19COC102 - Coursework Due on 16th March 2020, 12:00AM via Learn
Joe Graham
Student ID: B628327

Report

This report is intended to highlight the differences between different neural networks and the reasons for those differences.

Part A

Research + Design

Before tying to develop and train my own neural networks, I first researched the networks already available online in order to get an understanding of why some layers or methods were used and how efficient they were. Thus, allowing me to decide which tools to use. In order to use the jupyter notebook, firstly, I installed anaconda, then installed the jupyter notebook package from there. Once I had jupyter notebook up and running in a java environment, the next step was to choose which framework to use. Following extensive research, I decided to opt for the tensorflow framework, this I felt was more user friendly, and more importantly, the requirement to set up a tensorflow environment involved a broader spectrum of research on tensorflow verses pytorch in the initial stages of the project setup, therefore, my understanding of tensorflow was greater than that of pytorch. Following the setup, I installed all the necessary libraries and packages within the tensorflow environment, including pytorch, matplotlib and numpy. The initial phase for developing my model involved choosing which data to build a classifier for. For this project I opted for the mnist data set, primarily as it is a well-known data set and is easy to load the data. [1] In addition, it was a data set I was familiar with before commencing the project. Once the data set was selected, I researched the "best" classifier for the mnist data set, the results provided various models claiming to be the best, therefore, I opted to research thoroughly the one which was most readable. [3] with an accuracy of 0.998. It also detailed why the methods worked, where an explanation could be given, this helped me understand which methods worked best on this data set and why. With this in mind, and further research on other classifiers. I tried to develop a my own classifiers that would work well on the data set.

Implementation

Details of the tools and methods used during this project, and the reasons for why and how I used them are detailed below. After the research period, my thoughts were to create a simple neural network to see what effects the simple layers had. The initial part of code includes importing all the required libraries and initialising variables. I also followed the keras documentation and other documented models for loading and reshaping the data. The images are in the format of 28x28 pixels. To be able to input these images into the initial layer I first needed to convert it into a 2 dimensional space. The initial layers include two dense layers, using the selu activation method, with parameters 32 and 64 respectively. The dense layer is responsible for converting the input from its initial shape to a shape given by the parameter, for example, the first dense layer takes input as (*,784) to (*,32). as (*,784) is the input size. I then added a dropout of 0.25. This function randomly assigns 0 to inputs at a rate of 0.25. It is done for convergence purposes. The last layer, a dense layer using softmax as the activation protocol allows 10 outputs, corresponding to the 10 digits from 0-9. The loss function used was categorical cross entropy, this being the optimal function as the outputs are categorical, meaning they are placed in one of several categories, in this case, 10. Using the optimizer adadelta. I trained the data over 15 epochs, and used the test data as the validation data. I then decided to create two different convolutional neural networks to compare both the neural network to a convolutional NN and also to discuss the differences between the two CNNs. The first one was implemented with the intention of creating an efficient algorithm, for this reason, I decided to use a similar structure to the most efficient algorithms found during the research period. The difference between the first part of the code for the neural network and convolutional networks, are the different libraries needed, and also the data needs more reshaping, as before I loaded the data and initialised variables for later use. The difference being I needed to change the input into a 3 dimensional shape. The image must be stripped to grey scale, and normalised before use. For the sequential model, the first layer I used was a conv2D layer with paramater 32. This was done as the conv2D is considered the most efficient layer for image processing.[4] The parameter was selected based on the input size (28x28). The input was then initialised as a 3 dimensional shape (28,28,1) This was done as the layer needs a 4 dimensional input, yet it adds one itself, therefore, needing only a 3 dimensional input. Throughout the model I used the relu activation, as it is considered the most efficient for the mnist dataset [2]. Coupled with this layer is the maxpooling method, which was used with a pool size of (2,2). This is also considered the most effective for this scenario. I then repeated this coupled layer once more, and added a dropout of 0.25 after the second for a better convergence rate. The next layer consists of a flatten method, which is responsible for collapsing or reducing spatial dimension. Followed by a dense layer with parameters (128) which as discussed previously, changes the input to the given parameter followed by a dropout of 0.5, again for convergence sake. The final layer, as with the other model, is a dense layer with parameter 10, and activation softmax. This is to allow 10 outputs for the digits 0-9. I again used the categorical cross entropy loss function as it is the most effective given the type of output. I changed the optimiser to adam, as it is considered the most effective, only rivaled by the adadelta and selu optimisers. I trained this model over 15 epochs using the test data as the validation data. The second convolutional network, was implemented to show the difference in efficiency whilst changing the methods, optimisers, activation functions and others to less utilised ones on the mnist data set, I then explain the reasons for the differences. I kept a similar structure to the previous convolutional model, yet instead of using the conv2D layer, I used a SeperableConv2D layer, the main difference being seperableconv2D is intended to work faster, but can sometimes be less effective. I then used the Average pooling layer instead of the Max pooling, where it takes the average rather than the maximum of the given inputs. this pooling layer is less effective in this scenario as we want the part of the image with the greatest ”dark area”. I also used the sigmoid activation function throughout, which is ineffective in this scenario, as its main purpose is for probability related projects, as it takes output from 0-1, as with probability. The last two layers I decided to keep consistent with the neural network to compare the above methods more efficiently. When training the data, I used the mean squared error loss function and the sgd optimiser, both of which are not considered to be well known on this data set. Again, using the test data as the validation data over 15 epochs. I kept some parts of the models similar in order for a more effective comparison.

Part B

Results

When discussing the results, it is necessary to consider the loss and accuracy values for each algorithm. The neural network achieved a test loss of: 13.97 and test accuracy of: 0.2824 which is not great, yet was considerably higher than the second convolutional neural network, which achieved a test loss of: 0.090 and a test accuracy of: 0.1135.

In order to understand the difference, it is prudent to discuss the term loss and accuracy. Loss is the term given to how far off the model was at predicting the outcome, therefore, the larger the loss function the worse the prediction was.

Accuracy refers to how many correct predictions the model makes out of a total of 1, for normalisation reasons. Therefore, multiplying by 100, will give the percentage accuracy.

There have been numerous debates as to which is most representative of judging a model. In my opinion the 'loss' is important, as it can illustrate that your algorithm is clearly doing something not expected even if the accuracy is high.

The accuracy is simply an indication of how many it got right, although it may not work on unseen data.

For this reason, I consider the convolutional network to be greater than the neural network, simply as it has less of a loss.

Yet it is unwise to dismiss that the neural network was better at classifying the test data and with a few minor changes, the neural network has the potential to become more efficient than the convolutional algorithm.

The first convolutional neural network I created with the intention of making an efficient algorithm for classifying the data. It managed to produce a test loss of 0.038 and test accuracy of: 0.989, which, in my opinion worked extremely well on the test data and due to the loss has the potential to work well on unseen data also.

The main reason being the added dropout layers, as before adding them, the loss was considerably higher. Included in the jupyter network are the loss and accuracy graphs per epoch for each of the models discussed.

I opted to display these graphs to have an understanding of whether the model was over fitting or under fitting the data and to what extreme. As the graphs are in a separate part of the notebook, I trained the model again over 3 epochs for the purpose of the graph; this accelerated the process, yet caused the data to be slightly inaccurate. The graphs have the same layout throughout the different model. Using the pyplot from matplotlib, which i installed and included at the start of the code. The history object of the model just created needed to be stored in a local variable, namely, results. I had to store the object twice, as the validation data must be separate from the validation split. The next block of code is responsible for initialising variables for easier readability and easier repeated use. Once the initial steps in the code were complete. The next part was to plot the actual graphs. In order to do so, I followed the matplotlib documentation, manipulating the labels and other. The plot consists of writing the labels, title, the legend and where to place it and the x and y axis. This was repeated, for the loss and accuracy values respectively.

Although for the purpose of seeing how the model fits the data, it works. As the results illustrate, the simple neural network over fits the data, this explains the high accuracy and loss rate, as it overcompensates to predict correctly, this can be a serious problem when trying to implement the model on unseen data. The second convolutional network, also over fits the data, although not so much, and seems to follow the same curve as the data itself, unlike the neural network. The second convolutional neural network again over fits the data, although after each epoch, it seems to improve.

Table of results

This table contains the results of the different models

Table 1: Results table

Model	Loss	Accuracy
Simple neural network	13.97	0.2824
Convolutional neural network	0.090	0.1135
Convolutional neural network 2	0.038	0.989

Analysis

When analysing the results, the differences between neural networks and convolutional networks, and also the difference between the individual convolutional networks became apparent, albeit not conclusive. It is clear to see that the simple neural network classifier is the least reliable due to its high loss rate.

The main reason for this can be attributed to a number of decisions. The first being the layers themselves. The initial layer converts the shape from 784 to 32, to then be passed on to the following layer, which takes that input and converts it to 64. This in itself is not effective, as it is fist reduced to then be increased, where as when using dense, the output should normally be increased in size.

The dropout function may also be partly responsible, as the random nature of the layer can lead to loss in accuracy. The activation function selu, which stands for self-normalising can also be accredited with partial blame for the ineffectiveness of the algorithm, this is due to the function being known to not produce the best results when comparing it to its close relative, relu. A big factor in how well a given algorithm performs, is the circumstances under which it is compiled, that is to say, which optimiser and loss function is used, and how many epochs it is trained over. The optimiser used in this case was the adadelta, which is known to be a fairly good optimiser, although not used very often on the mnist dataset, where people tend to use the adam optimiser. The loss function used is also a common one, the categorical cross entropy. The results indicate that this one makes the most sense as the outputs are split into a finite number of categories. The model was trained over 15 epochs, which is enough to produce reliable results about the model.

Therefore, the way in which the model was compiled should not have hindered the model in a grievous manner, determining the layer choice to be the main reason for the inefficiency of this model when comparing this model with the second convolutional network it is clear that it performed with greater accuracy. Therefore, although it is the most inefficient, in my opinion, if the layers were altered slightly, I believe a simple neural network could perform more effectively than a convolutional network.

In fact, during the research period, I discovered a simple NN which had a similar accuracy to my most effective convolutional neural network, confirming my prediction that on average complex neural networks perform better than simple ones, yet this is not exclusively the case. [5]. Next, Comparisons between the two convolutional neural networks are made.

As the results show, the first convolutional model is the most effective for classifying mnist data, achieving close to perfect results, compared to a mere accuracy of 0.1. The sizeable difference between these two figures (88 per cent difference) is down to a number of reasons, due to the notable change in the layers, the activation function and the compiling method.

Following my research it is my opinion that the difference between the layers conv2D and seperableconv2D are not too great as to produce such a large contrast between the results. Therefore, although the difference may have had a slight effect on the accuracy, it could not have been detrimental. The first major difference is the pooling method used in the second model, the average pooling method was used in contrast to the max pooling method.

This has an effect on the results, for this project we are not interested in averaging the input, as the requirement is for the maximum figure to be spread. The fact that this coupled layer with its pooling method is repeated twice only exacerbates this difference 2-fold. Another difference is the absence of the dropout function in the second model. This could be a factor in the difference, yet it is hard to tell due to its random nature. A major downfall in the second model was the choice of activation function, sigmoid, which should not be used unless the output is in the region of 0-1, as that is its intended purpose. The use of this activation function in my opinion is the main reason for the large difference between the two models.

The last layer I kept consistent throughout both models in order to make a more reliable comparison, where the only significant difference between the layers of the two models is the activation function and the pooling layer. The next big difference between the two models can be seen when compiling the model. In the first case, I used the most renowned optimiser, adam, which is said to be the most efficient optimiser with little competition. Along with the loss function categorical cross entropy, which again, is the most renown for this data set, as the output is split into a finite number of categories. For the second algorithm, I opted to use the mean squared error loss function, as it is a well known function for its use not only as a loss function, but also in other areas of mathematics. Although in this case, it is not the most efficient. The optimiser used for the second model, was the sgd, stochastic gradient descent, which although not the most efficient, performs fairly well on most examples. Therefore, I believe the major reasons for the large difference in efficiency between the two models is due to the choice of activation function, the pooling layer used and less so, the loss function and optimiser chosen.

To conclude, this project identifies that in order to build a classifier it is crucial to not only identify but also fully understand the data at hand, this allows the developer to decide which functions to use based on the input and the output required. The well renown functions are the most effective as seen in the results, therefore, sticking to methods that are known to be efficient is the best option for producing an effective model.

When building the layers, try to keep it simple and understandable, using complex layers when necessary, without overusing them as to not understand what the model is doing. When compiling the model, use an appropriate loss function based on the output of the data, for instance, if the output is one of two objects, you should use a binary cross entropy loss function.

The optimiser should also be well known, adam for example is accepted as the most effective at the moment, therefore, should probably be employed.

As a final note this analysis determines that it is crucial when training the data, to ensure there are sufficient epochs to allow it to 'learn'.

References

[1] https://medium.com/data-science-bootcamp/famous-machine-learning-datasets-you-need-to-know-dd031bf74dd

[2] https://www.geeksforgeeks.org/activation-functions-neural-networks/

[3] https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d

[4] https://towardsdatascience.com/image-recognition-with-keras-convolutional-neural-networks-e2af10a10114

[5] https://medium.com/tebs-lab/how-to-classify-mnist-digits-with-different-neural-network-architectures- 39c75a0f03e