AI539 Natural Language Processing with Deep Learning – Homework 2

Recurrent Neural Networks and Sentence Representations

Ameyassh Nagarajan

1.0 Written Responses

▶ TASK 1.1 [5 pt] Manually find weights and biases for the univariate LSTM defined above such that the final hidden state will be greater than or equal to 0.5 for odd parity strings and less than 0.5 for even parity. The parameters you must provide values for are w_{ix}, w_{ih}, b_i, w_{fx}, w_{fh}, b_f, w_{ox}, w_{oh}, b_o, w_{gx}, w_{gh}, b_g and are all scalars. The LSTM will take one bit of the string (0 or 1) as input x at each time step. A tester is set up in `univariate_tester.py` where you can enter your weights and check performance.

▶ TASK 1.1 Answer These are the weights I have used for this task, I have realized that the magnitude for the candidate cell mattered a lot and specifically, the w_gx and b_g mattered when I was trying to get the LSTM to predict the correct parity.

```
        # i gate
        w_ix = 200.0
        w_ih = 0.0
        b_i = -40.0

        # f gate
        w_fx = -80.0
        w_fh = 0.0
        b_f = 20.0

        # o gate
        w_ox = 0.0
        w_oh = 0.0
        b_o = 10.0

        # g
        w_gx = 0.0
        w_gh = -100.0
        b_g = 20.0

```

▶ TASK 2.1 [5 pt] Implement the `ParityLSTM` class in `driver_parity.py`. Your model's `forward` function should process the batch of binary input strings and output a $B \times 2$ tensor y where $y_{b,0}$ is the score for the b^{th} element of the batch having an even parity and $y_{b,1}$ for odd parity. You may use any PyTorch-defined LSTM functions. Larger hidden state sizes will make for easier training in my experiments but often generalize more poorly to new sequences. Running `driver_parity.py` will train your model and output per-epoch training loss and accuracy. A correctly-implemented model should approach or achieve 100% accuracy on the training set. In your write-up for this question, describe any architectural choices you made.

For this task specifically, I have defined the model as it can be seen in the code below:

```python
class ParityLSTM(torch.nn.Module) :

    # __init__ builds the internal components of the model (presumably an LSTM and linear layer for classification)
    # The LSTM should have hidden dimension equal to hidden_dim

    def __init__(self,input_dim = 1, hidden_dim = 16, output_dim = 2) :
        super(ParityLSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.lstm = nn.LSTM(input_dim, self.hidden_dim, batch_first=True)
        self.fc = nn.Linear(self.hidden_dim, output_dim)





    # forward runs the model on an B x max_length x 1 tensor and outputs a B x 2 tensor representing a score for
    # even/odd parity for each element of th ebatch
    #
    # Inputs:
    #   x -- a batch_size x max_length x 1 binary tensor. This has been padded with zeros to the max length of
    #         any sequence in the batch.
    #   s -- a batch_size x 1 list of sequence lengths. This is useful if you want to get the hidden state at
    #         the end of a sequence, not at the end of the padding (may not matter here)
    #
    # Output:
    #   out -- a batch_size x 2 tensor of scores for even/odd parity

    def forward(self, x, s):
      #TODO
      packed_input = nn.utils.rnn.pack_padded_sequence(x, s, batch_first=True, enforce_sorted=False)
      packed_output, (ht,ct) = self.lstm(packed_input)
      out, input_sizes = nn.utils.rnn.pad_packed_sequence(packed_output, batch_first=True)
      logits = self.fc(ht[-1])
      return logits


    def __str__(self):
        return "LSTM-"+str(self.hidden_dim)

```

▶ TASK 2.2 [1 pt] `driver_parity.py` also evaluates your trained model on binary sequences of length 1 to 256 (for 500 samples each) and saves a corresponding plot of accuracy vs. length. Include this plot in your write-up and describe the trend you observe. Why might the model behave this way?

Based on this architecture and the parameters, I found that as the hidden dimensions increased, the accuracy increased. At 16 hidden dimensions, this is what my plot looks like. As the hidden dimensions decreased the accuracy also decreased.

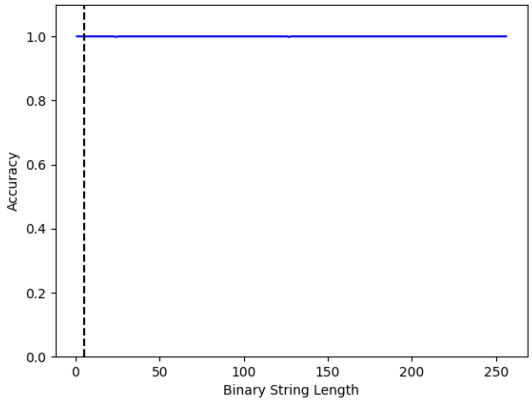

Figure 1: Parity LSTM performance

I have played around with different numbers for hidden dimensions.

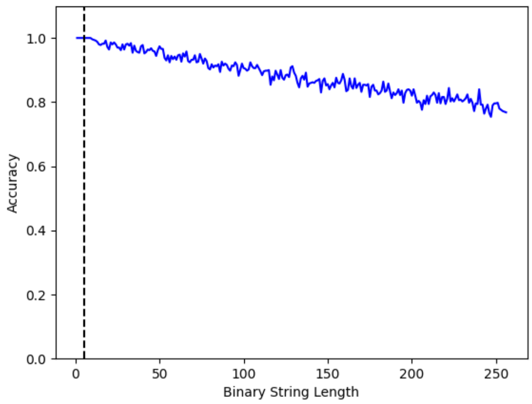

Figure 2: 10 Hidden Dimensions

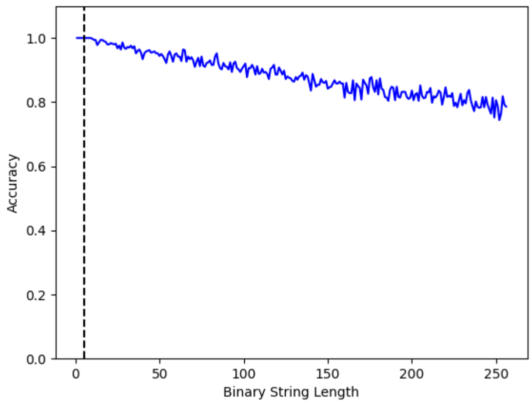

Figure 3: 4 Hidden Dimension

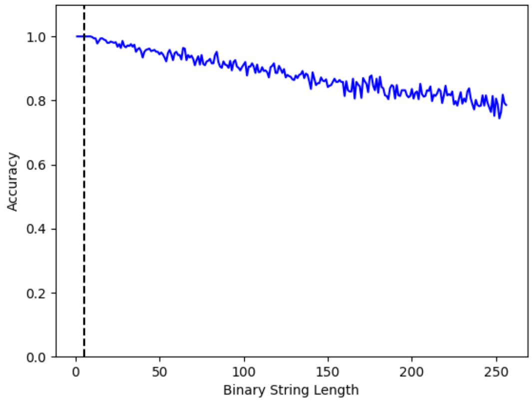

Figure 4: 2 Hidden Dimensions

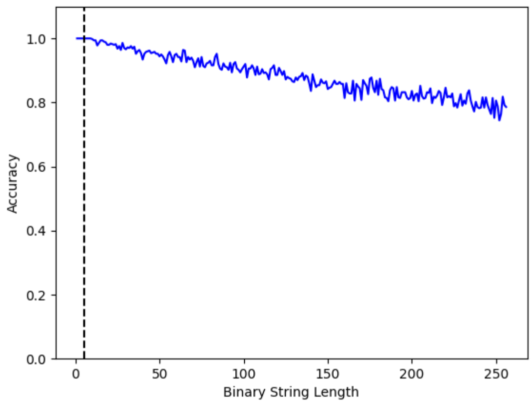

Figure 5: 1 Hidden Dimension

From these experiments, it is seen that as the hidden dimensions decrease, the accuracy over longer strings decreases.

▶ TASK 2.3 [3 pt] We know from 1.1 that even a univariate LSTM (one with a scalar hidden state) can theoretically solve this problem. Run a few (3-4) experiments with different hidden state sizes, what is the smallest size for which you can still train to fit this dataset? Feel free to adjust any of the hyper-parameters in the optimization in the `train_model` function if you want. Describe any trends you saw in training or the generalization experiment as you reduced the model capacity.

Some of the trends I saw when I was training the model were that, as the number of hidden dimensions decreased, the accuracy also decreased. This is because, as the length of the strings increases it is difficult for the model to remember long term dependency. I believe that with fewer hidden states the model would benefit if the number of epochs were increased.

▶ TASK 2.4 [1 pt] It has been demonstrated that vanilla RNNs have a hard time learning to classify whether a string was generated by an ERG or not. LSTMs on the other hand seem to work fine. Based on the structure of the problem and what you know about recurrent networks, why might this be the case?

Vanilla RNNs find it challenging to classify strings generated by Elementary Recursive Grammars (ERGs) because they cannot effectively manage long-term dependencies, a problem exacerbated by vanishing gradients. In contrast, LSTMs utilize memory cells and gates that enable them to preserve essential information across extended sequences, enhancing their capability for such tasks.

▶ TASK 3.1 [2 pt] The first step for any machine learning problem is to get familiar with the dataset. Read through random samples of the dataset and summarize what topics it seems to cover. Also look at the relationship between words and part-of-speech tags – what text preprocessing would be appropriate or inappropriate for this dataset? Produce a histogram of part-of-speech tags in the dataset – is it balanced between all tags? What word-level accuracy would a simple baseline that picked the majority label achieve?

Here is the histogram of the part of speech tags, this histogram shows which part of speech is used more in sentences.

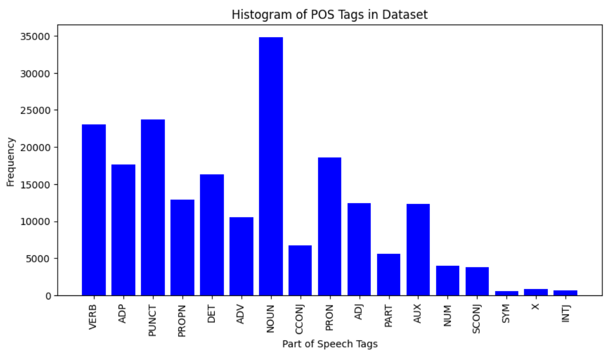

Figure 6: POS Histogram

The histogram of Part of Speech (POS) tags from the dataset is imbalanced, with NOUN, VERB, and PUNCT tags being the most common, while tags like SYM, X, and SCONJ are less frequent. A basic model predicting the most common tag, NOUN, would achieve around 30% accuracy, assuming NOUNs constitute 30,000 out of 100,000 total tags. Tokenization is essential for preprocessing as each word requires tagging, but case normalization may be unnecessary unless it significantly affects POS tagging. Punctuation must be retained due to its prevalence. To address tag imbalances, strategies might include data augmentation for rare tags or employing advanced models like LSTMs or Transformers that capture contextual subtleties more effectively. Starting with a simple baseline model and progressing to more complex ones can help manage the uneven distribution of tags.

```


        Sample 1:
 Token                                    POS Tag
--------------------------------------------------
4                                         X
Sample 2:
 Token                                    POS Tag
--------------------------------------------------
Small                                  ADJ
polygamous                        ADJ
groups                                NOUN
have                                    AUX
existed                             VERB
in                                        ADP
the                                     DET
southwestern                    ADJ
US                                        PROPN
under                                  ADP
the                                     DET
watchful                          ADJ
yet                                     CCONJ
fairly                              ADV
benign                              ADJ
eye                                     NOUN
of                                        ADP
authorities                     NOUN
ever                                    ADV
since                                  SCONJ
a                                         DET
sect                                    NOUN
known                                  VERB
as                                        ADP
the                                     DET
Fundamentalist               PROPN
Latter                              PROPN
Day                                     PROPN
Saints                              PROPN
(                                         PUNCT
FLDS                                    PROPN
)                                         PUNCT
separated                         VERB
itself                              PRON
from                                    ADP
mainstream                        ADJ
Mormonism                         PROPN
in                                        ADP
1890                                    NUM
.                                         PUNCT
Sample 3:
 Token                                    POS Tag
--------------------------------------------------
Our                                     PRON
attorneys                         NOUN
and                                     CCONJ
internal                           ADJ
audit                                  NOUN
area                                    NOUN
have                                    AUX
made                                    VERB
one                                     NUM
language                           NOUN
revision                           NOUN
concerning                       VERB
Section                              NOUN
XIII                                    NUM
Audit                                  NOUN
Rights                              NOUN
.                                         PUNCT
```

```

Sample 4:
 Token                                  POS Tag
----------------------------------------------------
the                                    DET
time                                  NOUN
:                                        PUNCT
10:00                                NUM
AM                                      NOUN
-                                        SYM
11:00                                NUM
AM                                      NOUN
CST                                    PROPN
Sample 5:
 Token                                  POS Tag
----------------------------------------------------
They                                  PRON
are                                    AUX
beautiful                        ADJ
and                                    CCONJ
will                                  AUX
add                                    VERB
a                                        DET
lot                                    NOUN
to                                      ADP
our                                    PRON
collection                      NOUN
.                                        PUNCT
```

▶ TASK 3.2 [10 pt] Create a file `driver_udpos.py` that implements and trains a bidirectional LSTM model on this dataset with cross entropy loss. The BiLSTM should predict an output distribution over the POS tags for each token in a sentence. In your written report, produce a graph of training and validation loss over the course of training. Your model should be able to achieve >70% per-word accuracy fairly easily.

To achieve stronger performance, you will likely need to tune hyper-parameters or model architecture to achieve lower validation loss. Using pretrained word vectors will likely help as well. You may also wish to employ early-stopping – regularly saving the weights of your model during training and then selecting the saved model with the lowest validation loss. In your report, describe any impactful decisions during this process. Importantly – `DO NOT EVALUATE ON TEST DURING THIS TUNING PROCESS`.

Once you are done finetuning, evaluate on the test split of the data and report the per-word accuracy.

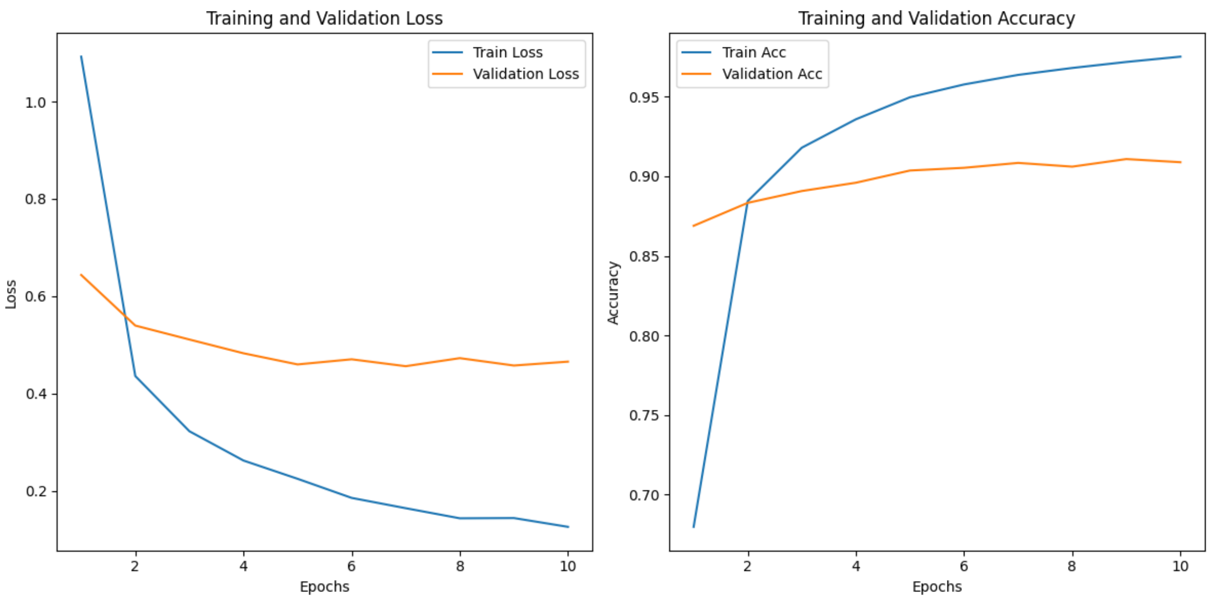

Figure 7: Graph that studies the accuracy over training and validation

Results of evaluating the model on test set.

```
        Model Evaluation on Test Data - Loss: 0.448, Accuracy: 90.66%


```

▶ TASK 3.3 [3 pt] Implement a function `tag_sentence(sentence, model)` that processes an input sentence (a string) into a sequence of POS tokens. This will require you to tokenize/numeralize the sentence, pass it through your network, and then print the result. Use this function to tag the following sentences:

The old man the boat.
The complex houses married and single soldiers and their families.
The man who hunts ducks out on weekends.

Here is the output of my function

```
The/DET old/ADJ man/NOUN the/DET boat/NOUN ./PUNCT
The/DET complex/ADJ houses/NOUN married/VERB and/CCONJ single/ADJ soldiers/NOUN and/CCONJ their/PRON families/NOUN ./PUNCT
The/DET man/NOUN who/PRON hunts/VERB ducks/VERB out/ADP on/ADP weekends/NOUN ./PUNCT
```

Figure 8: