Assignment 2 - Implementation of Recurrent Perceptron

Aziz Shameem, 20d070020
Rohan Rajesh Kalbag, 20d170033
Amruta Mahendra Parulekar, 20d070009
Keshav Singhal, 20d070047

30-03-2024

Problem Statement

- **Input**: POS-tagged input tokens
- **Output**: Noun chunk labels on tokens .The beginning of the chunk will be labeled 1 and the rest of the words in the chunk will be labeled 0. All other words are labeled 1.

Implementation Details

Model architecture:

- A single perceptron
- Sigmoid activation
- 9 weighted inputs
- One weighted input flow coming from the output.

Model weights:

- [w^, w_nn_prev, w_dt_prev, w_jj_prev, w_ot_prev, w_nn, w_dt, w_jj, w_ot] = [6.6279, -1.8724, -3.7012, -2.8595, 1.8169, 0.4002, 4.3571, 0.7107, 5.5893])
- V(previous output)= -0.9459

Overall performance

	Train	Test	Val
Size	2400	3453	600

5-Fold CV	Accuracy	Precision	Recall	F1 Score
Fold 1	0.8886	0.8633	0.9947	0.9244
Fold 2	0.8911	0.8685	0.9937	0.9269
Fold 3	0.8979	0.8774	0.9927	0.9315
Fold 4	0.8965	0.8753	0.9931	0.9305
Fold 5	0.8968	0.8750	0.9924	0.9300
Overall 5f	0.8942	0.8942	0.9933	0.9286
Train on Full Dataset	0.8434	0.8100	0.9925	0.8920

Language constraint table

Thresh=RHS= 0.5 (fixed).

w^ + w_dt>= thresh,	6.6279 +4.3571>= 0.5,
w^ + w_jj >= thresh,	6.6279 + 0.7107 >= 0.5,
w^ + w_nn >= thresh,	6.6279 +0.4002 >= 0.5,
w^ + w_ot>= thresh,	6.6279 + 5.5893>= 0.5,
v +w_dt_prev + w_jj <= thresh,	-0.9459 +-3.7012 + 0.7107 <= 0.5,
v + w_dt_prev + w_nn <= thresh,	-0.9459 + -3.7012 + 0.4002 <= 0.5,
w_jj_prev + w_jj <= thresh,	-2.8595 + 0.7107 <= 0.5,
w_jj_prev + w_nn <= thresh,	-2.8595 +0.4002 <= 0.5,
v + w_jj_prev+ w_jj <= thresh,	-0.9459 +-2.8595+ 0.7107<= 0.5,
v + w_jj_prev + w_nn <= thresh,	-0.9459 + -2.8595+ 0.4002 <=0.5,
w_nn_prev + w_ot>= thresh,	-1.8724 +5.5893>=0.5,
v + w_nn_prev + w_ot>= thresh,	-0.9459 + -1.8724 + 5.5893>=0.5,
v + w_ot_prev+ w_dt >= thresh,	-0.9459 + 1.8169+ 4.3571>= 0.5,
v + w_ot_prev+ w_jj >= thresh,	-0.9459 + 1.8169+ 0.7107 >= 0.5,
v + w_ot_prev+ w_nn >= thresh,	-0.9459 + 1.8169+ 0.4002 >= 0.5,
v + w_ot_prev +w_ot>= thresh	-0.9459 + 1.8169 +5.5893>= 0.5

Some Error Cases in our Model

- The model does not take the first pronoun in the noun chunk :
  for eg : instead of "their asian cup title", the model considers "asian cup title" as the noun chunk.
  This makes sense, since majority of the nouns in the dataset are of this form (infact, in many places the labels do not take the first pronoun in the noun chunk)
- In some places, the context is ambiguous
  eg : "United Arab Emirates, 1996-12-06" is a noun chunk, but the model takes "United Arab Emirates". Similarly : "Rome 1996-12-06"
- Some common noun chunks are not labelled :
  eg : "the win"
  Our model identifies these
- In several places, the model does not identify the complete noun chunk If numbers are present.
  eg : "the last six minutes" is not identified properly, it only identifies upto the numeric word. Similarly : "The 1995 World Cup" : taken as two separate chunks

Some Error Cases in our Model

- In some places, combination of pronoun and preposition are taken incorrectly as a noun chunk.
  eg : "each from", "which made"
- All other mistakes can be attributed to mistakes in the original Dataset
  eg : "about a comeback" is labelled a noun chunk, while our model correctly classifies "a comeback" as a noun chunk

In almost all cases of incorrect classification, the mistake is of a singular label per sentence, where the model either takes an extra word in a noun chunk, or misses a word from the noun chunk.
This means the model never glosses over a noun chunk entirely, or label any group of three or more words as a noun chunk incorrectly.

Some examples of these mistakes are shown in "Analysis.ipynb"

Learnings

- Backpropagation through time (BPTT) extends backpropagation to recurrent neural networks (RNNs), allowing them to learn from sequential data by propagating errors over multiple time steps during training. It involves a forward pass to compute predictions, a backward pass through time to calculate gradients, and parameter updates to optimize the network's performance.
  Sometimes, the BPTT is truncated to manage Vanishing/Exploding gradients (particularly if the time frames are very long).
  We implemented (truncated) BPTT from scratch in this assignment (although the truncation was not used in the final model, since results were better without it).
- Controlling the grads : We had to add Gradient Clipping for the model to be able to train.
- We used sigmoid Activation with BCE loss, since that performed best compared to other combinations (tanh, MSE etc)

Evaluation Scheme

- Correct implementation of BPTT from scratch: 10 marks (show the code parts that implement weight change rules)
- Overall Performance: accuracy >=90: 10 marks; 80-89: 9; 70-79: 8; 60-69: 7; 50-59: 40-49: 5; 30-39: 4. And so on
- Error Analysis: 10
- Inequality Table: 10
- Demo: 10