CS224n

Thomas Lu

1 Lecture 1: Introduction

This lecture basically just gives a high level overview of the fields of Natural Language Processing (NLP) and Deep Learning (DL). Some highlights from the lecture:

- NLP is a study that aims for computers to understand natural (human) language well enough to perform useful tasks.
- NLP has a lot of useful applications (virtual assistant, customer support automation, etc.) and has taken off commercially in the past few years.
- What is special about NLP (vs. other subfields of ML)?
  - Language is specifically constructed to convey information. It's not an environmental signal (e.g. an Amazon purchase, which I probably made for a reason other than conveying information).
  - Language is (mostly) a discrete/symbolic/categorical system, but our brains, which process language, are continuous systems.
- What is special about DL (vs. other ML techniques)?
  - Traditionally, learning an ML model requires an engineer/researcher to design and build relatively human-interpretable features. The machine then basically does some numerical optimization on these features to produce something useful. Note that this still requires a lot of intuitive human work to interpret data and figure out what parts of it might be useful.
  - In contrast, deep learning just feeds "raw data" or very simplistic representations of data to a deep neural net, which then learns its own effective representations of the data, obviating the need for feature engineering.
  - DL has performed very well recently, e.g. speech recognition, ImageNet.
- Why is NLP hard?
  - Human language contains lots of contextual information. The meaning of a sentence may depend on something that was said a few sentences ago, a piece of information from another document, current events, or something observable in the environment ("Did you see that dog?" refers presumably to a dog in the speaker's field of vision).
  - Human language is also ambiguous by design; since speech is pretty slow at transmitting information, people will almost always leave "relatively obvious" things unsaid in order to improve communication efficiency. Listeners are expected to infer these things. This is very different from computer languages, which must specify every posibility.

2 Lecture 2: word2vec

One part of NLP is word meaning representation. How do we represent the definition of a word so that a computer can do something useful with that representation?

One method is WordNet, which is a database of words that groups words with similar meaning together into synsets, which are also arranged in a kind of hierarchy. But this has some problems:

- Synonym/hypernym relationships miss a lot of nuance (e.g. "expert", "proficient", "good" don't really mean the same thing).
- It's hard to keep the database up-to-date when new words are added or existing words gain/lose usages and meanings.
- Creating the database is subjective and labor-intensive.
- Given two words, there isn't a good way to get a measure of how similar they are.

This is an example of a discrete/categorical/symbolic representation of words, where each word represents a single concept. Words might be related, but separate words are generally regarded as separate entities. If these words were vectors, each one would be a V-dimensional vector of zeroes with a single 1, where V is the vocabulary size. This is sometimes referred to as a "localist" or "one-hot" representation.

A problem with these representations is that they lack similarity measures; we could try to manually encode some (as with WordNet), but they don't exist in the representations themselves (e.g. any two distinct word vectors are orthogonal). We can instead try to encode words so that the include similarity information, where a dot product of two words can give you a measure of their similarity. This gives us representations of words as shorter vectors, and this is known as a "distributed" representation: instead of the meaning of a word being "localized" to a single component of a vector, it's "distributed" across multiple components.

One idea for generating distributed representations is the idea of distributional similarity, where we figure out the meaning of a word based on the contexts it appears in, i.e. its distribution in text. (The word "distributional" in "distributional similarity" is not related to the word "distributed" in "distributed representation".) This idea is summed up in a well-known quote: "You shall know a word by the company it keeps." (John Rupert Firth)

word2vec is a well-known distributed representation model. Its key idea is to predict **outer words** that will appear in the context of a **center word**. More specifically, given a center word w_c, it attempts to predict the probabilities $p(w_o|w_c)$ that various outer words w_o will appear in the context of w_c. Supposing we have a training corpus of T words, the training process will attempt to maximize

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

where θ is the parametrization of the model, the context of a word is defined as the m words before and after it, w_i is the i-th word in the corpus, and $p(w_i|w_j;\theta)$ denotes the model's predicted probability of w_i appearing in the context of w_j given its parametrization θ. Equivalently, the training process attempts to minimize the objective function

$$J(\theta) = -\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m, \\ j \neq 0}} \log p(w_{t+j}|w_t;\theta).$$

More concretely, the model is parametrized by $2|V|$ word vectors, where V is the vocabulary. Each word $w \in V$ has two vector representations: a center word representation v_w and an outer word representation

u_w. The model then predicts

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w \in V} \exp\left(u_w^T v_c\right)}.$$

More intuitively, this means that the model predicts that o is likely to appear in the context of c if the outer word representation of o is similar to the center word representation of c.

Because the training corpus for a word2vec model is typically very large, the model is usually trained using stochastic gradient descent, regarding each context window as a single training example. Now the gradient of J with respect to θ is a little tricky to notate, but since J is just a sum of terms of the form $\log p(w_o|w_c)$, the partial derivatives of that term will be quite informative. We have:

$$\begin{aligned}
\frac{\partial}{\partial u_o} \log p(o|c) &= \frac{\partial}{\partial u_o} \log \frac{\exp\left(u_o^T v_c\right)}{\sum_{w \in V} \exp\left(u_w^T v_c\right)} \\
&= \frac{\partial}{\partial u_o} \left[\log \exp\left(u_o^T v_c\right) - \log \left(\sum_{w \in V} \exp\left(u_w^T v_c\right) \right) \right] \\
&= v_c - \frac{1}{\sum_{w \in V} \exp\left(u_w^T v_c\right)} \frac{\partial}{\partial u_o} \sum_{w \in V} \exp\left(u_w^T v_c\right) \\
&= v_c - \frac{1}{\sum_{w \in V} \exp\left(u_w^T v_c\right)} \frac{\partial}{\partial u_o} \exp\left(u_o^T v_c\right) \\
&= v_c - \frac{\exp\left(u_o^T v_c\right)}{\sum_{w \in V} \exp\left(u_w^T v_c\right)} v_c \\
&= v_c \left(1 - p(o|c)\right) \ .
\end{aligned}$$

Similarly, we can calculate

$$\begin{aligned}
\frac{\partial}{\partial v_c} \log p(o|c) &= \frac{\partial}{\partial v_c} \log \frac{\exp\left(u_o^T v_c\right)}{\sum_{w \in V} \exp\left(u_w^T v_c\right)} \\
&= \frac{\partial}{\partial v_c} \left[\log \exp\left(u_o^T v_c\right) - \log \left(\sum_{w \in V} \exp\left(u_w^T v_c\right) \right) \right] \\
&= u_o - \frac{1}{\sum_{w \in V} \exp\left(u_w^T v_c\right)} \frac{\partial}{\partial v_c} \sum_{w \in V} \exp\left(u_w^T v_c\right) \\
&= u_o - \sum_{w \in V} \frac{\exp\left(u_w^T v_c\right)}{\sum_{w' \in V} \exp\left(u_{w'}^T v_c\right)} u_w \\
&= u_o - \sum_{w \in V} p(w|c) u_w \\
&= u_o \left(1 - p(o|c)\right) - \sum_{\substack{w \in V \\ w \neq o}} p(w|c) u_w \ .
\end{aligned}$$

Intuitively, this means that when we encounter a word o in the context of another word c, we should:

- Move the outer word representation u_o of o towards the center word representation v_c of c by an amount negatively correlated with the prior prediction $p(o|c)$. This means that we should make a larger update for more unexpected events.
- Move the center word representation v_c of c:
  - towards the outer word representation u_o of o by an amount negatively correlated with the prior prediction $p(o|c)$, meaning that we should make a larger update for more unexpected events; and

– away from the outer word representations u_w for each w_o by an amount positively correlated with the prior prediction $p(w|c)$, meaning that we should make a larger update for more unexpected events (since observing o in the context of c means that we did not observe w in the location of o).

This matches our intuition that we expect to see o in the context of c if u_o is similar to v_c.