custom-model-tf2

August 20, 2019

1 Example of custom training in TF2

[1]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2 Build training data

[2]:
```
n = 100
TRUE_W = 3.0
TRUE_b = 2.0

# random samples from normal distribution
np.random.seed(1)
r = np.random.normal(loc=0, scale=0.5, size=n)

# build data
inputs = np.random.normal(loc=0, scale=0.5, size=n)
outputs = TRUE_W * inputs + TRUE_b + r
```

3 Define model

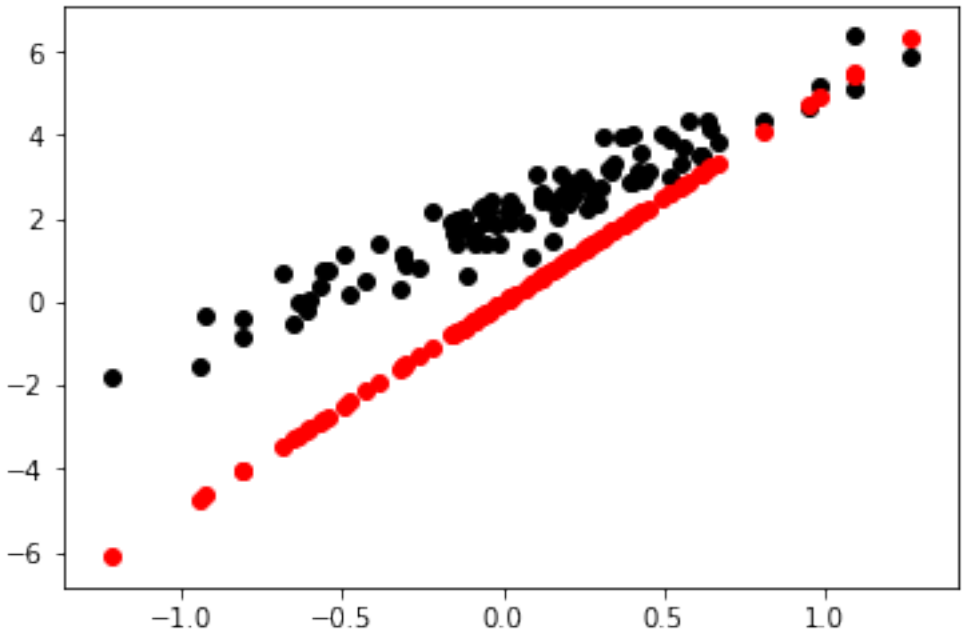
[3]:
```
class Model(object):
    def __init__(self):
        # Initialize the weights to `5.0` and the bias to `0.0`
        # In practice, these should be initialized to random values (for␣
 ↪example, with `tf.random.normal`)
        self.W = tf.Variable(5.0)
        self.b = tf.Variable(0.0)

    def __call__(self, x):
        return self.W * x + self.b

model = Model()
```

[4]:
```
def plot_model(pred):
    plt.scatter(inputs, outputs, c='black')
    plt.scatter(inputs, pred, c='r')
    plt.show()

predictions = model(inputs)
plot_model(predictions)
```



4 Train model

[5]:
```
N_EPOCHS = 10
LEARNING_RATE = 0.4

def loss(predicted_y, target_y):
    return tf.reduce_mean(tf.square(predicted_y - target_y))

def train(model, inputs, outputs, learning_rate):
    # gradient descent
    with tf.GradientTape() as t:
        current_loss = loss(model(inputs), outputs)
    dW, db = t.gradient(current_loss, [model.W, model.b])
    model.W.assign_sub(learning_rate * dW)
    model.b.assign_sub(learning_rate * db)
```

[6]:
```
Ws, bs = [], []
epochs = range(N_EPOCHS)
for epoch in epochs:

    # get variables and loss before training of each epoch
    Ws.append(model.W.numpy())
    bs.append(model.b.numpy())

    # train
    train(model, inputs, outputs, learning_rate=LEARNING_RATE)

# plot the evolution of trainable variables
plt.plot(epochs, Ws, 'r',
         epochs, bs, 'b')
plt.plot([TRUE_W] * len(epochs), 'r--',
         [TRUE_b] * len(epochs), 'b--')
plt.legend(['W', 'b', 'True W', 'True b'])
plt.show()
```