QUEST 0 : PyTorch Warmup

Before you begin, make sure your environment is set up. Verify you have something similar to this.

```
$ python --version
Python 3.8.3

$ python -c 'import torch; print(torch.__version__)'
1.12.1

$ python -c 'import torch; print(torch.cuda.is_available())'
True
```

The following exercises assume you are familiar with PyTorch. If you want a refresher, check out the Introduction to PyTorch series from Brad Heintz.

PyTorch warmup

These exercises are meant to jog your PyTorch skills. Except for #9, you should be able to do them all in a few minutes.

1. Use `torch.randn` to create two tensors of size `(29, 30, 32)` and `(32, 100)`.
2. Use `torch.matmul` to matrix multiply the two tensors.
3. What is the difference between `torch.matmul`, `torch.mm`, `torch.bmm`, and `torch.einsum`, and the `@` operator?
4. Use `torch.sum` on the resulting tensor, passing the optional argument of `dim=1` to sum across the 1st dimension. Before you run this, can you predict the size?
5. Create a new long tensor of size `(3, 10)`.
6. Use this new long tensor to index into the tensor from step 2.
7. Use `torch.mean` to average across the last dimension in the tensor from step 6.
8. Redo step 2. on the GPU and compare results from step 2.
9. Write a pure PyTorch program to compute the value of $\sqrt{2}$ up to 4 decimal places without using the square root or other math functions from any of the libraries.

   > Hint: Notice that the answer is the (positive) root of the equation, $x^2 - 2 = 0$. To find the root, you might want to use "Newton's Method": $x_{n+1} = x_n - \frac{f(x)}{f'(x)}$.

Fail-fast prototyping

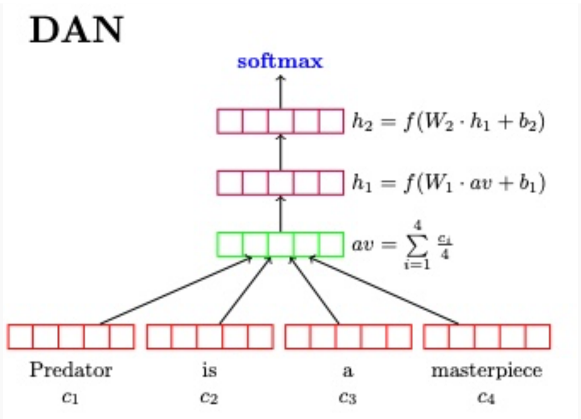
> When building neural networks, you want things to either work or fail fast. Long iteration loops are the worst enemy of a machine learning practitioner.
>
> For e.g., while writing code, you might want to incrementally test your code by doing something like this:

```
batch_size = 32
num_features = 512
embedding_size = 16

# construct a dummy input
x = torch.randn(batch_size, num_features)

# we want to project the input to embedding_size
fc = torch.nn.Linear(num_features, embedding_size)

# test if that works
print(fc(x).shape)
```

Fail-fast exercises

1. Glove has 300 dimension embeddings. Design an `nn.Module` that takes a sentence of `max_len` words, tokenizes words by spaces, represents the sentence by averaging the glove embeddings of constituent words. What is the shape of the resulting sentence embedding? When you implement this, you will need to make some assumptions. What are they?
2. How will you modify step 1. so that the sentence embeddings are in $\mathbb{R}^{50}$?
   BONUS: Can you think of more than one way to do this? What are the implications of each method?
3. Quickly test your answer in step 2. with a batch of 512 sentences on the GPU.

Congratulations! You almost implemented the model in the Deep Averaging Networks (DAN) paper!


DAN
softmax
$h_2 = f(W_2 \cdot h_1 + b_2)$
$h_1 = f(W_1 \cdot av + b_1)$
$av = \sum_{i=1}^{4} \frac{c_i}{4}$
Predator
c_1
is
c_2
a
c_3
masterpiece
c_4

4. Task: Create a `MultiEmbedding` Module that can take two sets of indices, embed them, and concat the results. You might remember it from the previous lecture where we had to produce an embedding for "green apple" from embeddings of "green" and "apple". Your `MultiEmbedding` class should work with the following test code.

```
# Test code: instantiate a MultiEmbedding with the sizes for each embedding.
# For this example, you can just randomly initialize each interior embedding.
# In a practical setting, you might support methods for initializing with
# combinations of embeddings, such as GloVe 300d vectors and word2vec 200d
# vectors, yielding 500d embeddings. Both embeddings share a vocabulary/range
# of supported indices indicated by `num_emb`

multiemb = MutliEmbedding(num_emb, size_emb1, size_emb2)
# You can then call this with a pair of indices
# where each value is in 0 <= i < num_emb
 indices1 =  ... # long tensor of shape (batch, num_length)
 indices2 =  ... # long tensor of shape (batch, num_length)
 output = multiemb(indices1, indices2)
 # print(output.shape) # should be (batch, num_length, size_emb1 + size_emb2)
```

5. **Datasets and DataLoaders.** Read this short post on PyTorch Dataset and DataLoaders. Often in prototyping we need to generate dummy datasets to test our models. Implement a PyTorch Dataset class that generates up to `num_sentences` random sentences of length up to `max_len` words. For each sentence, generate a binary label. You should be able to test your code as follows:

```
model = DeepAveragingNetwork()
dataset = DummySentenceLabelDataset(num_sentences=10, max_len=20)

# let's measure the error rate for one epoch

error = 0.0
for sentence, label in dataset:
  prediction = model(sentence)
  error += abs(prediction - label)

print(f'error rate: {error/len(dataset)}')
```

Don't worry if the answer makes little sense (it's random). Next lecture onwards, you will use real datasets via. the HuggingFace `dataset` module, but the `DummySentenceLabelDataset` class is quite useful. You plug it in any model that does supervised text classification and quickly test if your code is working. This all a part of fail-fast prototyping.