Q.1.1.1)

Looking at the graphs, clearly vocab size is more sensitive with respect to min_freq, so I chose min_freq = 2 and remove_frac = 0.4 because I wanted a vocabulary size of ~30,000. I think if min_freq were too high we'd start removing too many NE's and if remove_frac were too large we'd have too many unknown tokens to make valuable predictions.

Q1.2.1)

The distribution is centered around 24 tokens per sentence and the max sentence length in the data is around ~150. If we used static padding EVERY sentence would have to be padded to that max length while in Dynamic Padding we only need to pad to the length of the longest sentence in the batch thereby reducing the number of Pad Tokens we use in total.

Q1.3)

CE loss normalizes its inputs before computing loss which removes the necessity to add a softmax layer while NLL does not.

Q3.2)

We add @torch.no_grad() to avoid calculating unnecessary gradients. The gradients are very useful during training, as they enable backpropagation, but at evaluation time our network is already trained and we are just making predictions so calculating the gradients would be wasted calculation.

Q4.3.1)

I will evaluate how the FFNN performs relative to HMM & MEMM on the basis of its f1 score and efficiency. Also, note that my submission for HMM & MEMM was broken so I will be comparing my FFNN to the provided baselines instead.

In terms of performance (f1) my FFNN scores just slightly above both the baseline for HMM & MEMM (0.428 ish on validation data, 0.45 on the test data). I would expect better performance than the HMM as it isn't a very "complicated" model, but I think MEMM's are probably better suited to the task of NLP than an FFNN. If we were clever about feature selection (or just better at implementing in my case) I think MEMM could easily outperform even a 2-hidden layer FFNN!

In terms of efficiency, clearly, HMM is "best" as it is the simplest model with very little computation relative to an FFNN. More comparably, the MEMM took ~20 min to train (100 iterations) on a CPU while each epoch in my FFNN on CPU took ~6 min per epoch with the best value occurring around epoch 5 for a total of 30 minutes. However, in defense of FFNN's the f1 score in EARLY epochs was only slightly lower than what it ended up being (~.4) so a single layer FFNN would be much quicker than the MEMM.

Q4.3.2)

I experimented with batch size and the number of hidden dimensions.

When batch size was 64, my model still performed reasonably well (roughly the same) with equivalent runtimes. But as I used lower and lower batch sizes the runtime ballooned rapidly. At a batch size of 10 I couldn't wait long enough to allow training to finish, but it was well over 30 minutes. This is because there is a lot of computation done on/for each batch as a unit, and by decreasing the batch size we increase how many times these relatively expensive computations need to be done leading to much longer training times.

I also played with the number of dimensions in the hidden layer. Initially I scored around .43 on validation and .45 on the test set with 5 hidden dimensions. With 10 hidden dimensions I scored around .44 as well (just a smidge lower, with the best value coming in epoch 5). With 20 hidden dimensions it went back down to .42. It seems like increasing the hidden dimension is slightly positive to a point, before it overfits (but honestly the changes in f1 were underwhelming so I guess it could be random variation).

Q4.3.3)

My intuition was that as we get deeper into the network the activations for named entities would get sparser as we can conceptualize deeper "nodes" as conditioning on higher order features. The higher order the feature – in general – the fewer inputs it would activate for leading to sparser activations.

Q5.3.1)

I experimented with batch size and the number of hidden dimensions.

Just like in the FFNN I did not see meaningful changes in performance as I varied batch size, but its effect on runtime was VERY clear. Lowering the batch size even to 64 resulted in a few extra minutes of training, as I decreased batch size further it was clear that even at 50 or 40 the training time would simply outweigh any benefits this would bring.

The number of hidden dimensions was varied from 8 to 25, it performed best at 10 achieving an entity f1 .08 higher than the baseline of hidden_dim of 5. I thought this would have a larger effect given how prominent the hidden layers are in the structure of an RNN.

Q5.3.2)

If we used an RNN on an input sequence that long the whole "context" expressed by the hidden layers would get all condensed into one representation. This reduces its effectiveness as everything is all "mushed up" into one hidden state meaning early information is easily lost. In this case I would expect to see reduced effectiveness of the RNN model, but LSTMs might solve this!

Q5.3.3)

No! The context right before "Bank" in both cases is "I went to" and "Bank" appears the same in both cases as well. This means the computation would be identical leading to the same prediction in both cases.

To fix this we'd need to fix the grammar in the second sentence so it reads "I went to the Bank" which would change the previous hidden state at "Bank" in the second example, hopefully enough to classify it correctly!