VE 445 2019 Spring Lab 2 Convolution Neural Network

I. Introduction

In the last lab, you have implemented SVM and trained you own linear classifier in a rigorous mathematical perspective. However, you will implement another machine learning model, Convolution Neural Network in this lab. In lab 2, you will need to generate image identification models on two image data sets.

II. CNN

Convolution Neural Networks, also known as CNN, put up by Yann LeCun in 1989, is a specialized kind of neural network for processing data that has a known, grid-like topology structure. For more details, you can refer to the reference material on Canvas, chapter 7, 8, 9, *Deep Learning*.

1. Convolution layer.

   The convolution defined for 2-dimension space is different from the one for 1-dimension space. Given I as an image and K a matrix known as convolution kernel, the convolution between the image and the kernel is given by:

   $$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

   Using Convolution, we can extract features from the original image with different convolution kernels. The tensorflow module provides the function

   ```
   tensorflow.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=True,
                        data_format='NHWC', dilations=[1, 1, 1, 1], name=None)
   ```

   to generate the convolution layer in networks.

2. Activation function.

   Like BP neural networks, CNN requires activation function to increase the generalization ability. The common activation function includes sigmoid function, tanh function and ReLU function.

3. Dropout.

   A neural network with complicated structure and plenty of full-connected layers tends to overfit. To avoid of overfitting, we will use the dropout method to reduce the connection between layers. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. The details of dropout are tedious and hard to implement, but luckily, tensorflow provides a good function to apply dropout.

   ```
   tensorflow.nn.dropout(x, keep_prob=None, noise_shape=None, seed=None, name=None,
                         rate=None)
   ```

4. Pooling.

Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. There are many pooling method and you can refer to the tensorflow website for details.

5. One-hot.

   In order to describe the discrete label, we apply the method of one-hot to encode the label.

   For example:

```
# In the mnist datasets, there are 10 labels in total
# Label 0
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# Label 1
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
```

6. Softmax.

   Typically, the output result of the CNN represents for the probability of each label with one-hot encoding. A softmax output s can be obtained by the calculated output e.

$$s_i = \frac{e_i}{\sum_k e_k}$$

   For example, a softmax output for the MNIST data set is:

```
s = [0.9, 0.01, 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]
```

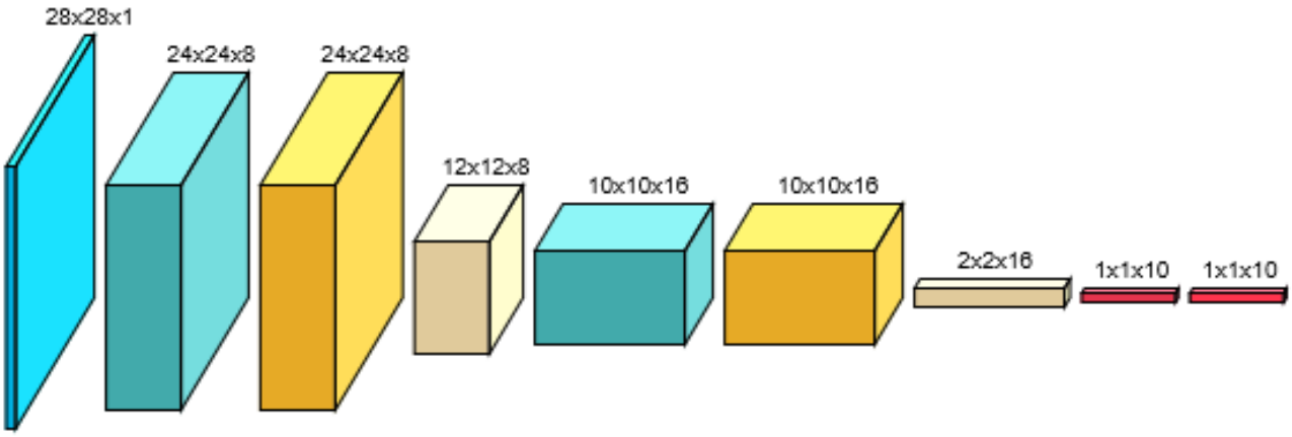
7. Optimization.

   For the training of the CNN, cross entropy is a good choice to be the loss function. For a batch of training labels $\mathbf{y}$, and softmax output $\mathbf{s}$, the cross entropy is given by:

$$L(\mathbf{y}, \mathbf{s}) = -\sum_n \sum_i y_{i,n} \log(s_{i,n})$$

For example, a CNN structure can be drew in the following form.

```
input(28, 28, 1)
conv(24, 24, 8)
relu(24, 24, 8)
pool(12, 12, 8)
conv(10, 10, 16)
relu(10, 10, 16)
pool(2, 2, 16)
fullyconn(1, 1, 10)
softmax(1, 1, 10)
```



The input is grayscale image with shape 28 x 28. And there are 8 5 x 5 convolution kernels applied to form the first convolution layer. The following is an activation layer with ReLU function. A pooling layer with 5 x 5 max pool is applied after the activation layer.

III. Data Sets

There will be two data sets needed in this lab. The first is the MNIST data set and the other is the CIFAR-10 data set. Both these two sets are picture set and you can import the data set directly from tensorflow.

```
mnist = tf.keras.datasets.mnist.load_data(path='mnist.npz')
# return: tuple of numpy arrays: (x_train, y_train), (x_test, y_test)
cifar = tf.keras.datasets.cifar10.load_data()
# return: tuple of numpy arrays: (x_train, y_train), (x_test, y_test)
```

The data of MNIST is in the form of 28x28 grayscale picture. For more information about the data set you can refer to http://yann.lecun.com/exdb/mnist/.

The CIFAR-10 dataset consists of 60000 32x32 RGB-colored images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. You can refer to https://www.cs.toronto.edu/~kriz/cifar.html to learn more about the data set.

IV. Model Training and Saving

Notice that there is no boundary conditions for deep learning model training, therefore you can use the optimizer provided by tensorflow directly for the loss function.

After finishing your training, you should save your model using tensorflow.train.Saver(). Only one model should be saved for each dataset. The model for MNIST data should be named as "**mnist_model**" and "**cifar_model**" for CIFAR_10.

V. Submission

Only one .zip file is expected to be submitted to the Canvas which contains your python file and a report.

The due is April 21st 11:59 pm. There is no due extension and late submission will not be accepted.