tf2-custom

August 22, 2019

1 Example of TF-2 custom model

[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

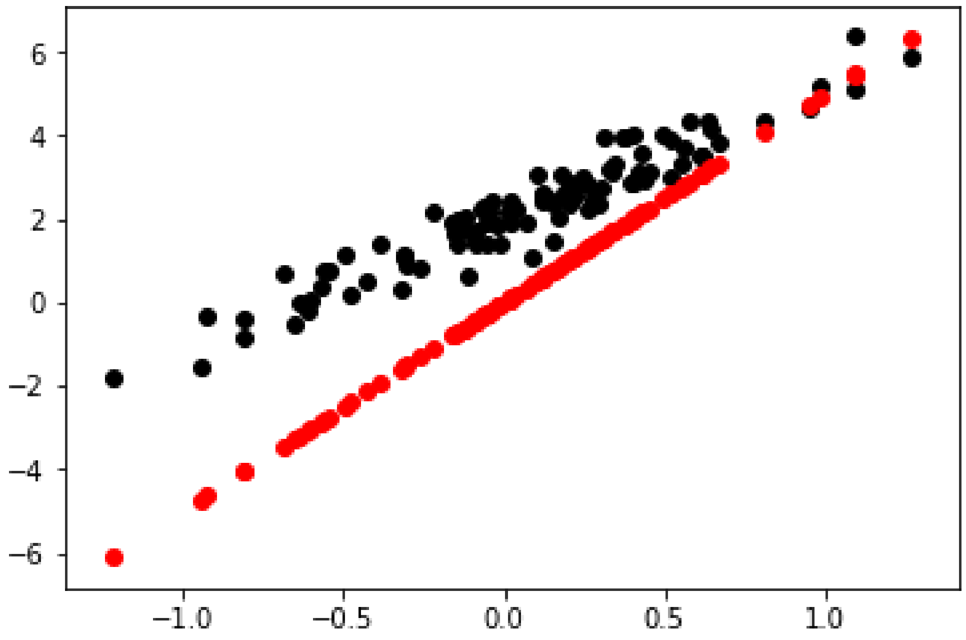
3.1 Construct model

[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()
```

3.2 Show model prediction before training

[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

4.1 Configure training

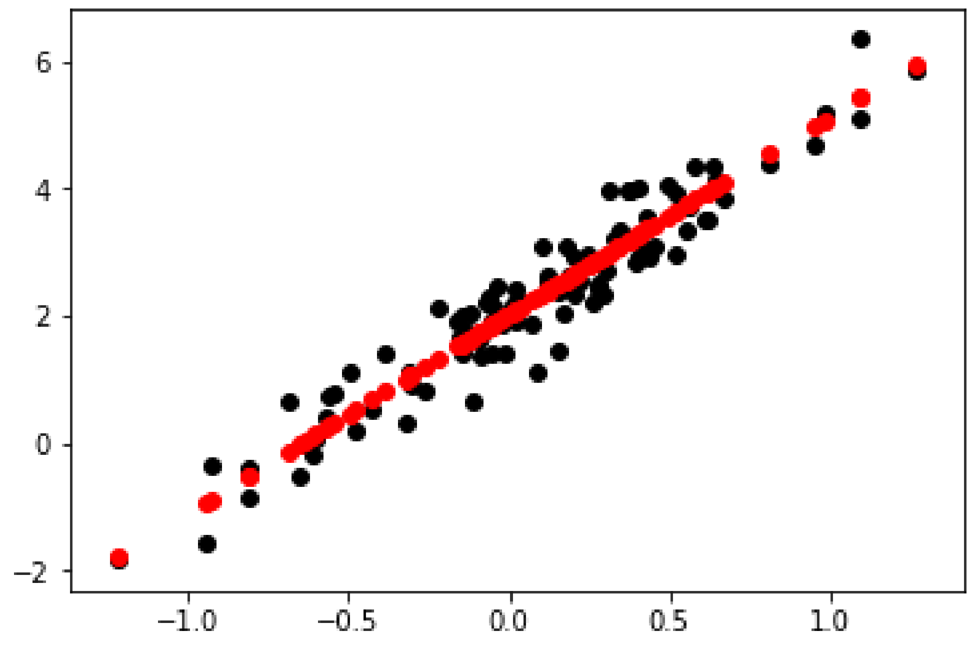
[5]:
```
N_EPOCHS = 100
LEARNING_RATE = 0.1

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)
```

4.2 Execute training

[6]:
```
train(model, inputs, outputs, learning_rate=LEARNING_RATE, epochs=N_EPOCHS)
```

4.3 Show model prediction after training

[7]:
```
new_predictions = model(inputs)
plot_model(new_predictions)
```


6
4
2
0
−2
−1.0
−0.5
0.0
0.5
1.0