MLP Coursework 1

s2043919

Abstract

Overfitting is a typical problem in machine learning. Many methods have been developed to solve this problem. This paper focuses on dropout and weight penalty, both of which prove to be effective in mitigating overfitting. By exploring the classification of images of handwritten digits in EMNIST dataset using neural network, this paper first identifies and discusses overfitting which exists during training, then explores technical details of dropout and weight penalty by implementing dropout and weight penalty. What's more, this paper performs several experiments using dropout and weight penalty in baseline network and discovers how to combine dropout and weight penalty to find the optimal model that generalizes best. Finally, this paper reviews the original essay of dropout and discusses it briefly.

Keywords: overfitting, machine learning, neural network, dropout, weight penalty

1. Introduction

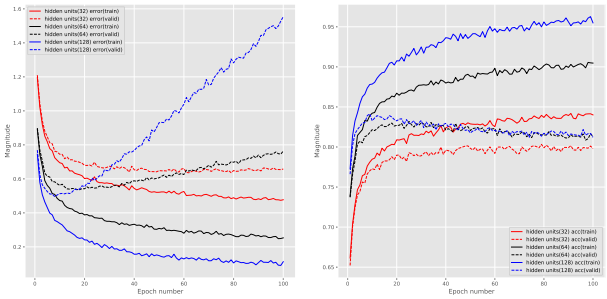
The motivation of the work is to modify the given baseline network to solve overfitting and find the optimal model in classication of images in EMNIST dataset. The research questions that this paper explores include whether varying number of hidden units and varying number of hidden layers mitigate or worsen overfitting and how dropout and overfitting work in detail and how to combine them to find the optimal model. In terms of EMNIST dataset, it extends MNIST by including images of handwritten letters(both upper and lower case) and handwritten digits. The dimention of each image is 28*28 and there are 62 potential classes in EMNIST, however, we only need 47 different classes because there are 15 letters which are difficult to distinguish between upper-case and lower-case. The dataset is divided up into training set, validation set and test set. The input size of training set is 100000. For validation and test set, the input size is 15800.

2. Problem identification

As we can see from the two graphs, after training 100 times, the validation error is far larger than training error, which is a typical overfitting problem in machine learning. Overfitting is usually caused by too many training parameters or too complex training models or small datasets. A typical feature of overfitting is that the model fits almost perfectly on training sets, however, terribly on validation sets and test sets. There are many ways to reduce overfitting, for example, cross-validation, using larger dataset, dropout, regularization, early stopping and data augmentation. In order to find whether hidden units size and hidden layers are related to mitigating overfitting, we can perform two experiments. The first one is changing the number of hidden units and another one is changing the number of layers. From the two experiments, we can understand how varying hidden unit sizes and number of hidden layers affect the results.

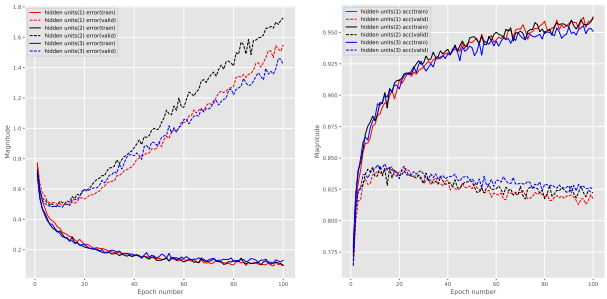
In the first experiment, we change the number of hidden units to compare the results. The motivation to do this experiment is to discover whether increasing number of hidden units mitigates or worsens overfitting problem. What we expect to discover is that increasing hidden units worsens overfitting because overfitting is usually caused by too complex models. Here are the procedures of this experiment. Firstly, we need to do some hyperparameter setting, which includes initializing the number of hidden layer to be 1 and training epochs to be 100. Then we train the new model three times by changing the hidden units size to be 32, 64 and 128 subsequently. Finally, we plot the graphs of error curve each time we change the hidden units size. The figure 1 shows the error and accuracy curves when hidden units size equals 32, 64, 128 subsequently. It's obvious that increasing hidden units size worsens overfitting problem. The reason is that when we increase hidden units size, we are also using more parameters and making the training model more complex.

Then we come to the second experiment, which is studying how varying number of hidden layers affects the results. The motivation to do this experiment is to find whether varying number of hidden layers worsens or mitigates overfitting. Here are the details of this experiment. Firstly, we set hidden units per layer to be 128 and epochs to be 100. Then again we train the model three times by setting number of hidden layers to be 1, 2, 3 subsequently. Finally we plot the graphs of error curve each time we change the number of hidden layer. The figure 2 shows the error and accuracy curves when number of hidden layers equals 1, 2, 3 subsequently. We can see from the graphs that overfitting still exits. So increasing number of hidden layers does not mitigate overfitting, but we are not sure whether it could worsen overfitting as the gap between validation error and training error remains basically the same across the three graphs. The reason is probably that the number of hidden layers is not large enough.



(a) Error curve (b) Accuracy curve

Figure 1. Varying number of hidden units



(a) Error curve (b) Accuracy curve

Figure 2. Varying number of hidden layers

3. Dropout and Weight Penalty

Dropout and weight penalty are two effective ways to mitigate overfitting. Dropout reduces overfitting by deleting hidden units randomly according to dropout probability. Normally we divide the batches into many mini-batches and each time we train a mini-batch, dropout layers delete part of hidden units based on drop probability. So we can consider dropout as training and combining many different networks with different ways of overfitting together. Dropout reduces overfitting efficiency in very complex or large-scale neural network. The key reason why dropout works is that it makes features more robust and less dependent on other features which might be missing when training because each of the hidden units must learn to cooperate with other randomly chosen units, which enables hidden units in each layer create their own features without relying on other units too much. Another reason is that dropout can also be explained as a method of regularization by adding noise to its hidden units and also a way of averaging different models. Technically, each dropout layer deletes units by setting their value to be 0. For the missing units, we need to compensate for them to guarantee performance. There are two ways to do this, the first one is called inverted dropout, which scales hidden units by 1/(1-p)(p is drop probability) when training. Another one is to scale the hidden units by 1-p when testing. Details of implementation of how the two ways work during training, presented as pseudo code, are shown in Algorithm 1 and 2.

For weight penalty, it's totally different from dropout. There

Algorithm 1 Inverted dropout

Input: data y_n, drop probability p

$mask = matrix[Uniform(0, 1, y_n.shape) <= 1 - p].$

$y_n new = mask * y_n/(1 - p)$

$z_{n+1} = w_{n+1} * y_n new + b_{n+1}$

$y_{n+1} = activation(z_{n+1})$

Algorithm 2 Normal dropout

Input: data y_n, drop probability p

$mask = matrix[Uniform(0, 1, y_n.shape) <= 1 - p].$

$y_n new = mask * y_n$

$z_{n+1} = w_{n+1} * y_n new + b_{n+1}$

$y_{n+1} = activation(z_{n+1})$

are two ways of weight penalty, which is L1 penalty and L2 penalty. L1 penalty reduces overfitting by penalizing the sum of absolute value of weights. L2 penalty works by penalizing the sum of square value of weights. Both L1 and L2 regularization can reduce overfitting, however, L2 is particularly useful in complex models while L1 is only feasible in simple models. The reason why weight penalty works is that it penalizes weights more if the model gets more complex. The formulas are as follow.

L1 Penalty:

$$E^n_{L1} = \sum_{i=1}^{k} |w_i|$$

$$E^n = E^n_{train} + \alpha * E^n_{L1}$$

L2 Penalty:

$$E^n_{L2} = \frac{1}{2} * \sum_{i=1}^{k} w_i^2$$

$$E^n = E^n_{train} + \alpha * E^n_{L2}$$

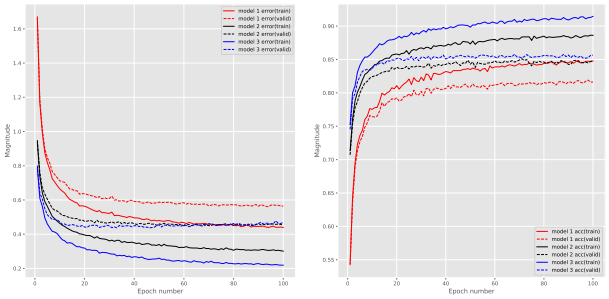
To compare dropout with weight penalty, the key difference is that dropout reduces overfitting by changing the model structure while for weight penalty, is by changing the cost function.

4. Balanced EMNIST Experiments

In order to mitigate overfitting, we need to modify the baseline network using dropout and weight penalty. The key problem is how to use dropout or weight penalty or combine both of them to find the optimal model. Before we do the experiment, we need to understand what is the optimal model. The optimal model is the one which shows the best ability of generalization. In some cases, if we penalize the model too much, the model might generalize terribly even though it prevents overfitting. Therefore, good performance on test set is our goal, which means that we need to mitigate overfitting and make the model generalize well at the same time. In order to know how models generalize, we use cross-validation to observe the generalization of models in validation sets when training.

Before we do the experiment, some basic settings need to be clarified. The neural network in this experiment has three hidden layers and 128 hidden units in each layer with Relu as the activation function. The learning rule is AdamLearningRule and learning rate is 0.1. Finally, the number of training epochs is 100.

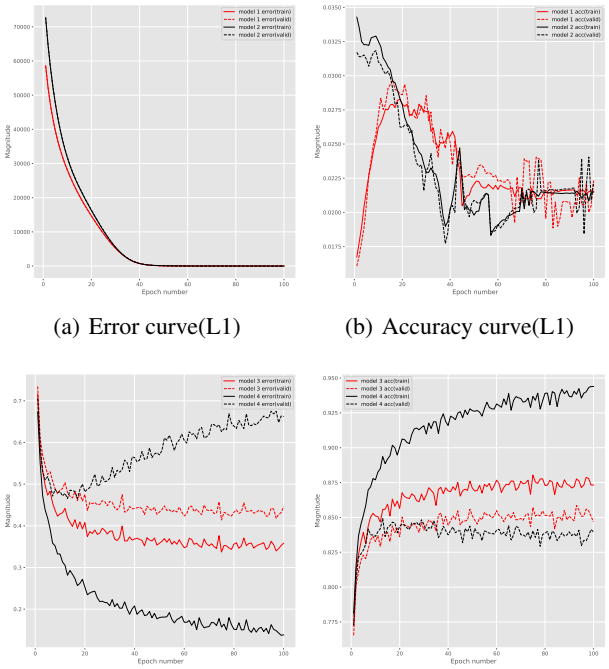
In the first experiment, we only add dropout layers to see the performance. We select the model which is proposed in the original essay (Srivastava et al., 2014), in which all dropout nets use retention probability p = 0.5 for hidden units and p = 0.8 for input units. We name this model as model1. The error and accuracy graph of model1 is shown in figure 3. We can see that training error and validation error matches well, however, the accuracy is not high. The reason is that we drop too many hidden units, which affects generalization. Therefore, we need to increase the inclusion probability and drop less hidden units. So we set the inclusion probability to be 0.8 in all hidden layers and input layer and name this model as model2. Although the gap between training error and validation error gets slightly larger, the accuracy gets improved. We could move a step further by setting probability to be 0.9. The results in figure 3 show that overfitting exists again, however, the accuracy is good on validation set. To conclude from the graphs above, if we only consider how the model solves overfitting, model1 is definitely the best, however, in terms of generalization, it's not the model we want. In terms of the other two models, although it doesn't reduce overfitting as much as the model1 does, it generalizes better. Therefore, we should select the latter two models, however, it does not mean one of these two model can be selected as the optimal model, we still need to make some efforts in mitigating overfitting that still exists in them. That's why we need to turn to weight penalty.



(a) Error curve (b) Accuracy curve

Figure 3. Add dropout only

In the next experiment, we only add L1 and L2 penalty in affine layers. Firstly, let's come to L1 penalty. Initially, we set the penalty value to be 0.001 in all affine layers. Again, we name the model as model1 in this context. As is shown in figure 4 (a) and (b), it is terrible in generalization. Then we set penalty value to be 0.0001 and name this model model2, again it generalizes terribly. Therefore, we can conclude that L1 is not feasible in this neural network model. Then we come to L2 penalty. Initially, we set the penalty value to be 0.001(name the model as model3), the graphs are shown in (c) and (d) in 4. We can see that the model perfectly prevents overfitting but generalizes badly with low validation accuracy. Then we set penalty value to be 0.0001(name the model as model4). Although the accuracy gets improved, overfitting exists again.



(a) Error curve(L1) (b) Accuracy curve(L1)

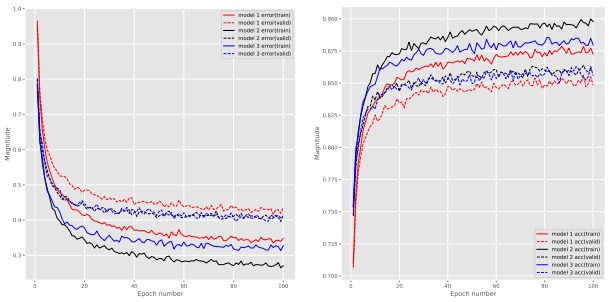
(c) Error curve(L2) (d) Accuracy curve(L2)

Figure 4. Add weight penalty only

As is discussed in the two experiments above, we can possibly gain an empirical knowledge that if we drop more hidden units in dropout layers or increase penalty value in weight penalty, overfitting might be mitigated to a larger extent but the model might generalize worse and if we drop less units or decrease penalty value in weight penalty, overfitting might be worsen but the model generalizes better. This is a very important empirical knowledge because we will use it to make adjustment to hyperparameters to find the optimal model.

We need to find combination of dropout and weight penalty in order to prevent overfitting and generalize well at the same time. In the first experiment, we chose the latter two models and expect to use weight penalty to reduce overfitting. L1 penalty would be excluded first because of its terrible generalization. So the only choice is L2 penalty. We start by combining model2 in the first experiment and model4 in the second experiment(name this new model model1 in this context) and hopefully we could get a model that reduces overfitting on the basis of model2 in the first experiment. The graphs of error and accuracy curves are shown in (a) and (b) in figure 5. We can see that overfitting does get mitigated. What's more, the new model also generalizes well on validation set with an accuracy of approximately 0.85. We can also have a try on other possible combinations. For example, if we set the inclusion proba-

bility to be 0.9, we could infer that the gap between training error and validation error will be larger but the accuracy might be higher based on the empirical knowledge we discussed before. We name this new model model2 in figure 5. The result is exactly the same as we predicted. Actually, this model is pretty good by now but we can possibly reduce overfitting further. So we increase the weight penalty value slightly in order to mitigate overfitting in this model. We choose penalty value to be 0.0003 and name the new model as model3 in figure 5. We get a good model with a relatively good accuracy(above 0.85), what's more, overfitting is also mitigated. So this is the model that we want. Although there are still many other possible combinations, we can stop here because the accuracy starts to converge no matter how we change the parameters and continuing to test other possible combinations won't help too much.



(a) Error curve (b) Accuracy curve

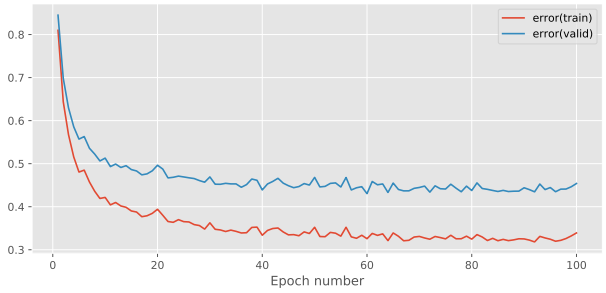
Figure 5. Combine dropout and weight penalties

From the analysis and design above, we finally find the best model. Now we need to use test set to evaluate this model's ability of generalization. The result is shown in graphs in figure 6. We can see that this model generalizes well on test set with an accuracy slightly above 0.84.

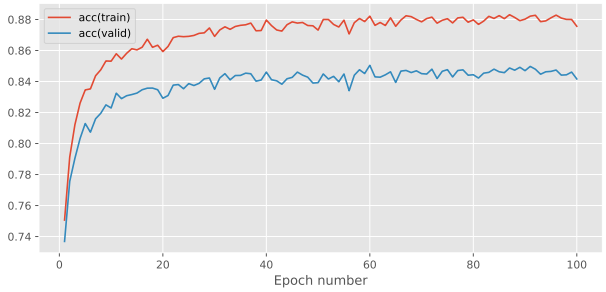
5. Literature Review: Dropout: A Simple Way to Prevent Neural Networks from Overfitting

This paper introduces dropout as an efficient method to prevent overfitting in neural networks and improve the performance of neural networks in many areas.

Generally speaking, the paper is structured in 9 sections. In section 1, this paper have a brief introduction of what dropout is and how it works. Then in section 2, it clarifies the motivation of this idea using the theory of how sexual reproduction affects evolution. In section 3, it describes some related work. Section 4 gives a detailed description about dropout model. Section 5 discusses the training of dropout. Section 6 applies dropout in many different domains of neural networks and presents experimental results in detail and also compares dropout with other ways of regularization. Section 7 analyzes how hyperparameters in dropout affect the performance of dropout. Section 8 introduces RBM model and finally section 9 is about marginal dropout.



(a) Test error



(b) Test accuracy

Figure 6. Error and accuracy in test set

This paper has lots of strengths. The first one is that it gives us an intuition about how dropout makes hidden units more robust by reducing co-adaptation between units in the first part using a theory in biology as an analogy of how dropout works, which is both interesting and convincing. Secondly, it analyzes in depth about the mechanism of dropout and clearly explains how and why dropout works using graphs and simple mathematical formulas. Thirdly, this paper performs a wide range of different experiments of the performance of dropout in different domains including computer vision, speech recognition, document classification and computational biology. The motivation to do so many experiments is to prove that dropout is a general technique rather a specific technique in any domain. Presented by detailed figures with many complicated curves and tables with precise numbers, the experimental results, which show that dropout does mitigate overfitting and lowers error rates in all the domains included, are very convincing and clear to understand. Another strength is that this paper compares dropout with other regularization models such as Bayesian Neural Networks and Standard Regularizers and also discusses how hyperparameters such as dataset size and dropout probability affect the performance of dropout.

However, there is also something that can be improved. For example, in section 7.3 which is about the effect of drop probability, this paper could move a step further to give a more detailed explanation of why classification error becomes flat when p is between 0.4 and 0.8. Another one is that this paper could cover how the structure of neural network affect the performance of dropout and selection of drop probability. Finally, this paper gives an empirical way of using dropout which sets retention probability to be

0.8 in input layer and 0.5 in hidden layers, however, does not give a convincing reason, which is not rigorous because selection of drop probability varies with different structures of neural network.

6. Conclusions

To conclude from all the experiments, increasing hidden units size worsens overfitting and increasing number of hidden layers does not help mitigate overfitting. Dropout and weight penalty can be efficient to mitigate overfitting. Specifically, dropout works by dropping hidden units randomly according to drop probability in each mini-batch when training. Weight penalty includes L1 penalty and L2 penalty which penalize the sum of absolute value and square value of weights respectively. The best model that we can extract from the given baseline network combines dropout and L2 penalty by putting dropout layers with inclusion probability 0.9 on each hidden layer as well as input layer and using L2 penalty with penalty value 0.0003 on each affine layer.

However, there are also some further work that can be expected. Although we find the relatively optimal model in this given baseline network, the accuracy on test is only slightly above 0.84. However, what we expected might be 0.95 or even more. It might be the structure of the baseline neural network that limits the accuracy, therefore, we could potentially improve the neural network by adding more hidden units or using different activation functions or learning rules in future work.

References

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15 (56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.