I 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 wix, wih, bi, wfx, wfh, bf , wox, woh, bo, wgx, wgh, bg 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.

Answer:

```
# i gate
w_ix =8
w_ih = 8
b_i = -5

# f gate
w_fx = 0
w_fh = 0
b_f = -1

# o gate
w_ox = -10
w_oh = -10
b_o = 15

# g
w_gx = 0
w_gh = 0
b_g = 10
```

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 × 2 tensor y where yb,0 is the score for both elements 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. Running driver_parity.py will train your model and output per-epoch training loss and accuracy. A correctly implemented model should approach 100% accuracy on the training set. In your write-up for this question, describe any architectural choices you made.

Answer:

Architecture

LSTM

- nn.LSTM with input dimension 1
- hidden dimension =64 (low=2)
- number of layers=1

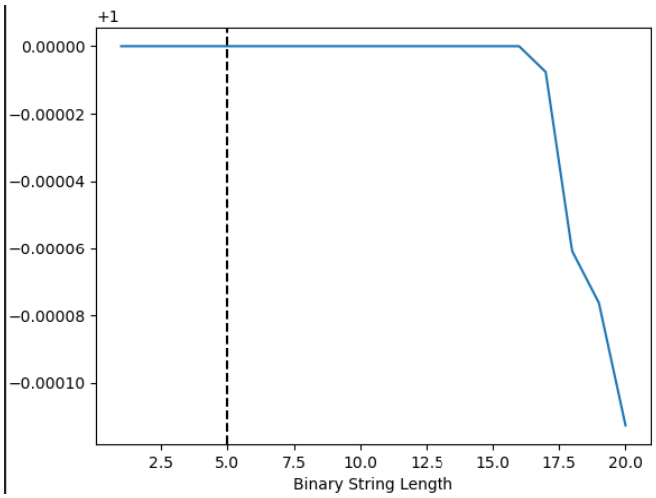
● batch_first= True.

Linear

● Input: Output of LSTM formatted “out = out[rs, si, :]
● Output of Liner is of size two for binary classification.

TASK 2.2 [1 pt] driver_parity.py also evaluates your trained model on binary sequences of length 0 to 20 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?**

Plot:



LSTM hidden_dim=64 Parity Generalization

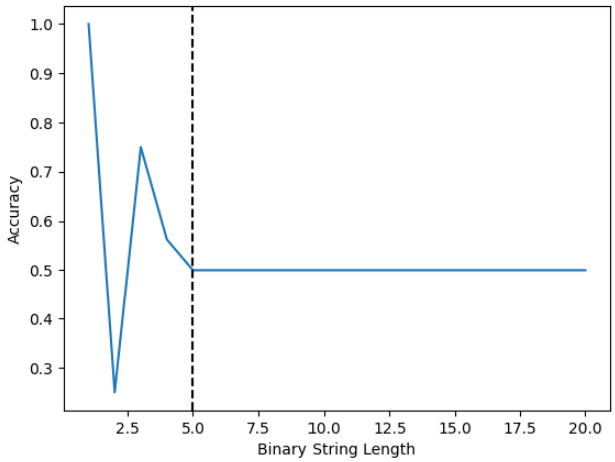
Trend and Why?:

100% accuracy for sequences of length < 16. Then the LSTM cannot **perfectly** predict the parity, and accuracy falls.LSTM has been trained to remember with a max length set to 5 *(maximum_training_sequence_length = 5)*. Therefore it doesn’t generalize well when we test with longer sequences. Generally, LSTMs have a memory life of 30-40 sequences, this LSTM seems to have a ‘perfect memory life” of 16.

.
TASK 2.3 [3 pt] **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?**

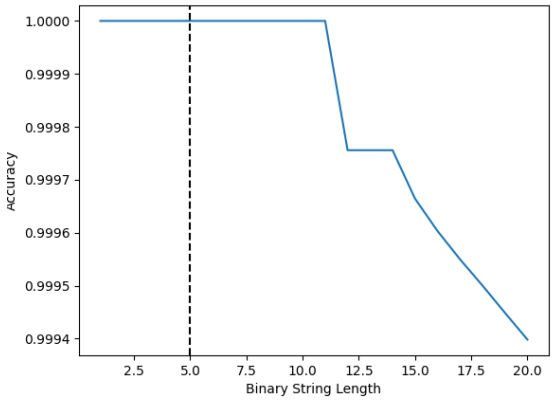
→ **Size 2**

LSTM-1 :



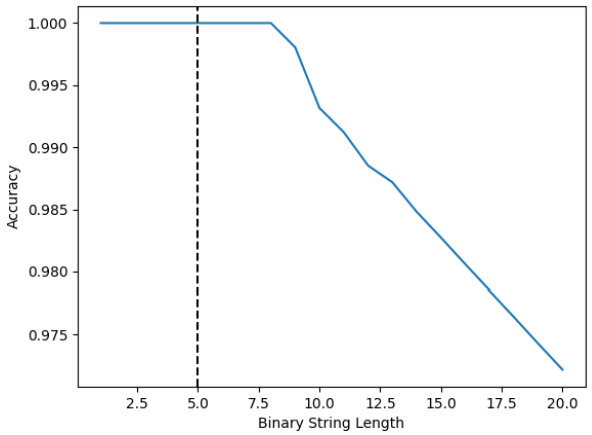
- epoch 1990 train loss 0.687, train acc 0.548. : Not fitting on hidden state size 1

LSTM-2 :



- epoch 1990 train loss 0.003, train acc 1.000
- epoch 6990 train loss 0.001, train acc 1.000 : Fitting on hidden state size 2
- length=20 val accuracy 0.999

LSTM 20: Lower learning rate, Lower Dimension: Hidden Dimension = 20, Learning Rate = 0.001, Batch Size=1000



- epoch 1990 train loss 0.003, train acc 1.000
- length=8 val accuracy 1.000, length=9 val accuracy 0.998
- length=20 val accuracy 0.972

Going lower than 20 to 6. The perfect Memory limit stays around nine but with a steeper decrease in accuracy compared to LSTM-20

TASK 2.4 [1 pt]It has been demonstrated that vanilla RNNs need help learning to classify whether a string was generated by an Embedded Reber Grammar (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?

→ Using Vanilla RNN on the problem, the computational graph is long-chain with only a single hidden state that is updated at each time step based on the previous hidden state(previous graph) and the current input(node). As weights(Wh) get multiplied during backpropagation, they can head toward zero (vanishing) or head toward infinity (exploding).

TASK [3.1] 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?

Answer:

Read through random samples of the dataset and summarize what topics it seems to cover
Some Topics Covered: News, Reviews, Blogs, Website scrapes

```
['dpa', ':', 'iraqi', 'authorities', 'announced', 'that', 'they', 'had', 'busted', 'up', '3', 'terrorist', 'cells', 'operating', 'in', 'baghdad', '.']
```

Topic : News

```
['i', 'will', 'never', 'return', 'there', 'again', '(', 'and', 'now', 'have', 'some', 'serious', 'doubts', 'about', 'the', 'quality', 'of', 'work', 'they', 'actually', 'performed', 'on', 'my', 'car', ')', '.']
```

Topic : Reviews

```
['so', 'i', 'checked', 'it', 'out', 'with', 'the', 'u.s.', 'post', 'office', '(', '1-800-238-5355', ')', 'and', 'they', 'confirmed', 'that', 'it', 'is', 'indeed', 'legal', '!!']
```

Topic: Blogs

```
['proceed', 'with', 'usual', 'login', '/', 'password']
['http://www.nea.fr/html/rp/chernobyl/c01.html']
['http://www.21stcenturysciencetech.com/articles/chernobyl.html']
```

Topic: Websites

Also look at the relationship between words and part-of-speech tags – what text preprocessing would be appropriate or inappropriate for this dataset?

→
Part of speech labels include tags for punctuations and symbols, which we generally remove during text processing.We wouldn't want to remove punctuations and symbols as these have POS labels (SYM, PUNC)

Also, Lemmatization and word stemming could change a word's part of speech.

For instance: Loving (Adjective) → Love(Verb)
Similarly, for stemming Baring(Verb) → Bar(Noun)

Produce a histogram of part-of-speech tags in the dataset – is it balanced between all tags:
→ **No, it is not balanced** between all tags. It is skewed with Nouns as the maximum tag and tags like ‘X’(others),’ INTJ(interjection)’, and’ SYM’(symbols) at the tail.

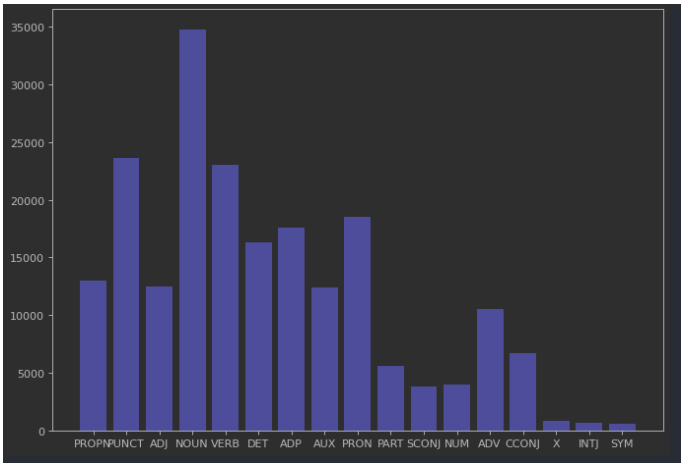


Fig: Histogram of part-of-speech tags in the dataset

What word-level accuracy would a simple baseline that picked the majority label achieve?
→ Baseline that picked the majority label (NOUN) would achieve an **accuracy of around 17%**.

Count['Noun']/ALL Tags=0.16999095818772758.

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.

→

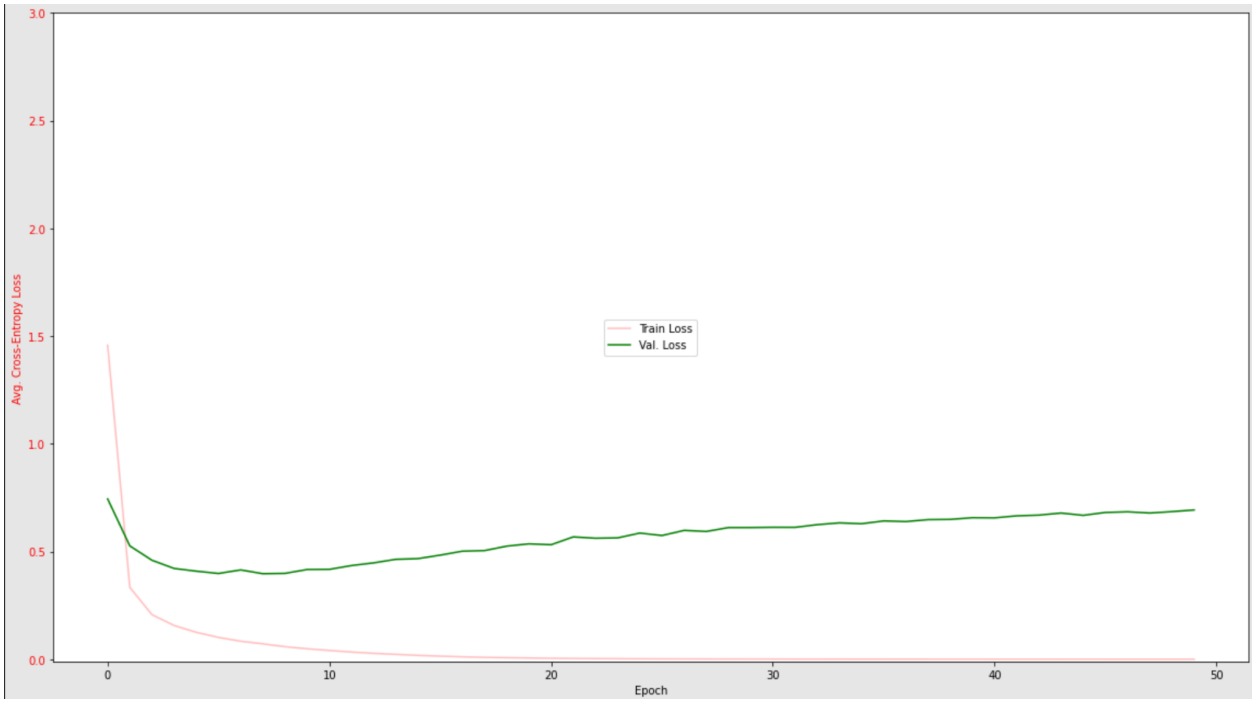


Fig: Average Training and Validation Loss over 50 Training Epochs

Impactful decisions:

- Using pre-trained Glove Embeddings:
  - `model.embedding.weight.data.copy_(pretrained_embeddings)`

        - Without using these embeddings, the first epoch training accuracy was around 60%; using these bumped it up to 71%

- Increasing the hidden state size:
    - Hidden state 18: initial training accuracy 44%

- Increasing the number of layers wasn't that useful but was really slow.
    - Number of Layers 64: initial training accuracy 18.17707071409067%
    - Number of Layers 10: initial training accuracy 18.271947372802398%

- Initially started with optimizers lr=0.003, and weight_decay=0.00001, but using the default learning rate was fine.
- Saving the best model periodically and early stopping was helpful because the training was overfitting.

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/neutralize the sentence, pass it through your network, and print the result. Use this function to tag the following sentences:

The old man the boat.

['DET', 'ADJ', 'NOUN', 'DET', 'NOUN', 'PUNCT']

The complex houses married and single soldiers and their families.

['DET', 'ADJ', 'NOUN', 'VERB', 'CCONJ', 'ADJ', 'NOUN', 'CCONJ', 'PRON', 'NOUN', 'PUNCT']

The man who hunts ducks out on weekends.

['DET', 'NOUN', 'PRON', 'VERB', 'VERB', 'ADV', 'ADP', 'NOUN', 'PUNCT']

References:

- https://torchtext.readthedocs.io/en/latest/datasets.html
- https://pytorch.org/text/0.8.1/datasets.html
- https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
- https://medium.com/@pankajchandravanshi/nlp-unlocked-pos-tagging-004-447884b6030a
- https://suneelpatel18.medium.com/nlp-pos-part-of-speech-tagging-chunking-f72178cc7385
- https://github.com/delip/PyTorchNLPBook/issues/14
- https://universaldependencies.org/u/pos/