BME 230B HW assignment part B

In this assignment you will train a neural network to identify the tissue type that produced an RNA expression vector. The dataset is comprised of RNA-seq data obtained from tumors.

For the in-class exercise, you learned how to adjust parameters of a model and its training algorithm to get desirable results for the approximation of a function (regression). In this assignment, we are performing classification, which has different properties from regression. The main problems you will deal with are:

- How to adjust the model to account for multivariable input and output vectors
- How to assess the performance of a classifier beyond its loss
- Understanding the flow of data in a training/testing pipeline

Evaluation:

Your assignment will be evaluated primarily on the presentation of your notebook and your results (see part 5). Secondary to this, your written answers to any questions here should be included in the final cell of your notebook, but they are primarily intended to help you through the process of building a classifier.

Getting started:

Copy the "HW_primary_site_classifier" notebook from the class folder into your own directory before starting the assignment. You will be turning in a completed notebook, as well as some short response answers about the pipeline.

Part 1:

Use the code that has already been provided in this assignment to answer the following questions:

1. How many training examples are in the entire dataset?
2. What portion of the full dataset has been set aside for testing?
3. How many gene expression values will be used as input to classify each sample in the dataset?
4. How many tissue types are there in the data labels (AKA the y vector)?
5. When performing classification with a neural network, the output vector is an array with n variables, where each variable describes the probability that your input belongs to one class in your classification problem. Based on this information, what should be the size of your output layer in the neural network?
6. How will you know which index in the output corresponds to which tissue type once you feed your data into the model and get your output?

Part 2:

For this section you will write the remaining methods needed to train and test the network. You can reuse the methods from the in-class exercise, but you will need to edit them before they will work properly with the new data.

1. Start by copying train(), train_batch(), test(), test_batch(), plot_loss(), run(), and the model class ShallowLinear into your notebook. Give each method and class an independent cell in your notebook. Write a very brief explanation about the purpose they serve, and very briefly describe each of their inputs and outputs (parameters and returned objects) in your own terms. If you choose not to use PyTorch, don't worry about replicating these methods exactly, but you still need to write a description of your methods in your Notebook.
2. Draw a diagram to describe the pipeline from end to end, with your inputs as the gene set and expression data files, and your outputs as a table of predicted vs true values from the trained model. Include as much or as little detail as you feel like… don't spend too much time on this. Freehand drawing is fine.
3. Rewrite the final code block that declares the dataset and calls run(). You will have to initialize the PrimarySiteDataset class for your training and testing set like so:

```
x_train_path = "data/x_train.npz"
y_train_path = "data/y_train.npz"
x_test_path = "data/x_test.npz"
y_test_path = "data/y_test.npz"

dataset_train = PrimarySiteDataset(x_path=x_train_path,
y_path=y_train_path)
dataset_test = PrimarySiteDataset(x_path=x_test_path,
y_path=y_test_path)
```

   Since you are developing a classifier, it won't make sense to plot the predictions, so leave out any scatter plot that might have been a part of the in-class sine model exercise. Again if you're using another library, you don't need to have this exact implementation, but we recommend you use object oriented programming and the best practices for your library.

Now that you have all the methods needed to run your pipeline, try calling run(). You will get an error associated with the size of your input layer if you haven't already adapted your model to accommodate the new data. Fix this error for your input *and* your output layer as needed. Once your model can successfully train without errors, move on to part 3.

Part 3:

For this section you will be assessing the performance of a draft model.

1. The initial model will be so bad that you will be unable to use it as a starting point. Perform some changes to your model architecture and training parameters until your final loss is no longer undefined (nan). Don't edit the loss function (we will do this later in the assignment). Describe the changes you made. Once your model consistently obtains a loss below 1 you can start working on assessing its performance.
2. The first step towards assessing your model will be to compare your predictions to their expected values. Edit your test methods so they return the true y vectors and the y prediction vectors back to your final code block. Print the shapes of these arrays to ensure they are exactly the same dimensions. What are their dimensions?
3. Do these match your training set size from part 1?
4. Each training example in the output has the form:

$$[y_1, y_2, \dots, y_c], \qquad for\ c = number\ of\ classes$$

   To get a sense of what this looks like, make a table of the first training example with the following format, with 46 rows:

y_predict	y
0.324	0
0.235	0
0.879	1
0.167	0
0.011	0
...	...

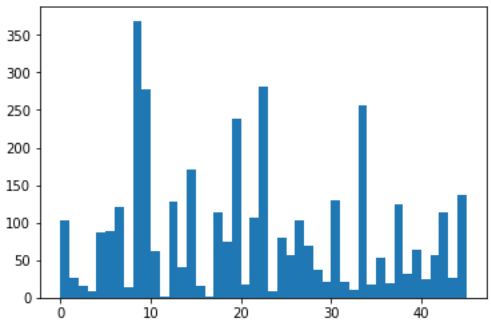
5. We only want to know which class the model predicted to be the most probable. Use numpy.argmax() to take the raw predicted and expected output vectors and find the index of their maximum value.

$$y_{predict} = \left[[y_1^*, y_2^*, \dots, y_c^*], [y_1^*, y_2^*, \dots, y_c^*], [y_1^*, y_2^*, \dots, y_c^*]\right]$$
$$y = \left[[y_1, y_2, \dots, y_c], [y_1, y_2, \dots, y_c], [y_1, y_2, \dots, y_c]\right]$$

$$y_{predict_{argmax}} = [i_1^*, i_2^*, \dots, i_n^*], \qquad for\ n = number\ of\ training\ examples$$
$$y_{argmax} = [i_1, i_2, \dots, i_n], \qquad for\ n = number\ of\ training\ examples$$

   Now you can quickly compare whether there is a match or a mismatch between your model's classificat
   true value. Print the shape of your argmax vectors to verify they have *n* entries. Write the first value of y_argmax and y_predict_argmax and compare them to the table you made in #4. Does the relationship between these values and your #4 values make sense?

6. Add to your above method to calculate the percentage of matches between your predictions and known values. What is your overall classification accuracy? It's expected that it won't be great at this stage.
7. Plot a histogram of your argmax values for y, and for predicted y. How do they compare? Your truth set for y should have a histogram that looks something like this:



with y=frequency and the x=class index.

Part 4 (no written answer for this part):

You might have noticed that our output vector doesn't have the form of a vector of random variables. So far, we have performed regression without calculating the relative probability of each class. The sum of the output variables is not 1, and is therefore not a valid probabilistic representation of a "collectively exhaustive set" of events. Also, if we minimize the loss of this output vector, we will usually find that our trained accuracy is not as great as if we had minimized the loss of an output vector that has already been transformed to reflect each class' relative probability. For these reasons, we will transform the model output with a function called *softmax*. Softmax is essentially the multivariate generalization of the sigmoid function. It has an output between 0 and 1, and it is normalized by the scores for each class in the output vector:

$$sigmoid(x) = \frac{e^x}{e^x + 1}$$

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

To do this, we just need to add a line to the end of our model's forward algorithm:

```
x = F.log_softmax(x)
```

We use the log of the softmax to prevent underflow.

You might have also noticed that our loss function is just the Mean Squared Error of the output vector, but a loss function that is better suited for a multi-class discriminator is Cross Entropy

Loss. This loss function doesn't take a one-hot vector for y… instead it takes indexes. So you will need to find the argmax of y inside of the train_batch() method, and pass that to the loss function instead. This edit is just 2 lines:

```
# convert 1-hot vectors back into indices
max_values, target_index = y.max(dim=1)
target_index = target_index.type(torch.LongTensor)

loss = loss_fn(y_predict, target_index)
```

This remaps the one-hot vector to an index and converts its data type to a long instead of a float. If you are not using PyTorch, we still recommend that you implement log softmax and cross entropy loss in your output layer, if you want to use another setup, you may, as long as you justify it in your part 5.

Once you have your model training with these new edits, go on to part 5.

Part 5:

Here you are free to make any edits to the model you think are worthwhile, and you can try your best to maximize the accuracy of its predictions by tuning each hyperparameter. If instead you are interested in generating a minimal model, or exploring which gene subset has the greatest predictive value, these are both interesting questions that you can answer by rewriting the model. As long as you explain your results using prediction accuracy and confusion, any modified pipeline is fine.

1. Once you have something you are satisfied with, report how the data was processed (if different from the provided script), architecture, and training parameters. Write briefly about what motivated you to choose these setup. Finally, make a normalized confusion matrix with your results, and plot it as a heat map. Comment briefly on your strategy for choosing parameters, and the performance of your model based on its accuracy and confusion. What tissue types were most confusing for the model?