Mini-Project Report

Haixiang Xu hxu@caltech.edu

Nov. 17, 2016

Abstract

This project is intended to explore the relative impact of regularization on neural network models, with detailed comparison between regularization coefficient of L1 and L2 norms. At the same time, the ability of L1 and L2 norms in creating sparse weight matrix will be investigated and we will see that L1 norm indeed have more generalization power in terms of creating sparse weights. The dropout layer is also introduced and compared with the result of L2 regularization.

Moreover, we also investigate

- the effects of different network architectures (deep neural network v.s. shallow neural network),
- the effects of different non-linear activation functions, mainly relu v.s. sigmoid v.s. tanh.

The visualization of network is appended in the end as part of the conclusion.

Some key factors in building the model of this project are listed as follows: the DNN model is trained with MLP method, and using gradient-based stochastic optimizer ('adam' optimizer). The 60000 training data is split into 1/5 for validation and the rest for training, and shuffles for each epoch; the test data involves 10000 data samples; the learning rate is adjusted to a smaller value when the validation loss plateaus; and we use training error to determine when to stop the training process early; the maximum training epoch is set to be 200, yet no model reaches such complexity. I will use Keras as frontend and Theano as backend. The dataset I use is MNIST. Code and documentation are included in this python notebook.

Regularization of DNN

1. Benchmark case

First of all, we will look at a DNN model with 6 hidden layers [128, 128, 64, 64, 64, 64]. Remember input layer is of length 784 and output layer is of length 10, and we will ignore regularization to use this as a benchmark case. The activation function is relu.

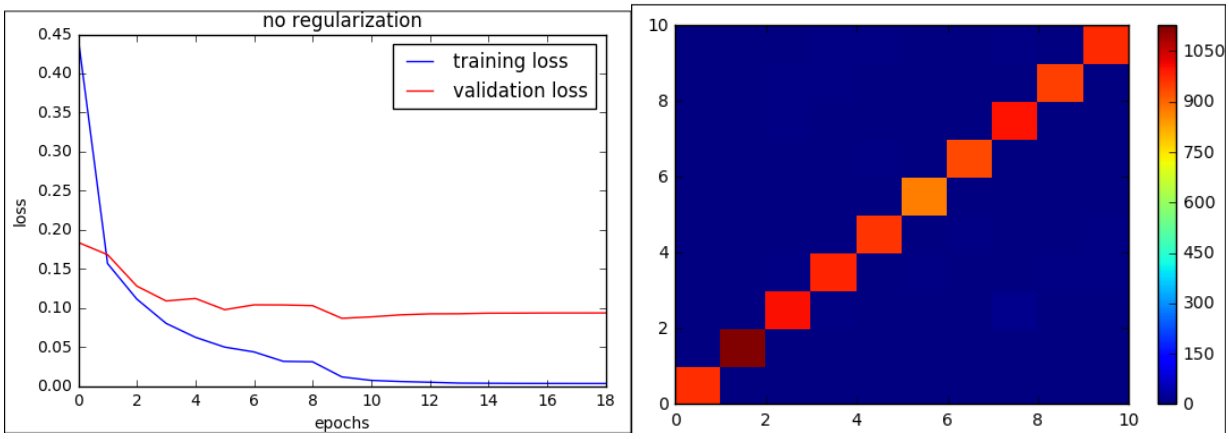


Fig. 1 Bench mark, (a), training and validation loss; (b), testing accuracy confusion heat map

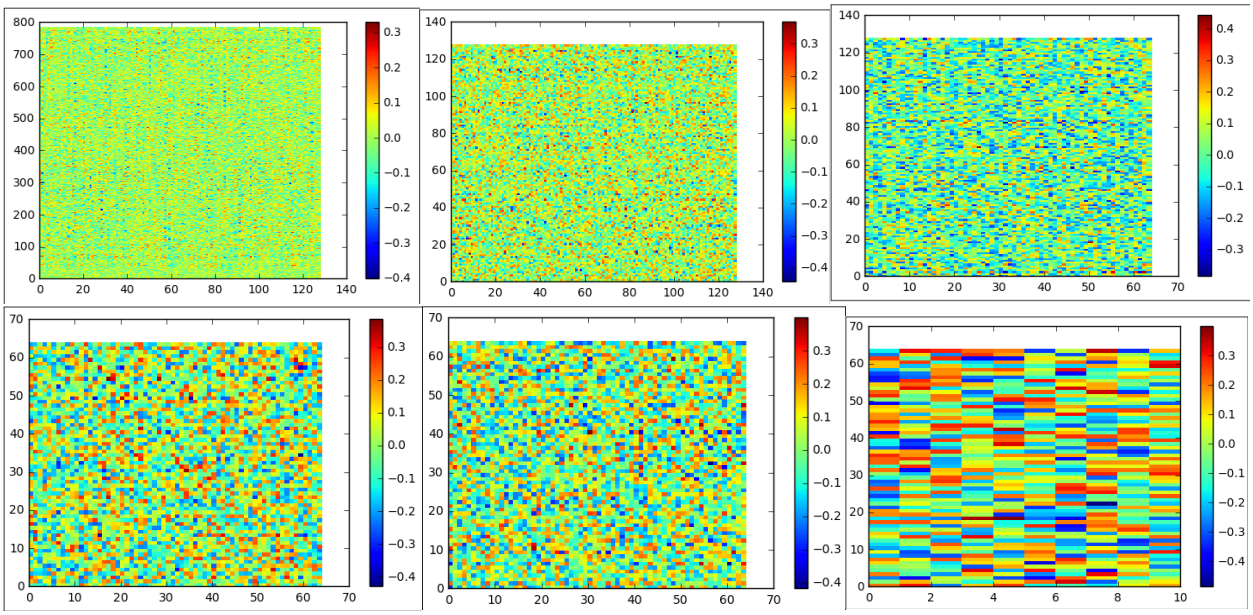


Fig. 2, Bench mark, hidden layers weight heat map

From the benchmark case, we can conclude that:

- The deep neural network already reached a very high test accuracy without regularization, it's due to the MNIST set is highly separable in high-dimensional non-linear space, and the DNN model is perfect in doing the job.

min training error	min validation error	Test error	Test accuracy
0.0031	0.0865	0.02120	0.9788

- During the training process, there is overfitting happened in that the validation loss plateaus or even increases when training loss still going down. However, because of previous reason, it does not result in very bad models.
- We can see from the weight heat map that all layers are randomized, while most hidden layers, have polarized distribution of weights in some neurons. Only very small amount of the weights in all layers are close to 0, as we show in sparsity (we pick criteria of < 0.0001 as close to 0).

	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	0.0011	0.0006	0.0006	0.0007	0	0

2. L2-normalization

2.1 L2 C = 1.0

It's too heavy on regularization, as the weights are mostly 0, model shifts too much towards original point in high-dimensional space.

Notice that all weights are sparse in the hidden layers are 0, except for output layer, and at the output layer, all neurons predicts the next weight to be 1, that's why we have almost uniform incorrect predictions.

min training error		min validation error		Test error		Test accuracy
2.3010		2.3020		0.8865		0.1135
	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	1.0	1.0	1.0	1.0	1.0	1.0

2.2 L2 C = 0.1

In this case, it is very similar to c = 1.0, we conclude that because the regularization term is still too large for this DNN, we are unable to train a useful model. Plots are similar, weights are extremely sparse, and error rates are similarly high. Please refer to the notebook for details.

min training error		min validation error		Test error		Test accuracy
2.3010		2.3020		0.8865		0.1135
	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	1.0	1.0	1.0	1.0	1.0	0.88906

2.3 L2 C = 0.01

In this case, regularization is not too big for the model weights, and it takes more epoch to train because weights are now converging to useful local minimal.

Model training is finished within 37 epochs, and the performance is much better than over-regulated cases. (Note that because training sample number is 4 times validation set, the error should be normalized in order to reflect correct number)

min training error		min validation error		Test error		Test accuracy
0.7913		0.2074		0.0523		0.9477
	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	0.39378	0.33941	0.38464	0.42017	0.35888	0.125

We can see very strange yet interesting distribution of weights below, especially in the input layer, that the weights in different layers are distributed into ‘grids’, which probably indicates:

- Weights in MNIST data are originally sparse in high-dimensional space,
- Because MNST is 2D data set, while MLP algorithm takes them as 1D vector, the weights are naturally repetitive in 2D patterns.

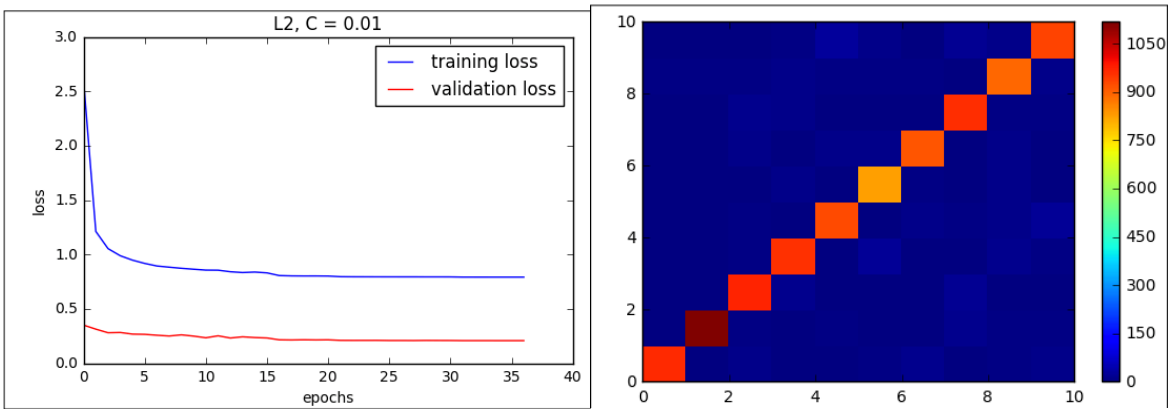


Fig. 3 c = 0.01, (a), training and validation loss; (b), testing accuracy confusion heat map

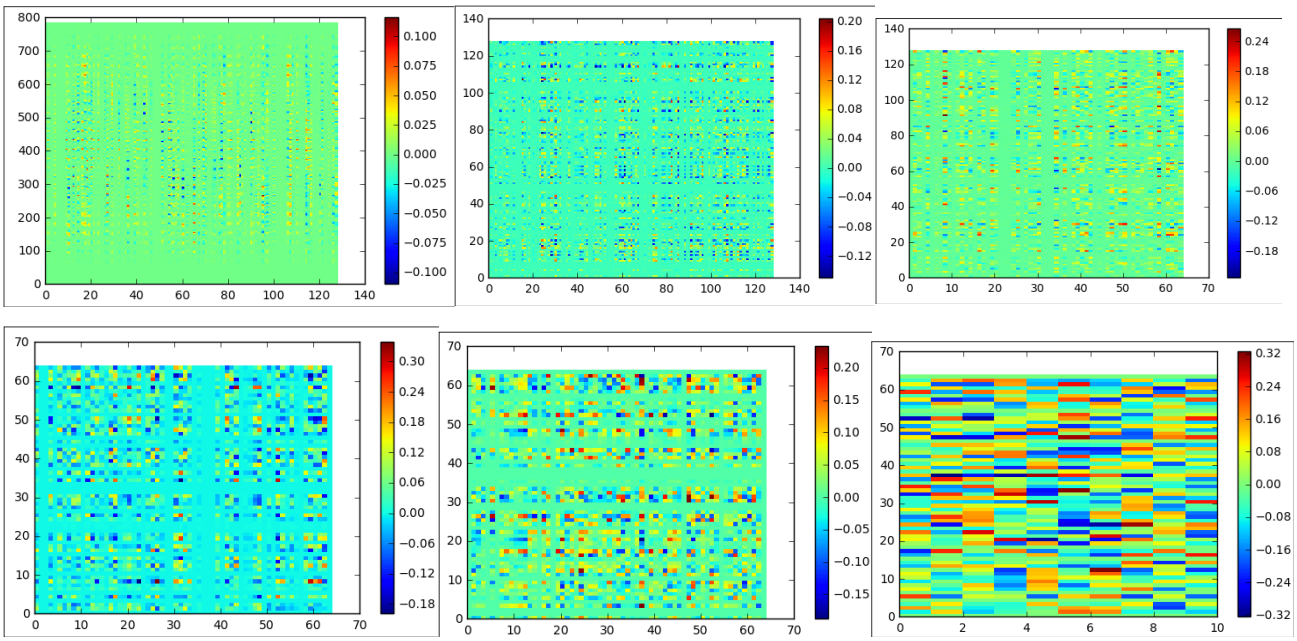


Fig. 4, c = 0.01, hidden layers weight heat map

2.4 L2 C = 0.001 and 0.0001

This model has even better than other l2-normalized models, as the test accuracy could reach as high as 98.01%, and the weights especially in the input layer has meshed grid patterns, as the table showing below and plots included in appendix:

min training error		min validation error		Test error		Test accuracy
0.1705		0.0719		0.01980		0.9802
	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	0.23307	0.12451	0.30737	0.31910	0.22119	0.125

The test accuracy is better than benchmark case, although there is no big difference. They are both very high accuracy indeed. That showed us that the model is less overfitting, therefore yields better performance.

When c is 0.0001, the result does not become better, so the magic of L2 regularization might already been fully exploited.

- Training error becomes smaller, as in bench mark case, which means the regularization power is not strong enough, and model tend to be overfitting.

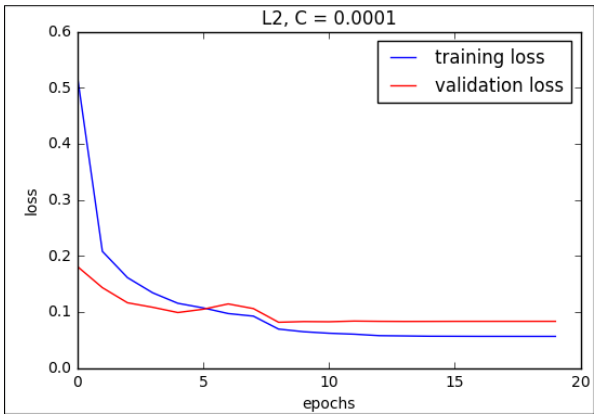


Fig. 5, c = 0.0001, L2 training loss

min training error		min validation error		Test error		Test accuracy
0.0562		0.0814		0.020299		0.9797
	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	0.19669	0.06427	0.18115	0.21460	0.18847	0.092188

3. L1-normalization

3.1 L1 C =1.0, 0.1, 0.01

Under L1 normalizations, the weight matrix is presumed to be even sparser. As it turns out, when c = 1, 0.1, 0.01, the resulted models have weights too sparse to hold any useful information, and the accuracies are equally bad. Please refer to the notebook for graphs.

min training error		min validation error		Test error		Test accuracy
2.3010		2.3021		0.8865		0.1135
	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	1.0	1.0	1.0	1.0	1.0	1.0

3.2 L1 C = 0.001

This is a reasonable regularization parameter to generate useful models. As we can see from the result, weights are much sparser (only less than 5% weights are non-zero) but we still have good prediction of test set.

min training error		min validation error		Test error		Test accuracy
0.4390		0.1426		0.0403		0.9597
	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	0.96387	0.98767	0.98132	0.96435	0.95434	0.75

- The MNIST set can be classified with sparse model, which means most of the features are intrinsically less important and L1 regularization will give us a satisfying result.
- Most of the weights in benchmark or L2 normalization cases can be further dropped, which lead us to think if L2 regularization with drop out layers will generate even better model.
- Recall that L2 norm will result in grid meshing weights, here L1 norm 'concentrates' weights more so that we can barely see mesh grids in the input layer.

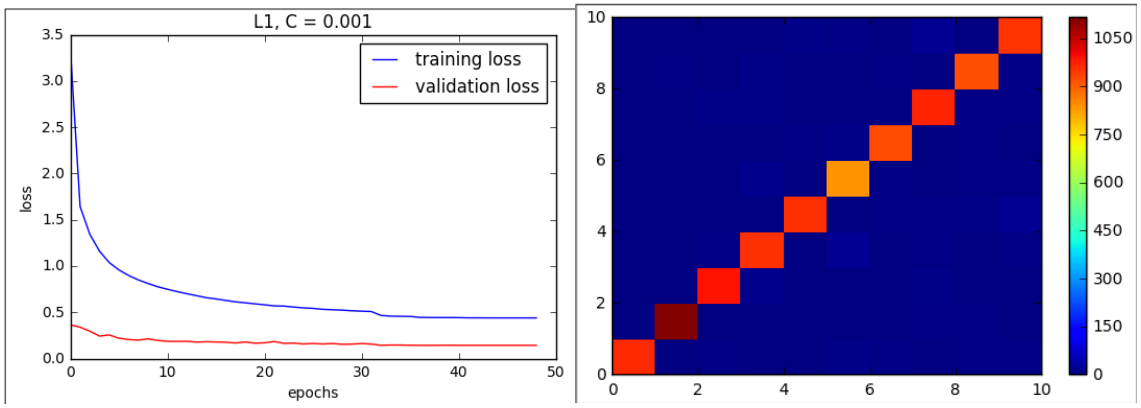


Fig. 6 c = 0.001, (a), training and validation loss; (b), testing accuracy confusion heat map

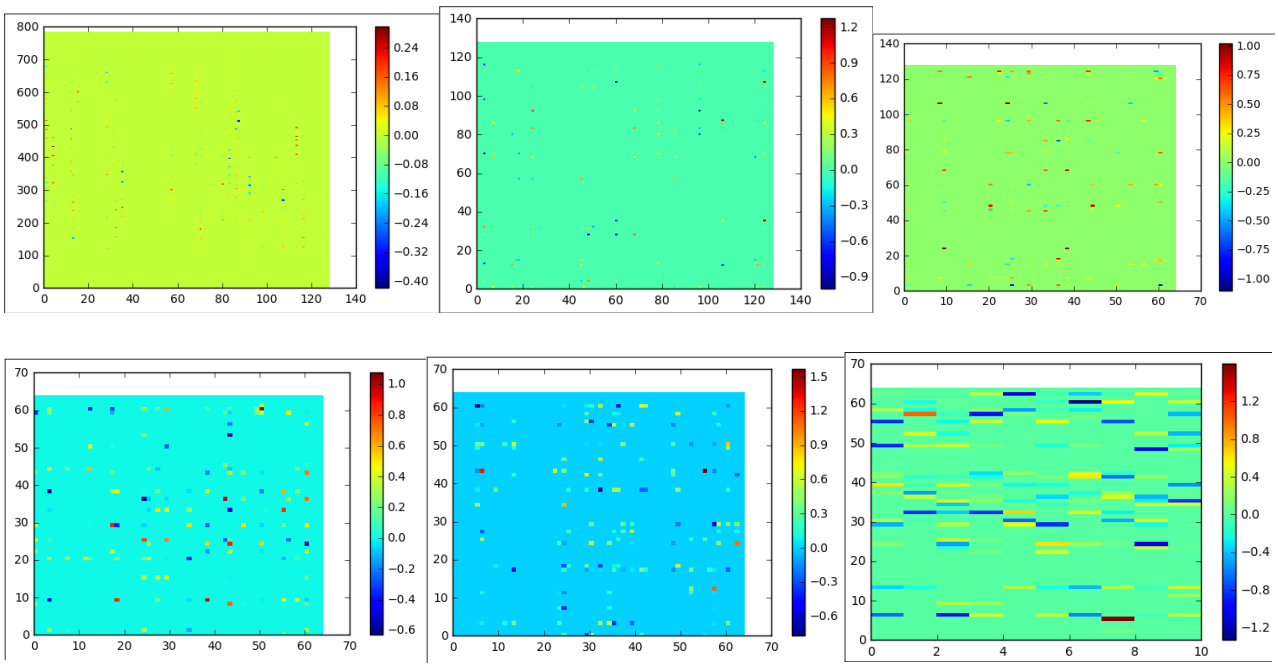


Fig. 7, c = 0.001, hidden layers weight heat map

3.3 L1 C = 0.0001

This model generates best performance comparing with all above, with sparse weights, which showed us that overfitting exists in MNIST DNN model, and L1 regularization will resolve it.

- With less sparse model than c = 0.001, we can capture relatively more details in dataset, and it helps us improve performance.
- Too much L1 regularization will also harm the performance.
- Meshed grid indeed exist in the input layer, as now we can conclude that the most important weights are kept and showed below.
- The output layer denser.

min training error	min validation error	Test error	Test accuracy
0.1763	0.0675	0.01939	0.9806

	Input layer	Layer 1	Layer 2	Layer 3	Layer 4	Output layer
Sparsity	0.82465	0.69952	0.64258	0.56592	0.52222	0.2

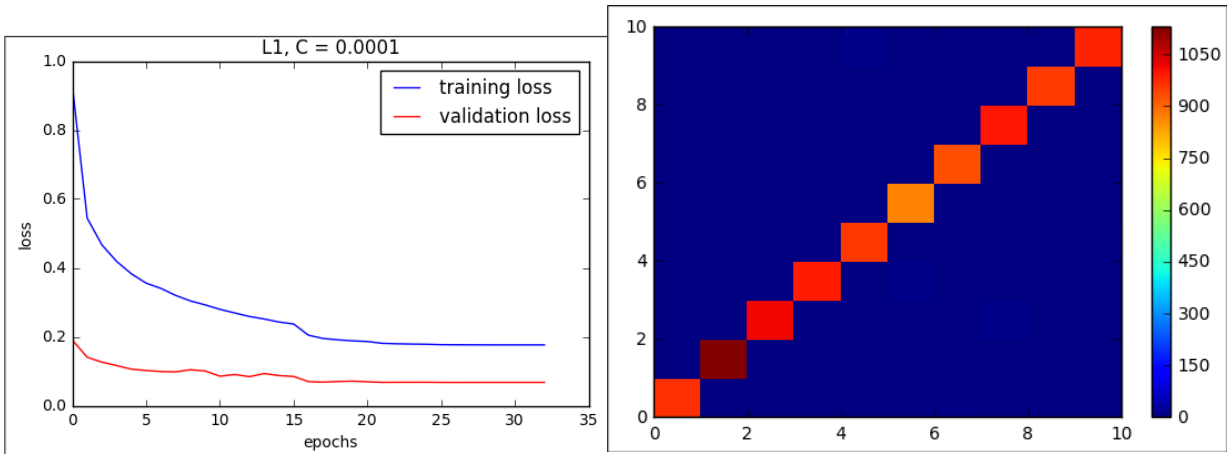


Fig. 8 c = 0.0001, (a), training and validation loss; (b), testing accuracy confusion heat map

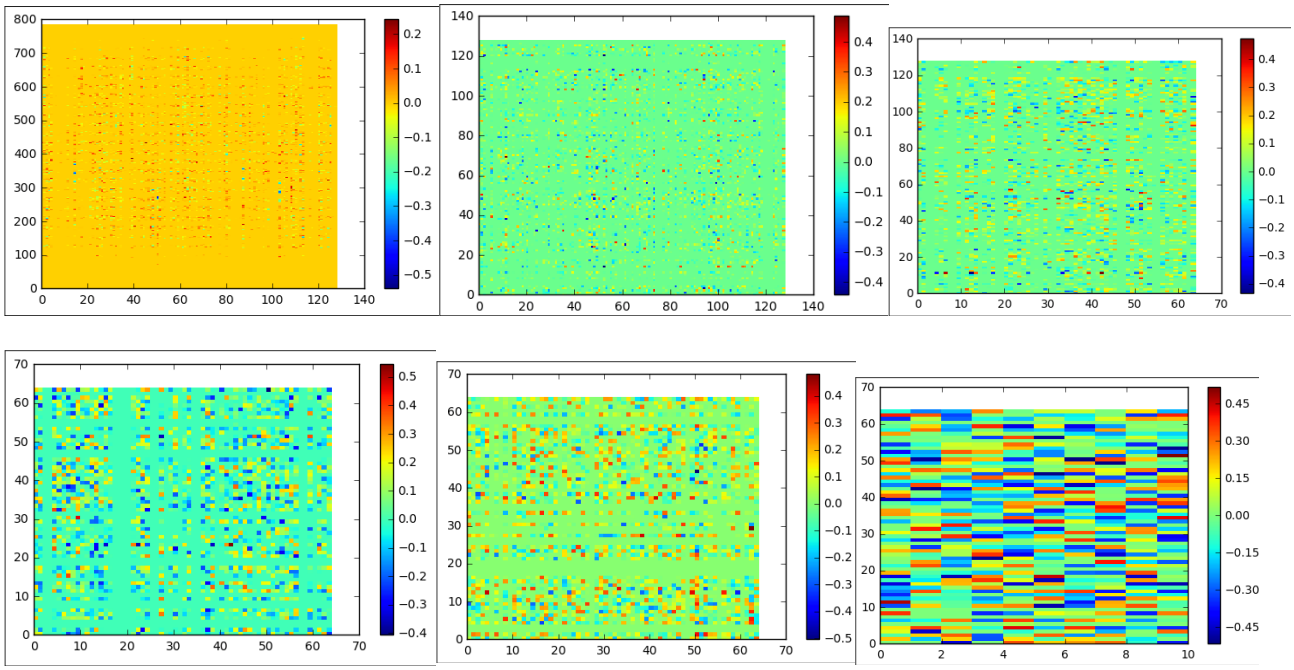
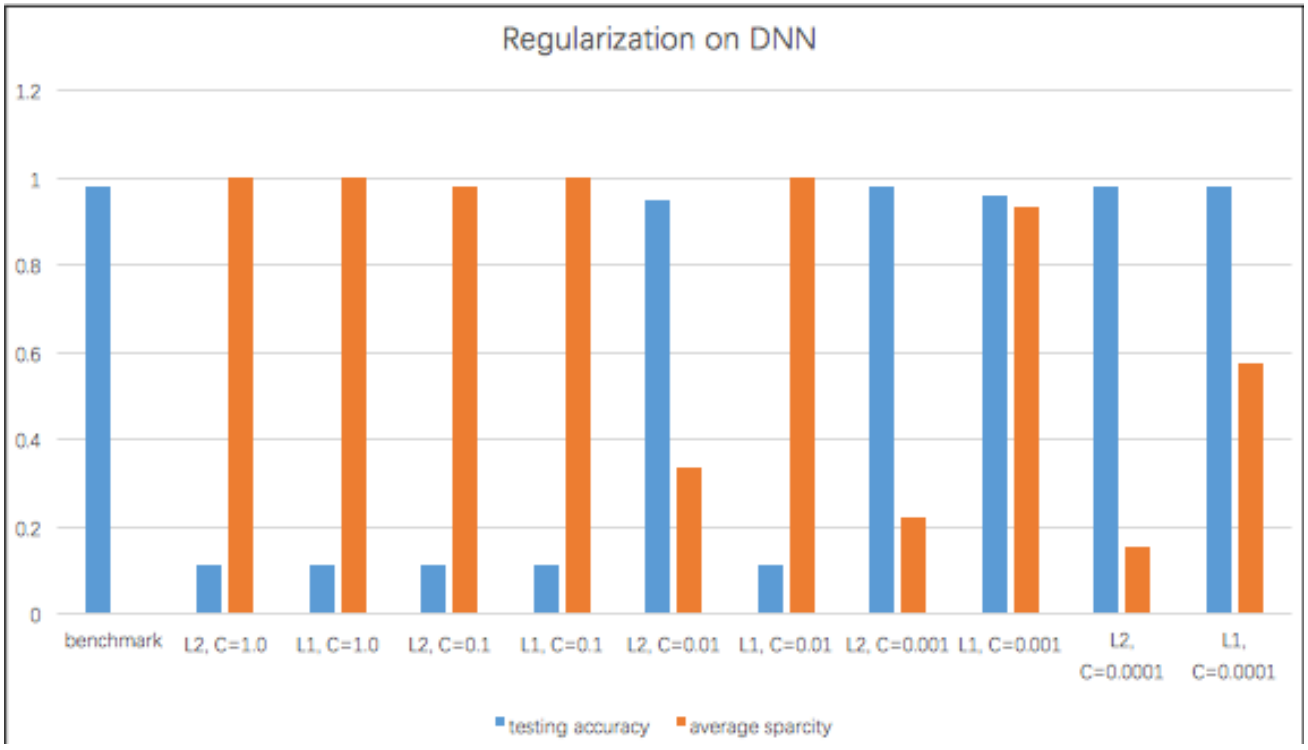


Fig. 9, c = 0.0001, hidden layers weight heat map

Conclusions and general findings on regularization



From the comparison above, we can conclude that L1 regularization with c = 0.0001 is the best fit for our MNIST classify dataset. Some wrap-up general points are listed below:

- For easily separable datasets (not necessarily linear), DNN is a perfect tool to generalize a model in high-dimensional space

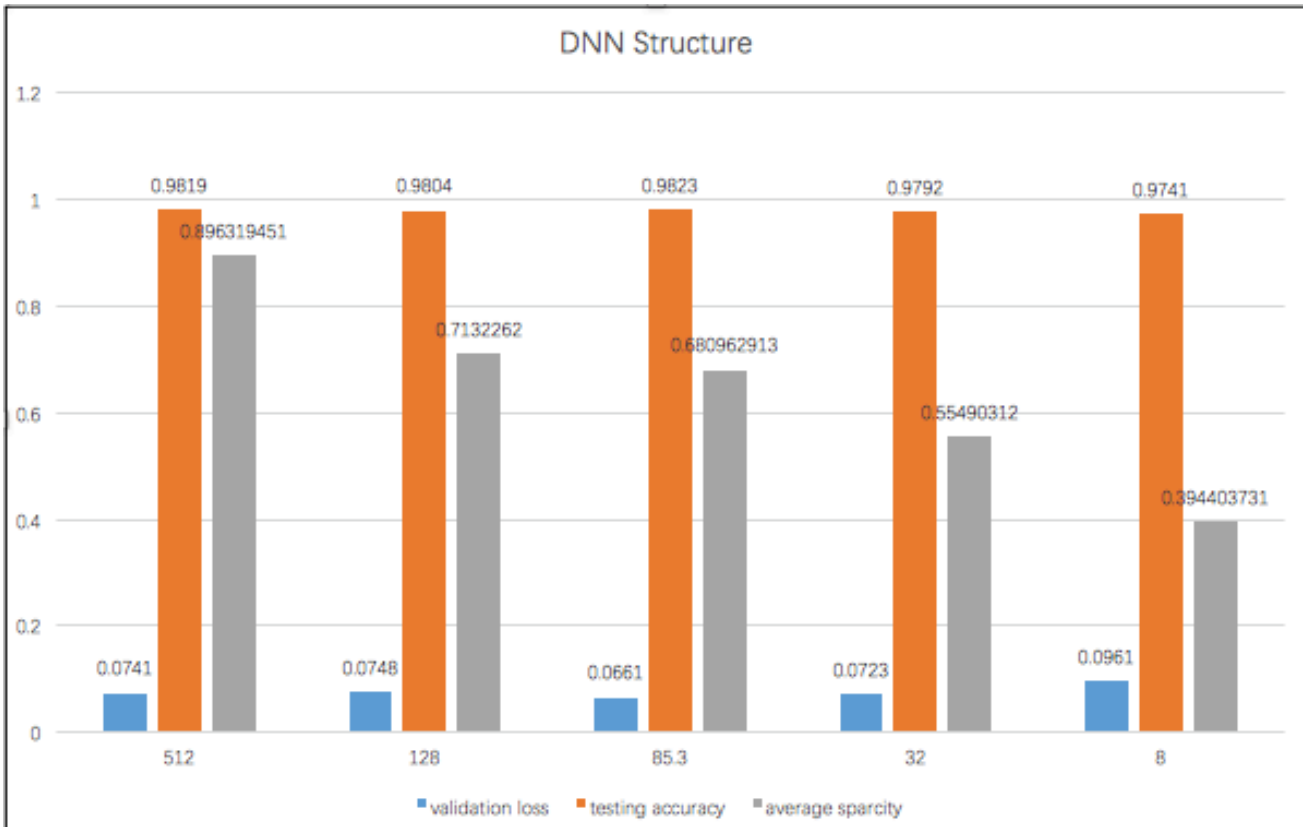
- If the data is simple and some high-dimensional abstraction is available for feature selection, sparse model (simple model) will most probably yield better result than dense weight DNN. So L1 regularization is more helpful in this case
- C too big will result in weights decay too much, therefore cannot predict anything
- C too small will not have enough decaying power (most obvious in L2 norms), so weights will become denser, and tends to over fit, as shown above
- Best model should choose L1 and L2 norm accordingly, and adjust regularization parameter very carefully so as to have high accuracy in testing case, while maintaining high sparsity
- Features that self-assemble into grids will maintain grid property in weight space, as in 2D input

Structure of DNN

When referring to deep neural network, depth is of vital importance. Here we tested several DNN models on MNIST dataset, varying the depth and width of each layer to see how it will affect the final testing accuracy.

- We keep total number of neurons as a constant, in this case, we pick N = 512 to be consistent with above experiments
- From shallow to deep, we vary network layers as
  (784) 512 (10);
  (784) 256 256 (10);
  (784) 256 128 128 (10);
  (784) 128 128 128 128 (10);
  (784) 64 64 64 64 64 64 64 64 (10);
  five cases
- We choose L1-regularization with c = 0.0001, because it is the best case so far.

The results are listed as below with N = 512:

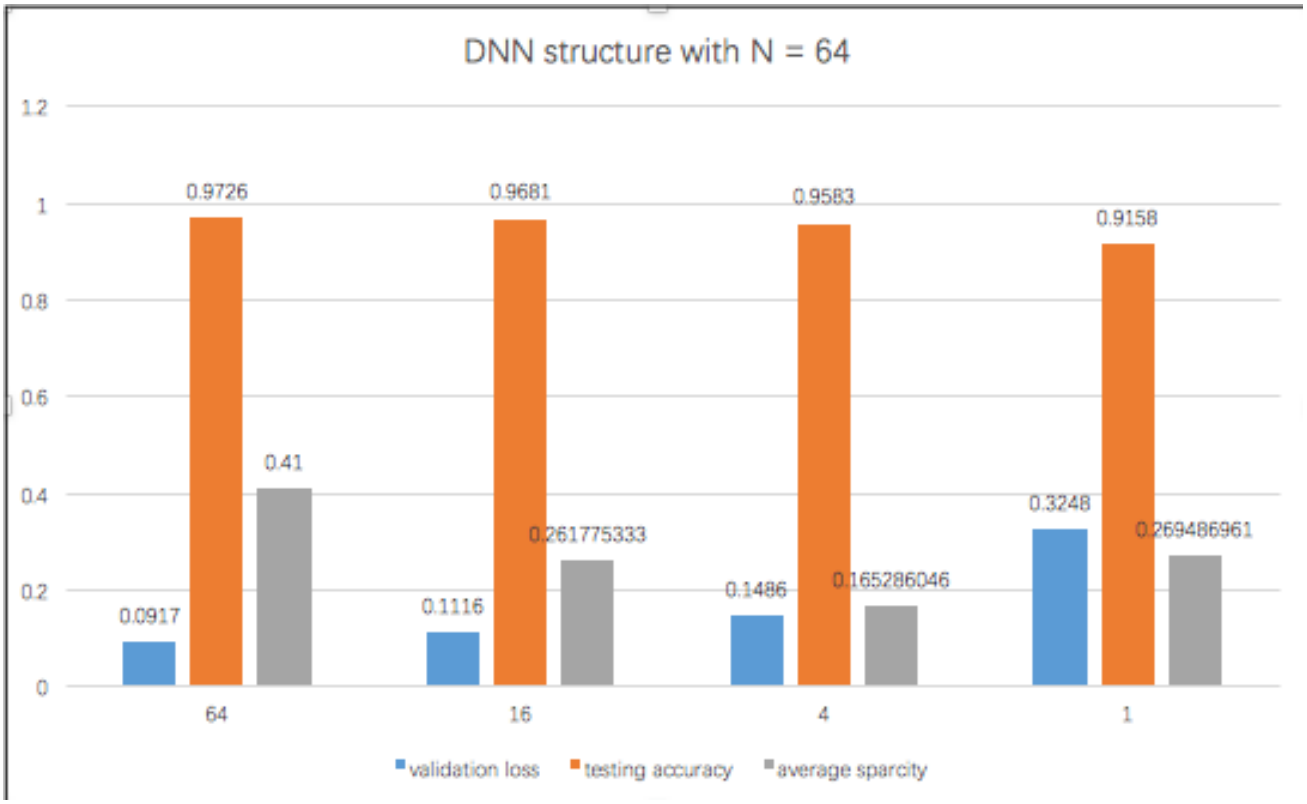


the horizontal axis is max network width divided by network depth.

Conclusions on structure of DNN

- These DNN models share almost same testing accuracy, so deep network does not necessarily mean better performance. However, MNIST set is easy to split, that is probably the main reason deeper network does not have significant performance boost.
- Shallow network is sparse, and lower validation loss. This conclusion is not obvious to me.
- To try smaller scale of neurons, I limit the number of total neuron to 64, and the result is shown

below.



- Less total number of neurons will generally weaken the ability of the classifier model.
- When each layer does not have sufficient number of neurons, the network will not reach better result even if the structure is deeper. That's because less neurons in a layer means less generalizing power for the layer, and therefore it becomes more difficult for the next layer to get information from the previous one. Intuitively, I recommend each layer (except output layer) should be no less than 1/10 of original input dimension, as in this case, 64 neuron layer still have pretty good result.
- As networks going deeper, the regularization term has less power in limiting the weights sparsity, that's most likely due to the difficulty of weights propagation from top output layer to middle hidden layer increases when the net grows deeper, therefore the restraining power become less averaging to each layer for the same c constant. So it is reasonable to argue that when DNN goes deeper, regularization term should increase in order to achieve similar sparsity.

Different non-linear activation functions

In this chapter, we pick the best performing model from above results and apply different non-linear activation functions to them. Up till now, all the results are from relu function, and we will compare that with sigmoid and tanh function to see the difference.

	sigmoid	tanh	relu
value space	(0,1)	(-1,1)	[0,+inf)
training epoch	36	26	23
tvalidation loss	0.1793	0.0841	0.0925
testing accuracy	0.9527	0.9797	0.9792

average sparsity	0.121277417	0.086090688	0.201346117

From the comparison above we can see that

- From all training processes, sigmoid takes more time and epoch to finish the training, while yield highest validation loss
- It's because the set is separable, and our target hypothesis has sparse weights, while sigmoid never reaches 0 in its value space, it is difficult for it to generalize some weights to 0, and we are solely depending on regularization. However, sigmoid is the most sensitive activation function among all 3 in terms of regularization. C should be smaller and has very narrow space to take value from, and the result shows that its generality is not very good.
- I assume sigmoid would be more useful in some non-separable high dimensional datasets.
- The loss in training process is higher for sigmoid, however, under the same circumstances testing accuracy is lower. So it's not very good fit for the model.
- Tanh function reaches best performance in this set of comparison, but the model is very dense, should be applying more regularization.
- Surprisingly, relu is the best-performing activation function in this set. Probably because its simplicity, and yields zero to negative inputs, and put weights decay in a linear formation. So the model can be trained to certain sparsity and only keeping the most important weights.

Conclusion

As we have compared and concluded, the best performing DNN model in classfying MNIST dataset is deep structured, L1-regularized and with a relu activation function. However, a simple model could also achieve satisfying test accuracy, so there is no big difference.

In real-world datasets, we might be facing with a more complex and scattered, non-separable data, in that case, I conclude that we should choose model parameters for regularization carefully, try L1 to see if the model can be sparse, and avoid overfitting.

If we have sufficient computation power to build a large network, it must be better, however, considering the input dimensions, we should try to build a network that concludes each previous layer properly, and goes as deep as possible. This will also give us a hint of how much neurons might be used in the minimal setting.

Last, activation function does not play an important role in this model and dataset, choose relu will probably speed things up.

The visualization of best model from Keras is attached in appendix.

Appendix

I) plots and graph

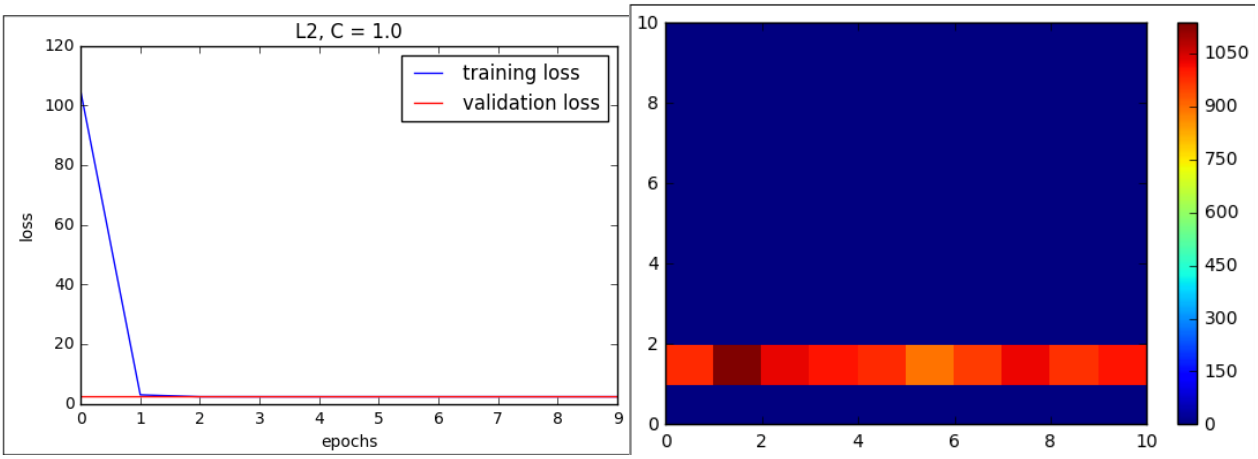


Fig. 3 L2, c= 1.0, (a), training and validation loss; (b), testing accuracy confusion heat map

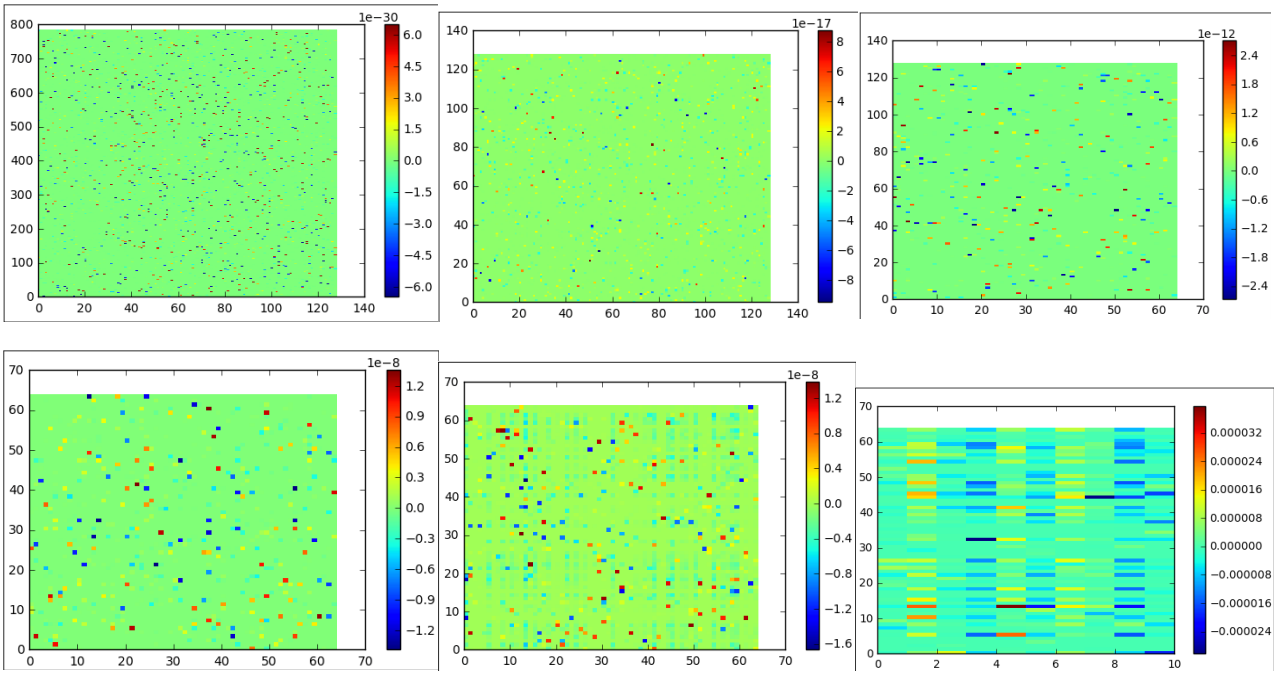


Fig. 4, L2, c= 1.0, hidden layers weight heat map

L2, C = 0.1

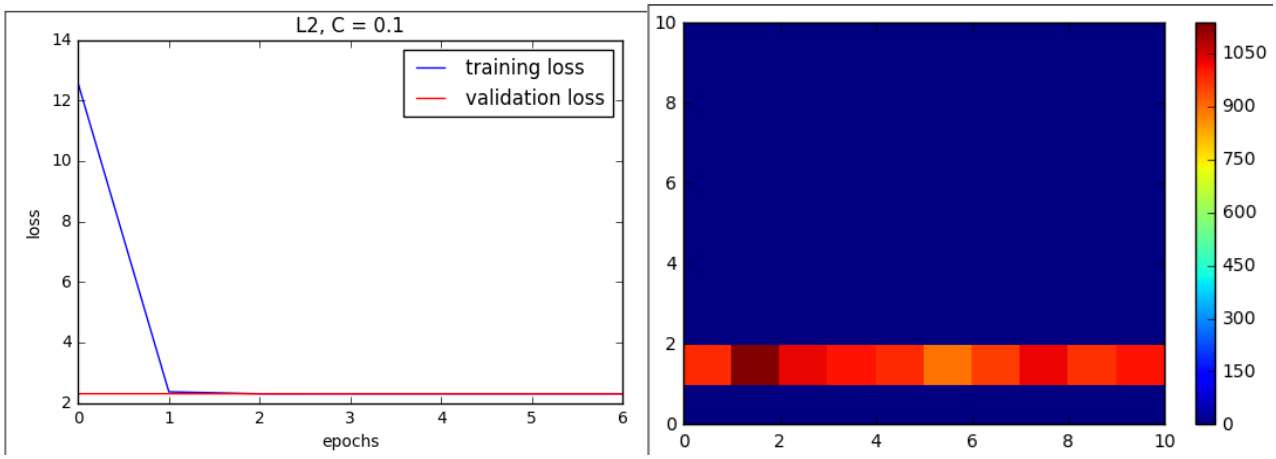


Fig. 5 L2, c= 0.1, (a), training and validation loss; (b), testing accuracy confusion heat map

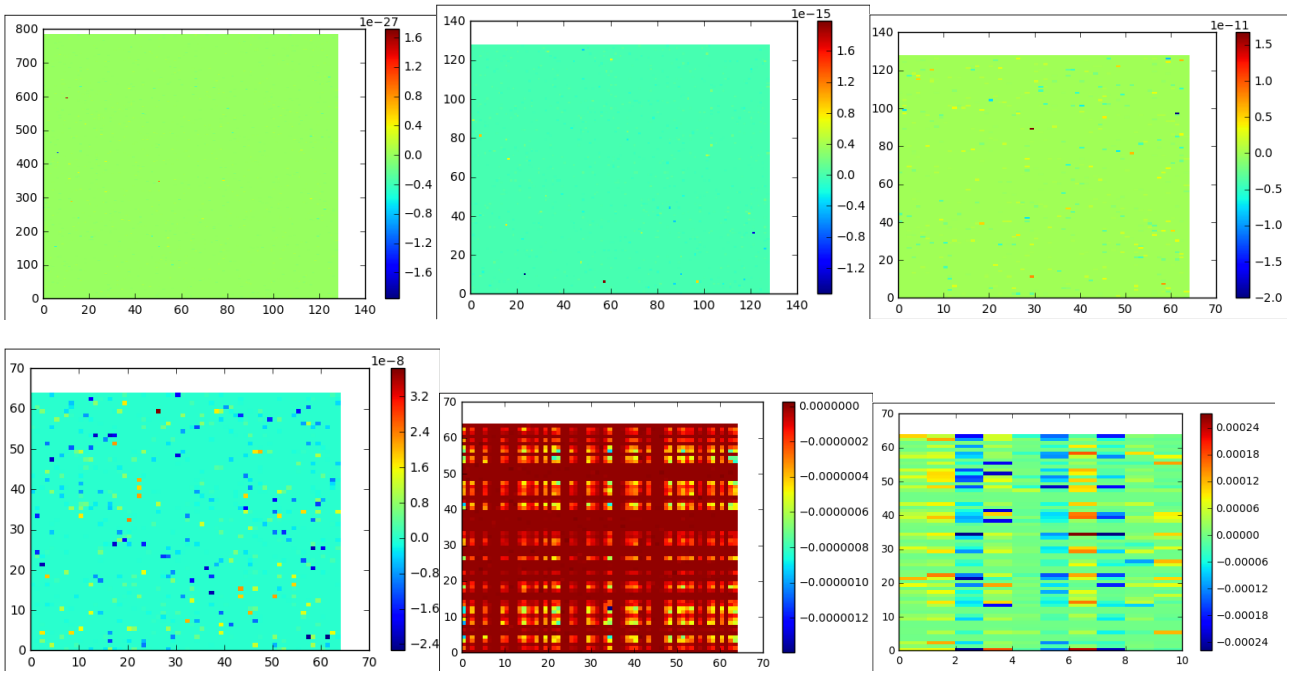


Fig. 6, L2, c= 0.1, hidden layers weight heat map

L2, C = 0.001

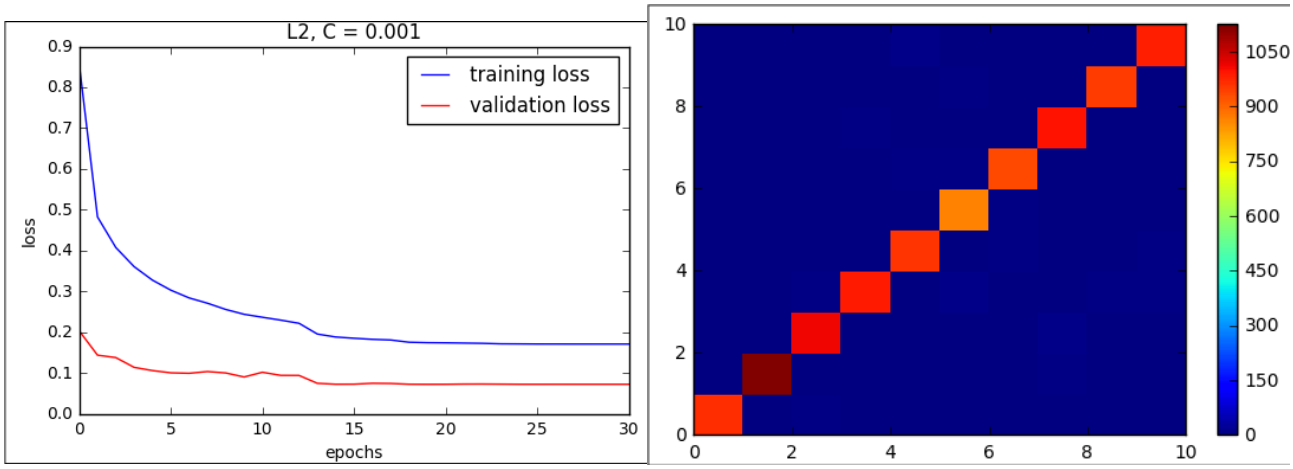


Fig. 5 L2, c= 0.001, (a), training and validation loss; (b), testing accuracy confusion heat map

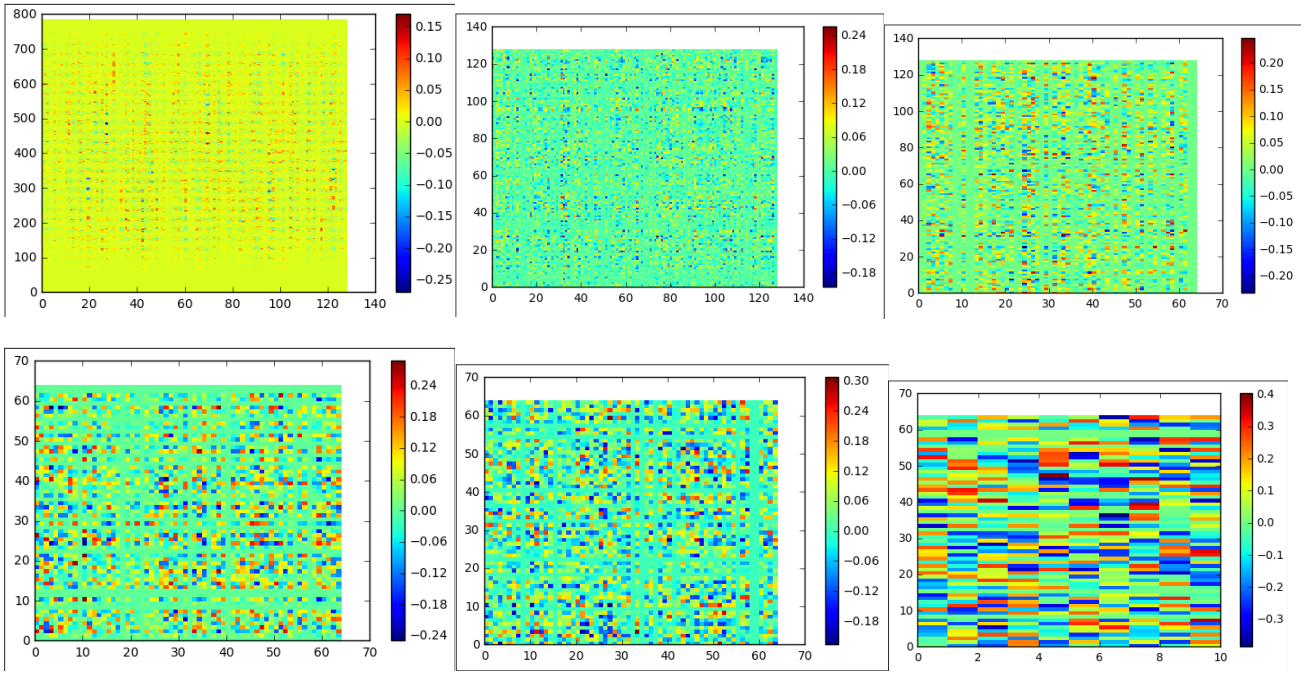
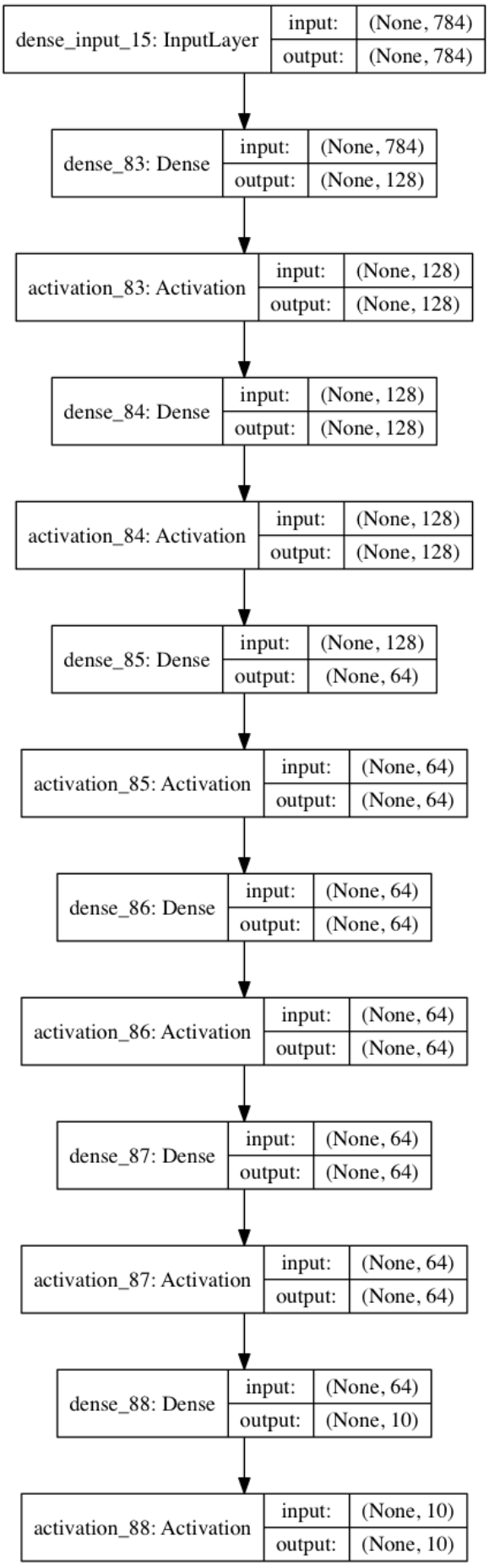


Fig. 6, L2, c= 0.001, hidden layers weight heat map


dense_input_15: InputLayer
input: (None, 784)
output: (None, 784)
dense_83: Dense
input: (None, 784)
output: (None, 128)
activation_83: Activation
input: (None, 128)
output: (None, 128)
dense_84: Dense
input: (None, 128)
output: (None, 128)
activation_84: Activation
input: (None, 128)
output: (None, 128)
dense_85: Dense
input: (None, 128)
output: (None, 64)
activation_85: Activation
input: (None, 64)
output: (None, 64)
dense_86: Dense
input: (None, 64)
output: (None, 64)
activation_86: Activation
input: (None, 64)
output: (None, 64)
dense_87: Dense
input: (None, 64)
output: (None, 64)
activation_87: Activation
input: (None, 64)
output: (None, 64)
dense_88: Dense
input: (None, 64)
output: (None, 10)
activation_88: Activation
input: (None, 10)
output: (None, 10)