**Deep Learning - 236606 - HW2**

Ido Glanz - 302568936 Matan Weksler – 302955372

1. **DNN architecture:**

See mydnn.py for the DNN class as well as all relevant routines to run the class. For each experiment see relevant .py file which uses the above class.

1. **Experiments**
   1. **Batch size:**

In this set of experiments, we used a network consisting of a single hidden layer with 128 neurons and using ReLU and softmax activations in aim to reveal the relationship between the batch size (i.e. how many samples are trained on before each backpropagation step) to the learning performance.

We ran the network with batch sizes 128, 1024, 10K and 20K and for each one accounted for the total runtime and overall accuracy and loss on the test set.

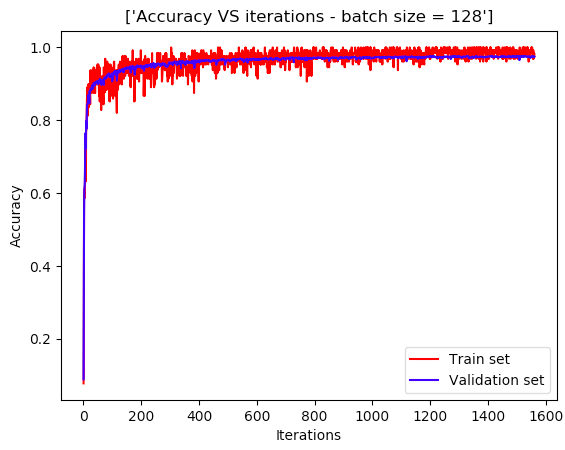
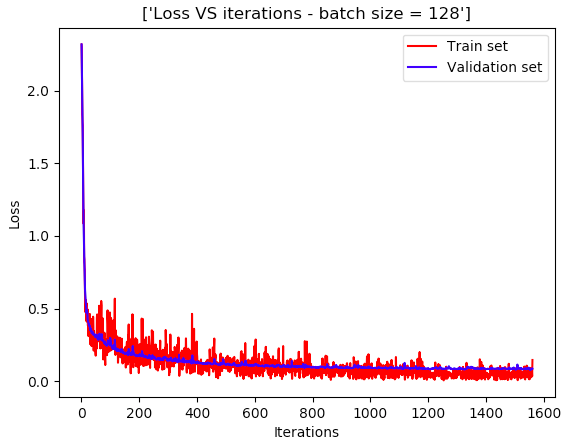


Figure - Loss and accuracy VS iterations for 128 batch size

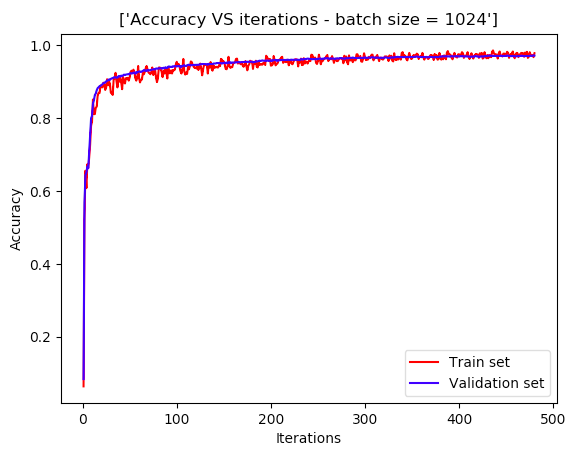
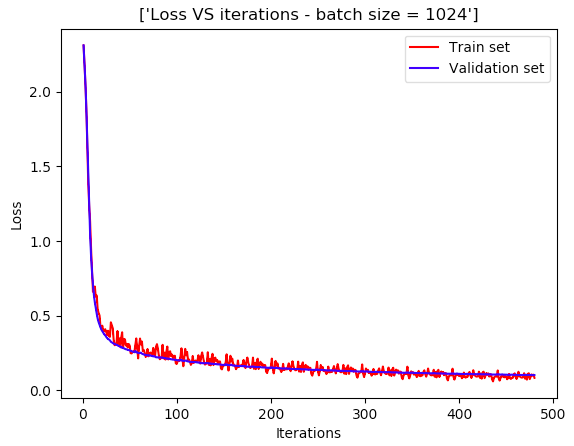


Figure - Loss and accuracyt for the 10K batch size

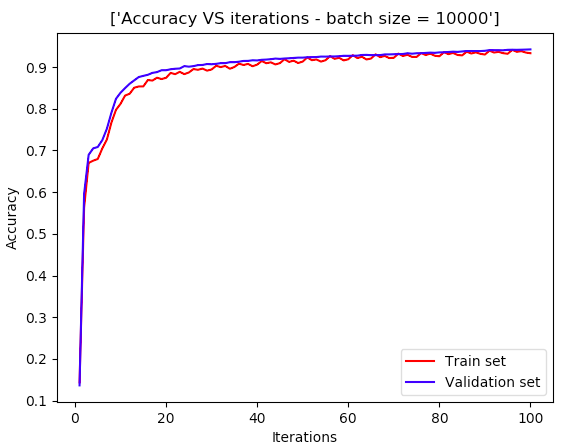
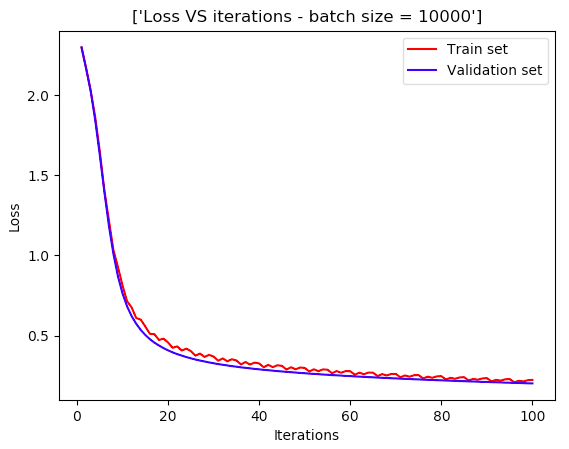


Figure - Loss and accuracy for 10K samples batch

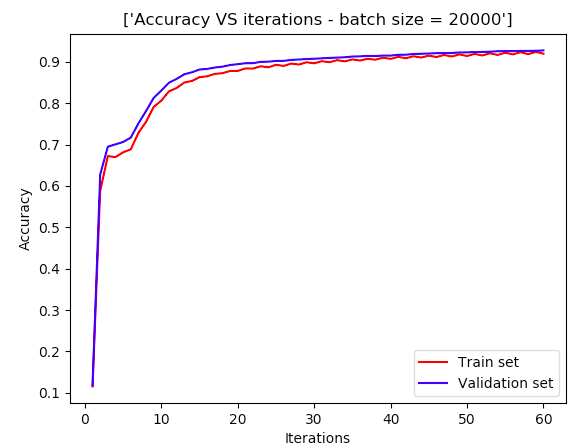
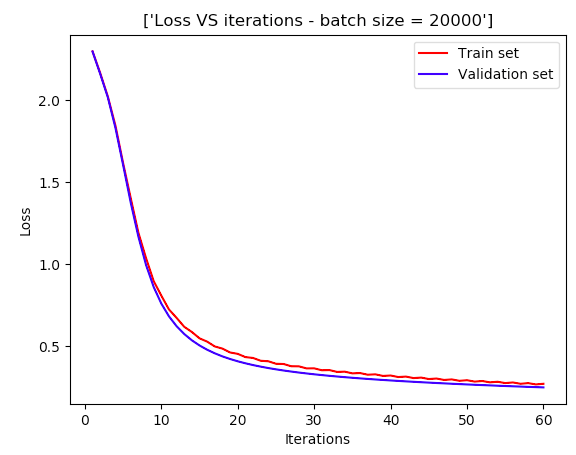


Figure - Loss and accuracy to 20K batch size

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Batch size | epochs | Total iterations | Run-time | Loss train set | Accuracy train set | Loss test set | Accuracy test set |
| 128 | 4 | 1560 | 1280 sec | 0.0840874 | 0.9754 | 0.0831195 | 0.9747 |
| 1024 | 10 | 480 | 400 sec | 0.0829664 | 0.978515 | 0.1041465 | 0.9691 |
| 10K | 30 | 150 | 270 sec | 0.2006139 | 0.9425 | 0.1775361 | 0.9482 |
| 20K | 30 | 60 | 270 sec | 0.2700259 | 0.9195 | 0.2555614 | 0.9261 |

* 1. **Regularization:**

In these tests, we used the same architecture as in section 2.a only now we are looking for the regularization effects on generalization when using different norms (L1, L2 and without). That said, we also ran a brief optimization on the weight decay parameter for each norm and used it when comparing the results.

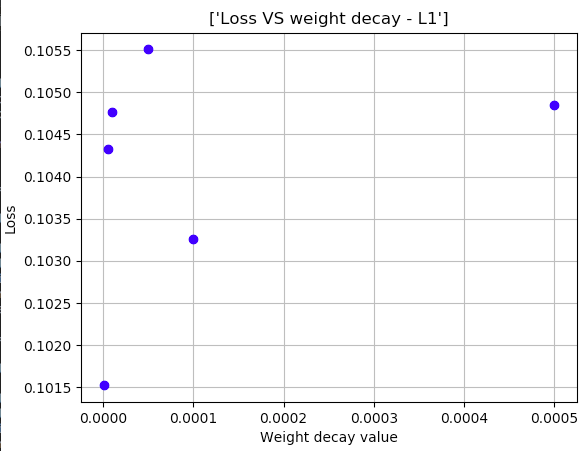
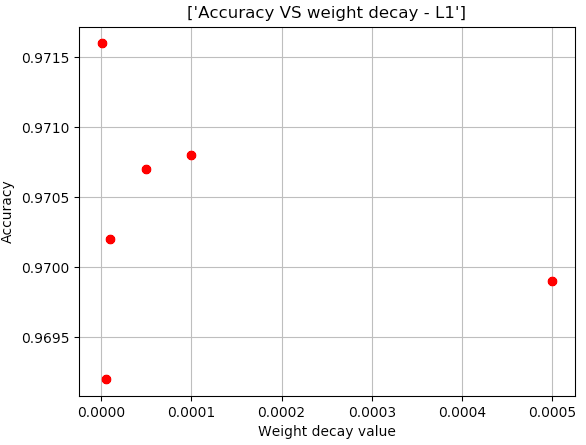


Figure - Loss and accuract VS weight decay under L1 norm

The weight decay with the best accuracy is 5e-07, giving a loss of 0.101523 and accuracy of 0.9716 (on the test set)

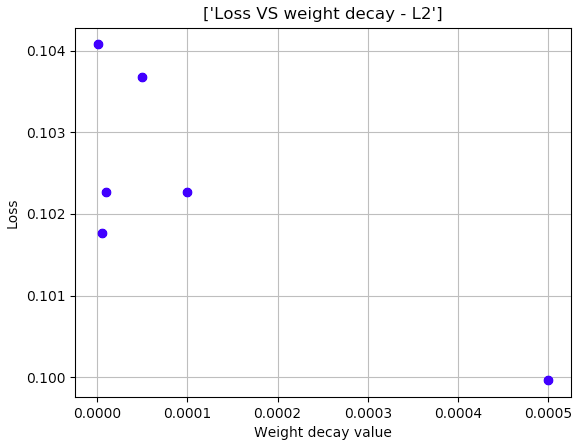
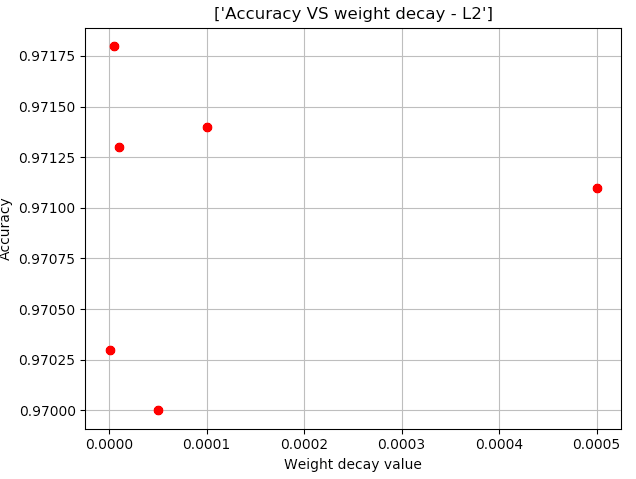


Figure - Loss and accuracy VS weight decay under L2 norm

The weight decay with the max accuracy is 5e-06 giving a loss of 0.10176 and accuracy of 0. 0.9718 (again on the test set)

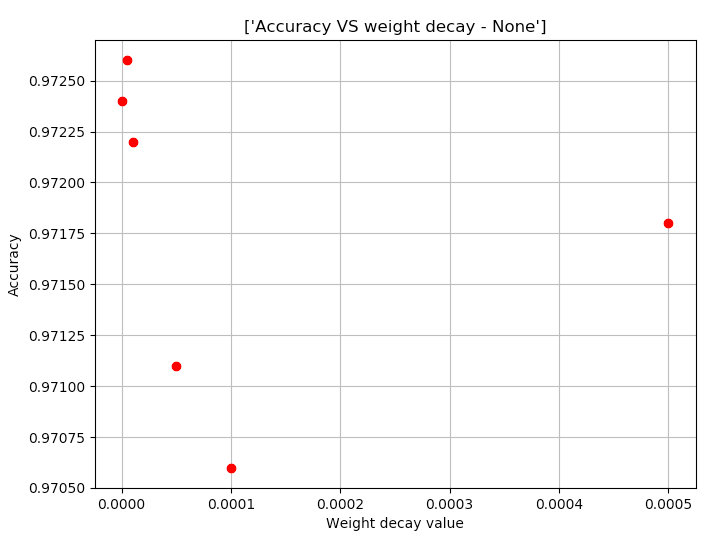
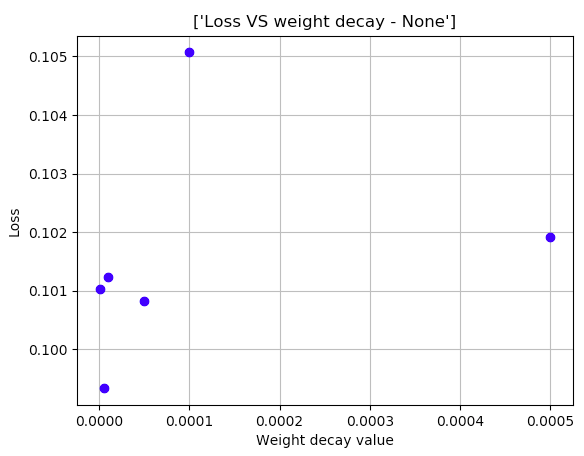


Figure - Loss and accuracy VS weight decay without regulaization

The weight decay with the max accuracy = 0.0001, where the loss is 0.1016795774770573, and accuracy = 0.9717

After optimizing the weight decay parameter, we now train the net for each regularization norm in accordance (**changing only the norm type**):

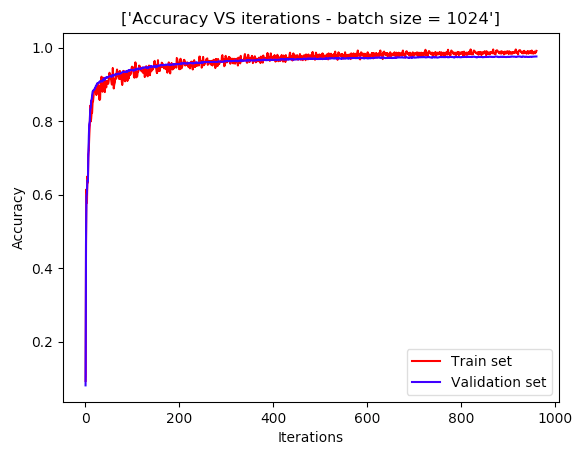
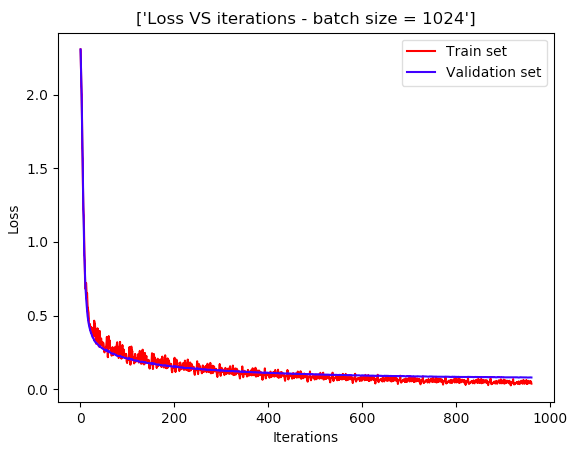


Figure - Loss and accuracy plots for batch size 1024 with L1 regularization

And zooming in we can see the generalization error is starting to grow as we run more iterations:

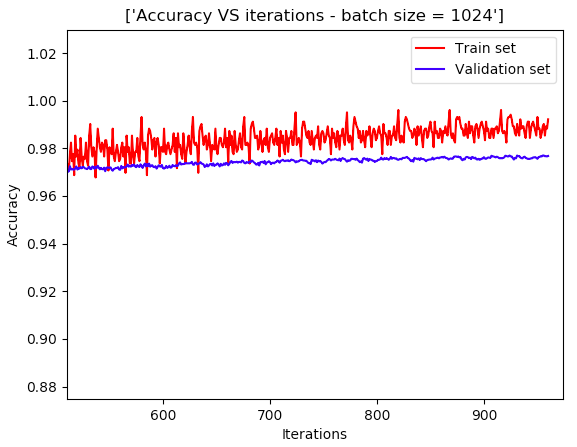
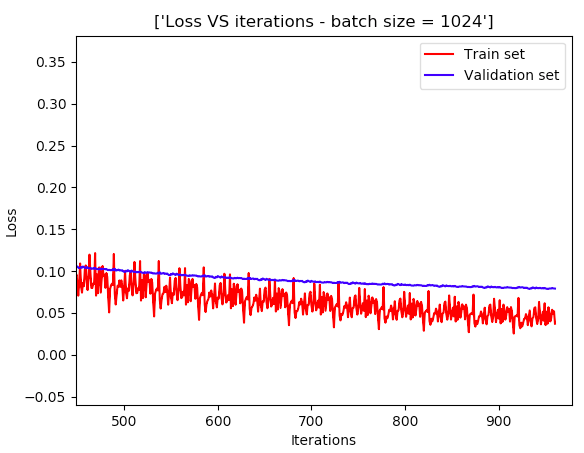


Figure - Zooming in on figure 8 we can see the generalization error starting to develop (while the train set error is decreasing the validation set isn’t effected – hence we are starting to overfit)

And running it over the test set we obtain: Loss = 0.08187, accuracy= 0.9749

For L2 regularization

Looking at the zoom in of the accuracy and loss curve we can see the generalization error development:

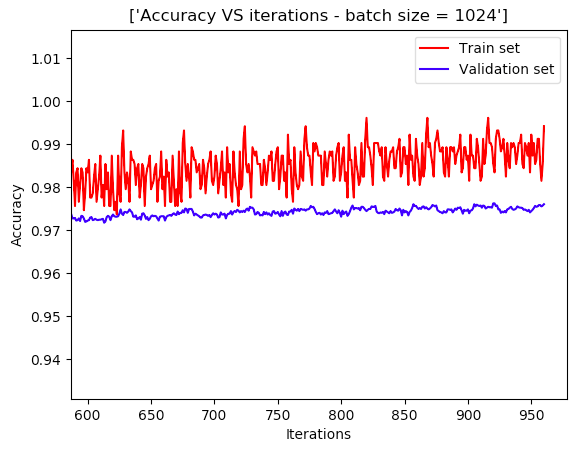
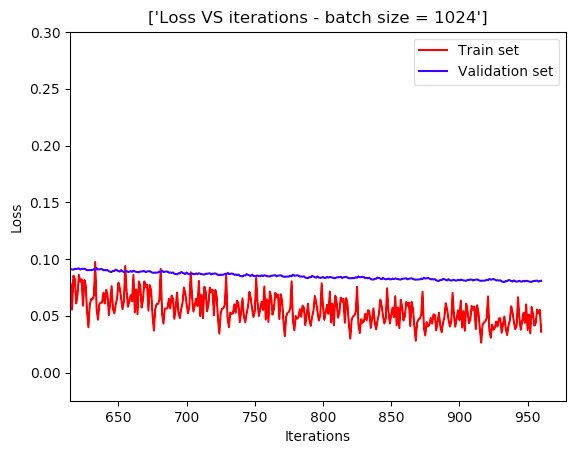


Figure - Accuracy and loss curves for L2 regularization

And running it over the test set we obtain: Loss =0.081762, accuracy= 0.9737 – slightly better than the L1 regularization

Without regularization

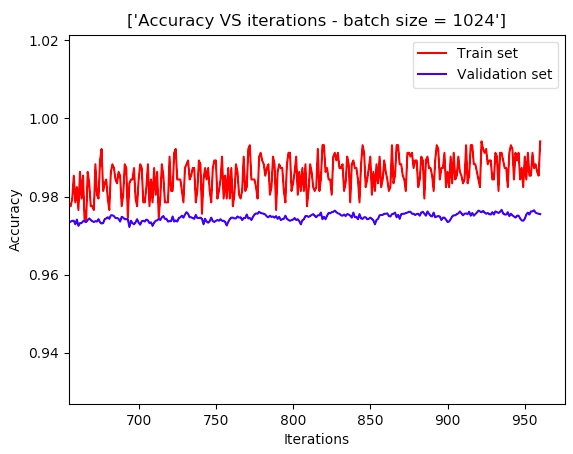
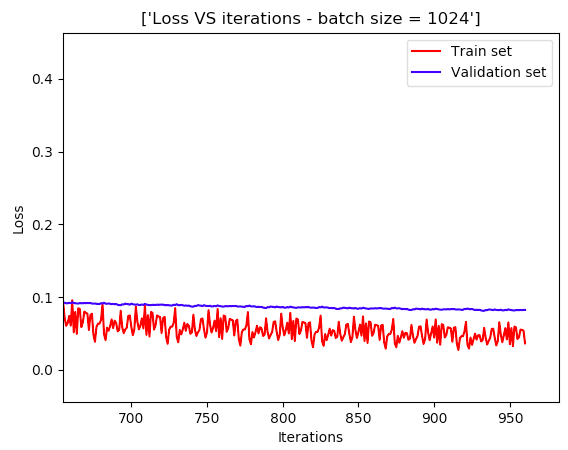


Figure - Loss and accuracy without using regularization

And running it over the test set we obtain: Loss = 0.08237, accuracy= 0.9741

Hence the best accuracy on the test set was using L2 regularization – that said, we also could have tried running for more training cycles and in theory the generalization error would have decreased more when not using generalization.

* 1. **Architecture**:

In the following sets of experiments, we ran different configurations of architectures ranging from depth 2-3 and width up to 512 neurons aiming to investigate the effects on the accuracy of the test and train sets. We chose different configurations as to test both extremes of the overall weights as well as various configurations which add up to the same overall number of weights (give or take) only with different architectures (i.e. varying each layer opposite to the other).

* + 1. **Configurations table and results:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | Depth | Width | Overall weights | Runtime | Loss train set | Accuracy train set | Loss test set | Accuracy test set |
| 1 | 2 | 64 | 50,816 | 22.428 | 0.048452 | 0.9892 | 0.0856 | 0.9729 |
| 2 | 2 | 128 | 101,632 | 36.619 | 0.03774 | 0.99218 | 0.07867 | 0.9766 |
| 3 | 2 | 256 | 203,264 | 68.513 | 0.032775 | 0.991210 | 0.07744 | 0.9752 |
| 4 | 2 | 512 | 406,528 | 126.4119 | 0.03132 | 0.99414 | 0.071216 | 0.9782 |
| 5 | 3 | [64, 64] | 54,912 | 27.664 | 0.06569 | 0.98535 | 0.097230 | 0.9697 |
| 6 | 3 | [512, 512] | 1,342,578,688 | 229.146 | 0.0708698 | 0.981445 | 0.099679 | 0.9706 |
| 7 | 3 | [128, 64] | 109,184 | 43.134 | 0.053017 | 0.98730 | 0.092121 | 0.9707 |
| 8 | 3 | [80, 512] | 108,800 | 60.797 | 0.09717 | 0.96679 | 0.1155 | 0.9637 |
| 9 | 3 | 100, 200 | 100,400 | 48.432 | 0.051773 | 0.9882 | 0.09499 | 0.9702 |
| 10 | 3 | 512,80 | 443,168 | 136.785 | 0.045381 | 0.9892 | 0.083578 | 0.9749 |

* + 1. **Conclusions**
       1. There seems to be a minimal number of weights needed to obtain better performance, which is logical under the input dimension (784) while using too much can also decrease accuracy, possibly because the more weights you use the more learning you need (as so they all converge and not to any local minimas)
       2. As for generalization (or over fitting), it seems using one layer we obtain greater generalization errors – possibly due to the regularization effects or its lack of expressiveness.
       3. Balancing run-time and performance, a single hidden layer, with 256 weights seemed to perform well while also consume less time and in that sense, might be a good option to keep optimizing other parameters on.
  1. **Regression**

In this section, we generated a synthetic data set from the function comprising of 100 and 1000 samples (each containing a uniformly sampled x1 and x2 from the above interval). The data was then used to train a network as so optimize the loss for the two sample set sizes.

We tested multiple architectures for both cases, where the general guideline was to test regularization for the smaller data set (as to allow generalization) and try and use less weights (partially to allow quicker runtime and more trials)

We first tested different weight decays using the architecture:

*{"input": 2, "output": 128, "nonlinear": "relu", "regularization": "l2"},*

*{"input": 128, "output": 64, "nonlinear": "sigmoid", "regularization": "l2"},*

*{"input": 64, "output": 1, "nonlinear": "none", "regularization": "l2"}*

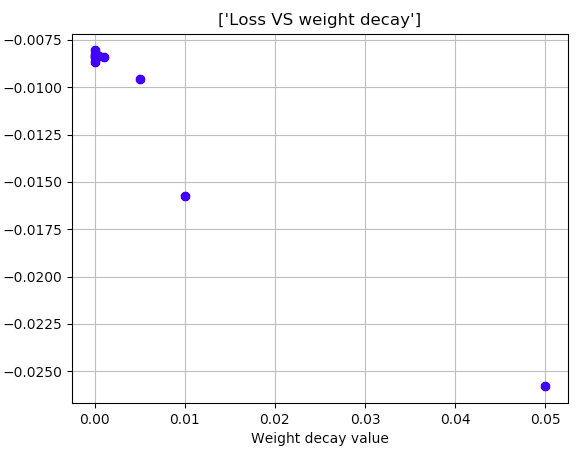


Figure -Test set loss VS Weight decay for 100 samples. Seeing the above results, we chose to work weight minimal weight decay factor

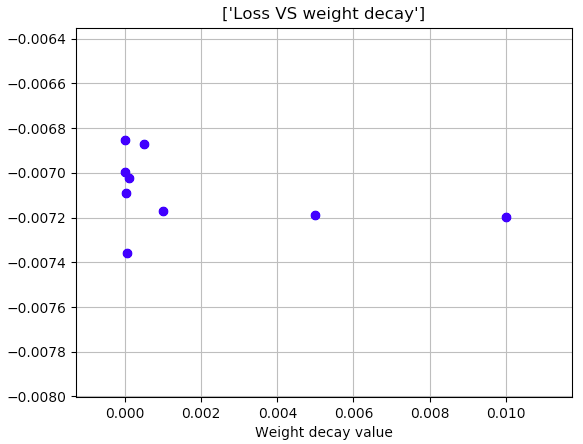


Figure - Test set loss VS weight decay for 1000 training samples. Here also we chose to work with minimal weight decay to obtain minimal loss

We then ran the network for a few architectures and recorded the test set loss;

For depth 2 we used ReLU activation and L2 regularization, and for depth 3 we used ReLU and sigmoid (in this order) and L2 regularization as well.

For both train sample sizes we ran for 200 epochs with batch size 10 for 100 train samples and 100 for 1000 train samples.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Depth | Width | Accuracy train set | Accuracy test set | Accuracy train set | Accuracy test set |
| 100 train samples | | 1000 train samples | |
| 1 | 2 | 64 | -0.00359728 | -0.01040215 | -0.00806548 | -0.00835385 |
| 2 | 2 | 128 | -0.01758234 | -0.01108232 | -0.00676666 | -0.00834461 |
| 3 | 2 | 256 | -0.01116733 | -0.01198914 | -0.00708678 | -0.00870745 |
| 4 | 2 | 400 | -0.00277952 | -0.00914418 | -0.0049995 | -0.00661973 |
| 5 | 3 | [64, 64] | -0.00419778 | -0.00678353 | -0.00682882 | -0.00707502 |
| 6 | 3 | [128, 64] | -0.00516225 | -0.00743214 | -0.00367851 | -0.00429218 |
| 7 | 3 | [64, 128] | -0.00732223 | -0.00769308 | -0.00682058 | -0.00662957 |
| 8 | 3 | 100, 200 | -0.01928708 | -0.03556572 | -1.59984514 | -1.39835986 |

Following the rough optimization on architecture, we settled for the architecture marked orange for the 100 train-sample-set and the one marked yellow for the 1000 train samples. Generally speaking, the more weights we induce the more samples are needed to properly train it, yet too few cause a generalization error as seen in the shallow architectures above.

Below are the training and validation loss curves for the chosen schemes:

For the 100 sample train set:

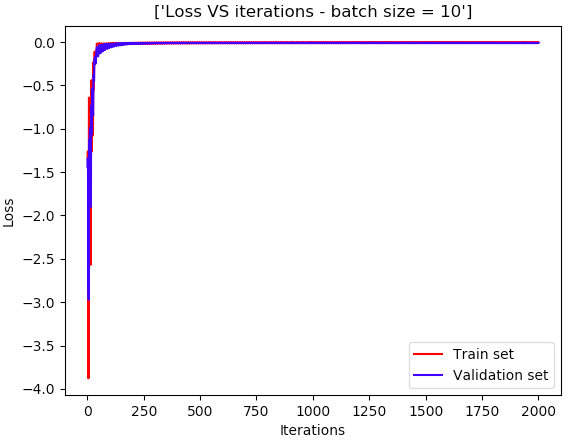


Figure - Loss VS iterations for test and validation sets

And for the 1000 sample train set:

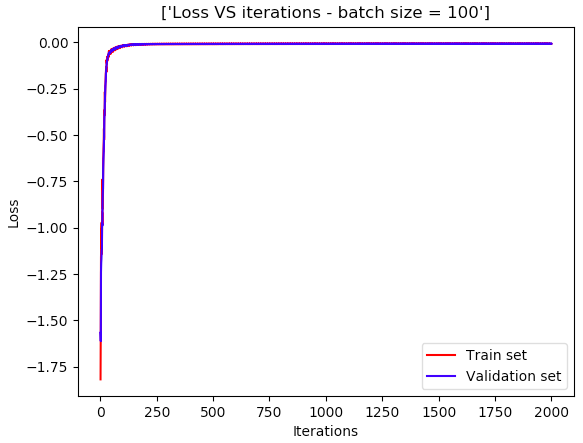
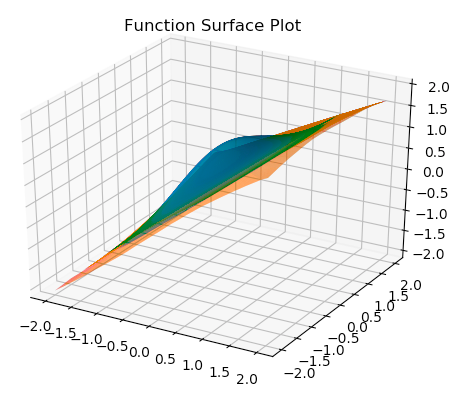


Figure - Loss VS iterations for the 1000 sample train set

To conclude, we plot the function over the [-2, 2] grid with the true and estimated values (test set and network output):

For the 1000 sample set:



And for the 100 sample set:

