Text Normalization Challenge - English Language

Presented by
Dahou Abdelhalim Hafedh
Pavithra POORNACHANDRAN
Tahani FENNIR

December 18th, 2020

Table of Content

Introduction

- Text normalization is the process of transforming a text into a standard form.

Examples:

- Transform word numerals into numbers (eg.: ‘twenty three’→‘23’)
- Acronym normalization (e.g.: ‘US’→‘United States’/‘U.S.A’)
- Removal or substitution of special characters(e.g.: remove hashtags).
- Removal of duplicate whitespaces and punctuation.

Sequence to Sequence Network

- Two recurrent neural networks work together to transform one sequence to another.

Encoder

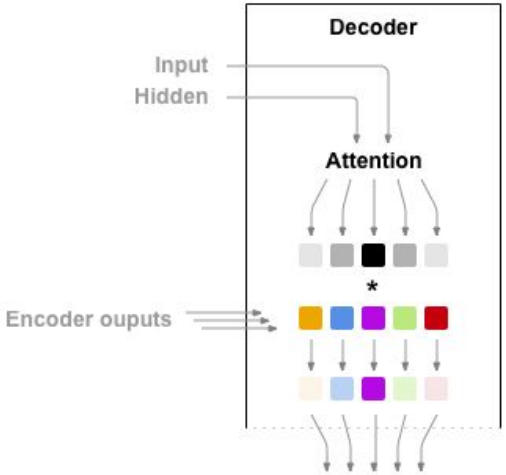
- The encoder reads an input sequence and outputs a single vector

Decoder

- Decoder network unfolds that vector into a new sequence
- **Simple Decoder** use only last output of the encoder. This last output is sometimes called the *context vector* as it encodes context from the entire sequence.

Attention Mechanism

- The decoder learn to focus over a specific range of the input sequence.
- **Attention Decoder** focus on a different part of the encoder outputs for every step of the decoder's own outputs.



Loading Data

- Representing each character in the data as a one-hot vector, or giant vector of zeros except for a single one (at the index of the characters).
- Set a unique index per character to use as the inputs and targets of the networks.
- To read the data file we split the file into pairs and filter it by length and content.

```
Read 258348 sentence pairs
Counting chars...
Counted chars:
after 62
before 26
['2002', 'two thousand two']
```

Training the Model

We run the input data through the encoder, and keep track of every output and the latest hidden state. Then the decoder is given the <SOS> token as its first input, and the last hidden state of the encoder as its first hidden state.

"**Teacher forcing**" is the concept of using the real target outputs as each next input, instead of using the decoder's guess as the next input.

Training process looks like :

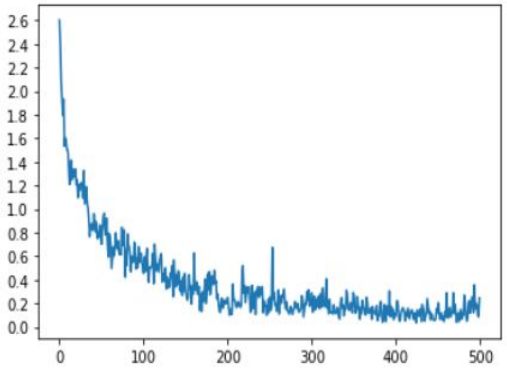
- Start a timer
- Initialize optimizers and criterion
- Create set of training pairs
- Start empty losses array for plotting

Evaluation

- We can evaluate random data and print out the input, target, and output to make some subjective quality judgements.
- Every time it predicts a character we add it to the output string, and if it predicts the EOS token we stop there. We also store the decoder’s attention outputs.

Training & Evaluation

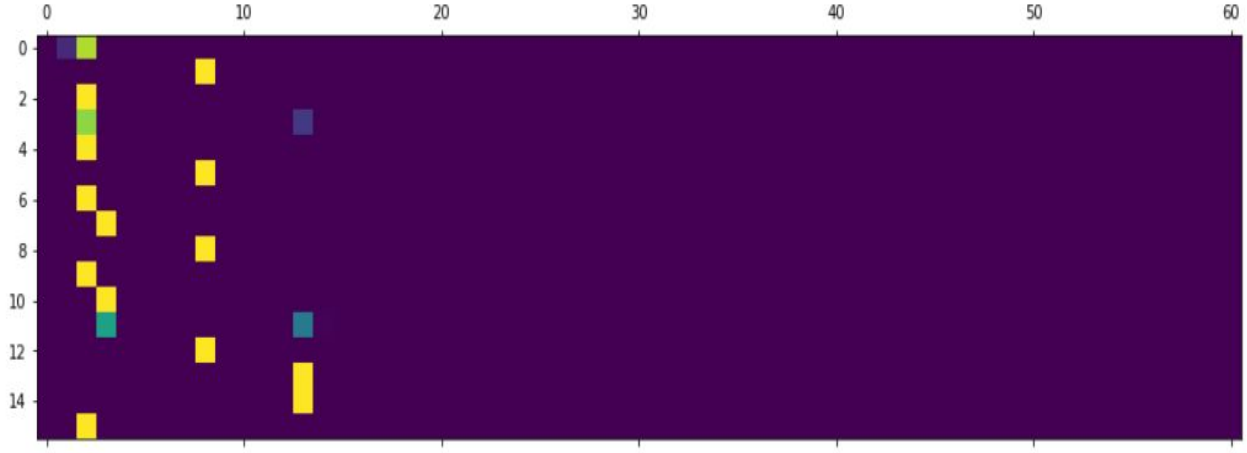
```
12m 12s (- 109m 54s) (5000 10%) 1.2427
24m 1s (- 96m 5s) (10000 20%) 0.6667
36m 6s (- 84m 14s) (15000 30%) 0.4568
47m 50s (- 71m 45s) (20000 40%) 0.2922
57m 23s (- 57m 23s) (25000 50%) 0.2346
78m 2s (- 52m 1s) (30000 60%) 0.2024
89m 7s (- 38m 11s) (35000 70%) 0.1828
100m 6s (- 25m 1s) (40000 80%) 0.1392
111m 8s (- 12m 20s) (45000 90%) 0.1143
122m 15s (- 0m 0s) (50000 100%) 0.1401

<Figure size 432x288 with 0 Axes>
```



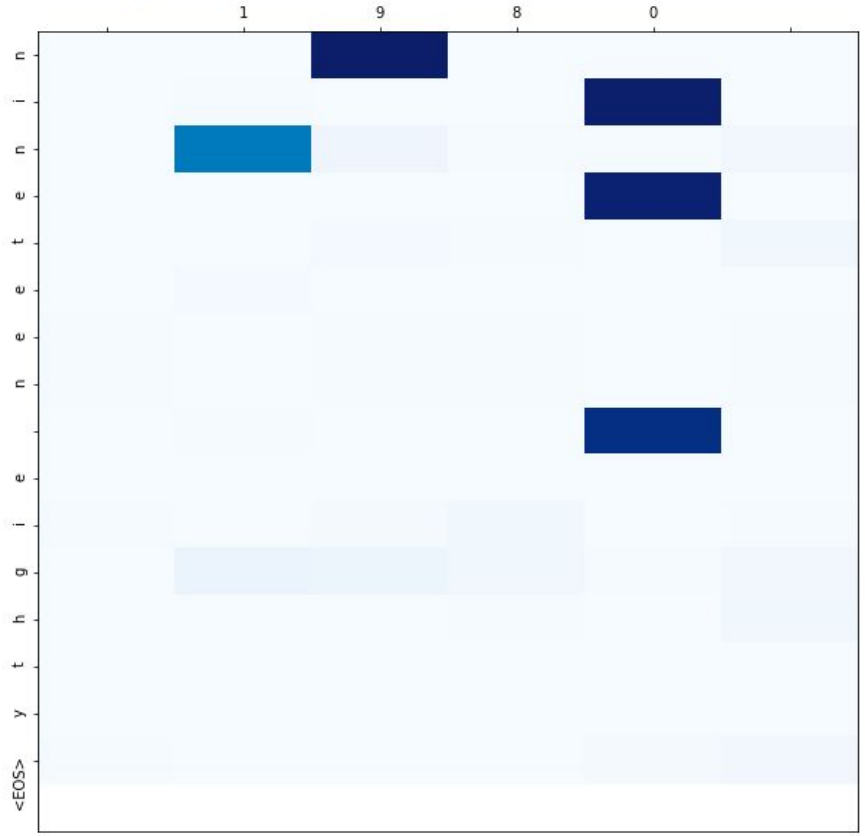

```
> 17 April 2011
= the seventeenth of april twenty eleven
< the seventeenth of april twenty eleven<EOS>

> 1999
= nineteen ninety nine
< nineteen ninety nine<EOS>

> 2010
= twenty ten
< twenty ten<EOS>

> 1952
= nineteen fifty two
< nineteen fifty two<EOS>

> 7 September 2015
= the seventh of september twenty fifteen
< the seventh of september twenty fifteen<EOS>

> 1995
= nineteen ninety five
< nineteen ninety five<EOS>

> November 29
= november twenty ninth
< november twenty ninth<EOS>
```

Visualizing Attention

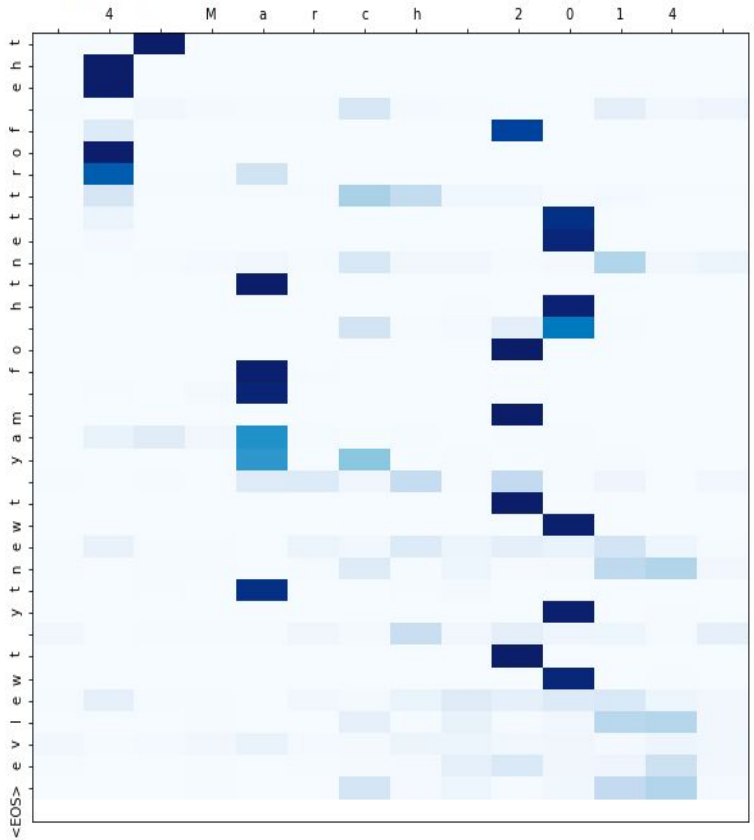
Out[24]: <matplotlib.image.AxesImage at 0x23db0420708>



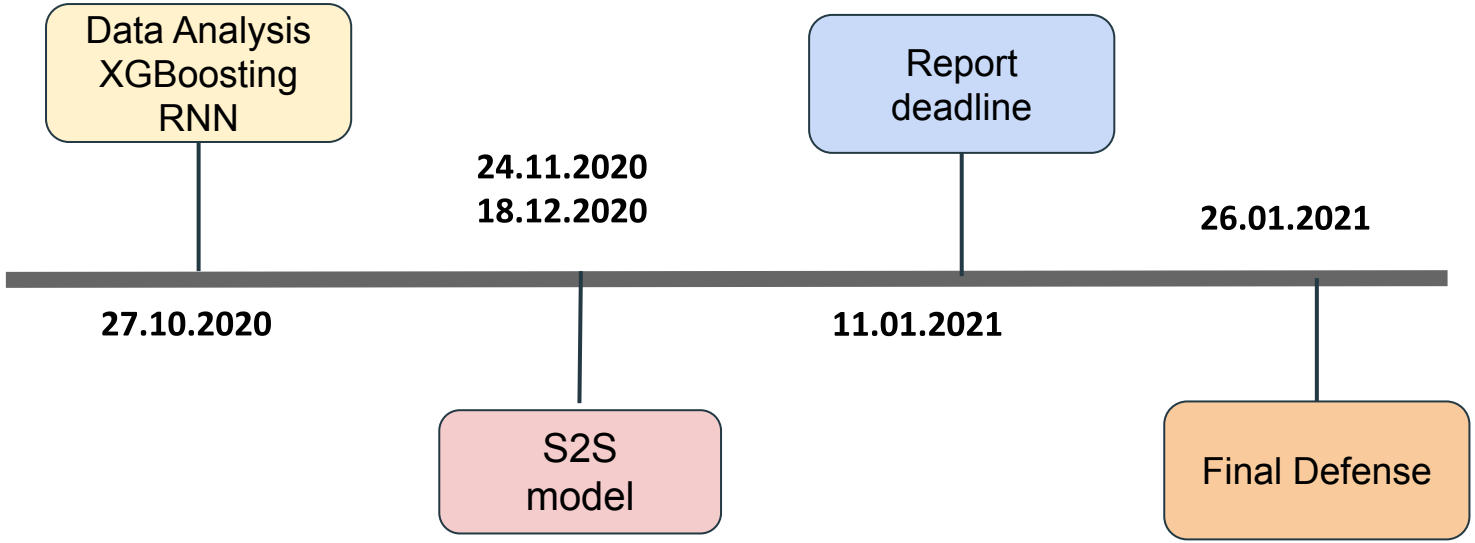
input = 1980
output = nineteen eighty<EOS>
torch.Size([16, 61])



input = 4 March 2014
output = the forttenth of may twenty twelve<EOS>
torch.Size([35, 61])



Previous work:



Thank you!