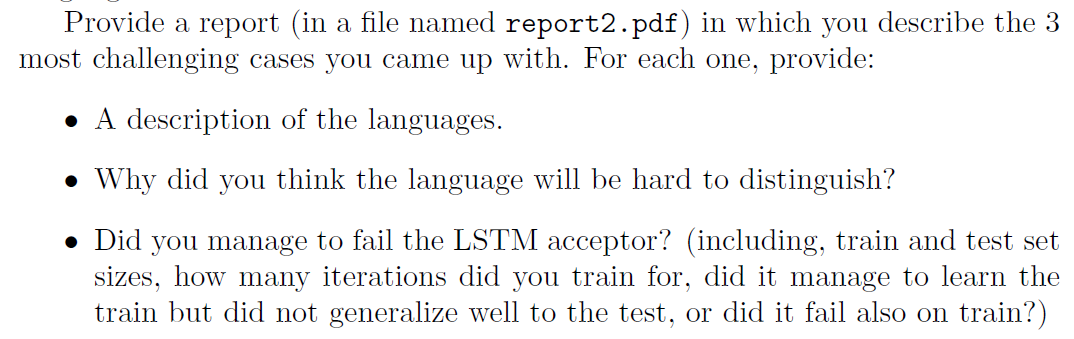
**ID 011862141**



After trying a few sequences that the LSTM was able to “crack”, I came up with the following:

General form (both positive and negative: ([0-9]+a){5} (a sequence of numbers followed by an ‘a’, repeat 5 times).

* Positive: Each of the 5 number segments is constructed so that the sum of the numbers is even.
* Negative: same structure but number segments sum to odd numbers.

At first I believed this was a good challenge, but after allowing the LSTM to run for 40 epochs (40,000 samples) it was able to learn this sequence.

For all the following, if not stated otherwise, embedding size is 10, LSTM memory vector size is 21, and the MLP’s hidden dimension is 15.

# Structures the LSTM did not pick up

Note: I gave a short name to each syntax. The name is for readability purposes only. The syntax definitions are detailed.

## Modulu

General form (both positives and negatives): ([1-9]+[a|b])+

Number sequences terminated by either a single ‘a’ or a single ‘b’, depending on the sequence’s modulus 7 (when interpreted as a decimal number):

* for the positive case, number sequences with values of 0, 1, or 2 modulus 7 were followed by an ‘a’, otherwise by a ‘b’
* for the negative case, modulus of 0, 1 or 2 are terminated by a ‘b’, otherwise by an ‘a’

Even after 200 epochs, the accuracy of this LSTM fluctuated between 60% and 70%. I increased the train set to 5000 cases and let it run for 50 epochs but it still swung between 55% and 75%. Even when the test set was a subset of the train set, results were the same. Increasing the LSTM state vector from 10 to 20 did not improve either.

While it’s possible to construct a streaming algorithm that keeps track of the number’s value modulus 7, getting there from a random initialization without “cues” is very hard. The algorithm was able however to “guess” some answers right, probably based on the last two digit in short sequences.

# Mind the first 4 digits

Both positive and negative have the following form: [1-9]{4,}a+b+c+d+

Positive:

* The length of the ‘a’ segment corresponds to the value of the first digit in the preceding number sequence.
* The length of the ‘b’ segment corresponds to the 2nd digit
* Similarly for c and d

Negative:

* The number of ‘a’-s corresponds to the value of the 4th digit in the number sequence.
* The number of ‘b’-s corresponds to the 3rd
* similarly for c and d

Care was taken that the first 4 digits are not symmetric which would mean an example would agree both with the positive and negative definitions.

This run for several dozen iterations without any improvement in accuracy (training set of 500 positive and 500 negative, accuracy remained flat at 50%). This syntax represents a combination of long-term relations and non-trivial rules (without “knowing” the value of each digit there are many “guesses” to “make right” for this to work).

# Letter Counter

General form: ([abcd]{1,10}[1-9])+

Positive:

* A random sequence of letters followed by a single digit. The digit is equal the number of letters appearing before it.
* Repeat this pattern a few times in each sequence

Negative:

* A random sequence of letters followed by a single digit. The number of letters plus the digit sum to 11.
* The letter sequence length is at least 2.

This ran for hundreds of iterations without any improvement in accuracy. After 150 epochs, it scored 51% on the training set itself. I actually half-expected the LSTM to crack this one. It is possible that if I limited the lengths of the letter sequences and used only a single letter, this would be solved.