機器學習作業 4

B06502028
莊立楷
機械三

由於我在 ipynb 上運行程式會有 error，所以這次作業我皆在自己的 IDE 上執行 py 檔，故作業也交成 py 檔，希望助教見諒！

經過設計後，我的 test accuracy 可以達到約 88.4%正確。下圖為我運行 40 個 epoch 後的結果，附上我的神經網路層。

```
class Residual(nn.Block):
    def __init__(self,**kwargs):
        super(Residual, self).__init__(**kwargs)
        self.net = nn.Sequential()
        self.net.add(nn.Dense(64, activation='relu'),nn.BatchNorm(),
                     nn.Dense(64, activation='relu'),nn.BatchNorm(),
                     nn.Dense(64, activation='relu'),nn.BatchNorm(),
                     nn.Dense(64, activation='relu'),nn.BatchNorm(),
                     nn.Dense(64, activation='relu'),nn.BatchNorm(),
                     nn.Dense(64, activation='relu'),nn.BatchNorm(),)
    def forward(self, X):
        return (nd.relu(self.net(X)+X))
print('Constructing network...\n')
batch_size = 64
net = nn.Sequential()
net.add(nn.Dense(64),
        Residual(), nn.BatchNorm(), nn.Dropout(0.3),
        Residual(), nn.BatchNorm(),
        Residual(), nn.BatchNorm(),
        Residual(), nn.BatchNorm(),
        Residual(), nn.BatchNorm(),
        Residual(), nn.BatchNorm(), nn.Dropout(0.3),
        Residual(), nn.BatchNorm(),
        Residual(), nn.BatchNorm(),
        Residual(), nn.BatchNorm(),
        Residual(), nn.BatchNorm(),
        nn.Dense(10))
```

```
    (2): Dense(None -> 64, Activation(relu))
    (3): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False,
    (4): Dense(None -> 64, Activation(relu))
    (5): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False,
    (6): Dense(None -> 64, Activation(relu))
    (7): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False,
    (8): Dense(None -> 64, Activation(relu))
    (9): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False,
    (10): Dense(None -> 64, Activation(relu))
    (11): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False
  )
 )
 (22): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, us
 (23): Dense(None -> 10, linear)
)
Start training...

epoch 0: loss 0.344, train acc 0.879, test acc 0.100, in 57.2 sec
epoch 1: loss 0.348, train acc 0.877, test acc 0.867, in 63.4 sec
epoch 2: loss 0.338, train acc 0.880, test acc 0.868, in 65.9 sec
epoch 3: loss 0.343, train acc 0.879, test acc 0.878, in 68.7 sec
epoch 4: loss 0.332, train acc 0.882, test acc 0.101, in 61.1 sec
epoch 5: loss 0.327, train acc 0.882, test acc 0.102, in 63.1 sec
epoch 6: loss 0.329, train acc 0.882, test acc 0.882, in 69.7 sec
epoch 7: loss 0.325, train acc 0.883, test acc 0.884, in 66.1 sec
epoch 8: loss 0.322, train acc 0.885, test acc 0.878, in 68.7 sec
epoch 9: loss 0.322, train acc 0.885, test acc 0.881, in 67.2 sec
請按任意鍵繼續 . . .
```

我的每層皆為 64 個 neuron，每層之間都有 relu 以及 batch normalization。

另外，我使用了 residual 殘差的方法，即神經元的 output 會是本身這層以及前面數層相加並取 relu 的 ouput，這個方法可以有效將深度較深的神經網路 train 起來。

並且，由於 overfit 的關係我加了兩層 drop out。

參考資料：

殘差網路：https://discuss.gluon.ai/t/topic/1663/20

Mxnet 相關教學：

https://mxnet.apache.org/api/python/docs/tutorials/packages/gluon/blocks/nn.html

https://gluon.mxnet.io/chapter03_deep-neural-networks/mlp-dropout-gluon.html