MLP Coursework 1

s2115429

Abstract

Over-fitting can occur in nearly any deep learning model due to the complexity of the model. This happens from many ways such as having strict constraints on hyper-parameters and unclean data. We will discuss a few potential solutions for this issue, such as dropout, L1 regularization, and L2 regularization. These techniques have been proven to improve results amongst many deep neural models.

1. Introduction

Over-fitting is the concept of the model over-training on the training data and providing high accuracy for this while providing very low accuracy for validation data. This becomes an issue since we want the training and testing accuracy to be similar in numbers to show the model can handle predictions on new data. In this paper, the capability and effectiveness of regularization techniques such as dropout, L1 and L2 regularization will be tested and analyzed to see if this improves the over-fitting problem within deep neural networks.

Our experiments will be using the Balanced EMNIST data which will contain 28x28 handwritten characters used for classification (Cohen et al., 2017). The training set is a size of 100,000 characters. The validation size is 15,800 characters. The testing set is a size of 15,800 characters as well. Total data set is a size of 131,600 characters.

The following research questions will be examined and analyzed within this paper:

- What parameters can we tweak before applying any dropout or weight penalties?
- Do these regularization techniques increase the accuracy while fixing our over-fitting problem?

2. Problem identification

The issue of over-fitting can clearly be seen when visualizing training and testing accuracy and error rates. The goal for any model is to report low error rates and high accuracy for training and testing sets. Over-fitting occurs when there is over-training within the network leading to the function to produce data outputs fit too closely to the training set. As the epochs increase for the error rates, we see the training error becomes very low while the testing error increases. In an ideal model, we want to see both error rates decrease as we approach the number of epochs used. Over-fitting can be seen as well in the accuracy plots. As the epochs increase for the accuracy, both the training and testing accuracy increase but there is a big gap separating them. This large gap is an indicator over-fitting is occurring since we want the training and testing accuracy to be high and in a close range.

This problem occurs when hyper-parameters, such as the learning rate, number of hidden units, and number of hidden layers, are too high or too low for the model to function properly. The size of parameters such as these matters because the values do not have to be extreme to make a negative impact on the model. If one hyper-parameter is off, it can throw off the entire network. When these hyper-parameters are not carefully chosen, it greatly affects the model's ability to make accurate predictions.

We can help reduce this issue of over-fitting by adjusting the individual hyper-parameters and applying the regularization methods. First, we will need to determine the learning algorithm and learning rate used for these experiments. A stochastic gradient descent model is a requirement for this assignment so we will use the default Adam. The learning rate is determined by trial and error which includes running various rates against the hidden unit options. The learning rate experiment was conducted by testing a range of potential rates of 0.0005, 0.0001, 0.001, 0.002, and 0.005 with 32, 64, and 128 hidden units. The final choices were between 0.001 and 0.0001 due to providing the best accuracy rates out of all the options.

Even though 0.0001 has the highest accuracy for training and validation, 0.001 was chosen as the learning rate. This value shows great improvement of accuracy amongst all hidden units as well as still shows the issue of over-fitting. We did not choose learning rate of 0.0001 because it was not obvious there was an over-fitting issue within the model. This can be seen in the below Figures 1-6 showing the training and accuracy decreasing as the learning rate gets smaller. Therefore, for the rest of the experiments, we will use a learning rate of 0.001.

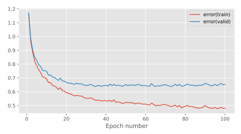

Figure 1. This plot shows the training and validation error using a learning rate = 0.001 and hidden units = 32.

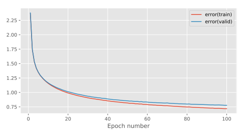

Figure 2. This plot shows the training and validation error using a learning rate = 0.0001 and hidden units = 32.

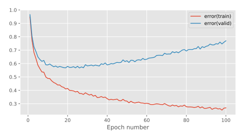

Figure 3. This plot shows the training and validation error using a learning rate = 0.001 and hidden units = 64.

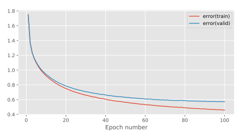

Figure 4. This plot shows the training and validation error using a learning rate = 0.0001 and hidden units = 64.

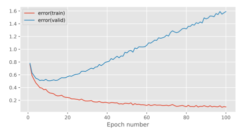

Figure 5. This plot shows the training and validation error using a learning rate = 0.001 and hidden units = 128.

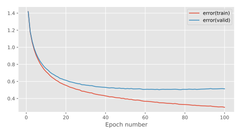

Figure 6. This plot shows the training and validation error using a learning rate = 0.0001 and hidden units = 128.

After deciding the learning rate, we will test if the number of hidden units affects the model's accuracy and error rate. Usually, the number of hidden units is chosen based off a few factors (Panchal et al., 2011). A few of the reasons for choosing the best number of hidden units is chosen from the number of input and output units, number of training cases, and the complexity of the problem we are trying to solve. To test this, one hidden layer was used to verify if changing the number of hidden units affects the model's accuracy and over-fitting issue. In our case, 32, 64, and 128 hidden units will fit these guidelines. The number of hidden units is passed into each Affine Layer function. The results are shown in Figure 1, Figure 3, and Figure 5 where we use the chosen learning rate of 0.001 and increase the number of hidden units. The plots show a drastic increase in validation error while the training error rate remained very low. This also proves that if we increase the number of hidden units, over-fitting is very recognizable.

We will also test the various number of hidden layers of 1, 2, and 3. Testing the number of hidden layers with our chosen learning rate will allow us to determine if increasing the hidden layers improves accuracy and decreases over-fitting. Normally, we expect to see an improvement as we increase the number of layers since more parameters become available which leads to an increase in accuracy as well (Trenn, 2008). However, adding two hidden layers increases the accuracy to 82.3%, only a 0.6% difference between having one hidden layer and two hidden layers. Increasing the hidden layers to three gives our model an accuracy of 82.4%, only a 0.2% increase from having two hidden layers. Although our model shows a slight improvement with increasing the number of hidden layers, this was not initially expected. Since there was not a dramatic increase from increasing the number of hidden layers, this leads us to believe that increasing the number of hidden layers will not always improve the accuracy. This shows we also need additional techniques to increase the accuracy such as dropout and weight penalties. In next section , we will discuss these regularization techniques using our baseline model of 3 hidden layers and 128 units.

Hidden Layers	Training Error	Validation Error
1	9.63E-02	1.59E+00
2	1.05E-01	1.70E+00
3	1.23E-01	1.54E+00

Table 1. This table reflects the number of hidden layers and the error rates associated with the training and validation set.

3. Dropout and Weight Penalty

As explained in Section 2, increasing the number of hidden units and increasing the number of hidden layers does not always yield significant improvements to our over-fitting problem in deep neural networks. Further adjustments can be made to optimize our model's predictions.

This is when dropout becomes useful for reducing over-fitting data within a deep neural model. Dropout is a technique used to directly help over-fitting as well as efficiently combine various network architectures (Srivastava et al., 2014). This is performed during training time by randomly dropping a number of nodes within a layer. Dropping these nodes thins the neural network which allows predictions to be made on the nodes not dropped during testing time. This process is beneficial because it is not as computationally expensive and shows significant results with reducing over-fitting in many deep neural networks.

Dropout helps with mitigating the problem by dropping a number of nodes during training time. This is done by randomly dropping a node in a hidden layer by comparing it to a probability variable (Ba & Frey, 2013). This inclusive probability variable, usually between 0.5-0.8, determines whether the node is dropped or kept in the network. This is also known as the mask, which is then multiplied with the input from the previous layers. This provides the output for a feed-forward propagation.

The following equation used in this assignment shows the relationship between the mask and the inputs for each layer as well as how this is applied to the rest of the deep neural network (Srivastava et al., 2014):

$$
\begin{aligned}
m^l &\sim UniformDistribution(p), \\
\widetilde{y}^l &= m^l * y^l, \\
z_i^{(l+1)} &= w_i^{l+1}\widetilde{y} + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)})
\end{aligned}
$$

In Srivastava's implementation, a Bernoulli distribution is used in the equation. For this assignment we will use Uniform distribution.

For each layer, l, a uniform distribution is used to randomly choose binary values to be the mask, m. The mask is then multiplied by each layer output. This is used to draw a comparison between this value and the chose probability value. Once the node is decided to be kept or dropped, it then follows a normal feed-forward network calculations.

Weight penalty is another technique that can be applied to help reduce over-fitting a model. This regularization method penalizes the larger weights in a network. Weight penalties are applied to the weights in a deep neural network where a penalty value is calculated as well as the penalty gradient value with respect to the input. This will add a term to the error function. This can be seen in the equation below for calculating the L1 penalty value and adding the term to the error function :

$$E_w = E_{l1} = \sum \|w_i\| \tag{1}$$

$$E^n = E^n_{train} + \beta E_w \tag{2}$$

where E^n is the equation for network error function, E^n_{train} is the data term, and β is the regularization coefficient. The equation for calculating the L1 penalty gradient with respect to the parameter can be seen as:

$$\frac{\beta \partial E_{L1}}{\partial w_i} = \beta sgn(w_i) \tag{3}$$

where sgn is the sign of w_i which is then used to calculate the gradient:

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n_{train}}{\partial w_i} + \frac{\beta \partial E_{L1}}{\partial w_i} \tag{4}$$

The other weight penalty method we will analyze is L2 regularization. L2 regularization, also known as weight decay, pushes the training data in a specific direction. If the training data does not respond to this push in weights, then it will decay to 0. This method allows the data to choose which parameter to reduce effectively.

The L2 penalty value can be seen in the equation below:

$$E_w = E_{l2} = 0.5 * \sum_i w_i^2 \tag{5}$$

The equation for calculating the L2 penalty gradient with respect to the parameter can be seen as:

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n_{train}}{\partial w_i} + \frac{\beta \partial E_{L2}}{\partial w_i} = \frac{\partial E^n_{train}}{\partial w_i} + \beta * (w_i) \tag{6}$$

Although L1 and L2 regularization both penalize larger weights, the rate of at which the weight values shrink is noticeably different. L1 tends to shrink to 0 at a constant rate while L2 shrinks to 0 at a rate proportional to the size of the weight. L1 is it encourages sparsity to the point it will shrink some weights to 0 which can leave out important connections

In fact, the combination of dropout and the addition of weight penalties can show significant improvement in over-fitting and accuracy.

4. Balanced EMNIST Experiments

In this section, we will analyze the results from the experiments performed with dropout alone, dropout combined with weight penalties, and weight penalties without dropout. Our goal for this assignment is to minimize over-fitting to receive higher accuracy for our deep neural network. Examining the above cases will give us an idea as to which technique gives the best results with this goal in mind. As mentioned in (Srivastava et al., 2014), the most improvement was seen with the combination of dropout and weight penalties. This will be something to keep in mind when analyzing our results.

For the following experiments on all regularization techniques, we set the number of hidden units to 128 and the number of hidden layers to 3 as well as set the learning rate to 0.001. Each network will be trained over 100 epochs with batch size of 100. These should be consistent with all the tests so we can make accurate and effective comparisons between the various changes with the hyper-parameters.

The first experiment examines the dropout method by itself. We give the dropout layer method a parameter of p to represent the probability we will compare the nodes to in order to decide which nodes to keep or toss in the network. In this case, the only hyper-parameter that will be examined is the changing probability value and how it makes a difference with affecting the model's accuracy. After deciding on the inclusive dropout probability value, the dropout method is applied to every layer, including input, hidden, and output layers.

Dropout Layer

Dropout Value	Training Accuracy	Validation Accuracy
0.5	7.52e-01	7.42e-01
0.6	8.05e-01	7.97e-01
0.7	8.42e-01	8.28e-01
0.8	8.71e-01	8.47e-01

Table 2. This table shows a relationship as we increase the dropout value to 0.8, there is an increase in the accuracy. It is important to note this does not mean if we continue to increase the dropout value, we will continue to see this pattern.

As can be seen above in Table 2, we see as we increased the dropout value to 0.8, we received a training accuracy of 87.1% and validation accuracy of 84.7%. This means if we have a dropout value of 0.8, we intend to keep 80% of the nodes in each layer and disregard 20% of the rest of the nodes. It is important to note that continuing to increase the dropout value will not guarantee better results. If we continued to increase the dropout value, then we would be closer to the original network architecture we had before resulting with over-fitting. The same can be said about lowering the dropout value to an unacceptable small value. Continuing to lower the hyper-parameter would result in dropping many of the important nodes we need to make accurate predictions. In conclusion for this particular experiment, a reasonable, larger dropout value increases accuracy and reduces over-fitting.

We can also adjust L1 and L2 regularization hyper-parameters without dropout to verify if any results of reducing over-fitting is seen. For L1 regularization, the hyper-parameter which will be adjusted is the L1 coefficient. The coefficients were chosen based on the lab specifications giving a typical range of 0.1-0.00001. Coefficients used were the minimum, maximum and values in the middle of the specified range.

L1 Weight Penalties

L1 Coefficient	Training Accuracy	Validation Accuracy
0.1	2.17E-02	1.96E-02
0.001	7.68E-01	7.63E-01
0.0001	8.75E-01	8.49E-01
0.00001	9.41E-01	8.26E-01

Table 3. This table shows the relationship between L1 coefficient values to the accuracy of training and validation data sets.

The results in Table 3 show as we decrease the L1 coefficients, the validation accuracy increases to a certain extent. A L1 coefficient of 0.0001 achieved the highest validation accuracy of 84.9%. In the table we can also see that if we decrease the L1 coefficient even smaller than the coefficient with the highest accuracy, the validation accuracy begins to decrease again. This can be due to the fact that if we reduce the coefficient severely, the weights will be decreased severely as well. As the weights become too low, we can see in our model that it becomes over-generalized and even exaggerates the over-fitting problem more with a training accuracy of 94.1% and a lower validation accuracy of 82.6%. We can see this behavior with too high of a coefficient. If the coefficient is too high, the training and validation accuracy decreases dramatically to about 2.17% and 0.196%. These low values lead us to believe a high coefficient is too restrictive on a model and can cause under-fitting as well.

Similarly, we see the same trend in results after examining the various L2 coefficients. The highest accuracy for using L2 regularization with the model is 84.9% with a coefficient of 0.001.

Table 4 shows the highest validation accuracy achieved is 84.9%. If we decrease the L2 coefficient to a value much smaller than the coefficient with the highest accuracy, the validation accuracy begins to decrease again as the training accuracy increases. This is the same reasoning due to the fact that if we reduce the coefficient severely, the weights will be decreased severely as well. As the weights become too low, we can see in our model that it becomes too lenient and exaggerates the over-fitting problem more with a training accuracy of 94.5% and a lower validation

L2 Weight Penalties

L2 Coefficient	Training Accuracy	Validation Accuracy
0.1	2.17E-02	1.98E-02
0.001	8.72-01	8.49E-01
0.0001	9.45E-01	8.40E-01
0.00001	9.45E-01	8.29E-01

Table 4. This table shows the relationship between L2 coefficient values to the accuracy of training and validation data sets.

accuracy of 82.6%. If the coefficient is too high, the training and validation accuracy decreases dramatically to about 2.17% and 0.198% and under-fitting occurs.

Combining the dropout method with weight penalties shows promising results. Our expectations is to see better accuracy results with the combination than the previous experiments. For the test cases, the experiment was conducted by using dropout inclusive probability values ranging from 0.5-0.8 as well as testing L1 and L2 coefficients of 0.1 - 0.00001.

Weight Penalties with Dropout

L1 Coeff.	DV 0.5	DV 0.6	DV 0.7	DV 0.8
0.1	2.2E-01	1.98E-02	2.01E-01	1.98E-02
0.001	2.09E-01	2.01E-02	2.01E-01	1.96E-02
0.0001	2.01E-01	7.27E-01	8.01E-01	8.27E-01
0.00001	7.40E-01	7.97E-01	8.30E-01	8.45E-01

Table 5. This table shows the validation accuracies for different dropout values (DV) with L1 coefficients.

From examining the above table, we see that the best outcome is a dropout value of 0.8, coefficient of 0.00001 and an accuracy of 84.5%. The trend we see as we decrease the L1 coefficient value, we see the validation accuracies rise. As for L2 with dropout, we see a similar pattern with an increase in values.

Weight Penalties with Dropout

L2 Coeff.	DV 0.5	DV 0.6	DV 0.7	DV 0.8
0.1	1.98-02	1.96E-02	1.96E-02	2.01E-02
0.001	6.50E-01	7.32E-01	7.84E-01	8.18E-01
0.0001	7.30E-01	7.87E-0	8.28E-01	8.51E-01
0.00001	7.48E-01	7.93E-01	8.28-01	8.48E-01

Table 6. This table shows the validation accuracies for different dropout values (DV) with L2 coefficients.

As can be seen from Table 5 and Table 6, the best outcome on the validation set is using L2 coefficient of 0.0001 with a dropout value of 0.8 which gives a validation accuracy of 85.1%. For both experiments of L1 with dropout and

L2 with dropout, we see a pattern. As we increase the dropout values as well as decrease the coefficients it will achieve higher accuracies. The higher the coefficients, the lower the accuracies. When comparing these experiments to the previous experiments, we can confirm using dropout with L2 regularization achieves the highest accuracy out of all the experiments. This matches the results found in the results of Srivastava's experiment (Srivastava et al., 2014).

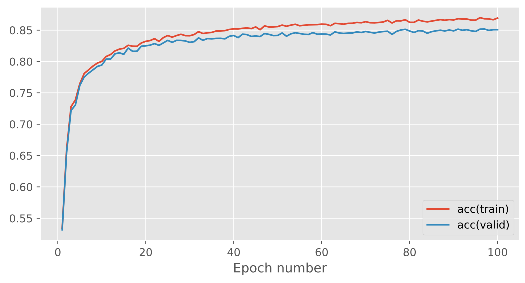

Figure 7. The best output for training and validation accuracy with dropout 0.8 and L2 coefficient of 0.0001.

Therefore, after examining the data from the experiments, the highest accuracy achieved is a 85.1% with a combination of L2 and Dropout for the validation set. After gathering the results, we can conclude the best model to run on the test set would be dropout with L2 regularization. The hyper-parameters we will use will be a dropout value of 0.8 and L2 coefficient of 0.0001. After running this model on the testing set, we achieved an accuracy of 83.57%.

5. Literature Review: Dropout: A Simple Way to Prevent Neural Networks from Overfitting

In the literature, the author introduces the concept of incorporating dropout within deep neural networks to reduce the common occurrence of data over-fitting. While the paper thoroughly explains the theory and application of dropout in relation with over-fitting, it is also mentioned this method helps to efficiently combine different neural network architectures (Srivastava et al., 2014).

The paper provided a simple yet informative formula for calculating the dropout and applying it with a feed-forward network. After reviewing how to apply the dropout network, the paper reviews the process of conducting and analyzing their experiment.

Srivastava and the team of writers approached this paper with the intent to easily explain how to apply dropout amongst various data sets for object classification, digit recognition, speech recognition, document classification, and bioinformatics (Srivastava et al., 2014). This wide use of data is crucial when studying a new technique since we want to test this technique's capability amongst different fields of study. Along with this, the author provides great detail of insight as the background information of the data. This can be seen in section 6.1.4. The section included the details such as data set and feature size. This is important to include to give readers of all backgrounds the ability to understand what the data is, where it is used, and how it can be used. Many papers assume this is already known details but for this case, there were no assumptions made about what the reader should already know.

This paper can be furthered strengthened, than it already is, by adding more analysis on the dropout training time issue. It is briefly mentioned in the conclusion that the training time for dropout increases by 2-3 times more than a normal neural network (Srivastava et al., 2014). While a decision was made to prefer accuracy over training time, since there is a trade off between the two, it would validate the paper more while including more details on the downfalls of this approach. While this study is supposed to introduce a new and exciting concept to helping a common over-fitting issue, it would not hurt the paper's overall intent to include a section to address the negatives.

While the authors' discussed the application of dropout amongst the many data sets analyzed in this experiment, the text data set in Section 6.3 appears to need further investigation with the results. It is briefly mentioned dropout still improved the error rate, but only by a few percentages. This result is expressed as not expected. A recommendation is to provide an explanation as to possible theories to clarify and potentially provide new experiments that can be done to justify these results.

6. Conclusions

This paper provided a discussion and analysis with techniques to improve the common over-fitting issue with deep neural networks. Based upon the results from the experiments, the best outcome for the neural network was using dropout with L2 regularization applied to every layer. The learning rate of 0.001, dropout of 0.8, and L2 coefficient of 0.0001 helped achieve the highest results.

Performing a few experiments inspired further experiments to be ran. Initially, the first experiments were comparing L1 regularization with dropout against L2 regularization with dropout. After performing these tests I realize these would not be effective comparisons if we did not test the individual dropout and weight penalties. After reading the literature, the conclusion was the best result was a combination of weight penalties and dropout but with experiments we need to come to our own conclusions and experiment with the necessary possibilities.

Further experiments can be performed to potentially achieve higher accuracy values with dropout and weight penalties. As mentioned in the paper, some results proved to show high accuracy when applied to either input layer or only hidden layers. This would be an interesting experiment to run to see if we can improve the accuracy to be greater than the testing accuracy of 83.57%.

References

Ba, Jimmy and Frey, Brendan. Adaptive dropout for training deep neural networks. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 3084–3092. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5032-adaptive-dropout-for-training-deep-neural-networks.pdf.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters, 2017.

Panchal, Gaurang, Ganatra, Amit, Kosta, YP, and Panchal, Devyani. Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers. *International Journal of Computer Theory and Engineering*, 3 (2):332–337, 2011.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

Trenn, Stephan. Multilayer perceptrons: Approximation order and necessary number of hidden units. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 19:836–44, 06 2008. doi: 10.1109/TNN.2007.912306.