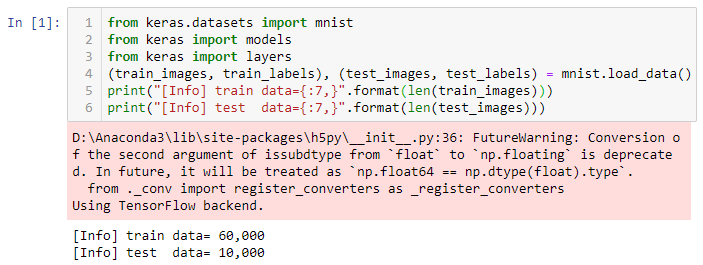
**使用不同優化器、調整epoch、batch size、隱藏層數、隱藏層節點數**

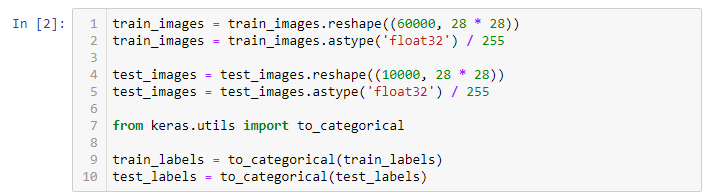
資管數二 7107029022 邱靖詒

* **MNIST手寫辨識資料集**

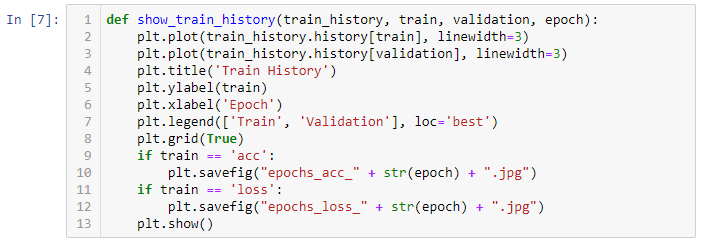
1. 分割測試集、訓練集



1. 前置作業



1. 定義繪製圖形之函式



* **優化器Optimizer**

使用以下參數配置：

network = models.Sequential()

network.add(layers.Dense(512, activation='relu', input\_shape=(28 \* 28,)))

network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop', loss='categorical\_crossentropy', metrics=['accuracy'])

rmsprop\_hist = network.fit(train\_images, train\_labels, epochs=5, batch\_size=128)

|  |  |
| --- | --- |
| 1. **RMSprop** | |
| 1. **SGD** | |
| 1. **Adam** | |
| 1. **AdaGrad** | |
| 1. **AdaDelta** | |
|  | |
| 1. **Nadam** | |
| 1. **Accuracy of different optimizer** | 1. **Loss of different optimizer** |

**Conclusions**

* 以手寫辨識資料集來說，使用RMSprop優化器準確度最高、損失最少。
* 其餘優化器在相似的情况下表現差不多。
* RMSprop是一種自我調整學習速率的方法。
* 任何一種優化器，準確度隨著Epoch增加，損失隨著Epoch減少。
* 自適應學習率方法Adagrad、AdaDelta、RMSprop、Adam幾乎很快就可以達到收斂的效果。
* **數據訓練的總輪數Epoch**

使用以下參數配置：

network = models.Sequential()

network.add(layers.Dense(512, activation='relu', input\_shape=(28 \* 28,)))

network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop', loss='categorical\_crossentropy', metrics=['accuracy'])

train\_history = network.fit(train\_images, train\_labels, validation\_split=0.2, epochs=5, batch\_size=128)

|  |  |
| --- | --- |
| 1. **epochs=5** | 1. **epochs=20** |
|  |  |
| 1. **epochs=50** | 1. **epochs=100** |
| 1. **Accuracy of different epochs** | |

**Conclusions**

* 隨著Epoch的增加，準確率也逐漸提升。
* Epoch 5、Epoch 20、Epoch 50、Epoch 100準確率都表現得不錯。
* 資料的多樣性會影響合適的epoch的數量。
* **批次處理數據量Batch Size**

使用以下參數配置：

network = models.Sequential()

network.add(layers.Dense(512, activation='relu', input\_shape=(28 \* 28,)))

network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='sgd', loss='categorical\_crossentropy', metrics=['accuracy'])

train\_history = network.fit(train\_images, train\_labels, validation\_split=0.2, epochs=5, batch\_size=64)

|  |  |
| --- | --- |
| 1. **batch\_size=64** |  |
| 1. **batch\_size=128** |  |
| 1. **batch\_size=256** |  |
| 1. **batch\_size=512** |  |
| 1. **batch\_size=1024** |  |
| 1. **Accuracy/Loss of different batch size** |  |
| 1. **Accuracy of different batch size** | |

**Conclusions**

* 隨著Batch Size的增加，準確率逐漸下降，損失也逐漸下降。
* Batch Size 64、Batch Size 128、Batch Size 256、Batch Size 512、Batch Size 1024，以Batch Size 64表現最好，準確率高達95.34%。
* 對於大的資料集，不能使用全批次(大Batch Size)，因為會得到更差的結果。
* **隱藏層Hidden Layer**

使用以下參數配置：

network = models.Sequential()

network.add(layers.Dense(512, activation='relu', input\_shape=(28 \* 28,)))

network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='sgd', loss='categorical\_crossentropy', metrics=['accuracy'])

train\_history = network.fit(train\_images,train\_labels, validation\_split=0.2, epochs=20, batch\_size=128)

|  |  |
| --- | --- |
| 1. **1 hidden layer** | |
|  |  |
|  |  |
| 1. **2 hidden layers** | |
|  |  |
| 1. **3 hidden layers** | |
|  |  |
| 1. **Accuracy/Loss of different hidden layers** |  |
| 1. **Accuracy of different hidden layers** | |

**Conclusions**

* 隨著Hidden Layers的增加，準確率逐漸上升，損失也逐漸下降。
* 1 Hidden Layer、2 Hidden Layers、3 Hidden Layers，以3 Hidden Layers表現最好，準確率高達94.88%。
* 輸入（可見層）和第一個隱藏層之間加入一層Dropout。丟棄率設為50%，就是說每輪迭代時每2個輸入值就會被隨機拋棄1個，準確率完美地提升。
* Dropout被用於兩個隱藏層之間和隱藏層與輸出層之間。丟棄率同樣設為50%，準確率效果更好。
* 實驗過程中，以Dropout控制在20%~50%，可從20%開始嘗試。如果比例太低則起不到效果，比例太高則會導致模型的欠學習。
* **隱藏層節點Hidden Node**

使用以下參數配置：

network = models.Sequential()

network.add(layers.Dense(512, activation='relu', input\_shape=(28 \* 28,)))

network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='sgd', loss='categorical\_crossentropy', metrics=['accuracy'])

train\_history = network.fit(train\_images,train\_labels, validation\_split=0.2, epochs=20, batch\_size=128)

|  |  |
| --- | --- |
| 1. **32 hidden nodes** | |
|  |  |
|  |  |
| 1. **64 hidden nodes** | |
|  |  |
| 1. **128 hidden nodes** | |
|  |  |
| 1. **256 hidden nodes** | |
|  |  |
|  |  |
| 1. **512 hidden nodes** | |
|  |  |
| 1. **Accuracy/Loss of different hidden nodes** |  |
|  |  |
| 1. **Accuracy of different hidden nodes** | |

**Conclusions**

* 隨著Hidden Nodes的增加，準確率逐漸上升，損失也逐漸下降。
* 32 Hidden Nodes、64 Hidden Nodes、128 Hidden Nodes、256 Hidden Nodes、512 Hidden Nodes，以512 Hidden Nodes表現最好，準確率高達94.74%。