課題 1 レポート

羽路 悠斗

2021 年 12 月 14 日

1　課題内容

1.1　3 層ニューラルネットワークの構築

MNIST の画像 1 枚を入力とし、3 層ニューラルネットワークを用いて、0 9 の値のうち 1 つを出力するプログラムを作成せよ。

- キーボードから 0 9999 の整数を入力 i として受け取り、0 9 の整数を標準出力に出力すること。
- MNIST のテストデータ 10000 枚の画像のうち i 番目の画像を入力画像として用いる。
- MNIST の画像サイズ (28 × 28)、画像枚数 (10000 枚)、クラス数 (C = 10) は既知とする。ただし、後々の改良のため変更可能な仕様にしておくことを薦める。
- 中間層のノード数 M は自由に決めて良い。
- 重み $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$ については乱数で決定すること。ここでは、手前の層のノード数を N として 1/N を分散とする平均 0 の正規分布で与えることとする。適切な重みを設定しないため、課題 1 の段階では入力に対してデタラメな認識結果を返す。ただし、実行する度に同じ結果を出力するよう乱数のシードを固定すること。

2　作成したプログラムの説明

2.1　設計方針

それぞれの層を関数として実装する。それらを最後にまとめることでニューラルネットとする。これによりプログラムの可読性が上がり、また後の拡張や再利用も容易になる。

各層の入力と出力のインターフェースを統一する。具体的には i 番目の層のノード数を n_i とすると、入力は n_{i-1} 行 1 列の array で、出力は n_i 行 1 列の array とする。

2.2　パラメータ

パラメータとしては、中間層の数と各層の重みとバイアスがある。中間層は 32 層、重みとバイアスは乱数で決定する。

ソースコード 1　ex1.py

```
1    random.seed(71)
2    units = 32
3    w1, b1 = random.normal(loc=0, scale=np.sqrt(1/784), size=784*units).reshape(units,
          784), random.normal(loc=0, scale=np.sqrt(1/784), size=units)
4    w2, b2 = random.normal(loc=0, scale=np.sqrt(1/units), size=units*10).reshape(10,
         units), random.normal(loc=0, scale=np.sqrt(1/units), size=10)
```

2.3 前処理

標準入力から受け取った整数をインデックスとして、MNIST 手描き文字画像を取り出す。

ソースコード 2　ex1.py

```
1    def preprocessing(input):
2      return mnist.download_and_parse_mnist_file('train-images-idx3-ubyte.gz')[input]
```

2.4 入力層

MNIST の画像を $28 * 28 = 784$ 次元のベクトルに変換する。後の行列演算のために 784 行 1 列の array として保持する。

ソースコード 3　ex1.py

```
1    def input_layer(im):
2      return im.flatten().reshape(-1, 1)
```

2.5 全結合層

2 つの全結合層は、パラメーターは違うが処理は同じなので、まとめてしまう。

ソースコード 4　ex1.py

```
1    def dense_layer(input_vec, weight, bias):
2      return weight@input_vec+bias.reshape(-1, 1)
```

2.6 中間層

ソースコード 4 を利用して、中間層への入力を計算する。

ソースコード 5　ex1.py

```
1    def dense_layer1(input_vec):
2      return dense_layer(input_vec, w1, b1)
```

次に活性化関数として、シグモイド関数を用いる。

ソースコード 6　ex1.py

```
1    def sigmoid(input_vec):
2      return 1/(1+np.exp(-input_vec))
```

2.7 出力層

ソースコード 4 を利用して、出力層への入力を計算する。

ソースコード 7　ex1.py

```
1    def dense_layer2(input_vec):
2      return dense_layer(input_vec, w2, b2)
```

次に活性化関数として、ソフトマックス関数を用いる。出力層は、0 9 の数に対応する 10 クラス分類の確率を出力する。

ソースコード 8　ex1.py

```
1    def softmax(input_vec):
2      return np.exp(input_vec-np.max(input_vec, axis=0)) / (np.sum(np.exp(input_vec-np.
           max(input_vec, axis=0))))
```

2.8 後処理

出力層の出力を用いて、尤度最大のクラスを認識結果として出力する。

ソースコード 9　ex1.py

```
1    def postprocessing(input_vec):
2      return np.argmax(input_vec)
```

2.9 モデル

以上の層を一つにまとめて、0 9999 の整数を受け取り、0 9 の整数を出力するニューラルネットとする。

ソースコード 10　ex1.py

```
1    def model(input):
2      return postprocessing(softmax(dense_layer2(sigmoid(dense_layer1(input_layer(
           preprocessing(input)))))))
```

3 実行結果

実際に標準入力から入力を受け取り、モデルの処理結果を標準出力に出力するプログラムの、実行と実行結果は次の通りである。

ソースコード 11　ex1.py

```
1      if __name__ == '__main__':
2        print('input␣0␣~␣9999␣number')
3        stdin = int(input())
```

```
4         print(f'mnist[{stdin}] is ...')
5         stdout = model(stdin)
6         print(stdout)
```

実行結果

```
input 0 ~ 9999 number
4
mnist[4] is ...
1
```

4　工夫点

各層の入力と出力のインターフェースを先に仕様として決めてしまい、仕様に合わせるように実装していった事で、形状の把握が容易である。特に入力層において、平坦化するだけではなく仕様に合わせて形状を変えてから出力したことで、中間層の全結合層の行列演算が明快である。そのまま 1 次元ベクトルとしていても、numpy の仕様としてキャストしてくれるようだが、明示的に変換しておくほうが良いだろう。

各層を関数として記述したことで、デバッグも容易である。初めは実行するとエラーが出たが、一つずつ層の入力と出力を確認して解決できた。

また、for 文を一度も使わずに行列演算を駆使したことで、高速化できた。

5　問題点

特になし。