MLP Coursework 1

s1869292

Abstract

In this report we study the problem of overfitting, which is **[when a model fits the data in the training set well, while incurring larger generalisation error. In practice this often realised by a large gap between the training and test error.]** . 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 **[(in the absence of countermeasures) systematically worsens the problem of overfitting, impairing model performance.]**. 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 **[a combination of Dropout and L2 weight penalty have the best performance in the dataset, however both Dropout and Weight penalty perform almost as well by themselves]**. 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 **[Dropout and weight decay are simple and cheap methods to improve performance and reduce overfitting on the EMNIST dataset.]**.

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 **[when a model fits the data in the training set well, while incurring larger generalisation error. In practice this often realised by a large gap between the training and test error.]** . 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 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 **[a combination of Dropout and L2 weight penalty have the best performance in the dataset, however both Dropout and Weight penalty perform almost as well by themselves]**. 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 **[Dropout and weight decay are simple and cheap methods to improve performance and reduce overfitting on the EMNIST dataset.]**.

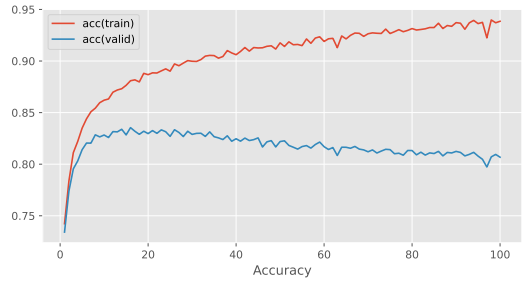
2. Problem identification

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 **[it fits the data in the training set well, while incurring larger generalisation error, meaning the model will not perform accurately on unseen data. This is a principle issue in machine learning as performance on unseen data is the fundamental test of an ML model and what separates it from a pure information-complete optimisation problem]**.

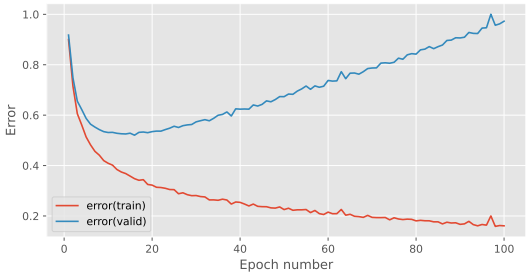
[Overfitting occurs due to the presence of random variation, noise, and the limited size of the training data. These factors will naturally lead to a model of sufficient size evolving to use and depend on this random, non-(externally-)predictive information. Overfitting is prototypically identified by stagnating or regressing performance on unseen data as a model continues to improve on previously seen data.].

Fig. 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that **[initially accuracy on both the training validation data improve rapidly, however they quickly diverge. performance on the training set continues to improve (logarithmically) meanwhile the validation accuracy quickly peaks (at approx. epoch=15) and**

[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.

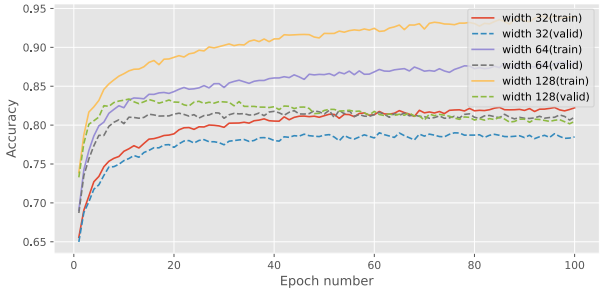


(a) accuracy by epoch

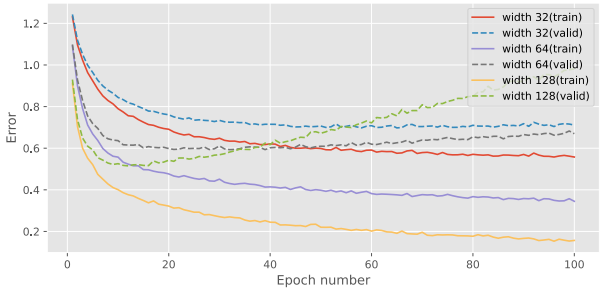


(b) error by epoch

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) 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.

# hidden units	val. acc.	generalization gap
32	0.782	0.045
64	0.810	0.053
128	0.807	0.141

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

then begins over fitting; with a brief plateau followed by a steadily decline to a final *generalisation gap*[2] of 0.13. Similar results are shown in 1b, this time with a more extreme final generalisation gap of ≈ 0.80.].

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

[ ] [ ]

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 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 **[the generalisation gap increases as you increase the width (size) of the model (Table 1). We can see how this gap evolves in Figures 2a and 2b, where overfitting can be seen visibly– beginning where the performance on the validation data (dotted line) stops improving or starts regressing, whilst performance on the training data (solid) continues to improve its fit.]**.

[From analysing Figures 2a and 2b, we observe that as the model width increases, this systematically worsens the problem of overfitting, as is considerably worse in the largest model than the smaller two. Of particular note is the error graph of the model with a width of 128, having by far the most extreme performance drop-off on the unseen data, strongly supporting our experimental hypothesis that increasing model size increases overfitting (in line with the reviewed theory and literature (Ch. 5 in Goodfellow et al. 2016))].

2.2. Network depth

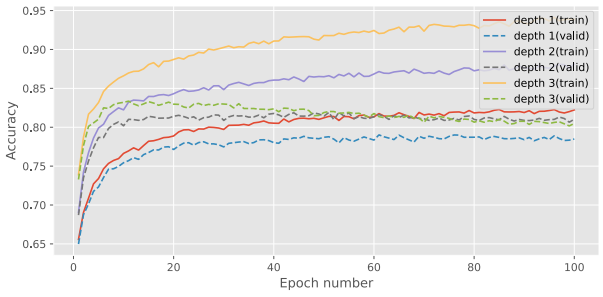
[ ] [ ]

Next we investigate the effect of varying the number of

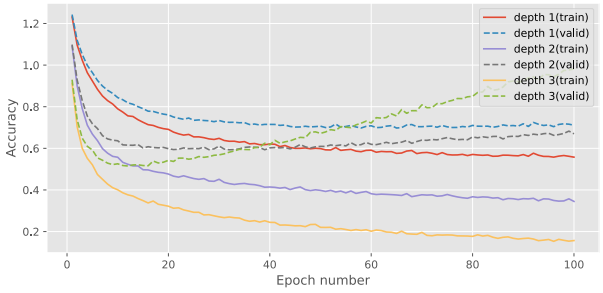
[2]The generalisation gap is the models final training accuracy/error minus its final validation accuracy/error

# hidden layers	val. acc.	generalization gap
1	0.789	0.043
2	0.815	0.071
3	0.803	0.144

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.

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 **[similar to increasing the width of the network, we can see significant overfitting in every model. Adding hidden layers to the network, increased the generalisation gap and worsened overfitting.]**.

[Overfitting was considerably more pronounced in the largest model (3 hidden layers), again mirroring the results of the network width experiment. This is in line with the theory as both increase the number of free parameters in the model.].

[Overall our experimental results on varying the width and depth of neural networks strongly support the literature, suggesting that vanilla neural networks are prone to overfitting and that increasing the width or depth of such models will worsen overfitting. The functional relationship here seems somewhat complex but in our limited scope, the problem of overfitting seems to worsen gradually then more rapidly as you increase the size of the model. To test further, we would want to run more experiments with a greater range of model input parameters.].

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. 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 (Ng, 2004) are simple but effective methods to mitigate overfitting to training data. **[Both L1 and L2 weight penalties are a form of *regularisation* ??. Regularisation is a countermeasure to overfitting that generally involves imposing some sort of smoothness constraint on a learned model. L1/L2 regularisation works by a penalty term that encourages the sum/squared-sum of the (absolute) values of the weights to be smaller, resulting in weights that shrink at a small rate towards 0, determined by hyperparameter λ.**

In both L1 and L2 regularisation, forward propagation remains identical whilst the error function has a penalty term added scaled by a constant factor as shown:

$$E_{L_1} = E(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i| \tag{5}$$

$$E_{L_2} = E(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2 \tag{6}$$

where $E(y, \hat{y})$ is an arbitrary error function, and w is a parameter vector of size N. The scaling factor, λ, is the single manually tuned hyperparameter, controlling the size of the penalty. It typically takes values from 0.1 to 1^{-5}].

[A intuitive way to think about the difference between L1 and L2 regularisation is that in L1 regularisation weights shrink at a constant rate towards zero so will tend to shrink a lot of weights to zero. L2 regularisation puts a "spring" on weights that tends to zero. If a connection has consistent force, it should outweigh the L2 decay. Both L1 and L2 regularisation can be thought of as encouraging only the most important features in the network (Ng, 2004). This is an obvious countermeasure to overfitting as tunes the model to ignore weakly predictive inputs in favour of a more simple, sparse model.].

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 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.

[Next we used these preliminary results to pick the most promising hyperparameters for combining the methods, running six further experiments.

Overall all three methods tested improved performance. The highest single-method accuracy score was 0.855 (its generalisation gap was 0.035), from $L2 = 1e-3$ whilst Dropout and L1 managed accuracies 0.853 and 0.851 respectively.

Clearly looking at Table 3, there are some very poorly performing models, most notably $L1 = 1e-1$. Looking at the training graph (not shown) its confirmed that that this model failed to converge at all. One probable reason for this is that the penalty hyperparameter is set to high and overpowers the model. A similar but less extreme story occurs $L2 = 1e-1$. There's clearly a trade-off here between accuracy and overfitting when increasing penalty size, as shown in Fig. 4b.

When combining the methods, I immediately discounted the underperforming high penalty models discussed earlier, then tried to mix the remaining value all of which I thought were viable hyperparameter values, with a bias to the best accuracy performers.

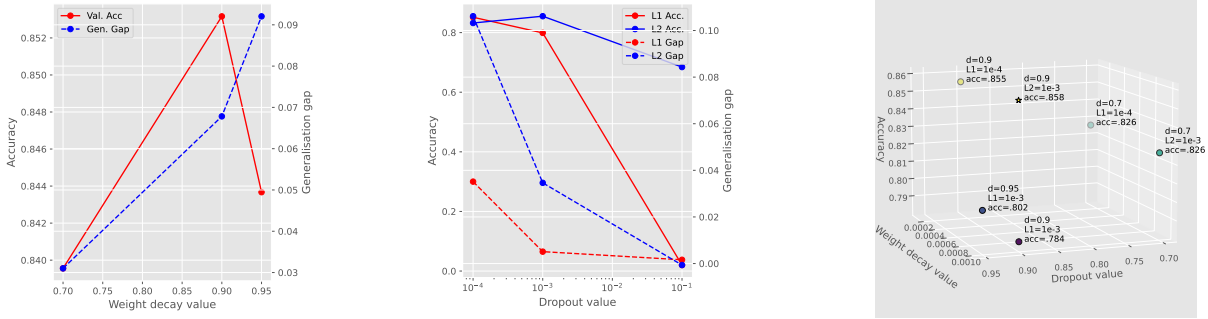
The chosen hyperparameter values can be seen in Table 3. All the results are shown in Fig. 4. The best overall performance found was with dropout=0.9 and L2=1e-3, yielding an validation set accuracy of 0.858 and an test set accuracy of 0.848 This was a direct combination of the best performing individual models.].

5. Literature Review: Maxout Networks

Summary of Maxout Networks In this section, we briefly discuss another generalization method: Maxout networks (Goodfellow et al., 2013). This paper further explores 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, **[it had not yet been proven to truly be performing model averaging, specifically for deep architectures. However, it turns out this was confirmed by (Srivastava et al., 2014) before the paper was published. Additionally, they felt dropout was a indiscriminate "blunt" tool that could be better tuned to the problem instance.]**. Following this motivation, they propose the Maxout activation layers. These can be considered learnable activations that work as a universal convex func-

Model	Hyperparameter value(s)	Validation accuracy	Generalisation gap
Baseline	-	0.836	0.290
Dropout	0.7	0.840	*0.031*
	0.9	*0.853*	0.068
	0.95	0.844	0.092
L1 penalty	1e-4	*0.851*	0.035
	1e-3	0.799	0.550
	1e-1	0.020	*0.002*
L2 penalty	1e-4	0.832	0.106
	1e-3	*0.855*	0.035
	1e-1	0.648	**-0.01**
Combined	0.7, L1 1e-4	0.826	0.006
	0.9, L1 1e-4	0.855	0.014
	0.9, L2 1e-3	**0.858**	0.012
	0.7, L2 1e-3	0.826	0.004
	0.9, L1 1e-3	0.784	0.003
	0.95, L1 1e-3	0.802	*0.002*

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

tion 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.

[Maxout can be used in combination with dropout and is designed to complement it. This is because maxout is simply a type of learned activation function.].

Strengths and limitations The author proposed a novel neural activation unit that further exploits the dropout technique. **[- tested on 5 benchmarks data sets - Only the best if you dont count unsupervised pre-training/clustering/pre-regularisation]**.

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

[- Simpler models are less likely to over-fit than complex ones.] .

References

Goodfellow, Ian, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In *International conference on machine learning*, pp. 1319–1327. PMLR, 2013.

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.