โครงงานที่ 1: การสร้างนิวรอนเน็ตเวิร์กสำหรับจำแนกตัวเลข ด้วยชุดข้อมูล MNIST

รหัสนักศึกษา _____57130500098_____ ชื่อ-นามสกุล _____กรรชัย สดหอม_____

เป้าหมายของโครงงาน:

สร้างโปรแกรม neural network ที่สามารถระบุตัวเลขที่เป็นลายมือเขียนจากรูปภาพ แบบ gray scale ได้ที่ระดับ accuracy สูงกว่า 98.5%

ชุดข้อมูล:

http://github.com/myleott/mnist_png/raw/master/mnist_png.tar.gz

1. **การทำความเข้าใจข้อมูล (Data Understanding)**

   ข้อมูล MNIST จะเป็นข้อมูลที่เป็นภาพ Grayscale ที่เป็นเลขตั้งแต่ 0 ถึง 9 และก็เป็นลายมือที่เขียนแบบต่างๆ ขนาดของรูปภาพก็คือ 28 x 28 Pixel ซึ่งมีจำนวนอยู่ 60,000 สำหรับตัวอย่างในการสอนและอีก 10,000 ไว้สำหรับทดสอบ

2. **การเตรียมข้อมูล (Data Pre-processing)**
   - ซึ่งตัวข้อมูลของ MNIST นั้นได้แบ่งมาให้แล้วเป็น 2 แฟ้มก็คือ Testing ซึ่งมีข้อมูลอยู่ 10,000 รูปโดยมีเลข 0-9 แบบคละกันไป และ Training มีข้อมูลอยู่ 60,000 รูป โดยมีเลข 0-9 คละกันไปเช่นกัน
   - ต่อมาก็จะแปลงข้อมูลเป็น Type float32 และจับหารด้วย 255 ทำให้ค่าสีอยู่ในค่าช่วง 0 ถึง 1
   - แปลงข้อมูลจากภาพ 28 x 28 Pixel ให้เป็นเวคเตอร์ขนาด 784 มิติโดยใช้ reshape

3. **การสร้างโมเดลเครือข่ายประสาทเทียม (Model Building)**
   - ซึ่งจาก Source Code ในส่วนการสร้าง Model ก็จะใช้ Raw Pixel เป็น Feature ทำให้เรานั้นมี Input Nodes = 784 Node และมี Output Node ตาม Class = 10 ตามจำนวน Class ของเรา (0-9)
   - โดยมี Hidden Node อยู่ 2 แถว แถวละ 512 Node
   - โดย Dense ในที่นี้จะหมายถึงโครงสร้างแบบ Fully Connected ระหว่างชั้นที่ติดกัน และเลือกใช้ Activation Function ชนิด relu

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.25))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])
```

- เนื่องจากเป็นการ Classification ดังนั้นเราจะใช้ Loss Function คือ Cross Entropy และเอาข้อมูล Gradient Descent มาปรับใช้โดยใช้อัลกอลิทึมคือ RMSprop

4. การเทรนโมเดล (Model Training)

4.1 Macbook Pro 2.7GHz - 128GB 2.7GHz dual-core Intel Core i5 processor (Turbo Boost up to 3.1GHz) with 3MB shared L3 cache and RAM 8 GiB with macOS High Sierra (version 10.13) and Jupyter Lab with Keras version 2.1.6

4.2 ค่า hyperparameter ต่างๆ

- Batch_size = 128
- Epochs = 10 รอบ
- Loss Function ใช้ Cross Entropy Cross Entropy ตามได้อธิบายไว้ตามข้อ 3
- Optimizer ใช้ RMSprop (ref : http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf)

4.3 รายงานผลการเทรน โดยแสดงค่า Training Accuracy และระยะเวลาทั้งหมดที่ใช้ในการเทรนเน็ตเวิร์ก

```
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Epoch 1/10
60000/60000 [==============================] - 10s 172us/step - loss: 0.0300 - acc: 0.9937
Epoch 2/10
60000/60000 [==============================] - 12s 193us/step - loss: 0.0302 - acc: 0.9938
Epoch 3/10
60000/60000 [==============================] - 10s 172us/step - loss: 0.0298 - acc: 0.9941
Epoch 4/10
60000/60000 [==============================] - 11s 175us/step - loss: 0.0306 - acc: 0.9942
Epoch 5/10
60000/60000 [==============================] - 11s 188us/step - loss: 0.0296 - acc: 0.9944
Epoch 6/10
60000/60000 [==============================] - 12s 200us/step - loss: 0.0323 - acc: 0.9941
Epoch 7/10
60000/60000 [==============================] - 12s 198us/step - loss: 0.0309 - acc: 0.9947
Epoch 8/10
60000/60000 [==============================] - 11s 190us/step - loss: 0.0322 - acc: 0.9944
Epoch 9/10
60000/60000 [==============================] - 11s 191us/step - loss: 0.0332 - acc: 0.9939
Epoch 10/10
60000/60000 [==============================] - 12s 194us/step - loss: 0.0295 - acc: 0.9943
```

ระยะเวลาของการเทรนแต่ละรอบจะตกอยู่ประมาณรอบละ 11 วินาที รวมทั้งหมด 112 วินาที ~ 1 นาที 52 วินาที

5. การทดสอบประสิทธิภาพของโมเดล (Model Evaluation)

```
score = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 1s 106us/step
Test loss: 0.13621695852212487
Test accuracy: 0.9851
```

ผลของการทดสอบจากโมเดลค่า Accuracy ที่ได้จะอยู่ที่ 0.9851 หรือ 98.51% ครับผม

6. สรุปผลและข้อเสนอแนะ

จากที่ได้ทำการทดลองการสร้างโมเดลในรูปแบบต่างๆ โดยเปลี่ยน Activation ในการเทรนกับเพิ่มรอบในการเทรน ก็ทำให้พบว่าบาง Activation ก็ไม่ได้เหมาะกับข้อมูลประเภทนี้ทำให้ค่า Accuracy ที่ได้นั้นคงที่ ส่วนรอบในการเทรนก็ต้องดูว่าต้องเทรนกี่รอบ

ถึงจะได้ค่า Accuracy ที่เหมาะสมที่สุด ข้อเสนอในการใช้งาน เนื่องจากผมได้ตั้งค่า Epoch ไปไว้แค่ 10 รอบทำให้เกิดความรวมเร็วในการเทรน แต่ถ้าหากเราเพิ่มรอบไปอีกโมเดลของเราก็อาจจะดีขึ้นแล้วของปรับแต่งตามความเหมาะสมของข้อมูลด้วย

****

การส่งงาน: โครงงานนี้เป็นโครงงานเดี่ยว นักศึกษาต้องส่ง

1. **รายงานในรูปแบบไฟล์ PDF (บันทึกชื่อเป็น mnist-report.pdf)**
2. **ซอร์สโค้ด**

โดยการอัพโหลดข้อมูลทั้งหมดไปไว้บน Dropbox folder ชื่อ **01_MNIST**

****