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

Lecture 1

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

What does the course offer?

- Understanding of the modern methods for DL - from basics to methods for NLP
- Understanding of human languages
- Understanding and ability to build systems (in PyTorch)

$->$ **A view on Languages**: Language is this uncertain evolved system of communication but somehow we have enough agreed meaning that allows us to communicate. On the same time, we are also doing some probabilistic inference of guessing what people mean.

$->$ Languages ($\sim$ 100,000 years old) are recent than vision.

$->$ Human networks are very slow at communication (Avg: $\sim$ 15 words a second). So humans have come up with this adaptive compression mechanism assuming the listener/reader has a lot of knowledge and can use the magical hammer of probabilistic guessing to unzip the heard message and interpret it as a large-sized visual image!! 😄

How to represent meaning of a word? - Commonest linguistic way of thinking of meaning is by using denotational semantics (also used in programming languages),

$$signifier(symbol) \iff signified(idea\ or\ thing)$$

So, how do we have a usable meaning in a computer?
Common Solution: Use e.g. WordNet, a thesaurus containing lists of synonym sets and hypernyms ("is-a" relationships).

But there are many problems with it -

- Great as a resource but missing nuance. E.g. "proficient" is listed as synonym for "good". This is only correct in some contexts.
- Missing new meaning of words. For e.g. - wicked, badass, nifty ...Impossible to keep up-to-date.
- Subjective
- Requires human labor to create and adapt
- Can't compute accurate word similarity

Representing words as discrete symbols
In traditional NLP ($\sim$ till 2012), words were regarded as discrete symbols: e.g. - hotel, motel, conference,

etc. - a localist representation (as referred to in Neural Networks where each word has one particular place). Words can be represented as one-hot vectors:

$$motel = [00000000100000] \tag{1.1}$$
$$hotel = [00000100000000] \tag{1.2}$$

Problems in above representation -

- The number of words in English language is infinite (by using the concept of derivational morphology where we can create more words by adding endings to existing words. For e.g. ideally, maternally, etc..)
- The above two vectors are orthogonal and there is no natural notion of similarity for one-hot vectors! For e.g. On doing a web search "Seattle motel" I want the results for "Seattle hotel" as well.

Solution -

- Could make a similarity table and work with that by using WordNet's synonyms list - Too many pairs ($\sim$ 2.5 Trillion) 😑
- Instead use such a representation that encodes the similarity in the vectors themselves.

Representing words by their context: Based on the idea by J.R. Firth 1957 - *"You shall know a word by the company it keeps"*

- Distributional Semantics: A word's meaning is given by the words that frequently appear close-by. Ideology - *"If you understand in what context it would be correct to use a certain word vs. in what context it would be wrong to use it, this implies that you understand the meaning of the word"*
- What is context? - the set of words that appear nearby (within a fixed size window) a word w in a sentence.

Word Vectors: (Also called as word embeddings/word representation) Use the many context of the word w for representation.

- Build a dense vector for each word, chosen so that it is similar to the vector of words that appear in it's contexts.

$$banking = \begin{bmatrix} 0.132 \\ -0.513 \\ 0.21 \\ \vdots \\ 0.076 \end{bmatrix}$$

- It is a type of **distributed** representation.

Q) *Are the above decimal values constant for every NLP task in which the word appears?*
A) These values are not chosen but are actually learned by a learning algorithm on subjecting it to lots of text.

Word2Vec:(Mikolov et al. 2013) Biggest impact in NLP - a framework for learning word vectors.

Basic Algorithm:

1. Take a large corpus of text
2. Every word in a fixed vocab is initialized by a random vector
3. Iterate through each position t in text which has a center word c and context ("outside") words o.
4. Use the similarity of the word vectors o and c to calculate the probability of o given c (or vice-versa) $\longrightarrow P(w_{t+j}|w_t)$
5. Keep adjusting the word vectors to maximize this probability.

Objective Function:

- Calculate the Likelihood - which is how good a job the model does in predicting the words around it.

$$Likelihood,\ L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} logP(w_{t+j}|w_t;\theta)$$

- Objective Fn -

$$J(\theta) = -\frac{1}{T} logL(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} logP(w_{t+j}|w_t;\theta)$$

- $Minimizing\ obj.\ function \iff Maximizing\ prediction\ accuracy$

How to calculate $P(w_{t+j}|w_t;\theta)$? - Simplest way is to create two vectors for each word: v_w where w is a center word, u_w where w is a context word. For one center word and one context word:

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \epsilon V} exp(u_w^T v_c)}$$

E.g., ...problems turning **into** banking crisis....., Calculating $P(u_{problems}|v_{into})$

Train a model and optimize parameters: Recall that θ has all model params in one long vector. In this case we have d-dimensional vectors and V-many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \epsilon \mathbb{R}^{2dV}$$

Optimize this by walking in the direction of gradient descent.
Idea: Word vectors will eventually learn what words should appear in it's context and so would give high probability estimates for those context words.

TODO: Calculate the gradient of $J(\theta)$ w.r.t. v_c and u_w. For center word, the inference we get is that the $gradient\ =\ observed\ representation\ of\ context\ word\ -\ expected\ representation\ of\ context\ word$