laplacian Filtering: Laplacian filter is an edge detector used to compute the second derivatives of an image, measuring the rate at which the first derivatives change. This determines if a change in adjacent pixel values is from an edge or continuous progression.

Fourier transform:

The Fourier Transform is an important image processing tool which is used to **decompose an image into its sine and cosine components**.

The **output of the transformation** represents **the image in the Fourier or frequency domain**, while the input image is the **spatial domain equivalent**.

In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

The Fourier Transform is used in a wide range of applications, such as image analysis, image filtering, image reconstruction and image compression.

The DFT(Discrete Fourier Transform) is **the sampled Fourier Transform** and therefore does not contain all frequencies forming an image, but only a **set of samples which is large enough to fully describe the spatial domain image**. The number of frequencies corresponds to the number of pixels in the spatial domain image, i.e. the image in the spatial and Fourier domain are of the same size.

How do we prevent misclassification of noise as an edge?

1. Using LPF, **remove the impulsive increase in the intensity values occurring in the image**

2. Once we have the result of Laplacian()(medianBlur(), and GaussianBlur().), we can invert it to get black edges on a white background. we can normalize it (so that its values range from 0 to 1) and multiply it with the source image to darken the edges

Contour detection

Another important computer vision technique is called contour detection that has gained popularity among scientists and researchers. It deals with the aspect of detecting contours in ROI in an image. It gains importance because of its derivative operations connected with identifying contours.

What are the operations related to contour detection?

1. generating bounding boxes

2. approximating shapes

3. Calculating ROIs

LARGE KERNEL PAN:

Large Kernel PAN (Large Kernel Pyramid Attention Network) is a type of deep neural network architecture that is used for image classification tasks. It is a variant of the Pyramid Attention Network (PAN) architecture, which is a type of convolutional neural network that uses a pyramid-like structure to capture multi-scale features from an input image.

The Large Kernel PAN architecture is based on the idea of using large kernels in the convolutional layers of the network to capture more global context and enhance the representation power of the network. This is achieved by replacing the traditional 3x3 kernels in the convolutional layers with larger kernels, such as 5x5 or 7x7.

In addition to the large kernels, the Large Kernel PAN architecture also includes an attention module, which allows the network to focus on the most relevant features in the input image. This helps the network to better handle complex and cluttered images, and can improve the overall accuracy and efficiency of the image classification system.

Overall, the Large Kernel PAN architecture is a powerful and widely used tool for image classification tasks, and has achieved good performance on a number of benchmarks and real-world datasets.

Evaluation Matrics in computer Vision:

Peak signal-to-noise ratio (PSNR):

it is the ratio between the maximum possible power of an image and the power of corrupting noise that affects the quality of its representation. To estimate the PSNR of an image, it is necessary to compare that image to an ideal clean image with the maximum possible power.

PSNR is defined as follows:

Machine Learning theoretical concepts

By: Vikram Pal

$$PSNR = 10log_{10}(\frac{(L-1)^2}{MSE}) = 20log_{10}(\frac{L-1}{RMSE})$$

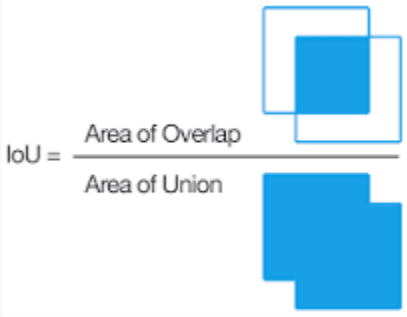
Structural Similarity Index metric:

SSIM is used as a metric to measure the similarity between two given images.
The Structural Similarity Index (SSIM) metric extracts 3 key features from an image:

- Luminance (averaging over all the pixel values)
- Contrast (standard deviation (square root of variance) of all the pixel values.)
- Structure

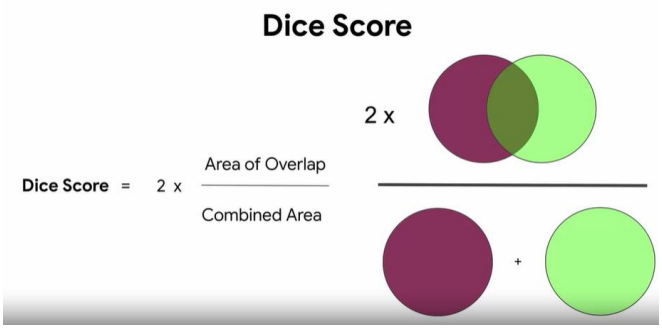
It varies b/w -1 and +1. +1 means two images are similar and -1 means two images are totally different.

IOU (intersection over union):



Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. Any algorithm that provides predicted bounding boxes as output can be evaluated using IoU.

Dice Score:



The subtle difference between them is that the dice score tends to veer towards the average performance. Whereas the IOU helps you understand worst case performance.

Inception Score:

The Inception Score, or IS for short, is an objective metric for evaluating the quality of generated images, specifically synthetic images output by generative adversarial network models. They developed the inception score as an attempt to remove the subjective human evaluation of images.

Machine Learning theoretical concepts

By: Vikram Pal

Frechet Inception Distance:

FID is a **metric for evaluating the quality of generated images** and specifically developed to evaluate the performance of generative adversarial networks

Model Visualization Technique:

feature maps :

The **feature maps** of a CNN capture the **result of applying the filters to an input image**. I.e at each layer, the feature map is the output of that layer. The reason for visualizing a feature map for a specific input image is to try to **gain some understanding of what features our CNN detects**. Perhaps **it detects some parts of our desired object and not others or the activations die out at a certain layer**.

saliency map:

The saliency map is an explanation method used for interpreting the predictions of Artificial neural networks (ANNs). This is probably the oldest and the most frequently used method of interpretation in deep learning. Basically, **the saliency map of an input image specifies parts of it that contribute the most to the activity of a specific layer in the network, or the decision of the network as a whole.**

- The main difference is in saliency maps, we are just shown the **relevant pixels instead of the learned features.**
- You can generate saliency maps by getting the **gradient of the loss with respect to the image pixels.**
- This means that changes in certain pixels that **strongly affect the loss will be shown brightly in your saliency map.**

Activation maps:

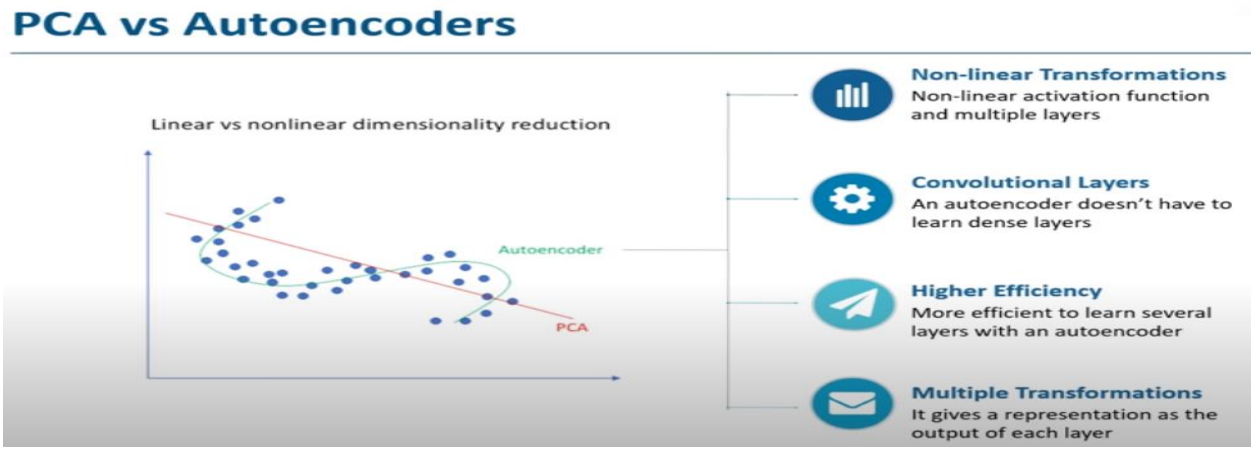
are a simple technique to get the discriminative image regions used by a CNN to identify a specific class in the image. In other words, **a class activation map (CAM) lets us see which regions in the image were relevant to this class**.

Autoencoder:

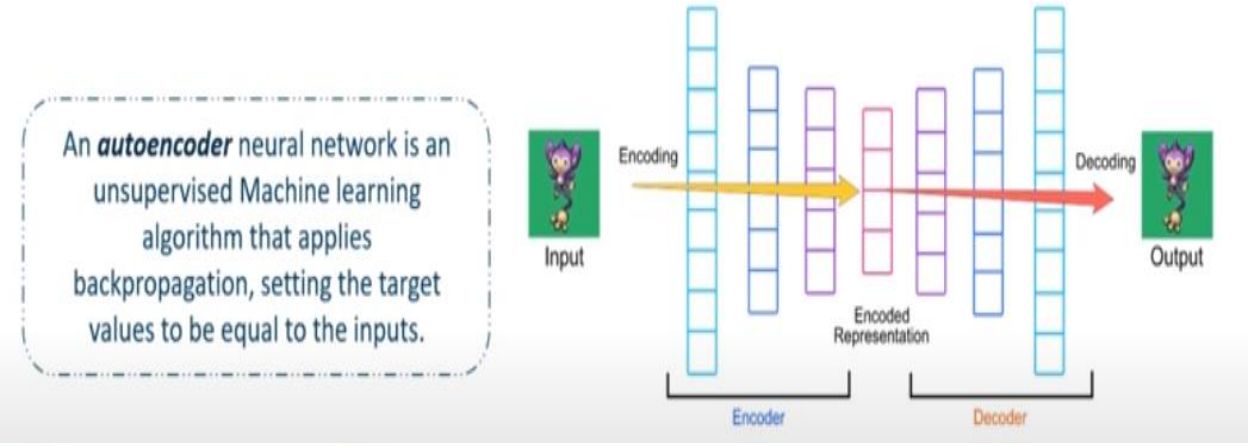
https://www.youtube.com/watch?v=nTt_ajul8NY
https://blog.keras.io/building-autoencoders-in-keras.html

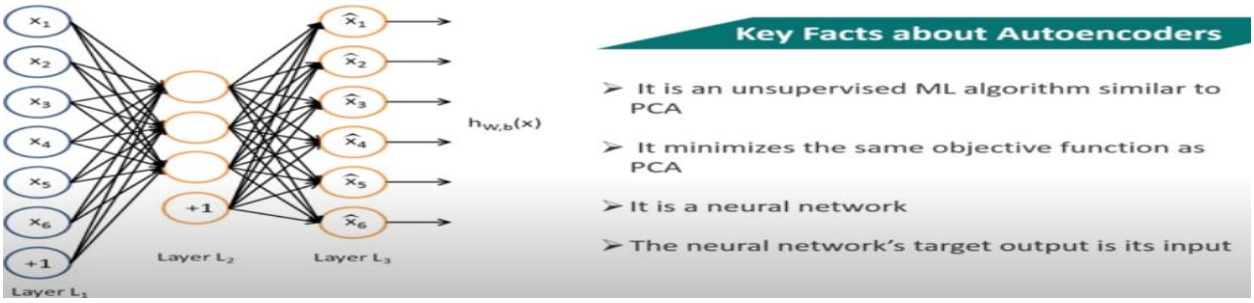
Machine Learning theoretical concepts

By: Vikram Pal



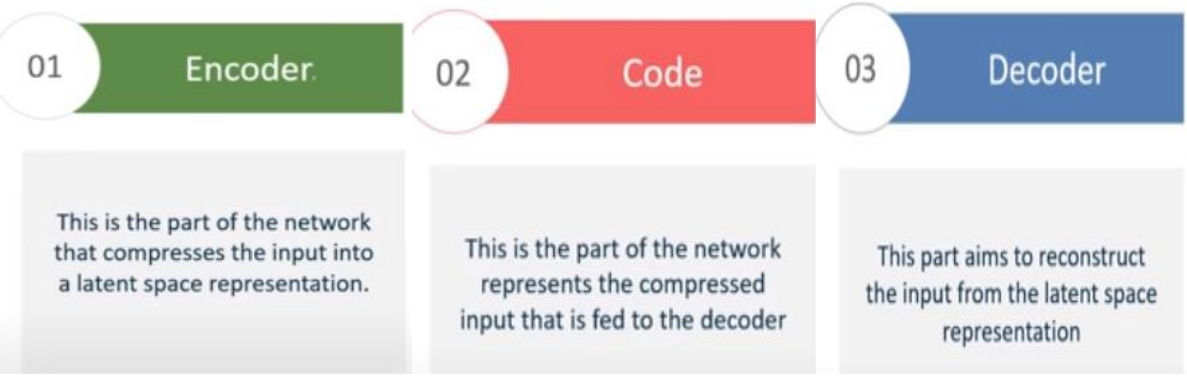
The biggest advantage of Autoencoder is that we can apply fine tuning. That mean we can use already trained learning from some place to new model. By that means we don't need to train model again and again. Two main applications of autoencoder are: 1. Denoising 2. Dimensionality Reduction.

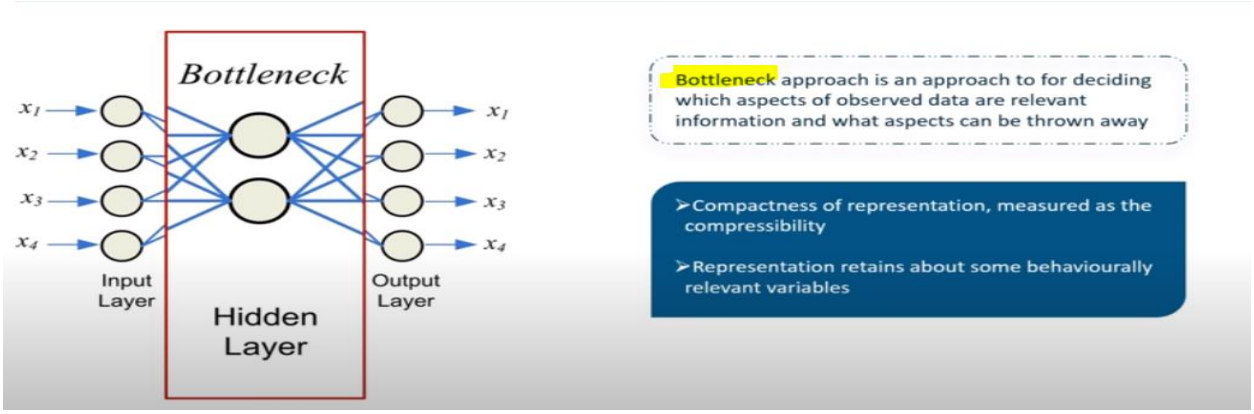




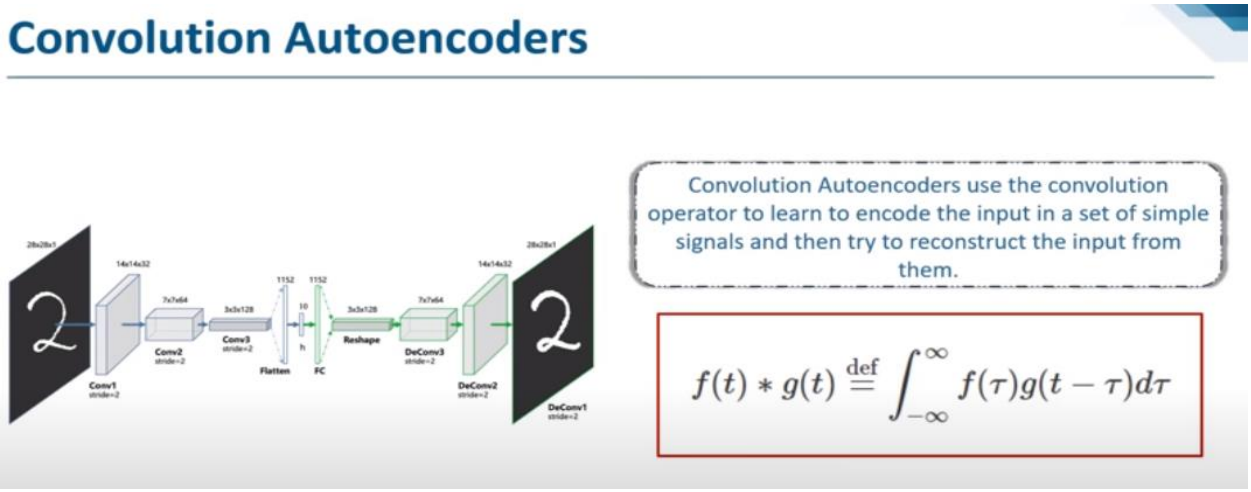
Machine Learning theoretical concepts

By: Vikram Pal





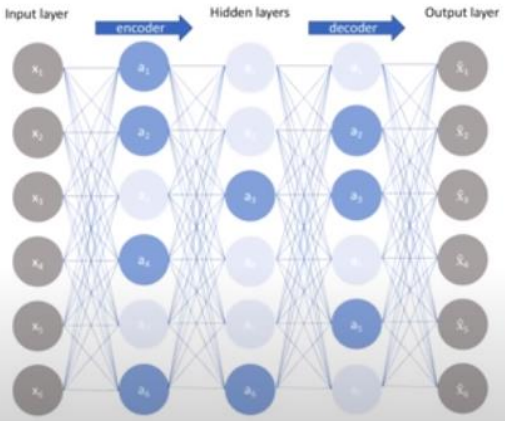
Convolution Autoencoders



Machine Learning theoretical concepts

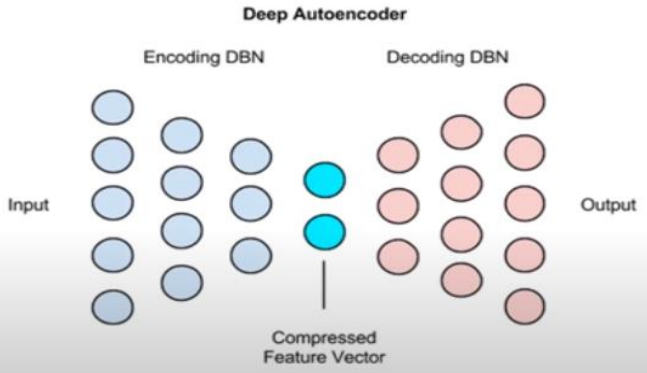
By: Vikram Pal

Sparse Autoencoders



Sparse autoencoders offer us an alternative method for introducing an information bottleneck without requiring a reduction in the number of nodes at our hidden layers

Deep Autoencoders



A ***deep autoencoder*** is composed of two, symmetrical deep-belief networks-

- First four or five shallow layers representing the encoding half of the net
- second set of four or five layers that make up the decoding half.

Use Case: Deep Autoencoders

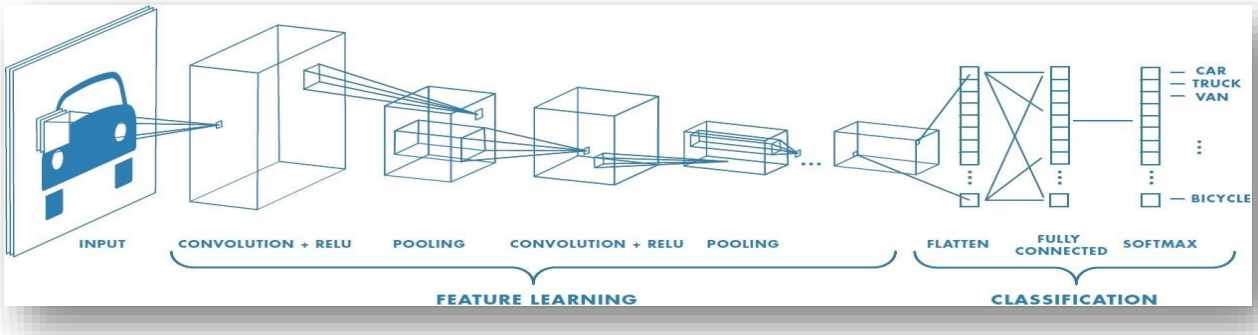
1. Image Search
2. Data Compression
3. Topic Modeling & Information Retrieval

Machine Learning theoretical concepts

By: Vikram Pal

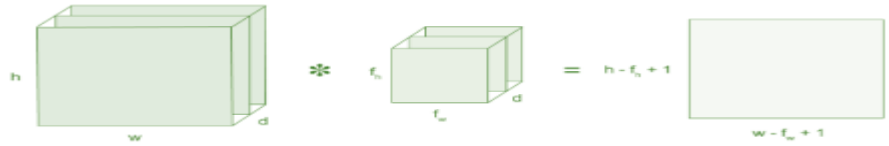
Convolution Neural Network:

CNNs are fully connected feed forward neural networks. CNNs are very effective in reducing the number of parameters without losing on the quality of models. Images have high dimensionality

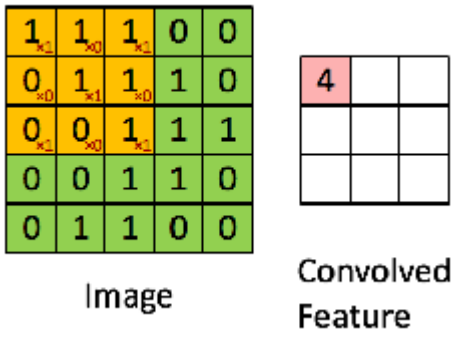


$$output = \frac{input - kernel\_size + 2 * padding}{stride} + 1$$

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**



Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “Feature Map” as output shown in below



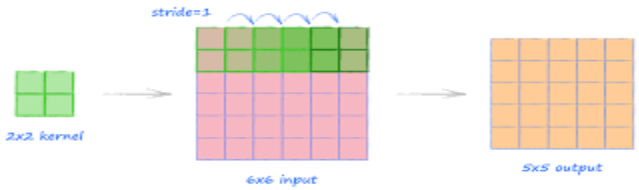
Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

Strides:

Stride is the number of pixels shifts over the input matrix.

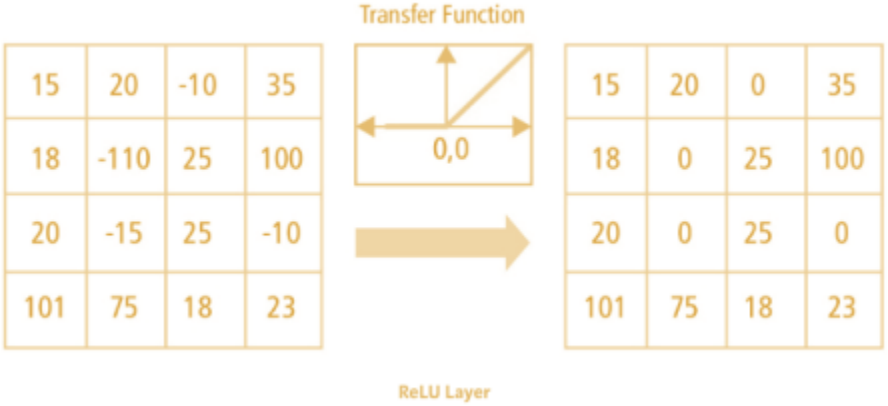
Machine Learning theoretical concepts

By: Vikram Pal



Non-Linearity (ReLU):

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ƒ(x) = max(0,x).
Why ReLU is important : ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real-world data would want our ConvNet to learn would be non-negative linear values.



Pooling Layer: It reduce the number of parameter when the image is too large.

a. Max Pooling
b. Average Pooling
c. Sum Pooling

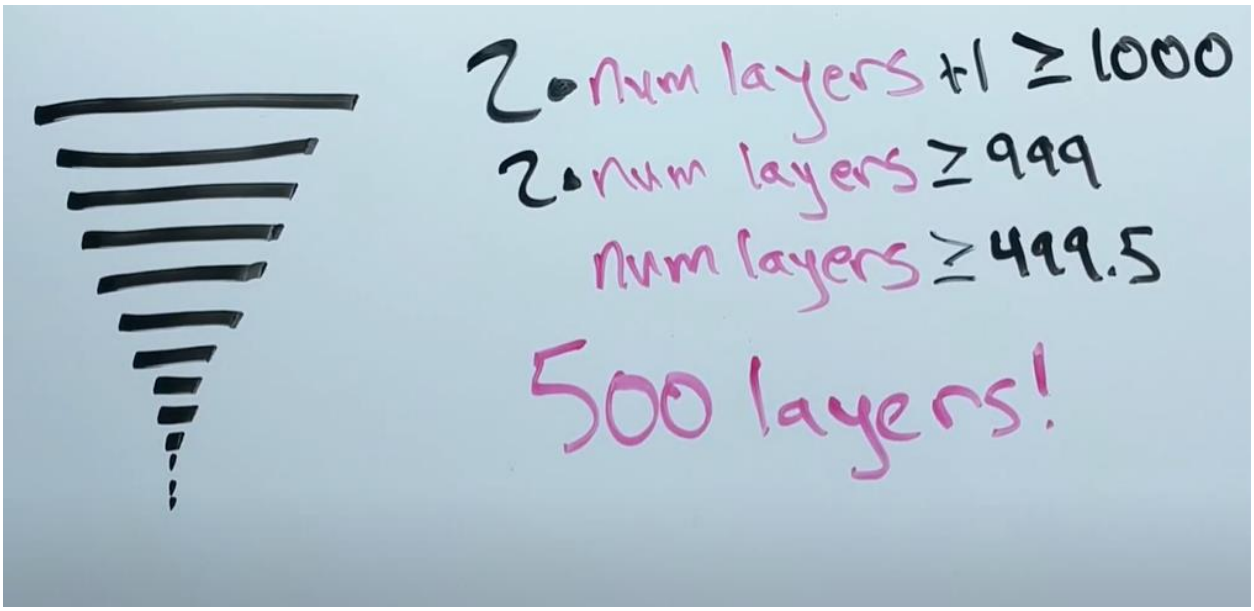
Fully Connected Layer:

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

Receptive Field:

Machine Learning theoretical concepts

By: Vikram Pal



receptive field formula: 2* number of layers+1 >= image dimension (e.g., image size 1000*1000, in this case image dimension will be 1000)
It is used to calculate how many numbers of layers required to focus on all details of one image.

https://www.youtube.com/watch?v=70A3uYfM1qA

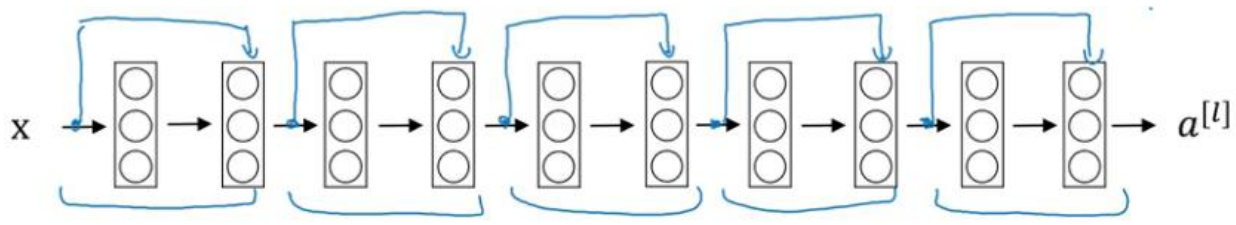
ResNet:

Training very deep networks can lead to **problems like vanishing and exploding gradients**.

How do we deal with these issues?

We can **use skip connections where we take activations from one layer and feed it to another layer that is even more deeper in the network**. There are residual blocks in ResNet which help in training deeper networks.

The residual network can be shown as:

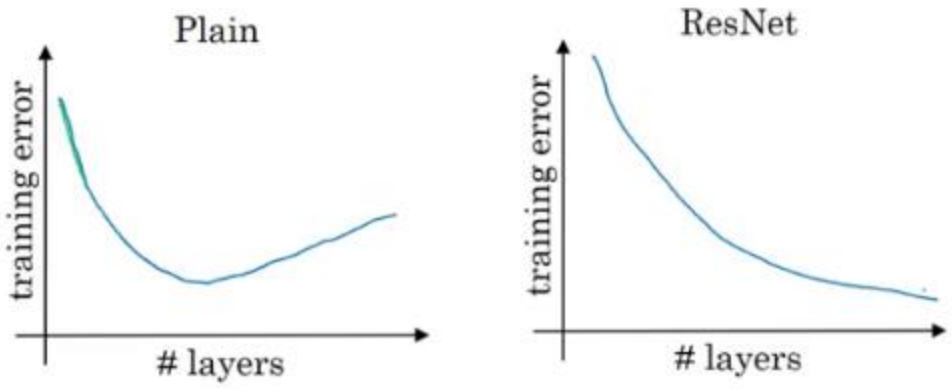


Benefit of training a residual network

The benefit of training a residual network is that even **if we train deeper networks, the training error does not increase.** Whereas in case of a plain network, the training error first decreases as we train a deeper network and then starts to rapidly increase:

Machine Learning theoretical concepts

By: Vikram Pal



Why ResNets Work?

In order to make a good model, we first have to make sure that it's performance on the training data is good. That's the first test and there really is no point in moving forward if our model fails here. We have seen earlier **that training deeper networks using a plain network increases the training error after a point of time. But while training a residual network, this isn't the case**. Even when we build a deeper residual network, the training error generally does not increase.

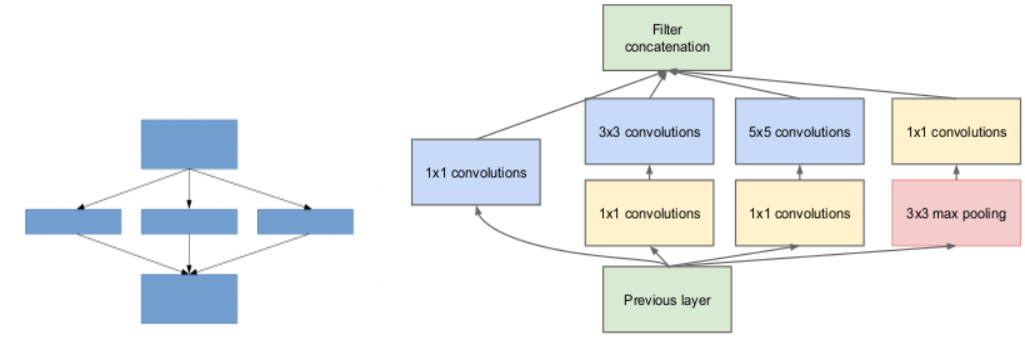
Google-Net/Inception module:

There's a simple but powerful way of creating better deep learning models. **We can just make a bigger model, either in terms of deepness, i.e., number of layers, or the number of neurons in each layer**. But as we can imagine, this can often create complications:

- **Bigger the model, more prone it is to overfitting**. This is particularly noticeable when the training data is small
- **Increasing the number of parameters** means you need to increase your existing computational resources

A solution for this is to move on to **sparsely connected network architectures** which will replace fully connected network architectures, especially inside convolutional layers. This idea can be conceptualized in the below images:

<u>Sparsely Connected Architecture:</u>



Machine Learning theoretical concepts

By: Vikram Pal

Inception Layer:

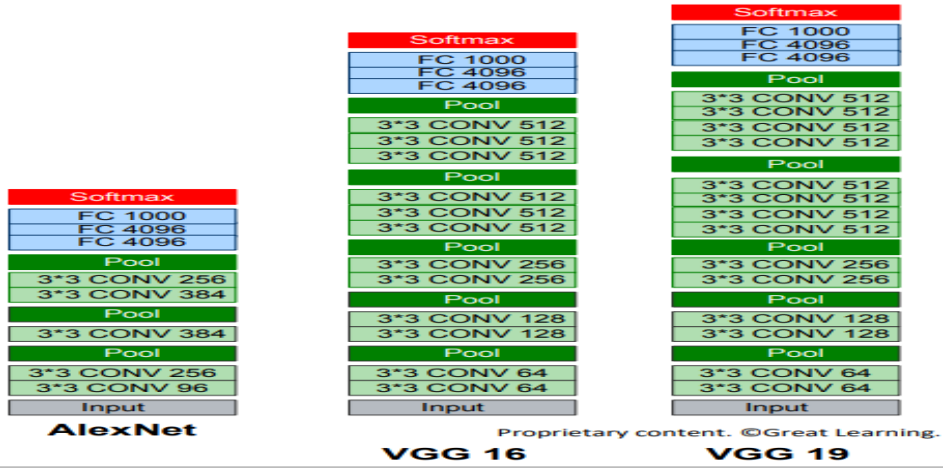
It allows the **internal layers to pick and choose which filter size will be relevant to learn the required information**. So even if the size of the face in the image is different, the layer works accordingly to recognize the face. For the first image, it would probably take a higher filter size, while it'll take a lower one for the second image.

VGGNet:

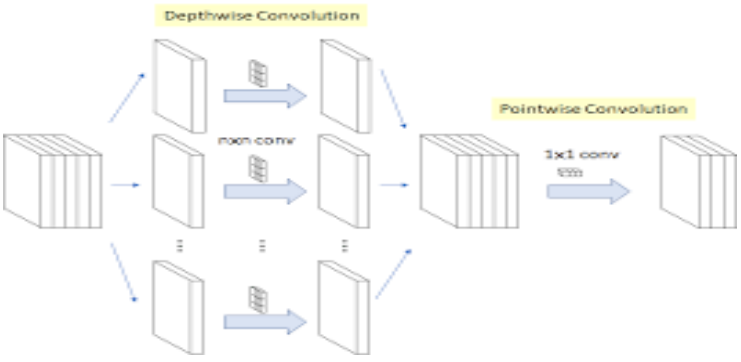
This model used:

- Smaller filters But
- Deeper networks

Why use smaller filters? (3x3 conv) Answer: Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 but deeper, more non-linearities and fewer parameters. 3x3 CONV stride 1, pad 1 2x2 MAX POOL stride



MobileNet:



Depth-wise convolution

Machine Learning theoretical concepts

By: Vikram Pal

In this convolution, we apply a **2-d depth filter at each depth level of input tensor**. Let's understand this through an example. Suppose our input tensor is 3* 8 *8 (input_channels*width* height). Filter is 3*3*3. In a standard convolution we would directly convolve in depth dimension as well (fig 1).

In depth-wise convolution, we use each filter channel only at one input channel. In the example, we have 3 channel filter and 3 channel images. What we do is — **break the filter and image into three different channels and then convolve the corresponding image with corresponding channel and then stack them back** (Fig 2)

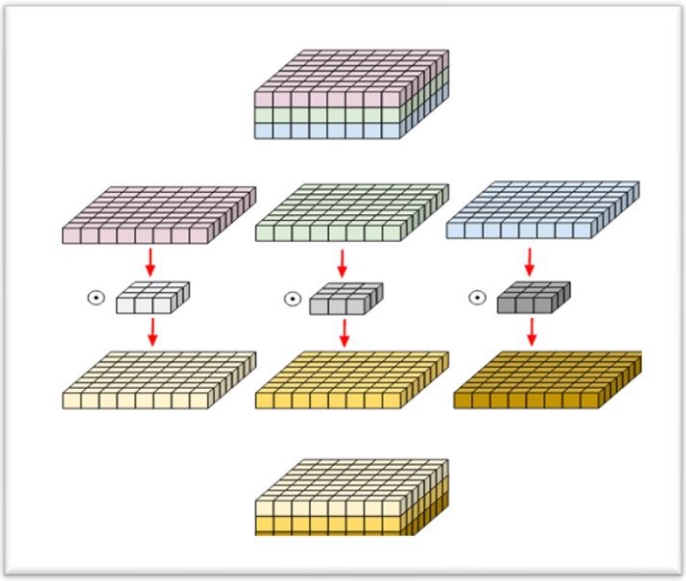
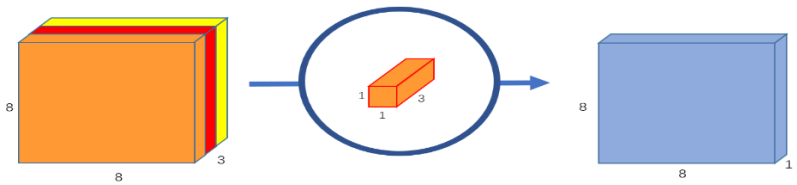


Fig 2. Depth-wise convolution.

Filters and image have been broken into three different channels and then convolved separately and stacked thereafter

Pointwise Convolution:

Pointwise Convolution is a type of convolution that uses a 1x1 kernel: a kernel that iterates through every single point. This kernel has a depth of however many channels the input image has. It can be used in conjunction with depthwise convolutions to produce an efficient class of convolutions known as depthwise-separable convolutions.



Machine Learning theoretical concepts

By: Vikram Pal

SLANet:

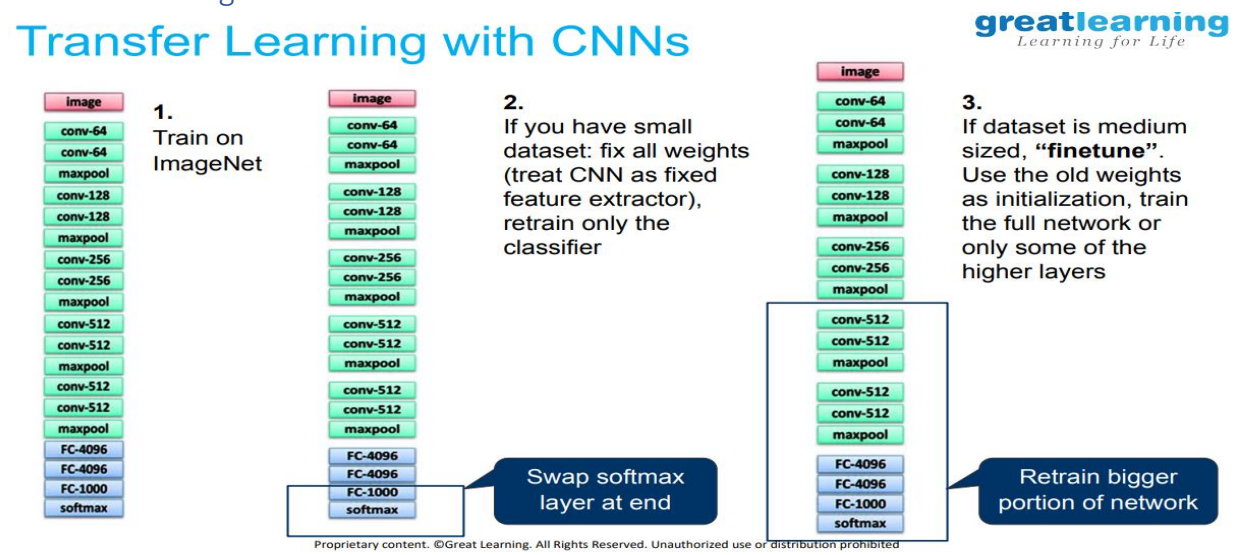
SLANet (Single-Level Attention Network) is a deep learning model that was proposed in the paper "Single-Level Attention Network for Robust Scene Text Recognition" (Zhang et al., 2020). It is a type of convolutional neural network (CNN) that is specifically designed for scene text recognition tasks, which involve recognizing and transcribing text from images of natural scenes.

The main idea behind the SLANet model is to use a single-level attention mechanism to dynamically weight the features extracted from the input image, and to use these weighted features to classify the text. The attention mechanism is implemented as a self-attention module, which allows the network to focus on the most relevant features in the input image and to learn context-aware representations.

The SLANet model is trained using a combination of supervised learning and self-supervised learning, which allows it to learn robust and generalizable features. It has been evaluated on a number of benchmarks and real-world datasets and has achieved good performance on a variety of scene text recognition tasks.

Overall, the SLANet model is a promising approach for scene text recognition and has the potential to be applied to a wide range of applications. However, it is important to note that the performance of any machine learning model will depend on the specific task and dataset it is being used on, and it may not always be the best choice for a given problem.

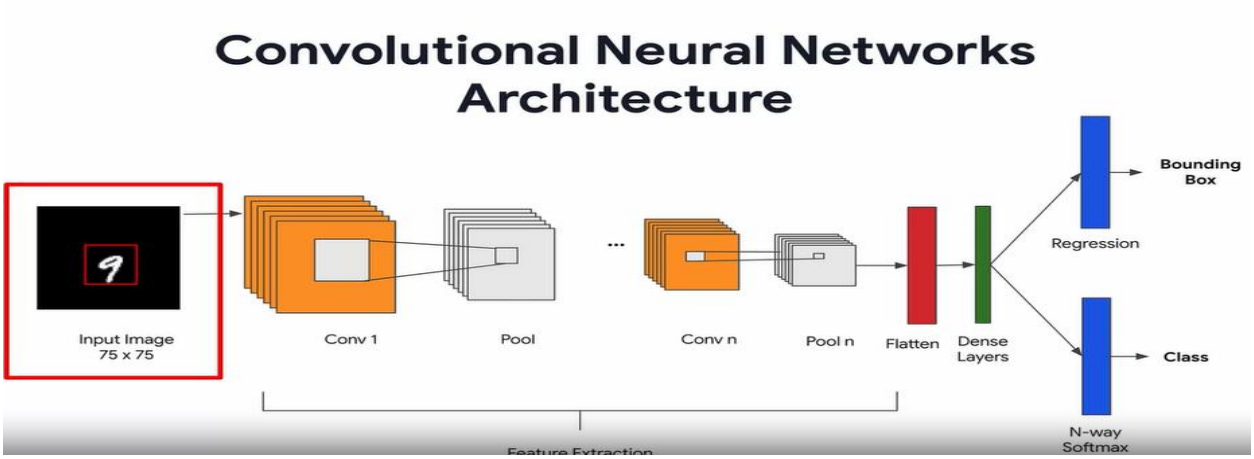
Transfer Learning with CNNs:



Machine Learning theoretical concepts

By: Vikram Pal

Network architecture for Object Localization:

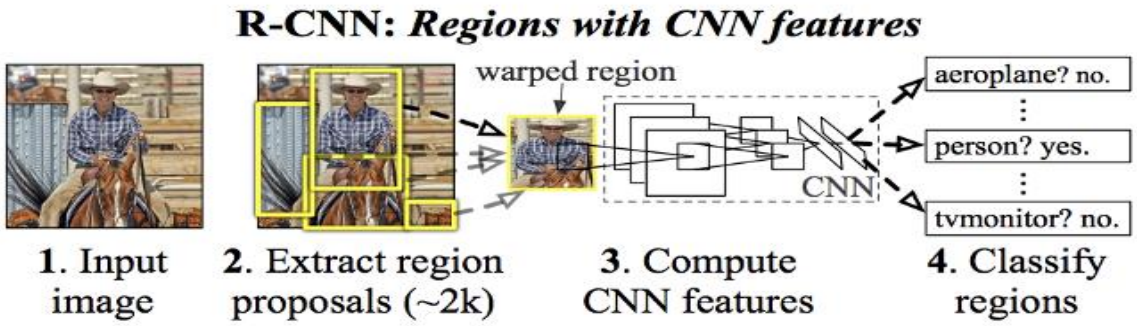


R-CNN:

To bypass the problem of selecting a huge number of regions , we use selective search to extract just 2000 regions from the image, those are region proposals. We generate using the **selective search algorithms**

Selective Search:

1. Generate initial sub-segmentation, we generate many candidate regions
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals



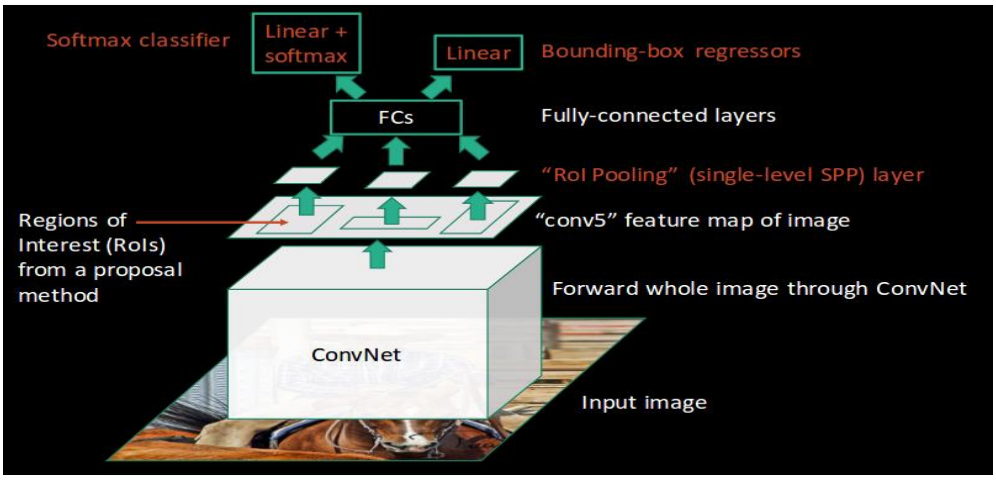
The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an **SVM to classify the presence of the object** within that candidate region proposal

Fast R-CNN:

It is similar to R-CNN. But instead of generating regions, we use CNN to generate a convolutional feature map.

Machine Learning theoretical concepts

By: Vikram Pal



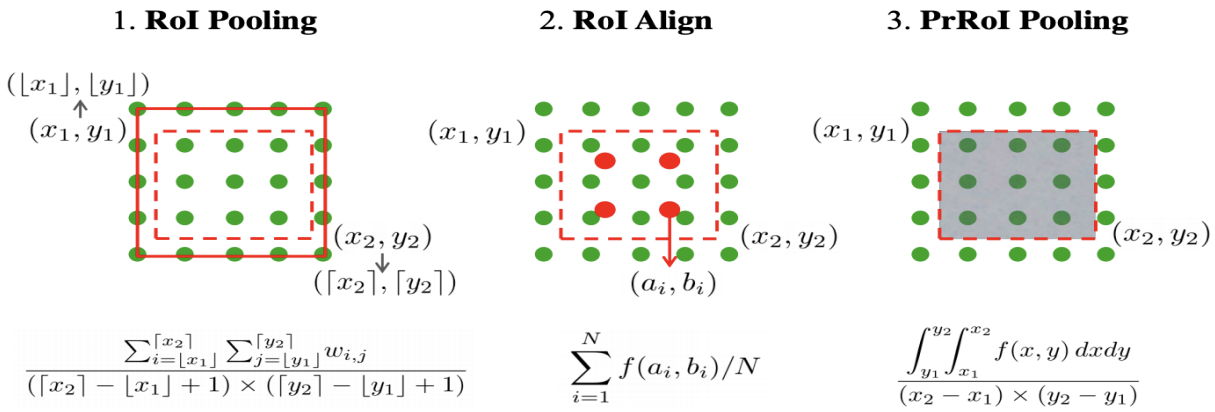
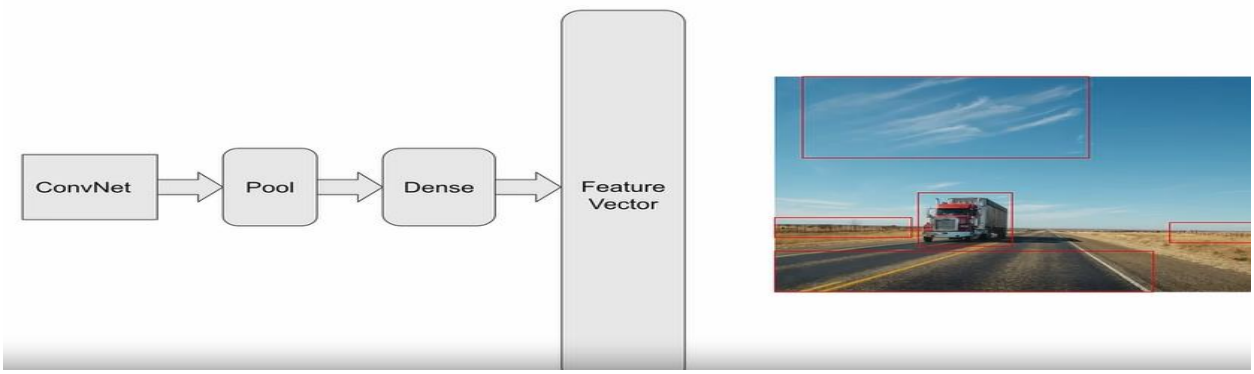
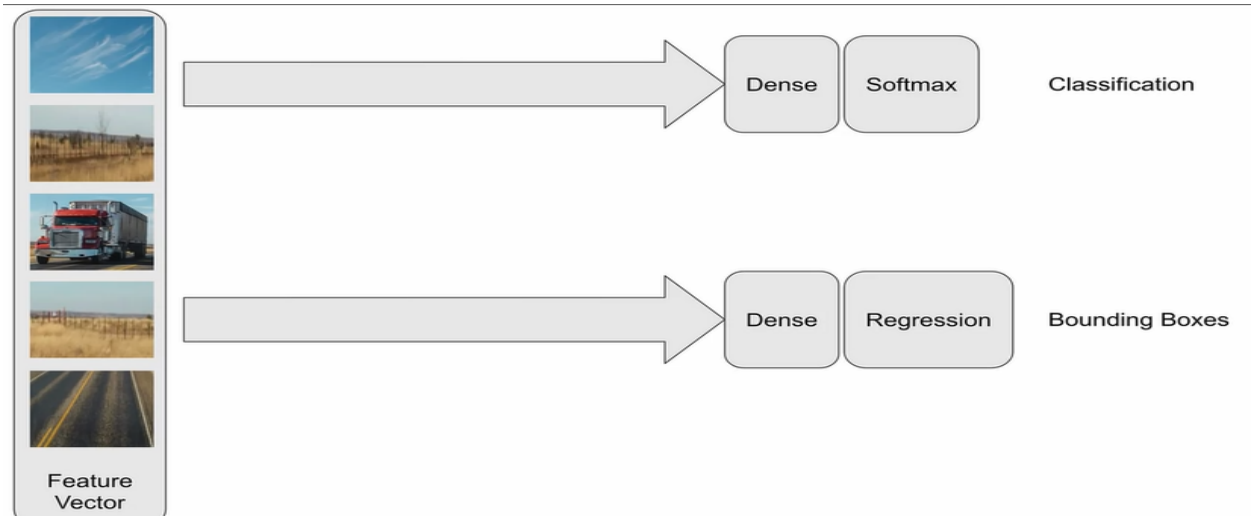


Fig. 6: Illustration of RoI Pooling, RoI Align and PrRoI Pooling.

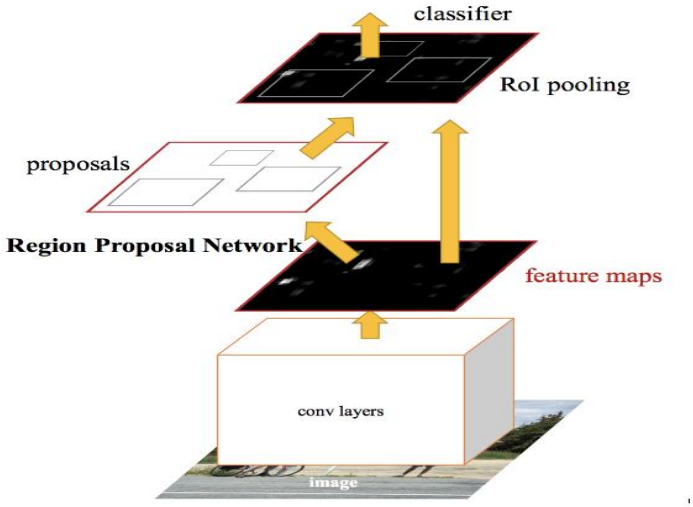




From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI(Region of interest) pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.

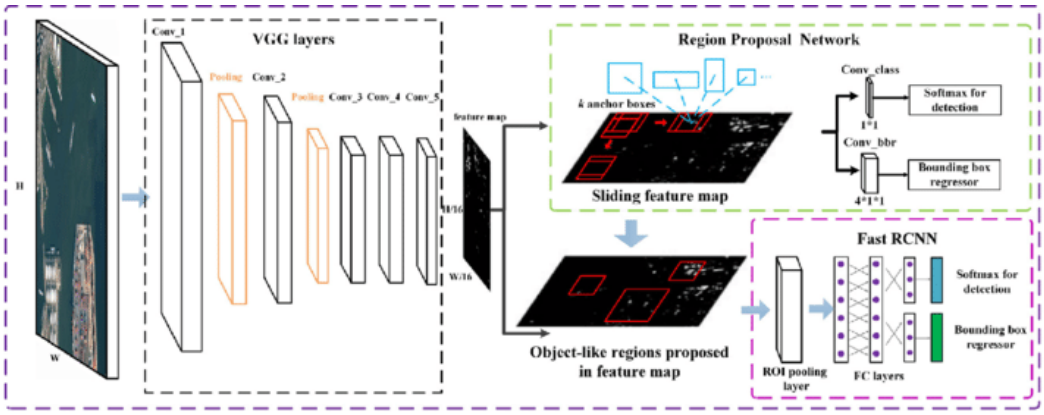
From the ROI feature vector, we use **a Softmax layer to predict** the class of the proposed region and also the offset values for the bounding box.

Faster R-CNN:



Machine Learning theoretical concepts

By: Vikram Pal

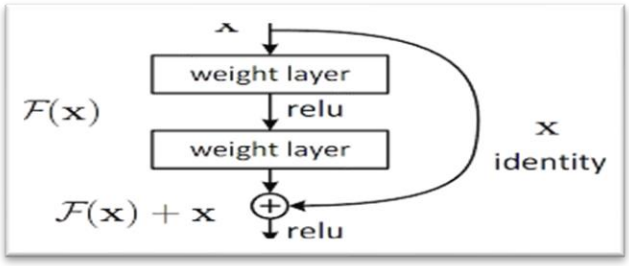


RPN is a **fully convolutional network, trained end-to-end**, that **simultaneously predicts object boundaries and object scores at each detection**

Instead of using selective search algorithm on the feature map to identify the region proposals, a **separate network is used to predict the region proposals**. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

All of the previous object detection algorithms use regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which have high probabilities of containing the object.

Residual Network:



YOLO:

In YOLO a single convolutional network predicts the **bounding boxes and the class probabilities** for these boxes

Machine Learning theoretical concepts

By: Vikram Pal

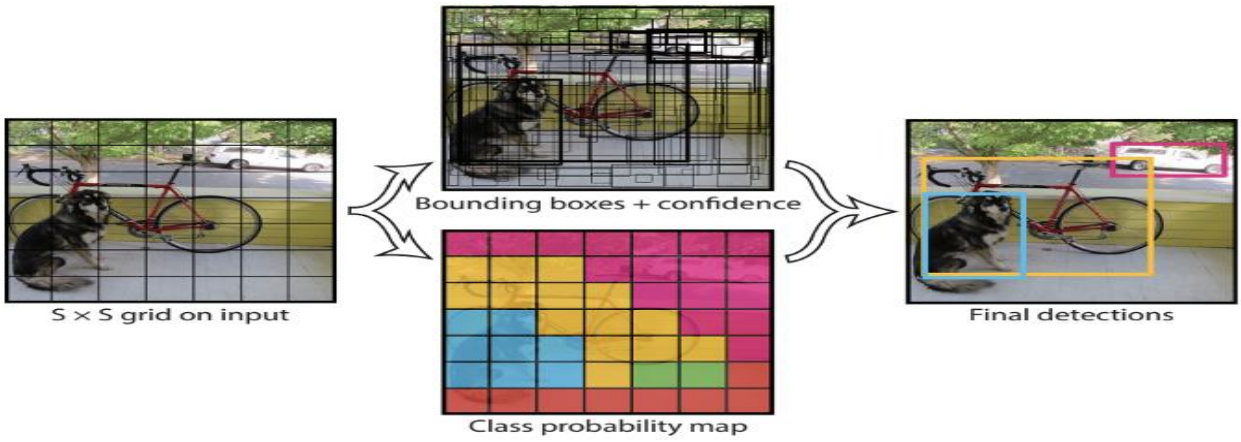
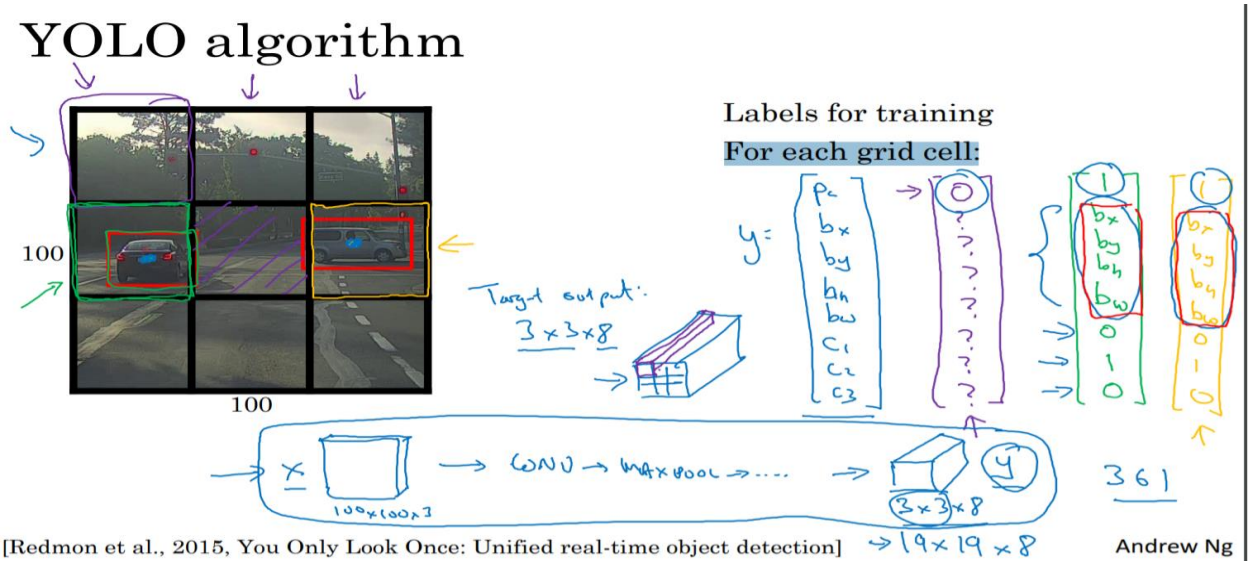


Image split (S*S) grid → each grid m bounding boxes → Each bounding box (network o/p class probability and offset values) → bounding box having probability above a threshold value is selected

How YOLO works is that we take an i**mage and split it into an SxS grid,** within each of the grid we take m bounding boxes. For each of the bounding box, the network outputs a class probability and **offset values for the bounding box**. **The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image**.

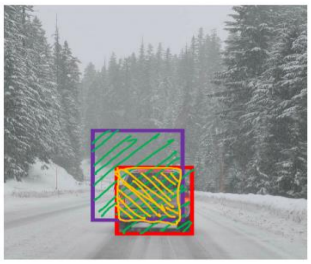


IOU= Intersecton/overall region

Machine Learning theoretical concepts

By: Vikram Pal

Evaluating object localization



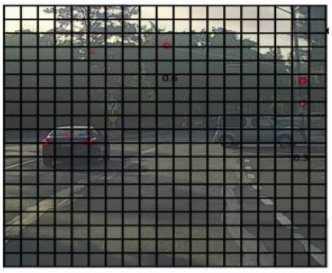
Intersection over Union (IoU)

$= \frac{\text{size of}}{\text{size of}}$

"Correct" if IoU $\geq$ 0.5 ←

0.6 ←

Non-max suppression algorithm



19×19

Each output prediction is: p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3

Discard all boxes with $p_c \leq 0.6$

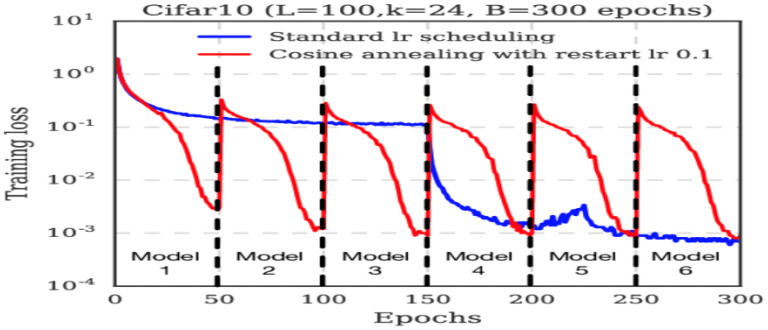
While there are any remaining boxes:

- Pick the box with the largest p_c Output that as a prediction.
- Discard any remaining box with IoU $\geq$ 0.5 with the box output in the previous step

How can I improve my Yolo accuracy?

Different Training Heuristics for Object Detection

- Image mix-up with geometry preserved alignment.
- Using **cosine learning rate scheduler**.
- Synchronized batch normalization.
- Data augmentation.
- Label smoothing.



Machine Learning theoretical concepts

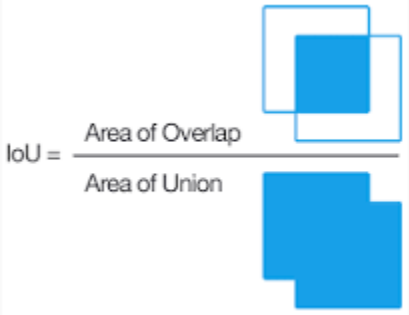
By: Vikram Pal

Evaluating Object Detection Models Using Mean Average Precision:

https://www.kdnuggets.com/2021/03/evaluating-object-detection-models-using-mean-average-precision.html

IOU:

The object detection models are evaluated with different IoU thresholds where each threshold may give different predictions from the other thresholds.



Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. Any algorithm that provides predicted bounding boxes as output can be evaluated using IoU.

$$class(IoU) = \begin{cases} Positive \rightarrow IoU \geq Threshold \\ Negative \rightarrow IoU < Threshold \end{cases}$$

Average Precision:

The average precision (AP) is a way to **summarize the precision-recall curve into a single value representing the average of all precisions**. The AP is calculated according to the next equation. Using a loop that goes through all precisions/recalls, the **difference between the current and next recalls is calculated and then multiplied by the current precision**.

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k)$$

$$Recalls(n) = 0, Precisions(n) = 1$$

$$n = Number\ of\ thresholds.$$

Mean Average Precision:

To calculate the mAP, start by calculating the AP for each class. The mean of the APs for all classes is the mAP.

Machine Learning theoretical concepts

By: Vikram Pal

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

$$AP_k = \text{the AP of class } k$$

$$n = \text{the number of classes}$$

YOLO3

Regarding the YOLO3 architecture:

- YOLO v3 uses a **variant of Darknet**, which originally has 53-layer network **trained on ImageNet**.
- For the task of detection, 53 more layers are stacked onto it, giving us **a 106 layer fully convolutional** underlying architecture for YOLO v3.
- In YOLO v3, the **detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network**.
- YOLO v3 uses **binary cross-entropy** for calculating the **classification loss** for each label while **object confidence and class predictions are predicted through logistic regression**.

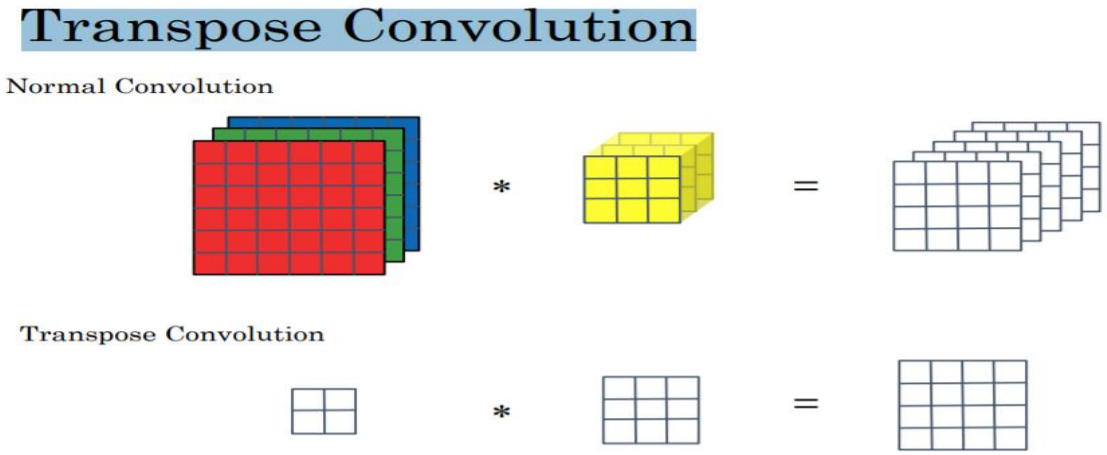
Hyper-parameters used

- **class_threshold** - Defines probability threshold for the predicted object.
- **Non-Max suppression Threshold** - It helps overcome the problem of detecting an object multiple times in an image. It does this by taking boxes with maximum probability and suppressing the close-by boxes with non-max probabilities (less than the predefined threshold).

CNN architecture of Darknet-53

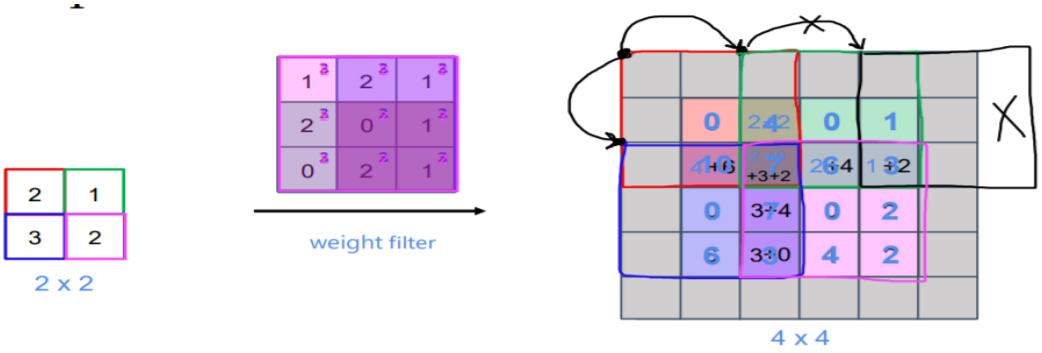
- **Darknet-53** is used as a feature extractor.
- **Darknet-53 mainly composed of 3 x 3 and 1 x 1 filters with skip connections like the residual network in ResNet.**

Transpose Convolution or Deconvolution:

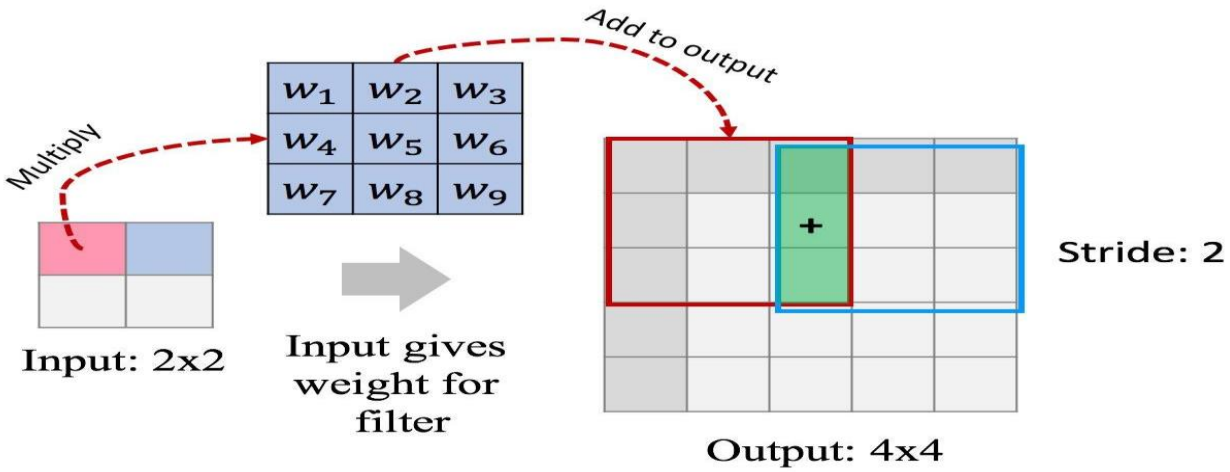


Machine Learning theoretical concepts

By: Vikram Pal



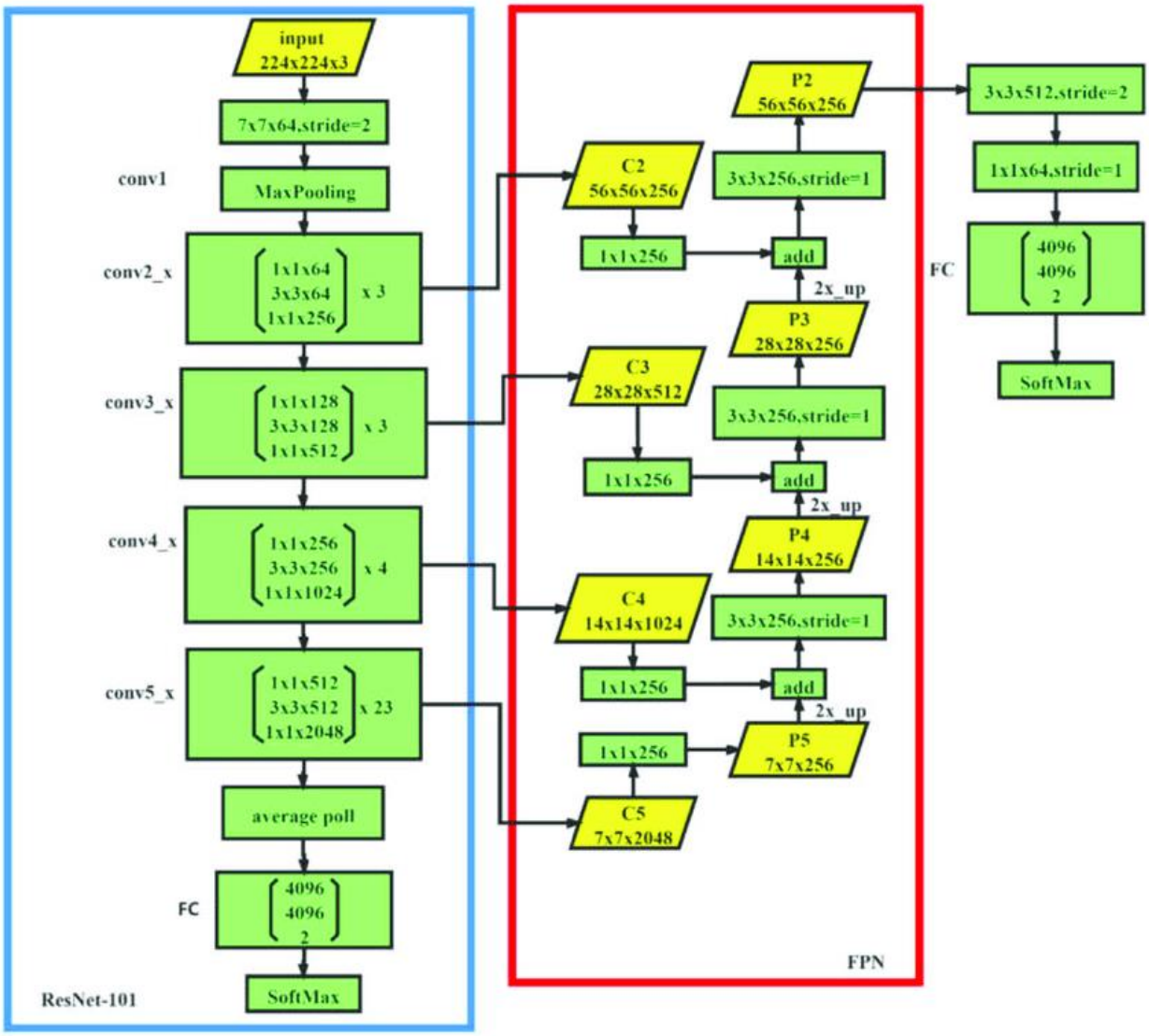
filter f x f = 3 x 3 padding p = 1 stride s = 2



Machine Learning theoretical concepts

By: Vikram Pal

Residual SE-FPN:



Residual SE-FPN (Residual Squeeze-and-Excitation Feature Pyramid Network) is a type of deep neural network architecture that is commonly used for object detection tasks in computer vision. It is a variant of the Feature Pyramid Network (FPN) architecture, which is itself a modification of the traditional Residual Network (ResNet) architecture.

The main idea behind the Residual SE-FPN architecture is to combine the strengths of ResNet, which is known for its ability to learn deep and complex representations, with the strengths of FPN, which is known for its ability to capture spatial hierarchies and handle large scale variations in object sizes. To achieve this, the Residual SE-FPN architecture uses a combination of residual blocks, which are the building blocks of ResNet, and feature pyramid layers, which are the building blocks of FPN.

The Residual SE-FPN architecture also includes a Squeeze-and-Excitation (SE) module, which is a type of attention mechanism that allows the network to focus on the most important features in an image. This

helps the network to better handle complex and cluttered images, and can improve the overall accuracy and efficiency of the object detection system.

Overall, the Residual SE-FPN architecture is a powerful and widely used tool for object detection tasks in computer vision, and has achieved state-of-the-art performance on a number of benchmarks and real-world datasets.

Segmentation:

Semantic Segmentation

- Same class → one segment.
- Each pixel is associated with one class.
  - All person(s) in an image are treated as one segment
- Popular models are Fully Convolutional Neural Networks, U-Net, DeepLab

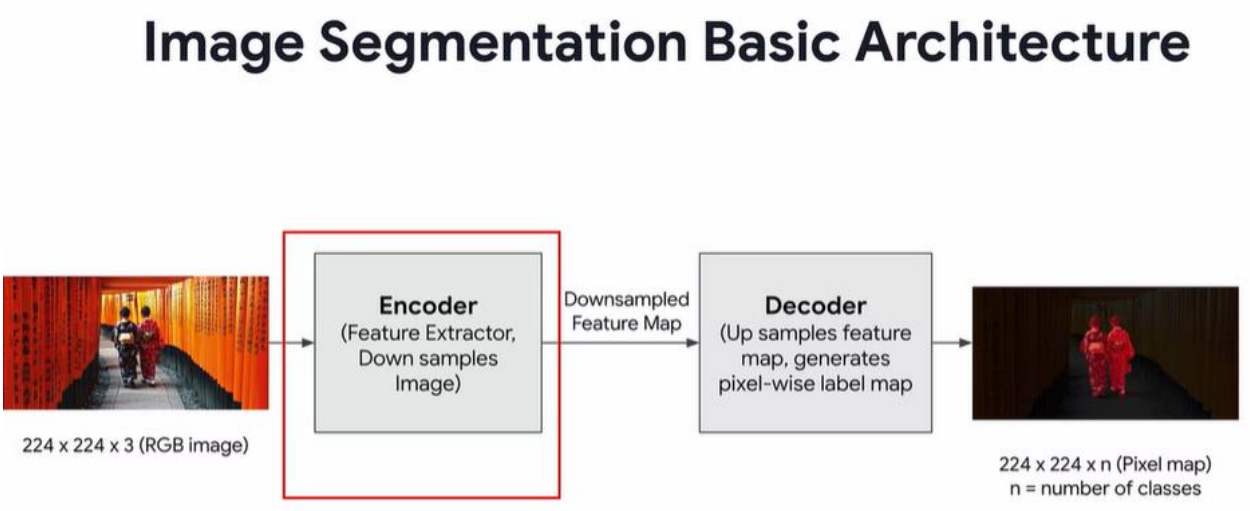
Instance Segmentation

- Multiple "instances" of same class are separate segments.

- Popular algorithms are Mask R-CNN.

Machine Learning theoretical concepts

By: Vikram Pal



- Encoder
  - CNN without fully connected layers
  - Aggregates low level features to high level features
- Decoder
  - Replaces fully connected layers in a CNN
  - Up samples image to original size to generate a pixel mask

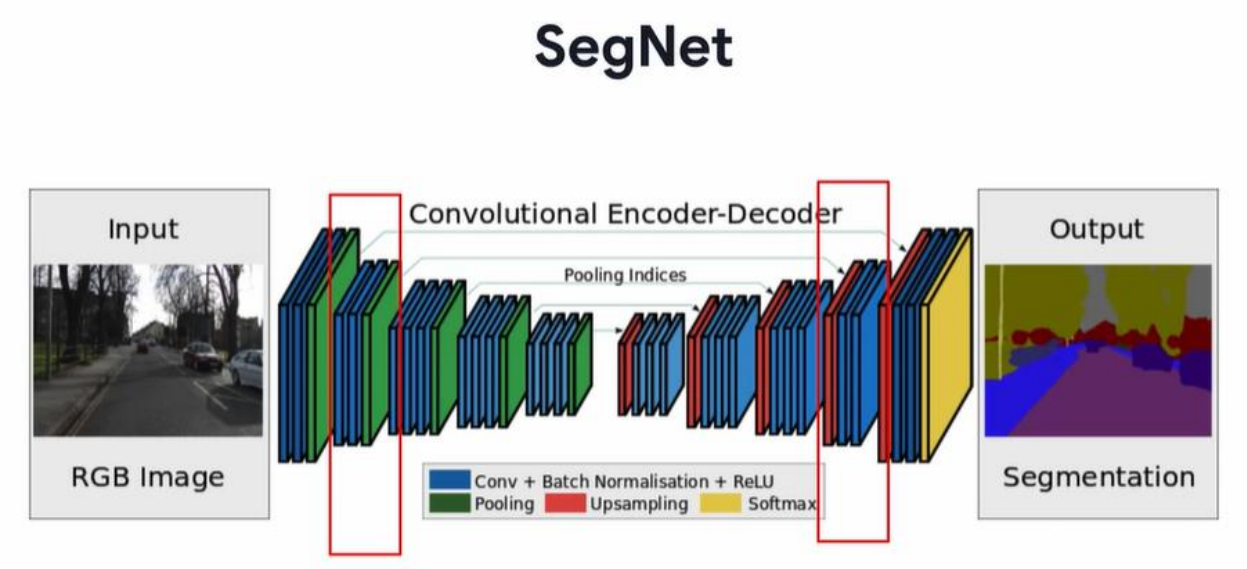
Segmentation Architecture:

- Fully Convolutional Neural Networks.
  - SegNet
  - UNet
  - PSPNet
  - Mask-RCNN

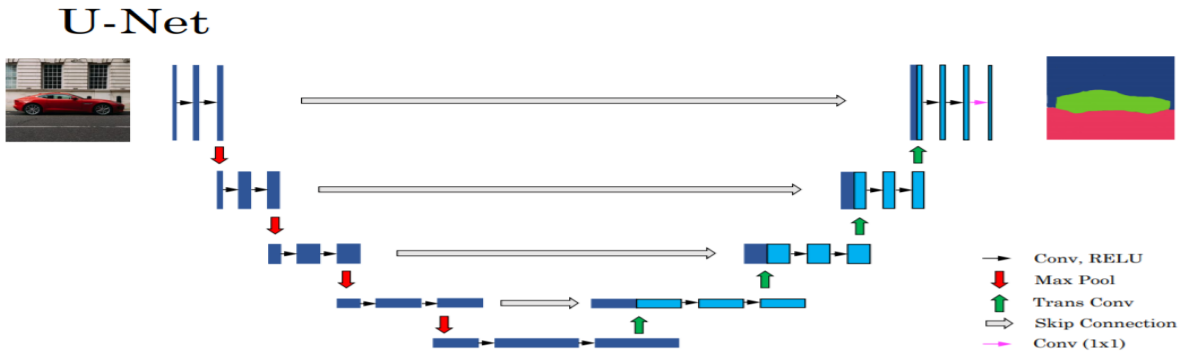
Machine Learning theoretical concepts

By: Vikram Pal

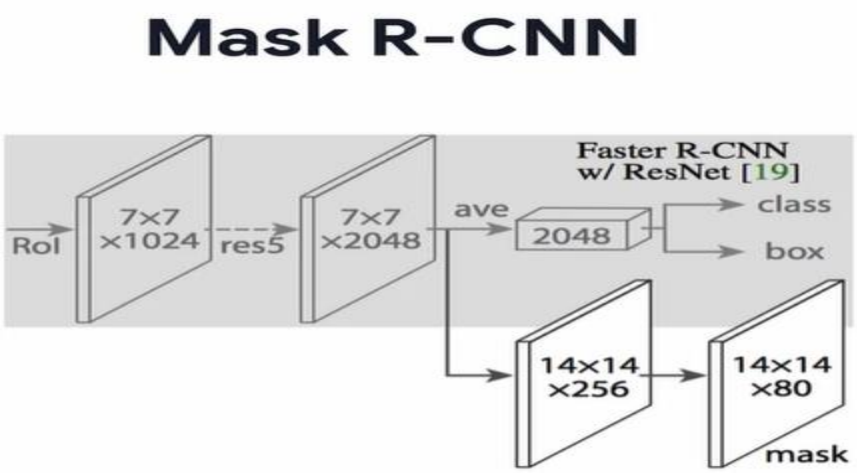
SegNet:



U-Net:



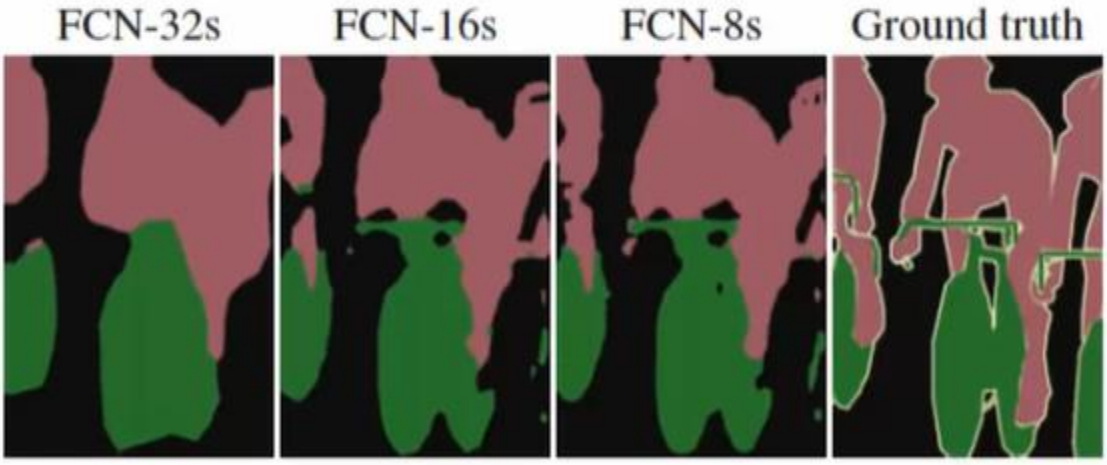
Mask R-CNN:



Encoder

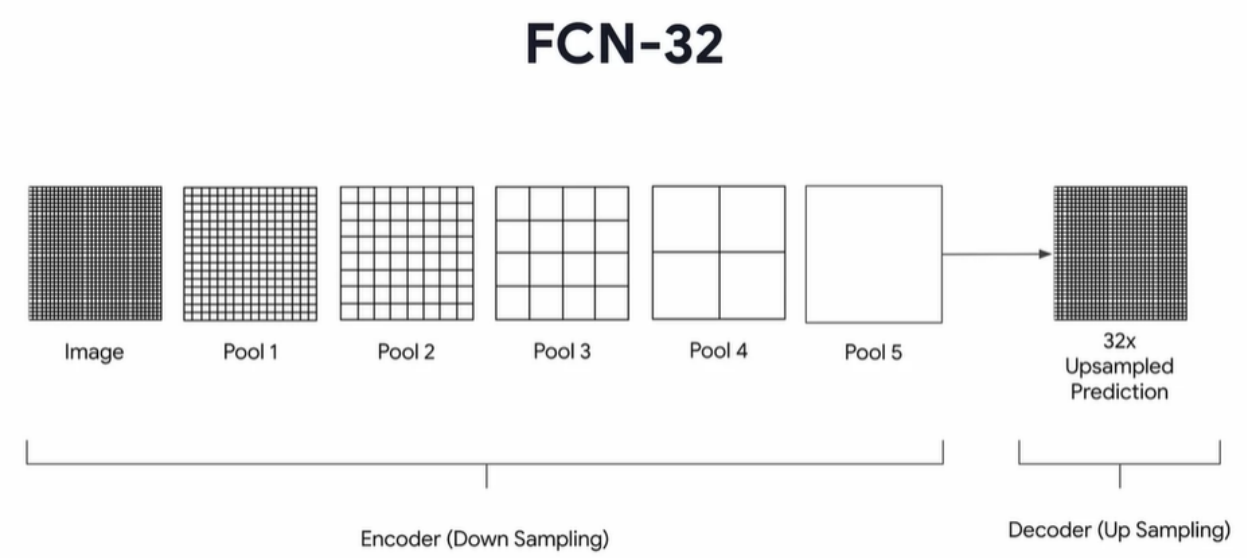
- Popular encoder architectures:
  - VGG-16
  - ResNet-50
  - MobileNet
- Reuse convolutional layers for feature extraction.
  - Do not reuse fully connected layers

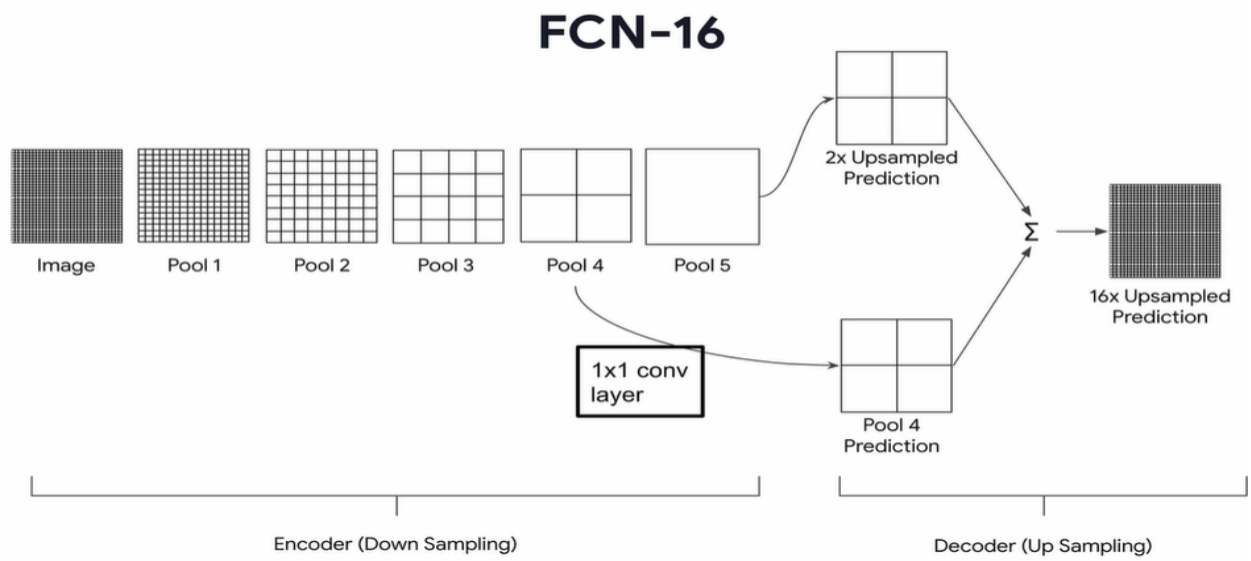
Decoder

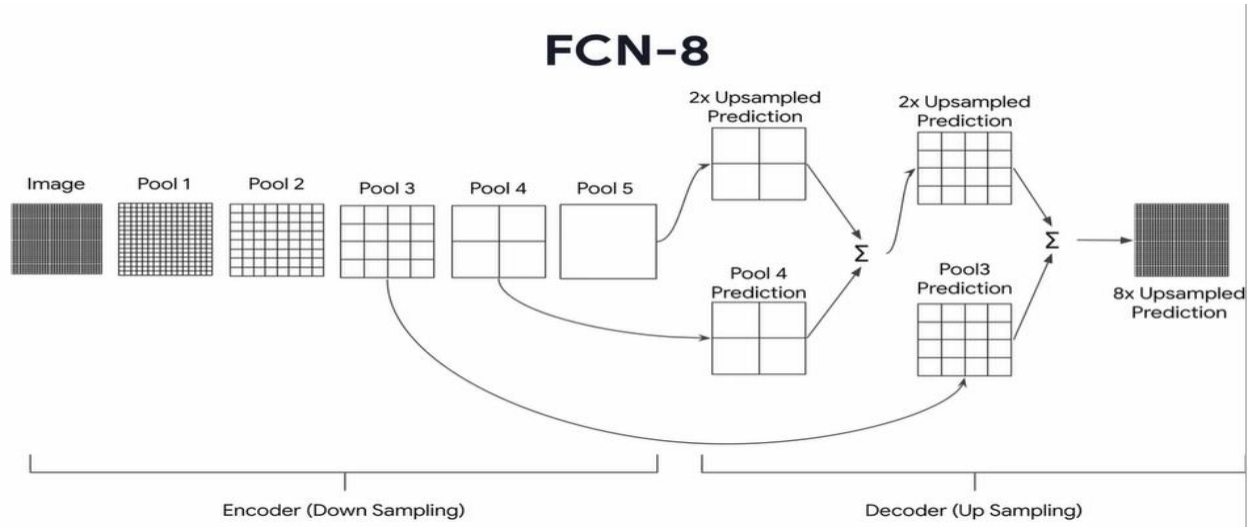


Machine Learning theoretical concepts

By: Vikram Pal







Machine Learning theoretical concepts

By: Vikram Pal

Natural Language Processing

NLP model evaluation Metrics:

Preplexity:

Dan Jurafsky

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

BLEU (Bi-Lingual Evaluation Understudy):

BLEU scores range from **0-1**, the higher the score, the more the translation correlates to a human translation.

The closer the BLEU score is to one, the better your model is. The closer to zero, the worse it is.

To get the BLEU score, the candidates and the references are usually based on an average of uni, bi, tri or even four-gram precision. To demonstrate, I'll use uni-grams as an example. Look at the following table:

Candidate	I	I	am	I	I
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

Le professeur est arrivé en retard à cause de la circulation. **(Source Original)**

The teacher arrived late because of the traffic. **(Reference Translation)**

The professor was delayed due to the congestion .	#1 Very low BLEU score
Congestion was responsible for the teacher being late	#2 Slightly higher but low BLEU
The teacher was late due to the traffic.	#3 Higher BLEU than #1 and #2
The professor arrived late because of circulation .	#4 Higher BLEU than #3
The teacher arrived late because of the traffic .	#5 ***Best BLEU Score***

Many accurate and correct translations can score lower simply because they use different words

SDL

To calculate the BLEU score you can do the following.

2 BLEU

Papineni et al. (2002) originally define BLEU n-gram precision p_n by summing the n-gram matches for every hypothesis sentence S in the test corpus C:

$$p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{matched}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)} \quad (1)$$

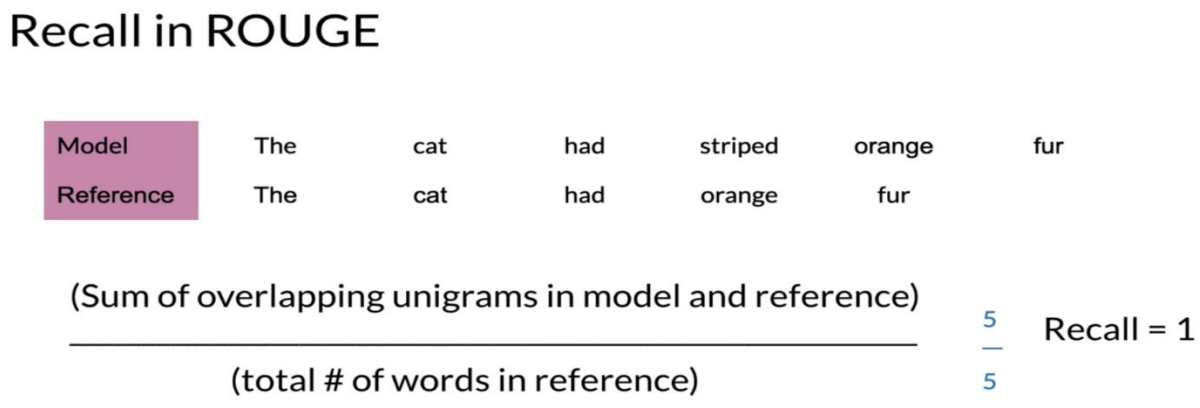
You would sum over the unique n-gram counts in the candidate and divide by the total number of words in the candidate. The same concept could apply to unigrams, bigrams, etc. One issue with the BLEU score is that it does not take into account semantics, so it does not take into account the order of the n-grams in the sentence.

```
>>> from bleu import multi_list_bleu
>>> multi_list_bleu([ref, ref1], [hyp, hyp1])
[34.99, 57.91]
```

Rogue:

ROUGE score, returned as a scalar value in the range **[0,1] or NaN** . A ROUGE score close to zero indicates poor similarity between candidate and references . A ROUGE score close to one indicates strong similarity between candidate and references

Another similar method for evaluation is the ROUGE score which calculates precision and recall for machine texts by counting the **n-gram overlap between the machine texts and a reference text**. Here is an example that calculates recall:



Rouge also allows you to compute precision as follows:

Machine Learning theoretical concepts

By: Vikram Pal

Precision in ROUGE

Model	The	cat	had	striped	orange	fur
Reference	The	cat	had	orange	fur	

$$\frac{\text{(Sum of overlapping unigrams in model and reference)}}{\text{(total \# of words in model)}} \qquad \frac{5}{6} \quad \text{Precision} = 0.83$$

Using Rouge to Evaluate abstractive Summary

```
from rouge import Rouge
r = Rouge()
r.get_scores(abstractve_summarization, reference_text)
```

Trade off b/w Bleu and Rogue score:

If you have **many words/ngrams from the system results** appearing in the human references you will **have high Bleu**, and if you have **many words/ngrams from the human references** appearing in the system results you **will have high Rouge.**

There's something called **brevity penalty**, which is quite important and has already been added to standard **Bleu** implementations. **It penalizes system results(candidate sentence) which are shorter than the general length of a reference**. This complements the n-gram metric behavior which in effect penalizes longer than reference results, since the denominator grows the longer the system result is.

We can also implement something similar for Rouge, but this time **penalizing system results which are longer than the general reference length,** which would otherwise enable them to obtain artificially higher Rouge scores (since the longer the result, the higher the chance you would hit some word appearing in the references). In Rouge we divide by the length of the human references, so we would need an additional penalty for longer system results which could artificially raise their Rouge score.

Finally, you could use the **F1 measure** to make the metrics work together: F1 = 2 * (Bleu * Rouge) / (Bleu + Rouge)

Bleu and Rogue usage: These two scores can be used for machine translation, Spellchecker, and Summarization (Abstractive and Extractive).

Machine Learning theoretical concepts

By: Vikram Pal

GLUE Benchmark

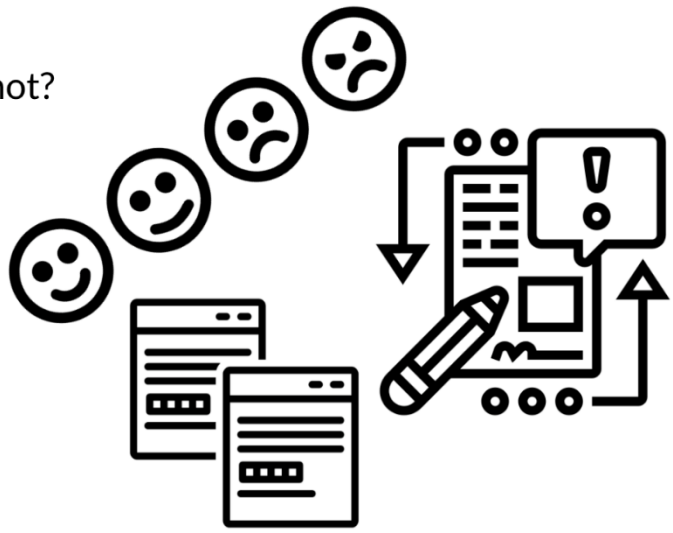
General Language Understanding Evaluation (GLUE) is containing:

- A collection used to train, evaluate, analyze natural language understanding systems
- Datasets with different genres, and of different sizes and difficulties
- Leaderboard

question (due to lack of content or misunderstanding).

Tasks Evaluated on

- Sentence grammatical or not?
- Sentiment
- Paraphrase
- Similarity
- Questions duplicates
- Answerable
- Contradiction
- Entailment
- Winograd (co-ref)



Currently T5 is state of the art according to this GLUE benchmark and you will be implementing it for homework this week! This GLUE benchmark is used for research purposes, it is model agnostic, and relies on models that make use of transfer learning.

Chatbot Evaluation metrics:

1. **Usage rate per login:** volume of active user sessions on the chatbot. To balance out with the average number of sessions on your website.

2**. Bounce rate:** volume of sessions where the chatbot was opened but not used

3. **Satisfaction rate**: average grade given when evaluating the chatbot's answers (to balance out with the evaluation rate).

4. **Average chat time:** allows you to evaluate your users' interest for your chatbot.

5. **Average number of interactions:** used to evaluate the Customer Effort Score on the chatbot and must be correlated to the satisfaction rate. If the latter is very low, the bot may be engaging the users in too many branches and steps to meet their needs. In this case, a resolution can be to correct the decision trees or knowledge base architecture.

6.**Goal completion rate:** in case your bot contains targeted actions like CTAs, a form or some cross-selling, that is the rate of users who have reached that specific action through the chatbot.

Word Embedding and Text Vectorization:

Different Types of Word Embeddings

- Pre Word-Embedding era – Frequency or Statistical based word Embedding approaches
- Recent Word-Embedding era – Prediction based word Embedding approaches

Pre word embedding era Techniques

- One-hot Encoding (OHE)
- Count Vectorizer
- Bag-of-Words (BOW)
- N-grams
- Term Frequency-Inverse Document Frequency (TF-IDF)

One-hot Encoding:

Let's consider the following sentence:

```
Sentence: I am teaching NLP in Python
```

A word in this sentence may be "NLP", "Python", "teaching", etc.

Since a dictionary is defined as the list of all unique words present in the sentence. So, a dictionary may look like –

```
Dictionary: ['I', 'am', 'teaching',' NLP',' in', 'Python']
```

Therefore, the vector representation in this format according to the above dictionary is

```
Vector for NLP: [0,0,0,1,0,0]
Vector for Python:  [0,0,0,0,0,1]
```

This is just a very simple method to represent a word in vector form.

Disadvantages of One-hot Encoding

1. One of the disadvantages of One-hot encoding is that the **Size of the vector is equal to the count of unique words in the vocabulary**.

2. One-hot encoding does not capture the **relationships between different words.** Therefore, it does not convey information about the context.

Count Vectorizer:

Let's consider the following example:

```
Document-1: He is a smart boy. She is also smart.
Document-2: Chirag is a smart person.
```

The dictionary created contains the list of unique tokens(words) present in the corpus

```
Unique Words: ['He', 'She', 'smart', 'boy', 'Chirag', 'person']
```

Here, D=2, N=6

So, the count matrix M of size 2 X 6 will be represented as –

He She smart boy Chirag person

D1	1	2	1	0	0
D2	0	1	0	1	1

Now, a column can also be understood as a word vector for the corresponding word in the matrix M.

For Example, for the above matrix formed, let's see the word vectors generated.

```
Vector for 'smart' is [2,1],
Vector for 'Chirag' is [0, 1], and so on.
```

What do rows and columns indicate in the above Count matrix?

- The rows indicate the documents in the corpus and
- The columns indicate the tokens in the dictionary.

Bag of Words:

Let's take a good understanding with the help of the following example:

```
This burger is very tasty and affordable
This burger is not tasty and is affordable
This burger is very very delicious
```

These 3 sentences are example sentences and combined it forms a corpus and our first step is to perform tokenization. Before tokenization, we will convert all sentences to lowercase letters or uppercase letters. Here, for normalization, we will convert all the words in the sentences to lowercase.

Now, the Output of sentences after converting to lowercase is:

```
this burger is very tasty and affordable.
this burger is not tasty and is affordable.
```

```
this burger is very very delicious.
```

Now we will perform tokenization.

After dividing the sentences into words and generate a list with all unique words in alphabetical order, we will get the following output after the tokenization step:

```
Unique words: [“and”, “affordable.”, “delicious.”,  “is”, “not”, “burger”, “tasty”, “this”, “very”]
```

Now, what is our next step?

Creating vectors for each sentence with the frequency of words. This is called a sparse matrix. Below is the sparse matrix of example sentences.

	and	affordable	delicious	is	not	pasta	tasty	this	very
this pasta is very tasty and affordable.	1	1	0	1	0	1	1	1	1
this pasta is not tasty and is affordable	1	1	0	2	1	1	1	1	0
this pasta is very very delicious.	0	0	1	1	0	1	0	1	2

N-grams Vectorization:

1. Similar to the count vectorization technique, in the N-Gram method, a document term matrix is generated, and each cell represents the count.

2. The columns represent all columns of adjacent words of length n.

3. Count vectorization is a special case of N-Gram where n=1.

4. N-grams consider the sequence of n words in the text; where n is (1,2,3..) like 1-gram, 2-gram. for token pair. Unlike BOW, it maintains word order.

For Example,

```
“I am studying NLP” has four words and n=4.

if n=2, i.e bigram, then the columns would be – [“I am”, “am reading”, ‘studying NLP”]
```

Machine Learning theoretical concepts

By: Vikram Pal

```
if n=3, i.e trigram, then the columns would be – [“I am studying”, ”am studying NLP”]

if n=4,i.e four-gram, then the column would be -[‘“I am studying NLP”]
```

Disadvantages of N-Grams

1. It has too many features.

2. Due to too many features, the feature set becomes too sparse and is computationally expensive.

3. Choose the optimal value of N is not that easy task.

TF-IDF:

$$TF(\boldsymbol{term}) = \frac{Number\ of\ times\ \boldsymbol{term}\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}$$

$$IDF(\boldsymbol{term}) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ \boldsymbol{term}\ in\ it}\right)$$

Let’s say the word ‘is’ is present in all the documents is a corpus of 1000 documents. The idf for that would be:

The idf(‘is’) is equal to log (1000/1000) = log 1 = 0

Thus, common words would have lesser importance.

$$TFIDF(\boldsymbol{term}) = TF(\boldsymbol{term}) * \mathrm{IDF}(\mathbf{term})$$

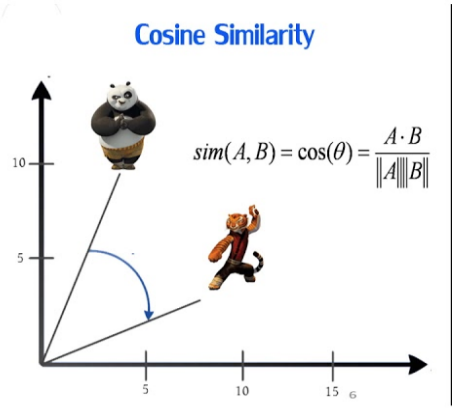
Word Embedding:

Machine Learning theoretical concepts

By: Vikram Pal

Word Embeddings

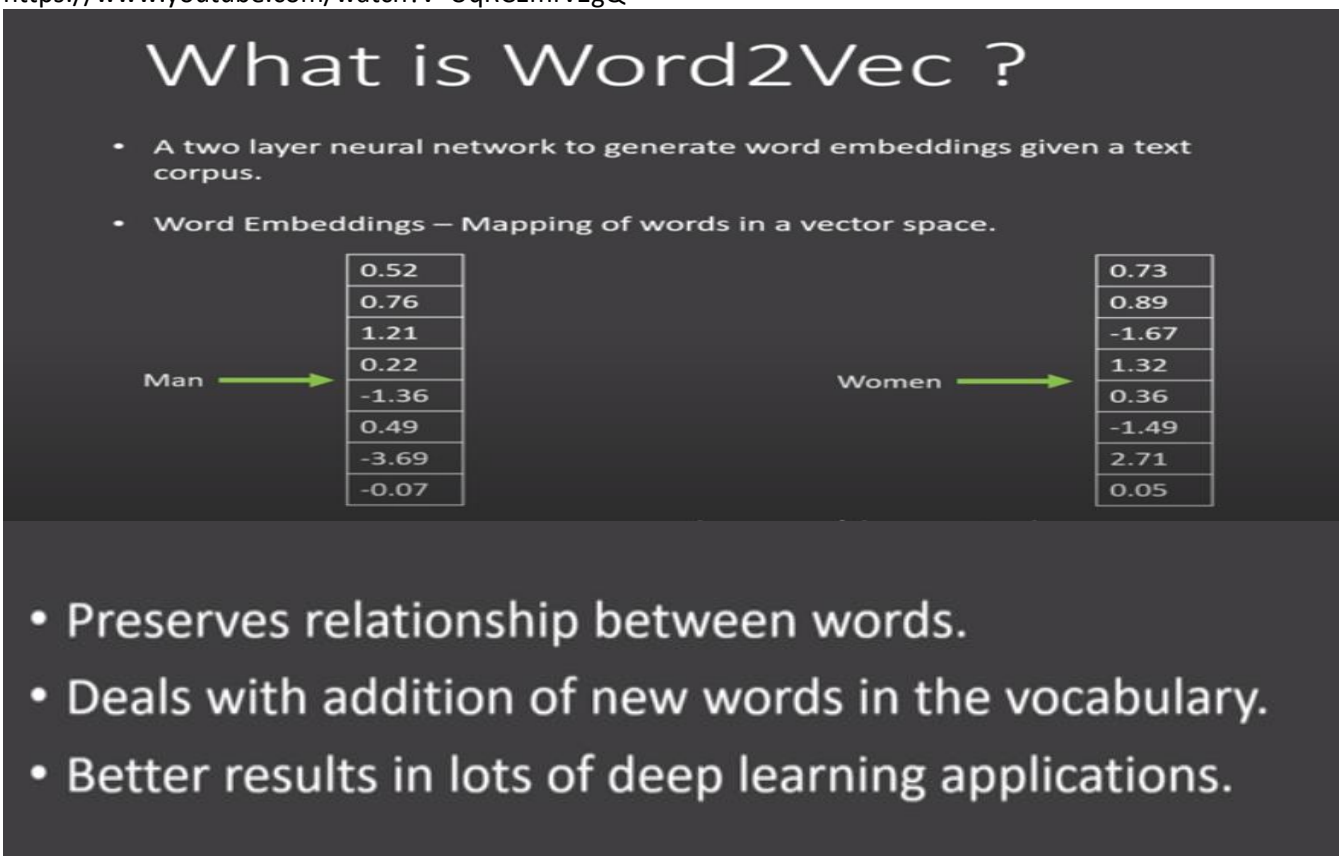
- Word Embedding's are the texts converted into numbers and there may be different numerical representations of the same text.
- Machine Learning algorithms and Deep Learning Architectures are incapable of processing *strings* or *plain text* in their raw form.
- They require numbers as inputs to perform any Task on Texts!
- Our objective is to have words with similar context occupy close spatial positions.
- Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.



Word2Vec:

http://www.claudiobellei.com/2018/01/06/backprop-word2vec/#skipgram

https://www.youtube.com/watch?v=UqRCEmrv1gQ



Machine Learning theoretical concepts

By: Vikram Pal

Working of word2Vec

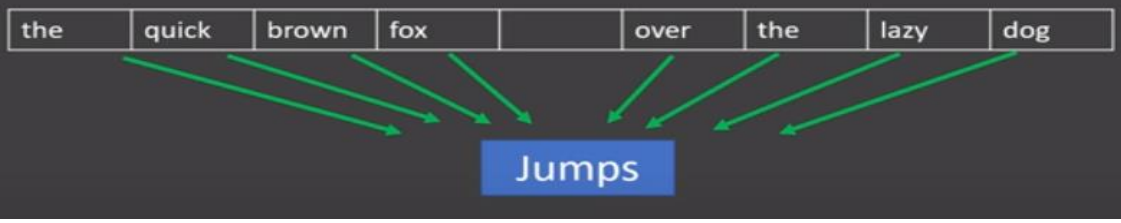
- The word2vec objective function causes the words that occur in similar contexts to have similar embeddings.

Example: The **kid** said he would grow up to be superman.
The **child** said he would grow up to be superman.

The words kid and child will have similar word vectors due to a similar context.

CBOW

- Predict the target word from the context.

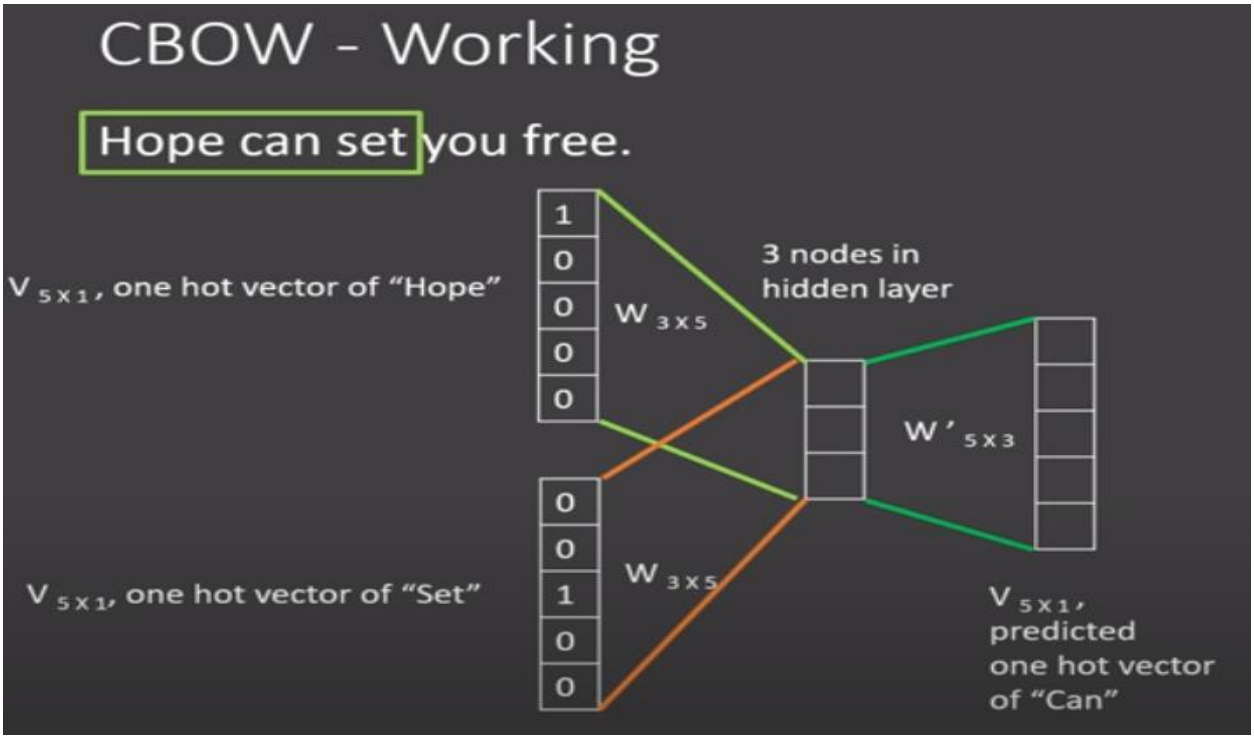


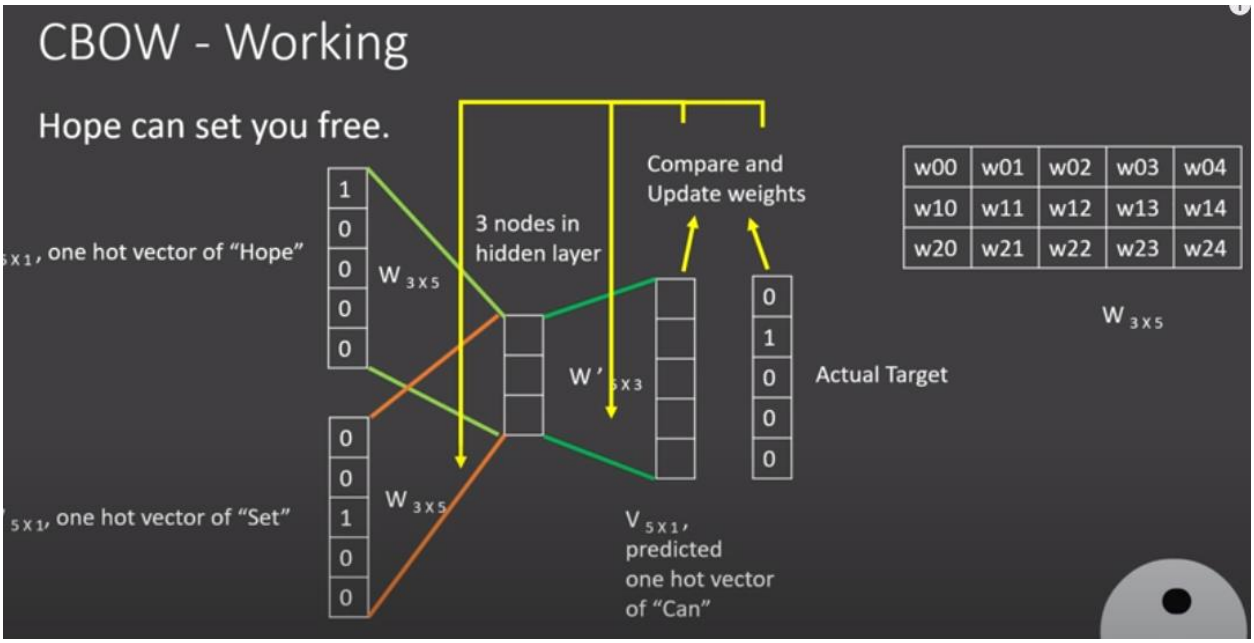
Skip Gram

- Predict the context words from target.

Machine Learning theoretical concepts

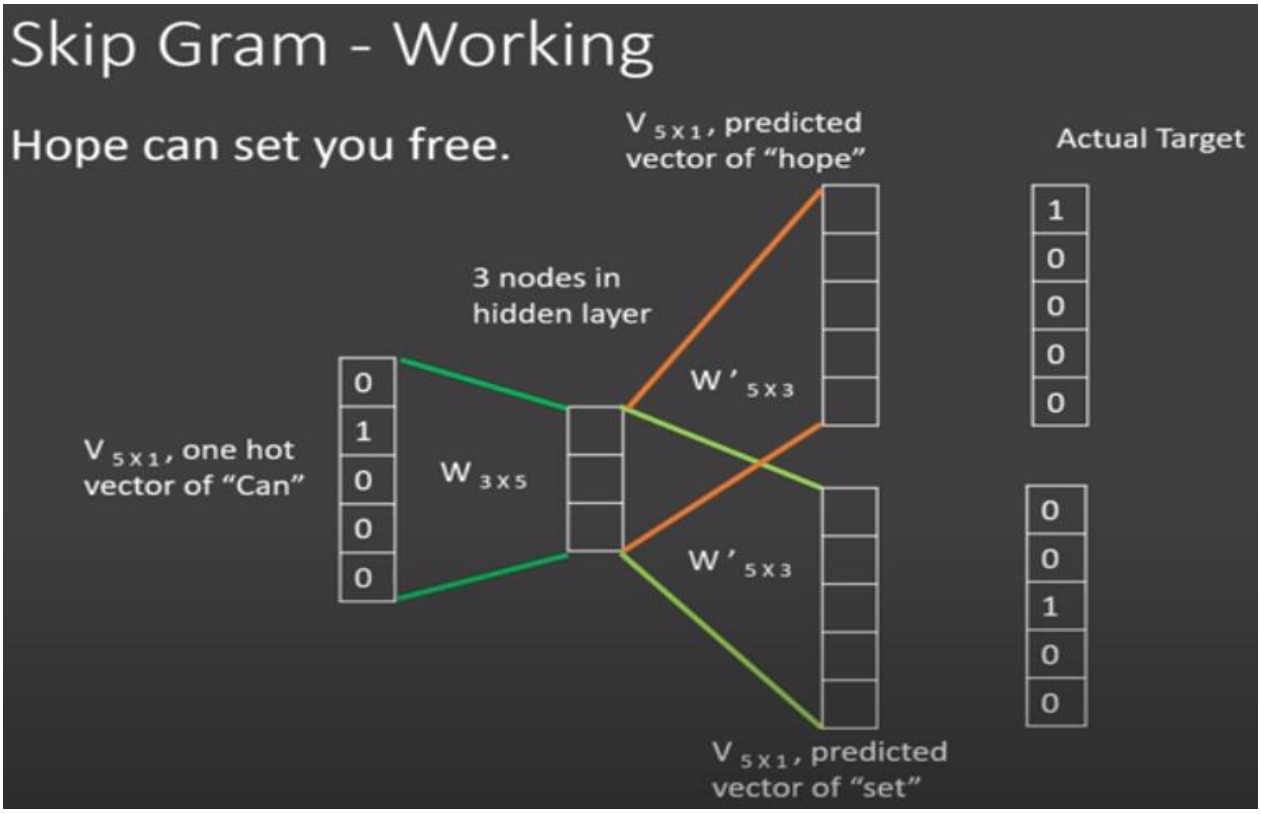
By: Vikram Pal

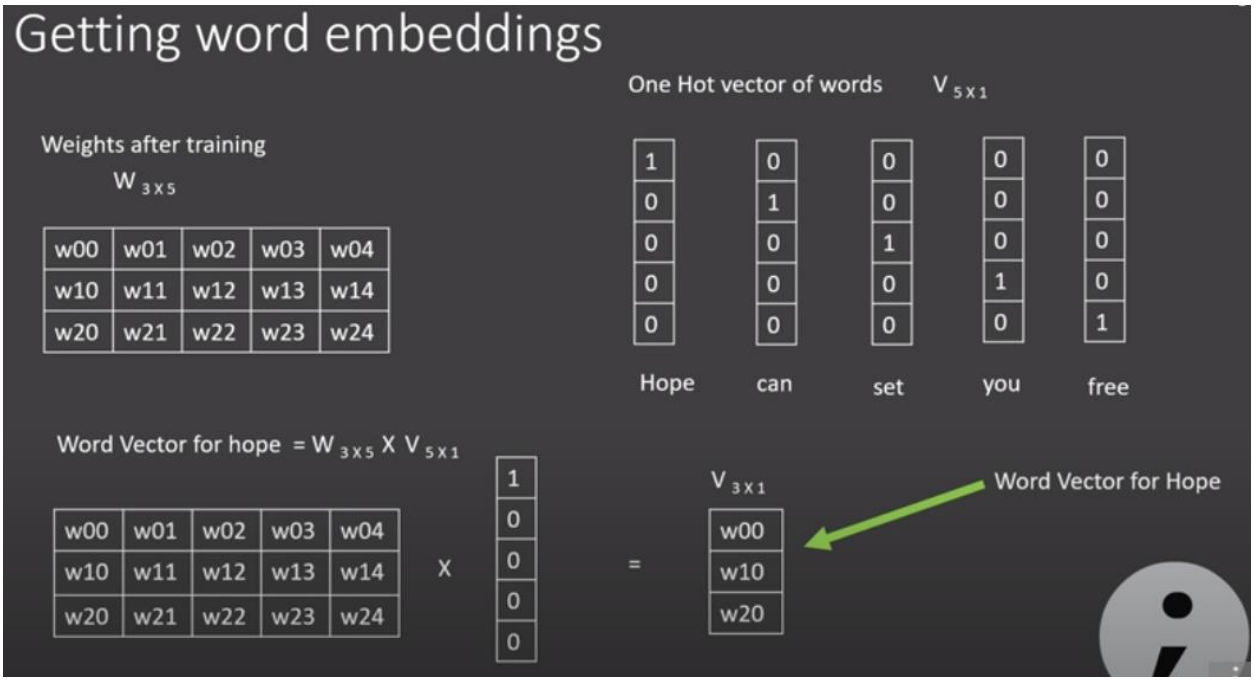




Machine Learning theoretical concepts

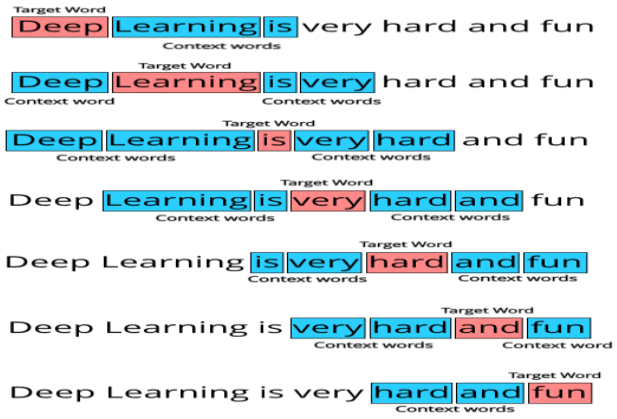
By: Vikram Pal





Machine Learning theoretical concepts

By: Vikram Pal



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

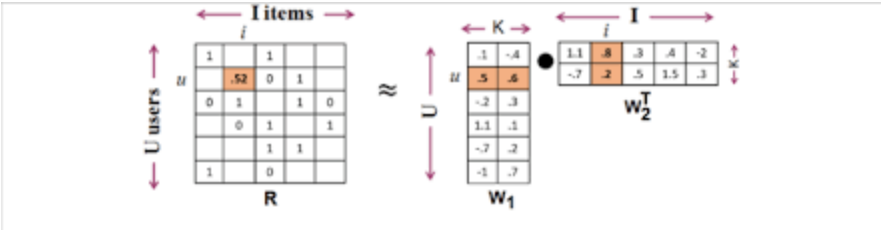
Glove

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

Topic modeling:

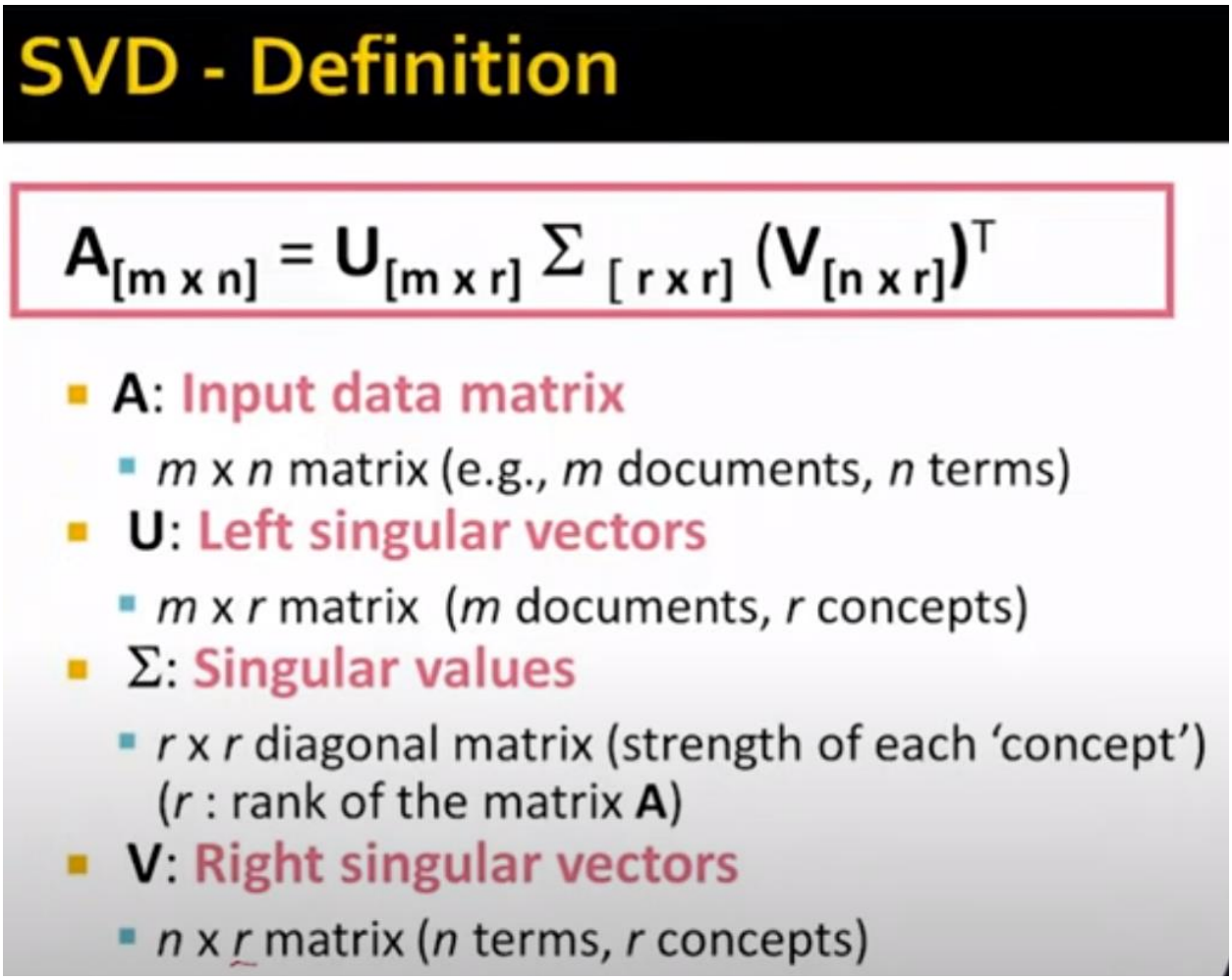
Note*: Word→document→ Corpus.

Topic modeling: 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.

SVD:

Machine Learning theoretical concepts

By: Vikram Pal



LSA: Latent Semantic Analysis: raw count in (Documents, terms) replace with tf-idf score ➔ (documents, topic) * (topic*term)

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 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.**

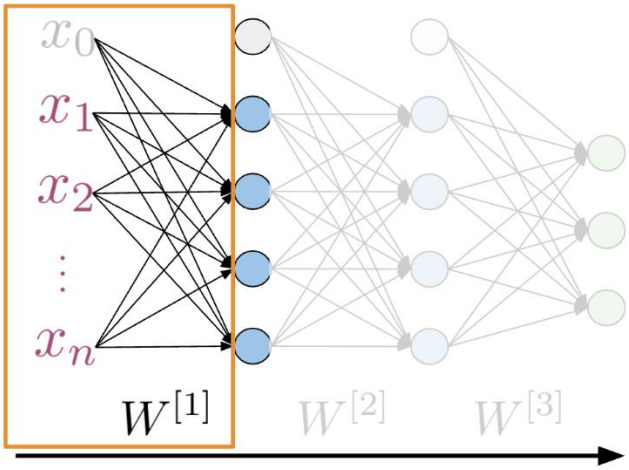
Machine Learning theoretical concepts

By: Vikram Pal

Different Layers & Training:

Dense and ReLU layer:

The Dense layer is the computation of the inner product between a set of trainable weights (weight matrix) and an input vector. The visualization of the dense layer could be seen in the image below.

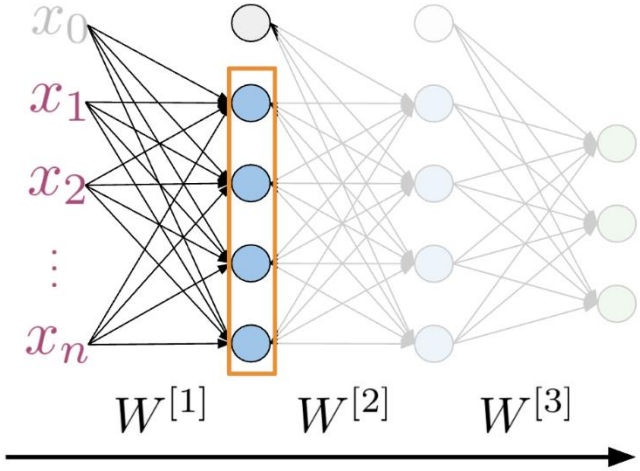


Fully connected layer

$$z^{[i]} = W^{[i]} a^{[i-1]}$$

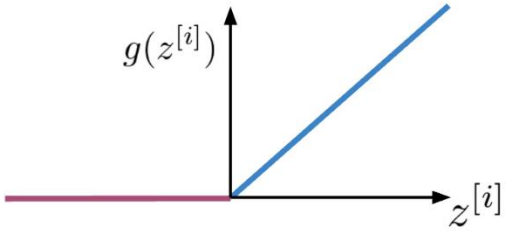
Trainable parameters

The orange box shows the dense layer. An activation layer is the set of blue nodes. Concretely one of the most commonly used activation layers is the rectified linear unit (ReLU).



ReLU = Rectified linear unit

$$a^{[i]} = g(z^{[i]})$$
$$g(z^{[i]}) = \max(0, z^{[i]})$$



ReLU(x) is defined as *max(0,x)* for any input x.

Other Layers:

Other layers could include embedding layers and mean layers. For example, you can learn word embeddings for each word in your vocabulary as follows:

Machine Learning theoretical concepts

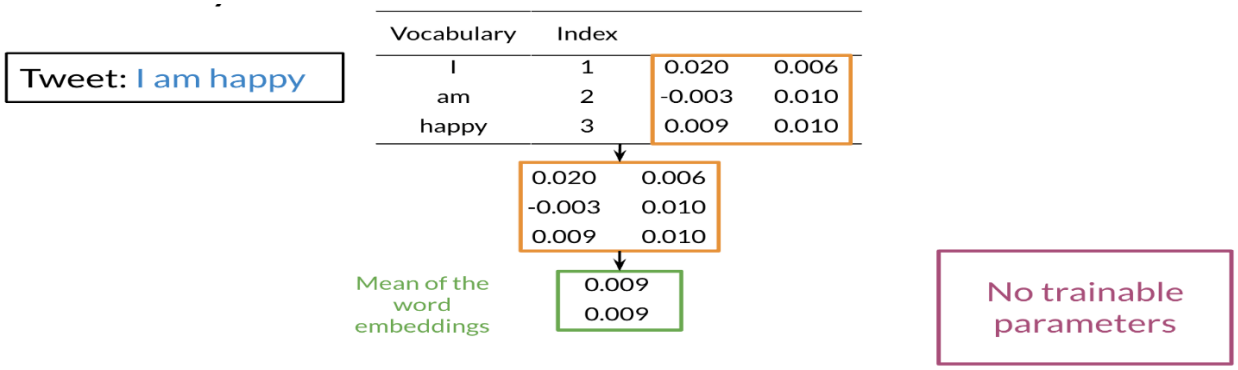
By: Vikram Pal

Vocabulary	Index		
I	1	0.020	0.006
am	2	-0.003	0.010
happy	3	0.009	0.010
because	4	-0.011	-0.018
learning	5	-0.040	-0.047
NLP	6	-0.009	0.050
sad	7	-0.044	0.001
not	8	0.011	-0.022

Trainable weights

Vocabulary x Embedding

The mean layer allows you to take the average of the embeddings. You can visualize it as follows:



This layer does not have any trainable parameters.

Training:

In Trax, the function grad allows you to compute the gradient. You can use it as follows:

$$f(x) = 3x^2 + x$$

$$\frac{\delta f(x)}{\delta x} = 6x + 1$$

Gradient

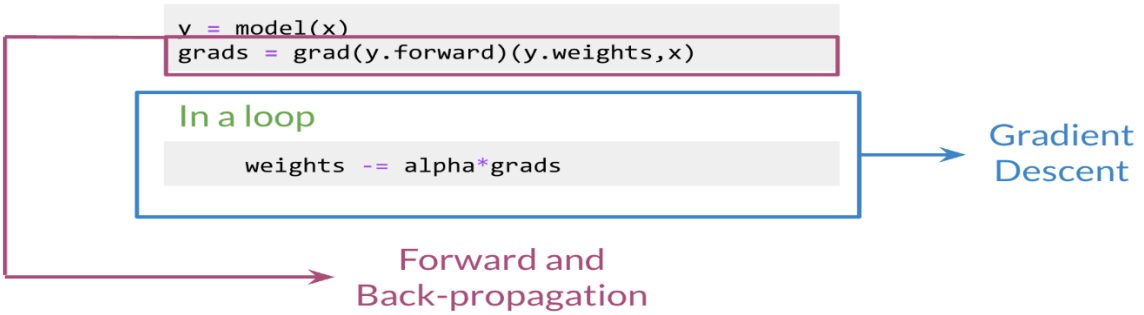
```
def f(x):
    return 3*x**2 + x

grad_f = trax.math.grad(f)
```

Returns a function

Now if you were to evaluate *grad_f* at a certain value, namely *z*, it would be the same as computing *6z+1*. Now to do the training, it becomes very simple:

Machine Learning theoretical concepts

By: Vikram Pal



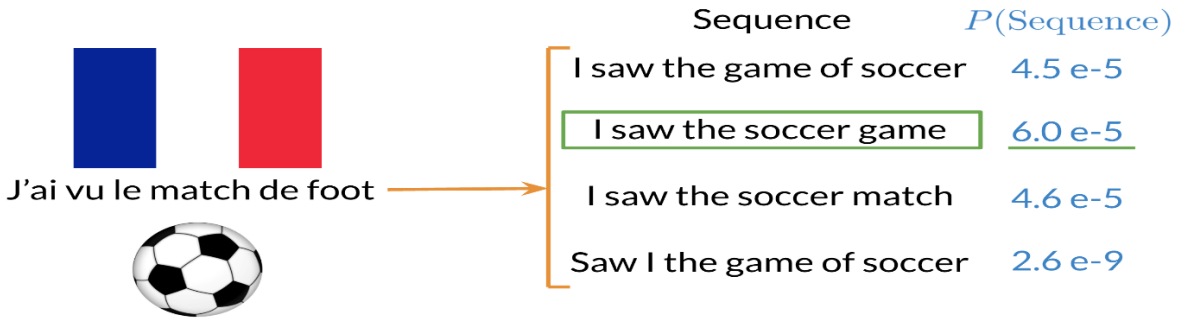
You simply compute the gradients by feeding in y.forward (the latest value of y), the weights, and the input x, and then it does the backpropagation for you in a single line. You can then have the loop that allows you to update the weights (i.e. gradient descent!).

TimeDistributed Layer:

We are dealing with **Many to Many RNN Architecture, where we expect output from every input sequence.** Here is an example, in the sequence (a1 →b1, a2 →b2…an →bn), a, and b are inputs and outputs of every sequence. The TimeDistributeDense layers allow Dense(fully-connected) operation across every output over every time-step. No using this layer will result in one final output.

Traditional Language models:

Traditional language models make use of probabilities to help identify which sentence is most likely to take place.



In the example above, the second sentence is the one that is most likely to take place as it has the highest probability of happening. To compute the probabilities, you can do the following:

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

$$P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2)$$

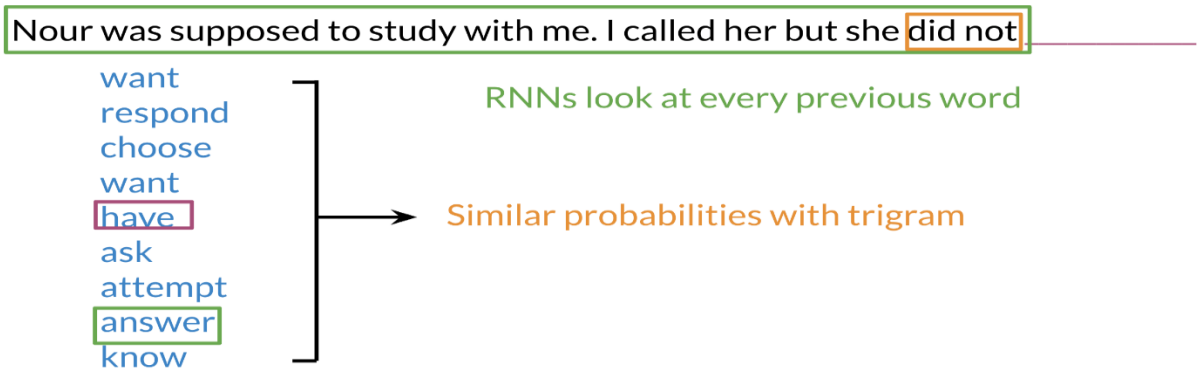
Machine Learning theoretical concepts

By: Vikram Pal

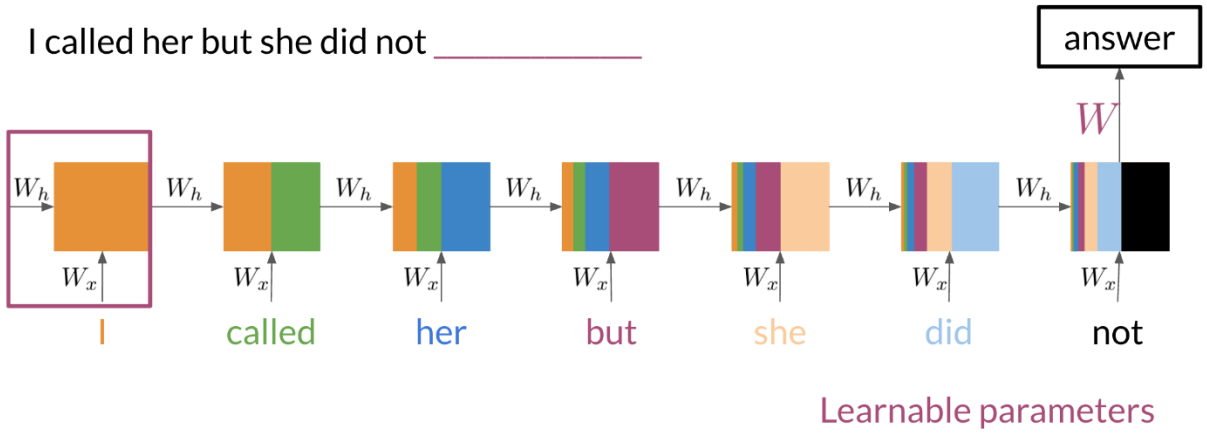
Large N-grams capture dependencies between distant words and need a lot of space and RAM. Hence, we resort to using different types of alternatives.

Recurrent Neural Network (RNN):

Previously, we tried using traditional language models, but it turns out they took a lot of space and RAM. For example, in the sentence below:



An N-gram *(trigram)* would only look at "did not" and would try to complete the sentence from there. As a result, the model will not be able to see the beginning of the sentence "I called her but she". Probably the most likely word is *have* after "did not". RNNs help us solve this problem by being able to track dependencies that are much further apart from each other. As the RNN makes its way through a text corpus, it picks up some information as follows:



Note that as **you feed in more information into the model, the previous word's retention gets weaker, but it is still there**. Look at the orange rectangle above and see how it becomes smaller as you make your way through the text. This shows that your model is capable of capturing dependencies and remembers a previous word although it is at the beginning of a sentence or paragraph. Another advantage of RNNs is that a lot of the computation shares parameters.

Machine Learning theoretical concepts

By: Vikram Pal

It is a generalization of **feedforward neural network that has an internal memory**. RNN is recurrent in nature as it performs the **same function for every input of data while the output of the current input depends on the past one computation.**

In other neural networks, **all the inputs are independent of each other. But in RNN, all the inputs are related to each other.**

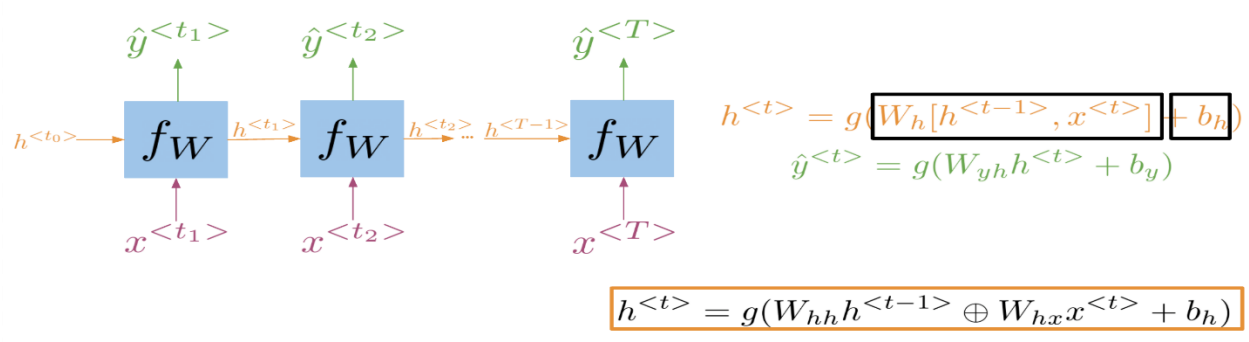
Application of RNNs

RNNs could be used in a variety of tasks ranging from machine translation to caption generation. There are many ways to implement an RNN model:

- One to One: **given some scores of a championship, you can predict the winner.**
- One to Many: **given an image, you can predict what the caption is going to be.**
- Many to One: **given a tweet, you can predict the sentiment of that tweet.**
- Many to Many: **given an English sentence, you can translate it to its German equivalent.**

Math in Simple RNNs

It is best to explain the math behind a simple RNN with a diagram:



Machine Learning theoretical concepts

By: Vikram Pal

Note that:

$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

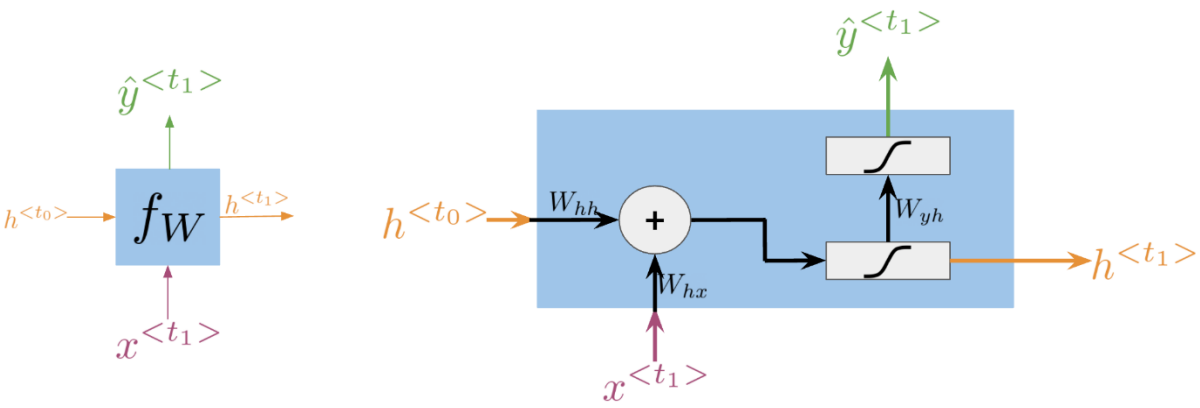
Is the same as multiplying W_{hh} by h and W_{hx} by x. In other words, you can concatenate it as follows:

$$h^{<t>} = g\left(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h\right)$$

For the prediction at each time step, you can use the following:

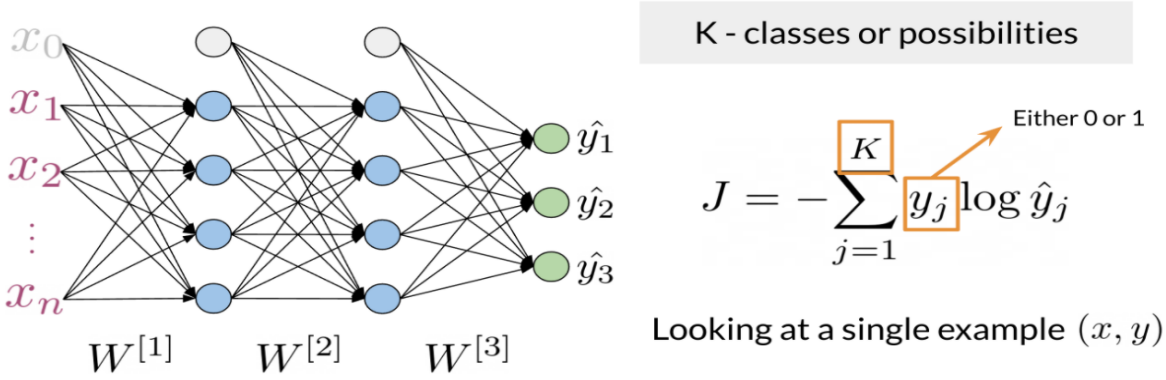
$$\hat{y}^{<t>} = g\left(W_{yh}h^{<t>} + b_y\right)$$

Note that you end up training $W_{hh}, W_{hx}, W_{yh}, b_h, b_y$. Here is a visualization of the model.



Cost Function for RNNs

The cost function used in an RNN is the cross entropy loss. If you were to visualize it



you are basically summing over the all the classes and then multiplying y_j times $\log \hat{y}_j$. If you were to compute the loss over several time steps, use the following formula:

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{K} y_j^{<t>} \log \hat{y}_j^{<t>}$$

Note that we are simply summing over all the time steps and dividing by *T*, to get the average cost in each time step. Hence, we are just taking an average through time.

Machine Learning theoretical concepts

By: Vikram Pal

Gradient issues in RNN:

While training an RNN algorithm, sometimes gradient can become too small or too large. So, the training of an RNN algorithm becomes very difficult in this situation. Due to this, following issues occur:

1. Poor Performance
2. Low Accuracy
3. Long Training Period

Exploding Gradient: When we **assign high importance to the weights, exploding gradient issue occurs**. In this case, values of a gradient become too large, and slope tends to grow exponentially. This can be solved using following methods:

1. Penalty

2. Truncated Backpropagation

3. Gradient Clipping: With gradient clipping**, pre-determined gradient threshold** be introduced, and **then gradients norms that exceed this threshold are scaled down** to match the norm. This prevents any **gradient to have norm greater than the threshold** and thus the gradients are clipped. There is **an introduced bias in the resulting values from the gradient, but gradient clipping can keep things stable.**

Vanishing Gradient: This issue occurs when the values of a **gradient are too small, and the model stops learning** or takes way too long because of that. This can be solved using following methods:

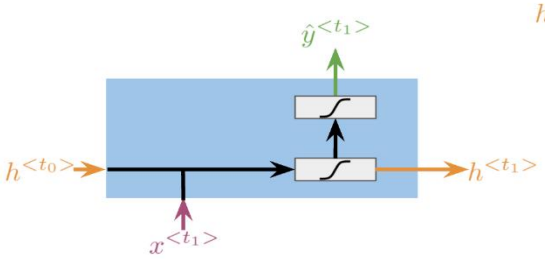
1. Weight Initialization

2. Choosing the right Activation Function

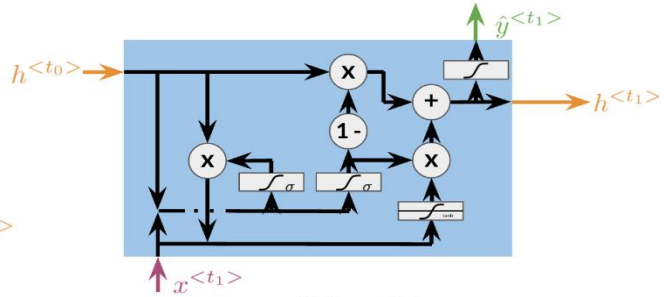
3. LSTM (Long Short-Term Memory) Best way to solve the vanishing gradient issue is the use of LSTM (Long Short-Term Memory).

Gated Recurrent Units

Gated recurrent units are very similar to vanilla RNNs, except that they have a **"relevance" and "update" gate** that allow the model to update and get relevant information. I personally find it easier to understand by looking at the formulas:



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$



$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$
$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

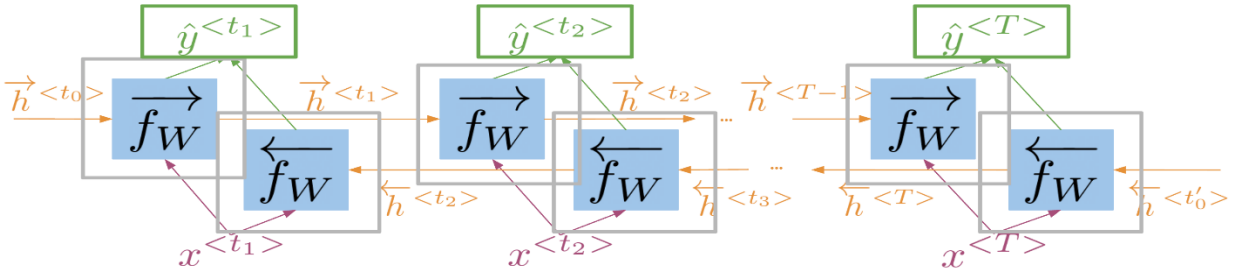
To the left, you have the diagram and equations for a simple RNN. To the right, we explain the GRU. Note that we add 3 layers before computing *h* and *y*.

$$\Gamma_u = \sigma\left(W_u\left[h^{<t_0>}, x^{<t_1>}\right] + b_u\right)$$
$$\Gamma_r = \sigma\left(W_r\left[h^{<t_0>}, x^{<t_1>}\right] + b_r\right)$$
$$h'^{<t_1>} = \tanh\left(W_h\left[\Gamma_r * h^{<t_0>}, x^{<t_1>}\right] + b_h\right)$$

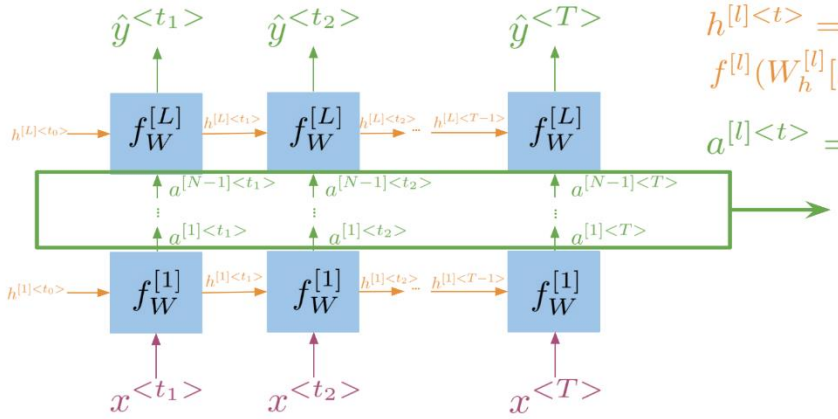
The first gate Γ_u allows you to decide how much you want to update the weights by. The second gate Γ_r, helps you find a relevance score. You can compute the new h by using the relevance gate. Finally you can compute *h,* using the update gate. GRUs "decide" how to update the hidden state. GRUs help preserve important information.

Deep and Bi-directional RNNs

Bi-directional RNNs are important, **because knowing what is next in the sentence could give you more context about the sentence itself.**



So, you can see, in order to make a prediction $y^{\wedge}$, we will **use the hidden states from both directions and combine them to make one hidden state**, we can then proceed as you would with a simple vanilla RNN. When implementing Deep RNNs, you would compute the following.



$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$
$$a^{[l]<t>} = f^{[l]}(W_a^{[l]} h^{[l]<t>} + b_a^{[l]})$$

Machine Learning theoretical concepts

By: Vikram Pal

Note that at layer *l*, you are using the input from the bottom $a^{[l-1]}$ and the hidden state h^l. That allows you to get your new h, and then to get your new a, you will train another weight matrix W_a, which you will multiply by the corresponding h add the bias and then run it through an activation layer.

Tensorflow Syntex:

```
tf.keras.layers.Bidirectional(
    layer,
    merge_mode='concat',
    weights=None,
    backward_layer=None,
    **kwargs
)
```

Layer: keras.layers.RNN instance, such as keras.layers.LSTM or keras.layers.GRU.

Merge: Mode by which outputs of the forward and backward RNNs will be combined. One of {'sum', 'mul', 'concat', 'ave', None}

```
model = Sequential()
forward_layer = LSTM(10, return_sequences=True)
backward_layer = LSTM(10, activation='relu', return_sequences=True,
                      go_backwards=True)
model.add(Bidirectional(forward_layer, backward_layer=backward_layer,
                        input_shape=(5, 10)))
```

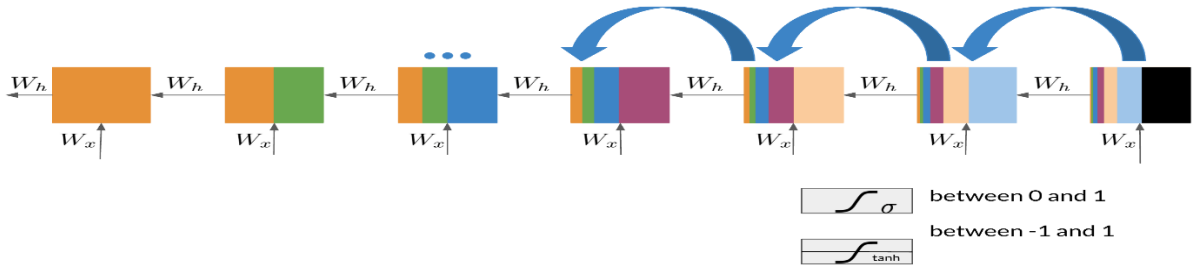
RNNs and Vanishing Gradients

Advantages of RNNs

RNNs allow us to capture dependencies within a short range and they take up less RAM than other n-gram models.

Disadvantages of RNNs

RNNs struggle with longer term dependencies and are very prone to vanishing or exploding gradients. Note that as you are back-propagating through time, you end up getting the following:

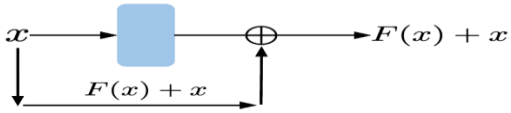


Note that the *sigmoid* and *tanh* functions are bounded by 0 and 1 and -1 and 1 respectively. This eventually leads us to a problem. If you have many numbers that are less than |1|, then as

you go through many layers, and you take the product of those numbers, you eventually end up getting a gradient that is very close to 0. This introduces the problem of vanishing gradients.

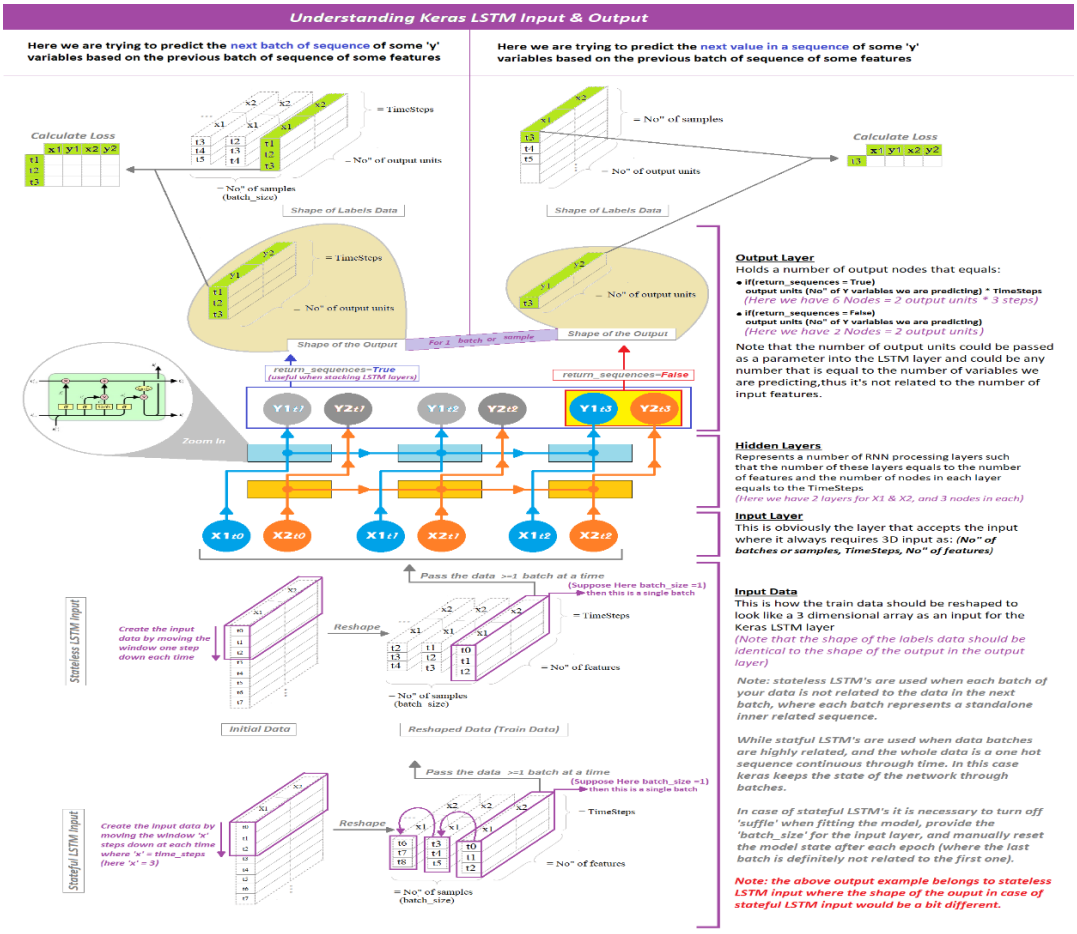
Solutions to Vanishing Gradient Problems

- Identity RNN with ReLU activation $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ -1 ⟶ 0
- Gradient clipping 32 ⟶ 25
- Skip connections



LSTM:

Introduction to LSTMs



Machine Learning theoretical concepts

By: Vikram Pal

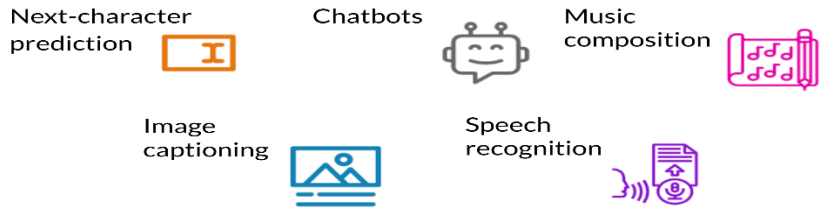
The LSTM allows your model to remember and forget certain inputs. It consists of a cell state and a hidden state with three gates. The gates allow the gradients to flow unchanged. You can think of the three gates as follows:

Input gate: tells you how much information to input at any time point.

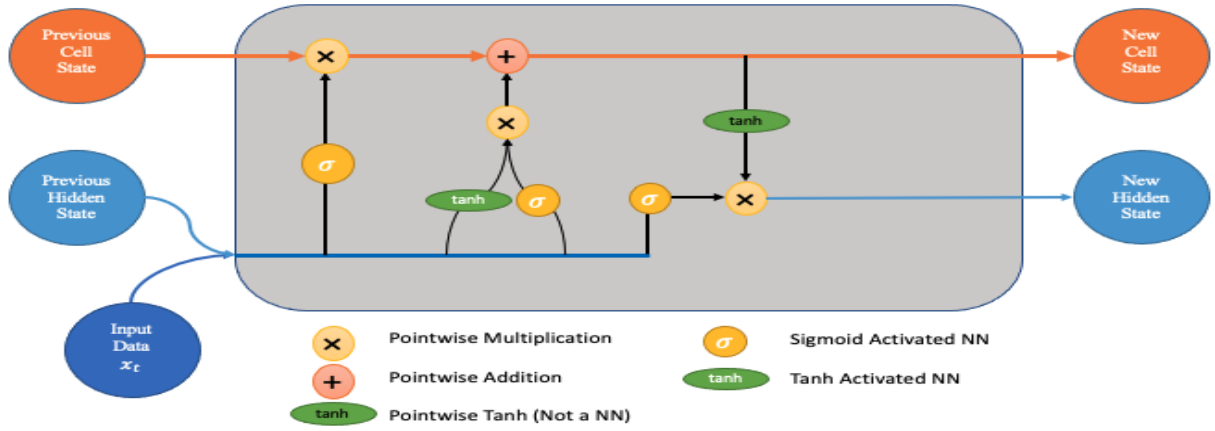
Forget gate tells you how much information to forget at any time point.

Output gate: tells you how much information to pass over at any time point.

There are many applications you can use LSTMs for, such as:

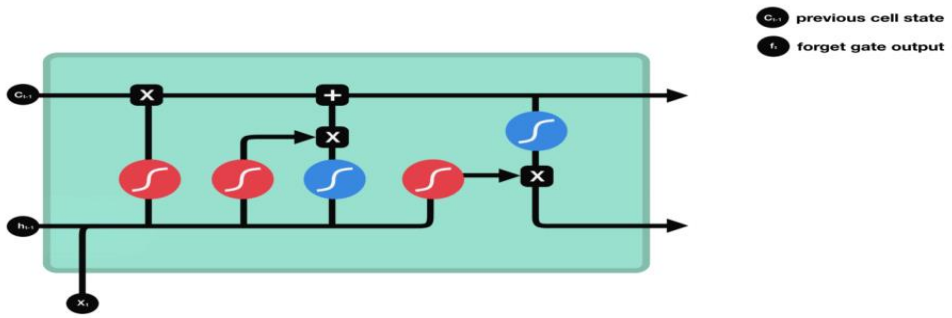


LSTM Architecture:



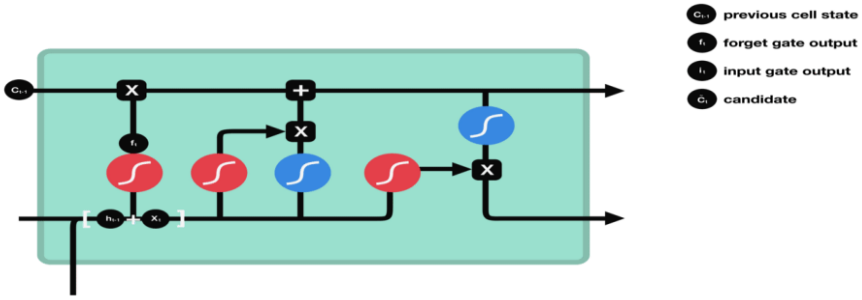
The cell state act as a transport highway that transfers relative information all the way down the sequence chain. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory.

Forget Gate:



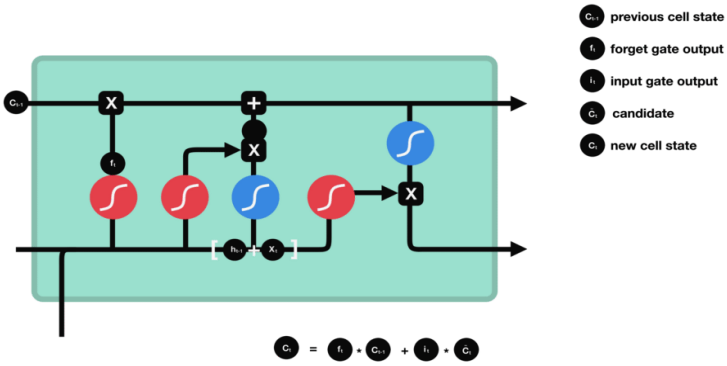
This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the **Truncated Backpropagation**. **Values come out between 0 and 1**. **The closer to 0 means to forget, and the closer to 1 means to keep.**

Input Gate:



First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be **between 0 and 1. 0 means not important, and 1 means important.** You also pass the hidden state and current input into the tanh function to **squish values between -1 and 1 to help regulate the network**. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

Cell State:

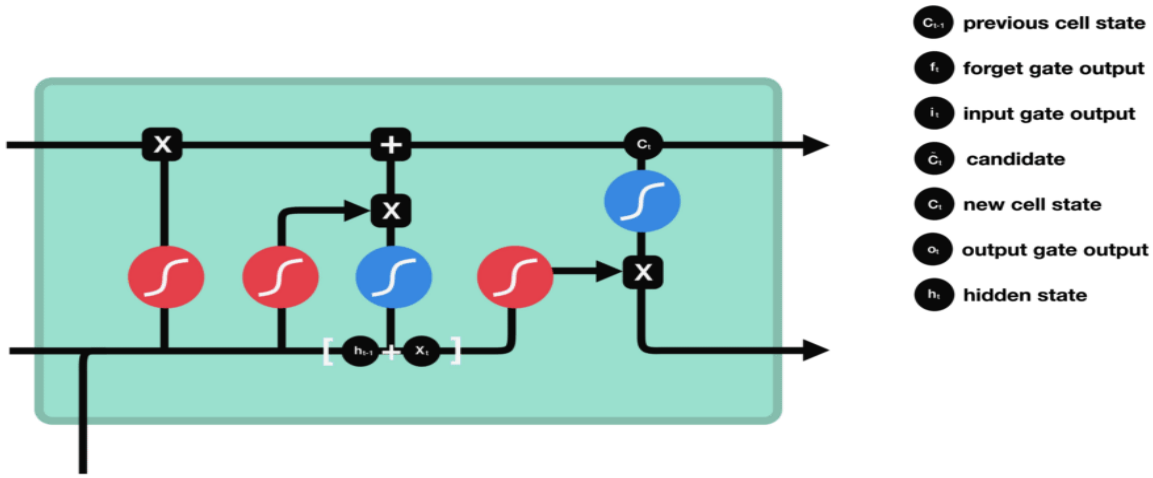


First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.

Machine Learning theoretical concepts

By: Vikram Pal

Output Gate:



Last, we have the output gate. The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

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.

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

Machine Learning theoretical concepts

By: Vikram Pal

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.

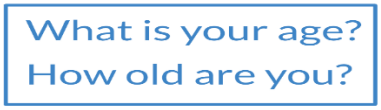
Return States=True and Sequences=True:
We can access both the sequence of hidden state and the cell states at the same time.
This can be done by configuring the LSTM layer to both return sequences and return states.
lstm1, state_h, state_c = LSTM(1, return_sequences=True, return_state=True)

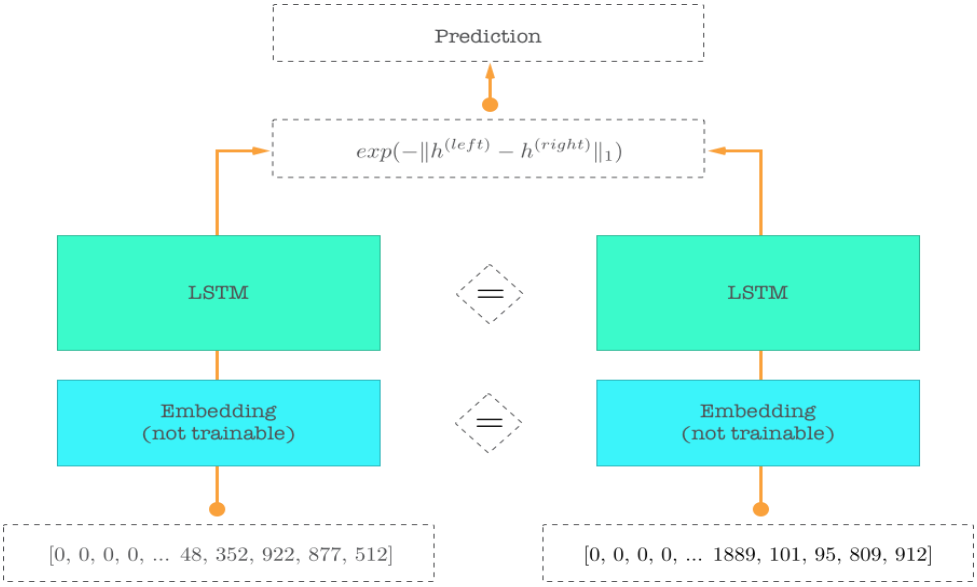
Note:

- 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.

Siamese Network:

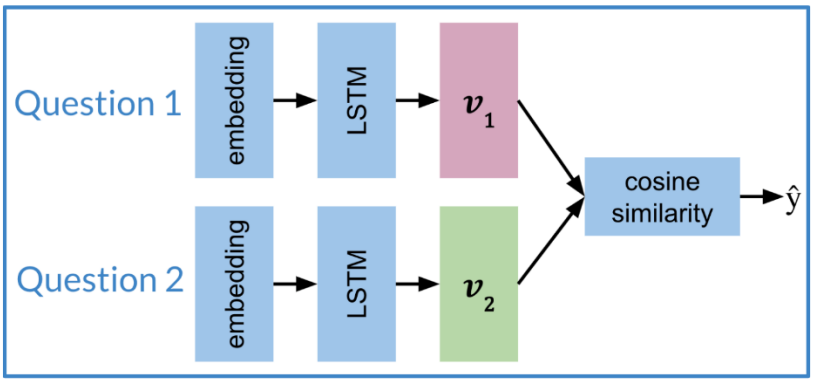
It learns what makes two inputs the same. Siamese networks are networks that have two or more identical sub-networks in them.
The weights and embedding are shared b/w both sides.





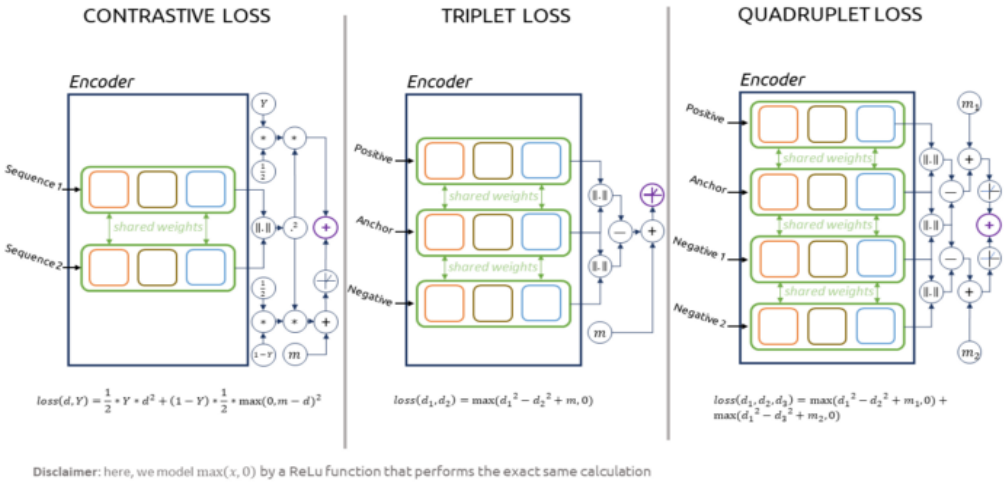
Machine Learning theoretical concepts

By: Vikram Pal



1) Embedding
2) LSTM
3) Vectors
4) Cosine Similarity

Cost Function



Let us take a close look at the following slide:

How old are you? Anchor $\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{||v_1||\,||v_2||}$

What is your age? Positive $\cos(A, P) \approx 1$

Where are you from? Negative $\cos(A, N) \approx -1$

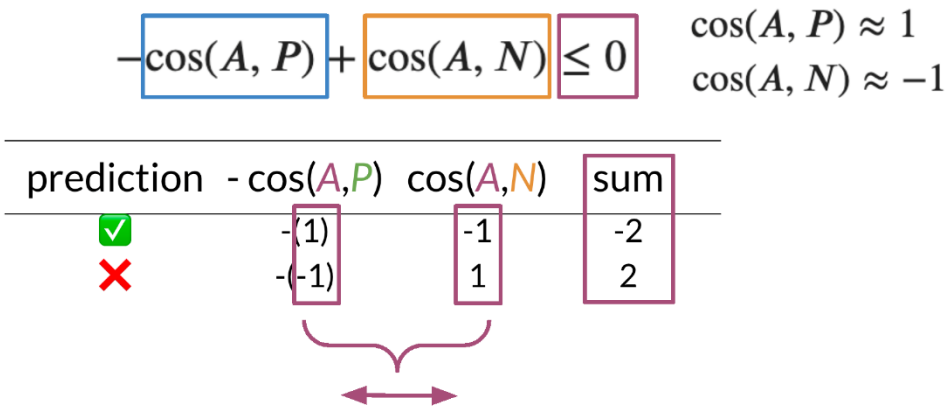
$$-\cos(A, P) + \cos(A, N) \leq 0$$

Note that when trying to compute the cost for a **Siamese network we use the triplet loss.** The triplet loss usually consists of an Anchor and a Positive example. Note that the anchor and the positive example have a cosine similarity score that is very close to one. On the other

hand, the anchor and the negative example have a cosine similarity score close to -1. Now we are ideally trying to optimize the following equation:

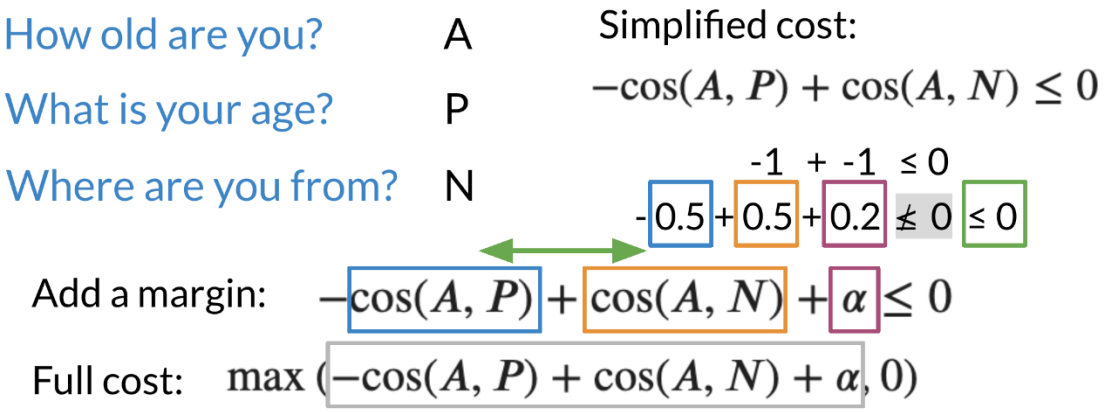
$$-\cos(A,P)+\cos(A,N)\leq 0$$

Note that if $\cos(A,P)=1$ is 1 and $\cos(A,N)=-1$, then the equation is definitely less than 0. However, as cos(A,P) deviates from 1 and cos(A,N) deviates from -1, then you can end up getting a cost that is > 0. Here is a visualization that would help you understand what is going on. Feel free to play with different numbers.



Triplets

We will now build on top of our previous cost function. To get the full cost function you will add a margin.



Note that we added an α in the equation above. This allows you to have a margin of "safety". When computing the full cost, we take the max of that the outcome of $-\cos(A,P)+\cos(A,N)+\alpha$ and 0. Note, we do not want to take a negative number as a cost.

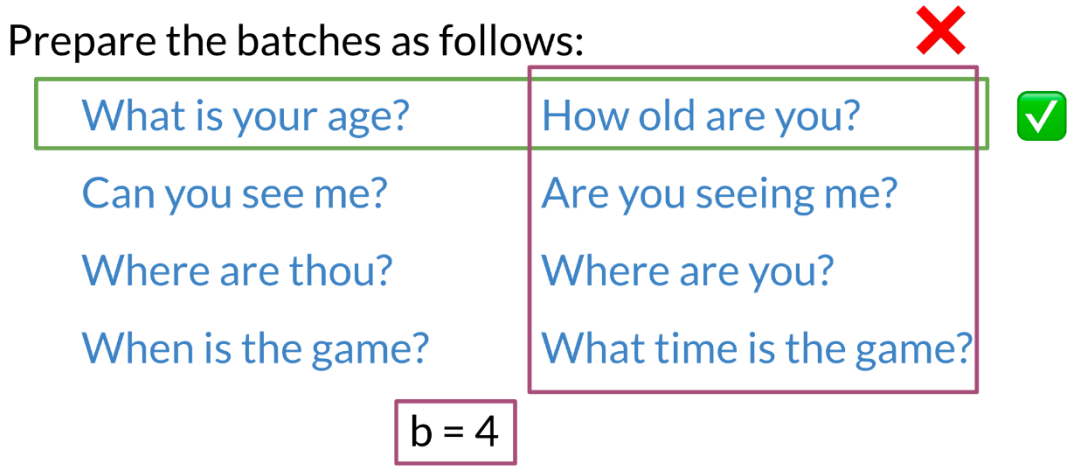
Here is a quick summary:

- α **:** controls how far cos(*A*,*P*) is from cos(*A*,*N*)

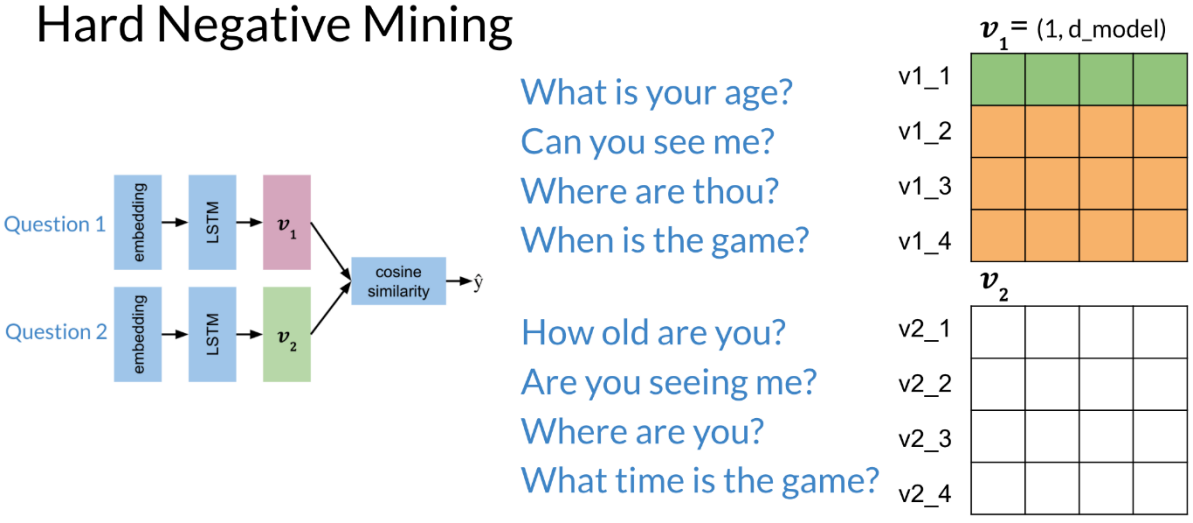
- **Easy** negative triplet: cos(*A*,*N*) < cos(*A*,*P*)
- **Semi-hard** negative triplet: cos(*A*,*N*) < cos(*A*,*P*) < cos(*A*,*N*) + α
- **Hard** negative triplet: cos(*A*,*P*) < cos(*A*,*N*)

Computing the Cost I

To compute the cost, we will prepare the batches as follows:



Note that each example, has a similar example to its right, but no other example means the same thing. We will now introduce hard negative mining.



Each horizontal vector corresponds to the encoding of the corresponding question. Now when you multiply the two matrices and compute the cosine, you get the following:

Machine Learning theoretical concepts

By: Vikram Pal

$\cos(v_1, v_2)$

v_2 \ v_1	_1	_2	_3	_4
_1	0.9	-0.8	-0.3	-0.5
_2	-0.8	0.5	-0.1	-0.2
_3	-0.3	-0.1	0.7	-0.8
_4	-0.5	-0.2	-0.8	1.0

$$\text{Cost} = \max(-\cos(A, P) + \cos(A, N) + \alpha, 0)$$

The diagonal line corresponds to scores of similar sentences, (normally they should be positive). The off diagonals correspond to cosine scores between the anchor and the negative examples.

Computing the Cost II

Now that you have the matrix with cosine similarity scores, which is the product of two matrices, we go ahead and compute the cost.

$\cos(v_1, v_2)$

v_2 \ v_1	_1	_2	_3	_4
_1	0.9	-0.8	-0.3	-0.5
_2	-0.8	0.5	-0.1	-0.2
_3	-0.3	-0.1	0.7	-0.8
_4	-0.5	-0.2	-0.8	1.0

mean_neg: speeds up training

closest_neg: helps penalize the cost more

We now introduce two concepts, the **mean_neg,** which is the **mean negative of all the other off diagonals in the row**, and the **closest_neg,** which corresponds to the **highest number in the off diagonals**.

$$\text{Cost}=\max(-\cos(A,P)+\cos(A,N)+\alpha,0)$$

So we will have two costs now:

$$\text{Cost1}=\max(-\cos(A,P)+mean_neg)+\alpha,0)$$

$$\text{Cost2}=\max(-\cos(A,P)+closest_neg+\alpha,0)$$

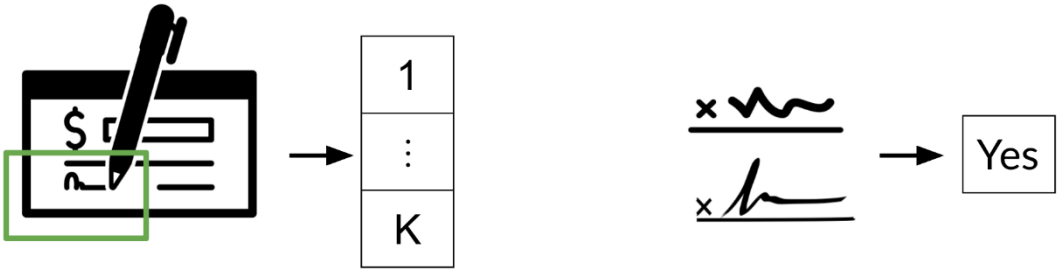
Machine Learning theoretical concepts

By: Vikram Pal

The full cost is defined as: Cost 1 + Cost 2.

One Shot Learning

Imagine you are working in a bank and you need to verify the signature of a check. You can either build a classifier with K possible signatures as an output or you can build a classifier that tells you whether two signatures are the same.



Classification: classifies input as 1 of K classes

One Shot Learning: measures similarity between 2 classes

Hence, we resort to one shot learning. Instead of retraining your model for every signature, you can just learn a similarity score as follows:

Learn a similarity score!

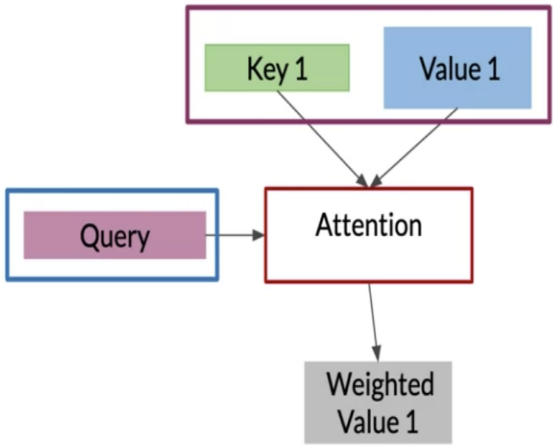
$\cos(sig1, sig2) > \tau$ ✅

$\cos(sig1, sig2) \leq \tau$ ❌

Machine Learning theoretical concepts

By: Vikram Pal

Attention:



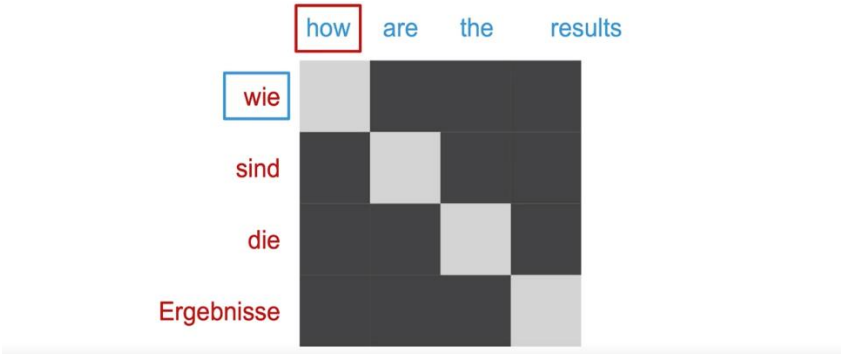
The intuition here is that you want to identify the corresponding words in the queries that are similar to the keys. This would allow your model to "look" or focus on the right place when translating each word. We then run a SoftMax

$$softmax(QK_T)$$

That allows us to get a distribution of numbers between 0 and 1. One more step is required. We then would multiply the output by VV. Remember VV in this example was the same as our **keys,** corresponding to the English word embeddings. Hence the equation becomes

$$softmax(QK_T)V$$

This tells us how much of each word, or which combination of words we will be feeding into our decoder to predict the next German word. This is called scale dot product attention. Note we add a normalization constant to it later, but you do not have to worry about that right now.



Seq-2-Seq:

Encoder encodes the input into a new representation
Decoder decodes the encoded representation of the input into output

Machine Learning theoretical concepts

By: Vikram Pal

- Training: During training, we train the encoder and decoder such that they work together to create a **context (representation) between input and output**
- Inference (Prediction): After learning how to create the context (representation), they can work together to predict the output
- **Encode all- decode one at a time**: Mostly, the **encoder** reads all the input sequence and creates a **context (representation) vector**. Decoder use this **context (representation) vector** and previously decoded result to create new output step by step.
- Teacher forcing: **During** training **decoder receives the** correct **output from the training set as the previously decoded result to predict the next output. However, during** inference **decoder receives the previously decoded result to predict the next output.** Teacher forcing improves the training process.

HOW TO TRAIN AN ENCODER — DECODER WITH TEACHER FORCING

The initial steps are the same:

- Decoder produces the output sequence one by one
- For each output, the decoder consumes a **context vector and an input**
- The initial context vector is created by the encoder
- The initial input is a special symbol for the decoder to make it start, e.g. 'start'
- Using initial context and initial input, the decoder will generate the first output

Sampling and Decoding

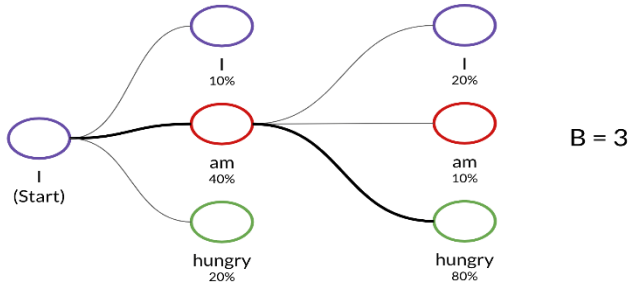
You use decoding after all the calculations have been performed on the encoder hidden states and when you are ready to predict the next token.

Decoding types:

1. Greedy Decoding: it is a simplest way to decode the model's predictions as it select the most probability word at every step and concatenate all predicted tokens for the output sequence.

2. Random Sampling: **In random sampling you provide probabilities for each word,** and **sample accordingly for the next outputs**. In sampling, **temperature is a parameter you can adjust to allow for more or less randomness in your predictions**. It is measured **on a scale of 0-1, indicating low to high randomness**.

3. Beam Search: Instead of offering a single best output like in greedy decoding, **beam search selects multiple options based on conditional probability**. The search restriction is the **beam width parameter B**, which **limits the number of branching paths based on a number that you choose**, such as three. Then at each time step, the beam search selects B number of best alternatives with the highest probability as the most likely choice for the time step. Once you have these B possibilities, you can choose the one with the highest probability.



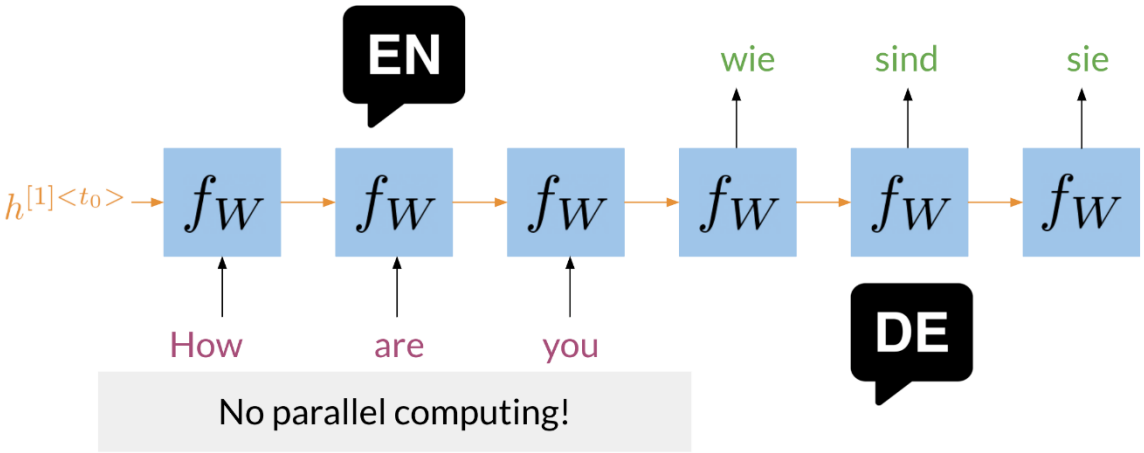
4.

Minimum Bayes Risk (MBR)

Compares many samples against one another. To implement MBR:

- Generate several random samples
- Compare each sample against all the others and assign a similarity score (such as ROUGE!)
- Select the sample with the highest similarity: the golden one ✨

Transformers vs RNNs

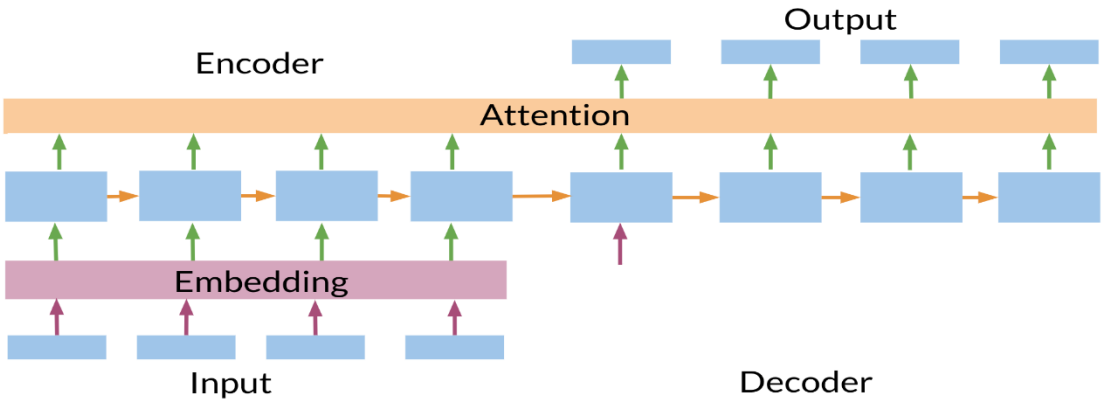


In the image above, you can see a typical RNN that is used to translate the English sentence "How are you?" to its German equivalent, "Wie sind Sie?". One of the biggest issues with these RNNs, is that they make use of sequential computation. That means, in order for your code to

process the word "you", it has to first go through "are" and then "you". Two other issues with RNNs are the:

- Loss of information: For example, it is harder to keep track of whether the subject is singular or plural as you move further away from the subject.
- Vanishing Gradient: when you back-propagate, the gradients can become really small and as a result, your model will not be learning much.

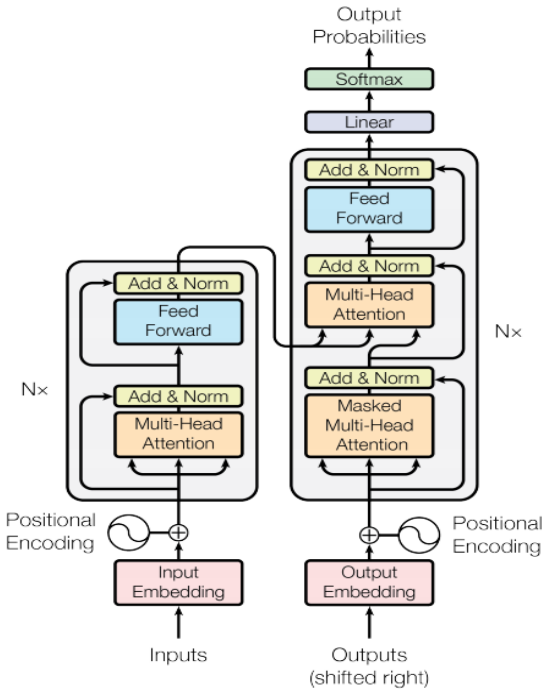
In contrast, transformers are based on attention and don't require any sequential computation per layer, only a single step is needed. Additionally, the **gradient steps that need to be taken from the last output to the first input in a transformer is just one**. For RNNs, the number of steps increases with longer sequences. Finally, transformers don't suffer from vanishing gradients problems that are related to the length of the sequences. Here is an image that might help you visualize it.



Machine Learning theoretical concepts

By: Vikram Pal

Transformer:



A transformer model follows the same general pattern as a **standard sequence to sequence with attention model.**

- **The input sentence is passed through N encoder layers that generates an output for each token in the sequence.**
- **The decoder attends to the encoder's output and its own input (self-attention) to predict the next word.**

Encoder layer:

Each encoder layer consists of sublayers:

- **Multi-head attention** (with padding mask)
- **Point wise feed forward networks.**

Each of these sublayers has a residual connection around it followed by a layer normalization. Residual connections help in avoiding the vanishing gradient problem in deep networks.

Decoder layer

Each decoder layer consists of sublayers:

- **Masked multi-head attention** (with look ahead mask and padding mask)
- **Multi-head attention** (with padding mask). V (value) and K (key) receive the encoder output as inputs. Q (query) receives the output from the masked multi-head attention sublayer.
- **Point wise feed forward networks**

Machine Learning theoretical concepts

By: Vikram Pal

Each of these sublayers has a residual connection around it followed by a layer normalization. The output of each sublayer is LayerNorm(x + Sublayer(x)). The normalization is done on the d_model (last) axis.

Hyperparameters:

- `num_layers = 4`
- `d_model = 128`
- `dff = 512`
- `num_heads = 8`
- `dropout_rate = 0.1`

Input and positional embedding:

We don't feed text input directly to transformer. We use indices of input. In input embedding, the above vector initialized at random number.

Position Embedding:

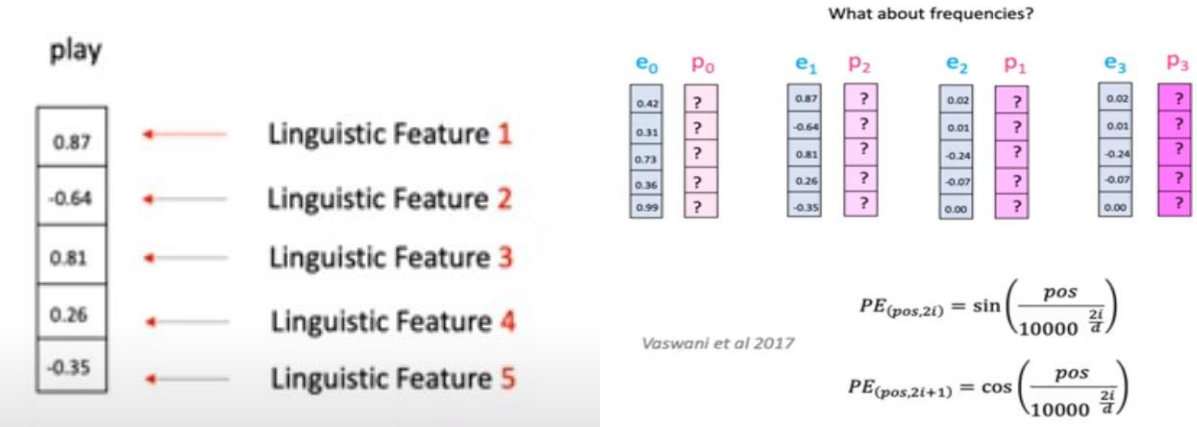
Attention layers see their input as a set of vectors, with no sequential order. This model also doesn't contain any recurrent or convolutional layers. Because of this a **"positional encoding" is added to give the model some information about the relative position of the tokens in the sentence**.

The positional encoding vector is added to the embedding vector. Embeddings represent a token in a d-dimensional space where tokens with similar meaning will be closer to each other. But the embeddings do not encode the relative position of tokens in a sentence. So, after adding the positional encoding, **tokens will be closer to each other based on the similarity of their meaning and their position in the sentence**, in the d-dimensional space.

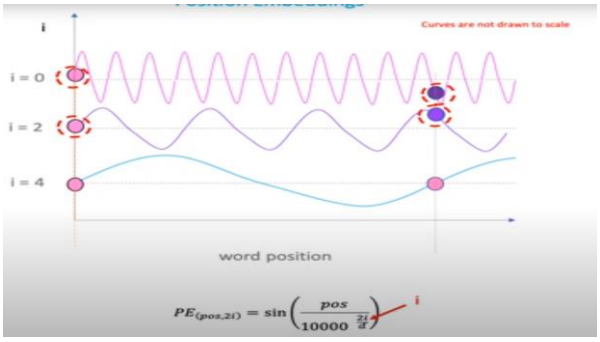
The formula for calculating the positional encoding is as follows:

Machine Learning theoretical concepts

By: Vikram Pal



Each number in the vector represent some linguistic features. We use wave frequency to capture position information.

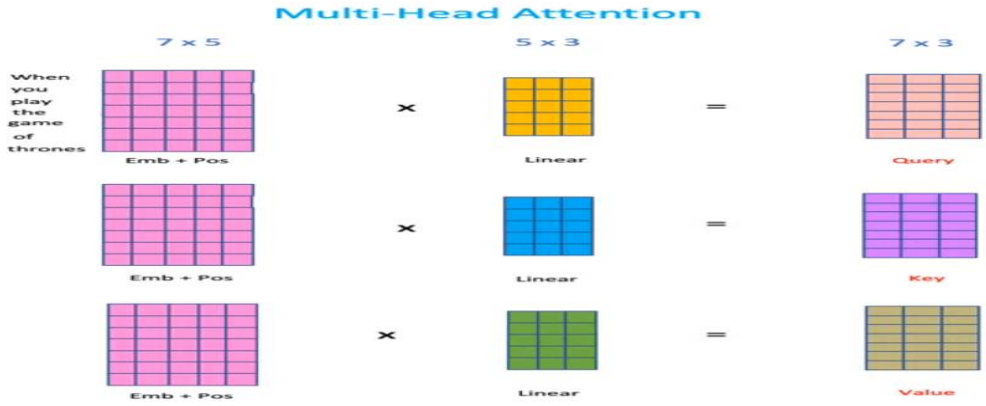


Linear Layer:

It has two functions

1. Mapping inputs onto the outputs.
2. Changing matrix/vector dimensions.

Linear layer weights are feed as matrix to model.

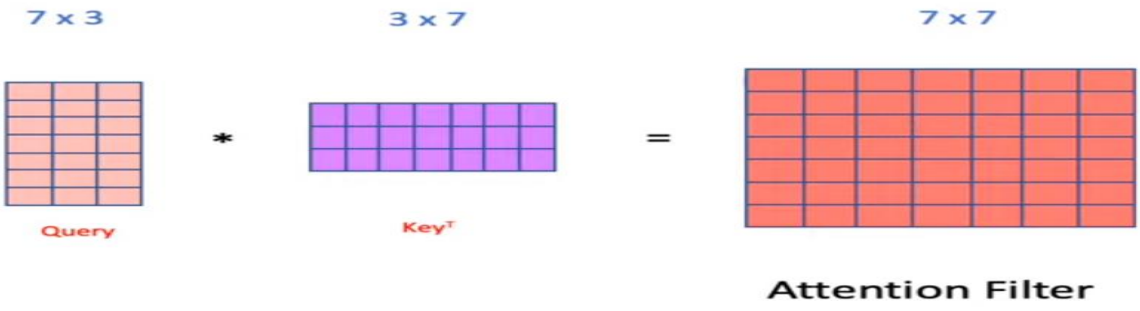


Attention filters/Scaled Dot Product:

It is a dot product of query and key.

Machine Learning theoretical concepts

By: Vikram Pal



7 x 7

	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99

$$\frac{}{\sqrt{d_k}}$$

dk=7 in our case.

Softmax

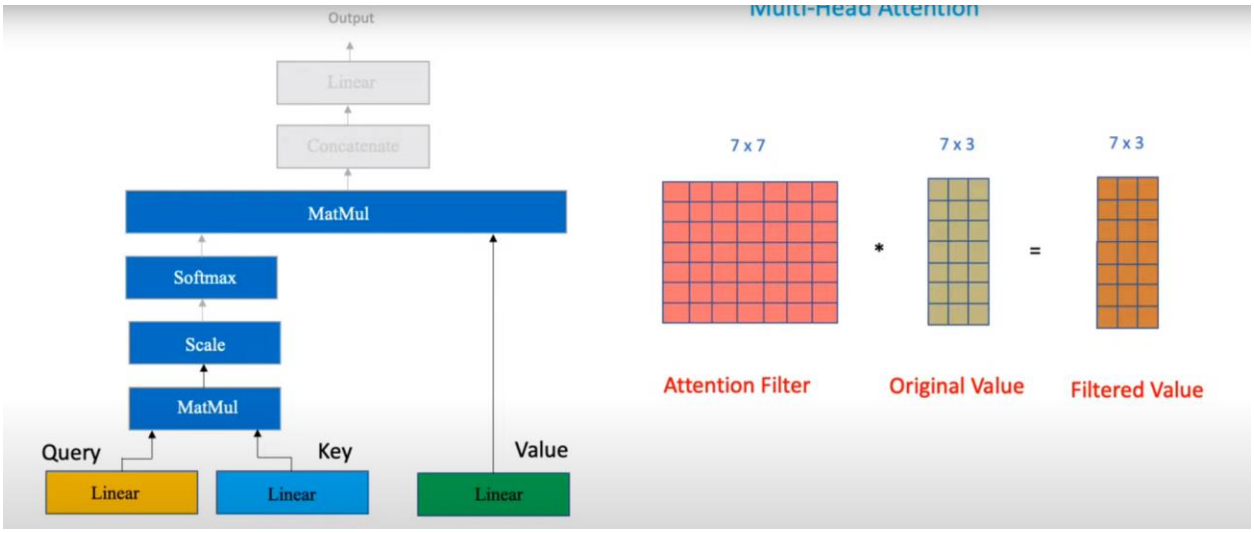
7 x 7

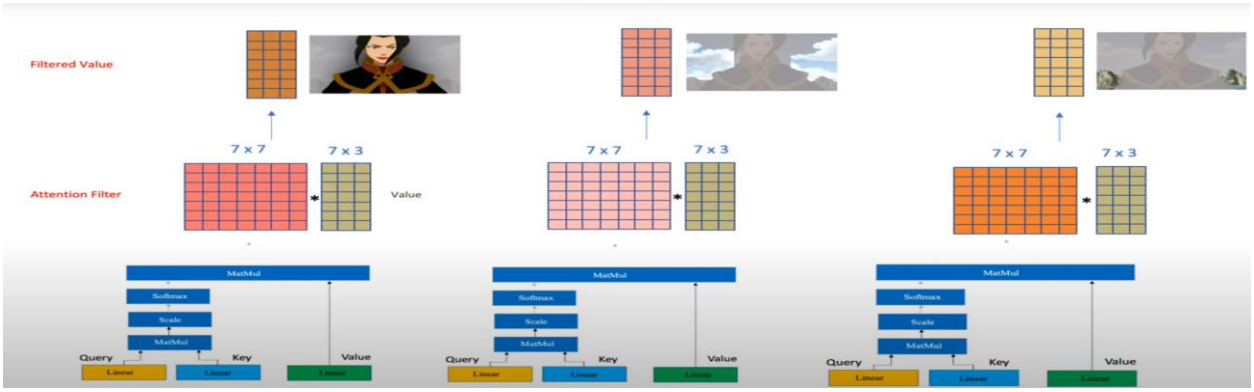
	When	you	play	the	game	of	thrones
When	33.6	7.6	15.5	3.8	20.8	3.8	22.3
you	7.6	34.0	30.6	8.3	26.5	5.7	27.2
play	15.5	30.6	35.9	3.8	34.0	11.3	34.8
the	3.8	8.3	3.8	34.8	33.3	15.1	33.6
game	20.8	26.5	34.0	33.3	37.0	16.6	35.9
of	3.8	5.7	11.3	15.1	16.6	32.1	22.3
thrones	22.3	27.2	34.8	34.0	35.9	22.3	37.4

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_i)}$$

Machine Learning theoretical concepts

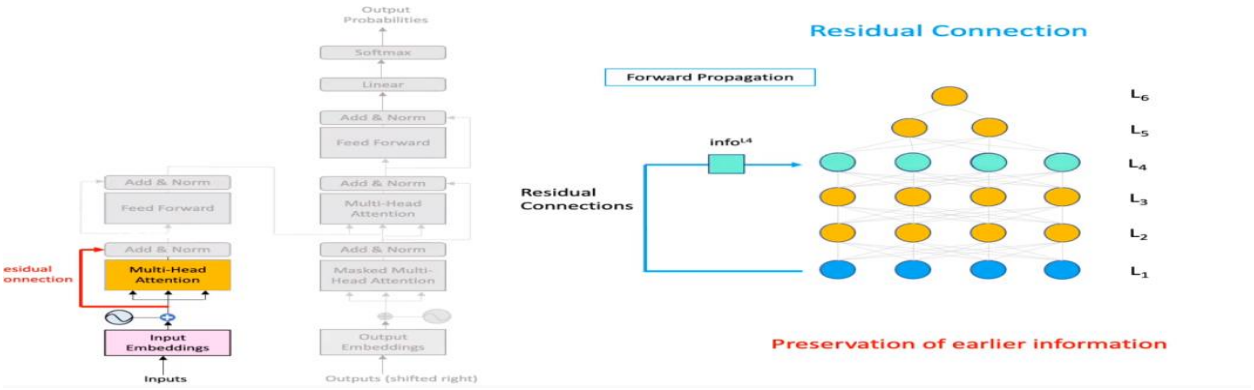
By: Vikram Pal





Residual connections: It has mainly two tasks

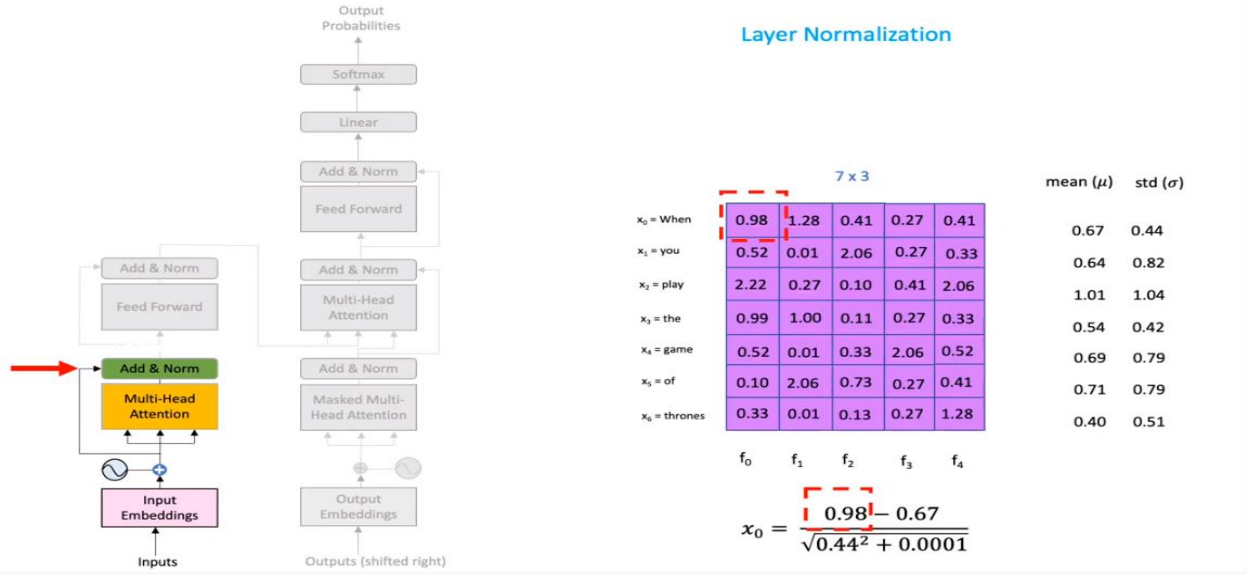
1. **Knowledge Preservation.**
2. **Vanishing Gradient Problem.**



In Normalization, we computer z-score of matrix.

Machine Learning theoretical concepts

By: Vikram Pal

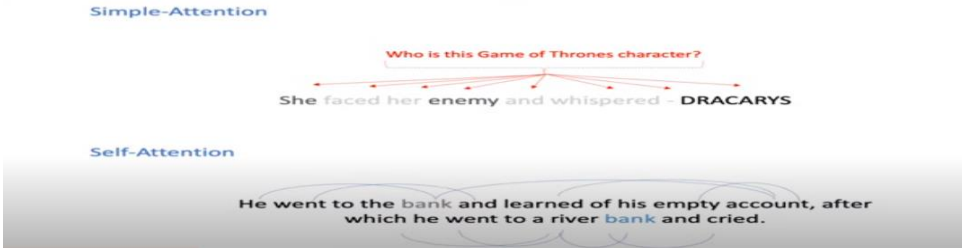


Masking:

Mask all the pad tokens in the batch of sequence. **It ensures that the model does not treat padding as the input.** The mask indicates where pad value 0 is present: it outputs a 1 at those locations, and a 0 otherwise.

Simple Attention vs self Attention:

The simple attention pays attention based on external query and the self attention pays attention with themselves.



Why different multiple heads?

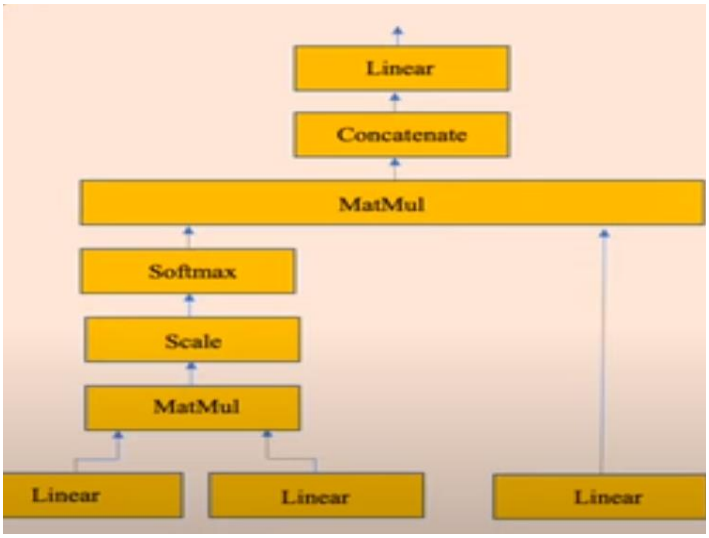
The reason each head is different is because they each learn a different set of weight matrices $\{W_i^Q, W_i^K, W_i^V\}$ where i is the index of the head. To clarify, the input to each attention head is the same. For attention head i:

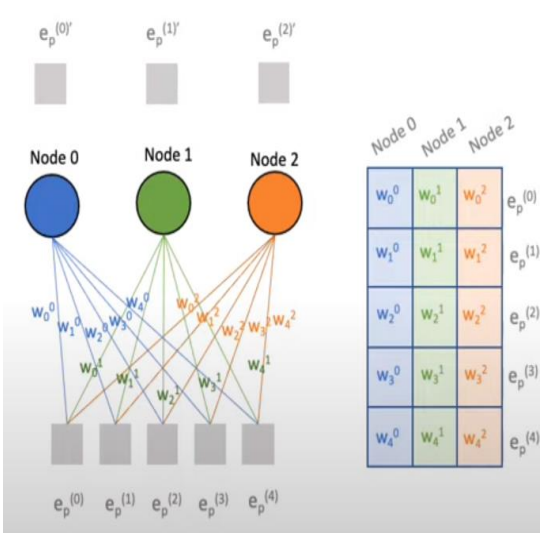
$$Q_i(x) = xW_i^Q$$
$$K_i(x) = xW_i^K$$
$$V_i(x) = xW_i^V$$
$$\text{attention}_i(x) = \text{softmax}\left(\frac{Q_i(x)K_i(x)^T}{\sqrt{d_k}}\right)V_i(x).$$

We have **"narrow self-attention"** in which the original input is split into smaller chunks and each head get its own small input.

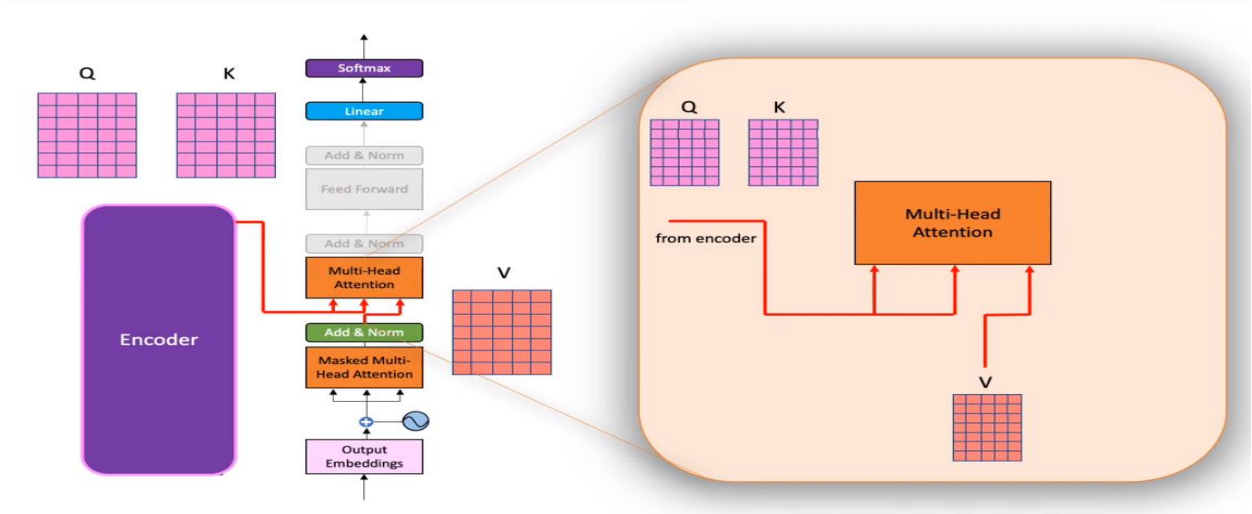
We also have **"wide self-attention"** in which the whole input gets fed into each head separately. Wide self-attention gives better results at the cost of memory and computation time.

Multiheaded Attention:





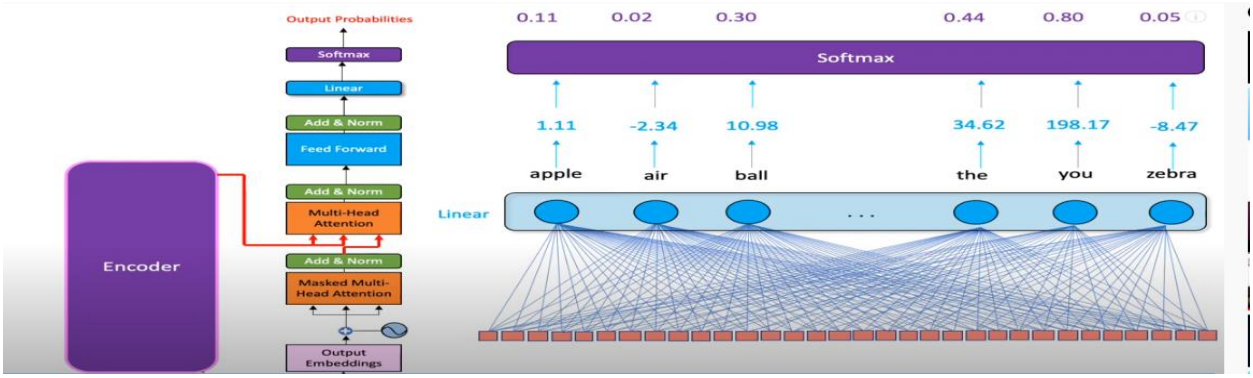
Decoder:



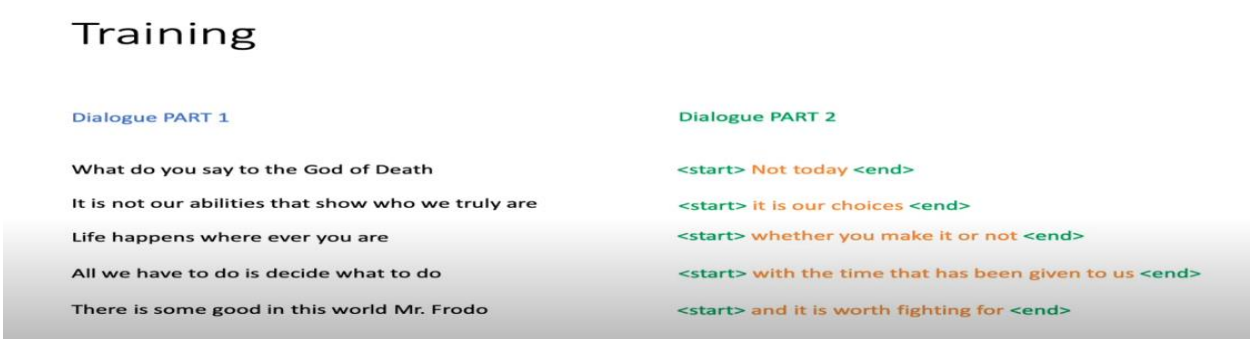
Machine Learning theoretical concepts

By: Vikram Pal

Decoder linear layer



Training Data example:



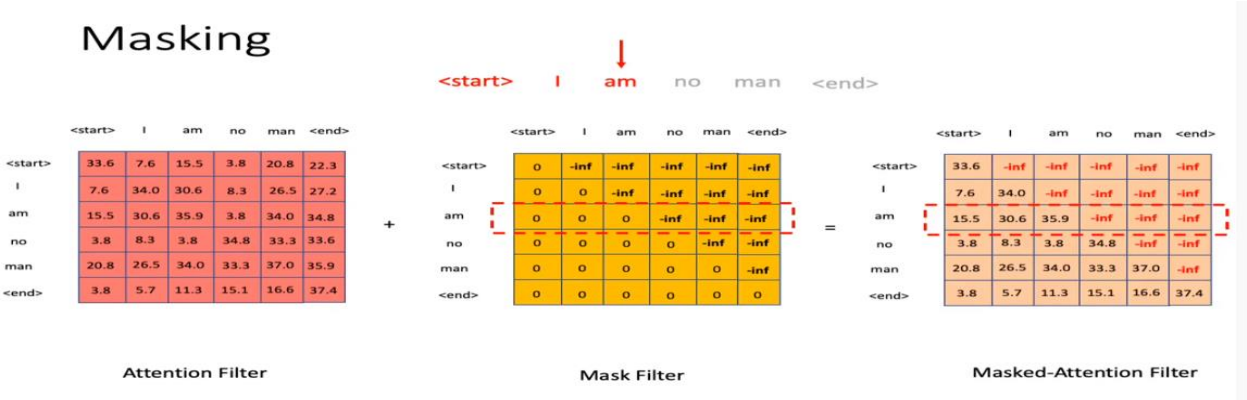
Decoder Masking:

Masking is needed to prevent the **attention mechanism of a transformer** from "cheating" in the decoder when training (on a translating task for instance). This kind of " cheating-proof masking" is not present in the encoder side.

We do not want the **attention mechanism to share any information regarding the token at the next positions**, when giving a prediction using all the previous tokens.

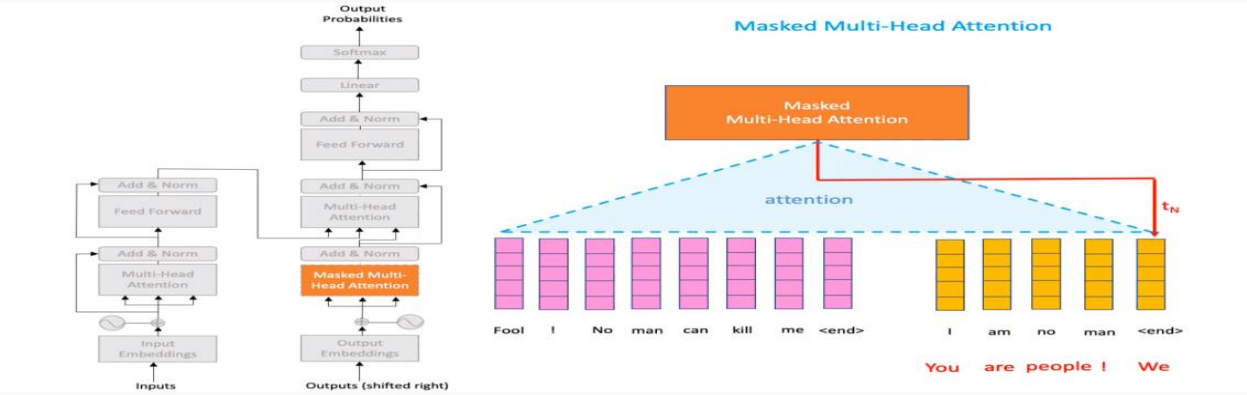
To ensure that this is done, **we mask future positions (setting them to -inf) before the softmax step in the self-attention calculation**.

Below is a commented example summing up how the masking works with a two token long sequence ("Hello there." for instance)



Machine Learning theoretical concepts

By: Vikram Pal



Transformer XL:

Mechanism: segment-level recurrence mechanism

Transformers have a potential of learning longer-term dependency but are limited by a **fixed-length context in the setting of language modeling**.

We propose a novel neural architecture Transformer-XL that enables **learning dependency beyond a fixed length without disrupting temporal coherence**.

It consists of a **segment-level recurrence mechanism and a novel positional encoding scheme**.

Our method not only enables capturing **longer-term dependency, but also resolves the context fragmentation problem.**

As a result, Transformer-XL learns dependency that **is 80% longer than RNNs and 450% longer than vanilla Transformers**, achieves better performance on **both short and long sequences, and is up to 1,800+ times faster than vanilla Transformers during evaluation**.

Working:

The notion of recurrence into our deep self-attention network. In particular, instead of computing the hidden states from scratch for each new segment, **we reuse the hidden states obtained in previous segments.**

The **reused hidden states serve as memory for the current segment**, which builds up a **recurrent connection between the segments**. As a result, modeling very **long-term dependency becomes possible because information can be propagated through the recurrent connections**.

Meanwhile, passing information from the previous segment can also resolve the problem of context fragmentation. More importantly, we show the necessity of using **relative positional encodings rather than absolute ones**, in order to enable state reuse without causing temporal confusion. Hence, as an additional technical contribution, we introduce a simple but more effective relative **positional encoding formulation that generalizes to attention lengths longer than the one observed during training**.

Machine Learning theoretical concepts

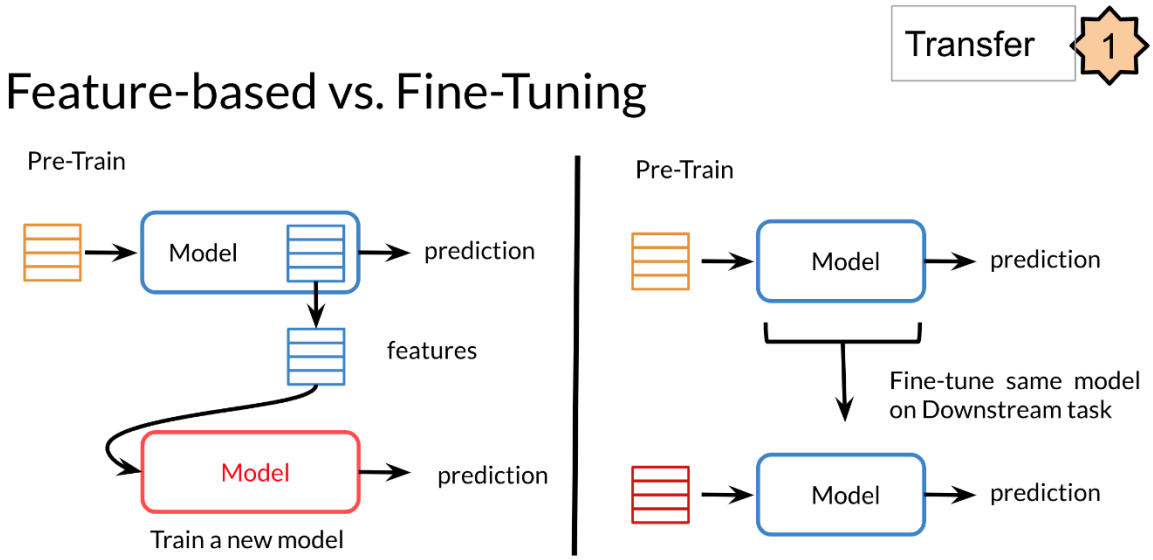
By: Vikram Pal

Transfer Learning in NLP:

There are three main advantages to transfer learning:

- Reduce training time
- Improve predictions
- Allows you to use smaller datasets

Two methods that you can use for transfer learning are the following:

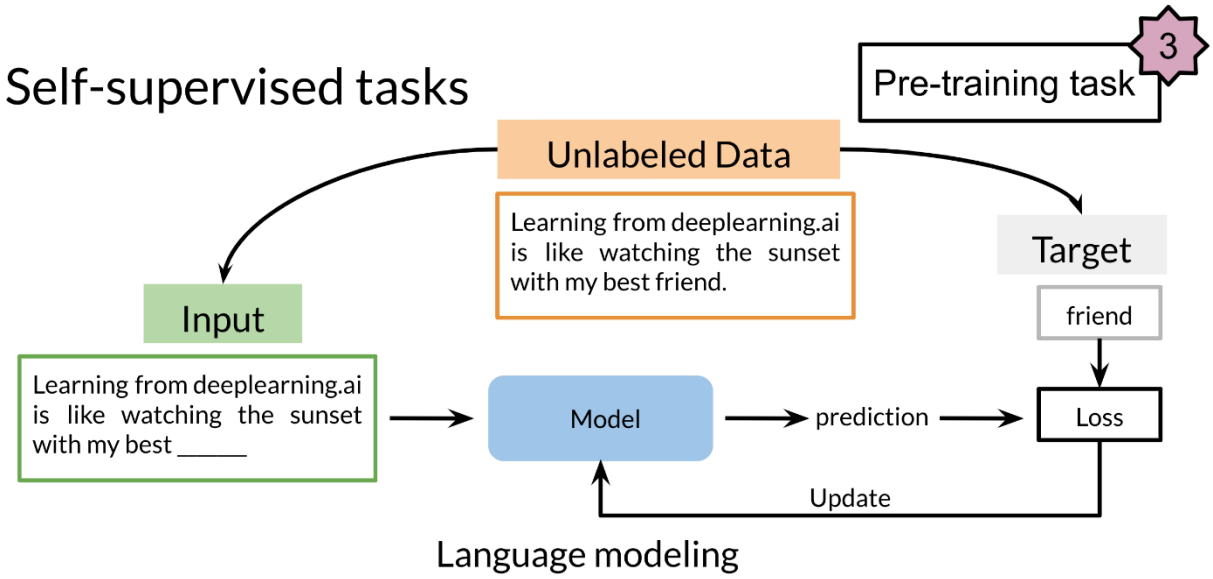


In feature based, you can train word embeddings by running a different model and then using those features (i.e. word vectors) on a different task.

When fine tuning, you can use the exact same model and just run it on a different task. Sometimes when fine tuning, you can keep the model weights fixed and just add a new layer that you will train. Other times you can slowly unfreeze the layers one at a time. You can also use unlabelled data when pre-training, by masking words and trying to predict which word was masked.

Machine Learning theoretical concepts

By: Vikram Pal



For example, in the drawing above we try to predict the word "friend". This allows your model to get a grasp of the overall structure of the data and to help the model learn some relationships within the words of a sentence.

BERT: (Bidirectional Encoder Representation from Transformers):

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google.

The original English-language BERT has two models:

(1) the BERTBASE: 12 encoders with 12 bidirectional self-attention heads

(2) the BERTLARGE: 24 encoders with 16 bidirectional self-attention heads.

Bert Training:

Bert training is combination of Unsupervised and supervised machine learning.

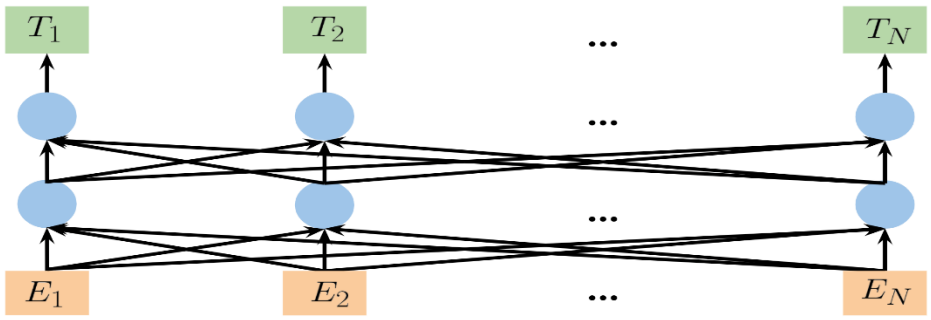
Unsupervised training: Bert learns contextual representation of the data.

Both models are pre-trained from unlabeled data extracted from the Books Corpus with 800M words and English Wikipedia with 2,500M words.

Machine Learning theoretical concepts

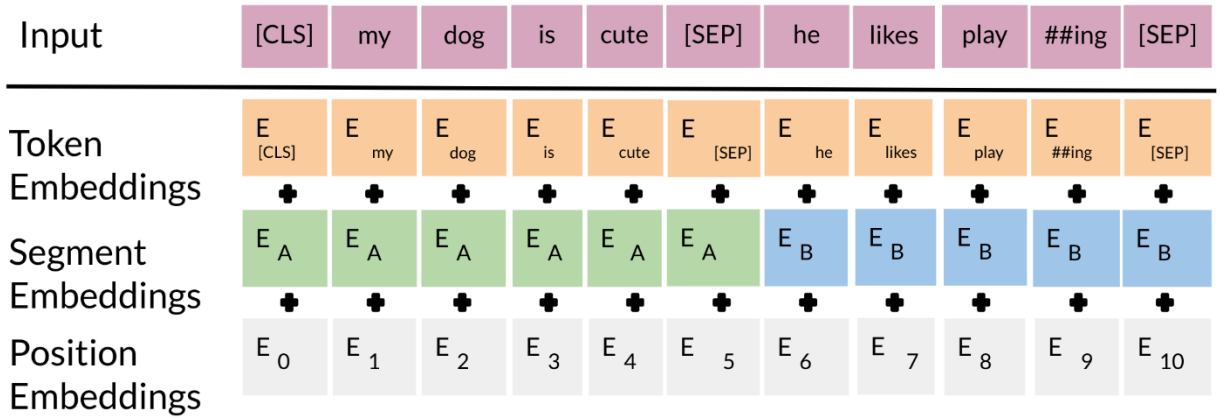
By: Vikram Pal

- Makes use of transfer learning/pre-training:



BERT Objective

We will first start by visualizing the input.

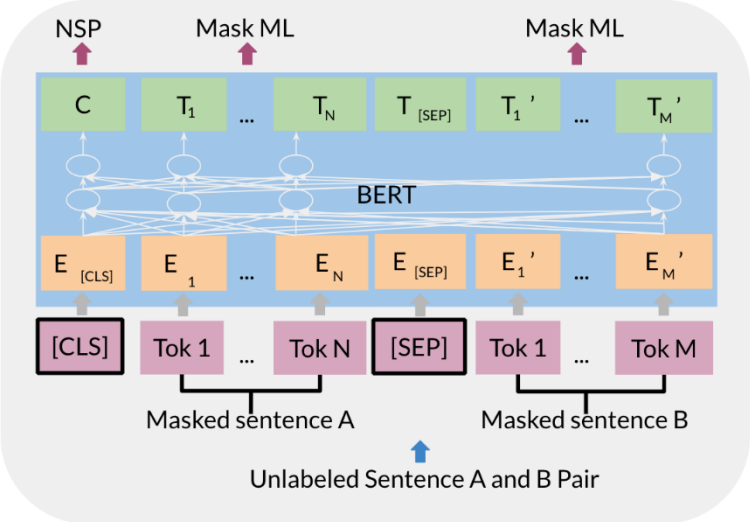


The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

- **The input embeddings:** you have a CLS token to indicate the beginning of the sentence and a sep to indicate the end of the sentence
- **The segment embeddings** allow you to indicate whether it is sentence a or b.
- **Positional embeddings:** allows you to indicate the word's position in the sentence.

Machine Learning theoretical concepts

By: Vikram Pal



- [CLS]: a special classification symbol added in front of every input
- [SEP]: a special separator token

Supervised Training:

The C token in the image above could be used for classification purposes. The unlabeled sentence A/B pair will depend on what you are trying to predict, it could range from question answering to sentiment. (in which case the second sentence could be just empty). The BERT objective is defined as follows:

Objective 1:
Multi-Mask LM

Loss: Cross Entropy Loss

Objective 2:
Next Sentence Prediction

Loss: Binary Loss

You just combine the losses!

There are two steps in the BERT framework: pre-training and fine-tuning.

During pre-training, the model is trained on unlabeled data over **different pre-training tasks**.

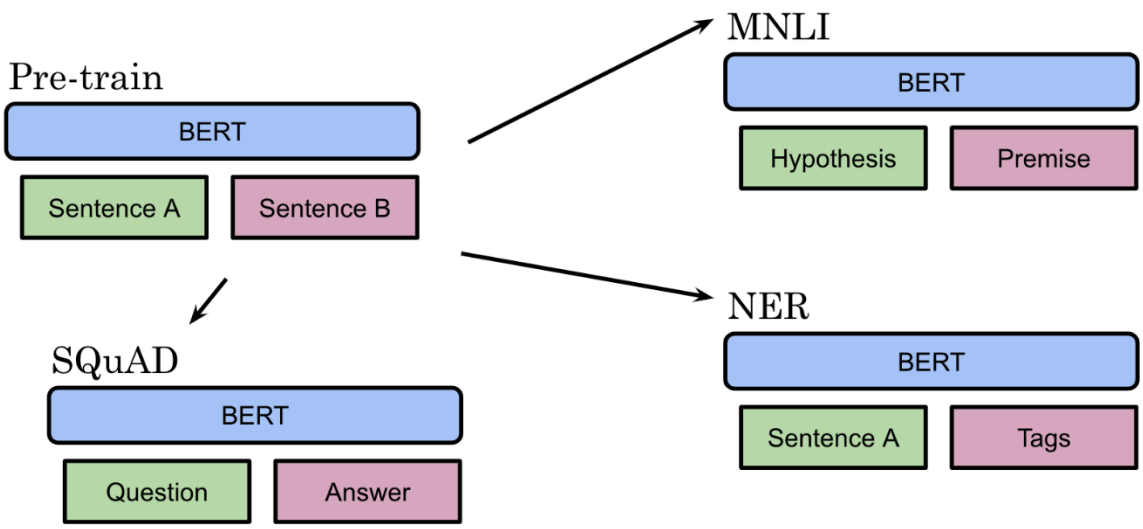
For fine tuning, the BERT model is first **initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks.** For example, in the figure above, you get the corresponding embeddings for the input words, you

run it through a few transformer blocks, and then you make the prediction at each time point T_i.

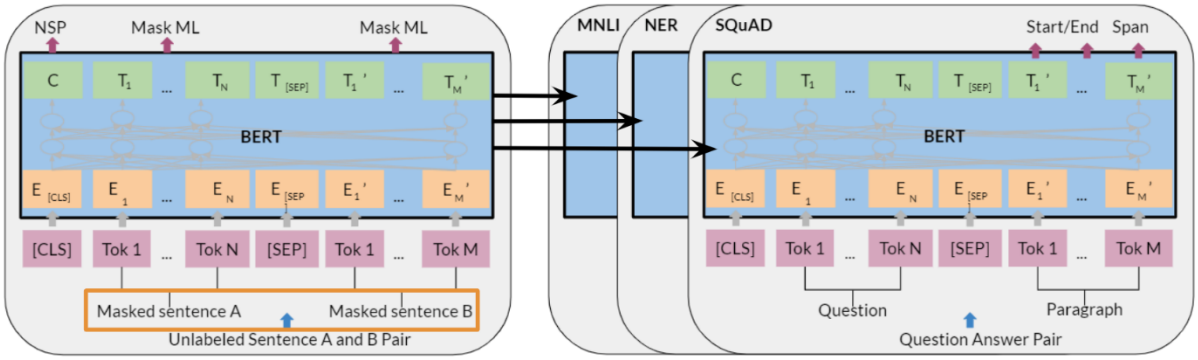
- Choose 15% of the tokens at random: mask them 80% of the time, replace them with a random token 10% of the time, or keep as is 10% of the time.
- There could be multiple masked spans in a sentence
- Next sentence prediction is also used when pre-training.

Fine tuning BERT

Once you have a pre-trained model, you can fine tune it on different tasks.



For example, given a hypothesis, you can identify the premise. Given a question, you can find the answer. You can also use it for named entity recognition. Here is a summary of the inputs.



- You can replace sentences A/B
- Paraphrase from sentence A
- Question/passage

- Hypothesis premise pairs in entailment
- Text and a Ø for classification/sequence tagging
- Output tokens are fed into a layer for token level tasks otherwise use [CLS] embedding as input.

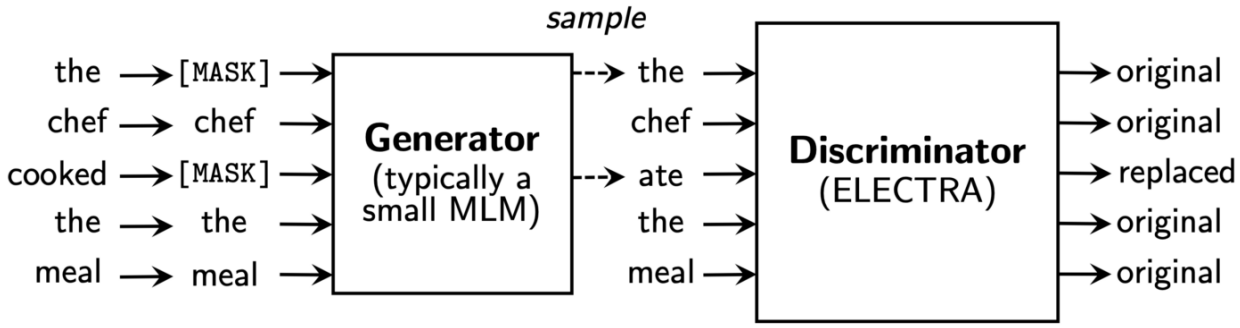
Electra:

ELECTRA replaces the MLM of BERT with Replaced Token Detection (RTD), which looks to be more efficient and produces better results

ELECTRA introduces a pre-training framework that enables BERT-small's GLUE performance to be achieved with the same size model for 12x less compute. It even achieves state-of-the-art results from Roberta with 4x less compute.
Its success can be contributed to two factors: its transformer-encoder architecture (enabling deep, bi-directional language understanding) and its pre-training process.

- MLM replaces 15% of tokens with the special [MASK] token and train BERT to reconstruct the original sequence with the remaining, unmasked context. This approach severely limits the token efficiency, the amount of language understanding gained per token during the pre-training phase.

- The [MASK] token is only present during pre-training, which precedes the fine-tuning step. This results in different token distributions for the two stages, even though Devlin et al. implement measures to limit its negative impact.



As shown in the figure above the pre-training framework utilize a small BERT model, trained through MLM, to construct a corrupted sequence This is achieved through sampling the most likely token at each position where the original token was masked.

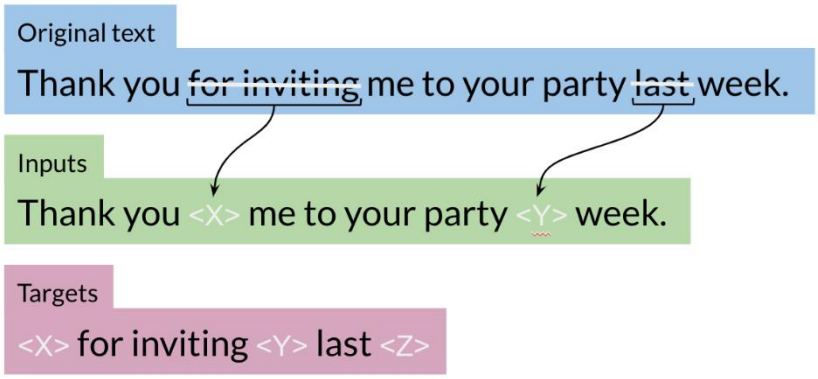
The corrupted sequence, constituting of replaced tokens and some original ones, makes up the input to the discriminator — ELECTRA (which by the way is short for Efficiently Learn an Encoder that Classifies Token Replacements Accurately). This model's task, as the abbreviation subtly suggests, is to take each token and predict whether it has been replaced by the generator.

Transformer T5

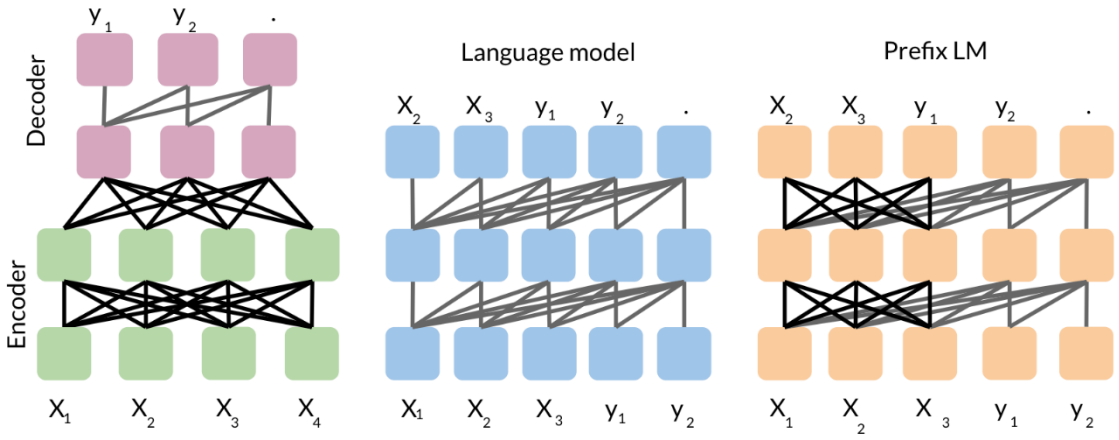
One of the major techniques that allowed the T5 model to reach state of the art is the concept of masking:

Machine Learning theoretical concepts

By: Vikram Pal



For example, you represent the “for inviting” with <X> and last with <Y> then the model predicts what the X should be and what the Y should be. This is exactly what we saw in the BERT loss. You can also mask out a few positions, not just one. The loss is only on the mask for BERT, for T5 it is on the target.



So we start with the basic encoder-decoder representation. There you have a fully visible attention in the encoder and then causal attention in the decoder. So light gray lines correspond to causal masking. And dark gray lines correspond to the fully visible masking.

In the middle we have the language model which consists of a single transformer layer stack. And it's being fed the concatenation of the inputs and the target. So it uses causal masking throughout as you can see because they're all gray lines. And you have X1 going inside, you get X2, X2 goes into the model, and you get X3 and so forth.

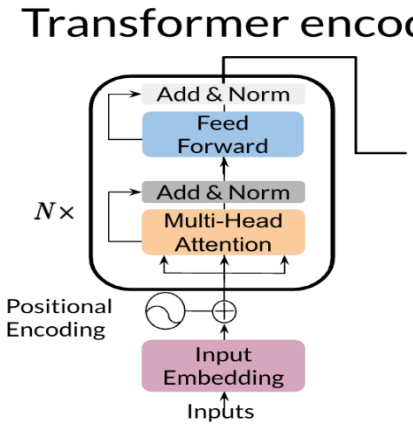
To the right, we have prefix language model which corresponds to allowing fully visible masking over the inputs as you can see with the dark arrows. And then causal masking in the rest.

Machine Learning theoretical concepts

By: Vikram Pal

Question Answering

You will be implementing an encoder this week. Last week you implemented the decoder. So here it is:



Feedforward:

```
[
      LayerNorm,
      dense,
      activation,
      dropout_middle,
      dense,
      dropout_final
]
```

Encoder block:

```
[
      Residual(
          LayerNorm,
          attention,
          dropout_,
      ),
      Residual(
          feed_forward,
      )
]
```

You can see there is a feedforward and the encoder-block above. It makes use of two residual connections, layer normalization, and dropout.

The steps you will follow to implement it are:

- Load a pre-trained model
- Process data to get the required inputs and outputs: "question: Q context: C" as input and "A" as target
- Fine tune your model on the new task and input
- Predict using your own model

Extractive Summary Using Bert:

Bert Embedding and input sequences:

it only has two labels (sentence A or sentence B), instead of multiple sentences as in extractive summarization.

Therefore, we modify the input sequence and embeddings of BERT to make it possible for extracting summaries.

Methodology

Let d denote a document containing several sentences [sent1, sent2, · · · , sent(m)], where sent(i) is the i-th sentence in the document. Extractive summarization can be defined as the task of assigning a label yi ∈ {0, 1} to each sent(i), indicating whether the sentence should be included in the summary. It is assumed that summary sentences represent the most important content of the document.

Machine Learning theoretical concepts

By: Vikram Pal

Dataset:

Both datasets contain abstractive gold summaries, which are not readily suited to training extractive summarization models.

A greedy algorithm was used to generate an oracle summary for each document. The algorithm greedily selects sentences which can maximize the ROUGE scores as the oracle sentences. We assigned label 1 to sentences selected in the oracle summary and 0 otherwise.

We evaluated on two benchmark datasets, namely the CNN/Daily Mail news highlights dataset (Hermann et al., 2015) and the New York Times Annotated Corpus

Fine-tuning with Summarization Layers

After obtaining the sentence vectors from BERT, we build several summarization-specific layers stacked on top of the BERT outputs, to capture document-level features for extracting summaries.

For each sentence sent(i), we will calculate the final predicted score Yˆ(i)
. The loss of the whole model is the Binary Classification Entropy of Yˆ(i) against gold label Y(i)
. These summarization layers are jointly fine-tuned with BERT.

Trigram Blocking:

During the predicting process, Trigram Blocking is used to reduce redundancy. Given selected summary S and a candidate sentence c, we will skip c is there exists a trigram overlapping between c and S. This is similar to the Maximal Marginal Relevance (MMR)

Learning to Rank:

What is Learning to Rank?

Learning to Rank (LTR) is a class of techniques that apply supervised machine learning (ML) to solve ranking problems. The main difference between LTR and traditional supervised ML is this:

- Traditional ML solves a prediction problem (classification or regression) on a single instance at a time. E.g., if you are doing spam detection on email, you will look at all the features associated with that email and classify it as spam or not. The aim of traditional ML is to come up with a class (spam or no-spam) or a single numerical score for that instance.
- LTR solves a ranking problem on a list of items. The aim of LTR is to come up with ***optimal ordering of those items***. As such, LTR doesn't care much about the exact score that each item gets but cares more about the relative ordering among all the items.

The training data for a LTR model consists of a list of items and a "ground truth" score for each of those items. In all techniques, **ranking is transformed into a pairwise classification or regression problem**.

RankNet:

Neural network → minimize the number of inversions in ranking → means an incorrect order among a pair of results → optimizes the cost function using Stochastic Gradient Descent.

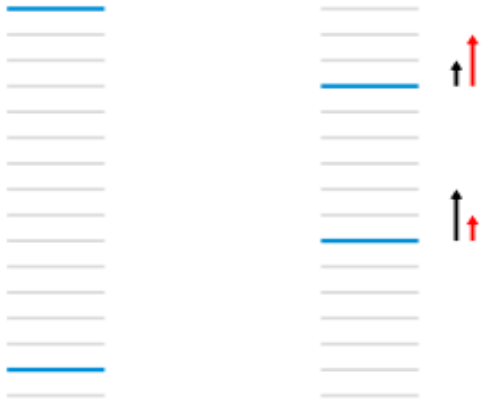
Machine Learning theoretical concepts

By: Vikram Pal

RankNet was originally **developed using neural nets**, but the underlying model can be different and is not constrained to just neural nets. The cost function for RankNet aims to **minimize the number of inversions in ranking**. Here an inversion **means an incorrect order among a pair of results**, i.e. when we rank a lower rated result above a higher rated result in a ranked list. RankNet optimizes the cost function using Stochastic Gradient Descent.

LambdaRank:

In LambdaRank, **we don't need the costs**, only need the gradients (λ) of the cost with respect to the model score**.** We can think of these **gradients as little arrows** attached to each document in the ranked list, indicating the direction we'd like those documents to move. The core idea of LambdaRank is to use this new cost function for training a RankNet. On experimental datasets, this shows both speed and accuracy improvements over the original RankNet.



LambdaMart:

MART uses **gradient boosted decision trees** for prediction tasks

LambdaMART uses gradient boosted decision trees using a cost function derived from LambdaRank

LambdaMART combines LambdaRank and MART (Multiple Additive Regression Trees). While MART uses **gradient boosted decision trees** for prediction tasks, LambdaMART uses gradient boosted decision trees using a cost function derived from LambdaRank for solving a ranking task. On experimental datasets, LambdaMART has shown better results than LambdaRank and the original RankNet.

Machine Learning theoretical concepts

By: Vikram Pal

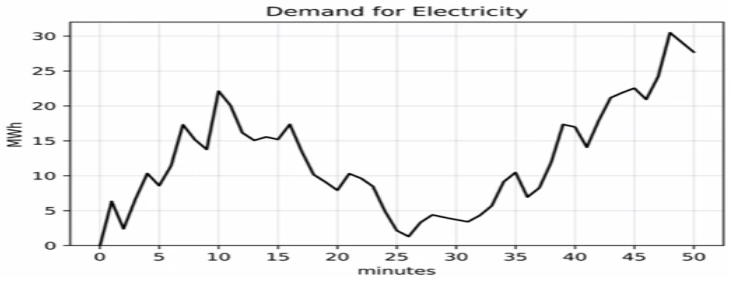
Time Series Analysis

Introduction to Time Series Analysis:

What is a Time Series?

Time Series: a sequence of data points organized in time order.

- The sequence captures data at equally spaced points in time.
- Data collected irregularly is not considered time series.



Why is Time Series Data Different?

Evaluating forecast results can be challenging

- Standard measures:
  - Forecast miss (%)
  - Error rates by horizon

  Can be misleading
- Custom measures based on business impact are often required
- Takes longer to learn outcomes

Types of Forecasting Problems

Univariate

Single data series
- Continuous data
- Binary data
- Categorical data

Multiple (un)related series

Conditional series

Panel / Multivariate

Multiple related series

Identifying groups
- Customer types
- Department / channel
- Geographic

Joint estimation across series

Machine Learning theoretical concepts

By: Vikram Pal

Time series data is common across many industries:

- **Finance**: stock prices, asset prices, macroeconomic factors
- **E-Commerce**: page views, new users, searches
- **Business**: transactions, revenue, inventory levels

Motivations for Using Time Series

Time series methods are used to:

- Understand the processes driving observed data
- Fit models to monitor or forecast a process
- Understand what influences future results of various series
- Anticipate events that require management intervention

Applications of Time Series Modeling

Time Series Components

A time series can be decomposed into several components:

- **Trend** – long term direction
- **Seasonality** – periodic behavior
- **Residual** – irregular fluctuations

Generally, models perform better if we can first remove **known sources** of variation

Machine Learning theoretical concepts

By: Vikram Pal

Trend

Trend captures the general direction of the time series.

- For example: increasing job growth year over year despite seasonal fluctuations

Trend

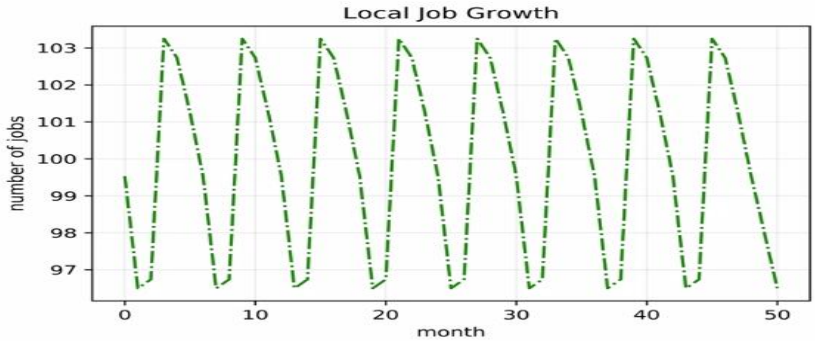
Seasonality

Seasonality captures effects that occur with specific frequency.

It can be driven by many factors:

- Naturally occurring events like weather fluctuations caused by time of year
- Business or administrative procedures like the start and end of a school year
- Social or cultural behavior like holidays or religious observances
- Fluctuations due to calendar events:
  - Number of Mondays per month
  - Holidays that shift from year to year (Easter, Chinese New Year)

Seasonality



Machine Learning theoretical concepts

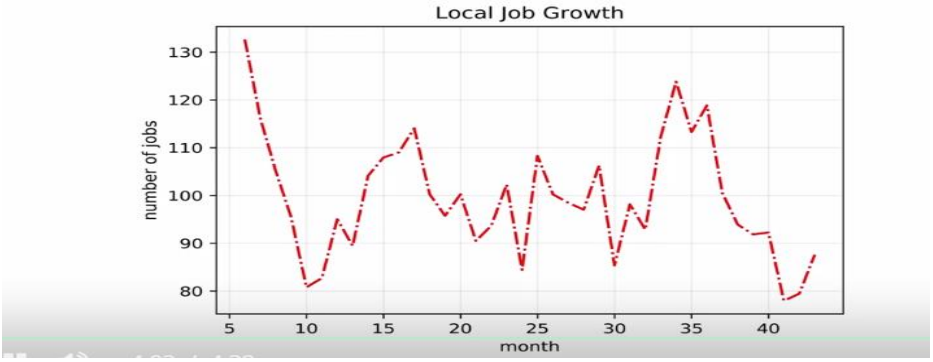
By: Vikram Pal

Residuals

Residuals are the random fluctuations leftover once trend and seasonality are removed.

- They are what is left after trend and seasonality are removed from the original time series.
  - There should not be a trend or seasonal pattern in residuals.
- They represent short term fluctuations and may be random.
- There may be a portion of the trend or seasonality components missed in the decomposition.

Residuals



Decomposition Models

Time series components can be decomposed with the following models:

- **Additive** decomposition
- **Multiplicative** decomposition
- **Pseudo-additive** decomposition

Additive Model

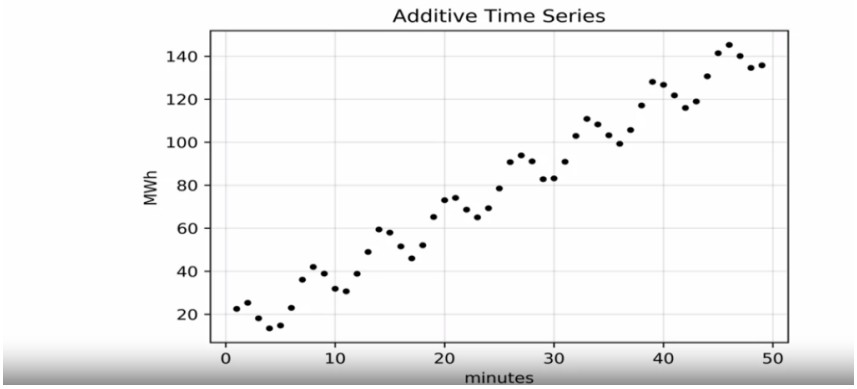
Additive models assume the observed time series is the sum of its components.

- **Observation = Trend + Seasonality + Residual**
- Additive models are used when the magnitudes of the seasonal and residual values are independent of trend.

Machine Learning theoretical concepts

By: Vikram Pal

Additive Model

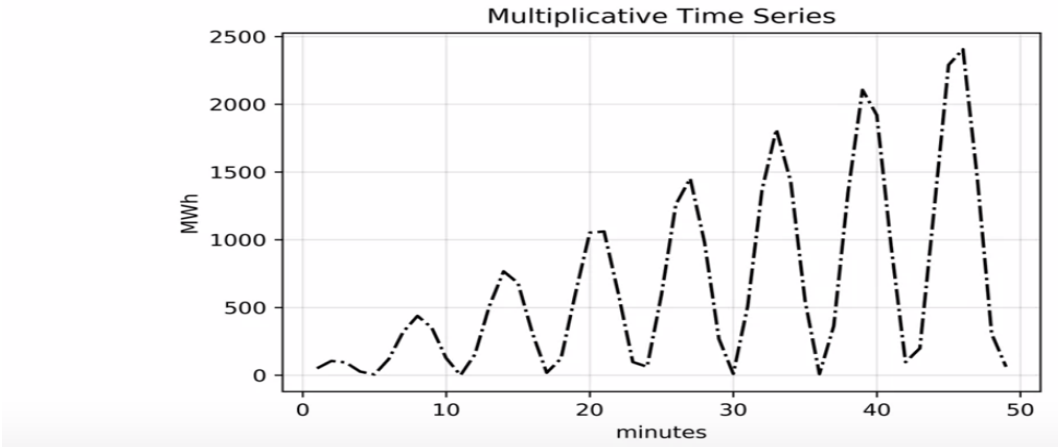


Multiplicative Model

Multiplicative models assume the observed time series is the product of its components.

- **Observation = Trend * Seasonality * Residual**
- A multiplicative model can be transformed to an additive by applying a log transformation:
  - **log(Time*Seasonality*Residual) = log(Time) + log(Seasonality) + log(Residual)**
- These are used if the magnitudes of the seasonal and residual values fluctuate with trend.

Multiplicative Model



Machine Learning theoretical concepts

By: Vikram Pal

Pseudo-Additive Model

Pseudo-additive models combines elements of the additive and multiplicative models.

They can be useful when:

- Time series values are close to or equal to zero.
- We expect features related to a multiplicative model.
- Division by zero often becomes a problem when this is the case.
- The fix: $O_t = T_t + T_t(S_t - 1) + T_t(R_t - 1) = T_t(S_t + R_t - 1)$

How to Decompose Time Series

There are many ways to decompose a time series. Some well-known approaches:

- Single, double, or triple exponential smoothing
- Locally Estimated Scatterplot Smoothing (LOESS)
- Frequency-based methods

Stationarity

Stationarity impacts our ability to model and forecast

- A **stationary** series has the same mean and variance over time
- **Non-stationary** series are much harder to model

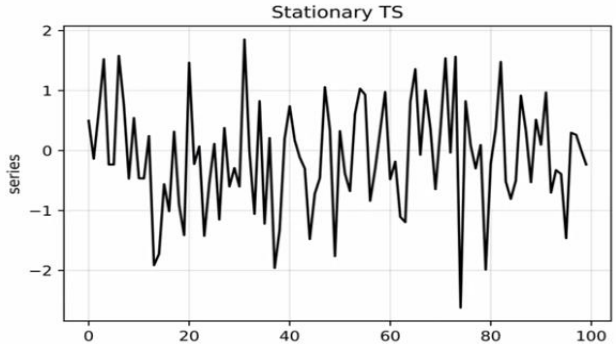
Common approach:

- Identify sources of non-stationarity

Assessing Stationarity

There are four key properties that a time series must exhibit over time for it to be **stationary**:

- Constant mean
- Constant variance
- Constant autocorrelation structure
- No periodic component



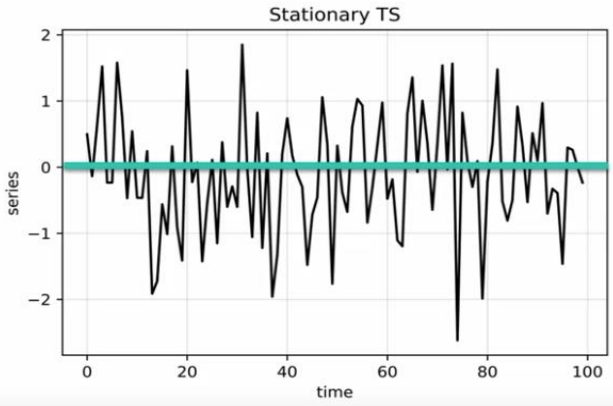
Machine Learning theoretical concepts

By: Vikram Pal

Assumption 1: Constant Mean

A stationary time series has **constant mean** throughout the entire series.

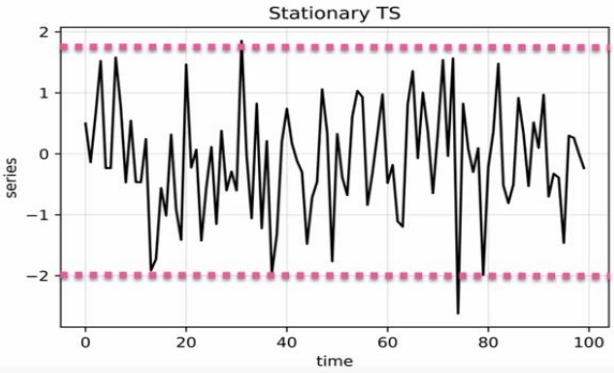
- Example: the series plotted on the right has constant mean throughout.



Assumption 2: Constant Variance

A stationary time series has **constant variance** throughout the entire series.

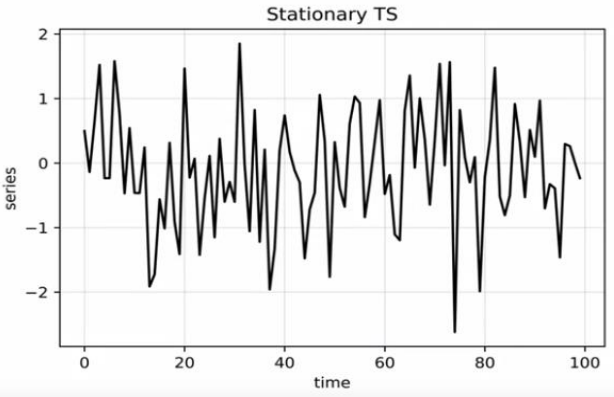
- Example: the series on the right has roughly constant variance throughout.



Assumption 3: Constant Autocorrelation

A stationary time series has **constant autocorrelation structure** throughout the entire series.

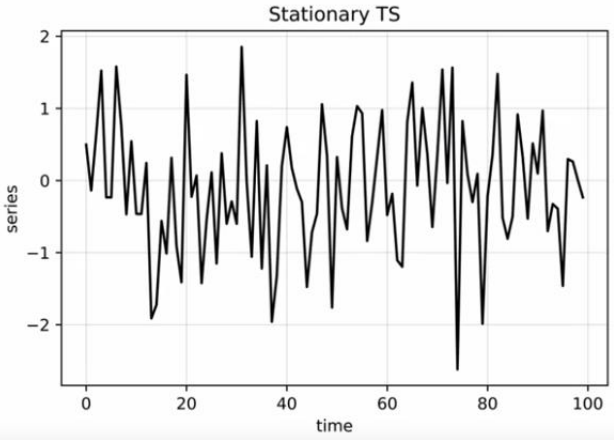
- Autocorrelation is a key concept in time series.
- It means today's measurement is highly dependent on some past value.
- The time interval between correlated values is called a lag.
- Example: stock prices may be correlated from one day to the next (lag=1).



Assumption 4: No Periodic Component

A stationary time series has **no periodic** throughout the entire series.

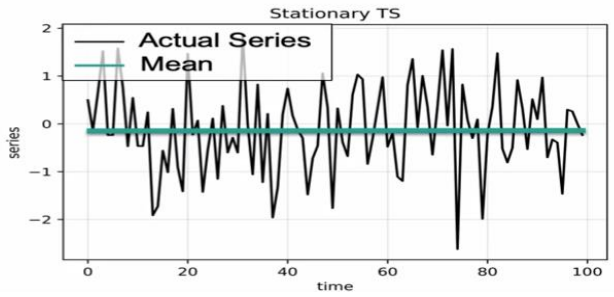
- Example: the plot on the right shows no seasonality or periodicity



Forecasting with Simple Average

A solution is to calculate the mean of the series and predict that value into the future.

- Looks quite reasonable in this case.
- However, we should be more rigorous and calculate how far off our estimate is from reality.
- Next is a quick detour about Mean Squared Error (MSE).



Moving Average

Moving Average smoothing techniques allow us to avoid sensitivity to local fluctuations.

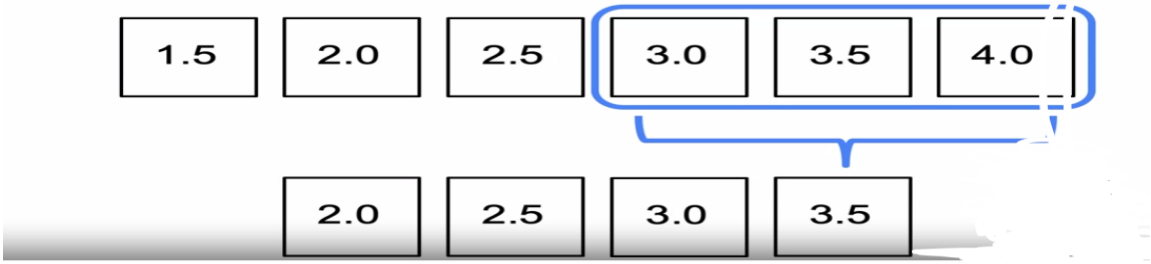
- Two primary approaches to Moving Average calculation:
  - Equally weighted
  - Exponentially weighted

Machine Learning theoretical concepts

By: Vikram Pal

Equally-Weighted Moving Average

We continue this process until we reach the end.

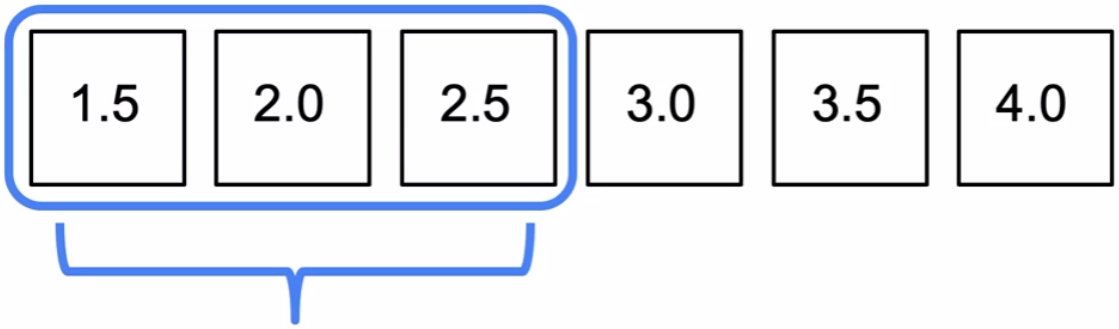


Exponential-Weights

There are many ways to create exponential weights.
To keep things simple we'll leverage this simple formula:

- $w + w^2 + w^3 = 1$
- $w = w_{t-1} \sim 0.543$
- $w^2 = w_{t-2} \sim 0.294$
- $w^3 = w_{t-3} \sim 0.160$

We slide the window over the first 3 values and calculate the mean, but differently.



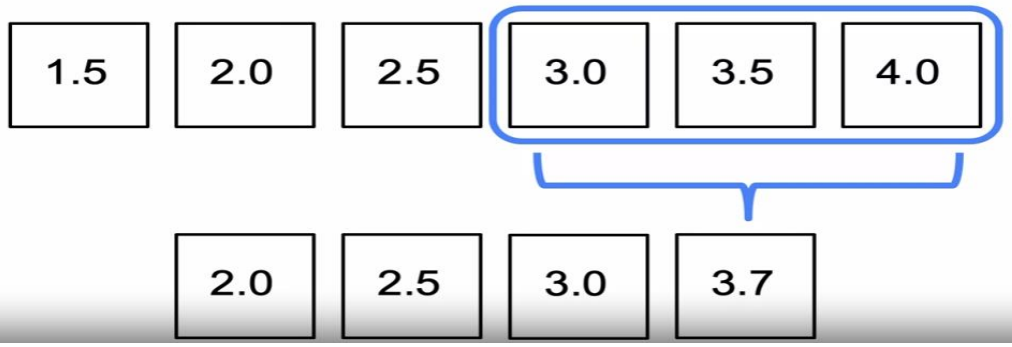
$(w_{t-3} \text{ x } 1.5) + (w_{t-2} \text{ x } 2.0) + (w_{t-1} \text{x } 2.5) = 2.2$

Machine Learning theoretical concepts

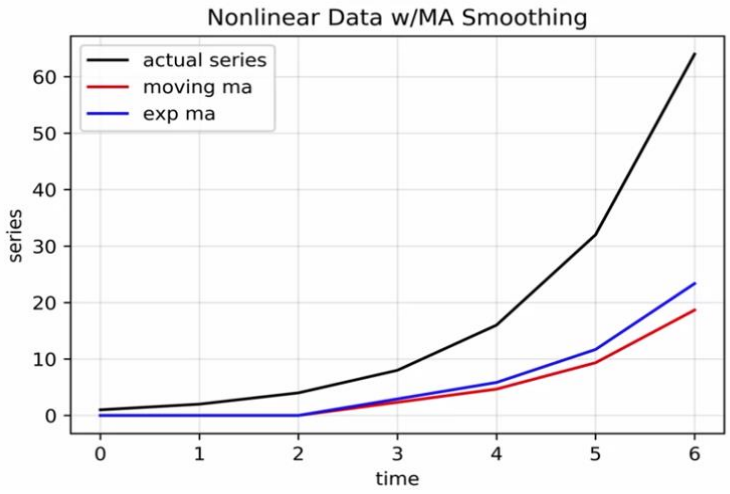
By: Vikram Pal

Exponentially-Weighted Moving Average

We continue this process until we reach the end.



Exponentially-Weighted Moving Average



IBM

Machine Learning theoretical concepts

By: Vikram Pal

What is Smoothing?

Smoothing is a process that often improves our ability to forecast series by reducing the impact of noise.

There are many ways to smooth data. Some examples:

- Simple average smoothing
- Equally weighted moving average
- Exponentially weighted moving average

Single Exponential Formulation

What we have been examining so far is exponential weighted average smoothing. This is also known as **Single Exponential Smoothing**, and has the formula:

$$\hat{S_t} = \alpha * X_t + (1-\alpha) * \hat{S_{t-1}} + (1-\alpha)^2 * \hat{S_{t-2}} + \ldots$$

$$t \geq 3$$

$\hat{S_t}$ = Smoothed value at time t

X_t = Actual value at time t

α = Parameter optimized to fit past data

Double Exponential Smoothing

Double Exponential Smoothing has the ability to pick up trend.
It does this by adding a second component into its formulation that smooths out trend.

$$S_t = \alpha * X_t + (1-\alpha) * (S_{t-1} + b_{t-1})$$ **Smooths the value of the series**

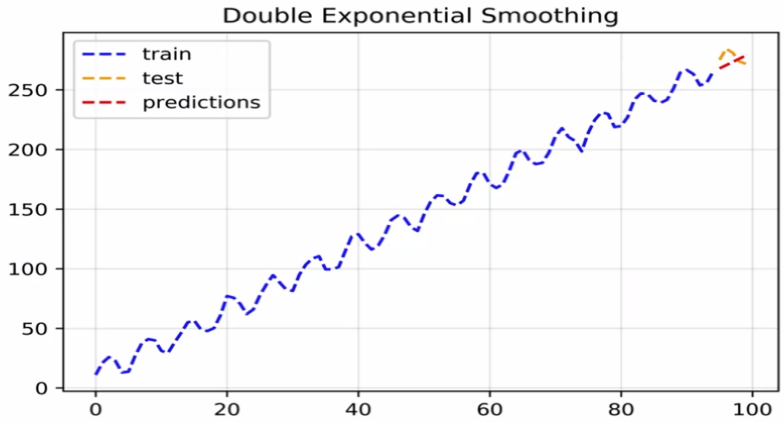
$$b_t = \beta * (S_t - S_{t-1}) + (1-\beta) * b_{t-1}$$ **Smooths the trend of the series**

$$\hat{X_{t+1}} = S_t + b_t$$ **Future prediction of series = sum of value and trend**

Machine Learning theoretical concepts

By: Vikram Pal

Double Exponential Smoothing



MSE = 354

IBM

Triple Exponential Smoothing

Triple Exponential Smoothing has the ability to pickup trend and seasonality.
It does this by adding a third component to its formulation that smooths out seasonality.

$$S_t = \alpha * (X_t - c_{t-L}) + (1 - \alpha) * (S_{t-1} + b_{t-1})$$ Smooths the value

$$b_t = \beta * (S_t - S_{t-1}) + (1 - \beta) * b_{t-1}$$ Smooths the trend

$$c_t = \gamma * (X_t - S_{t-1} - b_{t-1}) + (1 - \gamma) * c_{t-L}$$ Smooths the seasonality

$$(\hat{X})_{t+m} = (S_t + m * b_t) * c_{t-L+(m-1)} \mod L$$

$L =$ Length of time of the seasonality

IBM

Triple Exponential Smoothing has the ability to pickup trend and seasonality.
It does this by adding a third component to its formulation that smooths out seasonality.

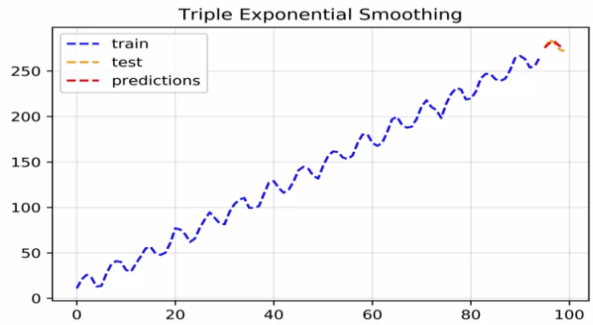
$$(\hat{X})_{t+m} = (S_t + m * b_t) * c_{t-L+(m-1)} \mod L$$

$L =$ Length of time of the seasonality

Machine Learning theoretical concepts

By: Vikram Pal

Triple Exponential Smoothing



MSE = 50

IBM

A comparison of MSE shows just how significant an impact using the best modeling strategy has on a forecast.

Single Exponential	830
Double Exponential	354
Triple Exponential	50

ARMA Model

ARMA models combine two models:

- The first is an autoregressive (AR) model. Autoregressive models anticipate series' dependence on **its own past values**.
- The second is a moving average (MA) model. Moving average models anticipate series' dependence on **past forecast errors**.
- The combination (ARMA) is also known as the **Box - Jenkins approach**.

ARMA Model: Autoregressive (AR) part

ARMA models are often expressed using orders ***p*** and ***q*** for the ***AR*** and ***MA*** components.
For a time series variable *X* that we want to predict for time *t*, the last few observations are:

$$X_{t-3}, X_{t-2}, X_{t-1}$$

AR(p) models are assumed to depend on the ***last p values*** of the time series. For ***p***=2, the forecast has the form:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \omega_t$$

Here, ω_t is the ***forecast error***; ϕ_1 and ϕ_2 are the (***p***=2) parameters (estimated by regression).

MA(q) models are assumed to depend on the ***last q values*** of the forecast error. For ***q***=2, the forecast has the form:

$$X_t = \theta_2 \omega_{t-2} + \theta_1 \omega_{t-1} + \omega_t$$

ω_t is the forecast error, ω_{t-1} is the previous forecast error, etc. θ_1 and θ_2 are the (***q***=2) parameters.

IBM

Combining the ***AR(p)*** and ***MA(q)*** models yields the ***ARMA(p, q)*** model. For ***p***=2, ***q***=2, the ARMA(2, 2) forecast has the form:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \theta_2 \omega_{t-2} + \theta_1 \omega_{t-1} + \omega_t$$

ω_t is the forecast error, ϕ_1, ϕ_2, θ_1, and θ_2 are the ($p + q = 4$) parameters.

ARMA Model Notes

There are some things to keep in mind when dealing with ARMA models:

- The time series is assumed to be stationary.
- A good rule of thumb is to have at least 100 observations when fitting an ARMA model.

There are three stages in building an ARMA model: identification, estimation, and evaluation.

Determine Seasonality

We can determine if seasonality is present by using:

- Autocorrelation and Partial Autocorrelation Plots
- Seasonal Subseries Plot
- Intuition (possible in some cases, i.e. seasonal sales of consumer products, holidays, etc.)

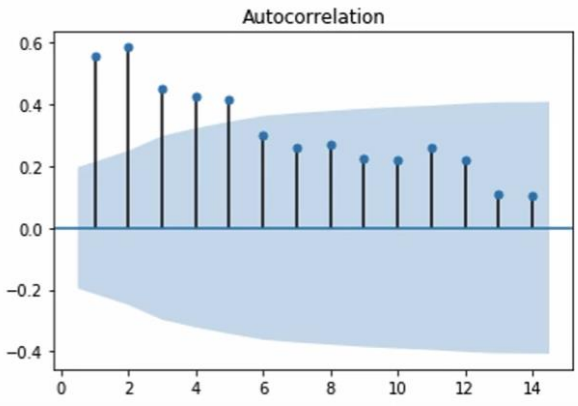
These plots are a common initial step to help us understand what type of seasonal patterns impact our data, using visual representation.

Autocorrelation Plot

An Autocorrelation Plot is commonly used to detect dependence on prior observations.

It summarizes total (2-way) correlation between the variable and its past values.

In this example, the last five observations have correlations that are statistically significant.

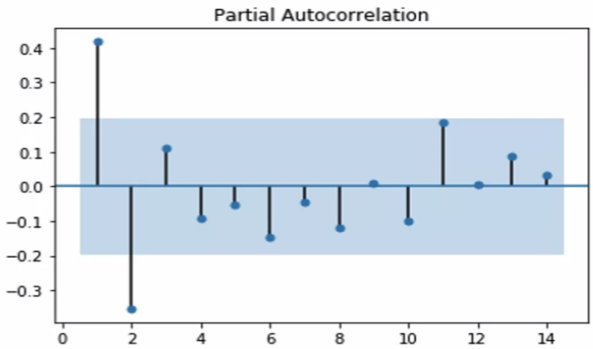


Partial Autocorrelation Plot

The Partial Autocorrelation Plot also summarizes dependence on past observations.

However, it measures partial results (including all lags)

In this example, when we include all past values, only the two most recent values appear significant.



Machine Learning theoretical concepts

By: Vikram Pal

Identifying *p* and *q*

Once we have a stationary series, we can estimate AR and MA models. We need to determine ***p*** and ***q***, the order of the AR and MA models.

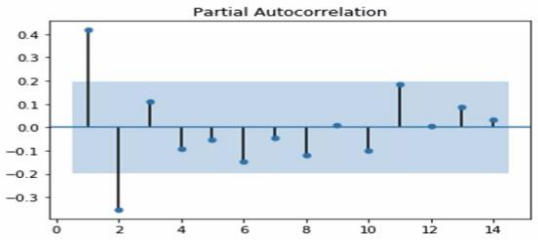
One approach here is to look at the:
- Autocorrelation Plot
- Partial Autocorrelation Plot

Another approach is to treat ***p*** and ***q*** as hyperparameters and apply standard approaches (grid search, cross validation, etc.)

AR(p)

How do we determine the order p of the AR model?

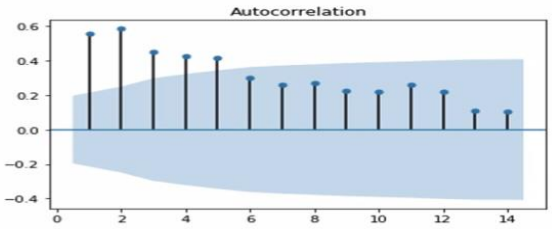
- Plot confidence intervals on the Partial Autocorrelation Plot.
- Choose lag ***p*** such that partial autocorrelation becomes insignificant for ***p + 1*** and beyond.



MA(q)

How can we determine the order ***q*** of the MA model?

- Plot confidence intervals on the Autocorrelation Plot
- Choose lag ***q*** such that autocorrelation becomes insignificant for ***q + 1*** and beyond.



Guidelines for Choosing AR/MA Models

SHAPE	MODEL
Exponential Decaying to zero	AR models
Alternating positive and negative decaying to zero	AR models
One or more spikes, the rest are close to zero	MA model
Decay after a few lags	Mixed AR and MA
All zero or close to zero	Data is random
High values at fixed intervals	Include seasonal AR term
No decay to zero	Series is not stationary

ARMA Model Estimation

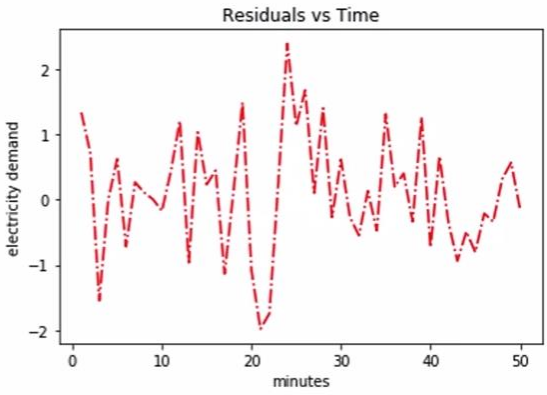
Estimating the parameters of an ARMA model can be a complicated non-linear problem.

- Non-linear least squares and Maximum Likelihood Estimation (MLE) are common approaches.
- Most statistical software will fit the ARMA model, and potentially help choose the order.

ARMA Model Validation

How do we know if your ARMA model is any good?

- The residuals will approximate a Gaussian distribution (aka white noise).
- Otherwise, we need to iterate to obtain a better model.



*** Note:

- Moving average smoothing is useful for estimating trend and seasonality of past data. MA models, on the other hand, are a useful forecasting model that regresses on past forecast errors in order to forecast future values.

- The moving-average model is conceptually a linear regression of the current value of the series against current and previous (unobserved) white noise error terms or random shocks. The random shocks at each point are assumed to be mutually independent and to come from the same distribution, typically a normal distribution, with location at zero and constant scale.

A p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis (stationary), otherwise a p-value above the threshold suggests we fail to reject the null hypothesis (non-stationary).

ARIMA Model Details

There are a few things you should know about ARIMA models:

- The ARIMA model is denoted **ARIMA (p, d, q)**.
- **p** is the order of the AR model.
- **d** is the number of times to difference the data.
- **q** is the order of the MA model.
- **p, d, q** are nonnegative integers.

Differencing

Differencing nonstationary time series data one or more times can make it stationary. That is the integrated (I) component of ARIMA.

- d is the number of times to perform a lag-1 difference on the data.
- d=0: no differencing
- d=1: difference once
- d=2: difference twice

$Y_i = Z_i - Z_{i-1}$ ⟶ (d=1)

SARIMA Model

SARIMA is short for **Seasonal ARIMA**, an extension of ARIMA models to address seasonality. This model is used to remove seasonal components.

- The SARIMA model is denoted **SARIMA (p, d, q) (P, D, Q)**.
- **P, D, Q** represent the same as p, d, q but they are applied across a season (e.g. yearly).
- **M** = one season

$Y_i = Z_i - Z_{i-M}$

Machine Learning theoretical concepts

By: Vikram Pal

Choosing ARIMA/SARIMA Parameters

How do you choose p, d, q and P, D, Q?

- Visually inspect a run sequence plot for trend and seasonality.
- Generate an ACF Plot.
- Generate a PACF Plot.
- Treat as hyperparameters (cross validate).
- Examine ***information criteria*** (*AIC*, *BIC*) which penalize the number of parameters the model uses.

ARIMA Summary

Overview of ARIMA models:

- Flexible family of models that capture autocorrelation.
- Based on strong statistical foundation.
- Requires stationary time series.
- Choosing optimal parameters manually requires care.
- Some software will automatically find parameters.
- Can be challenging to explain and interpret.
- Can be prone to overfitting.

SARIMA Assumptions

It is useful to keep the following ARMA, ARIMA, SARIMA assumptions in mind:

- Time series models require data that is stationary.
- If nonstationary, remove: trend, seasonality, apply differencing, and so on.
- Stationary data has no trend, seasonality, constant mean, and constant variance.
- The past is assumed to represent what will happen in the future, in a probabilistic sense.

Some rules to highlight from the Duke ARIMA Guide:

1. If the series has positive autocorrelations out to a high number of lags, then it probably needs a higher order of differencing
2. If the lag-1 autocorrelation is zero or negative, or the autocorrelations are all small and patternless, then the series does not need a higher order of differencing. If the lag-1 autocorrelation is -0.5 or more negative, the series may be overdifferenced. BEWARE OF OVERDIFFERENCING!!
3. A model with no orders of differencing assumes that the original series is stationary (mean-reverting). A model with one order of differencing assumes that the original series has a constant average trend (e.g. a random walk or SES-type model, with or without growth). A

model with two orders of total differencing assumes that the original series has a time-varying trend (e.g. a random trend or LES-type model)

Box-Jenkins Method

ACF Shape	Indicated Model
Exponential, decaying to zero	Autoregressive model. Use the partial autocorrelation plot to identify the order of the autoregressive model.
Alternating positive and negative, decaying to zero	Autoregressive model. Use the partial autocorrelation plot to help identify the order.
One or more spikes, rest are essentially zero	Moving average model, order identified by where plot becomes zero.
Decay, starting after a few lags	Mixed autoregressive and moving average (ARMA) model.
All zero or close to zero	Data are essentially random.
High values at fixed intervals	Include seasonal autoregressive term.
No decay to zero	Series is not stationary.

Statistical Tests:

- Normality (Jarque-Bera)
  - Null hypothesis is normally distributed residuals (good, plays well with RMSE and similar error metrics)
- Serial correlation (Ljung-Box)
  - Null hypothesis is no serial correlation in residuals (independent of each other)
- Heteroskedasticity
  - Tests for change in variance between residuals.
  - The null hypothesis is no heteroskedasticity. That means different things depending on which alternative is selected:
    - Increasing: Null hypothesis is that the variance is not increasing throughout the sample; that the sum-of-squares in the later subsample is not greater than the sum-of-squares in the earlier subsample.
    - Decreasing: Null hypothesis is that the variance is not decreasing throughout the sample; that the sum-of-squares in the earlier subsample is not greater than the sum-of-squares in the later subsample.
    - Two-sided (default): Null hypothesis is that the variance is not changing throughout the sample. Both that the sum-of-squares in the earlier subsample is not greater than the sum-of-squares in the later subsample and that the sum-of-squares in the later subsample is not greater than the sum-of-squares in the earlier subsample.
- Durbin Watson
  - Tests autocorrelation of residuals: we want between 1-3, 2 is ideal (no serial correlation)

Machine Learning theoretical concepts

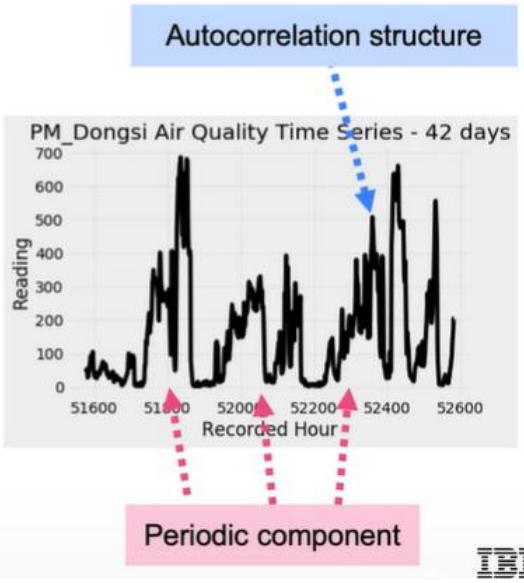
By: Vikram Pal

Deep Learning Time Series Analysis:

Why Deep Learning?

Neural networks offer several benefits over traditional time series forecasting models:

- Automatically learn how to incorporate series characteristics like trend, seasonality, and autocorrelation into predictions.
- Able to capture very complex patterns.
- Can simultaneously model many related series instead of treating each separately.



IBM

Why Not Deep Learning?

Neural network benefits don't come for free:

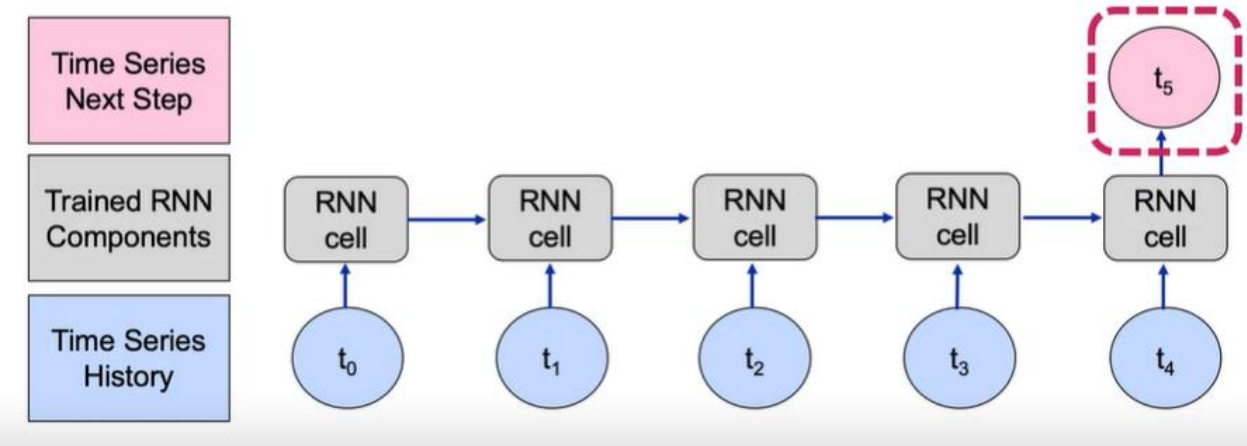
- Models can be complicated to build, computationally expensive to build (GPUs can help).
- Deep Learning models often overfit.
- It is very challenging to explain / interpret predictions made by the model ("black box").
- Tend to perform best with large training datasets.

Layer (type)	Output Shape	Param #
lstm_14 (LSTM)	(None, 70)	20160
dense_53 (Dense)	(None, 1)	71

Total params: 20,231
Trainable params: 20,231
Non-trainable params: 0

IBM

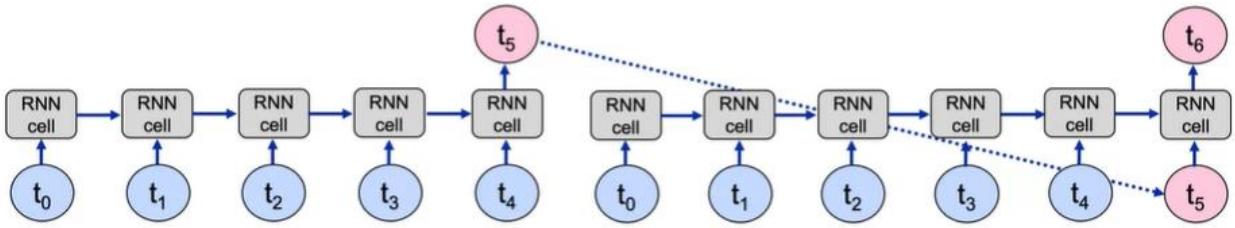
Applying RNN to Time Series Forecasting



Time Series Many-To-One RNN

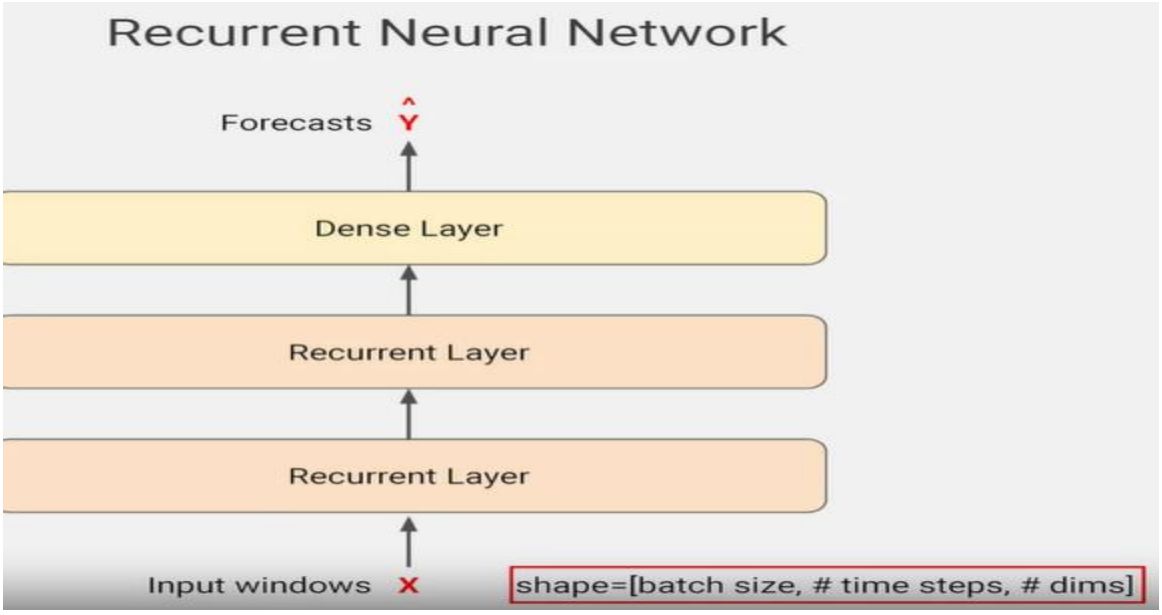
Use past time steps to forecast future time steps.

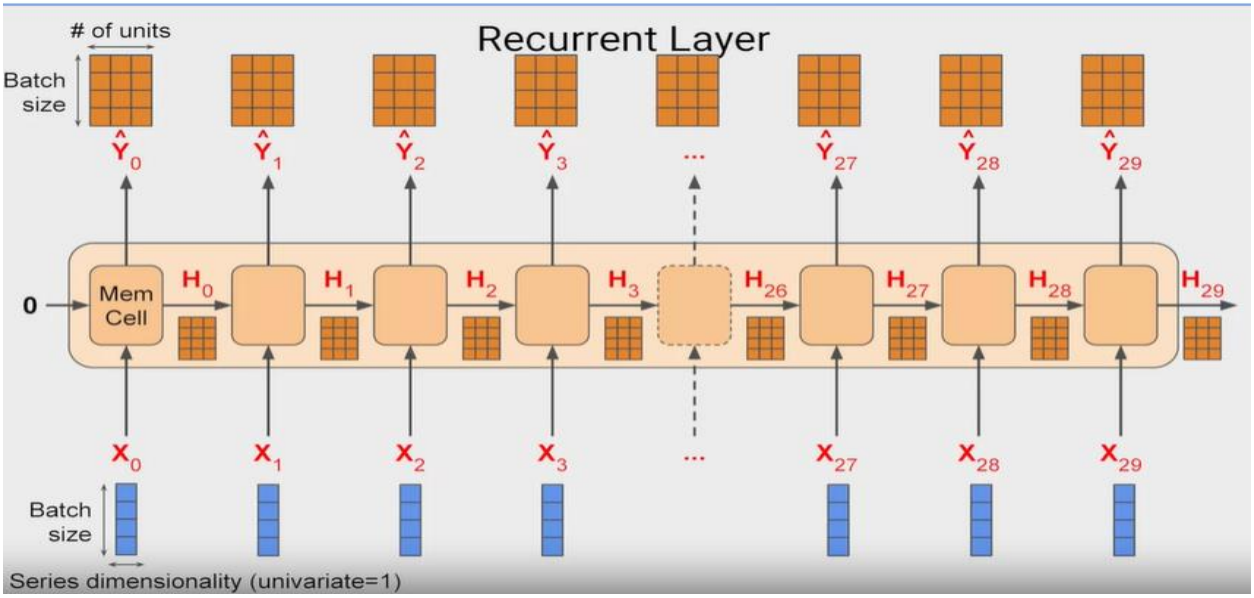
- Input: time series' historical steps.
- Output: time series' next step.
- Can forecast multiple time steps by adding previous predicted step to input sequence.



Machine Learning theoretical concepts

By: Vikram Pal





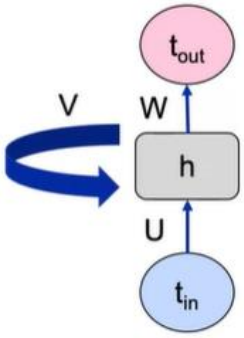
Machine Learning theoretical concepts

By: Vikram Pal

What's Going on Under the Hood?

How do we obtain the weight matrices ***U, V***, and *W*?

- When we train an RNN, we are actually finding weights via the **backpropagation algorithm.**
- In backpropagation, we repeatedly process the training data, updating the weights in order to minimize a **cost function.**
- For time series forecasting, a typical cost function would be **mean squared error** or a similar metric.
- Intuitively, we find values for *U*, *V*, and *W* that cause our predicted outputs t_{out} to be as close to the true target values as possible.



Survival Analysis:

Survival Analysis focuses on estimating the length of time until an event occurs.

Examples:

- How long will a customer remain before churning
- How long until equipment will need repairs

Standard regression-based approaches do not work due to the issue of Censoring.

Censoring: Customer Churn Example

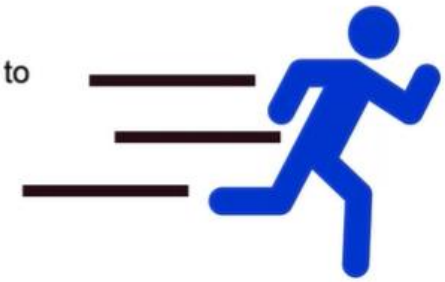
Customer churn occurs when a customer leaves a company.

If we try to estimate the churn risk for each group of customers, we only know outcomes for customers who have already churned.

For customers still with the company, we do not know their total time until churn, but we can use the fact they have not left yet.

If we exclude these customers, we restrict attention only to churned customers, and our estimates will be biased.

This bias is due to censored data (data where we do not observe churn).



Survival Function

The Survival Function: measures the probability that a subject will survive past time t.

$$S(t) = \mathrm{P}(T > t)$$

This function:

- Is decreasing (non-increasing) over time.
- Starts at 1. (when t=0)
- Ends at 0. (for high enough t)

Hazard Rate

The **Hazard Rate** represents the instantaneous rate at which events occur, given that it has not occurred already.

$$h(t) = \frac{f(t)}{S(t)}$$

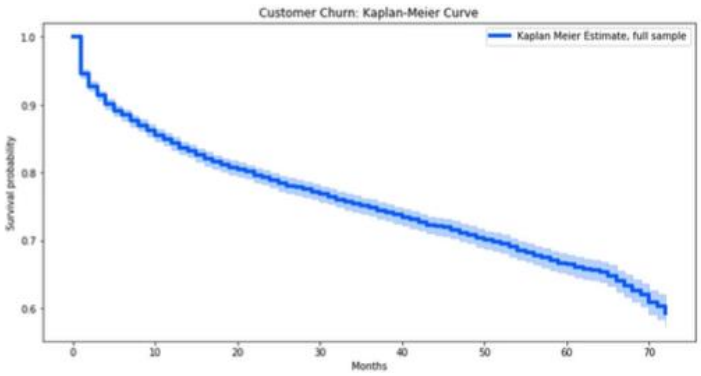
The cumulative hazard rate (sum of $h(t)$ from $t = 0$ to $t = t$) represents accumulated risk over time.

Kaplan-Meier Curve

The Kaplan-Meier estimator is a non-parametric estimator.

It allows us to use observed data to estimate the survival distribution.

The **Kaplan-Meier Curve** plots the (cumulative) probability of survival beyond each given time period.



Customer Churn Example

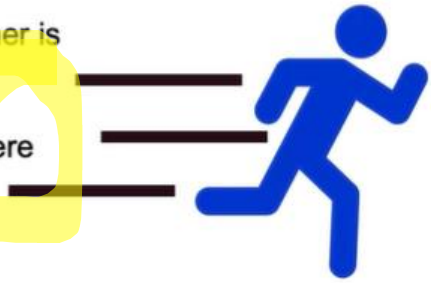
Considering our customer churn example, suppose we want to see whether certain features impact churn risk.

Using the Kaplan-Meier Curve allows us to visually inspect differences in survival rates by category.

This is similar to EDA (we are not using any model).

In our churn example, we have data on whether a customer is purchasing multiple services.

We can use Kaplan-Meier Curves to examine whether there appear to be differences based on this feature.

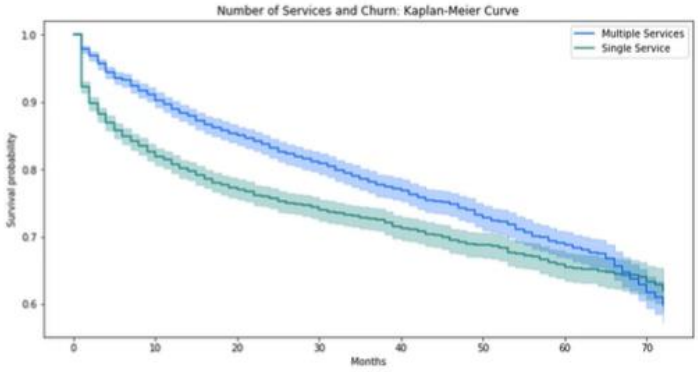


Kaplan-Meier Curve

To see whether survival rates differ based on number of services, we estimate **Kaplan-Meier** curves for each group.

We see that churn risk seems greater for those with only a single service.

This difference is most stark early on, gradually declines over time.



Survival Regression Approaches

The Kaplan-Meier approach provides sample averages. However, we may want to make use of individual-level data to predict survival rates.

Here, we turn to Survival Regression approaches, which:

- Allow us to generate estimates of total risk as a function of time
- Make use of censored and uncensored observations to predict hazard rates
- Allow us to estimate feature effects

Approaches to Modeling Hazard Rates

Some well-known Survival models for estimating Hazard Rates include:

- The Cox Proportional Hazard model
- Accelerated Failure Time (AFT) models (several variants including the Weibull AFT model)

These models differ with respect to assumptions they make about the hazard rate function, and the impact of features.

Approaches to Modeling Hazard Rates

The most common survival model is the ***Cox Proportional Hazard*** (CPH) model, which assumes features have a **constant proportional impact** on the hazard rate.

For a single non-time-varying feature *X*, the hazard rate $\boldsymbol{h(t)}$ is modeled as:

$$h(t) = \boldsymbol{\beta_0}(t)e^{\beta_1 X}$$

$\boldsymbol{\beta_0(t)}$ is the time-varying ***baseline hazard***, and $e^{\beta_1 X}$ is the (constant) proportional adjustment to the baseline hazard due to ***X***.

Approaches to Modeling Hazard Rates

Consider our Customer Churn example, with $\boldsymbol{X}$ as a binary variable that indicates whether a customer has multiple accounts. If we apply the Cox Proportional Hazard model:

$$h(t) = \boldsymbol{\beta_0}(t)e^{\boldsymbol{\beta_1} X}$$

The model will estimate the hazard rate function for customers without multiple accounts:

$$h(t; X = 0) = \boldsymbol{\beta_0}(t)$$

For customers with multiple accounts, the hazard rate becomes:

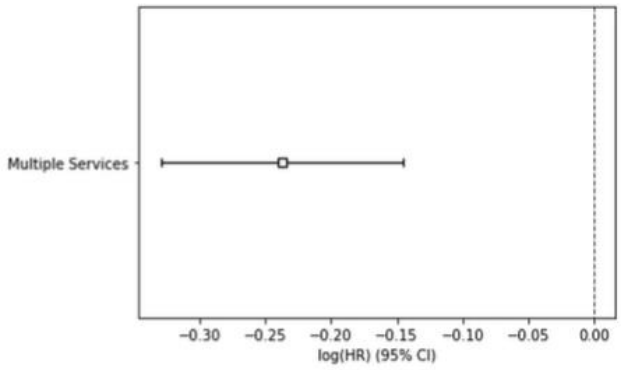
$$h(t; X = 1) = \boldsymbol{\beta_0}(t)e^{\boldsymbol{\beta_1}}$$

Customer Churn Example, CPH Model

Running the CPH model using the single variable yields an estimate of $\boldsymbol{\beta_1} = \boldsymbol{-0.24}$.

We can examine its confidence interval to assess its significance.

This suggests that having multiple services significantly improves churn risk.



Machine Learning theoretical concepts

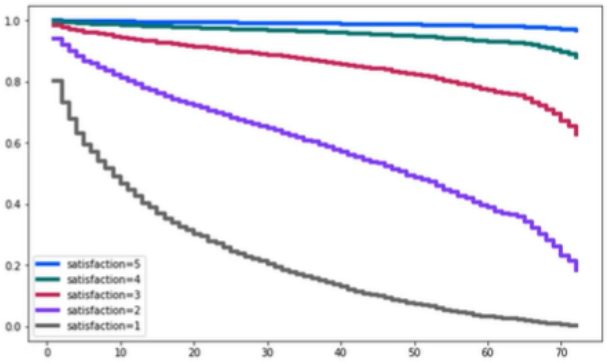
By: Vikram Pal

Customer Churn Example, CPH Model

Using the CPH model, we can plot estimated survival curves for various categories.

Using the customer satisfaction variable, we see survival likelihoods for various levels of satisfaction.

Lower satisfaction leads to substantially higher rates of predicted churn

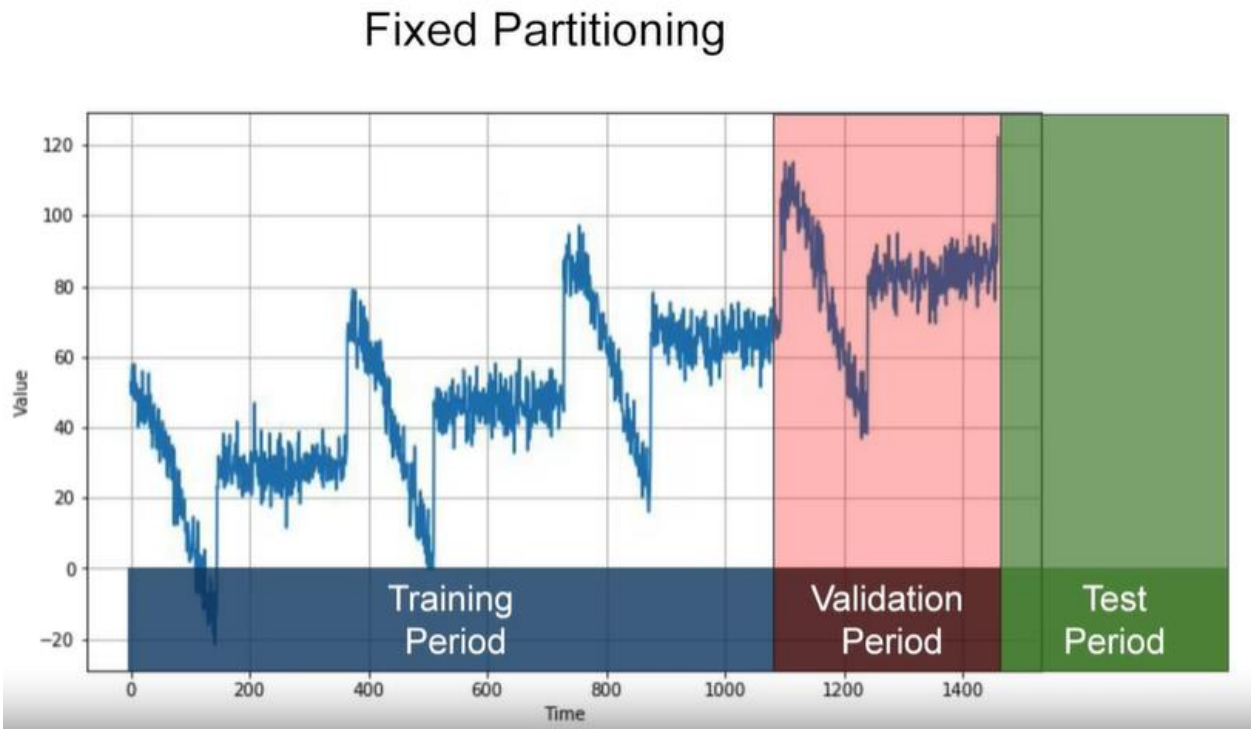


Cross validation techniques are better suited for time series data:

Forward Chaining Cross Validation: Forward-chaining cross-validation, also called rolling-origin cross-validation, is **similar to k-fold** but suited to sequential data such as time series. There is no random shuffling of data to begin but a test set may be set aside.

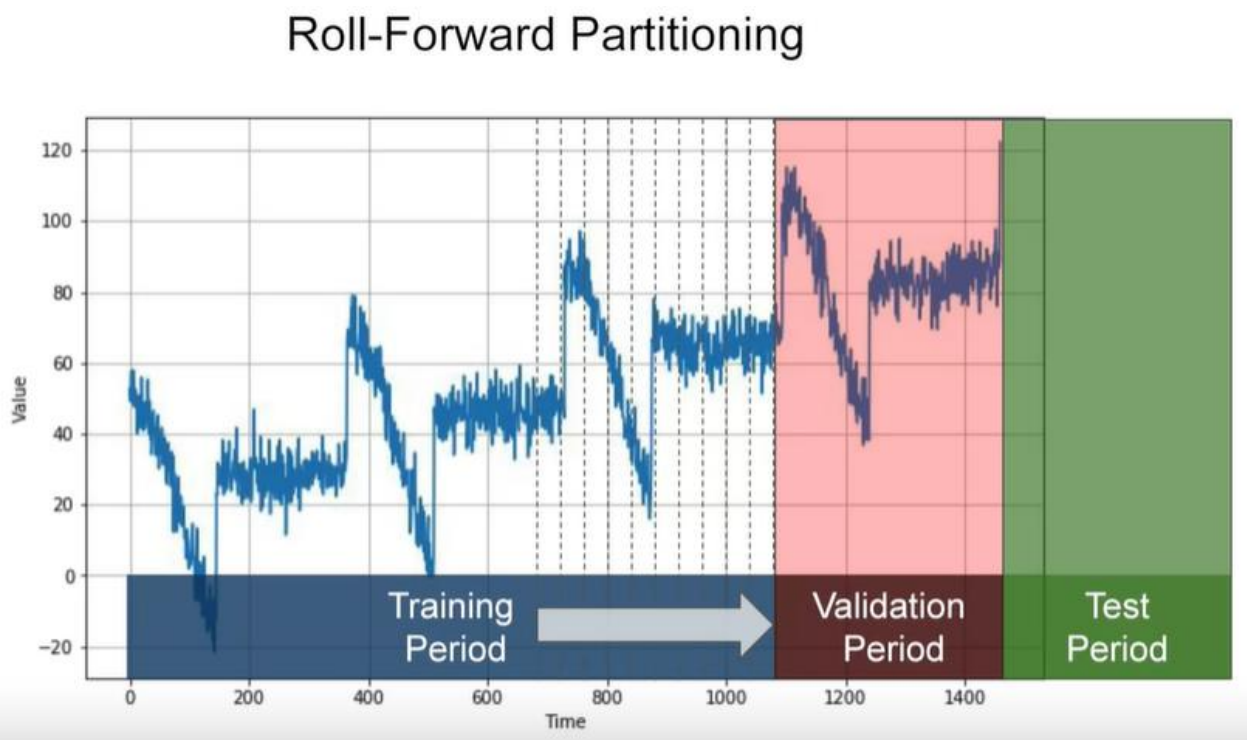
Stationary Process is a stochastic process whose unconditional joint probability distribution does not change when shifted in time.

Time-series data split:



Machine Learning theoretical concepts

By: Vikram Pal



Evaluation Metrics:

Metrics

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

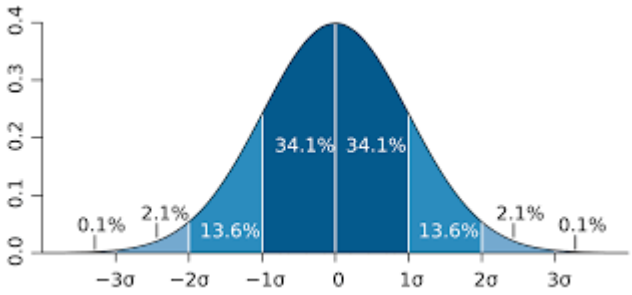
Machine Learning theoretical concepts

By: Vikram Pal

Statistics

Standard deviation & Variance:

	Population	Sample
Variance	$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}$	$S^2 = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}$
Standard deviation	$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}}$	$S = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$

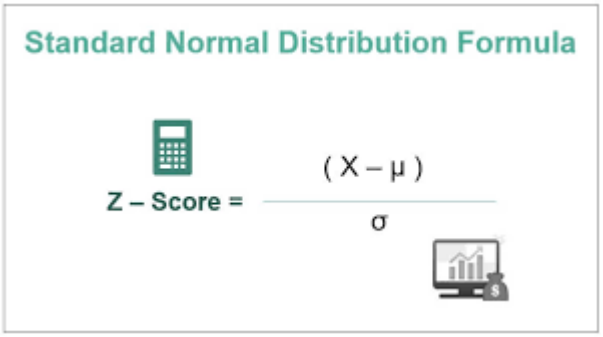


Different Type of Distribution:

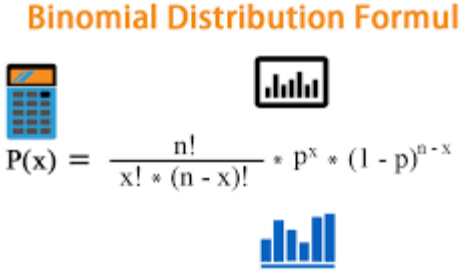
Normal distribution: It is also called Gaussian distribution. The curve of a normal distribution is called Bell-shaped curve.

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$\mu =$ Mean
$\sigma =$ Standard Deviation
$\pi \approx 3.14159\cdots$
$e \approx 2.71828\cdots$



Binomial Distribution: It is a discrete distribution. It describes the outcome of a binary scenarios. E.g toss a coin.



Machine Learning theoretical concepts

By: Vikram Pal

Difference Between Normal and Binomial Distribution: The main difference is that **normal distribution is continuous whereas binomial is discrete**, but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.

Poisson Distribution: it is a Discrete Distribution. It estimates how many times an event can happen in a specified time. e.g. If someone eats twice a day what is probability he will eat thrice?

Poisson Distribution Formula

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where
x = 0, 1, 2, 3, ...
λ = mean number of occurrences in the interval
e = Euler's constant ≈ 2.71828

Difference Between Normal and Poisson Distribution: Normal distribution is continuous whereas poison is discrete. But we can see that similar to binomial for a large enough poison distribution it will become similar to normal distribution with certain std dev and mean.

Uniform Distribution: Used to describe probability where every event has equal chances of occurring. E.g. Generation of random numbers.

Uniform Distribution Formula

cuemath THE MATH EXPERT

$$F(x) = \frac{1}{(b - a)} \qquad \text{Mean} = \frac{(a + b)}{2}$$

$$\sigma = \sqrt{\frac{(b - a)^2}{12}}$$

Logistic Distribution: It is used to describe growth.

Difference Between Logistic and Normal Distribution: Both distributions are near identical, but logistic distribution has more area under the tails. i.e. It represents more possibility of occurrence of an event further away from mean. For higher value of scale (standard deviation) the normal and logistic distributions are near identical apart from the peak.

Multinomial Distribution: It is a generalization of binomial distribution. It describes outcomes of multi-nominal scenarios unlike binomial where scenarios must be only one of two. e.g. Blood type of a population, dice roll outcome.

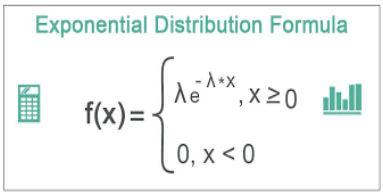
$$\mathrm{P} = \frac{n!}{(n_1!)(n_2!)\ldots(n_x!)} P_1{}^{n1} P_2{}^{n2} \ldots P_x{}^{nx}$$

Machine Learning theoretical concepts

By: Vikram Pal

Difference Between Normal distribution and Multinomial distribution: As they are generalization of binomial distribution their visual representation and similarity of normal distribution is same as that of multiple binomial distributions.

Exponential Distribution: It is used for describing time till next event e.g., failure/success etc.



Relation Between Poisson and Exponential Distribution:

- Poisson distribution deals with **number of occurrences of an event in a time period**
- Exponential distribution deals with **the time between these events.**
-

Bernoulli distribution: The Bernoulli distribution represents the **success or failure of a single Bernoulli trial.** The Binomial Distribution represents the **number of successes and failures in n independent Bernoulli trials for some given value of n**.

Formula

$$f(k;p) = pk + (1-p)(1-k)$$

p = probability
k = possible outcomes
f = probability mass function

What Is a Null Hypothesis?

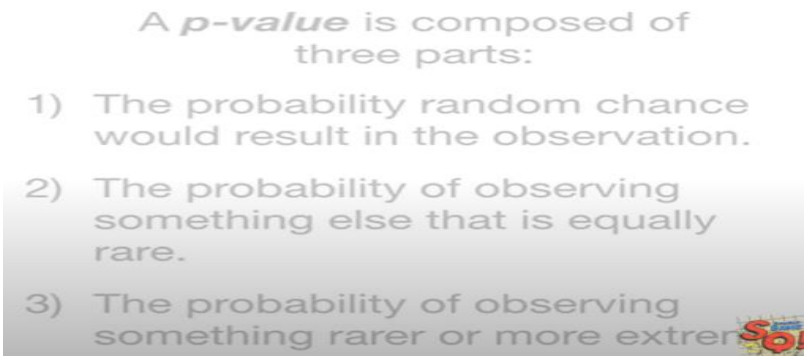
A null hypothesis is a type of hypothesis used in statistics that proposes that there is **no difference between certain characteristics of a population** **(**or data-generating process).

Hypothesis Testing for Investments:
As an example, related to financial markets, assume Alice sees that her investment strategy produces higher average returns than simply buying and holding a stock. The null hypothesis states that there is no difference between the two average returns.

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).

Machine Learning theoretical concepts

By: Vikram Pal



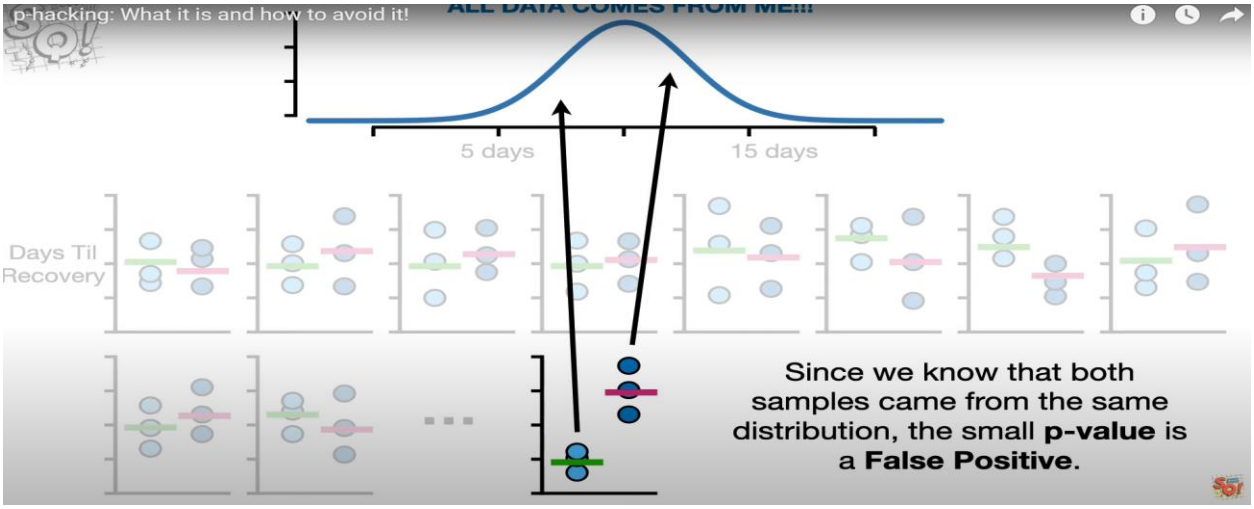
P-Value of two heads:
One coin tossed three times:
HHH, HHT, HTH, THH,TTH, TTT,THT,HTT

1. probability random change: 3/8
2. Probability of equal rare: 3/8
3. Probability of rarer or extreme rarer: 2/8

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

P-hacking:

it refers to the misuse and abuse of analysis techniques and results in being fooled by false positives.



Covariance:

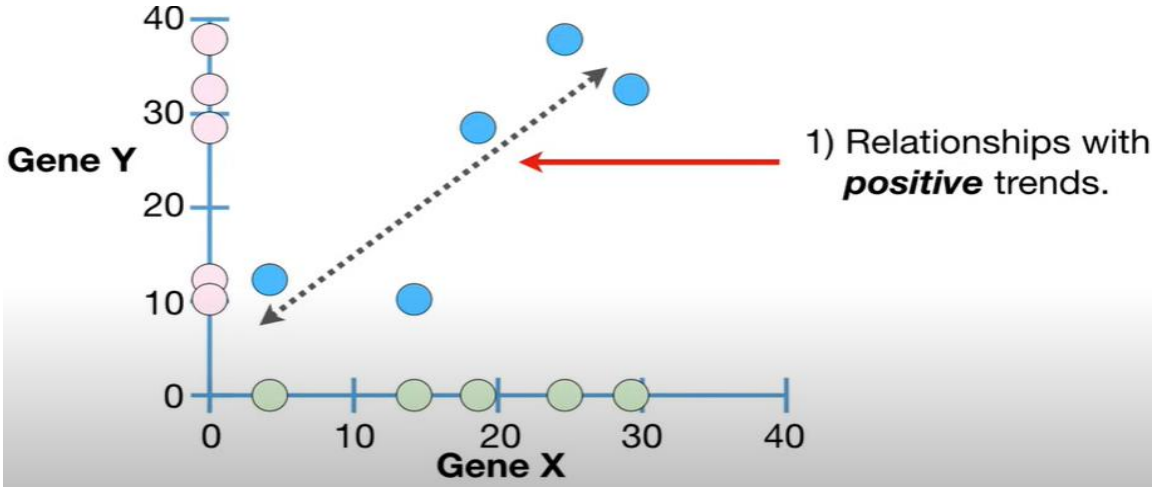
is a measure of the **relationship between two random variables**. The metric evaluates how much – to what extent – the variables change together.

$$\mathrm{Cov}\,(X, Y) = \frac{\sum(X_i - \overline{X})(Y_j - \overline{Y})}{n}$$

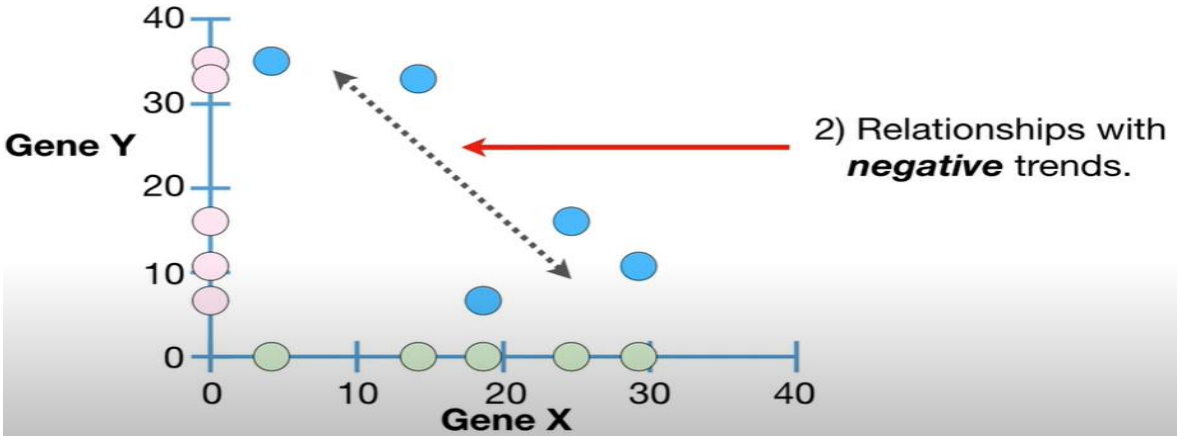
Covariance can classify three types of relationships.
1.Positive trend(slope):

Machine Learning theoretical concepts

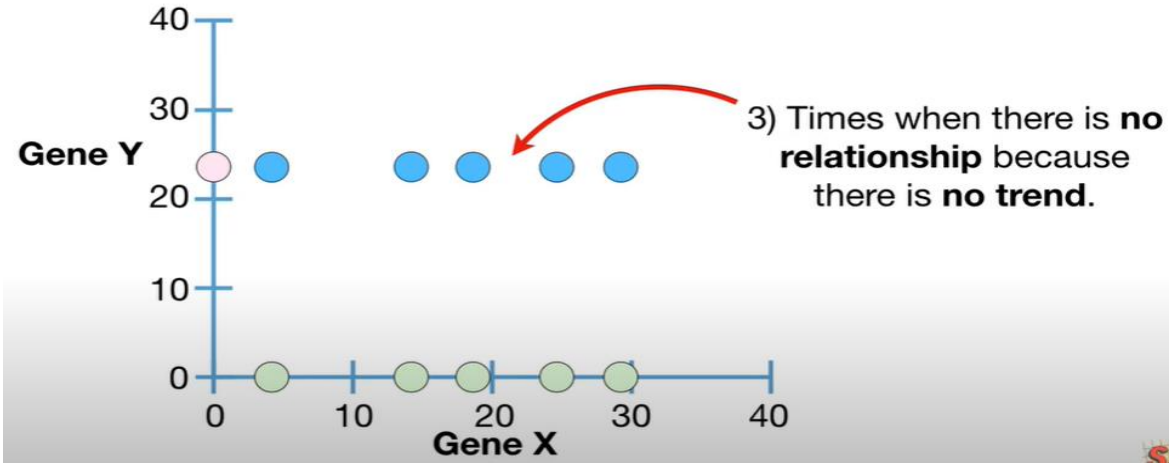
By: Vikram Pal



2.Negative Trend:



3.No relationship:



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.

Machine Learning theoretical concepts

By: Vikram Pal

$$\rho_{xy} = \frac{\mathrm{Cov}(x,y)}{\sigma_x \sigma_y}$$

where:

$\rho_{xy} = \text{Pearson product-moment correlation coefficient}$

$\mathrm{Cov}(x,y) = \text{covariance of variables } x \text{ and } y$

$\sigma_x = \text{standard deviation of } x$

$\sigma_y = \text{standard deviation of } y$

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.

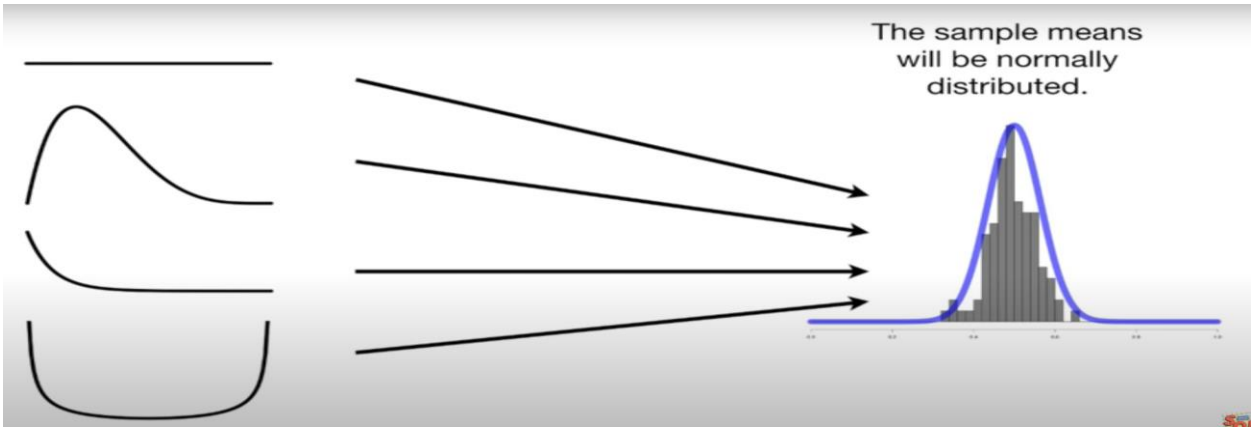
The use of a **controlled study is the most effective** way of establishing causality between variables. In a controlled study, the sample or population is split in two, with both groups being comparable in almost every way. The two groups then receive different treatments, and the outcomes of each group 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.**

e.g.: **(age and experience)**

Central limit theorem:

The Central Limit Theorem states that the sampling distribution of the sample means approaches a normal distribution as the sample size gets larger — no matter what the shape of the population distribution.



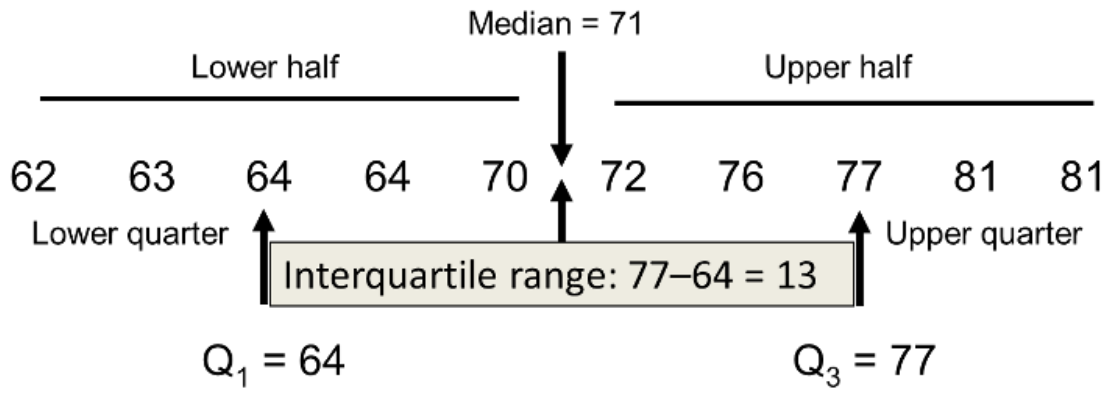
Note:The Standard deviation of means is called Standard error.

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)



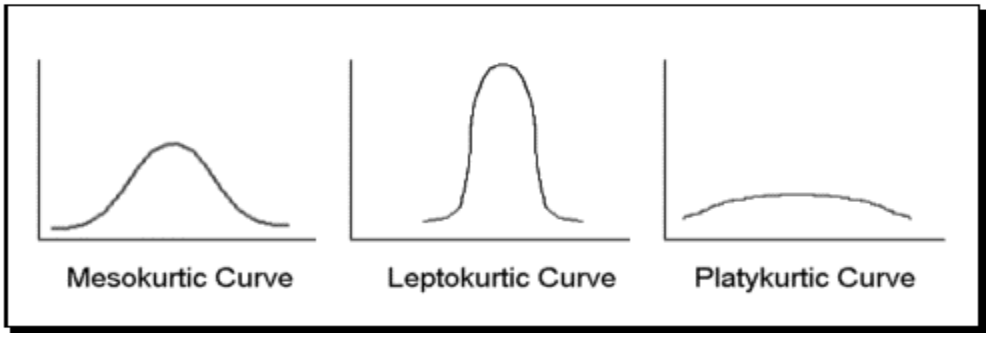
Skewness:

Karl Pearson's coefficient of skewness:

- $S_k = \dfrac{\text{Mean} - \text{Mode}}{\text{Standard deviation}}$
- $-3 \le S_k \le 3$

Kurtosis:

Kurtosis is a statistical measure used to describe the degree to which scores cluster in the tails or the peak of a frequency distribution. The peak is the tallest part of the distribution, and the tails are the ends of the distribution.
There are three types of kurtosis: mesokurtic, leptokurtic, and platykurtic.



- Mesokurtic: Distributions that are moderate in breadth and curves with a medium peaked height.
- Leptokurtic: More values in the distribution tails and more values close to the mean (i.e. sharply peaked with heavy tails)
- Platykurtic: Fewer values in the tails and fewer values close to the mean (i.e. the curve has a flat peak and has more dispersed scores with lighter tails).

What kurtosis is normal?
When kurtosis is equal to 3, the distribution is mesokurtic. This means the kurtosis is the same as the normal distribution, it is mesokurtic (medium peak).

What does it mean when kurtosis is zero?
When kurtosis is equal to 0, the distribution is platykurtic. A platykurtic distribution is flatter (less peaked) when compared with the normal distribution, with fewer values in its shorter (i.e., lighter, and thinner) tails.

Hypothesis testing:

https://towardsdatascience.com/hypothesis-testing-in-machine-learning-using-python-a0dc89e169ce

What is hypothesis testing ?
It is a statistical decision-making technique. In which we assume about population.

why do we use it ?
A hypothesis test evaluates two **mutually exclusive statements about a population to determine which statement is best supported by the sample data.** When we say that a finding is statistically significant, it's thanks to a hypothesis test.

what is basis of hypothesis ?
The basic of hypothesis is normalisation and standard normalisation. all our hypothesis revolves around basic of these 2 terms

Normal Distribution (3 M are equal) –
A variable is said to be normally distributed or have a normal distribution if its distribution has the shape of a normal curve — a special bell-shaped curve.
which has all of the following properties: 1. The mean, median, and mode are equal.

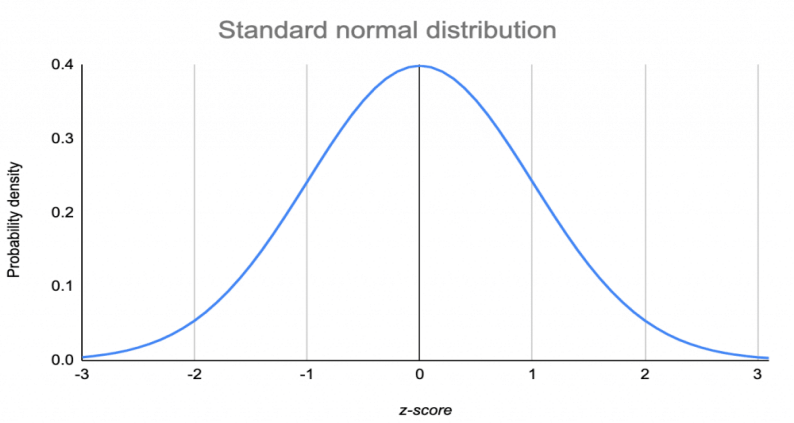
Standardised Normal Distribution (mean 0 and SD 1) :
A standard normal distribution is a normal distribution with **mean 0 and standard deviation 1**

Standard Normal Distribution

Machine Learning theoretical concepts

By: Vikram Pal

$$x_{new} = \frac{x - \mu}{\sigma}$$



Which is important parameter of hypothesis testing ?
Null hypothesis :-
The null hypothesis is a general statement or default position that there is **no relationship between two measured phenomena, or no association among groups**

Example : a company production is = 50 unit/per day etc.

Alternative hypothesis :-
The alternative hypothesis is the hypothesis used in hypothesis testing that is contrary to the null hypothesis. **It is usually taken to be that the observations are the result of a real effect** (with some amount of chance variation superposed)

Example : a company production is !=50 unit/per day etc.

Level of significance (0.05): Refers to the degree of significance in which we accept or reject the null hypothesis. 100% accuracy is not possible for accepting or rejecting a hypothesis, so we therefore select a level of significance that is usually 5%.

Type I error: When we reject the null hypothesis, although that hypothesis was true.

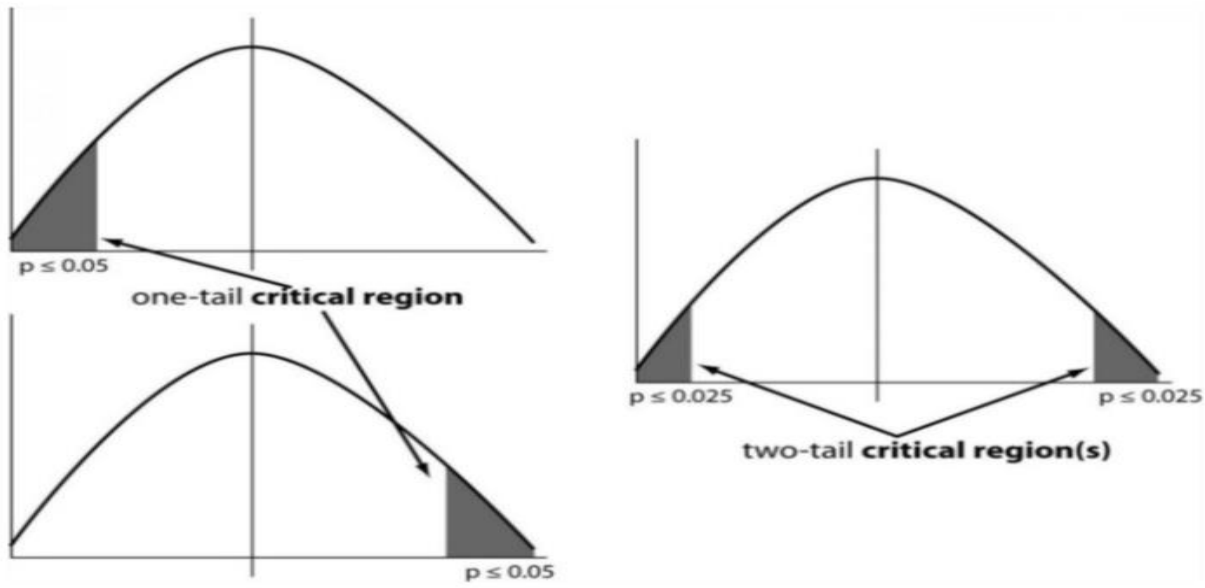
Type II errors: When we accept the null hypothesis, but it is false.

One tailed test (>=) :- A test of a statistical hypothesis , where the region of rejection is on only one side of the sampling distribution , is called a one-tailed test.
Example :- **a college has ≥ 4000 student** or data science ≤ 80% org adopted.

Two-tailed test (not equal to) :- A two-tailed test is a statistical test in which the critical area of a distribution is two-sided and tests whether a sample is greater than or less than a certain range of values.
Example : a college != 4000 student or data science != 80% org adopted

Machine Learning theoretical concepts

By: Vikram Pal



one and two-tailed images

P-value :- The P value, or calculated probability, is the probability of finding the observed, or more extreme, results when the null hypothesis (H 0) of a study question is true — the definition of 'extreme' depends on how the hypothesis is being tested.

Widely used hypothesis testing type :-

T-test:

It is also used for **comparing two population mean when SD is unknown.**

A t-test is a type of inferential statistic which is used to determine if there is a **significant difference between the means of two groups which** may be related in certain features.

T-test has 2 types : 1. one sampled t-test 2. two-sampled t-test.

One sample t-test : The One Sample T Test determines whether the sample mean is statistically different from a known or hypothesised population mean.

Two sampled T-test :- The Independent Samples t Test or 2-sample t-test compares the means of two **independent groups in order to determine whether there is statistical evidence that the associated population means are significantly different.**

Z-test:

It is used for **comparing two population mean when SD is known**. You would use a Z test if:

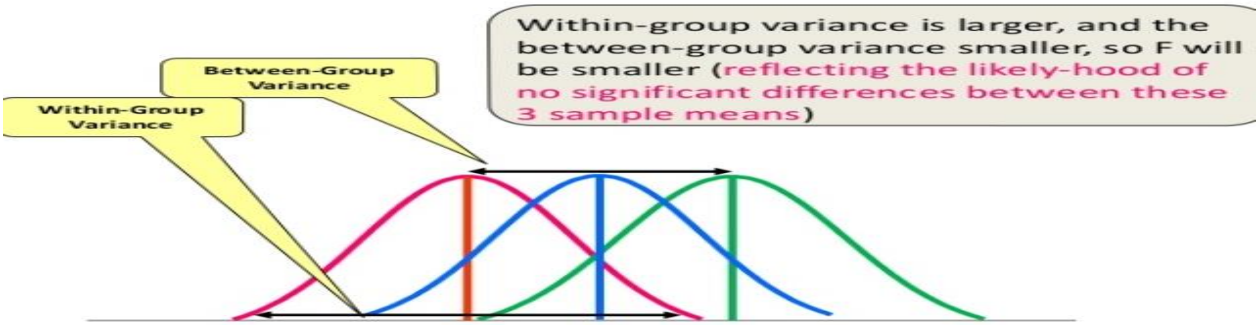
- Your sample size is greater than 30. Otherwise, use a t test.
- Data points should be independent from each other. In other words, one data point isn't related or doesn't affect another data point.
- Your data should be normally distributed. However, for large sample sizes (over 30) this doesn't always matter.
- Your data should be randomly selected from a population, where each item has an equal chance of being selected.
- Sample sizes should be equal if possible.

Machine Learning theoretical concepts

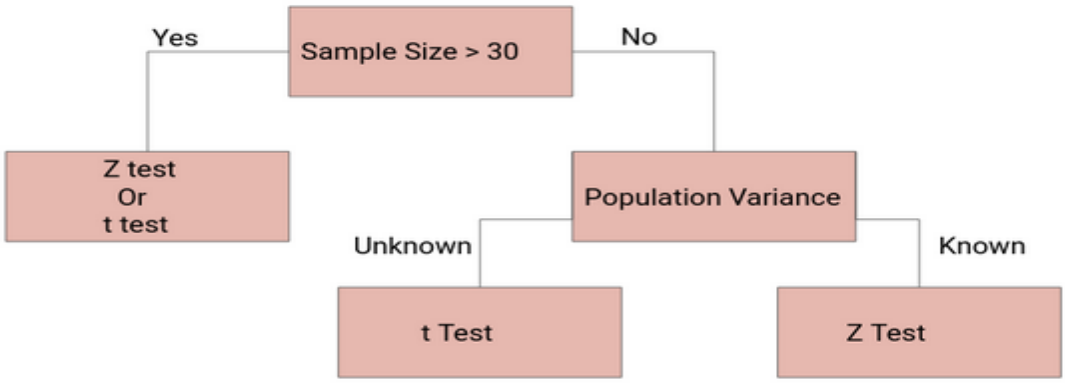
By: Vikram Pal

Two-sample Z test- In two sample z-test , like t-test here we are checking **two independent data groups and deciding whether sample mean of two group is equal or not**.
H0 : mean of two group is 0
H1 : mean of two group is not 0
Example : we are checking in blood data after blood and before blood data.

ANOVA: It is used for comparing more than two groups. The analysis of variance or ANOVA is a statistical inference test that lets you compare multiple groups at the same time.
F = Between group variability / Within group variability



Number of Independent Variables	ANOVA Used	Example
One	One-way ANOVA	Independent: *-fertilizer type* Dependent: -fruit produced
Two or more	Two-way ANOVA	Independent: *-fertilizer type* *-frequency of watering* Dependent: -fruit produced



Machine Learning theoretical concepts

By: Vikram Pal

Chi-square Test:

Chi Square test :**difference between the expected frequencies and the observed frequencies in one or more categories** of a contingency table

Contingency Table:

		Sport Preference			
		Archery	Boxing	Cycling	
Gender	Female	35	15	50	100
	Male	10	30	60	100
		45	45	110	200

Assume that we want to test if there is a statistically significant difference in Genders(M, F) population between Smokers and Non-Smokers.

Example: a scientist wants to know if education level and marital status are related for all people in some country.

Sampling Techniques:

Sampling is a method that allows us to get information about the population based on the statistics from a subset of the population (sample), without having to investigate every individual.

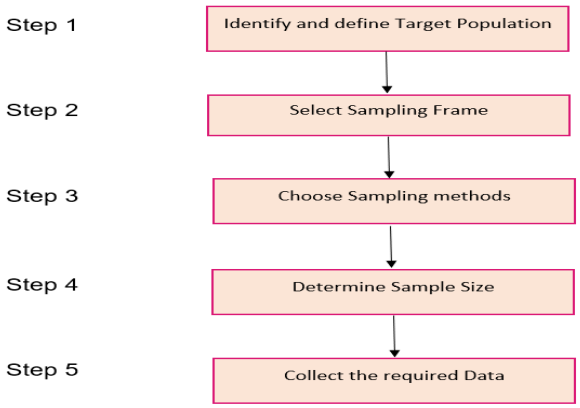
Let's say we go to a basketball court and take the average height of all the professional basketball players as our sample. This will not be considered a good sample because generally, a basketball player is taller than an average male and it will give us a bad estimate of the average male's height.

Here's a potential solution – find random people in random situations where our sample would not be skewed based on heights.

Steps involved in Sampling:

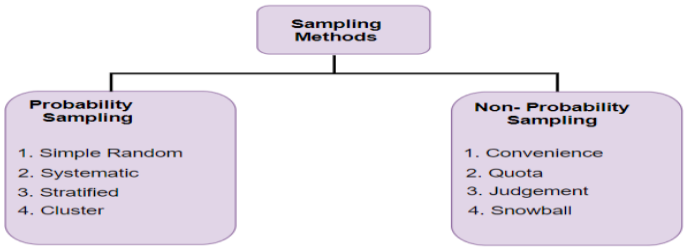
Machine Learning theoretical concepts

By: Vikram Pal



Different Types of Sampling Techniques

- **Probability Sampling**: In probability sampling, every element of the population has an equal chance of being selected. Probability sampling gives us the best chance to create a sample that is truly representative of the population
- **Non-Probability Sampling**: In non-probability sampling, all elements do not have an equal chance of being selected. Consequently, there is a significant risk of ending up with a non-representative sample which does not produce generalizable results

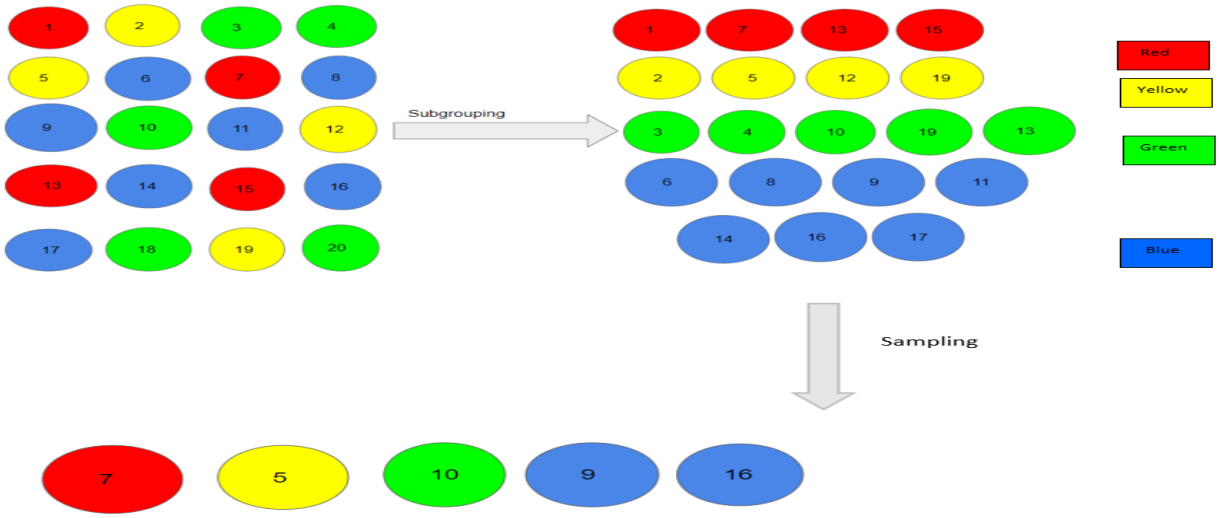


1. **Simple Random (random sampling):** This is a type of sampling technique you must have come across at some point. Here, **every individual is chosen entirely by chance and each member of the population has an equal chance of being selected**.

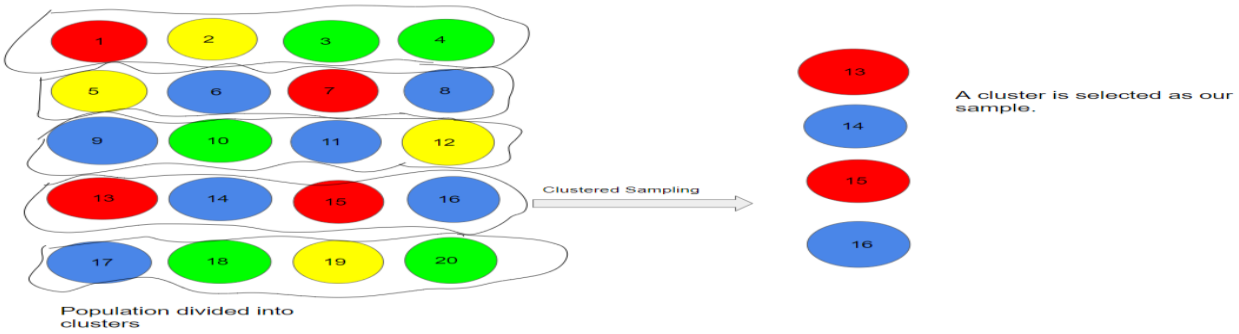
2. Stratified Sampling: In this type of sampling, we divide the population into subgroups (called strata) based on different traits like gender, category, etc. And then we select the sample(s) from these subgroups:

Machine Learning theoretical concepts

By: Vikram Pal



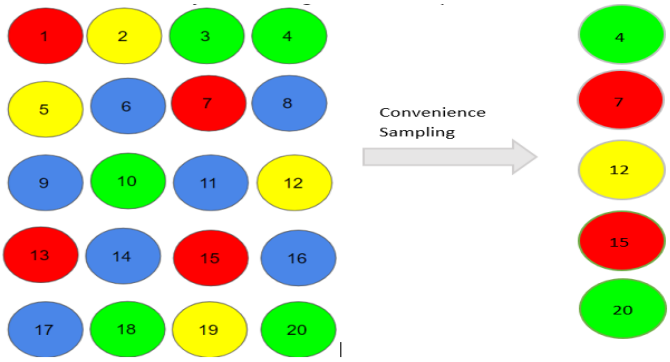
Cluster Sampling: In a clustered sample, we use the subgroups of the population as the sampling unit rather than individuals. The population is divided into subgroups, known as clusters, and a whole cluster is randomly selected to be included in the study:



Types of Non-Probability Sampling:

1. Convenience Sampling: This is perhaps the easiest method of sampling because individuals are selected based on their availability and willingness to take part.

Here, let's say individuals numbered 4, 7, 12, 15 and 20 want to be part of our sample, and hence, we will include them in the sample

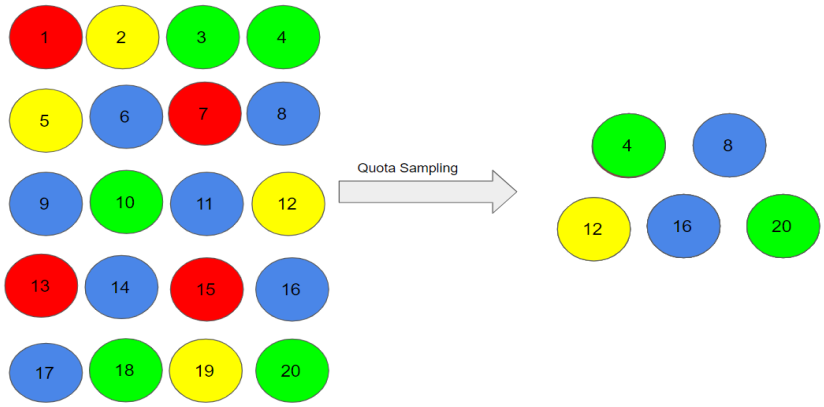


Convenience sampling is prone to significant bias, because the sample may not be the representation of the specific characteristics such as religion or, say the gender, of the population.

Machine Learning theoretical concepts

By: Vikram Pal

2. Quota Sampling: In this type of sampling, we choose items based on predetermined characteristics of the population. Consider that we have to select individuals having a number in multiples of four for our sample:



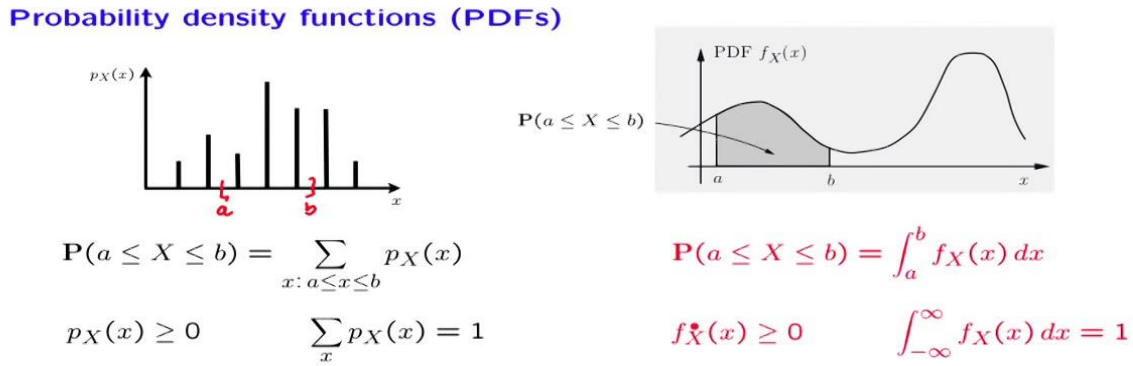
3. Judgment Sampling

It is also known as selective sampling. It depends on the judgment of the experts when choosing whom to ask to participate.

4. Snowball Sampling:

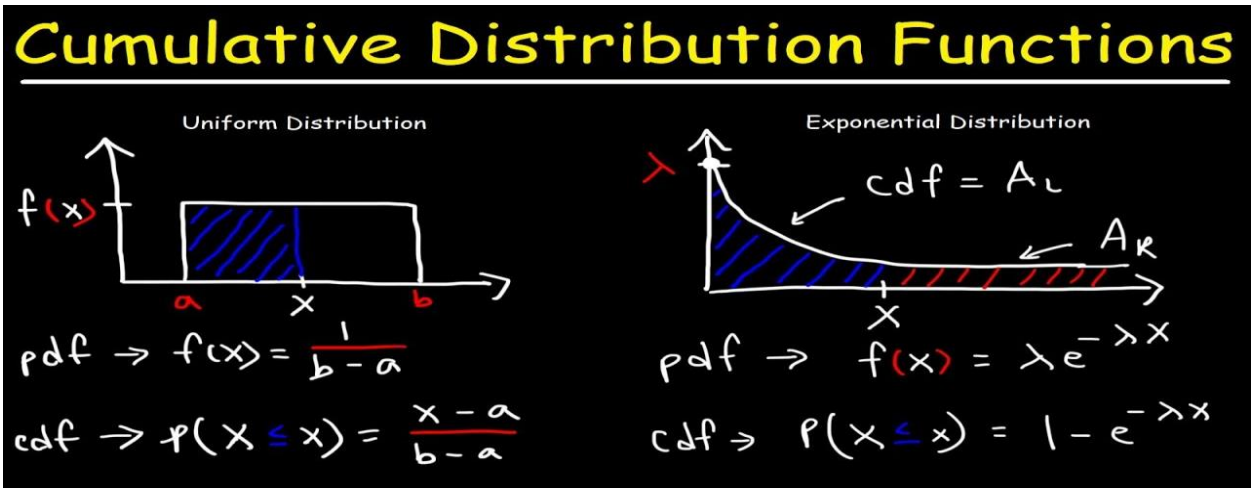
I quite like this sampling technique. Existing people are asked to nominate further people known to them so that the sample increases in size like a rolling snowball. This method of sampling is effective when a sampling frame is difficult to identify.

Probability Density Function:

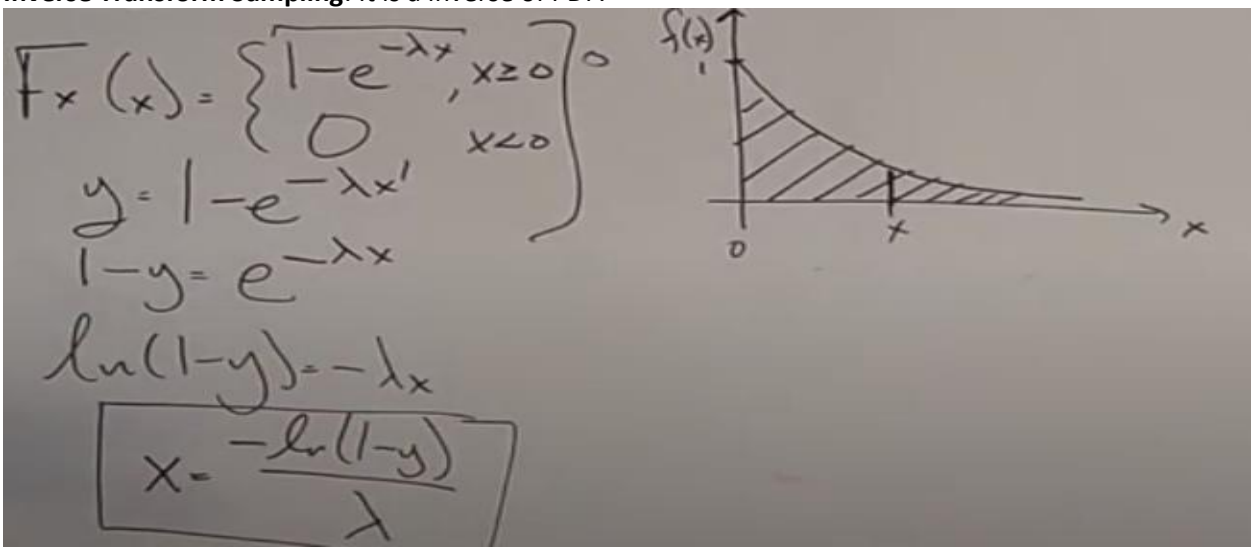


Machine Learning theoretical concepts

By: Vikram Pal



Inverse Transform Sampling: It is a inverse of PDF.



Monte Carlo sampling methods that are able to **draw independent samples from the distribution.**
Markov Chain Monte Carlo: method **draw samples where the next sample is dependent** on the existing sample, called a Markov Chain.

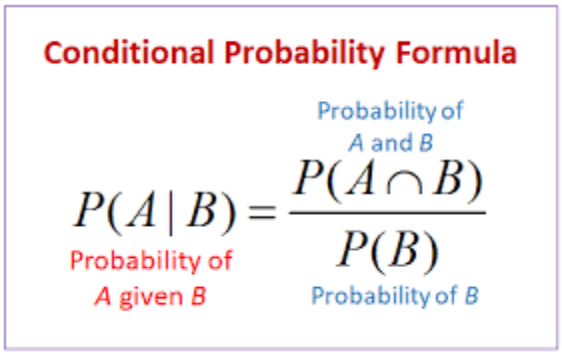
Machine Learning theoretical concepts

By: Vikram Pal

Probability:

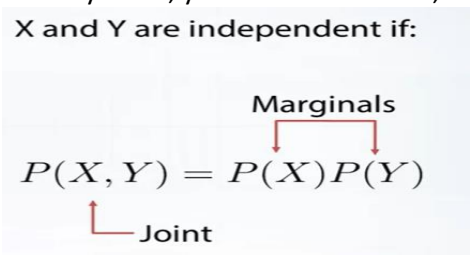
https://pasaentuciudad.com.mx/30-probability-and-statistics-interview-questions-for-data-scientists/

Conditional Probability:



Independent Probability:

Two events, A and B, are **independent** if the outcome of A **does not affect** the outcome of B.
In many cases, you will see the term, **"With replacement"**.



Independent Events are **not affected** by previous events.

A coin does not "know" it came up heads before.

And each toss of a coin is a perfect isolated thing.

Example: what is the probability of getting a "4" or "6" when rolling a die?

Number of ways it can happen: 2 ("4" and "6")

Total number of outcomes: 6 ("1", "2", "3", "4", "5" and "6")

So the probability = *2***6** = *1***3** = 0.333...

Machine Learning theoretical concepts

By: Vikram Pal

Mutually exclusive:

events are things that can't happen at the same time. For example, you can't run backwards and forwards at the same time.

When two events (call them "A" and "B") are Mutually Exclusive it is **impossible** for them to happen together:

$$\mathbf{P(A \text{ and } B) = 0}$$

"The probability of A and B together equals 0 (impossible)"

But, for Mutually Exclusive events, the probability of A **or** B is the sum of the individual probabilities:

$$\mathbf{P(A \text{ or } B) = P(A) + P(B)}$$

*"The probability of A **or** B equals the probability of A **plus** the probability of B"*

Example: King OR Queen

In a Deck of 52 Cards:

- the probability of a King is 1/13, so **P(King)=1/13**
- the probability of a Queen is also 1/13, so **P(Queen)=1/13**

When we combine those two Events:

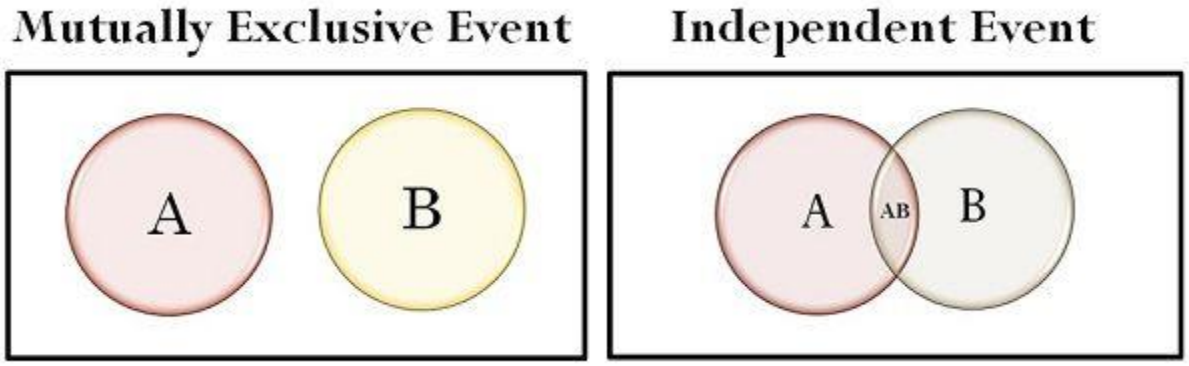
- The probability of a King **or** a Queen is (1/13) + (1/13) = **2/13**

Which is written like this:

$$P(\text{King or Queen}) = (1/13) + (1/13) = 2/13$$

Machine Learning theoretical concepts

By: Vikram Pal



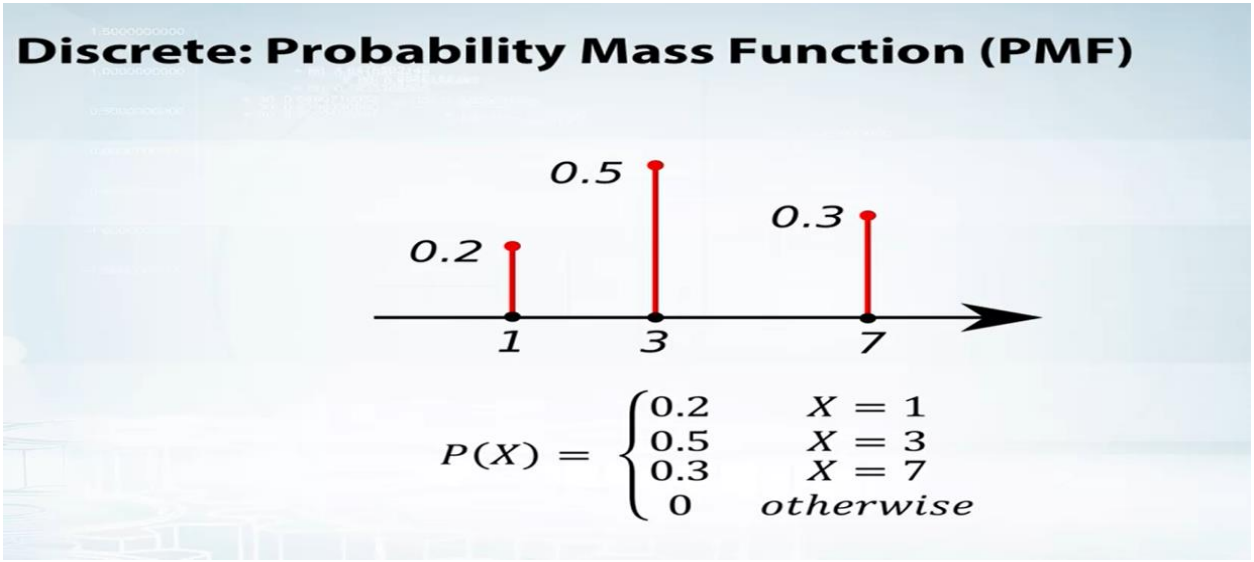
Bayes Theorem:

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

$$P(B_J \mid A) = \frac{P(A \mid B_J)P(B_J)}{\sum_{i=1}^{n} P(A \mid B_i)P(B_i)}$$

Calcworkshop.com

Discrete: Probability Mass Function:

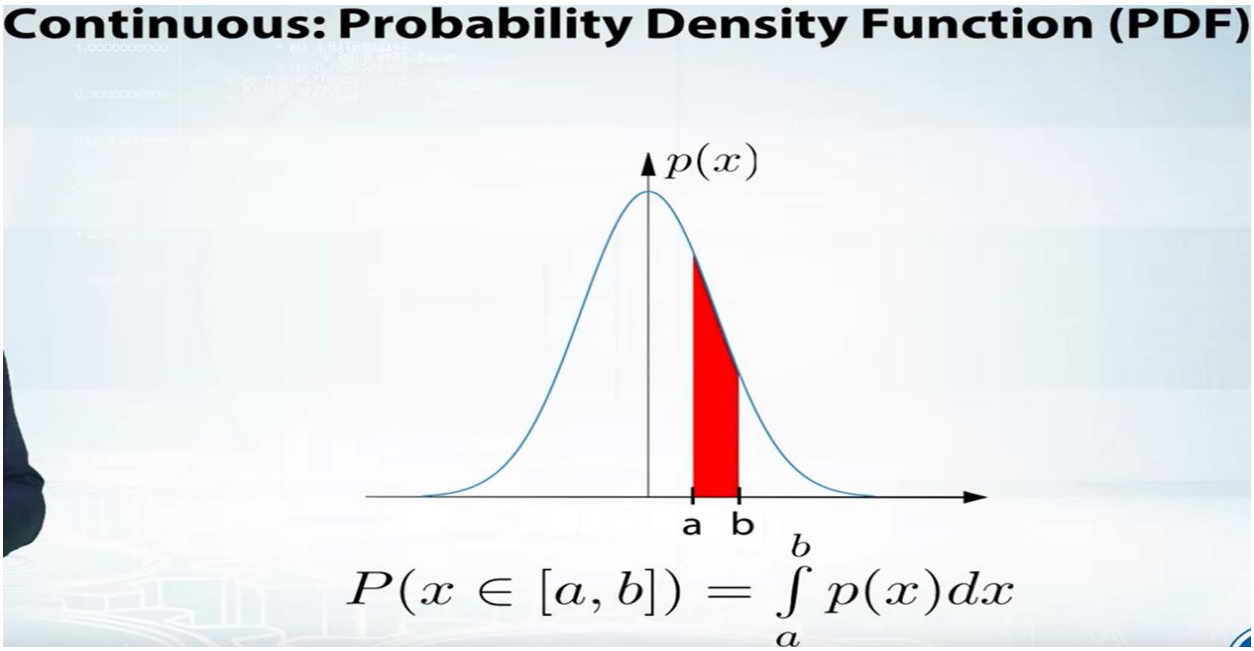


Continuous: Probability Density Function:

https://opentextbc.ca/introbusinessstatopenstax/chapter/using-the-normal-distribution/

Machine Learning theoretical concepts

By: Vikram Pal



Recommendation System:

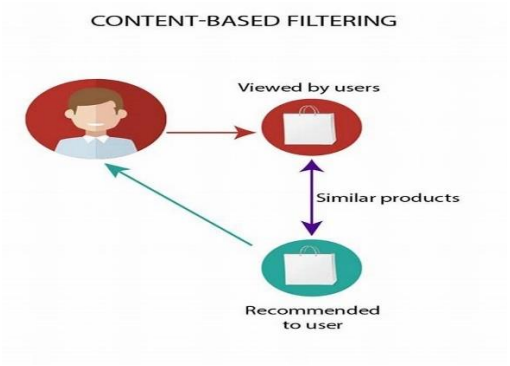
Popularity based recommendation:

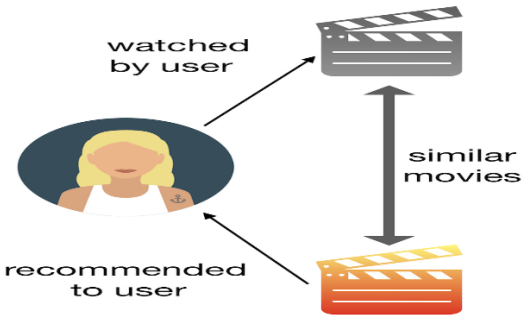
Popularity based recommendation system works with the trend. It basically uses the items which are in trend right now.

E.g.: new user to OTT platform and suggesting them a movie without any idea of user preferences.

Content based recommendation:

Content based means recommending **content to you based on its similarities with other content** without regarding the preferences of other users.



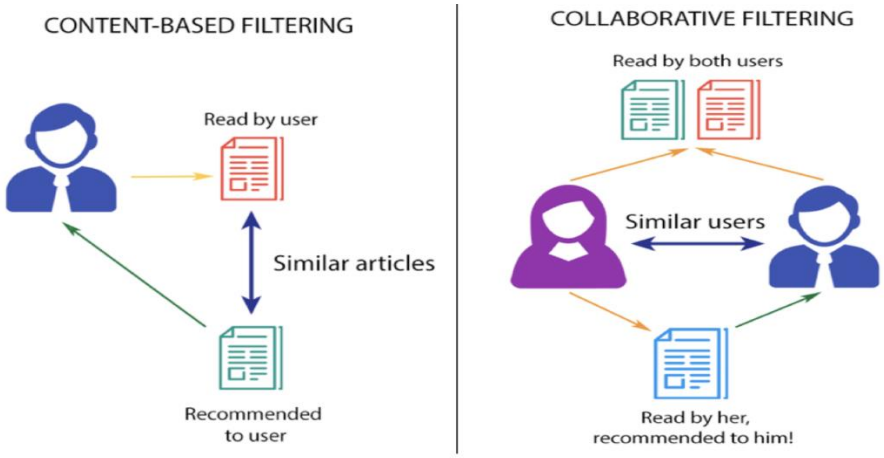


Searching a book on Amazon website for example “Long Live” and Amazon suggesting book with having title “Long Live”

Machine Learning theoretical concepts

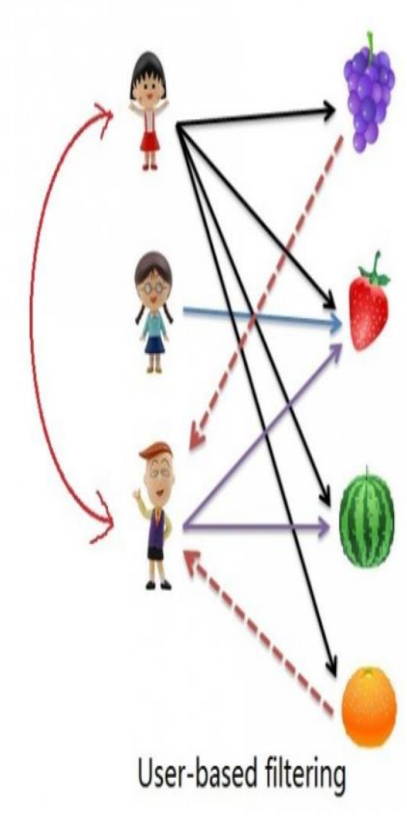
By: Vikram Pal

Collaborative filtering:



User-based vs Item-based Collaborative Filtering:

User-User based Collaborative filtering:



User-User collaborative filtering

User/Movie	x1	x2	x3	x4	x5	Mean User Rating
A	4	1	-	4	-	3
B	-	4	-	2	3	3
C	-	1	-	4	4	3

$$r_{AC} = [(1-3)*(1-3) + (4-3)*(4-3)]/[((1-3)^2 + (4-3)^2)^{1/2} * ((1-3)^2 + (4-3)^2)^{1/2}] = 1$$

$$r_{BC} = [(4-3)*(1-3) + (2-3)*(4-3) + (3-3)*(4-3)]/[((4-3)^2 + (2-3)^2 + (3-3)^2)^{1/2} * ((1-3)^2 + (4-3)^2 + (4-3)^2)^{1/2}] = -0.866$$

Rating of the item is done using the rating of neighbouring users. In simple words, it is based on the notion of **users' similarity**. **Pearson Correlation provides superior results.**

Pros:

Machine Learning theoretical concepts

By: Vikram Pal

1. Performance increase as the neighbourhood size increase

2. LSA has a beneficial effect on user-based recommendation.

Cons:

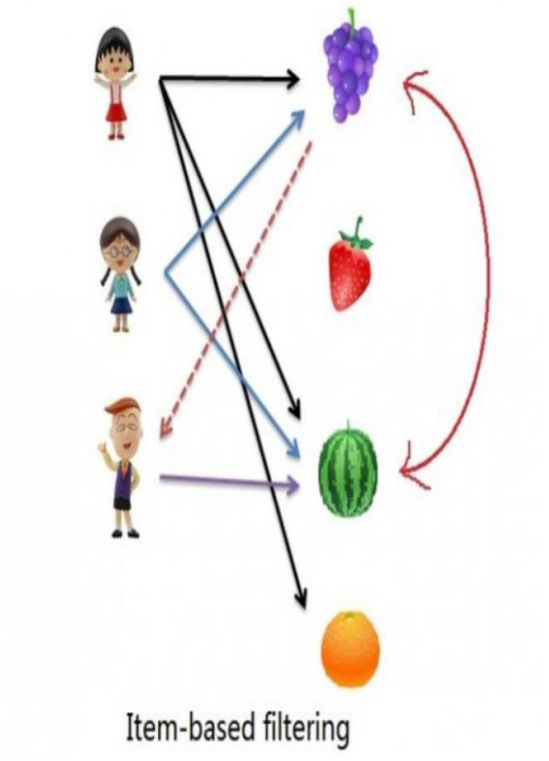
1. **Sparsity**: the percentage of people who rate items is really low.

2. **Cold Start**: New user will have little information about them to be compared with other users.

3. **Scalability**: the more user there are in the system, the greater the cost of finding nearest K neighbours will be.

4. **High space requirements**: The number of users will be greater than the number of items, the space requirement is more for user-based recommendation as compared to item-based recommendation system.

Item-item based Collaborative filtering:



Item-Item collaborative filtering

User/Movie	x1	x2	x3	x4	x5
A	4	1	2	4	4
B	2	4	4	2	1
C	–	1	–	3	4
Mean Item Rating	3	2	3	3	3

$C_{14} = [(4-3)*(4-3) + (2-3)*(2-3)] / [((4-3)^2 + (2-3)^2)^{1/2} * ((4-3)^2 + (2-3)^2)^{1/2}] = 1$

$C_{15} = [(4-3)*(4-3) + (2-3)*(1-3)] / [((4-3)^2 + (2-3)^2)^{1/2} * ((4-3)^2 + (1-3)^2)^{1/2}] = 0.94$

The rating of the item is predicted using the user's own rating on neighbouring items. In simple words, it is based on the notion of **item similarity. Adjusted Cosine similarity provides superior results.**

Pros:

1. The item-based similarity is a common choice for **high load services**.

2. The reason is that usually, **item's neighbourhood changes much slower in comparison to the user's neighbourhood**.

3. The **greater prediction accuracy of the item-based method is its main advantage.**

4. Item-based methods are also more robust to shilling attacks in recommender systems.

Cons:

1. Item-based methods might sometimes recommend obvious items or items that are **not novel from previous user experiences**.

Within the collaborative filtering ecosystem, there are number of options:

Singular value decomposition (SVD): is one of the first such techniques, which (as the name suggests) decomposes the user-item preference matrix into three elements: **the user-feature eigenvectors, the feature-item eigenvectors, and a diagonal matrix of eigenvalues.**

SVD → (user, item) --- > (user, feature) (feature, items) and a diagonal matrix of eigenvalues.

Gradient descent-based matrix factorization: The core idea here is to create **parameters and iteratively update them to minimize some cost function**.

1 Cost Function and Gradient

1.1 Mean Squared Error

$$MSE(U_1I_1) = (U_1I_1 - [(U_1F_1 * F_1I_1) + (U_1F_2 * F_2I_1)])^2 \quad (1)$$

1.2 Gradient with respect to U_1F_1

$$\frac{\partial U_1I_1}{\partial U_1F_1} = -2 * (U_1I_1 - [(U_1F_1 * F_1I_1) + (U_1F_2 * F_2I_1)]) * (F_1I_1) \quad (2)$$

Challenges Faced by Recommendation Systems:

Any predictive model or recommendation systems with no exception rely heavily on data. They make reliable recommendations based on the facts that they have. It's only natural that the finest recommender systems come from organizations with large volumes of data, such as Google, Amazon, Netflix, or Spotify. To detect commonalities and suggest items, good recommender systems evaluate item data and client behavioral data. Machine learning thrives on data; the more data the system has, the better the results will be.

Data is constantly changing, as are user preferences, and your business is constantly changing. That's a lot of new information. Will your algorithm be able to keep up with the changes? Of course, real-time recommendations based on the most recent data are possible, but they are also more difficult to maintain. Batch processing, on the other hand, is easier to manage but does not reflect recent data changes.

The recommender system should continue to improve as time goes on. Machine learning techniques assist the system in "learning" the patterns, but the system still requires instruction to give appropriate

results. You must improve it and ensure that whatever adjustments you make continue to move you closer to your business goal.

Evaluation Metrics for Recommendation System:

https://analyticsindiamag.com/how-to-measure-the-success-of-a-recommendation-system/

Predictive Accuracy Metrics:

Predictive accuracy or rating prediction measures address the subject of how near a recommender's estimated ratings are to genuine user ratings. This sort of measure is widely used for evaluating non-binary ratings.

Classification Accuracy Metrics:

Classification accuracy measures attempt to evaluate a recommendation algorithm's successful decision-making capacity (SDMC). They are useful for user tasks such as identifying nice products since they assess the number of right and wrong classifications as relevant or irrelevant things generated by the recommender system.

The exact rating or ranking of objects is ignored by SDMC measures, which simply quantify correct or erroneous classification. This type of measure is particularly well suited to e-commerce systems that attempt to persuade users to take certain actions, such as purchasing products or services.

Rank Accuracy Metrics:

In statistics, a rank accuracy or ranking prediction metric assesses a recommender's ability to estimate the correct order of items based on the user's preferences, which is known as rank correlation measurement. As a result, if the user is given a long, sorted list of goods that are recommended to him, this type of measure is most appropriate.

The relative ordering of preference values is used in a rank prediction metric, which is independent of the exact values assessed by a recommender. A recommender that consistently overestimates item ratings to be lower than genuine user preferences, for example, might still get a perfect score as long as the ranking is correct.

Model Evaluation metrics:

1. **Mean Average precision at K:**

It gives **how much relevant is the list of recommended items**. Here precision at K means Recommended items in top k sets that are relevant.

2. **Coverage:**

It is the percentage of items in the training data model able to recommend in test sets. Or Simply, the percentage of a possible recommendation system can predict.

3. **Personalization:**

It is basically how many same items the model recommends to different users. Or, the dissimilarity between users lists and recommendations.

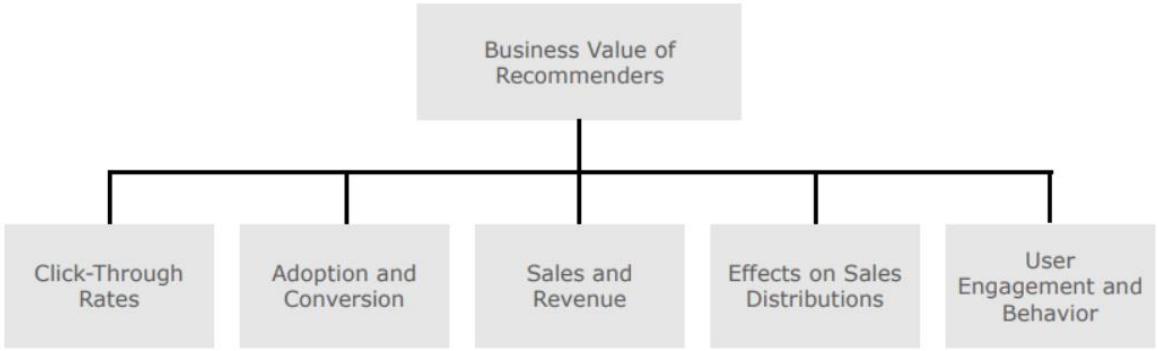
4. **Intralist Similarity:**

It is an average cosine similarity of all items in a list of recommendations.

Machine Learning theoretical concepts

By: Vikram Pal

Business Value of Recommender:



Click-Through Rates

The click-through rate (CTR) is a metric that measures how many people click on the recommendations. The basic notion is that if more people click on the recommended things, the recommendations are more relevant to them.
For Example: In news recommendations, the CTR is a widely used metric.

Adoption and Conversion

Click-through rates are often not the final success measure to pursue in recommendation scenarios, unlike online business models dependent on adverts. While the CTR can measure user attention or interest, it can't tell you whether users liked the recommended news article they clicked on or if they bought something based on a recommendation.

For Example: YouTube employs the idea of "long CTRs," in which a user's clicks on suggestions are only tallied if they view a particular percentage of a video. Similarly, Netflix utilizes a metric called "take-rate" to determine how many times a video or movie was watched after being recommended.

Sales and Revenue:
In many cases, the adoption and conversion measures outlined in the previous section are more telling of a recommender's prospective business value than CTR measures alone. When customers choose more than one item from a list of suggestions, this is a good indicator that a new algorithm was successful in identifying later purchases or views. stuff that the user is interested in.

Machine Learning theoretical concepts

By: Vikram Pal

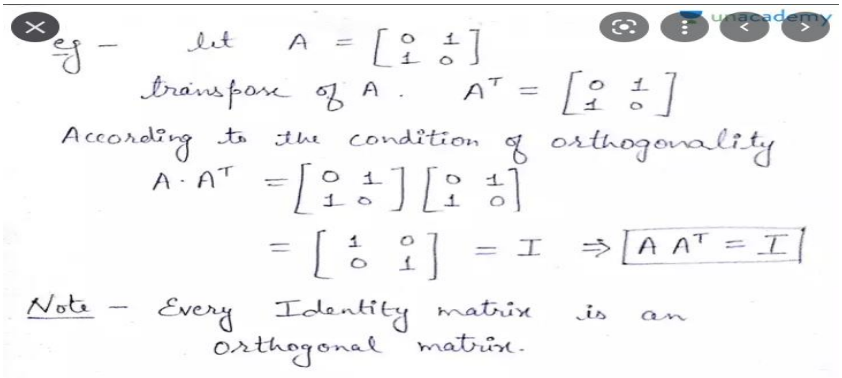
Measurement	Remarks
Click-Through Rates	Easy to measure and established, but often not the ultimate goal.
Adoption and Conversion	Easy to measure, but often requires a domain- and application specific definition. Requires interpretation and does not always translate directly into business value.
Sales and Revenue	Most informative measure, but cannot always be determined directly.
Effects on Sales Distribution	A very direct measurement; requires a thorough understanding of the effects of the shifts in sales distributions.
User Engagement and Behavior	Often, a correspondence between user engagement and customer retention is assumed; still, it remains an approximation.

Apart from all of this, we can also leverage our standard ML evaluation metrics to evaluate ratings and predictions that are as follows.

- Precision
- Recall
- F1-measure
- False-positive rate
- Mean average precision
- Mean absolute error
- The area under the ROC curve (AUC)

Matrix Operations:

Orthogonal Matrix:



Orthonormal Matrix:

$$Q^TQ = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ \vdots \\ q_n^T \end{bmatrix} [q_1\ q_2\ q_3\ \dots\ q_n] = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} = I$$

Machine Learning theoretical concepts

By: Vikram Pal

Diagonal Matrix:

Diagonal Matrices

A square matrix is called a diagonal matrix if nondiagonal entries are all zero. The main diagonal can be constants or zeros. A diagonal matrix must fit the following form:

$$D = \begin{bmatrix} d_{11} & 0 & 0 & \dots & 0 \\ 0 & d_{22} & 0 & \dots & 0 \\ 0 & 0 & d_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_{nn} \end{bmatrix}$$

- *If any of the diagonals of a diagonal matrix are zero, the matrix is singular, meaning it is not invertible or does not have an inverse matrix.*

Determinant of a matrix:

Finding the Determinant of a Three-By-Three Matrix

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

$$\det(A) = a_1 \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}$$

$$= a_1(b_2c_3 - b_3c_2) - a_2(b_1c_3 - b_3c_1) + a_3(b_1c_2 - b_2c_1)$$

Egin Values of a matrix:

$$\det(A - \lambda I) = 0$$

$$(A - \lambda I)x = 0$$

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}$$

$$\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

$$\det(A - \lambda I) = 0$$

$$\det\left(\begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0$$

$$\det\left(\begin{bmatrix} 1-\lambda & 4 \\ 3 & 2-\lambda \end{bmatrix}\right) = 0$$

$$(1-\lambda)(2-\lambda) - 12 = 0$$

$$\lambda^2 - 3\lambda - 10 = 0$$

$$(\lambda - 5)(\lambda + 2) = 0$$

$$\lambda = 5, -2$$

Lambda is eginvalues.

Egin Vectors:
By plugging in the discovered eigenvalues into our originally derived equation, we can find the eigenvectors.

$$\lambda = 5, -2$$

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}$$

$$(A - \lambda I)x = 0$$

$$\begin{bmatrix} 1-\lambda & 4 \\ 3 & 2-\lambda \end{bmatrix} x = 0 \qquad \begin{bmatrix} 1-\lambda & 4 \\ 3 & 2-\lambda \end{bmatrix} x = 0$$

$$\begin{bmatrix} -4 & 4 \\ 3 & -3 \end{bmatrix} x = 0 \qquad \begin{bmatrix} 3 & 4 \\ 3 & 4 \end{bmatrix} x = 0$$

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad x = \begin{bmatrix} -4 \\ 3 \end{bmatrix}$$

Machine Learning theoretical concepts

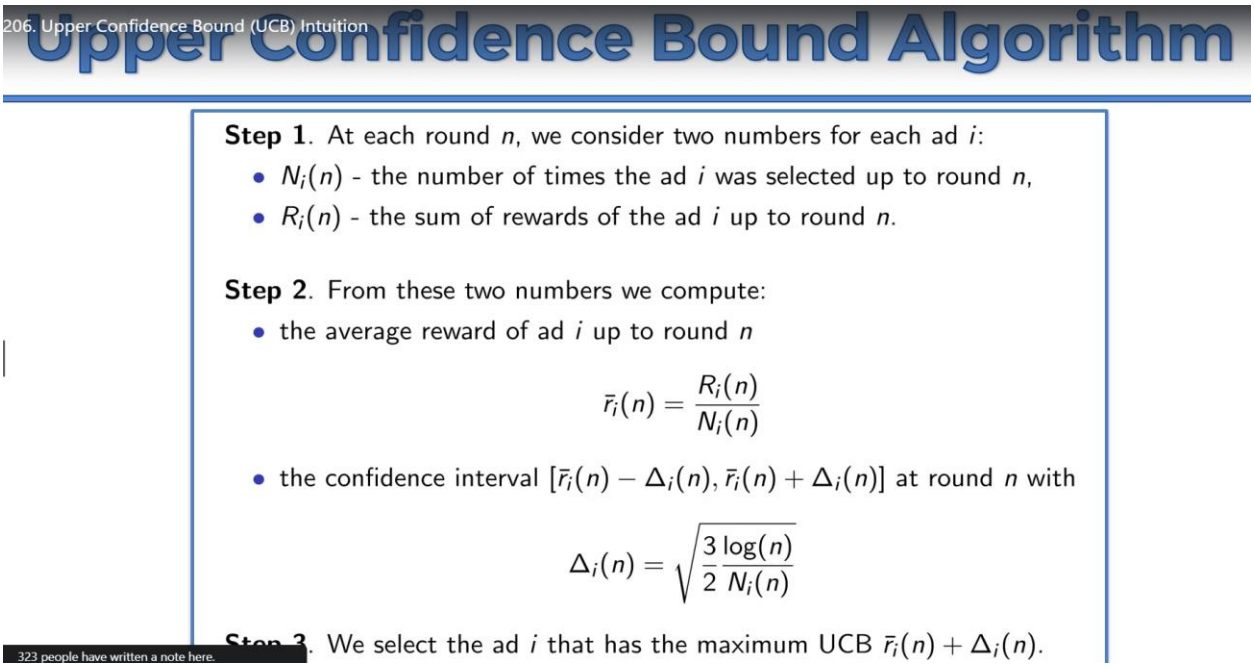
By: Vikram Pal

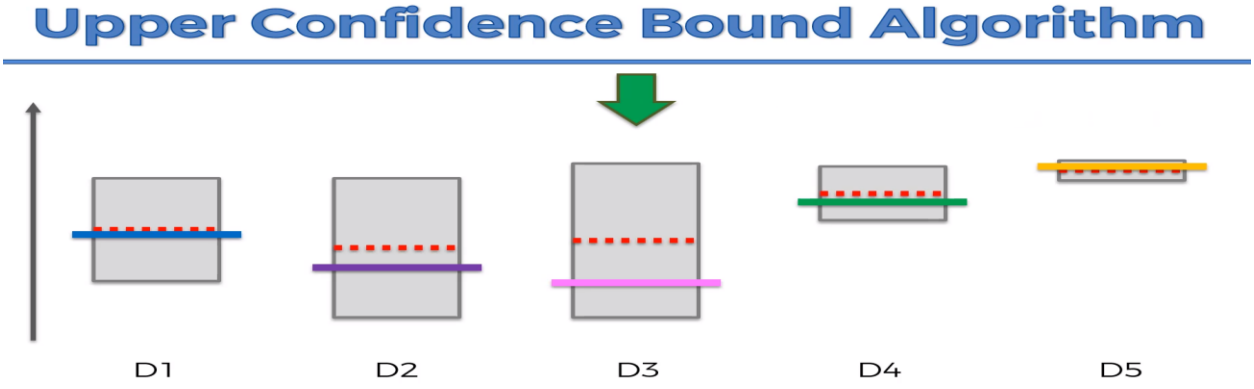
Pointwise multiplication and addition (element wise):

$$\begin{matrix} & G & & & H & & & N & \\ \begin{bmatrix} 3 & 5 & 7 \\ 4 & 9 & 8 \end{bmatrix} & \circ & \begin{bmatrix} 1 & 6 & 3 \\ 0 & 2 & 9 \end{bmatrix} & = & \begin{bmatrix} 3\times1 & 5\times6 & 7\times3 \\ 4\times0 & 9\times2 & 8\times9 \end{bmatrix} \end{matrix}$$

Reinforcement Learning:

Upper Confidence Bound:





Machine Learning theoretical concepts

By: Vikram Pal

Thomson Sampling Algorithms:

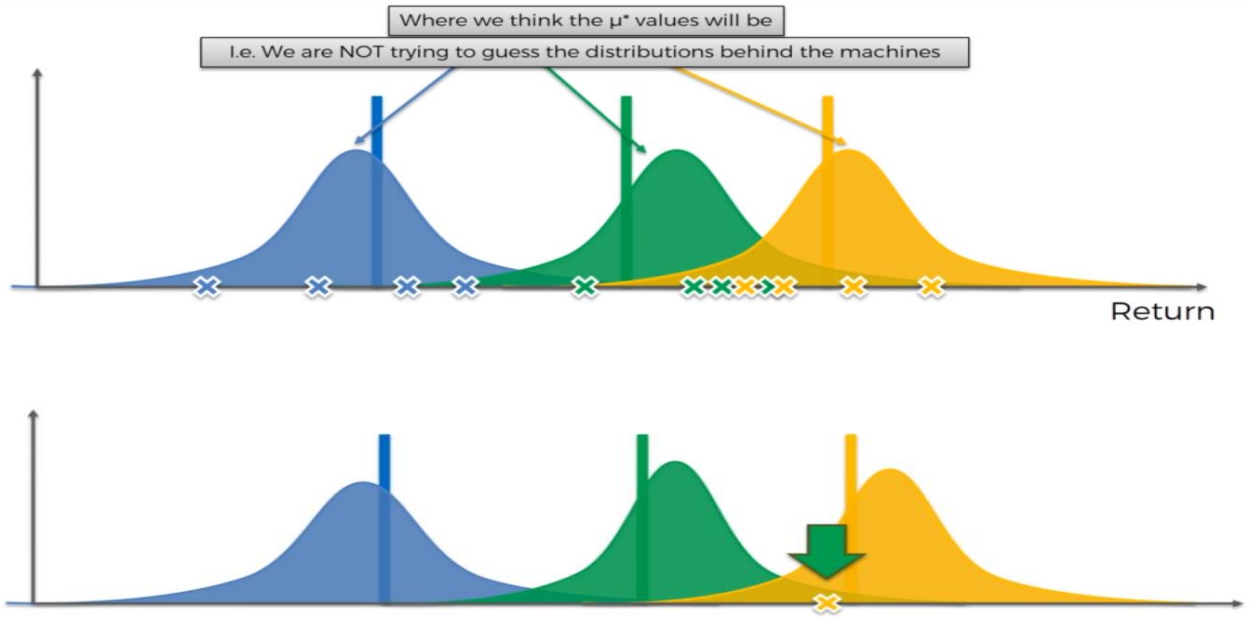
Bayesian Inference

- Ad i gets rewards $\mathbf{y}$ from Bernoulli distribution $p(\mathbf{y}|\theta_i) \sim \mathcal{B}(\theta_i)$.
- θ_i is unknown but we set its uncertainty by assuming it has a uniform distribution $p(\theta_i) \sim \mathcal{U}([0,1])$, which is the prior distribution.
- Bayes Rule: we approach θ_i by the posterior distribution

$$\underbrace{p(\theta_i|\mathbf{y})}_{\text{posterior distribution}} = \frac{p(\mathbf{y}|\theta_i)p(\theta_i)}{\int p(\mathbf{y}|\theta_i)p(\theta_i)d\theta_i} \propto \underbrace{p(\mathbf{y}|\theta_i)}_{\text{likelihood function}} \times \underbrace{p(\theta_i)}_{\text{prior distribution}}$$

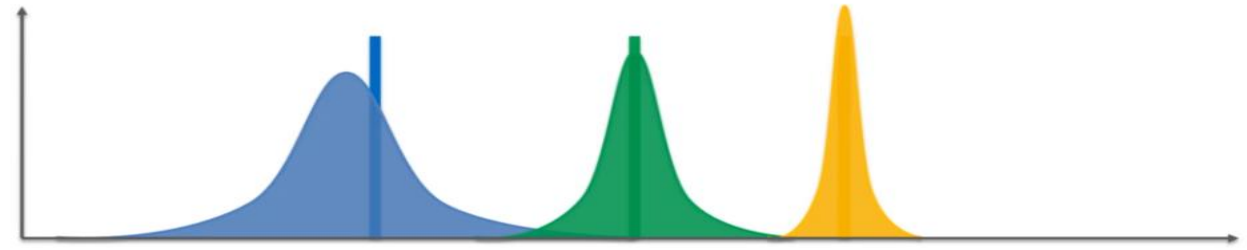
- We get $p(\theta_i|\mathbf{y}) \sim \beta(\text{number of successes} + 1, \text{number of failures} + 1)$
- At each round n we take a random draw $\theta_i(n)$ from this posterior distribution $p(\theta_i|\mathbf{y})$, for each ad i.
- At each round n we select the ad i that has the highest $\theta_i(n)$.

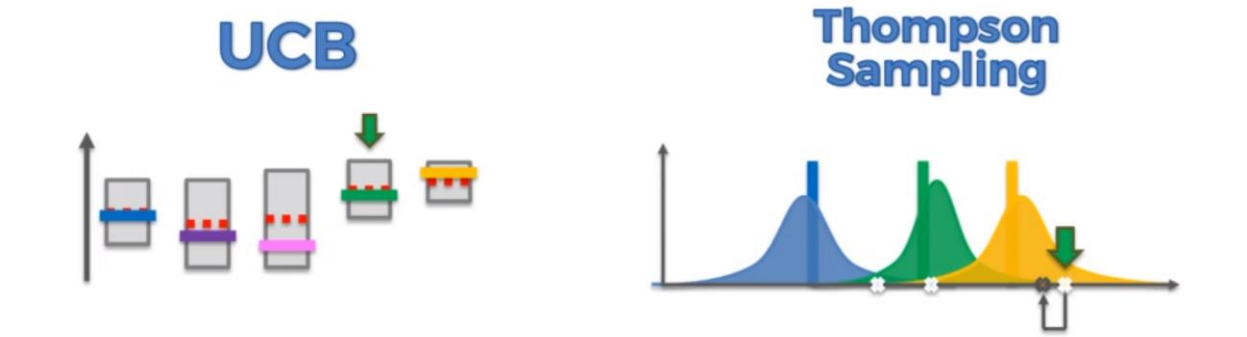
Thompson Sampling Algorithm



Machine Learning theoretical concepts

By: Vikram Pal





- Deterministic
- Requires update at every round

- Probabilistic
- Can accommodate delayed feedback
- Better empirical evidence

SQL & Pandas

MATCHING TEXT PATTERN WILDCARDS:

- LIKE 'TOM%' → FIND ALL MATCH STARTING WITH TOM
- LIKE '%TOM' → FIND ALL MATCH ENDING WITH TOM
- LIKE '[ABC]%' → FIND ALL MATCHES STARTING WITH A, B AND C
- LIKE '[A-Z]%' → FIND ALL MATCHES STARTING WITH A TO Z
- LIKE '%[0-9]%' → FIND ALL MATCHES STARTING SOMETHING BUT INBETWEEN 0-9 INTEGER (E.G ALIK98A)

We can use 'NOT LIKE' for opposite

TEXT MANIPULATION AND CLEANING:

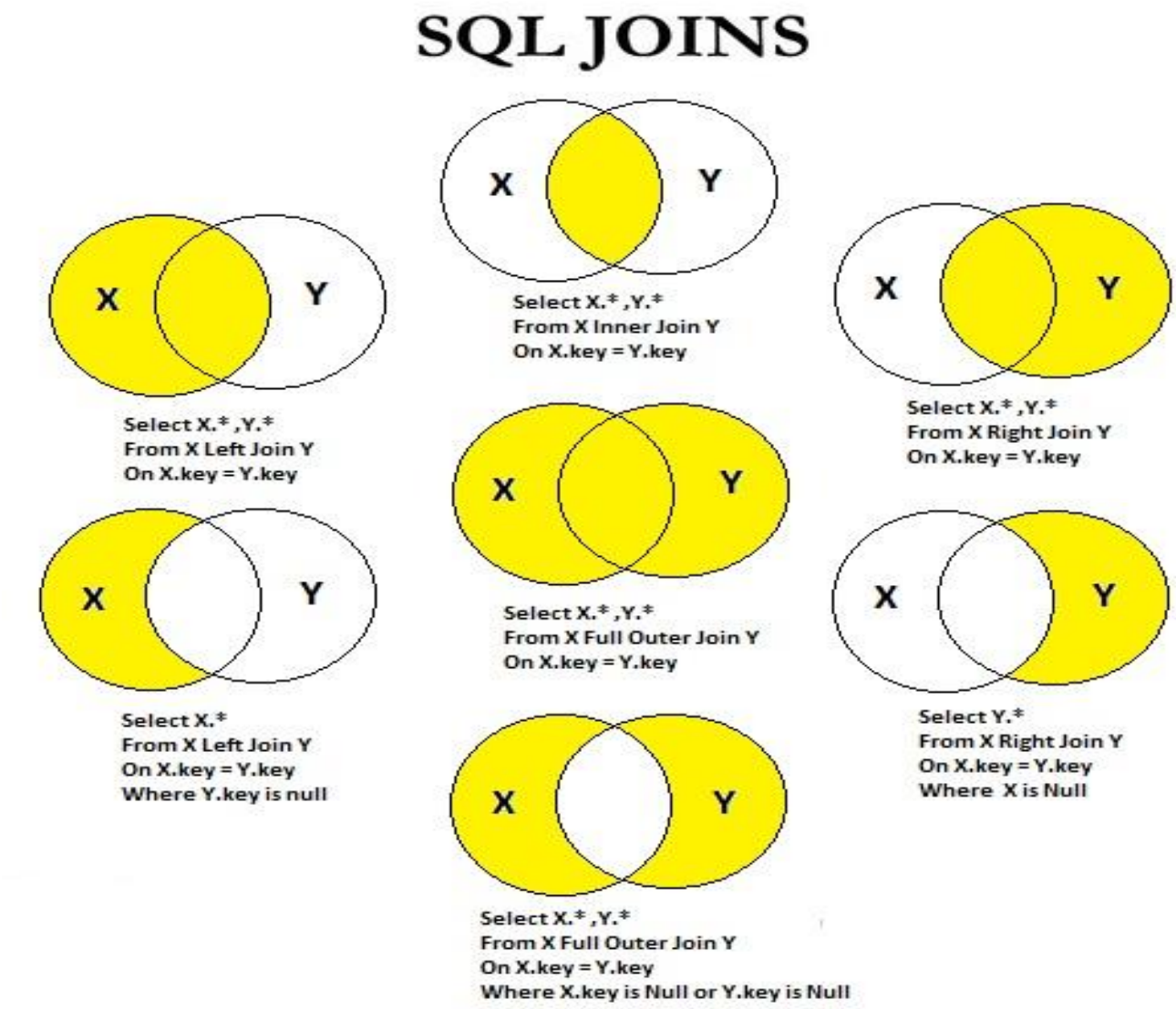
- LEFT(COLUMN_NAME, NUMBER_OF_CHAR)
- RIGHT(COLUMN_NAME, NUMBER_OF_CHAR)
- REPLACE(COLUMN_NAME, REPLACEMENT, NEW)
- PADDING LEFT OR LEFT: CALCULATE COLUMN VALUES LEN AND APPEND PADDED CHAR ON LEFT OR LEFT SIDE
  E.G `[PADDED]=RIGHT('0000000000'+[NationalIDNumber],10)`

Machine Learning theoretical concepts

By: Vikram Pal

SQL DATE:

- LAST SEVEN DAYS:
  WHERE DATEDIFF(DAY, [ORDERDATE], DATEFROMPARTS(2013,12,25)) BETWEEN 0 AND 7
- DATEDIFF(DAY,STARTING,ENDING)
- DATEADD(DAY,NUMBER_OF_DAY, COLUMNS_NAME)
- DATEFROMPARTS(YEAR, MONTH, DAY)



Pandas Left Only and right only:

df2=pd.merge(df_stream, df_transaction,on="customer_id",how="outer",indicator=True)

left only:

```
df[df["_merge"]=="left_only"]
```

right only:

```
df[df["_merge"]=="right_only"]
```

Self Join:

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

Having:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Between and Not Between:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

```
ANY:
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

Notes:

Multiple tables with summations:

https://www.codesadda.com/hackerrank/mysql/HackerRank%20MySQL%20-%20Interviews/

UNION: Returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table.

Machine Learning theoretical concepts

By: Vikram Pal

UNION ALL: Returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table.

UNION With WHERE: returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table.

UNION ALL With WHERE: returns the German cities (duplicate values also) from both the "Customers" and the "Suppliers" table.

Group By: The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

INNER JOIN: Select all orders with customer information or selects all orders with customer and shipper information

ROW_NUMBER vs DENSE_RANK

ROW_NUMBER returns for nth position if there is no duplicate in data, where as dense_rank with duplicate as well.

DATEDIFF

Last 7 days:

Select *, `DATEDIFF(day, Employees.hiredate, getdate()) as diff` from table_name where `DATEDIFF(day, Employees.hiredate, getdate())<8`

```
SELECT * FROM TABLE_NAME WHERE CAST(GETDATE() AS DATE) BETWEEN CAST(GETDATE() AS DATE)
AND DATEADD(DAY, -7, CAST(GETDATE() AS DATE))
```

YESTERDAY DATE:

```
SELECT DATEADD(DAY, -1, CAST(GETDATE() AS DATE))
```

Groupy by multiple columns:

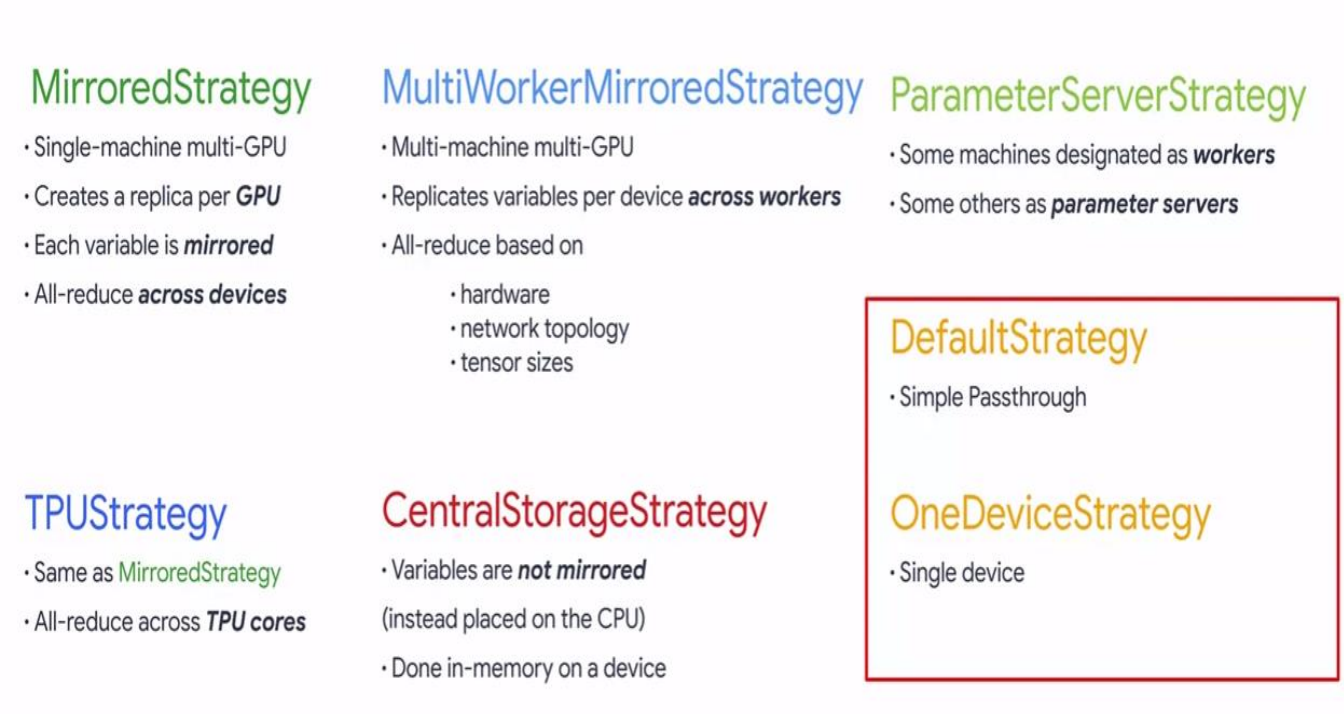
count(distinct l.lead_manager_code)

https://medium.com/jen-li-chen-in-data-science/hackerrank-sql-ff3e70081c4b

Machine Learning theoretical concepts

By: Vikram Pal

Custom Training with Tensorflow:

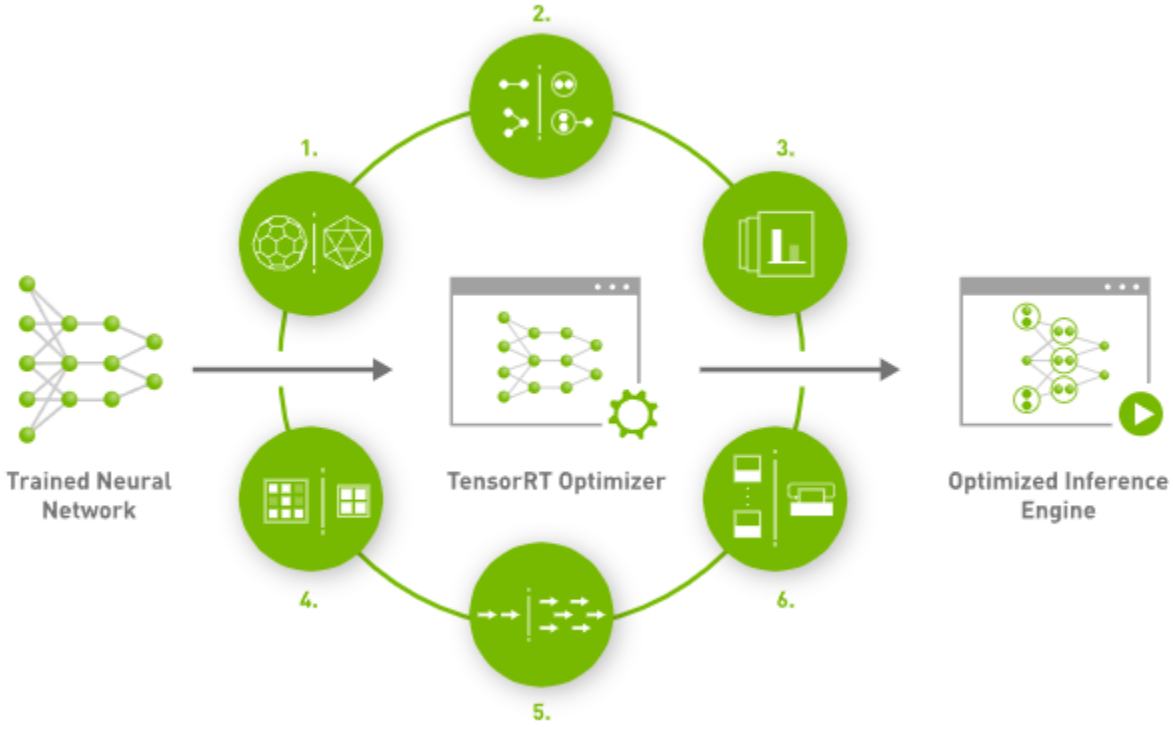


Model Inference

Why use TensorRT?

TensorRT-based applications perform up to 36x faster than CPU-only platforms during inference. It has a low response time of under 7ms and can perform target-specific optimizations. Thus enabling developers to optimize neural network models trained on all major frameworks, such as PyTorch, TensorFlow, ONNX, and Matlab, for faster inference. It can also be integrated with application-specific software development kits such as NVIDIA DeepStream, Riva, Merlin, Maxine, Modulus, Morpheus, and Broadcast Engine.

How does TensorRT perform optimization?



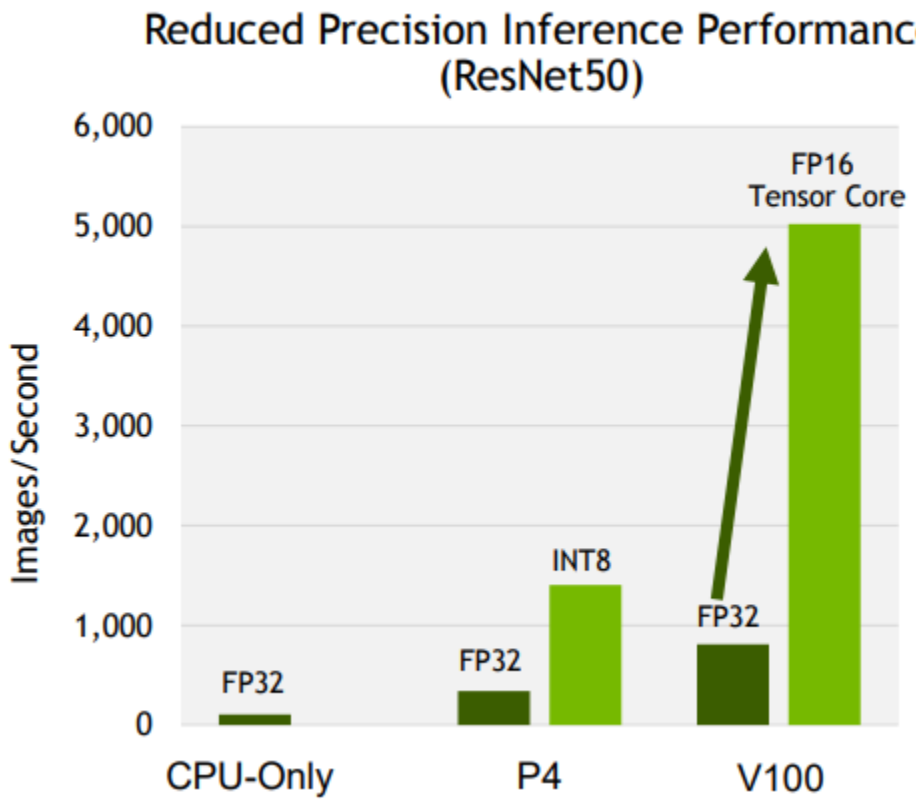
Source: [NVIDIA]

We have read about how TensorRT can help developers optimize, but now we will look at the six processes of TensorRT that can make it work.

Machine Learning theoretical concepts

By: Vikram Pal

1. Weight & Activation Precision Calibration

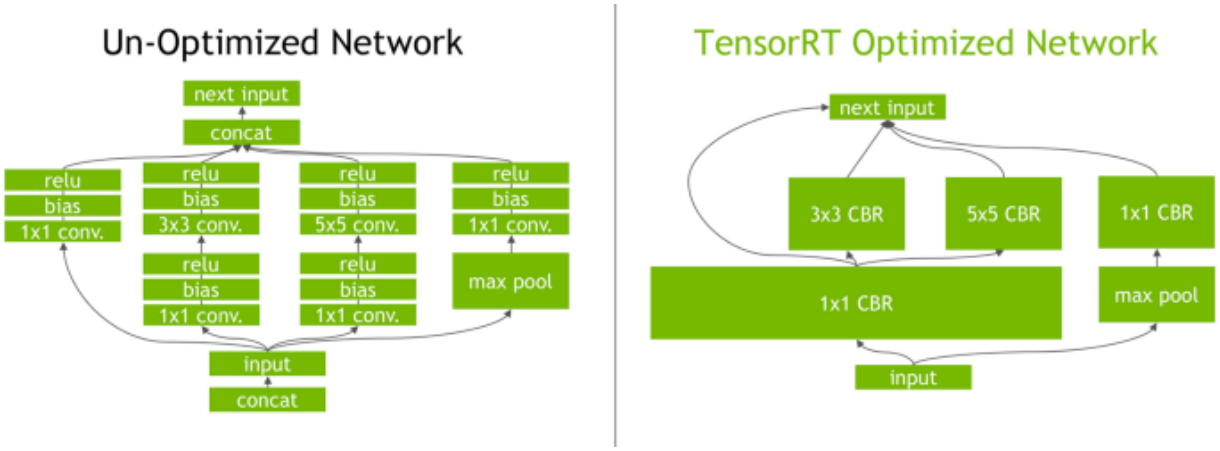


Source: NVIDIA

Nearly all deep learning models are trained in FP32 to take advantage of a wider dynamic range. However, these models require a long predicting time, setting back real-time responses.

In this process, model quantization converts the parameters and activations to FP16 or INT8. This will normally cause lower accuracy and a reduction in latency and model size. But by using KL-divergence, TensorRT is able to measure the difference and minimize it, thereby preserving accuracy while maximizing throughput. We can see the difference between FP32 and INT8/FP16 from the picture above.

2. Layer & Tensor Fusion



Source: NVIDIA

In this process, TensorRT uses layers and tensor fusion to optimize the GPU's memory and bandwidth by fusing nodes in a kernel vertically or horizontally (sometimes both). This reduces the overhead cost of reading and writing the tensor data for each layer.

We can see from the picture above that TensorRT recognizes all layers with similar inputs and filter sizes and merges them to form a single layer. It also eliminates concatenation layers, as seen in the picture above ("concat").

Overall, this will result in a smaller, faster, and more efficient graph with fewer layers and kernel launches, which will reduce inference latency.

3. Kernel Auto-Tuning

During this process, TensorRT selects the best layers, algorithms, and batch size based on the target GPU platform in order to find the best performance. This ensures that the deployed model is tuned for each deployment platform.

4. Dynamic Tensor Memory

For this process, TensorRT minimizes memory footprint and re-uses memory by allocating memory for each tensor only for the duration of its usage, avoiding any memory allocation overhead for faster and more efficient execution.

5. Multi-Stream Execution

TensorRT is designed to process multiple input streams in parallel during this process.

6. Time Fusion

For the last step before heading to the output stage, TensorRT is able to optimize recurrent neural networks over time steps with dynamically generated kernels.