Machine Learning Practical

Coursework 3 Report

2017年2月16日

Part 1: Exploring How Different Activation Function Influence the Effect of Training multi-layer networks model in TensorFlow for classification

Part 2: Exploring How Different Hidden Layer Depths and Widths Influence the Effect of Training multi-layer networks model in TensorFlow for classification

Part 3: Exploring How Different Method of Regularisation Influence the Effect of Training multi-layer networks model in TensorFlow for classification

Part 4: Exploring How Different Method of Learning Rate Schedules Influence the Effect of Training multi-layer networks model in TensorFlow for classification

Part 1: Exploring How Different Activation Function Influence the Effect of Training multi-layer networks model in TensorFlow for classificationStatement

Motivation

Compare the performance of training multi-layer networks for CIFAR-10/CIFAR-100 classification by using different activation function individually.
Invest if there is difference between the performance of using different activation function.
Invest how long do these methods run on training?
Invest how better do them perform on validation data set?

Experimental Methodology

Experiment/method design
In order to invest the research questions above, I design 3 experiments. In these three experiments(experiment 1-1, 1-2, 1-3) I using three different activation function: ReLU, sigmoid and tanh with the same other settings which include 3-layers network and 300 units per layers, using softmax cross entropy with logits function as error function, using reduce mean to calculate accuracy and adam optimiser as learning step(whose parameters are learning rate 0.0001 and epsilon 0.001). Also I run the same experiments on the CIFAR-10 and CIFAR-100 data set. In these experiment I record the error rate/accuracy/running time on training/validation data set. So that I could use them to compare the performance of training and invest the research questions.
Outline of Implementation

1. Import the necessary package and prepare the training and validation data set by using the CIFAR10DataProvider and CIFAR100DataProvider form mlp.data_provider.
2. Define a fully connected layer function*(nonlinearity=tf.nn.relu / tf.nn.sigmoid / tf.tanh* as incoming parameters and here I decide which activation function I want to use*)* which could calculate the weights and biases and return the outputs calculated by using weights and biases.
3. Reset the default graph on TensorFlow.
4. Set up the training model which is a 3-layer network with 300 units per layer by using fully_connected_layer function I defined before and the middle layer use the outputs of the previous layers as input.
5. Define the error function by using softmax cross entropy with logits function
6. Define the calculating accuracy function by using reduce mean method
7. Define the learning step function by using Adam optimizer(tf.train.AdamOptimizer*(learning_rate = 0.0001, epsilon = 0.001).minimize(error))* whicn has learning rate 0.0001 and epsilon 0.001
8. Set the global initial variables.

9. Define arrays to record the error rate / accuracy / running time on training / validation data set.
10. Use the session function of TensorFlow to train the data by using the model which I set up before and when training is running I print and record the error rate / accuracy / running time on training / validation data set.
11. After running the three experiment, plot the chart to analyse.

Experiment result

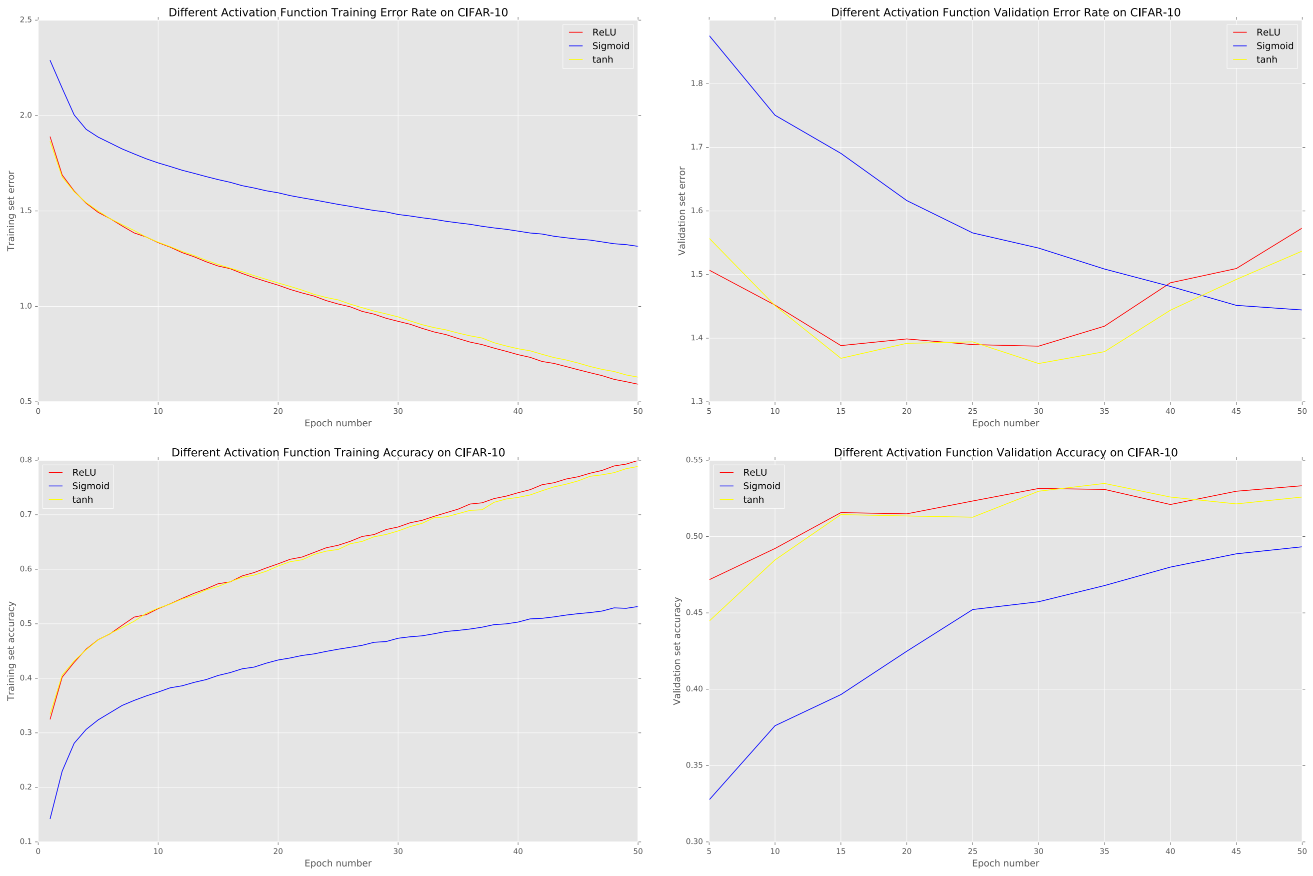


Figure 1-10

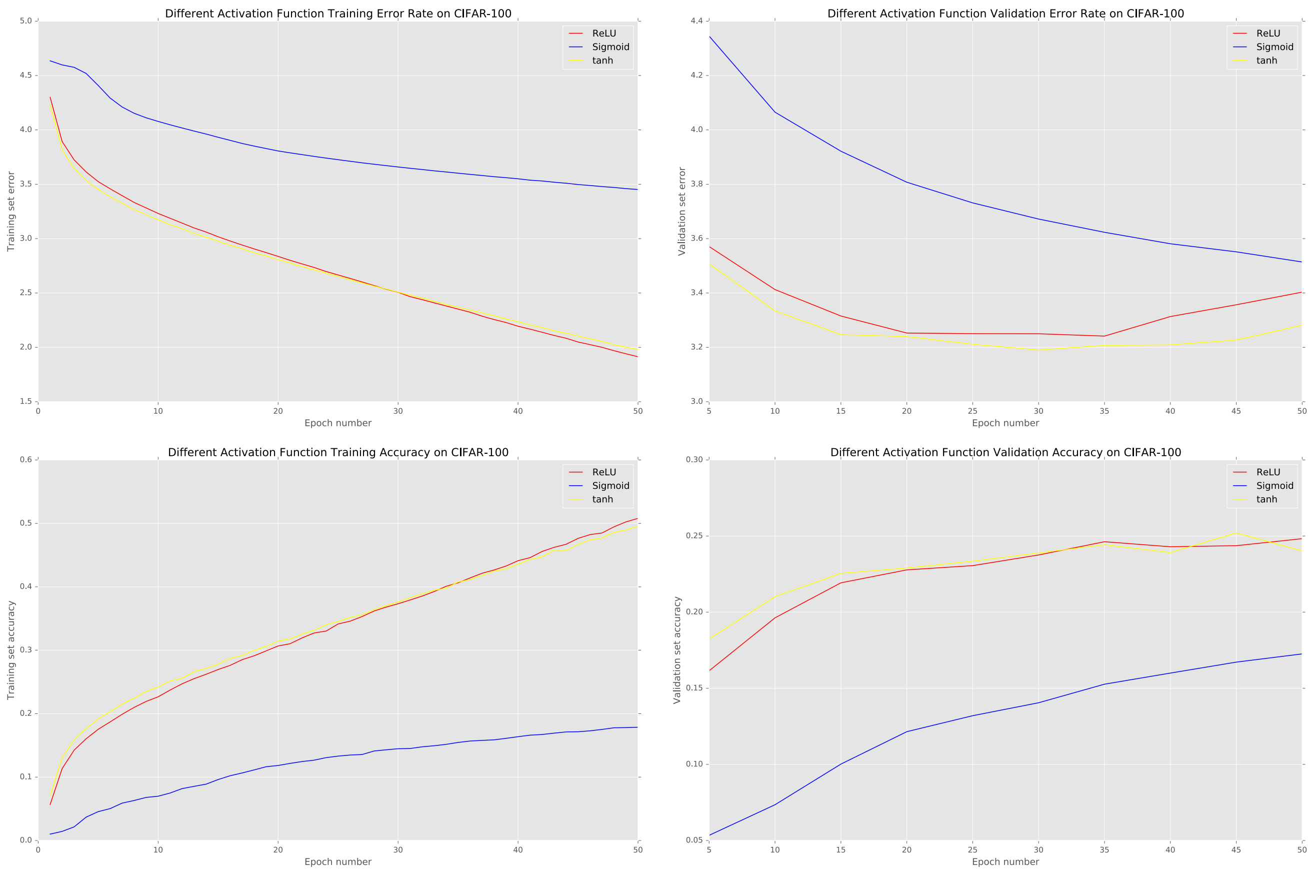


Figure 1-100

	final error(train)	final acc(train)	final error(valid)	final acc(valid)
ReLU	0.59	0.8	1.57	0.53
sigmoid	1.31	0.53	1.44	0.49
tanh	0.63	0.79	1.54	0.53

Chart 1

Discussion and Conclusion

1. From the figure 1-10 and figure 1-100 above we could see that the accuracy of validation with ReLU and tanh are larger than the accuracy of valid with logistic sigmoid and their error of validation are smaller than sigmoid. This result is more obvious on the CIFAR-100. And we could see that the using ReLU and tanh is almost similar on the performance of training.
2. And through the data of running time, we could find that ReLU cost less time than tan.

So there is different between the performance of using sigmoid activate function and using ReLU and tanh. Using ReLU and tanh is better a lot than using sigmoid. And using ReLU cost less time than others. The optimal non-linear transformations is the Relu, and the performances of using Relu and Tanh are better than using Sigmoid.

Part 2: Exploring How Different Hidden Layer Depths and Widths Influence the Effect of Training multi-layer networks model in TensorFlow for classification

Motivation

Compare the performance of training multi-layer networks for CIFAR-10/CIFAR-100 classification by using different hidden layer depths/widths individually.
Invest if there is difference between the performance of using different hidden layer depths/widths.
Invest how long do these different depths/widths run on training.
Invest how better do them perform on validation data set.
Invest whether there is a tendency of performance/time- consuming changes as changing the depths/widths.

Experimental Methodology

Experiment/method design
In order to invest the research questions above, I design 5 experiments(experiment 2-1, 2-2, 2-3, 2-4, 2-5). In these five experiments I divided them into two groups to invest the different research question about depths and widths. Each group has three experiment. So one experiment will be used in two different group commonly. In the first group(experiment 2-1, 2-2, 2-3), it uses three different experiments which has the same 300 units per layer but different in the number of lays. In the second group(experiment 2-2, 2-4, 2-5), it uses three different experiments which has the same 3-layers network but different in the number of the units per layer. All of the experiments have the same other settings which include using ReLU activation function, using softmax cross entropy with logits function as error function, using reduce mean to calculate accuracy and adam optimiser as learning step(whose parameters are learning rate 0.0001 and epsilon 0.001). Also I run the same experiments on the CIFAR-10 and CIFAR-100 data set. In these experiment I record the error rate/accuracy/running time on training/validation data set. So that I could use them to compare the performance of training and invest the research questions.
Outline of Implementation

1. Import the necessary package and prepare the training and validation data set by using the CIFAR10DataProvider and CIFAR100DataProvider form mlp.data_provider.
2. Define a fully connected layer function*(nonlinearity=tf.nn.relu)* as incoming parameters which could calculate the weights and biases and return the outputs calculated by using weights and biases.
3. Reset the default graph on TensorFlow.

4. Set up the training model which is a 1/3/5-layer network with 300/500/1000 units per layer by using fully_connected_layer function I defined before and the middle layer use the outputs of the previous layers as input.
5. Define the error function by using softmax cross entropy with logits function
6. Define the calculating accuracy function by using reduce mean method
7. Define the learning step function by using Adam optimizer(tf.train.AdamOptimizer*(learning_rate = 0.0001, epsilon = 0.001).minimize(error))* whicn has learning rate 0.0001 and epsilon 0.001
8. Set the global initial variables.
9. Define arrays to record the error rate/accuracy/running time on training/validation data set.
10. Use the session function of TensorFlow to train the data by using the model which I set up before and when training is running I print and record the error rate/accuracy/running time on training/validation data set.
11. After running the three experiment, plot the chart to analyse.

Experiment result

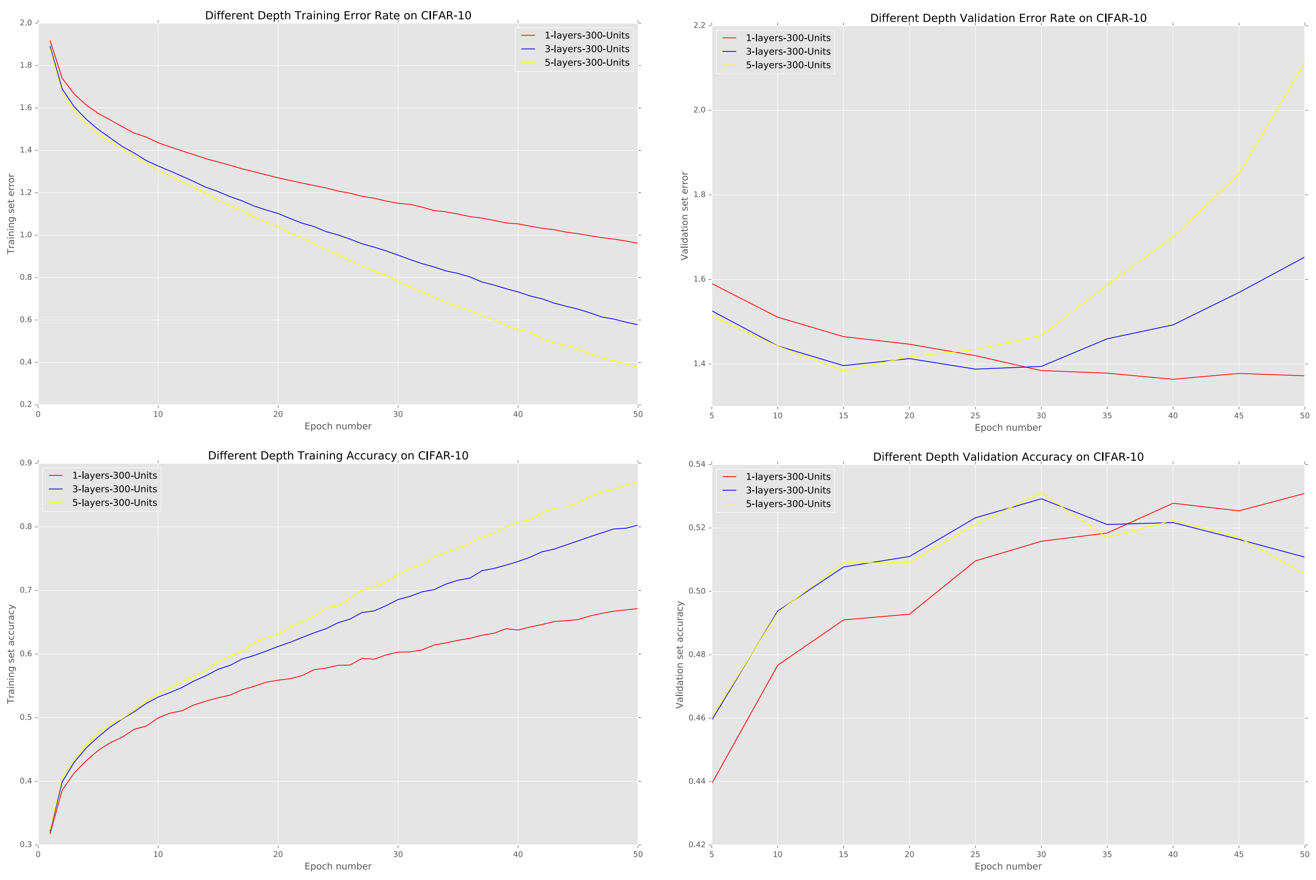


Figure 2-1-10

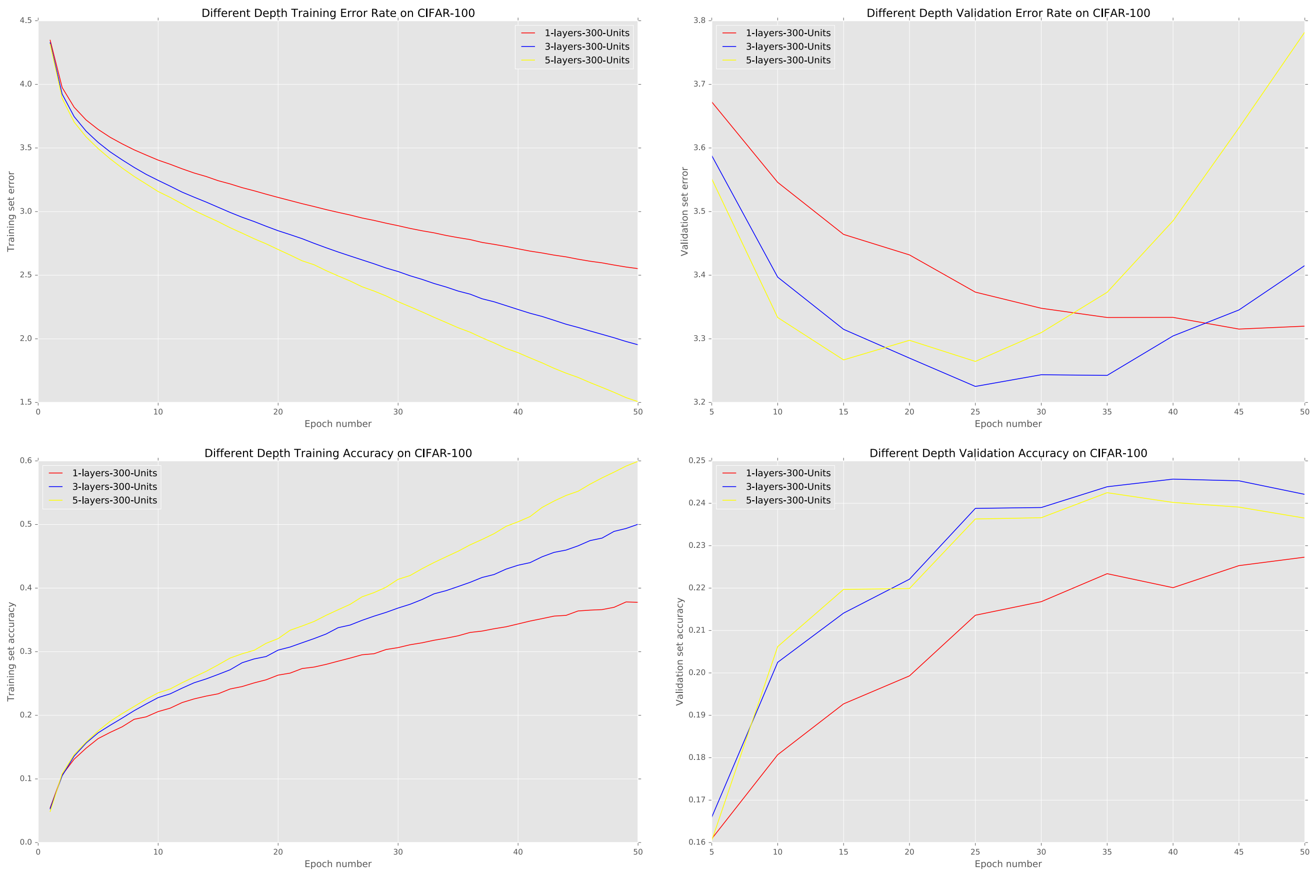


Figure 2-1-100

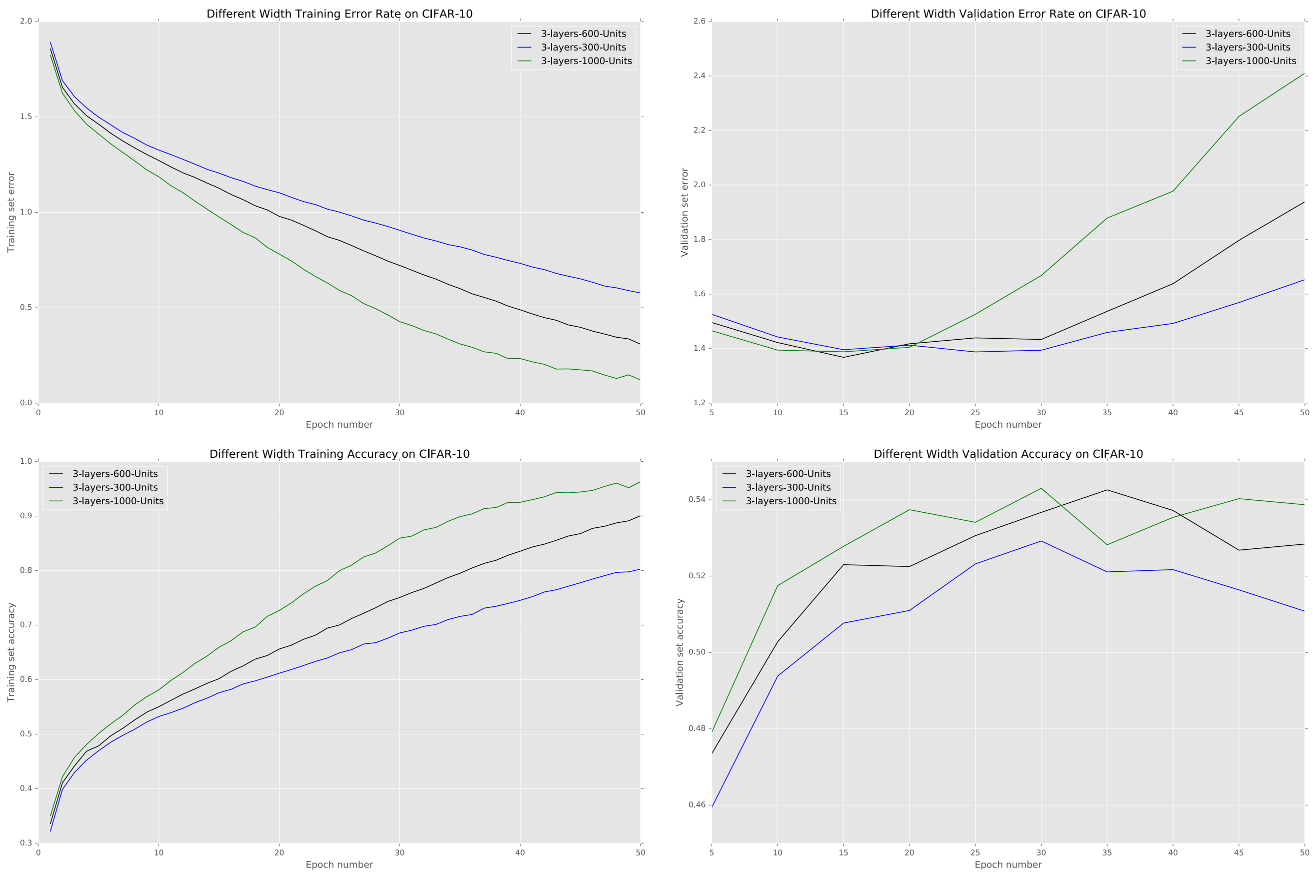


Figure 2-2-10

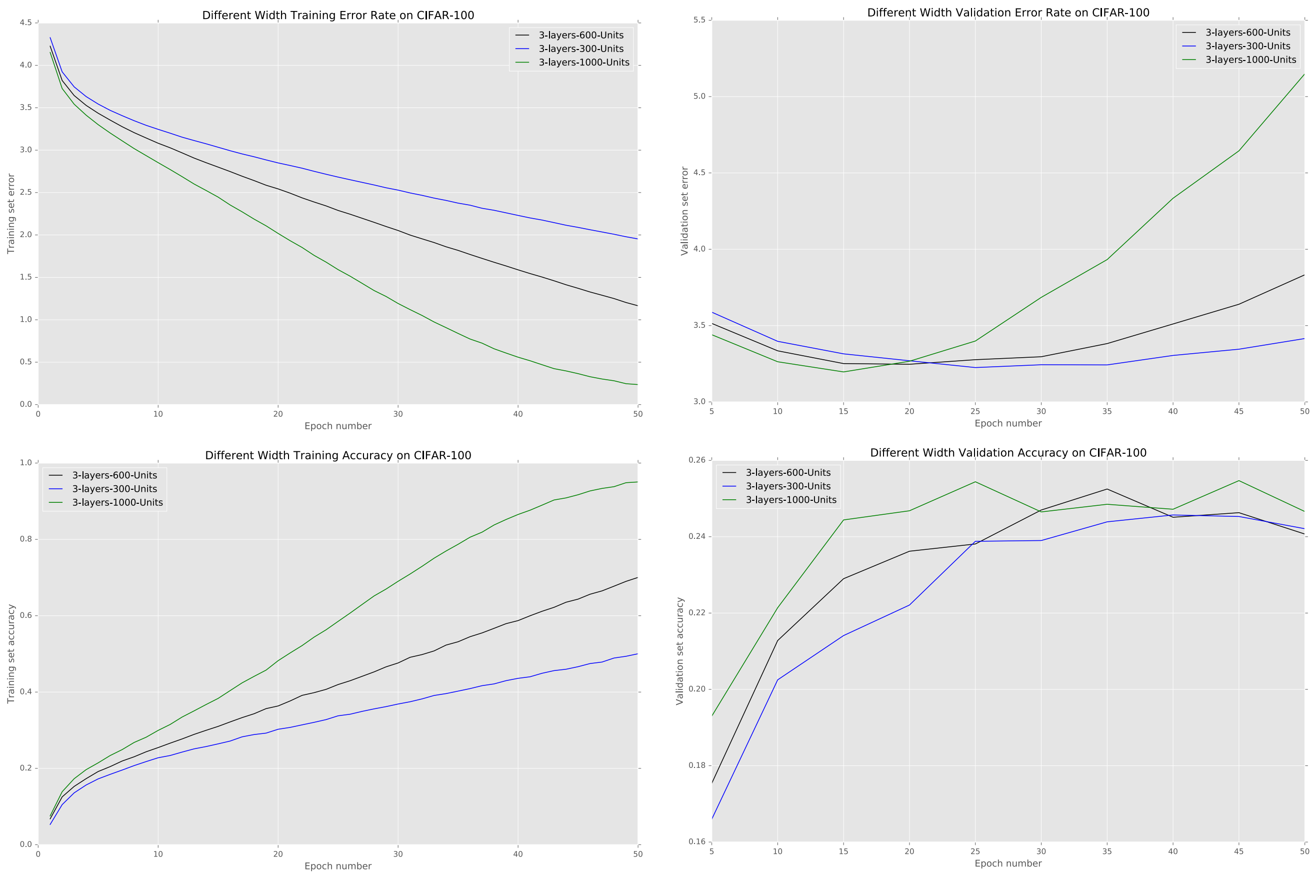


Figure 2-2-100

	final error(train)	final acc(train)	final error(valid)	final acc(valid)	running time per epoch
1-layer, 300 units	0.96	0.67	1.37	0.53	10.1
3-layer, 300 units	0.58	0.8	1.65	0.51	11.7
5-layer, 300 units	0.38	0.87	2.11	0.51	13.5
3-layer, 500 units	0.31	0.9	1.94	0.53	21.5
3-layer, 1000 units	0.12	0.96	2.41	0.54	48.7

Chart 2

Discussion and Conclusion

1. From the figure 2-1–10 and figure 2-1-100 above we could see that the accuracy and error rate of validation with different depths and same widths are similar. This result is similar on the CIFAR-100.
2. From the figure 2-2-10 and figure 2-2-100 and chart 2 above we could see that the accuracy and error rate is different when we using the same depths and different widths.
3. Using the same depths and a larger width, it will get a better accuracy when the number of units is not large. But it needs much more time and its error of validation will be larger. When the same situation but width is large, the increasing of widths will not improve the accuracy of validation and the error rate of validation will get larger, there will be over-fitting and the run time will be longer.
4. I found that 3 layers 500 units/layer needs more time than 5 layers 300 units/layer. However, 3 layers 500 units/layer has better accuracy on validation than 5 layers 300 units/layer both on CIFAR-10 andCIFAR-100.I think it is because that 3 layers 500units/layer calculate 32*32*3*500 + (500^2) *2 + 500*10 = 2041000 times and the 5 layers 300units/layer calculate 32*32*3*300 + (300^2) *4 + 300*10 =1284600 times, so the run time of 5 layers 300units/layer is shorter than 3 layers 500units/layer.

So there is no obviously different between the performance of using different depths and same widths. But there is different between the performance of using different width and same depth. I found that if we want to find the optimal depths and width, Their combination must fit this data set, not too large not too small. If we have a large width there may be over-fitting and will get less or no improvement on accuracy. Also it will cost more time. So there is no specific tendency of performance changes as changing the depths/widths. But I could sure that when we make the depth or width larger, the time-consuming gets larger.

Part 3: Exploring How Different Method of Regularisation Influence the Effect of Training multi-layer networks model in TensorFlow for classification

Motivation

Compare the performance of training multi-layer networks for CIFAR-10/CIFAR-100 classification by using different method of regularisation individually.
Invest whether it performs better by using regularisation than not.
Invest if there is difference between the performance of using different method of regularisation.
Invest how long do these different method of regularisation run on training?
Invest how better do them perform on validation data set.
Invest whether it performs better by using combination of L1/L2/dropout than not.

Experimental Methodology

Experiment/method design
In order to invest the research questions above, I design 6 experiments. In these six experiments I use six different regularisation including: using none regularisation, using L1 regularisation, using L2 regularisation, using L1+L2 regularisation, using dropout regularisation, using L1+L2+dropout regularisation. All of the experiments have the same other settings which include using ReLU activation function, using softmax cross entropy with logits function as error function, using reduce mean to calculate accuracy and adam optimiser as learning step(whose parameters are learning rate 0.0001 and epsilon 0.001). Also I run the same experiments on the CIFAR-10 and CIFAR-100 data set. In these experiment I record the error rate/accuracy/running time on training/validation data set. So that I could use them to compare the performance of training and invest the research questions.
Outline of Implementation

1. Import the necessary package and prepare the training and validation data set by using the CIFAR10DataProvider and CIFAR100DataProvider form mlp.data_provider.
2. Define a fully connected layer function*(nonlinearity=tf.nn.relu* as incoming parameters and add a new parameter called keep_prob) which could calculate the weights and biases. The function return weights and the outputs calculated by the dropout function*(tf.nn.dropout(nonlinearity(tf.matmul(inputs, weights) + biases), keep_prob)).*
3. Reset the default graph on TensorFlow.
4. Set up the training model which is a 3-layer network with 300 units per layer by using fully_connected_layer function I defined before and the middle layer use the outputs of the previous layers as input. And in addition, I need to record the different weight generated by the every layers.

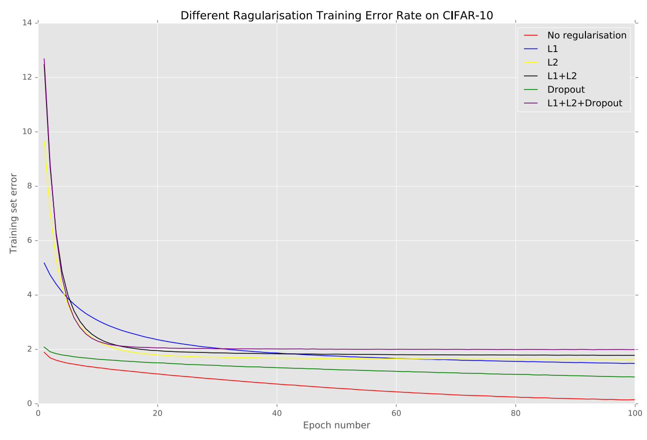
5. Define the L1/L2 regularisation function to calculate the value which I need to add to the error rate. In these function, I need the weight generated by the every layers.

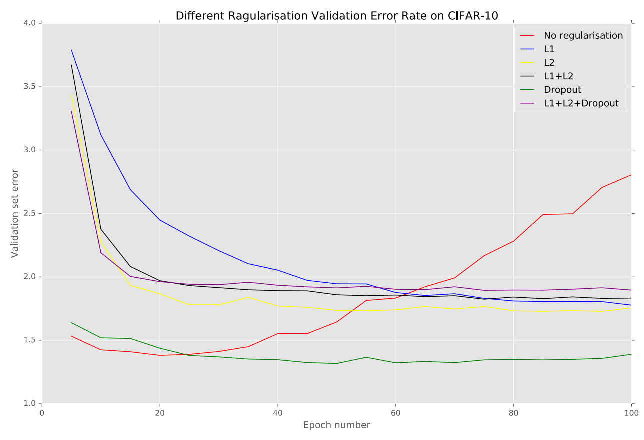
(l1_reg = (tf.reduce_sum(tf.abs(weight_1)) + tf.reduce_sum(tf.abs(weight_2)) +tf.reduce_sum(tf.abs(weight_3)) + tf.reduce_sum(tf.abs(weight_4))))

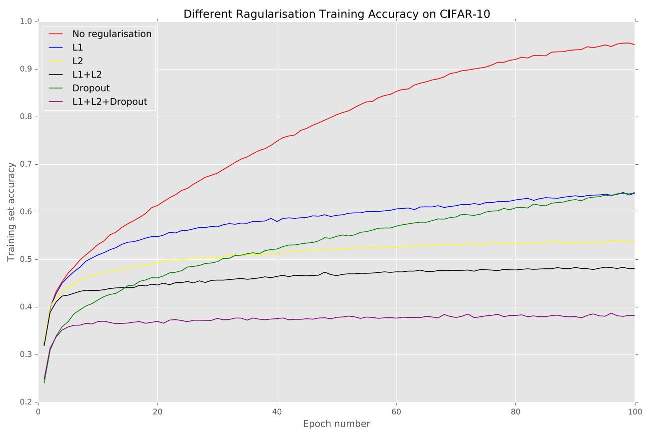
(l2_reg = tf.nn.l2_loss(weight_1) + tf.nn.l2_loss(weight_2) + tf.nn.l2_loss(weight_3) + tf.nn.l2_loss(weight_4))

6. Define the error function by using softmax cross entropy with logits function and add the L1/L2 regularisation generated by previous step
7. Define the calculating accuracy function by using reduce mean method
8. Define the learning step function by using adam optimizer(tf.train.AdamOptimizer(*learning_rate = 0.0001, epsilon = 0.001).minimize(error))* whicn has learning rate 0.0001 and epsilon 0.001
9. Set the global initial variables.
10. Define arrays to record the error rate/accuracy/running time on training/validation data set.
11. Use the session function of TensorFlow to train the data by using the model which I set up before and when training is running I print and record the error rate/accuracy/running time on training/validation data set.
12. After running the three experiment, plot the chart to analyse.

Experiment result







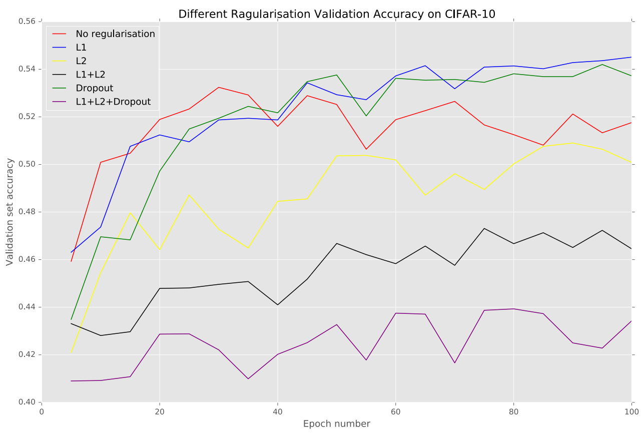


Figure 3-10

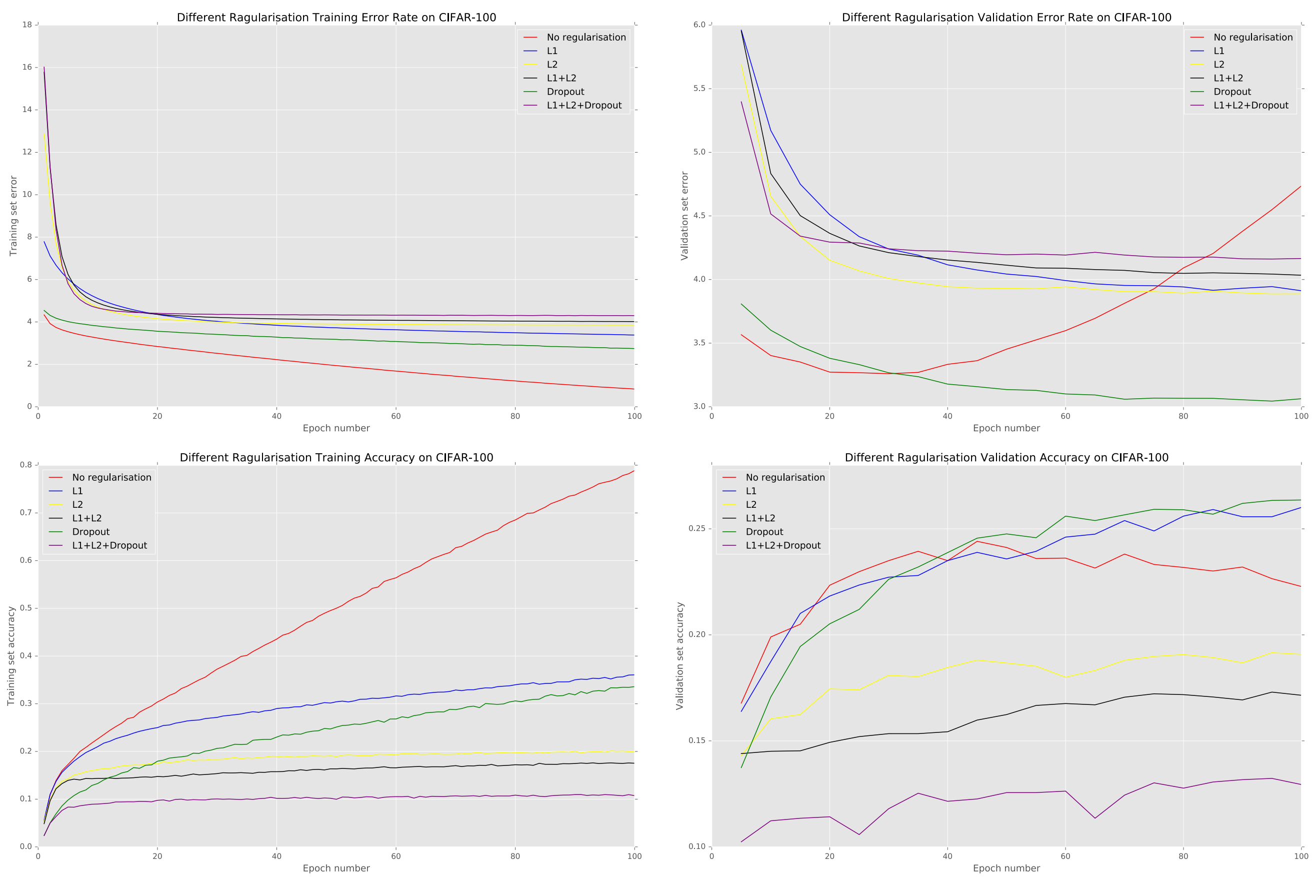


Figure 3-100

	final error(train)	final acc(train)	final error(valid)	final acc(valid)	running time per epoch
None	0.16	0.95	2.81	0.52	11.5
L1	1.49	0.64	1.78	0.55	34.7
L2	1.63	0.54	1.76	0.5	19.2
L1+L2	1.79	0.48	1.83	0.46	36.3
Dropout	0.99	0.64	1.39	0.54	13.5
L1+L2+Dropout	1.99	0.38	1.9	0.43	30.1

Chart 3

Discussion and Conclusion

1. From the figure 3–10 and figure 3-100 above we could see that the accuracy of validation are larger than others with Dropout and L1. And with dropout there is least error rate of validation. This result is similar on the CIFAR-100.
2. And it cost more time with Dropout and L2 than using L1 or using nothing.

3. And we could find that the performance of combining L1/L2/Dropout is not good even worse than using them individually. That may be because the regularization's role is to prevent over-fitting, using two or three methods together may will cause under-fitting like the results of L1L2 and L1L2+Dropout.
4. We could easily find that if we do not use regularisation, there is overfitting shown in plot.

So I think it performs better by using regularisation than not. Using regularisation could avoid over-fitting. And there is difference between the performance of using different method of regularisation. The optimal method is Dropout. Because it has better accuracy and costs less time. And using them individually is better than combining.

Part 4: Exploring How Different Method of Learning Rate Schedules Influence the Effect of Training multi-layer networks model in TensorFlow for classification

Motivation

Compare the performance of training multi-layer networks for CIFAR-10/CIFAR-100 classification by using different method of learning rate schedules individually.
Invest whether it performs better by using method of learning rate schedules than using a constant learning rate.
Invest if there is difference between the performance of using different method of learning rate schedules.
Invest how better do them perform on validation data set.

Experimental Methodology

Experiment/method design
In order to invest the research questions above, I design 3 experiments. In these three experiments I use three different a different learning step function: Gradient Descent Optimizer and I use 3 different learning rate: using constant learning rate 0.01, using exponential decay to learning rate, using inverse time decay to learning rate. All of the experiments have the same other settings which include using ReLU activation function, using softmax cross entropy with logits function as error function, using reduce mean to calculate accuracy. Also I run the same experiments on the CIFAR-10 and CIFAR-100 data set. In these experiment I record the error rate/accuracy/running time on training/validation data set. So that I could use them to compare the performance of training and invest the research questions. Outline of Implementation
Outline of Implementation
1. Import the necessary package and prepare the training and validation data set by using the CIFAR10DataProvider and CIFAR100DataProvider form mlp.data_provider.
2. Define a fully connected layer function*(nonlinearity=tf.nn.relu* as incoming parameters*)* which could calculate the weights and biases and return the outputs calculated by using weights and biases.
3. Reset the default graph on TensorFlow.
4. Set up the training model which is a 3-layer network with 300 units per layer by using fully_connected_layer function I defined before and the middle layer use the outputs of the previous layers as input.
5. Define the error function by using softmax cross entropy with logits function
6. Define the calculating accuracy function by using reduce mean method
7. Define a learning rate schedules function to set the learning rate schedules by three different way: one is using exponential decay to learning rate*(learning_rate = tf.train.exponential_decay(starter_learning_rate = 0.01, global_step, 10000, 0.96,*

staircase=True)), the other is using inverse_time_decay to learning rate*(learning_rate = tf.train.inverse_time_decay(starter_learning_rate = 0.01, global_step, 10000,k=0.5)),* last is using a constant learning rate 0.01.

8. Define the learning step function by using Gradient Descent Optimizer*(tf.train.GradientDescentOptimizer(learning_rate = learning_rate).minimize(error)).*
9. Set the global initial variables.
10. Define arrays to record the error rate/accuracy/running time on training/validation data set.
11. Use the session function of TensorFlow to train the data by using the model which I set up before and when training is running I print and record the error rate/accuracy/running time on training/validation data set.
12. After running the three experiment, plot the chart to analyse.Experiment result.

Experiment result

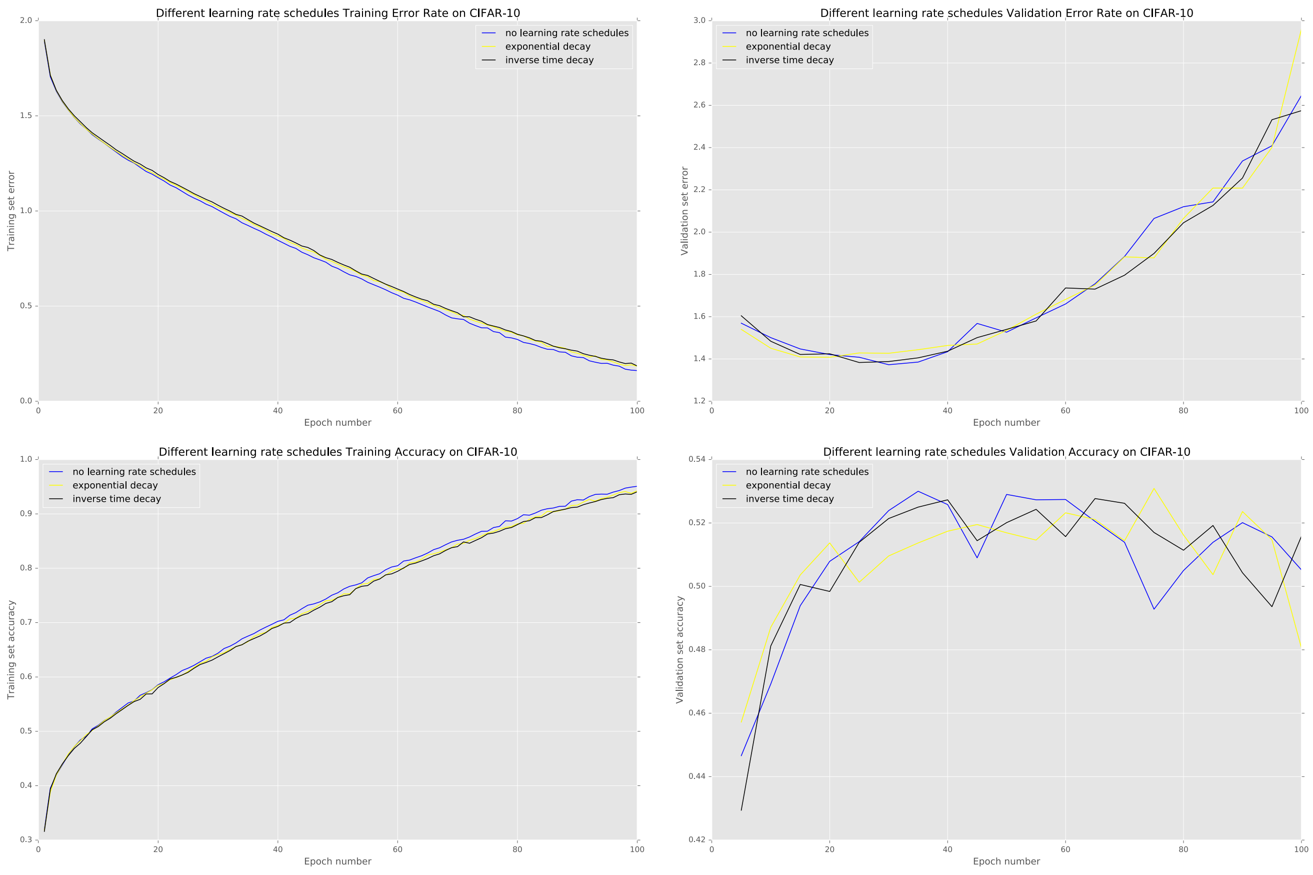


Figure 4-10

Figure 4-100

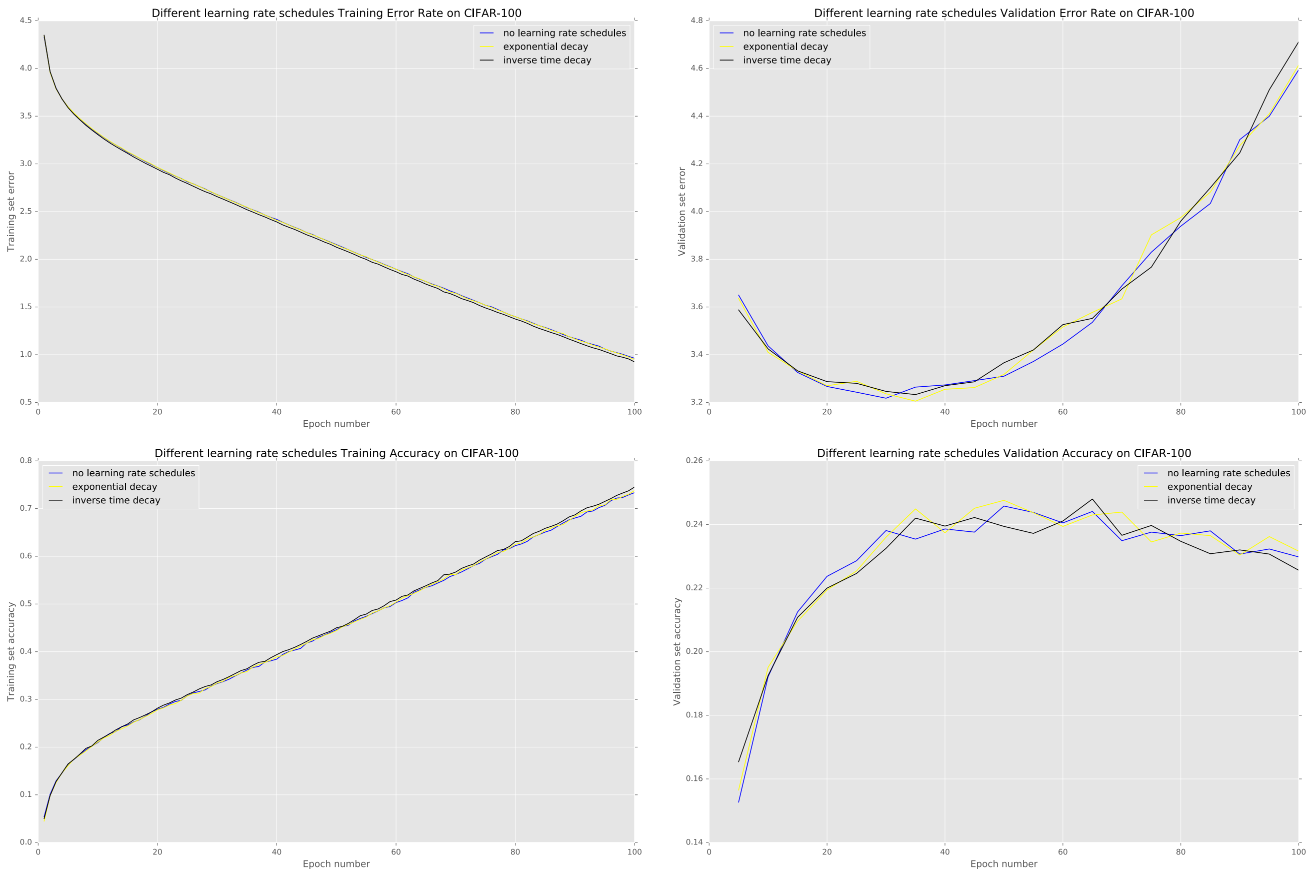


Chart 4

	final error(train)	final acc(train)	final error(valid)	final acc(valid)
constant learning rate	0.16	0.95	2.65	0.51
exponential decay	0.18	0.94	2.96	0.48
inverse time decay	0.19	0.94	2.57	0.52

Discussion and Conclusion

From the figure 4–10 and figure 4-100 above we could see that the accuracy of validation are larger than others with inverse time decay and to some extent they are almost similar on the CIFAR-100.

So I think it performs better by using inverse time decay to learning rate than not. The optimal method is inverse time decay. But on the whole on CIFAR-100 they are almost similar.

Future Work

In these four parts, I explore some basic research questions. In the advanced experiments I will explore more advanced approaches. I want to explore different network architectures(recurrent networks or convolutional networks), more advanced approaches to regularisation(explore many variants of dropout proposed).