

Qidi Han (6814891757)
EE-569
Homework 5
March-1-2020

Problem 1: Geometric Image Modification

I. Abstract and Motivation

CNN is the most popular technical in the field machine learning and AI. For the future of our life, CNN will still be used in many different perspectives. In this project, I will use this technology to do some analysis. Before I use this technology, I will read many different paper and class notes to understand the components and concepts of this process. Then, I will use those concepts to build a CNN model in python. This achievement will help me better understand the CNN and the process. There are also many different datasets for using in the LeNet5. For example, there are MNIST and CIFAR-10 dataset. In this project, I will use CIFAR-10.

II. Approach and Procedures

a. CNN Architecture

In this problem, I will answer the question, and explain the concept of CNN. There are four small question in this problem. The first two and last question is about the component and concept of CNN. The third question is about the why traditional approach is worth than CNNs. I will answer each question in the discussion part. The CNN in our project is called LeNet-5. The 5 represent the 5 different layers. I will mention each layer in the discussion part. I will use CIFAR-10 dataset in our question. CIFAR-10 contain 50k train data, and 10k test data. There are all RGB images. For part b, I will design and learn the process of LeNet-5 by using CIFAR-10.

b. CIFAR-10 Classification

1. There are four parts in my project. Since, I use the torch in Python, the first part is build the CNN kernel, also called the architecture of the CNN. The function called Net(). There are two important parts of this function, the first one is Conv2d() function. Since our image is RGB, therefore, we have 3 input image channel, 6 output channels, and we use 3*3 square convolution to do the process. For the fully connected layer, we will use the affine formula below. Figure 2 shows the kernel information for our process

$$y = Wx + b \quad (1)$$

The second step load the image, I write this function as load_data(batch_size). The batch_size can be changed for our test and train data. There are nothing special in this step, we just download the CIFAR10 from the datasets. And store the data in the train_load_data and test_load_data. Both results will be the return value for our case.

The third step is the train process function. I will use enumerate() function to divide the number of data evenly. Then each data will go through the net() function in the step 1, use the loss function to determine the missing class for all the data set. I will set data to be 0, when the data hits their run 200 times. As I train all the image, I will calculate the train data accuracy rate

and test data accuracy rate. The predicted result from the CNN process will compare with our data label. Then I will result the train data accuracy rate and test data accuracy rate.

The last step is run all the functions in the previous steps. I will run 50 epochs for our problem. And I will plot the figure to show the result with each epoch and accuracy rate. Since, we need 5 different initial parameters setting, I will set `learn_rate` to be 0.001 and 0.01, and `batch_size` with 64 and 128. Table 1 shows the result for 5 different parameters setting. Figure 2a, 2b, 2c, 2d, and 2e shows the plot for each parameter setting. Finally, I will discuss observation and effect of different settings.

2. For this question, I will pick the best result from our 5 different settings. The best result means that the test data give us the best accuracy. Then, I will do the same step as last question. I will plot the curves for test and train data.

c. State-of-the-Art CIFAR-10 Classification

For the last question, we will choose one paper from the list and discuss the process and result for the paper. Then I will compare the solution with LeNet-5 with both pros and cons.

III. Experimental Results

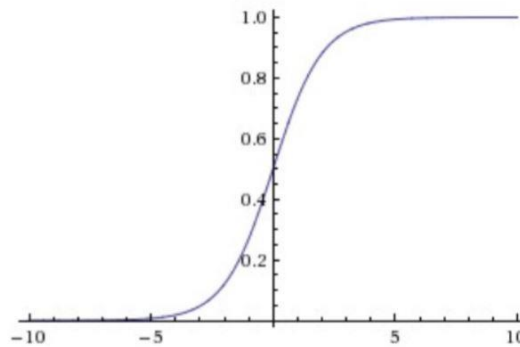


Figure 1a. sigmoid wave

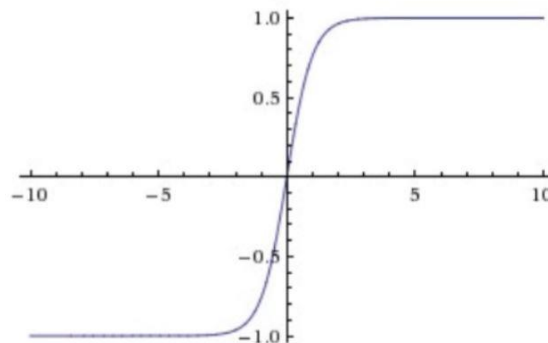


Figure 1b. Tanh wave

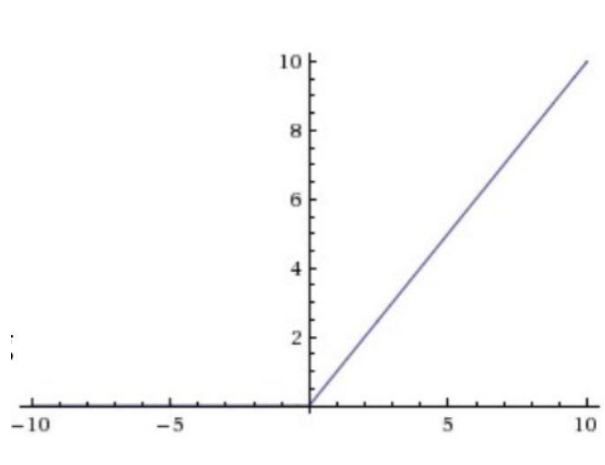


Figure 1c. Relu wave

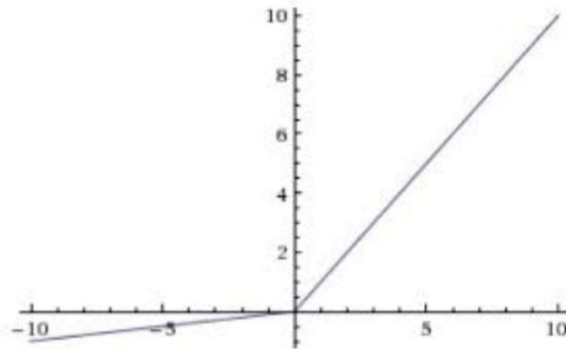
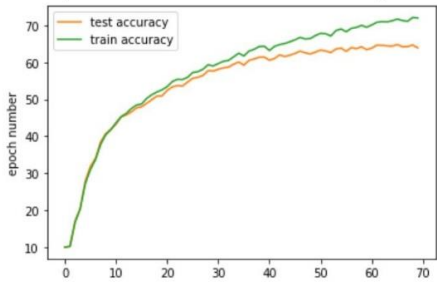
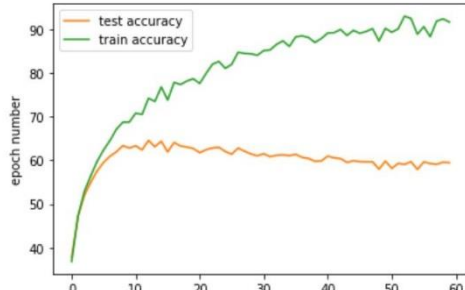


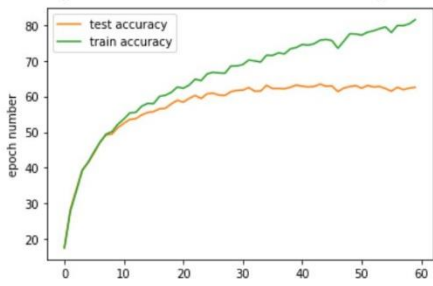
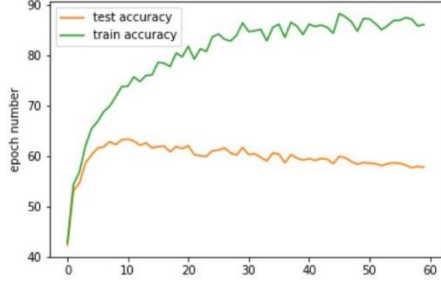
Figure 1d. Leaky Relu

```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

Figure a shows the kernel information for our process

	1	2
Batch_size	128	128
Learn_rate	0.001	0.01
Epoch_number	70	60

Figure2a,2b, 2c,2d,2e			
	Best train accuracy	72%	92%
	Best test accuracy	64%	63%

	3	4
Batch_size	64	64
Learn_rate	0.001	0.01
Epoch_number	60	60
Figure2a,2b, 2c,2d,2e		
Best train accuracy	81%	82%
Best test accuracy	63%	62%

	5
Batch_size	256
Learn_rate	0.01
Epoch_number	60

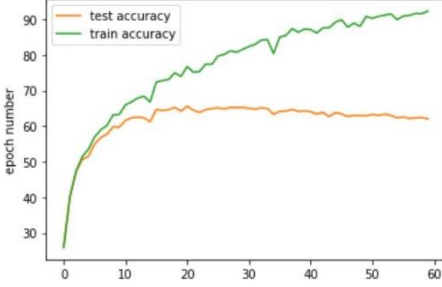
Figure2a,2b, 2c,2d,2e	
Best train accuracy	92%
Best test accuracy	65%

Table 1 shows the result for 5 different parameters setting.

IV. Discussion

a-1

1) The fully connected layer:

This is the layer that I mentioned in the part b with equation 1. This connected layer uses the equation 1 to reduce the dimension. There are totally two layers in this fully connected layer. The first connected layer takes the feature information from the max-pooling layer. Then, we compute the second connected layer by multiple a weight vector with our first connected layer. Also, it will reduce the dimension from 120 to 84. There are totally 4 layers, the first layer called input layer, it also calls max-pooling layers. There are two hidden layers. It used to classifier the feature information. The final layer called output layers, it will output the number between 0 and 1.

2) The convolutional layer:

The convolutional layer is the main corn layer in our process. I also mentioned convolutional layer in first approach. There are two convolutional layers. There are before each max-pooling layer. The first convolutional layer will use the original image multiple with 5*5 matrix. Because the image is RGB images, and there are 6 filter matrices. Therefore, the first convolutional layer will have 6*28*28, it's the combination of 6 different filtered layers. The second convolutional layer will have 16 filter matrices. Therefore, the second convolutional layer will have 16*10*10, it's the combination of 16 different filtered layers. Those two layers contain all the feature information. The next layer will extract the most important feature. It also called max pooling layer. We used zero padding in our question. As the discussion mentioned the formula for calculate the output width is below. In our question, the filter size is 5*5 and the padding is 0, the stride is 1. Make the first convolution as a example, we have 32*32 as the input width, 32-5+1=28. Therefore, our output wide is 28*28.

$$output\ wide = \frac{input_width - filter_{size} + 2 * padding}{stride} + 1 \quad (1)$$

3) The max pooling layer:

Max-pooling layer will reduce the dimension from the convolutional layer, there are two max-pooling layers. Each max-pooling layer are after each convolutional layer. In our project, the max-pooling layer will extract the greater value from the each 2*2 matrix. As I mentioned before, the layer extracts the most important feature information.

4) The activation function:

This activation function gives us the equation to determine the decision for non-linear situation. Because there is some non-linear situation in our problem. Activation function also make the decision layer from 0 to 1. It also called normalization. There is some example that mentioned in discussion, such as sigmoid, tanh, Relu and leaky Relu. Sigmoid will generate the number from 0 and 1. It takes a number and squashed a number between 0 and 1. It has the feature of sine. The gradient vanish from 0 to 1. Figure 1a shows the Sigmoid. Tanh will generate number between -1 and 1. The advantages of Tanh is that it converges faster. Figure 1b shows the Tanh. Relu and leaky Relu are similar to each other, but leaky Relu have some negative regions. Figure 1 c and Figure 1d show the Relu and leak Relu

5) The SoftMax layer:

From my perspective, I think SoftMax function layer also can be called decision layer, because this layer will give us the answer for the decision. This layer is from the second fully connected layer. It will give us a score from 0 to 1 for each component, there are 10 components. The highest score of those 10 components will be our result. The formula below is represent the standard SoftMax function.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Each function is discussed in each component. There are many different functions shows in the discussion. I will not show those duplicated formula in my report.

a-2

Overfitting means the model only good for the train data, but when we use same model to test other new data. It will have a really bad result. The also happened in many nonparametric systems, because those nonparametric system already define many parameters. When we use different data set, the parameter cannot be changed, the results can be bad in some situation. To avoid the overfitting problem, we have to resampling the data set, because when we re-sampling the data set, that mean we double the data set, it helps us to get a better result with larger data set. Another method is avoiding using non-parametric system. As I mentioned above, the nonparametric system will give us a worth case.

a-3

As we did last assignment with traditional computer vision method, like SIFT, SURF and bag-of-words, they have some many limitations. As we know the color of the object, may affect the result by using the bag-of words. The feature extraction and classification are not accuracy as the CNN process. It cannot update the parameter and steps of doing the traditional computer vision methods. But for the CNN process, it will update the parameter to learn those classification, it will use the known image to improve the algorithm.

The SIFT and SURF use the pixel value to extract the feature information, but the CNN use filters to do the convolution and max-pooling steps to extract the features. It's more accurate

for extract the important feature. CNN also use Gaussian noise to improve the extract features. There are many advantage of CNN to do the classification and recognition.

a-4

Loss function:

It is a function to calculate the different between the expect value and the actual value, it also called as cost function or criterion function, it helps us to determine how good our train or test data is. It also has a weight vector to calculate each data points, and sum those data point together, we will have a loss function. Usually, a small number of loss function have a better result.

Backpropagation:

It used to calculate the gradient of the loss function. It helps us to find a better parameter to get better scores. On other hand, backpropagation will help us to minimize the loss function to each input. As the discussion mentioned that, we can use those determination from the backpropagation to update the current parameters.

b-1

There are not huge different between each parameter. But there are some overfitting situations. As Figure b, d and e show the overfitting situation. As the epoch increase, the accuracy of the test data decreasing, but the accuracy of the train data still increasing. The batch size different effect the train data too much, but it will affect the test data. As we tried, when the batch size =256 and learning rate =0.01 has the best result. From my perspective of learning, I think if I have a larger learning rate and larger batch size, it will cause the overfitting, I think it's become the learning rate changes too big for the weight vector. The weight vector will change in the bad range. Therefore, the question has overfitting issue. The ways of avoiding the overfitting is run less epoch, and chose the highest test result with relatively epoch. Also the monument will not change to much according to the result

b-2

The best parameter setting is when batch size =256, learning rate =0.01. Figure 2e shows the result. But it has some overfitting issues. Therefore, I will choose when batch size=64, learning rate =0.001 and moment =0.9. Figure 2c shows the result.

c-1

I choose the article named "All you need is a good init" by Dmytro Mishkin and Jiri Matas. The article basically mentioned two steps. The first step is initializing the weight vector before each convolution process. The second step is normalizing each layer to normal distribution which make the variance equals 1. At first, he mentioned that they realized that the initializing weight vector is very important part of the machine learning. Figure 3a shows the weight vector and vs accurate in each iteration. As the article mentioned that the larger weight vector will cause the divergence. Small weight vector will update the data very slowly. Figure 3b shows the formula and the algorithm for generate the weight vector, it will use in non-linear process, such as max-out and tanh. And the formula cannot use it in the max-pooling layers. From the Figure 3b, we can see that we generate a weight vector in orthonormal way. There are two steps to update the wright vector. The first step is that multiple the weight vector with the Gaussian noise of unit variance. The second step is using the matrix to do a decomposition by using the SVD, then update the weight with SVD values. He also mentioned LSUV process (Layer-sequential unit-variance). It's uses for estimate the variance for each new matrix that come from convolution, and scaling the new weight vector

to become unit variance. As the author mentioned that we will do the same step as mini batch, but we add a orthonormal vector before the first convolution.

Next we will do the same step as mini-batch gradient descent, but each time, we will normalize the weight vector when we update the weight vector.

He also tries that multiple with Gaussian noise to our orthonormal weight vector, but the result is not good as the algorithm before.

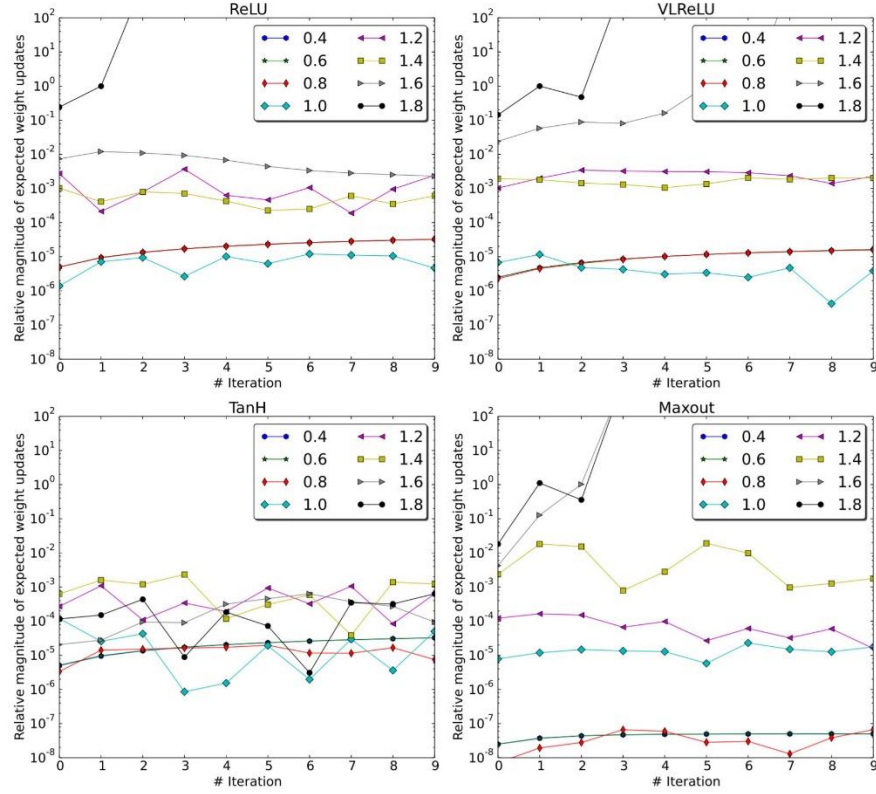


Figure 3a shows the weight vector and vs accurate in each iteration

Algorithm 1 Layer-sequential unit-variance orthogonal initialization. L – convolution or full-connected layer, W_L - its weights, B_L - its output blob, Tol_{var} - variance tolerance, T_i – current trial, T_{max} – max number of trials.

```

Pre-initialize network with orthonormal matrices as in Saxe et al. (2014)
for each layer  $L$  do
  while  $|Var(B_L) - 1.0| \geq Tol_{var}$  and  $(T_i < T_{max})$  do
    do Forward pass with a mini-batch
    calculate  $Var(B_L)$ 
     $W_L = W_L / \sqrt{Var(B_L)}$ 
  end while
end for

```

Figure 3b shows the formula and the algorithm for generate the weight vector

c-2

LeNet-5		Method from article	
Pro	Con	Pro	Con

Has bias in convolution layer	No Gaussian noise	Gaussian noise	Will not use it in non-linear process other than ReLU
Mini-batch gradient descent	Not normal distribution. (not zero mean and one standard deviation)	Normal distribution (unit variance)	Cannot use formula in max-pooling layers.
Simple steps. No need to normalize and no need for pro-process for the weight vector.	Linear process in connected layers	Has an orthonormal bias for convolution layers	Additional step of normalization and pro-process the weight vector.
Use the Relu and 5*5 filter.	Accuracy of test data around 60-75% (CIFAR-10)	High Accuracy of test data around 89 to 93.	
	Only did 50 epochs	Did 10 to 500 epochs	
		Try many different filter size and activation function	