11/12/2021

MET CS 767 Assignment 2T Neural Nets

*Nathan A. Horak*

Copy the implementation [here](https://colab.research.google.com/drive/1TI76Pg5RYmlA0sTNthkqDhTjZWfIoZwn?usp=sharing) to your Google drive. Modify this code in four ways, attempting to improve the output, and report the results, using this Word file as a template. Since the accuracy of the given implementation is already high, consider reducing the size of the MNIST training set so that the baseline implementation leaves more percentage room for improvement. You are free to combine changes but each of the sections below should contain at least one new change. If necessary, show changes that make the result worse, with your explanation. Note the evaluation criteria as you respond.

Please leave the gray text and the headings unchanged and honor all plagiarism rules carefully.

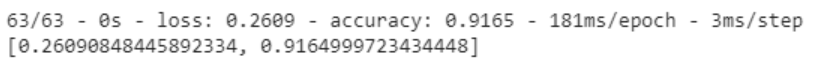
# First Code Modification

## 1.1 Description of what I did and reason(s) this could be an improvement (one paragraph)

After halving the training and testing set thrice (see COMMENTS), I began experimenting with the model parameters in order to improve (raise) the accuracy and lower the loss. Then I began reviewing various online literature as well as a little bit of experimenting and I determined that a good first step to improving the accuracy and loss would be introducing new hidden layers. According to d4datascience.in, increasing the number of hidden layers is a good way to improve the performance of a neural network algorithm[1]. This makes sense functionally- our original neural network algorithm is quite simple. After flattening the data by transforming it from a 28x28 two-dimensional array to a one-dimensional array of size 784 (28\*28), the algorithm utilizes ReLU activation with 128 densely connected neural units, drops off 20% of the data, and then processes the data through a final dense layer of 10 neural units sorting data corresponding to numerals 0-9. Adding additional hidden layers within the model creates additional instances within the execution of the algorithm on the training set that the neural network can correctly construct itself. I decided to add two additional hidden layers, both with ReLU activation instead of a differing activation like sigmoid (see Appendix A), following the initial data dropoff. I wanted each successive layer to have a smaller number of units so I went with 64 for the first hidden layer (half of 128) and 28 for the second hidden layer (a little more than half the second layer).

## 1.2 Comparison of the result with the original output

As stated above, my goal is to improve the accuracy and the loss of the model running on the test dataset. Accuracy and loss are two measurements that demonstrate the effectiveness of a neural net algorithm. Accuracy is the percentage of the test dataset whose output was correctly predicted while loss is a quantity that the model should seek to minimize during training- in this case, the negative log probability of the true class, according to the TensorFlow Core tutorials[3]. The original output gave a loss of 0.2609 and an accuracy of 0.9165.

 With the first code modification, the loss improved from 0.2609 to 0.2531 and the accuracy from 0.9165 to 0.9235.

This simple change improved loss and accuracy by 0.078 and 0.070 percentage points, respectively.

## 1.3 URL of your Colab code

<https://colab.research.google.com/drive/11Ug-B0_UdU_FsA_jgMQz1kWRUwPhX2b1#scrollTo=sh4G5rdVHrll>

# Second Code Modification

## 2.1 Description of what I did and reason(s) this could be an improvement (one paragraph)

I focused on two things for the second code modification. After adding additional hidden layers in the first modification, I tried a few things including lowering the dropout ratio (see Appendix B) and eventually I settled on increasing the number of neurons within the network as well as adding an additional LeakyReLU activation layer. Per d4datascience.in, increasing the number of neurons is a strategy a machine learning engineer can take in order to improve the effectiveness of a neural network algorithm[1]. According to the website, if an inadequate number of neurons are used, the resulting fit will be poor. However, one must also be mindful of using too many neurons as that can lead to excessive training times and/or overfitting if the additional neurons cause the algorithm to memorise the training data rather than learn from it[1]. Given the input array of size 784, I felt that the size of the first neuron was too low at 128 so I doubled it. I also increased the sizes of subsequent hidden layers but not the output layer (I wanted to keep that at size 10 for predicting numerals 0-9). In this step, I also decided to add in a ‘LeakyReLU’ layer. From the Keras API, a LeakyReLU layer is a leaky version of a Rectified Linear Unit (ReLU) activation method that allows for a small gradient when the unit is not active[5]. Per machinecurve.com, ReLU has a few problems of its own of which the main one is that since all outputs smaller than zero are zero, meaningful patterns of small data values can be lost[4]. LeakyReLU addresses this issue by introducing a small gradient for non-negative values.

## 2.2 Comparison of the result with the original output

As addressed in Section 1.2, the goal of the exercise is to improve the performance of the neural network algorithm by increasing accuracy and decreasing loss. After the first code modification, the resulting network gave an accuracy of 0.9235 and a loss of 0.2531.

Introducing the changes outlined in Section 2.1, the loss improved from 0.2531 to 0.2165 and the accuracy improved from 0.9235 to 0.9304.



These two alterations improved loss and accuracy by 0.366 and 0.069 percentage points, respectively.

## 2.3 URL of your Colab code

<https://colab.research.google.com/drive/11Ug-B0_UdU_FsA_jgMQz1kWRUwPhX2b1#scrollTo=FOlJ35DZO9VJ>

# Third Code Modification

## 3.1 Description of what I did and reason(s) this could be an improvement (one paragraph)

For the third code modification, I wanted to focus on the ‘dropout’ parameter. I considered other changes, such as an additional hidden layer (see Appendix C) but I had already done experimenting with adjusting this parameter (see Appendix B) but I felt like there was more to explore in this domain. According to the site towardsdatascience.com, dropout is used to randomly drop connections between neurons with the goal of forcing new connections between neurons as well as limiting overfitting[2]. In the original code, first code modification, and second code modification there was only one instance of randomly dropping data points (after the input). But there is no intuitive reason why this has to be the case. In fact, given the use of a dropout parameter to open up exploration between different nodes, it would make sense to use a dropout after each subsequent hidden layer to allow the algorithm new chances to learn, rather than memorize the data. I added three instances of randomized dropout- one after the ‘LeakyReLU’ layer, another after the hidden layer with 96 neurons, and another after the last hidden layer with 42 neurons.

## 3.2 Comparison of the result with the original output

Just as in Section 1.2 and 2.2, the aim is to improve upon accuracy and loss. After the second code modification, the network has an operating accuracy of 0.9304 and a loss of 0.216.

After the addition of the three instances of dropout following the ‘LeakyReLU’ layer as well as the two additional hidden layers, the loss stayed virtually the same from 0.2165 compared to 0.2163 while the accuracy improved from 0.9304 to 0.9344.



The additions of additional dropout instances improved loss and accuracy by a very marginal 0.002 and 0.040 percentage points, respectively.

## 3.3 URL of your Colab code

<https://colab.research.google.com/drive/11Ug-B0_UdU_FsA_jgMQz1kWRUwPhX2b1#scrollTo=DZVfmvqTCCdE>

# Fourth Code Modification

## 4.1 Description of what I did and reason(s) this could be an improvement (one paragraph)

In the fourth code modification, there were two changes I knew I wanted to address. First, in Section 3.1 I had established the merit behind additional dropout instances within the model after each subsequent layer. But I had not explored whether these dropout parameters should be the same as the initial dropout parameter. Secondly, I wanted to address the number of epochs. While 5 was a good starting number of epochs for the purposes of experimenting my code throughout the first three code modifications, I wanted to select a number that allowed the neural network algorithm to learn to an appropriate degree in an attempt to maximize accuracy and minimize loss. Per machinelearningmastery.com, an input layer usually uses a higher dropout rate of 0.8 (parameter=0.2) while hidden layers often use a smaller dropout rate between 0.5 and 0.8 (parameter between 0.5 and 0.2)[6]. It makes sense that a machine learning engineer would want the dropout parameter to be higher for hidden layers than for input layers. To prevent overfitting, it is important that enough data gets dropped off in the initial layer such that the model learns instead of memorizing the data[2]. However, connections built in hidden layers can be very beneficial to allowing a model to predict results accurately and although one should be wary of overfitting in these layers, it is important to not curtail the benefits of utilizing hidden layers with excessive dropouts. After some experimentation, I decided on leaving the initial layer dropout rate as is and changing the dropout parameter for the hidden layers to 0.30 from 0.20, effectively maintaining 33% more connections through each hidden layer. Finally, a straightforward way to improve the performance of a neural network is to increase the number of times that it runs on the training set. Per d4datascience.in, this is a proven way to increase accuracy and lower loss[1]. After some experimentation, I decided to go with 40 epochs (up from 5). While this leads to increased processing time, I was able to eek out much more improvement than in any other modification step giving the algorithm additional time to learn with the changed dropout parameters.

## 4.2 Comparison of the result with the original output

As in prior Comparison sections, the goal is to improve accuracy and loss. After the third code modification, the network has an operating accuracy of 0.9344 and a loss of 0.2163.



After adjusting the number of epochs and the dropout parameter values for dropout instances after hidden layers, the loss fell from 0.2163 to 0.2640 while the accuracy sharply improved from 0.9344 to 0.9528.



These modifications worsened loss and improved accuracy by 0.477 and 0.184 percentage points, each. Although unfortunately our loss worsened, accuracy did improve substantially. This result is not too unexpected as within the code, each model is being compiled based on the ‘accuracy’ metric and after 40 runs, benefits to loss were unfortunately lost.

## 4.3 URL of your Colab code

<https://colab.research.google.com/drive/11Ug-B0_UdU_FsA_jgMQz1kWRUwPhX2b1#scrollTo=ZnUwjige4Cg4>

For additional modifications, see Appendix D.

# Comments

I decided to reduce the sizes of the testing set and the training set. I did this by halving (per lecture) the respective sets three times from initial sizes of 60000 and 10000 to sizes of 7500 and 1250 for training and set, respectively. Given the variance in accuracy and loss of around a quarter to half percent when re-running the algorithm, I wanted the initial accuracy and loss to be worse such that changes leading to improvement could easily be seen. If the accuracy starts around 97-98% and I am expected to make four changes, it can be difficult to see the impact of each step given the variance in accuracy and loss described above.

# References

Show that you used a wide variety of resources by listing them below and clearly indicating in the body above where you used. Make sure to use proper referencing in your paper. We suggest using APA format, but other formats are fine if they clearly distinguish your work from work of others in your response. Each of your references should occur within the text; so for example [1] should occur below *and* within the body of your response at the relevant location. You can include specific sections of the notes and the textbooks.

[1] Ved. (2016, October 5). *How to improve performance of Neural Networks*. d4datascience.com. Retrieved November 13, 2021. <https://d4datascience.in/2016/09/29/fbf/>

[2] Gandhi, Rohith. (2018, May 17). *Improving the Performance of a Neural Network.* towardsdatascience.com. Retrieved November 13, 2021. <https://towardsdatascience.com/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d>

[3] *TensorFlow Tutorials*. tensorflow.org. Retrieved November 14, 2021. <https://www.tensorflow.org/tutorials/>

[4] Chris. (2019, September 4). *ReLU, Sigmoid and Tanh: today’s most used activation functions*. Retrieved November 14, 2021. <https://www.machinecurve.com/index.php/2019/09/04/relu-sigmoid-and-tanh-todays-most-used-activation-functions/>

[5] *Keras API Reference*. keras.io. Retrieved November 14, 2021. <https://keras.io/api/>

[6] Brownlee, Jason. (2018, December 3). *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. machinelearningmastery.com. Retrieved November 15, 2021. <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>

# Appendix

[A] I briefly experimented with changing the activation method of the hidden layers I added in Part 1. I specifically focused on using another popular activation method, the sigmoid method, instead of ReLU. From machinecurve.com, an activation method takes a linear neuron input and generates a non-linear output- sigmoid and ReLU (along with Tangens hyperbolicus {tanh}) are a few of the most popular activation methods[4]. Per the aforementioned website, a major difference among these is that ReLU is not sensitive to vanishing gradients (where ReLU/tanh are), which can slow learning down in your network[4]. However, the sigmoid activation function has some benefits of its own. According to the above source, sigmoid function is quite good at estimating a probability as probabilities have a range between 0 (never occurring) and 1 (100% chance of occurring) which fit the nature of the logistic curve. Specifically, when compared to a step-function, it provides some nuance concerning probabilities moderately less than or greater than 0.5. However, as machinecurve.com notes, this benefit is most pronounced within binary classification problems but a neural network built on the mnist dataset is not a binary classification problem, given that there are 10 outputs (as opposed to 2). When the algorithm is run with both hidden layers using the sigmoid activation function instead of ReLU, the results seem to be improved to a smaller degree with an accuracy of 0.9208 and loss of 0.2456 using the sigmoid method in the two hidden layers compared to an accuracy and loss of 0.9235 and 0.2531 using the ReLU method.

*Loss and Accuracy using sigmoid activation method on hidden layers:*

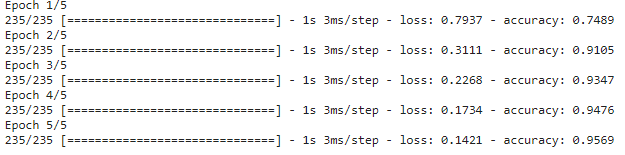


*Loss and Accuracy using ReLU activation method on hidden layers:*



[B] A change I considered was raising the dropout parameter from 0.20 to 0.30. From towardsdatascience.com, dropout is utilized to randomly drop connections between neurons with the aim of forcing the network to find new paths and generalize[2]. It is used to account for overfitting. Per the above listed source, overfitting occurs when your model starts to memorise values from the training value instead of learning from them and can be noticed if the training accuracy and loss are much better than the testing accuracy and loss. Given that the training accuracy is only a few percentage points higher than the test accuracy, it seems that overfitting is not too big of an issue. However, if upon modification the training accuracy/loss improve while the testing accuracy/loss get worse or don’t meaningfully improve, it seems as though we are creating additional overfitting at the expense of our model’s performance. In this case, it would be better to maintain the dropout parameter at 0.20.

*Loss and Accuracy from model fit for five epochs over training data (0.20 Dropout):*

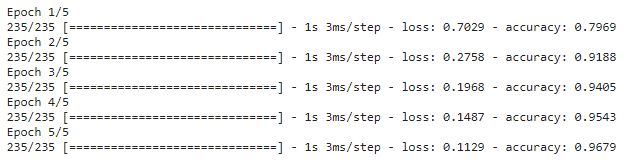


*Loss and Accuracy from testing data (0.20 Dropout):*



As the accuracy is only a few points improved for training compared to testing, it seems as though overfitting is not a concern. Perhaps our model could be improved by lessening the dropout ratio such that more connections between neurons are sustained as the model learns. Reducing the dropout parameter to 0.30 from 0.20 gives the following results:

*Loss and Accuracy from model fit for five epochs over training data (0.30 Dropout):*

*Loss and Accuracy from testing data (0.30 Dropout):*



While the training accuracy is more than a full percentage point better for the model with the lower dropout ratio, the accuracy is more than half a percentage point lower. The loss is improved by a few percentage points, but as stated above, these changes give credence to the idea that lowering the dropout ratio introduces more overfitting into the model and negatively impacts its performance. Thus, this change does not help to improve the model.

[C] Another change I considered for the third code modification was the addition of another hidden layer. As addressed in Section 1.1, the introduction of hidden layers can improve the accuracy/loss of a neural network algorithm (per d4datascience.in)[1]. I had already added one ‘LeakyReLU’ layer and two subsequent HeLU layers (with decreasing neural size) so I figured that I would try to add an additional layer. Although it seems unlikely that adding additional hidden layers indefinitely would lead to steadily increasing accuracy (while at the same time increasing processing time) I figured that I would examine the effects of an additional layer after the ‘LeakyReLU’ layer.

*Loss and Accuracy from model after Second Code Modification:*



*Loss and Accuracy from model with additional hidden layer of size 128 after ‘LeakyReLU’ layer:*



After the addition of another hidden layer of size 128 following the ‘LeakyReLU’ layer, the results don’t seem to be an improvement (and in fact, are slightly worse). Above, I theorized that adding additional hidden layers just for the sake of adding them was not going to keep improving the results and the loss/accuracy outputs give credence to the validity of this theory. Furthermore, there is a clear downside to this approach, namely increase in computation times. As such, a machine learning engineer should weigh the benefits of improved performance against the very real cost of increased processing time for each additional hidden layer.

[D] As I progressed through the assignment, a particular suggestion for improving performance of neural network algorithms kept sticking out. Per d4datascience.in, a proven way to improve the accuracy and loss of a neural network algorithm is to introduce more data[1] and according to towardsdatascience.com, a lack of training data can be a downfall to an underperforming neural network[2]. Given this, I wanted to apply the code I had written as of the fourth code modification to the original data. As noted in the comments, the original algorithm when operated gave an accuracy of generally between 0.97 and 0.98. I wanted to see if I could improve on this percentage and I succeeded in doing so:



0.9830 is indeed higher than anything in the 0.9701-0.9799 range and I was pleased with this result. From d4datascience.in, a large amount of training data is a bulwark against overfitting[1]. While adjusting the dropout parameter for the fourth code modification, I tried some higher dropout parameter values (equivalent to keeping more of the original data). According to machinelearningmastery.com, a good value for dropout rate in a hidden layer is between 0.50 and 0.80[6]. I tried out multiple values but I obtained worse results with 0.50 or 0.40 as a dropout parameter for hidden layer dropouts compared to 0.30.

*Accuracy and Loss of entire dataset using Fourth Code Modification code and dropout parameter at 0.50:*



*Accuracy and Loss of entire dataset using Fourth Code Modification code and dropout parameter at 0.40:*



While the accuracy value for hidden layer dropout parameter is less than 0.98 when the parameter is set to 0.50, it is greater than 0.98 when the parameter is set to 0.40. Both of these values are less than the accuracy result obtained from using the dropout rate parameter of 0.30. This leads me to believe that while the increase in training size was enough to increase the accuracy by multiple percentage points, it was not enough to change the nature of how the data interacts with the algorithm- changing the optimal dropout parameter after hidden layers.

# Evaluation

