**Report on Keras and Bokeh**

What is Keras?

Keras can be defined as a high-level neural network AI. It is implemented in Python. It is also compatible with TensorFlow, Theano etc. Basically, it is a wrapper on top of either Theano, Tensorflow, CNTK. It was developed mainly for fast experimentation.

There are 3 main reasons for using Keras:

1. It allows easy and fast prototyping. You don’t need to write all the boiler-plate code for creating and training the model. Just 3 or 4 lines will perform all the required actions. Fast prototyping means that you prepare and evaluate the data using keras first for demonstration purpose and then use tensorflow to prepare a similar model for actual project. Since keras may or not be supported in most of the environments, it makes sense to use tensorflow instead.
2. It supports convolutional networks and well as recurrent networks.
3. The most important point is that keras can utilize both the CPU and the GPU for computation.

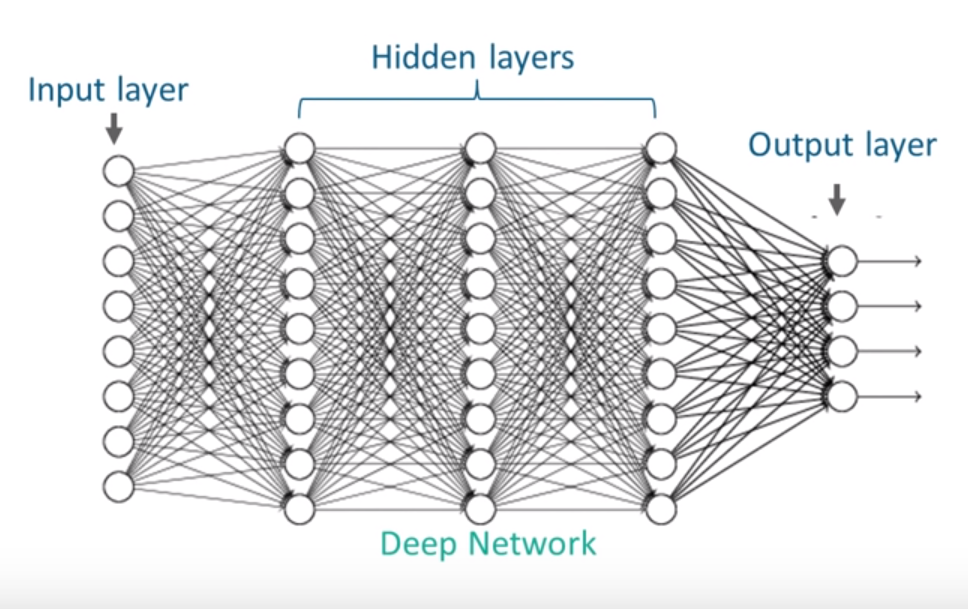
Types of Models

There are two types of models which can be set in keras:

1) Sequential composition.

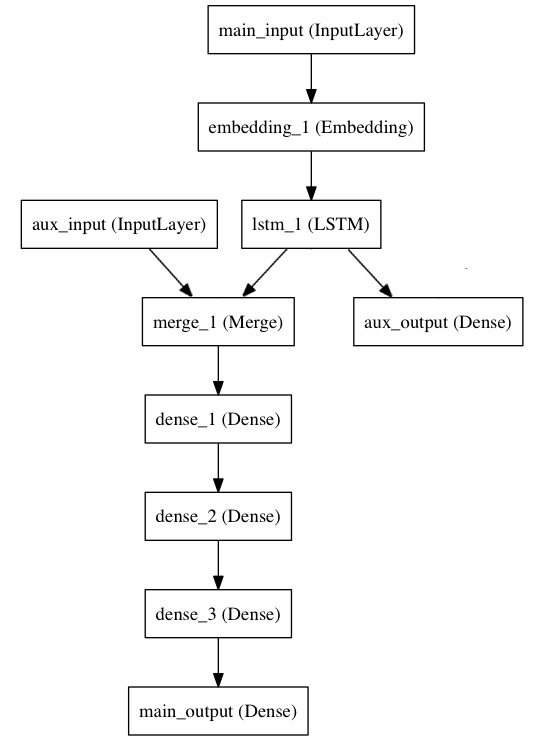
2) Functional Composition.

Sequential Composition:



Sequential models are nothing but a linear stack of layers and what it really means is that you take an input layer and you connect it to the hidden layer and when the weights which are coming from the input to the first hidden layer gets calculated and the node values are computed. After our activation function, this layer becomes input for the next layer and all we are doing is we are putting the next layer on top of the other and your model is calculated. The directionality in these models that you start from an input and you are making some predictions given the supervised learning nature of these models is called sequential composition.

Functional Composition:



Functional Model is another type of model in keras which is used for making more complex models. What I mean by complex is that I bring some inputs in between of the layers. In the case of sequential models, there are input layers, hidden layers and an output layer. The hidden layers are entirely dependent on the input layer and the output layer is also entirely dependent on the hidden layers. Each of the nodes in each of the layers gets its input from the previous layer. But in the case of Functional Composition, you can add an intervention between the layers. You can add a new information and pass that to the layers. So, if you want to create a model in such a way that it calculates some part of it from the input we have given and later on, you want to combine metadata information, then probably you can use functional APIs. Functional composition makes it possible to create complex models, such as acyclic graphs, or multi-output models. The figure above shows the working and behavior of the functional model. We can clearly see that the aux\_input is added or embedded in the next set of layers.

Building a model

Using keras, the pipeline for building a deep learning network looks like this:

1) Define Network.

2) Compile Network.

3) Fit Network.

4) Evaluate Network.

5) Make Predictions.

Churn-Prediction

Aim: The literal meaning of churn in this context is the propensity of customer to cease doing business with the bank. Churn prediction is a major concern of banks who wish to hold the clients. So, these banks will first predict whether a particular customer will churn or not. And then based on the result, it will take necessary steps to prevent the customer from leaving.

Description: The dataset has 13 columns and 10,000 rows. The columns had the all the customer information such as the Credit Score, Geography, Gender, Age, Tenure, Balance in Account, Number of Products, Credit Card Count & Estimated Salary which I have considered as the input columns. The other columns Customer Id & Surname is independent i.e. it does not have any importance, so I have omitted those 2 columns data. The last column is the ‘Exited’ column where 1 indicates that the customer has exited and 0 means not. So, based on the above information I will first check whether there is any relation between the other factors contributing to the churning and then perform prediction of the model using Keras.

Pre-process Data:

This is the first step for every machine learning project. Before we start building the model, we need to make our data suitable for processing. The input data may contain some categorical data which are represented in string format. The neural network models are compatible with only numbers. It cannot process string input. In my case, ‘Gender’ and ‘Geography’ column had string data. So, I had to transform those data into numerical format. There are different methods like Label Encoding, Dummy Encoding etc. I have used label encoding for doing the transformation.

lbl=LabelEncoder ()

bank\_data[:,col]=lbl.fit\_transform(bank\_data[:,col])

The above code will replace the column with the numerical data.

Now, the data must be divided into a set of training and testing data. The training data is used for training the model and the testing data is used for testing our model. Now, to divide our samples, we can either do it manually using computational logic or we can import a library train\_test\_split from sklearn.model\_selection.

The next step is to standardize/ normalise the data. This is highly recommended else the model will not give good results. For doing this, I have used the inbuilt class StandardScaler from sklearn.pre\_processing.

scaler=StandardScaler()

train\_data=scaler.fit\_transform(train\_data).

Creating an artificial neural network

For creating a model, we first create an object 'model' and then initialise that to Sequential Class.

model=Sequential()

Now, we need to specify all the hidden layers. I could have specified that in the constructor itself, but to make the code more readable, I have created it separately.

model.add(Dense(units=12,input\_dim=11,activation=”relu”)

The above line tells our model to add a dense hidden layer with 12 neurons or nodes. The model is expecting data with 11 columns. The activation function that I have used is the ‘relu’ activation function. We have only created one Dense(hidden) layer. Replicating the same line 'n' times will create 'n' hidden layers. If we don't specify the activation function, then keras will under the coverage use ‘linear activation function’.

For adding more hidden layers, copy below code

model.add(Dense(units=12 ,activation=”relu”).

Note that I haven’t specified the ‘input\_dim’ here because the output from the previous layer is passed as an input to this code. So, it will compute the input dim itself. Lastly, we are going to specify an output layer which is again going to be a dense layer. It's going to have 1 unit specified because the data on which I am working returns a 1-dimensional binary output. Here, I have used the sigmoid activation function.

You can then have a quick glimpse of your model using model.summary() which will display the number of hidden layers in your model and also specify the input and output dimension.

Training the model:

For compiling the model, I have used the Adam optimisation function. For this you may have to import it first.

from keras.optimizers import Adam

I have specified the learning rate as 0.001. The next argument is the loss. I will specify my loss as ‘binary\_crossentropy’ since my output is binary. If my output was categorical data, I would have specified my loss as ‘sparse\_categorical\_crossentropy’. There are different types of loss function such as mean\_squared\_error, mean\_absolute\_error, etc. The loss function is going to specify how the loss is calculated for your model during training.

Fitting the model:

For that, we need to pass the training data input and training data output as an argument. Next, we specify our batch size, and this is the number of samples we want our model to group together at a time during training. So rather than analysing each individual sample one by one, we want our model to analyse our samples in batches. And so, I have chosen a batch size of 10. Next is we have to specify epoch. A single epoch is a single run through the data. I have selected 200 here. Next, I have kept shuffle=true. Now, it means that for each epoch (each run) the input data (training data) is going to be in a different order.

I have also specified validation\_split=0.1. This means that it will split the last 10 percent of the training samples and perform the training separately. Generally, this is used for checking if our model is overfitting or not. Let’s assume if our accuracy reaches above 95 per cent but our validation accuracy is less, then that would mean that our model is overfitting. In my case, both the accuracies are almost the same. Note that previously I have kept shuffle=true. But this data, which is split, won’t get shuffled.

Making Predictions:

Now we will predict our values for the test data by using below code:

model.predict ( test\_data )

We can also use the evaluate function to get the actual score.

Eg: score=model.evaluate(test\_data,test\_op,verbose=1)

The above line will return the accuracy of the model. After multiple attempts, each time I got a decent prediction of around 85-90% accuracy. The validation accuracy was almost the same.

Plotting:

Before starting with the implementation, first we have to install bokeh. We can do this either typing pip install bokeh

OR conda install bokeh

The mean feature of bokeh is that the plots are visualised in the browser itself. And it provides great customizations like zooming in, zooming out, scrolling, etc.

The first step is to import the figure package from bokeh.plotting . Next, we will initialize the figure object. In doing so, we can specify the plot\_height and plot\_width. Eg:

p = figure(plot\_width=1000, plot\_height=600,title="Accuracy of the model for each epoch")

Now, suppose you want to plot a line, you specify the x and the y coordinate values, the line colour, line width and the labels. E.g.:

p.line(x,history.history['acc'],line\_color="red",line\_width=2,legend="Computed Accuracy")

You can also plot points by using the similar logic. E.g.:

p.circle(x,history.history['acc'], size=8, color="red", alpha=0.5)

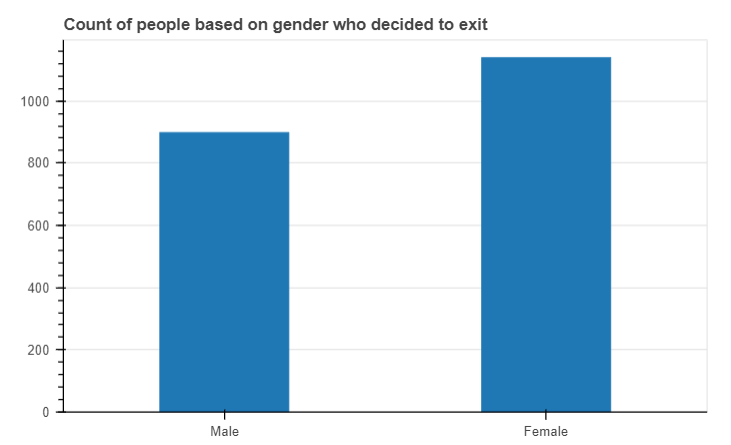
We can then specify all the other details like legend location, x axis labels, y axis labels etc.

For E.g.:

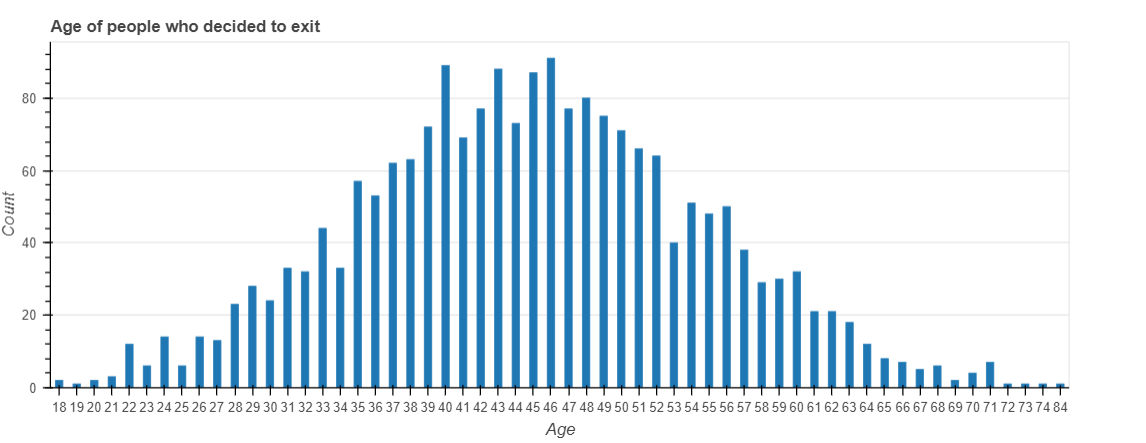
p.legend.location = "top\_left"  
p.legend.click\_policy="hide"  
p.xaxis.axis\_label='Iterations'

p.yaxis.axis\_label="Accuracy**"**

I have tried plotting different types of plotting like Bar chart, Scatter plot, Histogram, etc

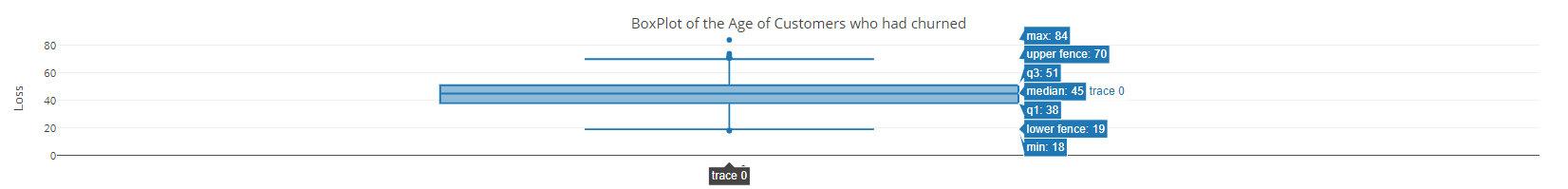


For this I have calculated the count of both the genders who have exited the bank and then added the values in a list. So, I have 2 lists, one containing the count and other containing the gender types (Male, Female)

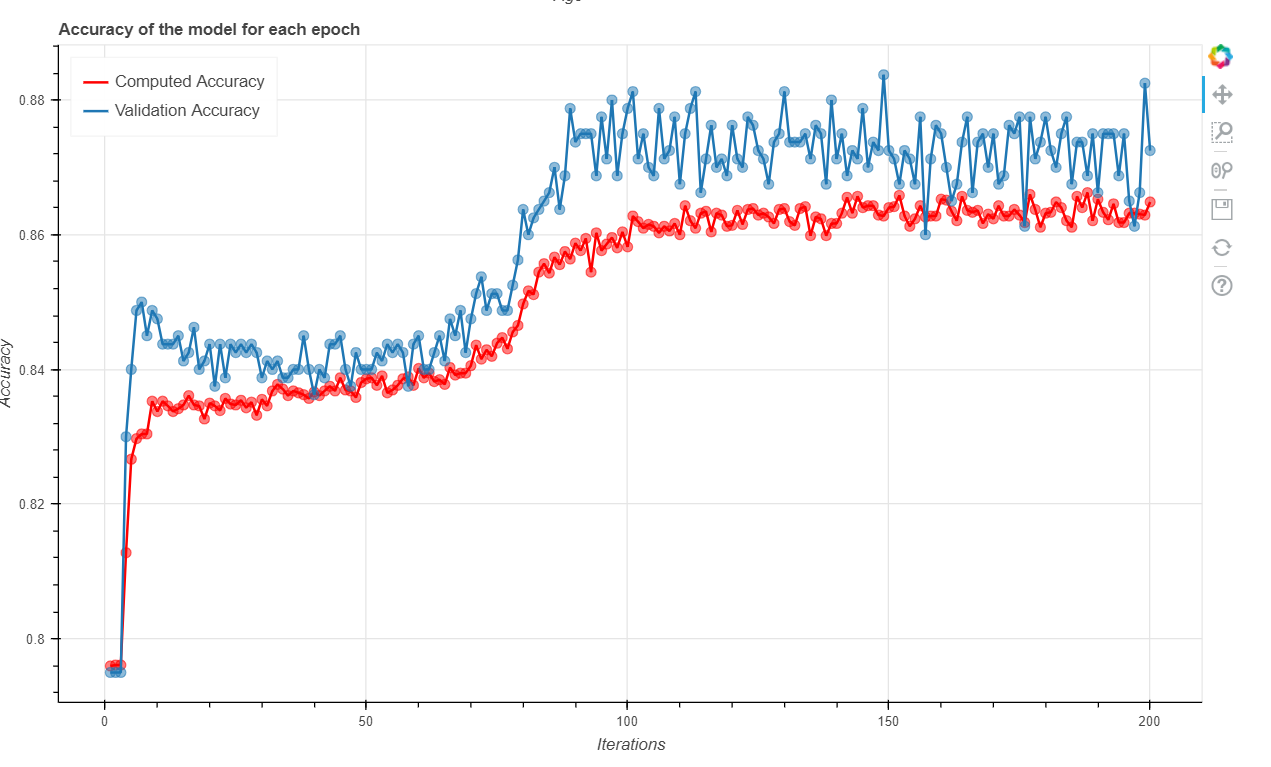


Here, I have computed the count of people of people who have churned(exited) grouped by the age group. In the dictionary object, I have stored the age group as the key and the value is the count. The figure shows that people with age-group 39-45 has the maximum probability of churning.

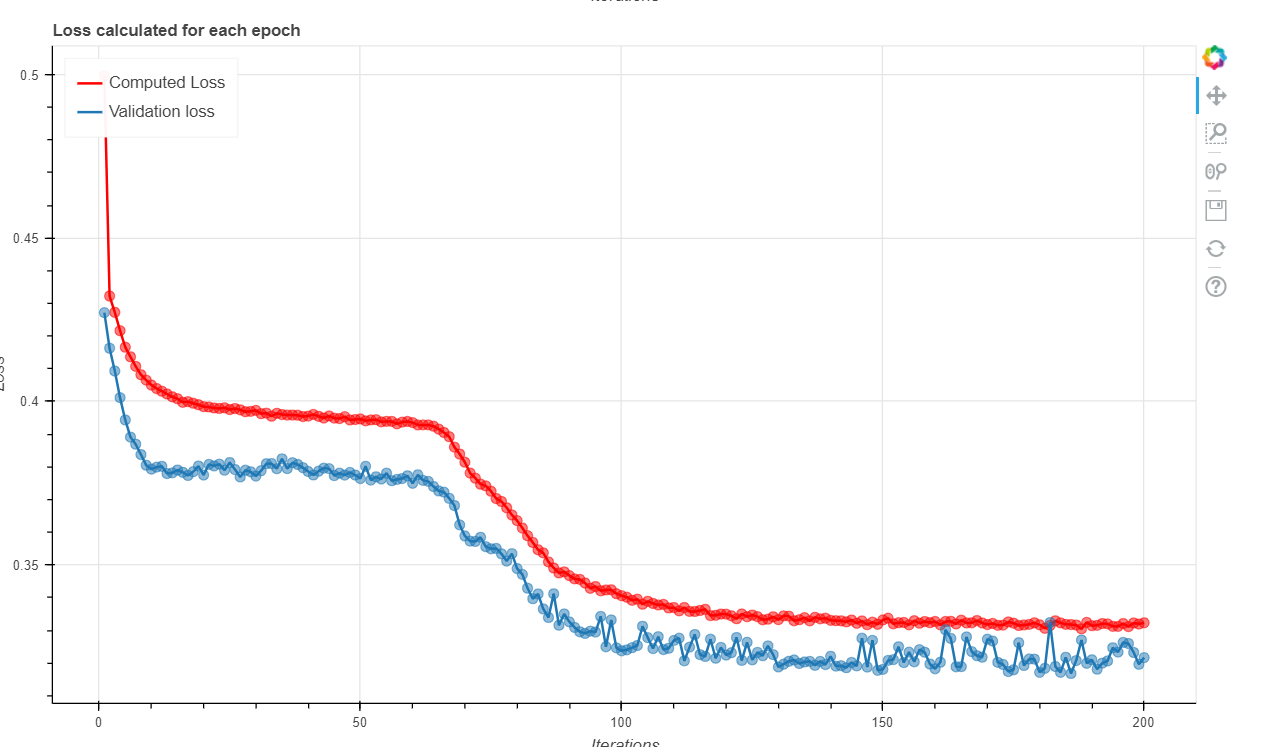
Below is the similar information using boxplot.



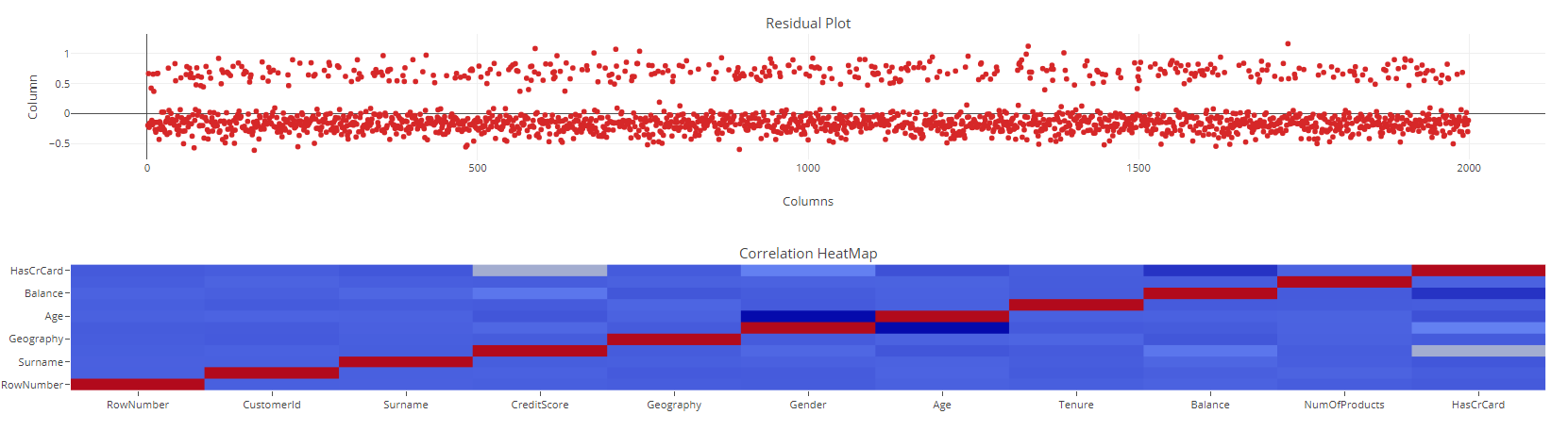
The plot below is the most important plot because it computes the accuracy. We can see the accuracy values are increasing as the iteration count increases. Also, the validation accuracies are also about the same. This means that our model is not overfitting.



The plot below shows the loss of the model. We can clearly see that the loss is drastically getting reduced as the iterations go up. Which means that our machine is learning at a better rate. Also, observe that the validation loss is also about the same. Therefore, our model is not generalising any input during any computations.



Both the plots below represent the residual plot and the correlation matrix. The difference between the actual values and the predicted values are called as residuals. It is important that the residual value should not very large or should not be zero. We can see that the model is predicted most of the values successfully. The final plot shows the correlation heatmap between all the columns of the dataset. The darker the line, the higher is the correlation.



Conclusion:

Keras is really a gem when it comes to deep learning. It totally abstracts the implementation layer making it easier for anyone to understand and implement or code the algorithm. Bokeh sure does have a lot of customizations when it comes to visualization. I believe that there are many methods which may have made the plotting more appealing or descriptive.