كلية الحوسبة والمعلوماتية |
College of Computing and Informatics
الجامعة السعودية الإلكترونية
SAUDI ELECTRONIC UNIVERSITY
2011-1432

Second Semester – 2021/2022

Course Code	DS660
Course Name	Deep Learning Techniques
Assignment type	Critical Thinking
Module	05
Total Points	105 Points

Student ID	G200007615
Student Name	Abdulaziz Aqlumayzi
CRN	21604

Solutions:

Critical Thinking Assignment 1

Acceleration by Hybridization

Introduction

In this activity, we will describe and explain how to give acceleration by hybridization, followed by a Python programming sample that demonstrates this concept.

Definition

Before delving into hybridization, let us first understand two key ideas in computing performance: symbolic programming and imperative programming.

Symbolic programming is faster and easier to transfer. Symbolic programming allows for simpler code optimization during compilation, as well as the option to convert the program into a Python-independent format. This allows the application to execute in a non-Python environment, avoiding any potential Python interpreter performance difficulties.

Imperative programming is less difficult. When imperative programming is employed in Python, the majority of the code is simple and straightforward. Imperative programming code is also easier to debug. This is due to the ease with which all important intermediate variable values can be obtained and printed, as well as the usage of Python's built-in debugging tools.

Acceleration by Hybridization

To highlight the benefit of compilation, we compare the time required to assess net(x) before and after hybridization. First, let us construct a class to quantify this time. It will come in helpful when we measure performance throughout the chapter.

Python Programming Code

Consider deep networks with numerous layers to gain an understanding of how hybridization works. Let's have a look as how we can handle this for substantial chunks of the code by replacing get net() with tf.function (). To begin, we define a basic MLP.

```
import tensorflow as tf
from tensorflow.keras.layers import Dense
from d2l import tensorflow as d2l

def get_net():
    net = tf.keras.Sequential()
    net.add(Dense(256, input_shape = (512,), activation = "relu"))
    net.add(Dense(128, activation = "relu"))
    net.add(Dense(2, activation = "linear"))
    return net
```

To demonstrate the performance improvement gained by compilation we compare the time needed to evaluate net(x) before and after hybridization. Let us define a class to measure this time first.

```
class Benchmark:
    """For measuring running time."""
    def __init__(self, description='Done'):
        self.description = description

    def __enter__(self):
        self.timer = d2l.Timer()
        return self

    def __exit__(self, *args):
        print(f'{self.description}: {self.timer.stop():.4f} sec')

net = get_net()
with Benchmark('Eager Mode'):
    for i in range(1000): net(x)

net = tf.function(net)
with Benchmark('Graph Mode'):
    for i in range(1000): net(x)
```

Program result:

```
Eager Mode: 0.8230 sec
Graph Mode: 0.3731 sec
```

References

Zhang, A., Lipton, Z., Li, M., Smola, A., Werness, B., Hu, R., Zhang, S., & Tay, Y. (2022). *Dive into Deep Learning*. https://d2l.ai/index.html.