Machine Learning Practical

Coursework 2 report

2016年11月24日

Part 1: Does combining L1 and L2 regularisation offer any advantage over using either individually?

Part 2: Data Augmentation

Part 3: Models with Convolutional Layers

Part 1: Does combining L1 and L2 regularisation offer any advantage over using either individually?

Statement

Is it better than using L1 L2 regularisation individually that combining L1 and L2 regularisation? And how better does it perform?

Motivation

Compare the performance of using combining L1 and L2 regularisation and using them individually. And invest how better it perform. I train data with combining L1 and L2 or without combining instead using them individually or without any regularisation and plot the performance on validation dataset.

Experimental Methodology

One method for trying to reduce overfitting is therefore to try to decrease the flexibility of the model. We can do this by simply reducing the number of free parameters in the model (e.g. by using a shallower model with fewer layers or layers with smaller dimensionality). A common method for doing this is to add an additional term to the objective function being minimised during training which penalises some measure of the complexity of a model as a function of the model parameters.

Two commonly used norms are the L1 and L2 norms.
For a D dimensional vector v the L1 norm is defined as

$$\|v\|_1 = \sum_{d=1}^{D} |v_d|$$

and the L2 norm is defined as

$$\|v\|_2 = \left[\sum_{d=1}^{D} |v_d^2|\right]^{\frac{1}{2}}$$

A L1 penalty on the ith vector parameter p(i) (or matrix parameter collapsed to a vector) is then commonly defined as

$$C_{L1}^{i} = \beta i \|p^{(i)}\|_2 = \beta i \sum_{d=1}^{D} |p_d^{(i)}|$$

This has a gradient with respect to the parameter vector

$$\frac{\partial C_{L1}^{(i)}}{\partial p_d^{(i)}} = \beta_i \operatorname{sgn}(p_d^{(i)})$$

Similarly a L2 penalty on the ith vector parameter p(i) (or matrix parameter collapsed to a vector) is commonly defined as

$$C_{L2}^{i} = \frac{1}{2}\beta i \left\| p^{(i)} \right\|_2^2 = \frac{1}{2}\beta i \sum_{d=1}^{D} \left[(p_d^{(i)})^2 \right]$$

This has a gradient with respect to the parameter vector

$$\frac{\partial C_{L1}^{(i)}}{\partial p_d^{(i)}} = \beta_i p_d^{(i)}$$

And as for combining the L1 and L2, I define the combining penalty as

$$C_{combining} = \frac{C_{L1} + C_{L2}}{2}$$

And this has a gradient as

$$\frac{\partial C_{combining}^{(i)}}{\partial p_d^{(i)}} = \frac{\beta_i p_d^{(i)} + \beta_i \operatorname{sgn}(p_d^{(i)})}{2}$$

1. So define the L1Penalty and L2Penalty class and define the L1 and L2 penalties for a parameter and corresponding gradients to implement the call and grad methods respectively.
2. The parameter the penalty term (or gradient) is being evaluated for will be either a one or two-dimensional NumPy array (corresponding to a vector or matrix parameter respectively).
3. Define the combining L1Penalty and L2Penalty class and define the combining L1 and L2 penalties for a parameter and corresponding gradients to implement the call and grad methods respectively.
4. Set up the data providers, a penalty object has been assigned to weights_penalty a seeded random number generator assigned to rng. logger objects needed for training runs.
5. For each regularisation scheme, train the model for 100 epochs with a batch size of 50 and using a gradient descent with momentum learning rule with learning rate 0.01 and momentum coefficient 0.9.

Experiment 1

Implement:

1. Train the model and for each regularisation scheme store the run statistics (output of Optimiser.train) and the final values of the first layer weights for each of the trained models.
2. Plot the training set error against epoch number for all different regularisation schemes on the same axis. On a second axis plot the validation set error against epoch number for all the different regularisation schemes.
3. Defines two functions for visualising the first layer weights of the models trained above. The first plots a histogram of the weight values and the second plots the first layer weights as feature maps, i.e. each row of the first layer weight matrix (corresponding to

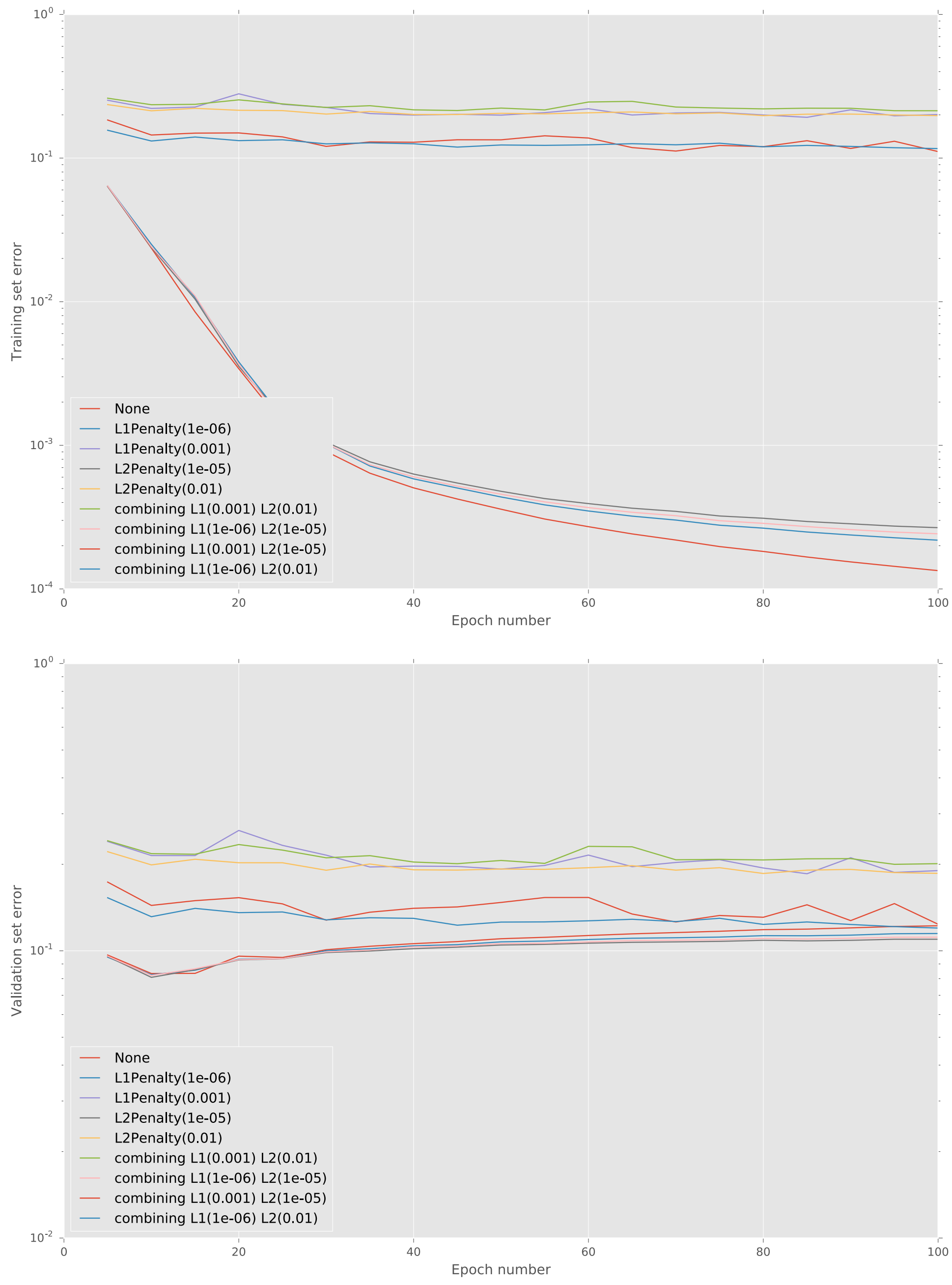

Figure 1

the weights going from the input MNIST image to a particular first layer output) is visualised as a 28×28 image. In these feature maps white corresponds to negative weights, black to positive weights and grey to weights close to zero.

Quantitative results:

In figure 1.1 here is the case where I use the L1 and L2 regularisation individually and using combining L1 and L2 and not using any regularisation method. And show the evolution of the **error** for both the **training** and **validation** sets.

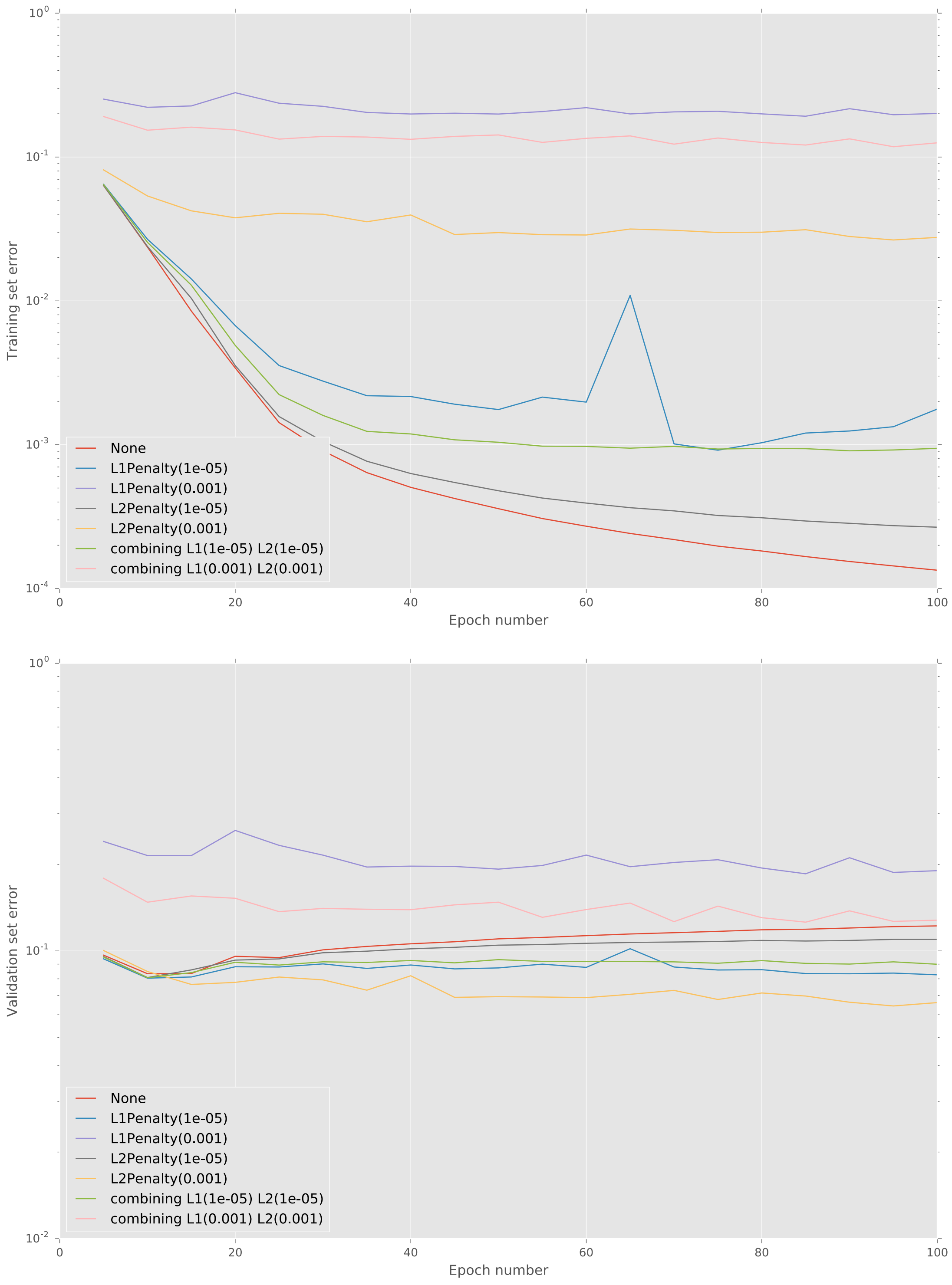

Figure 2

	None	L1 Penalty(1E-6)	L1 Penalty(1E-3)	L2 Penalty(1E-5)	L2 Penalty(1E-2)	L1(1E-3)+ L2(1E-2)	L1(1E-6)+L2(1E-5)	L1(1E-3)+L2(1E-5)	L1(1E-6)+L2(1E-2)
final error(train)	1.34E-04	2.19E-04	2.01E-01	2.67E-04	1.98E-01	2.14E-01	2.42E-04	1.11E-01	1.16E-01
final error(valid)	1.22E-01	1.15E-01	1.9E-01	1.1E-01	1.86E-01	2.01E-01	1.12E-01	1.24E-01	1.2E-01

	None	L1Penalty(1e-05)	L1 Penalty(1E-3)	L2(1E-5)	L2(1E-3)	L1(1E-5)+L2(1E-5)	L1(1E-2)+L2(1E-2)
final error(train)	1.34E-04	1.77E-03	2.01E-01	2.67E-04	2.76E-02	9.44E-04	1.26E-01
final error(valid)	1.22E-01	8.26E-02	1.9E-01	1.1E-01	6.61E-02	8.99E-02	1.28E-01

Discussion and conclusion:

1. From the figures 1 and table above we can see that using regularisation performs better than using none by compare the performance on validation data and training data. Whatever using combining regularisation or regularisation individually they all

slightly weekend the overfitting which we can see by comparing to the one without using regularisation on training and validation dataset.

2. From the figures 1, 2 table above we can see that smaller parameters performs better when using L1 or L2 regularisation by compare the L1(1e-5) and L1(1e-3) or L2(1e-4) and L2(1e-2) or combining(1e-3, 1e-2) and combining(1e-5, 1e-4).
3. From the figures 1, 2 and table above we can see that combining the L1 and L2 offers slight advantage over using them individually by comparing the validation set error when the parameters assigned the smaller or larger value at the same time.
4. From the figures 1,2 and table above we can see that combining the L1 which using smaller parameter(according to above discussion there will be a good performance) and L2 which using larger parameter(will perform worse) could perform almost as well as using smaller parameter by using regularisation individually.

So if we depends the result in this experiment we could find that combining L1 and L2 can weekend overfitting when we train the data and offer a little better performance on training a model than using nothing and using them individually. There may be advantages that combining can improve the performance of regularisation using a smaller parameter and performing bad.

Part 2: Data Augmentation

Statement

Does it improve the performance when using data augmentation? What data augmentation technology are there? And how better do them perform?

Motivation

Compare the performance of using different data augmentation technologies and without using them. And invest how better they perform. I train data with using different data augmentation and without them and plot the performance on validation dataset.

Experimental Methodology

Another technique mentioned in the lectures for trying to reduce overfitting is to artificially augment the training data set by performing random transformations to the original training data inputs. The idea is to produce further artificial inputs corresponding to the same target class as the original input. The closer the artificially generated inputs are to the appearing like the true inputs the better as they provide more realistic additional examples for the model to learn from.

One simple way to use data augmentation is to just statically augment the training data set - for example we could iterate through the training dataset applying a transformation function like that implemented above to generate new artificial inputs, and use both the original and newly generated data in a new data provider object.

An alternative is to randomly augment the data on the fly as we iterate through the data provider in each epoch. In this method a new data provider class can be defined that inherits from the original data provider to be augmented, and provides a new next method which applies a random transformation function like that implemented in the previous exercise to each input batch before returning it.

1. Define the function to apply rotation, shift, zoom out and add noise for the handwritten image inputs in the MNIST dataset
2. Define the function to to visualise the images in a batch before and after application of the those transformations.
3. test implementation.
4. For each data augmentation scheme, train the model for 100 epochs with a batch size of 100 and using a gradient descent with momentum learning rule with learning rate 0.01 and momentum coefficient 0.9

Experiment 2

Implement:

1. Randomly augment the data on the fly as we iterate through the data provider in each epoch. In addition to the arguments of the original MNISTDataProvider. Init method, this additional takes a transformer argument, which should be a function which takes as arguments an inputs batch array and a random number generator object, and returns an array corresponding to a random transformation of the inputs.
2. Train a model with the same architecture as in part 1 and with no L1 / L2 regularisation using a training data provider which randomly augments the training images using random_rotation, random_shift, random_noise transformer function.
3. Plot the training and validation set errors over the training epochs.

Quantitative results:

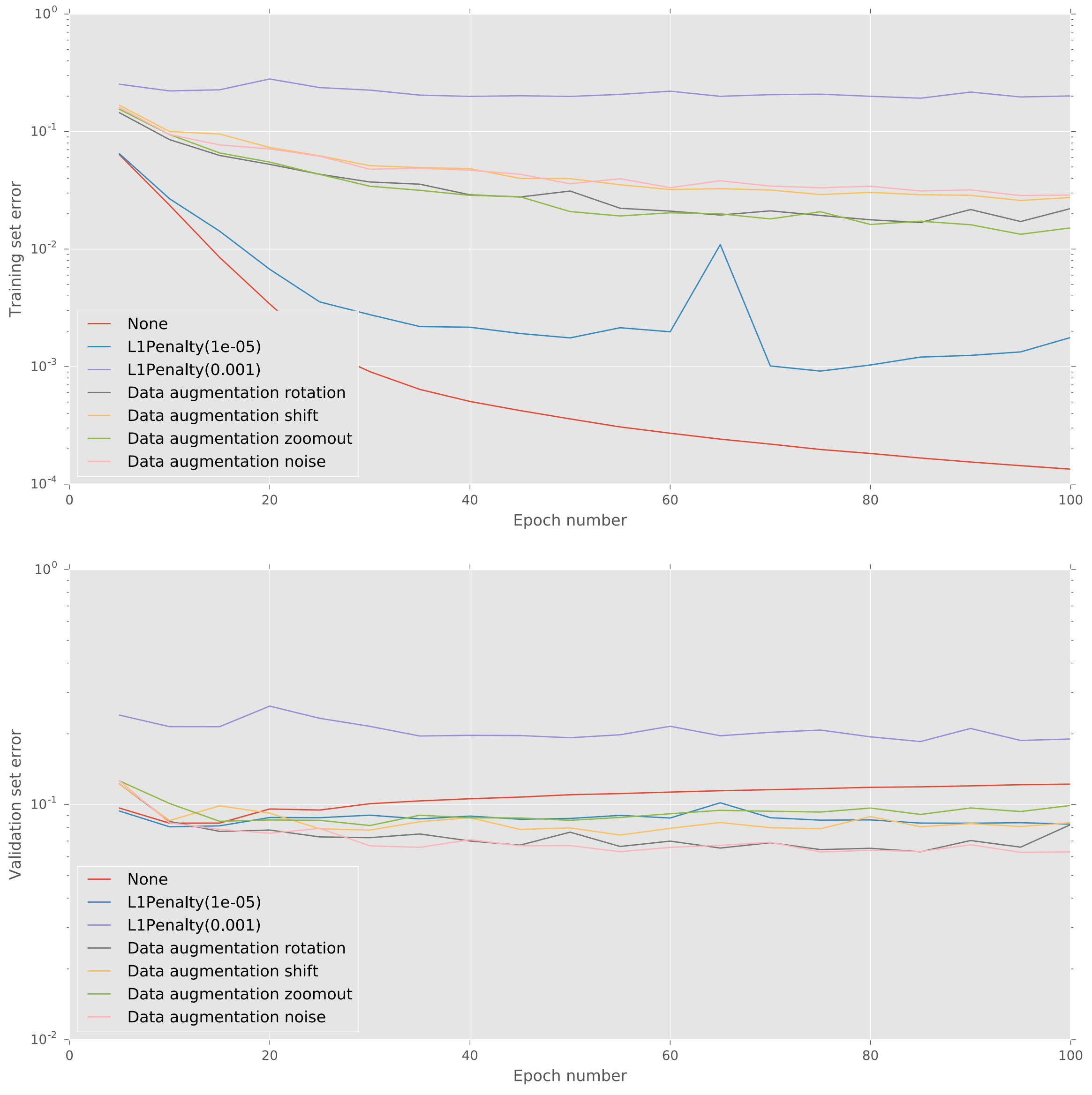

figure 3

	None	L1 Penalty(1E-5)	Rotation	Shift	Zoom out	Noise
final error(train)	2.21E-02	2.75E-02	2.01E-01	2.75E-02	1.52E+02	2.88E-02
final error(valid)	8.22E-02	8.37E-02	1.9E-01	8.37E+02	9.92E+02	6.29E-02

Discussion and conclusion:

1. From the figures 3 and table above we can see that using data augmentation performs better than using none by compare the performance on validation data and training data. Whatever using any data augmentation technology they all weekend the overfitting which we can see by comparing to the one without using regularisation on training and validation dataset.
2. From the figures 3 table above we can see that using adding noise method we can get better performance of weekend overfitting and the result is better than others. But theoretically using "realistic" distortions to create new data is better than adding random noise. I think the reason of adding noise performing better is that the I make the value of parameter in adding noise method so perfect and other parameters in other method not that perfect. So I need to compare the different parameters in adding noise method and with other methods.
3. From the figures 3 and table above we can see that sometimes using these data augmentation method better than combining the L1 and L2 to weekend overfitting. But sometimes it is not. This is not comparable, because combining method is influent by the parameter, so the two methods in general is about the same. And using data augmentation may cost more time than using combining the L1 and L2.

So if we depends the result in this experiment we could find that data augmentation performs better than using using nothing to weekend overfitting. And using data augmentation and regularisation is almost the same.

Part 3: Models with Convolutional Layers

Statement

How the parameters in the convolutional layer and the maxpooling layer affect the results of training?
How the maxpooling layer affect the evolution of training with convolutional layers?
Whether using three the affine transformations interleaved with logistic sigmoid, affect the performance of training with convolutional layer?

Motivation

Compare the 5 different kinds of models to evaluate the models with convolutional layers

Experimental Methodology

1. Make all the conditions needed to compare as default. Set different values of kernel dim or pool size and invest the result.
2. Make all the conditions needed to compare as default. There are two conditions needed to compare, with maxpooling layers, without maxpooling layers.
3. Make all the conditions needs to compare as default. There are two conditions needed to compare, using three the affine transformations interleaved or not.

4 . design 5 models implement this experiment:

	model1	model2	model3	model4	model5
kernel dim	5	9	5	5	5
maxpooling layers	yes	yes	yes	no	yes
pool size	4	4	8	4	4
affine transformation interleaved	three	three	three	three	one

Experiment 3

Implement:
1. Define the ConvolutionalLayer and implementation of fprop for convolutional layer, outputs are calculated only for 'valid' overlaps of the kernel filters with the input - i.e. without any padding. It is also assumed that if convolutions with non-unit strides are implemented the default behaviour is to take unit-strides, with the test cases only correct for unit strides in both directions.
2. The three test functions are defined in the cell below. All the functions take as first argument the class corresponding to the convolutional layer implementation to be tested (not an instance of the class). It is assumed the class being tested has an init method with at least all of the arguments defined in the skeleton definition above. A boolean second

argument to each function can be used to specify if the layer implements a cross-correlation or convolution based operation (see note in seventh lecture slides). An example of using the test functions if given in the cell below. This assumes you implement a convolution (rather than cross-correlation) operation. If the implementation is correct.
3. Set up
the data providers
a penalty object has been assigned to weights_penalty a seeded random number generator assigned to rng.
logger objects needed for training runs.
train the model using default parameteor below
num_epochs = 50
stats_interval = 5
batch_size = 50
learning_rate = 0.01
mom_coeff = 0.9
weights_init_gain = 0.5
biases_init = 0.
input_dim, output_dim, hidden_dim = 784, 10, 10
4. Train the 5 different model(ajust the parameter or other ajustment)
5. Plot the training set error against epoch number for all different models on the same axis. On a second axis plot the validation set error against epoch number for all the different models

Quantitative results:
1. kernel dim and pool size
group1:(model1 ,model2)   figure4

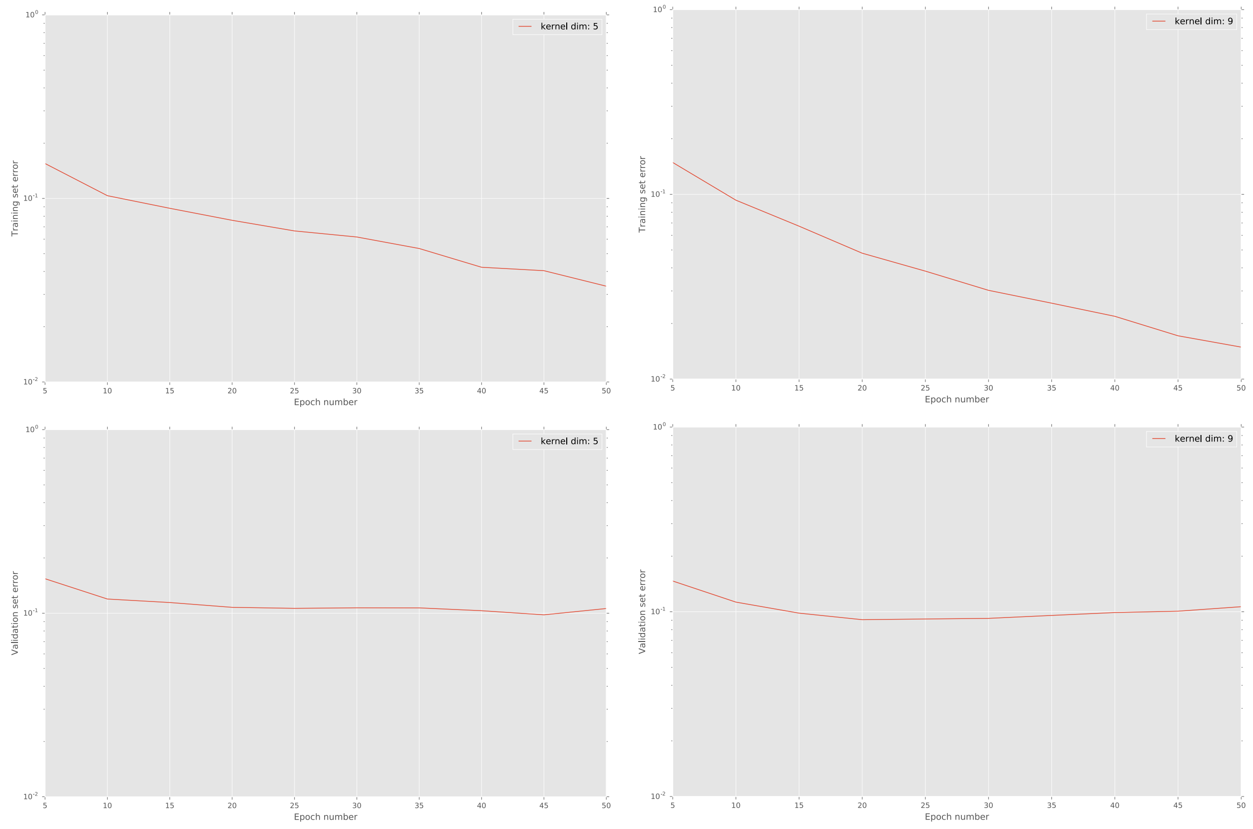

Figure 4

group2:(model1, model3) figure 5

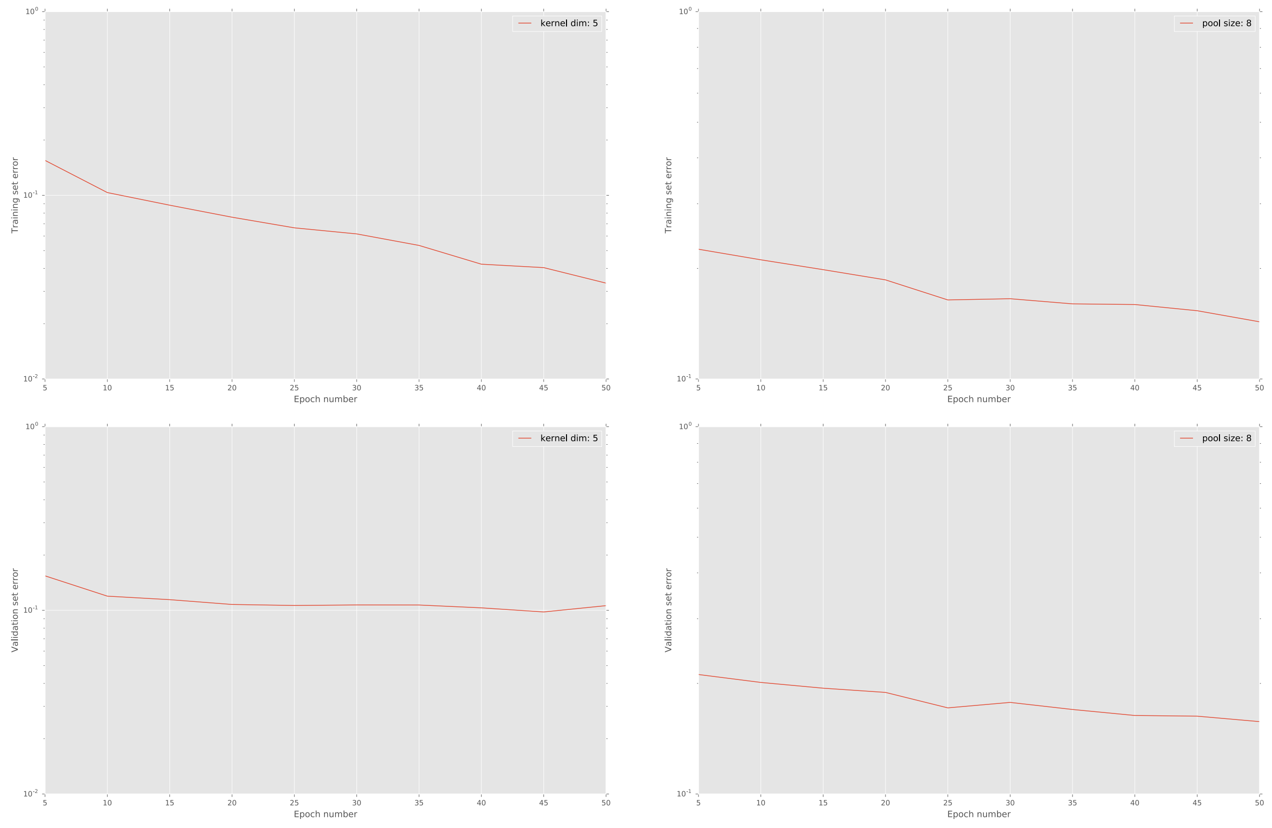

Figure 5

2. Maxpooling layers
group3:(model1, model4) Figure 6

3. Three affine transformations interleaved
group4:(model1, model5) Figure 7

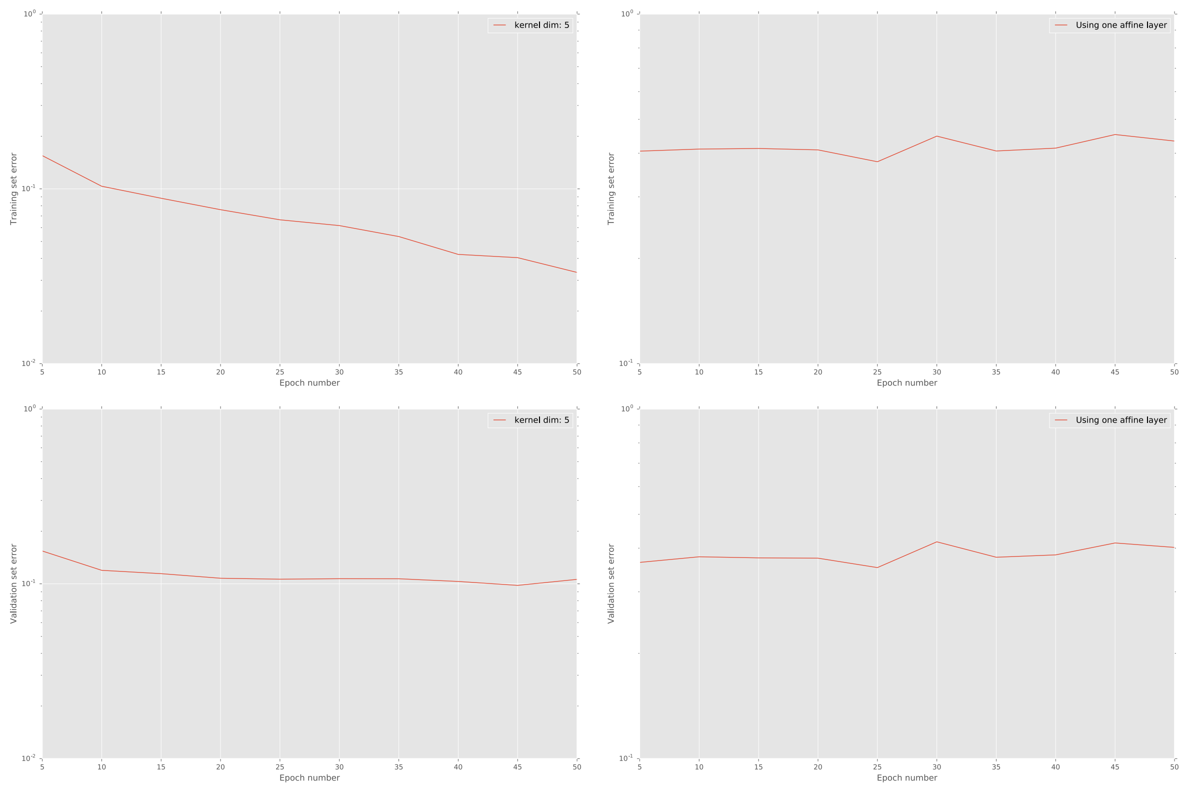

Figure 6

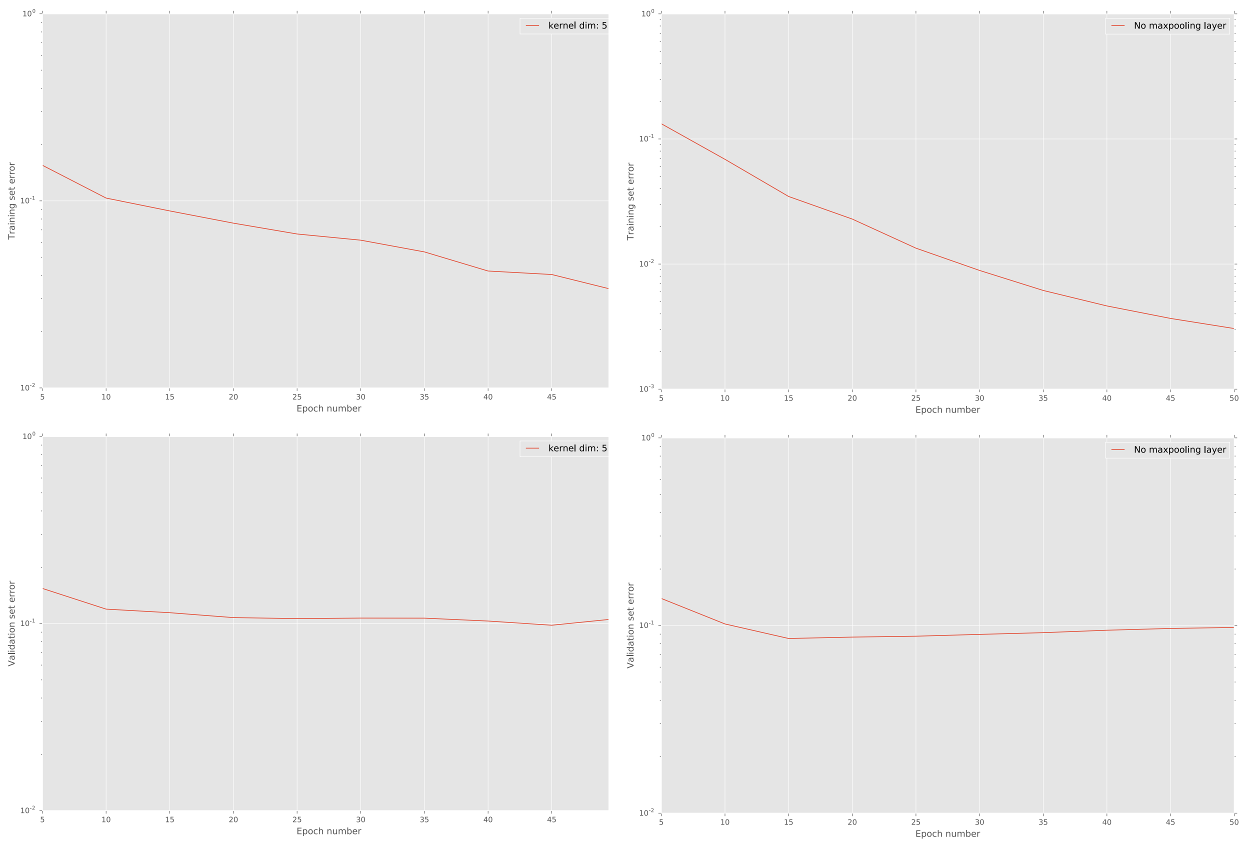

Figure 7

Discussion and conclusion:

1. From the group 1 we can find that it performs as good as when using 9 as value of kernel dim which use 5. But it is hard to say which one is better, using 9 is slightly better than using 5
2. From the group 2 we can find that it performs better when using 4 as value of pool size than 8.
3. From the group 3 we can find that it obviously performs better when using maxpooling layers than not using maxpooling layersthanSo.
4. From the group 3 we can find that it obviously performs better when using three affine transformations interleaved than just using one layer.
5. And I find that multiple layers neural networks with convolutional layers cost a huge number of time and computational complexity.

So we can find that when kernel dim is 9 and with maxpooling layers which size is 4 and with three affine layers give a positive affects to the result.