Inference

Let's put all these pieces together! One of the coolest ways to test a model like this is to give it user-generated data, without any true label, and see what happens. So, in this case, that data will just be a single string: a review that you can write and here's just one `test_review` as an example:

```
# negative test review
test_review_neg = 'The worst movie I have seen; acting was terrible and I
```

We can see that this review is a negative one, but let's see if our model can identify it's sentiment correctly!

Our task is to write a `predict` function that takes in a trained model, a `test_review` like this one that is just normal text and punctuation, a `sequence_length` for padding.

The process by which you make predictions based on user data, is called **inference**.

Pre-process the `test_review`

The first thing we'll have to do it to process the `test_review`, so that it is converted into a tensor that our model can see as input. In fact, this involves quite a lot of pre-processing, but nothing that you haven't seen before!

I broke this down into a series of steps.

I have a helper function `tokenize_review` that is responsible for doing some data processing on my test_review.

It takes in my test_review, and then does a couple of things:

1. First, I convert my `test_review` to lowercase, and remove any punctuation, so I'm left with all text.
2. Then I breaks it into individual words with split(), and I'm left with a list of words in the review.
3. I encode those words using the `vocab_to_int` dictionary that we already defined, near the start of this lesson.

Now, I am assuming a few things here, including: this review is one review, not a batch, and that this review only includes words *already* in our dictionary, and in this case that will be true, but you can add code to handle unknown characters, I just didn't do that in my model.

```
from string import punctuation

def tokenize_review(test_review):
    test_review = test_review.lower() # lowercase
    # get rid of punctuation
    test_text = ''.join([c for c in test_review if c not in punctuation])

    # splitting by spaces
    test_words = test_text.split()

    # tokens
    test_ints = []
    test_ints.append([vocab_to_int[word] for word in test_words])

    return test_ints
```

Okay, so this tokenize function returns a list of integers; my tokenized review!

Padding and converting into a Tensor

For my next couple of steps, I'm going to pad the ints, returned by the `tokenize_review` function and shape them into our `sequence_length` size; since our model was trained on sequence lengths of 200, I'm going to use the same length, here. I'll pad it using the `pad_features` function that we defined earlier.

Finally, I'm going to convert the padded result into a Tensor. So, these are all the steps, and I'm going to wrap this *all* up in my predict function.

```
def predict(net, test_review, sequence_length=200):
```

```
    net.eval()

    # tokenize review
    test_ints = tokenize_review(test_review)

    # pad tokenized sequence
    seq_length=sequence_length
    features = pad_features(test_ints, seq_length)

    # convert to tensor to pass into your model
    feature_tensor = torch.from_numpy(features)

    batch_size = feature_tensor.size(0)

    # initialize hidden state
    h = net.init_hidden(batch_size)

    if(train_on_gpu):
        feature_tensor = feature_tensor.cuda()

    # get the output from the model
    output, h = net(feature_tensor, h)

    # convert output probabilities to predicted class (0 or 1)
    pred = torch.round(output.squeeze())
    # printing output value, before rounding
    print('Prediction value, pre-rounding: {:.6f}'.format(output.item()))

    # print custom response
    if(pred.item()==1):
        print("Positive review detected!")
    else:
        print("Negative review detected.")
```

So, using the passed in arguments, I'm tokenizing my review using my helper function, then padding it using my pad function, and converting it into a Tensor that can be seen by my model.

Then, I'm passing this tensor into my trained net which will return an output of length one. With this output, I can grab the most likely class, which will be the rounded value 0 or 1; this is my prediction!

Lastly, I want to print out a custom message for a positive or negative detected review, and I'm doing that at the bottom of the above function!

You can test this out on sample positive and negative text reviews to see how this trained model behaves! Below, you can see how it identifies our negative test review correctly.

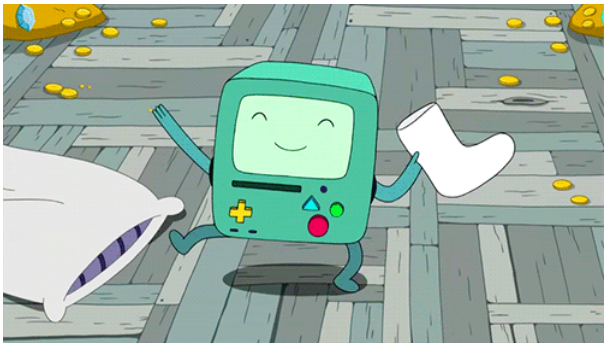
```
In [39]: # call function
         seq_length=200 # good to use the length that was trained on

         predict(net, test_review_neg, seq_length)
```

```
Prediction value, pre-rounding: 0.005722
Negative review detected.
```

Identifies negative review

Conclusion

Now that you have a trained model and a predict function, you can pass in any kind of text and this model will predict whether the text has a positive or negative sentiment. You can use this to try to find what words it associates with positive or negative sentiment.



Dancing Beemo from Adventure Time to celebrate!

Later, you'll learn how to deploy a model like this to a production environment so that it can respond to any kind of user data put into a web app!

For now, great job implementing so *many* kinds of recurrent neural networks!!

NEXT