DS620 Machine Learning and Deep Learning

HOS08 TensorFlow Low-level

05/21/2021 Developed by Minh Nguyen

05/21/2021 Reviewed by Shanshan Yu

School of Technology & Computing @City University of Seattle (CityU)

Learning objectives

- Lower-level Python API
- TensorFlow Keras
- TensorFlow Data API
- Custom model and training algorithms

Resources

- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.
- Paul, S. (2018, September 12). *Investigating Tensors with PyTorch*. Datacamp. https://www.datacamp.com/community/tutorials/investigating-tensors-pytorch
- Gadosey, P. (2019, August 19). *A beginner's guide to NumPy with Sigmoid, ReLu and Softmax activation functions*. Medium. https://medium.com/ai%C2%B3-theory-practice-business/a-beginners-guide-to-numpy-with-sigmoid-relu-and-softmax-activation-functions-25b840a9a272

Introduction

So far, what we've learned from TensorFlow happens at high-level APIs. Most of the operations were implemented with a single line of code such as passing a value to the function's parameter (e.g., loss function, activation function, optimizer, regularizer, weight initializer, etc.). Up until now, most of these algorithms have been a black box and we haven't had the flexibility to customize these algorithms on our own. In this HOS, we will dive deeper into TensorFlow, dissecting its component and learn how to build these algorithms on our own. Some of the code in this exercise can be found in Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.

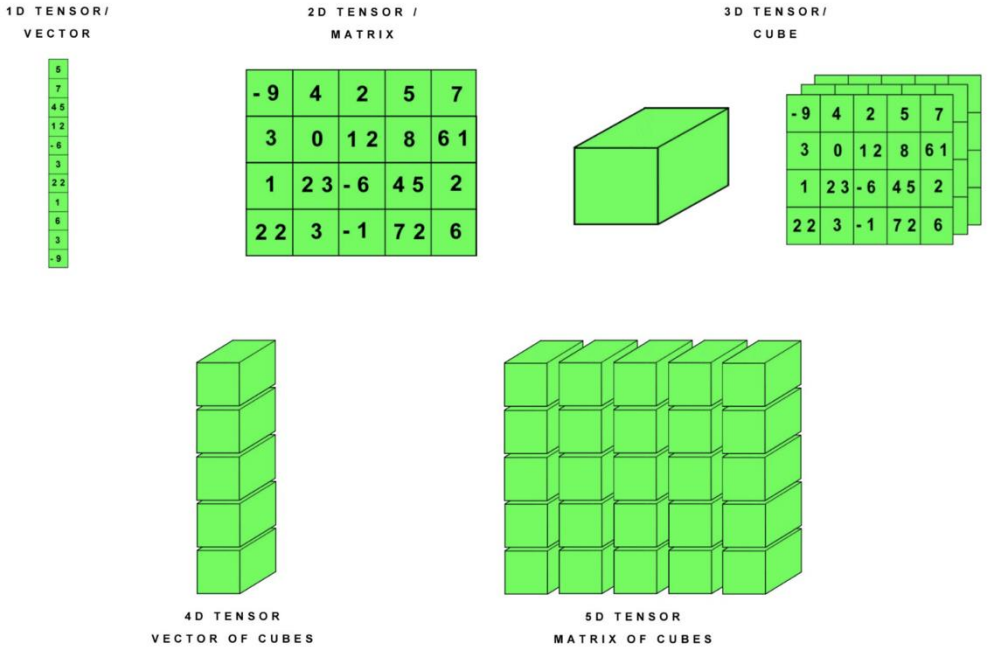
Preparing development environment

1. From Google Colab, create a new notebook, name it "Tensorflow.ipynb"
2. Type the following codes to import libraries.

```
import tensorflow as tf
```

1. Tensor

Tensor is the fundamental data structure that power TensorFlow. It's so crucial that its name is a part of TensorFlow itself. So, what is a tensor? You might already know what it is but under a different name – multi-dimensional array. Similar to the array, a tensor contains numerical values, the number of dimensions can be as many as you set it to be.



Type the following code to Tensorflow.ipynb

- We will start with building a matrix tensor. A tensor works exactly like a NumPy array. In fact, you can convert a tensor into a NumPy array.

Tensor

```
#create a tensor
matrix = tf.constant([[1,2,3],[4,5,6]])

matrix
```

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[1, 2, 3],
       [4, 5, 6]], dtype=int32)>
```

```
print(matrix.numpy())
print(matrix.shape)
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
```

- Tensor arithmetic operations is very similar to NumPy as well

Tensor arithmetic

```
# with a constant
matrix2 = matrix + 2
matrix2
```

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[3, 4, 5],
       [6, 7, 8]], dtype=int32)>
```

```
# with another tensor with the same shape
matrix + matrix2
```

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[ 4,  6,  8],
       [10, 12, 14]], dtype=int32)>
```

2. Custom Activation Functions

The activation function decides whether to turn on or off a neuron. Its name is inspired by how the neuron in our brain is "activated" by different stimuli. It takes in the weighted sum and bias of the previous layer and output a value.

Type the following code to Tensorflow.ipynb

- Below are the most common activation functions: relu, sigmoid, softmax and softplus.

Activation function

```
#relu
def my_relu(x):
    return tf.maximum(0,x)

#sigmoid
def my_sigmoid(x):
    return 1/(1+tf.exp(-x))

#softmax
def my_softmax(x):
    expo = tf.exp(x)
    expo_sum = tf.sum(tf.exp(x))
    return expo/expo_sum

#softplus
def my_softplus(x):
    return tf.math.log(tf.exp(x) + 1.0)
```

3. Custom Loss Functions

A quick revision on loss function, it's the way for the model to remedy the model. The more different the predicted value is from the true value, the larger the output of the loss function is, this gives the model a target value to minimize during the training process. The two variables it comprises of are the true value and the predicted value.

Type the following code to Tensorflow.ipynb

- Here an example of mean squared error, the most common regression loss function. It comprises only of 2 inputs, the predicted value and the true value.

Loss function

```
# mean squared error
def mse(y_true, y_pred):
    error = y_true - y_pred
    squared_error = tf.square(error)
    return tf.mean(squared_error)
```

- When the function requires a hyperparameter, which is an input other than the predicted value and the true value, the object-oriented approach is required as this ensure that the hyperparameter will be saved. An advantage of using this approach is that you can inherit directly from the parent class from Keras.

```python
# OOP approach required if function has a hyper parameter
class HuberLoss(tf.keras.losses.Loss):
    def __init__(self, threshold=1.0, **kwargs):
        self.threshold = threshold
        super().__init__(**kwargs)
    def call(self, y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) < self.threshold
        squared_loss = tf.square(error) / 2
        linear_loss = self.threshold * tf.abs(error) - self.threshold**2 / 2
        return tf.where(is_small_error, squared_loss, linear_loss)
    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "threshold": self.threshold}
```

4. Custom Regularizer

When training a dataset, the model will learn the underlying pattern of the sample dataset in hope that it can estimate the true pattern of the population. However, when the model learns too much from the pattern of the training data set, it risks missing the true pattern of the population thus result in an overfitted model because not all training data is a good estimate of the population. The goal of regularization is to reduce the learning of the model so that it won't overfit. This is done by reducing the weight of the model by a factor.

Type the following code to Tensorflow.ipynb

- Here's an implementation of the Functional programming approach. As mentioned above, the factor won't be saved using this approach

```python
# Functional Programming approach
def my_l1_regularizer(weight, factor = 0.01):
    return tf.reduce_sum(tf.abs(factor * weight))

def my_l2_regularizer(weight, factor = 0.01):
    return tf.reduce_sum(tf.square(factor * weight))
```

- Here's an example of the OOP approach

```python
# OOP approach
class MyL1Regularizer(tf.keras.regularizers.Regularizer):
    def __init__(self, factor):
        self.factor = factor
    def __call__(self, weight):
        return tf.reduce_sum(tf.abs(self.factor * weight))
    def get_config(self):
        return {"factor": self.factor}
```

5. Custom Layer

Sometime, the situation may require you to build your own layer. You can only create a layer using the Object-Oriented Approach as there are hyperparameters that need to be saved before using the layer.

Type the following code to Tensorflow.ipynb

- This is a simplified version of the Dense layer

```
class MyDense(tf.keras.layers.Layer):
    def __init__(self, units, activation = None, **kwargs):
        super().__init__(**kwargs)
        self.units = units
        self.activation = tf.keras.activation.get(activation)

    def build(self, batch_input_shape):
        self.kernel = self.add_weight(
            name ="kernel", shape=[batch_input_shape[-1], self.units],
            initializer="glorot_normal")
        self.bias = self.add_weight(
            name="bias", shape=[self.units], initializer="zeros")
        super().build(batch_input_shape)

    def call(self, X):
        return self.activation(X @ self.kernel + self.bias)

    def compute_output_shape(self, batch_input_shape):
        return tf.TensorShape(batch_input_shape.as_list()[:-1] + [self.units])

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "units": self.units,
                "activation": tf.keras.activations.serialize(self.activation)}
```

Push Your Work to GitHub

Download the notebook from Colab:

File -> Tensorflow.ipynb

Move the downloaded file into your **Module8** working folder.

Open terminal and Navigate to the GitHub folder of this week HOS.

Make sure the assignment files on the subfolder Module8 of hos08a_YouGithubUserName folder, enter the following command to upload your work:

>>>> git add .

>>>> git commit -m “Submission for HOS08”

>>>> git push origin master