CNN_week9

October 23, 2019

IST597 :- Implementing CNN from scratch Week 9 Tutorial
Author:- aam35

In [0]:
```
import tensorflow as tf
import numpy as np
import time
import tensorflow.contrib.eager as tfe
tf.enable_eager_execution()
tf.executing_eagerly()
seed = 1234
tf.random.set_random_seed(seed=seed)
np.random.seed(seed)
```

In [5]:
```
from tensorflow.examples.tutorials.mnist import input_data
data = input_data.read_data_sets("/tmp/data/", one_hot=True, reshape=False)
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

In [0]:
```
batch_size = 64
hidden_size = 100
learning_rate = 0.01
output_size = 10
```

In [0]:
```
class CNN(object):
  def __init__(self,hidden_size,output_size,device=None):
      filter_h, filter_w, filter_c , filter_n = 5 ,5 ,1 ,30
      self.W1 = tf.Variable(tf.random_normal([filter_h, filter_w, filter_c, filter_n], s
      self.b1 = tf.Variable(tf.zeros([filter_n]),dtype=tf.float32)
      self.W2 = tf.Variable(tf.random_normal([14*14*filter_n, hidden_size], stddev=0.01)
      self.b2 = tf.Variable(tf.zeros([hidden_size]),dtype=tf.float32)
      self.W3 = tf.Variable(tf.random_normal([hidden_size, output_size], stddev=0.01))
      self.b3 = tf.Variable(tf.zeros([output_size]),dtype=tf.float32)
      self.variables = [self.W1,self.b1, self.W2, self.b2, self.W3, self.b3]
      self.device = device
```

```
        self.size_output = output_size

    def flatten(self,X, window_h, window_w, window_c, out_h, out_w, stride=1, padding=0):

        X_padded = tf.pad(X, [[0,0], [padding, padding], [padding, padding], [0,0]])

        windows = []
        for y in range(out_h):
            for x in range(out_w):
                window = tf.slice(X_padded, [0, y*stride, x*stride, 0], [-1, window_h, win
                windows.append(window)
        stacked = tf.stack(windows) # shape : [out_h, out_w, n, filter_h, filter_w, c]

        return tf.reshape(stacked, [-1, window_c*window_w*window_h])

    def convolution(self,X, W, b, padding, stride):
        n, h, w, c = map(lambda d: d.value, X.get_shape())
        #print(X.get_shape())
        #print(data.train.images.get_shape())
        filter_h, filter_w, filter_c, filter_n = [d.value for d in W.get_shape()]

        out_h = (h + 2*padding - filter_h)//stride + 1
        out_w = (w + 2*padding - filter_w)//stride + 1

        X_flat = self.flatten(X, filter_h, filter_w, filter_c, out_h, out_w, stride, paddi
        W_flat = tf.reshape(W, [filter_h*filter_w*filter_c, filter_n])

        z = tf.matmul(X_flat, W_flat) + b     # b: 1 X filter_n

        return tf.transpose(tf.reshape(z, [out_h, out_w, n, filter_n]), [2, 0, 1, 3])



    def relu(self,X):
        return tf.maximum(X, tf.zeros_like(X))

    def max_pool(self,X, pool_h, pool_w, padding, stride):
        n, h, w, c = [d.value for d in X.get_shape()]

        out_h = (h + 2*padding - pool_h)//stride + 1
        out_w = (w + 2*padding - pool_w)//stride + 1

        X_flat = self.flatten(X, pool_h, pool_w, c, out_h, out_w, stride, padding)

        pool = tf.reduce_max(tf.reshape(X_flat, [out_h, out_w, n, pool_h*pool_w, c]), axis
        return tf.transpose(pool, [2, 0, 1, 3])
```

```
def affine(self,X, W, b):
    n = X.get_shape()[0].value # number of samples
    X_flat = tf.reshape(X, [n, -1])
    return tf.matmul(X_flat, W) + b

def softmax(self,X):
    X_centered = X - tf.reduce_max(X) # to avoid overflow
    X_exp = tf.exp(X_centered)
    exp_sum = tf.reduce_sum(X_exp, axis=1)
    return tf.transpose(tf.transpose(X_exp) / exp_sum)


def cross_entropy_error(self,yhat, y):
    return -tf.reduce_mean(tf.log(tf.reduce_sum(yhat * y, axis=1)))


def forward(self,X):
    if self.device is not None:
      with tf.device('gpu:0' if self.device == 'gpu' else 'cpu'):
        self.y = self.compute_output(X)
    else:
      self.y = self.compute_output(X)

    return self.y


def loss(self, y_pred, y_true):
    '''
    y_pred - Tensor of shape (batch_size, size_output)
    y_true - Tensor of shape (batch_size, size_output)
    '''
    y_true_tf = tf.cast(tf.reshape(y_true, (-1, self.size_output)), dtype=tf.float32)
    y_pred_tf = tf.cast(y_pred, dtype=tf.float32)
    return tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=y_pred_tf,


def backward(self, X_train, y_train):
    """
    backward pass
    """
    # optimizer
    # Test with SGD,Adam, RMSProp
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    #predicted = self.forward(X_train)
    #current_loss = self.loss(predicted, y_train)
    #optimizer.minimize(current_loss, self.variables)

    #optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
```

```
        with tf.GradientTape() as tape:
            predicted = self.forward(X_train)
            current_loss = self.loss(predicted, y_train)
        #print(predicted)
        #print(current_loss)
        #current_loss_tf = tf.cast(current_loss, dtype=tf.float32)
        grads = tape.gradient(current_loss, self.variables)
        optimizer.apply_gradients(zip(grads, self.variables),
                                  global_step=tf.train.get_or_create_global_step())


    def compute_output(self,X):
        conv_layer1 = self.convolution(X, self.W1, self.b1, padding=2, stride=1)
        conv_activation = self.relu(conv_layer1)
        conv_pool = self.max_pool(conv_activation, pool_h=2, pool_w=2, padding=0, stride=2
        conv_affine =self.affine(conv_pool, self.W2,self.b2)
        conv_affine_activation = self.relu(conv_affine)

        conv_affine_1 = self.affine(conv_affine_activation, self.W3, self.b3)
        return conv_affine_1
```

In [0]:
```
def accuracy_function(yhat,true_y):
    correct_prediction = tf.equal(tf.argmax(yhat, 1), tf.argmax(true_y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    return accuracy
```

In [9]:
```
# Initialize model using GPU
mlp_on_cpu = CNN(hidden_size,output_size, device='gpu')

num_epochs = 4
train_x =  tf.convert_to_tensor(data.train.images)
train_y = tf.convert_to_tensor(data.train.labels)
time_start = time.time()
num_train = 55000
z= 0

for epoch in range(num_epochs):
        train_ds = tf.data.Dataset.from_tensor_slices((data.train.images, data.train.lab
           .shuffle(buffer_size=1000)\
           .batch(batch_size=batch_size)
        loss_total = tf.Variable(0, dtype=tf.float32)
        accuracy_total = tf.Variable(0, dtype=tf.float32)
        for inputs, outputs in train_ds:
```

```
                preds = mlp_on_cpu.forward(inputs)
                loss_total = loss_total + mlp_on_cpu.loss(preds, outputs)
#                 accuracy_train = accuracy_function(preds,outputs)
#                 accuracy_total = accuracy_total + accuracy_train
                mlp_on_cpu.backward(inputs, outputs)
                #print(z)
                #z = z+ 1
            print('Number of Epoch = {} - loss:= {:.4f}'.format(epoch + 1, loss_total.numpy(
            preds = mlp_on_cpu.compute_output(train_x)
            accuracy_train = accuracy_function(preds,train_y)

            accuracy_train = accuracy_train * 100
            print ("Training Accuracy = {}".format(accuracy_train.numpy()))


#         preds_val = mlp_on_cpu.compute_output(data.validation.images)
#         accuracy_val = accuracy_function(preds_val,data.validation.labels)
#         accuracy_val = accuracy_val * 100
#         print ("Validation Accuracy = {}".format(accuracy_val.numpy()))

        #test accuracy
        test_x =  tf.convert_to_tensor(data.test.images)
        test_y = tf.convert_to_tensor(data.test.labels)
        preds_test = mlp_on_cpu.compute_output(test_x)
        accuracy_test = accuracy_function(preds_test,test_y)
        # To keep sizes compatible with model
        accuracy_test = accuracy_test * 100
        print ("Test Accuracy = {}".format(accuracy_test.numpy()))


        # time_taken = time.time() - time_start
        # print('\nTotal time taken (in seconds): {:.2f}'.format(time_taken))
        # #For per epoch_time = Total_Time / Number_of_epochs
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/data/util/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Number of Epoch = 1 - loss:= 0.0360
Training Accuracy = 11.234545707702637
Number of Epoch = 2 - loss:= 0.0274
Training Accuracy = 82.1927261352539
Number of Epoch = 3 - loss:= 0.0068
Training Accuracy = 88.91090393066406
Number of Epoch = 4 - loss:= 0.0053
Training Accuracy = 90.74727630615234
Test Accuracy = 91.50999450683594
```