Project Guide

2018.10.13

[Session 1: Image Classification]

1. Project Goal

실습에 사용할 MNIST는 28x28 사이즈의 작은 흑백 이미지로, 0-9까지의 손으로 쓴 숫자 데이터 셋이다. CNN을 설계하고 학습시켜 손으로 쓴 숫자를 인식하는 classifier를 학습시키는 것이 본 프로젝트의 목표이다. Deep Learning library인 Tensorflow를 사용하여 진행되며, 학습에 필요한 기본적인 코드를 제공한다. 실습의 주 내용은 다양한 구조의 딥러닝 아키텍처를 직접 설계하고 구현하는 것으로, 기본적인 형태의 모듈 (convolution layer, relu layer, pooling layer 등)을 제공하고, 주어진 간단한 구조의 네트워크를 구현하는 것일 일차적인 목표로 한다. 그 이후에는 classifier의 인식 성능 향상을 위해 다양한 구조를 직접 설계하고 구현하여 학습시킨다.

2. Image Classification – SIIT_KAIST_CLS

코드는 데이터를 제외하고 총 2개의 python 스크립트로 이루어져 있다 (main.py 와 model.py).

[model.py]

학습하고자 하는 네트워크의 구체적인 Architecture와 Cost function, Constraints 및 Optimizer 등이 정의되어 있는 스크립트. 네트워크의 구조를 변화시키거나, 학습시키는 방법 등 학습에 필요한 변수들을 조절할 수 있다. 실습에서는 주로 model.py를 수정하여 네트워크의 구조를 변화시키고 그 성능을 관찰한다.

[main.py]

학습에 사용할 데이터셋과 model.py에서 정의한 네트워크 등을 불러들여서 실제로 학습을 진행하기 위한 스크립트. Epoch, learning rate, batch size 등의 hyper parameter를 조절할 수 있다.

새로운 모델을 학습시키기 위해서는 train.sh 파일을, 학습된 모델을 불러와 테스트하기 위해서는 test.sh 파일을 실행시킨다.

$ sh train.sh $ sh test.sh

(1) MNIST dataset

위 이미지들은 실습에 사용될 MNIST hand written digit database의 일부들이다. MNIST는 위와 같은 숫자들을 인식하기 위한 데이터셋으로 학습데이터 5만장, 테스트 데이터 1만장으로, 총 6만 장으로 구성되어있다.

(2) Defined Functions

네트워크 구축을 위해 필요한 함수는 모두 model.py에 정의되어 있다. 본 프로젝트에 사용될 함수들은 Convolutional layer, ReLU layer, Batch Normalization layer, Pooling layer 등으로 이루어져 있다.

Convolution layer

```
def _conv(self, name, x, filter_size, in_filters, out_filters, strides):
    with tf.variable_scope(name):
        n = filter_size * filter_size * out_filters
        w = tf.get_variable(
            'weight/DW', [filter_size, filter_size, in_filters, out_filters],
            tf.float32, initializer=tf.uniform_unit_scaling_initializer(factor=2.0))

        y = tf.nn.conv2d(x, w, strides, padding='SAME')

    return y
```

ReLU layer

```
def _relu(self, x, leakiness=0.0):
```

```
    return tf.maximum(x, leakiness*x)
```

Global Average Pooling layer

```
def _global_avg_pool(self, x):
    assert x.get_shape().ndims == 4
    return tf.reduce_mean(x, [1, 2])
```

Batch Normalization layer

```
def _batch_norm(self, name, x):
    with tf.variable_scope(name):
      params_shape = [x.get_shape()[-1]]

      beta = tf.get_variable(
          'beta', params_shape, tf.float32,
          initializer=tf.constant_initializer(0.0, tf.float32))
      gamma = tf.get_variable(
          'gamma', params_shape, tf.float32,
          initializer=tf.constant_initializer(1.0, tf.float32))

      if self.mode == 'train':
```

```
        mean, variance = tf.nn.moments(x, [0, 1, 2], name='moments')



        moving_mean = tf.get_variable(
            'moving_mean', params_shape, tf.float32,
            initializer=tf.constant_initializer(0.0, tf.float32),
            trainable=False)
        moving_variance = tf.get_variable(
            'moving_variance', params_shape, tf.float32,
            initializer=tf.constant_initializer(1.0, tf.float32),
            trainable=False)
```

위의 레이어들을 모듈처럼 활용하여 서로 쌓거나 연결하여 하나의 딥 네트워크 구조를 만들 수 있다. 예를 들면 다음과 같다.

Example Network

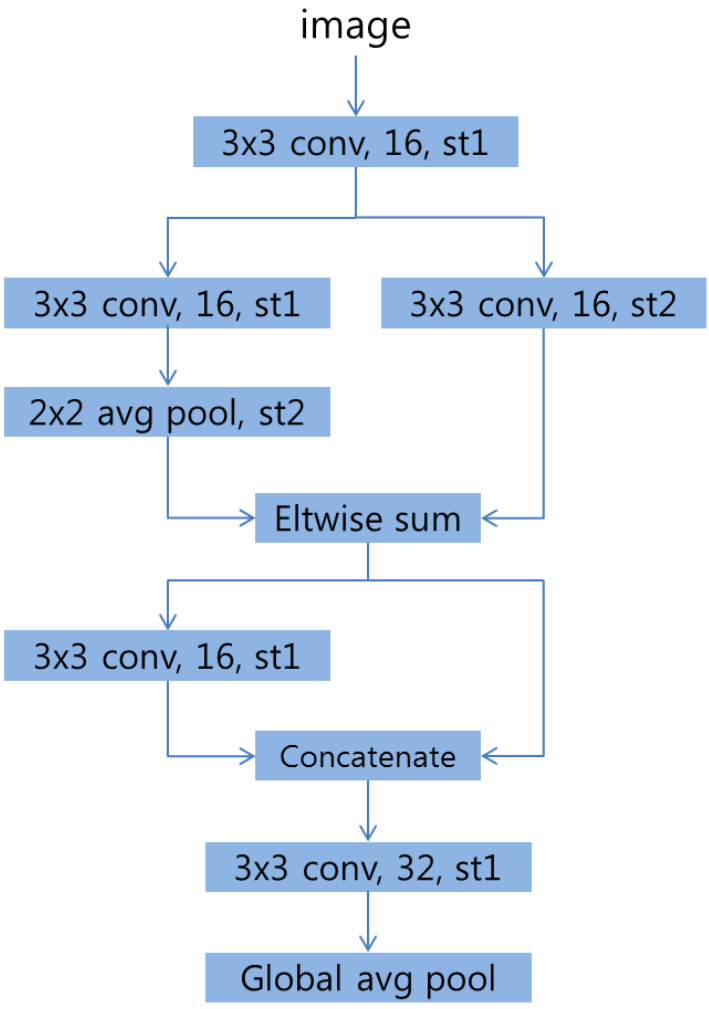
```
def _ex_net(self, x):
    x = self._conv('conv1', images, 3, 1, 16, 1)
    x = self._batch_norm('bn1', x)
    x = self._relu(x, 0.)
    x = self._conv('conv2', x, 3, 16, 16, 1)
    x = self._batch_norm('bn2', x)
    x = self._relu(x, 0.)
    x = tf.nn.avg_pool(x, 2, 2, 'VALID')
```

```
    return x
```

위의 예시는 2개의 convolution layer를 쌓았고, batch normalization을 사용하였다. 그 위에 2x2 average pooling을 쌓았으며, activation function으로는 relu를 사용하였다. 위의 예시에는 드러나지 않으나 average pooling위에 1개의 fully connected layer를 추가로 쌓아서 output node를 class의 개수와 일치시킨다. (model.py의 build_graph() 함수 참조)

(3) TODO: Simple network

본 실습에서 구현해야 하는 구조는 다음 그림과 같다. model.py 스크립트에서 simple_network 함수의 빈칸에 아래와 같은 구조의 네트워크를 구현해 본다.



Eltwise sum이란 element-wise summation으로 두 개의 텐서의 단순 합이다. 실제 코드에서 구현할 때에는 간단하게 x+y로 구현하면 된다. Concatenate는 두 개의 텐서를 정해진 축에 대하여 이

어 붙이는 operation으로, tf.concat() 함수를 사용하며, 실습에서 axis는 3을 사용한다. 자세한 사용법은 tensorflow API에서 검색할 수 있다.

네트워크의 구조를 정의한 뒤에는 터미널에서

$ sh train.sh

를 실행하여 학습이 진행되는 과정과 성능을 확인할 수 있다. 학습이 진행되는 과정 중에는 learning rate (Lr)와 학습중인 현재의 training loss (Loss), 학습 데이터에 대한 인식률 (Acc)을 확인할 수 있다. 또한 정기적으로 테스트 데이터에 대한 성능도 확인할 수 있다.

그림에서 모든 Convolutional layer는 Batch Normalization layer와 ReLU layer와 함께 쓰인다 (CONV-BN-RELU).

(4) TODO: Be creative!

model.py 스크립트에는 VGGnet과 Residual network가 정의되어 있다. 위의 네트워크를 구현하였다면, VGGnet과 Residual network를 참고하여 창의적인 네트워크를 직접 설계하여 구현해 보고, 직접 학습시켜 구조에 따른 성능 변화를 확인해본다.

새로운 네트워크의 구조는 DIY_network() 함수 아래에 정의하고, build_graph()에서

feats = self.simple_network(self.images) 를 주석처리하고, 그 윗줄을 주석 해제함으로써 학습시킬 수 있다. (feats = self.DIY_network(self.images))

3. 보고서

보고서는 별도의 포맷, 분량 제한을 두지 않으며 아래의 내용을 포함한다.

- Discriminator network code
- Train operation for discriminator network