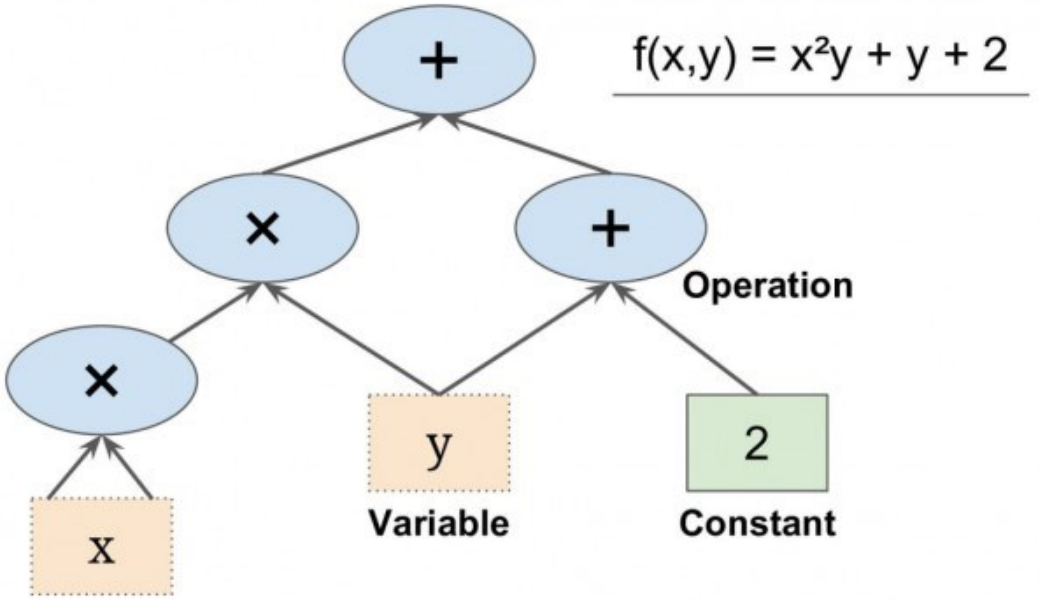
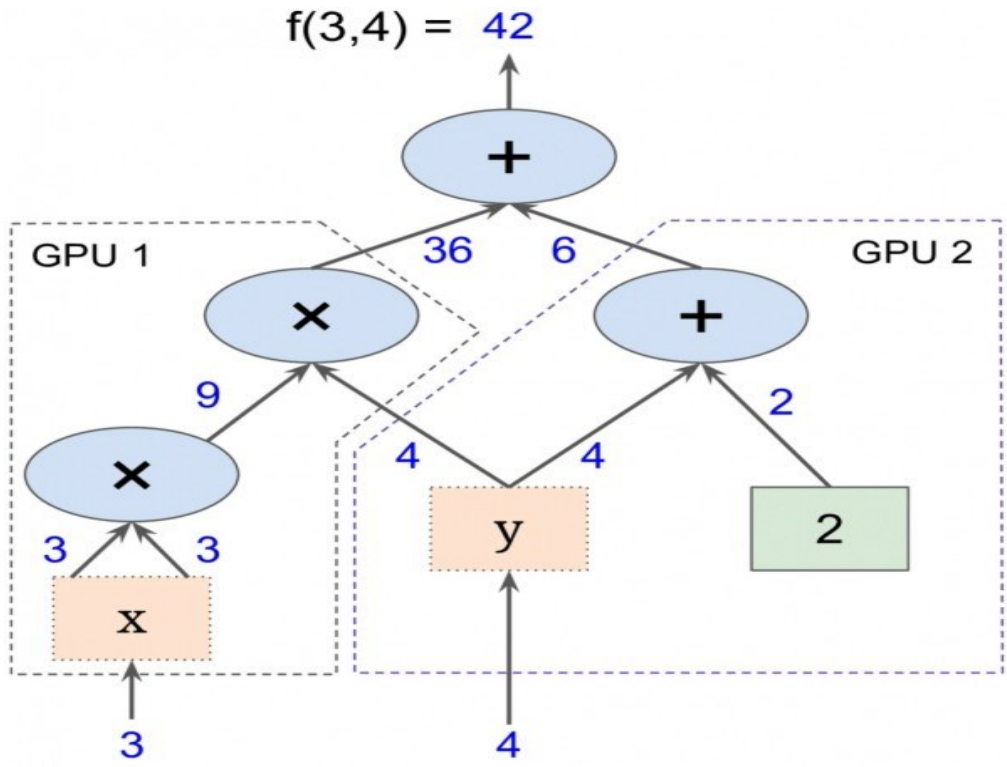
مقدمه

TensorFlow یک کتابخانه نرم افزاری منبع باز قدرتمند برای محاسبات عددی است، که به طور خاص برای یادگیری ماشین در مقیاس بزرگ، ارائه و تنظیم شده است. اصل اساسی آن ساده است: شما در ابتدا درپایتون یک نمودار از محاسباتی که باید انجام شود، تعریف می کنید (برای مثال شکل زیر)، و سپس تنسورفلو آن نمودار را می گیرد و آن را با استفاده از کد بهینه سازی شده C++ اجرا می کند.



مهمتر از همه این که می توان نمودار را به چند تکه شکست و آنها را به صورت موازی بر روی چند CPU یا GPU اجرا کرد. تنسورفلو از محاسبات توزیع شده نیز پشتیبانی میکند به طوری که شما می توانید با تقسیم محاسبات در صدها سرور، شبکه های عصبی عظیم را روی مجموعه های آموزش بسیار بزرگ در مدت زمانی معقول آموزش دهید. TensorFlow می تواند یک شبکه با میلیون ها پارامتر را روی یک مجموعه آموزشی متشکل از میلیاردها نمونه هر یک با میلیونها ویژگی، آموزش دهد. این جای تعجب ندارد، چرا

که TensorFlow توسط تیم Google Brain توسعه داده شده و از قدرت بسیاری از سرویسهای بزرگ مقیاس گوگل، مانندGoogle Cloud Speech ، Google Photos و Google Search بهره می‌برد.



هنگامی که TensorFlow در ماه نوامبر سال ۲۰۱۵ منبع باز اعلام شد، تقریبا تعداد قابل توجهی کتابخانه معروف منبع باز برای یادگیری عمیق (شکل بالا) وجود داشت و با نگاهی عادلانه، بسیاری از ویژگی های تنسورفلو پیش از این در یک کتابخانه و یا دیگری وجود داشته است.

با این وجود، طراحی تمیز TensorFlow، مقیاس پذیری، انعطاف‌پذیری و اسناد جامع و کامل آن (بدون در نظر گرفتن نام گوگل) به سرعت آن را به بالای لیست رساند.

ساختار تمرین

در این تمرین ما شبکه های مختلف را پیاده سازی می کنیم و خروجی هر کدام را به شکل نمودارهای مختلف نشان می دهیم.

قبل از هر چیز ما از یک تابع استفاده می کنیم که بررسی رسم نمودارهای مختلف که بتوانیم شبکه های را بررسی کنیم. این تابع در فایل visualizations با نام losses_accuracies_plots قرار دارد که نمودارهای مورد نظر را برای بررسی رسم می کند. پارامترهای ورودی تابع مربوط به خطا و دقت زمان آموزش و خطلا و دقت زمان تست و عنوان نمودار و مرحله می باشد.

تابع رسم میزان خطا و دقت برای حالت آموزش و تست

افزودن کتابخانه مورد نظر برای رسم

```
import matplotlib.pyplot as plt

def losses_accuracies_plots(train_losses, train_acc, test_losses,
test_acc,plot_title="Loss, train acc, test acc",step=100):

    training_iters = len(train_losses)
    # iters_steps
    iter_steps = [step *k for k in range(training_iters)]

    imh = plt.figure(1, figsize=(15, 14), dpi=160)
    # imh.tight_layout()
    # imh.subplots_adjust(top=0.88)

    final_acc = test_acc[-1]
    img_title = "{}, test acc={:.4f}".format(plot_title,final_acc)
    imh.suptitle(img_title)
    plt.subplot(221)
    #plt.plot(iter_steps,losses, '-g', label='Loss')
    plt.semilogy(iter_steps, train_losses, '-g', label='Trn Loss')
    plt.title('Train Loss ')
    plt.subplot(222)
    plt.plot(iter_steps, train_acc, '-r', label='Trn Acc')
    plt.title('Train Accuracy')
    plt.subplot(223)
    plt.semilogy(iter_steps, test_losses, '-g', label='Tst Loss')
    plt.title('Test Loss')
    plt.subplot(224)
    plt.plot(iter_steps, test_acc, '-r', label='Tst Acc')
    plt.title('Test Accuracy')
    plt.subplots_adjust(top=0.88)
    plot_file = "./plots/{}.png".format(plot_title.replace(" ","_"))
    plt.savefig(plot_file)
    plt.show()
```

نمودار ۲

ساخت یک شبکه پنج لایه با ۳ لایه Convolution

در ابتدا یک شبکه یک لایه را پیاده سازی می کنیم. با ساختار زیر که در آن لایه اول با ورودی با اندازه ۷۸۴ و ,

و سه لایه کانالوشن با اندازه ۴ و ۸ و ۱۶ قرار داده شده است و لایه بعد با اندازه ۲۵۶ قرار دارد.

ساختار لایه ها

```
5 layer neural network with 3 convolution layers, input layer 28*28= 784, output
10 (10 digits)
Output labels uses one-hot encoding
input layer               - X[batch, 784]
1 conv. layer             - W1[5,5,,1,C1] + b1[C1]
                            Y1[batch, 28, 28, C1]

2 conv. layer             - W2[3, 3, C1, C2] + b2[C2]
2.1 max pooling filter 2x2, stride 2 - down sample the input (rescale input by 2)
28x28-> 14x14
                            Y2[batch, 14,14,C2]
3 conv. layer             - W3[3, 3, C2, C3]  + b3[C3]
3.1 max pooling filter 2x2, stride 2 - down sample the input (rescale input by 2)
14x14-> 7x7
                            Y3[batch, 7, 7, C3]
4 fully connecteed layer  - W4[7*7*C3, FC4]   + b4[FC4]
                            Y4[batch, FC4]
5 output layer            - W5[FC4, 10]   + b5[10]
One-hot encoded labels      Y5[batch, 10]
```

سورس کد mnist_3.0_3layer_convnet.py

اضافه کردن کتابخانه تنسورفلو و دیتاست mnist برای کار

```
import tensorflow as tf
from tensorflow.contrib.learn.python.learn.datasets.mnist import
read_data_sets
import visualizations as vis

NUM_ITERS=5000

DISPLAY_STEP=100
BATCH=100

tf.set_random_seed(0)
```

بارگذاری داده های mnist برای شروع کار

```
mnist = read_data_sets("./input_data", one_hot=True, reshape=False,
validation_size=0)
```

ساخت مدل یک لایه که شامل ۷۸۴ ورودی می باشد یک تسنور ۴ بعدی در ابتدا ساخته شده است که براساس این که عکس های به صورتgrayscale با اندازه ۲۴*۲۴ می باشند اندازه وردی ۷۸۴ در نظر گرفته شده است و این تک لایه دارای ۱۰ خروجی می باشد.

x مقدار ورودی را نشان می دهد

```
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
```

y تعداد خروجی ها را نشان می دهد.

```
Y_ = tf.placeholder(tf.float32, [None, 10])
```

```
pkeep = tf.placeholder(tf.float32)
```

اندازه لایه ها را در این مرحله تعیین می کنیم.

```
C1 = 4  # first convolutional layer output depth
C2 = 8  # second convolutional layer output depth
C3 = 16 # third convolutional layer output depth

FC4 = 256  # fully connected layer
```

وزن ها از توزیع نرمالبا میانگین صفر و انحراف معیار ۰.۱ مقدار دهی اولیه می شوند

W ها تعداد وزن های این پنج لایه را نشان می دهد

B هاهم تعداد بایاس این پنج لایه را را نشان می دهد

```
# 5x5 conv. window, 1 input channel (gray images), C1 - outputs
W1 = tf.Variable(tf.truncated_normal([5, 5, 1, C1], stddev=0.1))
b1 = tf.Variable(tf.truncated_normal([C1], stddev=0.1))
# 3x3 conv. window, C1 input channels(output from previous conv. layer ), C2 - outputs
W2 = tf.Variable(tf.truncated_normal([3, 3, C1, C2], stddev=0.1))
b2 = tf.Variable(tf.truncated_normal([C2], stddev=0.1))
# 3x3 conv. window, C2 input channels(output from previous conv. layer ), C3 - outputs
W3 = tf.Variable(tf.truncated_normal([3, 3, C2, C3], stddev=0.1))
b3 = tf.Variable(tf.truncated_normal([C3], stddev=0.1))
# fully connected layer, we have to reshpe previous output to one dim,
# we have two max pool operation in our network design, so our initial size 28x28 will
be reduced 2*2=4
# each max poll will reduce size by factor of 2
W4 = tf.Variable(tf.truncated_normal([7*7*C3, FC4], stddev=0.1))
```

```
b4 = tf.Variable(tf.truncated_normal([FC4], stddev=0.1))

# output softmax layer (10 digits)
W5 = tf.Variable(tf.truncated_normal([FC4, 10], stddev=0.1))
b5 = tf.Variable(tf.truncated_normal([10], stddev=0.1))
```

مقدار ورودی X را در یک بردار قرار می دهیم.

```
XX = tf.reshape(X, [-1, 784])
```

حالا مدل خود را تعریف می کنیم.

```
stride = 1   # اندازه خروجی

Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, stride, stride, 1],
padding='SAME') + b1)

k = 2 # max pool filter size and stride, will reduce input by factor
of 2
Y2 = tf.nn.relu(tf.nn.conv2d(Y1, W2, strides=[1, stride, stride, 1],
padding='SAME') + b2)
Y2 = tf.nn.max_pool(Y2, ksize=[1, k, k, 1], strides=[1, k, k, 1],
padding='SAME')

Y3 = tf.nn.relu(tf.nn.conv2d(Y2, W3, strides=[1, stride, stride, 1],
padding='SAME') + b3)
Y3 = tf.nn.max_pool(Y3, ksize=[1, k, k, 1], strides=[1, k, k, 1],
padding='SAME')

# reshape the output from the third convolution for the fully
connected layer
YY = tf.reshape(Y3, shape=[-1, 7 * 7 * C3])

Y4 = tf.nn.relu(tf.matmul(YY, W4) + b4)
#Y4 = tf.nn.dropout(Y4, pkeep)
Ylogits = tf.matmul(Y4, W5) + b5
Y = tf.nn.softmax(Ylogits)
```

تابع تعیین میزان خطاcross-entropy در لایه را طراحی می کنیم که به ازای هر ۱۰۰ تصویر که معادل اندازه batch ما هست خطا را در پارامترهای بدست آمده اصلاح می کند

```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=Ylogits, labels=Y_)
cross_entropy = tf.reduce_mean(cross_entropy)*100
```

دقت مدل آموزش دیده بین ۰ (بدترین) و ۱ (بهترین)

```
correct_prediction = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

نرخ یادگیری را در مرحله آموزش تعیین می کنیم. که مقدار 0.003 را داده ایم

```
learning_rate = 0.003
train_step = train_step =
tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

allweights = tf.concat([tf.reshape(W1, [-1]), tf.reshape(W2, [-1]),
tf.reshape(W3, [-1]), tf.reshape(W4, [-1]), tf.reshape(W5, [-1])], 0)
allbiases  = tf.concat([tf.reshape(b1, [-1]), tf.reshape(b2, [-1]),
tf.reshape(b3, [-1]), tf.reshape(b4, [-1]), tf.reshape(b5, [-1])], 0)
```

مقدار دهی اولیه را انجام می دهیم. برای مقادیری که در آخر می خواهیم در روی نمودارها رسم کنیم و همین طور مقدار تنسورها را با مقادیر اولیه پر میکنیم

```
init = tf.global_variables_initializer()

train_losses = list()
train_acc = list()
test_losses = list()
test_acc = list()
```

در این مرحله آموزش شروع می شود.

```
saver = tf.train.Saver()
```

داده های برای رسم گراف را در این مرحله بدست می آوریم

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)


    for i in range(NUM_ITERS+1):
```

آموزش روی دسته ها انجام می شود. که تعداد دسته ها ۱۰۰ تصویر می باشد پس از هر ۱۰۰ تصویر میزان خطا بررسی و در پارامترها اصلاح می شود.

```
        batch_X, batch_Y = mnist.train.next_batch(BATCH)
```

```
        if i%DISPLAY_STEP ==0:
```

محاسبه مقدار پارامترها و خطا و دقت در این مرحله آموزش را ذخیره می کنیم برای رسم بر روی نمودار

```
                acc_trn, loss_trn, w, b = sess.run([accuracy,
cross_entropy, allweights, allbiases], feed_dict={X: batch_X, Y_:
batch_Y, pkeep: 1.0})
```

محاسبه مقدار پارامترها و خطا و دقت در این مرحله تست را ذخیره می کنیم برای رسم بر روی نمودار

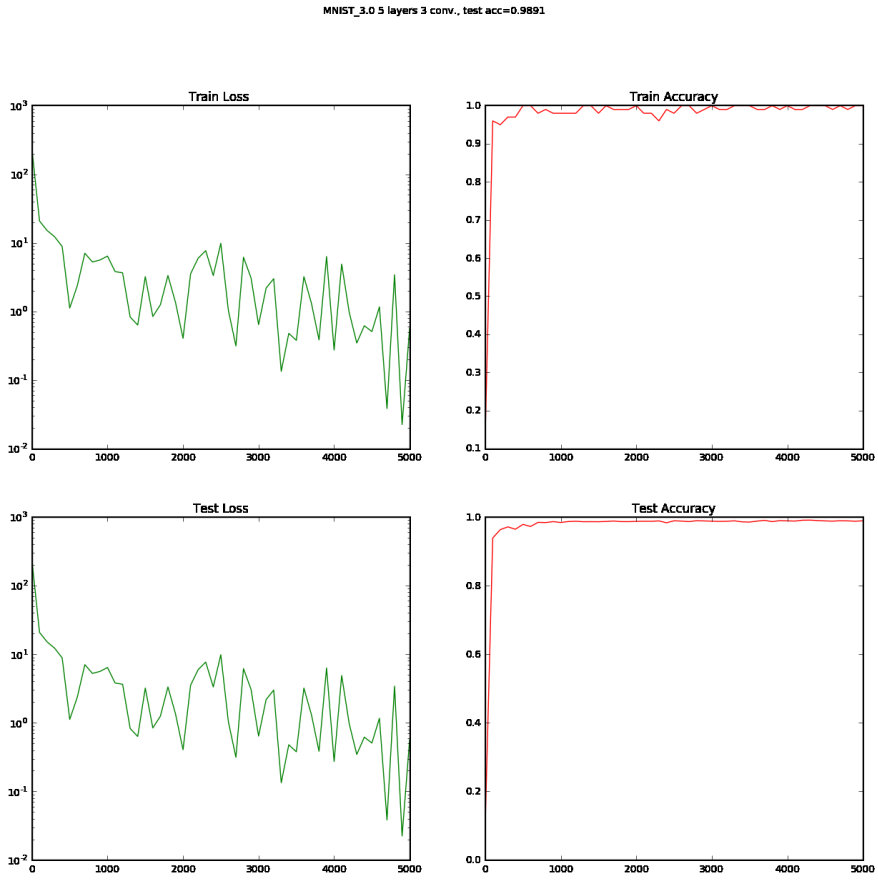
```
acc_tst, loss_tst = sess.run([accuracy, cross_entropy], feed_dict={X:
mnist.test.images, Y_ : mnist.test.labels, pkeep: 1.0})

            print("#{} Trn acc={} , Trn loss={} Tst acc={} , Tst
loss={}".format(i,acc_trn,loss_trn,acc_tst,loss_tst))

            train_losses.append(loss_trn)
            train_acc.append(acc_trn)
            test_losses.append(loss_trn)
            test_acc.append(acc_tst)

        # the backpropagationn training step
        sess.run(train_step, feed_dict={X: batch_X, Y_: batch_Y, pkeep: 0.75})
```

در این مرحله نمودارها رسم می شوند.

```
title = " MNIST 2.2 5 layers relu adam  dropout
 "
vs.losses_accuracies_plots(train_losses,train_acc,test_losses,
test_acc,title,DISPLAY_STEP)
```

خروجی

خروجی مدل با دقت ۰.۹۸۹۰ بدست آمده است. همان طور که در نمودار ۱-۱ می بینید ما میزان خطا و دقت در دو حالت آموزش و تست را رسم کرده ایم نمودار های سطر اول مربوط به آموزش و سطر دوم مربوط به مرحله تست می باشد. میزان خطا خطی به رنگ سبز و میزان دقت با رنگ قرمز نشان داده شده است.


MNIST_3.0 5 layers 3 conv., test acc=0.9891
Train Loss
Train Accuracy
Test Loss
Test Accuracy

نمودار ۵

نتیجه گیری

من در ابتدا با کدهای زیادی کار کردم و منابع زیادی رو از سایت های مختلف گرفتم و روش های مختلفی هم پیاده سازی کردم با تابع Relu و Tainh و Sigmoid و Softmax پیاده سازی کردم که نتیجه های مختلف بدست آمد و نتیجه خروجی از ترکیبات روش های بالا از بهترین ها بودن که بهترین روش کانولوشن بود که در آخرین روش پیاده سازی شد به عنوان کارکرد فایل های دیگر رو هم در کنار سورس های این گزارش قرار داده ام ما می توانیم با تغییر توابع روش های بالا و حتی میزان تغییر هایپر پارامترها مدل های مختلفی را بدست بیاوریم تعداد اندازه نرون ها و تغییر تابع فعالیت و غیر را می توانیم تغییر بدهیم و یا تعداد دسته ها و نرخ یادگیری تا به مدل های مختلف با دقت های بیشتر برسیم.

منبع

https://github.com/martin-gorner/tensorflow-mnist-tutorial

https://github.com/ksopyla/tensorflow-mnist-convnets

https://medium.com/tensorist/classifying-fashion-articles-using-tensorflow-fashion-mnist-f22e8a04728a

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/mnist

https://www.oreilly.com/learning/not-another-mnist-tutorial-with-tensorflow

پیامبر خدا (ص):

العِلمُ خَزائنُ ومَفاتِيحُهُ السُّؤالُ ، فَاسألُوا رَحِمَكُمُ اللّه فإنّهُ يُؤجَرُ أربَعةٌ : السائلُ ، والمُتَكلِّمُ ، والمُستَمِعُ ، والمُحِبُّ لَهُم.

دانش گنجینه هایی اســت و کــلــیدهای آن پرسش است ؛ پس ، خدایتان رحمت کند ، بپرسید ، که با این کار چهار نفر اجر می یابند : پرسشگر ، پاسخگو ، شنونده و دوستدار آنان.

(تحف العقول : ۴۱ منتخب میزان الحکمهٔ : ۲۶۰)