Homework 7: Neural Network Classification

Due October 30th at 10pm

1 Deep Learning Math

1.1 Tracing a Feedforward Network

Consider a feedforward network for classifying vectors of length 3, with two hidden layers followed by a softmax output layer (no input weights).

Assume the following random initialization for the input weights to each unit:

Layer 1:

$h_{1,1}$: $[0.2, 0.15, 0.7]$

$h_{1,2}$: $[0.1, 0, 0.3]$

$h_{1,3}$: $[0, 0.5, 0.4]$

Layer 2:

$h_{2,1}$: $[0.45, 0.3, 0.9]$

$h_{2,2}$: $[1, -1, 1]$

Each hidden layer applies the ReLu activation function.

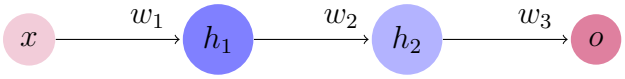
Calculate the model predictions for each of the following inputs:

	Features	Label
Input 1	$[9, 0, 4]$	1
Input 2	$[3, 5, 3]$	0
Input 3	$[2, 7, 0]$	1

Assume the inputs are batched (i.e., no weight update is done between inputs).

1.2 Gradient Descent

Consider a feedforward network with two hidden layers followed by a sigmoid output layer.



Each hidden layer has a single unit and applies a *Gaussian activation function:*

Gaussian activation: $g(x) = e^{-x^2}$ Partial derivative: $\frac{\delta g}{\delta x} = -2xe^{-x^2}$

Assume that the simple square loss function is used, as in the in-class gradient example.

Compute (1) the partial derivations for all three weights and (2) their updates. I have not given you the actual weight values, so you should give the equations for calculating these six terms.

Show your work.

2 Song Classification Revisited

Last week we used regression to classify songs by three artists: Taylor Swift, Drake, and Beyoncé. To input the lyrics to the regression classifier, you built features by hand. This week, we will revise our song classifier to use a powerful tool for automatic feature extraction: neural networks.

We will use a pre-trained neural network called DistilBERT to calculate word embeddings. DistilBERT is a more efficient version of a model called BERT. Here is the original BERT paper. Like many machine learning practitioners, we will use DistilBERT as a blackbox this week. But it is always good to understand what is happening under the hood of any model you use.

You can also test out the model online:
https://huggingface.co/distilbert/distilbert-base-uncased.

2.1 Simplifying the Data Preprocessing

Start by transferring your **`make_data`** function from last week to this week's file, **`nn.py`**. Previously, we turned the songs into vectors of numbers using **`make_features`**; we won't need this function anymore.

Instead, modify your **`make_data`** function to return a list of lyrics as strings. You do not need to lowercase the lyrics or preprocess them in any way.

2.2 Preprocessing Data for DistilBERT

Write a function called **`prep_bert_data`** to preprocess your data for DistilBERT. This will take the place of your old **`make_features`** function.

DistilBERT has its own tokenizer, which maps character sequences to token indexes. I have defined a global variable called **`tokenizer`** in the starter code, which constructs a DistilBERT tokenizer. Read the tokenizer class documentation to understand the tokenizer parameters.

Your function should take in a list of lyrics and call the tokenizer on each. We will use a maximum length so that we can control how much memory the model consumes. The starter code contains a global variable called **`MAX_LENGTH`**, set to 10 by default.

Your function should call the tokenizer with arguments to truncate and pad the songs to maximum length. Songs that are longer than the maximum length will be truncated; songs that are shorter will be padded (padding='max_length').

Your function should return the list of songs as a list of tensors, where each lyric is a tensor of longs (dtype=torch.long).

Note: Many examples in the documentation show examples that pass return_tensors="pt" as an argument to the tokenizer. I recommend that you **do not** do this; by default, the tokenizer returns a list of integers, which is easier for debugging. You can then turn these lists into tensors before returning them.

3 Neural Network Classification

This week's model is larger and will take longer to train. I have given you code in the starter file that periodically creates model *checkpoints*. These are save files for the model; if your program crashes or you kill it, you can reload the model from file and continue training where you left off.

Once you have a model that you are happy with, you will be able to load it from the checkpoint for inference, without retraining it.

Create a directory called **`ckpt`**. This is where the checkpoints will be saved. The code that I have given you checks if there is a saved checkpoint; if not, it initializes the model from scratch.

Warning: If you want to retrain your model from scratch (i.e., something has gone wrong), you will need to delete the contents of the checkpoint directory.

3.1 Incorporating DistilBERT

Next, modify your **`make_model`** function so that it feeds your input through a pretrained DistilBERT model. You will need to change your model definition in two places. First, in **`__init__`**, you will want to add the DistilBERT model that I have loaded at the top of the file.

Next, in **`forward`**, you will need to feed the input into DistilBERT. DistilBERT returns multiple outputs: we want only the **`last_hidden_state`**. You can retrieve this by calling **`last_hidden_state`** on the output from the DistilBERT layer.

3.2 Flattening

In our regression model, we represented each song as a one-dimensional vector and a batch as a two-dimensional tensor. The output from DistilBERT, however, is three-dimensional: the first dimension is the item in the batch, the second dimension is the number of tokens in the lyric, and the third is the number of units in the last hidden layer of the DistilBERT model: 768.

To feed the output of DistilBERT into a softmax layer for classification, we will need to flatten it to one dimension. After all, we want a prediction per song lyric, not per word in the song lyric.

Insert a flatten after the DistilBERT layer and before the Linear layer. Set **start_dim=1** so that your data is still batched.

You will also need to adjust the input dimensions to the Linear layer. The input dimensions should be the number of tokens (MAX_LENGTH) times the number of DistilBERT units (768).

Check in

You have now implemented a regression classifier on DistilBERT embeddings. Before adding any more layers, check to make sure that your current version runs.

4 Going Neural

A regression model is equivalent to a feedforward neural network with no hidden layers. Add 4 more Linear layers, each with a ReLU layer, to your model to make it more powerful.

You may find it useful to use the **`nn.Sequential`** function in your **`__init__)`** to group together layers (nn.Sequential documentation).

The first hidden layer should take the number of dimensions of the DistilBERT embeddings as input (768), and output half as many dimensions. Each subsequent layer should reduce the dimensionality by half.

You should apply the Flatten layer after the hidden layers, but before the softmax layer.

4.1 Questions

1. How well does your model perform? Report its validation accuracy for all three classes.

4.2 Fixing Our Inputs

It turns out that there is a subtle mistake in how we are feeding our data into the model. Because song lyrics can be different lengths, we are truncating long lyrics and padding short ones. However, we haven't told the model to ignore the padding. This has a small but noticeable effect on performance.

To fix this, we can pass in an *attention mask* to tell the model which tokens to ignore. An attention mask is a list containing a 1 for each token the model should pay attention to, and a 0 for each token it should ignore.

We will construct the attention mask for each item before feeding it into the model in **train**, **predict**, and **test**.

Write a function called **get_attention_mask**. This function should take a batched item from a dataloader (i.e., like what is stored in the variable X), and return a tensor containing attention masks for each item in the batch.

Lyric: "in red lipstick"
Tokenized Lyric: [CLS, in, red, lipstick, SEP, PAD, PAD, PAD, PAD, PAD]
Token Ids: [101, 1999, 2417, 22359, 102, 0, 0, 0, 0, 0]
Attention Mask: [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]

Hint: try constructing a list of attention mask lists first, and then converting it to a tensor before returning it.

You should then add a call to **get_attention_mask** in the **train**, **predict**, and **test** functions. You should pass a tuple containing the batch and its mask as input to the *model* call.

Finally, you will need to modify your **forward** function to unpack this tuple and use the attention mask:

```
embeddings = self.bert(input_ids=x[0],attention_mask=x[1]).last_hidden_state
```

4.3 Questions

1. How well does your model perform now that we are masking PAD tokens? Report its validation accuracy for all three classes.

5 Evaluation revisited

5.1 Sampling Predictions

It can be useful to manually inspect some of the model's predictions. Write a function called **`sample_and_print_predictions`**. This function should take 4 arguments: a dataset, the set of features for the dataset, the set of labels for the dataset, and the model.

It should randomly sample 10 indexes within the dataset, and retrieve model predictions for those 10 lyrics.

For each of the sampled lyrics, it should print the lyric, its predicted class, and its class:

```
show me a gray sky, a rainy cab ride was predicted to be class 0 and is class 0.0

(turn the lights on) was predicted to be class 1 and is class 1.0

cause i, i know places we can hide was predicted to be class 0 and is class 0.0
```

5.2 Adjusting for Class Imbalance

A major challenge of our song lyric dataset is *class imbalance*: there are twice as many Taylor Swift song lyrics as Beyoncé or Drake. We can mitigate this imbalance problem by weighting less frequent classes more heavily in the loss function.

Calculate the number of lyrics by artist in the training dataset. Add class weights as an argument to the loss function, **`nn.NLLLoss`**. You can do this by passing in a dictionary that maps labels to weights, as described in the loss function documentation.

HINT: a good weighting scheme is to weight each class inversely to its frequency in the training data.

5.3 Questions

1. How well does your model perform now? Report its validation accuracy for all three classes.

6 Intellectual Curiosity (10pts)

As usual, you will receive 90 points for implementing everything described above. To increase your score further, you can extend your investigation in some way. **If you choose to do this, please briefly describe what you've done in your report.**

If you would like to use data from another artist, I have provided some scripts for webscraping lyrics and post-processing them into the same format as the Swift, Beyoncé, and Drake lyrics.

6.1 Wrapping Up

When you're finished, you should submit all of your Python files and your answers to the questions (as a PDF) to Gradescope.