NN-Pytorch

March 28, 2023

1 Fully Connected Neural Networks in PyTorch (65 points)

In this assignment, you will implement a fully connected neural net with one hidden layer in PyTorch and evaluate it on 2 artificial datasets: *flower* and *spiral*.

1.1 Write Your Name Here:

2 Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version showing the code and the output of all cells, and save it in the same folder that contains the notebook file.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly.
7. Submit **both** your PDF and notebook on Canvas.
8. Make sure your your Canvas submission contains the correct files by downloading it after posting it on Canvas.

[ ]:
```
import numpy as np
from scipy.sparse import coo_matrix
from numpy.random import randn, randint
from numpy.linalg import norm
import matplotlib.pyplot as plt

import torch

import utils

np.random.seed(1)
```

2.1 Forward Propagation and Loss computation in a Neural Network (20 points)

Implement the function `nnLoss()`. You will need to write code that computes the loss, based on the current values of the parameters. This will be done by computing the values at every layer in the network, usign forward propagation equations.

You are supposed to write the code for computing the loss yourself. In particular, do not use functions from PyTorch (e.g. from the `torch.nn` module) that define the NN structure and that compute the loss.

```
[ ]: def nnLoss(X, groundTruth, params, decay):
         # Unpack parameters from 'params'
         W1, b1, W2, b2 = params

         # You might fidn this useful when computing the 'ramp' output for ReLU␣
      ↪neurons.
         zero = torch.FloatTensor([0])

         ############ YOUR CODE HERE ############

         # Forward propagation: compute Z2, A2, Z3.
         Z2 =
         A2 =

         Z3 =

         # Compute the output softmax probabilities in A3.
         # Do not forget to address overflow, usign the approach from a previous␣
      ↪assignment.
         A3 =

         # Compute the negative log-likelihood loss + regularization loss.
         loss =

         ########################################


         return loss
```

2.2 Training the NN using gradient descent (20 points)

```
[ ]: def nnTrain(X, y, n_y, hyperparams):
         """Train the LR model using gradient descent in PyTorch.
         Args:
             X: The data matrix as a NumPy array (numFeatures x numExamples)
             y: The vector of labels as a NumPy array.
             n_y: The number of classes.
             hyperparams: a tuple containing (hidden_size, decay, learning_rate,␣
      ↪num_epochs)

         Returns:
```

```
        W1, b1, W2, b2: The torch Tensors containing the NN parameters.
    """
    ftype = torch.FloatTensor

    # Load training data into Tensors that do not require gradients.
    X = torch.from_numpy(X).type(ftype)

    n_x = X.shape[0]
    m = X.shape[1]

    groundTruth = coo_matrix((np.ones(m, dtype = np.uint8), (y, np.arange(m))).
↪toarray()
    groundTruth = torch.from_numpy(groundTruth).type(ftype)

    # Set hyper-parameters.
    n_h = hyperparams[0]
    decay = hyperparams[1]
    learning_rate = hyperparams[2]
    num_epochs = hyperparams[3]

    # Randomly initialize parameters.
    W1 = 0.01 * torch.randn(n_h, n_x).type(ftype)
    W1.requires_grad_(True)
    b1 = torch.zeros((n_h, 1)).type(ftype)
    b1.requires_grad_(True)
    W2 = 0.01 * torch.randn(n_y, n_h).type(ftype)
    W2.requires_grad_(True)
    b2 = torch.zeros((n_y, 1)).type(ftype)
    b2.requires_grad_(True)

    # Gradient descent loop.
    # In this section, run gradient descent for num_epochs.
    # At each epoch, first compute the loss tensor, then update W and b params
    # using the gradient automatically computed by calling loss.backtrack().
    for epoch in range(num_epochs):
        # Forward propagation.
        loss = nnLoss(X, groundTruth, (W1, b1, W2, b2), decay)

        if (epoch + 1) % 1000 == 0:
            print("Epoch %d: cost %f" % (epoch + 1, loss))

        ############ YOUR CODE HERE ############
    ########################################

    return W1, b1, W2, b2
```

2.3 Forward Propagation for Prediction using a Neural Network (20 points)

Implement the function `nnPredict()`, which computes the most likely label for each input instance.

```
[ ]: def nnPredict(X, params):
         """Computes and returns the accuracy on the test instances.
         Args:
             X: the input example(s), represented in NumPy.
             params: the NN parameters, representend iun NumPy.

         Returns:
             pred - the most likely label(s), one label for each example in X.
         """

         # Unpack parameters from 'params'.
         W1, b1, W2, b2 = params

         ## ---------- YOUR CODE HERE -------------------------------------------
         #  Instructions: Compute for each instance the most likely label.
         #  Hint: You do not need to compute the probability of each label to find
         #        the most likely label, use the pre-softmax,  logit scores instead.

         # Compute logit scores using forward propagation equations.
         Z2 =
         A2 =
         Z3 =

         # Compute predictions, i.e. labels with the largest logit scores from Z3.
         pred =

         # ----------------------------------------------------------------------
```

```
    return pred
```

2.3.1 Evaluate the NN model on the *flower* dataset.

The expected accuracy on this dataset is 88.5%.

[ ]:
```
X, y, n_y = utils.load_flower_dataset()

# Train a NN with one hidden layer of size 20, for 20,000 epochs,
# with a learning rate of 0.05, and an L2 decay of 0.
hyperparams = 20, 0, 0.05, 20000

params = nnTrain(X, y, n_y, hyperparams)

# Get params in NumPy arrays.
np_params = tuple(param.detach().numpy() for param in params)

pred = nnPredict(X, np_params)
acc =  # YOUR CODE HERE
print('Accuracy: %0.3f%%.' % (acc * 100))

# Plot the decision boundary.
utils.plot_decision_boundary(lambda x: nnPredict(x, np_params), X, y)
plt.title("Neural Network")
plt.savefig('flower-boundary.jpg')
plt.show()
```

2.3.2 Evaluate the NN model on the *spiral* dataset.

The expected accuracy on this dataset is 99.0%.

[ ]:
```
X, y, n_y = utils.load_spiral_dataset()

# Train a NN with one hidden layer of size 100, for 10,000 epochs,
# with a learning rate of 1, and an L2 decay of 0.001.
hyperparams = 100, 0.001, 1, 10000

params = nnTrain(X, y, n_y, hyperparams)

# Get params in NumPy arrays.
np_params = tuple(param.detach().numpy() for param in params)

pred = nnPredict(X, np_params)
acc =  # YOUR CODE HERE
print('Accuracy: %0.3f%%.' % (acc * 100))

# Plot the decision boundary.
```

```
utils.plot_decision_boundary(lambda x: nnPredict(x, np_params), X, y)
plt.title("Neural Network")
plt.savefig('spiral-boundary.jpg')
plt.show()
```

2.4 Bonus points (20 + 20 + 20 + ... points)

1. Use the `torch.nn` module to implement the neural network and run the same experiments. Feel free to create a separate notebook for this. Relevant functions and classes are `torch.nn.Sequential`, `torch.nn.Linear`, `torch.nn.ReLU`, `torch.nn.CrossEntropyLoss`, `torch.optim.SGD`.
2. Verify experimentally only the positive conclusions (parts you answered “yes” to) that you reached for the Universal Approximation theory problem. Write your code in NumPy, PyTorch with `autograd`, or PyTorch with `torch.nn` and `torch.optim`.
3. Implement in NumPy the gradient formula that you derived for the Computation Graphs theory problem. Set all the variables to random values. Check the value of the analytical gradient against the numerical gradient and the gradient computed through autograd in PyTorch.
4. Anything extra goes here.

[ ]:
```

```

2.5 Analysis (5 points)

Include here a nicely formatted report of the results, comparisons with the LR performance, etc. Include explanations and any insights you can derive from the algorithm behavoir and the results. This section is important, so make sure you address it appropriately.