Neural network language models

Lecture, Feb 16
CS 690N, Spring 2017
Advanced Natural Language Processing
http://people.cs.umass.edu/~brenocon/anlp2017/

Brendan O’Connor
College of Information and Computer Sciences
University of Massachusetts Amherst
Neural Language Models

Feed forward network

\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]
Nonlinear activation functions

$$\text{sigmoid}(x) = \frac{e^x}{1 + e^x}$$

$$\text{tanh}(x) = 2 \times \text{sgm}(x) - 1$$

$$(x)_+ = \max(0, x)$$  
*a.k.a. “ReLU”*
Trigram NN language model

Word embeddings

\[ h_n = g(V[w_{n-1}; w_{n-2}] + c) \]
\[ \hat{p}_n = \text{softmax}(W h_n + b) \]
\[ \text{softmax}(u)_i = \frac{\exp u_i}{\sum_j \exp u_j} \]

- \( w_i \) are one hot vectors and \( \hat{p}_i \) are distributions,
- \( |w_i| = |\hat{p}_i| = V \) (words in the vocabulary),
- \( V \) is usually very large \( > 1e5 \).

[Slide: Phil Blunsom]
Neural Language Models: Sampling

\[ w_n \mid w_{n-1}, w_{n-2} \sim \hat{p}_n \]
Neural Language Models: Sampling

\[ w_n | w_{n-1}, w_{n-2} \sim \hat{p}_n \]
Neural Language Models: Sampling

\[ w_n | w_{n-1}, w_{n-2} \sim \hat{p}_n \]
Neural Language Models: Sampling

$$w_n | w_{n-1}, w_{n-2} \sim \hat{p}_n$$
Neural Language Models: Sampling

\[ w_n \mid w_{n-1}, w_{n-2} \sim \hat{p}_n \]
Neural Language Models: Training

The usual training objective is the cross entropy of the data given the model (MLE):

$$\mathcal{F} = -\frac{1}{N} \sum_n \text{cost}_n(w_n, \hat{p}_n)$$

The cost function is simply the model’s estimated log-probability of \(w_n\):

$$\text{cost}(a, b) = a^T \log b$$

(assuming \(w_i\) is a one hot encoding of the word)
Calculating the gradients is straightforward with back propagation:

\[
\frac{\partial F}{\partial W} = -\frac{1}{N} \sum_n \frac{\partial \text{cost}_n}{\partial \hat{p}_n} \frac{\partial \hat{p}_n}{\partial W}
\]

\[
\frac{\partial F}{\partial V} = -\frac{1}{N} \sum_n \frac{\partial \text{cost}_n}{\partial \hat{p}_n} \frac{\partial \hat{p}_n}{\partial h_n} \frac{\partial h_n}{\partial V}
\]
Neural Language Models: Training

Calculating the gradients is straightforward with back propagation:

$$\frac{\partial F}{\partial W} = -\frac{1}{4} \sum_{n=1}^{4} \frac{\partial \text{cost}_n}{\partial \hat{p}_n} \frac{\partial \hat{p}_n}{\partial W}, \quad \frac{\partial F}{\partial V} = -\frac{1}{4} \sum_{n=1}^{4} \frac{\partial \text{cost}_n}{\partial \hat{p}_n} \frac{\partial \hat{p}_n}{\partial h_n} \frac{\partial h_n}{\partial V}$$

Note that calculating the gradients for each time step $n$ is independent of all other timesteps, as such they are calculated in parallel and summed.
Comparison with Count Based N-Gram LMs

**Good**

- Better generalisation on unseen n-grams, poorer on seen n-grams. Solution: direct (linear) ngram features.
- Simple NLMs are often an order magnitude smaller in memory footprint than their vanilla n-gram cousins (though not if you use the linear features suggested above!).

**Bad**

- The number of parameters in the model scales with the n-gram size and thus the length of the history captured.
- The n-gram history is finite and thus there is a limit on the longest dependencies that can be captured.
- Mostly trained with Maximum Likelihood based objectives which do not encode the expected frequencies of words a priori.

[Slide: Phil Blunsom]
Training NNs

- Dropout (preferred regularization method)
- Minibatching
- Parallelization (GPUs)

- Local optima?
Word/feature embeddings

- “Lookup layer”: from discrete input features (words, ngrams, etc.) to continuous vectors
  - Anything that was directly used in log-linear models, move to using vectors

- Learn or not?
  - Learn: they’re just model parameters
  - Fixed: use pretrained embeddings
    - Use a faster-to-train model on very large, perhaps different, dataset [e.g. word2vec, glove pretrained word vectors]
  - Both: initialize with pretrained, then learn
    - Word at test but not training time?

- Shared representations for domain adaptation and multitask learning
Local models
\[ w_t \mid w_{t-2}, w_{t-1} \]

- Fully observed
direct word models

- Latent-class
direct word models

Long-history models
\[ w_t \mid w_1, \ldots, w_{t-1} \]

- Log-linear models

- Markovian neural LM

- Recurrent neural LM