Emoji is a module used to add emojis

To install we use pip install emoji

Import emoji

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

import matplotlib.pyplot as plt

->matplotlib is a library in python for data visualization

.pyplot is a module within the matplotlib library

As plt is an alias so you don’t have to write the library name on every call

**Data visualization** is the discipline of trying to understand **data** by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed.

n computer programming, **pandas** is a software library written for the **Python** programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

The package contains multiple methods for convenient data filtering. **Pandas** has a variety of utilities to **perform** Input/Output operations in a seamless manner. It **can** read data from a variety of formats such as CSV, TSV, MS Excel, etc.

Read\_csv() is a function from pandas used to read files of the format csv,excel,etcc

Csv-comma separated value: .csv files are usually used to import or export important data ;it is used commonly because it is smaller in size and can be uploaded faster

single line of code involving read\_csv() from pandas, you:

* Located the CSV file you want to import from your filesystem.
* Corrected the headers of your dataset.
* Dealt with missing values so that they're encoded properly as NaNs.
* Corrected data types for every column in your dataset.
* Converted a CSV file to a Pandas DataFrame

 the read\_csv() function correctly infers that the first observation contains the headers for the dataset. Not only that, read\_csv() can infer the data types for each column of your dataset as well.

The N-dimensional **array** ( **ndarray** ) An **ndarray** is a (usually fixed-size) multidimensional container of items of the same type and size.

The training set is a subset of the data set used to train a model.

* x\_train is the training data set.
* y\_train is the set of labels to all the data in x\_train.

The test set is a subset of the data set that you use to test your model after the model has gone through initial vetting by the validation set.

* x\_test is the test data set.
* y\_test is the set of labels to all the data in x\_test.

The variables **train\_data** and test\_data **are** lists of reviews; each review **is** a list of word indices (encoding a sequence of words). train\_labels and test\_labels **are** lists of 0s and 1s, where **0** stands for negative and 1 stands for positive:

"Word embeddings" are a family of natural language processing techniques aiming at mapping semantic meaning into a geometric space. This is done by associating a numeric vector to every word in a dictionary, such that the distance (e.g. L2 distance or more commonly cosine distance) between any two vectors would capture part of the semantic relationship between the two associated words. The geometric space formed by these vectors is called an embedding space.

A word embedding is a class of approaches for representing words and documents using a dense vector representation.

Word embeddings are computed by applying dimensionality reduction techniques to datasets of co-occurence statistics between words in a corpus of text. This can be done via neural networks (the "word2vec" technique), or via matrix factorization.

GloVe word embeddings

We will be using GloVe embeddings, which you can read about [here](http://nlp.stanford.edu/projects/glove/). GloVe stands for "Global Vectors for Word Representation". It's a somewhat popular embedding technique based on factorizing a matrix of word co-occurence statistics.

Specifically, we will use the 100-dimensional GloVe embeddings of 400k words computed on a 2014 dump of English Wikipedia. You can download them [here](http://nlp.stanford.edu/data/glove.6B.zip) (warning: following this link will start a 822MB download).

GloVe: global vector for word representation

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Embeddings:

 we compute an index mapping words to known embeddings, by parsing the data dump of pre-trained embeddings:

embeddings\_index = {}

At this point we can leverage our embedding\_index dictionary and our word\_index to compute our embedding matrix:

Neural Networks are set of algorithms which closely resemble the human brain and are designed to recognize patterns. They interpret sensory data through a machine perception, labelling or clustering raw input. They can recognize numerical patterns, contained in vectors, into which all real-world data ( images, sound, text or time series), must be translated. Artificial neural networks are composed of a large number of highly interconnected processing elements (neuron) working together to solve a problem

Recurrent Neural Network(RNN) is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation.

**Keras** is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

* The resulting dimensions are: (batch, sequence, embedding).

The simplest model in keras is sequential; class which is a linear stack

The Sequential model is a linear stack of layers.

The model needs to know what input shape it should expect. For this reason, the first layer in a Sequential model (and only the first, because following layers can do automatic shape inference) needs to receive information about its input shape. There are several possible ways to do this:

* Pass an input\_shape argument to the first layer. This is a shape tuple (a tuple of integers or None entries, where None indicates that any positive integer may be expected). In input\_shape, the batch dimension is not included.
* A metric is a function that is used to judge the performance of your model.
* Metric functions are similar to loss functions, except that the results from evaluating a metric are not used when training the model. Note that you may use any loss functions as a metric function.
* Computes the crossentropy metric between the labels and predictions.
* Use this crossentropy metric when there are two or more label classes. We expect labels to be provided as integers. If you want to provide labels using one-hot representation, please use CategoricalCrossentropy metric. There should be # classes floating point values per feature for y\_pred and a single floating point value per feature for y\_true.
* In the snippet below, there is a single floating point value per example for y\_true and # classes floating pointing values per example for y\_pred. The shape of y\_true is [batch\_size] and the shape of y\_pred is [batch\_size, num\_classes].

The compile() method takes a metrics argument, which is a list of metrics:

Much like loss functions, any callable with signature metric\_fn(y\_true, y\_pred) that returns an array of losses (one of sample in the input batch) can be passed to compile() as a metric.

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

an **epoch** refers to one cycle through the full training dataset.

Train the model by slicing the data into "batches" of size "batch\_size", and repeatedly iterating over the entire dataset for a given number of "epochs"

With this approach our model reaches a validation accuracy of around 88% (note the model is overfitting, training accuracy is significantly higher).

The EarlyStopping function has various metrics/arguments that you can modify to set up when the training process should stop. Here are some relevant metrics:

* **monitor**: value being monitored, i.e: val\_loss
* **min\_delta**: minimum change in the monitored value. For example, min\_delta=1 means that the training process will be stopped if the absolute change of the monitored value is less than 1
* **patience**: number of epochs with no improvement after which training will be stopped
* **restore\_best\_weights**: set this metric to True if you want to keep the best weights once stopped

# ModelCheckpoint

This callback saves the model after every epoch. Here are some relevant metrics:

* **filepath**: the file path you want to save your model in
* **monitor**: the value being monitored
* **save\_best\_only**: set this to True if you do not want to overwrite the latest best model
* **mode**: auto, min, or max. For example, you set mode=’min’ if the monitored value is val\_loss and you want to minimize it.

When given a batch of sequences as input, an embedding layer returns a 3D floating point tensor, of shape (samples, sequence\_length, embedding\_dimensionality).

The returned "history" object holds a record of the loss values and metric values during training

A class prediction is given the finalized model and one or more data instances, predict the class for the data instances.

We do not know the outcome classes for the new data. That is why we need the model in the first place.

We can predict the class for new data instances using our finalized classification model in Keras using the *predict\_classes()* function. Note that this function is only available on *Sequential* models, not those models developed using the functional API.

train\_embeddings = embedding\_output(x\_train)

test\_embeddings = embedding\_output(x\_test)

// we are loading the data

To train a model with fit, you need to specify a loss function, an optimizer, and optionally, some metrics to monitor.

You pass these to the model as arguments to the compile() method:

hist = model.fit(train\_embeddings,y\_train,validation\_split=0.1,shuffle=True,epochs=100,batch\_size=64)

//is used to train the model and make it fit