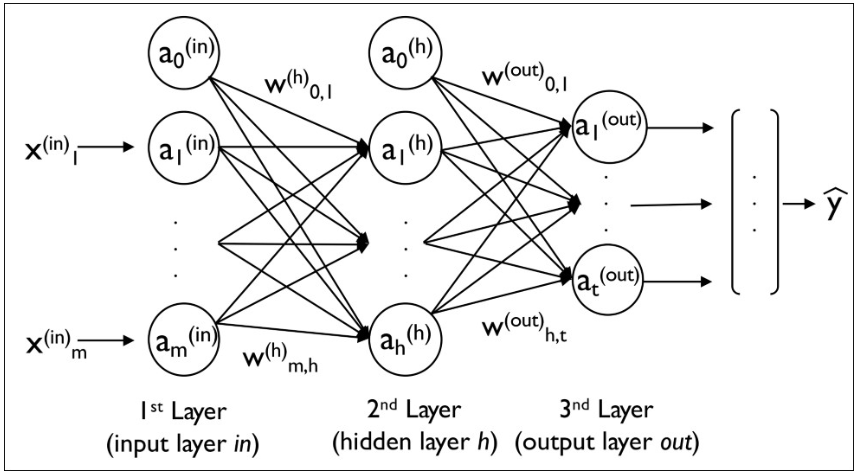
**PART I: ML (Fully Connected Neural Networks) with Keras**

**Multilayer Perceptions**

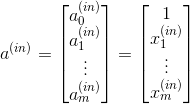
Multilayer feedforward neural networks are a special type of *fully connected* network with multiple single neurons. They are also called **Multilayer perceptions** (**MLP**). The following figure illustrates the concept of an MLP consisting of three layers:



The MLP depicted in the preceding figure has one input layer, one hidden layer, and one output layer. The units in the hidden layer are fully connected to the input layer, and the output layer is fully connected to the hidden layer. If such a network has more than one hidden layer, we also call it a **deep artificial neural network**. We can add an arbitrary number of hidden layers to the MLP to create deeper network architectures. Practically, we can think of the number of layers and units in a neural network as additional hyperparameters that we want to optimize for a given problem task.

As shown in the preceding figure, we denote the *i*th activation unit in the *i*th layer as *a\_i^(l).*To make the math and code implementations a bit more intuitive, we will use the *in* superscript for the input layer, the *h* superscript for the hidden layer, and the *o* superscript for the output layer.

For instance, *a\_i^(in)*​ refers to the *i*th value in the input layer, *a\_i^(h)*​ refers to the *i*th unit in the hidden layer, and *a\_i^(out)*​ refers to the *i*th unit in the output layer. Here, the activation units *a\_0^(in)* and *a\_0^(out)*​ are the **bias units**, which we set equal to *1*. The activation of the units in the input layer is just its input plus the bias unit:



Each unit in layer *l* is connected to all units in layer *l + 1* via a weight coefficient. For example, the connection between the *k*th unit in layer *l* to the *j*th unit in layer *l + 1* will be written as *w\_{k, j}^(l)*​. Referring back to the previous figure, we denote the weight matrix that connects the input to the hidden layer as *W^(h)*​, and we write the matrix that connects the hidden layer to the output layer as *W^(out)*​.We summarize the weights that connect the input and hidden layers by a matrix:



There are two ways to build Keras models:

sequential and functional.

* The sequential API

• It allows you to create models layer-by-layer for most problems.

• It is limited in that it does not allow you to create models that share layers or does not help you much in defining complex models.

* Functional API – using the class Model – allows you to create models with a lot more flexibility

• As you can easily define models where layers connect to more than just the previous and next layers. • In fact, you can connect layers to (literally) any other layer.

• As a result, creating complex networks such as Siamese networks and residual networks become possible.

* AI: Deep Learning: Activation Functions

• Many activation functions can be used in the layers of a neural networks.

• In recent years, two activation functions – ReLU and Softmax, have become very popular thanks to their superior performance in comparison with the alternatives.

* ReLU (Rectified Linear Unit) function:

• The Rectified Linear Unit (ReLU) has become very popular in the last few years.

• It computes the function f(x)=max(0,x). • In other words, the activation is simply thresholded at zero (see image to the right).

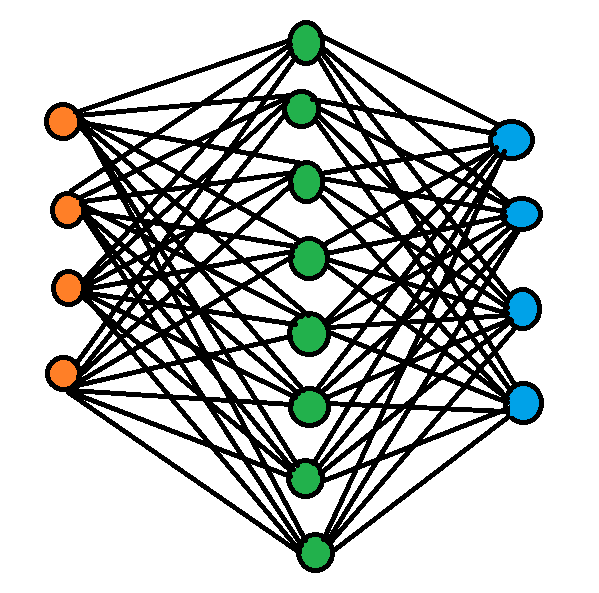
* Softmax function

The softmax function is often used in the final layer of a neural network-based classifier. • Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a nonlinear variant of multinomial logistic regression.

**Designing model with one hidden layer for Iris.csv dataset**

This is very good sign with this percentage of accuracy for prediction.

**MLPs (Fully Connected Neural Networks: 2 Layers) with Keras**



Input Hidden Output

Building DENSE model with Keras sequential API:

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(8, input\_dim=4, activation='relu‘)

model.add(Dense(3, activation='softmax'))

In the above example with IRIS dataset: 2-Layered MLP

• Two layers: The hidden layer + OUTPUT layer

* The input neurons: 4 neurons for 4 predictors, e.g., with IRIS dataset
* The 1st layer (the hidden layer): 8 neurons that receives data from the 4 input variables.
* The 2nd layer (the output layer): 3 neurons to predict the flower species (three species)

**Explaining the steps/results of the code**

Please refer to the code of Assignment 3 Part II while going through these lines.

* Dataset

This is an example of a notebook to demonstrate concepts of Data Science. In this example we will do some exploratory data analysis on the famous Iris dataset.

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). These measures were used to create a linear discriminant model to classify the species. The dataset is often used in data mining, classification and clustering examples and to test algorithms.

* Input Cell 1, 2

You can see that the python libraries we have imported which we will be using such as numpy, pandas, modules for data visualization, test/train, K Fold validation.

* Input Cell 3

Here we will be setting the value for the random seed and then we call the function.

* Input Cell 4

We will be loading the data set that contains the header row of attributes by specifying what and where the data file is by using the pandas dataframe.

* Input Cell 5

So, after loading the data we preprocess the data check if we have any missing values.

* Input Cell 6, 7

Now Exploring the Data by analyzing it. We do this by getting the dimensions by the code we got (150, 6) that is 150 rows and 6 columns.

Then we get the data types of the six columns.

* Input Cell 8, 9

Now we just look the sample dataset by calling the top with the python code.

Then the summary statistics of the numeric variables of the dataset by using the “df.describe()”.

* Input Cell 10

Now distribution of the species into respective that is there are 3 classes and 50 records for each class as a result basically we are grouping them.

* Input Cell 11

Here come the data visualization of the dataset, we plot histogram for each numeric variable/attribute of the dataset by using the python code “df.hist()”.

* Input Cell 12

Here comes the second part of the data visualization that is Density plots we did that by 5 numeric variables at least five plots. A layout of (3, 3) 3 rows and 3 plots.

* Input Cell 13

Now the third part with the help of scatter plot matrix.

* Input Cell 14

Now we separate the dataset into the input and output arrays in 1 to 4 in X and last 5 in Y.

* Input Cell 15

Now we split these arrays into training and testing datasets. We do this by first splitting the dataset 67% training sub dataset and 33% test sub dataset.

The selection of records to include in which sub dataset must be done randomly using the input cell 3 that is with seed 7.

* Input Cell 16

Here comes the encoding the class values into one hot coding for better performance as we have discussed in the above description.

So, the first step would be to encode the class values into the integers then we convert the integers to one hot coding format.

* Input Cell 17, 18

So once the it is encoded into the one hot coding which is suitable for the performance, we will be building an Multi-layer perceptron’s or a Fully Connected Neural Network Model.

Two layers: The hidden layer + OUTPUT layer

* The input neurons: 4 neurons for 4 predictors, e.g., with IRIS dataset
* The 1st layer (the hidden layer): 8 neurons that receives data from the 4 input variables.
* The 2nd layer (the output layer): 3 neurons to predict the flower species (three species).

First, we define a function to create a baseline model then the model with layers as discussed.

After creating a model we compile the model by optimization algorithm “adam” and loss function “categorical\_crossentropy”. Softmax is the iteration we need to use for the iteration.

Then in the next cell we will be calling this function so as to run this model.

* Input Cell 19

Once the model was built, we have to train the model. Using python code “model.fit()”. We need to provide the datasets and then the output that is one hot coding of the training, epochs, batch\_size.

So we have training dataset has 100 samples, we select the best that is of 10 batch size. We train the neural network one by one batch. We need to setup neural network many times almost 150 times.

Result, when we execute the system train and gives result. It will train 150 times. So,

Epoch 1/150 Loss: 1.2311 Accuracy: 0.3600

Epoch 2/150 Loss: 1.1945 Accuracy: 0.3600

Epoch 3/150 Loss: 1.1562 Accuracy: 0.3700

Epoch 4/150 Loss: 1.0816 Accuracy: 0.4200

.

.

.

.

Epoch 149/150 Loss: 0.2267 Accuracy: 0.9600

Epoch 150/150 Loss: 0.2237 Accuracy: 0.9600

Every time it will fit the better, by the end of the batches you will get the best we can say this by looking at the loss get decreased and the accuracy gets increased.

* Input Cell 20, 21, 22

So, once we train the model now time to evaluate the model so we do that by creating a Keras Classifier to evaluate the model.

Once the Classifier is created, we evaluate the model with K Fold cross validation.

In the end we evaluate the neural network using “cross\_val\_score()” by passing the parameter like evaluator, the dataset that we use to evaluate, the shuffle, the output of the dataset, then need to send the instance of kfold.

The function will return us the results now we can pin out the mean, standard deviation. Evaluate each neural network, it improves over and over in the end we can see the loss is low and the accuracy is more.

Baseline: 96.00% (8.00%) the average level the is mean of accuracy and the standard deviation.

* Input Cell 23

Now we evaluate the model with the python code “model.evaluate()” by using the default cross validation. We can also evaluate the performance of the neural network by using a method of sequential API. We can directly call by passing the parameters were X\_test and the onehot\_Y\_test.

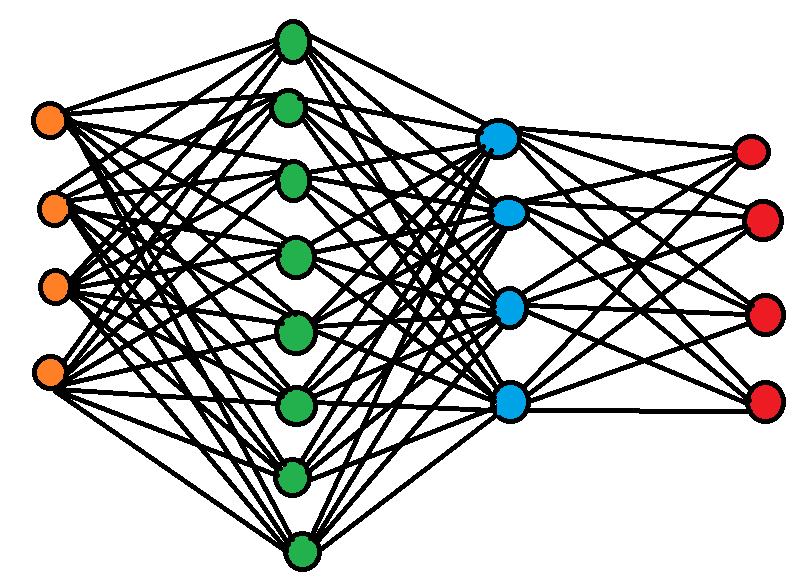
As a result, we get the accuracy 98.00%. This is very good sign with this percentage of accuracy for prediction.

**PART II: Redesign the MLP**

**Redesigning model with two hidden layers for Iris.csv dataset**

I have used Microsoft paint tool for this diagram.

**MLPs (Fully Connected Neural Networks: 3 Layers) with Keras**



Input Hidden Hidden Output

Layer 1 Layer 2

Examples of building DENSE model with Keras sequential API:

• Add another hidden layer into the MLP in the previous example

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(8, input\_dim=4, activation='relu‘)

model.add(Dense(4, activation='relu‘))

model.add(Dense(3, activation='softmax'))

In the above example with IRIS dataset: 3-Layered MLP

• Three layers: Two hidden layer + One OUTPUT layer

* The input neurons: 4 neurons for 4 predictors, e.g., with IRIS dataset
* The 1st layer (the hidden layer): 8 neurons that receives data from the 4 input variables.
* The 2nd layer (the hidden layer): 4 neurons and the activation function relu.
* The 3rd layer (the output layer): 3 neurons to predict the flower species (three species)

**Explaining the steps/results of the code**

Please refer to the code of Assignment 3 Part III while going through these lines.

Please refer to the code of Assignment 3 Part II while going through these lines.

* Dataset

This is an example of a notebook to demonstrate concepts of Data Science. In this example we will do some exploratory data analysis on the famous Iris dataset.

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). These measures were used to create a linear discriminant model to classify the species. The dataset is often used in data mining, classification and clustering examples and to test algorithms.

* Input Cell 1, 2

You can see that the python libraries we have imported which we will be using such as numpy, pandas, modules for data visualization, test/train, K Fold validation.

* Input Cell 3

Here we will be setting the value for the random seed and then we call the function.

* Input Cell 4

We will be loading the data set that contains the header row of attributes by specifying what and where the data file is by using the pandas dataframe.

* Input Cell 5

So, after loading the data we preprocess the data check if we have any missing values.

* Input Cell 6, 7

Now Exploring the Data by analyzing it. We do this by getting the dimensions by the code we got (150, 6) that is 150 rows and 6 columns.

Then we get the data types of the six columns.

* Input Cell 8, 9

Now we just look the sample dataset by calling the top with the python code.

Then the summary statistics of the numeric variables of the dataset by using the “df.describe()”.

* Input Cell 10

Now distribution of the species into respective that is there are 3 classes and 50 records for each class as a result basically we are grouping them.

* Input Cell 11

Here come the data visualization of the dataset, we plot histogram for each numeric variable/attribute of the dataset by using the python code “df.hist()”.

* Input Cell 12

Here comes the second part of the data visualization that is Density plots we did that by 5 numeric variables at least five plots. A layout of (3, 3) 3 rows and 3 plots.

* Input Cell 13

Now the third part with the help of scatter plot matrix.

* Input Cell 14

Now we separate the dataset into the input and output arrays in 1 to 4 in X and last 5 in Y.

* Input Cell 15

Now we split these arrays into training and testing datasets. We do this by first splitting the dataset 67% training sub dataset and 33% test sub dataset.

The selection of records to include in which sub dataset must be done randomly using the input cell 3 that is with seed 7.

* Input Cell 16

Here comes the encoding the class values into one hot coding for better performance as we have discussed in the above description.

So, the first step would be to encode the class values into the integers then we convert the integers to one hot coding format.

* Input Cell 19, 20

So once the it is encoded into the one hot coding which is suitable for the performance, we will be building an Multi-layer perceptron’s or a Fully Connected Neural Network Model.

Three layers: Two hidden layer + One OUTPUT layer

* The input neurons: 4 neurons for 4 predictors, e.g., with IRIS dataset
* The 1st layer (the hidden layer): 8 neurons that receives data from the 4 input variables.
* The 2nd layer (the hidden layer): 4 neurons and the activation function relu.
* The 3rd layer (the output layer): 3 neurons to predict the flower species (three species)

First, we define a function to create a baseline model then the model with layers as discussed.

After creating a model we compile the model by optimization algorithm “adam” and loss function “categorical\_crossentropy”. Softmax is the iteration we need to use for the iteration.

Then in the next cell we will be calling this function so as to run this model.

* Input Cell 21

Once the model was built, we have to train the model. Using python code “model.fit()”. We need to provide the datasets and then the output that is one hot coding of the training, epochs, batch\_size.

So we have training dataset has 100 samples, we select the best that is of 10 batch size. We train the neural network one by one batch. We need to setup neural network many times almost 150 times.

Result, when we execute the system train and gives result. It will train 150 times. So,

Epoch 1/150 Loss: 1.1559 Accuracy: 0.2600

Epoch 2/150 Loss: 1.1945 Accuracy: 0.2600

Epoch 3/150 Loss: 1.1562 Accuracy: 0.2900

Epoch 4/150 Loss: 1.0816 Accuracy: 0.3000

.

.

.

.

Epoch 149/150 Loss: 0.1660 Accuracy: 0.9400

Epoch 150/150 Loss: 0.1658 Accuracy: 0.9500

Every time it will fit the better, by the end of the batches you will get the best we can say this by looking at the loss get decreased and the accuracy gets increased.

* Input Cell 22, 23, 24

So, once we train the model now time to evaluate the model so we do that by creating a Keras Classifier to evaluate the model.

Once the Classifier is created, we evaluate the model with K Fold cross validation.

In the end we evaluate the neural network using “cross\_val\_score()” by passing the parameter like evaluator, the dataset that we use to evaluate, the shuffle, the output of the dataset, then need to send the instance of kfold.

The function will return us the results now we can pin out the mean, standard deviation. Evaluate each neural network, it improves over and over in the end we can see the loss is low and the accuracy is more.

Baseline: 96.00% (8.00%) the average level the is mean of accuracy and the standard deviation.

* Input Cell 25

Now we evaluate the model with the python code “model.evaluate()” by using the default cross validation. We can also evaluate the performance of the neural network by using a method of sequential API. We can directly call by passing the parameters were X\_test and the onehot\_Y\_test.

As a result, we get the accuracy 98.00%. This is very good sign with this percentage of accuracy for prediction.