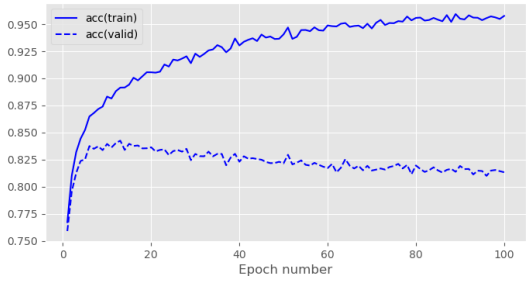
MLP Coursework 1

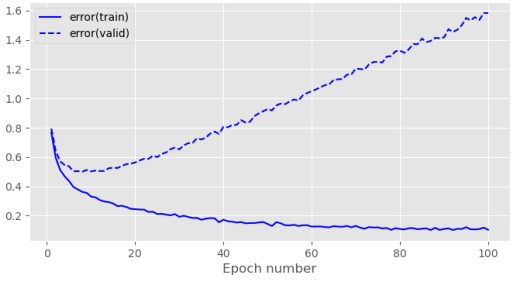
S2617343

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 prevents our trained model from generalizing well to 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 tend to enable further overfitting. Next we discuss how 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 are able to mitigate overfitting. Finally, we conclude the report with our observations and related work. Our main findings indicate that preventing overfitting is achievable through regularization, although combining different methods together is not straightforward.



(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 analysing the given problem in Figure 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 are able to mitigate overfitting.

Finally, we conclude our study in Section 5, noting that preventing overfitting is achievable through regularization, although combining different methods together is not straightforward.

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. Overfitting leads to bad generalization per-

[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.

# Hidden Units	Val. Acc.	Train Error	Val. Error
32	78.9	0.563	0.705
64	80.8	0.337	0.660
128	80.4	0.17	0.907

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

formance in unseen data, as performance on validation data is indicative of performance on test data and (to an extent) during deployment.

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.

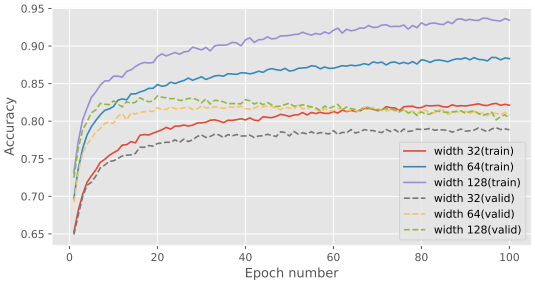
Figure 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that **[the training performance of the model continues to improve, and the curve representing the accuracy remains monotonic on the training set; while the generalization performance of the model becomes very good at the beginning, but after 10 epochs, the generalization performance of the model begins to deteriorate, that is, the model reaches the highest accuracy point on the 10th epoch, and then overfitting occurs, and the gap between training performance and generalization performance becomes larger and larger. Similarly, in Figure 1b, we can also observe a similar overfitting phenomenon, where the model error decreases on the training set and the validation set, and after the 10th epoch, the model error continues to decrease on the training set, and continues to increase after reaching the lowest point on the validation set]** .

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 regularization will also limit the extent to which the model overfits.

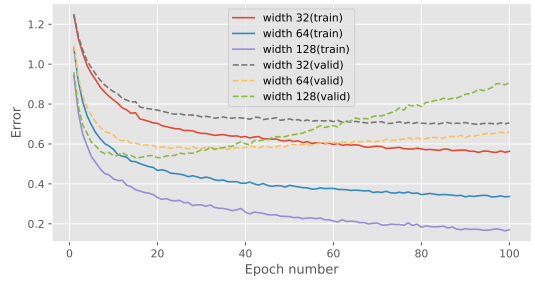
2.1. Network width

[ ] [ ]

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.



(a) accuracy by epoch



(b) error by epoch

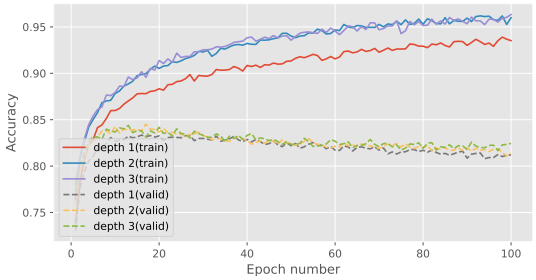
Figure 2. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for different network widths.

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. Figure 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 model width increases, the generalization ability of the model will increase, but if the model width is increased indefinitely, overfitting will occur. From Table 1, we can see that after the number of neurons in the hidden layer of the model increases from 32 to 64, the performance of the model on the validation set improves, but after the number of neurons increases from 64 to 128, although the performance of the model on the training set becomes better, the performance on the validation set becomes worse. From Figure 2, we can see more intuitively that the performance of the model with a width of 32 continues to improve, the model with a width of 64 overfits after the 15th epoch, and the model with a width of 128 overfits after the 10th epoch]** .

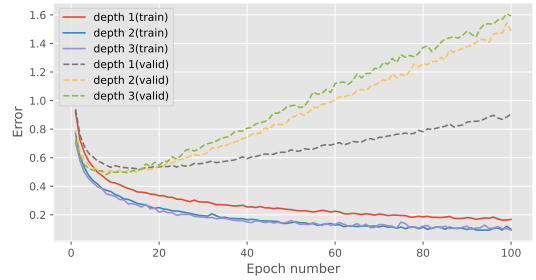
[From the above results, we can see that blindly increasing the model width cannot continuously enhance the generalization ability of the model. Instead, it will cause the model to fall into an overfitting state. This conclusion is consistent with our machine learning theory. When the amount of data remains unchanged, the increase in model capacity will lead to the model's ability to memorize training data during the training process, which may cause the model to begin to capture the noise in the data instead of capturing the essential distribution

# Hidden Layers	Val. Acc.	Train Error	Val. Error
1	81.2	0.168	0.905
2	81.4	0.098	1.488
3	82.4	0.093	1.592

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.

of the data] .

2.2. Network depth

[ ] [ ]

Next we investigate the effect of varying the number of hidden layers in the network. Table 2 and Figure 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 **[as the depth of the model increases, the generalization ability of the model continues to deteriorate, and overfitting occurs. From Table 2, we can see that as the number of hidden layers of the model increases from 1 to 2 and then to 3, although the performance of the model on the training set improves, the performance on the validation set deteriorates. From Figure 3, we can see more intuitively that all three models enter the overfitting state around the 10th epoch, and the models with 2 and 3 hidden layers overfit faster than the model with only 1 hidden layer]** .

[Similar to the conclusion of network width, blindly increasing the model depth does not continuously enhance the generalization ability of the model, but instead causes the model to fall into an overfitting state. The increase in model depth leads to an increase in model capacity, which also causes the model to try to memorize training data instead of capturing the essential distribution of the data] .

3. Regularization

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

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 behaviour 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.

3.3. Label smoothing

Label smoothing regularizes a model based on a softmax with K output values by replacing the hard target 0 labels and 1 labels with $\frac{\alpha}{K-1}$ and $1-\alpha$ respectively. α is typically set to a small number such as 0.1.

$$\begin{cases} \frac{\alpha}{K-1}, & \text{if} \quad t_k = 0 \\ 1-\alpha, & \text{if} \quad t_k = 1 \end{cases} \tag{7}$$

The standard cross-entropy error is typically used with these *soft* targets to train the neural network. Hence, implementing label smoothing requires only modifying the targets of training set. This strategy may prevent a neural network to obtain very large weights by discouraging very high output values.

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.

Here, we apply the L1 or L2 regularization with dropout to our baseline and search for good hyperparameters on the validation set. Then, we apply the label smoothing with $\alpha = 0.1$ to our baseline. We summarize all the experimental results in Table 3. For each method except the label smoothing, we plot the relationship between generalization 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 the experiment of each method, we control the hyperparameters of the baseline unchanged, and only set the hyperparameters of one of the methods (Dropout, L1 Penalty, L2 Penalty, label smoothing). Through multiple experiments, the hyperparameters of the method are observed to affect the model performance while keeping the hyperparameters of other models unchanged. From the results in Table 3 and Figure 4, we can see that as the probability p value of the dropout method decreases, the capacity of the model also decreases, so the generalization performance of the model is alleviated at first, but then it deteriorates and enters the underfitting state in Figure 4a. The L1 penalty performs the same as the dropout in Figure 4b. As the coefficient of the L2 penalty increases, the generalization effect is worse than the baseline, and the overfitting state is not alleviated. The label smoothing method also alleviates the overfitting state of the baseline. From Table 3, we can see that the model with the dropout layer and p set to 0.85 has the best generalization performance, because its validation set error is the lowest at 0.434, and the gap between the validation set and the training set is the smallest] .

[From Table 3, we can see that both Dropout and L2 penalty have a certain effect on alleviating overfitting of the baseline. Therefore, we may want to try to combine dropout and L2 penalty in the next experiment to test whether the model can further improve performance. At the same time, we also want to know whether we can choose appropriate hyperparameters for L1 penalty to improve model performance. Therefore, we will set up 8 groups of experiments to try to answer the above questions, where p denotes the probability in Dropout Layer, and λ denotes the coefficient in Penalty. For L2 penalty: $(p, \lambda) = \{(0.85, 5e-4), (0.85, 1e-3), (0.97, 5e-4), (0.97, 1e-3)\}$**, for L1 penalty:** $(p, \lambda) = \{(0.85, 1e-4), (0.85, 1e-5), (0.97, 1e-4), (0.97, 1e-5)\}$**. At the same time, we keep other model hyperparameters unchanged,**

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	83.2	0.444	0.504
	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
Label smoothing	0.1	85.4	1.017	1.152

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

such as batchsize of 100**, learning rate of** $1e-4$**, number of epochs of** 100**, the network has** 3 **hidden layers, and the number of neurons in each layer is** 128**]** .

5. Conclusion

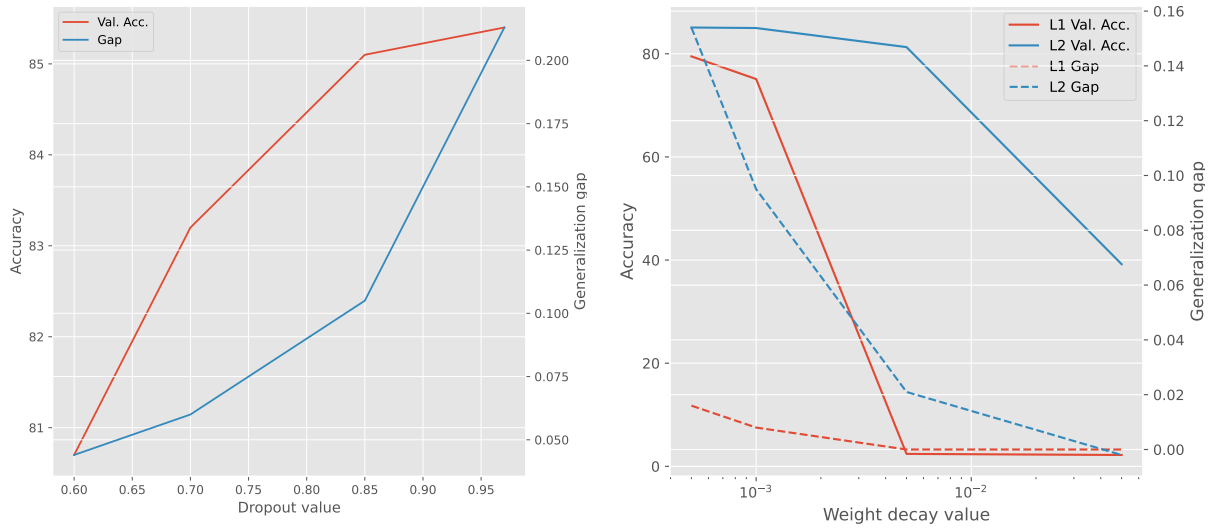
[In this report, we mainly study the overfitting in machine learning theory through the neural network structure and the setting of regularization methods. By adjusting the width and depth of the neural network, we concluded that increasing the width and depth of the neural network can enhance the generalization performance of the model to a certain extent, but it may also lead to overfitting of the model. Therefore, choosing the right width and depth can achieve high performance while avoiding overfitting. After that, we introduced three regularization methods, namely Dropout, L1/L2 penalty and label smoothing. We conducted experiments on each regularization method and found that adjusting the hyperparameters of these methods can alleviate the overfitting of the model to a certain extent, but choosing the right hyperparameters is necessary. Finally, we also tried to discuss the combination of different regularization methods to see if it can improve the performance of the model. Trying to find the optimal combination configuration and hyperparameters for improving model performance may be a possible research direction in the future, which may be better than using a single regularization method] .

References

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

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.



(a) Accuracy and error by inclusion probability.

(b) Accuracy and error by weight penalty.

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