MLP Coursework 1

s2298839

Abstract

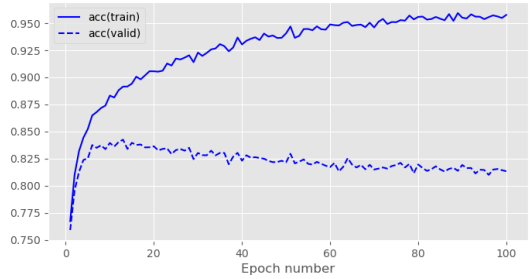
In this report we study the problem of overfitting, which is the training regime where performance increases on the training set but decrease on validation data. **[When overfitting occurs, the model can only be able to fit well on the training set(we do have labels/results) and unable to generalize well to new data(we do not know, results to be discovered). So the model can not perform the classification or prediction tasks that it was intended for]** . 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 **[aggravates the effects of overfitting.]** . Next we discuss why two standard methods, Dropout and Weight Penalty, can mitigate overfitting, then describe their implementation and use them in our experiments to reduce the overfitting on the EMNIST dataset. Based on our results, we ultimately find that **[the use of dropout can reduce the impact of overfitting. When the probability of a neuron to be included decreases, the gap between training error and validation error will become smaller, which means the overfitting problem is mitigated. However, in such case the performance of the model(we think higher validation accuracy means better performance) gets worse. As is the same, the use of both L1 penalty and L2 penalty can mitigate problem of overfitting, but it will severely degrades the performance of model when the penalty hyperparameter gets too big.]** . Finally, we briefly review a technique that studies use of weight decay in adaptive gradient algorithms, discuss its findings, and conclude the report with our observations and future work. Our main findings indicate that **[1: When quantity of parameters increases(biggernetwork width or depth), the model can fitting better on training set. However it will generate overfitting problem.**
2: Dropout and L1, L2 regularization can mitigate the problem of overfitting, but low probability to include units or setting the hyperparameter of penalty too high will do harm to the performance of the model.] .

1. Introduction

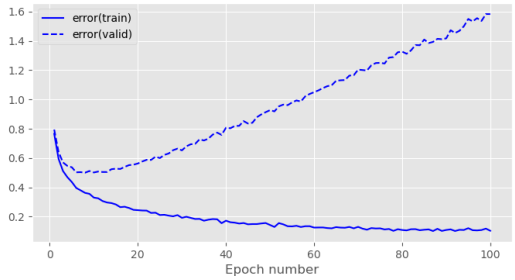
In this report we focus on a common and important problem while training machine learning models known as overfitting, or overtraining, which is the training regime where performances increase on the training set but decrease on unseen data. We first start with analyzing the given problem in Fig. 1, study it in different architectures and then investigate different strategies to mitigate the problem. In particular, Section 2 identifies and discusses the given problem, and investigates the effect of network width and depth in terms of generalization gap (see Ch. 5 in Goodfellow et al. 2016) and generalization performance. Section 3 introduces two regularization techniques to alleviate overfitting: Dropout (Srivastava et al., 2014) and L1/L2 Weight Penalties (see Section 7.1 in Goodfellow et al. 2016). We first explain them in detail and discuss why they are used for alleviating overfitting. In Section 4, we incorporate each of them and their various combinations to a three hidden layer[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 generalization gap and performance, and discuss the results and effectiveness of the tested regularization strategies. Our results show that **[the use of dropout can reduce the impact of overfitting. When the probability of a neuron to be included decreases, the gap between training error and validation error will become smaller, which means the overfitting problem is mitigated. However, in such case the performance of the model(we think higher validation accuracy means better performance) gets worse. As is the same, the use of both L1 penalty and L2 penalty can mitigate problem of overfitting, but it will severely degrades the performance of model when the penalty hyperparameter gets too big.]** . In Section 5, we study a related work that studies weight decay in adaptive gradient algorithms.[2] Finally, we conclude our study in section 6, noting that **[1: When quantity of parameters increases(biggernetwork width or depth), the model can fitting better on training set. However it will generate overfitting problem.**
2: Dropout and L1, L2 regularization can mitigate the problem of overfitting, but low probability to include units or setting the hyperparameter of penalty too high will do harm to the performance of the model.] .

[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.

[2] 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 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.

[Since the goal of the model is to extract of the commonalities of some classes from the training set, and use these commonalities to do prediction or classification on new data(same distribution as training set). While overfitting leads the model to memorizing too many details(like the outliers, or small disturbances due to noise), which results in a more complex model and weakens generalization of data commonalities in general. For example, considering the task of human identification intuitively. If the "real" human in training set all wear red clothes. When overfitting occurs, although the model can correctly recognize almost every person in training set, it may classify a naked person(in new data set) as non-human, despite the person has enough commonality with "real" human. For the model remembers more commonalities than the model without overfitting do] .

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 memorize complex data patterns that only apply to a restricted amount of samples. A prominent symptom of overfitting is the generalization gap, defined as the difference between the validation and training error. A steady increase in this quantity is usually interpreted as the model entering the overfitting regime.

[When the "network capacity"(the width and depth of a network) becomes larger, we think the network is more complex. Complex network has a more powerful ability to fit the data, which often demonstrated by higher training accuracy. However, too many network parameters will lead to the model fitting "too close" to the training data, which means the model extracts more information than it should do. So when the model is applied on new data, it may perform much worse than on the training data set. And the gap between the performance on training and validation dataset is called overfitting problem.] .

Fig. 1a and 1b show a prototypical example of overfitting. We see in Fig. 1a that **[The Fig.1a contains the accuracy(training and validation) by epoch and the Fig.1b contrains the error(training and validation)by epoch, the range of epoch is from 0 to 100. In Fig.1a, the training accuracy increases fast before epoch 10 and after that it increases more slowly and get to the peak of about 0.950 at epoch 90. The validation accuracy increases at the beginning of training and reaches the maximum of 0.835 at around epoch 10. After epoch 10 the validation continuous to decline to 0.810. In Fig.1b, the training error decreases fast before epooch 10 and after that it decreases more slowly and get to the minimum of about 0.05 at epoch 90. The validation error decreases and get to the minimum of 0.5 at epoch 10. After epoch 10, the validation error grows almost linearly with respect to epoch and grows rapidly. So the overfitting happens at epoch 10, and after that the gap between training error and validation error gets larger and larger.]** .

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.

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.]

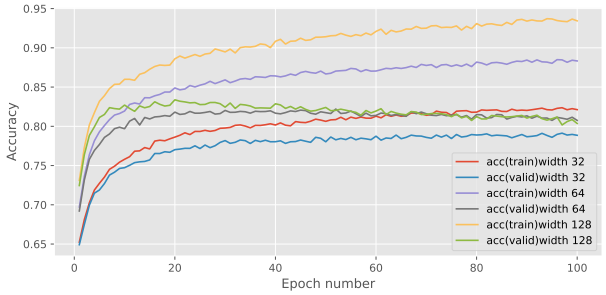
First we investigate the effect of increasing the number of hidden units in a single hidden layer network when training

# Hidden Units	Val. Acc.	Train Error	Val. Error
32	78.9	0.563	0.705
64	80.8	0.337	0.660
128	80.4	0.170	0.907

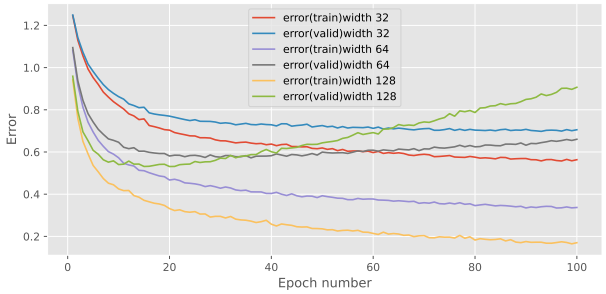
Table 1. Validation accuracy (%) and training/validation error (in terms of cross-entropy error) for varying network widths on the EMNIST dataset.

# Hidden Layers	Val. Acc.	Train Error	Val. Error
1	80.4	0.167	0.942
2	81.8	0.0925	1.46
3	81.5	0.105	1.68

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

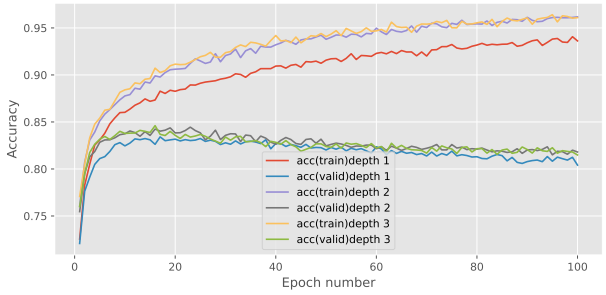


(a) accuracy by epoch

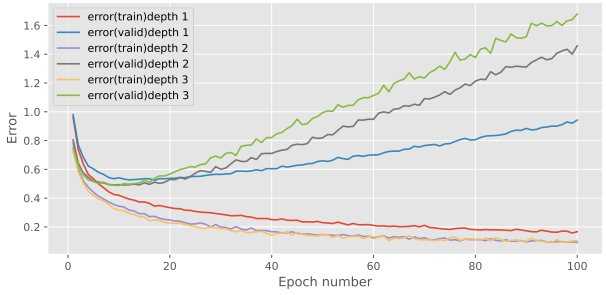


(b) error by epoch

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



(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.

on the EMNIST dataset. The network is trained using the Adam optimizer 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 respectively. Fig. 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 **[When using one hidden layer as training model, the 64 hidden units layer performs best(*Table 1*) because it has the highest validation accuracy. However, the gap between training error and validation error increases with the number of units(*Fig 2b*). In particular, after epoch 10, the gap becomes larger with the increase of epoch number.]** .

[The increase of network width enlarges the gap between training and validation error in a consistent way, which we can see from both (*Fig 2b*) and (*Table 1*). And the results are in line with the prior theory that complex model has a better capacity to fit while suffering from a higher risk of overfitting.].

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

We observe that **[When using fixed 128 units per layer as training model, the 2 hidden layer model show best performance(*Table 2*) due to highest validation accuracy. Similarly, the gap between training error and validation**

error increases as the number of layer increases from one to three(*Fig 3b*). After epoch 10, the gap becomes larger with the increase of epoch number. But unlike the change of units number, changing the number of layer does not have much impact on the training error and training accuracy(*Fig 2, Fig 3*)] .

[The increase of network depth enlarges the gap between training and validation error in a consistent way, which we can see from both (*Fig 3b*) and (*Table 2*). And the results are in line with the prior theory that complex model has a better capacity to fit while suffering from a higher risk of overfitting. Though when the number of hidden layers increase from 2 to 3, the training accuracy and training error do not show a explicit improvement, which I think maybe the 2 hidden layers model's fitting capacity is enough(model is complex enough), yet the more complex model leads to more severe overfitting.] .

[Using the validation accuracy as criterion for performance and gap between training and validation error as the degree of overfitting. As shown in (*Table 1, Table 2, Fig 2, Fig 3*), the increases of either width or depth will cause aggravates overfitting. While neither increasing width nor depth of the network can ensure a better performance(*Table 1, Table 2*). The best performance is obtained for both experiments with intermediate value of the hyperparameter variables(width and depth). As a result, even though the simple model has lowest risk of getting stuck in overfitting problem, and the complex model can fit training data quite well, we should select carefully because neither too simple nor too complex model shows competitive performance.] .

3. Dropout and Weight Penalty

In this section, we investigate three regularization methods to alleviate the overfitting problem, specifically dropout layers and the L1 and L2 weight penalties.

3.1. Dropout

Dropout (Srivastava et al., 2014) is a stochastic method that randomly inactivates neurons in a neural network according to an hyperparameter, the inclusion rate (*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 regularization (Ng, 2004) are simple but effective methods to mitigate overfitting to training data. The application of L1 and L2 regularization strategies could be formulated as adding penalty terms with L1 and L2 norm square of weights in the cost function without changing other formulations. The idea behind this regularization method is to penalize the weights by adding a term to the cost function, and explicitly constrain the magnitude of the weights with either the L1 and L2 norms. The optimization problem takes 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 regularization.

Weight penalty works by constraining the scale of parameters and preventing them to grow too much, avoiding overly sensitive behavior on unseen data. While L1 and L2 regularization are similar to each other in calculation, they have different effects. Gradient magnitude in L1 regularization does not depend on the weight value and tends to bring small weights to 0, which can be used as a form of feature selection, whereas L2 regularization tends to shrink the weights to a smaller scale uniformly.

[The grad of L1 regularisation is not related to the size of the weight, and it depend on the sign of the weight. So L1 regularisation can shrink the small weights to

zero. It is of great help when the input matrix is sparse, the L1 can select important features by setting some weights to 0. While L2 can effectively deal with large weights for the grad of weight is proportional to weight itself, so L2 can easily scaling weights to $(0, 1)$, which prevetes excessive weights from being too sensitive to data changes or outliers. So we can use both L1 and L2 by setting different λ for them($\lambda_1\|\omega\|_1 + \lambda_2\|\omega\|_2$). By tuning the hyperparameters, we can simultaneous achieve the goal of selecting important features and scaling the model parameters. Though the L1 is equivalent to the prior probability distribution of the parameter ω satisfying the Laplace distribution and the L1 is equivalent to the prior probability distribution of the parameter ω satisfying the Gaussian distribution, combining them may be quite challenging to find the proper λ, yet it is promising if suitable λ is found skillfully.] .

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 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 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 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.

[In the experiment, we use a 3 hidden layers network with 128 units in each layer as model. The size of each batch is set to 100, the probability of units included in dropout layer and λ for L1 and L2 is shown in Hyperparameter value(s) in *Table 3*. The first experiment in this section is to test the effect of different units included probability hyperparameter using dropout between hidden layers. Compared to Baseline, when hyperparameter $incl\_prob = 0.85$ and $incl\_prob = 0.85$, the performance of model is better than Baseline. While $incl\_prob = 0.6$ and $incl\_prob = 0.7$, the performance is worse than Baseline(*Table 3*). When $incl\_prob$ is close to 1, the performance is best among all the tests. Meanwhile, as shown in *Fig 4a*, when dropout value $incl\_prob$ gets smaller, the generalization gap will also shrink although with some trade-offs in performance. The second experiment in this section is to test the selection of hyperparameter λ for L1 penalty and L2 penalty. When λ is quite close to zero($5e-4$), both models using L1 penalty and L2 penalty achieve best performance among their own test space(*Table 3*). The *Fig 4b* shows the increase of weight decay value λ will lead to performance decreasing quickly, together with generalization gap.

] .

[Questions to answer:
1: If there exists a smaller λ by which the L1 or L2 penalty(single Hyperparameter test,just like the test in this section) can achieve better performance.
2: If I can find the combination of Dropout and L1 and/or Dropout and L2 to achieve the best performance(at least better than the bold result in *Table 3*) with possible low generalization gap(at least lower than the first test result of L2 in *Table 3*)
The first two experiments I will choose weight decay error $\lambda = 2e-4$ in both L1 and L2 to see if the lower λ brings better performance. After that I shall choose the best two performance λ among all the experiments did on L1 and L2. Then I will select $0.85, 0.91, 0.97$ as the dropout value list. Then the rest 6 experiments will be the 2(best performance weight decay hyperparameter) multiply 3(dropout value).
The reason why I choose this solution is that. After the previous experiments, I think the L2 is better at performance than L1 overall. But I still want to find if there is a better performance by using single L1 or L2 since the weight decay value still has space to decline and it is negatively correlated with the accuracy from *Fig 4b*. After selecting the best two weight decay value, I the rest 6 experiments can be used to arrange the combination of the hyperparameters.] .

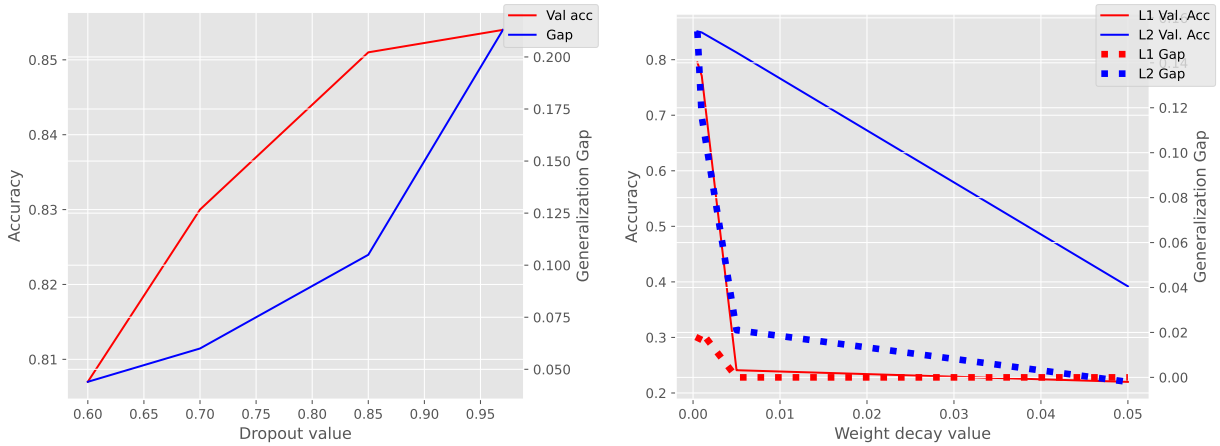
5. Literature Review: Decoupled Weight Decay Regularization

Summary In this section, we briefly study a method (Loshchilov & Hutter, 2019) that decouples the weight penalty from the weight update. The authors shed light onto the relation between weight decay and L2 weight penalty for SGD and Adam optimizers, pointing out that weight decay and L2 weight penalty are not equivalent when used with the Adam optimizer.

In particular, the authors claim that SGD with L2 weight penalty and weight decay are equivalent when their coeffi-

Model	Hyperparameter value(s)	Validation accuracy	Train Error	Validation Error
Baseline	-	83.7	0.241	0.533
Dropout	0.6	80.7	0.549	0.593
	0.7	83.0	0.442	0.502
	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	77.1	0.744	0.764
	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	84.9	0.334	0.450
	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 Regularization, L2 Regularization) and **bold** indicates the best overall.



(a) Accuracy and error by inclusion probability.

(b) Accuracy and error by weight penalty.

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

cients have the relation $\lambda' = \frac{\lambda}{\alpha}$ (see their **Proposal 1**), which they prove with Equations (5) and (6) (proof is in their Appendix A). In addition, they claim that such a simple scalar relation does not hold in Adam optimization (see **Proposal 2**) and show, in their Appendix A, that the necessary relation for equivalence requires use of a preconditioner matrix $\boldsymbol{M}_t$. This requirement can be explained with the fact that [

Lets define the weight of network as w, its gradient as $grad\_w$, and the hyperparameter $\lambda, \beta_1, \beta_2, \epsilon$.

intuitively, the L2 regularisation in Adam optimizer is that we first add the λw to the gradient $grad\_w$ and then compute a moving average of the gradients and their squares before using both of them for the update. While the weight decay will first compute the momentum of gradient(moving averages of the parameter w), then update and finally subtract a quantity proportional to λw. Due to the existence of the momentum(more complex in Adam than SGD with momentum because Adam has two momentums), the equivalence between L2 weight penalty and weight decay neeeds a quite complex matrix.

Here I give a simple derivation

In L2 weight penalty:

It first compute the gradients and moving average

$gradients \longleftarrow grad\_w + \lambda w$

$m_1 \longleftarrow \beta_1 m_1 + (1 - \beta_1) gradients$

$m_2 \longleftarrow \beta_2 m_2 + (1 - \beta_2) gradients^2$

The penalty information is involved in the gradients

$w \longleftarrow w - learning\_rate(m_1 / (\sqrt{m_2} + \epsilon))$

In weight decay:

The gradients equals to $grad\_w$

Directly compute the moving average

$m_1 \longleftarrow \beta_1 m_1 + (1 - \beta_1) grad\_w$

$m_2 \longleftarrow \beta_2 m_2 + (1 - \beta_2) grad\_w^2$

In the final step to introduce a decay

$w \longleftarrow w - learning\_rate(m_1 / (\sqrt{m_2} + \epsilon) + \lambda w)$

The m_1, m_2 in L2 weight penalty and weight decay are different matrix. So in order to equal L2 weight penalty

and weight decay, we need a quite complex matrix.] .

[In mlp.learning_rules.AdamLearningRule(), update_params() function: it neither realize the "Adam with L2 regularization" nor the AdamW. It views the L2 as a seperate layer with its own gradient, but not an extra bias implemented during the update procedure.] .

Finally the authors validate their findings and proposed decoupled optimization strategies in various learning rate schedules, initial learning rates, network architectures and datasets. [3]

6. Conclusion

[1: When quantity of parameters increases(biggernetwork width or depth), the model can fitting better on training set. However it will generate overfitting problem.
2: Dropout and L1, L2 regularization can mitigate the problem of overfitting, but low probability to include units or setting the hyperparameter of penalty too high will do harm to the performance of the model.
Take-away message:
1: Realize the principle and realization of Adam, AdamW.
2: Know how to build a multilayer model from scratch.
Future directions:
implement more complex model, and build AI architecture.] .

[3] Instructor note: Omitting this for this coursework, but normally you would give an evaluation of the experiment setup in (Loshchilov & Hutter, 2019) here.

References

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Loshchilov, Ilya and Hutter, Frank. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

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.