nn_impl

September 13, 2021

1 A demo of writing a multiple-layer perceptron from scratch

Author: March Jiaolin Luo

I write this demo for tutoring a friend who is learning deep learning. This demo shows how we model a general neural network layer and how we implement a MLP classifier for handwriting recognisition based on the general neural network layer, which we merely use numpy.

2 Importing library

[24]:
```
import numpy as np
import os
```

3 Setting random seed for reproducibility

[25]:
```
np.random.seed(123)
```

4 A general neural network layer

[26]:
```
class NNLayer:
    def __init__(self,
                 n_inputs,
                 n_neurons,
                 alpha = 0.01
                ):

        self.alpha = alpha
        self.n_inputs = n_inputs   #N
        self.n_neurons = n_neurons #M

        sigma = 0.1

        self.W = np.random.uniform(-sigma, sigma, (self.n_neurons, self.
↪n_inputs))
        self.b = np.random.uniform(-sigma, sigma, (self.n_neurons, 1))
```

```
    # We compute the Jacobian matrix for sigmoid
    # dz / dy
    def dsigmoid(self, y):

        z = self.sigmoid(y.flatten())

        dzdy_flatten = z*(1 - z)

        dzdy_Jacobian = np.diagflat(dzdy_flatten)

        return dzdy_Jacobian

    # We compute the sigmoid
    def sigmoid(self, y):
        z = 1.0 / (1.0 + np.exp(-y))
        return z

    def forward(self, x):

        x = x.copy().reshape(-1, 1)


        y = self.W @ x + self.b

        z = self.sigmoid(y)

        self.y = y
        self.x = x

        return z



    def backward(self, grad_in):

        # M x 1
        dLdz = grad_in.copy().reshape(-1, 1)


        # M x M
        dzdy = self.dsigmoid(self.y)

        #       1 x M    M x M = 1 x M
        dLdy = (dLdz.T @ dzdy).T
        #now dLdy is M x 1
```

```
        dLdW = dLdy @ self.x.reshape(1, -1)
        dLdb = dLdy

        grad_out = dLdy.T @ self.W

        self.W = self.W - self.alpha * dLdW
        self.b = self.b - self.alpha * dLdb


        return grad_out
```

4.1 Testing the NN layer

[27]:
```
alayer = NNLayer(n_inputs = 3, n_neurons = 5)
```

[28]:
```
xi = np.random.rand(3)
```

[29]:
```
xi
```

[29]:
```
array([0.63440096, 0.84943179, 0.72445532])
```

[30]:
```
alayer.forward(xi)
```

[30]:
```
array([[0.49915458],
       [0.4922888 ],
       [0.50618145],
       [0.49979187],
       [0.47726428]])
```

[31]:
```
yi = np.array([1,2,3])
```

[32]:
```
Jacobian = alayer.dsigmoid(yi)
Jacobian
```

[32]:
```
array([[0.19661193, 0.        , 0.        ],
       [0.        , 0.10499359, 0.        ],
       [0.        , 0.        , 0.04517666]])
```

[33]:
```
grad_in = np.array([1,2,3,4,5])
```

[34]:
```
alayer.backward(grad_in)
```

[34]:
```
array([[ 0.05015562, -0.10224841, -0.00383099]])
```

5 A Three-layer Multi-layer Perception Classifier

[35]:
```
class AThreeLayerSimpleMLPC:
    def __init__(self, n_inputs, n_classes):

        self.n_classes = n_classes

        self.hidden_layer1 = NNLayer(n_inputs = n_inputs, n_neurons = 32)
        self.hidden_layer2 = NNLayer(n_inputs = 32, n_neurons = 24)
        self.output_layer = NNLayer(n_inputs = 24, n_neurons = n_classes)

    def evaluate(self, X, y_truth):

        y_pred = [self.predict(xi) for xi in X]

        accuracy = np.mean(y_pred == y_truth)

        return accuracy

    def predict(self, x):

        x = x.copy().reshape(-1, 1)

        prob = self.forward(x)

        return np.argmax(prob)

    def forward(self, x):

        x = x.copy().reshape(-1, 1)

        x = self.hidden_layer1.forward(x)
        x = self.hidden_layer2.forward(x)
        x = self.output_layer.forward(x)

        return x

    def backward(self, grad_in):

        grad_in = grad_in.reshape(-1, 1)

        g = self.output_layer.backward(grad_in)
        g = self.hidden_layer2.backward(g)
        g = self.hidden_layer1.backward(g)

    def fit(self, X, y, epochs = 50):
```

```
        indices = np.random.permutation(len(X))

        onehots = np.eye(self.n_classes)

        loss_hist = []

        for epoch in range(epochs):

            np.random.shuffle(indices)

            run_loss = 0

            # for each epoch we train over all data
            for idx in indices:

                xi = X[idx]
                yi = y[idx]

                yi_truth = onehots[yi]

                yi_pred = self.forward(xi)


                mse_loss = (np.sum(yi_pred - yi_truth)**2) / 2.0
                mse_grad_in = yi_pred.reshape(-1, 1) - yi_truth.reshape(-1, 1)

                run_loss += mse_loss

                self.backward(mse_grad_in)

            run_loss = run_loss / len(indices)

            loss_hist.append(run_loss)

            #if epoch % 5 == 0:
            print(f"Epoch {epoch} loss = {run_loss:.6f}")

            run_loss = 0

        return loss_hist
```

6 Loading MNIST dataset

[36]:
```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.
↪load_data(path="mnist.npz")
```

[37]:
```
# normalisation MNIST to [-1, 1]
x_train = x_train.reshape(-1, 28 * 28) / 128. - 0.5
x_test = x_test.reshape(-1, 28 * 28) / 128. - 0.5
```

7 Performaning handwriting recognisition task for MNIST

[38]:
```
mlpc = AThreeLayerSimpleMLPC(n_inputs = 28*28, n_classes = 10)
```

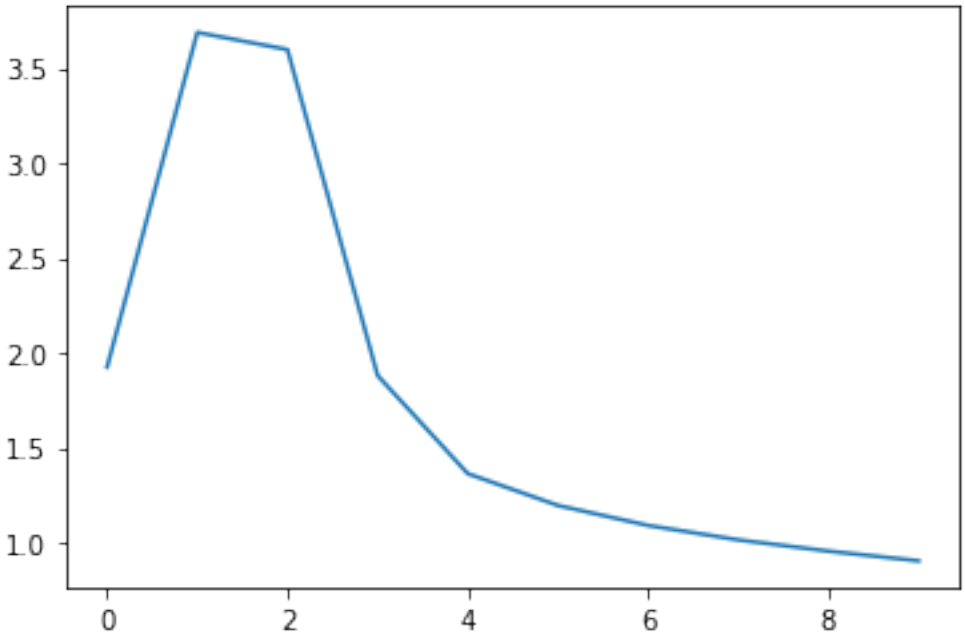
[39]:
```
loss_hist = mlpc.fit(x_train, y_train, epochs = 10)
```

```
Epoch 0 loss = 1.923932
Epoch 1 loss = 3.687358
Epoch 2 loss = 3.597052
Epoch 3 loss = 1.878976
Epoch 4 loss = 1.361905
Epoch 5 loss = 1.194306
Epoch 6 loss = 1.088908
Epoch 7 loss = 1.012949
Epoch 8 loss = 0.953642
Epoch 9 loss = 0.902428
```

[40]:
```
import matplotlib.pyplot as plt

plt.plot(loss_hist)
```

[40]:
```
[<matplotlib.lines.Line2D at 0x7feb6dfce580>]
```



8 Testing the model

[41]:
```
x0 = x_test[0]
y0 = y_test[0]

y0
```

[41]: 7

[42]:
```
mlpc.predict(x0)
```

[42]: 7

9 Evaluating on unseen testing dataset

[43]:
```
mlpc.evaluate(x_test, y_test)
```

[43]: 0.9457

[ ]: