Assignment_5

March 12, 2023

1 Imports

[ ]:
```
from nnet.model import Sequential
from nnet.layers import (
    Input,
    Dense,
    Conv2D,
    MaxPool2D,
    Flatten,
)
from nnet.utils import one_hot
import numpy as np
import os
from scipy.io import loadmat
import matplotlib.pyplot as plt
```

2 Loading Data

[ ]:
```
DATA_DIR = "data"
X_train = loadmat(os.path.join(DATA_DIR, "train_images.mat"))["train_images"]
X_test = loadmat(os.path.join(DATA_DIR, "test_images.mat"))["test_images"]
y_train = loadmat(os.path.join(DATA_DIR, "train_labels.mat"))["train_labels"]
y_test = loadmat(os.path.join(DATA_DIR, "test_labels.mat"))["test_labels"]
```

2.1 Preprocessing

[ ]:
```
# Making the data compatible with the model
X_train = X_train.T
X_test = X_test.T
X_train = X_train[:, :, np.newaxis, :]
X_test = X_test[:, :, np.newaxis, :]
X_train.shape, X_test.shape
```

[ ]:
```
((28, 28, 1, 1000), (28, 28, 1, 1000))
```

[ ]:
```
# Normalizing the data
X_train = X_train / X_train.max() - 0.5
X_test = X_test / X_test.max() - 0.5
```

[ ]:
```
# One hot encoding the labels
y_train = np.squeeze(y_train.T)
y_test = np.squeeze(y_test.T)
y_train = one_hot(y_train, 10)
y_test = one_hot(y_test, 10)

y_train.shape, y_test.shape
```

[ ]:
```
((10, 1000), (10, 1000))
```

3 Problems

3.1 Problem 1

> **Note:** I'm using the updated version of the module which I created for the previous assignment. To see how to work with the module, please refer to the previous assignment.

Here is the model:

[ ]:
```
model = Sequential("CNN Model")
inp = Input((28, 28, 1), name="Input")
conv = Conv2D(9, (3, 3), padding="valid", activation="relu", name="Conv")
pool = MaxPool2D((2, 2), 2, name="MaxPool")
flat = Flatten(name="Flatten")
dense = Dense(10, name="Output", activation="softmax")

model.add(inp)
model.add(conv)
model.add(pool)
model.add(flat)
model.add(dense)

model.summary()
```

```
Model: CNN Model
--------------------------------------------------------------------------------
-----------------

  Name       Input Shapes     Output Shapes     Weight Shapes     Bias Shapes
# Parameters
```

```
 Input     (28, 28, 1)      (28, 28, 1)       None              None
0


 Conv      (28, 28, 1)      (13, 13, 9)       (3, 3, 1, 9)      (9, 1)
90


 MaxPool   (13, 13, 9)      (6, 6, 9)         None              None
0


 Flatten   (6, 6, 9)        (324,)            None              None
0


 Output    (324,)           (10,)             (10, 324)         (10, 1)
3250


================================================================================
=================
Total Parameters: 3340
--------------------------------------------------------------------------------
-----------------
```

So, the model has just 3340 parameters. Would this be enough to learn the data? Let's find out. For this, we compile the model:

[ ]:
```python
model.compile(
    loss="categorical_cross_entropy",
    metrics=["accuracy"],
    initializer="glorot",
)
```

Finally, fit the model:

[ ]:
```python
history = model.fit(X_train, y_train, epochs=50, batch_size=32, lr=0.05,␣
 ↪verbose=10)
```

```
Epoch 0001/0050 | Loss: 2.56050 | Accuracy: 0.18200 |
Epoch 0002/0050 | Loss: 2.44336 | Accuracy: 0.20800 |
Epoch 0003/0050 | Loss: 2.35812 | Accuracy: 0.21400 |
Epoch 0004/0050 | Loss: 2.29177 | Accuracy: 0.21200 |
Epoch 0005/0050 | Loss: 2.23784 | Accuracy: 0.21100 |
Epoch 0006/0050 | Loss: 2.19254 | Accuracy: 0.21200 |
Epoch 0007/0050 | Loss: 2.15412 | Accuracy: 0.30300 |
Epoch 0008/0050 | Loss: 2.12118 | Accuracy: 0.30900 |
Epoch 0009/0050 | Loss: 2.09269 | Accuracy: 0.31700 |
```

```
Epoch 0010/0050 | Loss: 2.06748 | Accuracy: 0.32400 |
Epoch 0011/0050 | Loss: 2.04555 | Accuracy: 0.32700 |
Epoch 0012/0050 | Loss: 2.02632 | Accuracy: 0.32900 |
Epoch 0013/0050 | Loss: 2.00957 | Accuracy: 0.33200 |
Epoch 0014/0050 | Loss: 1.99492 | Accuracy: 0.33000 |
Epoch 0015/0050 | Loss: 1.98208 | Accuracy: 0.32900 |
Epoch 0016/0050 | Loss: 1.97072 | Accuracy: 0.33200 |
Epoch 0017/0050 | Loss: 1.96073 | Accuracy: 0.33300 |
Epoch 0018/0050 | Loss: 1.95208 | Accuracy: 0.33000 |
Epoch 0019/0050 | Loss: 1.94455 | Accuracy: 0.33000 |
Epoch 0020/0050 | Loss: 1.93779 | Accuracy: 0.32900 |
Epoch 0021/0050 | Loss: 1.93199 | Accuracy: 0.33200 |
Epoch 0022/0050 | Loss: 1.92699 | Accuracy: 0.33500 |
Epoch 0023/0050 | Loss: 1.92250 | Accuracy: 0.33300 |
Epoch 0024/0050 | Loss: 1.91830 | Accuracy: 0.33600 |
Epoch 0025/0050 | Loss: 1.91448 | Accuracy: 0.33700 |
Epoch 0026/0050 | Loss: 1.91109 | Accuracy: 0.33600 |
Epoch 0027/0050 | Loss: 1.90838 | Accuracy: 0.33900 |
Epoch 0028/0050 | Loss: 1.90624 | Accuracy: 0.34300 |
Epoch 0029/0050 | Loss: 1.90439 | Accuracy: 0.34700 |
Epoch 0030/0050 | Loss: 1.90300 | Accuracy: 0.34900 |
Epoch 0031/0050 | Loss: 1.90178 | Accuracy: 0.35000 |
Epoch 0032/0050 | Loss: 1.90051 | Accuracy: 0.34900 |
Epoch 0033/0050 | Loss: 1.89907 | Accuracy: 0.35000 |
Epoch 0034/0050 | Loss: 1.89647 | Accuracy: 0.34700 |
Epoch 0035/0050 | Loss: 1.89163 | Accuracy: 0.34800 |
Epoch 0036/0050 | Loss: 1.88653 | Accuracy: 0.35200 |
Epoch 0037/0050 | Loss: 1.88185 | Accuracy: 0.35200 |
Epoch 0038/0050 | Loss: 1.87717 | Accuracy: 0.35400 |
Epoch 0039/0050 | Loss: 1.87196 | Accuracy: 0.35500 |
Epoch 0040/0050 | Loss: 1.86731 | Accuracy: 0.35600 |
Epoch 0041/0050 | Loss: 1.86298 | Accuracy: 0.35800 |
Epoch 0042/0050 | Loss: 1.85855 | Accuracy: 0.35800 |
Epoch 0043/0050 | Loss: 1.85395 | Accuracy: 0.35700 |
Epoch 0044/0050 | Loss: 1.84980 | Accuracy: 0.36000 |
Epoch 0045/0050 | Loss: 1.84476 | Accuracy: 0.36000 |
Epoch 0046/0050 | Loss: 1.83999 | Accuracy: 0.36200 |
Epoch 0047/0050 | Loss: 1.83557 | Accuracy: 0.36200 |
Epoch 0048/0050 | Loss: 1.83127 | Accuracy: 0.36300 |
Epoch 0049/0050 | Loss: 1.82735 | Accuracy: 0.36700 |
Epoch 0050/0050 | Loss: 1.82316 | Accuracy: 0.36800 |
```

The accuracy is very low. This is because the model implemented above is very simple. It has just 3340 parameters. This is not enough to learn the data.

[ ]:
```
train_acc = model.evaluate(X_train, y_train, metric="accuracy")
print(f"Train Accuracy: {train_acc*100:.2f}%")
```

```
Train Accuracy: 36.80%
```

So, the training accuracy is about 37%. Let's see how the model performs on the test data.

3.2 Problem 2

[ ]:
```
test_acc = model.evaluate(X_test, y_test, metric="accuracy")
print(f"Test Accuracy: {test_acc*100:.2f}%")
```

```
Test Accuracy: 34.30%
```

Okay, the test accuracy is very close to the training accuracy. This means that the model is not overfitting.