UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB

Sentiment Analysis with Neural Networks

REVIEW

HISTORY

Meets Specifications

Congratulations!

- Your submission is now **excellent!!**
- All of your tests are passing
- All of your answers are correct.
- Your code is well written and easy to understand.

Bravo!

You have demonstrated a good understanding of the concepts in Project 6 and the ability to implement those concepts in Python.

It has been a pleasure reviewing your project.

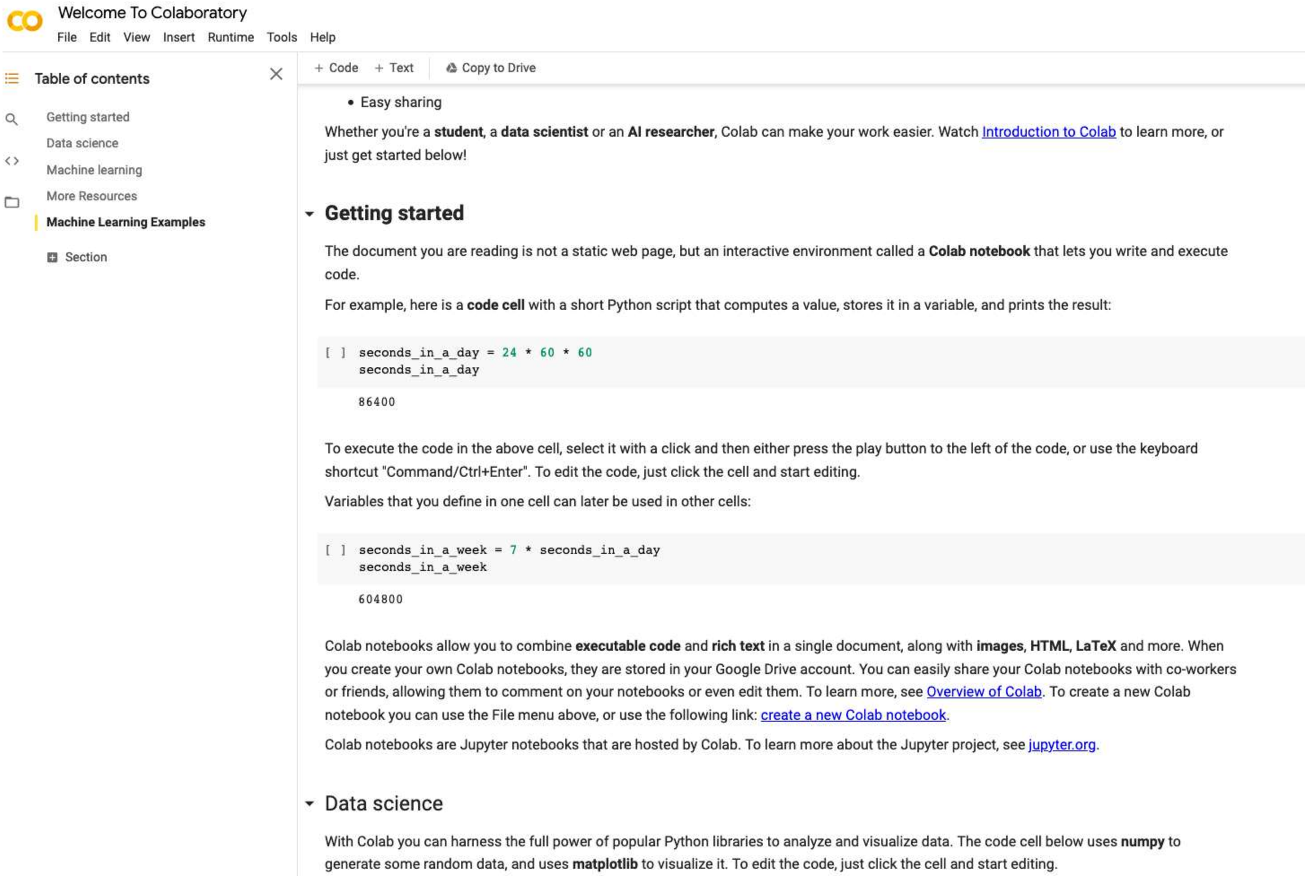
Good Luck on your final 2 projects in the nanodegree!

Now that you've completed this project (and since you are nearing

the end of this nanodegree) . . .

You might be interested to learn about Google Collab - https://colab.research.google.com/notebooks/intro.ipynb

- This is **Google's** Free Jupyter Notebooks Environment (including **"Free access to GPUs"**) that has most common 3rd party libraries already installed
- You can run code just like you have been doing in your nanodegree



- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day

    86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
    seconds_in_a_week

    604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Data science

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

Importing Twits

Print the number of twits in the dataset.

I was your previous reviewer. Nothing to add here.

Preprocessing the Data

The function `preprocess` correctly lowercases, removes URLs, removes ticker symbols, removes punctuation, tokenizes, and removes any single character tokens.

I was your previous reviewer. Nice job - your regular expression for urls is now properly removing all urls!

Preprocess all the twits into the `tokenized` variable.

I was your previous reviewer. Nothing to add here.

Create a bag of words using the tokenized data.

I was your previous reviewer. Nothing to add here.

Remove most common and rare words by defining the following variables: `freqs` , `low_cotoff` , `high_cutoff` , `K_most_common` .

I was your previous reviewer. Nothing to add here.

Defining the variables : 'vacab', 'id2vocab' and 'filtered' correctly.

I was your previous reviewer. Nothing to add here.

Neural Network

The init function correctly initializes the following parameters: `self.vocab_size` , `self.embed_size` , `self.lstm_size` , `self.lstm_layers` , `self.dropout` , `self.embedding` , `self.lstm` , and `self.fc` .

I was your previous reviewer. Nothing to add here.

The 'init_hidden' function generates a hidden state

I was your previous reviewer. Nothing to add here.

The 'forward' function performs a forward pass of the model the parameter input using the hidden state.

I was your previous reviewer. Nothing to add here.

Training

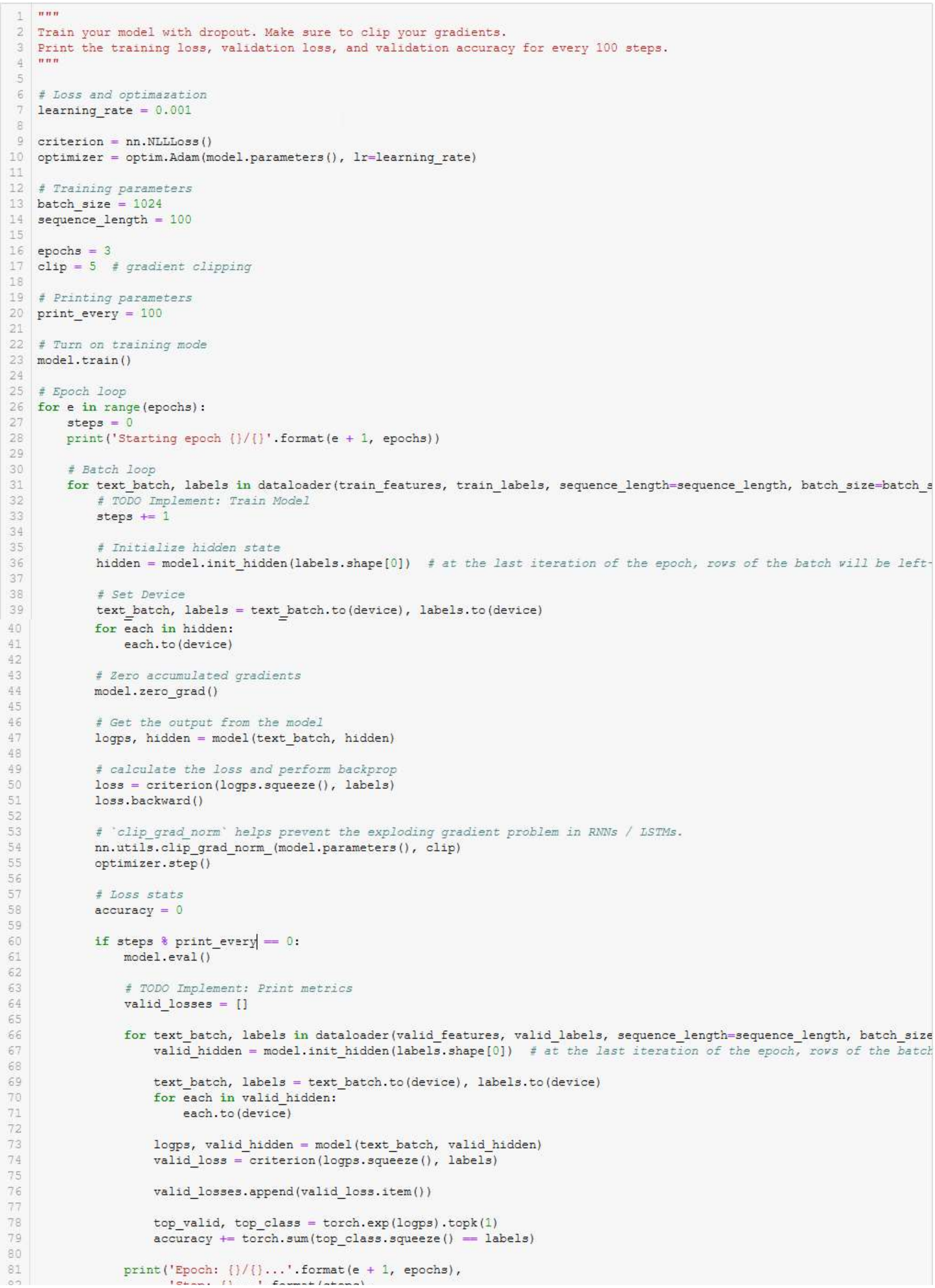
Correctly split the data into `train_features`, `valid_features`, `train_labels`, and `valid_labels`.

I was your previous reviewer. Nothing to add here.

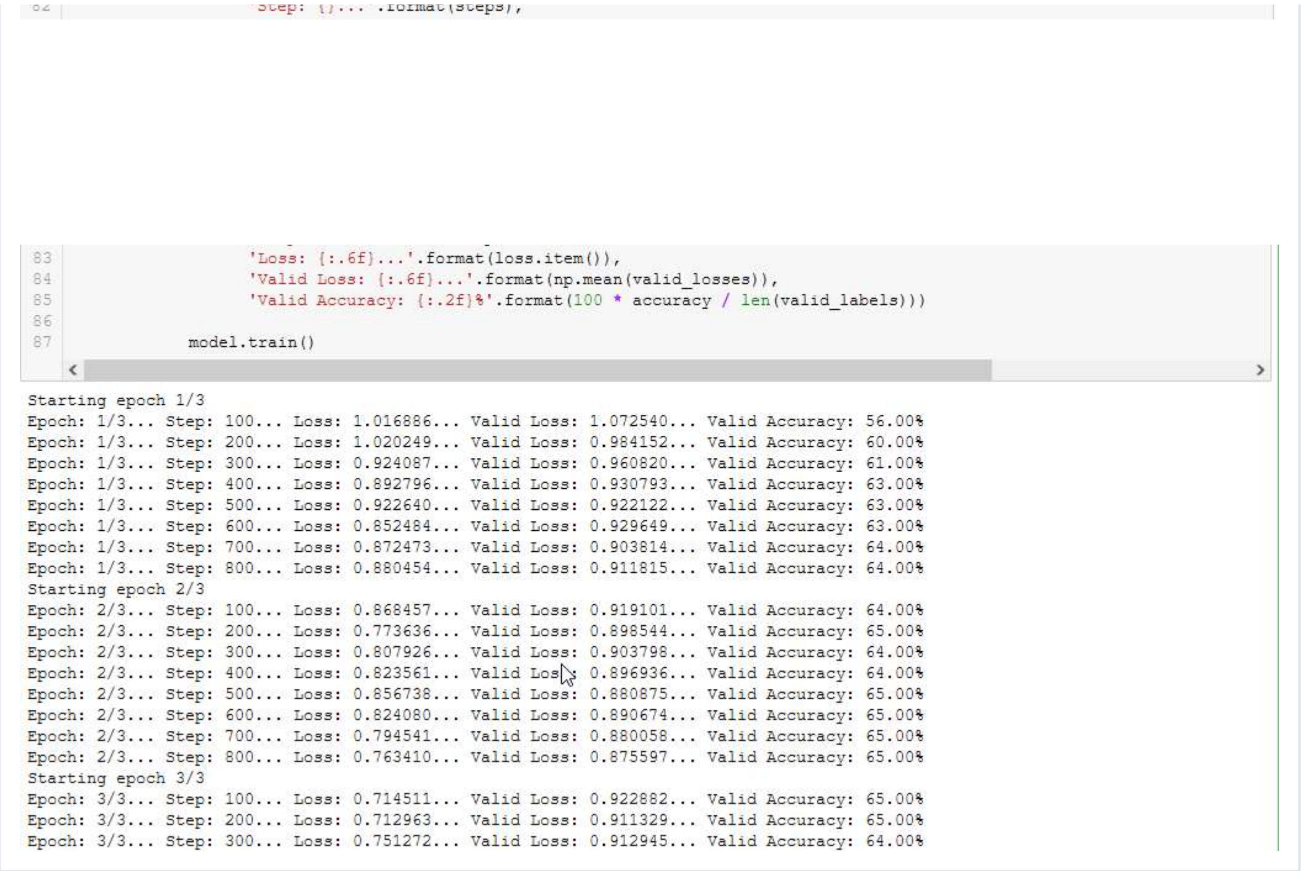
Train your model with dropout and clip the gradient. Print out the training progress with the loss and accuracy.

Excellent Job!

- You are calling **optimizer.zerograd()** properly.
- You are getting output from the model.
- You are using **criterion** to calculate the loss and calling **backward()** to perform backpropagation.
- You are using **clip_grad_norm** to prevent exploding gradient problem.
- You are calculating and printing **training loss, validation loss, and validation accuracy for every 100 steps**



```
"""
Train your model with dropout. Make sure to clip your gradients.
Print the training loss, validation loss, and validation accuracy for every 100 steps.
"""

# Loss and optimazation
learning_rate = 0.001

criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training parameters
batch_size = 1024
sequence_length = 100

epochs = 3
clip = 5  # gradient clipping

# Printing parameters
print_every = 100

# Turn on training mode
model.train()

# Epoch loop
for e in range(epochs):
    steps = 0
    print('Starting epoch {}/{}'.format(e + 1, epochs))

    # Batch loop
    for text_batch, labels in dataloader(train_features, train_labels, sequence_length=sequence_length, batch_size=batch_s
        # TODO Implement: Train Model
        steps += 1

        # Initialize hidden state
        hidden = model.init_hidden(labels.shape[0])  # at the last iteration of the epoch, rows of the batch will be left-

        # Set Device
        text_batch, labels = text_batch.to(device), labels.to(device)
        for each in hidden:
            each.to(device)

        # Zero accumulated gradients
        model.zero_grad()

        # Get the output from the model
        logps, hidden = model(text_batch, hidden)

        # calculate the loss and perform backprop
        loss = criterion(logps.squeeze(), labels)
        loss.backward()

        # `clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
        nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        # Loss stats
        accuracy = 0

        if steps % print_every == 0:
            model.eval()

            # TODO Implement: Print metrics
            valid_losses = []

            for text_batch, labels in dataloader(valid_features, valid_labels, sequence_length=sequence_length, batch_size
                valid_hidden = model.init_hidden(labels.shape[0])  # at the last iteration of the epoch, rows of the batch

                text_batch, labels = text_batch.to(device), labels.to(device)
                for each in valid_hidden:
                    each.to(device)

                logps, valid_hidden = model(text_batch, valid_hidden)
                valid_loss = criterion(logps.squeeze(), labels)

                valid_losses.append(valid_loss.item())

                top_valid, top_class = torch.exp(logps).topk(1)
                accuracy += torch.sum(top_class.squeeze() == labels)

            print('Epoch: {}/{}...'.format(e + 1, epochs),
                  'Step: {}...'.format(steps),
```



```
                      'Step: {}...'.format(steps),
```

```
                      'Loss: {:.6f}...'.format(loss.item()),
                      'Valid Loss: {:.6f}...'.format(np.mean(valid_losses)),
                      'Valid Accuracy: {:.2f}%'.format(100 * accuracy / len(valid_labels)))

              model.train()
```

```
Starting epoch 1/3
Epoch: 1/3... Step: 100... Loss: 1.016886... Valid Loss: 1.072540... Valid Accuracy: 56.00%
Epoch: 1/3... Step: 200... Loss: 1.020249... Valid Loss: 0.984152... Valid Accuracy: 60.00%
Epoch: 1/3... Step: 300... Loss: 0.924087... Valid Loss: 0.960820... Valid Accuracy: 61.00%
Epoch: 1/3... Step: 400... Loss: 0.892796... Valid Loss: 0.930793... Valid Accuracy: 63.00%
Epoch: 1/3... Step: 500... Loss: 0.922640... Valid Loss: 0.922122... Valid Accuracy: 63.00%
Epoch: 1/3... Step: 600... Loss: 0.852484... Valid Loss: 0.929649... Valid Accuracy: 63.00%
Epoch: 1/3... Step: 700... Loss: 0.872473... Valid Loss: 0.903814... Valid Accuracy: 64.00%
Epoch: 1/3... Step: 800... Loss: 0.880454... Valid Loss: 0.911815... Valid Accuracy: 64.00%
Starting epoch 2/3
Epoch: 2/3... Step: 100... Loss: 0.868457... Valid Loss: 0.919101... Valid Accuracy: 64.00%
Epoch: 2/3... Step: 200... Loss: 0.773636... Valid Loss: 0.898544... Valid Accuracy: 65.00%
Epoch: 2/3... Step: 300... Loss: 0.807926... Valid Loss: 0.903798... Valid Accuracy: 64.00%
Epoch: 2/3... Step: 400... Loss: 0.823561... Valid Loss: 0.896936... Valid Accuracy: 64.00%
Epoch: 2/3... Step: 500... Loss: 0.856738... Valid Loss: 0.880875... Valid Accuracy: 65.00%
Epoch: 2/3... Step: 600... Loss: 0.824080... Valid Loss: 0.890674... Valid Accuracy: 65.00%
Epoch: 2/3... Step: 700... Loss: 0.794541... Valid Loss: 0.880058... Valid Accuracy: 65.00%
Epoch: 2/3... Step: 800... Loss: 0.763410... Valid Loss: 0.875597... Valid Accuracy: 65.00%
Starting epoch 3/3
Epoch: 3/3... Step: 100... Loss: 0.714511... Valid Loss: 0.922882... Valid Accuracy: 65.00%
Epoch: 3/3... Step: 200... Loss: 0.712963... Valid Loss: 0.911329... Valid Accuracy: 65.00%
Epoch: 3/3... Step: 300... Loss: 0.751272... Valid Loss: 0.912945... Valid Accuracy: 64.00%
```

Making Predictions

The `predict` function correctly prints out the prediction vector from the trained model.

I was your previous reviewer. Nothing to add here.

Answer what the prediction of the model is and the uncertainty of the prediction.

I was your previous reviewer. Nothing to add here.

DOWNLOAD PROJECT

RETURN TO PATH