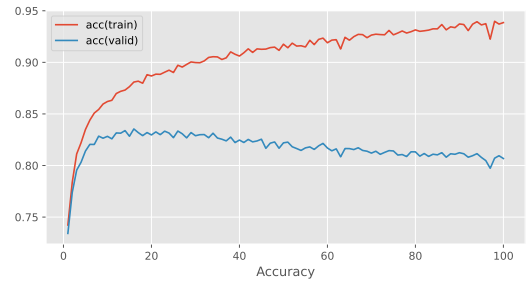
MLP Coursework 1

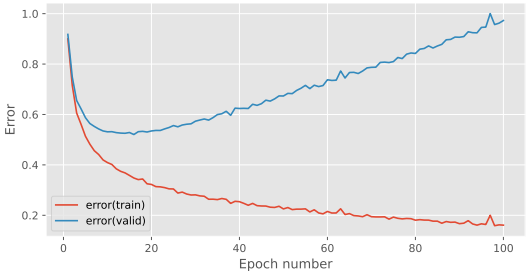
sXXXXXXX

Abstract

In this report we study the problem of overfitting, which is **[the training regime where performances increase on the training set but decrease on unseen data]** . We first analyse the given example and discuss the probable causes of the underlying problem. Then we investigate how the depth and width of a neural network can affect overfitting in a feedforward architecture and observe that increasing width and depth **[enable further 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 are able to mitigate overfitting]** . 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 **[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 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 **[both dropout and weight penalty are able to mitigate overfitting]** . 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 **[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 performances on the training data keep improving, while they are worsening on validation data. Effectively, the model stops learning useful patterns to recognize data and instead starts to memorize specificities of the train-**

[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.

# hidden units	val. acc.	generalization gap
32	0.794	0.137
64	0.803	0.339
128	0.804	0.814

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

ing samples that do not apply to new samples.] .

[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 parameter the model has, the easier it will be to memorize complex data pattern 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] .

Fig. 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that **[although the accuracy keeps increasing throughout on the training set, it reaches its maximum around epoch 15 on the validation set before decreasing. Similarly, figure 1b shows the same issue, with the generalization gap uncontrollably growing after epoch 15.]** .

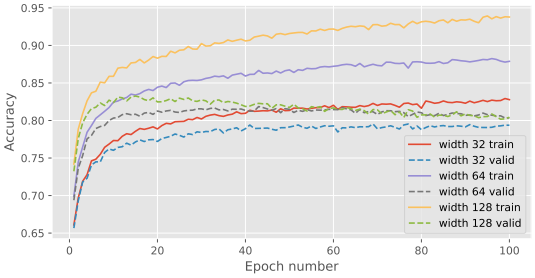
The extent to which our model overfits depends on many factors. For example, the quality and quantity of the training set and the complexity of the model. If we have a lot of varied training samples, or if our model is relatively shallow, it will in general be less prone to overfitting. Any form of regularisation will also limit the extent to which the model overfits.

2.1. Network width

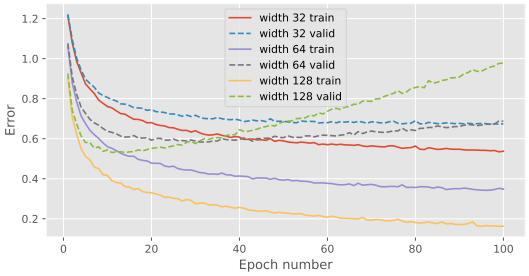
[ ] [ ]

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 **[networks with width 64 and 128 are clearly overfitting, with validation accuracy decreasing and generalization gap increasing when the last epoch is reached. Table 1 contains final validation accuracy and generalization gap for each run, and shows that**



(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 layers	val. acc.	generalization gap
1	0.805	0.807
2	0.815	1.532
3	0.814	1.367

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

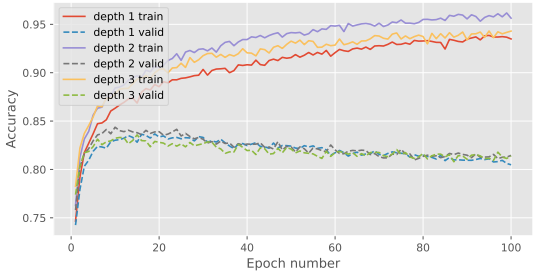
while accuracy is very slightly growing throughout experiments, the generalization gap is rocketing as width increases. In comparison with larger widths, the network with 32 neurons appears to be underfitting, as its final accuracy does not even match that of the larger widths, even though these are shrinking.] .

[Increasing the width allowed to observe a transition from under to overfitting, as expected by an increase in network parameters. the model with 64 could be argued to only be marginally overfitting and could therefor be considered the best of the three, although its final accuracy is lower than the 128-neuron model.] .

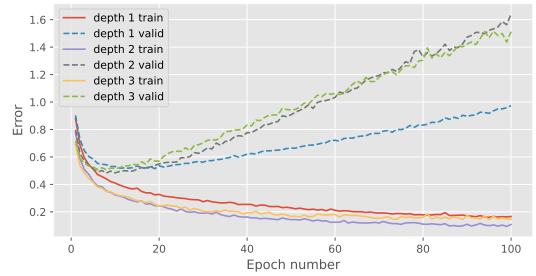
2.2. Network depth

[ ] [ ]

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



(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.

size of 100.

We observe that **[increasing network depth has a strong effect on overfitting: figure 3a shows that training accuracy increases with depth while validation stays constant when depth increases, and 3b illustrates a growing generalization gap. Table 2 reports the same phenomenon, although there is an apparent contradiction with 1% increase in accuracy for models with 2 and 3 hidden layers. This can however be attributed to stochasticity in the optimization, as the validation accuracy curves in figure 3a show almost similar curves for the three runs.]** .

[Increasing depth increased overfitting as expected, with validation error starting to grow from epoch 10 for deeper models.] .

[Both increasing width and depth allowed the network to overfit, as illustrated by an increase in generalization gap. However, validation accuracy did not suffer much from it, and in some cases did grow as the model were getting larger. This indicates that an increase in parameter was useful in this task, and finding a way to limit the overfitting would bring an increase in performance.] .

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 probability (*i.e.* the probability that a weight is included) 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. **[The application of L1 and L2 regularization strategies could be formulated as adding penalty terms with L1 and L2 norm 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 as different form:

$$\textbf{L1: } w* = \textbf{argmin}_w \mathcal{L}(X, y, w) + \lambda ||w||_1 \quad (5)$$
$$\textbf{L2: } w* = \textbf{argmin}_w \mathcal{L}(X, y, w) + \lambda ||w||_2 \quad (6)$$

where $w*$ denotes the optimized parameters, $\mathcal{L}$ represents the loss function, and $\{X, y\}$ 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 are different in the effect of regularization. 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.] .

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.

[The results from Table 3 show that all three regularization methods, when taken individually, can improve performances over the baseline when hyperparameters are set to adequate values, as well as catastrophically harm the system when regularization is too strong. Out of the three techniques, dropout with a strength of 0.95 performed best, with a validation accuracy of 0.853, an increase of 1.7% over the baseline. Interestingly, Figures 4a and 4b show that validation accuracy seems to be positively correlated with generalization gap. In reality, this is mostly due to high regularization deteriorating performances both in validation and training, artificially lowering the generalization gap. Throughout our experiments, the generalization gap is lower than that of the baseline model, showing that we are indeed mitigating overfitting.

Since dropout seems the most efficient regularization method, we will focus on finding the best inclusion rate when combined with the best weight decay strengths, respectively 10^{-4} for L1 and 10^{-3} for L2. We will test dropout rates around the optimum, that is .925, .95 and .975. Results shown in Table 3 and Figure 4c show that combining weight penalty and dropout does not lead to straightforward improvements. The best accuracy from these runs is 0.850, coming very close but not quite matching the best dropout run. However, these results are not sufficient to conclude that combining these technique is a bad design, it is possible that the best hyperparameters for combined runs lie far from their counterparts in isolated runs.

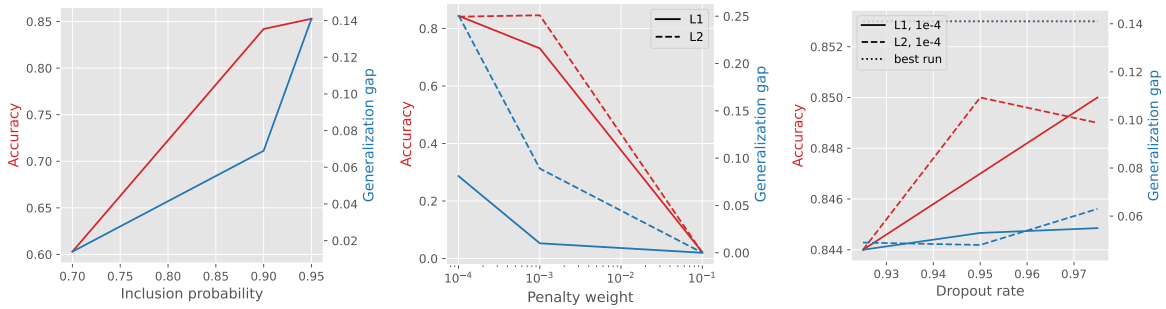
Interestingly once again, generalization gap is maximal for the best run, indicating that regularization might be too strong. Lower generalization gap however is likely to indicate better overall performances, so we can theorize that the model with best test performance is likely to be the one to use an inclusion probability $p = 0.95$ and a L2 weight penalty with strength $\lambda = 1e{-}3$. To verify this hypothesis, we evaluate both models on the test set: the dropout-only model yields 0.842 accuracy and the combined 0.835. Although the difference is small, we would have been better off relying only on validation accuracy and using the dropout-only model.] .

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, [the proof of dropout still remains in the one-layer Softmax regression model. Additionally, the idea of dropout is naturally poorly compatible with Stochastic Gradient Descent, as only selected neurons are updated during each forward pass. Furthermore, dropout tends to require large learning rates, since otherwise the averaged models will be similar to each other. Therefore, the authors argue there might be a direct design for the model to maximize the abilities of dropout during the model averaging.] . 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

Model	Hyperparameter value(s)	Validation accuracy	Generalization gap
Baseline	-	0.836	0.290
Dropout	0.7	0.603	0.014
	0.9	0.842	0.069
	0.95	**0.853**	**0.141**
L1 penalty	1e-4	*0.844*	*0.081*
	1e-3	0.731	0.010
	1e-1	0.022	0.000
L2 penalty	1e-4	0.841	0.251
	1e-3	*0.846*	*0.089*
	1e-1	0.020	0.000
Combined	p=0.925, L1=1e-4	0.844	0.046
	p=0.95, L1=1e-4	0.847	0.053
	p=0.975, L1=1e-4	*0.850*	*0.055*
	p=0.925, L2=1e-3	0.844	0.049
	p=0.95, L2=1e-3	*0.850*	*0.048*
	p=0.975, L2=1e-3	0.849	0.063

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



(a) Metrics by dropout inclusion rate (b) Metrics by weight penalty (c) Metrics for combined runs

Figure 4. Hyperparameter search for every method and combinations

independent affine transformations, then, for each element in output vectors, it takes the maximum value across all subspaces.

[Dropout is compatible with Maxout. One reason is Maxout could keep the variance of gradient on weights to the lower layer, thus alleviate the gradient forgetting problem and improve the bagging effect of dropout. The dropout training will also encourage the maxout unit to have larger linear regions with respect to inputs, and maxout unit may learn to use the averaging produced by dropout.] .

Strengths and limitations The author proposed a novel neural activation unit that further exploits the dropout technique. **[They also experimentally observe the bagging effect of dropout in a multilayer perceptron. In terms of experiments, they conduct multiple of convincing comparisons in different aspects of evaluating novel activations and networks on four benchmarks, while also investigate the gradient behavior (variance etc.) in optimization and the different saturation with rectifier units. They compared the Maxout with other similar ideas as well, *e.g.* learned and stochastic pooling, and experimentally proved that the Maxout achieves better performance in four benchmarks.]** .

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

[The best model when evaluating on the validation set during experiments discussed in Section 4 was a model using only dropout with an inclusion probability $p = 0.95$. Our experimental results confirm the usefulness of dropout and weight decay as a regularization methods, although combining them does not yield better results, probably because we did not find correct hyperparameters. Our best models achieved 0.842 test accuracy, an improvement of 1.2% over the unregulated baseline The experiments support the theory behind why dropouts work, and how averaging many submodels can be beneficial, as well as the theory behind weight penalty. We do however observe significantly larger improvement from using dropout. In this case, with a model with three hidden layers each with 128 units, the approximation of averaging together our submodels was sufficient and reduced overfitting significantly. In future work it would be interesting to experiment more with combining weight penalty and Dropout, and also implement the Maxout Layer(Goodfellow et al., 2013)] .

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.