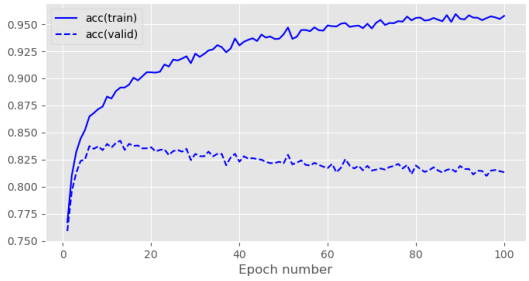
MLP Coursework 1

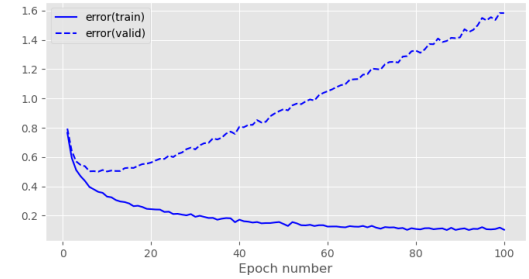
s2344822

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. **[This may result in the model failing to perform well on unseen set and having poor generalization ability.]** 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 tends to cause the problem of 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 **[both Dropout and Weight Penalty contribute greatly to reducing overfitting, and the combination of L2 regularization and Dropout performs best.]** 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 **[wider and deeper networks are more prone to overfitting, and there are many ways to address it, such as Dropout and L1/L2 regularization.]**



(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.

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 **[both Dropout and Weight Penalty contribute greatly to reducing overfitting, and the combination of L2 regularization and Dropout performs best]**. 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 **[wider and deeper networks are more prone to overfitting, and there are many ways to address it, such as Dropout, L1/L2 regularization.]**.

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.

[An overfitted model has poor generalization ability, which means it take into account too much information from training data and has limited applicability on new test datasets. And it may lead to wrong prediction results, so overfitting is indeed a problem that we must solve.]

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.

[Increasing network capacity will increase the risk of overfitting because it may memorize too many properties of training set that not useful on test set and something occurs by accident.]

Fig. 1a and 1b show a prototypical example of overfitting. **[We see in Fig. 1a, the accuracy of the training set continues increasing as the epoch increases, while the accuracy of the valid set increases quickly during the first 10 epochs and then decreases. Finally, there is a huge gap between accuracy on training and valid data. In Fig. 1b, the error of the training set continues to decrease with the increase of epoch number, while the error of the valid set decreases in rapid pace during the first 10 epochs and then decreases. We can conclude that overfitting occurs around 10th epoch. Because the maximum value of the valid set accuracy is around 10 and then decreases, the minimum value of the validation set error is around 10 and then increases. So 10 is the turning point, where overfitting occurs.]**

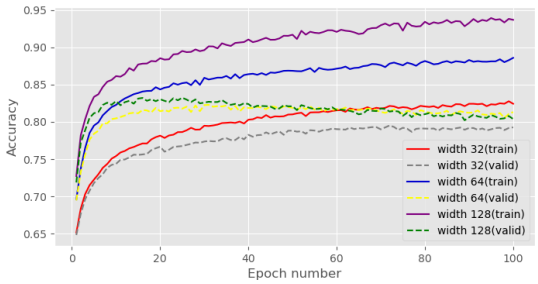
The extent to which our model overfits depends on many factors. For example, the quality and quantity of the training set and the complexity of the model. If we have sufficiently many diverse training samples, or if our model contains few hidden units, it will in general be less prone to overfitting.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

[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.

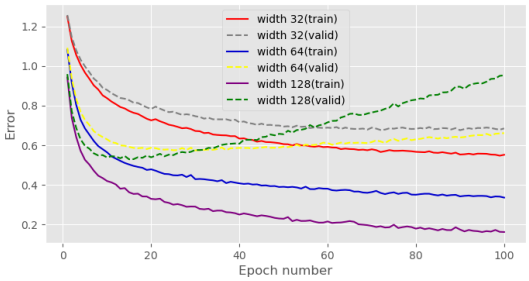
[2]Instructor note: Omitting this for this coursework, but normally you would be more specific and summarise your conclusions about that review here as well.

# Hidden Units	Val. Acc.	Train Error	Val. Error
32	78.3	0.566	0.703
64	80.9	0.338	0.659
128	80.7	0.157	0.928

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.

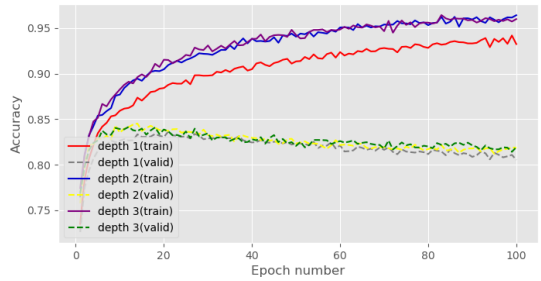
2.1. Network width

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.

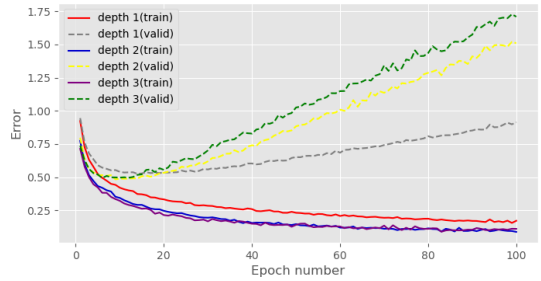
The input layer is of size 784, and output layer consists of 47 units. Three different models were trained, with a single hidden layer of 32, 64 and 128 ReLU hidden units respectively. Fig. 2 depicts the error and accuracy curves over 100 epochs for the model with varying number of hidden units. Table 1 reports the final accuracy, training error, and validation error. We observe that **[in these three experiments, models perform much better on the training set than the valid set after the 10th epoch, so they all face overfitting problems to varying degrees. The model with 128 hidden units has the largest gap between training**

# Hidden Layers	Val. Acc.	Train Error	Val. Error
1	79.3	0.182	0.92
2	81.9	0.088	1.51
3	81.9	0.111	1.72

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.

error and valid error, it performs about the same as the model with 64 hidden units in the training set but performs worst in the valid set, which means it has the worst overfitting problem. On the contrary, the model with 32 hidden units has the smallest gap between training error and valid error, which means it has a less severe overfitting problem. Accordingly, we can find overfitting become increasingly serious as the number of hidden units increase.]

[From our observation, we can find the width affects the results in a consistent way, the wider the network, the more serious the overfitting problem it faces. And this is expected and matches with prior knowledge. The model with more hidden units has more parameters and flexibility, which is more likely to be faced with overfitting.]

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 **[all these three models with different hidden layers are overfitted. It's clear that the model with 3 hidden layers reaches the highest accuracy at the 10th epoch and then decreases slowly. It has the largest gap between training error and valid error, it performs about the same as the model with 2 hidden layers in the training set but performs worst in the valid set, which means it has the worst overfitting problem. Oppositely, the model with 1 hidden layer has the smallest gap between the training error and valid error, so it has a less serious overfitting problem. We find that increasing the number of hidden layers will lead to a more severe overfitting problem.]**

[From our observation, we can find the depth affects the results in a consistent way which is similar to the width, the deeper the network, the more serious the overfitting problem. This is expected and matches with prior knowledge. The model with more hidden layers has more neurons, resulting in more complexity and flexibility, which is prone to become overfitting more easily.]

[From the experiments, we find that increasing the width or depth of the model will increase the performance of the model on the training set, but will reduce the performance on the validation set. They all increase the generalization gap and lead to overfitting because increasing width or depth means increasing model complexity, resulting in overfitting.]

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.

[L1 and L2 regularization can be combined, and the mathematical formula combines the two penalty terms of L1 and L2:

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

In theory, L1 has the property of shirking less important feature coefficients to zero while leaving a few large important weights, encouraging sparse weight space. L2 is less computationally expensive than L1, optimization with L2 leads to a closed form in many cases. So, combining L2 and L1 may achieve both precise calculation and sparsity when the number of features is very large.]

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.

[In experiment applying dropout, we have experimented with hyperparameters 0.6, 0.7, 0.85, 0.9, and the validation accuracy are 0.807, 0.843, 0.851 and 0.854. For the case of hyperparameters 0.7, 0.85, and 0.97, their validation accuracy is higher than the baseline, and the generalization gap is narrower than the baseline. This suggests that dropout can improve generalization performance and reduce overfitting problem. Among all there parameters, model with inclusive prob = 0.97 achieves the best performance on validation set, and the final accuracy on test set is 0.833.]

[In experiment with L1/L2 weight penalty, we set several weight decay value: 5e-4, 1e-3, 5e-3 and 5e-2. For the L1 penalty, we find all results are worse than the baseline, but the generalization gap is much smaller than the baseline. This may be because the penalty term is too large that the models become underfitting. It's noticeable that for the penalty parameter of 5e-2 and 5e-3, the generalization gap is close to zero, which hardly happen for a well-trained model. This may be caused by a large penalty term, and the model has not learned well on the training set. The best-performing model gets a result of 0.796 on the test set.]

[For L2 penalty with hyperparameter 5e-4 and 1e-3, the validation accuracy is better than baseline and the

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	84.3	0.349	0.459
	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	75.6	0.759	0.784
	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	0.365	0.431
	5e-3	81.3	0.586	0.607
	5e-2	39.2	2.258	2.256
Combined	Dropout 0.85, L1 5e-4	78.1	0.705	0.708
	Dropout 0.85, L1 1e-3	71.7	0.976	0.979
	Dropout 0.85, L2 5e-4	85.0	0.384	0.430
	Dropout 0.85, L2 1e-3	83.8	0.438	0.471
	Dropout 0.97, L1 5e-4	80.6	0.583	0.600
	Dropout 0.97, L1 1e-3	76.2	0.785	0.791
	Dropout 0.97, L2 5e-4	**85.9**	0.315	0.402
	Dropout 0.97, L2 1e-3	85.2	0.385	0.440

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.

generalization gap is smaller. So L2 penalty is also a useful method to reduce overfitting. The model with penalty=5e-4 performs best on validation set, and the final accuracy on test set is 0.841.]

[For the combined model, 8 combinations have experimented with different inclusive prob and L1/L2 penalty weights. We select the two parameters that perform the best of the three methods and combined them in pairs to get these eight cases. Generally, combined models perform well in the validation set compared with the baseline. But some cases perform even worse than each experiment, such as Dropout 0.85 L1 5e-4. This may be caused by over-regularisation, which makes the model underfitting. Among all cases, the best model is dropout 0.97 L2 5e-4. The final accuracy on the test set is 0.848.]

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 **[the major difference between L2 regularization and weight decay is while the former modifies the gradients to add $\lambda * \omega$, weight decay does not modify the gradients but instead it subtracts $\lambda * \omega * lr$ from the weights in the update step.]**

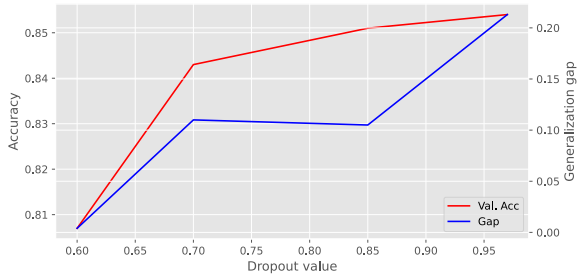
[We implemented AdamW in our experiments, as shown by the function update_params of class AdamLearningRule in mlp/learning_rules.py.]

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

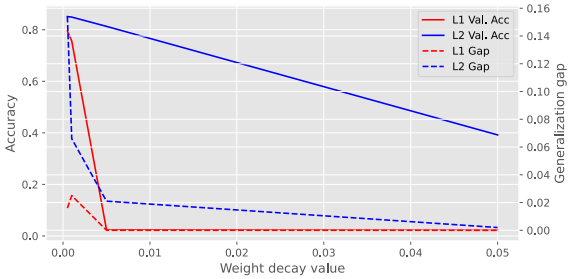
6. Conclusion

[In this report, we study how overfitting occurs with different network settings first. We find that increasing width or depth of the network will lead to a more severe

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



(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).

overfitting problem. To address this problem, we discuss three regularization methods, dropout and L1/L2 regularization. Experiment results show that they are all effective to alleviate the overfitting problem, the combination of dropout and L1/L2 weight penalty has better performance than the individual. We observe that the best performing model is "dropout 0.97, L2 penalty weight 5e-4", and the final accuracy on test set is 0.848. Finally, we briefly review another method, decoupled weight decay regularization, which is proposed in 2019. The author has proved that Weight decay should be used instead of L2 regularization on Adam. Inspired by this paper, we're able to use AdamW or other adaptive gradient methods instead of Adam to alleviate the overfitting problem and improve the model performance, just like what (Loshchilov & Hutter, 2019) did.]

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.