Coursework 3

Wenzhe Liu

February 16, 2017

1 Activation functions

1.1 Research questions

Every activation function has own advantages and disadvantages. So facing with different data type and test parameters, the performance of each activation function is uncertain. The aim of this experiment is to find out which activation function is suitable in this case.

1.2 Methods

1.2.1 Introduction of the methods

In neural networks, the activation function of a node defines the output of that node given an input or set of inputs.

1. Relu active function:
   Rectifier is an activation function defined as $f(x) = max(0, x)$, where x is the input to a neuron.

2. Sigmoid active function: A sigmoid function is a mathematical function having an "S" shaped curve (sigmoid curve). It can be defined as $S(x) = \frac{1}{1+e^{-}x}$.

3. Tanh active function: Tanh active function can be defined as $f(x) = tanh(x) = \frac{2}{1+e^{-}2x} - 1$.

4. Softsign active function:Computes softsign Tanh active function can be defined as $f(x) = \frac{x}{1+|x|}$.

5. Elu active function: Elu active function can be defined as:

$$h(\alpha, x) = \begin{cases} \alpha * (e^x - 1) & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$$

1.2.2 Outline of implementation in TensorFLow

1. Find the activation function module in TensorFLow. The activation ops provide different types of nonlinearities for use in neural networks.

2. Find the location to use the activation function. The activation function is used in the output of each layer. So it should be implemented in the full connected layer.

3. For example, if we want to use ReLU as the activation function, in tensorfolow, we can define the fully connected layer as below:

```
def fully_connected_layer(inputs, input_dim, output_dim,
    nonlinearity=tf.nn.elu):
```

1.3 Result and discussion

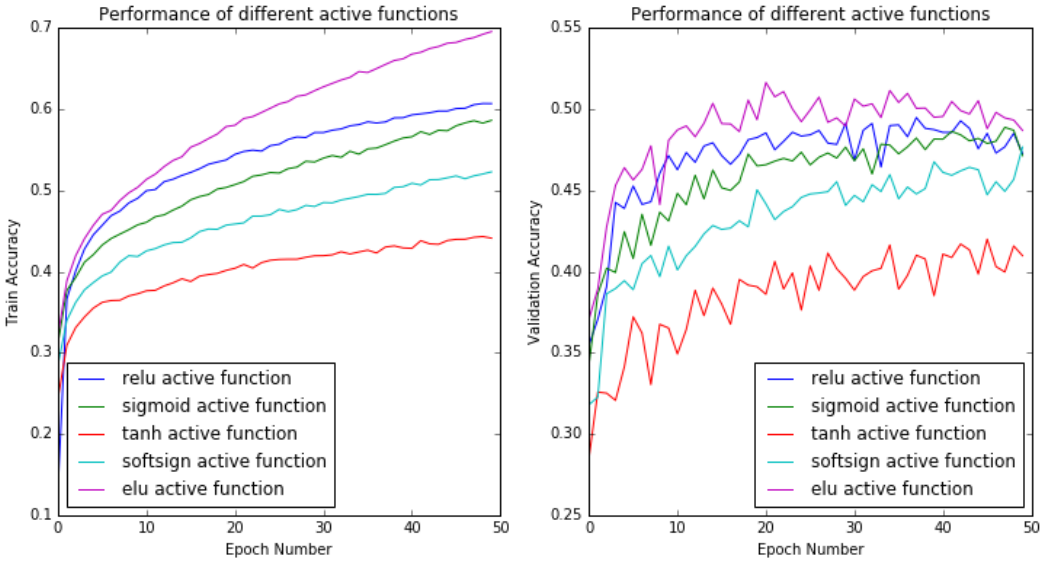

Figure 1: Performance of different activation function on CIFAR10

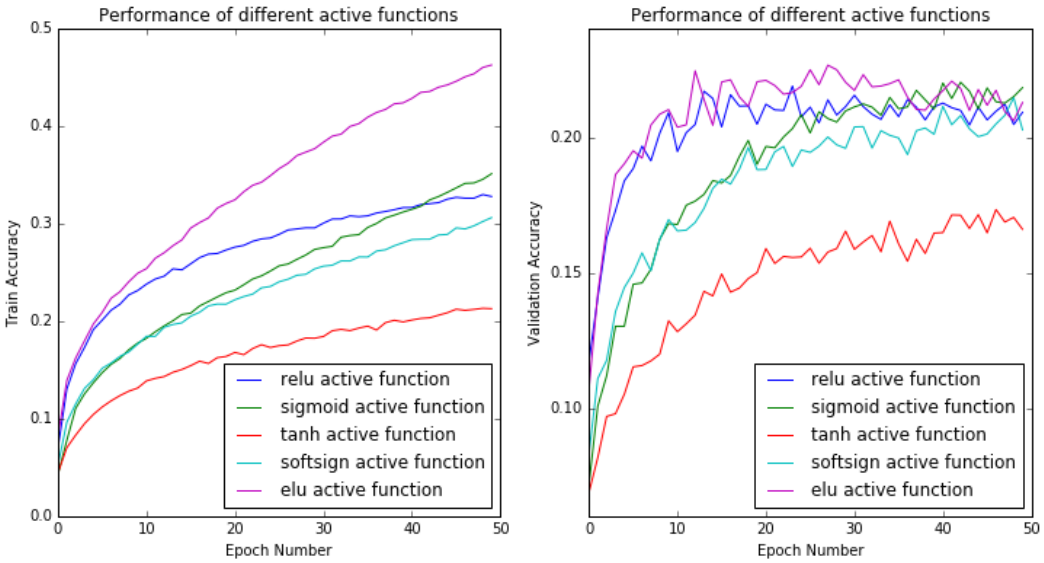

Figure 2: Performance of different activation function on CIFAR100

Table 1: The performance of different activation functions in CIFAR-10

	train_error	valid_error	train_accuracy	valid_accuracy
Relu	1.09	1.65	0.61	0.47
Sigmoid	1.15	1.54	0.59	0.47
Tanh	1.56	0.44	1.66	0.41
Softsign	1.33	1.49	0.52	0.48
Elu	0.84	1.87	0.70	0.49

Table 2: The performance of different activation functions in CIFAR-100

	train_error	valid_error	train_accuracy	valid_accuracy
Relu	2.61	3.72	0.33	0.21
Sigmoid	2.54	3.36	0.35	0.22
Tanh	3.23	3.58	0.21	0.17
Softsign	2.77	3.41	0.31	0.20
Elu	2.04	4.37	0.46	0.21

According to Figure 1 generated by the same epoch numbers and the same hidden layers, the elu activation function resulted in the most successful one for all the test parameters. Tanh function was not able to successfully use the data type and the network parameters. Besides, almost all the active functions result in overfitting since there is a big gap between training accuracy and validation accuracy. In addition, the convergence speed of Elu function is faster than sigmoid, tanh and softsign. From the Table 1, we can see the final accuray is almost the same except Tanh activation function. The trend on CIFAR100 is similar to the CIFAR10. The final accuracy of the algorithms was observed to be almost the same for the elu, relu, softsign, sigmoid functions. Elu and relu converge earlier than the other two. Because CIFAR100 is more complicated than CIFAR10, so the performance of CIFARR100 is worse than CIFAR10. In conclusion, from the experiment result, elu activation function is more suitable for this data type and test parameters.

2 Hidden layer depths and widths

2.1 Research questions

With the increase of hidden layer depth and width, the performance of the neural network is uncertain. So a lot of experiments need to be carried out to find the optimal depth and width for a specific database.

2.2 Methods

1. Introduction of the methods
   In feed-forward neural network, hidden layer can be defined as the layers except input layer and output layer. Hidden layer does not receive signals from the outside and also does not transmit signals to the outside. Hidden layer depths refer to the number of layers between input layer and output layer. Hidden layer width refers to the number of neurons in a hidden layer.

2. Outline of implementation in TensorFLow
   In the tensorflow, you need to define a fully connected layer which is the prototype of input layer, hidden layer and output layer. A fully connected layer function need to get input, input dimension, output dimension and activation function from the outside so that it can compute weight and bias for each layer and use the activation function to output the result.

   (a) Input layer: in this layer, train input should be given to fully connected layer function and input dimension is the size of the train input. The output dimension is the input dimension of the hidden layer.

   (b) Hidden layer: in this layer, the output of the input layer should be as the input of hidden layer and the output dimension is the width of hidden layer. If different hidden layer width needed to change, just use different values for the output dimension. If more hidden layers need to be added, just iterate the above step, but the only stuff need to change is to use the output of the previous hidden layer as the input of the next hidden layer.

   (c) Output layer: in this layer, the output of the final hidden layer should be as the input of output layer. Slightly different, the output dimension of the output layer should the number of the target.

2.3 Result and discussion

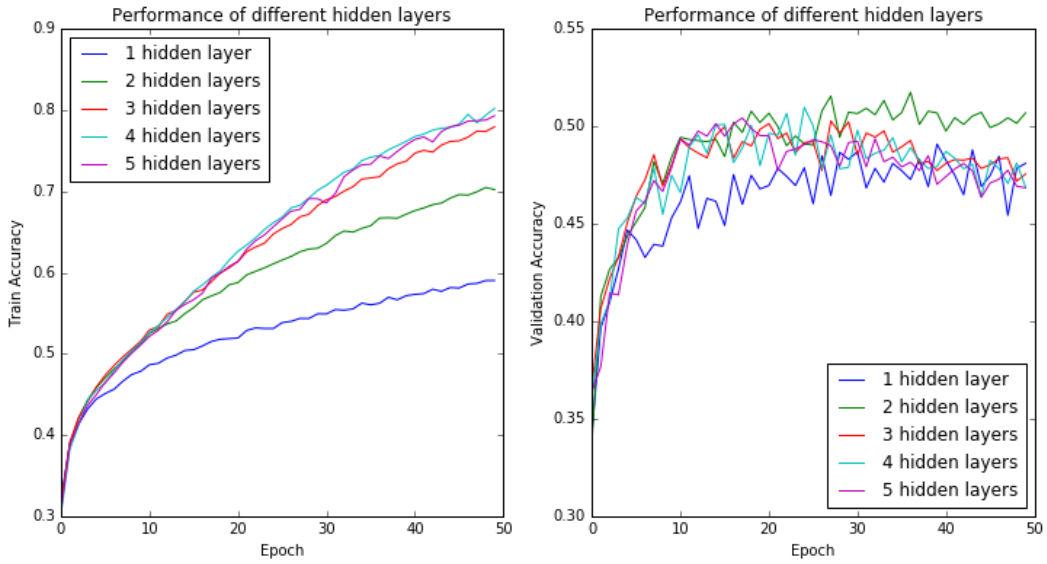

Figure 3: Performance of different hidden layers on CIFAR10

According to Figure 3 generated by the same epoch numbers and the elu activation function, 2 hidden layers resulted in the most successful one for all the test parameters. From the table 3, we can know the performance of the rest were around 0.48. Besides, the overfitting of 2 hidden layers is not as worse as the others. But the convergence speed is almost the same. In addition, with the increase of hidden layers, the computing time is getting longer. The performance of 2 hidden layers is best because if there are too many layers, it easily causes overfitting and if there are few hidden layers, it easily causes underfitting. The trend on CIFAR100 (Figure 4) is similar to the CIFAR10.

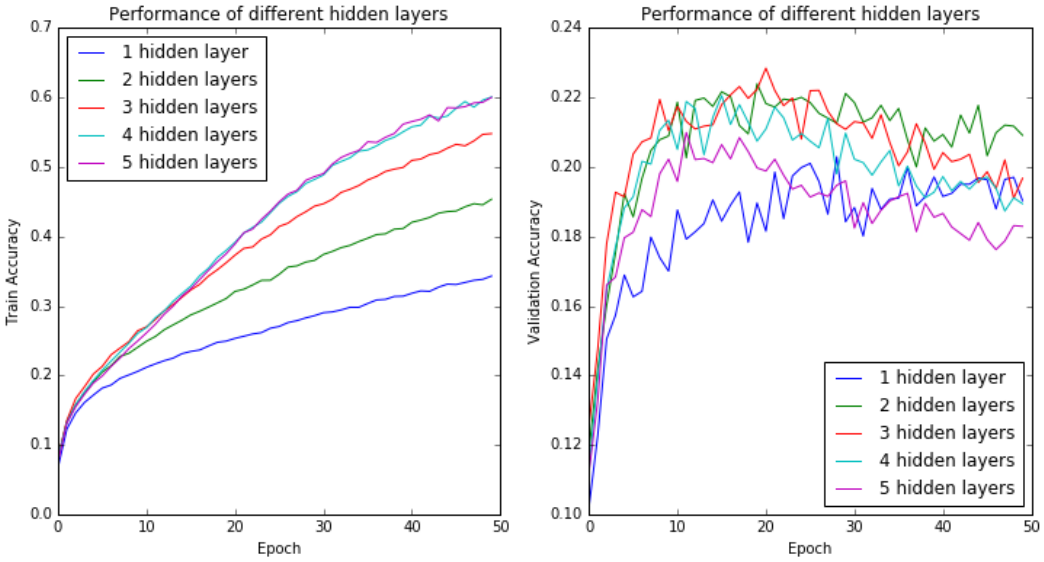

Figure 4: Performance of different hidden layers on CIFAR100

Table 3: The performance of different hidden layers in CIFAR10

	train_error	valid_error	train_accuracy	valid_accuracy
1 layer	1.15	1.59	0.59	0.48
2 layers	0.82	1.87	0.70	0.51
3 layers	0.60	2.61	0.78	0.48
4 layers	0.55	2.74	0.80	0.47
5 layers	0.57	2.49	0.79	0.47

Table 4: The performance of different hidden layers in CIFAR100

	train_error	valid_error	train_accuracy	valid_accuracy
1 layer	2.65	3.88	0.34	0.19
2 layers	2.08	4.32	0.45	0.21
3 layers	1.62	5.32	0.55	0.20
4 layers	1.41	6.15	0.60	0.19
5 layers	1.39	6.50	0.60	0.18

The accuracy of the algorithms was also observed to increase most for 2 hidden layers. According to Table 4, the rest performance is almost the same. In a word, 2 hidden layers are more suitable for both CIFAR 10 and CIFAR 100.

According to Figure 5 generated by the same epoch numbers and the elu activation function, there is not a lot of difference between different hidden units. No matter the convergence speed, the accuracy or the error rate, they are almost the same. So we cannot tell which is better. The trend on CIFAR100(Figure 6) is quite similar with CIFAR10. Because all the performance in this group is bad, from the respect of computing time, choosing 100 hidden units is more reasonable.

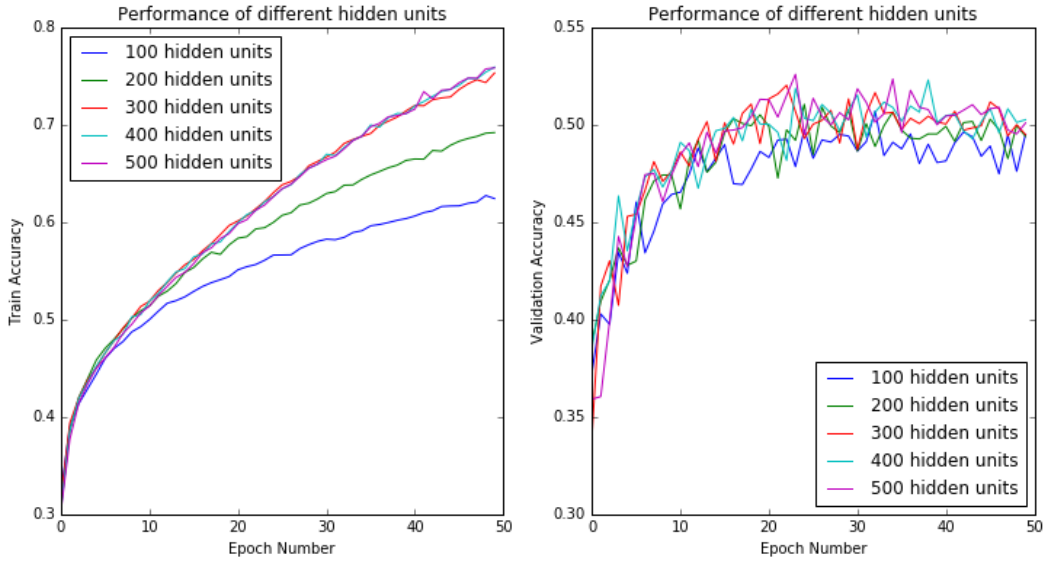

Figure 5: Performance of different hidden units on CIFAR10

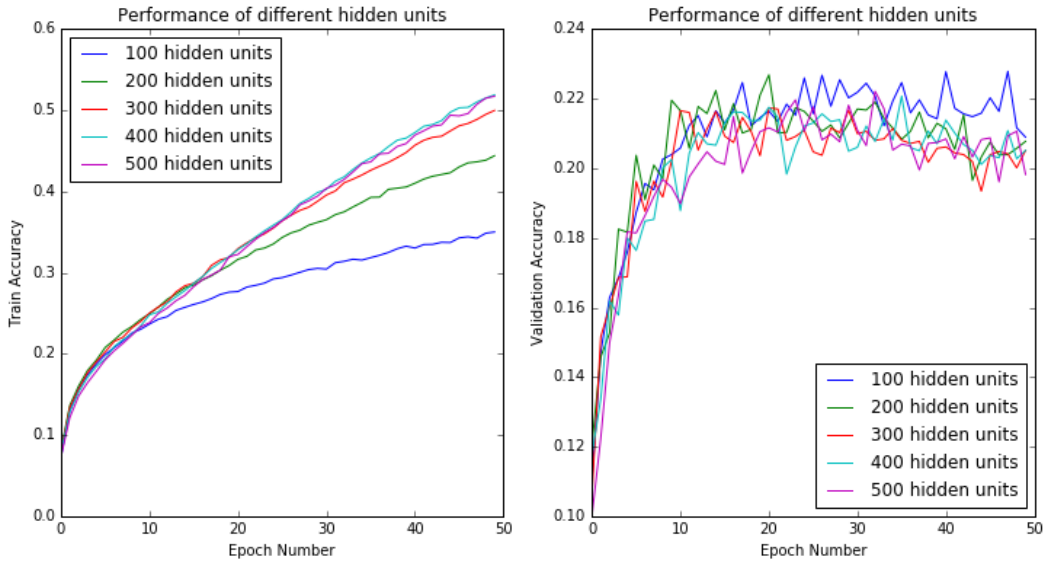

Figure 6: Performance of different hidden units on CIFAR100

In conclusion, it seems that the change of hidden units does not influence the performance a lot.

3 Learning rate schedules

3.1 Research questions

There are plenty of learning rate schedules and every learning rate schedule is suitable in a specific situation. So it is not be very clear which learning rate schedule suits this data type and test parameters. Therefore, a couple of experiments are carried out with different configuration options and

see what works best on this task.

3.2 Methods

1. Introduction of the methods
   Adapting the learning rate for stochastic gradient descent optimization procedure can increase performance and reduce training time.The approaches to change the learning rate is called learning rate schedules.

   (a) Momentum Optimizer: momentum algorithm is not a pure gradient descent algorithm because weight changes start by following the gradient. After a few updates they start to have velocity, so the weight change to go is in the previous direction. The learning rate update can be defined as: $\Delta W(t) = \lambda * D(t) + \alpha * \Delta W(t-1)$.

   (b) AdaGrad Optimizer: this algorithm is to separate and normalise update for each weight. The update step for a parameter W is normalised by the (square root of) the sum squared gradients for that parameter. Therefore, weights with larger gradient magnitudes will have smaller weight updates; small gradients result in larger updates. The learning rate update can be defines:$\Delta W(t) = \frac{-\lambda}{\sqrt{S(t)+\epsilon}} * \Delta W(t-1)$ and $S(t) = S(t-1) + D(t)^2$.

   (c) RMSProp Optimizer: RMSProp is a stochastic gradient descent version of RProp normalised by a moving average of the squared gradient similar to AdaGrad, but replacing the sum by a moving average for S.The learning rate update can be defines:$\Delta W(t) = \frac{-\lambda}{\sqrt{S(t)+\epsilon}} * \Delta W(t-1)$ and $S(t) = \beta * S(t-1) + (1-\beta) * D(t)^2$.

   (d) Adam Optimizer: it is a variant of RMSProp with momentum. The momentum-smoothed gradient is used for the update in place of the gradient.The learning rate update can be defines:$\Delta W(t) = \frac{-\lambda}{\sqrt{S(t)+\epsilon}} * \Delta M(t-1)$ and $S(t) = \beta * S(t-1) + (1-\beta) * D(t)^2$ and $M(t) = \alpha * M(t-1) + (1-\alpha) * D(t)$.

   (e) Gradient Descent Optimizer: Gradient descent is a first-order iterative optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. The learning rate update can be defines:$W(t) = W(t-1) - \lambda * G$.

2. Outline of implementation in TensorFLow

   (a) Find the optimizer module in TensorFLow. The optimizer base class provides methods to compute gradients for a loss and apply gradients to variables. A collection of subclasses implement classic optimization algorithms such as GradientDescent and Adagrad.

   (b) Find the location to use the optimizer. The optimizer function is used in the train step. So find the location of name scope called train.

   (c) Choose the learning rate the optimizer. You never instantiate the Optimizer class itself because it has the default learning rate setting, but instead instantiate one of the subclasses. For example, when Momentum Optimizer is used, learning rate should be filled and also momentum.

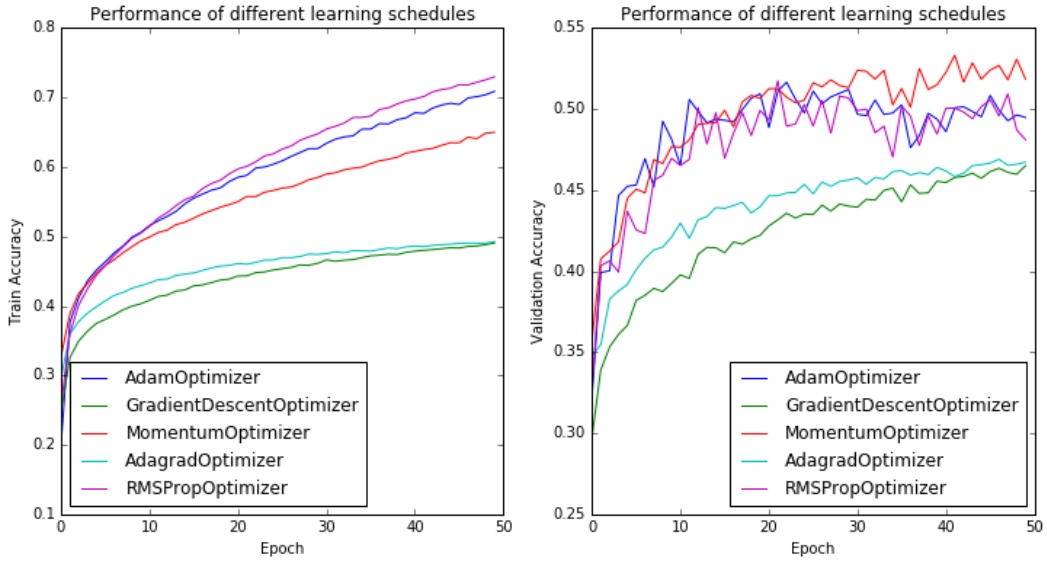

Figure 7: Performance of different learning rate schedules on CIFAR10

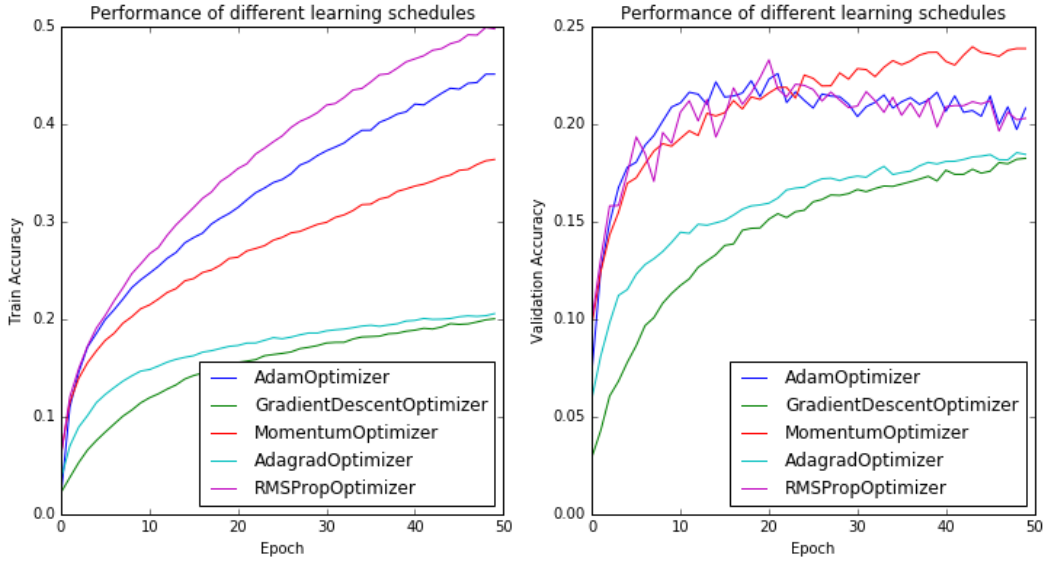

Figure 8: Performance of different learning rate schedules on CIFAR100

3.3 Result and discussion

According to figure 7, which is generate by the same epoch number, hidden layers and activation function, momentum learning rate schedule is the most successful one that increases the accuracy to 0.52. The convergence speed of Adam, RMSProp and Momentum is faster than GradientDescent and AdaGrad. Because the GradientDescent is the simplest way to change the learning rate, it is normal to have the worst performance. Because the momentum learning schedule changes the weight to go in the previous direction, in this dataset and parameter setting, it works well. The trend on CIFAR100(Figure 8) is similar to the CIFAR10. The accuracy of the algorithms was observed to increase a lot for momentum learning rate schedule. In conclusion, the momentum learning

Table 5: The performance of different learning rate schedule in CIFAR10

	train_error	valid_error	train_accuracy	valid_accuracy
AdamOptimizer	0.81	1.93	0.71	0.49
GradientDescentOptimizer	1.48	1.54	0.49	0.46
MomentumOptimizer	1.00	1.41	0.65	0.52
AdagradOptimizer	1.47	1.55	0.49	0.47
RMSPropOptimizer	0.77	2.52	0.73	0.48

Table 6: The performance of different learning rate schedule in CIFAR100

	train_error	valid_error	train_accuracy	valid_accuracy
AdamOptimizer	2.08	4.34	0.45	0.21
GradientDescentOptimizer	3.45	3.55	0.20	0.18
MomentumOptimizer	2.55	3.27	0.36	0.24
AdagradOptimizer	3.44	3.55	0.21	0.18
RMSPropOptimizer	1.95	4.84	0.50	0.20

rate schedule is suitable for both the CIFAR10 and CIFAR100. Because the CIFAR100 is more complicated than CIFAR10, the performance on that is worse.

4 Regularisation

4.1 Research questions

Does L2 regularisation offer any advantage to performance?

4.2 Method

1. Introduction of the methods
   Regularization, in the fields of machine learning, refers to a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting.It can be defined as the formula below:
   $$C = C_0 + \frac{\lambda}{2n} \sum_w W^2$$
   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 .

2. Outline of implementation in TensorFLow

   (a) Define a function to calculate the cross entropy to evaluate the error rate.

   (b) The first term should be the usual expression for the cross-entropy. A tensorflow inner function called tf.nn.softmax_cross_entropy_with_logits can be used. Besides, a penalty should be added as L2 regularisation. There is another inner function called tf.nn.l2_loss can be used. λ is a parameter for tuning.

(c) In tensorflow, a L2 regularisation can be written as below ($\lambda = 0.01$):

$$
\begin{aligned}
&with\ tf.name\_scope('error'):\\
&error = tf.reduce_mean(\\
&tf.nn.softmax\_cross\_entropy\_with\_logits(outputs, targets)\\
&+ 0.01 * tf.nn.l2\_loss(hidden_weights)\\
&+ 0.01 * tf.nn.l2_loss(hidden_weights)\\
&+ 0.01 * tf.nn.l2_loss(out_weights)\\
&+ 0.01 * tf.nn.l2_loss(out_biases))
\end{aligned}
$$

4.3 Result and discussion

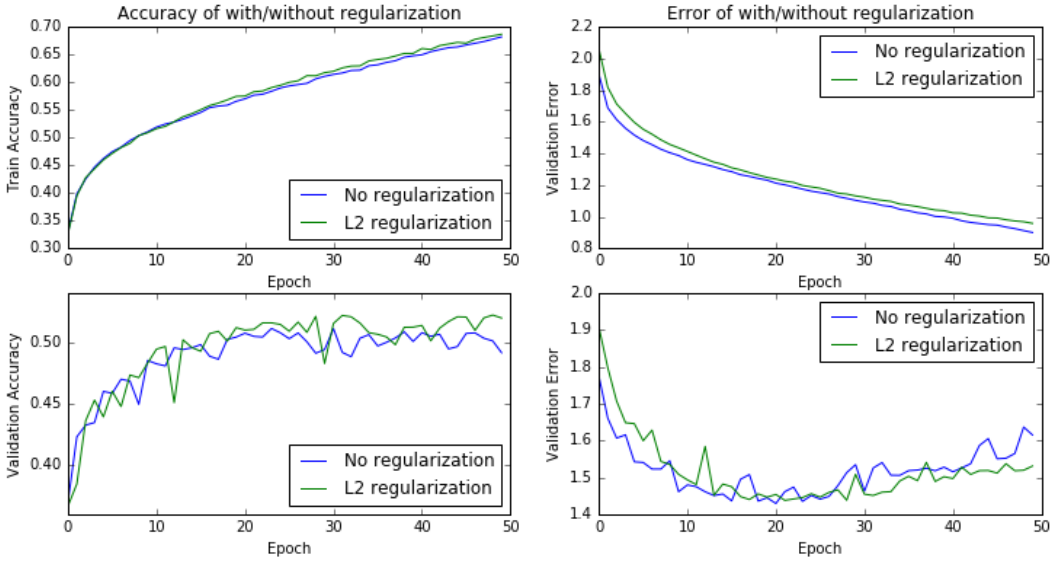

Figure 9: CNN architecture

Table 7: The performance of regularisation in CIFAR10

	train_error	valid_error	train_accuracy	valid_accuracy
Without regularisation	0.90	1.61	0.68	0.49
with regularisation	0.96	1.53	0.69	0.52

From the Figure 9, we can see that L2 regulisation slightly reduce the effect of overfitting, but it is still influenced by the overfitting. Because the gap between training accuracy and validation accuracy on L2 regularisation is not as large as the gap without L2 regularisation. Also, in the Table 7, we can find that L2 regularisation improves the accuracy from 0.49 to 0.52.

5 Baseline

5.1 CIFAR10

According to the experiments above, the optimal values of each parameter are used to build a baseline. So Elu activation function, 2 hidden layers, 400 hidden units, regularisation and momentum learning rate schedule are used. From the Table 8, the final accuracy can achieve at 0.53 and the error rate is 1.45.

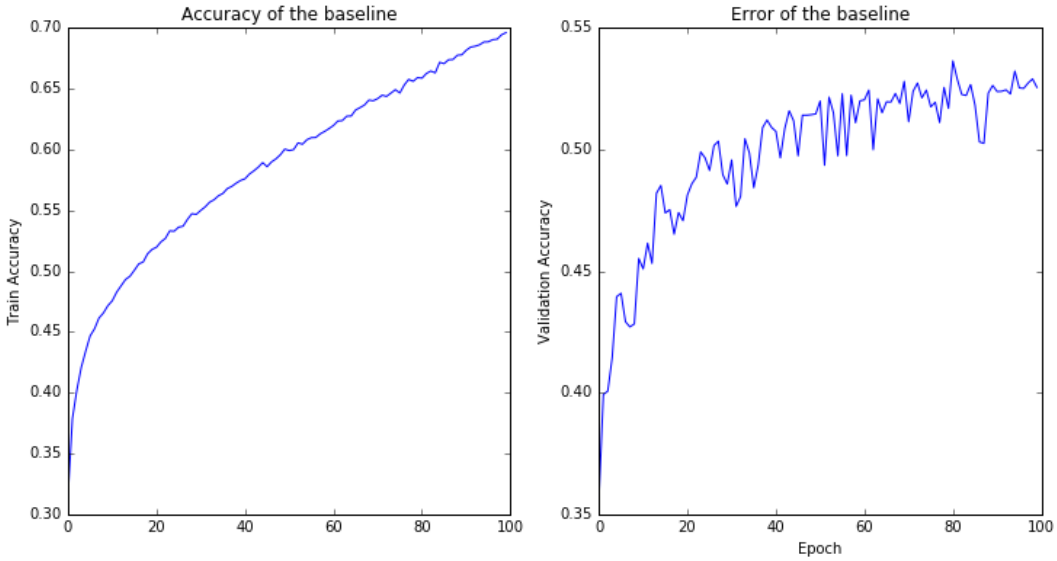

Figure 10: CIFAR10 baseline

Table 8: The performance of baseline

	train_error	valid_error	train_accuracy	valid_accuracy
CIFAR10	0.88	1.45	0.70	0.53
CIFAR100	2.70	3.36	0.34	0.23

5.2 CIFAR100

According to the experiments above, the optimal values of each parameter are used to build a baseline. So Elu activation function, 2 hidden layers, 100 hidden units, regularisation and momentum learning rate schedule are used. From the Table 8, the final accuracy can achieve at 0.23 and the error rate is 3.36.

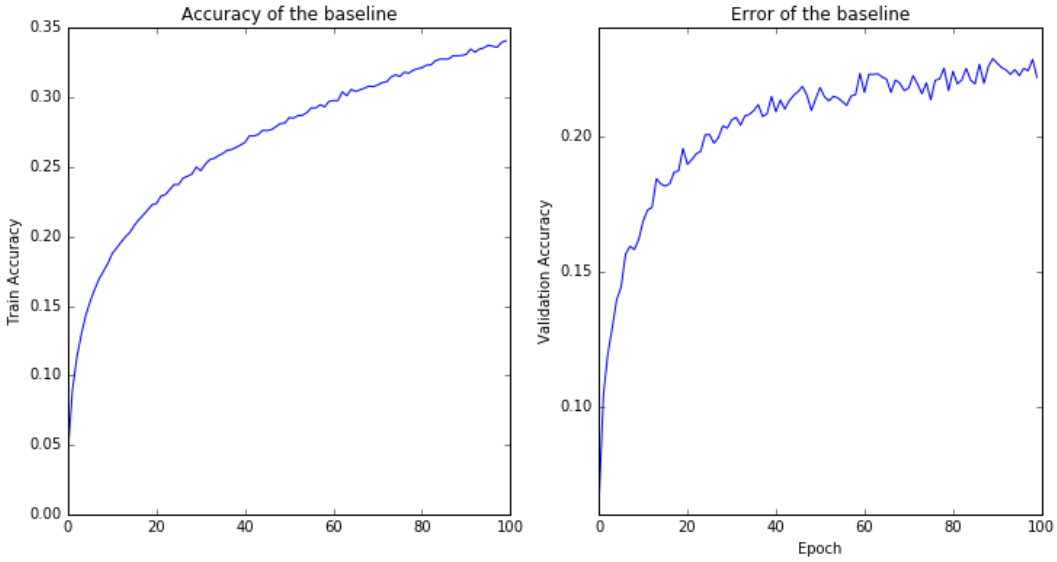

Figure 11: CIFAR100 baseline

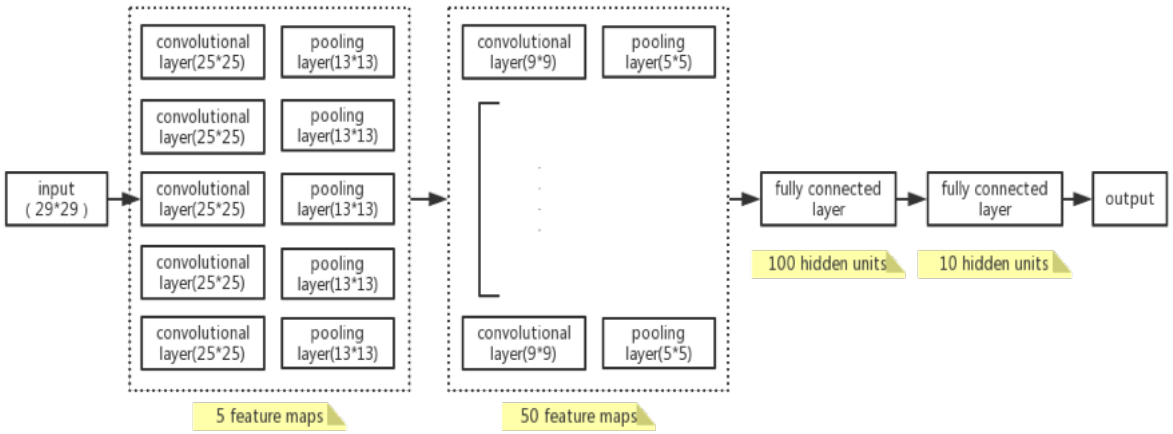

Figure 12: CNN architecture

6 Further work

6.1 Network Architecture

The whole architecture can be seen in Figure 12. Assume that the image from database is 28*28 pixels. For the pooling layer, the empty circle units cannot influence the final result. Therefore, pad the input from 28*28 to 29*29 so that these feature units can be centered on the border and can generate integer size output for the convolutional layer.

In the first convolutional layer, 5 different feature maps are applied. Each feature map is a 5*5 local receptive field. So after extracting the feature of the input, the output of the convolutional layer is 25*25 size. The next process is max pooling. The purpose of the pooling layer is to shrink the image stack. In the max pooling layer, the window size is 2*2 and the stride is 2. Walking the

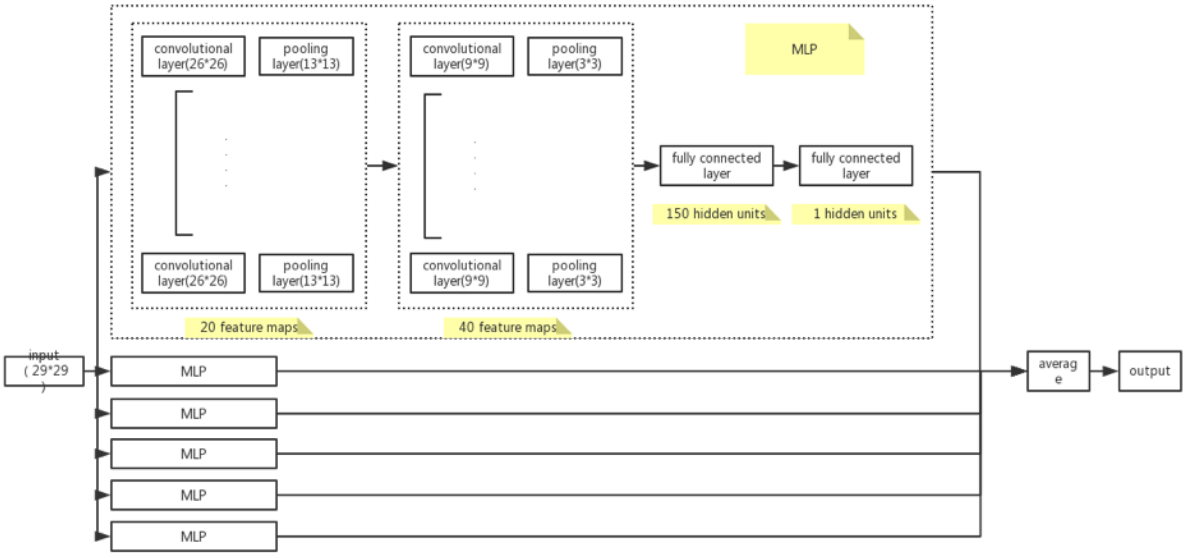

Figure 13: CNN committee architecture

window across the filtered image, from each window, take the maximum value. So the output of the pooling layer should be 13*13.
The aim of the second convolutional layer is to extract some specific features. For the second convolutional layer, 50 different feature maps and the 5*5 local receptive fields are used. Since the output of the second convolutional layer is too small (only 5*5) so that it cannot be used for the third convolution.
After doing the feature extraction, a trainable classifier in the form of 2 fully connected layers is connected. The fully connected layer can be stacked as well. For the second fully connected layer, because the number of the target(CIFAR10) is 10, the hidden unit value should be 10.
After finishing the basic CNN, a more complicated architecture is going to be tested. According to Figure 13, The structure of the committee can be viewed in the Figure 10. This committee is based the architecture of CNN. Seven different CNNs are trained by the original dataset and six different transformed datasets. Different from the setting of MLP above, the number of feature maps are changed from 5 to 20. Meanwhile, the size of local receptive field is modified from 5*5 to 4*4. So after the first convolutional layer, the output size is shifted from 25*25 to 26*26. But the window size for the first pooling layer is the same as previous (2*2). However, for the second pooling layer, the windows of size 3*3 is applied. After this, a fully connected layer that has 150 hidden units are connected, which has more than 50 hidden units than the first MLP. Every classifier will output the result independently. What the committee does is only to simply average the corresponding outputs.

6.2 Research questions

1. what is the optimal size of feature maps?How many fully connected layers is optimal?
2. Does data augmentation improve the final performance?
3. What is the more advanced approaches to regularisation? How about the performance?
4. How many CNNs need to be used in CNN committee?