MLP Coursework 1

s2308470

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. **[The training model fits noisy features in training set, which leads to bad performance on unseen data.]** . 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 **[intensify 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 **[dropout and L2 penalty could alleviate overfitting and improve the performance of the model on unseen data by choosing proper hyperparameters, but the effect of L1 penalty is less prominent.]** . 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 **[overfitting correlates to the width and depth of model architecture. Some methods such as dropout and weight penalty could mitigate overfitting by a certain extent.]** .

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 **[dropout and L2 penalty could alleviate overfitting and improve the performance of the model on unseen data by choosing proper hyperparameters, but the effect of L1 penalty is less prominent.]** . 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 **[overfitting correlates to the width and depth of model architecture. Some methods such as dropout and weight penalty could mitigate overfitting by a certain extent.]** .

2. Problem identification

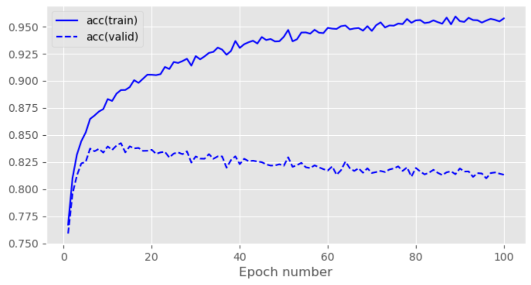
Overfitting to training data is a very common and important issue that needs to be dealt with when training neural networks or other machine learning models in general (see Ch. 5 in Goodfellow et al. 2016). A model is said to be overfitting when as the training progresses, its performance on the training 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 specificities of each training sample that are irrelevant to new samples.

[Overfitted model usually has better performance on training set, but worse performance on unseen data, for example, validation set. The model learns overmuch patterns from training data, which contains irrelevant patterns that have less meaning on unseen data, or even causes bad predictions. Since a model is eventually judged by its performance on unseen data, overfitting leads to worse model quality even its performance on training set increases.] .

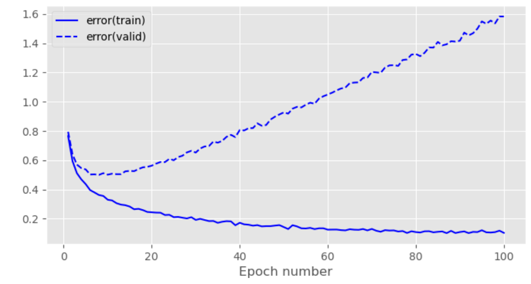
Although it eventually happens to all gradient-based training, it is most often caused by models that are too large with respect to the amount and diversity of training data. The more free parameters the model has, the easier it will be to memorize complex data patterns that only apply to

[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.

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.

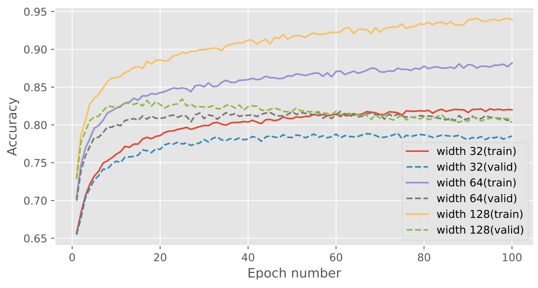
[A large capacity network tends to express more patterns. As the capacity of a simple model increases, it will capture more features and performs better. However, if the capacity increases continuously and exceeds a specific point, the model explores more features than we expect. The point is where a promising model we desired occurs and after that it starts to overfit.] .

Fig. 1a and 1b show a prototypical example of overfitting. We see in Fig. 1a that **[the accuracy on both training and validation set increases dramatically before 10 epochs. After that, the training accuracy continue increasing by a stable velocity while the validation accuracy reach its maximum and decreases. Similar phenomenon appears in Fig. 1b. Both errors decreasing with 0-10 epochs, and then the training error still reducing but the validation error start increasing. Overfitting occurs on the maximum of validation accuracy, which is the minimum of validation error. It continuously exists after that point since the generalization gap is steadily increasing during that period.]** .

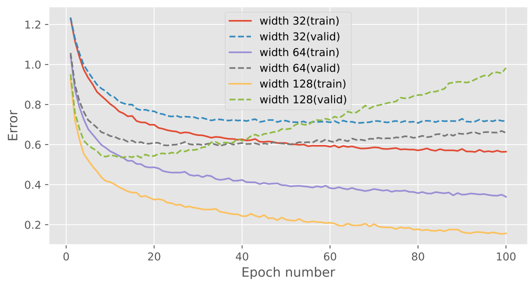
The extent to which our model overfits depends on many factors. For example, the quality and quantity of the training set and the complexity of the model. If we have sufficiently many diverse training samples, or if our model contains few hidden units, it will in general be less prone to overfitting.

# Hidden Units	Val. Acc.	Train Error	Val. Error
32	78.5	0.565	0.717
64	80.9	0.339	0.659
128	80.3	0.156	0.983

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



(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.

Any form of regularisation will also limit the extent to which the model overfits.

2.1. Network width

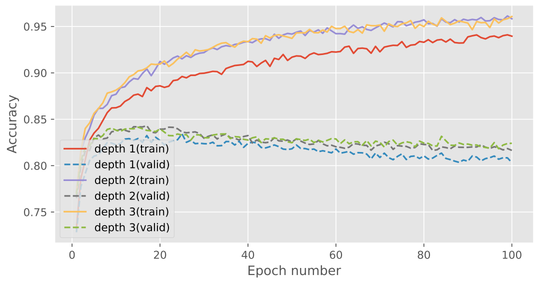
[ ] [ ]

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.

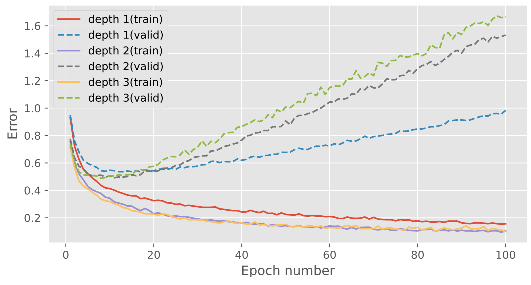
The input layer is of size 784, and output layer consists of 47 units. Three different models were trained, with a single hidden layer of 32, 64 and 128 ReLU hidden units respectively. Fig. 2 depicts the error and accuracy curves over 100 epochs for the model with varying number of hidden units. Table 1 reports the final accuracy, training error, and validation error. We observe that **[as the width increasing, the training error decreases and its accuracy increases. They have similar shape until the tail between the curves of different widths. The maximum value of**

# Hidden Layers	Val. Acc.	Train Error	Val. Error
1	80.3	0.156	0.983
2	81.6	0.105	1.530
3	82.4	0.099	1.670

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



(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.

validation error and accuracy improves, whereas they decreases by a larger speed as the width increases and the final validation error of width 128 is even larger than that of width 32 and 64. Moreover, the final validation accuracy] .

[As shown by the experiment result, increasing the width of a layer could improve the performance of models initially, but it usually leads to a higher speed of performance degradation. As the number of training epoch increasing, larger width models might have a worse prediction than the smallers. This result matches with our prior expectation. Wider layer means the model learns more features from the training data, but once it capture more than the diversity of training samples, it probably enters overfitting regime.] .

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 **[adding layers does not give significant improvement to the value on EMNIST dataset. As shown in Fig. 3a and 3b, increasing depth from 1 to 3 has a slight effect on both training and validation accuracy. Its effect on training error is also imperceptible, but a deeper model causes a high speed of error increasing as the training time increases.]** .

[By our experiment results, increase the number of hidden layers with 128 ReLU hidden units has slight improvement on EMNIST dataset. It might because of width 128 is a sufficient unit number for EMNIST dataset and simply increasing the number of hidden layers with 128 units cannot improve performance to a large extent. However, deeper increasing width gives a more apparent overfitting to the model, as the generalization gap is larger.] .

[In our experiments, changing the width of a single hidden layer network improves the model performance initially. Whereas, the effect of changing the depth of model is not that apparent. Both width and depth increasing causes overfitting as the training epoch passed.] .

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:

$$\text{mask} \sim \text{bernoulli}(p) \quad (1)$$
$$\boldsymbol{y}' = \text{mask} \odot \boldsymbol{y} \quad (2)$$

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:

$$\boldsymbol{y}' = \boldsymbol{y} * p \quad (3)$$

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:

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

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

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.

[It is possible to combine L1 and L2 regularization by adding both L1 and L2 penalty terms to the cost function. The following formula shows the optimized cost function:

$$\text{L1\&L2: } \min_w \ E_{\text{data}}(X, y, w) + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2 \tag{7}$$

where λ_1 and λ_2 are two weights to control the strength of L1 and L2 regularization separately. As mentioned above, L1 regularization performs as a feature selection. It bring the weight of some inessential features, for example, noisy features, to 0 and reduce their impact on predicted result. Meanwhile, L2 regularization can scale down the weight to smooth the decision boundary and avoid extreme weights. This optimized function could potentially combine their strengths by carefully choosing the hyperparameters λ_1 and λ_2.] .

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.

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.

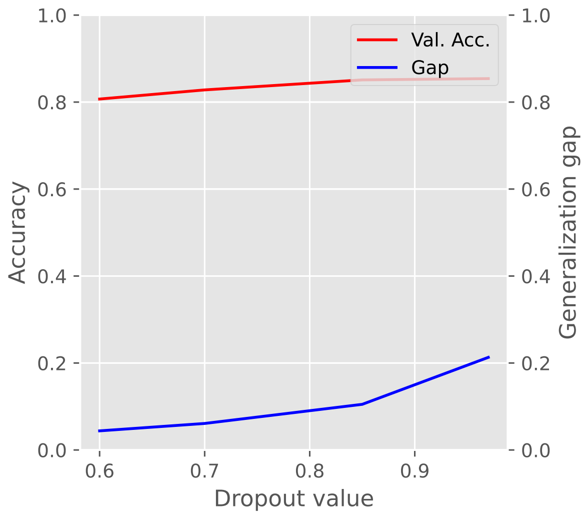
First we analyze three methods separately, train each over a set of hyperparameters and compare their best performing results.

[The experiments aims to discover how L1, L2 regularization and dropout with different hyperparameters mitigate overfitting regime and improve the generalization performance individually. As shown in Table 3, we run 12 experiments in total, 4 different hyperparameters for each method. The hyperparamer for dropout is the inclusion probability rate p and that for L1 L2 regularization is the penalty weight λ, mentioned in Section 3. The only difference between the model we used for each experiment and the baseline is the method we applied with a specific hyperparameter. All the other configurations, for example, learning rate, are the same with baseline, which is 10^{-4}.

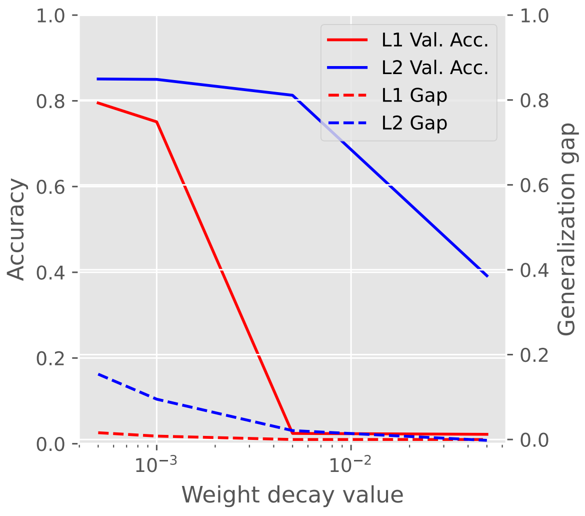
The best results among each method are shown in *italics* in Table 3. Through results of the models which have dropout applied, we can know that inclusion probability inside range 0.85-0.97 could be a promising selection to mitigate overfitting problem for EMNIST dataset since their validation accuracy are higher and the generalization gap are smaller compared with the baseline. Even though the generalization gap of inclusion probability 5×10^{-3} and 5×10^{-2} are small, which can be seen in Figure 4a, their accuracy on unseen data is smaller than the baseline. Similarly, L2 penalty weight 5×10^{-4} and

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
	0.7	82.8	0.447	0.508
	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	75.1	0.841	0.849
	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	85.0	0.361	*0.456*
	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).

1×10^{-3} are also good approaches. On the other hand, L1 penalty make less effects on our model since their best validation accuracy is even smaller the baseline. For L1 penalty, 5×10^{-3}, 5×10^{-2} or even larger wights are not suitable hyperparameters for our dataset. They are large so that the L1 regulation term $\lambda||w||_1$ dominate the cost function, which is formula (5). Under this situation, L1 regularization even reduces the weights of some essential features to 0 and finally leads to underfitting.

The best results over all 12 experiments is shown in bold in Table 3. All the best results appear in dropout. The best validation accuracy 85.4%, which can be seen as the best accuracy on unseen data, appears when the inclusion probability is 0.97. Thus, through selecting model by their performances on the validation set, we can conclude that the model applied dropout with $p = 0.97$ is the best model through all our experiments. Then, we test the best model on the test set and get accuracy 84.4% and error 0.483, which is a better result compared with our baseline validation performance.] .

[Then, we want to figure out what is the effect of combining L1 or L2 regularization with dropout. In particular, we wonder whether the combination of dropout layer and regularization penalty alleviate overfitting by a larger extent. Thus, we intend to run 8 more experiments on EMNIST dataset and compare them with the previous experiments. For this bunch of experiments, we will still set the learning rate to be 10^{-4} and keep the baseline model as the same, which is three hidden layers with 128 neurons. Moreover, we apply two methods on each of the experiment model with different combinations of the hyperparameters used previously, as listed

Model	inclusion probability	penalty weight
dropout & L1	0.85	5e-4
dropout & L1	0.85	1e-3
dropout & L1	0.97	5e-4
dropout & L1	0.97	1e-3
dropout & L2	0.85	5e-4
dropout & L2	0.85	1e-3
dropout & L2	0.97	5e-4
dropout & L2	0.97	1e-3

Table 4. The model used in 8 futher experiments and their corresponding hyperparameter combinations.

in Table 4. We have 4 experiments for applying both dropout and L1 regularizaiton and 4 experiments for dropout and L2 regularization. Their hyperparameter values are the two of best performance in the previous experiments, which is 0.85 and 0.97 for dropout inclusion probability, 5×10^{-4} and 1×10^{-3} for penalty weight.

] .

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 **[L2 weight penalty in Adam optimizer scales down all the weights x with the overall typical magnitudes, which leads to an uneven regularization. In particular, some larger weights are scaled down by a smaller extent compared with the smaller weights. Whereas, weight decay in Adam optimiser normalise each of the weights by the same rate. Under this condition, an approach called preconditioner matrix is applied to make them equivalent. The preconditioner matrix is a diagonal matrix with each value related to the magnitude of each weights, which counteracts the irregular scaling in L2 regularization.]** .

[The model we used implements the Adam learning rule with L2 regularization. L2 regularization is implemented by adding a penalty term to the cost. It also add the weights to the gradient and affects the weights indirectly. However, weight decay optimized the weight updating formula directly, as proposed by (Loshchilov & Hutter, 2019). In MLP library function grads_wrt_params() of class AffineLayer in module layer implements L2 regularization by adding the gradient of L2 penalty to the gradients of weights.] .

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

[Overfitting is a common problem in model training. It occurs when the training performance of a model improves but the validation performance degrades. It usually indicates a lower predicted accuracy on unseen data. A general quantity to indicate overfitting is the generalisation gap, which defines as the difference between the validation error and training error. If the generalization gap steady increases, there is a large probability that the training process enters overfitting.

We trained models with incremental width and depth on EMNIST dataset to figure out the correlation between network capacity and overfitting. Results shows that increasing width and depth could improve the validation performance, but the improvement is limited and larger capacity will finally leads to overfitting, which can be seen from the larger generalization gap.

We also explore some approaches to mitigate overfitting, which is dropout and weight penalty. Dropout is a method that randomly delete some neurons from each hidden layer to reduce the high connection between units of adjacent layers and utilize the sub-network of the whole structure. Weight penalty optimizes the cost function by adding a penalty of L1 or L2 norm of the weights parameter to restrict its magnitude. L1 regularization will filter out some specific features while L2 regularization scale down the weight globally. Both dropout and weight penalty are able to alleviate overfitting with suitable hyperparameters. We also proposed potentials of combining L1 L2 weight penalty. Some experiment schemes are also given in Section 4. The joint impact of these methods could be a direction of future exploration. Finally, a popular paper is reviewed to discover the difference of L2 regularization and weight decay on two learning rules. It is found that they have same effect on SGD, but need a more complicated transformation to act similar on Adam optimiser.] .

[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.

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.