Keras:

\* Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

\* TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. Both provide high-level APIs used for easily building and training models, but Keras is more user-friendly because it's built-in Python.

\* Steps to create a neural network model with Keras:

1. Define the training data

2. Define a neural network model

3. Configure the learning process

4. Train the model

**DEFINING THE TRAINING MODEL :**

\* The Sequential class is used to define a linear stack of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method. The Dense layer has an output of size 16, and an input of size INPUT\_DIM (user defined).

\*Activation function:

The activation functions are at the very core of Deep Learning. They determine the output of a model, its accuracy, and computational efficiency. In some cases, activation functions have a major effect on the model’s ability to converge and the convergence speed.

e.g.: rectified linear unit activation fnc, softmax

\*Layer activation fnc :

- ReLU :

1. The Rectified Linear Unit (ReLU) is the most commonly used activation function in deep learning. The function returns 0 if the input is negative, but for any positive input, it returns that value back. The function is defined as:

0 if x<=0

x if x>0

2. Graphically, the ReLU function is composed of two linear pieces to account for non-linearities. A function is non-linear if the slope isn’t constant. So, the ReLU function is non-linear around 0, but the slope is always either 0 (for negative inputs) or 1 (for positive inputs).

3. Returns a Tensor representing the input tensor, transformed by the relu activation function. Tensor will be of the same shape and dtype of input x. (tensor - mathematical obj that holds your value. Data container. Like a vector which is basically an array or a 2D matrix)

- Softmax :

1. Softmax converts a vector of values to a probability distribution. The elements of the output vector are in range (0, 1) and sum to 1. Each vector is handled independently. The axis argument sets which axis of the input the function is applied along. Softmax is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution.

2. The softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one.

- Formulae used by the funcs :

1. ReLU : With default values, this returns the standard ReLU activation: max(x, 0), the element-wise maximum of 0 and the input tensor.

2. Softmax : The softmax of each vector x is computed as exp(x) / tf.reduce\_sum(exp(x)).

\*BatchNormalization :

normalizes each layer of neural network

\*Dropout :

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

**COMPILING MODEL :**

\*Optimizer : Optimizers are Classes or methods used to change the attributes of your machine/deep learning model such as weights and learning rate in order to reduce the losses.

- Adam : **Adam**stands for **adaptive moment estimation**, which is another way of using past gradients to calculate current gradients. Adam utilizes the concept of momentum by adding fractions of previous gradients to the current one.

\*Losses : The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

- Probabilistic Loss : sparse\_categorical\_crossentropy

- Regression Loss : mean\_squared\_error

- Sparse categorical cross entropy : It is used when there are two or more label classes present in our case statement, and labels are expected to be provided in integers. It calculates the cross entropy between the true labels and predicted output.

**Now how the loss functions and optimizers are related?**

During the training of the model, we tune the parameters and weights to minimize the loss and try to make our prediction accuracy as correct as possible. Now to change these parameters the optimizer’s role came in, which ties the model parameters with the loss function by updating the model in response to the loss function output. Simply optimizers shape the model into its most accurate form by playing with model weights. The loss function just tells the optimizer when it’s moving in the right or wrong direction.

\*Metrics : A metric is a function that is used to judge the performance of your model.

**TRAINING MODEL :**

- fit() requires 2 arguments: input and target tensors

- batch size : it is the no of observations after which each weight is updated

- epochs : no of times we pass our training dataset through the ANN

**Callback (optional) before training model:**

* EarlyStopping : Assuming the goal of a training is to minimize the loss. With this, the metric to be monitored would be 'loss', and mode would be 'min'. A model.fit() training loop will check at end of every epoch whether the loss is no longer decreasing, considering the min\_delta and patience if applicable. Once it's found no longer decreasing, model.stop\_training is marked True and the training terminates.

Overfitting is a nightmare for Machine Learning practitioners. One way to avoid overfitting is to terminate the process early. The EarlyStoppingfunction 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
* ReduceLROnPlateau : ReduceLROnPlateau is a callback to reduce the learning rate when a metric has stopped improving. This callback monitors a quantity and if no improvement is seen for a patience number of epochs, the learning rate is reduced by factor value (new\_lr = lr \* factor). Let’s see how this works with the help of an example.

Resources:

1. <https://keras.io/api/layers/core_layers/dense/>
2. <https://www.kdnuggets.com/2018/06/keras-4-step-workflow.html>
3. For audio data, classification, feature extraction, genre classification : <https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html>