LR-Pytorch

November 14, 2024

1 Logistic Regression in PyTorch

In this assignment, you will implement the logistic regression model using PyTorch and evaluate it on 3 artificial datasets: *shapes*, *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 1. The Logistic Regression model (40 points)

In this part, you will write code to:

1. Train the logistic regression model (20 points)

2. Compute the model predictions and accuracy (20 points).

Implement the function `softmaxTrain()`. You will need to write code for the following:

1. **Tensors**: Create pytorch tensors for the input data and the model parameters. Specify that gradients are to be computed w.r.t. parameters. Initialize the bias vector with zeros, and the weight matrix with a standard Gaussian multiplied with 0.01.

2. **Loss**: Write code that computes the loss variable, based on the current values of the parameters. Once the loss is computed, the gradient w.r.t. the parameters will be automatically calculated by calling loss.backward(). 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 compute the cross entropy loss.

[ ]:
```
def softmaxTrain(instances, labels, numClasses):
    """Train the LR model using gradient descent in PyTorch.
    Args:
        instances: The data matrix as a tensor (numFeatures x numExamples)
        labels: The vector of labels as a tensor.
        numClasses: The number of classes.

    Returns:
        W, b: The tensors containing the LR parameters.
    """

    numExamples = instances.shape[1]
    inputSize = instances.shape[0]

    # Load training data into Tensors that do not require gradients.
    groundTruth = coo_matrix((np.ones(numExamples, dtype = np.uint8),
                              (labels.numpy(), np.arange(numExamples)))).
↪toarray()
    groundTruth = torch.from_numpy(groundTruth).type(torch.FloatTensor)

    # Set hyper-parameters.
    learning_rate = 1.0
    decay = 0.001
    num_epochs = 200

    # Randomly initialize parameters W and b, using same distribution / values␣
↪as in previous assignment.
    W = # YOUR CODE HERE
    b = # YOUR CODE HERE


    # 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
    # using the gradient automatically computed by calling loss.backtrack().
```

```
    for epoch in range(num_epochs):
        ## ---------- YOUR CODE HERE --------------------------------------
        #  Instructions: Compute the loss tensor, see NumPy solution from previous assignment.

        loss = None









        ## ---------------------------------------------------------------

        if epoch % 10 == 0:
            print("Epoch %d: cost %f" % (epoch, loss))

        # Use autograd to compute the backward pass. This call will compute the
        # gradient of loss with respect to all Tensors with requires_grad = True.
        # After this call W.grad and b.grad will be Tensors holding the gradient
        # of the loss with respect to W and b respectively.
        loss.backward()

        # Update weights using gradient descent; W and b are Tensors,
        # W.grad and b.grad are Tensors.
        W.data -= learning_rate * W.grad.data
        b.data -= learning_rate * b.grad.data

        # Manually zero the gradients after updating weights
        W.grad.zero_()
        b.grad.zero_()


    return W, b
```

Implement the function `softmaxTest()`, which computes the most likely label for each input instance and uses that to compute and return the final accuracy.

[ ]:
```
def softmaxTest(W, b, instances, labels):
    """Computes and returns the accuracy on the test instances.
    Args:
        W, b - tensors with model parameters trained using softmaxTrain.
```

```
        instances: The data matrix as a tensor (numFeatures x numExamples)
        labels: The vector of instance labels as a tensor.

    Returns:
        acc - the accuracy
    """

    ## ---------- 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.

    acc = 1.0

    # ----------------------------------------------------------------------

    return acc
```

2.2 2. Experimental Evaluations (10 points)

In this part, you will train and evaluate the LR model on 3 artificial datasets: *shapes*, *flower*, and *spiral*.

2.2.1 Evaluate the LR model on the *flower* dataset.

[ ]:
```
instances, labels, numClasses = utils.load_flower_dataset()

instances = torch.from_numpy(instances).type(torch.FloatTensor)
labels = torch.from_numpy(labels).type(torch.ByteTensor)

W, b = softmaxTrain(instances, labels, numClasses)
acc = softmaxTest(W, b, instances, labels)
print('Accuracy: %0.3f%%.' % (acc * 100))

# Plot the decision boundary.
utils.plot_decision_boundary(lambda x: np.argmax((W.detach().numpy().dot(x) + b.
 ↪detach().numpy()), axis = 0),
                             instances.numpy(), labels.numpy())
plt.title("Softmax Regression")
plt.savefig('flower-boundary.jpg')
plt.show()
```

2.2.2 Evaluate the LR model on the *spiral* dataset.

[ ]:
```
instances, labels, numClasses = utils.load_spiral_dataset()

instances = torch.from_numpy(instances).type(torch.FloatTensor)
```

```
labels = torch.from_numpy(labels).type(torch.ByteTensor)

W, b = softmaxTrain(instances, labels, numClasses)
acc = softmaxTest(W, b, instances, labels)
print('Accuracy: %0.3f%%.' % (acc * 100))

# Plot the decision boundary.
utils.plot_decision_boundary(lambda x: np.argmax((W.detach().numpy().dot(x) + b.
 ↪detach().numpy()), axis = 0),
                             instances.numpy(), labels.numpy())
plt.title("Softmax Regression")
plt.savefig('spiral-boundary.jpg')
plt.show()
```

2.2.3 Evaluate the LR model on the *shapes* dataset (10 points)

[ ]:
```
########## YOUR CODE HERE ##########
```

[ ]:
```

```

2.3 Bonus (10 points)

- Use the the cross entropy loss function available through `torch.nn` module to implement `softmaxTrain()` and run the same experiments.
- Anything extra goes here.

[ ]:
```

```

2.4 Analysis (10 points)

Include here a nicely formatted report of the results, comparisons. 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.

[ ]:
```

```