Part 2 Report: Challenging RNN Acceptor Cases

Submitters:

Ori Braverman 318917010

Elie nedjar 336140116

Model Hyperparameters (for all cases)

- Dev ratio = 0.1
- Batch size = 16
- Epochs = 25
- Embedding dim = 30
- Lstm hidden dim = 32
- Mlp hidden dim = 16
- Learning rate = 0.003
- Early stopping = 10 epochs with no change in loss

Case 1: Palindromes with Optional Middle Character

Description of the Language (L1)

L1 = {wtw^R | w ∈ Σ1*, |w| ≤ Max length, t ∈ Σ1 or t = ε}

- Σ1 = {a, ..., z}

- Max length = 50

This language consists of palindromes with an optional middle character. The strings are formed by a word w, followed by an optional single character t, and then the reverse of w (w^R).

Why This Language is Challenging

1. **Non-Regular Language**: Palindromes are non-regular language. This means they cannot be recognized by a Deterministic Finite Automaton (DFA), which can only handle regular languages. Instead, palindromes are a context-free language that can be recognized by a Pushdown Automaton (PDA) because they require a stack to match characters from the beginning and end of the string.
2. **One Directional LSTM Limitation**: Our LSTM is one directional, so it processes the sequence from start to end without considering future tokens. For palindromes, the model needs to compare the start of the sequence with the end, but one directional LSTM **don't have information of the end in the start position**. A bidirectional LSTM would have been more effective in recognizing palindromic patterns.

3. **Long-Range Dependencies**: The language requires the model to maintain and compare information across potentially long sequences. The LSTM must remember the first half of the string while processing the second half, which can be challenging for longer sequences or not possible because the **memory unit of the LSTM is limited**.
4. **Optional Middle Character**: The presence of an optional middle character adds complexity, as the model must handle **both even-length and odd-length** palindromes. This challenges the LSTM to generalize across different string lengths.

Results

- Train set size: 1,000 examples (500 positive, 500 negative)

- Test set size: 500 examples (250 positive, 2500 negative)

- Training iterations: 25 epochs with early stopping

The LSTM acceptor struggled with this language:

- Training accuracy: 91.8%

- Test accuracy: 50.8%

The LSTM struggled to learn this language. Because there are half positive and half negative examples, the model that say POS/NEG to all examples gets the same accuracy as the accuracy learned.

Case 2: String Repetition

Description of the Language (L2)

L2 = {ww | w ∈ Σ1*, |w| ≤ Max length}

- Σ1 = {a, ..., z}

- Max length = 50

This language consists of strings that are exact repetitions of a substring w.

Why This Language is Challenging

1. **Exact repetition detection**: The RNN needs to recognize that the second half of the string is an exact copy of the first half. This require the model to remember w without any changes, which could be difficult to impossible for long sequences.
2. **Variable-length substrings**: The length of w can be from 1 to max length, so the model needs to know when w ends and when the second w begins. This requires the model to learn how to compare substrings with changing sizes.

Results

- Train set size: 1,000 examples (500 positive, 500 negative)

- Test set size: 500 examples (250 positive, 2500 negative)

- Training iterations: 25 epochs with early stopping

The LSTM acceptor performed better on this language compared to L1, but still showed limitations:

- Training accuracy: 91.4%

-Dev accuracies: ranged from 49% to 56%

- Test accuracy: 53.2%

The LSTM struggled to learn this language. Because there are half positive and half negative examples, the model that say POS/NEG to all examples gets the same accuracy as the accuracy learned.

Case 3: Balanced Parentheses

Description of the Language (L3)

L3 = {w | w ∈ Σ2*, |w| ≤ 2*Max length, w is a correct parenthesis string}

- Σ2 = {'(', ')', '{', '}', '[', ']'}

- Max length = 50

This language consists of correctly balanced parentheses, brackets, and braces.

Why This Language is Challenging

1. **Non-Regular Language**: like case 1, to check correct parenthesis, there is a need to save the open parentheses using a stack.
2. **One Directional LSTM Limitation** like case 1, when the model encounters a open parentheses it needs to validate it based on information yet to be known by an one directional LSTM.
3. **Nested structure**: The RNN needs to keep track of multiple levels of nesting.
4. **Long-range dependencies**: Matching opening and closing symbols can span the entire sequence. So, the model needs to remember a lot of information for many time steps.
5. **Multiple symbol types**: The network must learn to match three different types of brackets correctly.

Results

- Train set size: 1,000 examples (500 positive, 500 negative)

- Test set size: 500 examples (250 positive, 2500 negative)

- Training iterations: 25 epochs with early stopping

The LSTM acceptor had significant difficulties with this language:

- Training accuracy: 91.8%

-Dev accuracies: ranged from 49% to 56%

- Test accuracy: 51.8%

The LSTM struggled to learn this language. Because there are half positive and half negative examples, the model that say POS/NEG to all examples gets the same accuracy as the accuracy learned.