Neural Network Analysis

Overview

This purpose of this analysis was to explore to see if data from AlphabetSoup company could be used to predict which charity applicant for funding would be successful. The goal was to get at least 75% predicted correctly using deep learning neural networks. The process was to use the Scikit Learn library for data regularization. Then the data had to be compiled and fitted to the model for predictive analysis. The process is as followed.

Results – Data Preprocessing

- The target for the model would be the column that said the project was successful or not. We are evaluating that very thing so that column would be the target.
- For the features we wanted to use as much unique data as possible. It was decided to use the two columns with the largest amount of unique data those being Application_Type which had seventeen distinct types and Classification which had seventy-one different types. Then we decided to bin the different data types into their own number of classifications for the model.
- It was decided to drop the EIN and Name since those would make the values too unique. Those are unique items and all the 34,000+ number of those entries would be different making it much harder to train the model.

Results – Compiling, Training, and Evaluating the Model

- It was decided to see if many different variables layers and activation functions could be used for this test. The first test used three layers just to see if 75% could be achieved with minimal inputs. The first test also used neurons of sixteen for the first layer sixteen for the second and eight for the last. The subsequent test used many distinct levels, with the last being 5 layers with neuron levels of 128 for the first, 128 for the second, 32 for the third layer, 16 for the fourth, and 1 for the output layer.
- All my models did not achieve the desired result of 75% optimization for the model.
- The steps that I took to achieve the desired results were extensive. I used various levels of neurons. First 100 and 50 thinking that would be better than sixteen but later into the project I realized that neurons work best in multiples of sixteen up to 128 so that was the max that I used for neurons. I tried adding different layers of neurons thinking that would allow for more params to be tested and that decreased the accuracy of my model over time. I then tried different activations for the layers. I tried to use other activations during the output level but then researched and found that sigmoid or linear is the best activation to use for a regression problem which is what this is. For the hidden layers I wanted to use different activations, so I tried relu, sigmoid, tanh, gelu, elu, selu, softmax,

softplus, to see which one was the best for the model. I settled on leaky relu because leaky relu is like relu which was used in the original model but leaky relu does not cause vanishing gradients. Meaning that over the life of the model leaky relu's model weights do not decay. It is not needed with this model, but I did see the model performing better with it instead of relu. After I settled on the layers and the activation units, I still was not getting the desired result, so I want to use a different optimizer than Adam. Adam was used in the first model and since it did not achieve the desired results, I tried to use other activation functions. I tried using Adadelta, Adamax, NAdam and even tried to use Adabelief but I could not find a way to use it with Keras and I also tried MADGRAD which according to the latest data does outperform Adam on regression and classification problems. It is a new optimizer that has only came out a couple of months ago and was developed by Facebook AI. I tried to get it to work with Keras, but I could not get it working. I settled on Adam because with the model and the available neurons it performed the best. I also created another feature for income type that carried 9 different classifications. All of these customizations helped with the model and I saw that the model did reach 75% briefly but then went back to 74 and sometimes even 73% accuracy.

Summary

In conclusion, all of these model customization options didn't reach the level of optimization that was needed for the model. I think that the model can be tweaked to reach 75% accuracy and if there was more time I could add more layers, create new features, tweak the cutoff level for those features so that more parameters could be analyzed. I could find a way for the optimization models of Adabelief and MADGRAD to work to find out if it will be effective for the model. Overall, I do believe that 75% accuracy can be achieved with this model and for this problem. It may be switching to completely TensorFlow environment so that more optimizations and activations can be effectively used.