HW1

January 12, 2023

#

Homework 1

In this homework you will code a simple 1 layer linear regression problem using PyTorch. You can use very similar code to the example shown in Lab 1, but will need to alter it to accomodate linear (instead of logistic) regression.

0.1 Instructions

Follow the outline code below and fill in code where you see "<YOUR CODE HERE>". Also, answer any questions in the provided"<YOUR ANSWER HERE>" space. Once completed, upload the Jupyter Notebook, a PDF export AND a HTML export version of your Jupyter notebook to Canvas under "Homework 1". This homework is due **Friday January 13th at 11:59pm.**

1 The Data

We now consider a list of statements and their sentiment score (provided). Sentiment scores reflect the sentiment of a sentence, with postive values reflecting positive sentiment and negative values reflecting negative sentiment. The magnitude of the score reflects the magnitude of the sentiment (negative or positive). A sentiment score of 0, means the sentence has neutral sentiment. We want to train a single layer linear model using the training data and then test our model on the test data provided below.

[ ]:
```
import torch
# The Data
training_data = [("This product is terrible, it broke on the first day".
 ↪split(), -0.53),
       ("Wednesday is the worst day of the week. ".split(), -0.61),
       ("I love going on hikes and eating tasty food".split(), 0.41),
       ("Blue is my favorite color, it makes me feel happy".split(), 0.62),
       ("Today I have to go to school".split(), -0.02)]

test_data = [("I am dreading my visit to the dentist.".split(), -0.31),
             ("I get to see the circus today, I am excited.".split(), 0.34)]

word_to_ix = {}
for sent, _ in training_data + test_data:
    for word in sent:
```

```
        if word not in word_to_ix:
            word_to_ix[word] = len(word_to_ix)
print(word_to_ix)

VOCAB_SIZE = len(word_to_ix)
```

```
{'This': 0, 'product': 1, 'is': 2, 'terrible,': 3, 'it': 4, 'broke': 5, 'on': 6,
'the': 7, 'first': 8, 'day': 9, 'Wednesday': 10, 'worst': 11, 'of': 12, 'week.':
13, 'I': 14, 'love': 15, 'going': 16, 'hikes': 17, 'and': 18, 'eating': 19,
'tasty': 20, 'food': 21, 'Blue': 22, 'my': 23, 'favorite': 24, 'color,': 25,
'makes': 26, 'me': 27, 'feel': 28, 'happy': 29, 'Today': 30, 'have': 31, 'to':
32, 'go': 33, 'school': 34, 'am': 35, 'dreading': 36, 'visit': 37, 'dentist.':
38, 'get': 39, 'see': 40, 'circus': 41, 'today,': 42, 'excited.': 43}
```

[ ]:
```
# Transform sentence into BoW
def make_bow_vector(sentence, word_to_ix):
    vec = torch.zeros(len(word_to_ix))
    for word in sentence:
        vec[word_to_ix[word]] += 1
    return vec.view(1, -1)


def make_target(label):
    # return the label as a tensor (instead of a float)
    return(torch.FloatTensor([[label]]))

# Example
print("Sample Input:", training_data[0][0])
print(make_bow_vector(training_data[0][0], word_to_ix))
print(make_bow_vector(training_data[0][0], word_to_ix).size())

print("")

print("Sample Output:", training_data[0][1])
print(make_target(training_data[0][1]))
print(make_target(training_data[0][1]).size())
```

```
Sample Input: ['This', 'product', 'is', 'terrible,', 'it', 'broke', 'on', 'the',
'first', 'day']
tensor([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0.]])
torch.Size([1, 44])

Sample Output: -0.53
tensor([[-0.5300]])
torch.Size([1, 1])
```

1.1 Create the nn.Module for Linear Regression

[ ]:
```
import torch.nn as nn
class LinearRegression(nn.Module):

    def __init__(self, output_size, input_size):
        super(LinearRegression, self).__init__()
        self.linear = nn.Linear(input_size,output_size)  # Add one layer of␣
 ↪linear regression

    def forward(self, bow_vec):
        return self.linear(bow_vec)

model = LinearRegression(1, len(word_to_ix)) # Input the correct output and␣
 ↪input sizes
```

So lets train! To do this, choose an appropriate loss function for linear regression from the list provided here. Hint: it should be different than the loss we used in Lab 1.

You can use the same optimizer from Lab 1 (torch.optim.SGD) or you can choose a different optimizer from the list here. However, we will need to smaller the learning rate (try 0.01).

[ ]:
```
loss_function = nn.MSELoss()
optimizer = torch.optim.SGD(params=model.parameters(),lr=0.01) # use a learning␣
 ↪rate = 0.01
```

Now we train the model. Use 100 epochs to train the model.

[ ]:
```
for epoch in range(100):
    for instance, label in training_data:
        # Step 1. PyTorch accumulates gradients. DO NOT CHANGE THIS
        model.zero_grad()

        # Step 2. Make our BOW vector and also we must wrap the target in a␣
 ↪Tensor.
        bow_vec = make_bow_vector(instance, word_to_ix)
        target = make_target(label)

        # Step 3. Run our forward pass.
        pred = model(bow_vec)

        # Step 4. Compute the loss, gradients, and update the parameters by␣
 ↪calling optimizer.step()
        loss = loss_function(pred, target)
        loss.backward()
        optimizer.step()

print("True Test Values: ", test_data)
```

```
with torch.no_grad():
    for instance, label in test_data:
        bow_vec = make_bow_vector(instance, word_to_ix)
        pred = model(bow_vec)
        print(pred)
```

```
True Test Values:  [(['I', 'am', 'dreading', 'my', 'visit', 'to', 'the',
'dentist.'], -0.31), (['I', 'get', 'to', 'see', 'the', 'circus', 'today,', 'I',
'am', 'excited.'], 0.34)]
tensor([[-0.0368]])
tensor([[-0.1063]])
```

1.2 Questions

1. Explain what the output represents.

The output represents a sentiment score with positive values indicating positive sentiment and negative values representing negative sentiment and the magnitude reflects how strong the sentiment is.

2. Use the "Run All" function to run the whole Jupyter Notebook multipe times. Do you get the same results on your test set each time? Why or why not?

No, we do not see the same output each time because the weight learning process is not deterministic, i.e. the intial weights are assigned randomly which means subsequent updates will be different based on the intial weights.

It is also possible that the learning process has not converged completely, due to which we see different values each time we train the model for 100 epochs. This could be because we have very little training data, additionally, the number of records compared to the number of features in the bag of words representation is also low, or because the learning rate is very high. Further diagnostics would be required/

3. Do you think this is a good model for this data? Why or why not?

This model is not good for this data, since the outputs are unconstrained. Which means that sentiment scores can take on any arbitrary real value which makes it hard to draw comparisons between scores. Also, the training data scores seem to be constained between -1 and 1, so we should try to make sure that our model is constrained in a similar manner so that we can get meaningful results.

Moreover, the model is still a simple linear model that does not capture any complex non-linearities that might be present in the data.

2 Bonus (optional)

Can you alter your nn.Module LinearRegression to include two linear layers? Write code for a new nn.Module called "TwoLayerLinearRegression" and re-train your model. Then print the predictions of your train model on the test set.

[ ]:
```
import torch.nn as nn
class TwoLinearRegression(nn.Module):

    def __init__(self, output_size, input_size):
        super(TwoLinearRegression, self).__init__()
        self.linear1 = nn.Linear(input_size,input_size)
        self.linear2 = nn.Linear(input_size, output_size)

    def forward(self, bow_vec):
        l1 = self.linear1(bow_vec)
        pred = self.linear2(l1)
        return pred

model = TwoLinearRegression(1, len(word_to_ix)) # Input the correct output and␣
 ↪input sizes
```

[ ]:
```
loss_function = nn.MSELoss()
optimizer = torch.optim.SGD(params=model.parameters(),lr=0.01) # use a learning␣
 ↪rate = 0.01
```

[ ]:
```
for epoch in range(100):
    for instance, label in training_data:
        # Step 1. PyTorch accumulates gradients. DO NOT CHANGE THIS
        model.zero_grad()

        # Step 2. Make our BOW vector and also we must wrap the target in a␣
 ↪Tensor.
        bow_vec = make_bow_vector(instance, word_to_ix)
        target = make_target(label)

        # Step 3. Run our forward pass.
        pred = model(bow_vec)

        # Step 4. Compute the loss, gradients, and update the parameters by␣
 ↪calling optimizer.step()
        loss = loss_function(pred, target)
        loss.backward()
        optimizer.step()

print("True Test Values: ", test_data)
with torch.no_grad():
    for instance, label in test_data:
        bow_vec = make_bow_vector(instance, word_to_ix)
        pred = model(bow_vec)
        print(pred)
```

```
True Test Values:  [(['I', 'am', 'dreading', 'my', 'visit', 'to', 'the',
'dentist.'], -0.31), (['I', 'get', 'to', 'see', 'the', 'circus', 'today,', 'I',
'am', 'excited.'], 0.34)]
tensor([[-0.1628]])
tensor([[0.0516]])
```