< Deep Learning - PART2 TF2 CNNs >

Ch 5. CNNs Workshop 1- MNIST : Digit Recognition with Batch Normalization

2021/10/01

[Reference]

1. TensorFlow Core - Tutorials : **`TensorFlow 2 quickstart for experts`** , 2019. https://www.tensorflow.org/tutorials/quickstart/advanced (https://www.tensorflow.org/tutorials/quickstart/advanced)
2. TensorFlow Core - Tutorials : **`Get started with TensorBoard`** , 2019. https://www.tensorflow.org/tensorboard/get_started (https://www.tensorflow.org/tensorboard/get_started)
3. IT邦幫忙, **`Day 17: Tensorflow 2.0 再造訪 tf.GradientTape`** , 2019/10/02. https://ithelp.ithome.com.tw/articles/10223779 (https://ithelp.ithome.com.tw/articles/10223779)
4. TensorFlow Core - Guides : **`Better performance with tf.function and AutoGraph`** https://www.tensorflow.org/guide/function (https://www.tensorflow.org/guide/function)
5. TensorFlow Core - API : **`tf.keras.layers.BatchNormalization`** https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization (https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization)
6. François Chollet, **Deep Learning with Python**, Section 7.3.1 (**BATCH NORMALIZATION**), pp. 260~261, Manning, 2018. http://www.deeplearningitalia.com/wp-content/uploads/2017/12/Dropbox_Chollet.pdf# (http://www.deeplearningitalia.com/wp-content/uploads/2017/12/Dropbox_Chollet.pdf)

In [1]:

```
from __future__ import absolute_import, division, print_function, unicode_l

import tensorflow as tf
import datetime

from tensorflow.keras.layers import Dense, Flatten, Conv2D, BatchNormalizat
from tensorflow.keras import Model
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.4.1'
```

1. Load and prepare the `MNIST` dataset

- **`MNIST`** dataset - http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/)

In [3]:

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Checking the data info...

In [4]:

```
x_train.shape, y_train.shape
```

Out[4]:

```
((60000, 28, 28), (60000,))
```

In [5]:

```
x_test.shape, y_test.shape
```

Out[5]:

```
((10000, 28, 28), (10000,))
```

In [6]:

```
y_train    #  y_test
```

Out[6]:

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

In [7]:

```
x_train.min(), x_train.max()    # x_test.min(), x_test.max()
```

Out[7]:

```
(0, 255)
```

Normalization

In [8]:

```
1 x_train, x_test = x_train / 255.0, x_test / 255.0
```

In [9]:

```
1 x_train.min(), x_train.max()
```

Out[9]:

```
(0.0, 1.0)
```

Changing the data format into 4D Tensors for CNN's Inputs

- **4D Tensor : (samples, height, width, channels)**

In [10]:

```
1 # Add a "channels" dimension
2 x_train = x_train[..., tf.newaxis]
3 x_test = x_test[..., tf.newaxis]
```

In [11]:

```
1 x_train.shape  # x_test.shape
```

Out[11]:

```
(60000, 28, 28, 1)
```

Using `tf.data` to batch and shuffle the dataset:

- **`tf.data.Dataset`** : https://www.tensorflow.org/api_docs/python/tf/data/Dataset (https://www.tensorflow.org/api_docs/python/tf/data/Dataset)

> Class **Dataset**
>
> - Represents a potentially large set of elements.

- **Use `tf.data` to batch and shuffle the dataset**:

> For example :
>
> ```
> train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train))  .shuffle(10000).batch(32)
> ```

> [NOTE]:

- `tf.data.Dataset.from_tensor_slices` - https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_tensor_slices (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_tensor_slices)
- `shuffle()` - https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#shuffle (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#shuffle)
- `batch()` - https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#batch (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#batch)

In [12]:

```
# Nodes in the Computation Graph...
train_ds = tf.data.Dataset.from_tensor_slices(
    (x_train, y_train)).shuffle(10000).batch(32)

test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

In [13]:

```
train_ds
```

Out[13]:

```
<BatchDataset shapes: ((None, 28, 28, 1), (None,)), types: (tf.fl
oat64, tf.uint8)>
```

2. Forward Propagation

- Build the `tf.keras` model using the Keras model subclassing API (https://www.tensorflow.org/guide/keras#model_subclassing):

In [14]:

```
class MyModel(Model):   # tf.keras.Model class
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(32, 3, activation='relu')  # tf.keras.layers.Co
        self.batchnorm1 = BatchNormalization()   # tf.keras.layers.BatchNor
        self.flatten = Flatten()                 # tf.keras.layers.Flatten(
        self.d1 = Dense(128, activation='relu')  # tf.keras.layers.Dense()
        self.batchnorm2 = BatchNormalization()   # tf.keras.layers.BatchNor
        self.d2 = Dense(10)

    def call(self, x):
        x = self.conv1(x)
        x = self.batchnorm1(x)
        x = self.flatten(x)
        x = self.d1(x)
        x = self.batchnorm2(x)
        return self.d2(x)

# Create an instance of the model
model = MyModel()
```

In [15]:

```
model
```

Out[15]:

```
<__main__.MyModel at 0x222e8c81e08>
```

3. Backpropagation

Choosing an optimizer and loss function for training

[Loss Function] : `tf.keras.losses.SparseCategoricalCrossentropy()` :
https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy
(https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentrop

- **Class `SparseCategoricalCrossentropy`** - Computes the crossentropy loss between the labels and predictions.
  - **Use this crossentropy loss function when there are two or more label classes.**
  - **We expect labels to be provided as integers.**

[NOTE] : + **If you want to provide labels using `one-hot` representation, please use `CategoricalCrossentropy` loss.**

- There should be `# classes` floating point values per feature for `y_pred` and a single floating point value per feature for `y_true` .

In [16]:

```
#  < tf.keras.losses.SparseCategoricalCrossentropy >
#    from_logits: Whether y_pred is expected to be a logits tensor.
#                 By default, we assume that y_pred encodes
#                 a probability distribution.
#   [Note]: Using from_logits=True may be more numerically stable.

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tru

optimizer = tf.keras.optimizers.Adam()
```

Selecting metrics to measure the loss and the accuracy of the model

- These metrics accumulate the values over epochs and then print the overall result.
- Module: **`tf.keras.metrics`** https://www.tensorflow.org/api_docs/python/tf/keras/metrics (https://www.tensorflow.org/api_docs/python/tf/keras/metrics)

Built-in metrics classes : (*to name a few*)

- **`class SparseCategoricalAccuracy`** : Calculates how often predictions matches integer labels. https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAc (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAc
- **`class CategoricalAccuracy`** : Calculates how often predictions matches labels.
- **`class Mean`** : Computes the (weighted) mean of the given values.

In [17]:

```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_acc

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accur
```

Building A Training Model with `tf.function` & `tf.GradientTape`

- Using `tf.function` decorator & `tf.GradientTape` method to build a custom-made training model instead of `tf.keras` 's `model.fit()` .
- **`tf.GradientTape()`** (https://www.tensorflow.org/api_docs/python/tf/GradientTape (https://www.tensorflow.org/api_docs/python/tf/GradientTape))
  - When training with methods such as **`tf.GradientTape()`** , we can use **`tf.summary`** to log the required information.

> - `tf.GradientTape()` 可以用在 training loop 裡，記錄並建構正向傳播的計算圖。
> - 在完成"記錄"後，tf.GradientTape() 的 tape 物件則呼叫 gradient()方法，並傳入損失值 (loss score) 和模型可訓練的參數。 [from **`Ref 3.`**]
> - 一旦計算出了梯度後，立即呼叫 optimizer.apply_gradients() 方法，傳入一個 list of tuple，每一個 tuple 的第二個則是參數變數，而第一個變數為針對該參數所計算出的梯度。 [from **`Ref 3.`**]

In [18]:

```
@tf.function
def train_step(images, labels):    # images : x_train , labels : y_train
    ## ---------------------------------------------------------------------
    ##  Forward propagation -
    ##    tf.GradientTape()可以用在 training loop 裡，記錄並建構正向傳播的計算
    ## ---------------------------------------------------------------------
    with tf.GradientTape() as tape:
        # training=True is only needed if there are layers with different
        # behavior during training versus inference (e.g. Dropout).
        predictions = model(images, training=True)
        loss = loss_object(labels, predictions)

    ## ---------------------------------------------------------------------
    ##  Backpropagation -
    ##    在完成"記錄"後，tf.GradientTape() 的 tape 物件則呼叫 gradient()方法
    ##    並傳入損失值 (loss score) 和模型可訓練的參數。 [from Ref 3.]
    ## ---------------------------------------------------------------------
    gradients = tape.gradient(loss, model.trainable_variables)

    ## ---------------------------------------------------------------------
    ##  Parameters' update -
    ##    一旦計算出了梯度後，立即呼叫 optimizer.apply_gradients() 方法，
    ##    傳入一個 list of tuple，每一個 tuple 的第二個則是參數變數，
    ##    而第一個變數為針對該參數所計算出的梯度。 [from Ref 3.]
    ## ---------------------------------------------------------------------
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    train_loss(loss)
    train_accuracy(labels, predictions)
```

Testing the model :

In [19]:

```
@tf.function
def test_step(images, labels):
    # training=False is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    predictions = model(images, training=False)
    t_loss = loss_object(labels, predictions)

    test_loss(t_loss)
    test_accuracy(labels, predictions)
```

4. TensorBoard with `tf.summary`

- Set up summary writers to write the summaries to disk in a different logs directory:

In [20]:

```
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

log_directory = 'logs/CNN_MNIST_BN/'
train_log_dir = log_directory + current_time + '/train'
test_log_dir = log_directory + current_time + '/test'

train_summary_writer = tf.summary.create_file_writer(train_log_dir)
test_summary_writer = tf.summary.create_file_writer(test_log_dir)
```

> **[NOTE] :**

- **Use `tf.summary.scalar()` to log metrics (loss and accuracy) during training/testing within the scope of the summary writers to write the summaries to disk.**
- You have control over which metrics to log and how often to do it.
- **Other `tf.summary` functions enable logging other types of data.**

5. Training & Testing Processes

In [21]:

```
EPOCHS = 10

for epoch in range(EPOCHS):
    # Reset the metrics at the start of the next epoch
    train_loss.reset_states()
    train_accuracy.reset_states()
    test_loss.reset_states()
    test_accuracy.reset_states()

    for images, labels in train_ds:
        train_step(images, labels)

    with train_summary_writer.as_default():
        tf.summary.scalar('loss', train_loss.result(), step=epoch)
        tf.summary.scalar('accuracy', train_accuracy.result(), step=epoch)

    for test_images, test_labels in test_ds:
        test_step(test_images, test_labels)

    with test_summary_writer.as_default():
        tf.summary.scalar('loss', test_loss.result(), step=epoch)
        tf.summary.scalar('accuracy', test_accuracy.result(), step=epoch)

    template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accur
    print(template.format(epoch+1,
                          train_loss.result(),
                          train_accuracy.result()*100,
                          test_loss.result(),
                          test_accuracy.result()*100))
```

```
WARNING:tensorflow:Layer my_model is casting an input tensor from
dtype float64 to the layer's dtype of float32, which is new behav
ior in TensorFlow 2.  The layer has dtype float32 because it's dt
ype defaults to floatx.

If you intended to run this layer in float32, you can safely igno
re this warning. If in doubt, this warning is likely only an issu
e if you are porting a TensorFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float64 by default, call `tf.k
eras.backend.set_floatx('float64')`. To change just this layer, p
ass dtype='float64' to the layer constructor. If you are the auth
or of this layer, you can disable autocasting by passing autocast
=False to the base Layer constructor.

Epoch 1, Loss: 0.13132937252521515, Accuracy: 96.10833740234375,
Test Loss: 0.07257194817066193, Test Accuracy: 97.72999572753906
Epoch 2, Loss: 0.047159213572740555, Accuracy: 98.54999542236328,
Test Loss: 0.05519557371735573, Test Accuracy: 98.22999572753906
Epoch 3, Loss: 0.027102164924144745, Accuracy: 99.1383285522461,
Test Loss: 0.0529349260032177, Test Accuracy: 98.30999755859375
Epoch 4, Loss: 0.017519904300570488, Accuracy: 99.42832946777344,
Test Loss: 0.06894338876008987, Test Accuracy: 98.20999908447266
Epoch 5, Loss: 0.012101971544325352, Accuracy: 99.61500549316406,
```

```
Test Loss: 0.06786699593067169, Test Accuracy: 98.29999542236328
Epoch 6, Loss: 0.011082613840699196, Accuracy: 99.62166595458984,
Test Loss: 0.06433113664388657, Test Accuracy: 98.29000091552734
Epoch 7, Loss: 0.007853485643863678, Accuracy: 99.75167083740234,
Test Loss: 0.14718002080917358, Test Accuracy: 97.61000061035156
Epoch 8, Loss: 0.006852753926068544, Accuracy: 99.7933349609375,
Test Loss: 0.07337167114019394, Test Accuracy: 98.31999969482422
Epoch 9, Loss: 0.008080849424004555, Accuracy: 99.73833465576172,
Test Loss: 0.06854166090488434, Test Accuracy: 98.37999725341797
Epoch 10, Loss: 0.004787638783454895, Accuracy: 99.8466644287109
4, Test Loss: 0.06653798371553421, Test Accuracy: 98.589996337890
62
```

The image classifier is now trained to ~98% accuracy on this dataset. To learn more, read the [TensorFlow tutorials (https://www.tensorflow.org/tutorials)](https://www.tensorflow.org/tutorials).

To run TensorBoard, run the following command on `Anaconda (Powershell) Prompt :`

`tensorboard --logdir=` *path/to/log-directory*

- For instance, **`tensorboard --logdir logs/CNN_MNIST_BN`**

Connecting to **`http://localhost:6006`**