MLP Coursework 1

s1893731

Abstract

In this report we study the problem of overfitting, which is **[the phenomenon by which a model generalises the training data incredibly well achieving a high accuracy and low loss on the training data while not generalising unseen data in the test set. Such models typically have low bias, high variance and a high generalization 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 **[It was observed that increasing the width and depth of the model increased the number of parameters of the model making it more complex, thereby increasing the generalization gap of the model. Therefore, if we keep increasing the width and depth of a model, after a certain point, it is bound to overfit even though it can be observed that increasing the depth from 1 hidden layer to 3 hidden layers allowed the model to generalize the validation data much better.]** . 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 **[It was observed from the experiments for dropout that the model achieved higher validation accuracy for dropout values of 0.7 and 0.9. The best validation accuracy was achieved for a dropout value of 0.9 which was 0.8559 with the generalization gap not exceeding 0.1874. For the dropout values less than 0.7, it can be observed that the model is clearly underfitting due to the incredibly low values of include probability which greatly reduce the number of parameters making the model less complex and lowering the capacity of the model, due to which the model isn't able to sufficiently reduce the training error even though it manages to minimize the generalization gap. Therefore, for models utilising include probabilities less than 0.7 with dropout, the models belong to underfitting zone. While the include probability of 0.9 provided optimal capacity performing well on the validation set.**

Looking at the L1 experiments conducted, we can observe that the best validation accuracy score of 0.8462 was achieved for a penalty of 1e-4 and the second best validation accuracy score of 0.8331 was achieved for penalty value of 1e-5. It can be observed that for penalty values of 1e-3 and greater, the model clearly starts underfitting, the reason this can be observed is because the more alue of the penalty hyperparameter keeps increasing, the more magnitude of weight decay keeps increasing, when increased to a certain extent it greatly reduces the weight vector causing the model to underfit, which is evident from the validation accuracies of approximately 0.02 when the penalty hyperparameter value is 1e-2 or 1e-1.

Looking at the L2 experiments it can be observed that the best validation accuracy score of 0.8498 is obtained for the penalty hyperparameter value of 1e-3. It can be observed that the best results were obtained for hyperparameter values 1e-5,1e-4 and 1e-3 while with the increasing hyperparameter values, the model tends to underfit due to the same reasons stated for L1 weight decay. However, the phenomenon of underfitting is less severe in the case of L2 weight decay due to the fact that the updated weight equation varies linearly with the weight whereas in L1 weight decay the updated weight equation depends on the hyperparameter value and the sign of the weight vector. This makes L2 more stable at higher values of the hyperparameter and is the reason why L2 weight decay gives more promising results than L1 weight decay, especially for hyperparameter values of 1e-2 and 1e-3.

] . 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 **[We received some great insight into the phenomenon of a model overfitting, and conducted experiments to ascertain how and why a model is overfitting or underfitting, learning about the optimal capacity of a model through various experiments on increasing depth and width. We also conducted various experiments in order to mitigate overfitting**

through regularization methods like dropout and L1 and L2 weight decay, experimenting with various hyperparameter values both independently and in combinations to find the best performing models and more importantly to understand how the regularization techniques actually work and how exactly do they mitigate overfitting by understanding the trends of the model performance on different hyperparameter values.

] .

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 phenomenon by which a model generalises the training data incredibly well achieving a high accuracy and low loss on the training data while not generalising unseen data in the test set. Such models typically have low bias, high variance and a high generalization 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 [It was observed from the experiments for dropout that the model achieved higher validation accuracy for dropout values of 0.7 and 0.9. The best validation accuracy was achieved for a dropout value of 0.9 which was 0.8559 with the generalization gap not exceeding 0.1874. For the dropout values less than 0.7, it can be observed that the model is clearly underfitting due to the incredibly low values of include probability which greatly reduce the number of parameters making the model less complex and lowering the capacity of the model, due to which the model isn't able to sufficiently reduce the training error even though it manages to minimize the generalization gap. Therefore, for models utilising include probabilities less than 0.7 with dropout, the models belong to underfitting zone. While the include probability of 0.9 provided optimal capacity performing well on the validation set.

Looking at the L1 experiments conducted, we can observe that the best validation accuracy score of 0.8462 was achieved for a penalty of 1e-4 and the second best validation accuracy score of 0.8331 was achieved for penalty value of 1e-5. It can be observed that for penalty values of 1e-3 and greater, the model clearly starts underfitting, the reason this can be observed is because the more alue of the penalty hyperparameter keeps increasing, the more magnitude of weight decay keeps increasing, when increased to a certain extent it greatly reduces the weight vector causing the model to underfit, which is evident from the validation accuracies of approximately 0.02 when the penalty hyperparameter value is 1e-2 or 1e-1.

Looking at the L2 experiments it can be observed that the best validation accuracy score of 0.8498 is obtained for the penalty hyperparameter value of 1e-3. It can be observed that the best results were obtained for hyperparameter values 1e-5,1e-4 and 1e-3 while with the increasing hyperparameter values, the model tends to underfit due to the same reasons stated for L1 weight decay. However, the phenomenon of underfitting is less severe in the case of L2 weight decay due to the fact that the updated weight equation varies linearly with the weight whereas in L1 weight decay the updated weight equation depends on the hyperparameter value and the sign of the weight vector. This makes L2 more stable at higher values of the hyperparameter and is the reason why L2 weight decay gives more promising results than L1 weight decay, especially for hyperparameter values of 1e-2 and 1e-3.

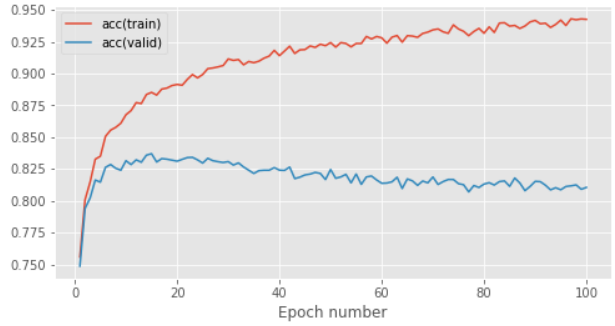
] . In Section 5, we discuss a related work on Maxout Networks and highlight its pros and cons.[1] Finally, we conclude our study in section 6, noting that [We received some great insight into the phenomenon of a model overfitting, and conducted experiments to ascertain how and why a model is overfitting or underfitting, learning about the optimal capacity of a model through various experiments on increasing depth and width. We also conducted various experiments in order to mitigate overfitting through regularization methods like dropout and L1 and L2 weight decay, experimenting with various hyperparameter values both independently and in combinations to find the best performing models and more importantly to understand how the regularization techniques actually work and how exactly do they mitigate overfitting by understanding the trends of the model performance on different hyperparameter values.

] .

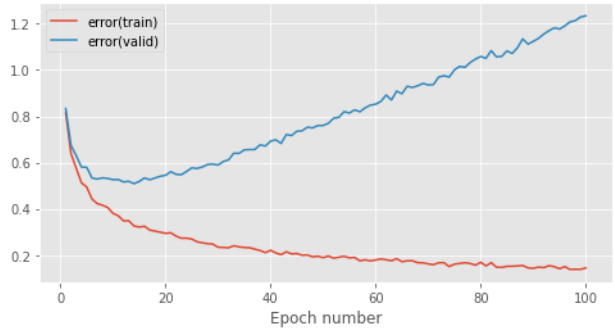
2. Problem identification

Overfitting to training data is a very common and important issue that needs to be dealt with when training neural

[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.

networks or other machine learning models in general (see Ch. 5 in Goodfellow et al. 2016). A model is said to be overfitting when **[a model generalises the training data incredibly well achieving a high accuracy and low loss on the training data while not generalising unseen data in the test set, achieving a relatively higher error on the test set. Such models typically have low bias, high variance and a high generalization error which is the difference between the error achieved by the model on the training set and that on the test set.]** .

[Overfitting can occur due to a model having a high capacity to fit a wide variety of functions due to which the model can memorize properties of the training set that does not serve the model well on the test set. One can identify overfitting by measuring the performance of the model on a training set and a test set and calculating the generalization gap.] .

Fig. 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that **[The figures represent the change in accuracy and error of the model on the training set and the validation set with respect to the number of epochs for which the model is trained.**

Looking at Fig 1(a), it can be observed that the model starts with a classification accuracy of 0 which rapidly increases on the training set with the increase in the number of epochs for which it is trained, increasing all the way to 0.95 on the 100th epoch, while the accuracy of the model on the validation set reaches a maximum of approximately 0.837 on the 15th epoch after which it suffers a slight and gradual decrease over the next 85 epochs, decreasing to around 0.81 at the 100th epoch, it can be observed from the difference in the trends of the training accuracy and validation accuracy that the model is clearly overfitting and the two accuracy curves diverge from around the 5th epoch with the accuracy of the model on the training set steeply increasing while that on the validation set largely staying constant and gradually decreasing.

Looking at Fig 1(b), it can be observed that the model starts with an error of 0.8 on both the training and the validation sets, it can be seen that the error of the model on the training set steeply reduces to less than 0.2 after 100 epochs, while the error of the model on the validation set reduces to a minimum of 0.55 on the 15th epoch, after which it experiences an almost linear increase and increases to more than 1.2 over the next 85 epochs. It can be observed that the divergence between the 2 curves takes place at the 5th epoch with the training error curve decreasing much faster than the validation error curve up to the 15th epoch. After the 15th epoch, the validation error curve starts increasing as the model overfits and the generalization gap gets larger and larger with each epoch as the training error curve continues to decrease while the validation error curve experiences an almost linear increase.] .

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.

# hidden units	val. acc.	generalization gap
32	78.4	0.149
64	80.9	0.337
128	80.7	0.804

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

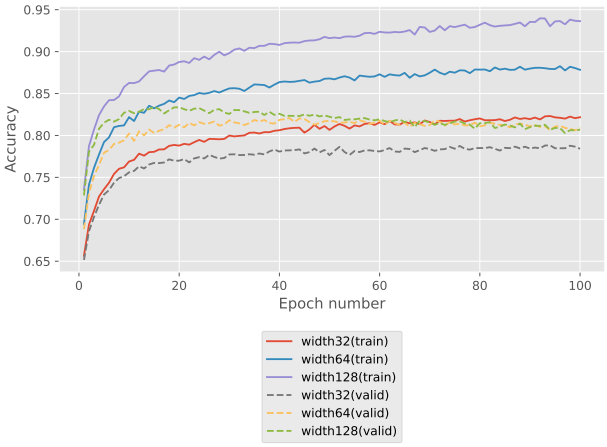
2.1. Network width

[Question Table 1 - Fill in Table 1 with the results from your experiments varying the number of hidden units.

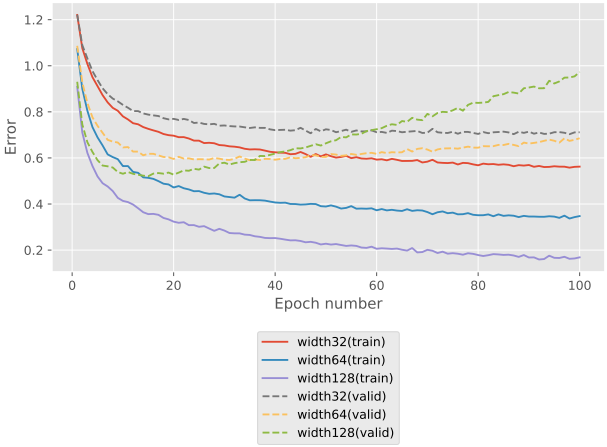
] [Question Figure 2 - Replace the images in Figure 2 with figures depicting the accuracy and error, training and validation curves for your experiments varying the number of hidden units.

]

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



(a) accuracy by epoch



(b) error by epoch

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

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

The input layer is of size 784, and output layer consists of 47 units. Three different models were trained, with a single hidden layer of 32, 64 and 128 ReLU hidden units respectively. Figure 2 depicts the error and accuracy curves over 100 epochs for the model with varying number of hidden units. Table 1 reports the final accuracy and generalization gap. We observe that **[Increasing the width (hidden units) of the network increases the capacity of the model to fit a wide variety of functions, making it more complex. It can be observed from the experiments conducted that the model with 32 hidden units has low complexity and underfits on the data achieving a training accuracy of approximately 0.82 and a validation accuracy of 0.784, looking at the error curves, it can be observed that the gradients of the training and validation curves are largely similar in magnitude with the generalization gap being low .**

We increase the model complexity by increasing the width to 64 hidden units instead of 32, the resulting model predictably achieves a higher accuracy on both training and validation sets than the one with 32 hidden units. The model achieves the highest validation accuracy out of the three models of 0.809, with a training accuracy of 0.875. The validation accuracy curve of the model increases up until epoch 30 after which it stays constant for approximately 20 epochs after which we can witness a slight decrease in the magnitude of the gradient of the validation accuracy curve, hinting that the model might be overfitting. This is also supported by the fact that the validation error curve of the model experiences a very slight linear increase from epoch 55 to epoch 100 increasing the generalization gap to 0.337.

The model complexity is further increased when the width of the model is increased to 128 hidden units, the model predictably achieves the highest accuracy of approximately 0.94 on the training set while achieving a validation accuracy of 0.807 just shy of the model with 64 hidden units which achieved a validation accuracy of 0.809 after 100 epochs. It can be observed that the slope of the validation accuracy curve starts reducing after the 20th epoch which suggests that the model might be overfitting, looking at the error curves, we can observe that the slope of the validation error curve starts increasing linearly from epoch 20 and the curve enters the overfitting zone, increasing the generalization gap of the model to 0.804 after 100 epochs. The model with the highest width also achieved the lowest training error. It is clear that the model is too complex and is clearly overfitting even though it achieves a much better validation accuracy score than the model with 32 hidden units which had the lowest capacity.

We can observe that the model with 64 units performed best in these experiments since it was more complex than the model with the lowest capacity but not overly complex such that it would overfit with a large generalization gap. The model with 32 units clearly underfits while the model with 128 units clearly overfits whereas, the model with 64 units was of the optimal capacity.] .

[Increasing width results in more complex models which have a higher capacity, in models with lower width (lesser capacity) we observed that while the generalization gap was indeed low, the model failed to obtain a sufficiently low training error putting the model in the underfitting regime. According to theory, in order to obtain a model of optimal capacity, the width has to be increased while taking into account that a significant increase in width would result in a much more complex model with a much higher capacity which would be able to optimally minimize the training error but however wouldn't be able to minimize the generalization gap, and we indeed observed this in the model with 128 hidden units which belongs to the overfitting regime. According to theory, the model with optimal capacity, is found somewhere in between the underfitting and overfitting regimes and we indeed found this in the model with 64

# hidden layers	val. acc.	generalization gap
1	80.7	0.800
2	81.4	1.487
3	82.3	1.570

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

hidden units, which not only managed to sufficiently reduce the training error but also managed to minimize the generalization gap simultaneously. Therefore, the experiments are consistent with the expectations from relevant theory and literature.] .

2.2. Network depth

[Question Table 2 - Fill in Table 2 with the results from your experiments varying the number of hidden layers.

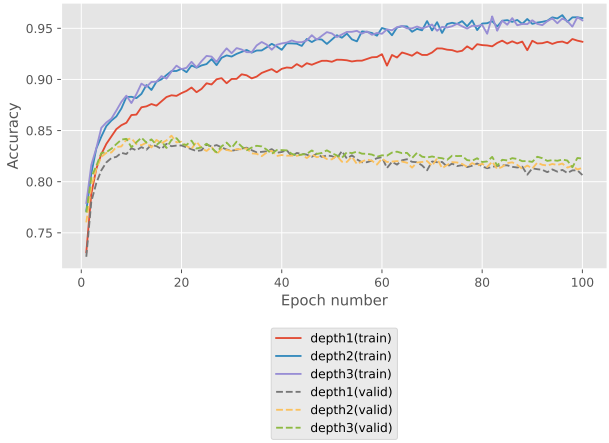
] [Question Figure 3 - Replace these images with figures depicting the accuracy and error, training and validation curves for your experiments varying the number of hidden layers.

]

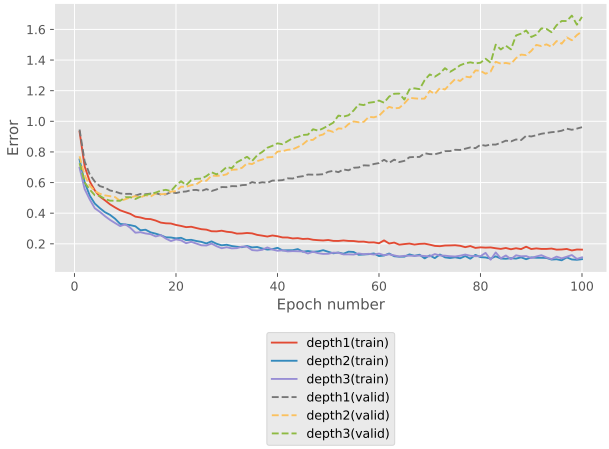
Next we investigate the effect of varying the number of hidden layers in the network. Table 2 and Fig. 3 shows the results from training three models with one, two and three hidden layers respectively, each with 128 ReLU hidden units. As with previous experiments, they are trained with the Adam optimizer with a learning rate of 10^{-3} and a batch size of 100.

We observe that **[the model with 1 hidden layer is the one with least capacity (least complex) out of the 3 models trained in the experiment and achieves a validation accuracy of 0.807 and a generalisation gap of 0.8, this is the same model trained in the network depth experiments with 128 hidden units. Looking at the accuracy of this model, we can observe that it achieves an accuracy just short of 0.95 on the training set, it can be observed that after the 20th epoch the slope of the validation accuracy curve starts gradually decreasing until it reaches an accuracy of 0.807 after 100 epochs. Looking at the error curves, the model sufficiently lowers the training error to less than 0.2 after 100 epochs while the slope of the validation error curve experiences a slight linear increase after the 20th epoch increasing the generalization gap of the model causing it to overfit.**

In contrast, the models with 2 and 3 hidden layers are more complex and have a higher capacity than the model with 1 hidden layer. Looking at the accuracy by epoch plot, it can be observed that the models with 2 and 3 hidden layers have a higher training accuracy (> 0.95) than the model with 1 hidden layer and also have marginally better validation accuracy than the model with 1 hidden layer, the model with 2 hidden layers



(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.

achieving a validation accuracy of 0.814 and the model with 3 hidden layers achieving a validation accuracy of 0.823. Looking at the error by epoch plot, it can be observed that the models with 2 and 3 hidden layers while managing to reduce the training error more than the model with 1 hidden layer, do not manage to minimize the generalization gap. In fact, the slopes of the validation error curves of the models with 2 and 3 hidden layers start increasing linearly by a large factor after 20 epochs which increases the generalization gap to 1.487 and 1.570 for the 2 hidden layer and 3 hidden layer models, respectively.] .

[From the experiments conducted, it can be observed that increasing the depth of the network results in a higher validation accuracy in addition to a higher generalization gap. The models with 2 hidden layers and 3 hidden layers are much more complex than the one with 1 hidden layer and have a higher capacity in addition to which we can observe from the experiments conducted that the models with 2-3 hidden layers simply failed to optimally minimize the generalization gap in addition to

which the validation errors of the models with greater depth are greater than the ones with lesser depth. It indeed seems to be the case that the model with only 1 hidden layer was clearly not complex enough to minimize the training error sufficiently and in order to do so, model depth had to be increased to obtain the model with the optimal capacity, however, if we kept increasing the depth we would end up with a much more complex model that would then fall into the overfitting regime.]

.

[From the above experiments conducted on width and depth, it can be concluded that in order to obtain the model with the optimal complexity, it is necessary to increase the values of width and depth to achieve a model which is complex enough to minimize both training error and generalization gap simultaneously while taking into account that the model is not complex enough to have a high generalization gap even though it sufficiently minimizing the training error.

From the network width experiments, we concluded that the model with 32 hidden units wasn't complex enough and was clearly underfitting with a low training error while the model with 64 hidden units performed the best since it was of optimal capacity and managed to minimize both the training error and the generalization gap simultaneously.

From the network depth experiments, we concluded that the model with only 1 hidden layer was not complex enough and had lesser capacity than the ones with 2-3 hidden layers due to which it didn't optimally minimize the training error. In comparison, the models with 2-3 hidden layers sufficiently minimized the training error, even though they achieved high generalization gaps the trade-off proved to be useful and the model with 3 hidden layers had the best validation accuracy.] .

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 dropout 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. **[L2 weight penalty - This is more commonly known as ridge regression, it is a regularization strategy that drives the weights of a model closer to the origin by adding a regularization term to the objective function in order to limit the capacity of the model.**

The regularization term added to the cost function is - $\Omega(\theta) = \frac{1}{2}||\mathbf{w}||_2^2$

Where θ **Represents both bias and weight parameters.**

This is how L2 regularization is applied to the objective function

$\bar{J}(\mathbf{w}; \mathbf{X}; \mathbf{y}) = \frac{\alpha}{2}\mathbf{w}^\top \mathbf{w} + J(\mathbf{w}; \mathbf{X}; \mathbf{y})$

Where a positive hyperparameter is utilised in α

L1 weight penalty - Commonly known as lasso regression, also helps reducing the model parameters with the help of the addition of a regularization term to the objective function.

The regularization term for L1 weight penalty is -

$\Omega(\theta) = \|\mathbf{w}\|_1 = \sum_i |\mathbf{w}_i|$

Which is the sum of the absolute values of the individual parameters

L1 penalty applied to the cost function looks something like -

$\bar{J}(\mathbf{w}; \mathbf{X}; \mathbf{y}) = \alpha\|\mathbf{w}\|_1 + J(\mathbf{w}; \mathbf{X}; \mathbf{y})$] .

[For L2 regularization, it is known that the regularization term is added to the objective function in the following manner -

$\bar{J}(\mathbf{w}; \mathbf{X}; \mathbf{y}) = (\alpha/2)\mathbf{w}^\top\mathbf{w} + J(\mathbf{w}; \mathbf{X}; \mathbf{y})$

The corresponding parameter gradient can be calculated as -

$\alpha\mathbf{w} + \nabla_w J(\mathbf{w}; \mathbf{X}; \mathbf{y})$

Therefore, the updated weight equation would look like-

$\mathbf{w} \leftarrow \mathbf{w} - \epsilon(\alpha\mathbf{w} + J(\mathbf{w}; \mathbf{X}; \mathbf{y}))$

Therefore, the addition of the L2 regularization term helps in weight shrinkage by a constant factor for each iteration, thereby reducing overfitting.

For L1 regularization, the regularization term is added to the obective function in the following way -

$\bar{J}(\mathbf{w}; \mathbf{X}; \mathbf{y}) = \alpha\|\mathbf{w}\|_1 + J(\mathbf{w}; \mathbf{X}; \mathbf{y})$

The corresponding parameter sub-gradient would be -

$\alpha sign(\mathbf{w}) + \nabla_w J(\mathbf{w}; \mathbf{X}; \mathbf{y})$

It can be observed from the above equations that in L1 regularization, the regularization term's contribution to the gradient does not scale linearly with each w_i but it is a constant factor with a sign equal to $sign(w_i)$ and this is the main difference between L1 and L2 regularization.(Goodfellow et al., 2016)

] .

4. Balanced EMNIST Experiments

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

]

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

]

Here we evaluate the effectiveness of the given regularization methods for reducing the overfitting on the EMNIST dataset. We build a baseline architecture with three hidden layers, each with 128 neurons, which suffers from overfitting in EMNIST as shown in section 2. We follow the previous training settings where we deliberately let the baseline overfit on the training set as in previous experiments. These settings ensure the fairness of the evaluation of three methods to alleviate overfitting. 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.

[Experiments were conducted for 16 different combinations of hyperparameters. For this 4 values of include probability were taken for implementing dropout - 0.95,0.85,0.8,0.75 along with 2 values of penalty 1e-5,1e-4 trained for both L1 and L2 for each value of include probability. The experiments were conducted on a model with 3 hidden layers and 128 units in each hidden layer, with a batch size of 100 and a learning rate of 1e-3. Additionally, both bias and weight penalties were utilised on each layer of the model.

It was observed that, the best results were obtained for the model with an include probability of 0.95, penalty of 1e-4 using L2 weight decay, the model achieved a validation accuracy of 0.862 and a generalization gap of 0.1698.

From the table and the generated figure it can be observed that the models with include probabilities of 0.95 and 0.85 for dropout performed the best, consistently achieving validation accuracies of 0.85+ while having optimally low generalization gap. This is intuitive since we apply dropout to each layer of the model and for lower values of include probability, we would be lowering the size of parameters of the model causing the models to become less complex and lowering their capacity, which could lead to underfitting, from the experiments we can conclude that the optimal range of include probabilities for dropout lies in between 0.85-0.95.

Values of 1e-4 and 1e-5 were chosen for the penalties since models trained with L1 weight decay with penalties greater than 1e-4 tended to underfit, whereas models trained with L2 weight decay achieved the highest validation accuracy for penalty value of 1e-3, therefore, training the hyperparameter combinations with penal-

Model	Hyperparameter value(s)	Validation accuracy	Generalization gap
Baseline	-	0.823	1.570
Dropout	0.1	0.0390	0.0045
	0.3	0.6374	0.0071
	0.5	0.7727	0.0168
	0.7	0.8374	0.0537
	0.9	*0.8559*	*0.1874*
L1 penalty	1e-5	0.8331	0.8118
	1e-4	*0.8462*	*0.0955*
	1e-3	0.7553	0.0180
	1e-2	0.0212	4.132e-07
	1e-1	0.0215	1.032e-06
L2 penalty	1e-5	0.8222	1.2404
	1e-4	0.8363	0.5198
	1e-3	*0.8498*	*0.0695*
	1e-2	0.7596	0.0077
	1e-1	0.0212	5.45e-05
Combined	0.95, L1 1e-5	0.8610	0.2142
	0.95, L2 1e-5	0.8522	0.2744
	0.95, L1 1e-4	0.8524	0.0460
	0.95, L2 1e-4	**0.8620**	**0.1698**
	0.85, L1 1e-5	0.8597	0.0826
	0.85, L2 1e-5	0.8604	0.1106
	0.85, L1 1e-4	0.8406	0.0236
	0.85, L2 1e-4	0.8570	0.0726
	0.8, L1 1e-5	0.8565	0.0644
	0.8, L2 1e-5	0.8545	0.0781
	0.8, L1 1e-4	0.8312	0.0174
	0.8, L2 1e-4	0.8537	0.0514
	0.75, L1 1e-5	0.8466	0.0443
	0.75, L2 1e-5	0.8470	0.0597
	0.75, L1 1e-4	0.8250	0.0136
	0.75, L2 1e-4	0.8419	0.0394

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

ties of 1e-5 and 1e-4 provided a more fair comparison to combinations involving L1 with those involving L2

] .

5. Literature Review: Maxout Networks

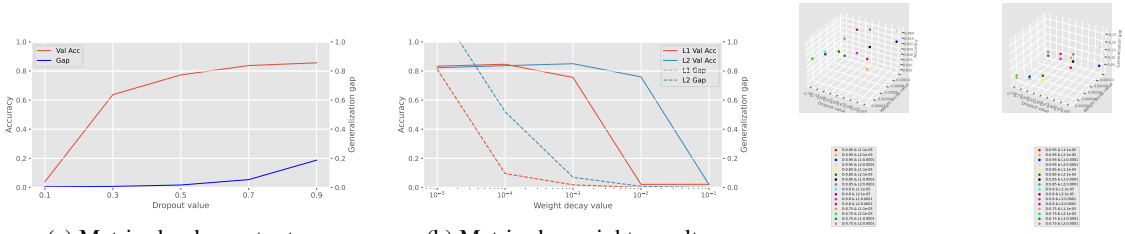
Summary of Maxout Networks In this section, we briefly discuss another generalization method: Maxout networks (Goodfellow et al., 2013). This paper further explores the dropout method and proposes a new "maxout" layer which can complement dropout. The authors evaluate the performance of Maxout Networks 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 researchers mentioned the limitations of dropout in not being able to carry out model averaging for deep architectures and suggested that instead of using dropout as a slight enhancement for arbitrary models, it was best to design a model that would enhance the model averaging techniques of dropout as well as have beneficial characteristics in optimization.]** . 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.

[A maxout unit can learn a piecewise linear, convex function with up to k pieces which makes maxout layers universal approximators .

For example, when k=2 the ReLU activation function can be approximated, and the maxout unit computes the following -

$max(\mathbf{w}_1^\top \mathbf{x} + \mathbf{b}_1, \mathbf{w}_2^\top + \mathbf{b}_2)$

Of which ReLU is a special case with $\mathbf{w}_1 = 0$ and $\mathbf{b}_1 = 0$



(a) Metrics by dropout rate (b) Metrics by weight penalty (c) Extra experiments

Figure 4. Hyperparameter search for every method and combinations

In fact, the researchers claim that two maxout units can approximate any continuous function which makes it ideal to be used along with dropout and can help dropout carry out approximate weight averaging which is something dropout does after training every random subnetwork for an iteration.] .

Strengths and limitations The author proposed a novel neural activation unit that further exploits the dropout technique. **[On the MNIST dataset, the authors trained a model consisting of two densely connected maxout layers followed by a softmax layer on 60,000 training and 10,000 test samples. The regularized the model with dropout and selected the hyperparameters by minimizing the error on the validation set which consisted of the last 10,000 training images. To make full use of the training set, they made use of the log likelihood on the first 50,000 examples at the point of minimal validation error and then continued training on the enitre dataset of 60,000 images until the validation set log likelihood matched this number and achieved a test error of 0.94e-02.**

They used a similar procedure for the CIFAR10 dataset, however this time they trained the model from scratch instead and stopped when the new likelihood matched the old one. Their best model consisted of 3 convolutional maxout layers, a fully connected maxout layer and a fully connected softmax layer.] .

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

[Maxout networks although provided a great addition to dropout layers essentially double the number of parameters greatly increasing model complexity. Although maxout layers are being consistently used in all the State Of The Art Convolutional Neural Networks in the form of a Max Pooling Layer, I think that it is not something that is capable of replacing activation functions like ReLU or PReLU or Swish due to the complexity it adds to the model resulting in overfitting] .

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.