MLP Coursework 1

B133237

Abstract

In this report we study the problem of overfitting, which is the training regime where performance increases on the training set but decrease on validation data. **[Overfitting is a problem as it means that our model has learned the noise present in the training data which leads to poor generalisation and inaccurate predictions for 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 **[improved the performance of the models initially, but led to a greater level of overfitting over 100 epochs]** . Next we discuss how 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 **[each of these techniques reduced the effect of overfitting when applied to the model. The best performing model was dropout with a dropout value of 0.85, this gave us 85.1% validation accuracy and a generalisation gap of 0.105.]** . Finally, we conclude the report with our observations and related work. Our main findings indicate that **[techniques such as dropout, L1, and L2 regularisation were effective in reducing the effects of overfitting resulting in models that generalised and performed better to unseen data. The best performing model was found to be dropout with a dropout value of 0.85.]** .

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 analysing the given problem in Figure 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 generalisation gap (see Ch. 5 in Goodfellow et al. 2016) and generalisation performance. Section 3 introduces two regularisation 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[1] neural network, train it on the EMNIST dataset, which contains 131,600 images of characters and digits, each of size 28x28, from 47 classes. We evaluate them in terms of generalisation gap and performance, and discuss the results and effectiveness of the tested regularisation strategies. Our results show that **[each of these techniques reduced the effect of overfitting when applied to the model. The best performing model was dropout with a dropout value of 0.85, this gave us 85.1% validation accuracy and a generalisation gap of 0.105.]** .

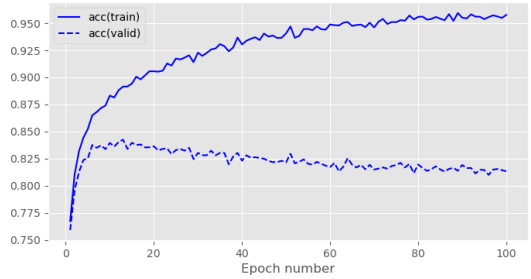
Finally, we conclude our study in Section 5, noting that **[techniques such as dropout, L1, and L2 regularisation were effective in reducing the effects of overfitting resulting in models that generalised and performed better to unseen data. The best performing model was found to be dropout with a dropout value of 0.85.]** .

2. Problem identification

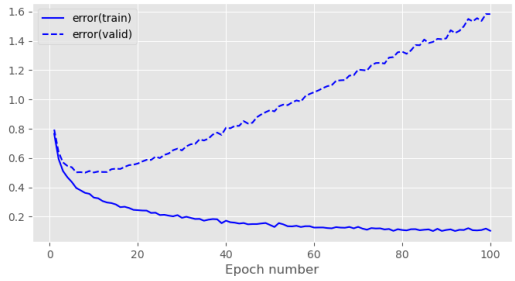
Overfitting to training data is a very common and important issue that needs to be dealt with when training neural networks or other machine learning models in general (see Ch. 5 in Goodfellow et al. 2016). A model is said to be overfitting when as the training progresses, its performance on the training data keeps improving, while its is degrading on validation data. Effectively, the model stops learning related patterns for the task and instead starts to memorize specificities of each training sample that are irrelevant to new samples.

[Overfitting occurs when a model becomes too specific, learning the noise and variance in training data that is not representative of the underlying pattern which we are trying to learn. The main goal of machine learning is to develop models that can make accurate predictions or decisions when presented with new data, however, the over-specificity caused by overfitting means that the model will not generalise well to unseen data and will therefor give inaccurate predictions. Strictly speaking overfitting is inevitable in all gradient-based training, however, there are various methods which can reduce

[1] We denote all layers as hidden except the final (output) one. This means that depth of a network is equal to the number of its hidden layers + 1.



(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.

its effect or catch it early in order to stop it affecting a models results. As it is inevitable, understanding how to implement these techniques is extemely important to achieve good results.] .

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 parameters the model has, the easier it will be to memorise complex data patterns that only apply to a restricted amount of samples. A prominent symptom of overfitting is the generalisation 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.

[The network capacity of a neural network refers to it's ability to approximate a range or scope of functions - it's flexibility and complexity. The network capacity of a neural network is changed by increasing or decreasing the width - the number of nodes in the network, and the depth - the number of hidden layers. It is relevant to overfitting as if the capacity of the network is too large, then this allows the model to try and perfectly fit the training data as opposed to representing the underlying pattern. This causes the model to overfit to the training data and generalise poorly. Similarly, if the capacity is not large enough this leads to underfitting, where the model is not flexible enough to capture the pattern in the data which will again generalise poorly.] .

Figure 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that **[initially the training and validation accuracy climb at a similar rate as the model improves at fitting the underlying pattern in the data. However, at around epoch 6 we see the velocity of the validation accuracy change dramatically, with the rate of increase tailing off before it reaches its maximum value at around epoch 10. After this point the validation accuracy starts to decrease, while the training accuracy continues to improve as we continue each epoch. This shows that the model is overfitting to the training data getting better and better at representing the training data while getting worse at generalising to the validation data.**

We see a similar pattern in figure 1b. Initially both lines are on a similar path, however, the validation error starts to change shape at around epoch 6, slowing down in terms of it's improvement. The minimum validation error is reached at around epoch 9, at which point the error starts to consistently increase. From this point onwards the training error continues to improve and the validation error gets worse. This is symptomatic of overfitting where the model fits the training data better and better, improving the error, but generalises worse to the validation data causing it's error to 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 sufficiently many diverse training samples, or if our model contains few hidden units, 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.	Train Error	Val. Error
32	78.4	0.566	0.713
64	80.8	0.347	0.651
128	80.6	0.160	0.920

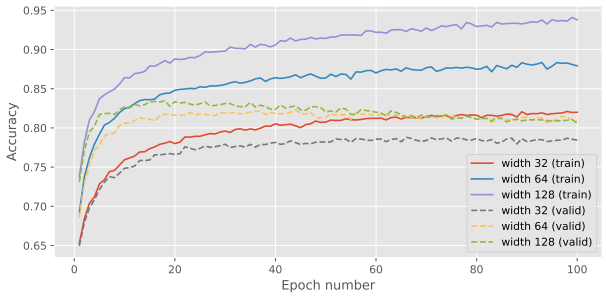
Table 1. Validation accuracy (%) and training/validation error (in terms of cross-entropy error) for varying network widths on the EMNIST dataset.

2.1. Network width

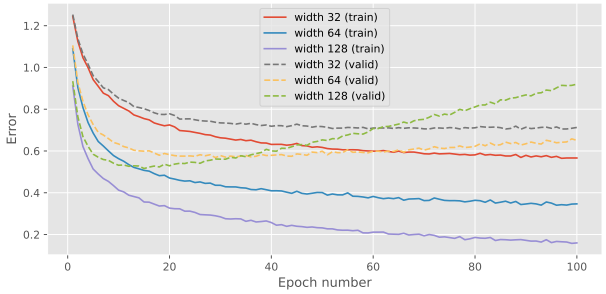
[Question Table 1 - Fill in Table 1 with the results from your experiments varying the number of hidden units.] [Question Figure 2 - Replace the images in Figure 2 with figures depicting the accuracy and error, training and validation curves for your experiments varying the number of hidden units.]

First we investigate the effect of increasing the number of hidden units in a single hidden layer network when training on the EMNIST dataset. The network is trained using the Adam optimiser with a learning rate of 9×10^{-4} 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 respec-



(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.

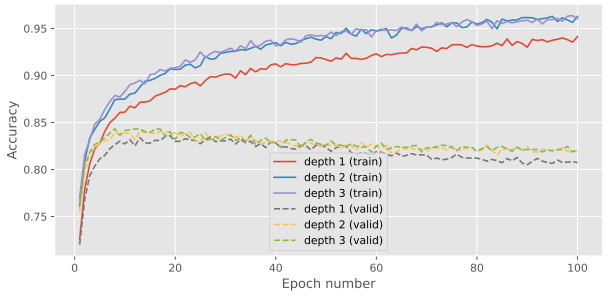
tively. Figure 2 depicts the error and accuracy curves over 100 epochs for the model with varying number of hidden units. Table 1 reports the final accuracy, training error, and validation error. We observe that **[increasing the number of hidden units consistently improves the training error, however while increasing hidden units from 32 to 64 improves the validation accuracy and error we see that increasing it again from 64 to 128 actually makes the validation accuracy and error slightly worse at epoch 100. This is the result of increasing the complexity of the model - allowing it to capture more of the noise in the data and therefore increasing the level of overfitting.**

However, the results in table 1 don't paint the full picture. Looking at figure 2 we can see that increasing the number of hidden units allows the model to capture a more complex underlying pattern, this initially leads to a more optimal maximum accuracy and minimum error observed at aroundepoch 15 for accuracy and epoch 10 for error. However, once overfitting begins to occur the rate of overfitting is higher in the width 128 model compared to the width 64 model eventually leading to worse performance in accuracy and error at epoch 100. Essentially, increasing the complexity of the model gives us a higher ceiling for performance, but once the model starts overfitting it does so at a faster rate eventually resulting in worse performance.] .

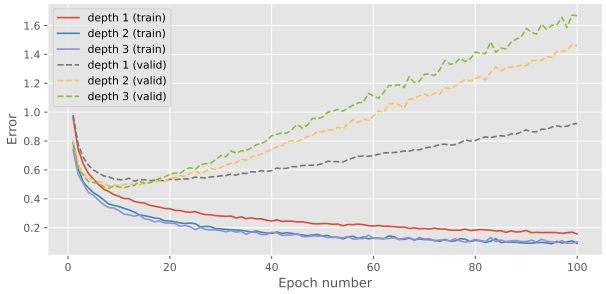
[From figure 2 we can say that increasing the width of the model consistently improves performance on training data While this initially translates to generalising better to unseen data - improving the peak performance

# Hidden Layers	Val. Acc.	Train Error	Val. Error
1	80.7	0.157	0.921
2	81.9	0.091	1.460
3	82.1	0.101	1.67

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



(a) accuracy by epoch



(b) error by epoch

Figure 3. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for different network depths.

of the model - once overfitting begins it occurs at an increased rate. This is an intuitive result given the relationship between model complexity and overfitting. By increasing the width of the model, we increase it's complexity, allowing the model to better learn the patterns in the training data. This leads to the initial improvement in performance. However, there is a trade off and as the number of epochs increases we give the model more opportunity to fit the training data perfectly leading to the poor generalisation and poor validation error and accuracy.] .

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 Figure 3 depict

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 optimiser with a learning rate of 9×10^{-4} and a batch size of 100.

We observe that **[increasing the number of hidden layers leads to an improvement in validation accuracy, showing that the model generalises better to unseen data when there are 3 hidden layers. In terms of validation error we see from table 2 that by epoch 100 the error increases as we increase the number of layers. However looking at figure 3 we see a similar pattern to our width experiments: increasing the depth initially leads to improved performance in terms of validation error. However, after around 10 epochs the model begins overfitting, at which point the rate of overfitting is higher with the more layers we have.]** .

[Increasing the number of layers consistently improves the initial performance of the models; leading to a better peak performance in both error and accuracy as we increase the depth. For accuracy the rate of change as we increase epoch number is fairly consistent regardless of depth, however, in terms of error we see that as we increase the depth the rate at which error increases once overfitting begins to occur is increased by increasing the depth. A recent literature review https://ieeexplore.ieee.org/document/9318195?denied= found that 3 tends to be the optimum number of layers in terms of time complexity and accuracy, so it should come as no surprise that 3 layers performed better in terms of it's maximum accuracy and minimum error that 1 and 2 layer models. The increase in complexity of the model allows us to better capture the patterns in the data leading to better optimal performance.] .

3. Dropout and Weight Penalty

In this section, we investigate three regularisation 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 (*i.e.* the rate that an unit is included). Dropout is commonly represented by an additional layer inserted between the linear layer and activation function. Its forward pass during training is defined as follows:

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

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

where $\boldsymbol{y}, \boldsymbol{y}' \in \mathbb{R}^d$ 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 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 p:

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

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:

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

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 units between layers so that the neurons of the next layer will not rely on only few features from of the previous layer. Instead, it forces the network to extract diverse features and 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 regularisation (Ng, 2004) are simple but effective methods to mitigate overfitting to training data. The application of L1 and L2 regularisation strategies could be formulated as adding penalty terms with L1 and L2 norm square of weights in the cost function without changing other formulations. The idea behind this regularisation method is to penalise 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 optimisation problem takes a different form:

$$\text{L1: } \min_{\boldsymbol{w}} E_{\text{data}}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w}) + \lambda \|w\|_1 \tag{5}$$

$$\text{L2: } \min_{\boldsymbol{w}} E_{\text{data}}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w}) + \lambda \|w\|_2^2 \tag{6}$$

where E_{data} denotes the cross entropy error function, and $\{\boldsymbol{X}, \boldsymbol{y}\}$ denotes the input and target training pairs. λ controls the strength of regularisation.

Weight penalty works by constraining the scale of parameters and preventing them to grow too much, avoiding overly sensitive behaviour on unseen data. While L1 and L2 regularisation are similar to each other in calculation, they have different effects. Gradient magnitude in L1 regularisation 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 regularisation tends to shrink the weights to a smaller scale uniformly.

[L1 and L2 regularisation can be used in conjunction in a method called elastic net regularisation. Elastic net

regularisation offers a compromise between L1 and L2 allowing us to take the benefits of both, this is particularly useful in mitigating overfitting. The L1 component will push small weights towards 0, acting as a feature selection mechanism and removing features from the model entirely. This helps to reduce overfitting as it reduces the model complexity, making it less likely to fit small fluctuations in the trainnig data. On the other hand the L2 component will reduce model complexity by penalizing large coefficients or weights in the model. This encourages the model to distribute weight between features and not rely too heavily on specific ones, in the case of overfitting this means the model is more likely to fit the general pattern than just the exact features in the training data. By combining L1 and L2 the elastic net method allows us to control the number of features and distribute the contribution of each feature in our model helping us to reduce the chances of our model overfitting.] .

4. Balanced EMNIST Experiments

[Question Table 3 - Fill in Table 3 with the results from your experiments for the missing hyperparameter values for each of L1 regularisation, L2 regularisation, and Dropout (use the values shown on the table).

]

[Question Figure 4 - Replace these images with figures depicting the Validation Accuracy and Generalisation Gap (difference between validation and training error) for each of the experiment results varying the Dropout inclusion rate, and L1/L2 weight penalty depicted in Figure 3 (including any results you have filled in).]

Here we evaluate the effectiveness of the given regularisation 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 as shown in section 2.

Here we train the network with a lower learning rate of 10^{-4}, as the previous runs were overfitting after only a handful of epochs. Results for the new baseline (c.f. Table 3) confirm that lower learning rate helps, so all further experiments are run using it.

Then, we apply the L1 or L2 regularisation with dropout to our baseline and search for good hyperparameters on the validation set. We summarise 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.

[For dropout, the hyperparameter we adjusted was the inclusion probability, representing the likelihood that a node within a layer is deactivated during training. Figure 4 indicates that dropout values of 0.85 and 0.95 yield the highest validation accuracy. However, a dropout value of 0.95 results in a significantly larger generalization gap, implying a greater discrepancy in error between the validation and test datasets. Taking both accuracy and the generalization gap into consideration, a dropout value of 0.85 emerges as the optimal choice, striking a balance between robust validation accuracy and a minimal generalization gap. For L1 and L2 regularisation the hyperparameter were changing is the weight decay value which changes the level of penalty added to the loss function, regulating how much of an impact we want L1 or L2 to have. From figure 4 and table 3 we can see that both start with high validation accuracy which decreases rapidly as we increase the penalty weight. Increasing the penalty weight also results in a reduction in the generlisation gap. We can see there is a sweet spot for both models where validation accuracy is high and generalisation gap is low. For L1 this comes at 5e-4 and for L2 this comes at 1e-3.] .

[The experiments I would perform would aim to investigate first the combination of dropout and L1, then the combination of dropout and L2, and finally the combination of dropout, L1, and L2. This would give me a good general picture of how L1 and L2 interact with dropout and what the best combination would be. The hyperparameters I would select would be based upon the best performing models in the experiments ran in this project. The questions I would aim to answer would be: 1) How does the combination of Dropout and L1/L2 regularisation affect model performance compared to using them individually? 2) Does combining all three techniques lead to better generalisation than just using any two? 3) Are there adverse impacts when stacking too many regularisation techniques? The experiments would be as follows: Experiment 1: Dropout + L1: Dropout = 0.85, L1 = 5e-4.

Experiment 2: Dropout + L1: Dropout = 0.85, L1 = 1e-3.

Experiment 3: Dropout + L2: Dropout = 0.85, L2 = 5e-4.

Experiment 4: Dropout + L2: Dropout = 0.85, L2 = 1e-3.

Experiment 5: Dropout + L1 + L2: Dropout = 0.85, L1 = 5e-4, L2 = 5e-4.

Experiment 6: Dropout + L1 + L2: Dropout = 0.85, L1 = 1e-3, L2 = 5e-4.

Experiment 7: Dropout + L1 + L2: Dropout = 0.85, L1 = 5e-4, L2 = 1e-3.

Experiment 8: Dropout + L1 + L2: Dropout = 0.85, L1 = 1e-3, L2 = 1e-3.] .

5. Conclusion

Model	Hyperparameter value(s)	Validation accuracy	Train Error	Validation Error
Baseline	-	0.837	0.241	0.533
Dropout	0.6	80.7	0.549	0.593
	0.7	78.9	0.647	0.660
	0.85	**85.1**	**0.329**	**0.434**
	0.97	85.4	0.244	0.457
L1 penalty	*5e-4*	*79.5*	*0.642*	*0.658*
	1e-3	73.8	0.889	0.895
	5e-3	2.41	3.850	3.850
	5e-2	2.20	3.850	3.850
L2 penalty	5e-4	85.1	0.306	0.460
	1e-3	*85.0*	*0.359*	*0.453*
	5e-3	81.3	0.586	0.607
	5e-2	39.2	2.258	2.256

Table 3. Results of all hyperparameter search experiments. *italics* indicate the best results per series (Dropout, L1 Regularisation, L2 Regularisation) and **bold** indicates the best overall.

[Overfitting is a major issue in training deep neural networks, and while it is to a certain extent inevitable there are many ways of handling and mitigating its effects. Overfitting occurs when a model learns the training data too well, to the extent of capturing noise and outliers, resulting in poor generalization to new, unseen data. In our initial experiments, the absence of regularisations yielded models that often fit the training data exceptionally well but underperformed on the validation data.

However, we showed that techniques such as dropout, L1, and L2 regularisation were effective in reducing the effects of overfitting resulting in models that generalised and performed better to unseen data.

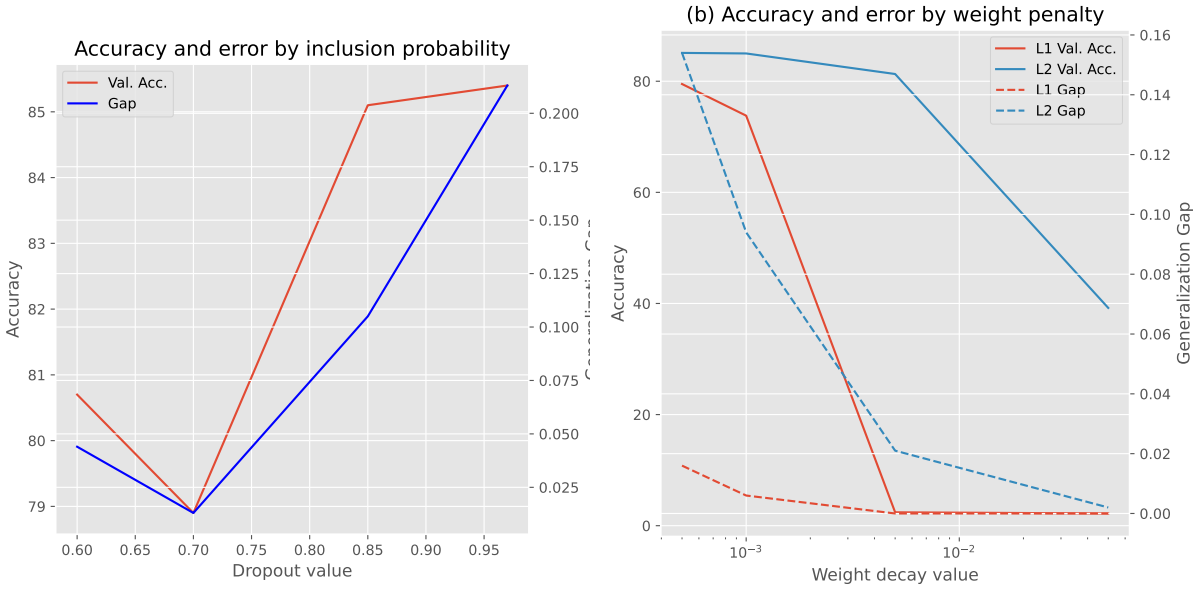
A potential future direction would be to investigate the effect of combining the regularisation techniques and to apply other techniques such as noise injection and adversarial training.] .

References

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.



(a) Accuracy and error by inclusion probability.

(b) Accuracy and error by weight penalty.

Figure 4. Accuracy and error by regularisation strength of each method (Dropout and L1/L2 Regularisation).