Natural Language Processing: High-Dimensional Ouputs

Sargur N. Srihari
srihari@cedar.buffalo.edu

This is part of lecture slides on Deep Learning:
http://www.cedar.buffalo.edu/~srihari/CSE676

Topics in NLP

1. N-gram Models
2. Neural Language Models
3. High-Dimensional Outputs
4. Combining Neural Language Models with n-grams
5. Neural Machine Translation
6. Historical Perspective

Topics in High-Dimensional Outputs

- Overview

1. Use of a Short List
2. Hierarchical Softmax
3. Importance Sampling
4. Noise-Contrastive Estimation and Ranking Loss

Word vocabularies can be large

- In many NLP applications we want our models to produce words (rather than characters) as the fundamental output
- For large vocabularies it is computationally expensive to represent output distribution over the choice of a word because the vocabulary size is large
  - Ex: In many applications V contains $100\mathrm{K}$ words

Naiive Approach

- Naiive approach
  1. Apply an affine transformation from hidden representation to output space
  2. Then apply a softmax function from hidden representation to output space
- The Weight matrix representing this affine transformation is very large because output dimension is $|\mathbb{V}|$
  – High memory cost to represent it and high computational cost to multiply by it

Cost at both training and testing

- Because softmax is normalized across all $|\mathrm{V}|$ outputs, it is necessary to perform full matrix multiplication at training time and testing time
- We cannot calculate only the dot product with the weight vector for the correct output
- Thus the high computational cost of the output layer arises at both
  - Training time (to compute likelihood and gradient)
  - Testing time (to compute probabilities for all selected words)

Cost of Naiive Mapping to Words

- Suppose $\boldsymbol{h}$ is the top hidden layer used to predict output probabilities $\hat{y}_i$
- If we parameterize the transformation from $\boldsymbol{h}$ to $\hat{y}_i$ with learned weights W and learned biases $\boldsymbol{b}$, then the affine-softmax layer performs the following computations

$$a_i = b_i + \sum_j W_{ij} h_j \quad \forall i \in \{1,..|V|\}$$

$$\hat{y}_i = \frac{e^{a_i}}{\sum_{i'=1}^{|V|} e^{a_{i'}}}$$

- If $\boldsymbol{h}$ contains n_h elements then the above operation is $O(|\mathrm{V}|n_h)$
  - n_h is in the thousands and $|\mathrm{V}|$ is in hundreds of thousands

Use of a Short List

- First neural language models to deal with high cost of using softmax over large V:
  - Split V into shortlist L of frequent words (say $10{,}000$ words handled by a neural net) and tail $\mathrm{T}=\mathrm{V}/\mathrm{L}$ of rare words (handled by an n-gram model)
    - To combine two predictions, NN has to predict probability that word appearing after context C belongs to tail list
      - By extra sigmoid output unit to provide an estimate $P(i \in \mathbb{T} \mid C)$.
      - The extra output can then be used to achieve an est probability estimate over all words in V as follows:

$$
\boxed{\begin{aligned} P(y=i \mid C) = & 1_{i\in\mathbb{L}} P(y=i \mid C, i \in \mathbb{L})(1 - P(i \in \mathbb{T} \mid C)) \\ & + 1_{i\in\mathbb{T}} P(y=i \mid C, i \in \mathbb{T}) P(i \in \mathbb{T} \mid C) \end{aligned}}
$$

      - where $P(y=i|C,\ i\ \varepsilon\ \mathrm{L})$ is provided by neural language model
      - and $P(y=i|C,\ i\ \varepsilon\mathrm{T})$ is provided by the n-gram model

Disavantage of Short-list approach

- Potential generalization advantage of the neural language models is limited to the most frequent words
  - Where it is least useful
- This disadvantage has stimulated exploration of alternative methods to deal with high-dimansional outputs
  - Described next

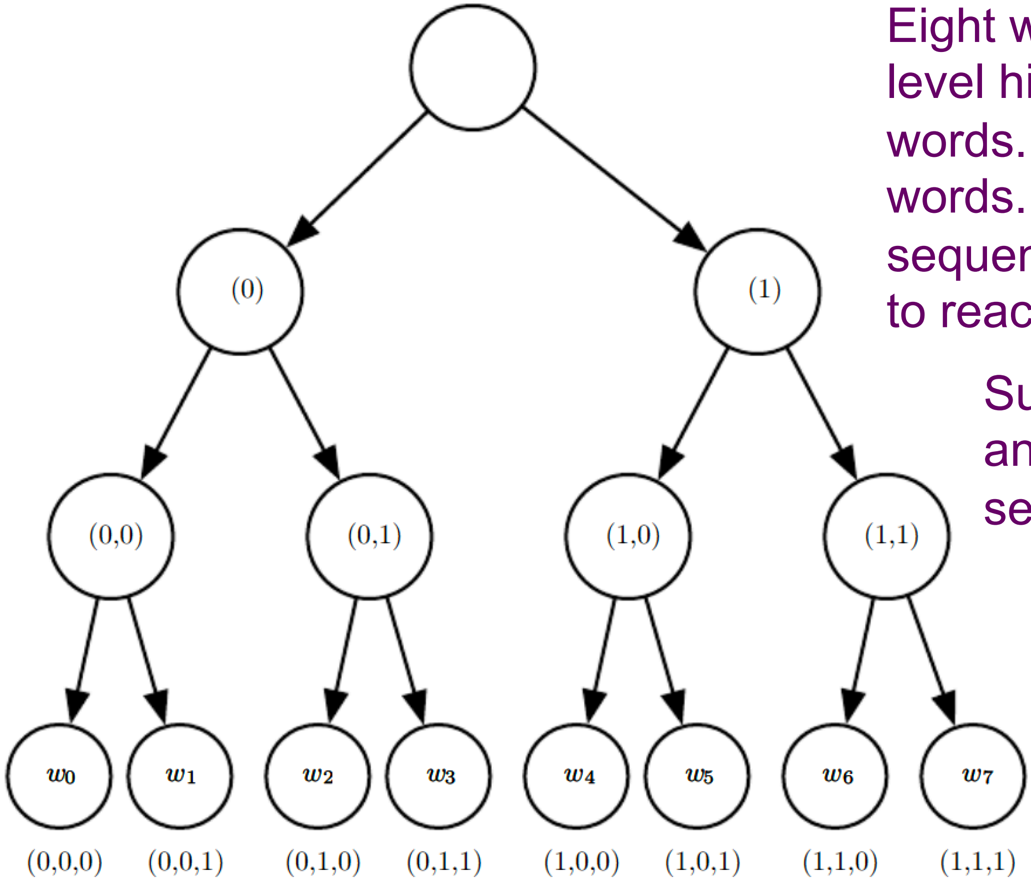
Hierarchical Softmax

- Classical approach to reducing computational burden of high-dimensional outputs over large vocabulary sets $\mathbb{V}$ is to decompose probabilities hierarchically
- Instead of necessitating a no. of computations proportional to $|\mathbb{V}|$ (and also proportional to no. of computations proportional to no. of hidden units n_h), the $|\mathbb{V}|$ factor can be reduced to as low as $\log |\mathbb{V}|$

Hierarchy of Words

- Hierarchy builds categories of words
- Then categories of categories of words, etc
- Nested categories form a tree
  - With words at the leaves
- In a balanced tree, tree has depth $O(\log|\mathrm{V}|)$
- Probability of choosing a word is given by:
  - The product of the probabilities of choosing the branch leading to that word at every node on a path from the root of the tree to the leaf containing the word
  - A simple example is given next

Simple Hierarchy of Word Categories



Eight words $w_0,..,w_7$ organized into a three level hierarchy. Leaves represent specific words. Internal nodes represent groups of words. Any node can be indexed by the sequence of binary decisions $(0=\text{left},\ 1=\text{right})$ to reach the node from the root

Superclass (0) contains the classes $(0,0)$ and $(0,1)$ which respectively contain the sets of words $\{w_0,w_1\}$ and $\{w_2,w_3\}$

Superclass (1) contains the classes $(1,0)$ and $(1,1)$ which respectively contain the sets of words $\{w_4,w_5\}$ and $\{w_6,w_7\}$

Node $(1,0)$ corresponds to the prefix $(b_0(w_4)=1, b_1(w_4)=0)$ and the probability of w_4 can be decomposed as:

$$\boxed{\begin{aligned} P(\mathrm{y}=w_4) &= P(\mathrm{b}_0=1,\mathrm{b}_1=0,\mathrm{b}_2=0) \\ &= P(\mathrm{b}_0=1)P(\mathrm{b}_1=0 \mid \mathrm{b}_0=1)P(\mathrm{b}_2=0 \mid \mathrm{b}_0=1,\mathrm{b}_1=0) \end{aligned}}$$

Importance Sampling

- Training of neural language models can be speeded up by avoiding explicit computation of contribution to the gradient from all words that do not appear in the next position
- Every incorrect word should have low probability under the model
- Instead of enumerating all words, it is possible to sample only a subset of words

Gradient using Sampling

Using notation:

$$a_i = b_i + \sum_j W_{ij} h_j \quad \forall i \in \{1, \ldots, |\mathbb{V}|\},$$

$$\hat{y}_i = \frac{e^{a_i}}{\sum_{i'=1}^{|\mathbb{V}|} e^{a_{i'}}}.$$

where $\boldsymbol{a}$ is the vector of pre-softmax activations (or scores) with one element per word.

the gradient written as follows:

$$\begin{aligned}
\frac{\partial \log P(y \mid C)}{\partial \theta} &= \frac{\partial \log \operatorname{softmax}_y(\boldsymbol{a})}{\partial \theta} \\
&= \frac{\partial}{\partial \theta} \log \frac{e^{a_y}}{\sum_i e^{a_i}} \\
&= \frac{\partial}{\partial \theta}\left(a_y - \log \sum_i e^{a_i}\right) \\
&= \frac{\partial a_y}{\partial \theta} - \sum_i P(y = i \mid C) \frac{\partial a_i}{\partial \theta}
\end{aligned}$$

The first term is the **positive phase term**
(pushing a_y up)
Second term is the **negative phase term**
(pushing a_i down for all i with weight $P(i|C)$

Since negative phase is expectation, can estimate with a Monte Carlo sample

Sampling from another distribution

- Gradient method based on sampling would require sampling from the model itself
- Sampling from the model requires computing $P(i|C)$ for all i in the vocabulary
  - Which is precisely what we are trying to avoid
- Instead of sampling from the model, one can sample from another distribution, called the proposal distribution (denoted q)
  - And use weights to correct for bias due to sampling from wrong distribution
  - This is an application of importance sampling

Biased Importance Sampling

- Even exact importance sampling is inefficient
- It requires computing weights p_i/q_i where $p_i = P(i|C)$
  - Which can only be computed if all the scores a_i are computed
- The solution adopted is called biased importance sampling
  - Where the importance weights are normalized to sum to 1