MLP Coursework 1

s1803764

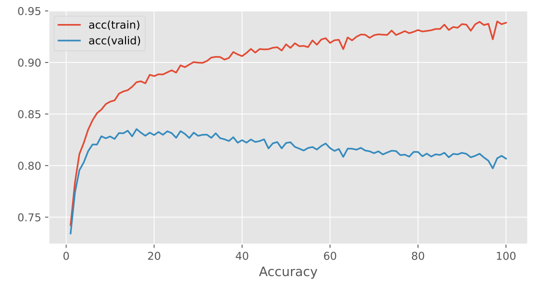
Abstract

In this report we study the problem of overfitting, which is **[when a model learns the ”noise” in the training data so that it does not do well when generalizing to new 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 **[improves the validation accuracy but worsens the validation error. This indicates that as we increase the width/depth of our model it begins to fit "noise" in the training data and thus implies that we should stop training earlier (when the validation accuracy is at it's maximum), and apply regularisation techniques to reduce the model's complexity and thereby reduce the amount of "noise" information retained in the network]** . 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 **[dropout is the most effective regularisation technique for mitigating overfitting as it does a bit more than just regularisation and actually adds robustness to the network. This is due to it's usage of deactivated neurons (which are ignored in the forward and backward propagations) which effectively compares our network with various different ones (given that deactivating neurons acts as a way to randomly change the dimensions of each layer) and chooses the best one. Although, the L1 and L2 regularisation techniques did not produce a validation accuracy as good as that of Dropout, they did produce the best generalisation gaps and came very close to the performance produced by dropout]** . 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 **[the Maxout unit is useful as a means to enhance Dropout's ability as a model averaging technique, however, it does require more trainable parameters than traditional activation functions and does not perform as well as the RELU unit for an increased amount of convolutional filters. We also found that for the EMNIST dataset the Dropout, L1 regularisation, and L2 regularisation techniques are all very useful for mitigating overfitting. From our Dropout experiments we found that for $0.7 \leqslant p \leqslant 0.95$ (where p is the Dropout probability) that p was directly proportional to validation accuracy and generalisation gap, and was optimal at a value of 0.95. From our L1/L2 regularisation experiments we found that for $1^{-1} \leqslant \beta \leqslant 1^{-4}$ (where β is the regularisation parameter) that β was inversely proportional to generalisation gap and validation accuracy, L2 regularisation produces better performing models, and L1 regularisation produces models with lower generalisation gaps. From our combined Dropout and L1/L2 regularisation experiments we found that for $p = 0.95$ and $1^{-6} \leqslant \beta \leqslant 1^{-4}$ L2 regularisation was the better weight penalty technique given it produced the best validation accuracies across all values of $\beta, \beta = 1^{-5}$ was the optimal parameter setting in this context for both L1 and L2 regularisation, and thus the parameters for the best performing model was $\beta = 1^{-5}$ for L2 regularisation and $p = 0.95$ for Dropout. In the future, to make our models perform even better there are a number of things we could still optimise further: the number of epochs used, the batch size, the depth and width of our model, the type of activation function used, the structure of the Dropout layers in our architecture, the L1/L2 regularization parameters, and the random seed used to initialize our weights and Dropout layers]** .

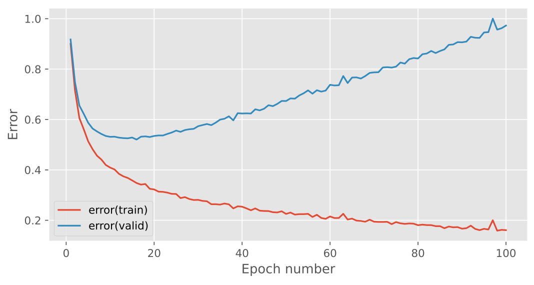
1. Introduction

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 learns the ”noise” in the training data so that it does not do well when generalizing to new 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 **[dropout is the most effective regularisation technique for mitigating overfitting as it does a bit more than just regularisation and actually adds robustness to the network. This is due to it's usage of deactivated neurons (which are ignored in the forward and backward propagations) which effectively compares our network with various different ones (given that deactivating neurons acts as a way to randomly change the dimensions of each layer) and chooses the best one. Although, the L1 and L2 regularisation techniques did not produce a validation accuracy as good as that of Dropout, they did produce the best generalisation gaps and came very close to the performance produced by dropout]** . 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 **[the Maxout unit is useful as a means to enhance Dropout's ability as a model averaging technique, however, it does require more trainable parameters than traditional activation functions and does not perform as well as the RELU unit for an increased amount of convolutional filters. We also found that for the EMNIST dataset the Dropout, L1 regularisation, and L2 regularisation techniques are all very useful for mitigating overfitting. From our Dropout experiments we found that for $0.7 \leqslant p \leqslant 0.95$ (where p is the Dropout probability) that p was directly proportional to validation accuracy and generalisation gap, and was optimal at a value of 0.95. From our L1/L2 regularisation experiments we found that for $1^{-1} \leqslant \beta \leqslant 1^{-4}$ (where β is the regularisation parameter) that β was inversely proportional to generalisation gap and validation accuracy, L2 regularisation produces better performing models, and L1 regularisation produces models with lower generalisation gaps. From our combined Dropout and L1/L2 regularisation experiments we found that for $p = 0.95$ and $1^{-6} \leqslant \beta \leqslant 1^{-4}$ L2 regularisation was the better weight penalty technique given it produced the best validation accuracies across all values of β, $\beta = 1^{-5}$ was the optimal parameter setting in this context for both L1 and L2 regularisation, and thus the parameters for the best performing model was $\beta = 1^{-5}$ for L2 regularisation and $p = 0.95$ for Dropout. In the future, to make our models perform even better there are a number of things we could still optimise further: the number of epochs used, the batch size, the depth and width of our model, the type of activation function used, the structure of the Dropout layers in our architecture, the L1/L2 regularization parameters, and the random seed used to initialize our weights and Dropout layers]** .

[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.

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 is trained for too long resulting in an increasingly large generalization gap (the difference between the training error and validation error) and a decreasing validation accuracy. These trends indicate that the model is starting to learn "noise" in the training data and thus hinders the model's ability to generalize well to new unseen data such as the validation set]** .

[Overfitting occurs due to the mathematical techniques we use to train and optimise our ML models. In order to optimise our model we define an error function that represents how far our model's predictions are away from the real labels, thus to create a model with perfect accuracy we want a model that produces 0 error. So in order to minimize the error of our model we can try adjust the weights of our model that moves our error function towards a local/global minimum (by finding where the derivative of this error function is equal to 0).

# hidden units	val. acc.	generalization gap
32	77.94%	0.148
64	80.91%	0.344
128	80.92%	0.803

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

However, the problem here is that this error function is defined by the training data which inevitably stores some "noise" due to various reasons (data collection errors due to inaccurate/miscalibrated sensors, or varying conditions; or noise added by computer processing such as floating point round-off errors etc.), and so as we begin to become close to 0 error we start to fit this "noise" into our model. Thus we must be wary as to when we should stop training our model to prevent fitting this "noise". We can identify when our model starts to overfit when the generalisation gap starts to increase and the validation accuracy starts to decrease, as this indicates that this further training is actually worsening the performance of this model] .

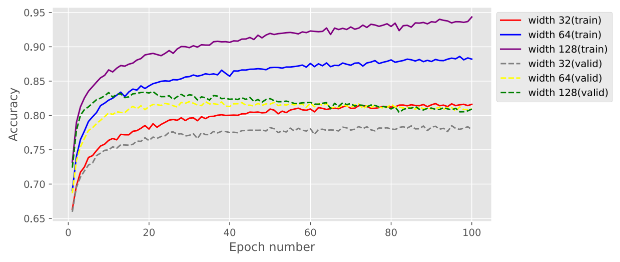
Fig. 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that **[the training and validation accuracies both improve logarithmically with the number of epochs, however, after epoch 17 we can see that the validation accuracy begins to decrease linearly with the number of epochs. This indicates that this model is starting to overfit to the "noise" in the training set after epoch 17 as the validation accuracy starts to decrease and generalisation gap starts to increase with the number of epochs. This is further supported by figure 1b which shows the validation and training errors as we increase the number of epochs. We can see that after epoch 17 the validation error starts to slowly increase in a linear fashion and the generalisation gap ($E_{train} - E_{valid}$) starts to increase exponentially (due to the fact that the training error continues to decrease linearly and the validation error continues to increase linearly).]** .

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.

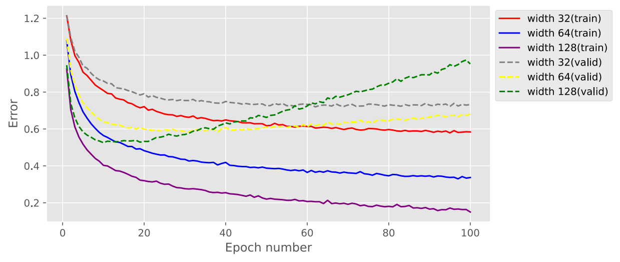
2.1. Network width

[ ] [ ]

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.



(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 and generalization gap. We observe that **[as given by table 1 the widest model (128 units) performed the best given it produced the best validation accuracy, however, it was also the most overfitted given it produced the largest generalization gap. This seems to be the general trend for these width experiments, that as we increase the width of our model, our validation accuracy and generalization gap both increase. This is evident from both figure 2a and figure 2b. In figure 2a we can see that the validation accuracy of the model with 128 units peaks at epoch 20 and after this it decreases slowly in a linear fashion with the number of epochs and converges with the validation accuracies of the thinner models. In figure 2b we can see that the classification error of the model with 128 units reaches its minimum at epoch 9 and after this it increases slowly as the training error continues to decrease with the number of epochs, indicating a widening of the generalization gap. From these figures we can also see that these thinner models do not overfit as quickly as the wider ones due to their lower complexity. This lower complexity implies that these thinner networks are not able to store as much information as the wider networks making it harder for them to fit to the training data and thereby store the insignificant features/"noise" from the training data. This is evident in figure 2a as the model with 32 units continues to slowly increase in validation accuracy all the way up to epoch 94 (where it reaches it's maximum validation accuracy) in con-**

# hidden layers	val. acc.	generalization gap
1	80.92%	0.803
2	81.56%	1.456
3	82.51%	1.538

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

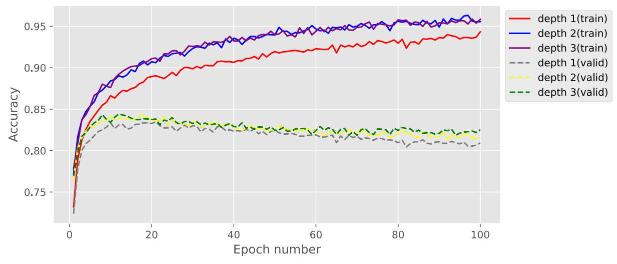
trast to the 128 unit model which achieved maximum validation accuracy at epoch 20, and the 64 unit model which achieved maximum validation accuracy at epoch 29. These exponential differences between the optimal epoch numbers for each model are to be expected given the exponential differences in the widths of all these models ($32 = 2^5$, $64 = 2^6$, **and** $128 = 2^7$)] .

[**From these results it is evident that as we increase the width of our model the validation accuracy increases, the classification error decreases, and the number of epochs needed to train our model decreases. However, we must note that this observation only holds up when we stop training our model before it becomes overfitted. We can define this point of overfitting to be the epoch when the given model reaches their optimum validation accuracy as this indicates this model is best at generalising to new unseen data. This overfitting is due to the fact that as we increase the number of training epochs our error function comes closer and closer to reaching a local minimum, but this local minimum is defined by the training samples in our dataset and thus as we get closer to this minimum our network starts to fit noise from our training data preventing it from being able to generalise well. This overfitting is evident across all our models here as they all reach their maximum validation accuracy scores before epoch 100, and after the epoch where this maximum validation accuracy is achieved we can see that for each model the validation accuracy starts to slowly decrease and the classification error starts to slowly increase. These results were to be expected as the wider the model implies more information can be stored in the network allowing it to make more informed predictions. Furthermore, being able to store more information can also make the model more susceptible to overfitting as if it is trained for too long it can result in the network storing the "noise" from the training data and thus not allowing it to generalize well to new unseen data, which explains why the wider models overfit more quickly than shallower ones**] .

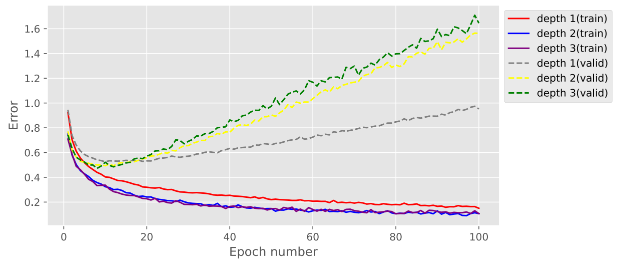
2.2. Network depth

[ ] [ ]

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 size of 100.



(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.

We observe that [**as given by table 2 the deepest model (3 layers) performed the best given it produced the best validation accuracy, however, it was also the most overfitted given it produced the largest generalization gap. This seems to be the general trend for these depth experiments, that as we increase the depth of our model, our validation accuracy and generalization gap both increase. This is evident from both figure 3a and figure 3b. In figure 3a we can see that the validation accuracy of the model with 3 layers peaks at epoch 11 and after this it decreases slowly in a linear fashion with the number of epochs and converges with the validation accuracies of the shallower models. In figure 3b we can see that the classification error of the model with 3 layers units reaches its minimum at epoch 7 and after this it increases slowly as the training error continues to decrease with the number of epochs, indicating a widening of the generalization gap. From these figures we can also see that these shallower models do not overfit as quickly as the deeper ones due to their lower complexity. This lower complexity implies that these shallower networks are not able to store as much information as the deeper networks making it harder for them to fit to the training data and thereby store the insignificant features/"noise" from the training data. This is evident in figure 3a as the model with 1 layer continues to slowly increase in validation accuracy all the way up to epoch 20 (where it reaches it's maximum validation accuracy) in contrast to the 3 layer model which achieved maximum validation accuracy at epoch 11, and the 2 layer model which achieved maximum validation accuracy at**

epoch 17. These linear differences between the optimal epoch numbers for each model are to be expected given the linear differences in the depths of all these models] .

[From these results it is evident that as we increase the depth of our model the validation accuracy increases, the classification error decreases, and the number of epochs needed to train our model decreases. However, we must note that this observation only holds up when we stop training our model before it becomes overfitted. We can define this point of overfitting to be the epoch when the given model reaches their optimum validation accuracy as this indicates this model is best at generalising to new unseen data. This overfitting is due to the fact that as we increase the number of training epochs our error function comes closer and closer to reaching a local minimum, but this local minimum is defined by the training samples in our dataset and thus as we get closer to this minimum our network starts to fit noise from our training data preventing it from being able to generalise well. This overfitting is evident across all our models here as they all reach their maximum validation accuracy scores long before epoch 100, and after the epoch where this maximum validation accuracy is achieved we can see that for each model the validation accuracy starts to slowly decrease and the classification error starts to slowly increase. These results were to be expected as the deeper the model implies more information can be stored in the network allowing it to make more informed predictions. Furthermore, being able to store more information can also make the model more susceptible to overfitting as if it is trained for too long it can result in the network storing the "noise" from the training data and thus not allowing it to generalize well, which explains why the deeper models overfit more quickly than shallower ones] .

[From table 1 and table 2 we can see that increasing depth is a more effective means to increase performance as it achieves better validation accuracy across all models (when looking at the optimal validation accuracy across 100 epochs). We can also see from these tables that increasing width makes the model less susceptible to overfitting than increasing the depth given the far smaller generalization gaps achieved across all the width experiment models. When looking at figure 2a and figure 3a we can see that there is a far larger gap in accuracies between the models of varying width which indicates the rapid improvement increasing width can have on a model's performance. Finally, when analysing these two methods for developing neural networks we must also keep in mind the computational costs of each of these, as different applications may demand different amounts of computational efficiency. In a time-critical application (where computation speed needs to be minimised as much as possible) it is more computationally effective to widen the network than increase the depth of the network as wider networks allow many multiplications to be completed in parallel, unlike deeper networks which require more sequential operations (since the computations depend on the outputs of the previous layers) (Zagoruyko & Komodakis, 2016)] .

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. **[These techniques work by adding a regularization term to the error function which acts as a penalty for complex models with large weights. This is so that when we train our model using gradient descent, our error function will gravitate towards a local minimum that does not take "noisy"/insignificant features into account. The mathematical formula for the error function without regularization is as follows:**

$$E^n = E^n_{train}$$

We incorporate L1 regularisation into training our model by adding a regularisation term to this error function that penalizes large weights:

$$E^n = E^n_{train} + \beta \sum_{i=1}^{n} |w_i|$$

where β is the regularization parameter

And now we can try to reach the *new* local/global minimum of our L1 error function by using the following equation:

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n_{train}}{\partial w_i} + \beta \mathrm{sgn}(w_i)$$

where sgn(w_i) is the sign of w_i

We incorporate L2 regularisation into training our model by adding a regularisation term to this error function that penalizes large weights:

$$E^n = E^n_{train} + \beta \sum_{i=1}^{n} w_i^2$$

where β is the regularization parameter

And now we can try to reach the *new* local/global minimum of our L2 error function by using the following equation:

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n_{train}}{\partial w_i} + \beta w_i$$

This regularization parameter β is evidently extremely important to both these regularization techniques as it directly affects the complexity and training-data fit of the model. Given this parameter is handpicked it is extremely important that it is set optimally. The choice of this parameter is entirely dependent on how far we want to simplify our model (as increasing it's magnitude strengthens this regularization effect) so we must keep the initial performance and fitting of our model in mind when choosing this. As shown in table 3 a good method to choose this parameter would be to test varying size β values on the validation set and choose the one that produces the best validation accuracy] .

[These regularisation techniques address overfitting by penalising large weights in models. Penalising these large weights helps to reduce the complexity of the model and ultimately retain less noise from the training set. The main difference between L1 and L2 regularisation is the regularisation terms they add to the error function. In L1 regression weights shrink to 0 at a constant rate (βsgn(w_i)) in contrast to L2 regression where weights shrink to 0 at a rate proportional to the magnitude of the weight (βw_i). This difference is due to the fact that L1 regularization penalizes the sum of the absolute value of weights whereas L2 regularization penalizes the sum of square weights. Such differences in these regularization terms give each of these methods different strengths for different types of data. L1 regularisation is particularly useful for feature selection as it produces a sparse solution and allows us to drop features based on the weights that go to 0 (Tyagi, 2021). L2 regularisation on the other hand produces non-sparse solutions and is useful when you have collinear/codependent features in your dataset. This is due to the fact that codependence tends to increase weight variance, which makes the weights unreliable/unstable and can end up hurting the model's generality. L2 fixes this by reducing the variance of these estimates which counteracts the effect of codependencies (exp, 2021)] .

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.

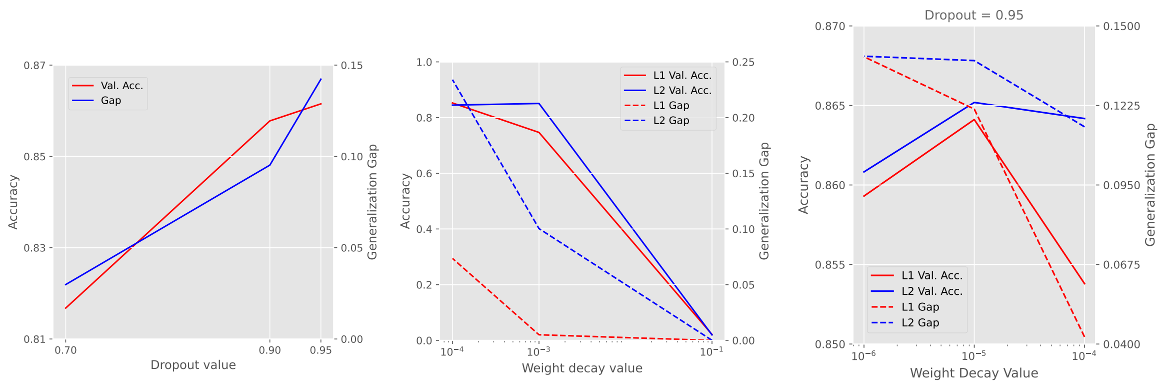
We start by lowering the learning rate to 10^{-4}, as the previous runs were overfitting in only a handful of epochs. Results for the new baseline (c.f. Table 3) confirm that lowering learning rate helps, so all further experiments are run using it.

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.

Model	Hyperparameter value(s)	Validation accuracy	Generalization gap
Baseline	-	0.836	0.290
Dropout	0.7	0.817	*0.030*
	0.9	0.858	0.095
	0.95	*0.862*	0.142
L1 penalty	1e-4	*0.853*	0.074
	1e-3	0.747	0.005
	1e-1	0.021	***0***
L2 penalty	1e-4	0.845	0.234
	1e-3	*0.851*	0.100
	1e-1	0.02	***0***
Combined	0.95, L1 1e-4	0.854	*0.042*
	0.95, L1 1e-5	0.864	0.121
	0.95, L1 1e-6	0.859	0.139
	0.95, L2 1e-4	0.864	0.115
	0.95, L2 1e-5	***0.865***	0.138
	0.95, L2 1e-6	0.861	0.140

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) Metrics by combining inclusion rate and weight penalty

Figure 4. Hyperparameter search for every method and combinations

[From our Dropout experiments it is evident to see that validation accuracy and generalisation gap are both directly proportional to the Dropout probability when $0.7 <= p <= 0.95$ (where p is the Dropout probability). This was achieved by applying a Dropout layer before each Affine layer in our network (resulting in 4 Dropout layers), and using the same Dropout probability for each Dropout layer. These results were to be expected given that the higher the Dropout probability implies less nodes to drop and thus more information is retained in the network meaning higher validation accuracy, but since we are retaining more information this also makes our model more susceptible to overfitting thus widening the generalisation gap. However, these results do not cover any cases where $0.5 <= p < 0.7$, this is problematic as we cannot extrapolate the trend found from our 3 experiments to learn about the performance for Dropout layers with $0.5 <= p < 0.7$. Such results would have been useful to get a better insight when choosing the Dropout probability for our combination experiments.

From our L1/L2 regularisation experiments it is evident that L1 regularisation is less susceptible to overfitting than L2 regularisation based on the far lower generalization gaps this method achieved as shown in table 3. From these results we can also see that the L1 regularisation parameter is inversely proportional to the validation accuracy and generalisation gap. The L2 regularisation parameter seemed to follow the same general trend except for the validation accuracy, in which after the L2 regularisation parameter was set to 1e-4 the validation accuracy decreased from that achieved by setting the parameter to 1e-3. However, we must be careful before making any deductions as this parameter producing a better network for the validation set could just be due to luck/coincidence given there was only a 0.6% difference in accuracy between these hyperparam-

eter settings. This trend is to be expected given that the smaller the magnitude of the regularisation parameter implies the smaller the penalty applied to large weights, meaning more information retained in the network and thus the validation accuracy will continue to improve until it reaches a point of overfitting, but since we are retaining more information this also makes our model progressively more susceptible to overfitting thus widening the generalisation gap.

For my combined regularisation method experiments I carefully chose what hyperparameters to use. I wanted to use my findings from my regularisation experiments as a way to inform what parameters would be best for Dropout and the L1/L2 weight penalties. However, when choosing these hyperparameter settings I also wanted to be representative of the range of possible parameter settings as although we know what parameters seem to perform best when using these regularisation techniques by themselves we cannot be sure as to how they will react when used in conjunction.

From the Dropout experiments it was evident that setting the Dropout probability to 0.95 was the optimal choice since it gave the highest validation accuracy of 86.15%. Thus for all my combination experiments I decided to keep the Dropout probability fixed at 0.95 to maximise validation accuracy. Although this parameter setting produced the largest generalisation gap we can be less worried about this given we will be using it in conjunction with L1/L2 regularisation which should help mitigate this overfitting even further.

From the results of our L1/L2 regularisation experiments as shown in table 3 L1 seems like the more sensible choice. This is due to the fact that it achieved the highest validation accuracy from these experiments at 85.25% for a parameter setting of 1e-4, and achieved far smaller generalisation gaps than L2 for the same parameter settings. However, the differences in validation accuracy were small and I wanted to be representative in my combination experiments so I decided to use both techniques.

For each of these regularisation techniques the smaller the parameter seemed to imply the better the accuracy, thus in conjunction with 1e-4 (since it achieved the best score on average across L1 and L2 experiments) I decided to use even smaller regularisation parameters for our combination experiments (1e-5 and 1e-6).

Thus as shown in table 3 the hyperparameter combinations I used were as follows:

- **Dropout probability** $= 0.95$
- **Weight penalty technique** $\in \{\mathbf{L1}, \mathbf{L2}\}$
- **L1/L2 regularisation parameter** $\in \{1^{-4}, 1^{-5}, 1^{-6}\}$

After running all these experiments my findings proved very useful. As shown in figure 4c L2 regularisation proved to be the superior weight penalty technique to L1 given it achieved higher validation accuracies across all possible weight decay settings. Although, it ended up producing worse generalisation gaps than L1 for all weight decay settings this was to be expected based on the generalisation gap results from the lone L1 and L2 experiments. In addition, again as shown in figure 4c 1e-5 proved to be the optimal regularisation parameter setting given it produced the best validation accuracies for both L1 and L2 regularisation.

Thus in line with the findings discussed above my best model ended up using L2 regularisation with parameter setting 1e-5, and Dropout with probability 0.95. This model achieved 86.52% validation accuracy and 84.87% test accuracy] .

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 has not previously been demonstrated to actually perform model averaging for deep architectures. Dropout is generally seen as an indiscriminately applicable tool that maxout networks are designed to both facilitate optimisation by dropout and improve the accuracy of dropout's fast approximate model averaging technique. The maxout network is a feed-forward architecture that uses a new type of activation function called the maxout unit. Given an input a maxout hidden layer implements the following function:**

$$h_i(x) = \max_{j\in[1,k]} z_{ij}$$

where $z_{ij} = x^T W_{...ij} + b_{ij}$ **, and** $W \in \mathbb{R}^{d\times m\times k}$ **and** $b \in \mathbb{R}^{m\times k}$

are learned parameters

A single maxout unit can be interpreted as making a piecewise linear approximation to an arbitrary convex function.

Maxout networks are useful because they learn not just the relationship between hidden units, but also the activation function of each hidden unit. The Maxout units are particularly useful as a multilayer perceptron containing 2 maxout units can be used to arbitrarily approximate any function] . 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.

[We know that Dropout is compatible with Maxout given Maxout was developed as a method to enhance Dropout's abilities as a model averaging technique. This is further supported by the founding paper on Maxout networks (Goodfellow et al., 2013) which actually used Maxout in conjunction with Dropout to test Maxout's performance on benchmark datasets (including MNIST)] .

Strengths and limitations The author proposed a novel neural activation unit that further exploits the dropout technique. **[The setup for this experiment was good as they decided to test their novel method on benchmark datasets, thus making it compareable to the performance of existing neural networks with different regularisation methods, and/or activation units. I also thought using logic proofs to prove that Maxout is a universal approximator was very good as it provides solid evidence for this claim, and ensures a conservation of logic.**

However, there were a few areas in this paper which were not exhaustively covered and added some ambiguity to the quality of this method. Firstly, I believe they should have provided error scores for models fitted with Dropout alone, and with Maxout alone. As although they said it was "particularly well suited for training with dropout" it is still useful to be able to compare the performances of these techniques independently. Secondly, given a network with Maxout activation units has a higher number of trainable parameters compared to traditional acitvation functions it is unclear whether Maxout's increased performance was mainly/solely due to an the increase in the number of trainable parameters or not. Thirdly, in the conclusion of this paper it was stated that "We have shown empirical evidence that dropout attains a good approximation to model averaging in deep models. We have shown that maxout exploits this model averaging behavior because the approximation is more accurate for maxout units than for tanh units". Although tanh is a very reliable and good activation unit this does not mean it is the optimal activation unit in every context. It would have been better if they performed tests on more types of activation units to make the performance of Maxout more compareable. This ambiguity actually resulted in a paper being published that investigated the quality of Maxout in comparison to many other activation units (RELU, Leaky RELU, SELU, and TANH) (Castaneda et al., 2019). This paper found that Maxout networks trained relatively slower than networks with traditional activation functions (which can be attributed to the increase in trainable parameters). Furthermore, the RELU activation function ended up performing better than any Maxout function when the number of convolutional filters was increased] .

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

[From these results it is evident that for the EMNIST dataset the Dropout, L1 regularisation, and L2 regularisation techniques are all very useful for mitigating overfitting. From our Dropout experiments (as shown in figure 4a) we found that for $0.7 \leqslant p \leqslant 0.95$ (where p is the Dropout probability) that p was directly proportional to validation accuracy and generalisation gap, and was optimal at a value of 0.95. From our L1/L2 regularisation experiments (as shown in figure 4b) we found that for $1^{-1} \leqslant \beta \leqslant 1^{-4}$ (where β is the L1/L2 regularisation parameter) that β was inversely proportional to generalisation gap and validation accuracy, L2 regularisation produces better performing models, and L1 regularisation produces models with lower generalisation gaps. From our combined Dropout and L1/L2 regularisation experiments (as shown in figure 4) we found that for $p = 0.95$ and $1^{-6} \leqslant \beta \leqslant 1^{-4}$ L2 regularisation was the better weight penalty technique given it produced the best validation accuracies across all values of β, $\beta = 1^{-5}$ was the optimal parameter setting in this context for both L1 and L2 regularisation, and thus the parameters for the best performing model was $\beta = 1^{-5}$ for L2 regularisation and $p = 0.95$ for Dropout.

In the future, to make our models perform even better there are a number of things we could still optimise further: the number of epochs used, the batch size, the depth and width of our model, the type of activation function used, the structure of the Dropout layers in our architecture, the L1/L2 regularization parameters, and the random seed used to initialize our weights and Dropout layers.

The first step I would take towards optimising my models would be through optimising the number of epochs used. Throughout all our experiments we set the number of epochs to be constant at 100. This is evidently not

a realistic setting to use in implementation as we would rather choose the number of epochs dynamically based on what achieves the best validation accuracy.

Secondly, I would optimise the batch size used by gradient descent in my model. Throughout all our experiments we have kept the batch size fixed at 100, however, we should rather set this dynamically based on what setting enables our model to perform best. Changing this batch size could dramatically improve our models ability to generalise: "it has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize." - (Keskar et al., 2016)

Thirdly, to optimise the shape (width and depth) of our model I would mainly focus on increasing the width as our experiments from figure 3 indicate that increasing the depth has minimal performance returns after layer 2, given the model with 3 layers achieved an optimal validation accuracy of 84.41% (a 0.01% performance decrease from the 2 layer model). In contrast, we can see in figure 2 that the performance returns for increasing the width are much greater (as illustrated by the difference between the optimal validation accuracies achieved by each width model), given the model with 128 units achieved an optimal validation accuracy of 83.48% (a 1.47% performance increase from the 64 unit model). However, we must note that this performance increase was due to an exponential increase in the number of hidden units ($2^6 -> 2^7$) and thus any further significant increases in performance may require another exponential increase in the number of hidden units and thus an exponential increase in computation.

Next, to optimise the activation unit used I would like to test varying different types of activation units and choose the one which performs best. In particular, I would like to test the Maxout unit in conjunction with Dropout to enhance Dropout's abilities as a model averaging technique.

After this, to optimise our Dropout layers even further we could optimise the number of layers used, their placement within the network (before which Affine layers), and the unique probabilities used for each of these layers. I would start off by setting p in the input layer as 0.95 (based on our results), to be between 0.5 and 0.8 in the hidden layers, and no Dropout on the output layer (Brownlee, 2018). From here I would would then optimise each of these layers' parameters by training multiple models while varying these parameter values. Once I have a sufficient amount of models with varying Dropout layer probabilities I would plot all this data like in figure 4a as a means to try find a trend in these parameter settings and ultimately determine which settings would be optimal.

Now, to optimise the L1 and L2 regularization models even further we could optimise the regularisation parameter for the weights and bias separately. This could be done by iterating through various regularisation parameter settings, plotting their performances and choosing the parameter settings that produced the best performing model.

Lastly, to optimise the random seed we would just iterate through various random seeds to find which produces the optimal start point for the weights and Dropout layers. This optimisation is important in the context of gradient descent due to the fact we want to find the global minimum of the error function. This is due to the fact that performing gradient descent will always bring us to a local minimum but not necessarily a global minimum as it finds the minimum closest to the initialised weights. Thus finding this global minimum is fully dependent on the initialised weights (and thus the random seed) as this denotes which minimum our model starts closest to] .

References

The difference between l1 and l2 regularization, 2021. URL https://explained.ai/regularization/L1vsL2.html.

Brownlee, Jason. A gentle introduction to dropout for regularizing deep neural networks, Dec 2018. URL https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.

Castaneda, Gabriel, Morris, Paul, and Khoshgoftaar, Taghi M. Evaluation of maxout activations in deep learning across several big data domains. *Journal of Big Data*, 6(1), Aug 2019. doi: 10.1186/s40537-019-0233-0. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0233-0.

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.

Keskar, Nitish Shirish, Mudigere, Dheevatsa, Nocedal, Jorge, Smelyanskiy, Mikhail, and Tang, Ping Tak Peter. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. URL http://arxiv.org/abs/1609.04836.

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.

Tyagi, Neelam. L2 vs l1 regularization in machine learning | ridge and lasso regularization, 2021. URL https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.