Neural Networks & Intro to Keras

1. Introduction

One of the subsets of machine learning is the topic of (Artificial) Neural Networks. The reason for the distinction is that these networks are *loosely* built to model the neurons in a biological brain. Neural networks are made up of an input layer and an output later with many different hidden layers in between. These layers in between can have different activations to show different characteristics in the data and within the model. These models are increasing the ability to work with data that has hundreds of inputs and have fairly high accuracy. Face recognition on iPhones is a machine learning algorithm, that, as we have learned, works well. Data that might have so much input data might be images or videos, self driving cars and video recognition. It may also be generating images and data or working with audio data. What is Keras? Keras is an easy to use front end tool for building neural networks using Tensorflow. Both these tools are built by google and are open source to modify and edit.

2. My experiments

2.1. Data preparation

I used data from the UCI Machine Learning repository. This repository offers many clean datasets ready for doing machine learning on. Usually this data has accurate values and has nothing missing. The specific data used in this experiment is the Student data which describes Portuguese students with data from a survey. Examples of different variables are: urban/rural address, romantic relationship, study time, family size, alcohol consumption, & sex. There are many other interesting variables in the data, take a look at the Jupiter code output to see a better description. The desired outcome variable is test grades. There were ~400 observations in the data. Eventually I took the data and split it into categorical, numerical, & outcome variable. I one-hot encoded (or dummyized) the categorical variables and z scored the numerical variables. This is the final dataset to use against the outcome test score variable.

2.2. Baseline model

The baseline model was a modified example from the text. It took in all of the variables as the input ~55 and had one dense final node for the regression prediction. The two hidden layers were 32 and 16 nodes with Relu activations. Between the two layers is a dropout layer which helps 'deactivate' random nodes during training to help with overfitting. The optimizer used the RMSPROP algorithm and used MSE as the loss to work against. The model had batch sizes of 20 and 20 epochs. This model had a testing mean squared error (MSE) of 15.24 and a mean absolute error (MAE) of 3.09

2.3. Layer Depth

Because neural networks are a growing tool in the world of machine learning, there are very few true and proven *rules*. Often a rule will be proven wrong when it has been said to be true. Compared to classical statistical methods, there isn't the same amount of theory that has been proven. Number of layers is an interesting topic because with too few layers (maybe 1) the network fails to convey the complexity of the data. A network with one layer is just a linear model. On the other hand, with too many layers, the gradient vanishes (see vanishing gradient problem). This happens because when the model is updating the weights in the model using back propagation, it uses a proportion of the partial derivative. Given a deep neural network, without any attempt to control this, the model will eventually not be able to increase the knowledge it can gain. Some weights closer to input layers don't get updated and stay at or close to their initial random value. But, in general, more layers do better. Here, I reduced the model to a single layer with dropout. The model in the data that was trained with the single layer got a testing MSE of 22.27 and a MAE of 3.94. This is a little worse than the baseline model with 2 hidden layers. This is likely because the single layer is close to a linear model. Multiple layers allow the network to more accurately portray non-linear data.

2.4. Neurons in a Layer

A heuristic way to define the number of nodes in a layer is a number between the number of input and the number of outputs. Maybe 100 inputs and 10 outputs, then

your layers should *heuristically* be between 100 and 10. In my model, using way more neurons than the number of inputs (55), my model did slightly better with a MSE or 15.17 and a MAE of 2.97. Maybe the extra layers helped better predict outliers which brought the mean squared error down, but didn't do as well on getting it close to the true value. For this reason, the model performed about equal on MAE, but did much better in MSE.

2.5. Loss Functions

Loss functions are used to help train the model. When doing gradient descent, or some other similar algorithm, the loss is what is being descent. When training on different loss functions it may optimize in a different way. MAE doesn't emphasize outliers as much as MSE. In my case I trained on MAE instead of MSE and got very similar errors with a MSE of 16.14 and a MAE of 3.07. With classification problems there are more loss functions to use and choose from.

2.6. Activation functions

I am not going to lie, activation functions are something I don't fully understand. But, I will do my best to explain them. There are many different activation functions such as Rely (rectified linear unit) or tanh, etc. What these all do is transform the number in the neuron. This can help models to work with non-linear data, just how doing log and exp transformations can be done in linear models. Also another fact to look at is the runtime of different functions. Relu is very simple and is the max(0,x), so all negative values are transformed into 0. This leads to a neural network that has some sort of additive parts. Relu is easy to compute. Tanh on the other hand may use more computations and take a longer time. However, tanh has negative values, which can be beneficial in a neural network. In my case, using tanh activation, the model got a MSE of 44.8 and a MAE of 5.6. Both much higher than the Relu activation. This is likely due to the bounding issues with z-scores. It might be better to use a MinMaxScaler to scale the values to between zero and one to avoid problems with tanh activation.

2.7. K-fold

The K-fold I used was out of the textbook, and is similar to the SciKit Learn K-Fold. In the future I'll use Sci-Kit's interpretation. Running this showed that the validation MAE lowers out at about 5-10 epochs. Using this hyper parameter, my model got a MAE of 3.9. Cross validation should be run more to verify results before trying it on testing data.

3.Conclusion

Though this is an intro to neural networks, it is promising to see the uses of them and the ability to work with massive sizes of input data. Looking back on the experiments there are many ways to optimize test and see what hyper parameters work best and what the general heuristics of neural networks are.

3.1.Future tasks

3.1.1. Run the model using MinMaxScaler and see how that affects the Tanh activation.

3.1.2. Use the traditional K-Fold function from Sci-Kit Learn.