MSc Artificial Intelligence - Deep Learning

Assignment 1 - TensorFlow & Keras

Michael McAleer R00143621

Note: For ease of reading, loss, accuracy, and duration values for low level TensorFlow operations have been reduced to five decimal places. Results provided from Keras models offer results to four decimal places.

Part A

i. Results of training and test data after being pushed once through:

a. Layer 1: 100 neurons (ReLu activation function)

b. Layer 2: 1 neuron (Sigmoid activation function)

c. Learning Rate: 0.01 with Gradient Descent

	Loss	Accuracy
Training	0.79478	0.47791
Test	0.71141	0.47749
Duration (s)	0.42038	

ii. Results of training and test data after 40 epochs through:

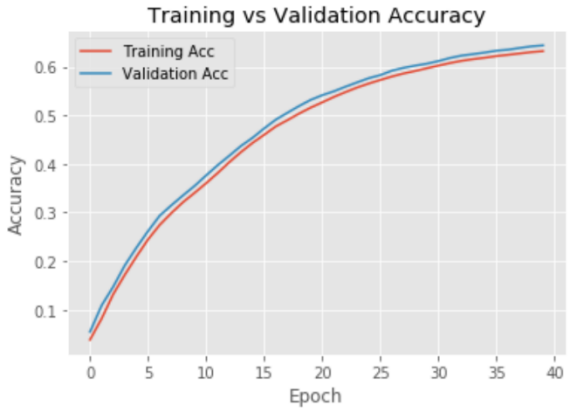
a. Layer 1: 300 neurons (ReLu activation functions).

b. Layer 2: 100 neurons (ReLu activation function)

c. Layer 3: Softmax Layer

d. Learning rate: 0.01 for this problem with Gradient Descent.

Iteration	Training Loss	Training Acc.	Validation Loss	Validation Acc.
1	2.42079	0.03805	2.37811	0.05433
5	2.16235	0.20933	2.14152	0.22783
10	1.98206	0.34040	1.96491	0.35466
15	1.83986	0.44348	1.82265	0.45399
20	1.71829	0.51603	1.70049	0.53200
25	1.61249	0.56561	1.59407	0.57666
30	1.51987	0.59705	1.50104	0.60650
35	1.43927	0.61877	1.42016	0.62949
40	1.36921	0.63231	1.34986	0.64416
Test	1.36358	0.62599		
Duration (s)	21.78385			



iii. Results of training and test data after 40 epochs using mini-batch size of 512 through:

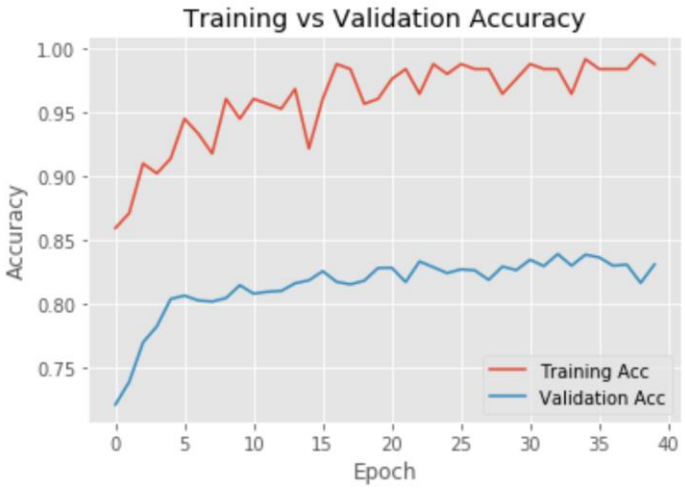
a. Layer 1: 300 neurons (ReLu activation functions).

b. Layer 2: 100 neurons (ReLu activation function)

c. Layer 3: Softmax Layer

d. Learning rate: 0.01 for this problem with Gradient Descent.

Iteration	Training Loss	Training Acc.	Validation Loss	Validation Acc.
1	0.69825	0.85937	0.88292	0.72066
5	0.33216	0.91406	0.57006	0.80366
10	0.25812	0.94531	0.52877	0.81449
15	0.27133	0.92187	0.53690	0.81833
20	0.20634	0.96093	0.52756	0.82800
25	0.14964	0.98046	0.53272	0.82400
30	0.13168	0.97656	0.55324	0.82633
35	0.11316	0.99218	0.51172	0.83850
40	0.10326	0.98828	0.52402	0.83083
Test	0.56705	0.81790		
Duration (s)	23.29650s			



Comparison of mini-batch gradient descent on accuracy and loss after 1 and 40 epochs:

	Benchmark (no mini-batch) Results			Mini-Batch Results		
	Training	**Test**	**Validation**	**Training**	**Test**	**Validation**
Loss (1)	2.42079	-	2.37811	0.69825	-	0.88292
Loss (40)	1.36921	1.36358	1.34986	0.10326	0.56705	0.52402
Accuracy (1)	0.03805	-	0.05433	0.85937	-	0.72066
Accuracy (40)	0.63231	0.62599	0.64416	0.98828	0.8179	0.83083

Implementing mini-batch gradient descent (MBGD) in the training process results in dramatic improvements across both loss and accuracy in the training, test, and validation data sets. In the training data, after the first epoch alone the training accuracy improves from 3.8% to 85.9%, and after 40 epochs from 63.2% to 98.8%. These improvements after one epoch show the significant impact that pushing mini-batches through the network has, updating the weights after each batch results in a far greater accuracy whilst having negligible impact on the total duration of the training process without MGBD – 21s without and 23s with.

Test and validation accuracy after 40 epochs improve from 62% to 81% in the test data, and 64% to 83% in the validation data set. The validation accuracy does not see the same rate of improvement as the training data, this could be due to the weights being updated multiple times for training data and thus becoming slightly overfit on those data points compared to the data within the validation set, however this behaviour if present is not significant as the loss values do not start to increase as the batches and epochs progress through the training process.

The loss values across all data sets greatly increases with the implementation of MBGD, after one epoch the training loss reduces from 2.42 to 0.698, and after forty from 1.369 to 0.103. The validation loss improves from 2.378 to 0.882 after one epoch, and from 1.349 to 0.524 after forty. The test loss values after all epochs are complete improves from 1.363 to 0.567.

This improvement in both loss and accuracy values with MBGD demonstrate the power of smaller batch sizes when training a neural network, the advantages gained through multiple updates of the weight and bias values per batch instead of just per epoch make it a very valuable inclusion in any neural network training process.

Part B

Note: In all tests run in part B, there is a validation split of 0.9/0.1 on the training data and a batch size of 256.

i) **Network Benchmark**

Results of training and test data after 1 epoch using:

a. Layer 1: Softmax classifier

	Loss	Accuracy
Training	0.7464	0.8034
Test	0.5769	0.8510
Validation	0.6603	0.8255

ii) **Multi-layer neural networks**

Two Layers

Results of training and test data after 15 epochs using:

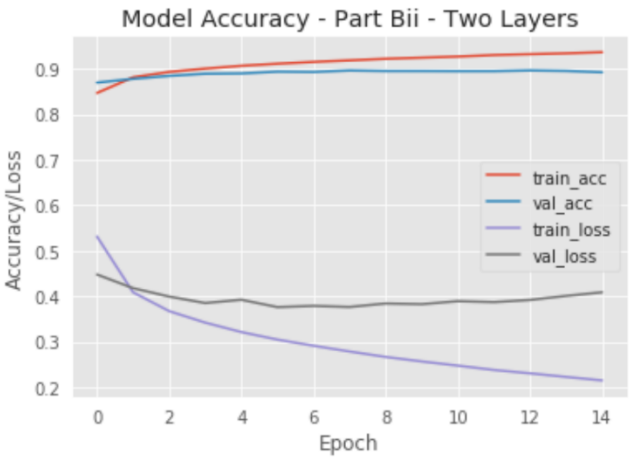
a. Layer 1: 200 neurons (ReLu activation functions)

b. Layer 2: 10 neurons (Softmax classifier)

Epoch	Training Loss	Training Acc.	Val. Loss	Val. Acc.
1	0.5313	0.8464	0.4478	0.8692
5	0.3213	0.9061	0.3923	0.8895
10	0.2567	0.9240	0.3825	0.8946
15	0.2151	0.9361	0.4087	0.8920
Test	0.3251	0.9128		

The accuracy across the training, test, and validation increased with the addition of a ReLu layer consisting of 200 neurons.

After the first epoch, similar increases were seen in both the training and validation results. The training accuracy had increased by 4.3% from 0.8034 to 0.8464, and the validation accuracy had increased by from 4.5% 0.8255 to 0.8692



After fifteen epochs the training accuracy had increased by 13.3% from the benchmark of 0.8034 to 0.9361, the validation accuracy had increased by 6.7% from 0.8255 to 0.8920. The test accuracy had increased by 6.1% from 0.8510 to 0.9128, giving similar increases as the validation data. The rate of improvement seen by both the test and validation data gives a truer indication of accuracy improvement. The rate of improvement seen by the training accuracy of 13.3% is an indication that even with only one layer of ReLu activation functions overfitting on the training data is becoming a factor.

Examining deeper neural networks with three, four, and five layers

Three layers

Results of training and test data after 15 epochs using:

a. Layer 1: 400 neurons (ReLu activation functions)
b. Layer 2: 200 neurons (ReLu activation functions)
c. Layer 3: 10 neurons (Softmax classifier)

Epoch	Training Loss	Training Acc.	Val. Loss	Val. Acc.
1	0.4778	0.8579	0.3939	0.8820
5	0.2463	0.9225	0.3386	0.9002
10	0.1501	0.9521	0.3833	0.9033
15	0.0961	0.9697	0.4864	0.9016
Test	0.3908	0.9211		

Four Layers

Results of training and test data after 15 epochs using:

a. Layer 1: 600 neurons (ReLu activation functions)
b. Layer 2: 400 neurons (ReLu activation functions)
c. Layer 3: 200 neurons (ReLu activation functions)
d. Layer 4: 10 neurons (Softmax classifier)

Epoch	Training Loss	Training Acc.	Val. Loss	Val. Acc.
1	0.4612	0.8599	0.3921	0.8813
5	0.2300	0.9265	0.3337	0.9025
10	0.1312	0.9573	0.4109	0.9050
15	0.0897	0.9714	0.4949	0.9090
Test	0.3847	0.9282		

Five Layers

Results of training and test data after 15 epochs using:

a. Layer 1: 800 neurons (ReLu activation functions)

b. Layer 2: 600 neurons (ReLu activation functions)

c. Layer 3: 400 neurons (ReLu activation functions)

d. Layer 4: 200 neurons (ReLu activation functions)

e. Layer 5: 10 neurons (Softmax classifier)

Epoch	Training Loss	Training Acc.	Val. Loss	Val. Acc.
1	0.4615	0.8593	0.3899	0.8803
5	0.2333	0.9242	0.3269	0.9033
10	0.1437	0.9531	0.3681	0.9085
15	0.0958	0.9690	0.4693	0.9078
Test	0.3836	0.9264		

Results Overview

Training

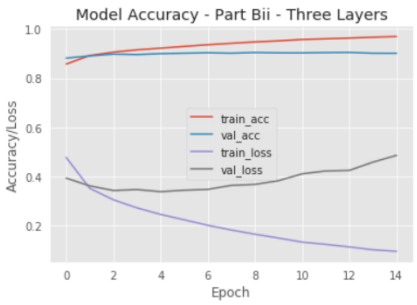
Epochs	Benchmark		Two Layers		Three Layers		Four Layers		Five Layers	
	Loss	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**
1	0.7464	0.8034	0.5313	0.8464	0.4778	0.8579	0.4612	0.8599	0.4615	0.8593
15	-	-	0.2151	0.9361	0.0961	0.9697	0.0897	0.9714	0.0958	0.9690

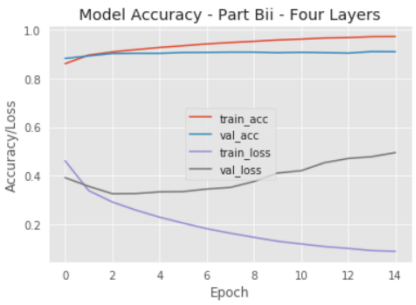
Tests

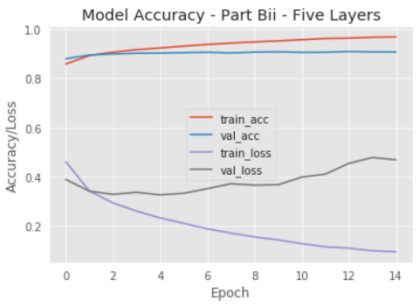
Epochs	Benchmark		Two Layers		Three Layers		Four Layers		Five Layers	
	Loss	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**
1	0.5769	0.8510	-	-	-	-	-	-	-	-
15	-	-	0.3251	0.9128	0.3908	0.9211	0.3847	0.9282	0.3836	0.9264

Validation

Epochs	Benchmark		Two Layers		Three Layers		Four Layers		Five Layers	
	Loss	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**
1	0.6603	0.8255	0.4478	0.8692	0.3939	0.8820	0.3921	0.8813	0.3899	0.8803
15	-	-	0.4087	0.8920	0.4864	0.9016	0.4949	0.9090	0.4693	0.9078







When comparing the results from the range of depths of neural networks from one layer with only a Softmax classifier to five layers with four ReLu activation layers and a single Softmax classifier, the best performing configuration was four layers with three ReLu activation layers and a single Softmax classifier. This configuration, with 600 neurons in the first layer, 400 neurons in the second, 200 in the third, and 10 neurons in the fourth, one for each class in the data going through the classifier, achieved an accuracy of 97.14% in the training data, 92.82% in the test data, and 90.90% in the validation data.

Across the range of depths, the training, test, and validation data increases in accuracy up to four layers in depth, and falls drops slightly in the five-layer neural network. Whilst the training data loss drops continually as the number of layers in the network increases, the loss in both the validation and test data only see decreases from the benchmark to two layers deep, from the third to fifth layers the loss rises as the number of layers increase. In each of these multi-layer configurations, the validation loss drops only from the first epoch to the third, then continues on a upward trend that increases as the number of layers in the network increases. There is a slight drop in the loss in the validation data from the four- and five-layer networks but following the trend of the loss in the network with five layers indicates that the loss will continue to increase if the number of epochs increase to a value greater than the 15 epochs used in these tests.

This increase in the gap between the training, test, and validation loss is indicative of overfitting in the network on the training data used to set the weight and bias values. This overfitting behaviour on the training data is increases as the number of epochs and layers increase which is to be expected, as in both the amount of times the training data is pushed through the network increases.

iii. Dropout, L1, and L2 Regularization

Results of training and test data after 15 epochs using two deepest networks from part ii:

a. Layer 1: 600 neurons (ReLu activation functions)
b. Layer 2: 400 neurons (ReLu activation functions)
c. Layer 3: 200 neurons (ReLu activation functions)
d. Layer 4: 10 neurons (Softmax classifier)

And…

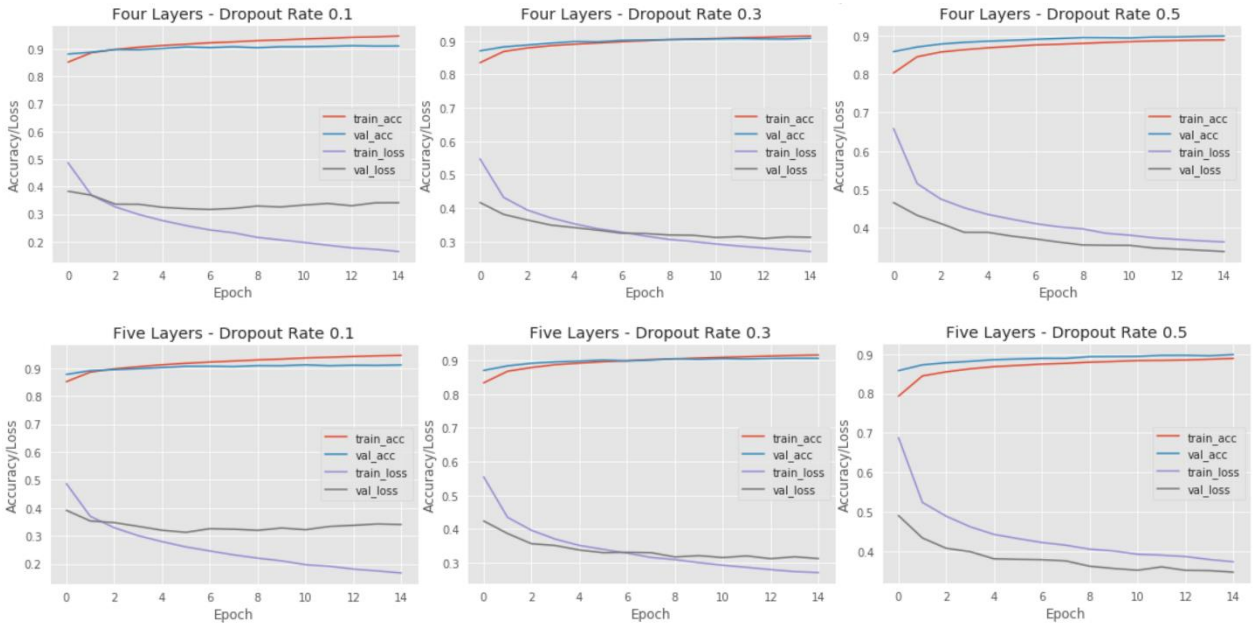
a. Layer 1: 800 neurons (ReLu activation functions)
b. Layer 2: 600 neurons (ReLu activation functions)
c. Layer 3: 400 neurons (ReLu activation functions)
d. Layer 4: 200 neurons (ReLu activation functions)
e. Layer 5: 10 neurons (Softmax classifier)

Dropout

Results are representative of process after 15 epochs are finished.

Dropout Rate	Four Layers - Loss			Five Layers - Loss		
	Training	Test	Validation	Training	Test	Validation
0.1	0.1644	0.2669	0.3415	0.1669	0.2718	0.3404
0.2	0.2202	0.2415	0.3158	0.2218	0.2414	0.3045
0.3	0.2700	0.2439	0.3128	0.2714	0.2437	0.3128
0.4	0.3144	0.2569	0.3245	0.3200	0.2585	0.3218
0.5	0.3642	0.2722	0.3393	0.3734	0.2767	0.3472

Dropout Rate	Four Layers - Accuracy			Five Layers - Accuracy		
	Training	Test	Validation	Training	Test	Validation
0.1	0.9465	0.9311	0.9104	0.9457	0.9299	0.9115
0.2	0.9291	0.9312	0.9098	0.9294	0.9299	0.9099
0.3	0.9147	0.9262	0.9083	0.9156	0.9277	0.9058
0.4	0.9016	0.9239	0.9018	0.9025	0.9226	0.9042
0.5	0.8887	0.9192	0.8986	0.8889	0.9188	0.8986



Across both four- and five-layer networks a decrease can be seen in the validation loss and gap between the training and validation loss when dropout regularization is applied to the data in the training process. As the rate of dropout increases in increments of 0.1 from 0.1 to 0.5, the gap between the training and validation loss continues to decrease until it is lower than the loss in the training data across all epochs.

The decrease in the difference in loss between the two cannot be attributed entirely to a drop the validation loss value as the dropout rate increases. The loss value in the training data increases as the dropout rate increases to the point it is greater than the validation loss, but this is expected behaviour as the dropout operation is applied to the training data to remove overfitting on the training data.

The table below shows the difference between the best performing neural network from part ii, and the best dropout rate/layer count combination for both loss and accuracy.

	Loss – Best 0.2 with 5 layers			**Accuracy – Best 0.1 with 5 layers**		
	No Dropout	**Dropout**	**Change**	**No Dropout**	**Dropout**	**Change**
Training	0.2151	0.2218	+0.0067	0.9714	0.9457	-0.0257
Test	0.3251	0.2414	-0.0837	0.9282	0.9299	+0.0017
Validation	0.4087	0.3045	-0.1042	0.9090	0.9115	+0.0025

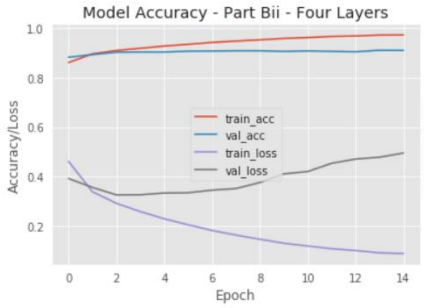
The improvement in the loss for the validation and test data does not provide the same level of improvement in the accuracy. The best performing dropout rate in terms of loss value does not translate into an improvement in accuracy, in fact most dropout rates seen a continuous but small drop in accuracy as the dropout rate increases. However, what is most interesting here is that as the rate of dropout increases it brings the training data accuracy and loss to more similar values seen by that of both the training and validation data, a clear indicator that dropout regularization is having the desired effect on the training data.

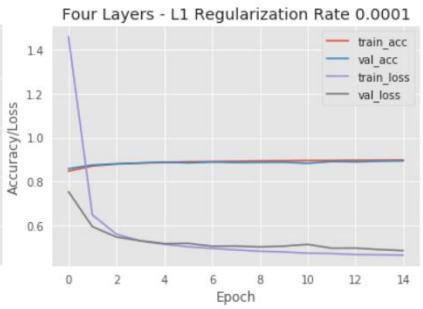
L1 Regularization

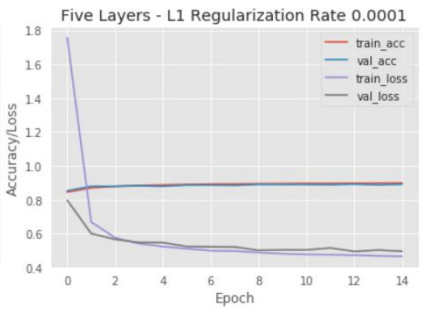
Results are representative of process after 15 epochs are finished.

Iteration	**Four Layers – L1 Regularization Loss**			**Five Layers – L1 Regularization Loss**		
	Training	**Test**	**Validation**	**Training**	**Test**	**Validation**
1	1.4614	-	0.7531	1.7562	-	0.7968
5	0.5122	-	0.5151	0.5233	-	0.5466
10	0.4772	-	0.5036	0.4810	-	0.5036
15	0.4625	0.4201	0.4831	0.4654	0.4307	0.4956

Iteration	**Four Layers – L1 Regularization Accuracy**			**Five Layers – L1 Regularization Accuracy**		
	Training	**Test**	**Validation**	**Training**	**Test**	**Validation**
1	0.8465	-	0.8575	0.8449	-	0.8526
5	0.8864	-	0.8871	0.8868	-	0.8791
10	0.8938	-	0.8870	0.8947	-	0.8895
15	0.8971	0.9115	0.8930	0.8982	0.9084	0.8911



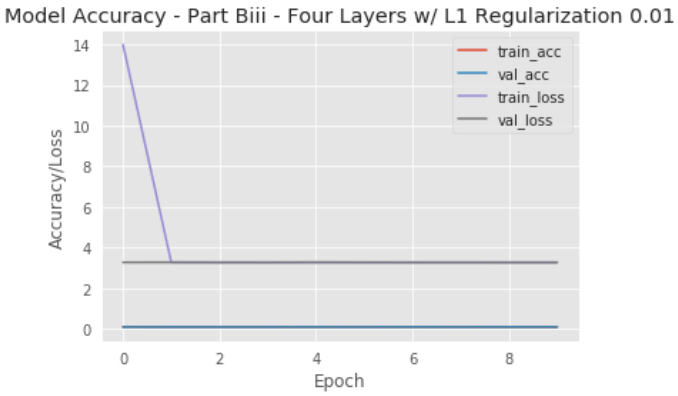




Applying L1 regularization to the training data exhibits a dramatic impact on both the four- and five-layer neural networks over the benchmark used, the best performing network from part ii. The loss values start considerably higher that those found in the benchmark, but these quickly converge around the third epoch and the loss continues to decrease at a very low rate.

The difference in loss seen around the third epoch is almost non-existent, but this cannot be attributed to an increase the performance of the model with respect to the validation loss value, but instead the training loss does not continue to decrease dramatically as the epochs progress. This is a clear indication that the L1 regularization technique is having the desired impact on overfitting on the training data, thanks to the adjustment of the weight and bias values using the 0.0001 L1 regularization rate provided.

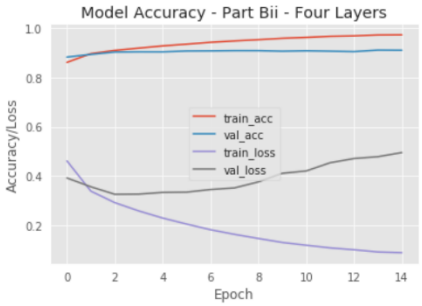
Larger lambda rates for L1 regularization were tested but these rendered the network unusable, this is demonstrated in the table to the left. After the first epoch the accuracy and loss values for both the training and validation data 'flatline', giving steady loss values of just under 4.0 and accuracy just over 0.0.

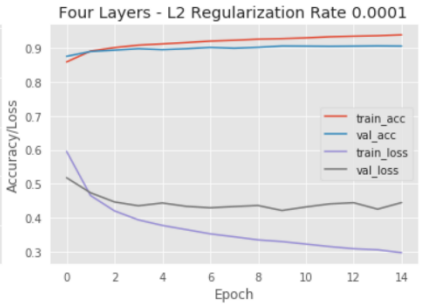


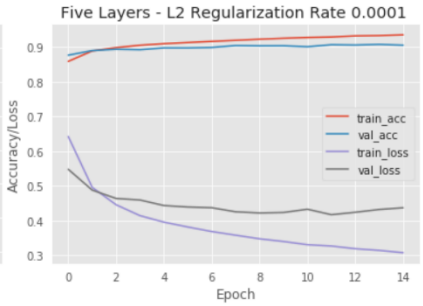
L2 Regularization

Iteration	Four Layers – L2 Regularization Loss			Five Layers – L2 Regularization Loss		
	Training	Test	Validation	Training	Test	Validation
1	0.5953	-	0.5176	0.6421	-	0.5476
5	0.3771	-	0.4433	0.3951	-	0.4430
10	0.3295	-	0.4211	0.3394	-	0.4230
15	0.2967	0.3730	0.4444	0.3071	0.3666	0.4366

Iteration	Four Layers – L2 Regularization Accuracy			Five Layers – L2 Regularization Accuracy		
	Training	Test	Validation	Training	Test	Validation
1	0.8591	-	0.8757	0.8586	-	0.8765
5	0.9123	-	0.8953	0.9097	-	0.8976
10	0.9278	-	0.9060	0.9252	-	0.9036
15	0.9390	0.9254	0.9058	0.9352	0.9242	0.9053

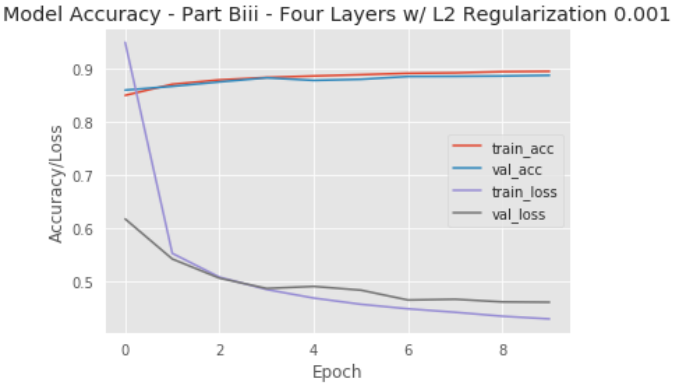






L2 regularization has a more subtle effect on the loss values achieved by the training data than that of L1 regularization. L2 exhibits a decrease in severity of the training loss downward trend whilst decreasing the validation loss over time, decreasing the gap between the two over each epoch. This is in stark contrast from the loss values achieved in the four-layer network from part ii, where the training loss decreases continuously as epochs progress, and the validation loss continues to increase after the third epoch, widening the gap between the two.

Slight increases improve the difference in the gap between the training and validation loss values, however it is very sensitive to changes in the L2 regularization rate. The table to the right shows the impact of changing the rate from 0.0001 to 0.001, where the initial loss values are higher but quickly converge after the second epoch and continue on a downward trend over each epoch.



Accuracy is not impacted at the same rate as loss, with accuracy rates on the test and validation data remaining at similar values to before L2 regularization is applied. The training accuracy is impacted however to reduce it from the high values seen when overfitting was a factor, with L2 regularization rate increases, the accuracy value of the training value decreases to bring it closer to accuracy values achieved by the test and validation data.

Part C – Research – Batch Normalisation

Two of the techniques used when pre-processing data for use in neural-networks are normalisation and standardisation, whereby all data-points are put on the same scale, usually 0-1, to prepare the data for the training process. This intention of the normalisation process is to reduce the impact of very high or very low data feature values, to eliminate the impact of gross influences within the data and training process. Very large or small values within the training data can cause imbalanced gradients, known as exploding gradients which cause activation gradients in successive layers to either reduce or increase in magnitude [1], and makes it harder to train the neural network. Normalising the data before it is input into the network prevents this issue with large or small values and also has the added advantage of increasing the speed of the training process, however normalising data is not without its own problems.

During training, even with normalised data, it is very likely that one of the weights will become larger than all the other weights. This large weight will then cause the output from its corresponding neuron to be extremely large and this imbalance will continue to propagate through the neural network causing instability. This issue becomes more pronounced as a neural network deepens [2]. With deep networks, the gradient will update each parameter under the assumption that the other layers do not change. In reality, all layers are updated at the same time and when the update is made it is possible that unexpected results can be returned because *"many functions composed together are changed simultaneously, using updates that were computed under the assumption that the other functions remain constant"* [3], put simply if the value X changes from its expected value, then the algorithm may be to be retrained. This is also known as co-variate shift, coined by Ioffe and Szegedy [4], where the distribution of covariates used as predictors changes between the training process and test/validation stage. Batch normalisation was created to address this problem.

Batch normalisation was proposed by Ioffe and Szegedy in their 2015 paper [4] providing a way of reparametrizing deep networks. The proposed solution for the co-variate shift problem is to reduce the amount of change seen in the distribution of the activation values of neurons in a layer by introducing the normalisation process to these hidden values on a per layer basis. Batch normalisation is applied to the layers within a network at the choosing of the person building the network. This ability to choose the networks to apply this normalisation technique to can results in normalised data flowing in and out of all layers in the network. The batch normalisation process consists of three steps:

1. Normalise the output from the activation function (X = output, M = mean, S = standard deviation)

$$Z = \frac{(X - M)}{S}$$

2. After normalising, multiply the output by the arbitrary parameter G (gamma).

$$Z\ x\ G$$

3. After multiplying the output by G, add arbitrary parameter B (beta) to resulting product.

$$(Z\ x\ G) + B$$

These new values introduced during the batch normalisation process, mean, standard deviation, gamma, and beta, are all trainable, meaning they can and will become optimised during the training process. As a result, the weights in the network don't become imbalanced with very large or small values since the normalisation process is included in the gradient process. There is no ideal starting value for beta and gamma, the values for the associated weights for both can be initialised in the batch normalisation layer parameters. This reduction in the imbalance in the weights in earlier layers makes the weights in the later layers more robust to changes in later layers by reducing the amount that the distribution of activation values shifts around between layers, the co-variate shift problem mentioned earlier.

Batch normalisation limits the amount to which updating the weights in the earlier layers can affect the distribution of activation values that later layers see and therefore has to learn on. Batch normalisation reduces the problem of the input values changing, causing the activation values in later layers to become more stable and although the input distribution changes some, it changes less than before. As earlier layers keep learning, the amount that this forces the later layers to adapt to earlier layer changes is reduced and allows each layer of the network to learn by itself more independently of other layers.

The process also has the added benefit of greatly increasing the speed at which training occurs as a result of normalised values and removal of outliers in the hidden layer values. Another advantage of batch normalisation is that it can also act as a regulariser [5]. Each mini-batch is scaled by the mean/variance computed on that mini-batch, which has the effect of adding some noise to the values within that mini-batch, so similar to the dropout regularisation technique it adds some noise to the update process [5].

References

[1] Aggarwal, C.C., 2018. *Neural networks and deep learning* (pp. 978-3319944623). Berlin, Germany:: Springer. 3.6 Batch Normalisation, P152

[2] Chollet, F., 2018. *Deep Learning with Python*: . MITP-Verlags GmbH & Co. KG., 7.2.1 Batch normalization P260

[3] Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. MIT press., 8.7.1 Batch Normalization, P309

[4] Ioffe, S. and Szegedy, C., 2015. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. arXiv preprint arXiv:1502.03167.

[5] Aggarwal, C.C., 2018. *Neural networks and deep learning* (pp. 978-3319944623). Berlin, Germany:: Springer, 3.6 Batch Normalisation, P156