Machine Learning Practical
Assignment 2

Siyuan Zhao

November 23, 2016

1 Combining L1 and L2 regularisation

1.1 Experimental hypothesis

In this section, An investigation was conducted to evaluate the method of combining L1 and L2 regularisation on overcoming the problem of overfitting. The method is also compared to L1 and L2 regularisation individually. The goal is to find whether combining L1 and L2 regularisation can offer any advantage over using either penalty individually.

1.2 Methodology

As L1 and L2, an complexity term E_w will be added to the error function.

$$E^n = E^n_{train} + E_w \tag{1}$$

To combine L1 and L2 regularisation, the term of E_w should included both the L1 term of $|w_i|$ and L2 term of $\frac{1}{2}w_i^2$. A linear relationship is applied here where α and β is the coefficient control the L1 and L2 term.

$$E_w = \alpha \sum_i |w_i| + \beta \frac{1}{2} \sum_i w_i^2 \tag{2}$$

This has a gradient with respect to the parameter vector

$$\frac{\partial E_w}{\partial w_i} = \alpha sgn(w_i) + \beta w_i \tag{3}$$

where $\text{sgn}(u) = +1$ if $u > 0$, $\text{sgn}(u) = -1$ if $u < 0$ (and is not well defined for $u = 0$ though a common convention is to have $\text{sgn}(0) = 0$). the total error with respect to the parametrs

1.3 Implementation and Results

In the python notebook file **CW2**, a class called **L1_L2_Penalty** is developed according to the definition of the method of new regularisation. The remaining part of the python notebook is about three plots. The first plot is the training and validation error against various values of coefficients for L1 term and L2 term. The second plot is the training and validation error against epoch number for all different regularisation schemes on the same axis. The third plot is about comparison between individual L1 and L2 regularisation, L1 and L2 combination regularisation and the model without regularisation.

coeff_L1	coeff_L2	error(train)	error(valid)	acc(train)	acc(valid)	params_penalty
1e-07	1e-06	1.53e-04	1.22e-01	1.00e+00	9.79e-01	7.05e-04
1e-06	1e-05	3.87e-04	1.03e-01	1.00e+00	9.80e-01	5.81e-03
1e-05	1e-04	4.39e-03	7.21e-02	1.00e+00	9.80e-01	3.38e-02
1e-04	1e-03	6.13e-02	8.31e-02	9.83e-01	9.76e-01	1.39e-01
1e-03	1e-02	3.37e-01	3.06e-01	9.09e-01	9.21e-01	4.13e-01

Table 1: Performance of different coefficients for combining method for L1 and L2

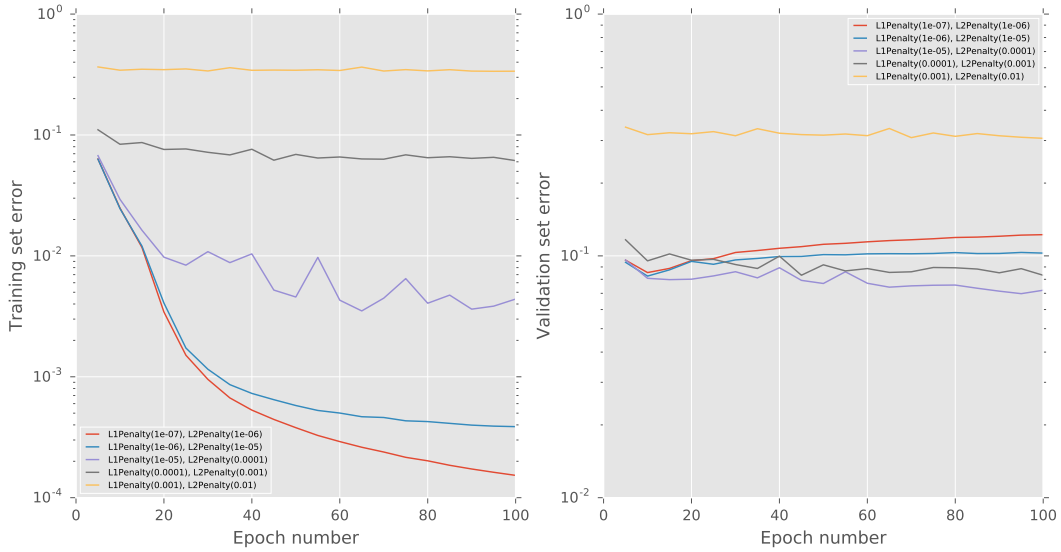


Figure 1: training error and validation error for L1 and L2 combination method

1.4 Discussion and Conclusion

From the Table 2, it can be shown that the smallest validation error is achieved when coefficient for L1 is $1e-05$ and coefficient for L2 is $1e-04$. The parameters penalty is neither too high nor too lower so that the model will not overfitting nor underfitting. For larger coefficients the parameters penalty is too large so that the speed of learning is very slow. However, For smaller coefficients, the validation error starts to increase with the decreasing of training error, which indicates a overfitting problem. On reason for that could be the relatively small value of coefficients on L1 and L2 term so that the effects of penalty is minor.

In conclusion, the combining method of L1 and L2 weights methods do provide better performance on validation error though the difference is small. In consideration to the computational cost, it is not worthwhile to use L1 and L2 regularisation.

coeff_L1	coeff_L2	error(train)	error(valid)	acc(train)	acc(valid)	params_penalty
None	None	1.34e-04	1.22e-01	1.00e+00	9.79e-01	0
1e-05	None	1.64e-03	8.29e-02	1.00e+00	9.80e-01	1.98e-02
None	1e-04	2.32e-03	7.57e-02	1.00e+00	9.81e-01	1.56e-02
1e-05	1e-04	4.39e-03	7.21e-02	1.00e+00	9.80e-01	3.38e-02

Table 2: Comparison between combined method and L1, L2 methods

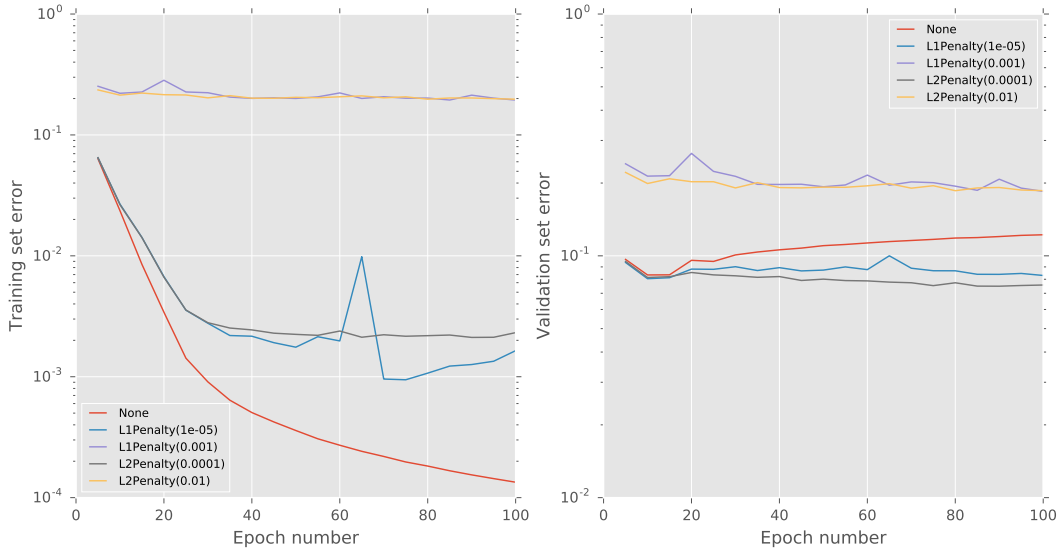


Figure 2: training error and validation error for L1 and L2 penaly method

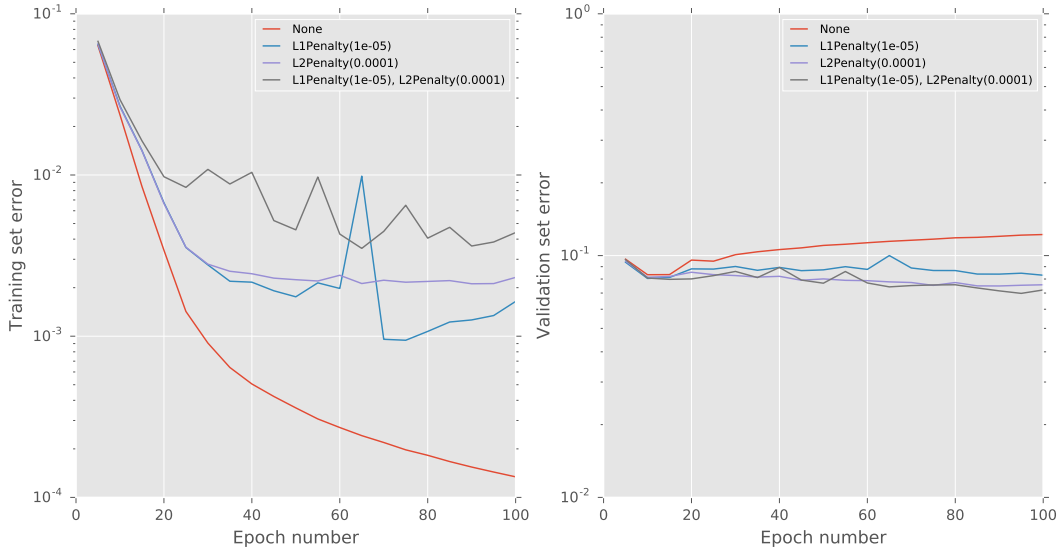


Figure 3: Comparison between combining method, L1 and L2 penalty method

2 Data Augmentation

2.1 Experimental hypothesis

Several ways was investigated to expand the training data such as rotating, zooming of existing data and elastic deformation [2]. In this experiment, we are going to find wether additional inputs generated from training data could provide better performance on validation set and how well do different ways of distortion perform on validation sets.

2.2 Methodology

In this section, the methodology of two ways of distortion is introduced.

2.2.1 Random Zoom

one quarter of training data will be randomly chosen to be zoomed by a factor of ± 0.2. Then, theses distorted data will be added to the training set. A function named **random_zoom** was developed. First, the indices of images to be zoomed was selected. Then, for each images, the zoom factor was chosen using uniform distribution in the range from 1.2 to 0.8. Then, if it's zoomed in, the dimension of the zoomed images will be larger than original dimension of 28×28. Therefore, the centeral 28×28 of pixels of zoomed images will be chosen to be the new images for training. Otherwise, the dimension of the zoomed images will be smaller than original dimension of 28×28. Therefore, the zoomed images will be added to the central of the new images to be training. Therefore, the dimensional size of the new image will be unchanged during zooming. Finally, the images was reshaped to the standard size for training.

2.2.2 Elastic Distortion

In the case of handwriting recognition, [2] provides a way to provide similar uncontrolled oscillation of the hand muscles. A function named **elastic_deformation** was developed. Firstly, similar to **random_zoom**, a quarter of training data will be randomly selected for elastic deformation. It is generated first by random displacements fields. $\Delta x(x, y) = rand(-1, +1)$ and $\Delta y(x, y) = rand(-1, +1)$, where $rand(-1, +1)$ is a random number uniformly distributed between -1 and $+1$. Then, Δx and Δy is convolved with a Gaussian distribution with standard deviation σ and then scaled by a factor of α.

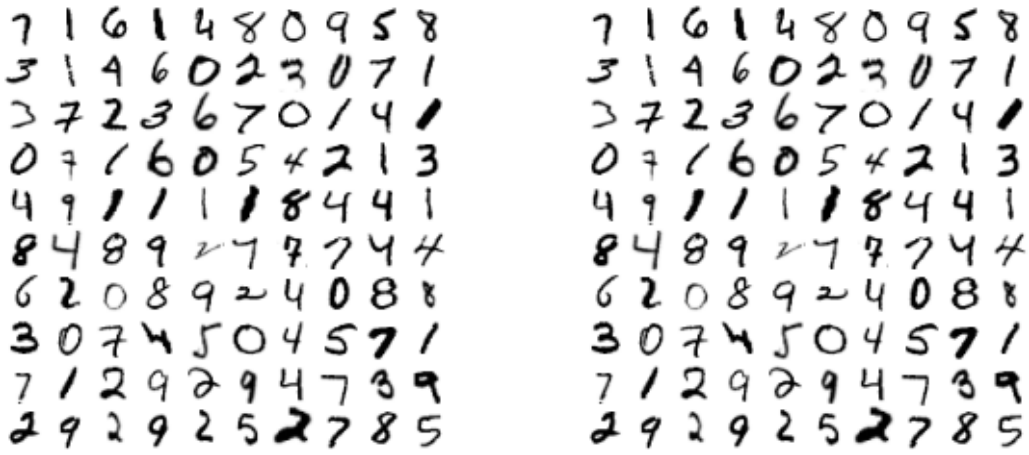
2.3 Implementation and Results

The final result for different type of augmentation was obtained by training each of them individually at first. Then, the results was shown by combining all the stats. The Table 3 shows the results of error and accuracy for both training and validation sets.

Type	error(train)	error(valid)	acc(train)	acc(valid)
None	1.34e-04	1.22e-01	1.00e+00	9.79e-01
rotation	2.21e-02	8.22e-02	9.93e-01	9.80e-01
Zoom	8.00e-03	8.45e-02	9.98e+00	9.81e-01
Elastic	5.71e-02	4.50e-02	9.83e+00	9.87e-01

Table 3: Performance of different ways of data augmentation

In the Figure 4, we can find that some of numbers is zoomed in and some of numbers is zoomed out. In the Figure 5, we can find the impact of elastic deformation with different value of σ. The α is set to 34 according to the authors [2]. The best σ is chosen to be 4 for training. The final training and validation results is shown in Figure.



(a) Original (b) Zoomed

Figure 4: Batch of images for zooming

2.4 Discussion and Conclusion

From the results above, it can be seen that by applying the method of data augmentation, the overfitting is reduced since the training error is increased while the validation error is decreased. From the three ways of data augmentation, the elastic deformation has the best performance. It not only avoid overfitting the training data, but also fitting well on validation sets.

The reason that the elastic deformation has the best performance is that it is a probably more realistic distortion compared to other methods. That means the additional data is more similar to the validation data. The more similar the artificial created data is, the better result the model will be.

In conclusion, we found that the additional augmented data can provide better performance. Elastic deformation has the best performance of all.

(a) Original

(b) $\sigma = 1$

(c) $\sigma = 2$

(d) $\sigma = 3$

(e) $\sigma = 4$

(f) $\sigma = 5$

Figure 5: Batch of images for elastic deformation

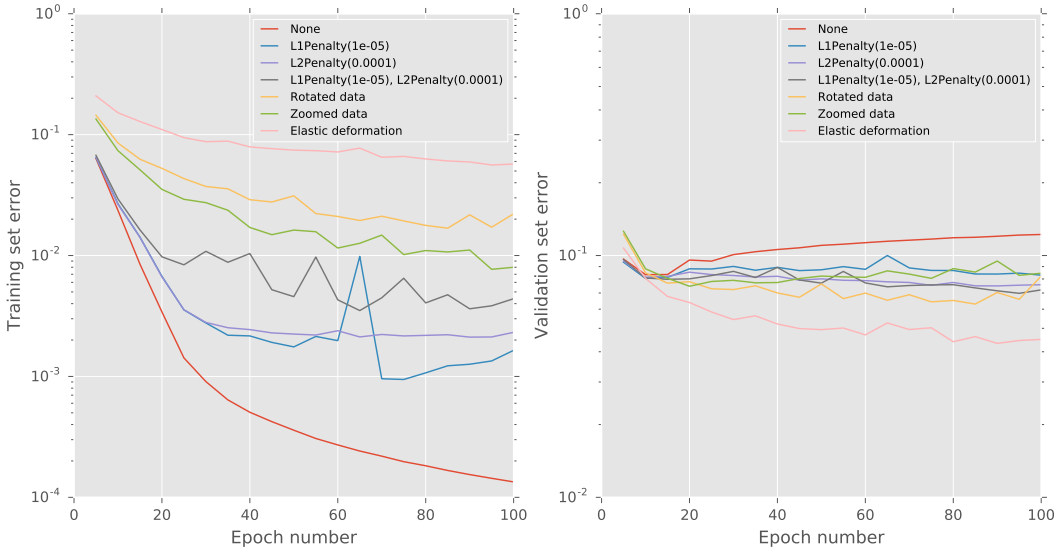


Figure 6: training error and validation error for data augmentation

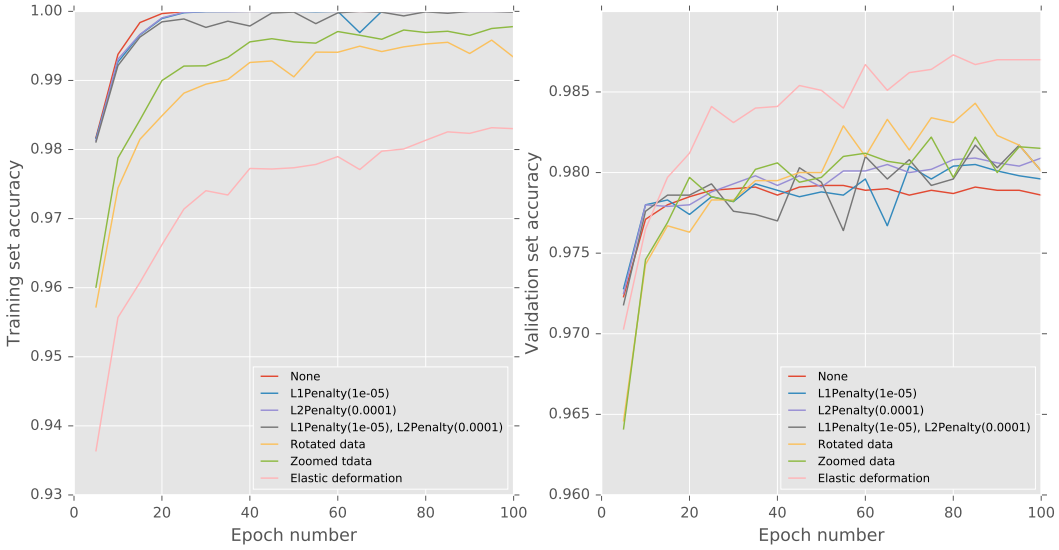


Figure 7: training accuracy and validation accuracy for data augmentation

3 Convolution layer

3.1 Experimental hypothesis

In this section, a convolution layer and a max-pooling layer was developed. The hypothesis is to evaluate the impact of the max-pooling layer on the convolutional neural network (CNN).

3.2 Methodology

3.2.1 Convolution layer

The inputs Assuming the inputs was a 4D tensor (batch size, number of input channels, input dimension 1, input dimension 2), the output for each unit in the next hidden layer is calculated by:

$$h_{i,j} = \Theta(\sum_{k=0}^{m-1}\sum_{l=0}^{m-1} w_{k,l}x_{i+k,j+l} + b) \tag{4}$$

where i, j is the indices for the row and column, Θ is the activation function such as the *ReLU*, *Sigmoid* or the *hyperbolic tangent* function, $w_{k,l}$ are elements of the shared $m \times m$ weight matrix $\mathbf{w}$.

For the back propagation, the gradient with respect to the inputs is:

$$\delta_n^{(x)} = \frac{\partial J}{\partial x_n} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x_n} = \sum_{i=0}^{m}\frac{\partial J}{\partial y_{n-i}}\frac{\partial y_{n-i}}{\partial x_n} = \sum_{i=0}^{m}\delta_{n-i}^{(y)}w_i = \Big(\delta^{(y)} * flip(w_i)\Big)[n] \tag{5}$$

the gradient with respect to the kernels is:

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_i} = \sum_{j=0}^{d-m+1}\frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial w_i} = \sum_{j=0}^{d-m+1}\delta_j^{(y)}x_{j+i} = \Big(\delta^{(y)} * flip(x)\Big)[i] \tag{6}$$

where d is the input dimension.

3.2.2 Pooling Layer

The max-pooling layer is investigated in this section. It subsamples a feature map and reduce its size. In this experiment, it reduces a set of 2×2 regions to a single unit. The way to reduce is max-out which means the maximum among the 4 units becomes the output of the pooling layer. In forward propagation, the output of the pooling layer is:

$$y(x) = max(x) \tag{7}$$

for back propagation,

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1 & \text{if } x_i = \max(x) \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial x} = \delta^{(y)}\frac{\partial y}{\partial x} \tag{8}$$

3.3 Implementation and Results

3.3.1 Convolution layer

For the convolution layer, the class **ConvolutionalLayer** was developed. The method **fprop** was developed for the forward propagation. The inputs image is 2-dimensions (x, y). The input of the method is variable **inputs**, a 4-dimensional tensor (batch size, number of input channels, x, y). The output of the method is variable **outputs**, a 4-dimensional tensor (batch size, number of output channels, x - m + 1, y - m +1), where m is the the 2-dimensional kernel ($m \times m$) size, with no-padding and unit strides.

For the method of **bprop**, it is used to calculate the gradients with respect to the inputs. The input arguments are **inputs**, **outputs** and a 4-dimensional tensor **grad_wrt_outputs** (batch size, number of output channels, x - m +1, y - m + 1). The return value of the method is **grad_wrt_inputs** which is an 4-D tensor (batch size, number of input channels, x, y).

For the method of **grads_wrt_params**, the inputs arguments are **inputs** and **grad_wrt_outputs**. The output is a list of arrays of gradients with respect to the layer parameters. The list contains **grads_wrt_kernels** contains **grads_wrt_biases**.

3.3.2 Pooling Layer

For the max-pooling layer, the class **MaxPoolingLayer** was developed. It has internal arguments **pool_size_1** and **pool_size_2**. In this implementation, the default value for both of them is 2.

For the method of **fprop**, the inputs of the method is **inputs**, which is a 4-dimensional tensor (batch size, number of channels, x, y). the output tensor is also 4-dimensional but the size of each size of the output image is halved. since both of the pool size is 2.

For the method of **bprop**, the inputs of the method are **inputs**, **outputs** and **grads_wrt_outputs**. The return tensor is **grads_wrt_inputs**.

3.3.3 Results

The code is implemented in the file **CW2_Q3.ipynb**. The total number of epochs is limited to 30 due to high time-consumption of training. The baseline was a model with two affine layer using *ReLu* as the activation function. The plots of error and accuracy on training and validation sets was shown in Figure 8 and Figure 9. The plot of error is log-scaled. The Table 4 shows the value of the performance for every model.

Type	error(train)	error(valid)	acc(train)	acc(valid)
Baseline	8.57e-04	1.02e-01	1.00e+00	9.80e-01
cnn_only	1.28e-04	1.29e-01	1.00e+00	9.80e-01
cnn_pooling	2.67e-04	8.91e-02	1.00e+00	9.84e-01

Table 4: Performance on experiment of convolution layer

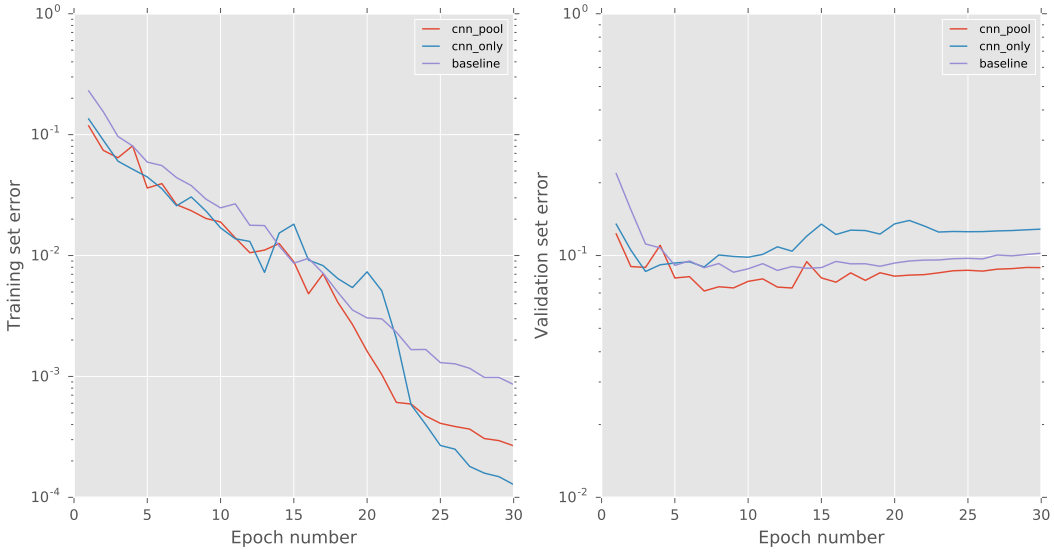


Figure 8: training error and validation error for CNN

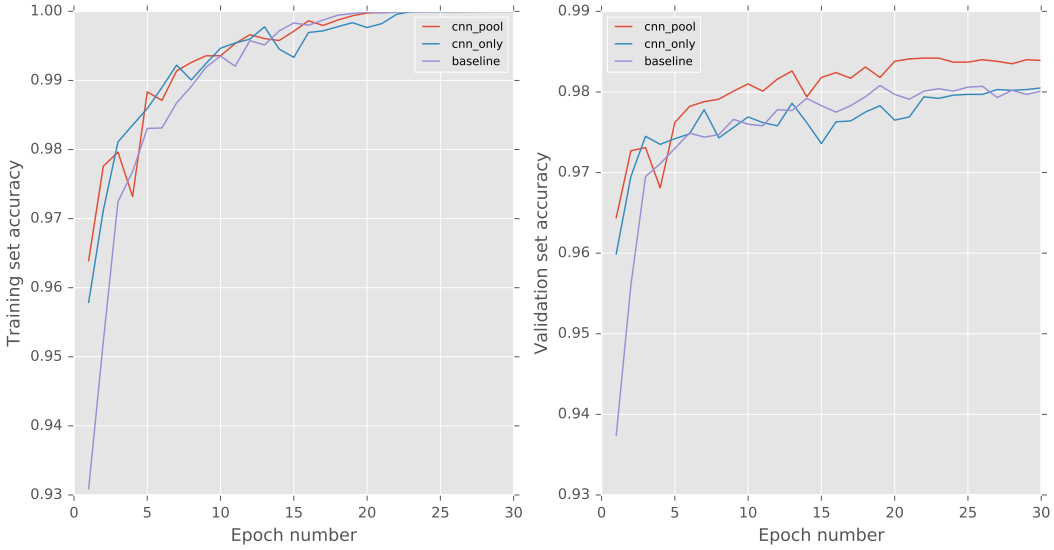


Figure 9: training accuracy and validation accuracy for CNN

3.4 Discussion and Conclusion

The plot illustrates that the CNN with pooling layer has higher accuracy and lower error on the validation sets, while it has lower error in training sets. Therefore, it reduces the overfitting.

Since max pooling operation replace the outputs of convolution layer at a certain location with a summary statistic of pixels nearby, it is helpful to make the representation become approximately invariant to small translation of the input [1].

In the MNIST classification problem, we care more about whether some feature is present rather than where it is. The use of pooling layer can be assumed to add an infinity strong prior that the feature learned by the neural network must be invariant to small translations. It can improve the efficiency of the network when this assumption is correct [1].

Another benefits of pooling layer is that it has fewer pooled features which helps to reduce the number of kernel parameters in later layers.

In conclusion, a max-pooling layer can improve the performance of a convolutional neural network in both accuracy, error but also in reducing overfitting.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". Book in preparation for MIT Press. 2016. URL: http://www.deeplearningbook.org.

[2] P. Y. Simard, D. Steinkraus, and J. C. Platt. "Best practices for convolutional neural networks applied to visual document analysis". In: *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*. Aug. 2003, pp. 958–963. DOI: 10.1109/ICDAR.2003.1227801.