CS224n: NLP with Deep Learning **Winter 2019**

Lecture 6

Course Coordinator: Prof. Chris Manning *Scribes: Akash Gupta*

Language Modeling - Task of predicting what word comes next. Formally, given a sequence $x^{(1)}, ..., x^{(t)}$ compute probability distribution of the next word $x^{(t+1)}$:

$$P(x^{(t+1)}|x^{(1)}, ..., x^{(t)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, ..., w_V\}$

$->$ LM can also be thought of as a system that assigns probability to a piece of text. For eg - for some text $x^{(1)}, ..., x^{(t)}$ the probability of this text is (according to LM):

$$P(x^{(1)}, ..., x^{(T)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times ...... \times P(x^{(T)}|P(x^{T-1}), ..., x^{(1)}) \tag{6.1}$$

$$= \prod_{t=1}^{T} P(x^{(t+1)}|x^{(1)}, ..., x^{(t)}) \tag{6.2}$$

(Pre deep learning) **n-gram Language Model** -

1. n-gram is a chunk of n consecutive words.
2. Simplifying Assumption - $x^{(t+1)}$ depends only on the preceding n-1 words.

$$P(x^{(t+1)}|x^{(t)}......, x^{(1)}) = \frac{P(x^{(t+1)}, x^{(t)}......, x^{(t-n+2)})}{P(x^{(t)}, ..., x^{(t-n+2)})}$$

3. How to get prob? - By counting the no. of n-gram and n-1 gram.

$$P(x^{(t+1)}|x^{(t)}......, x^{(1)}) = \frac{count(x^{(t+1)}, x^{(t)}......, x^{(t-n+2)})}{count(x^{(t)}, ..., x^{(t-n+2)})}$$

4. Sparsity problem -
   (a) What if numerator is 0? Prob assigned 0 (BAD!), Solution - Smoothing i.e. adding small delta.
   (b) What if denominator is 0? Cannot even calculate prob (BAD!), Solution - Back off i.e. n = n-1, So 4-gram LM to tri-gram LM.
5. Increasing n makes sparsity problems worse. Can't have > 5
6. Storage problem - As n increases, model size increases since we have to store counts of n-grams.
7. Generate text using n-gram by sampling and concatenating.

Neural LM -

1. Fixed-window Neural model - From NER in Lecture 3 - Using embedding matrix
   - Improvements over n-gram LM - No sparsity problem, No need to store n-grams

- Drawbacks - Fixed window is too small, Enlarging window increases W, $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how inputs are processed $->$ **Main idea behind introducing using RNNs for LM**

2. Recurrent Neural Networks (RNNs) -
   - input sequences, hidden states, outputs.
   - Same weight matrix applied all the words (at every time step).
   - Number of hidden states = no. of inputs.
   - Embeddings can be downloaded (e.g. GloVe), Can be fine tuned, or learned from scratch.
   - Advantages - Can process any length, Computation for may steps back, Model size doesn't increase for longer input, Same weights applied.
   - Disadvantages - Recurrent computation is slow, Not many steps back in practice.
   - Idea behind using same weights is that we are trying to learn a general function and not just one word and since each words are vectors they have a lot of positions(or numbers) to store information for different context.
   - Train using SGD and loss fn - cross entropy.
   - Backpropagation through time -

   $$\frac{\partial J^{(t)}}{\partial W} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W}|_i$$

   - Character level RNN-LM

3. Evaluating LM - Std eval metric - perplexity

   $$perplexity \ = \prod_{t=1}^{T} \left(\frac{1}{P_{LM}(x^{(t+1)}|x(t), ..., x(1))}\right)^{1/T}$$

   which is equal to exponential of cross entropy loss. **So low perplexity is better**

$->$ Language modeling is a benchmark task for measuring progress on Language Understanding. Other tasks - Predictive typing, Speech recognition, Authorship Identification, NMT, etc.

$->$ RNN $\neq$ LM

$->$ Sentiment Classification - use final hidden state as sentence encoding
QA - Question encoding + context

$->$ RNN = Vanilla RNN