MLP Coursework 1

Kyriakos Kyriakou - s2281922

Abstract

In this report we study the problem of overfitting, which is the training regime where performance increases on the training set but decrease on validation data. Overfitting is a major problem because a Neural Network learns only to describe seen data amazingly, but fails to understand and make new predictions. We first analyse the given example and discuss the probable causes of the underlying problem. Then we investigate how the depth and width of a neural network can affect overfitting in a feedforward architecture and observe that increasing width and depth make the network perform better on training data as more parameters exist which can make the network memorize more phenomena, but makes it perform worse on the unseen data and cause bigger overfitting. Next we discuss why two standard methods, Dropout and Weight Penalty, can mitigate overfitting, then describe their implementation and use them in our experiments to reduce the overfitting on the EMNIST dataset. Based on our results, we ultimately find that the three regularization methods applied dealt well with the problem of overfitting. Dropout regularization worked better for addressing that issue. A combination of the tested approaches is discussed as it could be an ideal solution of the overfitting problem as the netowork could get the best of the three worlds. Finally, we briefly review a technique that studies use of weight decay in adaptive gradient algorithms, discuss its findings, and conclude the report with our observations and future work. Our main findings indicate that L2 Weight Penalty is equivalent with weight decay when their coefficients have a relation $\lambda' = \frac{\lambda}{2}$, but when Adam optimizer is used, a more complex relation is needed to show that weight decay and L2 weight penalty are equivalent. Finally we discuss our implementation of L2 Weight Penalty that uses a combination of Adam with L2 regularization and weight decay.

1. Introduction

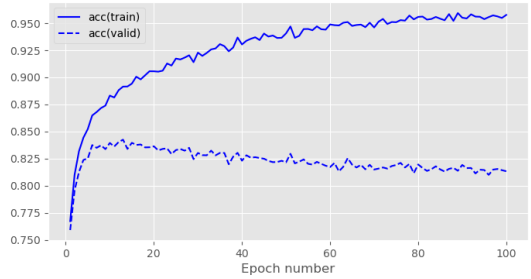
In this report we focus on a common and important problem while training machine learning models known as overfitting, or overtraining, which is the training regime where performances increase on the training set but decrease on unseen data. We first start with analyzing the given problem in Fig. 1, study it in different architectures and then investigate different strategies to mitigate the problem. In particular, Section 2 identifies and discusses the given problem, and investigates the effect of network width and depth in terms of generalization gap (see Ch. 5 in Goodfellow et al. 2016) and generalization performance. Section 3 introduces two regularization techniques to alleviate overfitting: Dropout (Srivastava et al., 2014) and L1/L2 Weight Penalties (see Section 7.1 in Goodfellow et al. 2016). We first explain them in detail and discuss why they are used for alleviating overfitting. In Section 4, we incorporate each of them and their various combinations to a three hidden layer[1] neural network, train it on the EMNIST dataset, which contains 131,600 images of characters and digits, each of size 28x28, from 47 classes. We evaluate them in terms of generalization gap and performance, and discuss the results and effectiveness of the tested regularization strategies. Our results show that the three regularization methods applied dealt well with the problem of overfitting. Dropout regularization worked better for addressing that issue. A combination of the tested approaches is discussed as it could be an ideal solution of the overfitting problem as the netowork could get the best of the three worlds. In Section 5, we study a related work that studies weight decay in adaptive gradient algorithms.[2] Finally, we conclude our study in section 6, noting that L2 Weight Penalty is equivalent with weight decay when their coefficients have a relation $\lambda' = \frac{\lambda}{2}$, but when Adam optimizer is used, a more complex relation is needed to show that weight decay and L2 weight penalty are equivalent. Finally we discuss our implementation of L2 Weight Penalty that uses a combination of Adam with L2 regularization and weight decay.

2. Problem identification

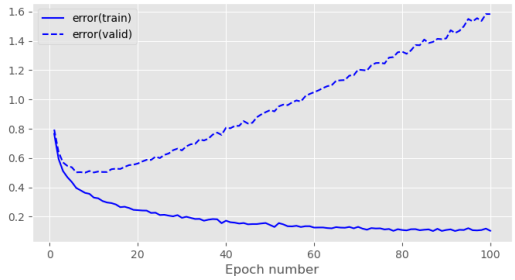
Overfitting to training data is a very common and important issue that needs to be dealt with when training neural networks or other machine learning models in general (see Ch. 5 in Goodfellow et al. 2016). A model is said to be overfitting when as the training progresses, its performance on the training data keeps improving, while its is degrading on validation data. Effectively, the model stops learning related patterns for the task and instead starts to memorize

[1] We denote all layers as hidden except the final (output) one. This means that depth of a network is equal to the number of its hidden layers + 1.

[2] Instructor note: Omitting this for this coursework, but normally you would be more specific and summarise your conclusions about that review here as well.



(a) accuracy by epoch



(b) error by epoch

Figure 1. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for the baseline model.

specificities of each training sample that are irrelevant to new samples.

Overfitting leads to problematic models that cannot guarantee correct predictions. Training a model to understand very well specific data, can lead to a better performance in the case of known data (training data), but will also lead to a low performance in the case of unseen data (test data). Thus, an overfitted model can perform amazingly on existing data, but when the time comes to generalize new data it will perform badly and make false assumptions.

Although it eventually happens to all gradient-based training, it is most often caused by models that are too large with respect to the amount and diversity of training data. The more free parameters the model has, the easier it will be to memorize complex data patterns that only apply to a restricted amount of samples. A prominent symptom of overfitting is the generalization gap, defined as the difference between the validation and training error. A steady increase in this quantity is usually interpreted as the model entering the overfitting regime.

Neural Network models tend to use a network capacity which controls how well the network will learn to describe many phenomena (see Ch. 3 in Nielsen 2019). A small capacity of the network will lead to not understanding the training data well (underfit). On the other hand, a big capacity of the network will lead to overfitting, where the model will trained more than enough on the training data to perform great there, but will lead to poor testing performance. A model overfit will eventually happen when creating a model with a big network capacity (many parameters).

# Hidden Units	Val. Acc.	Train Error	Val. Error
32	78.9	0.553	0.691
64	80.9	0.344	0.663
128	80.4	0.167	0.942

Table 1. Validation accuracy (%) and training/validation error (in terms of cross-entropy error) for varying network widths on the EMNIST dataset.

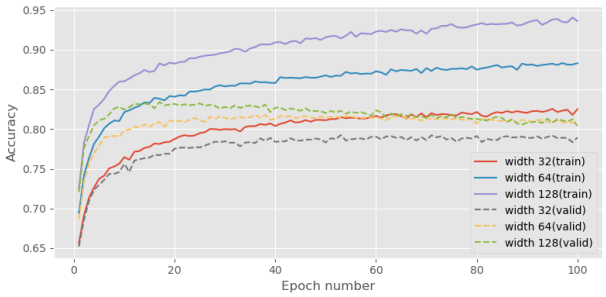
Fig. 1a and 1b show a prototypical example of overfitting. We see in Fig. 1a that curves of training and validation are shown in term of cross-entropy error by the epochs. From Fig.1a we can spot that through epochs, the training error is decreasing as the network learns very well the training data. The error captured on the validation of the model, at the beginning, improves on the validation data until epoch 10, and then starts to increase. We can spot the overfitting at around epoch 10 where the error starts to get the opposite direction and gets worse while the training continues to gets better with experience. In Fig.1b we can see that similarly curves of training and validation are show, but now in term of classification accuracy by the epochs. Throughout the epochs, we can spot that the training accuracy is getting better. In contrast, the validation curve shows us that the accuracy it started getting better until the epoch 10, and similarly to the error, starts to not getting better and eventually to gets worse as epochs are passed. This is another example where we can spot the overfitting happening at epoch 10 where the model no longer understands the test data.

The extent to which our model overfits depends on many factors. For example, the quality and quantity of the training set and the complexity of the model. If we have sufficiently many diverse training samples, or if our model contains few hidden units, it will in general be less prone to overfitting. Any form of regularisation will also limit the extent to which the model overfits.

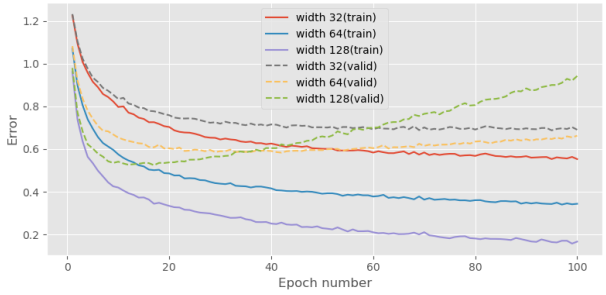
2.1. Network width

First we investigate the effect of increasing the number of hidden units in a single hidden layer network when training on the EMNIST dataset. The network is trained using the Adam optimizer with a learning rate of 9×10^{-4} and a batch size of 100, for a total of 100 epochs.

The input layer is of size 784, and output layer consists of 47 units. Three different models were trained, with a single hidden layer of 32, 64 and 128 ReLU hidden units respectively. Fig. 2 depicts the error and accuracy curves over 100 epochs for the model with varying number of hidden units. Table 1 reports the final accuracy, training error, and validation error. We observe that the network with the biggest width (most units), is the one achieving highest accuracy and lower cross-entropy error on the training data. On the validation data, we can view that 32 unit – width network achieved the worst validation accuracy and the network with the 64 – units width achieved the best. The

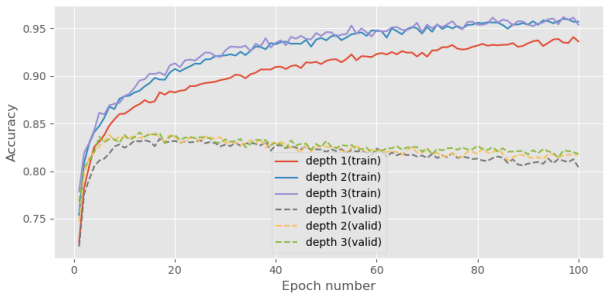


(a) accuracy by epoch

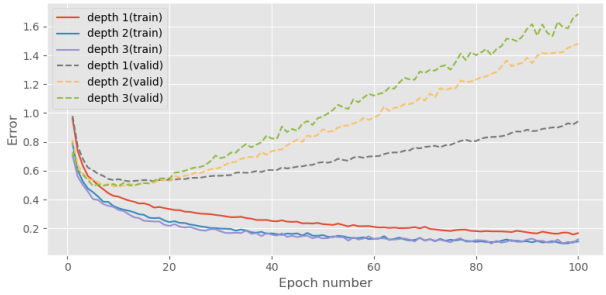


(b) error by epoch

Figure 2. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for different network widths.



(a) accuracy by epoch



(b) error by epoch

Figure 3. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for different network depths.

# Hidden Layers	Val. Acc.	Train Error	Val. Error
1	80.4	0.167	0.943
2	81.7	0.109	1.480
3	81.8	0.125	1.690

Table 2. Validation accuracy (%) and training/validation error (in terms of cross-entropy error) for varying network depths on the EMNIST dataset.

lowest validation error was seen on the 64 – unit network as well, where the highest validation error was captured on the network with the biggest width (128 hidden units). Overfitting starts appearing on the epoch 10 for the 128 units network, and on the 30th epoch for the 64 units network. Worth mentioning here is that the 32 units network has almost nothing overfit (only a bit at the end that the curve becomes flat), but the performance of the model is not good either.

Experiments with different various number of hidden layers showed that the results are affected in the way that the network width is formed. The results are affected in a way that - for networks having more hidden units, training and validation accuracy becomes better, but the validation error becomes much worse. Thus, wider networks are more likely to have bigger overfit effect.

2.2. Network depth

Next we investigate the effect of varying the number of hidden layers in the network. Table 2 and Fig. 3 depict results from training three models with one, two and three hidden layers respectively, each with 128 ReLU hidden units. As with previous experiments, they are trained with the Adam optimizer with a learning rate of 9×10^{-4} and a batch size of 100.

We observe that the network with the most depth (most layers), achieves the best accuracy score on the training error. The worst accuracy score on training phase is seen on the smallest depth network. The network with the smallest depth is also observed to be the one with the lowest validation error. The network with the 3 layers, attained the highest validation accuracy and cross-entropy error, while the network with the 2 hidden layers got the lowest training error. Based on the validation accuracy – error trade-off it is observed that the effect of overfitting is appeared less in the netowork with the smallest depth (1 layer). Overfitting starts appearing on the epoch 10 for all the different networks created, with the lower overfit appearing on the network with 1 depth layer.

After experimenting with various depths, more depth can be seen as one more way to improve the learning process. Results are affected consistently by the change of depth in a network. More depth leads to a better generalization, but also to a highest error. That's why overfitting is appeared stronger on deeper networkds..

More width and depth help the nework until the paramater number becomes very high. Having a network which is really width, depth, or both, will lead on creating many learning parameters. This will make the network memorize the training data so much that will not able to perform well

on the unseen data. Thus, the network with the 128 hidden units and network with most de pth (3 layers) performed the worst in the experiments (biggest overfits). If a network has a really small width, or low depth, might generate less than enough parameters that will also affect the model in a way that will not trained the best it could. This can be observed on the 32 hidden units and on the 1 layer networks which had the less accuracy and also the highest training error. If the depth and width of a network is balanced, the network will have the lowest overfit and the best performance. A balanced network will lead to achieving the highest validation accuracy score and lowest validation error as possible.

3. Dropout and Weight Penalty

In this section, we investigate three regularization methods to alleviate the overfitting problem, specifically dropout layers and the L1 and L2 weight penalties.

3.1. Dropout

Dropout (Srivastava et al., 2014) is a stochastic method that randomly inactivates neurons in a neural network according to an hyperparameter, the inclusion rate (*i.e.* the rate that an unit is included). Dropout is commonly represented by an additional layer inserted between the linear layer and activation function. Its forward pass during training is defined as follows:

$$\text{mask} \sim \text{bernoulli}(p) \tag{1}$$

$$\boldsymbol{y}' = \text{mask} \odot \boldsymbol{y} \tag{2}$$

where $\boldsymbol{y}, \boldsymbol{y}' \in \mathbb{R}^d$ are the output of the linear layer before and after applying dropout, respectively. mask $\in \mathbb{R}^d$ is a mask vector randomly sampled from the Bernoulli distribution with inclusion probability p, and $\odot$ denotes the element-wise multiplication.

At inference time, stochasticity is not desired, so no neurons are dropped. To account for the change in expectations of the output values, we scale them down by the inclusion probability p:

$$\boldsymbol{y}' = \boldsymbol{y} * p \tag{3}$$

As there is no nonlinear calculation involved, the backward propagation is just the element-wise product of the gradients with respect to the layer outputs and mask created in the forward calculation. The backward propagation for dropout is therefore formulated as follows:

$$\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}} = mask \tag{4}$$

Dropout is an easy to implement and highly scalable method. It can be implemented as a layer-based calculation unit, and be placed on any layer of the neural network at will. Dropout can reduce the dependence of hidden units between layers so that the neurons of the next layer will not rely on only few features from of the previous layer. Instead, it forces the network to extract diverse features and evenly distribute information among all features. By randomly dropping some neurons in training, dropout makes use of a subset of the whole architecture, so it can also be viewed as bagging different sub networks and averaging their outputs.

3.2. Weight penalty

L1 and L2 regularization (Ng, 2004) are simple but effective methods to mitigate overfitting to training data. The application of L1 and L2 regularization strategies could be formulated as adding penalty terms with L1 and L2 norm square of weights in the cost function without changing other formulations. The idea behind this regularization method is to penalize the weights by adding a term to the cost function, and explicitly constrain the magnitude of the weights with either the L1 and L2 norms. The optimization problem takes a different form:

$$\text{L1: } \min_{\boldsymbol{w}} E_{\text{data}}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w}) + \lambda \|w\|_1 \tag{5}$$

$$\text{L2: } \min_{\boldsymbol{w}} E_{\text{data}}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w}) + \lambda \|w\|_2^2 \tag{6}$$

where E_{data} denotes the cross entropy error function, and $\{\boldsymbol{X}, \boldsymbol{y}\}$ denotes the input and target training pairs. λ controls the strength of regularization.

Weight penalty works by constraining the scale of parameters and preventing them to grow too much, avoiding overly sensitive behavior on unseen data. While L1 and L2 regularization are similar to each other in calculation, they have different effects. Gradient magnitude in L1 regularization does not depend on the weight value and tends to bring small weights to 0, which can be used as a form of feature selection, whereas L2 regularization tends to shrink the weights to a smaller scale uniformly.

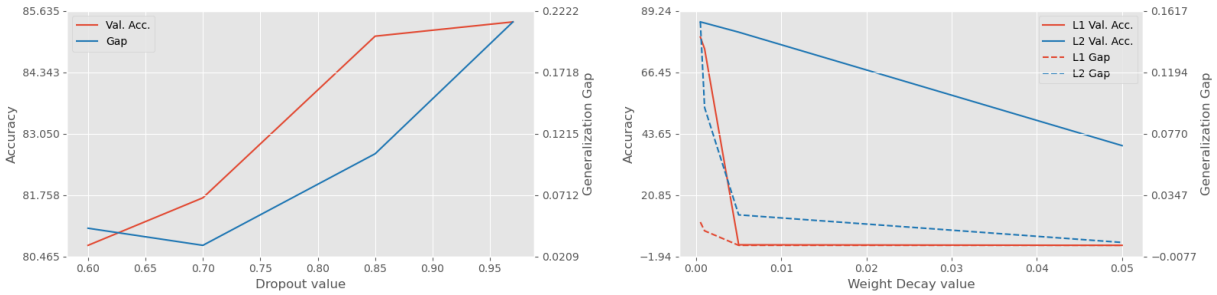
The L1 and L2 regularization techniques can be sometimes combined to help one network get the benefits of both worlds. The ability of L1 regularization to be more selective at adopting data features, can lead to get more sparsity on the network. More sparsity will increase the space and time efficiency of the network. On the other hand, L2's regularization ability to shrink the weights by distributing the error to them, will tend the network to make better predictions (accuracy increase). Combining L1 and L2 will result in building a network that is accurate and sparse as well, something that will improve the performance. One could add L1 regularization on the first half of layers and L2 on the rest, or the L2 first and then L1, or design a model and for each layer switch between L1 and L2 alternately. Other possible combinations could be possible.

4. Balanced EMNIST Experiments

Here we evaluate the effectiveness of the given regularization methods for reducing the overfitting on the EMNIST dataset. We build a baseline architecture with three hidden layers, each with 128 neurons, which suffers from overfitting as shown in section 2.

Model	Hyperparameter value(s)	Validation accuracy	Train Error	Validation Error
Baseline	-	0.837	0.241	0.533
Dropout	0.6	80.7	0.549	0.593
	0.70	82.8	0.448	0.509
	0.85	85.1	0.329	0.434
	0.97	***85.4***	***0.244***	***0.457***
L1 penalty	*5e-4*	*79.5*	*0.642*	*0.658*
	1e-3	75.0	0.849	0.856
	5e-3	2.41	3.850	3.850
	5e-2	2.20	3.850	3.850
L2 penalty	*5e-4*	*85.1*	*0.306*	*0.460*
	1e-3	84.9	0.356	0.453
	5e-3	81.3	0.586	0.607
	5e-2	39.2	2.258	2.256

Table 3. Results of all hyperparameter search experiments. *italics* indicate the best results per series (Dropout, L1 Regularization, L2 Regularization) and **bold** indicates the best overall.



(a) Accuracy and error by inclusion probability.

(b) Accuracy and error by weight penalty.

Figure 4. Accuracy and error by regularization strength of each method (Dropout and L1/L2 Regularization).

Here we train the network with a lower learning rate of 10^{-4}, as the previous runs were overfitting after only a handful of epochs. Results for the new baseline (c.f. Table 3) confirm that lower learning rate helps, so all further experiments are run using it.

Then, we apply the L1 or L2 regularization with dropout to our baseline and search for good hyperparameters on the validation set. We summarize all the experimental results in Table 3. For each method, we plot the relationship between generalisation gap and validation accuracy in Figure 4.

First we analyze three methods separately, train each over a set of hyperparameters and compare their best performing results.

The first method we experimented with was dropout regularization, that was applied to our baseline model. Changing the dropout rate of the layer was tested using different hyper-parameters. That hyper-parameter is the inclusion probability that controls the dropping rate. We started with the first experiment using an inclusion probability of 0.6, and then three more experiments were followed changing it to 0.70, 0.85, and 0.97 respectively. Having a bigger inclusion rate means less units are dropped from the layer. The dropout method did solve the problem with the overfitting of the baseline model. Experimenting with the different dropout values, we saw that higher dropout value, led to higher overall performance of the model. Dropout value of 0.97 achieved highest validation accuracy score of 85.4% without any overfitting.

Experimenting with L1 penalty regularization was followed. Different weight decay values were used to test how the model is performed with their change. L1 regularization dealt successfully with the problem of overfitting. The used hype-parameters values for this experiments were decays of 5e-4, 1e-3, 5e-3, and 5e-2. The results of the experiments showed that the networks with smaller value of weight decay performed better than the networks using bigger values. Using the weight coefficient of 5e-4 achieved the highest validation performance of 79.5%. For the last experiments, L2 regularization penalty was applied on the network using different weight decay values. The same values used for the L1 regularization experiments were used again. The results showed that the smallest weight decay value (5e-4) achieved again the highest performance with accuracy of 85.1% on the validation dataset. L2 regularization has overcome the overfitting as well. Smaller weight decay value seemed to had a bigger effect on the model's performance when L1 regularization was applied. Furthermore,

L2 penalty made the networks to achieve higher accuracy than L1 penalty.
Based on the experiments, the model that used only dropout for regularization and used specifically inclusion rate of 0.70 as hyper-parameter, attained the best overall performance of all models based on the trade-off of validation accuracy and generalization gap. That was also the model selected to be evaluated on the unseen test data. Its evaluation showed that the model achieved accuracy of 84.5%, and generalization gap of 0.242.

Combinations of the three regularization techniques could lead to an ideal solution, and give the best performing model without having the overfitting problem as well. Some possible tests of some combinations are discussed next. Two new experiments could use a combination of Dropout and L1 regularization and the other Dropout and L2 regularization. The hyper-parameter for dropout could be the best of the previous best experiments (0.97) and the hyper-parameter for L1 + L2 penalty could be again the best from the already tested values (5e-4). Experimenting with those two new architectures can give us a good understanding if Dropout and L1 and/or Dropout and L2 combinations can improve even more the model performance. For both experiments, Dropout layer with value 0.97 would be placed between all the layers and their activation layer. L1 and L2 penalties will be applied to each layer for each experiment respectively. Another possible experiment could be combining the L1 and L2 regularizations to see if their different effect on the weights actually improves the performance of the network. The weight decay values we could use to check this, are the ones achieved the greatest results from the previous experiments. That being the value 5e-4. The setup of the network could add L1 regularization on the first half of layers and L2 on the rest, or the L2 first and then L1, or design a model and for each layer switch between L1 and L2 alternately. A good fourth experiment would be a combination of all the regularization method (Dropout + L1 + L2). The hyper-parameters that could be selected for that would be the ones achieved the best results from the previous tests. Therefore, for Dropout would be 0.97, and for L1 and L2 the weight decay value would be 5e-4. The setup could be adding Dropout to all the layers and switching alternately the L1 and L2 penalties for each layer. This setup could tell us, if actually, the combination of the three affects the performance of the model positively. Another experiment that could be worth checking, will be to combine L2 regularization and Dropout, but use a smaller dropout value this time of a value 0.5 (to drop more units). For the hyper-parameter of L2 penalty we could stick to the best one gotten from previous tests (5e-4) so we can observe the effect of the combination of the two but with a change on the dropout layer. The setup should be the same as the previous one combining Dropout and L2 in order to compare the results with too (add dropout to every layer between hidden and activation functions, and L2 penalty to every layer). The same setup could be use for a sixth test, but instead of changing the dropout value, we could make the weight decay value bigger. The dropout value selected would be 0.97, but the penalty value would be 5e-3 this time. With this test, we could observe if higher inclusion probability combined with not a big weight decay can lead to a good balance and eventually to a good model. The already executed tests tell us that when the inclusion probability is bigger, better performance is recorded. Similarly, better performance is captured when the weight decay value is smaller (for both L1 and L2 regularizations). Therefore, a seventh and an eighth experiment, can be used to determine if this is consistent. The seventh experiment would be combining Dropout with L2 and the eighth combining Dropout and L1. This time, the inclusion probability for the Dropout layers could be 0.99, which is even higher, and for each experiment, the L2 and L1 penalty value respectively could be 5e-5 which is even smaller. The setup should stay the same as the previous experiments so not other factors would affect the results except of the specified hyper-parameters. We've just stated eight more possible experiments that could give us more insights about the regularizations techniques and their hyper-parameters but, there are also other experiments that could be carried out to give us more different observations.

5. Literature Review: Decoupled Weight Decay Regularization

Summary In this section, we briefly study a method (Loshchilov & Hutter, 2019) that decouples the weight penalty from the weight update. The authors shed light onto the relation between weight decay and L2 weight penalty for SGD and Adam optimizers, pointing out that weight decay and L2 weight penalty are not equivalent when used with the Adam optimizer.

In particular, the authors claim that SGD with L2 weight penalty and weight decay are equivalent when their coefficients have the relation $\lambda' = \frac{\lambda}{2}$ (see their **Proposal 1**), which they prove with Equations (5) and (6) (proof is in their Appendix A). In addition, they claim that such a simple scalar relation does not hold in Adam optimization (see **Proposal 2**) and show, in their Appendix A, that the necessary relation for equivalence requires use of a preconditioner matrix $\boldsymbol{M}_t$. This requirement can be explained with the fact that Adam optimizer scales gradients based on the previous recorded magnitudes of the weights. When L2 regularization is applied for larger gradients of weights, the regularization process has less effect, as they are regularized by a smaller values. On the other hand, when weight decay is used, the regularization effect doesn't have any complications, as all the weights are regularized with the same value of λ. That leads the larger magnitude weights to get scaled more effectively. Introducing a pre-conditioner matrix $\boldsymbol{M}_t$ can help.

The implementation of our L2 weight penalty regularization implements the same approached followed in (Loshchilov & Hutter, 2019) study at Algorithm 2 with that being Adam with L2 regularization and Adam with decoupled weight decay (AdamW) algorithm. Penalizing gradients is performed

in the *mlp penalties python file* file in the class *L2Penalty*. A *grad* method was created that calculates and returns the value of the penalty gradient with respect to the parameter. That is the application of Adam with L2 regularization as we use Adam learning rule when we train our network. The same technique is seen in the studied paper at line 6 of Algorithm 2. The Adam with decoupled weight decay (AdamW) was applied in our implementation when we set the learning rule as *AdamLearningRule*. This implementation is located in the *mlp learning rules python file* at the class of *AdamLearningRule*. In our application we assume that the scheduled multiplier of Algorithm 2 (line 11) is equal to 1 thus we ignore it. Our α value is not a constant compared to the study, and is updated based on the learning rate. The application of this method from is located in the function of *update params*, where we update the parameters using the formula stated in line 12 of Algorithm 2 (see (Loshchilov & Hutter, 2019)).

Finally the authors validate their findings and proposed decoupled optimization strategies in various learning rate schedules, initial learning rates, network architectures and datasets. [3]

6. Conclusion

In conclusion, our study observed the problem of overfitting that leads neural networks to fail understanding new data and make new predictions. We have investigated how the depth and width of different networks can consistently affect the performance of the networks by making them overfit more when more units and layers are used. Next to that, we introduced three ways of regularization including Dropout, L1 weight penalty, and L2 weight penalty, that help mitigate the problem of overfitting. Experiments with these methods were done using different hyper-parameters, and proved that the methods indeed help address that issue. We suggested some possible new experiments that could help in further studies to investigate how neural networks are affected when those methods are combined. Moreover, our study reviewed a technique from a paper that uses weight decay in adaptive gradient algorithms, and we observed why in Adam optimizer a more complex relation is needed to achieve equivalence between L2 weight penalty and weight decay. Finally we discussed our implementation of L2 Weight Penalty, which uses a combination of Adam with L2 regularization and with weight decay. Future direction of the current study could investigate and experiment with combinations of regularization techniques to see if there is an effect on the overall performance of the model, and discover the best possible combinations with their according network setup.

[3] Instructor note: Omitting this for this coursework, but normally you would give an evaluation of the experiment setup in (Loshchilov & Hutter, 2019) here.

References

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Loshchilov, Ilya and Hutter, Frank. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Ng, Andrew Y. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 78, 2004.

Nielsen, Michael. *Neural Networks and Deep Learning*. Determination Press, 2019. http://neuralnetworksanddeeplearning.com/.

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.