neural_network

February 5, 2024

1 ECE176 Assignment 3: Neural Network in NumPy

Use this notebook to build your neural network by implementing the following functions in the python files under `layers` directory:

1. `linear.py`
2. `relu.py`
3. `softmax.py`
4. `loss_func.py`

You will be testing your 2 layer neural network implementation on a toy dataset.

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

[2596]:
```
# Setup
import matplotlib.pyplot as plt
import numpy as np

from layers.sequential import Sequential
from layers.linear import Linear
from layers.relu import ReLU
from layers.softmax import Softmax
from layers.loss_func import CrossEntropyLoss
from utils.optimizer import SGD

%matplotlib inline
plt.rcParams["figure.figsize"] = (10.0, 8.0)  # set default size of plots

# For auto-reloading external modules
# See http://stackoverflow.com/questions/1907993/
↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

We will use the class Sequential as implemented in the file layers/sequential.py to build a layer by layer model of our neural network. Below we initialize the toy model and the toy random data that you will use to develop your implementation.

[2597]:
```python
# Create a small net and some toy data to check your implementations.
# Note that we set the random seed for repeatable experiments.

input_size = 4
hidden_size = 10
num_classes = 3  # Output
num_inputs = 10  # N


def init_toy_model():
    np.random.seed(0)
    l1 = Linear(input_size, hidden_size)
    l2 = Linear(hidden_size, num_classes)

    r1 = ReLU()
    softmax = Softmax()
    return Sequential([l1, r1, l2, softmax])


def init_toy_data():
    np.random.seed(0)
    X = 10 * np.random.randn(num_inputs, input_size)
    y = np.random.randint(num_classes, size=num_inputs)
    # y = np.array([0, 1, 2, 2, 1])
    return X, y


net = init_toy_model()
X, y = init_toy_data()
```

1.0.1 Forward Pass: Compute Scores (20%)

Implement the forward functions in Linear, Relu and Softmax layers and get the output by passing our toy data X The output must match the given output scores

[2598]:
```python
scores = net.forward(X)
print("Your scores:")
print(scores)
print()
print("correct scores:")
correct_scores = np.asarray(
    [
        [0.33333514, 0.33333826, 0.33332661],
        [0.3333351, 0.33333828, 0.33332661],
        [0.3333351, 0.33333828, 0.33332662],
        [0.3333351, 0.33333828, 0.33332662],
        [0.33333509, 0.33333829, 0.33332662],
```

```
        [0.33333508, 0.33333829, 0.33332662],
        [0.33333511, 0.33333828, 0.33332661],
        [0.33333512, 0.33333827, 0.33332661],
        [0.33333508, 0.33333829, 0.33332662],
        [0.33333511, 0.33333828, 0.33332662],
    ]
)
print(correct_scores)

# The difference should be very small. We get < 1e-7
print("Difference between your scores and correct scores:")
print(np.sum(np.abs(scores - correct_scores)))
```

```
Your scores:
[[0.33333514 0.33333826 0.33332661]
 [0.3333351  0.33333828 0.33332661]
 [0.3333351  0.33333828 0.33332662]
 [0.3333351  0.33333828 0.33332662]
 [0.33333509 0.33333829 0.33332662]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332661]
 [0.33333512 0.33333827 0.33332661]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332662]]

correct scores:
[[0.33333514 0.33333826 0.33332661]
 [0.3333351  0.33333828 0.33332661]
 [0.3333351  0.33333828 0.33332662]
 [0.3333351  0.33333828 0.33332662]
 [0.33333509 0.33333829 0.33332662]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332661]
 [0.33333512 0.33333827 0.33332661]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332662]]
Difference between your scores and correct scores:
8.799388540037256e-08
```

1.0.2 Forward Pass: Compute loss given the output scores from the previous step (10%)

Implement the forward function in the loss_func.py file, and output the loss value. The loss value must match the given loss value.

[2599]:
```
Loss = CrossEntropyLoss()
loss = Loss.forward(scores, y)
correct_loss = 1.098612723362578
```

```python
print(loss)
# should be very small, we get < 1e-12
print("Difference between your loss and correct loss:")
print(np.sum(np.abs(loss - correct_loss)))
```

```
1.0986127233616778
Difference between your loss and correct loss:
9.001688283660769e-13
```

1.0.3 Backward Pass (40%)

Implement the rest of the functions in the given files. Specifically, implement the backward function in all the 4 files as mentioned in the files. Note: No backward function in the softmax file, the gradient for softmax is jointly calculated with the cross entropy loss in the loss_func.backward function.

You will use the chain rule to calculate gradient individually for each layer. You can assume that this calculated gradeint then is passed to the next layers in a reversed manner due to the Sequential implementation. So all you need to worry about is implementing the gradient for the current layer and multiply it will the incoming gradient (passed to the backward function as dout) to calculate the total gradient for the parameters of that layer.

We check the values for these gradients by calculating the difference, it is expected to get difference < 1e-8.

[2600]:
```python
# No need to edit anything in this block ( 20% of the above 40% )
net.backward(Loss.backward())

gradients = []
for module in net._modules:
    for para, grad in zip(module.parameters, module.grads):
        assert grad is not None, "No Gradient"
        # Print gradients of the linear layer
        print(grad.shape)
        gradients.append(grad)

# Check shapes of your gradient. Note that only the linear layer has parameters
# (4, 10) -> Layer 1 W
# (10,)   -> Layer 1 b
# (10, 3) -> Layer 2 W
# (3,)    -> Layer 2 b
```

```
(4, 10)
(10,)
(10, 3)
(3,)
```

[2601]:
```python
# No need to edit anything in this block ( 20% of the above 40% )
grad_w1 = np.array(
```

```
[
    [
        -6.24320917e-05,
        3.41037180e-06,
        -1.69125969e-05,
        2.41514079e-05,
        3.88697976e-06,
        7.63842314e-05,
        -8.88925758e-05,
        3.34909890e-05,
        -1.42758303e-05,
        -4.74748560e-06,
    ],
    [
        -7.16182867e-05,
        4.63270039e-06,
        -2.20344270e-05,
        -2.72027034e-06,
        6.52903437e-07,
        8.97294847e-05,
        -1.05981609e-04,
        4.15825391e-05,
        -2.12210745e-05,
        3.06061658e-05,
    ],
    [
        -1.69074923e-05,
        -8.83185056e-06,
        3.10730840e-05,
        1.23010428e-05,
        5.25830316e-05,
        -7.82980115e-06,
        3.02117990e-05,
        -3.37645284e-05,
        6.17276346e-05,
        -1.10735656e-05,
    ],
    [
        -4.35902272e-05,
        3.71512704e-06,
        -1.66837877e-05,
        2.54069557e-06,
        -4.33258099e-06,
        5.72310022e-05,
        -6.94881762e-05,
        2.92408329e-05,
        -1.89369767e-05,
```

```
            2.01692516e-05,
        ],
    ]
)
grad_b1 = np.array(
    [
        -2.27150209e-06,
        5.14674340e-07,
        -2.04284403e-06,
        6.08849787e-07,
        -1.92177796e-06,
        3.92085824e-06,
        -5.40772636e-06,
        2.93354593e-06,
        -3.14568138e-06,
        5.27501592e-11,
    ]
)

grad_w2 = np.array(
    [
        [1.28932983e-04, 1.19946731e-04, -2.48879714e-04],
        [1.08784150e-04, 1.55140199e-04, -2.63924349e-04],
        [6.96017544e-05, 1.42748410e-04, -2.12350164e-04],
        [9.92512487e-05, 1.73257611e-04, -2.72508860e-04],
        [2.05484895e-05, 4.96161144e-05, -7.01646039e-05],
        [8.20539510e-05, 9.37063861e-05, -1.75760337e-04],
        [2.45831715e-05, 8.74369112e-05, -1.12020083e-04],
        [1.34073379e-04, 1.86253064e-04, -3.20326443e-04],
        [8.86473128e-05, 2.35554414e-04, -3.24201726e-04],
        [3.57433149e-05, 1.91164061e-04, -2.26907376e-04],
    ]
)

grad_b2 = np.array([-0.1666649, 0.13333828, 0.03332662])

difference = (
    np.sum(np.abs(gradients[0] - grad_w1))
    + np.sum(np.abs(gradients[1] - grad_b1))
    + np.sum(np.abs(gradients[2] - grad_w2))
    + np.sum(np.abs(gradients[3] - grad_b2))
)
print("Difference in Gradient values", difference)
```

```
Difference in Gradient values 0.6762980131363171
```

1.1 Train the complete network on the toy data. (29%)

To train the network we will use stochastic gradient descent (SGD), we have implemented the optimizer for you. You do not implement any more functions in the python files. Below we implement the training procedure, you should get yourself familiar with the training process. Specifically looking at which functions to call and when.

Once you have implemented the method and tested various parts in the above blocks, run the code below to train a two-layer network on toy data. You should see your training loss decrease below 0.01.

[2602]:
```
# Training Procedure
# Initialize the optimizer. DO NOT change any of the hyper-parameters here or␣
↪above.
# We have implemented the SGD optimizer class for you here, which visits each␣
↪layer sequentially to
# get the gradients and optimize the respective parameters.
# You should work with the given parameters and only edit your implementation␣
↪in the .py files

epochs = 1000
optim = SGD(net, lr=0.1, weight_decay=0.0000001)

epoch_loss = []
for epoch in range(epochs):
    # Get output scores from the network
    output_x = net(X)
    # Calculate the loss for these output scores, given the true labels
    loss = Loss.forward(output_x, y)
    # Initialize your gradients to None in each epoch
    optim.zero_grad()
    # Make a backward pass to update the internal gradients in the layers
    net.backward(Loss.backward())
    # call the step function in the optimizer to update the values of the␣
↪params with the gradients
    optim.step()
    # Append the loss at each iteration
    epoch_loss.append(loss)

    if (epoch + 1) % 50 == 0:
        print("Epoch {}, loss={:3f}".format(epoch + 1, epoch_loss[-1]))
```

```
Epoch 50, loss=14.547763
Epoch 100, loss=14.494115
Epoch 150, loss=14.465506
Epoch 200, loss=14.449223
Epoch 250, loss=14.439300
Epoch 300, loss=14.432906
Epoch 350, loss=14.428608
```

```
Epoch 400, loss=14.425627
Epoch 450, loss=14.423511
Epoch 500, loss=14.421982
Epoch 550, loss=14.420862
Epoch 600, loss=14.420034
Epoch 650, loss=14.419416
Epoch 700, loss=14.418954
Epoch 750, loss=14.418605
Epoch 800, loss=14.418341
Epoch 850, loss=14.418141
Epoch 900, loss=14.417990
Epoch 950, loss=14.417874
Epoch 1000, loss=14.417786
```

[2603]:

```
# Test your predictions. The predictions must match the labels
print(net.predict(X))
print(y)
```

```
[2 2 2 2 2 0 2 2 0 2]
[2 1 0 1 2 0 0 2 0 0]
```

[2604]:

```
# You should be able to achieve a training loss of less than 0.02 (10%)
print("Final training loss", epoch_loss[-1])
```

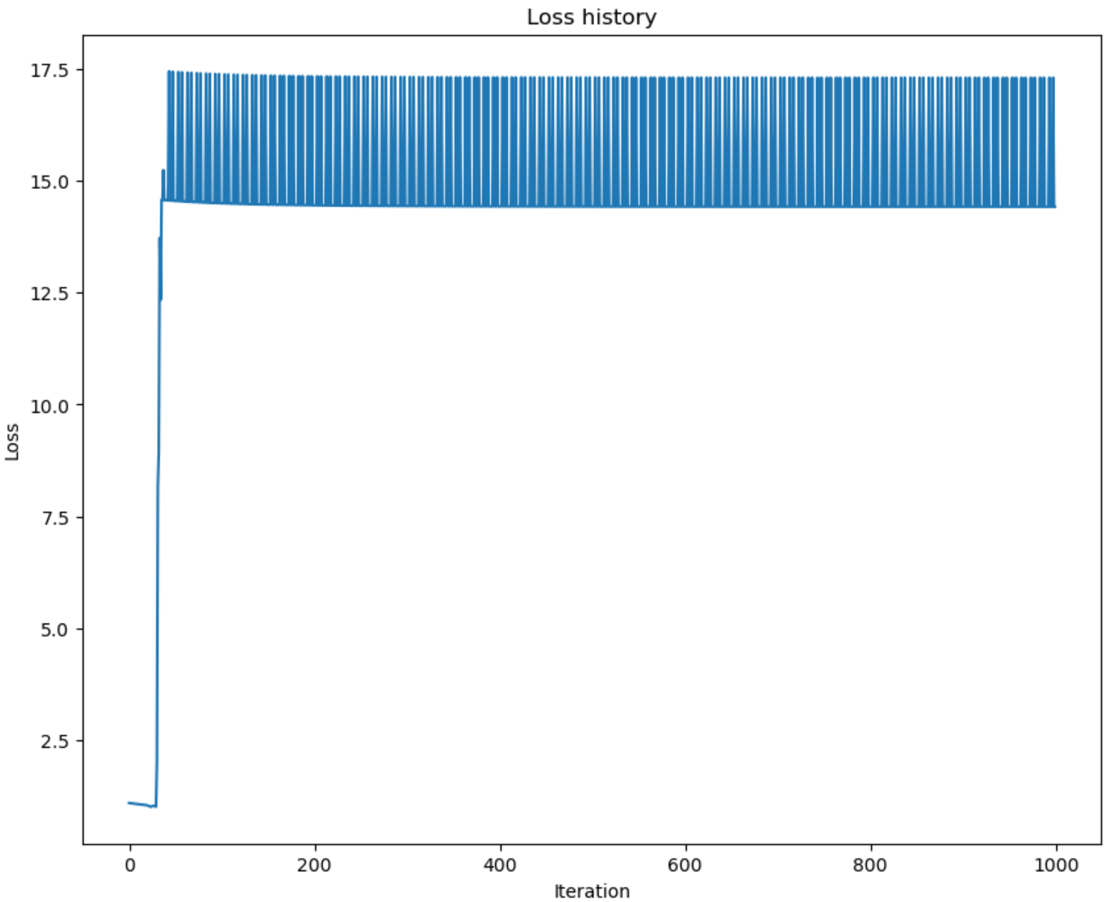
```
Final training loss 14.417785677623234
```

[2605]:

```
# Plot the training loss curve. The loss in the curve should be decreasing (20%)
plt.plot(epoch_loss)
plt.title("Loss history")
plt.xlabel("Iteration")
plt.ylabel("Loss")
```

[2605]:

```
Text(0, 0.5, 'Loss')
```



1.2 Survey (1%)

1.2.1 Question:

How many hours did you spend on this assignment?

1.2.2 Your Answer: 6 Hours.