

# K-Nearest Neighbors (KNN)



# K-Nearest Neighbors (KNNs)

- KNNs are one of the most popular machine learning algorithms as it gives somewhat decent performance and is also quite easy to understand and implement.
- It's called a **lazy learner** (as opposed to eager learners) because it doesn't have a training phase, what it does instead is compute its classification using the training data (can be slow). This also makes it a **non-parametric algorithm**.



# The KNN Algorithm – Step by Step

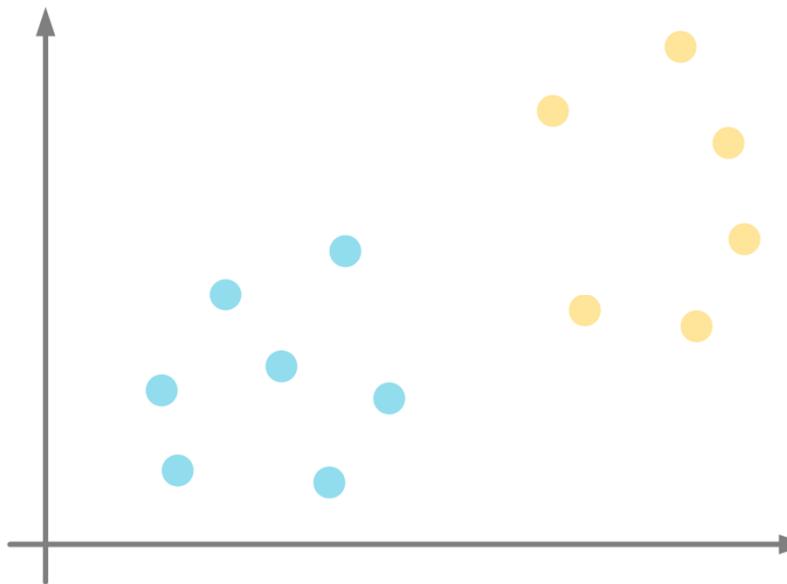
How KNNs classify a new data input:

1. Load all training data
2. Use a pre-set value of k (integer only)
3. Calculate the distance between the test data input and each value of the training data
4. Sort on our distances (ascending i.e. smallest distance to largest distance)
5. Choose the top K rows of our sorted data
6. Assign class to our input data to the most frequent class in our K rows

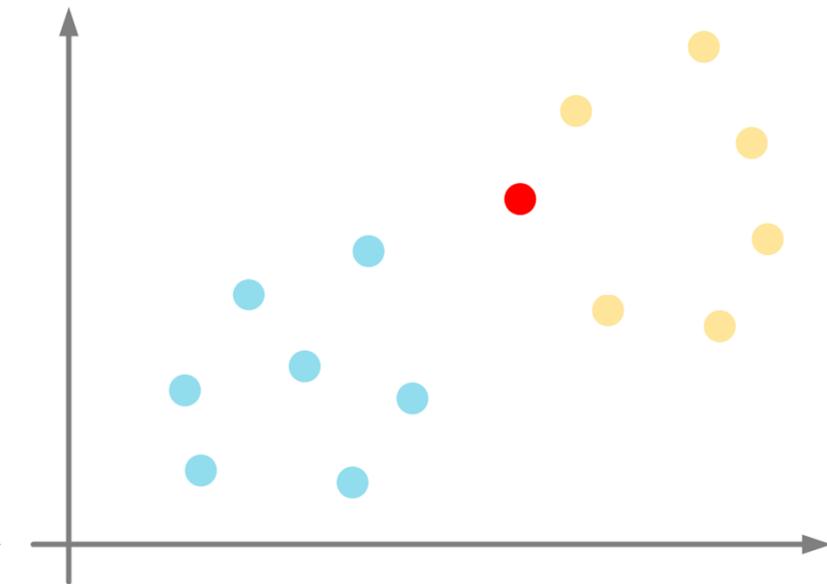


## The KNN Algorithm – Visually

Scatter plot of our training data



Let's classify our new red input

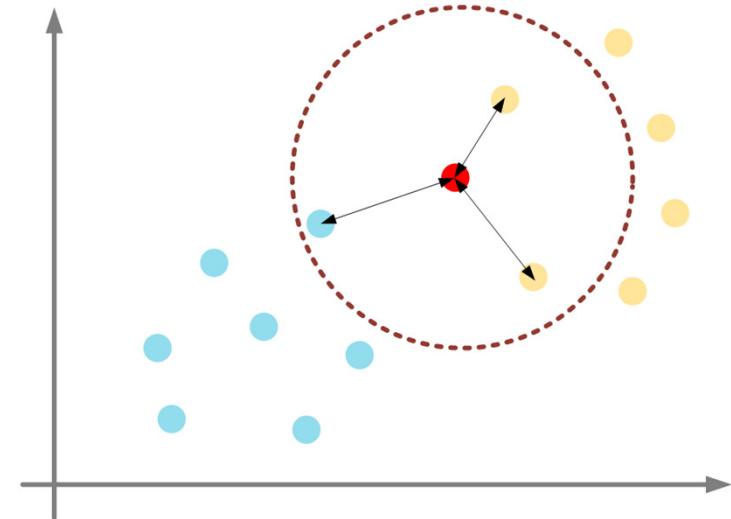




## The KNN Algorithm – Using k = 3

- As K = 3, let's draw a circle enclosing the 3 nearest points to our input (red circle)
- Two out our 3 points were yellow when sorted by distance, therefore the class KNN will assign the red point to will be the yellow class

| Point | Distance | Class  |
|-------|----------|--------|
| 1     | 10       | Yellow |
| 2     | 12       | Yellow |
| 3     | 15       | Blue   |

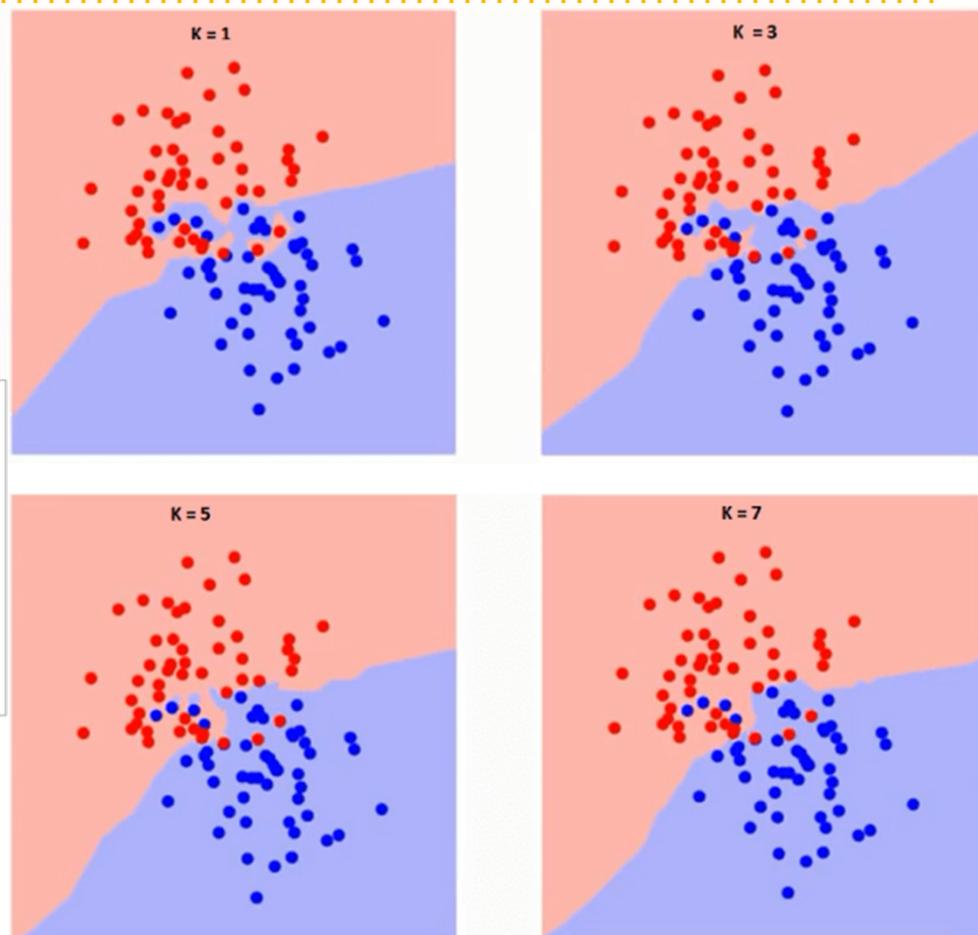
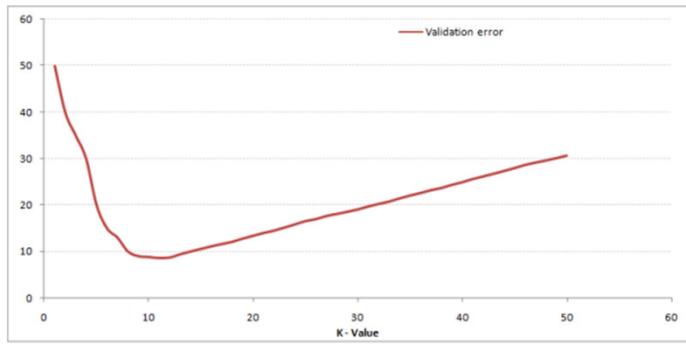




# Choosing K

- As we can see from our last diagram, the larger  $k$ , the more points it considers. But how do we go about knowing which  $k$  to choose?
- If  $k = 1$ , we will only assign classes considering the closest point, this can lead to a model that **overfits**!
- However, as  $k$  goes up, we begin to simplify the model too much thus leading to a model that **is under-fitting** (high bias and low variance)
- Let's visualize this!

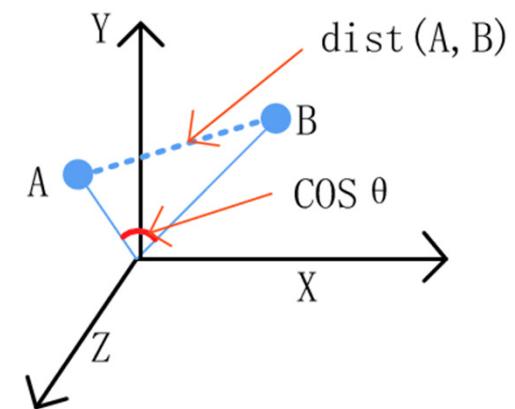
- Notice how our decision boundary becomes smoother as k increases (less overfitting) but only up to a point!





# Disadvantages of KNNs

- As you may have noticed, every time we wish to calculate a new input, we need **to load all our training data**, then calculate the distances to every point! This is exhaustive and negatively impacts performance, not to mention the high memory usage it requires.
- Also of note, for KNN to work well, we need to normalize or scale all the input data so that we can calculate the distances fairly. Common distance metrics used are Euclidean distance or Cosine Distance.
- Datasets with a large number of features (i.e. dimensions) will impact KNN performance. To avoid overfitting we thus need even more data for KNN which isn't always possible. This is known as the Curse of Dimensionality.



## **Assessing Performance – Accuracy, Confusion Matrix, Precision and Recall**



## Assessing Model Accuracy

- Accuracy is simply a measure of how much of our training data did our model classify correctly

$$\bullet \quad Accuracy = \frac{\text{Correct Classifications}}{\text{Total Number of Classifications}}$$



## Is Accuracy the only way to assess a model's performance?

- While very important, accuracy alone doesn't tell us the whole story.
- Imagine we're using a Model to predict whether a person has a life threatening disease based on a blood test.
- There are now 4 possible scenarios.
  1. **TRUE POSITIVE** - Test Predicts Positive for the disease and the person has the disease
  2. **TRUE NEGATIVE** - Test Predicts Negative for the disease and the person does NOT have the disease
  3. **FALSE POSITIVE** - Test Predicts Positive for the disease but the person does NOT have the disease
  4. **FALSE NEGATIVE** - Test Predicts Negative for the disease but the person actually has the disease



## For a 2 or Binary Class Classification Problem

- **Recall** – How much of the positive classes did we get correct

- $\circ \quad Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$

- **Precision** – When predicting positive, how many of our positive predictions were right?

- $\circ \quad Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$

- **F-Score** – Is a metric that attempts to measure both Recall & Precision

- $\circ \quad F - Score = \frac{2 \times Recall \times Precision}{Precision + Recall}$



## Confusion Matrix Real Example

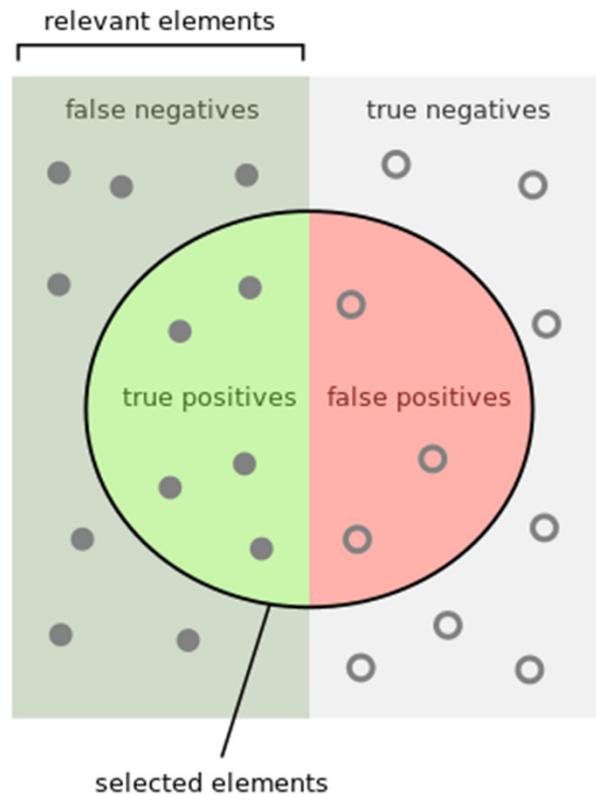
Let's say we've built a classifier to identify Male Faces in an image, there are:

- 10 Male Faces & 5 Female Faces in the image
- Our classifier identifies 6 male faces
- Out of the 6 male faces - 4 were male and 2 female.

$$Precision = \frac{TP}{TP + FP}$$

|                    | Male   | Not Male |
|--------------------|--------|----------|
| Predicted Male     | TP = 4 | FP = 2   |
| Predicted Not Male | FN = 6 | TN = 3   |

- Our Recall is  $4 / (4 + 6)$  (6 male faces were missed) or 0.4 or 40%
- Our Precision is  $4 / (4 + 2)$  or 0.66 or 66%
- Our F-Score is  $\frac{2 \times Recall \times Precision}{Precision + Recall} = \frac{2 \times 0.4 \times 0.66}{0.4 + 0.66} = \frac{0.528}{1.06} = 0.498$



How many selected items are relevant?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

[https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)



## Confusion Matrix for Multiple Classes

- We can use scikit-learn to generate our confusion matrix.
- Let's analyze our results on our MNIST dataset

```
[ [ 977    0    0    0    0    0    1    1    1    0]
  [  0 1130    3    1    0    0    1    0    0    0]
  [  1    0 1029    0    1    0    0    1    0    0]
  [  0    0    4 1003    0    1    0    1    1    0]
  [  0    0    0    0  976    0    0    0    2    4]
  [  3    0    0    5    0   881    3    0    0    0]
  [  6    2    0    0    1    1   948    0    0    0]
  [  1    2   11    2    0    0    0 1010    1    1]
  [  4    0    3    0    0    0    0    2   963    2]
  [  1    2    0    1    5    3    0    2    5 990]]
```



## Confusion Matrix for Multiple Classes

| Predicted | 0   | 1    | 2    | 3    | 4   | 5   | 6   | 7    | 8   | 9   | True Values |
|-----------|-----|------|------|------|-----|-----|-----|------|-----|-----|-------------|
| [         | 977 | 0    | 0    | 0    | 0   | 0   | 1   | 1    | 1   | 0   | 0           |
| [         | 0   | 1130 | 3    | 1    | 0   | 0   | 1   | 0    | 0   | 0   | 1           |
| [         | 1   | 0    | 1029 | 0    | 1   | 0   | 0   | 1    | 0   | 0   | 2           |
| [         | 0   | 0    | 4    | 1003 | 0   | 1   | 0   | 1    | 1   | 0   | 3           |
| [         | 0   | 0    | 0    | 0    | 976 | 0   | 0   | 0    | 2   | 4   | 4           |
| [         | 3   | 0    | 0    | 5    | 0   | 881 | 3   | 0    | 0   | 0   | 5           |
| [         | 6   | 2    | 0    | 0    | 1   | 1   | 948 | 0    | 0   | 0   | 6           |
| [         | 1   | 2    | 11   | 2    | 0   | 0   | 0   | 1010 | 1   | 1   | 7           |
| [         | 4   | 0    | 3    | 0    | 0   | 0   | 0   | 2    | 963 | 2   | 8           |
| [         | 1   | 2    | 0    | 1    | 5   | 3   | 0   | 2    | 5   | 990 | 9           |



## Confusion Matrix Analysis

- Classifying 7s as 2s, 6s as 0s, 9s as 4s and 9s as 8s.

| Predicted | 0   | 1    | 2    | 3    | 4   | 5   | 6   | 7    | 8   | 9   | True Values |
|-----------|-----|------|------|------|-----|-----|-----|------|-----|-----|-------------|
| [         | 977 | 0    | 0    | 0    | 0   | 0   | 1   | 1    | 1   | 0   | 0           |
| [         | 0   | 1130 | 3    | 1    | 0   | 0   | 1   | 0    | 0   | 0   | 1           |
| [         | 1   | 0    | 1029 | 0    | 1   | 0   | 0   | 1    | 0   | 0   | 2           |
| [         | 0   | 0    | 4    | 1003 | 0   | 1   | 0   | 1    | 1   | 0   | 3           |
| [         | 0   | 0    | 0    | 0    | 976 | 0   | 0   | 0    | 2   | 4   | 4           |
| [         | 3   | 0    | 0    | 5    | 0   | 881 | 3   | 0    | 0   | 0   | 5           |
| [         | 6   | 2    | 0    | 0    | 1   | 1   | 948 | 0    | 0   | 0   | 6           |
| [         | 1   | 2    | 11   | 2    | 0   | 0   | 0   | 1010 | 1   | 1   | 7           |
| [         | 4   | 0    | 3    | 0    | 0   | 0   | 0   | 2    | 963 | 2   | 8           |
| [         | 1   | 2    | 0    | 1    | 5   | 3   | 0   | 2    | 5   | 990 | 9           |

## Recall



- **Recall** – Actual true positives over how many times the classifier predicted that class
- Let's look at the number 7:
  - TP = 1010 and FN = 18
  - $1010 / 1028 = 98.24\%$

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

| 0   | 1    | 2    | 3    | 4   | 5   | 6   | 7    | 8   | 9   |   |
|-----|------|------|------|-----|-----|-----|------|-----|-----|---|
| 977 | 0    | 0    | 0    | 0   | 0   | 1   | 1    | 1   | 0   | 0 |
| 0   | 1130 | 3    | 1    | 0   | 0   | 1   | 0    | 0   | 0   | 1 |
| 1   | 0    | 1029 | 0    | 1   | 0   | 0   | 1    | 0   | 0   | 2 |
| 0   | 0    | 4    | 1003 | 0   | 1   | 0   | 1    | 1   | 0   | 3 |
| 0   | 0    | 0    | 0    | 976 | 0   | 0   | 0    | 2   | 4   | 4 |
| 3   | 0    | 0    | 5    | 0   | 881 | 3   | 0    | 0   | 0   | 5 |
| 6   | 2    | 0    | 0    | 1   | 1   | 948 | 0    | 0   | 0   | 6 |
| 1   | 2    | 11   | 2    | 0   | 0   | 0   | 1010 | 1   | 1   | 7 |
| 4   | 0    | 3    | 0    | 0   | 0   | 0   | 2    | 963 | 2   | 8 |
| 1   | 2    | 0    | 1    | 5   | 3   | 0   | 2    | 5   | 990 | 9 |

## Precision



- **Precision** – Number of correct predictions over how many occurrences of that class were in the test dataset.
- Let's look at the number 7:
  - TP = 1010 and FP = 7
  - $1010 / 1017 = 99.31\%$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

| 0   | 1    | 2    | 3    | 4   | 5   | 6   | 7    | 8   | 9   |   |
|-----|------|------|------|-----|-----|-----|------|-----|-----|---|
| 977 | 0    | 0    | 0    | 0   | 0   | 1   | 1    | 1   | 0   | 0 |
| 0   | 1130 | 3    | 1    | 0   | 0   | 1   | 0    | 0   | 0   | 1 |
| 1   | 0    | 1029 | 0    | 1   | 0   | 0   | 1    | 0   | 0   | 2 |
| 0   | 0    | 4    | 1003 | 0   | 1   | 0   | 1    | 1   | 0   | 3 |
| 0   | 0    | 0    | 0    | 976 | 0   | 0   | 0    | 2   | 4   | 4 |
| 3   | 0    | 0    | 5    | 0   | 881 | 3   | 0    | 0   | 0   | 5 |
| 6   | 2    | 0    | 0    | 1   | 1   | 948 | 0    | 0   | 0   | 6 |
| 1   | 2    | 11   | 2    | 0   | 0   | 0   | 1010 | 1   | 1   | 7 |
| 4   | 0    | 3    | 0    | 0   | 0   | 0   | 2    | 963 | 2   | 8 |
| 1   | 2    | 0    | 1    | 5   | 3   | 0   | 2    | 5   | 990 | 9 |



# Classification Report

Using scikit-learn we can automatically generate a **Classification Report** that gives us **Recall, Precision, F1 and Support**.

|   | precision | recall | f1-score | support |
|---|-----------|--------|----------|---------|
| 0 | 0.98      | 1.00   | 0.99     | 980     |
| 1 | 0.99      | 1.00   | 1.00     | 1135    |
| 2 | 0.98      | 1.00   | 0.99     | 1032    |
| 3 | 0.99      | 0.99   | 0.99     | 1010    |
| 4 | 0.99      | 0.99   | 0.99     | 982     |
| 5 | 0.99      | 0.99   | 0.99     | 892     |
| 6 | 0.99      | 0.99   | 0.99     | 958     |
| 7 | 0.99      | 0.98   | 0.99     | 1028    |
| 8 | 0.99      | 0.99   | 0.99     | 974     |
| 9 | 0.99      | 0.98   | 0.99     | 1009    |



# F1-Score and Support

- **F1 Score** is the weighted average of Precision and Recall. As a result, this score takes both false positives and false negatives into account. It is useful if you have an **uneven class distribution** as accuracy works best if false positives and false negatives have similar cost.
  - The f1-scores corresponding to every class will tell you the accuracy of the classifier in classifying the data points in that particular class compared to all other classes
- The **support** is the number of samples of the true response that lie in that class.



## More on Recall vs. Precision

- **High recall (or sensitivity) with low precision.**
  - This tells us that most of the positive examples are correctly recognized (low False Negatives) but there are a lot of false positives i.e. other classes being predicted as our class in question.
- **Low recall (or sensitivity) with high precision.**
  - Our classifier is missing a lot of positive examples (high FN) but those we predict as positive are indeed positive (low False Positives)



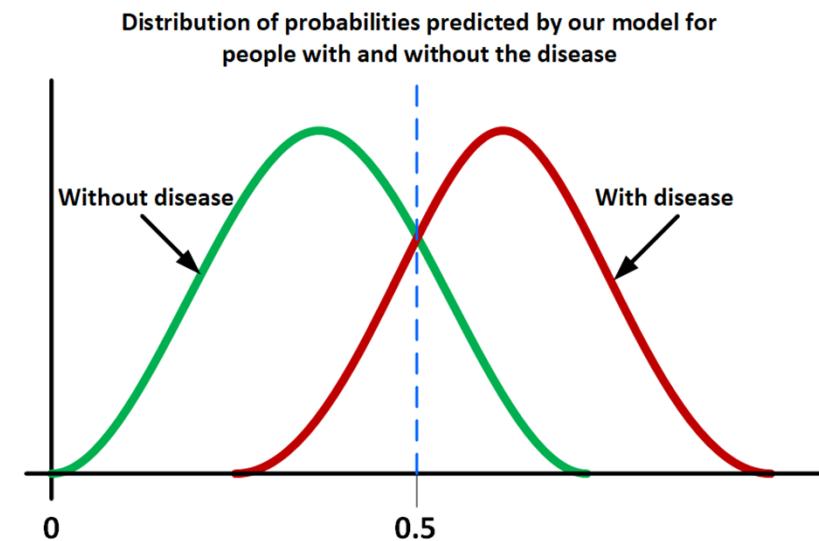
**Let's compare the  
performance between  
Logistic Regressions, SVMs  
and KNN Classifiers in Python**

# Understanding the ROC and AUC Curve

# ROC (Receiver Operating Characteristic)



- The ROC curve is very important as it tells us how good our model is at **distinguishing between two classes**.
- Imagine we have a model that predicts whether someone has a disease or not.
- In the diagram on the right we use 0.5 as our threshold to identify someone as having the disease.
- However, look at the area around 0.5, there can be a lot of ambiguity if a sample point lies in that region.





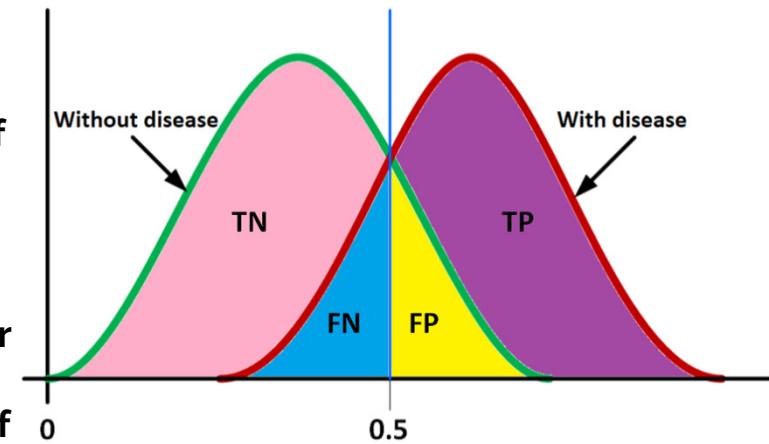
## Remember our Recall and Precision

- **Recall** - in this example, Recall would be the number of patients who our model identified as having the disease (TP) over the total number of patients that actually have the disease.

$$\circ \quad \text{Recall} = \frac{TP}{TP+FN}$$

- **Precision** - in this example would be the number of patients who our model identified as NOT having the disease (TN) over the total number of patients that actually did NOT have the disease.

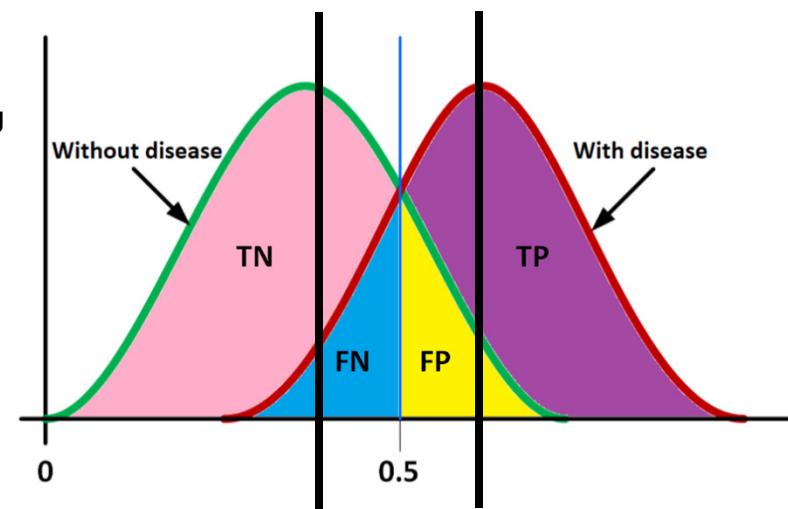
$$\circ \quad \text{Precision} = \frac{TN}{TN+FP}$$





## How Thresholds affect Recall and Precision

- Let's make a **lower threshold** moving it from **0.5 to 0.4**.
  - We get **more positive** predictions of a patient having the disease
  - Increase in FP
  - Decrease in FN
- This will **decrease the Precision** and **increase Recall**
- If we did the reverse and increased it **to 0.6** we get:
  - More negative predictions (i.e. less classifications saying we found the disease)
  - Decrease in FP
  - Increase in FN
- This will **increase the Precision** and **reduce Recall**.

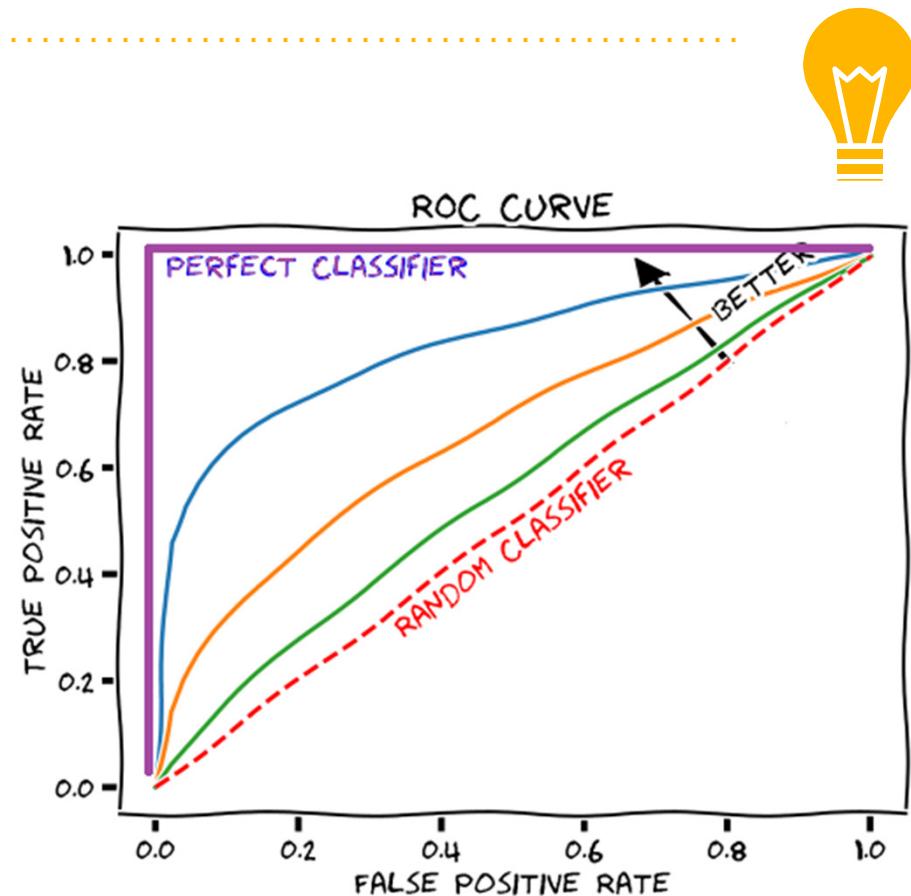


As Recall decreases Precision increases

As Precision decreases Recall increases

# The ROC Curve

- The ROC Curve is the plot of **Recall** or our **True Positive Rate (TPR)** against (**1-Precision**) or our **False Positive Rate (FPR)**.
- **TPR** is the proportion of people with the disease that were correctly identified by our model.
- The **FPR** is the proportion of people identified by our model to not have the disease that were false positives.
- The **AUC** or **Area under the Curve** is an overall measure of performance, the greater area the better the classifier.

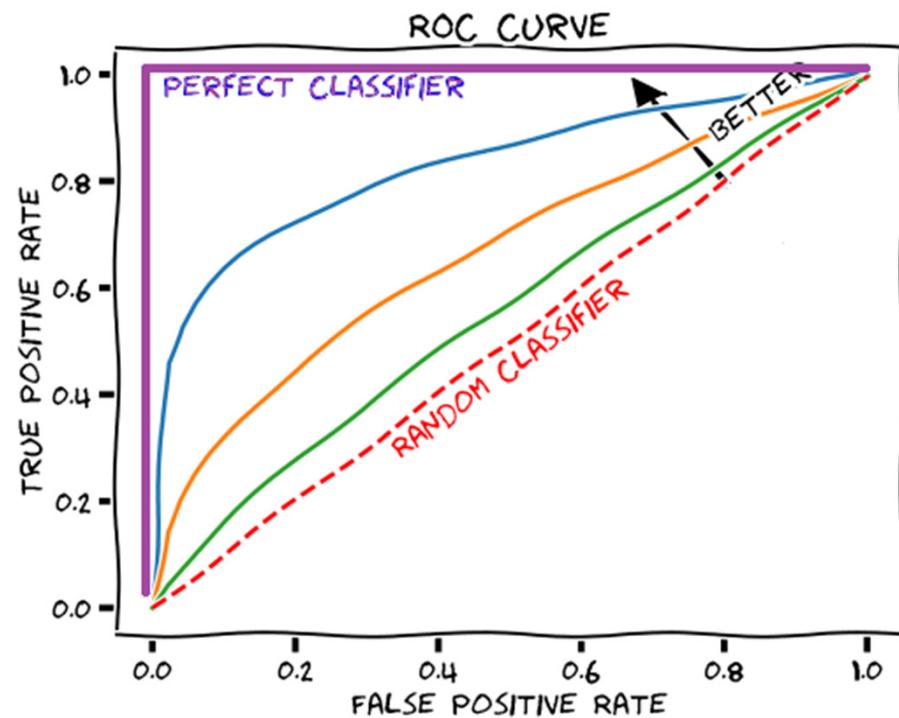


Source: <https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/>

# Creating the ROC Curve



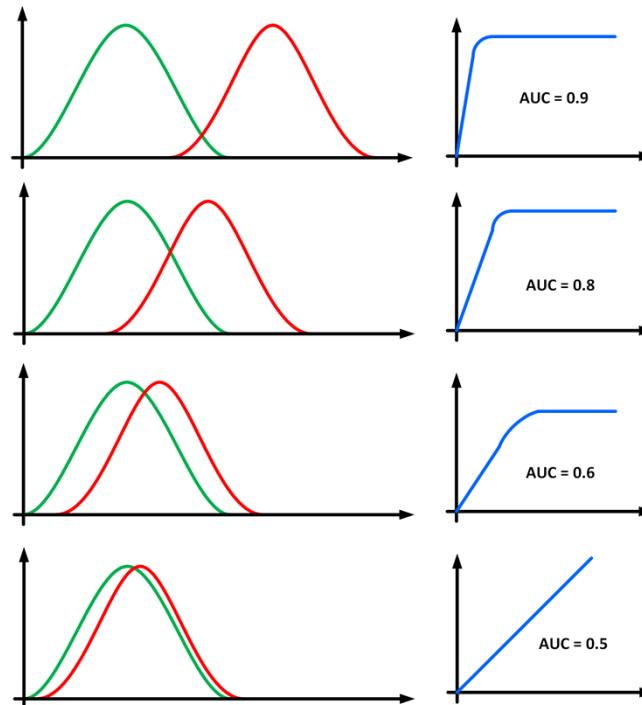
- The Area under the ROC Curve is very important when comparing models. Generally the model with the greater AUC is better.
- However, how is this curve generated?
- We simply use different thresholds in our classifier to get our TPR and FPR values and plot them on a graph, connecting the lines to gives us this curve.



Source: <https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/>



## How we use the ROC Curve and AUC



# **Overfitting – Regularization, Generalization and Outliers**



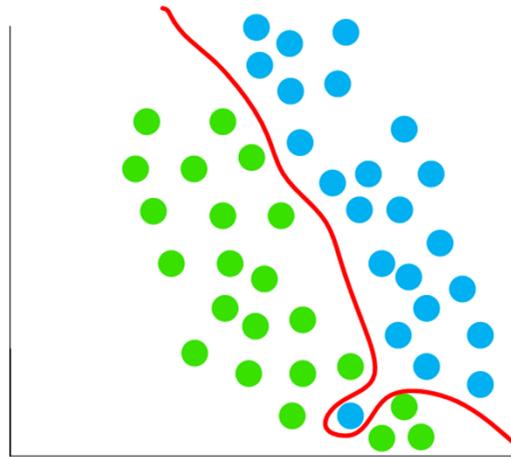
# What Makes a Good Model?

- A good model is accurate
- Generalizes well
- Does not overfit
- But what do these things mean?

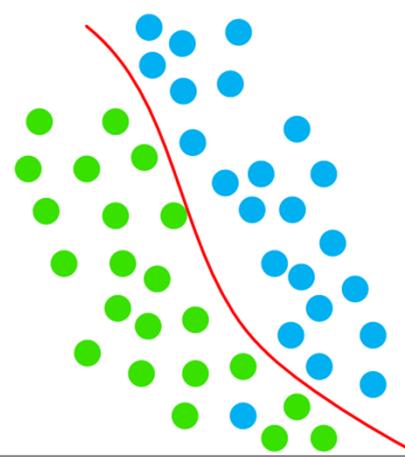
## Examine these Models and the Decision Boundary



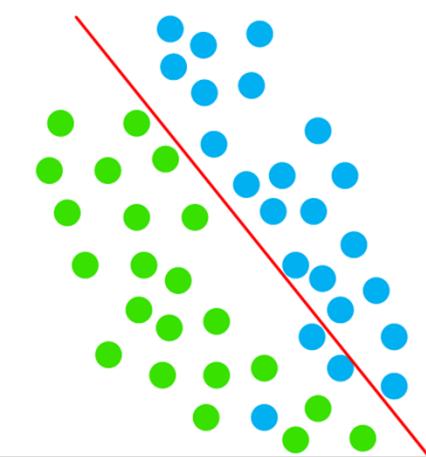
Model A



Model B



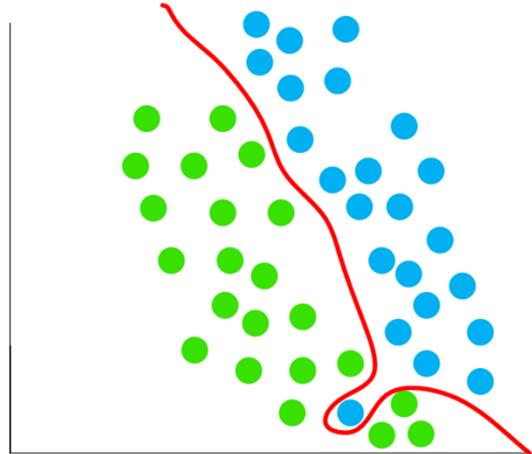
Model C



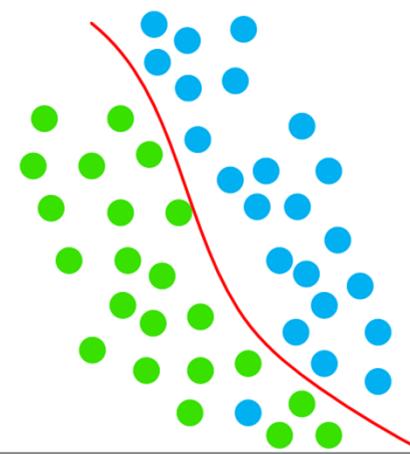


## Let's look at Each

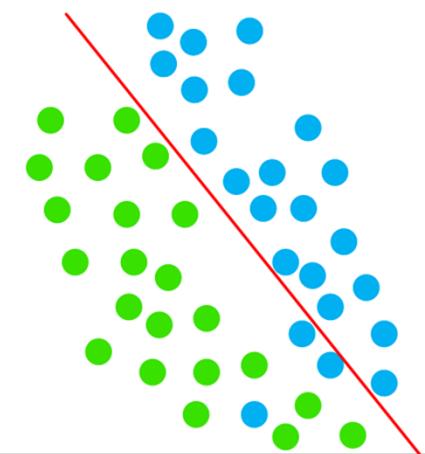
- Overfitting



- Ideal or Balanced



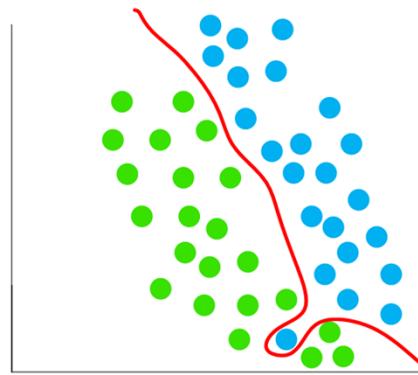
- Underfitting





# Overfitting and Underfitting

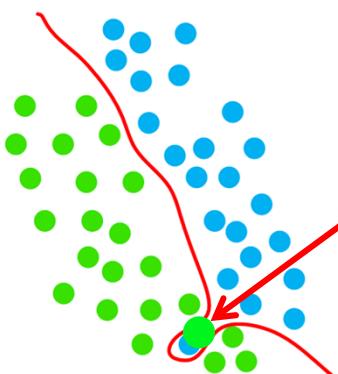
- Overfitting occurs when a statistical model or machine learning algorithm captures the noise of the data. Overfitting occurs if the model or algorithm shows **low bias** but **high variance**
- Underfitting occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data





# Overfitting

- Overfitting leads to poor models and is one of the most common problems faced developing in AI/Machine Learning/Neural Nets.
- Overfitting occurs when our Model fits near perfectly to our training data, as we saw in the previous slide with Model A. However, fitting too closely to training data isn't always a good thing.

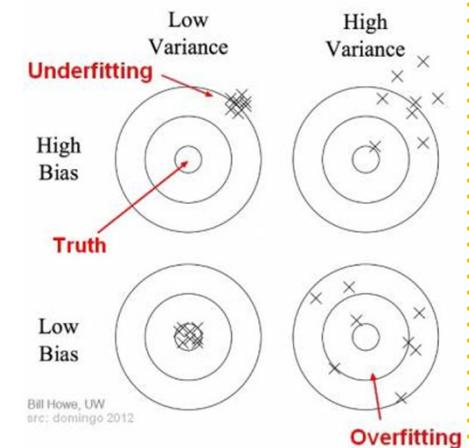


- What happens if we try to classify a brand new point that occurs at the position shown on the left? (who's true color is green)
- It will be misclassified because our model has overfit the test data
- Models don't need to be complex to be good



## Overfitting – Bias and Variance

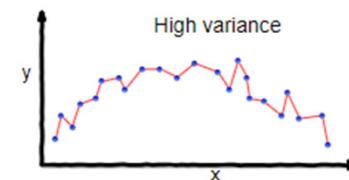
- **Bias Error** – Bias is the difference between the average prediction and the true value we're attempting to predict. Bias error results from simplifying assumptions made by a model to make the target function easier to learn. Common low bias models are Decision Trees, KNN and SVMs. **Parametric** models like Linear Regression and Logistic Regression have high-bias (i.e. more assumptions). They're often fast to train, but less flexible.
- **Variance Error** – Variance error originates from how much would your target model change if different training data were used. Normally, we do expect some variance in models for different training data, however, the underlying model should be generally the same. High variance models (Decision Trees, KNN and SVMs) are very sensitive to this whereas low variance models (Linear Regression and Logistic Regression) are not.
- Parametric or linear models have high bias and low variance
- Non-parametric or non-linear have low bias and high variance



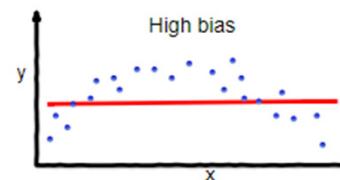


# The Bias and Variance Tradeoff

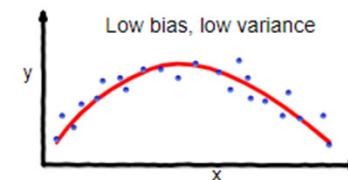
- **Trade-Off** – We want both low bias and low variance, however as we increase bias we decrease variance and vice versa. But how does this happen?
- In linear regression, if we increase the degrees in our polynomial, we are lowering the bias but increasing the variance, conversely if we reduce the complexity we increase bias but reduce variance
- In K-nearest neighbors, increasing the number of clusters ( $k$ ) increases bias but reduces variance



overfitting



underfitting



Good balance



# How do we know if we've Overfit?

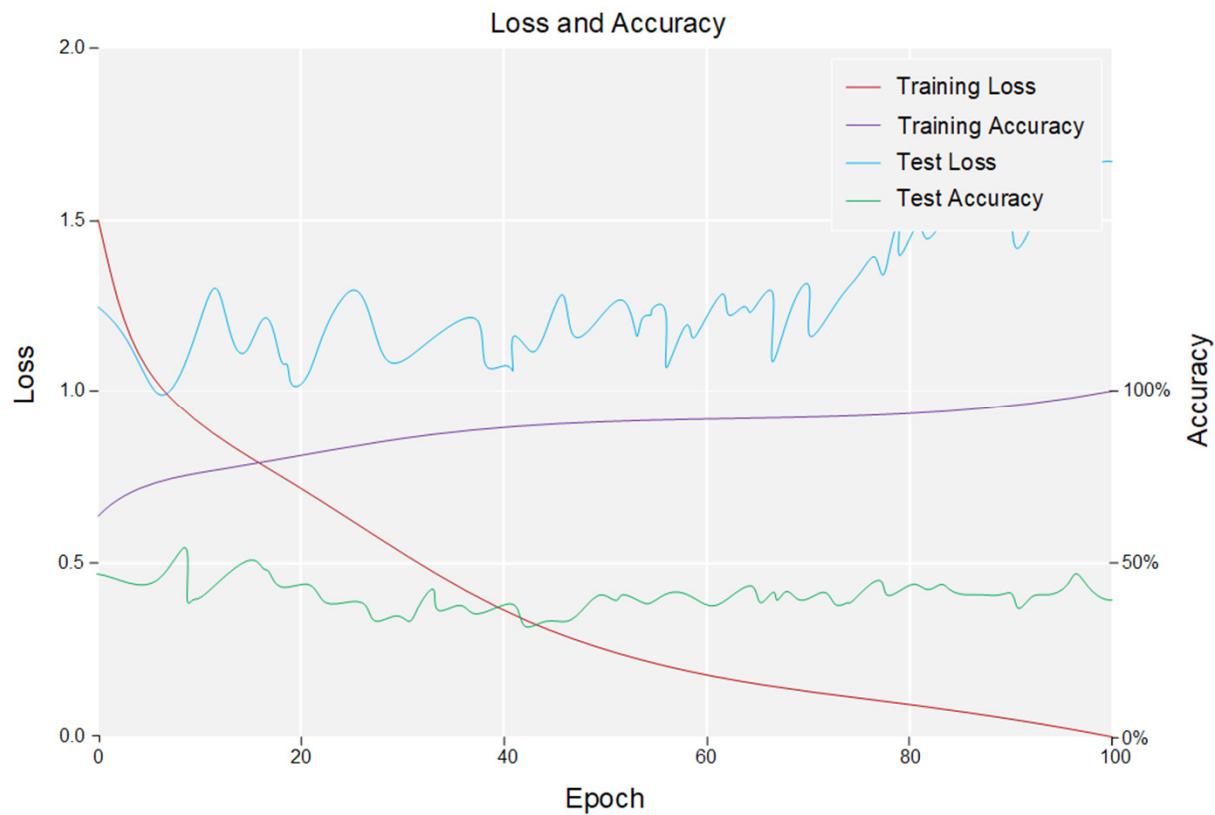
## Test on your model on.....Test Data!

- In all Machine Learning it is extremely important we hold back a portion of our data (10-30%) as pure untouched test data.



- Untouched meaning that this data is NEVER seen by the training algorithm. It is used purely to test the performance of our model to assess its accuracy in classifying new never before seen data.
- Many times when Overfitting we can achieve high accuracy 95%+ on our **training data**, but then get abysmal (~70%) results on the test data. That is a perfect example of Overfitting.

## Overfitting Illustrated Graphically

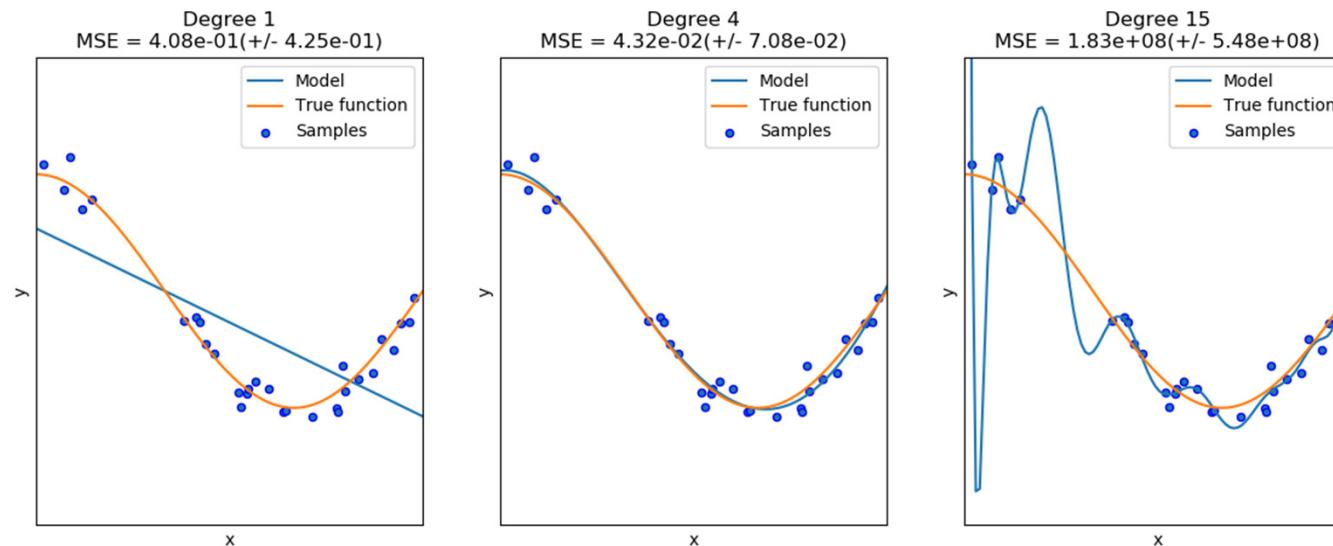


- Examine our Training Loss and Accuracy. They're both heading the right directions!
- But, what's happening to the loss and accuracy on our Test data?
- This is classic example of Overfitting to our training data



# How do we avoid overfitting?

- Rule of thumb is to reduce the complexity of your model





# How do we avoid overfitting?

- Why do less complex models overfit less?
- Overly complex (i.e. less degrees in the polynomial or in Deep Learning, shallower networks) can sometimes find features or interpret noise to be important in data, due to their abilities to memorize more features (called memorization capacity)

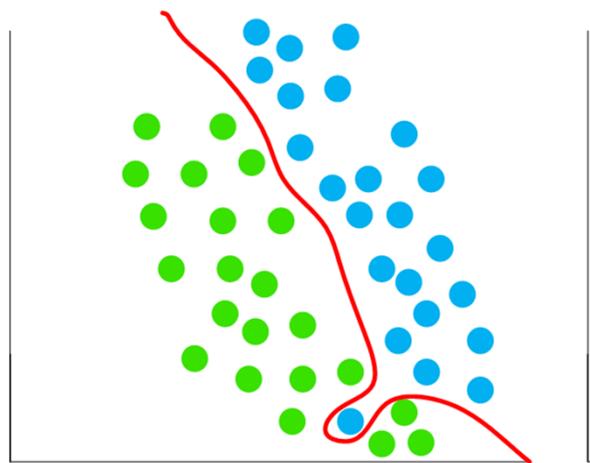
Another method is to use **Regularization!**

- It is better practice to regularize than reduce our model complexity.

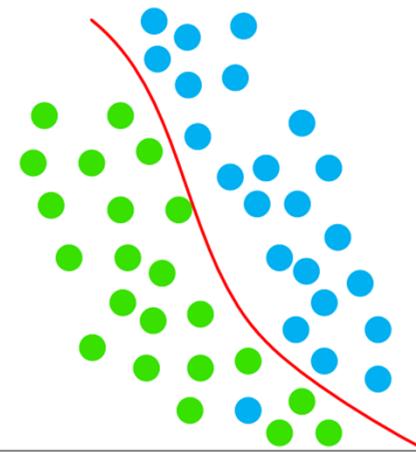


# What is Regularization?

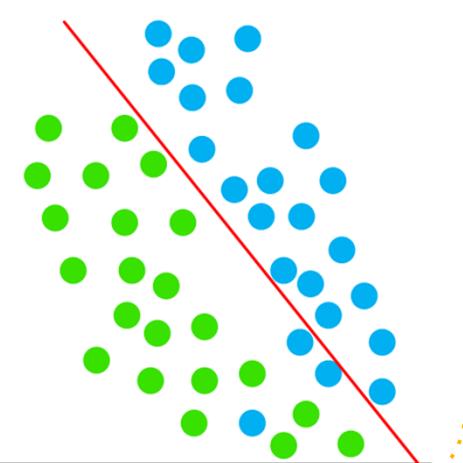
- It is a method of making our model more general to our dataset.
- Overfitting



- Ideal or Balanced



- Underfitting





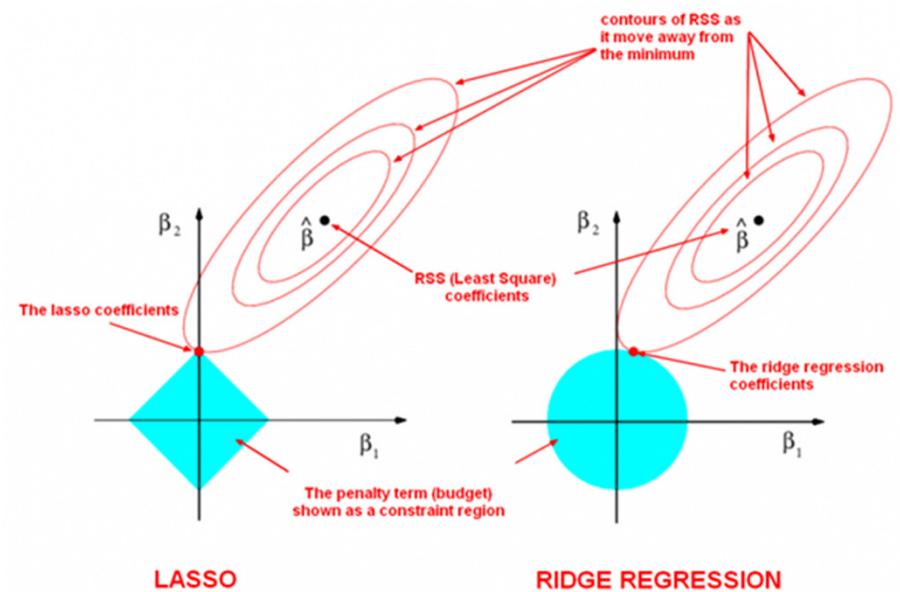
# Types of Regularization

- In most Machine Learning Algorithms we can use:
  - L1 & L2 Regularization
  - Cross Validation
- In Deep Learning we can use:
  - Early Stopping
  - Drop Out
  - Dataset Augmentation

# L1 And L2 Regularization

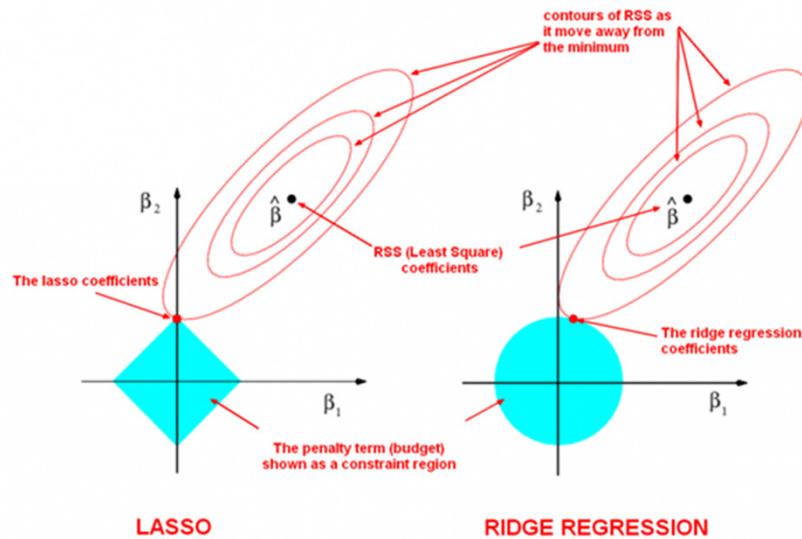


- L1 & L2 regularization are techniques we use to penalize large weights. Large weights or gradients manifest as abrupt changes in our model's decision boundary. By penalizing, we are really making them smaller.
- L2 also known as Ridge Regression
  - $Error = \frac{1}{2}(target_{01} - out_1)^2 + \frac{\lambda}{2} \sum w_i^2$
- L1 also known as Lasso Regression
  - $Error = \frac{1}{2}(target_{01} - out_1)^2 + \frac{\lambda}{2} \sum |w_i|$
- $\lambda$  controls the degree of penalty we apply.
- The difference between them is that L1 brings the weights of the unimportant features to 0, thus acting as feature selection algorithm (known as sparse models or models with reduced parameters.)





# L1 And L2 Regularization



On the left: LASSO regression (you can see that the coefficients, represented by the red rings, can equal zero when they cross the y-axis). On the right: Ridge regression (you can see that the coefficients approach, but never equal zero, because they never cross the y-axis).

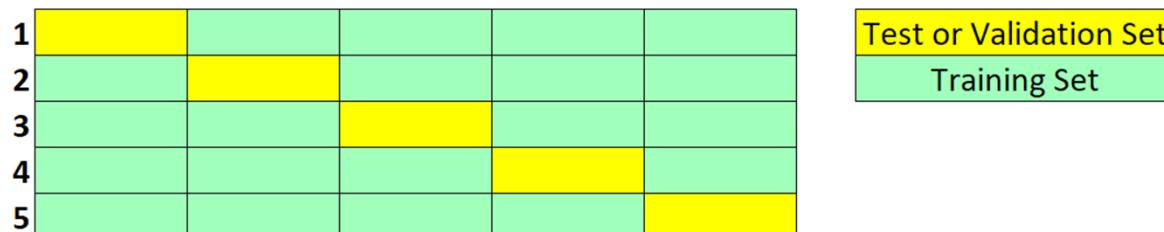
Meta-credit: "[Regularization in Machine Learning](#)" by [Prashant Gupta](#)



# Cross Validation

- Cross validation or also known as k-fold cross validation is a method of training where we split our dataset into k folds instead of a typical training and test split.
- For example, let's say we're using 5 folds. We train on 4, and use the 5th final fold as our test. We then train on the other 4 folds, and test on another.
- We then use the average weights across coming out of each cycle.
- Cross Validation reduces overfitting but slows the training process

k-folds (5 shown)



# **Introduction to Neural Netorks**

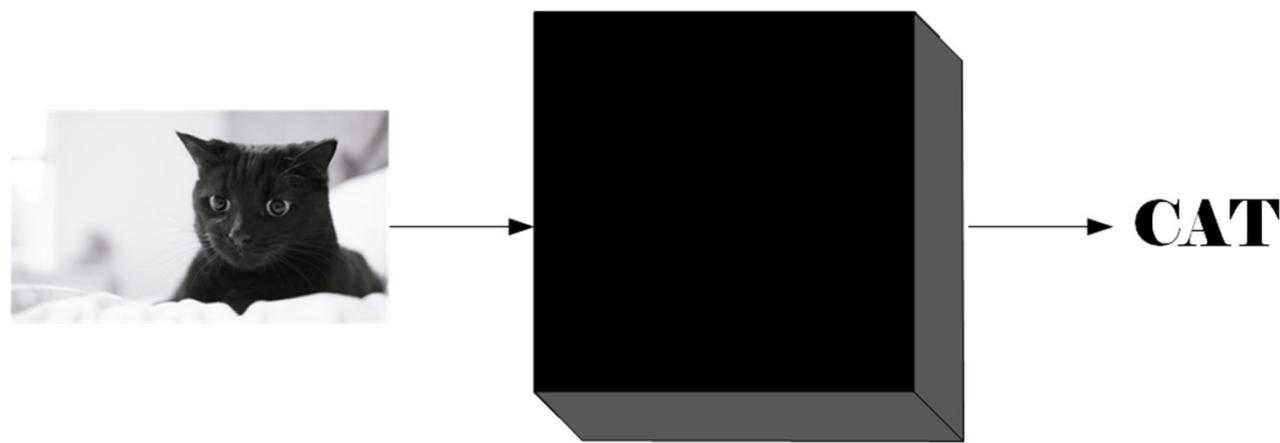


# What are Neural Networks

- Neural Networks act as a '**black box**' or brain that takes inputs and predicts an output.
- It's different and 'better' than most traditional Machine Learning algorithms because it **learns complex non-linear mappings** to produce far more accurate output classification results.

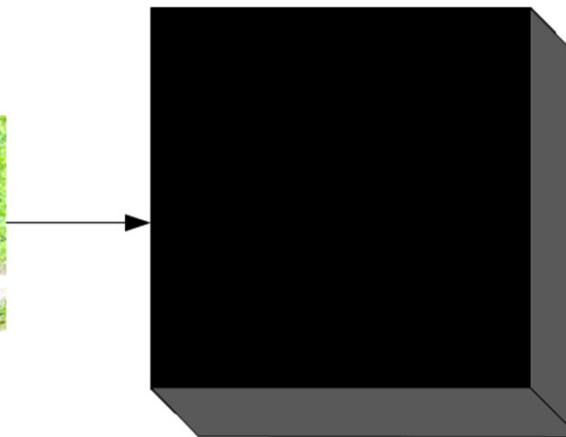


# The Mysterious 'Black Box' Brain





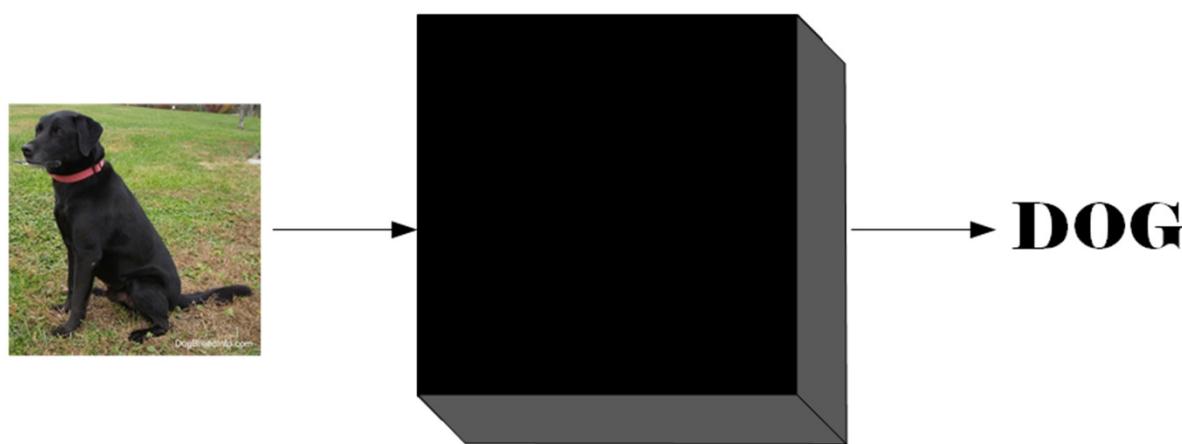
# The Mysterious 'Black Box' Brain



**GORILLA**



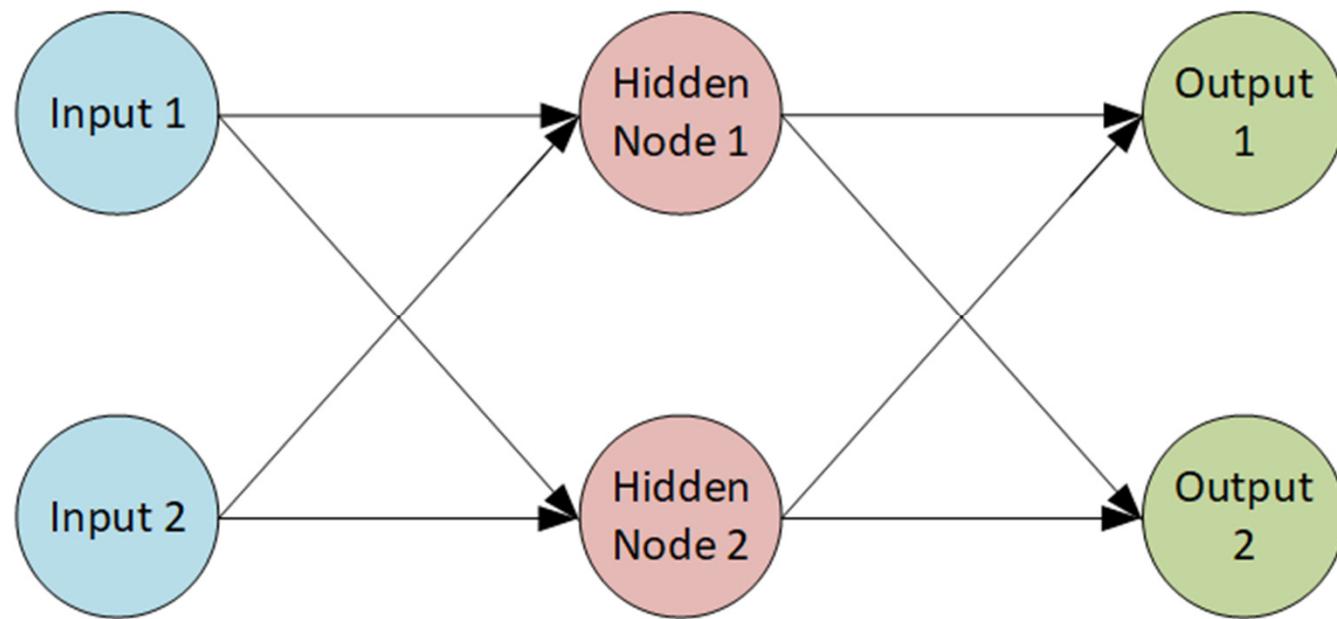
# The Mysterious 'Black Box' Brain





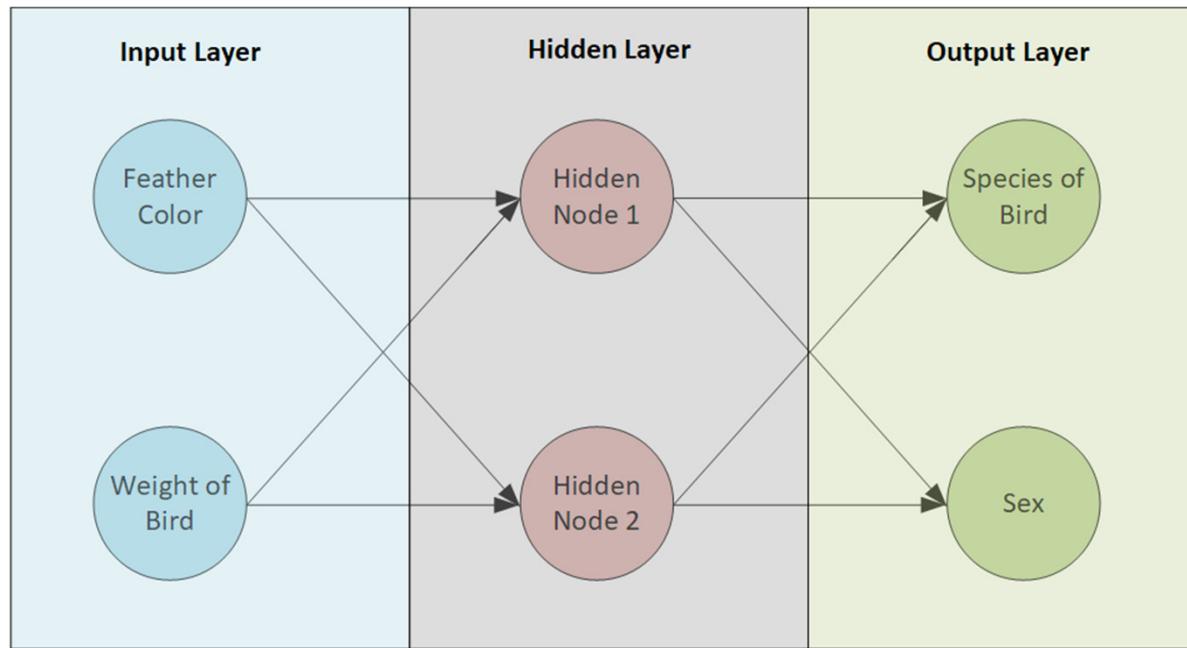
## How NNs 'look'

A Simple Neural Network with 1 hidden layer





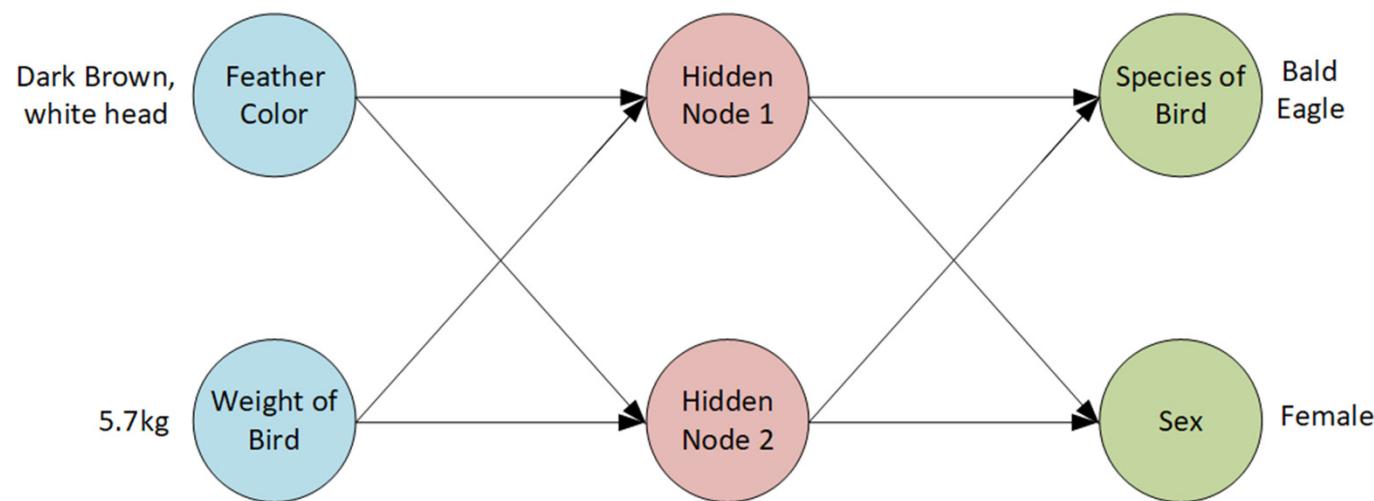
# Example Neural Network





# How do we get a prediction?

Pass the inputs into our NN to receive an output





# How we predict example 2

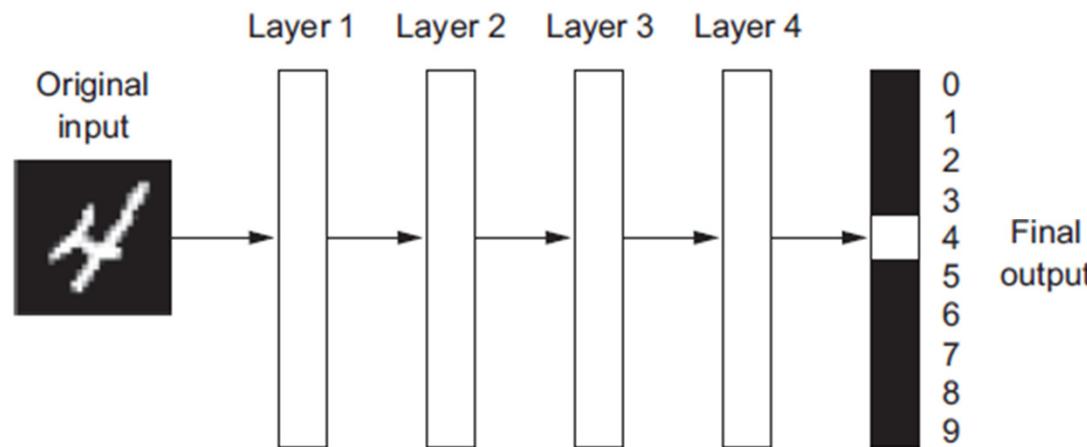


Figure 1.5 A deep neural network for digit classification

# **Types of Deep Learning Models – Feed Forward, CNNs, RNNs & LSTMs**



## Deep Learning has Spawned Dozens of Model Types

- With the advent of Deep Learning algorithms obtaining incredible performance, tweaking and designing intricate elements to these Neural Networks has spawned dozens and dozens different models.
- However, despite the dozens of variations they mostly fall into the following categories:
  - Feed Forward Neural Networks
  - Convolutional Neural Networks (**CNN**)
  - Recurrent Neural Networks (**RNN**)
  - Long Short Term Memory Networks (**LSTM**)

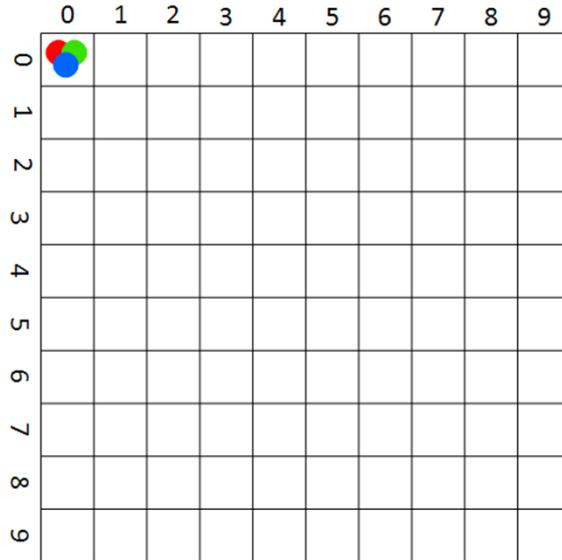


## **Convolutional Neural Networks (CNNs) – Why are they needed?**

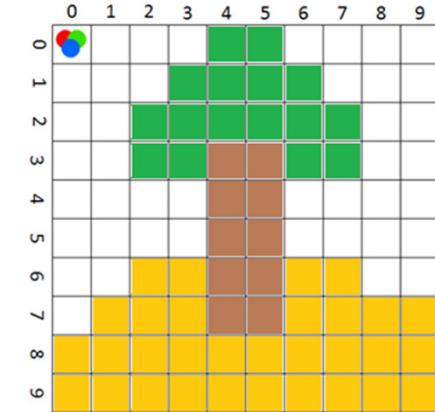
- Because Neural Networks don't scale well to image data



# How do Computers Store Images?



- Each pixel coordinate ( $x, y$ ) contains 3 values ranging for intensities of 0 to 255 (8-bit).
- Red
- Green
- Blue

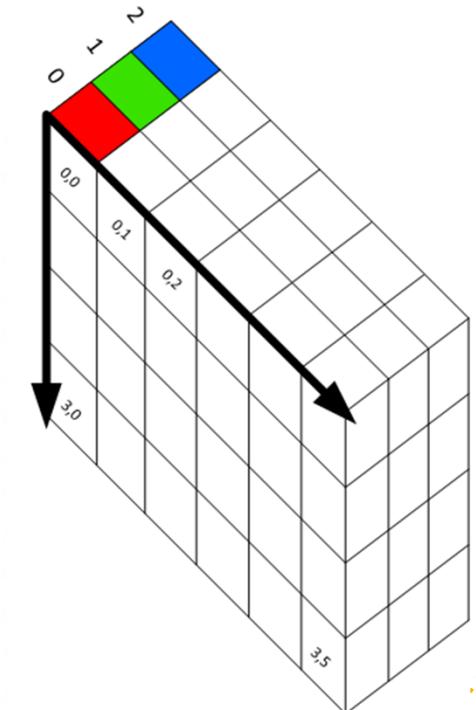


# How do Computers Store Images?



- A Color image would consist of 3 channels (RGB) - Right
- And a Grayscale image would consist of one channel - Below

|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 220 | 146 | 237 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 91  | 0   | 170 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 253 | 68  | 0   | 220 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 238 | 30  | 62  | 241 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 218 | 0   | 64  | 241 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 145 | 0   | 64  | 241 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 81  | 0   | 72  | 244 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 72  | 0   | 128 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 73  | 0   | 146 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 64  | 0   | 184 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 63  | 0   | 188 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 71  | 0   | 187 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 68  | 0   | 190 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 83  | 0   | 160 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 181 | 48  | 184 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |



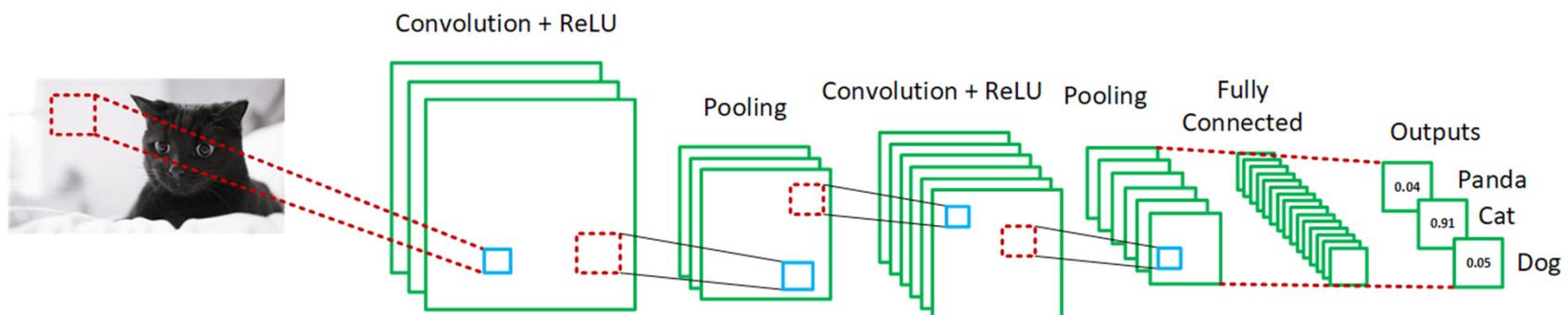


## Why NNs Don't Scale Well to Image Data

- Imagine a simple image classifier that takes color images of size  $64 \times 64$  (height, width).
- The Input size to our NN would be  $64 \times 64 \times 3 = 12,288$
- Therefore, our input layer will thus have 12,288 weights. While not an insanely large amount, imagine using input images of  $640 \times 480$ ? You'd have 921,600 weights! Add some more hidden layers and you'll see how fast this can grow out of control. Leading to very long training times
- However, our input is image data, consists of patterns and correlated inputs. There must be a way to take advantage of this.



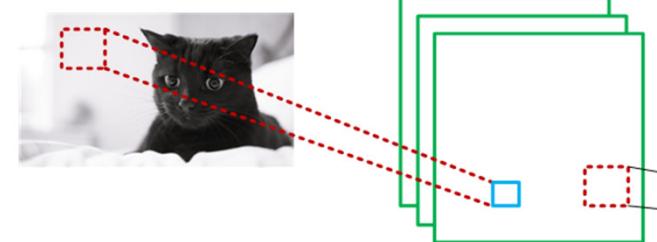
# CNNs are perfectly suited for Image Classifications





## CNN's use a 3D Volume Arrangement for it's Neurons

- Because our input data is an image, we can constrain or design our Neural Network to better suit this type of data
- Thus, we arrange our layers in **3 Dimensions**. Why 3?
- Because of image data consists of:
  - Height
  - Width
  - Depth (RGB) our colors components





## It's all in the name

- The **Convolution Layer** is the most significant part of a CNN as it is this layer that learns image features which aid our classifier.
- But what exactly is a **Convolution**?



# Convolutions

- Convolution is a mathematical term to describe the process of combining **two functions** to produce a **third function**.
- This **third function** or the output is called a **Feature Map**
- A convolution is the action of using a **filter** or **kernel** that is applied to the input. In our case, the input being our image.



# Examples of Image Features

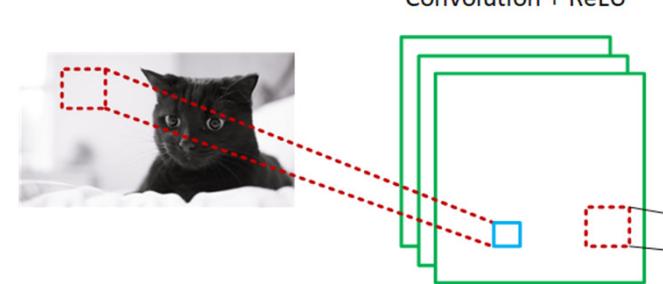


- Example filters learned by Krizhevsky et al.



# The Convolution Process

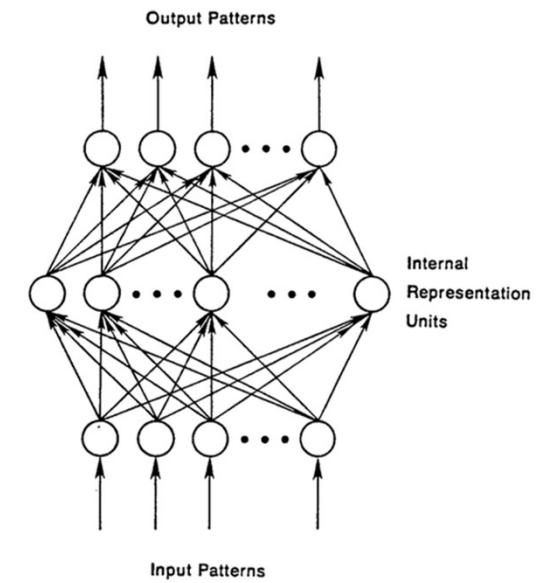
- **Convolutions are executed by sliding the filter or kernel over the input image.**
- **This sliding process is a simple matrix multiplication or dot product.**





# Recurrent Neural Networks

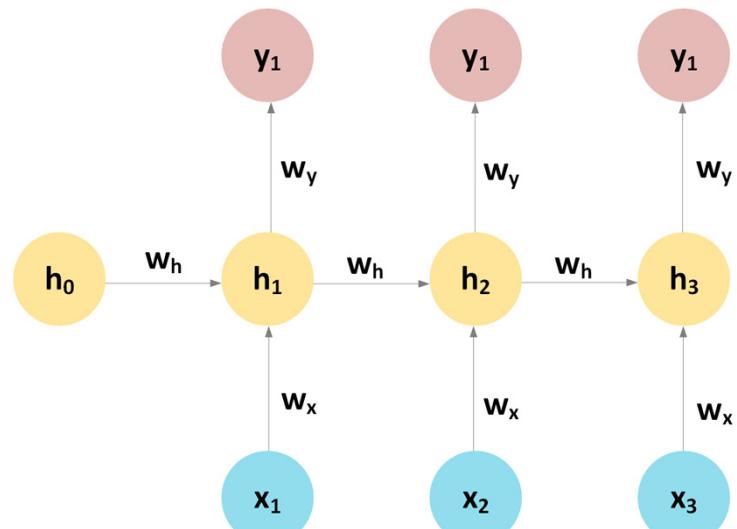
- Feed Forward Neural Networks have no concept of time, each classification is done only on the current inputs
- Recurrent networks**, take as their input not just the current input example they see, but also what they have perceived previously in time.
- Think about them as having two inputs, the present and past inputs.





# Recurrent Neural Networks

- RNNs can take the same input but produce **different outputs** depending on the **past inputs**.
- They are influenced not just by weights applied on inputs like a regular NN, but also by a “**hidden state vector (h)** representing the context based on prior input(s)/output(s).





## RNN Uses and Weaknesses

- RNNs have been very successfully applied to speech recognition, language modeling and image captioning.
- For RNNs, memory or 'context' awareness is very important. For example, saying the "The sky is...." we don't need much context to know the next word is most likely blue.
- However, saying, "I grew up in Brazil and I speak fluent....." knowing the next word is Portuguese isn't as easy and hence requires the ability to know deeper context. This is a situation where the gap between relevant information and the point where it's needed is very large.



## Introducing Long Short Term Memory Networks (LSTM)

- RNNs suffer with **vanishing gradient** and **exploding gradient**, which is some situations make it unusable.
- LSTMs solved this by introducing a memory unit (cell) into the network.
- LSTMs are a type of RNN, capable of learning long-term dependencies.
- LSTMs were designed to avoid the long-term dependency problem.
- They are excellent at remembering information for long periods of time
- LSTMs have the ability of having long chains of information that it decides whether to keep or not by using Gates.

6.0

# Neural Networks Explained



# Neural Networks Explained

- **6.3 Forward Propagation**
- **6.4 Activation Functions**
- **6.5 Training Part 1 – Loss Functions**
- **6.6 Training Part 2 – Backpropagation and Gradient Descent**
- **6.7. Backpropagation & Learning Rates – A Worked Example**
- **6.8 Regularization, Overfitting, Generalization and Test Datasets**
- **6.9 Epochs, Iterations and Batch Sizes**
- **6.10 Measuring Performance and the Confusion Matrix**
- **6.11 Review and Best Practices**

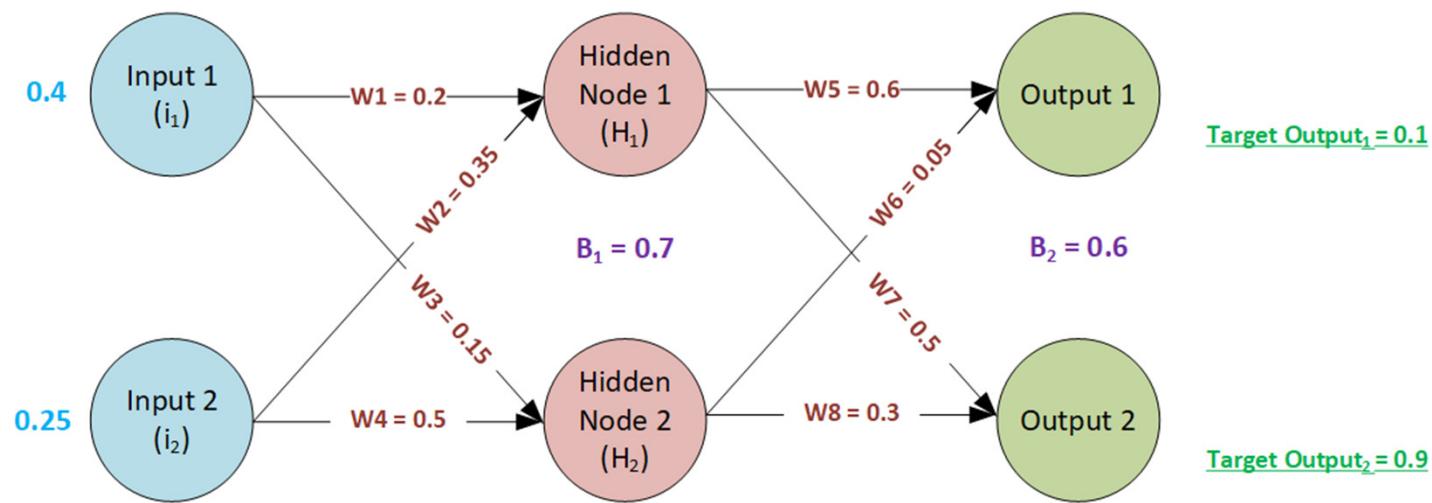
6.3

# Forward Propagation

How Neural Networks process their inputs to produce an output

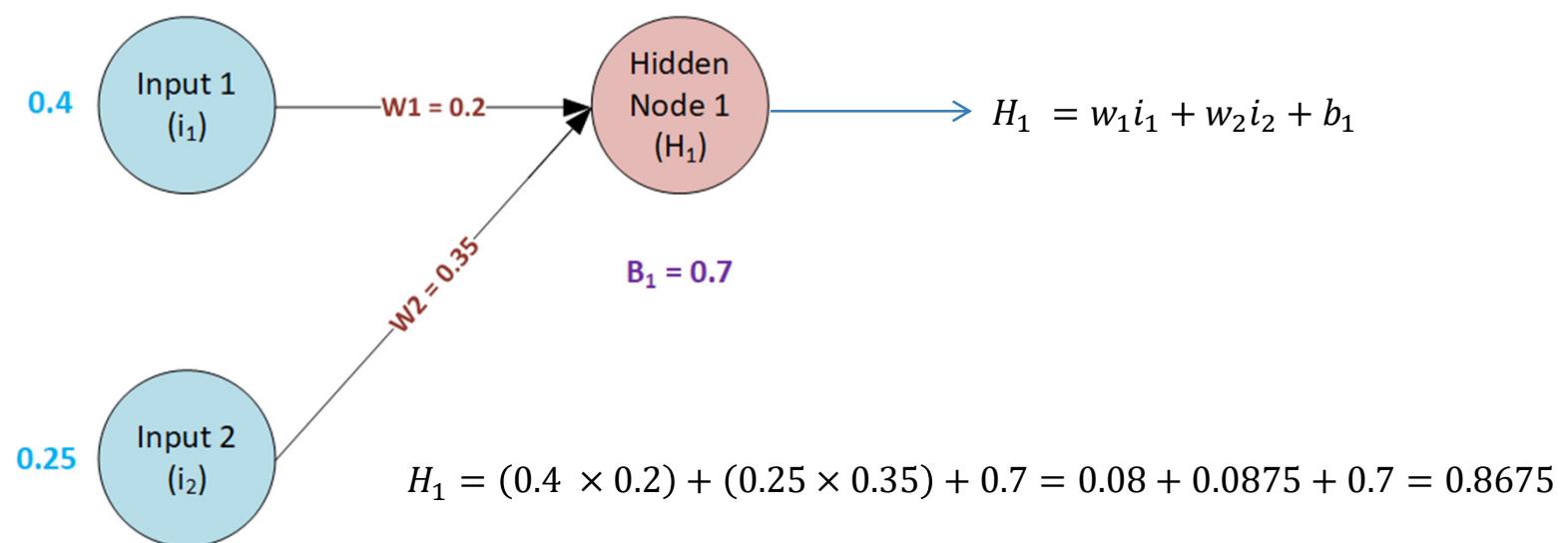


# Using actual values (random)



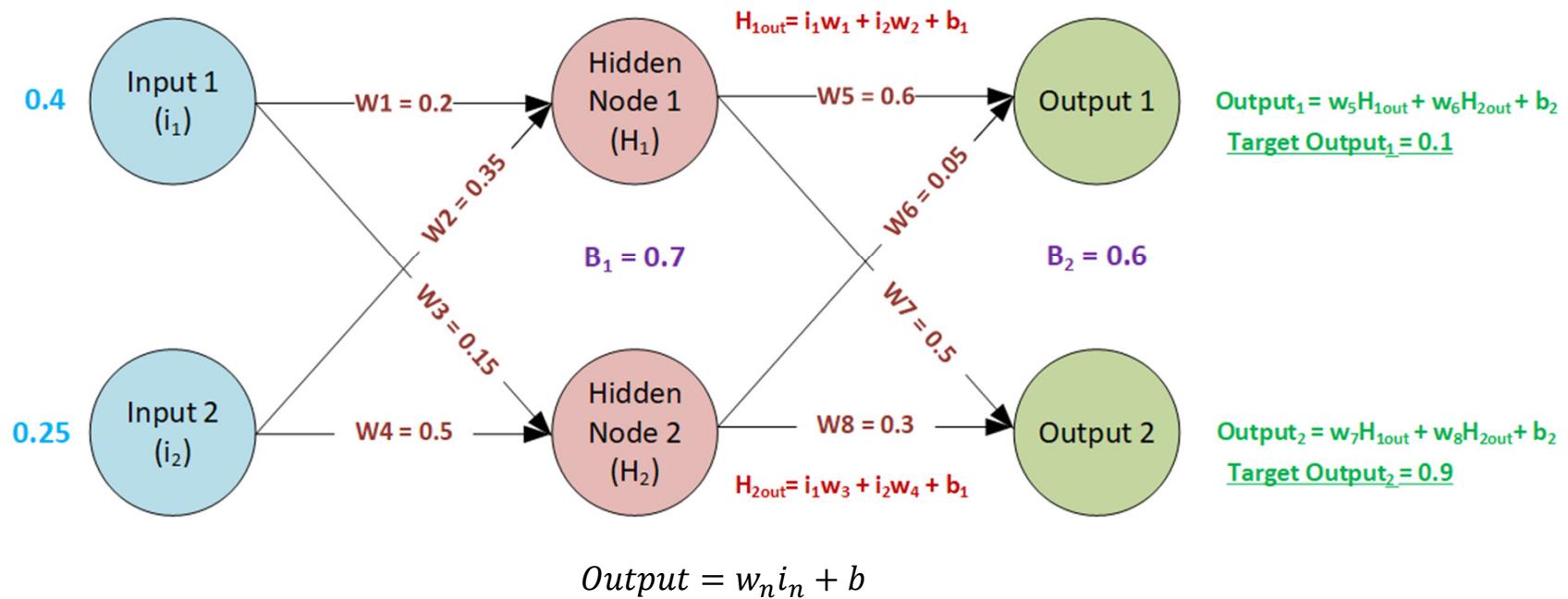


## Looking at a single Node/Neuron



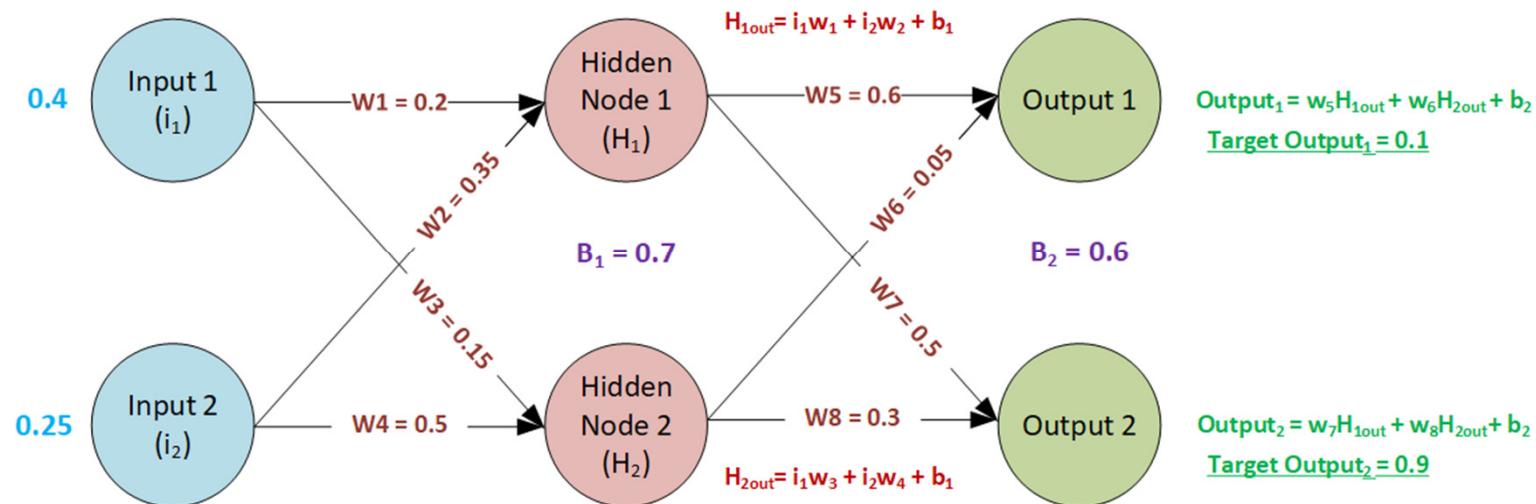


## Steps for all output values





In reality these connections are simply formulas that pass values from one layer to the next



$$\text{Output of Nodes} = w_n i_n + b$$

$$H_1 = i_1w_1 + i_2w_2 + b_1$$

$$\text{Output}_1 = w_5H_1 + H_2w_6 + b_2$$

# Calculating H<sub>2</sub>

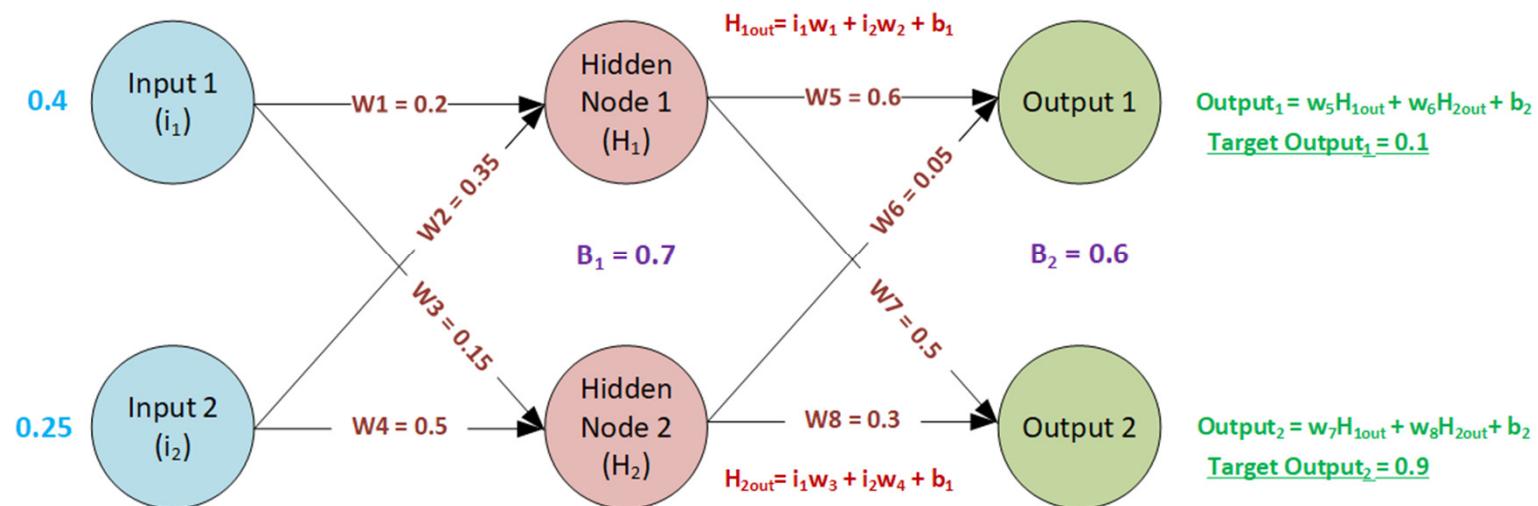


$$H_2 = i_1w_3 + i_2w_4 + b_1$$

$$H_2 = (0.4 \times 0.15) + (0.25 \times 0.5) + 0.7 = 0.06 + 0.125 + 0.7 = 0.885$$



# Getting our final outputs



$$\text{Output}_1 = w_5H_1 + H_2w_6 + b_2$$

$$\text{Output}_1 = (0.6 \times 0.8675) + (0.05 \times 0.885) + 0.6 = 0.5205 + 0.04425 + 0.6 = 1.16475$$

$$\text{Output}_2 = 0.43375 + 0.2655 + 0.6 = 1.29925$$



# What does this tell us?

- Our initial default random weights ( $w$  and  $b$ ) produced very incorrect results
- Feeding numbers through a neural network is a matter of simple matrix multiplication
- Our neural network is still just a series of linear equations



## The Bias Trick

### - Making our weights and biases as one

- Now is a good time as any to show you the bias trick that is used to simplify our calculations.

$$\begin{array}{c} \begin{matrix} 0.2 & 0.65 & 0.54 & -3.1 \\ 1.2 & -0.23 & 0.23 & 0.5 \\ 0.35 & 1.5 & -0.7 & 0.02 \end{matrix} \\ W \end{array} \cdot \begin{array}{c} \begin{matrix} 44 \\ 73 \\ 32 \\ 64 \end{matrix} \\ x_i \end{array} + \begin{array}{c} \begin{matrix} -3.1 \\ 0.5 \\ 0.02 \end{matrix} \\ b \end{array} \longleftrightarrow \begin{array}{c} \begin{matrix} 0.2 & 0.65 & -3.1 & -3.1 \\ 1.2 & -0.23 & 0.5 & 0.5 \\ 0.35 & 1.5 & 0.02 & 0.02 \end{matrix} \\ W \end{array} \cdot \begin{array}{c} \begin{matrix} 44 \\ 73 \\ 32 \\ 64 \\ 1 \end{matrix} \\ x_i \end{array} + b$$

- $x_i$  is our input values, instead of doing a multiplication then adding of our biases, we can simply add the biases to our weight matrix and add an addition element to our input data as 1.
- This simplifies our calculation operations as we treat the biases and weights as one.
- NOTE:** This makes our input vector size bigger by one i.e if  $x_i$  had 32 values, it will now have 33.

**6.4**

# **Activation Functions & What ‘Deep’ really means**

An introduction to activation functions and their usefulness



# Introducing Activation Functions

- In reality each Hidden Node also passes through an activation function.
- In the simplest terms an activation function changes the output of that function. For example, let's look at a simple activation function where values below 0 are clamped to 0. Meaning negative values are changed to 0.

$$\text{Output} = \text{Activation}(w_i x_i + b)$$

$$f(x) = \max(0, x)$$

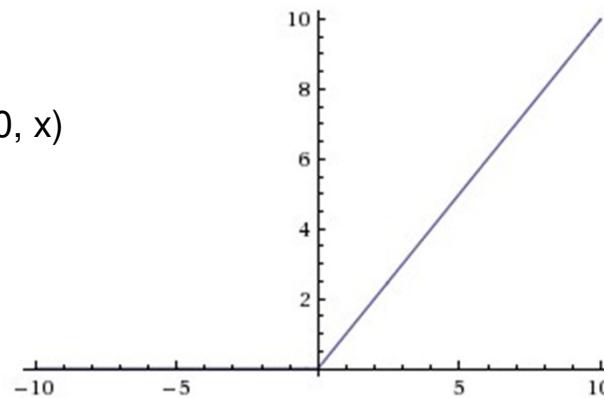
- Therefore, if  $w_i x_i + b = 0.75$
- $f(x) = 0.75$
- However, if  $w_i x_i + b = -0.5$       then  $f(x) = 0$



# The ReLU Activation Function

- This activation function is called the ReLU (Rectified Linear Unit) function and is one of the most commonly used activation functions in training Neural Networks.
- It takes the following appearance. The clamping values at 0 accounts for its non linear behavior.

$$f(x) = \max(0, x)$$





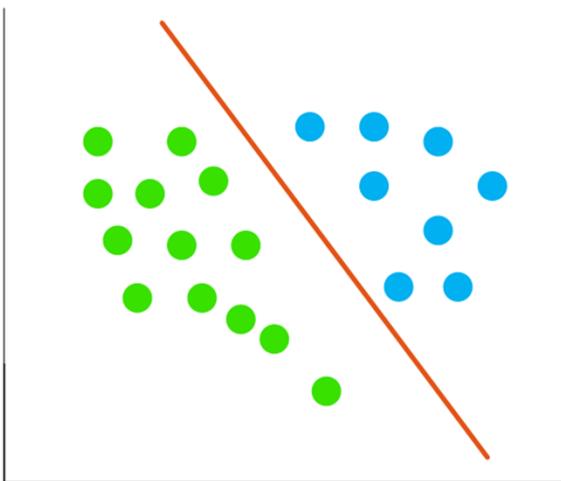
# Why use Activation Functions? For Non Linearity

- Most ML algorithms find non linear data extremely hard to model
- The huge advantage of deep learning is the ability to understand nonlinear models

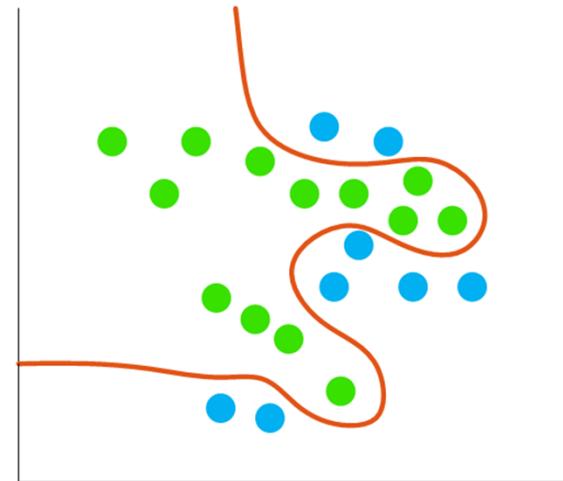


# Example of Non Linear Data

Linearly Separable



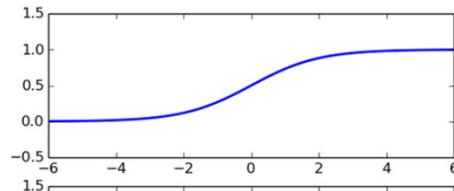
Non-Linearly Separable



**NOTE:** This shows just 2 Dimensions, imagine separating data with multiple dimensions

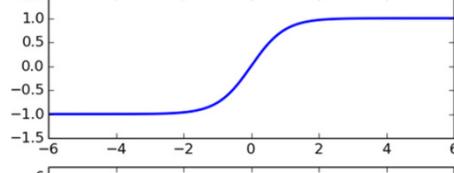


# Types of Activation Functions



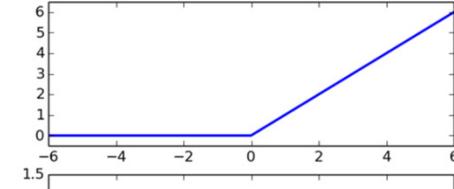
Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



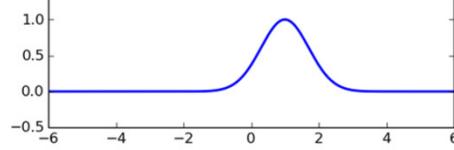
Hyperbolic Tangent

$$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



Rectified Linear

$$\phi(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$



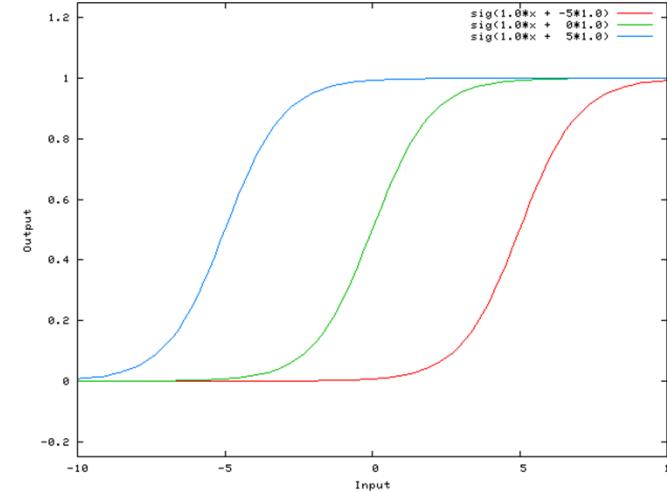
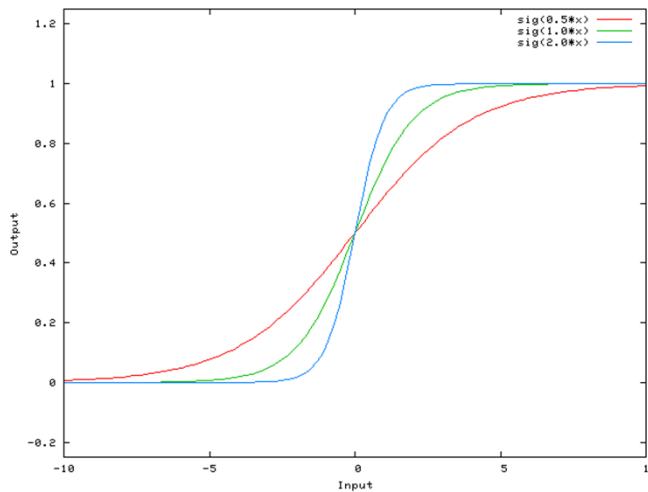
Radial Basis Function

$$\phi(z, c) = e^{-(\epsilon \|z - c\|)^2}$$



## Why do we have biases in the first place?

- Biases provide every node/neuron with a trainable constant value.
- This allows us to shift the activation function output left or right.

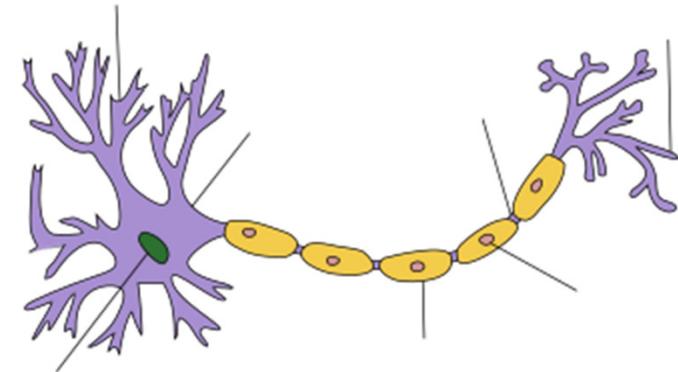


- Changes in weights simply change the gradient or steepness of the output, if we needed shift our function left or right , we need a bias.



# Neuron Inspiration

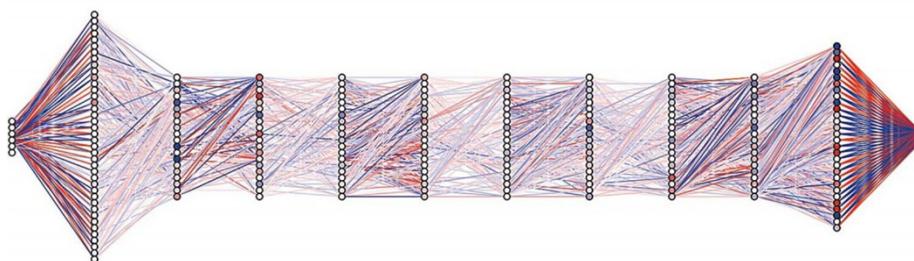
- Neuron only fires when an input threshold is reached
- Neural Networks follow that same principle





## The 'Deep' in Deep Learning: Hidden Layers

- Depth refers to the number of hidden layers
- The deeper the network the better it learns non-linear mappings
- Deeper is always better, however there becomes a point of diminishing returns and overly long training time.
- Deeper Networks can lead to over fitting



Source: A visualization of Meade's neural network for predicting earthquakes  
<https://harvardmagazine.com/2017/11/earthquakes-around-the-world>



## The Real Magic Happens in Training

- We've seen Neural Networks are simple to execute, just matrix multiplications
- How do we determine those weights and biases?

6.5

# Training Part 1: Loss Functions

The first step in determining the best weights for our Neural Network



# Learning the Weights

- As you saw previously, our random default weights produced some very bad results.
- What we need is a way to figure out how to change the weights so that our results are more accurate.
- This is where the brilliance of Loss Functions , Gradient Descent and Backpropagation show their worth.



## How do we beginin training a NN? What exactly do we need?

- Some (*more than some*) accurately labeled data, that we'll call a dataset
- A Neural Network Library/Framework (*Keras*)
- Patience and a decently fast computer

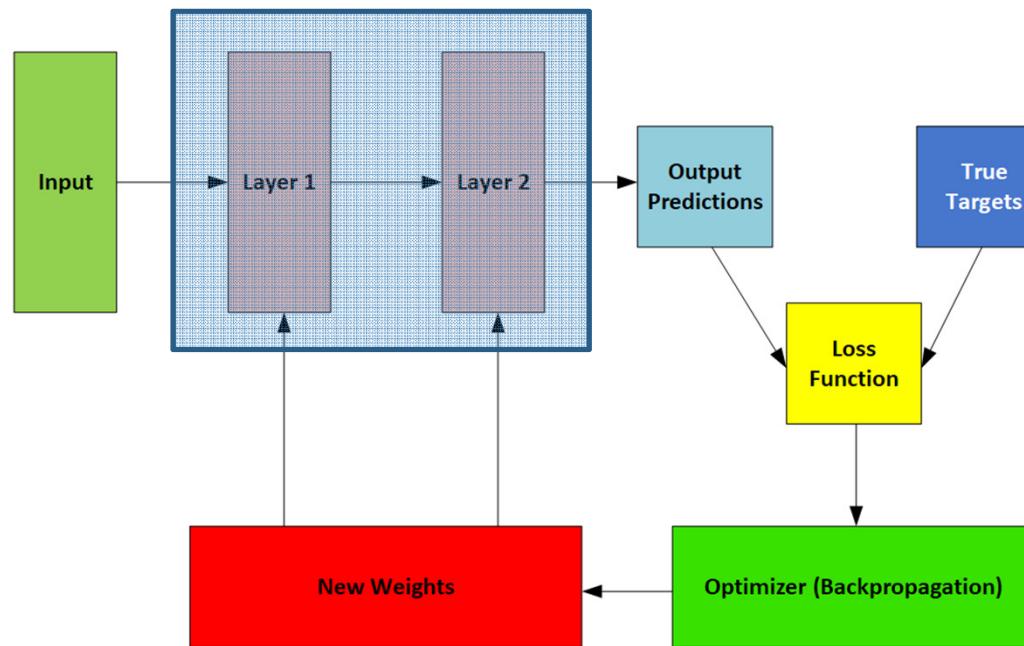


# Training step by step

1. Initialize some random values for our weights and bias
2. Input a single sample of our data
3. Compare our output with the actual value it was supposed to be, we'll be calling this our Target values.
4. Quantify how 'bad' these random weights were, we'll call this our Loss.
5. Adjust weights so that the Loss lower
6. Keep doing this for each sample in our dataset
7. Then send the entire dataset through this weight 'optimization' program to see if we get an even lower loss
8. Stop training when the loss stops decreasing.



# Training Process Visualized





# Quantifying Loss with Loss Functions

- In our previous example our Neural Network produced some very 'bad' results, but how do we measure how bad they are?

| Outputs | Predicted Results | Target Values | Difference (P-T) |
|---------|-------------------|---------------|------------------|
| 1       | 1.16475           | 0.1           | 1.06475          |
| 2       | 1.29925           | 0.9           | 0.39925          |



# Loss Functions

- Loss functions are integral in training Neural Nets as they measure the inconsistency or difference between the predicted results & the actual target results.
- They are always positive and penalize big errors well
- The lower the loss the 'better' the model
- There are many loss functions, Mean Squared Error (MSE) is popular
- $\text{MSE} = (\text{Target} - \text{Predicted})^2$

| Outputs | Predicted Results | Target Values | Error (T-P) | MSE          |
|---------|-------------------|---------------|-------------|--------------|
| 1       | 1.16475           | 0.1           | -1.06475    | 1.1336925625 |
| 2       | 1.29925           | 0.9           | -0.39925    | 0.1594005625 |



# Types of Loss Functions

- There are many types of loss functions such as:
  - L1
  - L2
  - Cross Entropy – Used in binary classifications
  - Hinge Loss
  - Mean Absolute Error (MAE)

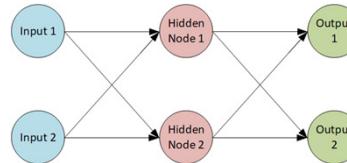
In practice, MSE is always a good safe choice. We'll discuss using different loss functions later on.

**NOTE:** A low loss goes hand in hand with accuracy. However, there is more to a good model than good training accuracy and low loss. We'll learn more about this soon.



## Using the Loss to Correct the Weights

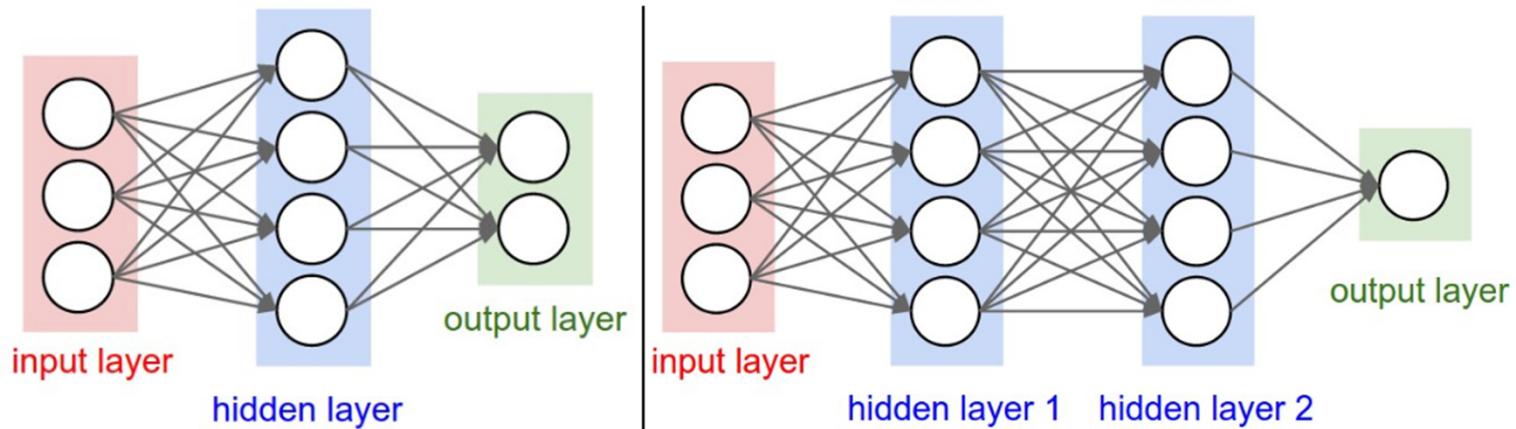
- Getting the best weights for our classifying our data is not a trivial task, especially with large image data which can contain thousands of inputs from a single image.



- Our simple 2 input, 2 output and 1 hidden layer network has X parameters.
  - Input Nodes X Hidden Layers + Hidden Layers x Output + Biases
  - In our case this is:  $(2 \times 2) + (2 \times 2) + 4 = 12$  Learnable Parameters



## Calculating the Number of Parameters



$$(3 \times 4) + (4 \times 2) + (4+2) = 26$$

$$(3 \times 4) + (4 \times 4) + (4 \times 1) + (4+4+1) = 41$$

6.6

# Training Part 2: Backpropagation & Gradient Descent

Determining the best weights efficiently



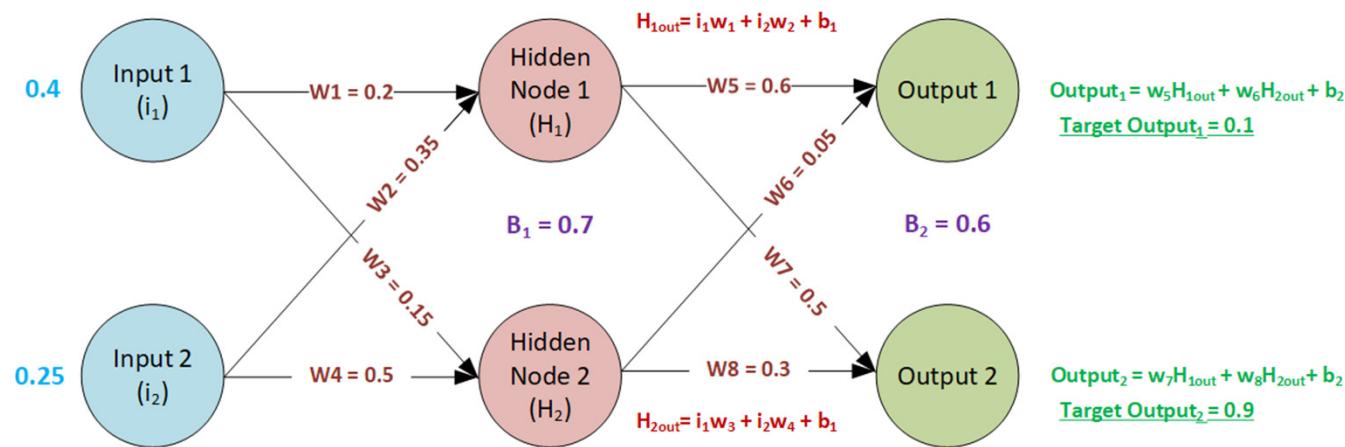
# Introducing Backpropagation

- What if there was a way we could use the loss to determine how to adjust the weights.
- That is the brilliance of Backpropagation

Backpropagation tells us how much would a change in each weight affect the overall loss.



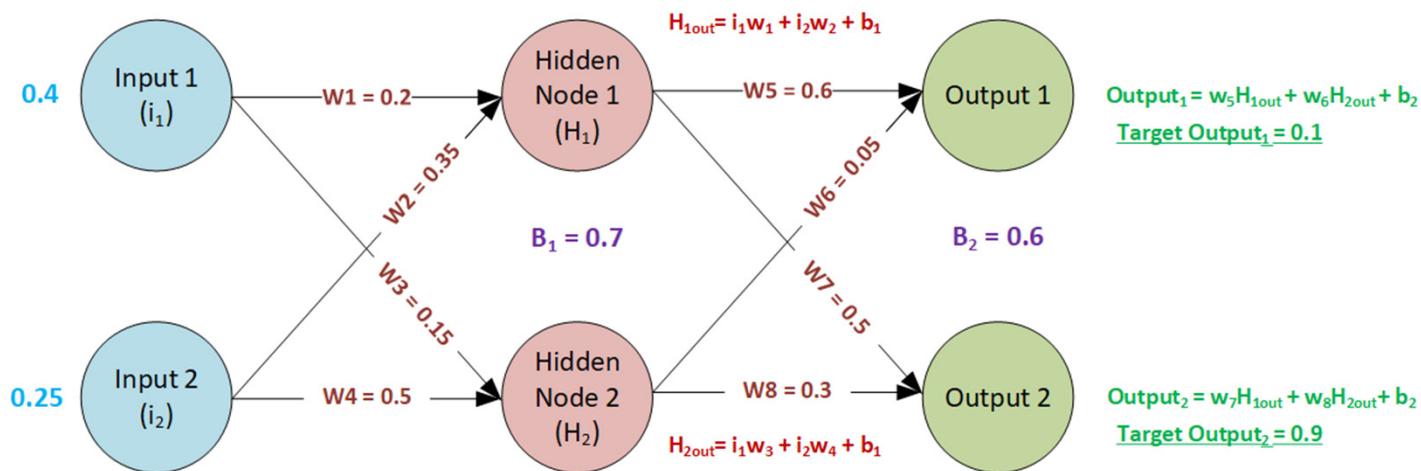
## Backpropagation: Revisiting our Neural Net



- Using the MSE obtained from Output 1, Backpropagation allows us know:
  - If changing  $w_5$  from 0.6 by a small amount, say to 0.6001 or 0.5999,
  - Whether our overall Error or Loss has increased or decreased.



## Backpropagation: Revisiting our Neural Net



- We then backpropagate this loss to each node (Right to Left) to determine which direction the weight should move (negative or positive)
- We do this for all nodes in the Neural Network



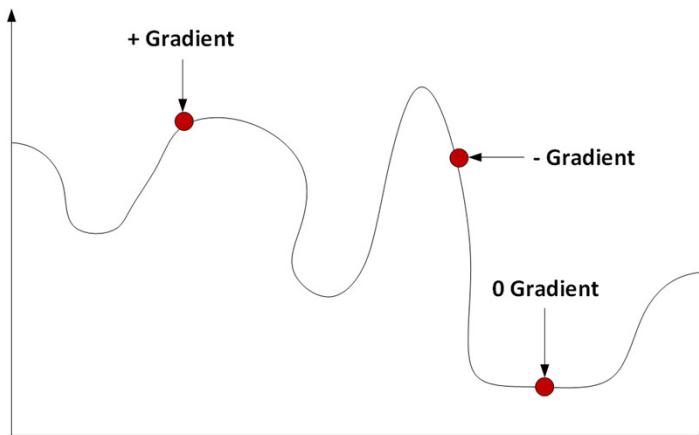
## Backpropagation – The full cycle

- Therefore, by simply passing one set of inputs of a single piece of our training data, we can adjust all weights to reduce the loss or error.
- However, this tunes the weights for that specific input data. How do make our Neural Network generalize?
- We do this for each training samples in our training data (called an Epoch or Iteration).



# Introducing Gradient Descent

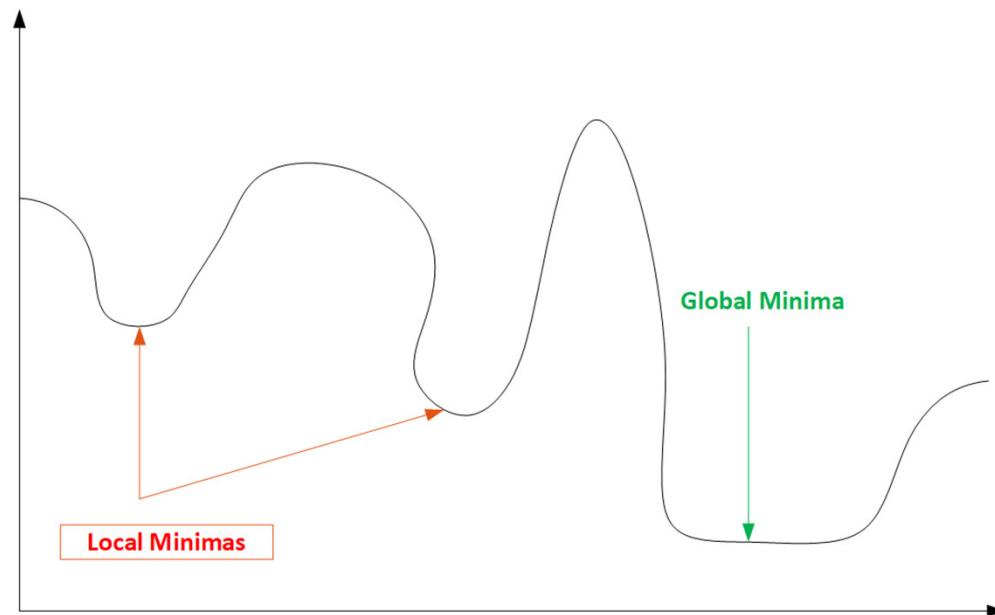
- By adjusting the weights to lower the loss, we are performing gradient descent. This is an 'optimization' problem.
- Backpropagation is simply the method by which we execute gradient descent
- Gradients (also called slope) are the direction of a function at a point, it's magnitude signifies how much the function is changing at that point.



- By adjusting the weights to lower the loss, we are performing gradient descent.
- Gradients are the direction of a function at a point



# Gradient Descent



Imagine our global minima is the bottom of this rough bowl. We need to traverse through many peaks and valleys before we find it



# Stochastic Gradient Descent

- Naïve Gradient Decent is very computationally expensive/slow as it requires exposure to the entire dataset, then updates the gradient.
- Stochastic Gradient Descent (SGD) does the updates after every input sample. This produces noisy or fluctuating loss outputs. However, again this method can be slow.
- Mini Batch Gradient Descent is a combination of the two. It takes a batch of input samples and updates the gradient after that batch is processed (batches are typical 20-500, though no clear rule exists). It leads to much faster training (i.e. faster convergence to the global minima)



# Overview

We learned:

- That Loss Functions (such as MSE) quantify how much error our current weights produce.
- That Backpropagation can be used to determine how to change the weights so that our loss is lower.
- This process of optimizing or lowering the weights is called Gradient Descent, and an efficient method of doing this is the Mini Batch Gradient Descent algorithm.

6.7

# Back Propagation & Learning Rates: A Worked Example

A worked example of Back Propagation.



# Backpropagation

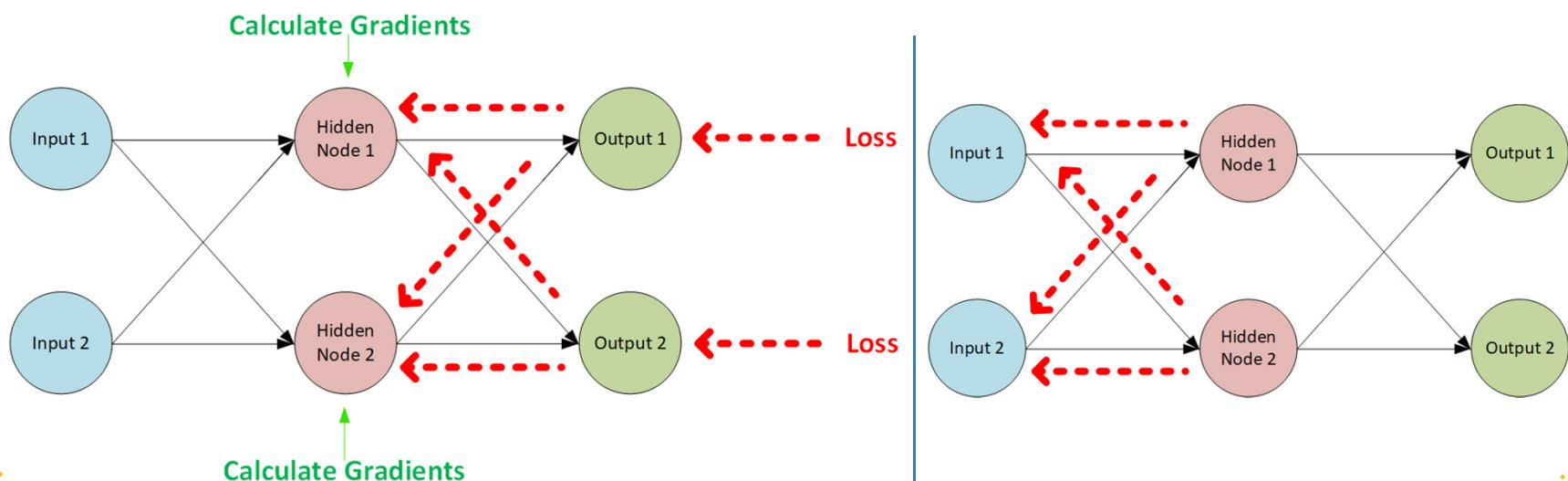
- From the previous section you have an idea of what we achieve with Backpropagation.
- It's a method of executing gradient descent or weight optimization so that we have an efficient method of getting the lowest possible loss.

How does this 'black magic' actually work?



# Backpropagation Simplified

- We obtain the total error at the output nodes and then propagate these errors back through the network using Backpropagation to obtain the new and better gradients or weights.





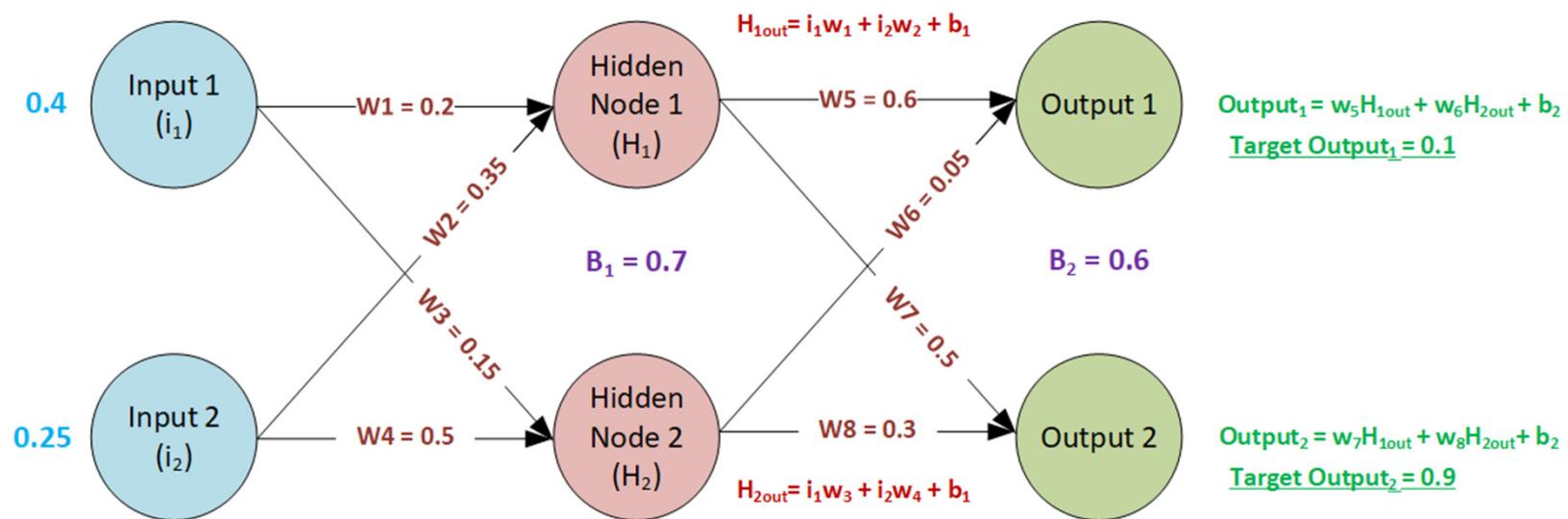
# The Chain Rule

- Backpropagation is made possible by the *Chain Rule*.
- What is the Chain Rule? Without over complicating things, it's defined as:
  - If we have two functions:  $y = f(u)$  and  $u = g(x)$  then the derivative of  $y$  is:

$$\frac{dy}{dx} = \frac{dy}{du} \times \frac{du}{dx}$$

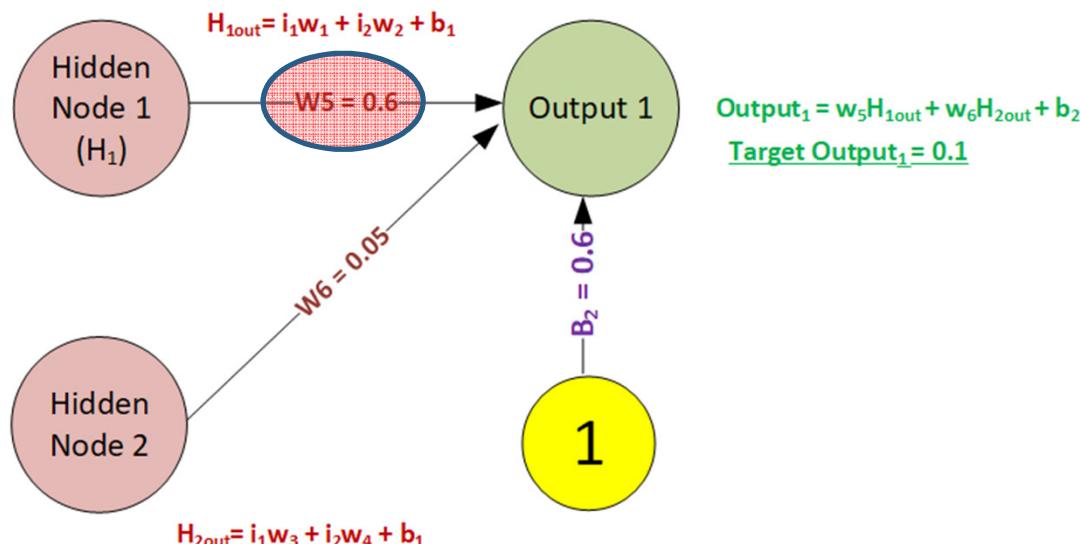


## Let's take a look at our previous basic NN





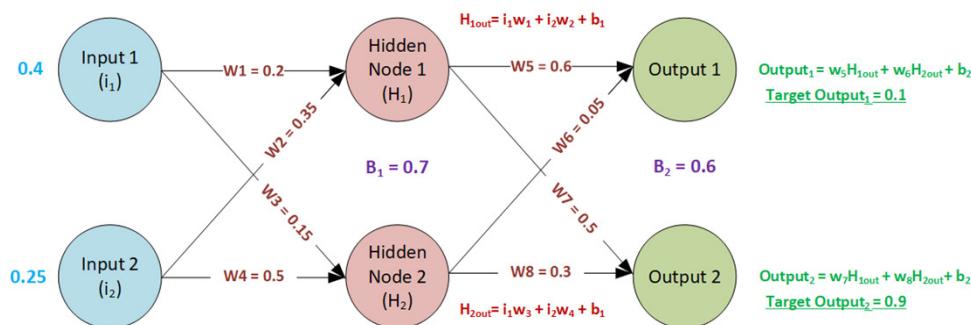
## We use the Chain Rule to Determine the Direction the Weights Should Take



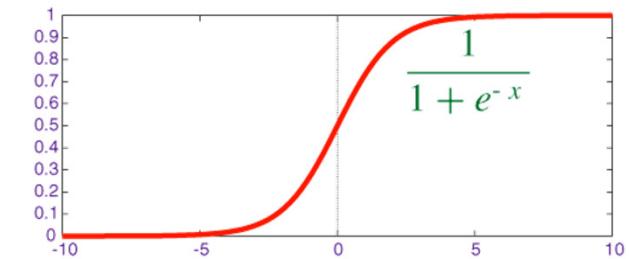
- Let's take a look at  $W_5$ , how does a change in  $W_5$  affect the Error at  $\text{Output}_1$ ?
- Should  $W_5$  be increased or decreased?



## Our Calculated Forward Propagation and Loss Values



Logistic Activation Function  
Squashes the output between 0 and 1



- Using a Logistic Activation Function at each node, our Forward Propagation values become:

- $H_1 = \mathbf{0.704225}$

- $H_2 = \mathbf{0.707857}$

- $\text{Output } 1 = \mathbf{0.742294}$

- $\text{Output } 2 = \mathbf{0.762144}$

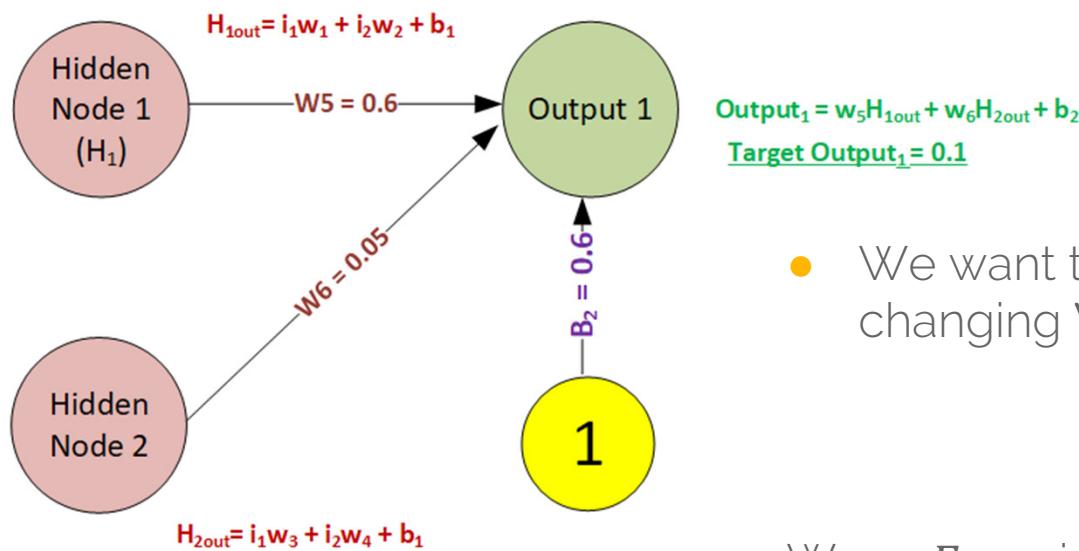


## Slight Change in Our MSE Loss Function

- Let's define the MSE as
  - = Error =  $\frac{1}{2}(\text{target} - \text{output})^2$
- The  $\frac{1}{2}$  is included to cancel the exponent when differentiated (looks better)
- Output 1 Loss = **0.206271039**
- Output 2 Loss = **0.009502145**



# Exploring $W_5$



$$\begin{aligned} \text{Output}_1 &= w_5 H_{1\text{out}} + w_6 H_{2\text{out}} + b_2 \\ \text{Target Output}_1 &= 0.1 \end{aligned}$$

- We want to know how much changing  $W_5$  changes to total Error.

$$\frac{dE_{total}}{dw_5}$$

Where  $E_{total}$  is the sum of the Error from Output 1 and Output 2



## Using the Chain Rule to Calculate $W_5$

- $\frac{dE_{total}}{dw_5} = \frac{dE_{total}}{dOut_1} \times \frac{dOut_1}{dNetOutput_1} \times \frac{dNetOutput_1}{dw_5}$
- $E_{total} = \frac{1}{2}(target_{o1} - out_1)^2 + \frac{1}{2}(target_{o2} - out_2)^2$

Therefore differentiating  $E_{total}$  with respect to  $out_1$  gives us

$$\frac{dE_{total}}{dout_1} = 2 \times \frac{1}{2}(target_{o1} - out_1)^{2-1} \times (-1) + 0$$

$$\frac{dE_{total}}{dout_1} = out_1 - target_{o1}$$

$$1 \quad \frac{dE_{total}}{dout_1} = (0.742294385 - 0.1) = 0.642294385$$



Let's get  $\frac{dOut_1}{dNetOutput_1}$

- $dOut_1 = \frac{1}{1+e^{-netO_1}}$
- Fortunately, the partial derivative of the logistic function is the output multiplied by 1 minus the output:  
$$\frac{dOut_1}{dnetO_1} = out_{o1}(1 - out_{o1}) = 0.742294385(1 - 0.742294385)$$
- 2 • 
$$\frac{dOut_1}{dnetO_1} = -0.191293$$



Let's get  $\frac{dnetO_1}{dw_5}$

- $netO_1 = (w_5 \times outH_1) + (w_6 \times outH_2) + (b_2 \times 1)$
- $\frac{dnetO_1}{dw_5} = 1 \times outH_1 \times w_5^{(1-1)} + 0 + 0$
- 3 •  $\frac{dnetO_1}{dw_5} = outH_1 = 0.704225234$



We now have all the pieces to get  $\frac{dE_{total}}{dw_5}$

- $\frac{dE_{total}}{dw_5} = \frac{dE_{total}}{dOut_1} \times \frac{dOut_1}{dNetOutput_1} \times \frac{dNetOutput_1}{dw_5}$
- $\frac{dE_{total}}{dw_5} = 0.642294385 \times 0.191293431* \times 0.704225234$
- $\frac{dE_{total}}{dw_5} = 0.086526$



## So what's the new weight for $W_5$ ?

- *New*  $w_5 = w_5 - \eta \times \frac{dE_{total}}{dw_5}$
- *New*  $w_5 = 0.6 - (0.5 \times 0.086526) = 0.556737$

## Learning Rate

- Notice we introduced a new parameter ' $\eta$ ' and gave it a value of **0.5**
- Look carefully at the first formula. The learning rate simply controls how a big a magnitude jump we take in the direction of  $\frac{dE_{total}}{dw_5}$
- Learning rates are always positive and range from  $\gamma > 0 \leq 1$
- A large learning rate will allow faster training, but can overshoot the global minimum (getting trapped in a local minima instead). A small learning will take longer to train but will more reliably find the global minimum.

## Check your answers



- $\text{new } w_6 = 0.400981$
- $\text{new } w_7 = 0.508845$
- $\text{new } w_8 = 0.558799$



## You've just used Backpropagation to Calculate the new $W_5$

- You can now calculate the new updates for  $W_6$ ,  $W_7$  and  $W_8$  similarly.
- $W_1$ ,  $W_2$ ,  $W_3$  and  $W_4$  are similar:

$$\frac{dE_{total}}{dw_1} = \frac{dE_{total}}{dOut_{H1}} \times \frac{dOut_{H1}}{dNetOutput_{H1}} \times \frac{dNetOutput_{H1}}{dw_1}$$

- I'll leave this as an exercise for you.

# 6.8

## **Regularization, Overfitting, Generalization and Test Datasets**

How do we know our model is good?



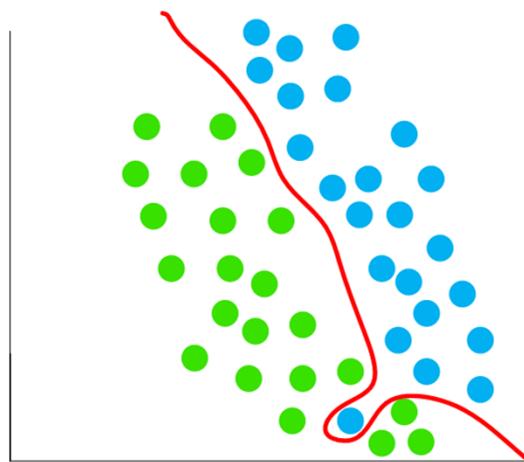
# What Makes a Good Model?

- A good model is accurate
- Generalizes well
- Does not overfit

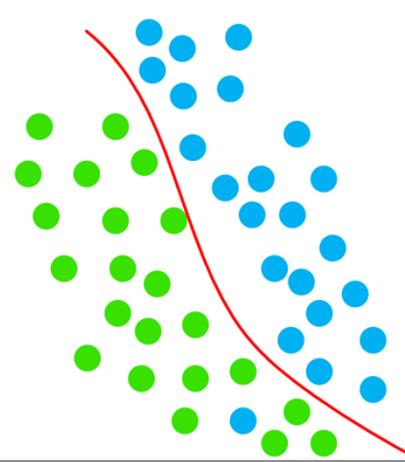


# What Makes a Good Model?

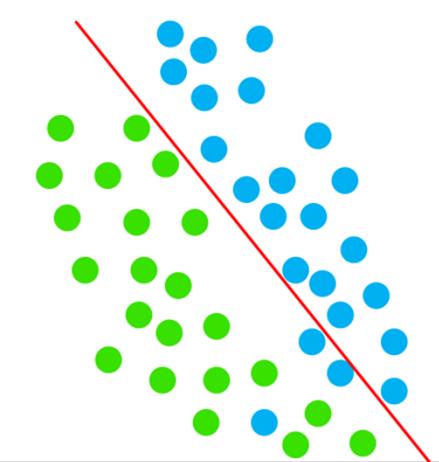
Model A



Model B



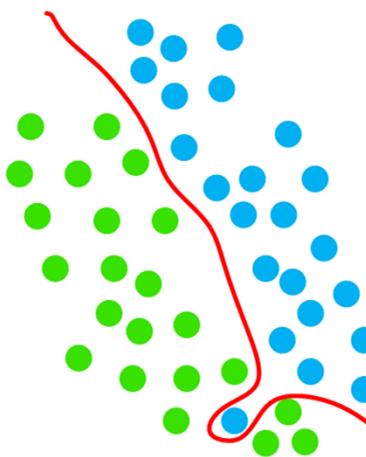
Model C



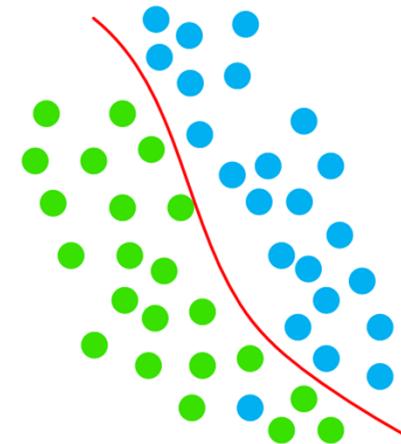


# What Makes a Good Model?

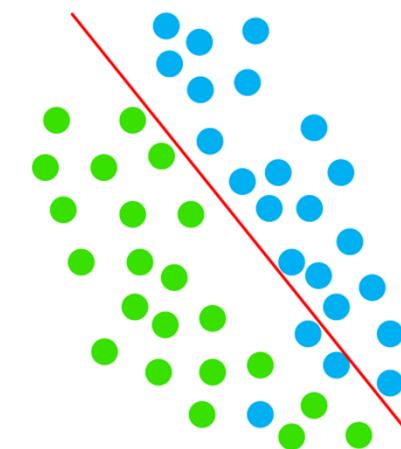
- Overfitting



- Ideal or Balanced



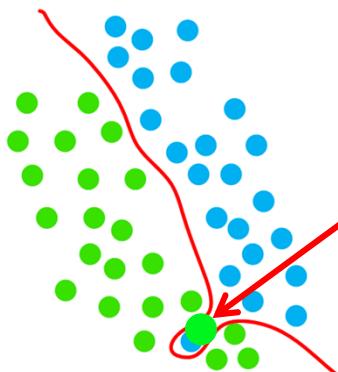
- Underfitting





# Overfitting

- Overfitting leads to *poor models* and is one of the most common problems faced developing in AI/Machine Learning/Neural Nets.
- Overfitting occurs when our Model fits near perfectly to our training data, as we saw in the previous slide with Model A. However, fitting to closely to training data isn't always a good thing.



- What happens if we try to classify a brand new point that occurs at the position shown on the left? (who's true color is green)
- It will be misclassified because our model has overfit the test data
- Models don't need to be complex to be good



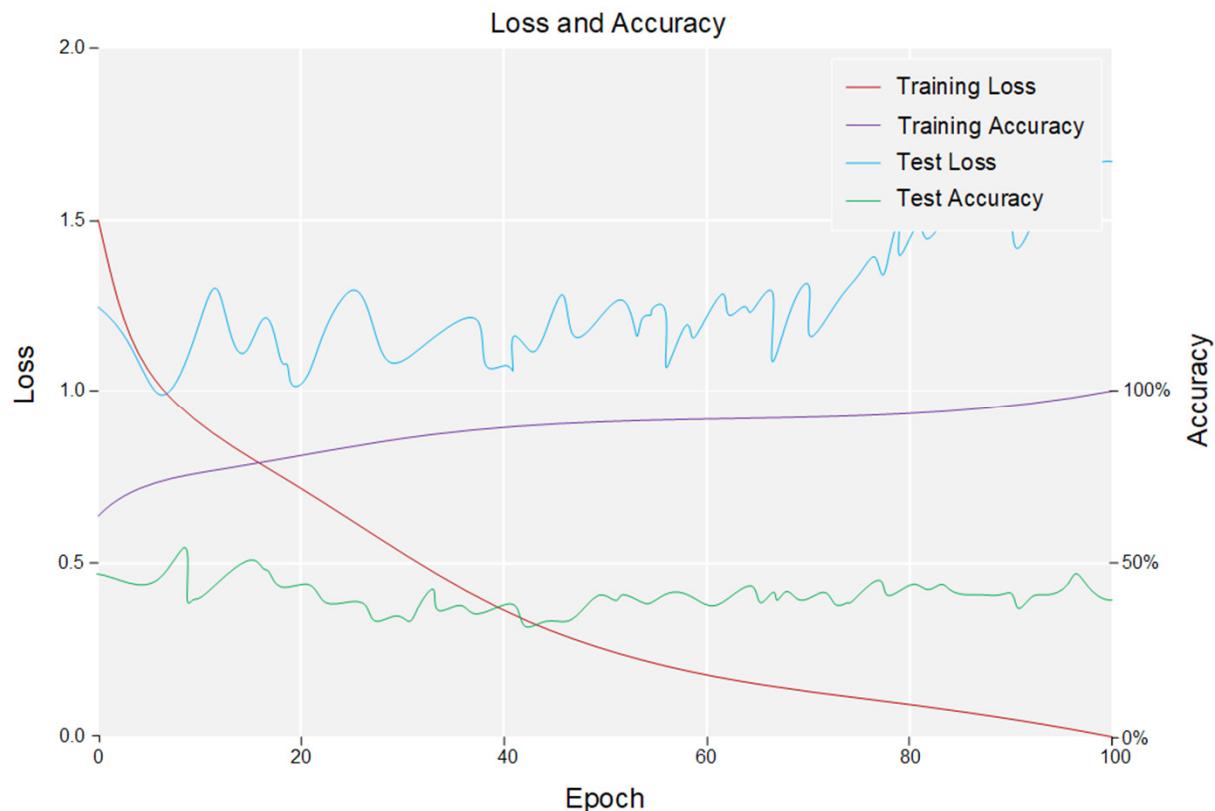
# How do we know if we've Overfit?

## Test on your model on.....Test Data!

- In all Machine Learning it is extremely important we hold back a portion of our data (10-30%) as pure untouched test data.

|               |           |
|---------------|-----------|
| Training Data | Test Data |
|---------------|-----------|
- Untouched meaning that this data is NEVER seen by the training algorithm. It is used purely to test the performance of our model to assess its accuracy in classifying new never before seen data.
- Many times when Overfitting we can achieve high accuracy 95%+ on our test data, but then get abysmal (~70%) results on the test data. That is a perfect example of Overfitting.

## Overfitting Illustrated Graphically

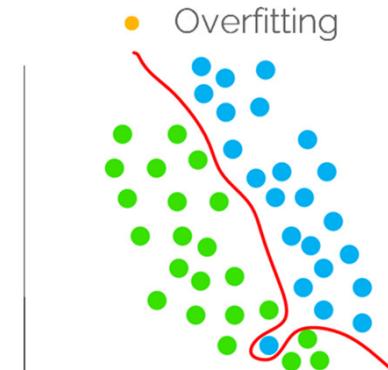


- Examine our Training Loss and Accuracy. They're both heading the right directions!
- But, what's happening to the loss and accuracy on our Test data?
- This is classic example of Overfitting to our training data



# How do we avoid overfitting?

- Overfitting is a consequence of our weights. Our weights have been tuned to fit our Training Data well but due to this 'over tuning' it performs poorly on unseen Test Data.
- We know our weights are a decent model, just too sensitively tuned. If only there were a way to fix this?





# How do we avoid overfitting?

- We can use less weights to get smaller/less deep Neural Networks
  - Deeper models can sometimes find features or interpret noise to be important in data, due to their abilities to memorize more features (called memorization capacity)



# How do we avoid overfitting?

- We can use less weights to get smaller/less deep Neural Networks
  - Deeper models can sometimes find features or interpret noise to be important in data, due to their abilities to memorize more features (called memorization capacity)

## Or Regularization!

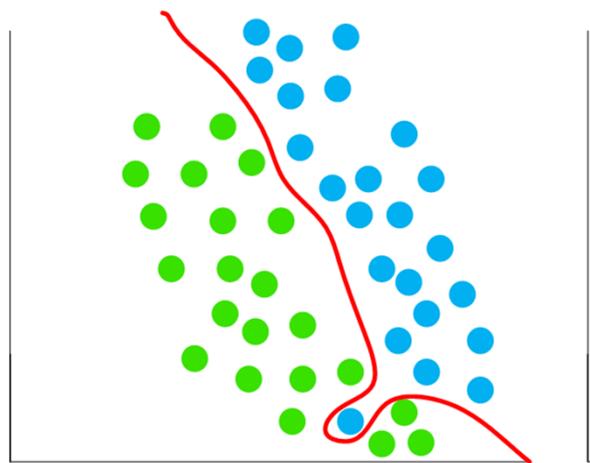
- It is better practice to regularize than reduce our model complexity.



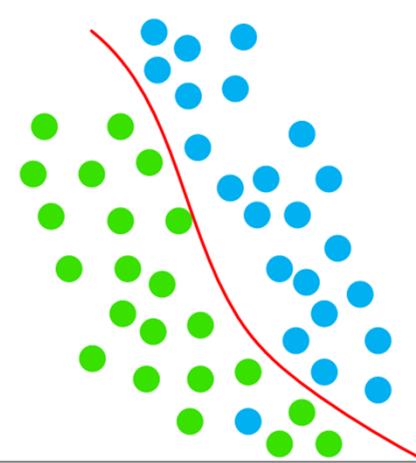
# What is Regularization?

- It is a method of making our model more general to our dataset.

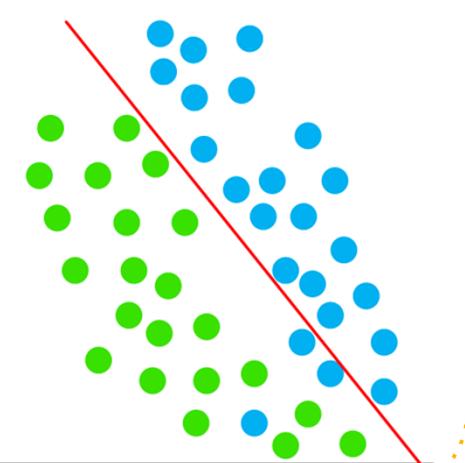
- Overfitting



- Ideal or Balanced



- Underfitting





# Types of Regularization

- L1 & L2 Regularization
- Cross Validation
- Early Stopping
- Drop Out
- Dataset Augmentation



# L1 And L2 Regularization

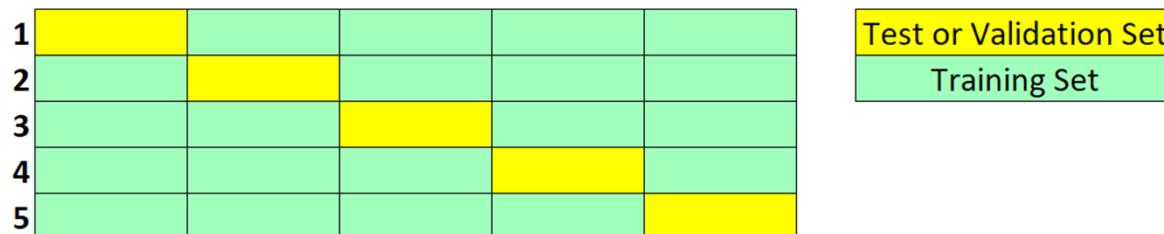
- L1 & L2 regularization are techniques we use to penalize large weights. Large weights or gradients manifest as abrupt changes in our model's decision boundary. By penalizing, we're really making them smaller.
- L2 also known as Ridge Regression
  - $Error = \frac{1}{2}(target_{01} - out_1)^2 + \frac{\lambda}{2} \sum w_i^2$
- L1 also known as Lasso Regression
  - $Error = \frac{1}{2}(target_{01} - out_1)^2 + \frac{\lambda}{2} \sum |w_i|$
- $\lambda$  controls the degree of penalty we apply.
- Via Backpropagation, the penalty on the weights is applied to the weight updates
- The difference between them is that L1 brings the weights of the unimportant features to 0, thus acting as feature selection algorithm (known as sparse models or models with reduced parameters.)



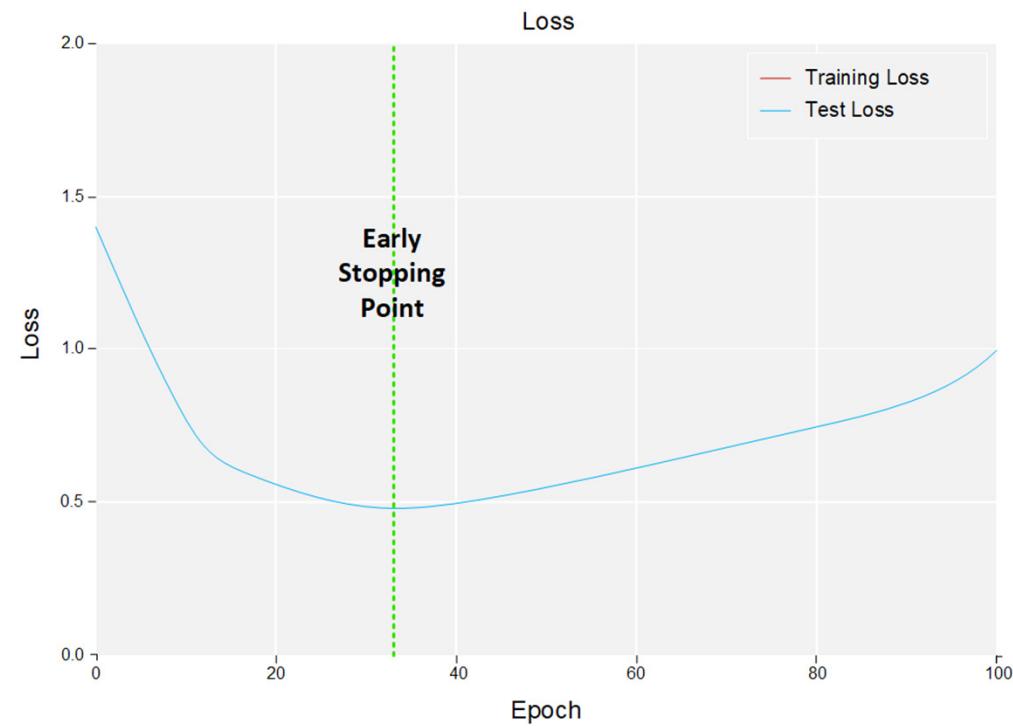
# Cross Validation

- Cross validation or also known as k-fold cross validation is a method of training where we split our dataset into k folds instead of a typical training and test split.
- For example, let's say we're using 5 folds. We train on 4, and use the 5<sup>th</sup> final fold as our test. We then train on the other 4 folds, and test on another.
- We then use the average weights across coming out of each cycle.
- Cross Validation reduces overfitting but slows the training process

**k-folds (5 shown)**



# Early Stopping



532

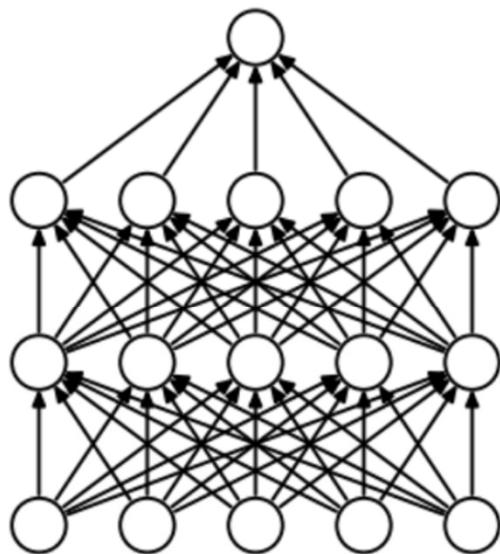


# Dropout

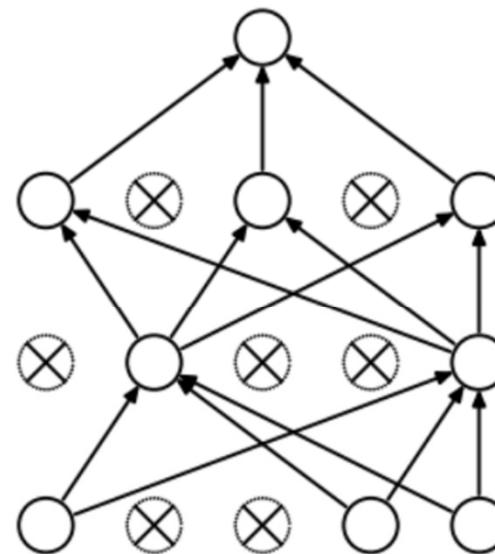
- Dropout refers to dropping nodes (both hidden and visible) in a neural network with the aim of reducing overfitting.
- In training certain parts of the neural network are ignored during some forward and backward propagations.
- Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons. Thus the NN learns more robust or meaningful features.
- In Dropout we set a parameter ' $P$ ' that sets the probability of which nodes are kept or  $(1-p)$  for those that are dropped.
- Dropout almost doubles the time to converge in training



# Dropout Illustrated



(a) Standard Neural Net

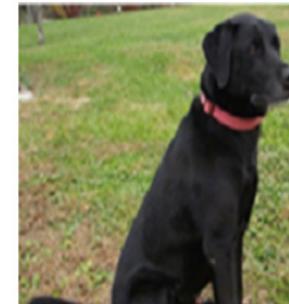
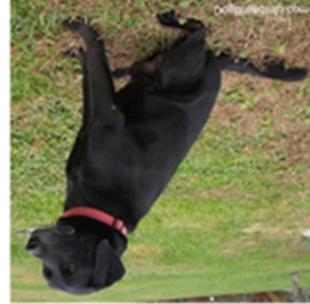


(b) After applying dropout.



# Data Augmentation

- Data Augmentation is one of the easiest ways to improve our models.
- It's simply taking our input dataset and making slight variations to it in order to improve the amount of data we have for training. Examples below.
- This allows us to build more robust models that don't overfit.



**6.9**

# **Epochs, Iterations and Batch Sizes**

Understanding some Neural Network Training Terminology



# Epochs

- You may have seen or heard me mention Epochs in the training process, so what exactly is an Epoch?
  - An Epoch occurs when the full set of our training data is passed/forward propagated and then backpropagated through our neural network.
  - After the first Epoch, we will have a decent set of weights, however, by feeding our training data again and again into our Neural Network, we can further improve the weights. This is why we train for several iterations/epochs (50+ usually)



# Batches

- Unless we had huge volumes of RAM, we can't simply pass all our training data to our Neural Network in training. We need to split the data up into segments or.....Batches.
- Batch Size is the number of training samples we use in a single batch.
- Example, say we had 1000 samples of data, and specified a batch size of 100. In training, we'd take 100 samples of that data and use it in the forward/backward pass then update our weights. If the batch size is 1, we're simply doing Stochastic Gradient Descent.



# Iterations

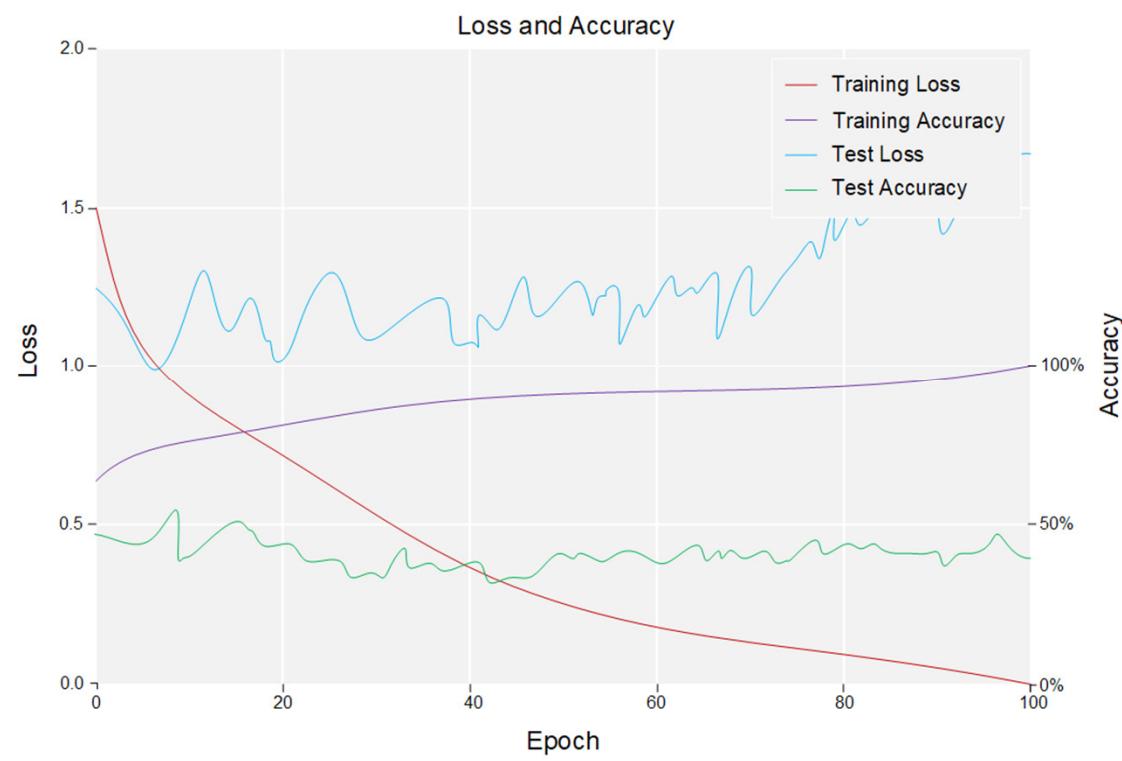
- Many confuse iterations and Epochs (I was one of them)
- However, the difference is quite simple, Iterations are the number of batches we need to complete one Epoch.
- In our previous example, we had 1000 items in our dataset, and set a batch size of 100. Therefore, we'll need 10 iterations ( $100 \times 10$ ) to complete one Epoch.

**6.10**

# **Measuring Performance**

How we measure the performance of our Neural Network

# Loss and Accuracy



541



## Loss and Accuracy

- It is important to realize while Loss and Accuracy represent different things, they are essentially measuring the same thing. The performance of our NN on our training Data.
- Accuracy is simply a measure of how much of our training data did our model classify correctly
  - $$\text{Accuracy} = \frac{\text{Correct Classifications}}{\text{Total Number of Classifications}}$$
- Loss values go up when classifications are incorrect i.e. different to the expected Target values. As such, Loss and Accuracy on the Training dataset WILL correlate.



## Is Accuracy the only way to assess a model's performance?

- While very important, accuracy alone doesn't tell us the whole story.
- Imagine we're using a NN to predict whether a person has a life threatening disease based on a blood test.
- There are now 4 possible scenarios.
  1. TRUE POSITIVE
    - Test Predicts **Positive** for the disease and the person **has the disease**
  2. TRUE NEGATIVE
    - Test Predicts **Negative** for the disease and the person **does NOT have the disease**
  3. FALSE POSITIVE
    - Test Predicts **Positive** for the disease but the person **does NOT have the disease**
  4. FALSE NEGATIVE
    - Test Predicts **Negative** for the disease but the person actually **has the disease**



## For a 2 or Binary Class Classification Problem

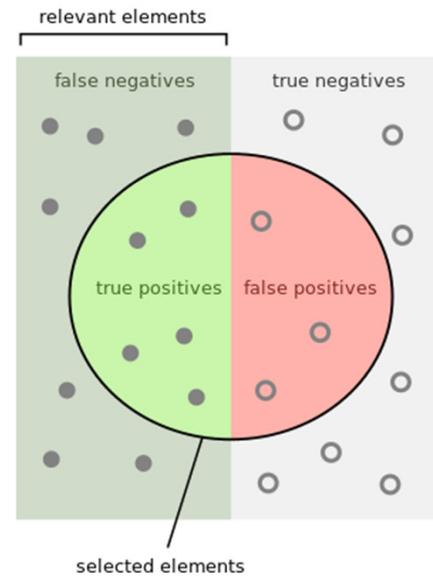
- Recall – How much of the positive classes did we get correct
  - $$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$
- Precision – Out of all the samples how much did the classifier get right
  - $$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$
- F-Score – Is a metric that attempts to measure both Recall & Precision
  - $$F - Score = \frac{2 \times Recall \times Precision}{Precision + Recall}$$



## An Example

Let's say we've built a classifier to identify gender (male vs female) in an image where there are:

- 10 Male Faces & 5 Female Faces
- Our classifier identifies 6 male faces
- Out of the 6 male faces - 4 were male and 2 female.
- Our Precision is  $4 / 6$  or 0.66 or 66%
- Our Recall is  $4 / (4 + 6)$  (**6 male faces were missed**) or 0.4 or 40%
- Our F-Score is  $= \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Precision} + \text{Recall}} = \frac{2 \times 0.4 \times 0.66}{0.4 + 0.66} = \frac{0.528}{1.06} = 0.498$



How many selected items are relevant?

$$\text{Precision} = \frac{\text{green}}{\text{green} + \text{red}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{green}}{\text{green} + \text{red}}$$

[https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)

**6.11**

## **Review and Best Practices**

Review on the entire training process and some general guidelines to designing your own Neural Nets

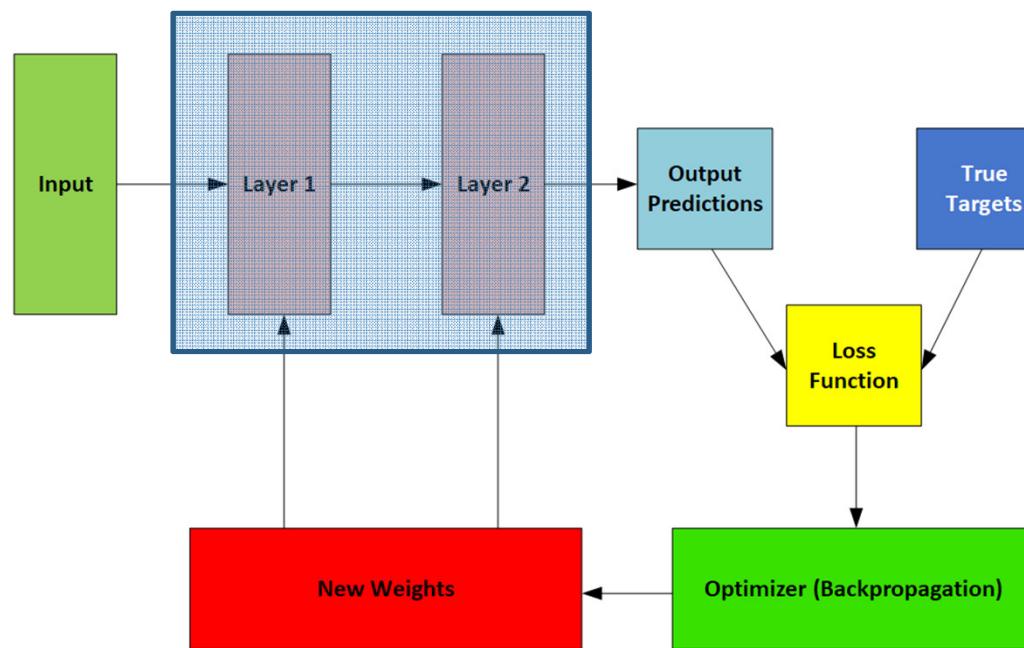


# Training step by step

1. Initialize some random values for our weights and bias
2. Input a single sample of our data or batch of samples
3. Compare our output with the actual value target values.
4. Use our loss function to put a value to our loss.
5. Use Backpropagation to perform mini batch gradient descent to update our weights
6. Keep doing this for each batch of data in our dataset (iteration) until we complete an Epoch
7. Complete several Epochs and stop training when the Accuracy on our test dataset



# Training Process Visualized





# Best Practices

- Activation Functions - Use ReLU or Leaky ReLU
- Loss Function - MSE
- Regularization
  - Use L2 and start small and increase accordingly (0.01, 0.02, 0.05, 0.1.....)
  - Use Dropout and set P between 0.2 to 0.5
- Learning Rate – 0.001
- Number of Hidden Layers – As deep as your machine's performance will allow
- Number of Epochs – Try 10-100 (ideally at least 50)

# A/B Testing A Fun Theoretical Example



# A/B Testing Introduction

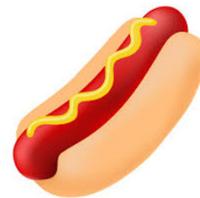
- A/B Testing has become almost a buzz word given how much it's thrown around in marketing circles and amongst web & UX/UI interfaces.
- But what exactly is it? Is it just comparing the results of A versus B? That doesn't sound too hard does it?
- It's not, but, there're a lot of things to consider when designing and analyzing your A/B Test
- Let's conduct a simple real world example to better understand



## Our Real Life A/B Test Example

- So now, let's test our understanding with looking at our real world example

**Hot Dogs!**



- We want to test which hot meat sausage is better between two meats.



# Our Hot Dogs

- Hot Dog Sausage (**A**) is what we've been using for years and it's worked fairly well.
- Along comes another brand of sausage meat (**B**) that claims to test better.



# Our Experiment Design

- So how do we go about testing which of these two hot dogs is best?
- Someone said A/B Testing? Correct! So how do we setup our experiment?

**Step 1 – Define our Hypothesis**

**Step 2 – How do we evaluate our experiment**

**Step 3 – How much of the effect do we want?**

**Step 4 – How many people or test subjects do we need?**



# Defining our Hypothesis



## Null Hypothesis

- There is no difference between the two hot dog sausage brands (A and B)

## Alternative Hypothesis

- The new hot dog sausage brand (B) is better than A.





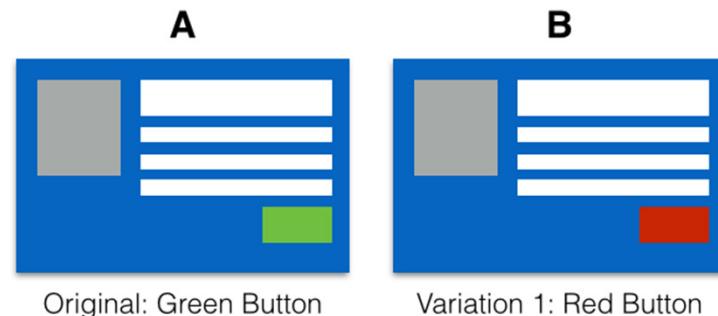
# Our Evaluation Metric

We need a way to define what our **test metric** is, in most cases this is quite simple. Did button A result in more clicks than button B, with click through rate being our metric.

However, in our hotdog experiment what should our metric be?

We can ask the tasters if they'd like a **second hotdog from the same batch?** This is an effective **metric**.

It removes some subjectivity of rating taste. Instead, a simple yes or no would suffice and it allows us to compare figures easily.





## Desired Effect

- Now, in our example the two hot dog sausages cost roughly the same so the cost impact is **negligible**.
- An example of understanding how desired effect comes into play, let's assume we can use Kobe beef sausages. Those are perhaps \$50 a sausage (if it even exists).
- We need to know if the cost investment of using B over A is worth the investment of B. B can perform 5% better than A but cost double. In a situation like that we need to work out the economics of whether our added benefit to determine if it's a feasible investment.

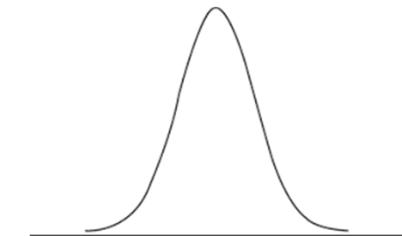
# Experiment Size



- This is where a lot A/B tests go wrong.
- Imagine we run an experiment on 10 people with our hotdog A only. In this example, we ask them the same question, “Would like another hotdog from the same batch”

| Group | Yes to another hotdog |
|-------|-----------------------|
| 1     | 4                     |
| 2     | 5                     |
| 3     | 6                     |
| 4     | 5                     |

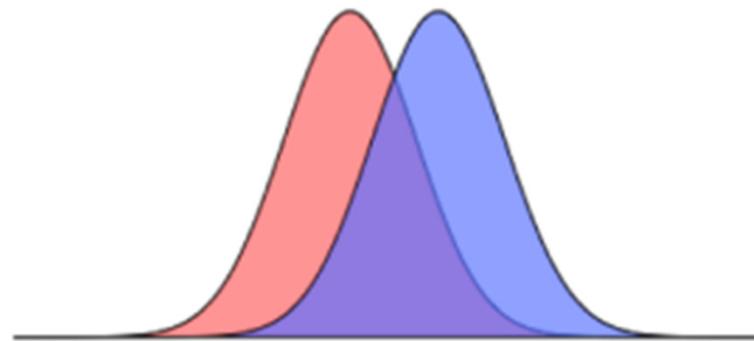
- Every time we run an experiment we can get different results. These results typically form a normal distribution curve.
- When testing for two different cases (A & B) how do we know if our results are real and not due to random variation?





## Experiment Size – Sample Size

- This is why we need to choose a sufficiently large sample size that allows us to know whether the difference in results is Statistically Significant





# Statistical Power

- When choosing sample size, we need to decide what the Statistical Power of our test will be.
- Statistical Power is the **probability of correctly rejecting the Null Hypothesis**
- Statistical Power is often referred to **1-Beta** as it's inversely proportional to making **Type II Error** (Beta Errors).
- Type II error is the probability of failing to reject the Null Hypothesis when you should have.
- We typically use a **statistical power of 80%** and don't worry, I'll explain what that 80% means shortly.
- Statistical power represents the **probability that you'll get a false negative**. A power of 0.80 means that there is an 80% chance that if there was an effect, we would detect it (or a 20% chance that we'd miss the effect)



# Statistical Significance Level

- Statistical Significance is how likely it is that the difference between your experiment's control version and test version **isn't due to error or random chance**.
- Basically it means, if we run an experiment test with a 95% significance level - you can be 95% confident that the differences are real and not attributed to chance.



## Baseline Metric and Effect Size

- Before moving ahead with testing variations of hotdogs, we need to establish our **baseline metric**.
- In our experiment, our **evaluation metric** was the number of people who wanted another hotdog after having one.
- But by what amount of change are looking to get from our new hotdogs? In our example let's look at getting a 25% increase.



# Calculating our Sample Size

```
▶ from scipy.stats import norm, zscore

def sample_power_probtest(p1, p2, power=0.8, sig=0.05):
    #two-sided t test
    z = norm.isf([sig/2])
    zp = -1 * norm.isf([power])
    d = (p1-p2)
    s = 2*((p1+p2) / 2)*(1-((p1+p2) / 2))
    n = s * ((zp + z)**2) / (d**2)
    return int(round(n[0]))

sample_power_probtest(p1=0.5, p2=0.75, power=0.8, sig=0.05)
```

□ 59

- Online Calculator - <http://www.evanmiller.org/ab-testing/sample-size.html>

P1 (baseline) = 50%  
P2 (p1+effect size) = 75%  
Statistical Power = 80%  
Significance Level = 5%

Sample Size N = 59



# Experiment Length

- In our example this doesn't apply, however one should be very careful with experiment lengths.
- In the business world there are many other factors that could mess with our results such as seasonality, trends etc.
- Example don't run a test during Christmas week or right before Mothers Day and expect your results to be relevant.
- However, one advantage of doing it during a peak time would be to get larger N over a shorter space of time.



# Back to our Hotdog Experiment

- So we ran our test on two groups of 60 persons (chosen at random)
- Null Hypothesis – Our new hotdog B does not make a difference.
- $H_0: \hat{d} = 0$  (*d 'hat' is the difference between the two groups*)
- $N_{control} = N_{experiment} = 60$

## Results

- $X_{control} = 33$
- $X_{experiment} = 42$
- $\hat{d} = \hat{p}_{experiment} - \hat{p}_{control} = \frac{42}{60} - \frac{33}{60} = 0.15$



## Discussing our Results

- $\hat{d} = \hat{p}_{experiment} - \hat{p}_{control} = \frac{42}{60} - \frac{33}{60} = 0.15$
- So we have a **0.15 improvement** with our Hotdog B, but was this by random chance or is this statistically significant?
- In order to test this, we need to calculate our **confidence interval** for the difference of the results of the two groups.
- This **interval** tells us the range of values the difference between the two groups can have.



# Calculating our Confidence Intervals

- We firstly need to calculate the pooled probability, which is really just the combined probability of the two samples.
- $\hat{p}_{pooled} = \frac{X_{control} + X_{experiment}}{N_{control} + N_{experiment}} = \frac{33 + 42}{120} = \frac{75}{120} = 0.625$
- The second and last step to get the Confidence Interval is getting the **Standard Error**. It estimates of how much variation the obtained results will have. This means how widespread the values in the distribution of the sample will be. We'll calculate the **Pooled Standard Error** which combines the standard error of both samples.
- $SE_{pooled} = \sqrt{\hat{p}_{pooled} \times (1 - \hat{p}_{pooled}) \times \left(\frac{1}{N_{control} + N_{experiment}}\right)}$
- $SE_{pooled} = \sqrt{0.625 \times (1 - 0.625) \times \left(\frac{1}{120}\right)} = 0.044$

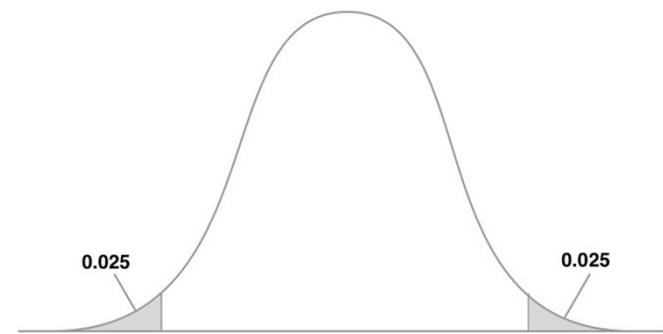


# Final Analysis

So now we have everything to get out confidence intervals and enough information to determine if we should reject our Null Hypothesis.

- $H_0: \hat{d} = 0$
- $H_1: \hat{d} > 1.96 \times StandardError_{pooled}$

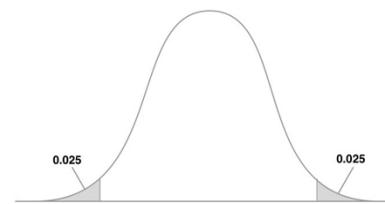
Remember we chose to use a 5% significance level? What we do now is split the 5% into the two ends of our normal distribution curve (we assume normally distributed data).





# Final Analysis – Obtaining our Z-Score

| <i>z</i> | .00   | .01   | .02   | .03   | .04   | .05   | .06   | .07   |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|
| -3.4     | .0003 | .0003 | .0003 | .0003 | .0003 | .0003 | .0003 | .0003 |
| -3.3     | .0005 | .0005 | .0005 | .0004 | .0004 | .0004 | .0004 | .0004 |
| -3.2     | .0007 | .0007 | .0006 | .0006 | .0006 | .0006 | .0006 | .0005 |
| -3.1     | .0010 | .0009 | .0009 | .0009 | .0008 | .0008 | .0008 | .0008 |
| -3.0     | .0013 | .0013 | .0013 | .0012 | .0012 | .0011 | .0011 | .0011 |
| -2.9     | .0019 | .0018 | .0018 | .0017 | .0016 | .0016 | .0015 | .0015 |
| -2.8     | .0026 | .0025 | .0024 | .0023 | .0023 | .0022 | .0021 | .0021 |
| -2.7     | .0035 | .0034 | .0033 | .0032 | .0031 | .0030 | .0029 | .0028 |
| -2.6     | .0047 | .0045 | .0044 | .0043 | .0041 | .0040 | .0039 | .0038 |
| -2.5     | .0062 | .0060 | .0059 | .0057 | .0055 | .0054 | .0052 | .0051 |
| -2.4     | .0082 | .0080 | .0078 | .0075 | .0073 | .0071 | .0069 | .0068 |
| -2.3     | .0107 | .0104 | .0102 | .0099 | .0096 | .0094 | .0091 | .0089 |
| -2.2     | .0139 | .0136 | .0132 | .0129 | .0125 | .0122 | .0119 | .0116 |
| -2.1     | .0179 | .0174 | .0170 | .0166 | .0162 | .0158 | .0154 | .0150 |
| -2.0     | .0228 | .0222 | .0217 | .0212 | .0207 | .0202 | .0197 | .0192 |
| -1.9     | .0287 | .0281 | .0274 | .0268 | .0262 | .0256 | .0250 | .0244 |
| -1.8     | .0359 | .0351 | .0344 | .0336 | .0329 | .0322 | .0314 | .0307 |
| -1.7     | .0446 | .0436 | .0427 | .0419 | .0410 | .0401 | .0392 | .0384 |



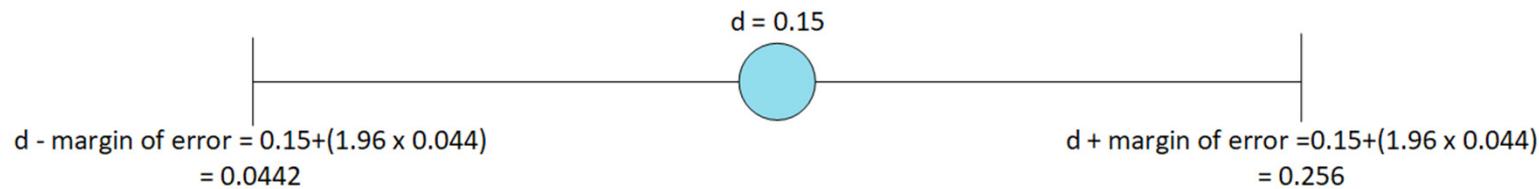
$$H_1: \hat{d} > 1.96 \times StandardError_{pooled}$$

- Notice the 1.96 in our Alternative Hypothesis?
- We used the Z-Table to get the value for 0.025
- Therefore, in order to reject the Null Hypothesis, we want to prove that the difference observed, is great than a certain interval around the value that refers to 5% of the results being due to chance.
- Hence why we use the z-score and the standard error.



# So are our new Hotdogs better?

- $H_1: 0.15 > 1.96 \times 0.044 \equiv 0.15 > 0.08624$
- So yes, we have statistically proven with our AB test that the **new hotdogs (B) is better than A.**
- Before we say an emphatic yes, to using our new hotdogs, let's dig into the interpretation of our results, given our specified parameters.



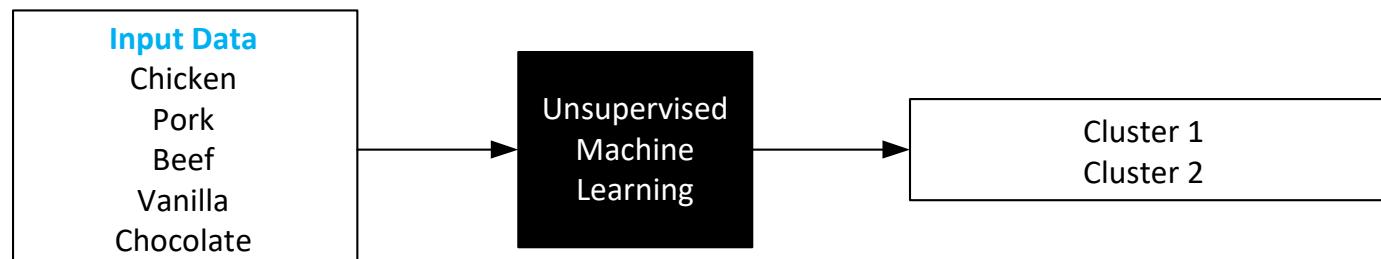
- Remember we chose a minimum effect size of 25%, as we can see given our Standard Error, the lower bound (left) is a situation where the minimum effect is not guaranteed as it's significantly less than 0.15
- We can therefore conclude that our new hotdogs have a strong possibility of resulting in 25% more sales

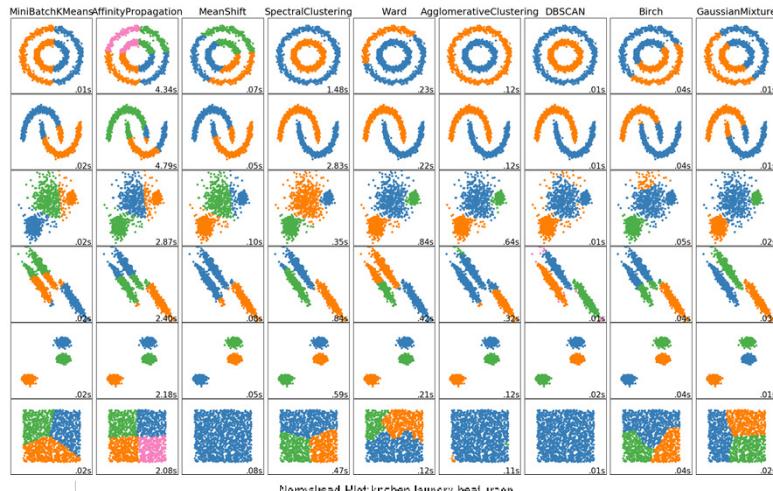
# **Clustering – Unsupervised Learning**



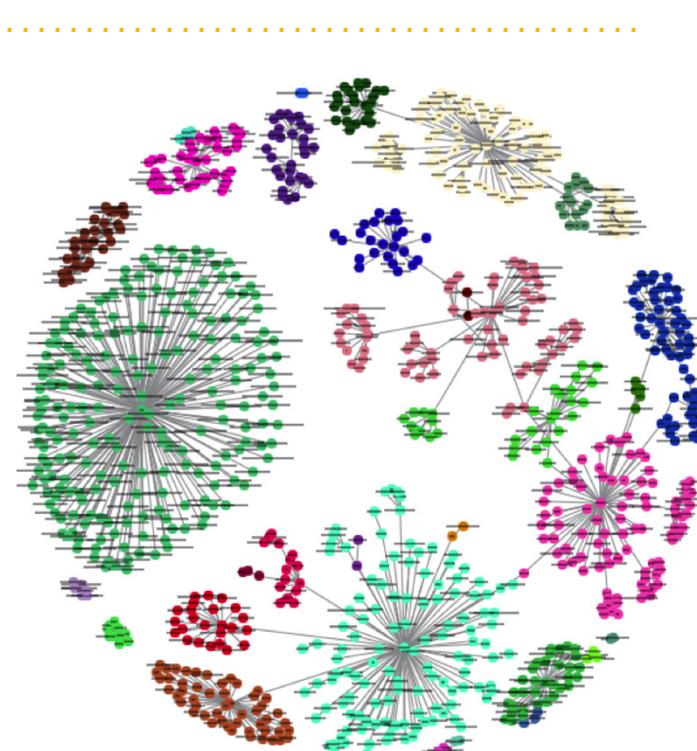
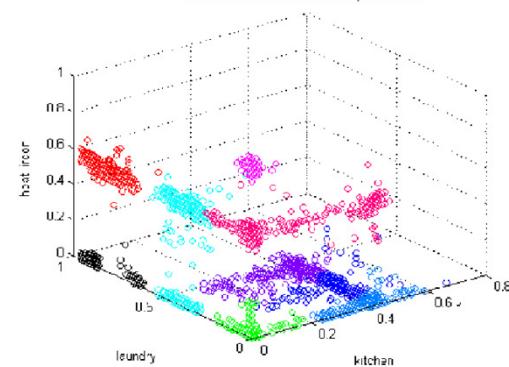
# Unsupervised Learning

- Unsupervised learning is concerned with finding interesting clusters of input data. It does so **without any help of data labeling**.
- It does this by creating interesting transformations of the input data
- It is very important in data analytics when trying to understand data
- Examples in the Business world:
  - Customer Segmentation





Normalised Plot kitchen laundry heat iron





# Goal of Clustering

- The goal of clustering is if given a set of data points, we can **classify each data point into a specific group or cluster.**
- We can use clustering analysis to gain some valuable insights from our data by seeing what groups the data points fall into naturally by using our clustering algorithms.
- *A cluster refers to a collection of data points aggregated together because of certain similarities.*
- There are several types of clustering methods which we'll now discuss.



# Types of Clustering Algorithms

There are many types of clustering algorithms, some far more widely used than others, however we will discuss 5 main types:

- K-Means Clustering
- Agglomerative Hierarchical Clustering
- Mean-Shift Clustering
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
- Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM)

# K-Means Clustering



# K-Means Clustering Algorithm

- K-Means is perhaps the most popular clustering algorithm in existence.
- It is extensively used in real world applications
- It's relatively simple, intuitive and great for beginners to conceptualize some machine learning concepts.
- Let's take a look at how it works!

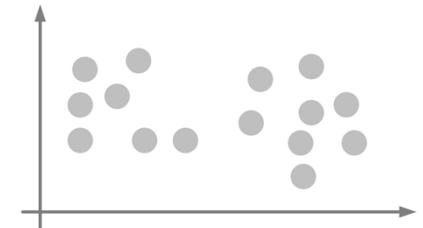


## K-Means Clustering – Step 1

Choose the number of clusters you wish to identify by choosing k

- Choosing K can be done either intuitively or via the **Silhouette Method or the Elbow method** (we'll discuss this next)
- You can choose K intuitively by:
  - **Understanding the data domain** – e.g. if you're trying to cluster amongst newspaper articles you might have quite a few types, whereas if you're clustering on customer types, it might be better to start with a smaller k (less than 5)
  - **Exploring it Visually** to see if we can spot natural clusters

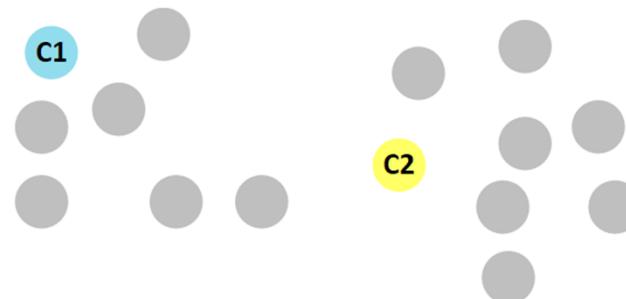
Let's try k=2





## K-Means Clustering – Step 2

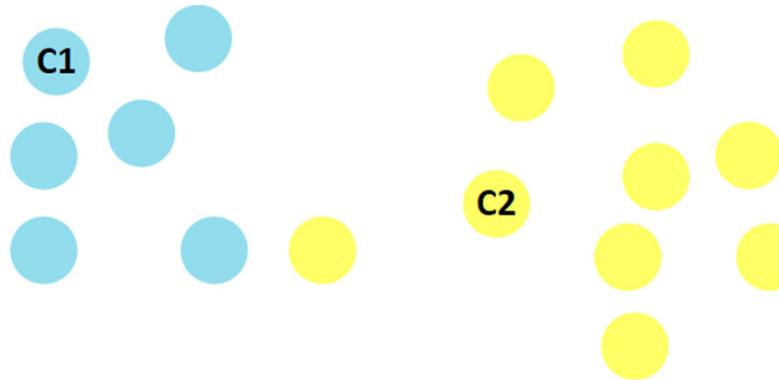
- Once we have  $k$  (which was equal to 2), we select  $k$  random points from our dataset and use these as centroids.
- Here we have  $c_1$  (blue) and  $c_2$  (yellow) that represent the centroid of these two clusters





## K-Means Clustering – Step 3

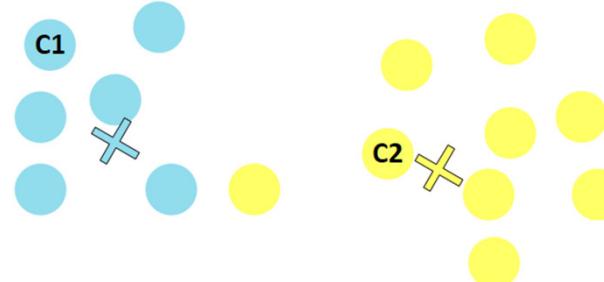
- Assign all the points to the closest cluster centroid
- So all points closest to C1 get assigned to the blue cluster and all points closest to C2 get assigned to the yellow cluster.





## K-Means Clustering – Step 4

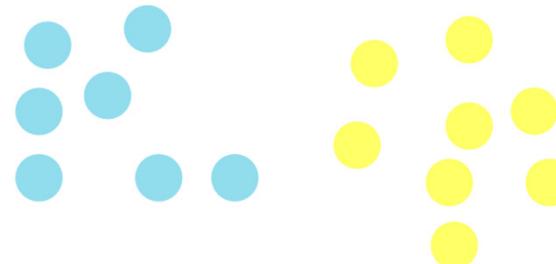
- We now compute the centroid of the newly formed clusters.
- The blue and yellow crosses represent the center of the newly formed clusters





## K-Means Clustering – Step 5

- **Repeat Step 3 & 4** – We now use the new centroids (the yellow and blue crosses) as the cluster centers and then assign the closest points to each centroid's cluster.
- We keep repeating this step until the newly formed clusters stop changing, all points remain in the same cluster and/or the number of specified iterations is reached.





## K-Means Clustering Algorithm Advantages

- Relatively simple to implement.
- Scales to large data sets.
- Guarantees convergence.
- Can warm-start the positions of centroids.
- Easily adapts to new examples.
- Generalizes to clusters of different shapes and sizes, such as elliptical clusters.



## K-Means Clustering Algorithm Disadvantages

- We still need to choose K manually
- It is dependent on initial values
- Can run into problems when clustering varying sizes and density
- Sensitive to outliers
- Doesn't scale well with large number of dimensions (can be mitigated by using PCA)
- Only works for numeric values, as such categorical values will either have to be translated into some numerical meaning (e.g. high, medium, low can be mapped to 3,2,1) this can't always work for categories like types of fruit. Alternatively, we can use K-Medians, or K-Modes to alleviate this issue..

# **Choosing K – Elbow Method & Silhouette Analysis**

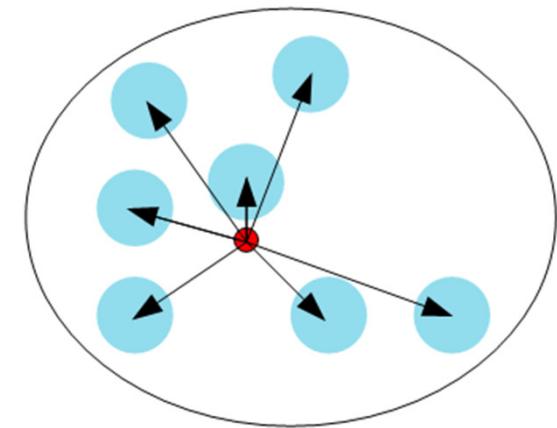


## Choosing K

- As I stated in the previous section, choosing K is very important and should be guided by your knowledge of the dataset together with some guidance by some more scientific methods.
- We can get a rough idea on what k is good by using the two following methods:
  - Elbow Method
  - Silhouette Analysis

## Elbow Method

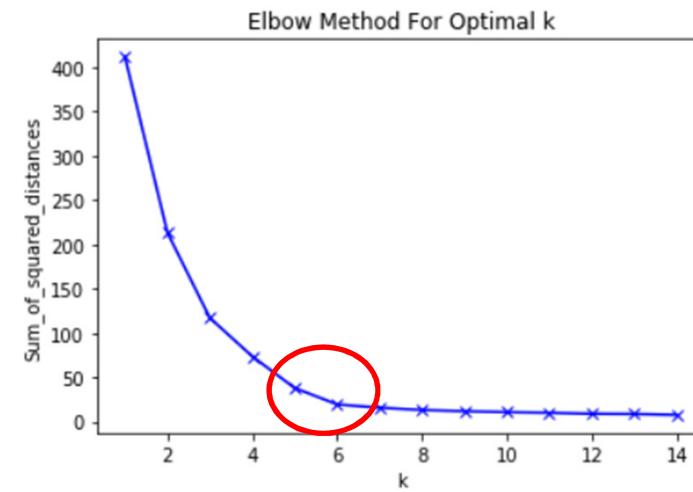
- The Elbow method works by running the clustering algorithm with a preset of cluster values (say 2 to 10).
- It then **computes the cost function** (or any other evaluation metric)
- Typically we use the **Inertia** (the within-cluster sum of distances from the centroid) and plot in a graph with the x-axis being the number of clusters and the y-axis being our evaluation metric. Small inertia is good.





## Elbow Method

- The ideal number of clusters is obtained when the addition of a new cluster doesn't significantly increases the cost function.
- This can be visually illustrated by looking on the 'elbow' of the line.
- In our plot we can see the elbow lies at **k = 5 and k=6**.





# Silhouette Analysis

- The Silhouette Analysis is a method of validating the consistency within clusters of data.
- The technique provides a succinct graphical representation of how well each object has been classified
- In this method a graph is plotted measuring how close the points in some cluster are to points of the other clusters.
- When the Silhouette coefficient is **near +1** this indicates that the points are far way from the other clusters
- When Silhouette coefficient is **near 0**, it indicates that the points are very close or intersecting other clusters.

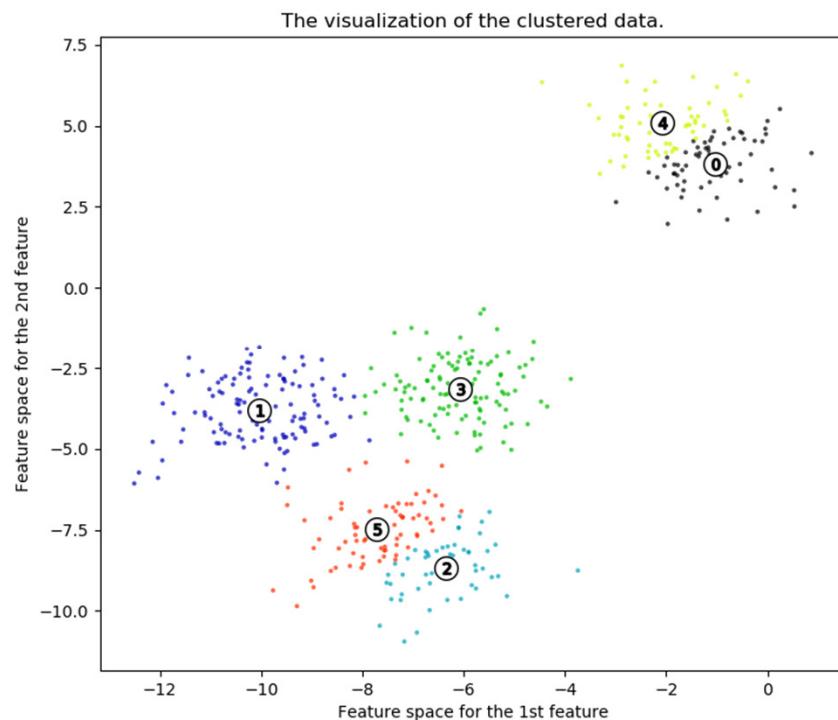
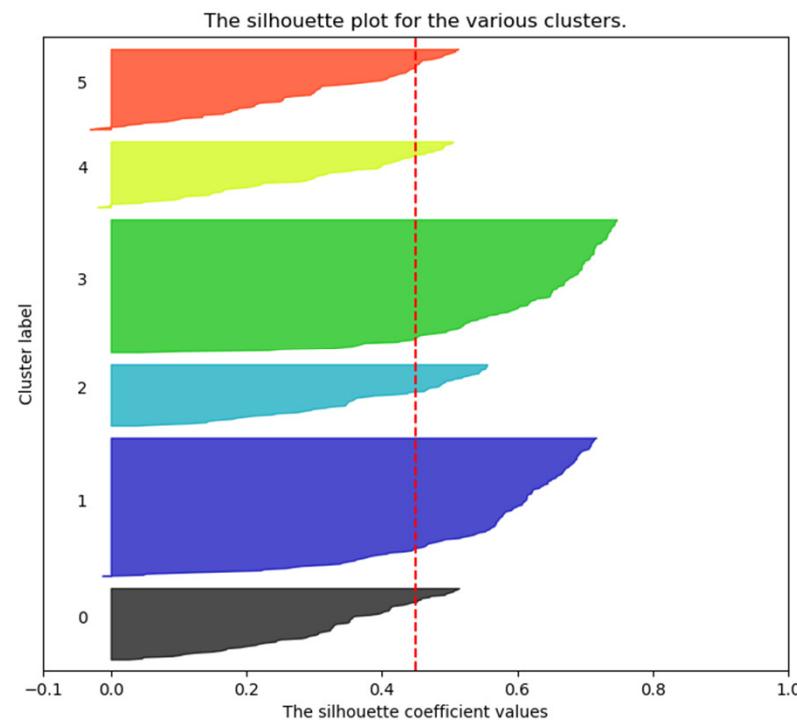


# Silhouette Analysis

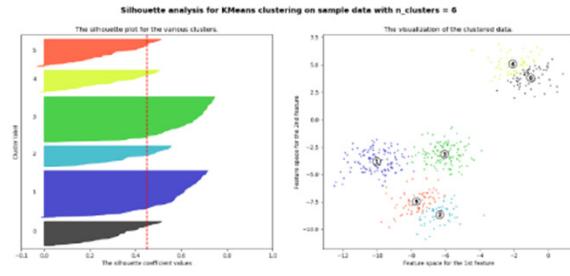
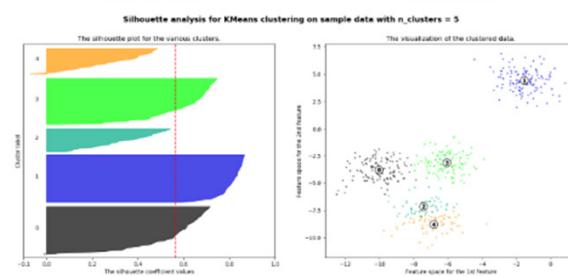
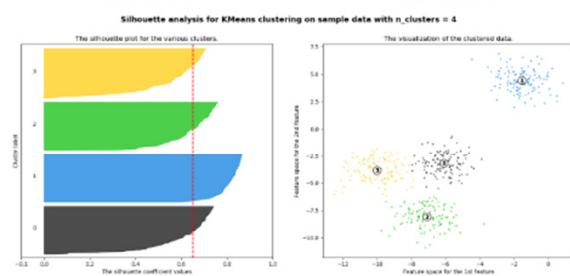
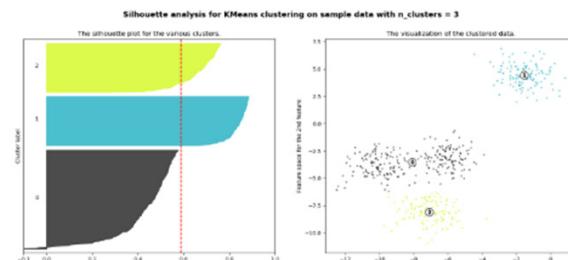
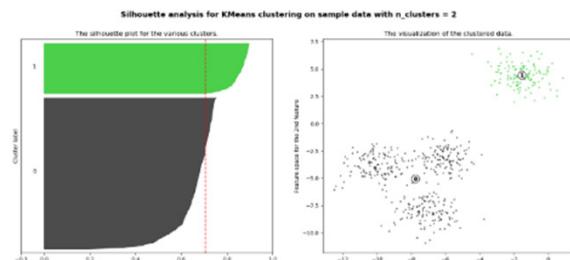
- The Silhouette Analysis is a method of validating the consistency within clusters of data.
- To calculate the Silhouette coefficient we need to define the mean distance of a point to all other points in its cluster (**a(i)**) and also define the mean distance to all other points in the closest cluster (**b(i)**). So, the Silhouette coefficient is:
- $$s(i) = \frac{(b(i)-a(i))}{\max(b(i),a(i))}$$



### Silhouette analysis for KMeans clustering on sample data with n\_clusters = 6



Source: [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_silhouette\\_analysis.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html)



n\_clusters = 2 The average silhouette\_score is : 0.704  
n\_clusters = 3 The average silhouette\_score is : 0.588  
n\_clusters = 4 The average silhouette\_score is : 0.650  
n\_clusters = 5 The average silhouette\_score is : 0.563  
n\_clusters = 6 The average silhouette score is : 0.450

Source: [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_silhouette\\_analysis.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html)

# **Agglomerative Hierarchical Clustering**

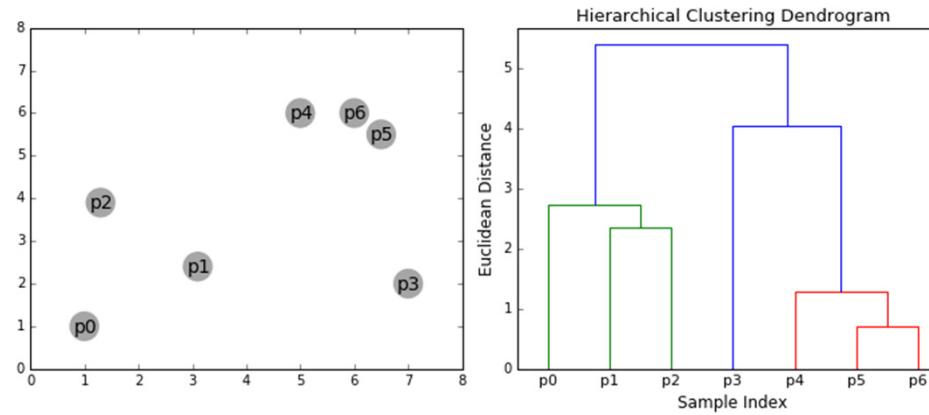


## Agglomerative Hierarchical Clustering Introduction

- Hierarchical clustering algorithms is a bottom-up approach to clustering.
- Bottom-up algorithms treat each data point as a single cluster at the beginning and then successively merge or agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data points.
- This hierarchy of clusters is represented as a tree or often referred to as a dendrogram.
- The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.



## Agglomerative Hierarchical Clustering Introduction

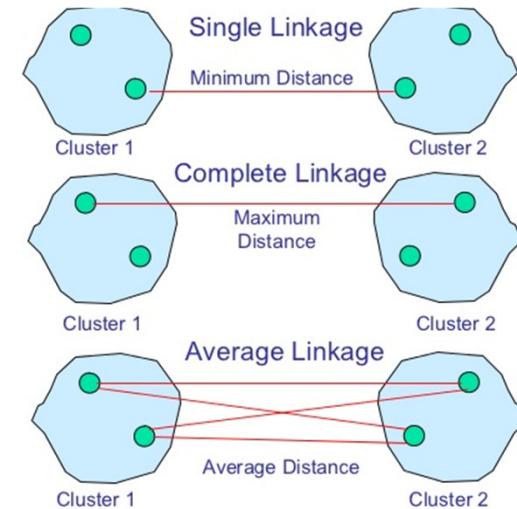


<https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68>



## Agglomerative Hierarchical Clustering: Step 1

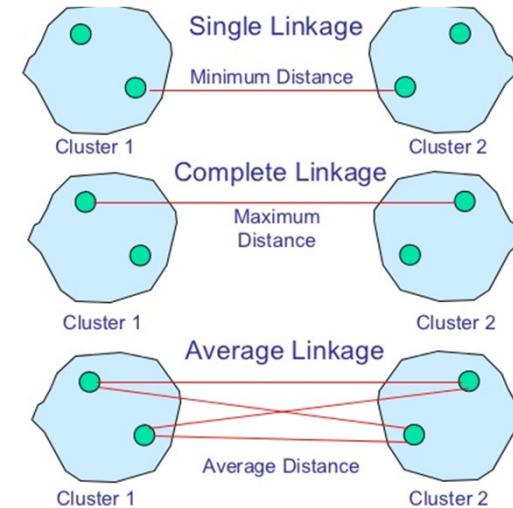
- We first begin by treating each data point as a single cluster
- For X data points we have X clusters.
- We use a distance metric (pre-selected) that measures the **distance between two clusters**.
- A commonly used metric is **average linkage** which defines the distance between two clusters to be the average distance between data points in the first cluster and data points in the second cluster.





## Agglomerative Hierarchical Clustering: Step 2

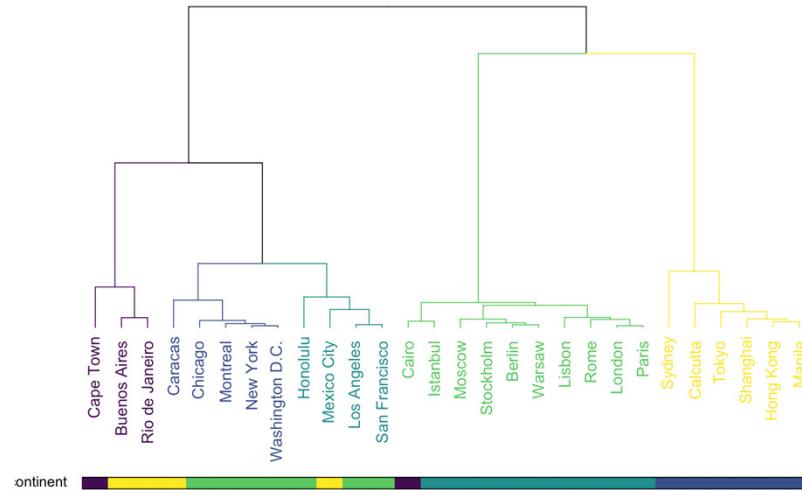
- For each iteration, we combine two clusters into one.
- The two clusters to be combined are selected as those with the smallest average linkage – this means that according to our average linkage distance metric, these two clusters have the smallest distance between each other and therefore are the most similar and should be combined.





## Agglomerative Hierarchical Clustering: Step 3

- We repeat Step 2 until we reach the root of the tree, which is the final cluster that encapsulates all data points.





## Advantages of Agglomerative Hierarchical Clustering

- We don't need to specify the number of clusters,  $k$
- The algorithm is not too sensitive to the choice of distance metric with all of them working equally well in most cases.
- It works well in uncovering naturally hierarchical data

### Disadvantages:

- It is quite slow and doesn't scale well

# Mean-Shift Clustering



# Mean-Shift Clustering

- Mean shift clustering is a sliding-window-based algorithm that attempts to find dense areas of data points.
- It is a centroid-based algorithm whereby its objective is to locate the center points of each cluster.
- It does this by **updating candidates for center points to be the mean of the points within the sliding-window**.
- These candidate windows are then filtered in a post-processing stage to eliminate near-duplicates, forming the final set of center points and their corresponding groups



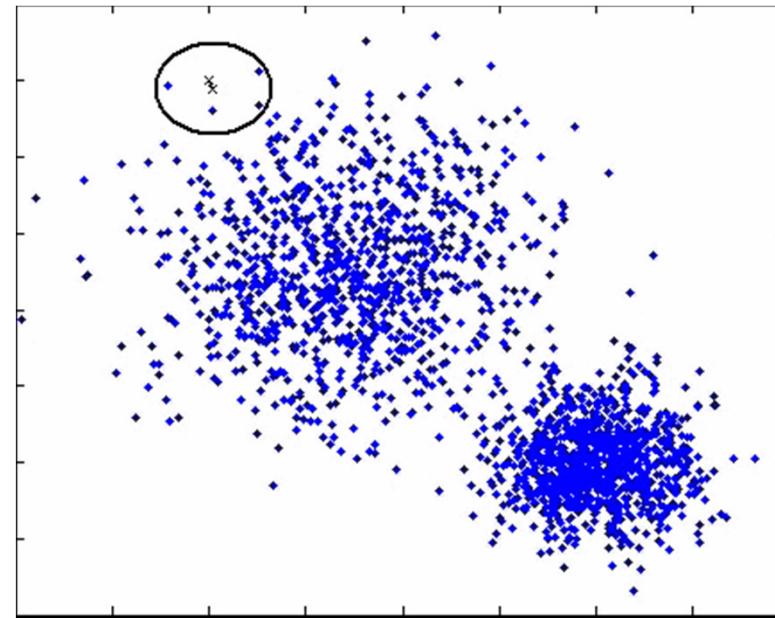
# Mean-Shift Clustering Steps

Let's start with thinking about a set of 2-dim x,y points.

1. We first initialize a circular sliding window (radius r) starting at a randomly selected point.
2. Think of Mean Shift as a density finding algorithm, that keeps shifting the window to higher and higher densities of points until it converges.
3. The density within the sliding window is proportional to the number of points inside it.
4. The sliding window is shifted according to the mean until there is no direction at which a shift can accommodate more points inside the kernel.
5. We do this entire process above is done with a few sliding windows until all points lie within a window. When multiple sliding windows overlap the window containing the most points is preserved. The data points are then clustered according to the sliding window in which they reside.



## Mean-Shift Clustering Demo



604

# **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**



## Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

- DBSCAN is similar to Mean Shift, where it too is a density-based clustered algorithm
- However, it has a few notable advantages.



## DBSCAN – Step 1 & 2

### Step 1

- DBSCAN starts off with an arbitrary starting data point that has not been visited.
- The neighborhood of this point is extracted using a distance epsilon  $\epsilon$  (All points which are within the  $\epsilon$  distance are neighborhood points)

### Step 2:

- If there are a sufficient number of points (according to minPoints) within this neighborhood then the clustering process starts and the current data point becomes the first point in the new cluster.
- If not, the point will be labeled as noise and the point is marked as “visited”.



## DBSCAN – Step 3 & 4

### Step 3

- For this first point in the new cluster, the points within its  $\epsilon$  distance neighborhood also become part of the same cluster. This procedure of making all points in the  $\epsilon$  neighborhood belong to the same cluster is then repeated for all of the new points that have been just added to the cluster group.

### Step 4

- This process of steps 2 and 3 is repeated until all points in the cluster are determined, this occurs when all points have been visited and assigned a cluster.



## DBSCAN – Step 5

### Step 5

- Once we're done with the current cluster, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.
- This process repeats until all points are marked as visited. Since at the end of this all points have been visited, each point will have been marked as either belonging to a cluster or being noise.



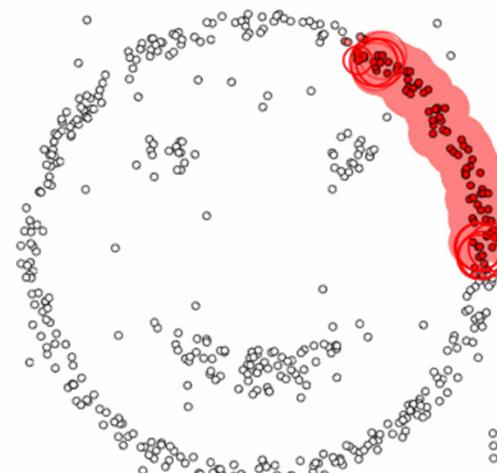
# DBSCAN Demo

epsilon = 1.00  
minPoints = 4

Restart



Pause





## DBSCAN Advantages and Disadvantages

### Advantages:

- No preset K number of clusters needs to be set
- It can identify outliers
- It can find erratically sized and shaped clusters well

### Disadvantages:

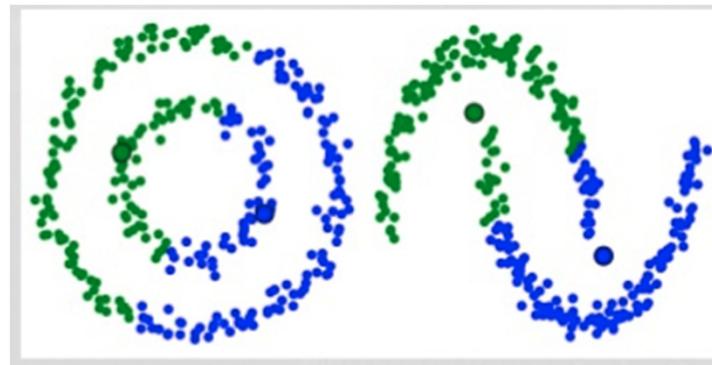
- It doesn't perform well when clusters are of varying densities (due to the setting of  $\epsilon$  and minPoints for identifying the neighborhood points, as these will vary from cluster to cluster when they have different densities).
- High-Dimensional data poses problems when choosing  $\epsilon$

# **Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM)**



## Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM)

- EM Clustering solves one the main weaknesses of K-Means, which is it's naïve use of the mean value for the cluster center.
- The image below shows two scenarios where K-means fails due to the mean values of the clusters are tightly packed.





## Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM)

- Gaussian Mixture Models (GMM) allows more flexibility than K-Means by assuming the points are Gaussian Distributed.
- This way we now have two parameters to describe the cluster shape, the mean and standard deviation.



## EM Clustering Using GMM **Steps 1 & 2**

### Step 1

- Like K-Means, we begin by selecting the number of clusters and randomly initializing the Gaussian distribution parameters for each cluster.

### Step 2

- Given these Gaussian distributions for each cluster, we compute the probability that each data point belongs to a particular cluster. The closer a point is to the Gaussian's center, the more likely it belongs to that cluster.
- This works well for Normally Distributed data as we are assuming that most of the data lies closer to the center of the cluster.



## EM Clustering Using GMM Steps

### Step 3

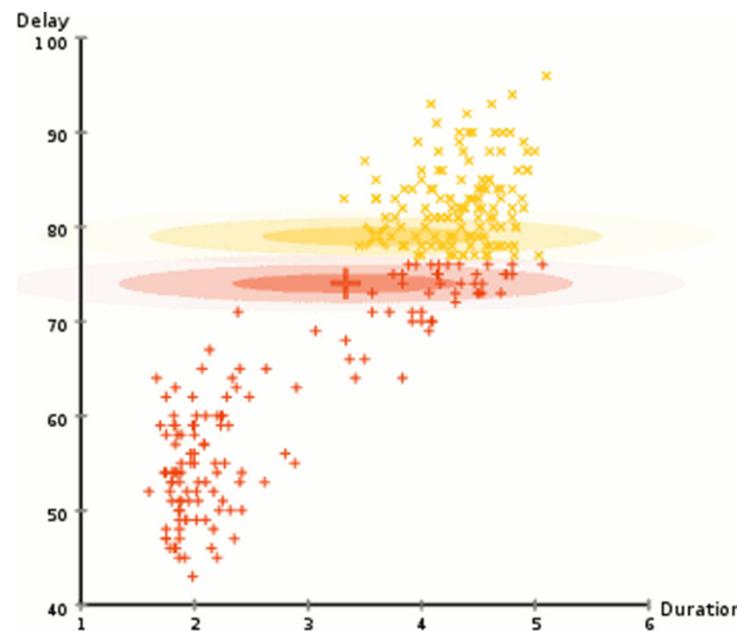
- Based on these probabilities a new set of parameters for the Gaussian distributions is computed such that we maximize the probabilities of data points within the clusters.
- We compute these new parameters using a weighted sum of the data point positions, where the weights are the probabilities of the data point belonging in that particular cluster.

### Step 4

- We repeat Step 2 and 3 iteratively until convergence.



## EM Clustering Using GMM Demo

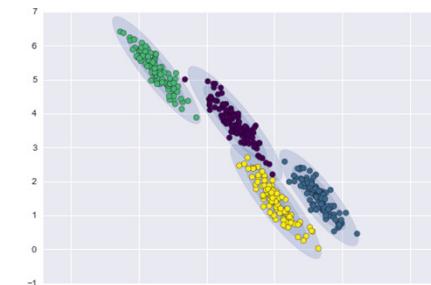




## EM Clustering Using GMM Advantages and Disadvantages

### Advantages

- GMMs are a lot more flexible in terms of cluster covariance than K-Means
- the clusters can take on any ellipse shape, rather than being restricted to circles
- Due to the use of probabilities data points can have multiple clusters e.g. a point can have 0.65 probability in being in one cluster and 0.35 in another.



### Disadvantages

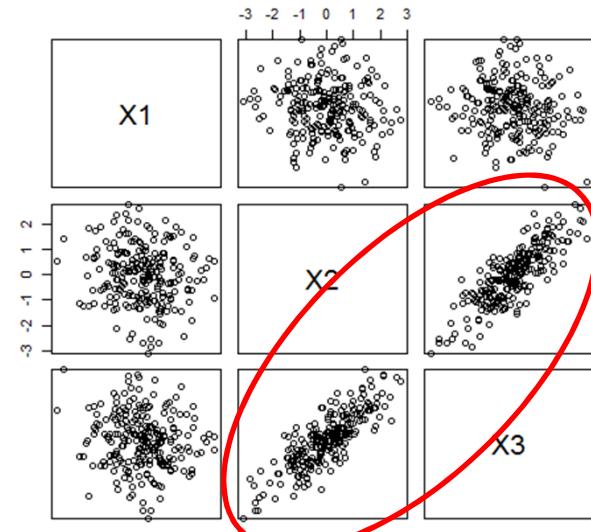
- Choosing K and scaling to higher dimensions

# Principal Component Analysis



## Why do we need PCA?

- Many times when working with large datasets with **many dimensions** things get too **computationally expensive** and confusing to keep track off
- However, many times we look at the data itself, we see variables that are **strongly correlated** and hence possibly redundant. E.g. a persons height and weight
- What if there was a way to **reduce the dimensionality** of our data while still retaining it's information?
- That is what **Principal Component Analysis** achieves





# What is PCA Exactly?

- PCA is an algorithm that compresses your dataset's dimensions from a **higher to lower dimensionality**.
- It does this based on the **eigenvectors** of the variance in your dataset
- PCA is extremely used in **Data Compression** saving loads in processing time, as well as in **Visualizations** of high dimensional data in 2 or 3 dimensions. This is very helpful when doing **cluster analysis**.
- More technically, PCA finds a **new set of dimensions** such that:
  - All the dimensions are orthogonal (and thus linearly independent)
  - Ranked according to the variance of data along them. This means the first principal component contains the most information about the data.



## How does PCA Work?

1. Calculate the Covariance Matrix ( $X$ ) of the our data points (i.e. how our variables all relate to one another)
  2. Calculate the **Eigenvectors** and their **Eigenvalues**
  3. Sort our Eigenvectors according to their Eigenvalues in from largest to smallest.
  4. Select a set of Eigenvectors ( $k$ ) to represent our data in k-dimensions
  5. Transform our original n-dimensional dataset into k-dimensions
- 
- What exactly are Eigenvectors & Eigenvalues?



## Eigenvectors & Eigenvalues

- The **eigenvectors** or principal components of our covariance matrix represent the **vector directions of the new feature space**.
- The **eigenvalues** represent the **magnitudes** of these vectors.
- As we are looking at our covariance matrix the eigenvalues thus quantify the **contributing variance of each vector** to the overall variance in our dataset

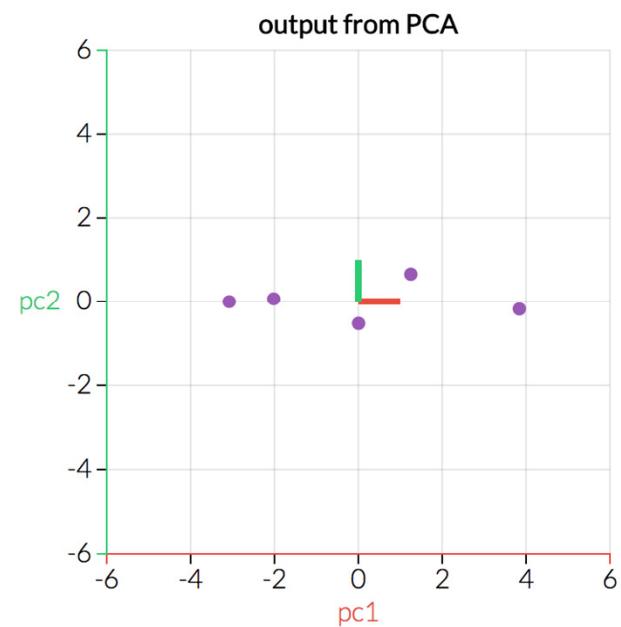
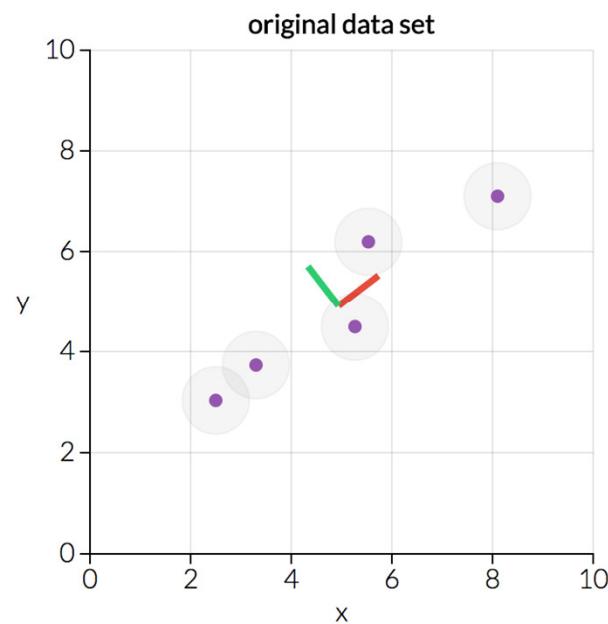
# Understanding them intuitively

- Imagine you have a bunch of people. Every person has a group of friends they talk to, with some frequency. And every person has a credibility rating (though the same person may have different credibility ratings from different people).
- Distribute an amount  $X$  of gossip to each person, and let them talk to each other.
- The largest eigenvalue gives you an idea of the fastest rate at which gossip can grow in this social circle.
- The corresponding eigenvector gives you an idea of how much gossip each person should start with in order to obtain this maximal growth rate. (In particular, if you want a story to spread rapidly, the largest components of the principal eigenvector identify who you should tell the story to)

<https://www.quora.com/What-is-the-physical-meaning-of-the-eigenvalues-and-eigenvectors>

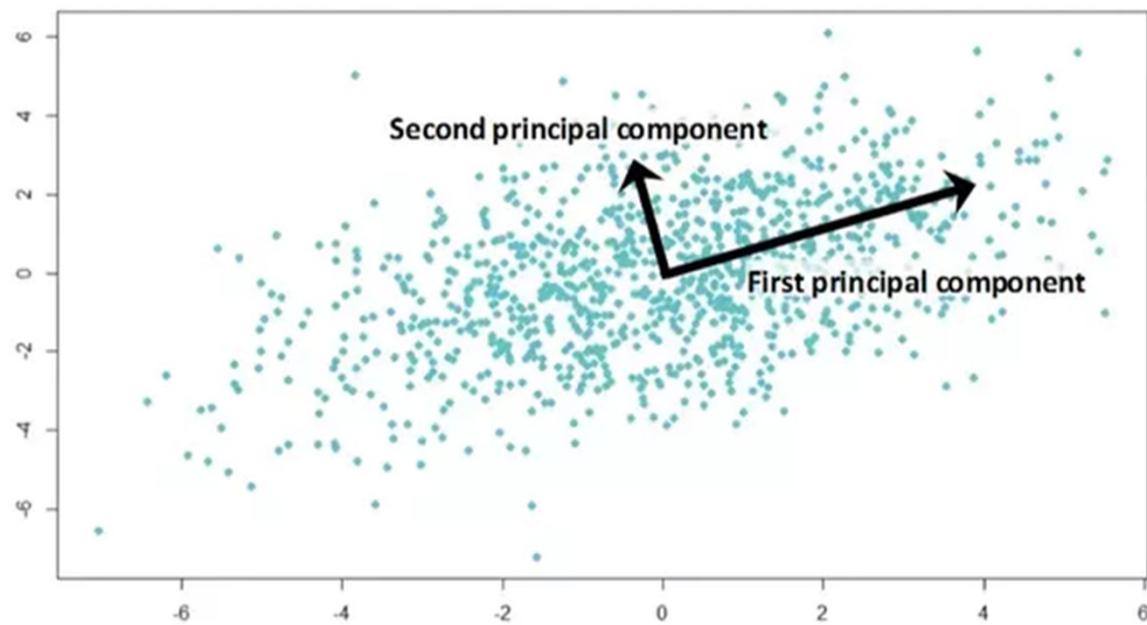


# PCA Output



625

# PCA Output



<https://medium.com/@sadatnazrul/the-dos-and-donts-of-principal-component-analysis-7c2e9dc8cc48>



## Points to Remember about PCA

- Always normalize your dataset before performing PCA
- Principal components are always linearly independent
- Every principal component will be Orthogonal/Perpendicular to every other principal component
- Using PCA we can use k-dimensions to represent some value (always less than 100%, your goal would be to get it as close to 100% as possible with as small as k as possible)
- Numerically we compute PCA using SVD (Singular Value Decomposition)
- The goal of PCA is to create a new data that represents the original with some loss due to compression/reduced dimensionality
- $\text{New Data}_{k \times n} = [\text{top } k \text{ eigenvector}]_{k \times m} [\text{original dataset}]_{m \times n}$



## PCA Disadvantages

- PCA works only if the observed variables or data is linearly correlated. If there is no correlation within our dataset PCA will fail to find the adequate components to represent our data with less dimensions
- PCA by nature is lossy and thus information will be lost when we reduce dimensionality
- It is sensitive to scaling which is why all data should be normalized
- Visualizations are hard to interpret real meaning as they do not relate to the original data features.



# PCA In Python!

629

# **t-Distributed Stochastic Neighbor Embedding (t-SNE)**



## **t-Distributed Stochastic Neighbor Embedding (t-SNE)**

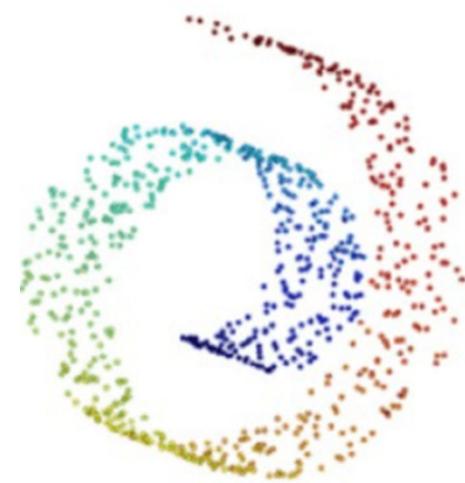
### **Introduction**

- t-SNE is another Dimensionality Reduction algorithm that is very popular.
- Published in 2008, in practice it has proven more effective than PCA.



## Why is t-SNE better than PCA?

- PCA is almost the defacto standard in dimensionality reduction tasks and is a part of possibly every Machine Learning class.
- PCA is good as it creates low-dimensional embeddings that preserve the overall variance of the dataset.
- However, PCA is a Linear Projection which means it's unable to capture non-linear dependencies.
- For example, PCA would be unable to unroll the data structure (right)
- t-SNE is not limited to linear projections which makes it applicable to many more datasets.





## How does t-SNE Work?

- t-SNE uses local structure. It attempts to map points of higher dimension onto a lower dimensional space so that the distances between points remain almost the same.
- There are in fact other dimensionality reduction algorithms that attempt this such as Local Linear Embeddings and Kernel PCA. However, t-SNE works best in practice.
- The reason t-SNE works so well is that it solves the crowding problem caused by the curse of dimensionality. When mapping these higher dimensional points onto a lower dimension we end up with all the points squashed, causing crowding.
- t-SNE solves this by making the optimization spread-out.



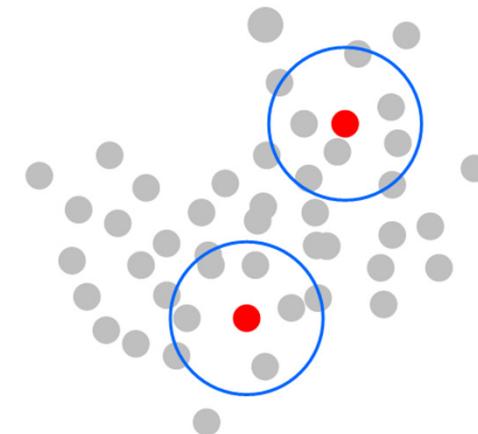
## How does t-SNE Work?

- Additionally t-SNE uses **stochastic neighbors**
- This allows us to have no clear line between which points are neighbors of the other points.
- The lack of defined boundaries is a huge advantage as it allows t-SNE to naturally take both the global and local structure into account.
- The local structure is more important than global structure, however the points that are far away aren't ignored completely. This allows for a well-balanced dimensionality reduction.

## The t-SNE algorithm – Step 1



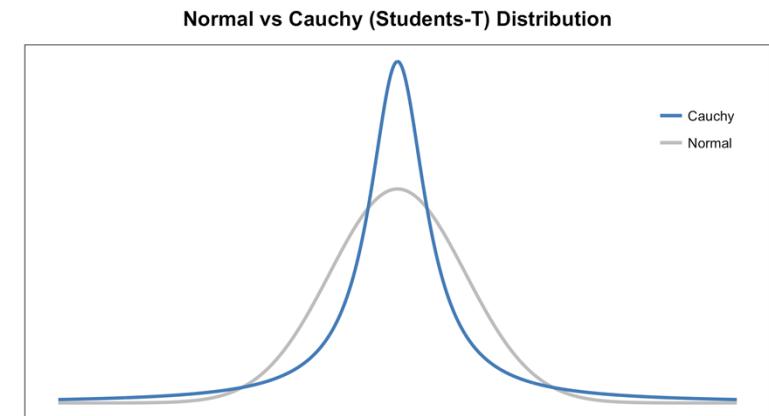
- Firstly, we measure similarities between points in the higher dimensional space.
- Imagine we have a bunch of 2d points scattered (see right). For each point we center a Gaussian distribution over the point.
- We then measure the density of all points under that Gaussian distribution.
- This is effectively converting the high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities.





## The t-SNE algorithm – Step 2

- We perform Step 1, however this time we use a Student t-distribution with one degree of freedom (also called a Cauchy distribution)
- The t-distribution shape allows better modeling for distances that are far apart.



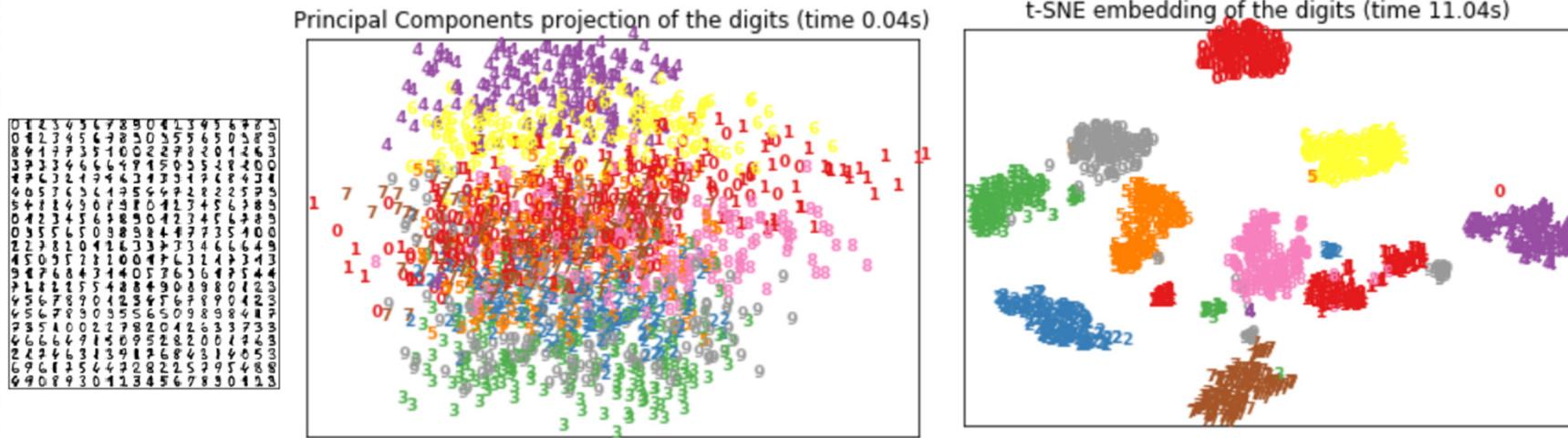


## The t-SNE algorithm – Step 3

- In the last step we want these set of probabilities from the low-dimensional space to reflect those of the high dimensional as best as possible (as we want the two map structures to be similar).
- We then measure the difference between the probability distributions of the two-dimensional spaces using Kullback-Liebler divergence (KL), often written as  $D(p,q)$ .
- KL-divergence is an asymmetrical approach that compares two distributions (“distance” between two distributions).
- We then minimize our KL cost function using gradient descent.



## Visual Comparison of t-SNE vs PCA



# **Introduction to Recommendation Engines**



640



# Netflix's Recommendations

The screenshot shows the Netflix homepage with a yellow dotted border around the main content area. At the top, there is a navigation bar with links for Home, TV Shows, Movies, Recently Added, and My List. To the right of the navigation are search, KIDS, notifications (with 9+ notifications), and profile icons.

**Because you watched Love, Death & Robots ➤**

A row of six show thumbnails: STRANGER THINGS, ALTERED CARBON, ASH VS EVIL DEAD, STAR TREK: THE NEXT GENERATION, KISS ME FIRST, and THE UMBRELLA ACADEMY.

**Top Picks for Rajeev**

A row of six show thumbnails: I AM A KILLER, FORENSIC FILES, TIMELESS (with a "NEW EPISODES" button), INSIDE THE WORLD'S TOUGHEST PRISONS, WOMEN BEHIND BARS, and LOUIS THEROUX.

**Because you watched Better Call Saul**

A row of six show thumbnails: WEEDS, RAKE, GODLESS, THE PEOPLE V. O.J. SIMPSON: AMERICAN CRIME STORY (with a "NEW EPISODES" button), POWER, and DAMNATION.

641



# Amazon's Recommendations



## Frequently bought together



Total price: £477.42

Add all three to Basket

i These items are dispatched from and sold by different sellers. Show details

**This item:** Canon EF 70-300 mm f/4-5.6 IS II USM Lens - Black £449.00

JJC Reversible Lens Hood Shade for Canon EF 70-300mm f/4-5.6 IS II USM Replaces Canon Lens Hood ET... £14.99

Hoya 67mm UV(C) Digital HMC Screw-in Filter,Y5UVC067 £13.43



YouTube GB

Search

Home

Trending

Subscriptions

Library

### Recommended

JVNA Live Ep:008 | Alchemy | Melodic Dubstep, Future... 38:49

Epic Music World 🔊 1:07:08

DUTY FREE 1:39:37

JVNA 🔊 53K views • 4 weeks ago

Relaxing Music Mix | BEAUTIFUL PIANO 3:20

REDY MIX DMTV 15K views • 1 month ago

Soca 2019 / Soca 2020- Duty Free 2019 (The Best...) 1:14:01

MOTHERBOARD 11:06

Paramore: Playing God [OFFICIAL VIDEO] 3:20

Buddha-Bar III - CD1 1:14:01

Nuclear Fusion Energy: The Race to Create a Star on... 3:46

Fueled By Ramen 🔊 68M views • 8 years ago

Buddha-Bar Official Channel 186K views • 5 months ago

YEAH yeah yeah 3:46

ピアノ曲 1:05:09

MACHINE LEARNING 41:17

Living in the Age of AI 41:17

7

643

# How do they know us so well?



644



## Are Big Tech Companies Spying on Us?

- No
- Well technically no.
- Through advances in Recommendation System Machine Learning methods, companies can create very accurate user specific recommendations!



## Why are Recommendation Systems so Important?

- **Information Overload** – Think about online retailers like Amazon, eBay or Netflix. What problem can you easily foresee? There is just too much products or movies to choose from. Yes we can search for something we want, but what about things we'd potentially like, but didn't know existed.
- In this new era of Big Data, we need systems with heuristic techniques that **make our process of selection easier!**

“

*“Recommender Systems aim to help  
a user or a group of users  
to select items from a crowded item  
or information space.”*

*(MacNee et. al 2006)*



# Section Overview

In this section we'll learn

- Intuition behind Recommendation Systems – How do we review items?
- Collaborative Filtering and Content-based filtering

**Before recommending, how do we  
rate or review Items?**



## Let's try to Build an Item Comparison Tool

- Suppose we have website called “**I like to Watch Movies**”, a Netflix competitor
- We allow users can rate movies by giving a thumbs **up** or **down** to rank it.
- And now we want to get a list of our highest rated movies
- **How do we do this?**



I Like to Watch Movies





# Approach 1 – Net Score

- We take the Net Score of Positive ratings– Negative ratings
  - $Net\ Score = (Positive\ Ratings) - (Negative\ Ratings)$

| Movie | Positive Ratings | Negative Ratings | Net Score | Ave Percent Positive |
|-------|------------------|------------------|-----------|----------------------|
| A     | 750              | 500              | 250       | 60%                  |
| B     | 5000             | 4000             | 1000      | 56%                  |

- Our algorithm scores Movie B higher, but is this right? **Nope**



## Examples of sites making this mistake

### Urban Dictionary

#### 2. **normal**

209 up, 50 down  

A word made up by this corrupt society so they could single out and attack those who are different

*Normal is nothing but a word made up by society*

[conformists](#) [worker bees](#) [in crowd](#) [followers](#) [mindless](#)  
by [Bill](#) Oct 6, 2005 [share this](#) [add comment](#)

#### 3. **normal**

118 up, 25 down  



## Approach 2 – Average Rating

- $\text{Average Rating} = \frac{\text{Positive Ratings}}{\text{Total Ratings}}$

| Movie | Positive Ratings | Negative Ratings | Net Score | Ave Percent Positive |
|-------|------------------|------------------|-----------|----------------------|
| A     | 750              | 500              | 250       | 60%                  |
| B     | 5000             | 4000             | 1000      | 56%                  |
| C     | 9                | 1                | 8         | 90%                  |

- Our algorithm now scores Movie C the highest but again is this right?  
**Nope.** Movies with very little reviews will dominate the rankings.

## Sites making this mistake - Amazon

4



EVGA GeForce GTX 1660 Ti XC Black Gaming, 6GB GDDR6, HDB Fan Graphics Card 06G-P4-1261-KR

★★★★★ ▾ 2

£265<sup>94</sup> £279.99

✓prime | Same-Day

FREE delivery Today, Oct 30

More buying choices

£247.32 (16 used & new offers)



5



MSI NVIDIA GTX 970 Gaming Twin Frozr HDMI DVI-I DP Graphics Card (4GB, PCI Express, DDR5, 256 Bit)

★★★★★ ▾ 908

£429<sup>00</sup>

£5.95 delivery

Only 1 left in stock.

More buying choices

£116.99 (5 used & new offers)

6



EVGA Geforce GTX 1050 TI GeForce GTX1050TI Graphic Card 4096 MB

★★★★★ ▾ 15

More buying choices

£226.42 (2 new offers)

654



## Approach 3 **CORRECT** Score = Lower bound of Wilson score confidence interval for a Bernoulli parameter

- It can be seen that we need to balance the proportion of positive ratings with the uncertainty of a small number of observations.
- Fortunately, the math for this was worked out in 1927 by Edwin B. Wilson.
- What we want to ask is: Given the ratings I have, there is a 95% chance that the “real” fraction of positive ratings is at least what?
- Wilson gives the answer. Considering only positive and negative ratings (i.e. not a 5-star scale), the lower bound on the proportion of positive ratings is given by:

$$\left( \hat{p} + \frac{z_{\alpha/2}^2}{2n} \pm z_{\alpha/2} \sqrt{\left[ \hat{p}(1 - \hat{p}) + z_{\alpha/2}^2 / 4n \right] / n} \right) / (1 + z_{\alpha/2}^2 / n).$$

Source: <https://www.evanmiller.org/how-not-to-sort-by-average-rating.html>

# User Collaborative Filtering and Item/Content-based Filtering



# Recommendation Systems

- As we discussed in the introduction to this section, when we have too much choices, we tend to avoid choosing.
- We need a tool that can 'know' what we'd like. Imagine someone who knows you inside out, could be your best friend, your spouse or parent. But they can easily go into a store and pick things you'd like.
- They, in essence act like a Recommendation System
- But how do they do this? And how would an AI system figure this out?



# Types of Recommendation Systems

Let's think about two ways we can do this for let's say, an online retailer like Amazon.

1. Do we have users that buy similar items? Let's say our system has records that we have a subset of users who bought Metallica albums, and most of these customers also bought Megadeath albums too. Therefore, we can infer if someone has purchases Metallica albums, they are a likely candidate to purchase a Megadeath album. This is called **Collaborative Filtering**.
2. Another approach is, what if we know a user has been searching for formal wear suits online. We know intrinsically that formal wear suits need to be paired with appropriate shoes. As such, we can recommend the user purchase our top rated shoes. This is called **Content or Item based filtering**.



## Collaborative filtering User to User

- The Collaborative filtering algorithm is used to recommend products based on the history of user behaviors and consequently looks at the **similarities between users**
- A Collaborative filter uses something called the **user-to-item matrix** to find similar users.



## User-to-item Matrix

|   | Image   | Book    | Video   | Game    |
|---|---------|---------|---------|---------|
| A | Like    | Dislike | Like    | Like    |
| B | Like    | Like    | Dislike | Dislike |
| C | Like    | Like    | Dislike |         |
| D | Dislike |         | Like    |         |
| E | Like    | Like    | ?       | Dislike |

| User | Item A | Item B | Item C | Item D |
|------|--------|--------|--------|--------|
| 1    | 1      | 0      | 1      | 1      |
| 2    | 0      | 1      | 0      | 0      |
| 3    | 1      | 1      | 0      | 0      |
| 4    | 0      | 0      | 1      | 0      |
| 5    | 1      | 1      | 0      | 0      |



## User-to-item Matrix Explained

- In our matrix, each row represents an individual customer
- Each column represents items in the inventory
- Therefore, a 1 or 0 indicates that the user in that row has purchased (clicked, watched, rated highly, liked, rated, etc.) the item
- From this matrix we constructed, we can calculate the similarity of users using some method to measure distance between users. A popular and successful method is the **Cosine Similarity**



## Using the Cosine Similarity

- $\text{Cosine Similarity}(a, b) = \cos(a, b) = \frac{a \cdot b}{\|a\| \times \|b\|}$

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

- A and B are used to represent two users, A and B
- $A_i$  and  $B_i$  represent each item User A and B purchased.



# Cosine Similarity Explained

Imagine we have two sentences to compare

- Amy likes mangoes more than apples
- Sam likes potatoes, Sam likes mangoes

Create a list of words: “Amy, likes, mangoes, more, than, apples, Sam, potatoes”



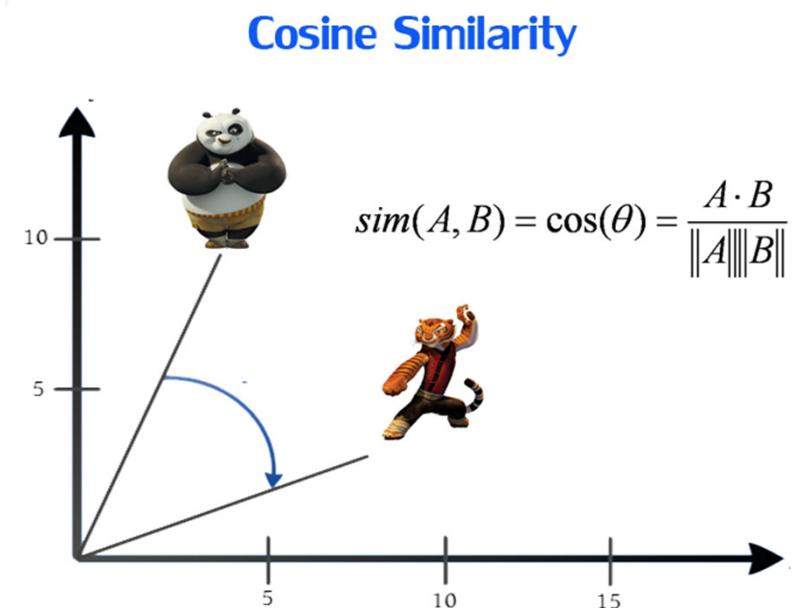
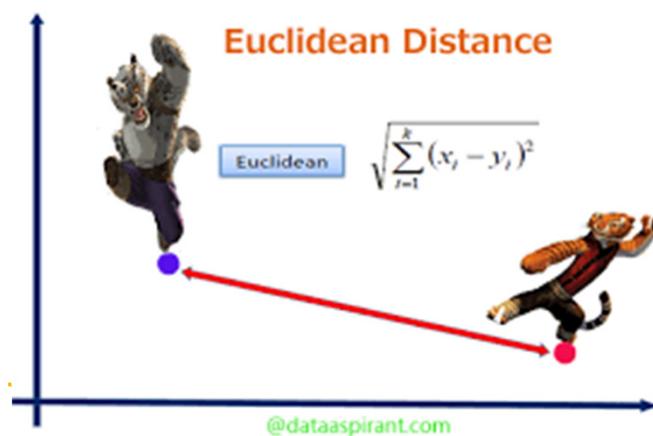
## Cosine Similarity Explained continued

| Words    | Sentence 1 | Sentence 2 |
|----------|------------|------------|
| Amy      | 1          | 0          |
| Likes    | 1          | 2          |
| Mangoes  | 1          | 1          |
| More     | 1          | 0          |
| Than     | 1          | 0          |
| Apples   | 1          | 0          |
| Sam      | 0          | 1          |
| potatoes | 0          | 1          |



## Cosine Similarity Explained continues

- Sentence 1 = [1,1,1,1,1,1,0,0]
- Sentence 2 = [0,2,1,0,0,0,1,1]
- Cosine Distance = 0.463



665



## Item-based approach collaborative filtering

- In Collaborative Filtering, we calculated similarities between the two users by building a user-to-item matrix.
- However, for a **item-based approach**, we need to calculate similarities between the two items.
- This means that we need to build and use an item-to-user matrix, (we do this by simply transposing the user-to-item matrix).



## User to User Collaborative Filtering Disadvantages

- Systems performed poorly when they had many items but comparatively few ratings
- Computing similarities between all pairs of users was expensive
- User profiles changed quickly and the entire system model had to be recomputed



## How item-item based filtering solved this problem

- Item-item models resolve these problems in systems that have more users than items.
- Item-item models use rating distributions per item, not per user. With more users than items, each item tends to have more ratings than each user, so an item's average rating usually doesn't change quickly.
- This leads to more stable rating distributions in the model, so the model doesn't have to be rebuilt as often. When users consume and then rate an item, that item's similar items are picked from the existing system model and added to the user's recommendations.

# **Case Study – User Collaborative Filtering and Item/Content-based filtering in Python**

# **The Netflix Prize and Matrix Factorization and Deep Learning as Latent-Factor Methods**



## Collaborative Filtering Recap

- Looks relatively simple to create and underlying theory is relatively simple
- Can be applied to many types of examples
- High accuracy

However, it's not all good, let's take a look at some of the potential issues we face when using Collaborative filtering.



## Collaborative Filtering Challenges

- **Sparsity** – the number of purchases made by each user is extremely small compared to the number of items that exist.
- **Cold-start issues** – we don't have anything to recommend to new users as they haven't made any purchases/reviews/likes yet
- How to recommend a **new item?**
- **Scalability** – the more users grow the larger the computational requirements to calculate distances between them.



## Latent-Factor Methods were introduced to help with Scalability Issues

- Latent-factor methods create a new and most times a reduced feature space of the original user or item vectors, leading to reduced noise and faster computations in real-time.

There are two latent-factor methods:

1. *Matrix factorization*
2. *Deep learning*



## Matrix Factorization

- **Matrix Factorization** was popularly used in the Netflix Prize Competition.
- Recommendations are a vital part of business in our modern era, so much so that Netflix offered \$1M USD in a competition to improve recommendation performance over their in-house algorithm.
- On September 21, 2009, the grand prize was given to the BellKor's Pragmatic Chaos team which bested Netflix's own algorithm for predicting ratings by 10.06%



# Netflix Prize

Home Rules Leaderboard Register Update Submit Download

## Leaderboard

Display top  leaders.

| Rank   | Team Name                              | Best Score | % Improvement | Last Submit Time    |
|--|--|------------|---------------|---------------------|
| --   | No Grand Prize candidates yet          | --         | --            | --                  |
| <b>Grand Prize - RMSE &lt;= 0.8563</b>   |  |            |               |                     |
| 1  | <a href="#">PragmaticTheory</a>        | 0.8584     | 9.78          | 2009-06-16 01:04:47 |
| 2  | <a href="#">BellKor in BigChaos</a>    | 0.8590     | 9.71          | 2009-05-13 08:14:09 |
| 3  | <a href="#">Grand Prize Team</a>       | 0.8593     | 9.68          | 2009-06-12 08:20:24 |
| 4  | <a href="#">Dace</a>                   | 0.8604     | 9.56          | 2009-04-22 05:57:03 |
| 5  | <a href="#">BigChaos</a>               | 0.8613     | 9.47          | 2009-06-15 18:03:55 |
| <b>Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos</b> |  |            |               |                     |
| 6  | <a href="#">BellKor</a>                | 0.8620     | 9.40          | 2009-06-17 13:41:48 |
| 7  | <a href="#">Gravity</a>                | 0.8634     | 9.25          | 2009-04-22 18:31:32 |
| 8  | <a href="#">Opera Solutions</a>        | 0.8640     | 9.19          | 2009-06-09 22:24:53 |
| 9  | <a href="#">xvector</a>                | 0.8640     | 9.19          | 2009-06-17 12:47:27 |
| 10   | <a href="#">BruceDengDaoCiYiYou</a>    | 0.8641     | 9.18          | 2009-06-02 17:08:31 |
| 11   | <a href="#">Ces</a>                    | 0.8642     | 9.17          | 2009-06-12 23:04:25 |
| 12   | majia2                                 | 0.8642     | 9.17          | 2009-06-15 03:35:00 |
| 13   | xiangliang                             | 0.8642     | 9.17          | 2009-06-13 16:35:35 |
| 14   | <a href="#">Feeds2</a>                 | 0.8647     | 9.11          | 2009-06-16 22:21:19 |
| 15   | <a href="#">Just a guy in a garage</a> | 0.8650     | 9.08          | 2009-05-24 10:02:54 |
| 16   | Team ESP                               | 0.8653     | 9.05          | 2009-06-16 05:25:11 |
| 17   | <a href="#">pengpengzhou</a>           | 0.8654     | 9.04          | 2009-05-05 18:18:03 |
| 18   | NewNetflixTeam                         | 0.8657     | 9.01          | 2009-05-31 07:30:22 |
| 19   | <a href="#">J.Dennis Su</a>            | 0.8658     | 9.00          | 2009-03-11 09:41:54 |
| 20   | Vandelay Industries !                  | 0.8658     | 9.00          | 2009-05-11 00:43:14 |



675



## Singular Value Decomposition (SVD)

- Singular Value Decomposition (SVD) decomposes the preference matrix as:

- $P_{m \times n} = U_{m \times m} \Sigma_{m \times n} V_{n \times n}$

- U and V are unitary matrixes. For 4 users and 5 items, we will have:

$$P_{4 \times 5} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ u_{21} & u_{22} & u_{23} & u_{24} \\ u_{31} & u_{32} & u_{33} & u_{34} \\ u_{41} & u_{42} & u_{43} & u_{44} \end{bmatrix} \bullet \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 \end{bmatrix} \bullet \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} \\ v_{41} & v_{42} & v_{43} & v_{44} & v_{45} \\ v_{51} & v_{52} & v_{53} & v_{54} & v_{55} \end{bmatrix}$$

- Where  $\sigma_1 > \sigma_2 > \sigma_3 > \sigma_4$ , therefore the preference for the first item is written as:
- $p_{11} = \sigma_1 \mu_{11} v_{11} + \sigma_2 \mu_{12} v_{21} + \sigma_3 \mu_{13} v_{31} + \sigma_4 \mu_{14} v_{41}$
- Vector Form -  $p_{11} = \sum_{\sigma} \vec{\mu}_1 \cdot \vec{v}_1$
- So now we can select the top two features based on the sigmas:
- $p_{11} \approx \sigma_1 \mu_{11} v_{11} + \sigma_2 \mu_{12} v_{21}$



## Simon Funk's SVD

There were still two issues with using SVDs

1. The way missing values are imputed can have an undesirable impact on the outcome
2. The computational complexity for training can be high when all the entries are considered

During the Netflix Prize content, Simon Funk developed a solution. Where only non-missing entries ( $p_{ij}$ ) are considered.

$$\min_{\vec{u}_i, \vec{v}_j} \sum_{p_{ij}} (p_{ij} - \vec{u}_i \cdot \vec{v}_j)^2$$

Therefore, the estimated score for the item j from user i is:

$$\vec{u}_i \cdot \vec{v}_j$$



## Deep Learning Embedding

- Deep learning offered a more flexible method to include various factors into modeling and creating embeddings.
- The workings for many of these methods can be a bit complicated to explain, however in essence, they formulate the problem as a classification problem where each item is one class.
- There have many advances that can deal with millions of users and items.
- Multi-Phase modeling such as used by YouTube divided the modeling process into two steps where the first uses only user-item activity data to select hundreds of candidates out of millions. Then in the second phase it uses more information about the candidate videos to make another selection and ranking.
- All Recommender systems should not overfit historical user-item preference data (exploitation), to avoid getting stuck in a local optimal.

# **Introduction to Natural Language Processing**



# Introduction to Natural Language Processing

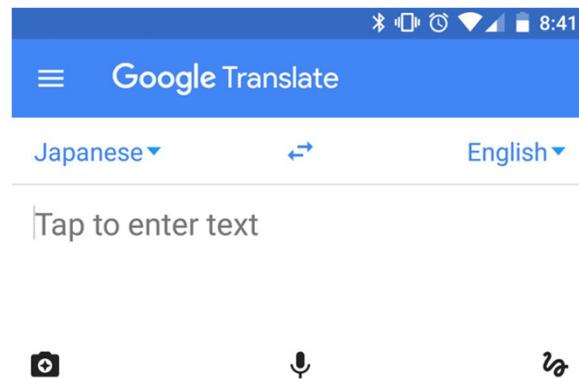
- Natural Language Processing (NLP) is a sub-field of Artificial Intelligence that deals with the processing and understanding of human language.
- It is the foundation of language translation, topic modeling, document indexing, information retrieval and extraction.
- Current hot topics in NLP involve search engines, chat bots, sentiment analysis, summarization of documents, marking essays and grammatical checking and improving writing.





## NLP's role in Businesses - Translating

- Imagine translating your website or business communication into different languages:





## NLP's role in Businesses - Summarizing information

### Reviews (265)

[Write a review](#)



#### Show reviews that mention

Search reviews

[All reviews](#) [chicken](#) [papadoms](#) [curry](#) [naan breads](#) [prawn puri](#) [dumplings](#) [king prawns](#)

[nepalese food](#) [delicious food](#) [tasted amazing](#) [fantastic restaurant](#) [small restaurant](#) [great menu](#)

[recommend this restaurant](#) [saturday night](#) [visited this restaurant](#) [attentive service](#)

1 - 10 of 255 reviews

#### Read reviews that mention

[small amount](#) [easy to apply](#) [dry hair](#) [short hair](#) [matt and messy](#)

[works well](#) [hair gel](#) [much better](#) [hair in place](#) [easy to use](#)

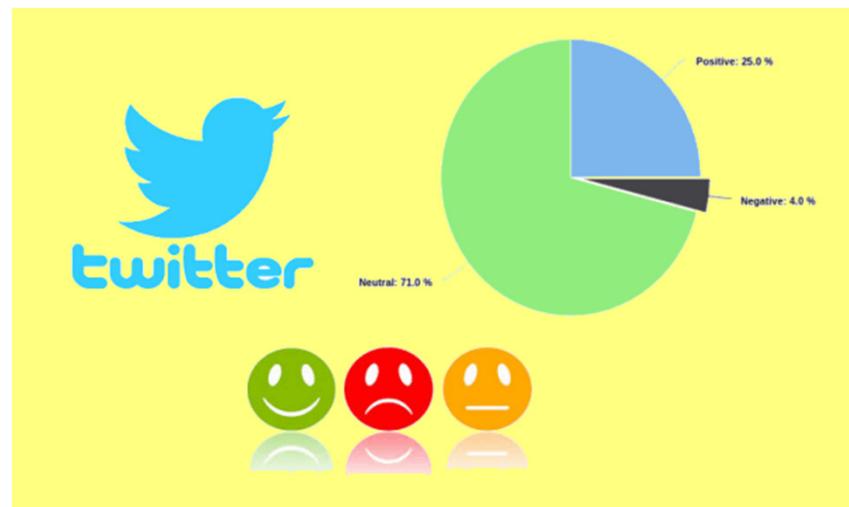
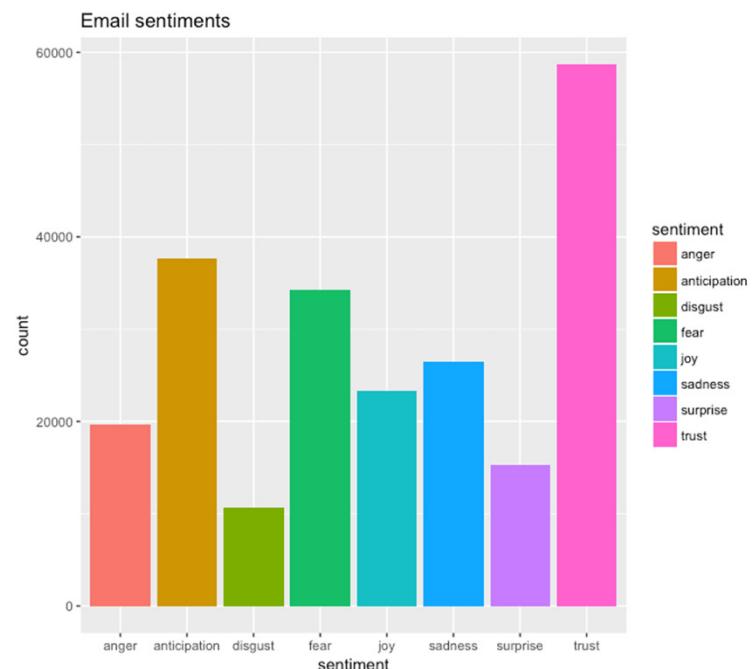
[fairly short](#) [rough paste](#) [strong hold](#) [makes it easy](#) [easy to wash](#)

198 customer reviews

682



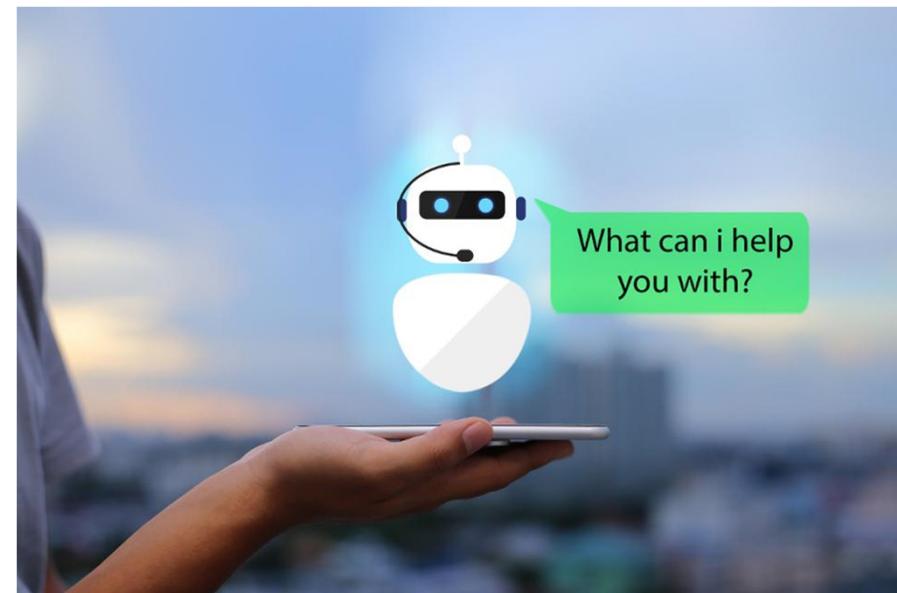
## NLP's role in Businesses - Sentiment analysis



683



## NLP's role in Businesses - Chatbots





## NLP's role in Businesses – Information Extraction and Search





# NLP's role in Businesses – Auto Responders and Auto Complete

A screenshot of a messaging application interface. At the top, there is a dark header bar with the text "Taco Tuesday". Below this, a message from a user named Jacqueline Bruzek is shown, with the text "Taco Tuesday". A reply message from the user is also visible, starting with "Hey Jacqueline," followed by the text "Haven't seen you in a while and I hope you're doing well." To the left of the messages, there are several vertical text snippets, such as "on re", "pcom", "natio", "rly A", "t vita", and "g dat", which appear to be part of a larger document or conversation.



# Main Topics in NLP

- Bag of Words Model
- Tokenization
- Predictions
- Sentiment Analysis - Case Study – Sentiment of Airline Tweets
- Summarization - Case Study – Amazon Review Summaries
- Text Classification - Case Study - Spam Filter

# **Modeling Language – The Bag of Words Model**



## Bag of Words Modeling

- When training NLP models, we need to understand how we create or format our data.
- In English we've got hundreds of thousands of words, which begs the question, how do we represent our words as data inputs to a ML model?



# Typical Machine Learning Inputs

| Input #     | Output - Gender |
|-------------|-----------------|
| Height      |                 |
| Weight      | 0 – Male        |
| Hair Length | 1 - Female      |

| Input #  | Output – Diabetes Risk |
|----------|------------------------|
| Height   |                        |
| Body Fat | Percentage Output      |
| Weight   |                        |

- Typical Datasets we've seen so far have simple or intuitive inputs.
- How do we represent language or text inputs in this?
- We use Tokenization!



# Tokenization

- **Tokenization** is the process where we take an input string of text and split it up (parse) into individual string elements.
- It is a vectorization technique that allows us to represent words as real numbers



## Bag of Words and Tokenization Example

1. *We have landed on the moon*
2. *The USA has landed on the moon*

[We, have, landed, on, the, moon]

[The, USA, has, landed, on, the, moon]

| # | We | the | have | USA | has | landed | on | moon |
|---|----|-----|------|-----|-----|--------|----|------|
| 1 | 1  | 1   | 1    | 0   | 0   | 1      | 1  | 1    |
| 2 | 0  | 1   | 0    | 1   | 1   | 1      | 1  | 1    |



## The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



|           |     |
|-----------|-----|
| it        | 6   |
| I         | 5   |
| the       | 4   |
| to        | 3   |
| and       | 3   |
| seen      | 2   |
| yet       | 1   |
| would     | 1   |
| whimsical | 1   |
| times     | 1   |
| sweet     | 1   |
| satirical | 1   |
| adventure | 1   |
| genre     | 1   |
| fairy     | 1   |
| humor     | 1   |
| have      | 1   |
| great     | 1   |
| ...       | ... |



## Understanding our input data

1. *We have landed on the moon*
2. *The USA has landed on the moon*

| # | We | the | have | USA | has | landed | on | moon |
|---|----|-----|------|-----|-----|--------|----|------|
| 1 | 1  | 1   | 1    | 0   | 0   | 1      | 1  | 1    |
| 2 | 0  | 1   | 0    | 1   | 1   | 1      | 1  | 1    |

| Original Document              | Document Vector   | Class Label |
|--------------------------------|-------------------|-------------|
| We have landed on the moon     | [1,1,1,0,0,1,1,1] | Negative    |
| The USA has landed on the moon | [0,1,0,1,1,1,1,1] | Positive    |

**Normalization, Stop Word Removal,  
Lemmatizing/Stemming**



## Words are Messy

*"We ain't gonna get it in time"*

*"This meal was pleasantly exquisite!"*

*"Vote 4 PSG.LGD at The International 2019"*

*"OMG lol jus txt me pls, ttyl"*

*"U wot mate?"*



- Words in any language (we use English here) can get extremely messy! Any NLP project requires extensive cleaning, but how do we go about doing this?



# Normalization Types

- Case Normalization
- Removing Stop Words
- Removing Punctuation and Special Symbols
- Lemmatising/Stemming



# Case Normalization

Case normalization is simply the standardizing of all case for words found in our document. Example, changing sentences like:

- “**Hi John, today we’ll go to the market**”:
- “**hi john, today we’ll go to the market**”

Typically, case doesn't actually change any meaning, however they are cases (pun intended) where it can, example:

- Reading is a city in the UK which is different to the act of reading.
- April and May are both common names and months.



# Removing Stop Words

- Stop words are common words that do not act additional information or predictive value due to how common they are in normal text. Examples of common stop words are:
  - I
  - is
  - and
  - a
  - are
  - the



## Removing Punctuation and Special Symbols

- Self explanatory, but very important step in our cleaning process. However, there are instances where punctuation adds mean such as it's vs. its. Example of punctuation and special symbol removals are:
- !"#\$%&'()\*+,-./;:<=>?@[\\]^\_`{|}~



# Lemmatising/Stemming

- Both lemmatizing and stemming are techniques that seek to reduce inflection forms to normalize words with the same lemma.
- Lemmatising does this by considering the context of the word while stemming does not.
- However, most current lemmatizers or stemmer libraries are not highly accurate.

## Example

|          |   |        |
|----------|---|--------|
| caresses | → | caress |
| ponies   | → | poni   |
| caress   | → | caress |
| cats     | → | cat    |

Stemming from Stanford NLP

# **TF-IDF Vectorizer (Term Frequency — Inverse Document Frequency)**



# Vectorizing Text

- Sometimes our good old Bag of Words model isn't always good enough, it's often a good way to explain how simple NLP classifiers are built, but in reality we need sometimes more complex methods.
- Methods that can account the importance of each term in a document/text.
- One of the simplest and most versatile vectorizers is the **TF-IDF Vectorizer** (Term Frequency — Inverse Document Frequency)



## TF-IDF Vectorizer

The TF-IDF statistic for a term  $i$  in document  $j$  is calculated as shown below:

- $TF - IDF(i, j) = TF \times IDF$

Where

$$TF(i, j) = \frac{\text{number of times term } i \text{ is in document } j}{\text{total number of terms in document}}$$

$$IDF(i) = \ln \left( \frac{\text{total number of documents}}{\text{number of documents containing term } i} \right)$$

# **Word2Vec - Efficient Estimation of Word Representations in Vector Space**



## Vectorizing Text

- In our previous slides we explained Bag of Words modeling and the TF-IDF vectorizer. Both are useful, but more advances in vectorizing documents have been developed since.
- One such is the Word2Vec model developed by a Google Research team.



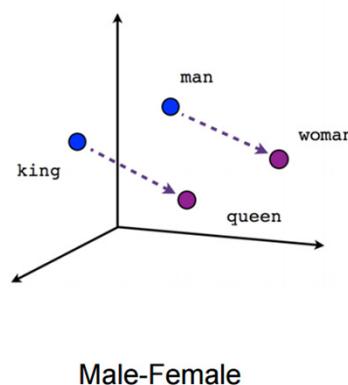
## Vectorizing Text

- Word2vec has been pre-trained over large corpora
- It uses an unsupervised learning method to determine semantic and syntactic meaning from word co-occurrence, which is then used to construct vector representations for every word in the vocabulary.

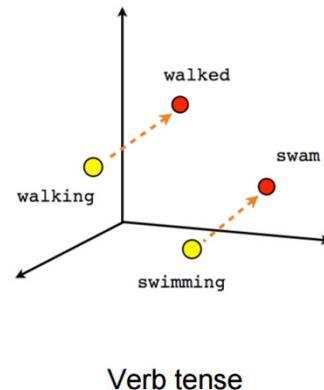


From the Word2Vec Paper - **Read** – “*Efficient Estimation of Word Representations in Vector Space*” <https://arxiv.org/pdf/1301.3781.pdf>

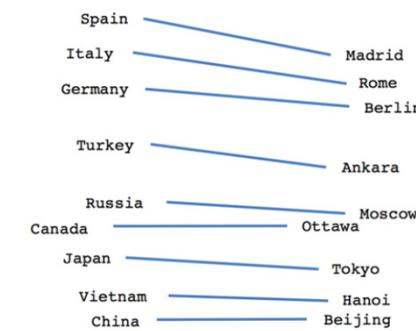
“Using a word offset technique where simple algebraic operations are performed on the word vectors, it was shown for example that  $\text{vector}(\text{"King"}) - \text{vector}(\text{"Man"}) + \text{vector}(\text{"Woman"})$  results in a vector that is closest to the vector representation of the word Queen.”



Male-Female



Verb tense



Country-Capital



# Training Word2Vec

- Two model architectures were used to train word2vec:
  - Continuous Bag of Words
  - Skip Gram
- They both use a **context window** to determine the contextually similar words, example, using a window with a fixed size  $n$  means that all the words within  $n$  units from the target word belong to its context.



## Context Window

- Let's use a fixed window of size 2 on the following sentence:
  - The **quick brown fox** jumped over the lazy dog
- Let's look at **Fox** as our target word example.
- The words in blue (**quick, brown, jumped, over**) belong to the context of **Fox**
- Word2Vec has the ability that with enough examples of contextual similarity, the network learns the correct associations between words.



# Context Window

- This assumption made by the design of the Word2Vec model is that, “*words which are used and occur in the same contexts tend to purport similar meaning*”
- The context window of Word2Vec is dynamic which has a max size.
- The context is samples from the max window size with a probability of  $1/d$  where  $d$  is the distance between the word to target.
- In our previous example, the target word **fox** using a dynamic context window with maximum window size of 2. (brown, jumped) have a **1/1** probability of being included in the context since they are one word away from fox. (quick, over) have a **1/2** probability of being included in the context since they are two words away from fox.



## Continuous Bag of Words & Skip Gram

- Using this method, the Continuous Bag of Words and the Skip Gram models separate the data into observations of target words and their context.
- Continuous Bag of Words** – In our Fox example, the context is ‘quick, brown, jumped, over’. These form the features for the Fox class.
- Skip Gram** – Here we structure the data so that the target is used to predict the context, so here we’ll use Fox to predict the context ‘quick, brown, jumped, over’.



# Building the Neural Network Model

- Word2Vec trains a shallow neural network over data structured using either the Continuous Bag of Words Model or Skip Gram.
- Example of a simple Continuous Bag of Words model with a fixed context window of 1 on a simple corpus (right)
- Let's use our context window to include words that follows the target. In this example we can assume the words at the end of the sentence is the first word of the next sentence.
- Here we can see that simple patterns emerge, where "Math" and "Programing" are both context to "like".

I like math.

I like programming.

Today is Friday.

Today is a good day.

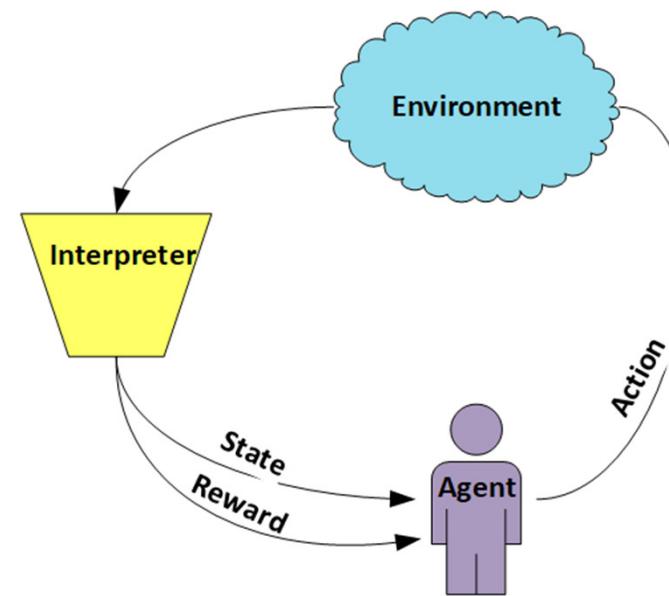
- like* is the context of target *I*
- math* is the context of target *like*
- programming* is also the context of target *like*

# **Reinforcement Learning**



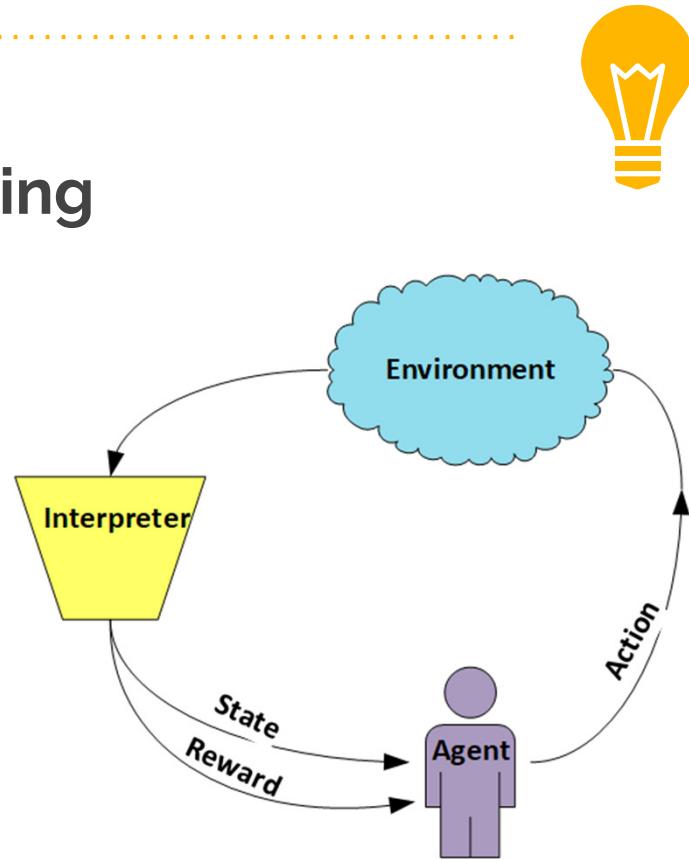
## Introduction to Reinforcement Learning

- In Reinforcement learning, we teach our algorithm, or '**agent**' in an **environment** which produces a **state** and **reward**.
- The agent performs **actions/interacts** with this environment which result in various responses.



## Learn in Reinforcement Learning

- The agent in this environment examines the **state** and the **reward** information it receives.
- It choose an action which **maximizes the reward** feedback it receives.
- The **agent learns** by repeated interaction with the environment.

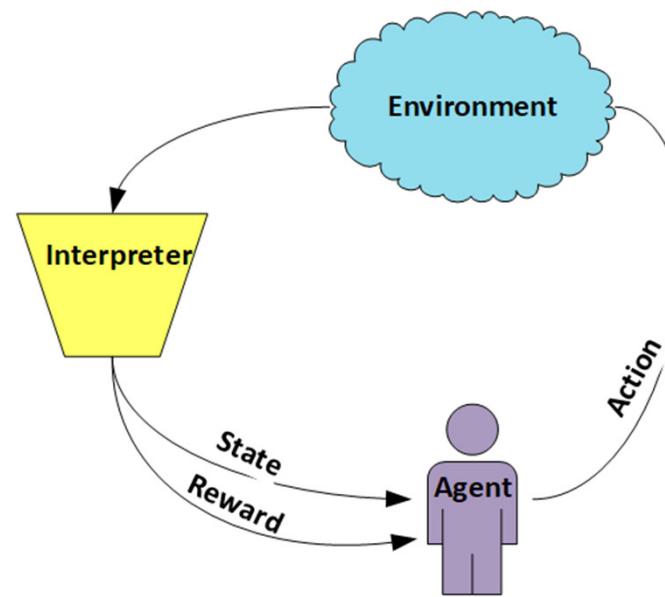




## Learn in Reinforcement Learning

A successful agent needs to:

- Learn the interaction between states, actions and their corresponding rewards
- Determine which action(s) are the best to take





# Q Learning

- Q-learning is a model-free reinforcement learning algorithm.
- The goal of Q-learning is to learn a **policy**, which tells an agent what action to take under what circumstances.
- It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.



# Q Learning State Reward Tables

|         | Action 1 | Action 2 |
|---------|----------|----------|
| State 1 | 0        | 5        |
| State 2 | 5        | 0        |
| State 3 | 0        | 5        |
| State 4 | 5        | 0        |

- This is a simple State-Action-Reward table where in this 'game' we can see at each state the best reward the agent needs to take.
- If an agent randomly explored this game, and summed up which actions received the most reward in each of the four states (and stored this in an array), then it would basically learn the functional form of the table above.



## State Reward Tables – Deferred Learning

|         | Action 1 | Action 2 |
|---------|----------|----------|
| State 1 | 10       | 0        |
| State 2 | 5        | 0        |
| State 3 | 5        | 0        |
| State 4 | 5        | 40       |

- After extensive trials an agent will be able to learn that taking Action 2 repeatedly for States 1,2,3 and 4 lead to the greatest reward..



# The Q Learning Rule

- We can define the Q-Learning update rule as:
- – Reward
- $\gamma$  – Discounts reward impact (0 to 1)
- $\max_a Q(s, a)$  - This is the maximum Q value possible in the next state. It represents the maximum future reward thus to encourage the agent to aim for the max reward in as little action steps



**Shane Legg**  
@ShaneLegg



### Learn:

1. linear algebra well (e.g. matrix math)
2. calculus to an ok level (not advanced stuff)
3. prob. theory and stats to a good level
4. theoretical computer science basics
5. to code well in Python and ok in C++

Then read and implement ML papers and \*play\* with stuff! :-)

ML is a broad field, so you can specialize in what interests you. If you're interested in NLP, check out my notes on that topic.

# Introduction to Big Data



# Big Data

- **Big Data** has been such a big buzz word over the last few years, but in reality very few people understand or use the term correctly.
- Big Data is often used as a catchall phrase for any data analytics, Artificial Intelligence or ML work where the dataset is often known to be 'quite big'.
- But what is 'Big'?



ALSO  
**BIG**  
EVERY  
EXAMPLES  
TOOLS  
DISK TARGET  
APPLIED  
SHARED SENSOR  
RECONSIDER DEFINITION CURRENT  
ZETIABYTES PRACTITIONERS  
INTERNET CAPTURE BUSINESS  
DESCRIBING RADIO-FREQUENCY  
MANAGEMENT DISTRIBUTED SETS GENOMICS  
TERABYTES CASE COMPLEXITY  
ELAPSED PETABYTES SYSTEMS  
INCLUDE TOLERABLE  
WORKING  
GARTNER  
CURRENTLY  
OPPORTUNITIES  
AMOUNT  
DIFFICULTY  
**DATA**  
HUNDREDS  
BIOLOGICAL PROCESSING  
CAPACITY  
NOW  
BURIED WORLD'S  
NETWORKS  
UBIQUITOUS  
RECORDS  
DATABASES  
SEARCH  
GROW  
PARALLEL  
SAN  
QUALITIES  
MPP  
STORAGE  
ABILITY  
SIZE  
CASE  
COMPLEXITY  
GENOMICS  
BUSINESS  
SETS  
CAPTURE  
WITHIN  
THOUGHT  
MOVING  
MAY  
CURRENT  
RECOGNITION  
PRACTITIONERS  
INTERNET  
DESCRIBING  
MANAGEMENT  
TERABYTES  
ELAPSED  
PETABYTES  
INCLUDE  
TOLERABLE  
WORKING  
GARTNER  
CURRENTLY  
OPPORTUNITIES  
AMOUNT  
DIFFICULTY  
**DATA**  
HUNDREDS  
BIOLOGICAL PROCESSING  
CAPACITY  
NOW  
BURIED WORLD'S  
NETWORKS  
UBIQUITOUS  
RECORDS  
DATABASES  
SEARCH  
GROW  
PARALLEL  
SAN  
QUALITIES  
MPP  
STORAGE  
ABILITY  
SIZE  
CASE  
COMPLEXITY  
GENOMICS  
BUSINESS  
SETS  
CAPTURE  
WITHIN  
THOUGHT  
MOVING  
MAY  
CURRENT  
RECOGNITION  
PRACTITIONERS  
INTERNET  
DESCRIBING  
MANAGEMENT  
TERABYTES  
ELAPSED  
PETABYTES  
INCLUDE  
TOLERABLE  
WORKING  
GARTNER  
CURRENTLY  
OPPORTUNITIES  
AMOUNT  
DIFFICULTY



## Big Data Defined

- Big Data refers to a **huge volume of data**, that cannot be **stored** and **processed** using the **traditional computing** approach within a given time frame.
- The size of the data in question can be in gigabytes, terabytes, petabytes or more! But this isn't always the case and it depends on the organization.
- You can transform a 500mb of data into much larger sizes if you were to create 10,000 features from it?

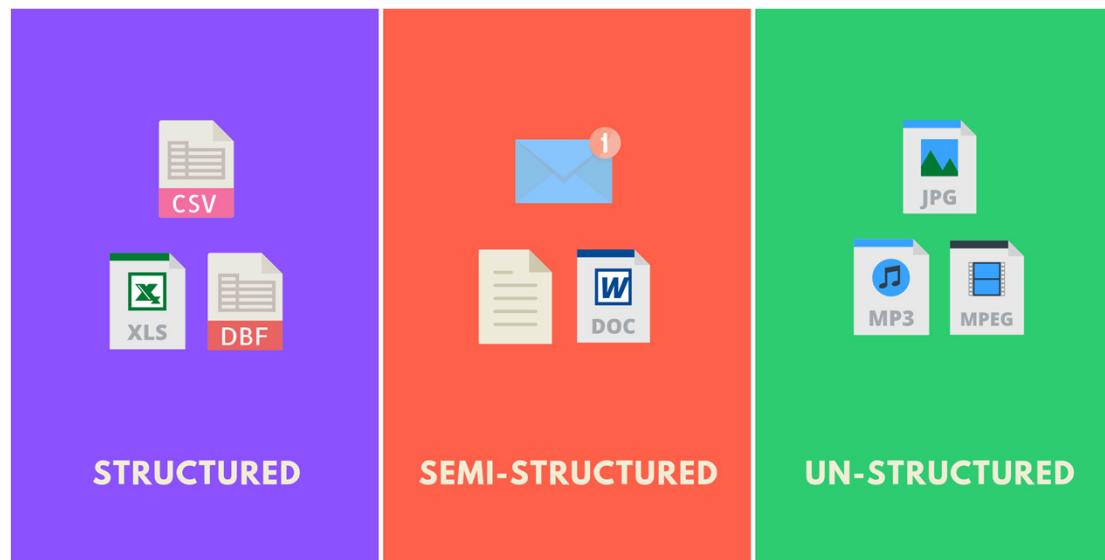


## Examples of Big Data

- User information for millions of customers
- User generated information, think of a social media behemoths like Facebook and Twitter
- Behavioral data (e.g tracking user movements on a website)
- Image data
- Text Data



# Classifications of Big Data





# Structured Data

- This is data that has a proper defined structure to it, examples include:
  - Existing Database data
  - CSV or any other structured or organized data



## Semi-Structured Data

- This is data that doesn't have a defined structure, example the text in documents can be messy and very disorganized

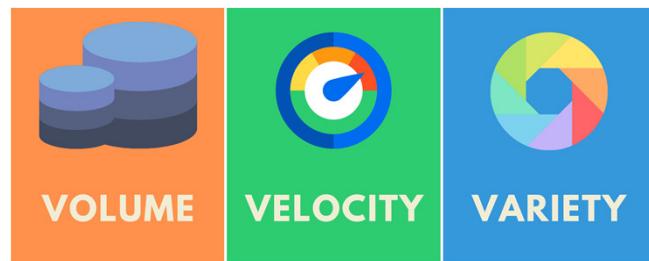


## Un-Structured Data

- This is data that has no structure at all, examples include:
  - Video
  - Audio
  - Images



## Big Data Characteristics - The 3Vs



- **Volume** – The amount of data generated, think of how much thousands of point of sale terminals can generate sales data
- **Velocity** – The speed at which this data is generated. Think about how fast thousands of IoT sensors can create data
- **Variety** – the various types of data being generated, eg. Think of an organizations cloud storage that holds everyone's documents, emails, audio, video etc.

# Challenges in Big Data



## Typical Data Storage Challenges

- Data in organizations is typically generated from several places, then via the ETL process (Extract, Transform and Load), is then stored in Databases
- Initially when an organization started, this existing infrastructure might have been sufficient. However, in modern day companies, this situation becomes untenable due to one big reason.
- **Data Growth!**



## Challenges of Dealing with Big Data

- Storage Requirements – hardware or cloud space can get very expensive!
- Deciding what to store and for how long
- Scalability
- Processing time constraints when performing Analysis and Data Science work



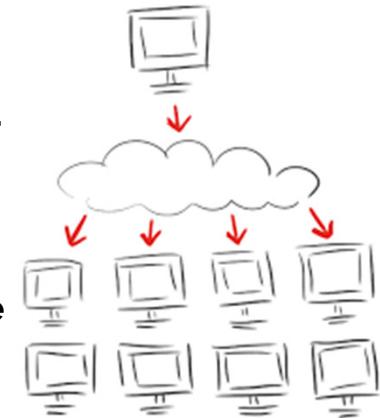
## Big Data Solutions

- While we haven't solved the storage issues just yet (though prices are cheaper than ever!), software tools have been developed that can help solve one of the biggest challenges in Big Data.
- Using **Distributed** workload across several (can be thousands) of machines can help us perform analytics and data science activities on massive datasets that wouldn't have been possible on a powerful desktop.



# Distributed Data/Computing

- Big Data operates in a world that is a bit different from our normal computer operations. When dealing in data that is so big, or data that is generated so fast it becomes impossible for even the most powerful single computer to handle.
- Hence the reason we need distributed computing comes in. Storage and computation tasks can be deployed over multiple computers.
- MapReduce was designed to used for Distributed Computing/Data and solved many of the initial Big Data issues.



# Hadoop, MapReduce and Spark



## Big Data History

- Even though CPU, RAM, storage performance and size increased exponentially over the last two decades. The processing requirements for Big Data grew even faster!
- Thus one of the first models used for processing Big Data was MapReduce.

# MapReduce



- MapReduce is a programming framework for Big Data processing on **distributed platforms** created by Google in 2004. It can run calculations over thousands of computers in parallel
- It evolved further with advances by Hadoop and Spark, but one of the key paradigms set out by MapReduce was the Map and Reduce phase, still used by Hadoop and Spark today.

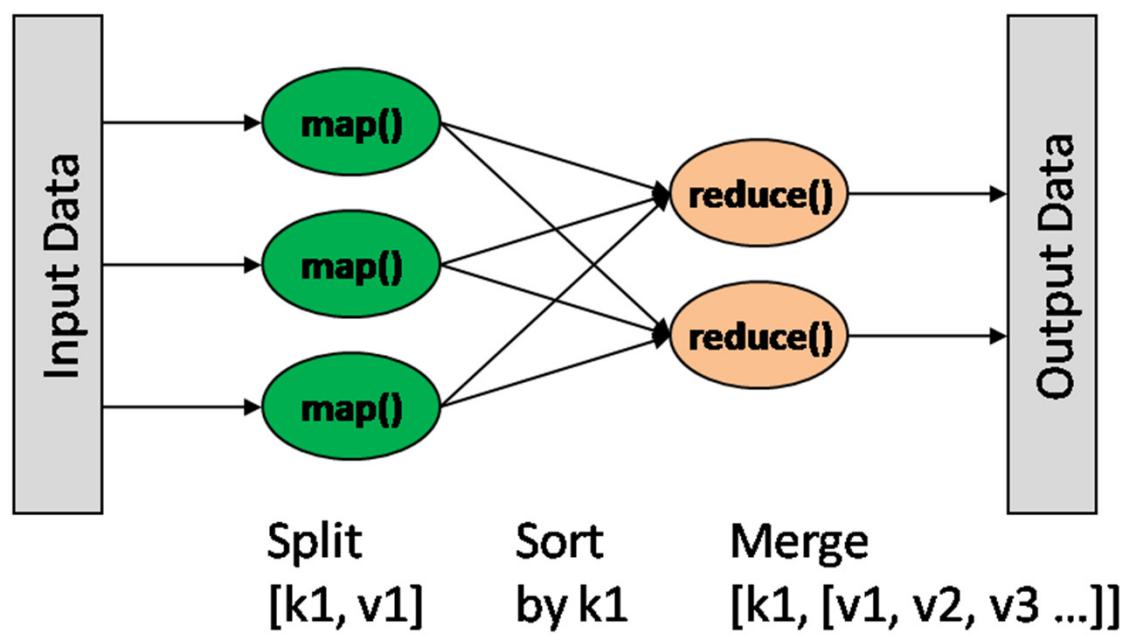
# MapReduce



- **Map phase:** The user specifies a map function that is applied to each key-value pair, producing other key-value pairs, referred to as intermediate key-value pairs.
- **Reduce phase:** In this phase the intermediate key-value pairs are grouped by key and for each group the user applies a reduce function, producing other key-value pairs, which is the output of the round.
- A program written in MapReduce is automatically parallelized without the programmer having to care about the underlying details of partitioning the input data, scheduling the computations or handling failures.



# MapReduce



# Hadoop

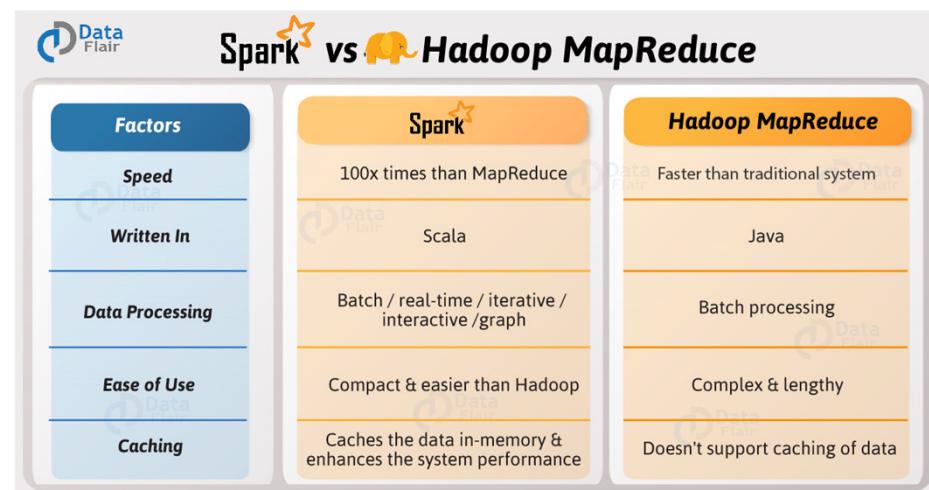


- Hadoop is a JAVA based open source system developed by Apache in 2006 and provides a software framework for distributed storage and processing of big data using the MapReduce programming model.
- Hadoop is a **Processing Engine/Framework** and introduced the HDFS (Hadoop Distributed File System), a batch processing engine (MapReduce) & a Resource Management Layer (YARN).
- Hadoop provided the ability to analyze large data sets. However, it relied heavily on **disk storage as opposed to memory** for computation. Hadoop was therefore very slow for calculations that require multiple passes over the same data.
- This allowed the hardware requirements for Hadoop operations to be cheap (hard disk space is far cheaper than ram) it made accessing and processing data much slower.
- However, Hadoop had poor support for SQL and machine learning implementations.

# Spark

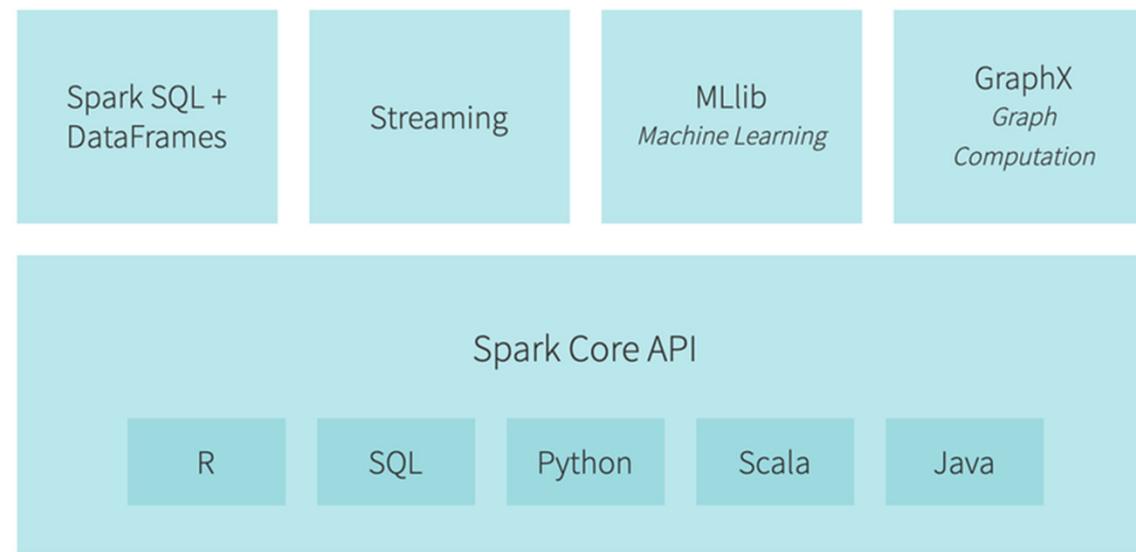


- The UC Berkeley AMP Lab and Databricks spearheaded the development of Spark that aimed to solve many of deficiencies, performance issues and complexities of Hadoop
- It was initially released in 2014 and donated to the Apache Software Foundation
- Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.
- As such, Spark was a big hit with Data Scientists!





# Spark Components



Source: <https://databricks.com/spark/about>



# RDDs – Resilient Distributed Data Set

- **RDD** is Spark's representation of a dataset which is distributed across the RAM of a cluster of machines. It is the primary data abstraction in Apache Spark and is often referred to as the Spark Core.

The features of RDDs are:

- **Resilient**, i.e. fault-tolerant with the help of RDD lineage graph and so able to re-compute missing or damaged partitions due to node failures.
- **Distributed** with data residing on multiple nodes in a cluster.
- **Dataset** is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).

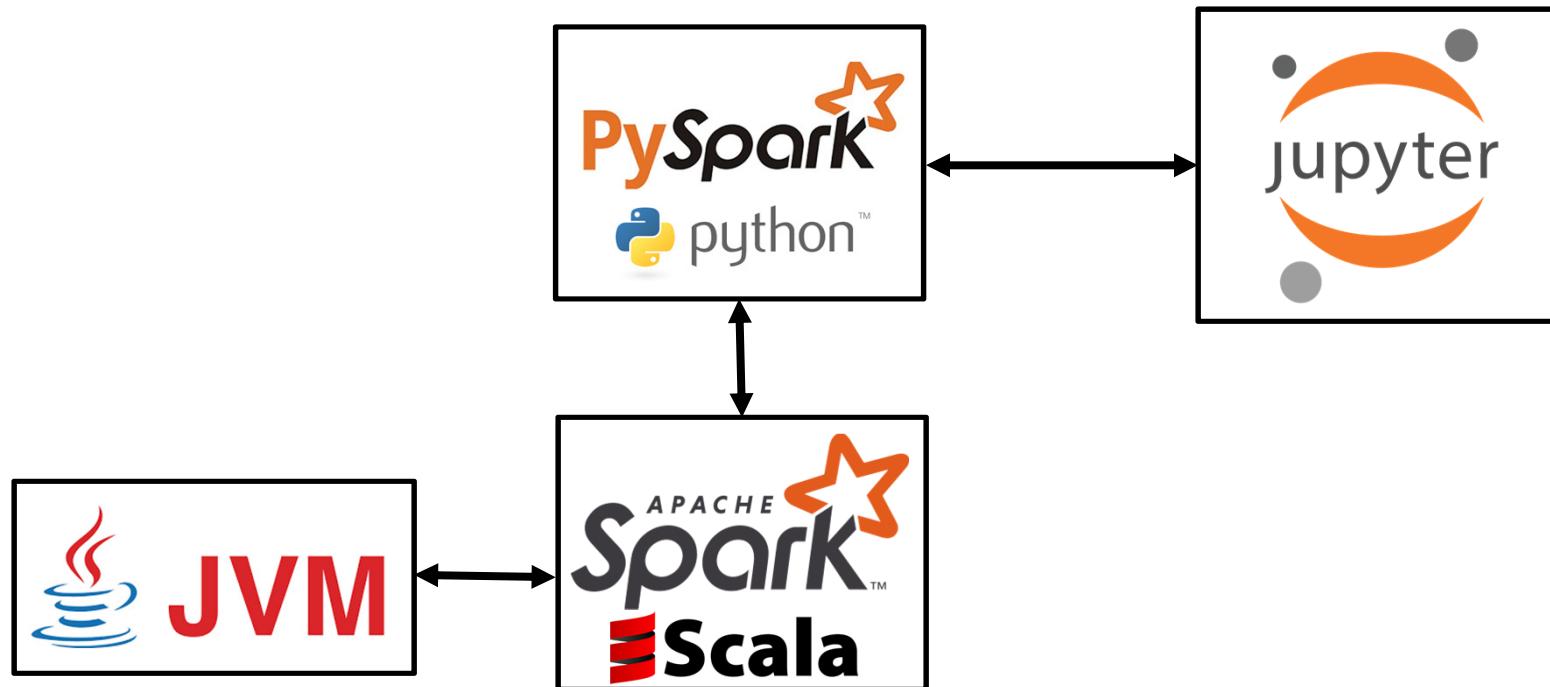
# Introduction to PySpark

# PySpark



- Even though the Spark toolkit was written in **Scala**, a language that compiles to byte code for the **Java Virtual Machine (JVM)**, as such numerous implementations or wrappers have been developed for R, Java, SQL and of course Python!
- It was made possible by **Py4j** which enables Python programs running in a Python interpreter to dynamically access Java objects in a JVM. As such, PySpark is API that allows us to interface with RDDs in Python
- As such, Python users can now work with **RDDs** in Python programming language also

# PySpark Overview



# PySpark in Industry

- **Netflix** has used PySpark internally to power many of their backend machine learning tasks (apparently almost one **trillion** per day!)
- The **healthcare** industry has used PySpark to perform analytics including Genome Sequencing
- The **financial** sector has made full use of PySpark for in-house trading, banking, credit risk and insurance use cases.
- **Retail and E-commerce** – Literally begs the use of PySpark given that these businesses have millions and millions of sales and retail data in their data warehouses. Both **eBay** and **Alibaba** are known to use PySpark.



# **RDDs, Transformations, Actions, Lineage Graphs & Jobs**



## RDDs (Resilient Distributed Data) in Python

- When we load data into python using PySpark (you'll be doing this soon), it creates an **RDD object**
- RDD is a data structure that is the core building block of Spark. It is an **immutable (means once it's made it can't be altered)**, **partitioned** collection of records or elements that we can use to hold lists of tuples, dictionaries, lists.
- RDDs do not have a schema.** This means they do not have a columnar structure. Records are just recorded row-by-row, and are displayed similar to a list. RDDs are analogous, but very different to Pandas DataFrame.
- Our loaded dataset is an RDD object and we can then run any of the methods accessible to that object.



# Lazy Evaluation and Pipelining

- Spark's RDD implementation allows us to evaluate code "lazily". Lazily means it **postpones running a calculation until it is needed**.
- A regular compiler (like Python) sequentially evaluates each expression it comes across. A lazy compiler doesn't continually evaluate expressions, instead it waits until it is actually told to generate a result, and then performs all the evaluation all at once.
- Pipelining is how we chain together calculations in Spark and understanding it is critical in working with Spark.



# Types of Spark Methods

- **Transformations:**
  - `map()`
  - `reduceByKey()`
- **Actions:**
  - `take()`
  - `reduce()`
  - `saveAsTextFile()`
  - `collect()`



# Transformations

- Transformations are one of the methods you can perform on an RDD in Spark.
- They **are lazy operations** that create one or more new RDDs (because RDDs are **immutable**, they can't be altered in any way once they've been created)
- Transformations take an RDD as an input and apply some function on them and outputs one or more RDDs.
- Let's talk about lazy evaluation - as the Scala compiler comes across each Transformation, it doesn't actually build any new RDDs yet. Instead, it **constructs a chain (or pipeline)** of hypothetical RDDs that would result from those Transformations which will only be evaluated once an **Action** is called.
- This chain of hypothetical, or "child", RDDs are all connected logically back to the original "parent" RDD, this concept is called the **lineage graph**.



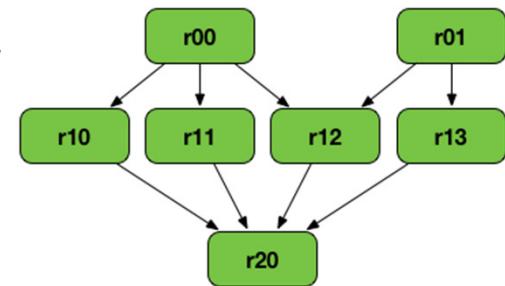
# Actions

- Actions are any operations on RDDs that do not produce an RDD output
- Typically these include operations such as getting a count, max, min etc.



# Lineage Graphs

- Remember the **chain or pipeline** constructed due to our Lazy Evaluation? These were **Transformation** operations that are only evaluated when an **Action** is called. This chain of hypothetical, or “child”, RDDs are all connected logically back to the original “parent” RDD, this concept is called the **lineage graph**.
- A **lineage graph** outlines a “logical execution plan”. The compiler begins with the earliest RDDs that aren’t dependent on any other RDDs, and follows a logical chain of Transformations until it ends with the RDD that an Action is called on
- Lineage Graphs are the drivers of Spark’s fault tolerance. If a Node fails, the information about what that node was supposed to do is held in the lineage graph and thus can be done elsewhere.

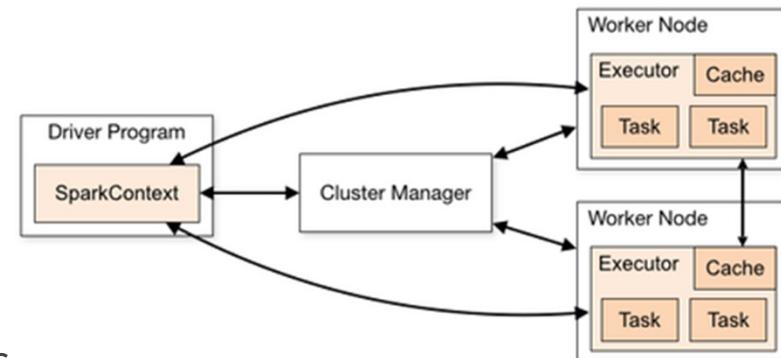


Visualization of example lineage graph;  
r00, r01 are parent RDDs, r20 is final  
RDD (source Jacek Laskowski -  
<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-lineage.html#logical-execution-plan>)



# Spark Applications and Jobs

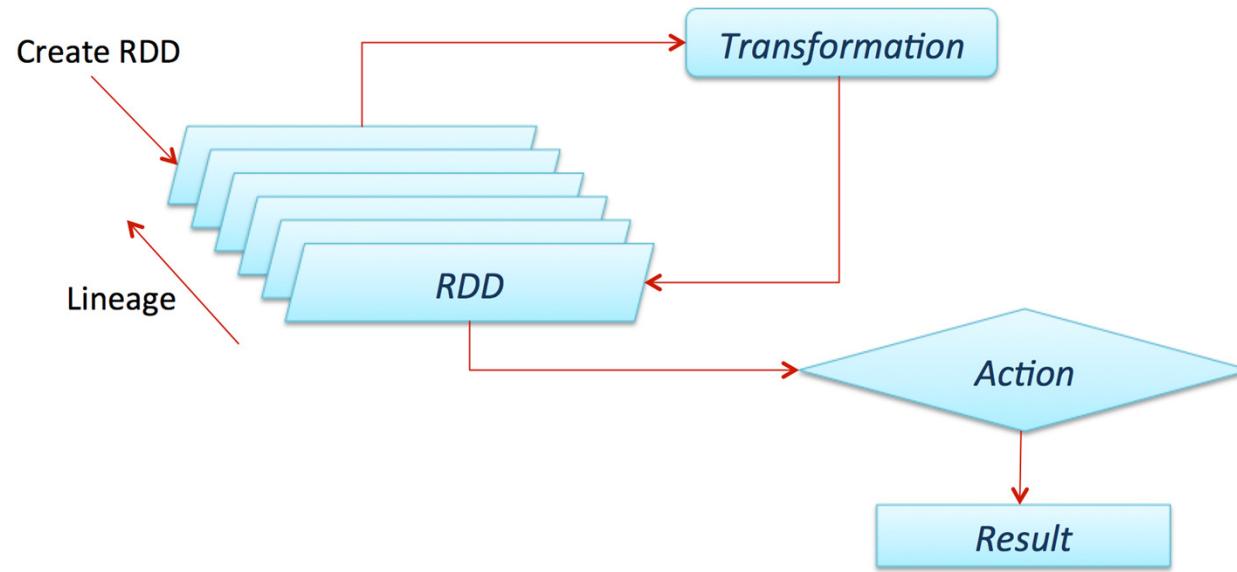
- In Spark, whenever processing needs to be done, there is a **Driver** process that is in charge of taking the user's code and converting it into a set of multiple tasks.
- There are also **executor** processes, each operating on a separate node in the cluster, that are in charge of running the tasks, as delegated by the driver.
- Each **driver** process has a set of **executors** that it has access to in order to run tasks.
- A Spark **application** is a user built program that consists of a driver and that driver's associated executors.
- A Spark **job** is task or set of tasks to be executed with executor processes, as directed by the driver.



Visualization of Spark Architecture (from Spark API  
- <https://spark.apache.org/docs/latest/cluster-overview.html>)



# Spark Overview



# **Simple Data Cleaning in PySpark**

# Machine Learning in PySpark

# **Customer Lifetime Value (CLV)**



# Customer Lifetime Value

- Why is CLV Important?
- Suppose our Marketing Department has told us the **Customer Acquisition Cost** (Marketing Spending divided by the number of new customers) is \$100 per customer.
- How do you know if that's worth it?
- That's where CLV comes in. we need to determine what the expected lifetime value a customer has to our business.



## Customer Lifetime Value Defined

***Definition - The present value of the expected sum of discounted cash flows of an individual customer.***

This effectively means, it's the total purchases made (cash flow) over the lifetime of that customer.



## Customer Lifetime Value Application

- For CLV to be useful we need to **accurately predict how much the customer will spend in future**
- CLV in our situation applies to non-contractual customers who have a continuous opportunity to purchase
- **Contractual** customers are subscription based customers
- **Discrete** (opposite of continuous in our case) has limited windows of purchase opportunity e.g. concert tickets or seasonal businesses



## Benefits of knowing your CLV

- Determine the traits of your most valuable customers and find similar customers
- Know how much you should be spending to acquire a particular type of customer
- Push the marketing channels that bring you your most valuable customers
- Use your best customers for market research and product feedback



## What CLV is Not!

- There is a lot of misconception about CLV, with many marketing texts and tutorials defining it's calculation as:

*"CLV is calculated by multiplying the Average Order Value by number of Expected Purchases and Time of Engagement."*

### Why is this wrong?

- CLV isn't a calculation per se, it's a prediction and this definition is leads one to think it's simply the spend of a customer over time (based only on past purchases).
- We're interested in the Expected Value of a customer's returns



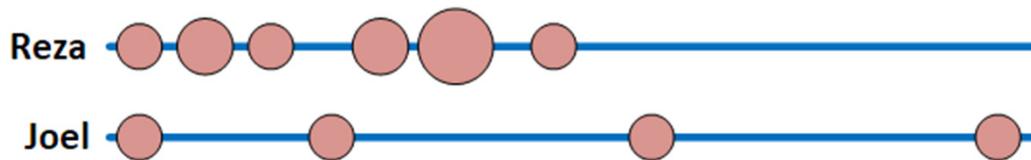
## Another CLV Pitfall

- Another common mistake used when determining CLV is that it is calculated over all customers.
- This neglects the fact that we have different types of customers.



## Another CLV Pitfall Example

- Imagine we have two customers, Reza and Joel.
- Reza discovered our business and liked it a lot, as such made many frequent purchases early on, but then lost interest and stopped buying
- Joel, buys less frequently but is still a regular customer.



- If we based on our CLV on purchase frequency and average order value, we'd be misled into thinking Reza is a more valuable customer.
- This type of analysis failed to consider that Reza churned.

## **Buy-til-you-die (BTYD) models**



# The buy-til-you-die model

- In 1987 by researchers at Wharton and Colombia developed a model called the Pareto/NBD (Negative Binomial Distribution) that could be used for estimating the number of future purchases a customer will make.
- This was the foundation of buy-til-you-die (BTYD) models.



# BTYD Method

- BTYD models predict purchasing activity of a customer using two stochastic probabilistic models.
  1. The probability of customer making a repeat purchase
  2. The probability of a customer churns or 'dies'
- To get the above we need two pieces of information:
  1. Recency – the time since a customers last purchase
  2. Frequency - the number of repeat purchases placed by that customer in the given time period



## Using BTYD to Estimate Expected No. of Future Purchases

- Our goal with the BTYD model is to estimate the expected number of future purchases each customer will place over a time period given their recency and frequency.
- This can be expressed as:
- $E[X(t)] = \text{expected number of transactions in time of length } t$   
*(given a customer's recency and frequency)*



## Obtaining a Customers Residual Lifetime Value

- Once we have the Expected number of future purchases for each customer, we can multiply it by that customers average order value to get their **Residual Lifetime Value (RLV)**.

$RLV = \text{expected future purchases} + \text{expected average order value}$

- Residual lifetime value is the amount of additional value we can expect to collect from a customer over a given time period.
- To get our CLV just add the sum of each customer's past purchases to their RLV!



## The Beta-Geometric/Negative Binomial Distribution

- In 2003, Peter Fader and Bruce Hardie published their seminal paper on a simplified version of the Pareto/NB.
- It was called the Beta-Geometric/NBD (BG/NBD) and it was much easier to implement (thanks to great packages created around it)



## The lifetimes Module in Python

- Implementing the BG/NBD model in Python is relatively simply using the **lifetimes** module created by Cameron Davidson-Pilon, the former head of Data Science at Shopify.
- All the lifetimes value model needs is a simple transaction log, with a customer ID, date of order, and order amount.
- Let's go into Python and experiment with it!