**Analysing the Relative Performance of Recommender Algorithms Across Song Domain**

Algorithms Used: KNN and ANN

**Abstract:** A music recommendation system was developed that can learn users' preferences. The system can classify a wide range of stored music using automatic music content analyses. Users can opt for music according to their mood, using such words as "bright", "exciting", "quiet", "sad" and "healing".

Develop an approach, to find the similarities between pair of songs using audio features and lyrics.

The approach used in this paper focuses on various features of songs. Here, the user interface allows user to play songs and at the same time recommends songs based on the current song being played. The recommendations are stored using firebase firestore and firebase storage.

The audio features and lyrics score of each pair of songs are provided to the algorithm. Artificial Neural Networks and KNN regression are the algorithms used. The algorithm predicts the similarity score between pair of songs. Songs which have highest similarity scores with respect to the currently playing song is recommended to the user.

**Problem Statement:** The recommendation system is because of information overload, and we can call it an information filter system. It greatly influences what we interact with the world: shopping (Amazon, Best Buy), music(Spotify), video(Youtube, Netflix), etc. To build a recommendation system providing recommendations to millions of users with millions of items.

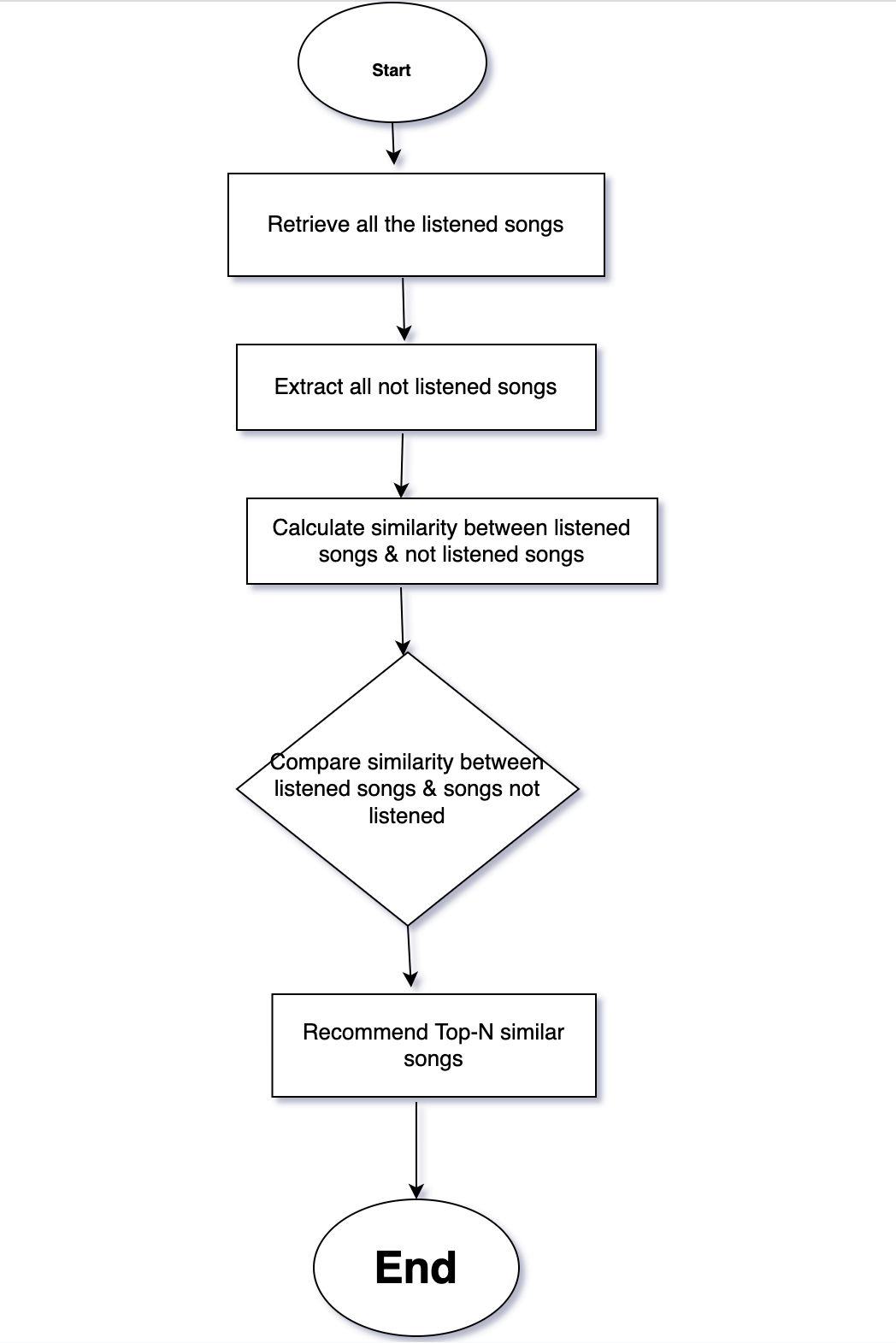
Customers/ Viewers can spend hours scrolling through hundreds, sometimes thousands of items/songs/movies never finding a thing they like.

**Scope:**

The Scope of the Project is to make Recommendation System using ANN & KNN Algorithms which will be able to recommend the relevant recommendations for the customer or viewer.

**Methodology:**

Pre-processing the dataset in proper manner using different techniques. Further using KNN & ANN Algorithms and Techniques trying to find the best recommendation for three different domains.

****

**Contribution:**

There are several experiments and researches already being done for Recommendation System. Although, the effort to test and compare several different techniques will be done in our project.

**Existing System:** Different recommendation systems with different approaches for giving recommendations.

**Proposed System:** We propose 6 different recommendation systems using datasets from 3 different domains using KNN & ANN algorithms with the combination of techniques like Data Normalization, Proper Handling of Nan Values etc. is used to give recommendation. This Approach will make the system more accurate.

**Success Criteria:**

The System must be able to give recommendation according to user/viewer interest.

**Requirements:**

* PC
* Python
* Necessary modules like (Numpy, Pandas, Scikit-learn, Keras etc.)

**Solution:**

Combination of several techniques with Cosine similarity & sentence transformers and other techniques is used to recommend for both KNN & ANN models. This Approach will make the system more accurate.

**Future Work:**

In this proposed model ANN is showing more accuracy, so for future model we can improve the accuracy of KNN by trying to find the similarities between the different movies & Users and other different approaches to increase the accuracy.

We can also try to get the better dataset with more relevant & useful features to improve the performance.

**Introduction:**

Recommender systems aim to predict users' interests and recommend product items that quite likely are interesting for them. They are among the most powerful machine learning systems that online retailers implement in order to drive sales.

Data required for recommender systems stems from explicit user ratings after watching a movie or listening to a song, from implicit search engine queries and purchase histories, or from other knowledge about the users/items themselves.

Companies using recommender systems focus on increasing sales as a result of very personalized offers and an enhanced customer experience.

Recommendations typically speed up searches and make it easier for users to access content they’re interested in, and surprise them with offers they would have never searched for.

What is more, companies are able to gain and retain customers by sending out emails with links to new offers that meet the recipients' interests, or suggestions of films and TV shows that suit their profiles.

The user starts to feel known and understood and is more likely to buy additional products or consume more content. By knowing what a user wants, the company gains competitive advantage and the threat of losing a customer to a competitor decreases.

Providing that added value to users by including recommendations in systems and products is appealing. Furthermore, it allows companies to position ahead of their competitors and eventually increase their earnings.

Recommender systems function with two kinds of information:

* *Characteristic information*. This is information about items (keywords, categories, etc.) and users (preferences, profiles, etc.).
* *User-item interactions.* This is information such as ratings, number of purchases, likes, etc.

Based on this, we can distinguish between three algorithms used in recommender systems:

* *Content-based* systems, which use characteristic information.
* *Collaborative filtering* systems, which are based on user-item interactions.
* *Hybrid systems*, which combine both types of information with the aim of avoiding problems that are generated when working with just one kind.

**METHODS:**

Following Methods and Techniques were used in the Project.

* KNN
* ANN
* Cosine similarity
* Keras

Details of the above Methods are mentioned below in the document

**Divided Work:**

Dataset Finding

Dataset Preprocessing

KNN Model Training  
ANN Model Training

**Timeline and Work Stages:**

4 Weeks

Finding Right Dataset – 1 week

Cleaning and Dataset Preprocessing – 1 week

Model Trainings and Validation – 1 Week

Testing and Finalization – 1 week

**Expected Algorithms to be used:**

KNN

ANN

1. Why KNN?

To implement an item based collaborative filtering, KNN is a perfect go-to model and also a very good baseline for recommender system development. But what is the KNN? KNN is a non-parametric, lazy learning method. It uses a database in which the data points are separated into several clusters to make inference for new samples.

1. KNN does not make any assumptions on the underlying data distribution but it relies on item feature similarity. When KNN makes inference about a movie, KNN will calculate the “distance” between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest neighbor item as the most similar item recommendations.
2. WhyANN?  
     
   Embeddings are a way to represent discrete — categorical — variables as continuous vectors. In order to create embeddings, we need a neural network embedding model and a supervised machine learning, hence artificial neural network is best suited.
3. What Cosine similarity done for Us.?

Cosine similarity is a metric used to measure how similar two items are. Mathematically it calculates the cosine of the angle between two vectors projected in a multidimensional space. Cosine similarity is advantageous when two similar documents are far apart by Euclidean distance(size of documents) chances are they may be oriented closed together. The smaller the angle, higher the cosine similarity.

5. Where did we found Dataset?

The dataset is taken from Kaggle.com, which is a famous website for hosting genuine datasets and organizing Machine Learning challenges. You can get more details about the dataset and download it using the link below.

# <https://www.kaggle.com/code/vatsalmavani/music-recommendation-system-using-spotify-dataset/data>

6. How do we know that this is right data set,?

[Spotify](https://www.spotify.com/us/) is a pro in is in having knowledge of its customers. The platform incorporates proprietary algorithms for comprehending the music taste of the user and to steer them towards fresh genres, songs, and artists.

**K-NN(K-Nearest Neighbor)**

K-NN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution i.e. the model structure determined from the dataset.

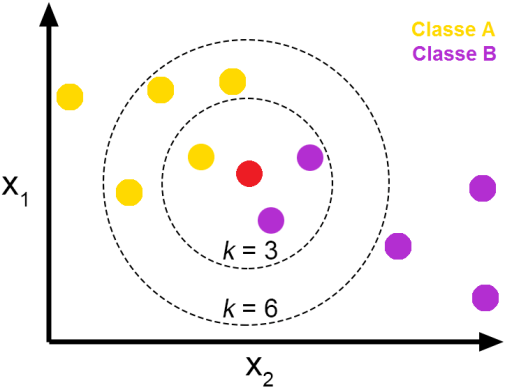
It is called Lazy algorithm because it does not need any training data points for model generation. All training data is used in the testing phase which makes training faster and testing phase slower and costlier.

K-Nearest Neighbor (K-NN) is a simple algorithm that stores all the available cases and classifies the new data or case based on a similarity measure.

K-NN classification

In K-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

To determine which of the K instances in the training dataset are most similar to a new input, a distance measure is used. For real-valued input variables, the most popular distance measure is the Euclidean distance



Source: Towards Data Science

**The Euclidean distance**

* The Euclidean distance is the most common distance metric used in **low dimensional data sets**. It is also known as the**L2 norm**. The Euclidean distance is the usual manner in which distance is measured in the real world.

where p and q are n-dimensional vectors and denoted by **p** = (*p*1, *p*2,…, *pn*) and **q** = (*q*1, *q*2,…, *qn*) represent the n attribute values of two records.

* While Euclidean distance is useful in low dimensions, it **doesn’t work well in high dimensions and for categorical variables**. The drawback of Euclidean distance is that it **ignores the similarity between attributes**. Each attribute is treated as totally different from all of the attributes.

**Other popular distance measures :**

* **Hamming Distance**: Calculate the distance between binary vectors.
* **Manhattan Distance**: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance.
* **Minkowski Distance**: Generalization of Euclidean and Manhattan distance.

**Steps to be carried out during the K-NN algorithm are as follows :**

1. Divide the data into training and test data.
2. Select a value K.
3. Determine which distance function is to be used.
4. Choose a sample from the test data that needs to be classified and compute the distance to its n training samples.
5. Sort the distances obtained and take the k-nearest data samples.
6. Assign the test class to the class based on the majority vote of its k neighbors.



Source: DataCamp

**Performance of the K-NN algorithm is influenced by three main factors :**

1. The **distance function** or distance metric used to determine the nearest neighbors.
2. The **decision rule used to derive a classification** from the K-nearest neighbors.
3. The **number of neighbors** used to classify the new example.

**Advantages of K-NN :**

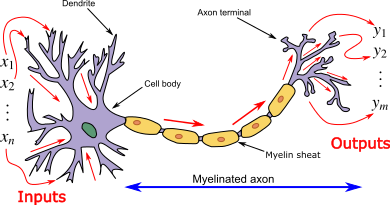
1. The K-NN algorithm is very easy to implement.
2. Nearly optimal in the large sample limit.
3. Uses local information, which can yield highly adaptive behavior.
4. Lends itself very easily to parallel implementation.

**ANN — Artificial Neural Network**

**ARTIFICIAL NEURAL NETWORK (ANN)**

**Neurons**

Biological Neurons (also called nerve cells) or simply neurons are the fundamental units of the brain and nervous system, the cells responsible for receiving sensory input from the external world via dendrites, process it and gives the output through Axons.



A biological Neuron

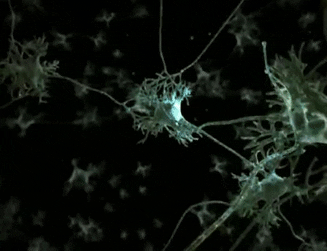
**Cell body (Soma):**The body of the neuron cell contains the nucleus and carries out biochemical transformation necessary to the life of neurons.

**Dendrites:**Each neuron has fine, hair-like tubular structures (extensions) around it. They branch out into a tree around the cell body. They accept incoming signals.

**Axon:** It is a long, thin, tubular structure that works like a transmission line.

**Synapse:**Neurons are connected to one another in a complex spatial arrangement. When axon reaches its final destination it branches again called terminal arborization. At the end of the axon are highly complex and specialized structures called synapses. The connection between two neurons takes place at these synapses.

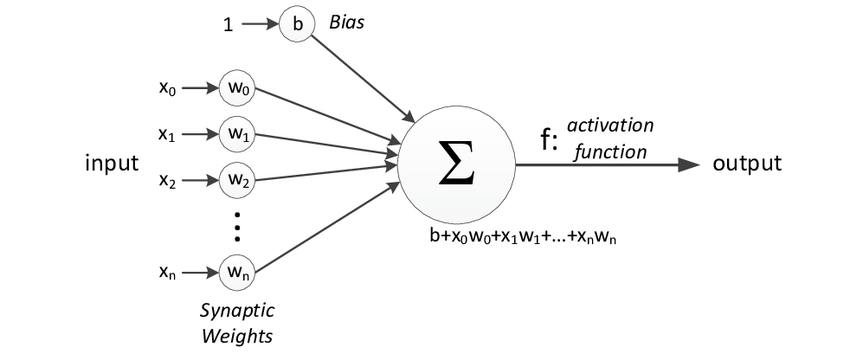
Dendrites receive input through the synapses of other neurons. The soma processes these incoming signals over time and converts that processed value into an output, which is sent out to other neurons through the axon and the synapses.



The flow of electric signals through neurons.

The following diagram represents the general model of ANN which is inspired by a biological neuron. It is also called Perceptron.

A single layer neural network is called a Perceptron. It gives a single output.



Perceptron

In the above figure, for one single observation, x0, x1, x2, x3...x(n) represents various inputs(independent variables) to the network. Each of these inputs is multiplied by a connection weight or synapse. The weights are represented as w0, w1, w2, w3….w(n) . **Weight shows the strength of a particular node.**

b is a bias value. A bias value allows you to shift the activation function up or down.

In the simplest case, these products are summed, fed to a transfer function (activation function) to generate a result, and this result is sent as output.

Mathematically, x1.w1 + x2.w2 + x3.w3 ...... xn.wn = ∑ xi.wi

Now activation function is applied 𝜙(∑ xi.wi)

**Activation function**

The Activation function is important for an ANN to learn and make sense of something really complicated. Their main purpose is to convert an input signal of a node in an ANN to an output signal. This output signal is used as input to the next layer in the stack.

***Activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The motive is to introduce non-linearity into the output of a neuron.***

If we do not apply activation function then the output signal would be simply linear function(one-degree polynomial). Now, a linear function is easy to solve but they are limited in their complexity, have less power. Without activation function, our model cannot learn and model complicated data such as images, videos, audio, speech, etc.

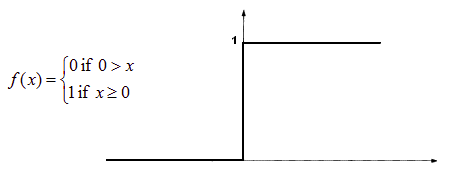
**Now the question arises why do we need Non-Linearity?**

Non-Linear functions are those which have a degree more than one and they have a curvature. Now we need a neural network to learn and represent almost anything and any arbitrary complex function that maps an input to output.

Neural Network is considered **“Universal Function Approximators”.**It means they can learn and compute any function at all.

**Types of Activation Functions:**

**1.Threshold Activation Function — (Binary step function)**  
A Binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.



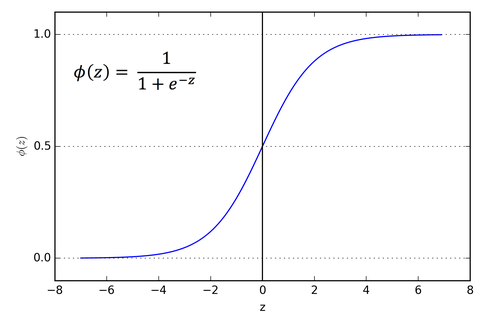
A Binary step function

Activation function A = “activated” if Y > threshold

else not or A=1 if y>threshold 0 otherwise.

The problem with this function is for creating a binary classifier ( 1 or 0), but if you want multiple such neurons to be connected to bring in more classes, Class1, Class2, Class3, etc. In this case, all neurons will give 1, so we cannot decide.

**2. Sigmoid Activation Function — (Logistic function)**  
A Sigmoid function is a mathematical function having a characteristic “S”-shaped curve or sigmoid curve which ranges between 0 and 1, therefore it is used for models where we need to predict the probability as an output.

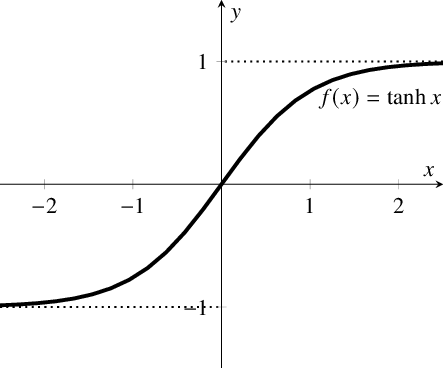


Sigmoid curve

The Sigmoid function is differentiable, means we can find the slope of the curve at any 2 points.

The drawback of the Sigmoid activation function is that it can cause the neural network to get stuck at training time if strong negative input is provided.

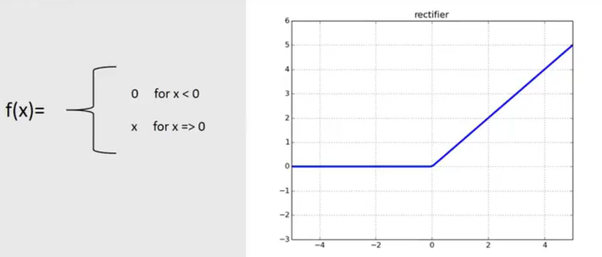
**3. Hyperbolic Tangent Function — (tanh)**  
It is similar to Sigmoid but better in performance. It is nonlinear in nature, so great we can stack layers. The function ranges between (-1,1).



Hyperbolic tangent function

The main advantage of this function is that strong negative inputs will be mapped to negative output and only zero-valued inputs are mapped to near-zero outputs.,So less likely to get stuck during training.

**4. Rectified Linear Units — (ReLu)**  
ReLu is the most used activation function in CNN and ANN which ranges from zero to infinity.[0,∞)



ReLu

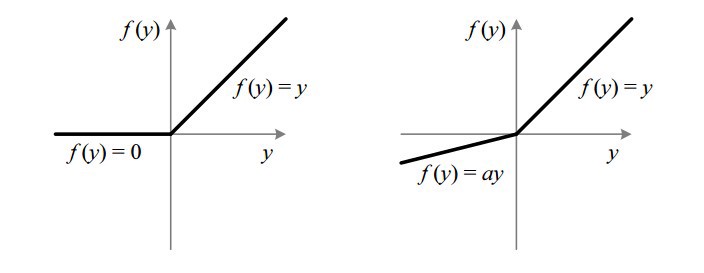
It gives an output ‘x’ if x is positive and 0 otherwise. It looks like having the same problem of linear function as it is linear in the positive axis. Relu is non-linear in nature and a combination of ReLu is also non-linear. In fact, it is a good approximator and any function can be approximated with a combination of Relu.

ReLu is 6 times improved over hyperbolic tangent function.

It should only be applied to hidden layers of a neural network. So, for the output layer use softmax function for classification problem and for regression problem use a Linear function.

Here one problem is some gradients are fragile during training and can die. It causes a weight update which will make it never activate on any data point again. Basically ReLu could result in dead neurons.

To fix the problem of dying neurons, **Leaky ReLu** was introduced. So, Leaky ReLu introduces a small slope to keep the updates alive. Leaky ReLu ranges from -∞ to +∞.



ReLu vs Leaky ReLu

Leak helps to increase the range of the ReLu function. Usually, the value of a = 0.01 or so.

When a is not 0.01, then it is called Randomized ReLu.

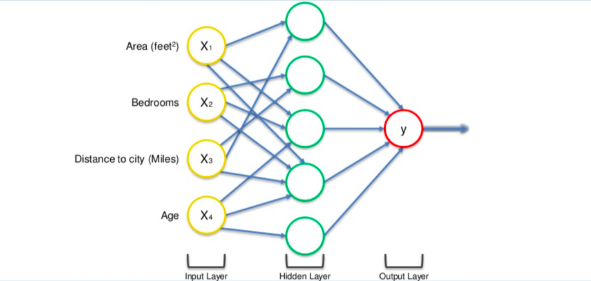
**How does the Neural network work?**

Let us take the example of the price of a property and to start with we have different factors assembled in a single row of data: Area, Bedrooms, Distance to city and Age.



The input values go through the weighted synapses straight over to the output layer. All four will be analyzed, an activation function will be applied, and the results will be produced.

This is simple enough but there is a way to amplify the power of the Neural Network and increase its accuracy by the addition of a hidden layer that sits between the input and output layers.



A neural network with a hidden layer(only showing non-0 values)

Now in the above figure, all 4 variables are connected to neurons via a synapse. However, not all of the synapses are weighted. they will either have a 0 value or non-0 value.

here, the non-0 value → indicates the importance

0 value → They will be discarded.

Let's take the example of Area and Distance to City are non-zero for the first neuron, which means they are weighted and matter to the first neuron. The other two variables, Bedrooms and Age aren’t weighted and so are not considered by the first neuron.

You may wonder why that first neuron is only considering two of the four variables. In this case, it is common on the property market that larger homes become cheaper the further they are from the city. That’s a basic fact. So what this neuron may be doing is looking specifically for properties that are large but are not so far from the city.

Now, this is where the power of neural networks comes from. There are many of these neurons, each doing similar calculations with different combinations of these variables.

Once this criterion has been met, the neuron applies the activation function and do its calculations. The next neuron down may have weighted synapses of Distance to the city and, Bedrooms.

This way the neurons work and interact in a very flexible way allowing it to look for specific things and therefore make a comprehensive search for whatever it is trained for.

**How do Neural networks learn?**

Looking at an analogy may be useful in understanding the mechanisms of a neural network. Learning in a neural network is closely related to how we learn in our regular lives and activities — we perform an action and are either accepted or corrected by a trainer or coach to understand how to get better at a certain task. Similarly, neural networks require a trainer in order to describe what should have been produced as a response to the input. Based on the difference between the actual value and the predicted value, an error value also called **Cost Function** is computed and sent back through the system.

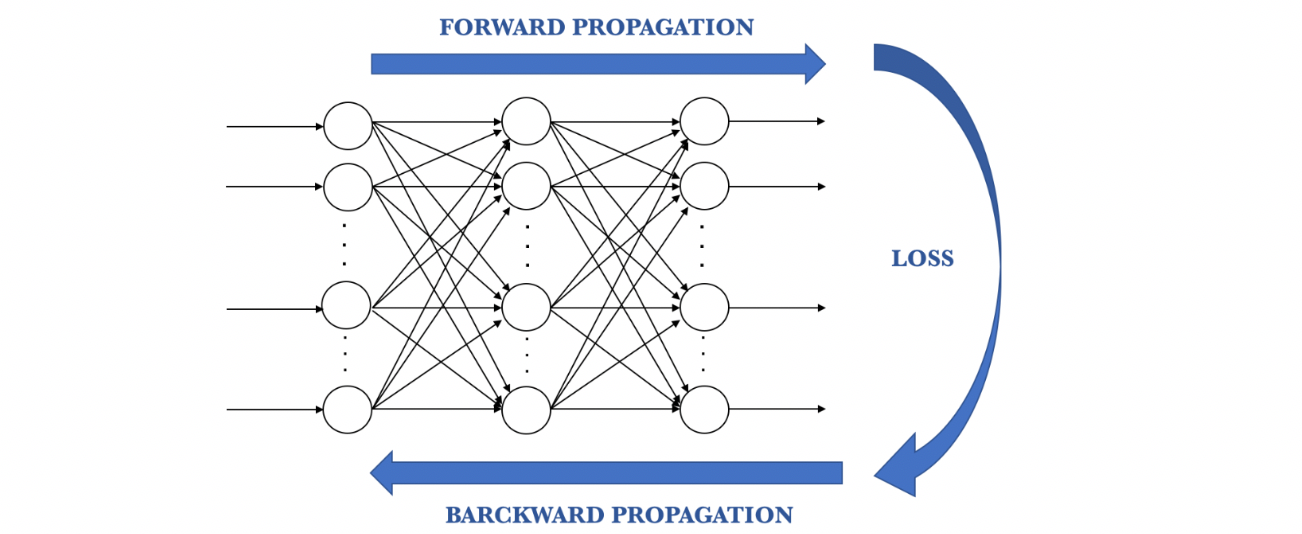
***Cost Function: One half of the squared difference between actual and output value.***

For each layer of the network, the cost function is analyzed and used to adjust the threshold and weights for the next input. Our aim is to minimize the cost function. The lower the cost function, the closer the actual value to the predicted value. In this way, the error keeps becoming marginally lesser in each run as the network learns how to analyze values.

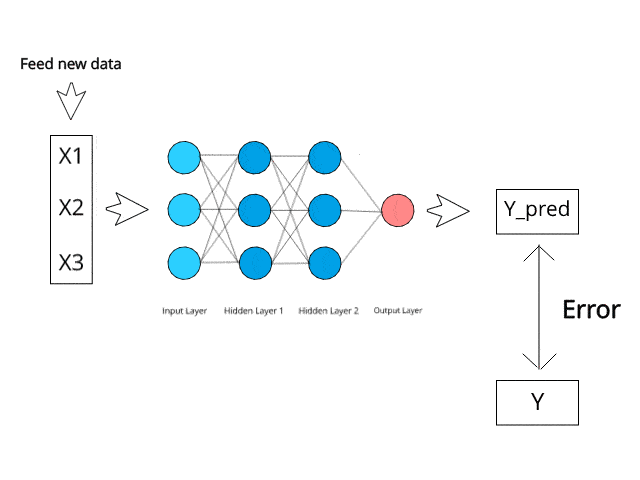
We feed the resulting data back through the entire neural network. The weighted synapses connecting input variables to the neuron are the only thing we have control over.

As long as there exists a disparity between the actual value and the predicted value, we need to adjust those wights. Once we tweak them a little and run the neural network again, A new Cost function will be produced, hopefully, smaller than the last.

We need to repeat this process until we scrub the cost function down to as small as possible.



The procedure described above is known as **Back-propagation** and is applied continuously through a network until the error value is kept at a minimum.



Back-propagation in action

There are basically 2 ways to adjust weights: —  
1. Brute-force method  
2. Batch-Gradient Descent

**Brute-force method**

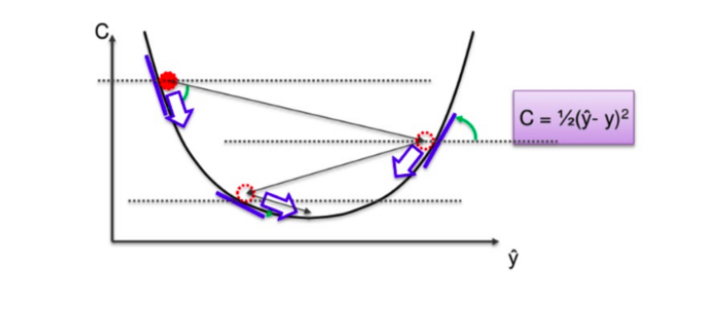
Best suited for the single-layer feed-forward network. Here you take a number of possible weights. In this method, we want to eliminate all the other weights except the one right at the bottom of the U-shaped curve.

Optimal weight can be found using simple elimination techniques. This process of elimination work if you have one weight to optimize. What if you have complex NN with many numbers of weights, then this method fails because of the **Curse of Dimensionality.**

The alternative approach that we have is called Batch Gradient Descent.

**Batch-Gradient Descent**

It is a first-order iterative optimization algorithm and its responsibility is to find the minimum cost value(loss) in the process of training the model with different weights or updating weights.



Gradient Descent

In Gradient Descent, instead of going through every weight one at a time, and ticking every wrong weight off as you go, we instead look at the angle of the function line.

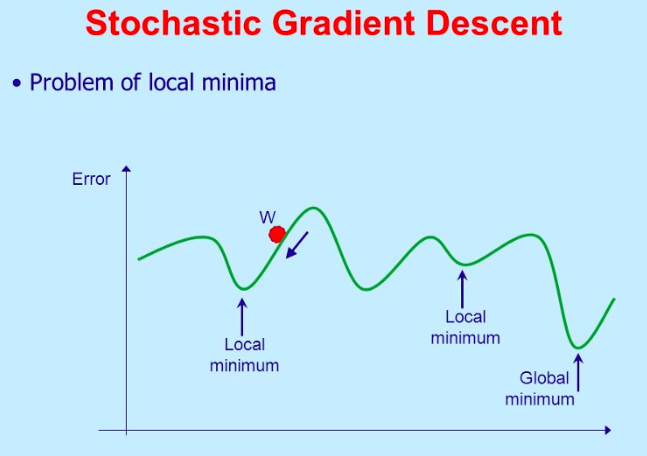
**If slope → Negative, that means yo go down the curve.  
If slope → Positive, Do nothing**

This way a vast number of incorrect weights are eliminated. For instance, if we have 3 million samples, we have to loop through 3 million times. So basically you need to calculate each cost 3 million times.

**Stochastic Gradient Descent(SGD)**

Gradient Descent works fine when we have a convex curve just like in the above figure. But if we don't have a convex curve, Gradient Descent fails.

The word ‘*stochastic*‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.



Stochastic Gradient Descent

In SGD, we take one row of data at a time, run it through the neural network then adjust the weights. For the second row, we run it, then compare the Cost function and then again adjusting weights. And so on…

SGD helps us to avoid the problem of local minima. It is much faster than Gradient Descent because it is running each row at a time and it doesn’t have to load the whole data in memory for doing computation.

One thing to be noted is that, as SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of its randomness in its descent. Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent. Hence, in most scenarios, SGD is preferred over Batch Gradient Descent for optimizing a learning algorithm.

**Training ANN with Stochastic Gradient Descent**

Step-1 → Randomly initialize the weights to small numbers close to 0 but not 0.

Step-2 → Input the first observation of your dataset in the input layer, each feature in one node.

Step-3 → **Forward-Propagation**: From left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted value.

Step-4 → Compare the predicted result to the actual result and measure the generated error(Cost function).

Step-5 → **Back-Propagation**: from right to left, the error is backpropagated. Update the weights according to how much they are responsible for the error. The learning rate decides how much we update weights.

Step-6 → Repeat step-1 to 5 and update the weights after each observation (Reinforcement Learning)

Step-7 → When the whole training set passed through the ANN, that makes and epoch. Redo more epochs.

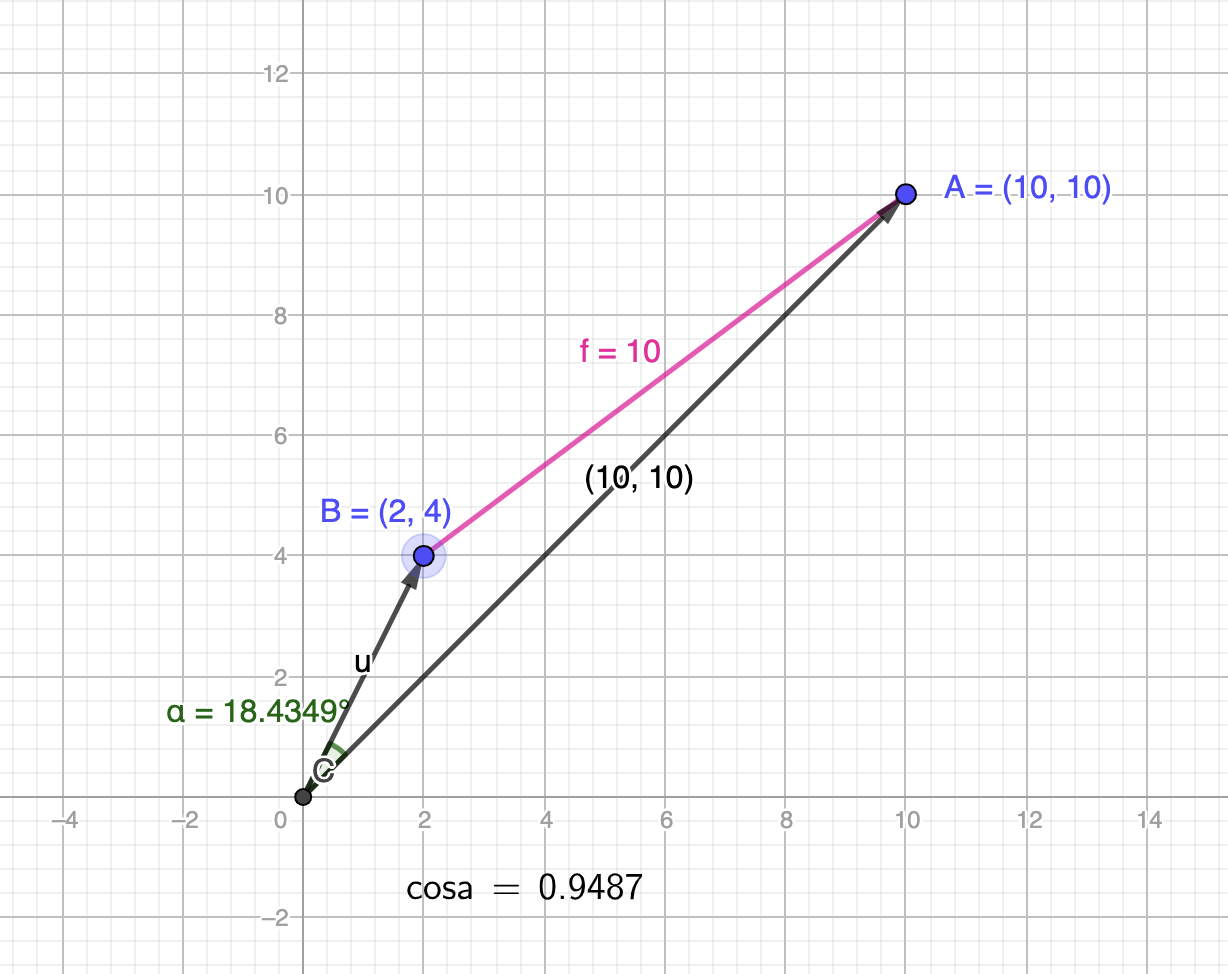
## **Cosine Similarity Matrix:**

Now suppose you work for a pay tv channel and you have the results of a survey from two groups of subscribers. One of the anaysis could be about the similarities of tastes between the two groups. For this type of analysis we are interested to select people sharing similar behaviours regardless of “how much time” they watch TV. This is well represented by the concept of cosine similarity which allow to consider as “close” those ‘observations’ aligned to some interesting for us directions regardless of how different the magnitude of the measures are from each other .

So as an example if “person A” watches 10 hours of sport and 4 hours movies and “person B” watches 5 hours of sport, 2 hours movies, we can see the twos are (perfectly in this case) aligned given the fact that regardless of how many hours in total they watch TV, in proportion they share the same behaviours.

By contrast if the objective is to analyse those watching similar number of hours in an interval, the euclidean distance would have been more appropriate as that evaluates the distance as we are used normally to think.

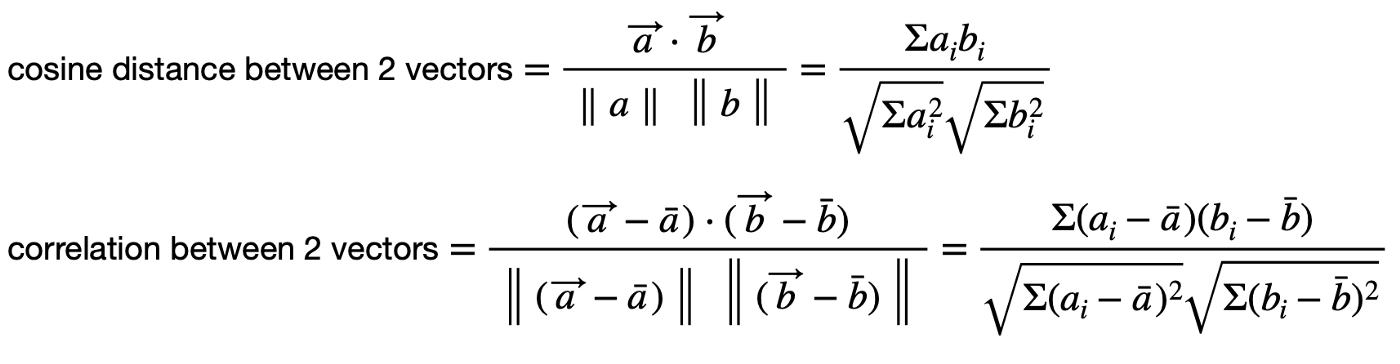
It’s rather intuitive from the chart below to see this comparing the two points A and B with the length of segment f=10 (euclidean distance) with cosine of angle alpha = 0.9487 which oscillates between 1 and -1 where 1 means same direction same orientation, -1 same direction but opposite orientation.



Simple example to how cosine of alpha (0.94) show a good alignment between the two vectors (OA) and (OB)

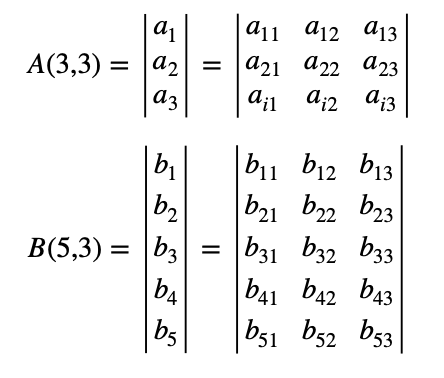
If the orientation is not important in our analysis the module of cosine would null this effect and consider +1 the same as -1.

In terms of formulas cosine similarity is related to Pearson’s correlation coefficient by almost the same formula as cosine similarity is Pearson’s correlation when vectors are centered on their mean:



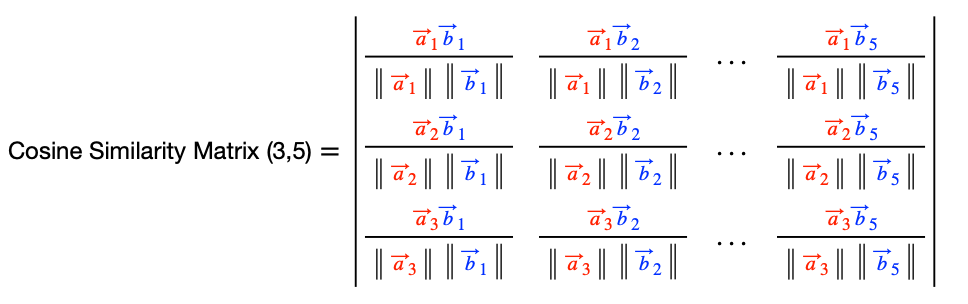
The generalization of the cosine similarity concept when we have many points in a data matrix A to be compared with themselves (cosine similarity matrix using A vs. A) or to be compared with points in a second data matrix B (cosine similarity matrix of A vs. B with the same number of dimensions) is the same problem.

So to make things different from usual we want to calculate the Cosine Similarity Matrix of a group of points A vs. a second group of points B, both with same number of variables (columns) like this:



(image by author)

Assuming the vectors to be compared are in the rows of A and B the Cosine Similarity Matrix would appear as follows where each cell is cosine of the angle between all the vectors of A (rows) with all the vectors of B (columns):



(image by author)

If you look at the color pattern you see that first vectors “a” replicate itself by row, while vectors “b” replicates itself by columns.

To calculate this matrix in (almost) one line of code we need to look for a way to use what we know of algebra for numerator and denominator and then put it all together.

# Keras

Keras is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) neural-network library written in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)). It is capable of running on top of [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow), [Microsoft Cognitive Toolkit](https://en.wikipedia.org/wiki/Microsoft_Cognitive_Toolkit), [Theano](https://en.wikipedia.org/wiki/Theano_(software)), or [PlaidML](https://en.wikipedia.org/wiki/PlaidML" \o "PlaidML). Designed to enable fast experimentation with [deep neural networks](https://en.wikipedia.org/wiki/Deep_learning), it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a [Google](https://en.wikipedia.org/wiki/Google) engineer. Chollet also is the author of the XCeption deep neural network model.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. [Microsoft](https://en.wikipedia.org/wiki/Microsoft) added a [CNTK](https://en.wikipedia.org/wiki/CNTK) backend to Keras as well, available as of CNTK v2.0.

**Use Keras if you need a deep learning library that:**

* Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
* Supports both convolutional networks and recurrent networks, as well as combinations of the two.
* Runs seamlessly on CPU and GPU.

**Guiding principles**

* User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
* Modularity. A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.
* Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
* Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

# Models

The core data structure of Keras is a model, a way to organize layers. The simplest type of model is the [Sequential](https://keras.io/getting-started/sequential-model-guide) model, a linear stack of layers. For more complex architectures, you should use the [Keras functional API](https://keras.io/getting-started/functional-api-guide), which allows to build arbitrary graphs of layers.

Here is the Sequential model:

from keras.models import Sequential

model = Sequential()

Stacking layers is as easy as .add():

from keras.layers import Dense

model.add(Dense(units=64, activation='relu', input\_dim=100))

model.add(Dense(units=10, activation='softmax'))

Once your model looks good, configure its learning process with .compile():

model.compile(loss='categorical\_crossentropy',

optimizer='sgd',

metrics=['accuracy'])

If you need to, you can further configure your optimizer. A core principle of Keras is to make things reasonably simple, while allowing the user to be fully in control when they need to (the ultimate control being the easy extensibility of the source code).

model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))

You can now iterate on your training data in batches:

# x\_train and y\_train are Numpy arrays --just like in the Scikit-Learn API.

model.fit(x\_train, y\_train, epochs=5, batch\_size=32)

Alternatively, you can feed batches to your model manually:

model.train\_on\_batch(x\_batch, y\_batch)

Evaluate your performance in one line:

loss\_and\_metrics = model.evaluate(x\_test, y\_test, batch\_size=128)

Or generate predictions on new data:

classes = model.predict(x\_test, batch\_size=128)

Building a question answering system, an image classification model, a Neural Turing Machine, or any other model is just as fast. The ideas behind deep learning are simple, so why should their implementation be painful?

In the [examples folder](https://github.com/keras-team/keras/tree/master/examples) of the repository, you will find more advanced models: question-answering with memory networks, text generation with stacked LSTMs, etc.

Installation

Before installing Keras, please install one of its backend engines: TensorFlow, Theano, or CNTK. We recommend the TensorFlow backend.

* [TensorFlow installation instructions](https://www.tensorflow.org/install/).
* [Theano installation instructions](http://deeplearning.net/software/theano/install.html#install).
* [CNTK installation instructions](https://docs.microsoft.com/en-us/cognitive-toolkit/setup-cntk-on-your-machine).

You may also consider installing the following optional dependencies:

* [cuDNN](https://docs.nvidia.com/deeplearning/sdk/cudnn-install/) (recommended if you plan on running Keras on GPU).
* HDF5 and [h5py](http://docs.h5py.org/en/latest/build.html) (required if you plan on saving Keras models to disk).
* [graphviz](https://graphviz.gitlab.io/download/) and [pydot](https://github.com/erocarrera/pydot) (used by [visualization utilities](https://keras.io/visualization/) to plot model graphs).

Then, you can install Keras itself. There are two ways to install Keras:

* Install Keras from PyPI (recommended):

sudo pip install keras

If you are using a virtualenv, you may want to avoid using sudo:

pip install keras

* Alternatively: install Keras from the GitHub source:

First, clone Keras using git:

git clone https://github.com/keras-team/keras.git

Then, cd to the Keras folder and run the install command:

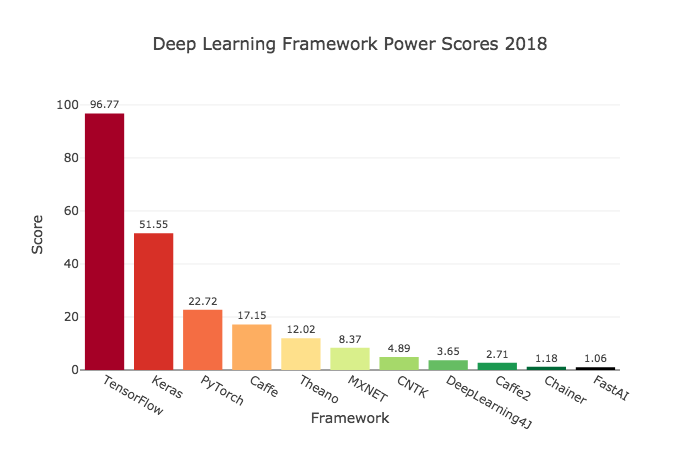
cd keras

sudo python setup.py install

Keras prioritizes developer experience

* Keras is an API designed for human beings, not machines. [Keras follows best practices for reducing cognitive load](https://blog.keras.io/user-experience-design-for-apis.html): it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
* This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn [helps you win machine learning competitions](https://www.quora.com/Why-has-Keras-been-so-successful-lately-at-Kaggle-competitions).
* This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.

Keras has broad adoption in the industry and the research community

[](https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a)

Deep learning frameworks ranking computed by Jeff Hale, based on 11 data sources across 7 categories

With over 250,000 individual users as of mid-2018, Keras has stronger adoption in both the industry and the research community than any other deep learning framework except TensorFlow itself (and the Keras API is the official frontend of TensorFlow, via the tf.keras module).

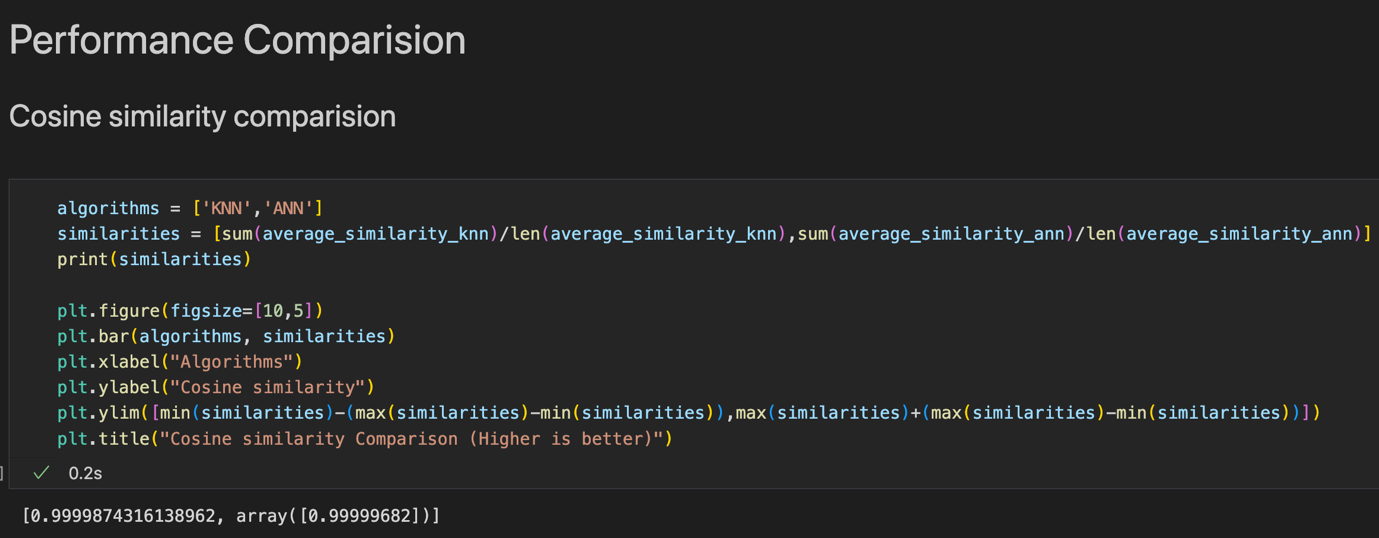
You are already constantly interacting with features built with Keras -- it is in use at Netflix, Uber, Yelp, Instacart, Zocdoc, Square, and many others. It is especially popular among startups that place deep learning at the core of their products.

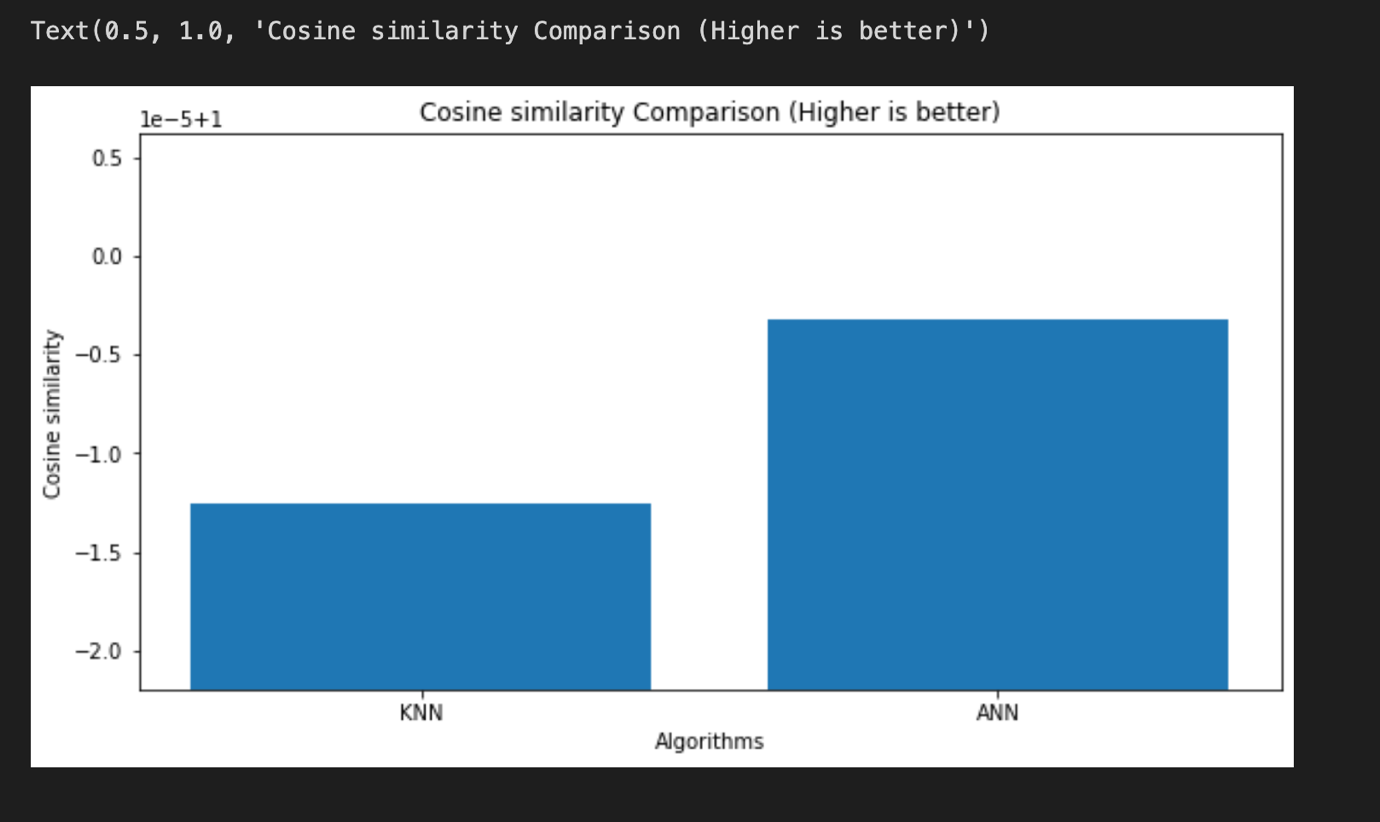
Keras is also a favorite among deep learning researchers, coming in #2 in terms of mentions in scientific papers uploaded to the preprint server [arXiv.org](https://arxiv.org/archive/cs). Keras has also been adopted by researchers at large scientific organizations, in particular CERN and NASA.

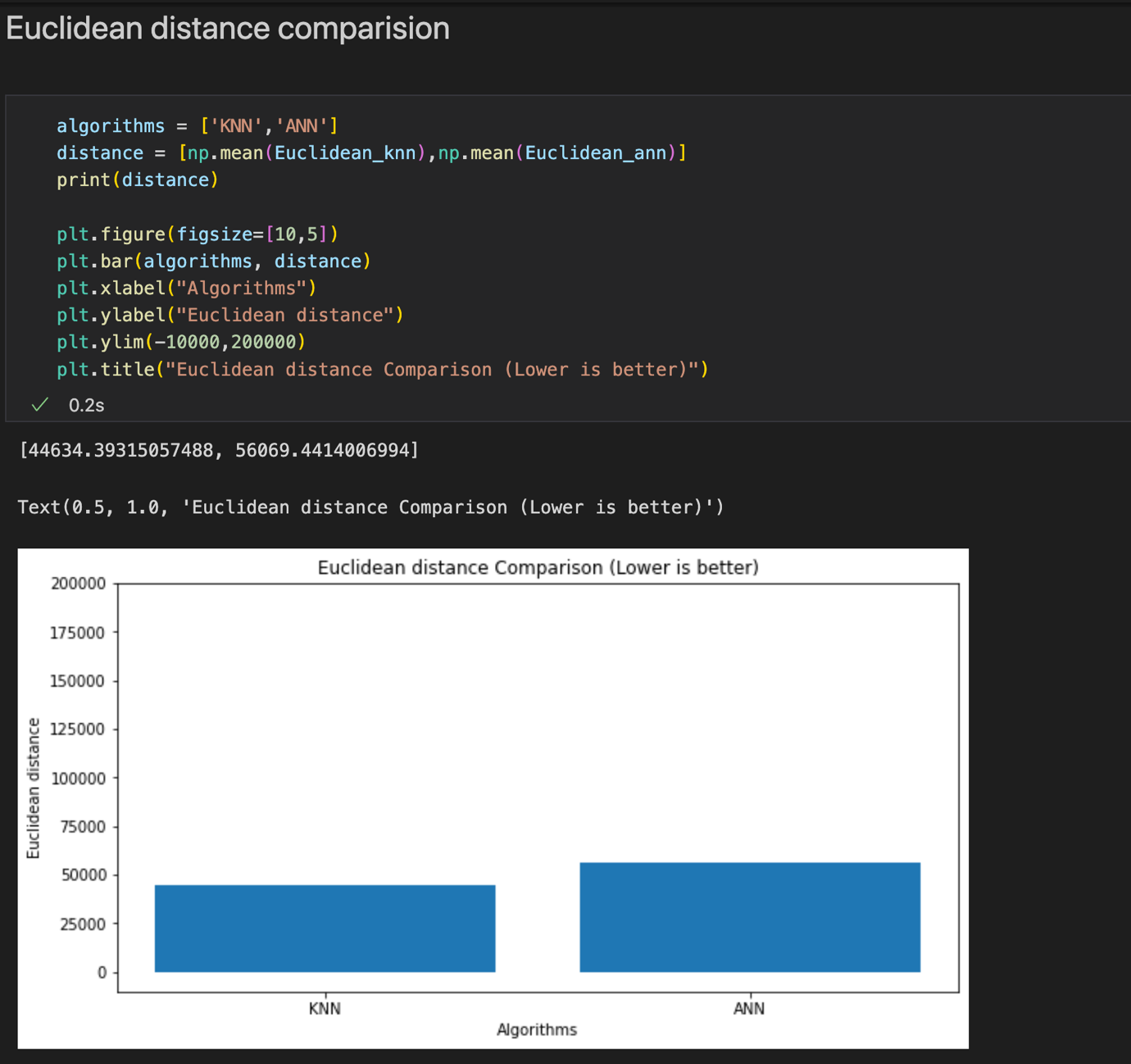
**Performance Comparision :**

**KNN model:** 0.999955

**ANN model:** 0.999994

****

****

****

**Conclusion:**

For the song dataset, ANN recommendation system is working more accurately and giving the higher performance.