MLP Coursework 2

Name: Wenzhe Liu   Student Number: s1631854

Part 1

1. Topic: Does combining L1 and L2 regularization offer any advantage over using either individually?
2. A description of the methods used and algorithms implemented.

   1) L1 regularization:
      - **Formula:** $C = C_0 + \frac{\lambda}{n}\sum_w |w|$
      - **Description:**
        The first term is the usual expression for the cross-entropy. The second term is the sum of the absolute value of all the weights.
      - **Parameters:**
        - ➢ C_0: the original, un-regularized cost function.
        - ➢ λ: regularization parameter, where $\lambda > 0$.
        - ➢ n: the size of training set.
      - **Implementation:**
        a) Create a new class named L1Penalty.
        b) Define a function named __init__ to initialize the penalty coefficient.
        c) Define a function named __call__ to calculate L1 penalty value for a parameter as the formula shows.
        d) Define a function named grad to calculate the penalty gradient with respect to the parameter as the formula shows.
        e) Use the implementation to train the dataset.
   2) L2 regularization:
      - **Formula:** $C = C_0 + \frac{\lambda}{2n}\sum_w w^2$
      - **Description:**
        The first term is the usual expression for the cross-entropy. The second term is the sum of the squares of all the weights, which is scaled by a factor $\frac{\lambda}{2n}$.
      - **Parameters:**
        - ➢ C_0: the original, un-regularized cost function.
        - ➢ λ: regularization parameter, where $\lambda > 0$.
        - ➢ n: the size of training set.

- **Implementation:**
  a) Create a new class named L2Penalty.
  b) Define a function named __init__ to initialize the penalty coefficient.
  c) Define a function named __call__ to calculate L2 penalty value for a parameter as the formula shows.
  d) Define a function named grad to calculate the penalty gradient with respect to the parameter as the formula shows.
  e) Use the implementation to train the dataset.

3) L1 and L2 combination regularization:

- **Formula:** $C = C_0 + \frac{1}{2}(\frac{\lambda_1}{n}\sum_w |w| + \frac{\lambda_2}{2n}\sum_w w^2)$
- **Description:**
  The first term is the usual expression for the cross-entropy. The second term is the combination of L1 penalty and L2 penalty.
- **Parameters:**
  - C_0: the original, un-regularized cost function.
  - λ_1: L1 regularization parameter, where $\lambda_1 > 0$.
  - λ_2: L2 regularization parameter, where $\lambda_2 > 0$.
  - n : the size of training set.
- **Implementation:**
  a) Create a new class named L1L2Penalty.
  b) Define a function named __init__ to initialize the L1 penalty coefficient and L2 penalty coefficient separately.
  c) Define a function named __call__ to calculate L1L2 penalty value for a parameter.
  d) Define a function named grad to calculate the penalty gradient with respect to the parameter.
  e) Use the implementation to train the dataset.

3. A motivation for the experiment completed.

1) To explore the influences of L1 regularization, L2 regularization and L1 and L2 combination regularization on the model training.
2) To compare the performances between L1 regularization, L2 regularization and L1 and L2 combination.
3) To know the features of L1 regularization, L2 regularization and L1 and L2 combination.

4. Quantitative results for the experiments.

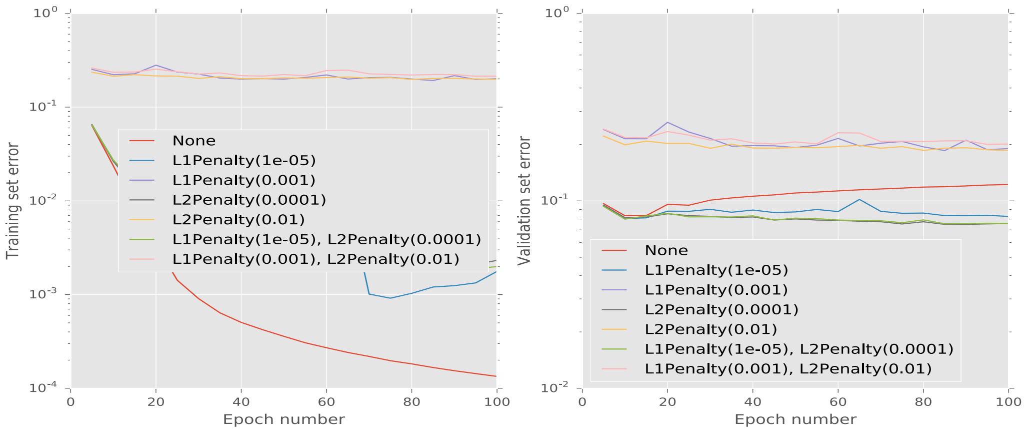
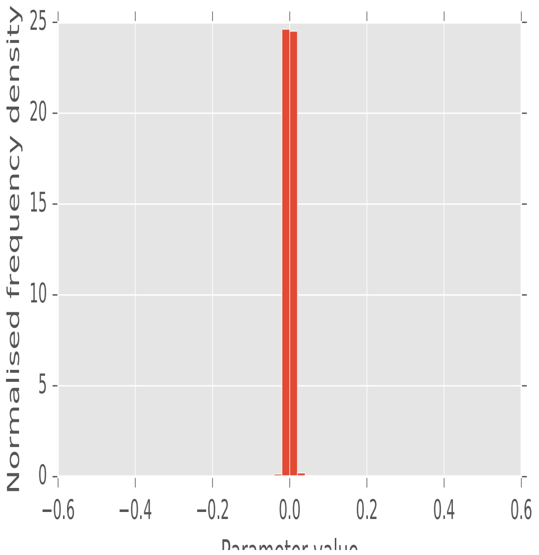


Figure 1-1 the error rate on both training set and validation set



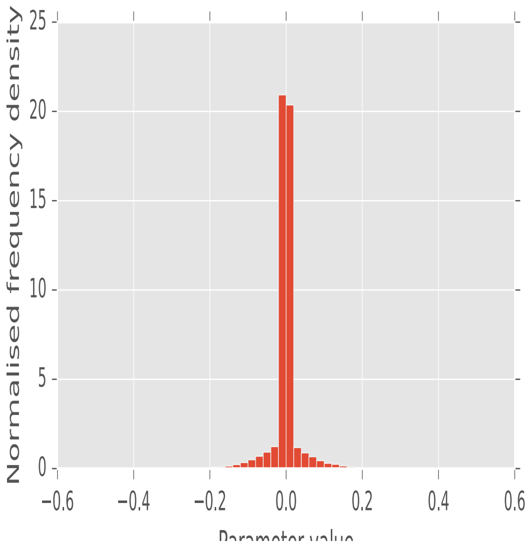
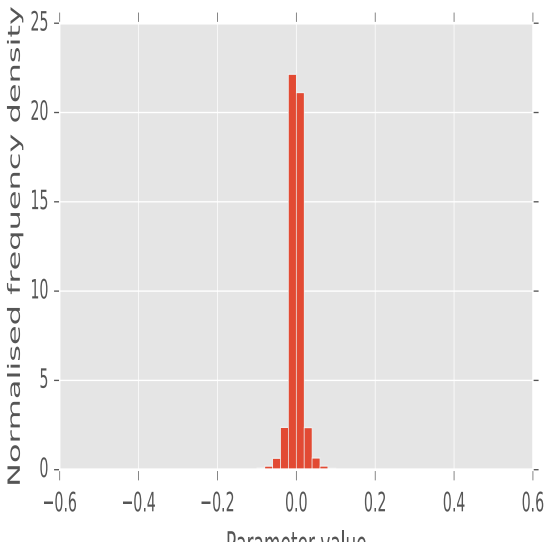


Figure 1-2 normalized frequency density on L1 penalty(10e-3) *Figure 1-3 normalized frequency density on L1 penalty(10e-5)*



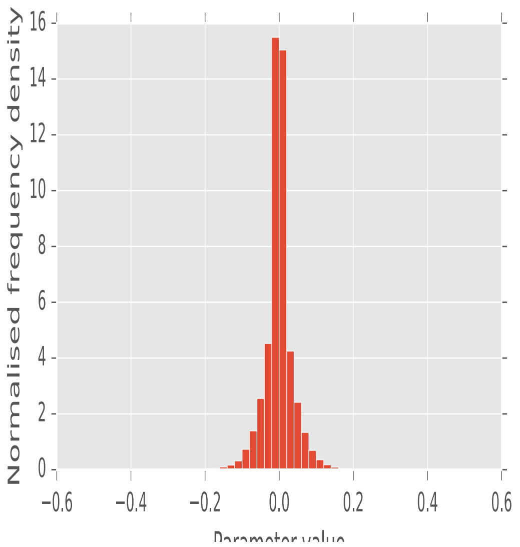


Figure 1-4 normalized frequency density on L2 penalty(10e-2) *Figure 1-5 normalized frequency density on L2 penalty(10e-4)*

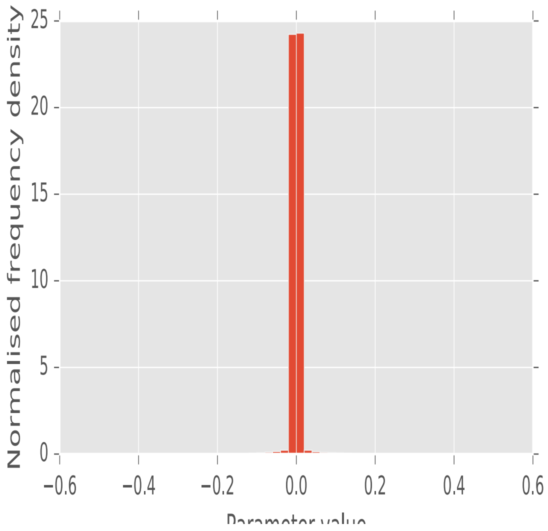


Figure 1-6 normalized frequency density on L1L2 penalty (L1=10e-3 L2 = 10e-2)

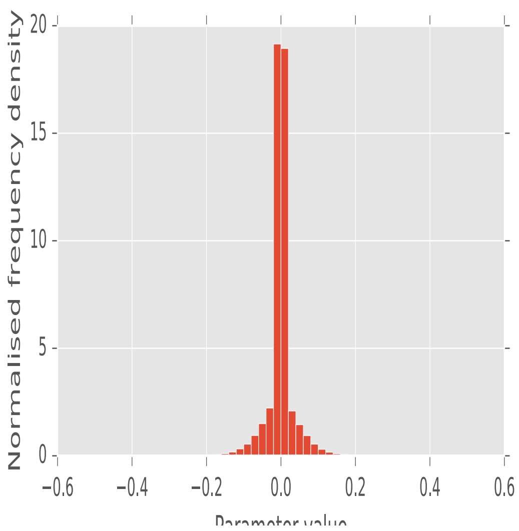


Figure 1-7 normalized frequency density on L1L2 penalty (L1=10e-5 L2 = 10e-4)

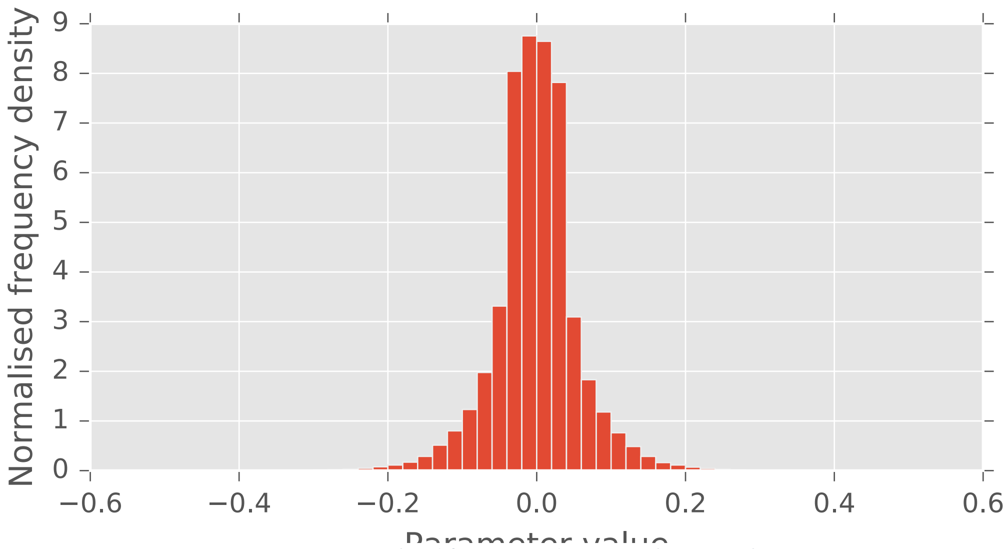


Figure 1-8 normalized frequency density without regularization

5. Discussion of the results of experiments and Conclusions.

   1) Comparing to the training set error and validation set error in figure 1-1, both L1 regularization and L2 regularization can considerably **reduce the effect of overfitting**, but L1 and L2 combination regularization can perform better. Because the training set error without regularization is near 0 at epoch 100, while in the validation set error figure, we can see the error goes down around epoch 10, but after that it actually starts to get worse. That is a sign that the model is overfitting. However, with the help of regularization, we find that the gap between training set error and validation set error decreases considerably and the validation set error becomes more stable, which shows the effect of overfitting is diminished.

Obviously, the gap of L1 and L2 combination regularization is the smallest, which means this regularization performs better than the other two.

2) Comparing to the training set error and validation set error in figure 1-1 and table 1-1, we can know both L1 regularization and L2 regularization can considerably **decrease the error rate**, but L1 and L2 combination regularization can perform better. Even the error rate on the training set looks better than others, the validation set result show the improvement is an illusion. From the table 1-1, we can conclude that the error rate falls from 1.22e-01 to 8.26e-02, 7.59e-02, 7.57e-02 separately comparing to the error rate without regularization. But still, the error rate of L1 and L2 combination is the lowest.

table 1-1 validation error on different conditions

	none	L1(10e-5)	L2(10e-4)	**L1(10e-)L2(10e-4)**
error	1.22e-01	8.26e-02	7.59e-02	7.57e-02

3) Comparing figure 1-2 to figure 1-8, we can know both L1 regularization and L2 regularization can **penalize large weights**, but L1 and L2 combination regularization can do it more reasonably. Firstly, comparing figure 1-3 with figure 1-8, we can find that the weights shrink by a constant amount toward 0 in L1 regularization. Secondly, comparing figure 1-5 with figure 1-8, the weights shrink by an amount which is proportional to w. So when a particular weight has a large magnitude, L1 shrinks the weight much less than L2 does, but when the weight is small, L1 shrinks the weight much more than L2 does. Comparing figure 1-7 with figure 1-8, L1 and L2 combination regularization find a balance between L1 regularization and L2 regularization. That is to make full use of both advantages so that no matter the weight is too large or too small, the penalty will not be too extreme.

Part 2

1. Topic: Data augmentation.
2. A description of the methods used and algorithms implemented.

   1) Random distortion:
      - **Description:**
        The method is use elastic deformations to make some “realistic” distortion. The image deformations are created by first generating random displacement fields, which are multiplied by a scaling factor α that controls the intensity of the deformation (shown in fig 2-3).
      - **Implementation:**
        a) Define a function named random_distortion.
        b) Use an array of shape (batch_size, 784) copied from the original inputs so that the inputs array cannot be modified.
        c) Use Gaussian_filter to generate random displacement fields call dx and dy.
        d) In every loop, add dx/dy with x and y to be new matrices. (x correlates to inputs.shape[1] and y correlates to inputs[0])
        e) Return transformed array.
   2) Random gaussian noise:
      - **Description:**
        The method is to add some noises to the image so that the color of the image background is changed randomly. Normally, the image background is pure white. After I add some noises, we can see images in different range of grayscales (shown in fig 2-4).
      - **Implementation:**
        a) Define a function named random_noise to add little noise to the image.
        b) Use an array of shape (batch_size, 784) copied from the original inputs so that the inputs array cannot be modified.
        c) Use np.random.normal to draw random samples from a gaussian distribution. So this is used to generate noises.
        d) Add the noise to the inputs in every loop.
        e) Return the changed array.
   3) Random Shift:
      - **Description:**
        The method is to change the position of the number in an image. Normally, the number lies in the center of an image. When I use shift method from the scipy we can see some numbers deviate from the center randomly (shown in fig 2-5).

- **Implementation:**
  a) Define a function named random_shift to change the position of the number in an image randomly.
  b) Use the rng.choice to generate indices randomly.
  c) Use scipy.shift function to shift the array using spline interpolation of the requested order.
  d) Return the shifted array.

3. A motivation for the experiment completed.

   1) To explore the performance of different data augmentation methods.
   2) To compare the influences of different data augmentation methods on the model training with the performance of the original dataset.
   3) To know the importance of tuning the parameters.

4. Quantitative results for the experiments.

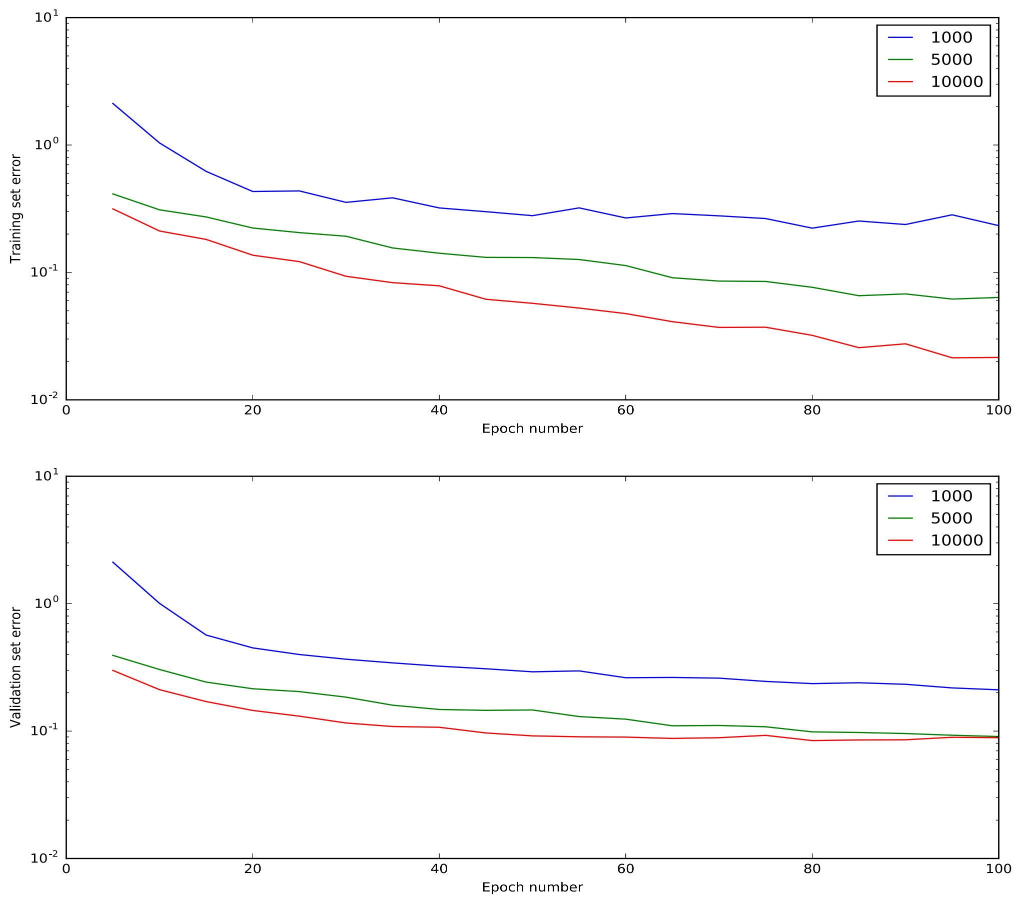


Figure 2-1 training/validation error based on different size databases

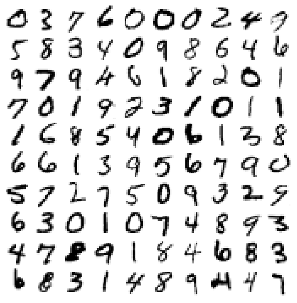


Figure 2-2 original image

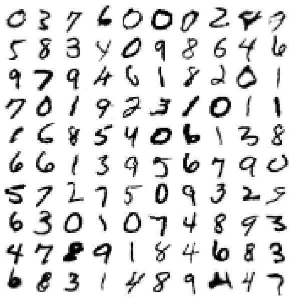


Figure 2-3 random distortion

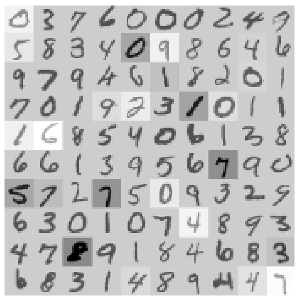


Figure 2-4 random gaussian noise

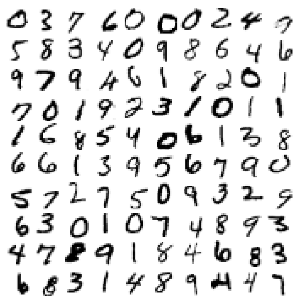


Figure 2-5 random shift

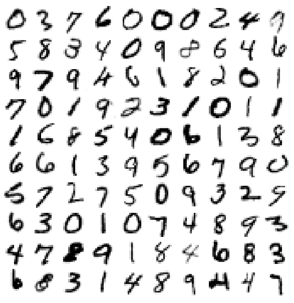
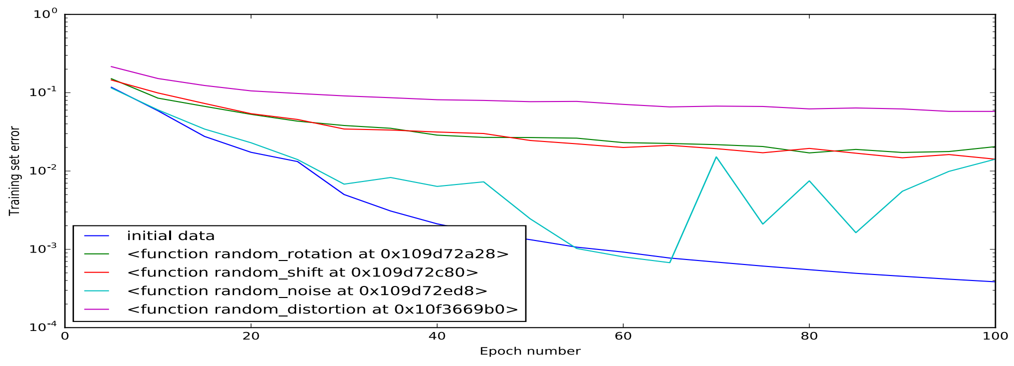


Figure 2-6 random rotation



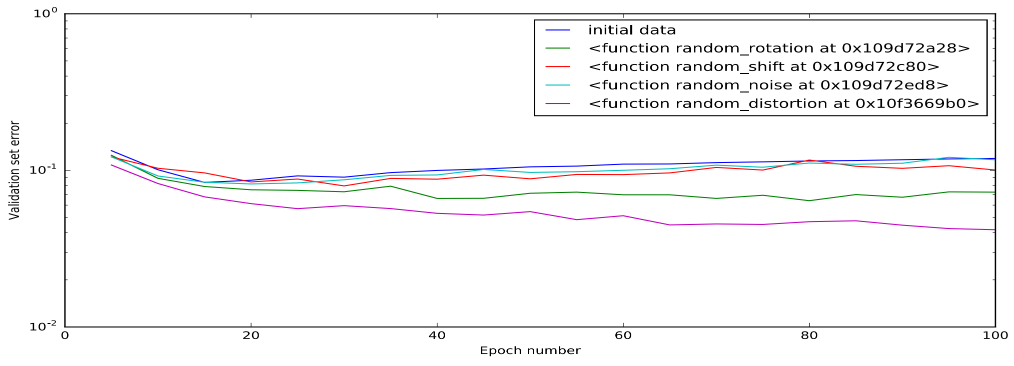


Figure 2-7 training/validation error based on different data augmentation method

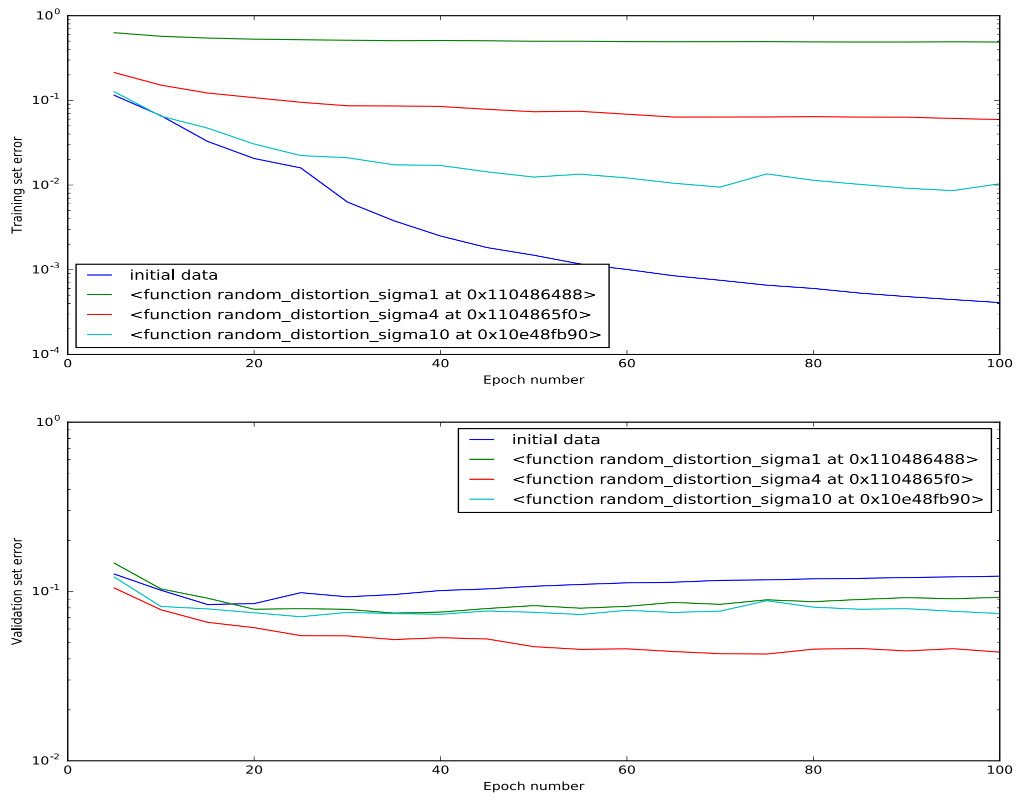


Figure 2-8 training/validation error based on different sigma values of random distortion

5. Discussion of the results of experiments and Conclusions.

    1) Comparing both training set and validation set error in figure 2-1 and table 2-1, it is easy to find that **the performance goes with the amount of training data**. Because when we use the 1000 data points to train the model, we get a bad result. But when we increase the amount of data points, the performance is getting better and better.

Table 2-1 error and accuracy based on different size datasets

	size = 1000		size = 5000		size=10000	
	training	validation	training	validation	training	validation
error	2.33e-01	2.11e-01	6.36e-02	9.06e-02	2.15e-02	8.88e-02
accuracy	9.25e-01	9.42e-01	9.81e-01	9.73e-01	9.94e-01	9.76e-01

2) Comparing both training set and validation set error in figure 2-7 and table 2-2, it can be concluded that **a good data augmentation method can reduce the effect of overfitting and decrease the error rate and increase the accuracy**. Because when using the original dataset, the gap between training error and validation error is very large and we can see the error rate of the validation set declines first and then goes up, which is a sign of overfitting. But when we use a good data augmentation to change the dataset to train, such as random distortion or random rotation, we can see the error rate on validation set reduces a lot and gap is decreased too.

Table 2-2 error and accuracy based on different data augmentation methods

	original dataset	random noise	random shift	random rotation	random distortion
validation error	1.04e-01	1.03e-01	9.76e-02	7.26e-02	4.18e-02
validation accuracy	9.80e-01	9.81e-01	9.80e-01	9.81e-01	9.88e-01

3) Comparing both training set and validation set error in figure 2-8 and table 2-3, it can be concluded that even use the same **data augmentation method, the choice of hyperparameters can affect the performance significantly.** For instance, when we use random distortion to do the data augmentation, if σ is large, the resulting values are very small because the random values average 0. If σ is small, the field looks like a completely random field after normalization. For intermediate σ values, the displacement fields look like elastic deformation, where σ is the elasticity coefficient. So there is no doubt that the dataset with intermediate ***σ*** value performs best.

Table 2-3 error and accuracy based on different σ values of random distortion

	$\sigma = 1$		$\sigma = 4$		$\sigma = 10$	
	training	validation	training	validation	training	validation
error	4.88-01	9.23e-02	5.93e-02	4.38e-02	1.03e-02	7.40e-02
accuracy	8.28e-01	9.79e-01	9.83e-01	9.87e-01	9.97e-01	9.83e-01

Part 3

1. Topic: Models with convolutional layers.
2. A description of the methods used and algorithms implemented.

   1) Forward propagation:
      - **Formula:** $\boldsymbol{y} = \boldsymbol{conv2d}(\boldsymbol{x}, \boldsymbol{K}) + \boldsymbol{b}$
      - **Parameters:**
        - x: Array of layer inputs.
        - K: Array of layer kernels.
        - b: Array of biases.
        - y: Array of layer outputs.
      - **Description:**
        The method takes a batch of activations at the inputs to the layer and forward propagates them to produce activates at the outputs.
      - **Implementation:**
        a) Define a function named fprop in the ConvolutionalLayer class, which forward propagates activations through the layer transformation.
        b) There are totally three loops in this function. The first loop is in the range of zero to batch size. The second loop is in the range of zero to output channels. The third loop is in the range of zero to input channels.
        c) In the innermost loop, use convolve2d to do the convolution between inputs and kernels.
        d) Append the outputs of the third loop in the second loop and append the outputs of the second loop in the first loop.
        e) Return the outputs of the first loop.

   2) Back propagation:
      - **Formula:** $\frac{\partial C}{\partial w_{a,b}^{l}} = \delta_{a,b}^{l} * \sigma(ROT180(z_{a,b}^{l-1}))$
      - **Parameters:**
        - $\delta_{a,b}^{l}$: Array of gradients with respect to the layer outputs.
        - $\sigma(ROT180(z_{a,b}^{l-1}))$: Array of kernels rotated by 180 degrees.
      - **Description:**
        The method calculates the gradient of a loss function with respect to all the weights in the network, so that the gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

- **Implementation:**
  a) Define a function named bprop in the CovolutionalLayer Class to back propagates gradients through a layer.
  b) There are totally three loops in this function. The first loop is in the range of zero to batch size. The second loop is in the range of zero to input channels. The third loop is in the range of zero to output channels.
  c) In the innermost loop, use convolve2d to do the convolution between grads_wrt_outputs and kernels. Importantly, we should rotation our kernels.
  d) Append the sum of outputs of the third loop in the second loop and append the outputs of the second loop in the first loop.
  e) Return the outputs of the first loop.

3) Grads_wrt_params:
- **Description:**
  The method is to calculate gradients with respect to layer parameters.
- **Implementation:**
  a) Define a function named grads_wrt_params.
  b) There are totally three loops in this function. The first loop is in the range of zero to batch size. The second loop is in the range of zero to input channels. The third loop is in the range of zero to output channels.
  c) In the innermost loop, use convolve2d to do the convolution between grads_wrt_outputs and inputs. Importantly, we should rotation our inputs.
  d) Store the sum of outputs of the third loop in the second loop called grads_wrt_kernels.
  e) In the first loop, store the sum of outputs called grads_wrt_biases.
  f) Return both grads_wrt_kernels and grads_wrt_biases.

3. A motivation for the experiment completed.
   1) To explore the performance of convolutional neural network comparing the neural network without convolutional layer.
   2) To test the influence of different size of local receptive fields.

4. Quantitative results for the experiments.

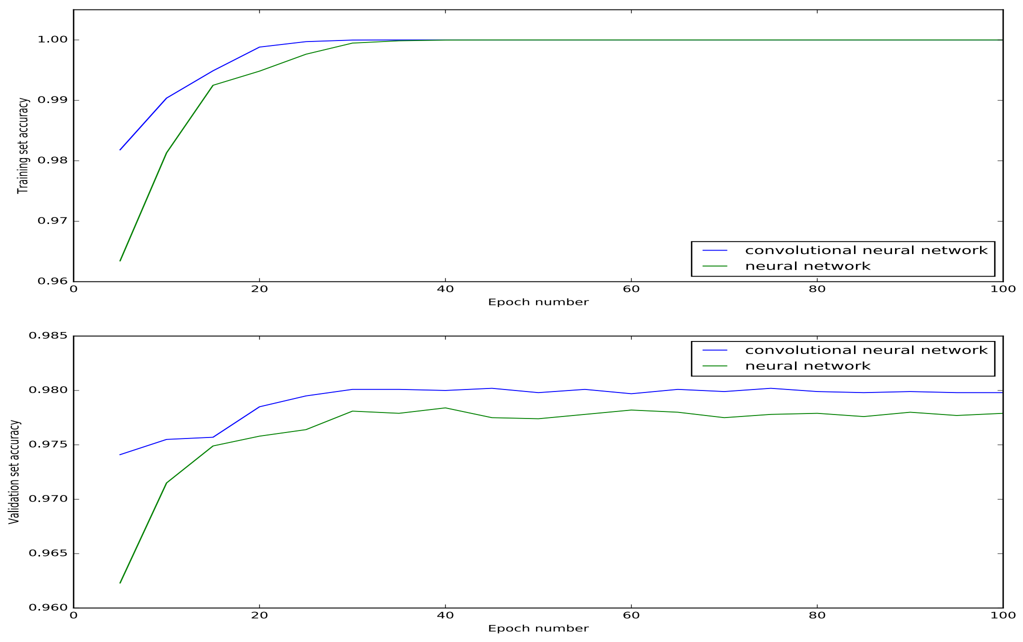


Figure 3-1 training/validation set accuracy of nn and cnn

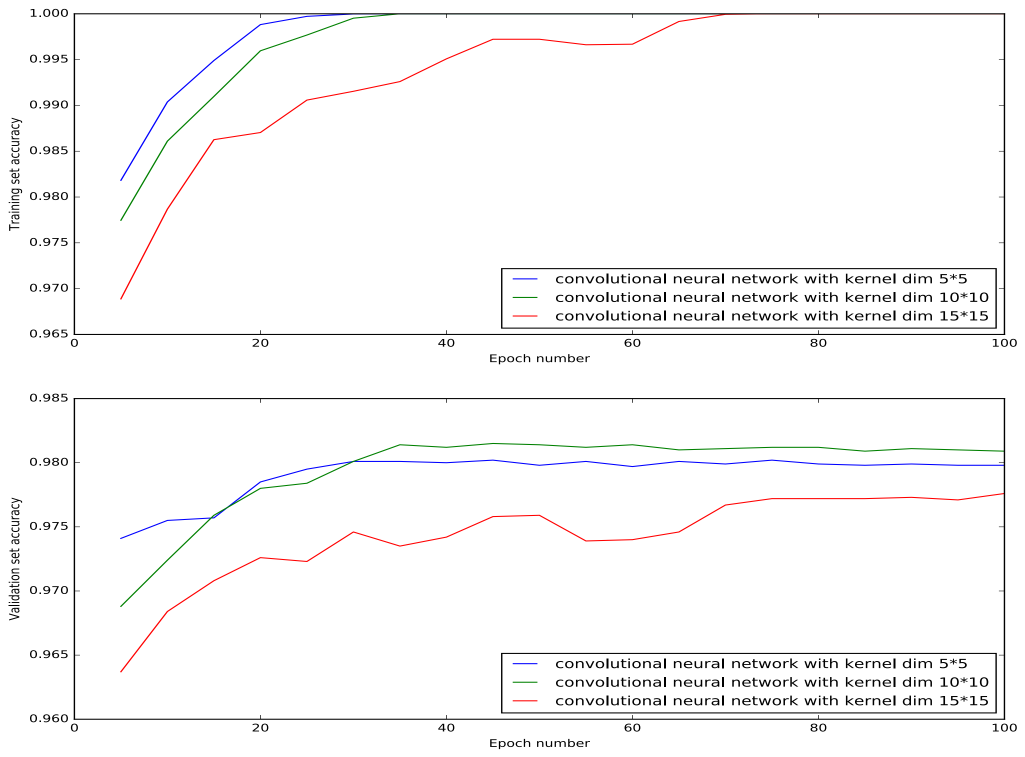


Figure 3-2 training/validation set accuracy based on different size of local receptive field

5. Discussion of the results of experiments and Conclusions.

1) Comparing both training set and validation set accuracy in figure 3-1 and table 3-1, it can be concluded that **convolutional neural network can speed up the speed of convergence and increase accuracy**. For example, in the accuracy of training set figure, we can see the convolutional neural network starts to converge at around epoch 20 while neural network without convolution starts to converge at epoch 35. And as for the accuracy, the accuracy of validation set improves from 97.8% to 98%. Even the improvement is very small, it still reduces 10% error rate ((98%-97.8%)/(1-98%)).

table 3-1 training/validation set accuracy of nn and cnn

	Neural network		Convolutional neural network	
	Train	Validation	Train	validation
Error	3.97e-04	1.20e-01	8.40e-05	1.40e-01
Accuracy	1.00e+00	9.78e-01	1.00e+00	9.80e-01

2) Comparing both training set and validation set accuracy in figure 3-2 and table 3-2, it can be concluded that **the choice of size of local receptive field can influence the final performance**. If the receptive fields are too small, a large number of neurons is needed to cover the input space. If the receptive fields are too large, it cannot extract enough information for the training. So a proper size of receptive field is significant.

table 3-2 training/validation set accuracy based on different size of local receptive

	Cnn kernel dim 10*10		Cnn kernel dim 5*5	
	Train	Validation	Train	validation
Error	9.80e-05	1.43e-01	8.40e-05	1.40e-01
Accuracy	1.00e+00	9.81e-01	1.00e+00	9.80e-01