MLP Coursework 1

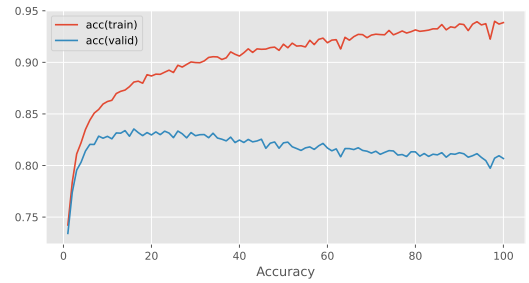
s2177402

Abstract

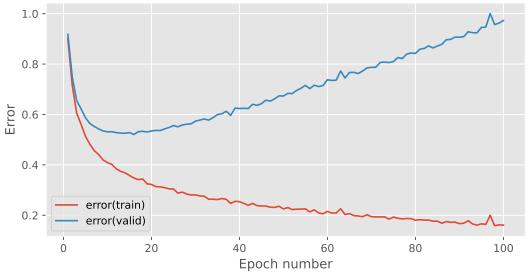
In this report we study the problem of overfitting, which is **[Overfitting is making prediction too strict in order to produce consistent predictions in a particular data set, so that it can not predict data well on test sets or other data sets.]** . We first analyse the given example and discuss the 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 **[With the increase of the width and depth of the architecture, the generalization gap increases and the overfitting becomes more obvious.]** . 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 **[Question 3 - Summarise what your results show you about the effect of the tested approaches on overfitting and the performance of the trained model]** . Finally, we briefly review another method, Maxout, discuss its strengths and weaknesses, and conclude the report with our observations and future work. Our main findings indicate that **[Question 4 - Give your overall conclusions]** .

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 **[Overfitting is making prediction too strict in order to produce consistent predictions in a particular data set, so that it can not predict data well on test sets or other data sets.]** . 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 **?**) and generalization performance. Section 3 introduces two regularization techniques to alleviate overfitting: Dropout (**?**) and L1/L2 Weight Penalties (see Section 7.1 in **?**). 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 neural network, train it on the EMNIST dataset, which contains 131,600 images of characters and digits, each of size 28x28 which are split into 47 classes, grouping together some difficult to distinguish characters. 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 **[Question 3 - Summarise what your results show you about the effect of the tested approaches on overfitting and the performance of the trained model]** . In Section 5, we discuss a related work on Maxout Networks and highlight its pros and cons.[1] Finally, we conclude our study in section 6, noting that **[Question 4 - Give your overall conclusions]** .



(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.

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 **?**). A model is said to be overfitting when **[Over-**

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

fitting is a phenomenon that model has too good results on the training set and not so good results on the test set. For example with the progress of learning epochs, the accuracy rate decreases or does not increase like that on training set and generalization gap increases sharply especially on test set.] .

[Overfitting occurs for a number of reasons. First, the modeling sample set is incorrectly selected. For example, the number of sample sets is too small, leading to the sample data is not enough to represent the predetermined classification. Second, the noise contained in the sample set is too large, leading to the machine to regard part of the noise as valid data and thus interfere with the classification. Thirdly, the model has too many parameters and too much complexity (too much width and depth). In general, when the error rate of the model in the validation set is not significantly reduced or even increases with the number of learning rounds, or the generalization gap is extremely large, we can think that overfitting has occurred. From a practical point of view, the most important point is the classification accuracy, and when the classification accuracy on the test set no longer increases with that of the training set, it can be considered that there is an overfitting.] .

Fig. 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that [These two figures reflect the performance of classification accuracy and cross entropy errors in training set and verification set. As you can see from these two graphs, both of them are steadily rising or falling in the training set. However, in the validation set, after the vertex of the image curve (the point with slope 0), the two develop in reverse. The cross entropy error shows an upward trend around the 15th epoch, which can be considered as a sign of overfitting. But from a practical point of view, what we really care about is improving the classification accuracy on the validation data, so we think that the vertex of the classification accuracy curve on the validation set is the point where overfitting begins to dominate learning.] .

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 a lot of varied training samples, or if our model is relatively shallow, it will in general be less prone to overfitting. Any form of regularisation will also limit the extent to which the model overfits.

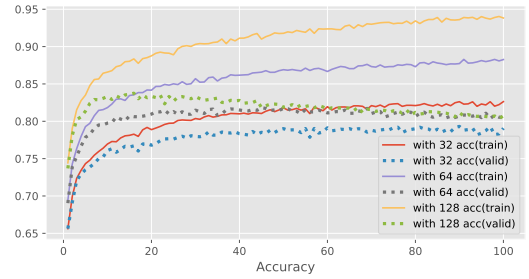
2.1. Network width

[Question Table 1 - Fill in Table 1 with the results from your experiments varying the number of hidden units.] [Question Figure 2 - Replace the images in Figure 2 with figures depicting the accuracy and error, training and validation curves for your experiments varying the number of hidden units.]

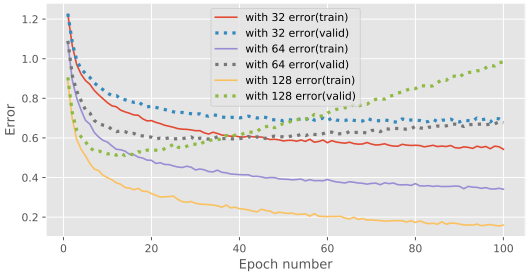
First we investigate the effect of increasing the number of hidden units in a single hidden layer network when training

# hidden units	val. acc.	generalization gap
32	0.788	0.148
64	0.805	0.344
128	0.807	0.811

Table 1. Validation accuracy (%) and generalization gap (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.

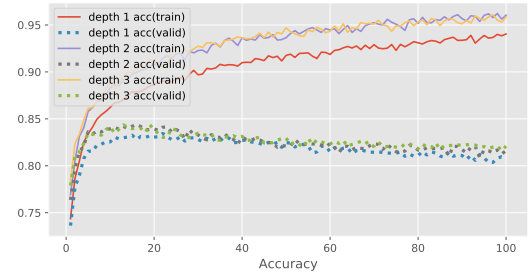
on the EMNIST dataset. The network is trained using the Adam optimizer with a learning rate of 10^{-3} and a batch size of 100, for a total of 100 epochs.

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. 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 and generalization gap. We observe that [From Table 1 and Figure 2, we can see that with the increase of model width, the accuracy rate of prediction increases continuously in the training set. But the results on the validation set were roughly the same (around 0.8). As the width of the model increases, the cross entropy error decreases in the training set, but the generalization gap increases (from 0.148 to 0.811).At the same time, around the 20th epoch, the models with width 64 and 128 appeared overfitting (the accuracy on validation set stops increasing and the errors in the validation set increased rather than decreased).] .

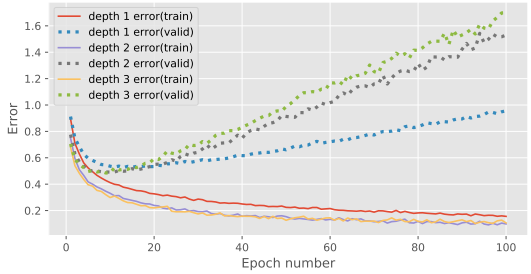
[From the images and data, it can be seen that the model

# hidden layers	val. acc.	generalization gap
1	0.809	0.800
2	0.816	1.432
3	0.821	1.594

Table 2. Validation accuracy (%) and generalization gap (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.

with width 32 is underfitting (the result on the verification set is much lower than the result on the training set), the model with width 64 and 128 is overfitting, and the model with width 128 is overfitting more seriously. To some extent, this confirms the previous explanation of the cause of overfitting and the possibility of overfitting is higher when the model complexity is too high.] .

2.2. Network depth

[Question Table 2 - Fill in Table 2 with the results from your experiments varying the number of hidden layers.] [Question Figure 3 - Replace these images with figures depicting the accuracy and error, training and validation curves for your experiments varying the number of hidden layers.]

Next we investigate the effect of varying the number of hidden layers in the network. Table 2 and Fig. 3 shows the 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 10^{-3} and a batch size of 100.

We observe that **[On the training set, the classification accuracy becomes higher and the errors become lower as the depth of the model increases. However, in the validation set, the classification accuracy of the three models is roughly the same (only slightly improved with the increase of depth), and the error and generalization gaps become larger with the increase of depth.]** .

[The effects of different depths on the results are consistent. As the depth of the model increases, the classification accuracy becomes higher in both sets, even if it is very small in the validation set. On the validation set, error and generalization gaps become larger with increasing depth. This is consistent with the previous description of over-complex models leading to overfitting. As can be seen from the generalization gap, the overfitting of the model becomes more serious with the increase of depth.] .

[With the increase of model complexity (width and depth), the performance of the model increases little, but the degree of overfitting increases significantly.] .

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 (**?**) is a stochastic method that randomly inactivates neurons in a neural network according to an hyperparameter, the inclusion rate. Dropout is commonly represented by an additional layer inserted between the linear layer and activation function. Its forward propagation during training is defined as follows:

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

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

where $y, y' \in \mathbb{R}^d$ 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 parameter of 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 rate p:

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

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:

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

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 features between layers so that the neurons of the next layer will not specifically depend on some features from of the previous layer. Instead, it force the network to 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 (**?**) are simple but effective methods to mitigate overfitting to training data. **[In machine learning, the loss function is usually followed by an extra term. There are two common extra terms, L1-norm and L2-norm. These two can be viewed as the penalty terms of the loss function.**

L1-norm refers to the sum of the absolute values of each element in the weight vector W, usually expressed as

$$\|\boldsymbol{v}\|_1 = \sum_{d=1}^{D} |v_d| \,, \tag{5}$$

To weigh the penalty term, add a positive scalar coefficient, β_i.

$$C_{\mathbf{L1}}^{(i)} = \beta_i \left\|\boldsymbol{p}^{(i)}\right\|_1 = \beta_i \sum_{d=1}^{D} \left|p_d^{(i)}\right| . \tag{6}$$

The gradient of the parameter vector is

$$\frac{\partial C_{\mathbf{L1}}^{(i)}}{\partial p_d^{(i)}} = \beta_i \, \mathbf{sgn}\left(p_d^{(i)}\right) \tag{7}$$

L2-norm refers to the sum square root of the squares of each element in the weight vector W, and is defined as.

$$\|\boldsymbol{v}\|_2 = \left[\sum_{d=1}^{D} \left(v_d^2\right)\right]^{\frac{1}{2}} . \tag{8}$$

Similarly, to weigh the penalty term, add a positive scalar coefficient, β_i.

$$C_{\mathbf{L2}}^{(i)} = \frac{1}{2}\beta_i \left\|\boldsymbol{p}^{(i)}\right\|_2^2 = \frac{1}{2}\beta_i \sum_{d=1}^{D} \left[\left(p_d^{(i)}\right)^2\right] . \tag{9}$$

And notice that $\frac{1}{2}$ isn't necessary here, we're just trying to figure out the gradient later, so that $\frac{1}{2}$ and 2 can multiply and cancel out to 1.

$$\frac{\partial C_{\mathbf{L2}}^{(i)}}{\partial p_d^{(i)}} = \beta_i p_d^{(i)} \tag{10}$$

L1 and L2 penalties are very easy to implement. It can be implemented in an AffineLayer, adding the gradient of the penalty term to the existing gradient and returning the penalty term of the layer dependent parameter. The gradient of the penalty term and the penalty term themselves can be calculated by the function according to the formula. The function parameter which can be adjusted freely is the coefficient before the penalty term.] .

[During the fitting process, it is generally preferred to keep the weights as small as possible and construct a model with relatively small parameters. Because it is generally considered that the model with smaller parameter value is relatively simple and can avoid the over-fitting phenomenon to a certain extent. In the gradient descent algorithm, L2-norm iteratively multiplizes a factor less than 1 to make the result progressively smaller, resulting in a smaller parameter (called weight decay). Unlike L2-norm, L1-norm can generate a sparse weight matrix for feature selection. Many elements in the sparse matrix are 0, and only a few elements are non-zero, that is, the coefficients of weights. The end result is that L1-norm tends to focus its weight on high-importance connections, while others have a weight of 0. This also reduces the complexity of the model, so it can prevent overfitting to a certain extent.] .

4. Balanced EMNIST Experiments

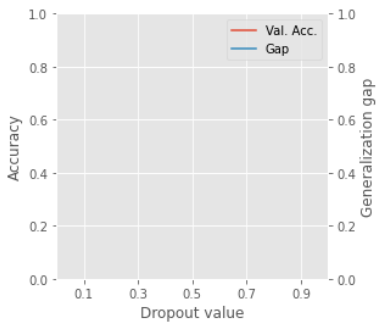
[Question Table 3 - Fill in Table 3 with the results from your experiments varying the hyperparameter values for each of L1 regularisation, L2 regularisation, and Dropout (use the values shown on the table) as well as the results for your experiments combining L1/L2 and Dropout (you will have to pick what combinations of hyperparameter values to test for the combined experiments; each of the combined experiments will need to use Dropout and either L1 or L2 regularisation; run an experiment for each of 8 different combinations). Use *italics* to print the best result per criterion for each set of experiments, and bold for the overall best result per criterion.]

[Question Figure 4 - Replace these images with figures depicting the Validation Accuracy and Generalisation Gap for each of your experiments varying the Dropout inclusion rate, L1/L2 weight penalty, and for the 8 combined experiments (you will have to find a way to best display this information in one subfigure).]

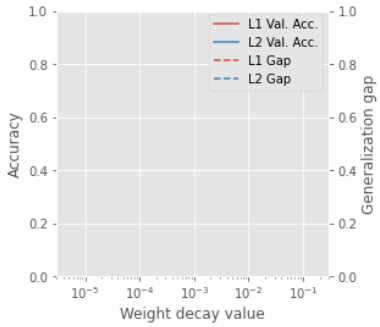
Here we evaluate the effectiveness of the given regularization methods for reducing the overfitting on the EMNIST dataset. We build a baseline architecture with three hidden layers, each with 128 neurons, which suffers from overfit-

Model	Hyperparameter value(s)	Validation accuracy	Generalization gap
Baseline	-	0.836	0.290
Dropout	0.7 0.9 0.95		
L1 penalty	1e-4 1e-3 1e-1		
L2 penalty	1e-4 1e-3 1e-1		
Combined	for example 0.95, L1 1e-6 ?, ? ?, ? ?, ? ?, ? ?, ?		

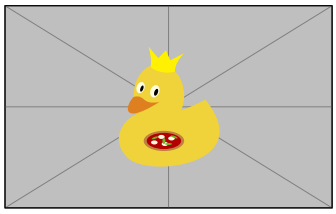
Table 3. Results of all hyperparameter search experiments. *italics* indicate the best results per series and **bold** indicate the best overall



(a) Metrics by inclusion rate



(b) Metrics by weight penalty



(c) Extra experiments

Figure 4. Hyperparameter search for every method and combinations

ting in EMNIST as shown in section 2.

We start by lowering the learning rate to 10^{-4}, as the previous runs were overfitting in only a handful of epochs. Results for the new baseline (c.f. Table 3) confirm that lowering 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.

[Question 15 - Explain the experimental details (e.g. hyperparameters), discuss the results in terms of their generalization performance and overfitting] .

5. Literature Review: Maxout Networks

Summary of Maxout Networks In this section, we briefly discuss another generalization method: Maxout networks (**?**). This paper further ex- plores the dropout method and proposes a new "maxout" layer which can complement dropout. The authors evaluate the performance of Maxout Networks in four standard datasets, namely MNIST, CIFAR-10 and 100, and SVHN. They point out that although dropout has been widely applied in deep models, **[Question 16 - Explain the motivation behind Maxout Networks as presented in (?)]** . Following this motivation, they propose the Maxout activation layers. These can be considered learnable activations that work as a universal convex function approximator. The Maxout layer first maps the hidden space to k subspaces through independent affine transformations, then, for each element in output vectors, it takes the maximum value across all subspaces.

[Question 17 - State whether Dropout is compatible (can be used together) with Maxout and explain why] .

Strengths and limitations The author proposed a novel neural activation unit that further exploits the dropout technique. **[Question 18 - Give an overview of the experiment setup in (?) and analyse it from the point of view of how convincing their conclusions are]** .

Although the Maxout activation units can maximize the averaging effect of dropout in a deep architecture, we can argue that the Maxout computation is expensive. The advantage of dropout lies in its high scalability and computational advantages. It can be arbitrarily applied to various network structures, and the calculation speed is fast, which is very suitable for heavy computing algorithms such as training and inference of neural networks. In comparison, the design of the Maxout network needs to project the hidden vector into k subspaces. Both the forward algorithm and the backward algorithm of dropout can be calculated in $O(D)$ complexity, but the complexity of Maxout is $O(kD)$. This can lead to increasing the number of training epochs needed to reach convergence. Furthermore, the universal approximation property of Maxout seems powerful, but it would be interesting to verify that it is useful in practice. Specifically, we can design an experiment where we increase the number of subspaces k and see where performances stop improving. In extreme cases, it is even possible that the function learned is too specific to the training data, effectively causing overfitting.

6. Conclusion

[Question 19 - Briefly draw your conclusions based on the results from the previous sections (what are the take-away messages?) and conclude your report with a recommendation for future directions] .