A Neuro Probabilistic Language Model
Bengio et. al. 2003

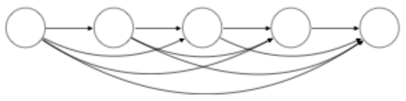
Class Discussion Notes
Scribe: Olivia Winn
February 1, 2016

Opening thoughts (or why this paper is interesting): Word embeddings currently have a massive impact in NLP, and this is the paper that began it all. Additionally, the authors are excellent writers; there is a lot of important introductory information, and since the paper discusses neural nets (NNs) before they became hyped it has a more 'sober' approach than more recent works. Finally, language modeling is a classical example of discrete data.

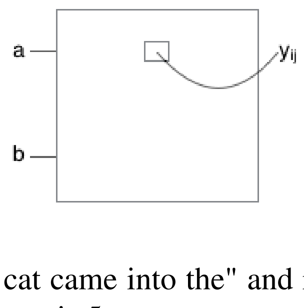
Language Models - Background

The goal in language modeling is to estimate the joint distribution of a sequence of words

$$p(w_1, ..., w_T) = \prod_{i=1}^{T} p(w_i | w_{1:(i-1)})$$



This is incredibly difficult to work with and difficult to estimate. One solution is to reduce the length of the chain. As Bengio puts it, "one considerably reduces the difficulty of this modeling problem by taking advantage of word order, and the fact that temporally closer words in the word sequence are statistically more dependent" [1138]. Thus n-gram models are used as an approximation. However, this still leaves the issue of sparsity:



Let's assume row a is "the cat came into the" and row b is "the dog came into the", word j is "room", and $y_{a,j}$ is 5.

Problem: $y_{b,j}$ should have a similar probability, but if we didn't see it in the training data then we can't predict it! The <u>curse of dimensionality</u>.

Current solutions: smoothing (add a small probability to unseen data), and back-off (if you don't have the trigram, go to the bigram, and if that wasn't present in the data use the unigram).

Bengio's Intuitions

1. Discrete Data:
   - "There is much more information in the sequence that immediately precedes the word to predict than just the identity of the previous couple of words" [1138]
   - Training does not understand similar semantics or grammatical rules between words
   - Each row is completely independent - no data can be transferred
2. Continuous Data:
   - "In high dimensions, it is crucial to distribute probability mass where it matters rather than uniformly in all directions around each training point"
   - Previous smoothing approaches do just this
   - Want to estimate the joint probability by embedding the discrete data in a continuous space
   - This way local smoothness can be assumed

Question: Clusterings vs embedding?

- There is work clustering words into classes - replaces large discrete data set with smaller discrete data set. Class based back-off model is helpful but not as flexible.
- Embeddings allow more relationships between points (vs. binary in class or not in class)
- Semantic clustering? How to handle polysemous words? There has been work in embedding the contexts of words and clustering those to gain semantic understanding.
- How do you handle new words with clustering? (New words discussed later)

Algorithm

1. Each term is associated with a (continuous space) representation
2. Express the joint distribution of words as a function of those representations
3. Simultaneously fit both the joint and the representations

- Training data: $w_1, ..., w_T$

$$p(w_i|w_{1:(i-1)}) = f(w_i, ..., w_{i-n}; \theta)$$

- Traditionally, θ was a matrix of probabilities; now it is a set of parameters.
- f has two elements:
  1) word representations $\lambda_v \in \mathbb{R}^m$ [λ_v is $\mathrm{C}(v)$ in the paper]
  2) $f(w_i, ..., w_{i-n}) = g_w(\lambda_{w_{i-1}}, ..., \lambda_{w_{i-n}}; v)$
- Each word has a single representation. λ_v can be thought of as a linear transformation from the one-hot (indicator) vocabulary-length vector per word to an m-dimensional real-valued vector per word.

$$g_v(\lambda) \propto \exp\{y_v\}$$

$$y_v = b + \sum_{i=1}^{n} \beta_{iv}^T \lambda_{w_i} + \sum_{j=1}^{h} u_{jv} \tanh(d + \sum_{i=1}^{n} \alpha_{ij} \lambda_{w_i})$$

  - parameters: α (per hidden unit), β, u
  - b and d: intercepts
  - summation over n: n represents n-gram (not # of data points)
  - $\sum_{i=1}^{n} \beta_{iv}^T \lambda_{w_i}$ is 'direct' component
  - β: corresponds to how log prob of word v changes. $\beta_{\dot{v}}$ is an embedding / representation of term v in m-space
  - u_{ij}: weight of hidden unit. $u_{\dot{v}}$ is a representation of term v in the space of hidden units
  - tanh: encapsulates linear model of how much each dimension matters, but has something akin to an on-off response (neurons are either firing or not; hence the NN idea)

  - **Question**: Why do we need to sum over both words and the hidden nodes?
    - According to the paper, the direct component isn't actually needed - generalization was better without, but it sped the training up

    - 'Highway' method: In deeper NNs, trend has been to cascade the direct component into a higher layer to ease with inference - can collapse weights (?) when doing gradient descent
    - Theoretically, with infinite dimensions and infinite data the hidden layer would be the 'universal function' and converge perfectly. Practically....

- $\log p(w_{1:T}) = \sum_{i=1}^{T} \log p(w_i | w_{i-1}, w_{i-2})$ is now an optimization problem
- Put this through softmax to get word probabilities (bottleneck!)

$$g_v(\lambda) = \exp\{y_v\} / \sum_{v'} \exp\{y_{v'}\}$$

- **Question**: Is there a way to avoid normalization? Possibly through putting priors on β and u?
  - Even with priors, wouldn't the dimensionality still be a problem?
  - You can sample in order to approximate the summation (how many words really have to contribute? Not many), but you still have to repeat this procedure over the entire vocabulary

Extensions, Future Work

- Out-of-vocabulary words: (also answering a previous question)
  - If we had priors on β and λ and no NN - with a new word, we know the λ of its context (surrounding words), which means we know something about β, and that helps us figure out λ
- Polysemous words: new work currently happening in this direction
- LSI: take SVD of the term-count matrix
  - Supervised factorization analysis (?)
- **Question**: People now use other people's word embeddings as plug-ins for their own work. Should these be modified for the new task they are being used for?
  - These embeddings were made for the downstream task of n-gram modeling - when they used LSI it didn't work as well!
  - If you have a smaller corpus, being able to use richer word embeddings than you could generate can be useful
  - Want to capture similarity of 'bird' and 'birds', but no stemming. Also, how to handle rare words? (Bayesian should help)

- Building a structure on top of the word embeddings (non-negativity structure)
- Tree structure??
  - Size of context determines whether more syntactic or semantic understanding is being generated
  - State-of-the-art parsers use NN, but some people actually use parse trees as input to their NNs to generate word embeddings