

Natural Language Processing: High-Dimensional Outputs

Sargur N. Srihari
srihari@buffalo.edu

This is part of lecture slides on [Deep Learning](#):
<http://www.cedar.buffalo.edu/~srihari/CSE676>

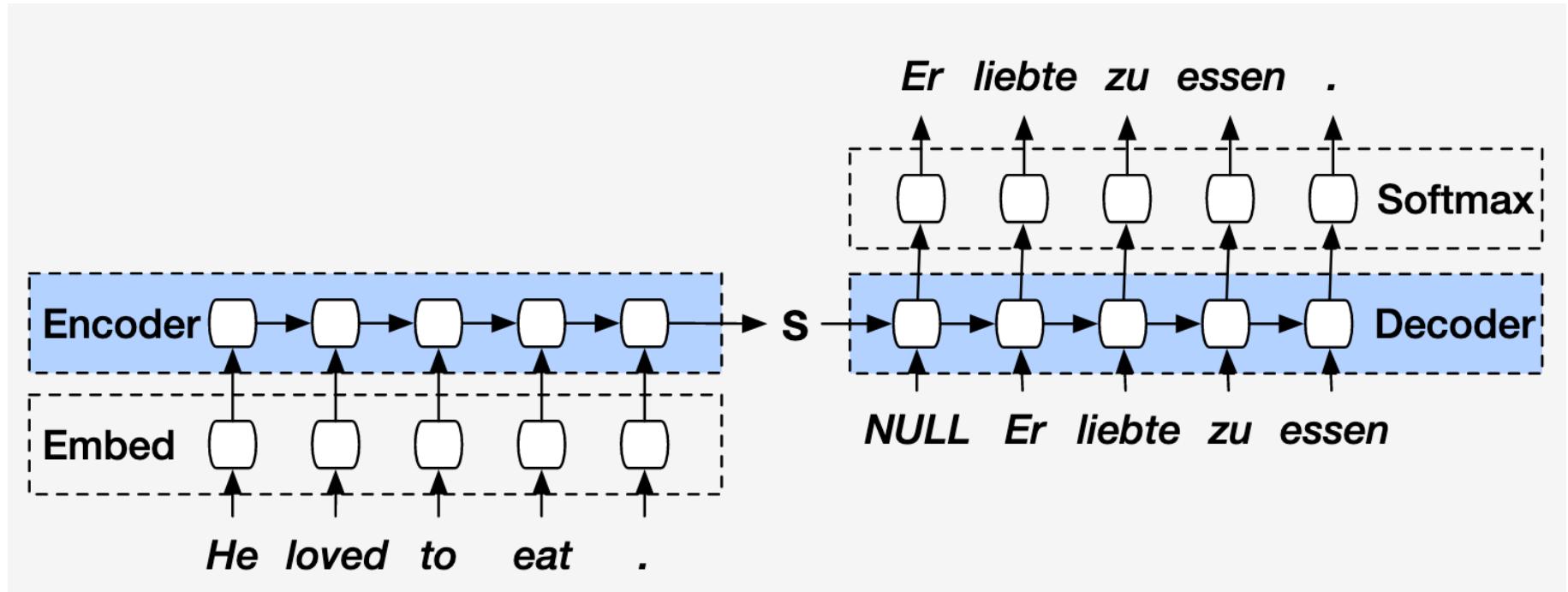
Topics in NLP

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

1. Outputs in Deep Learning
2. Computational Complexity over full vocabulary
3. Use of a Short List
4. Hierarchical Softmax
5. Speeding-up gradient descent using sampling
6. Noise-Contrastive Estimation and Ranking Loss

Outputs in Neural Machine Translation

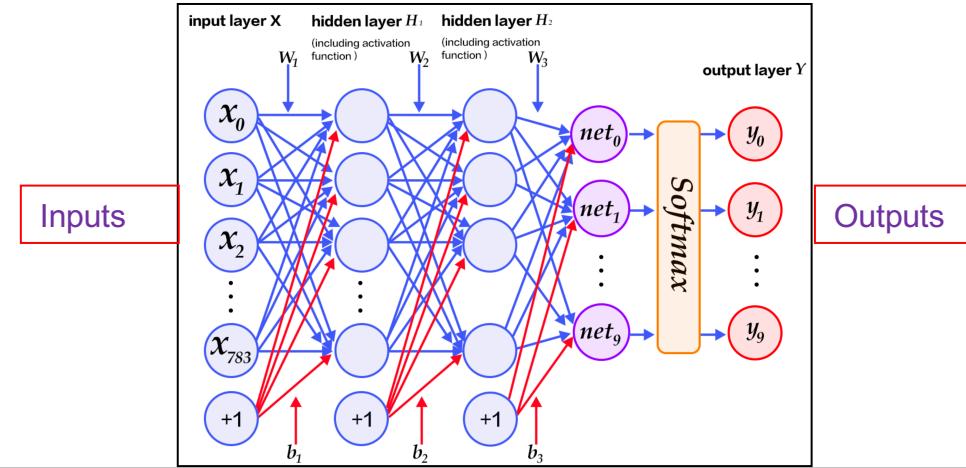
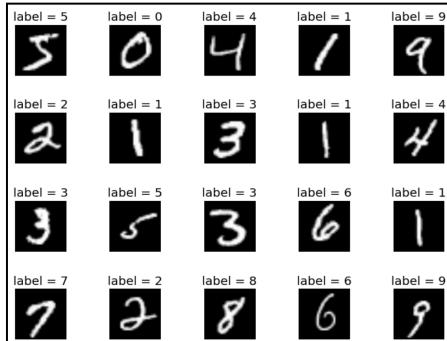


The decoder uses the summary of the input **S** and the previous output word to generate the next output word
Note the use of softmax which we discuss next

Softmax is used in all networks

Feedforward Network

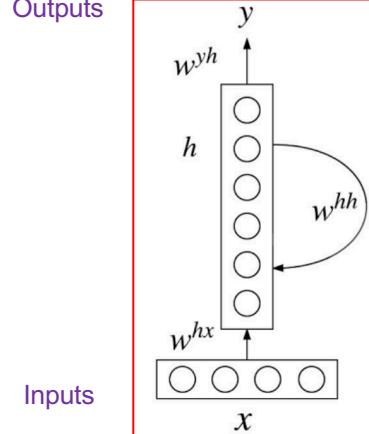
MNIST 28x28 images



Recurrent Neural Network

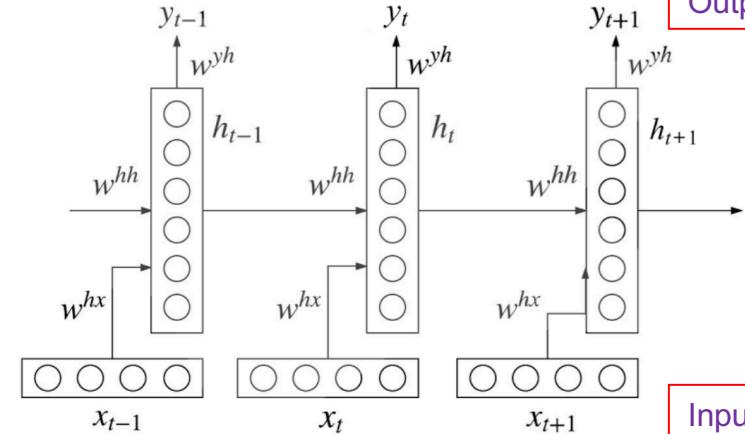
Folded network

Outputs



Unfolded sequence network with three time steps

Outputs



$$h_t = f(w^{hh}h_{t-1} + w^{hx}x_t)$$

$$y_t = \text{softmax}(w^{yh}h_t)$$

Definition of softmax

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

- for $i = 1, \dots, K$ and $\mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$

In NLP $K=|V|$

Unlike a feedforward neural network, which uses different parameters at each layer, RNN shares the same parameters (W^{hx}, W^{hh}, W^{yh}) across all steps

Computational Complexity of Output

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

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

$$\hat{y}_i = \frac{e^{a_i}}{\sum_{i'=1}^{|V|} e^{a_{i'}}} \quad \text{Softmax Output Probabilities}$$

- If h contains n_h elements then above operation is $O(|V|n_h)$
 - n_h is in the thousands and $|V|$ is in hundreds of thousands
 - Implies millions of operations

Word vocabularies can be large

- In many NLP applications models produce words (rather than characters) as output
 - E.g., MT, speech recognition
- High computational expense to represent output distribution over word vocabulary V
 - Ex: In many applications $|V| = 100K$

Size of Output Vector

- Three news datasets of different sizes:
 - Penn Treebank (PTB)
 - WMT11-1m (billionW)
 - English Gigaword,v5 (gigaword)
- Dataset statistics
 - No. of tokens for training and testing
 - Vocabulary size
 - Fraction of Out of vocabulary words

Dataset	Train	Test	Vocab	OOV
PTB	1M	0.08M	10k	5.8%
gigaword	4,631M	279M	100k	5.6%
billionW	799M	8.1M	793k	0.3%

Naïive Mapping to Vocabulary

1. Apply affine transform from hidden to output
 2. Apply softmax from hidden to output space
- Weight matrix for affine transform is large because output dimension is $|V|$
 - High memory cost to represent it
 - High computational cost to multiply by it

High cost at both training and testing

- At testing time
 - Softmax is normalized across all $|V|$ outputs
 - Thus need full matrix multiplication at testing time
 - We cannot calculate only a dot product with weight vector for the correct output
- At training time
 - High computational cost of output at training as well
 - To compute likelihood and gradient
- At testing:
 - To compute probabilities for selected words

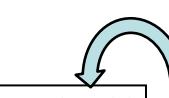
Methods for high-dimensional outputs

1. Use of a short list
2. Hierarchical softmax
3. Speeding-up gradient during training using sampling

Use of a Short List

- To deal with high cost of softmax over large V :
 - Split V into two
 1. Short list L of frequent words handled by a neural net
 2. Tail $T = V \setminus L$ of rare words (handled by an n -gram model)
 - To combine two predictions the NN also predicts:
 - Probability that word after context C belongs to tail list
 - By extra sigmoid output unit to provide an estimate $P(i \in T | C)$.
 - The extra output can then be used to estimate probability over all words in V as follows:

$$P(y = i | C) = \begin{cases} 1_{i \in L} P(y = i | C, i \in L) (1 - P(i \in T | C)) \\ + 1_{i \in T} P(y = i | C, i \in T) P(i \in T | C) \end{cases}$$

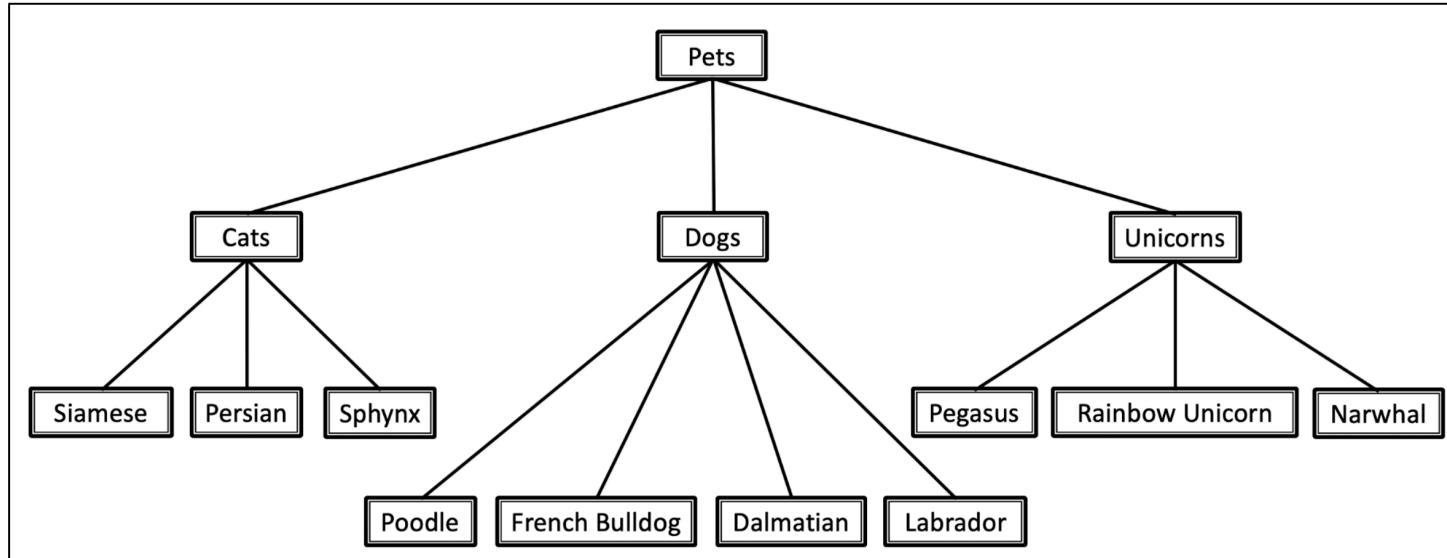


 $P(y=i|C, i \in L)$ is provided by neural language model
 $P(y=i|C, i \in T)$ is provided by the n -gram model

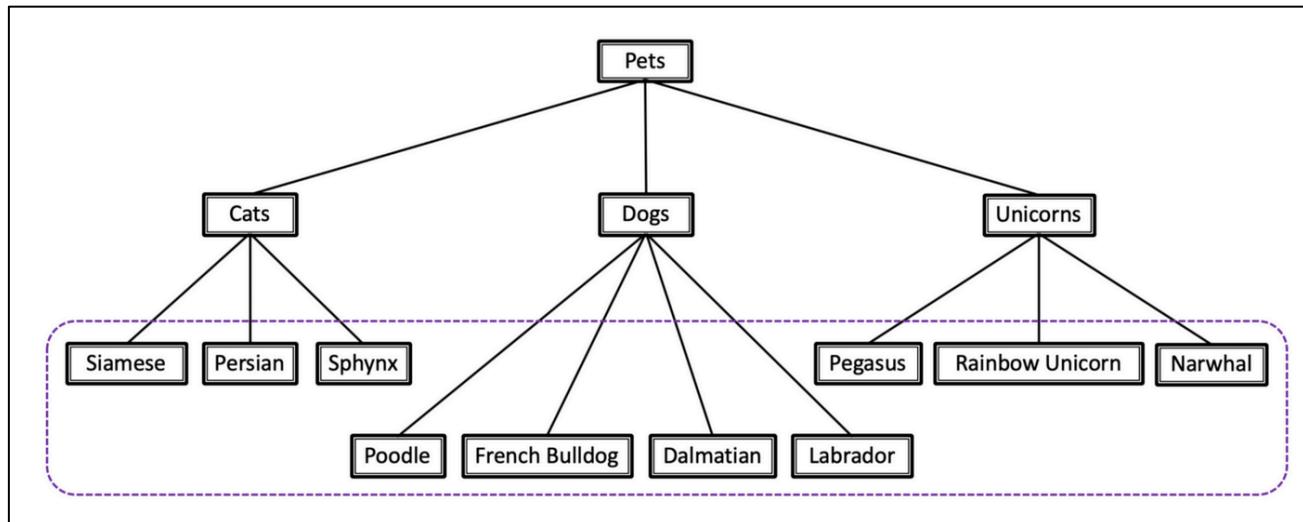
Disadvantage of Short-list approach

- Generalization advantage of NLM is limited to the most frequent words
 - Where it is least useful
- This disadvantage has stimulated exploration of alternative methods to deal with high-dimensional outputs
 - Hierarchical softmax described next

Hierarchy of classes



Hierarchical
Classification



Flat
Classification

Hierarchical Softmax

- Computational burden of large vocabulary V is reduced by decomposing probabilities hierarchically
- Instead of complexity of $|V|$ (and of n_h), the $|V|$ factor reduces to $\log |V|$

Hierarchy of Words

- Hierarchy of 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 $O(\log|V|)$
- Probability of choosing a word is given by:
 - 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, \dots, w_7 organized into a three level hierarchy

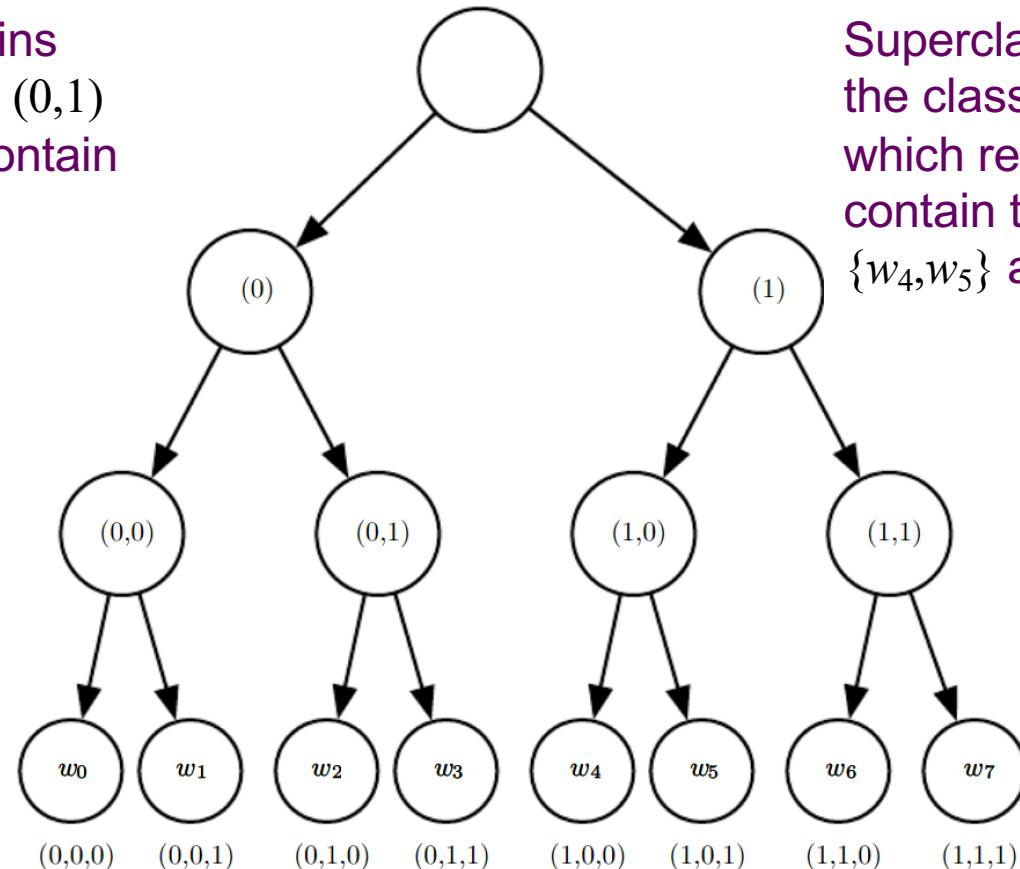
Any node can be indexed by the sequence of binary decisions (0=left, 1=right) to reach the node from the root

Leaves represent specific words. Internal nodes represent groups of words.

Superclass (0) contains the classes (0,0) and (0,1) which respectively contain sets of words

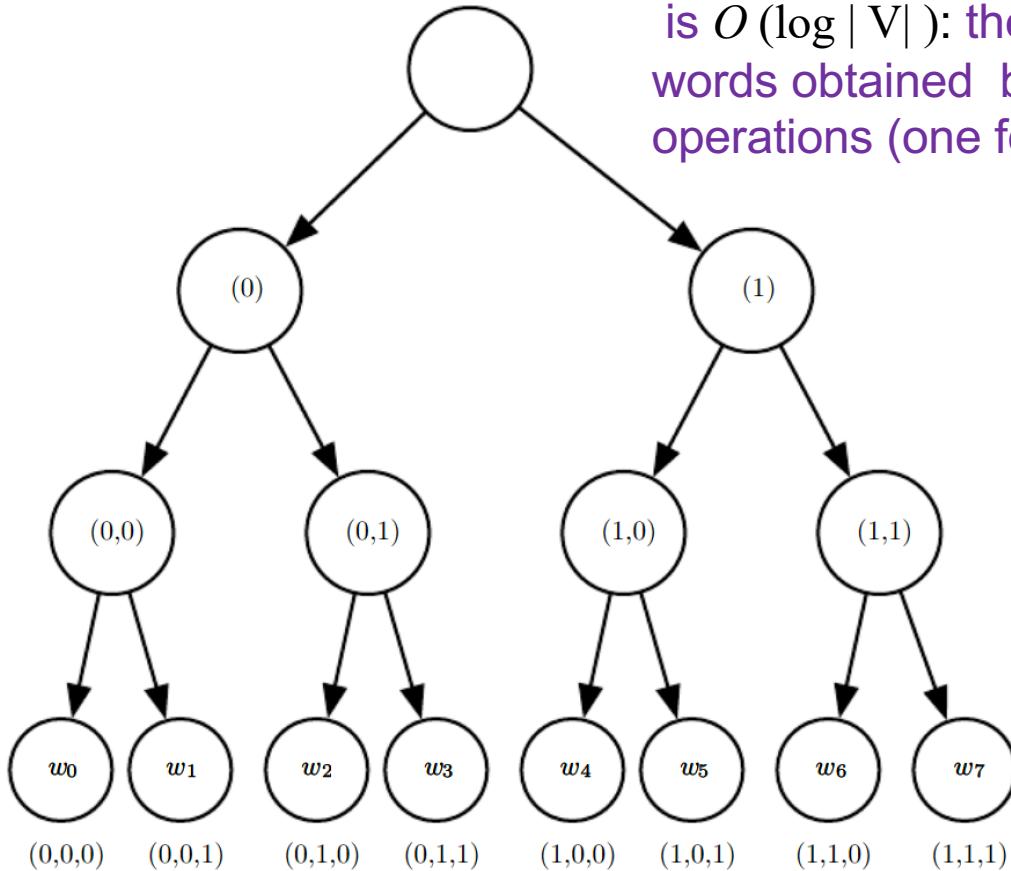
$\{w_0, w_1\}$ and $\{w_2, w_3\}$

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



Computing word probability from tree

If tree balanced, maximum depth (no. of binary decisions) is $O(\log |V|)$: the choice of one out of $|V|$ words obtained by doing $O(\log |V|)$ operations (one for each node on the path from the root)



Computing probability of a word y
 Multiply three node probabilities, associated with the binary decisions
 To move left or right at each node on the path from root to node y .
 $b_i(y)$: i -th binary decision when traversing tree towards value y .

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:

$$\begin{aligned}
 P(y = w_4) &= P(b_0 = 1, b_1 = 0, b_2 = 0) \\
 &= P(b_0 = 1)P(b_1 = 0 | b_0 = 1)P(b_2 = 0 | b_0 = 1, b_1 = 0)
 \end{aligned}$$

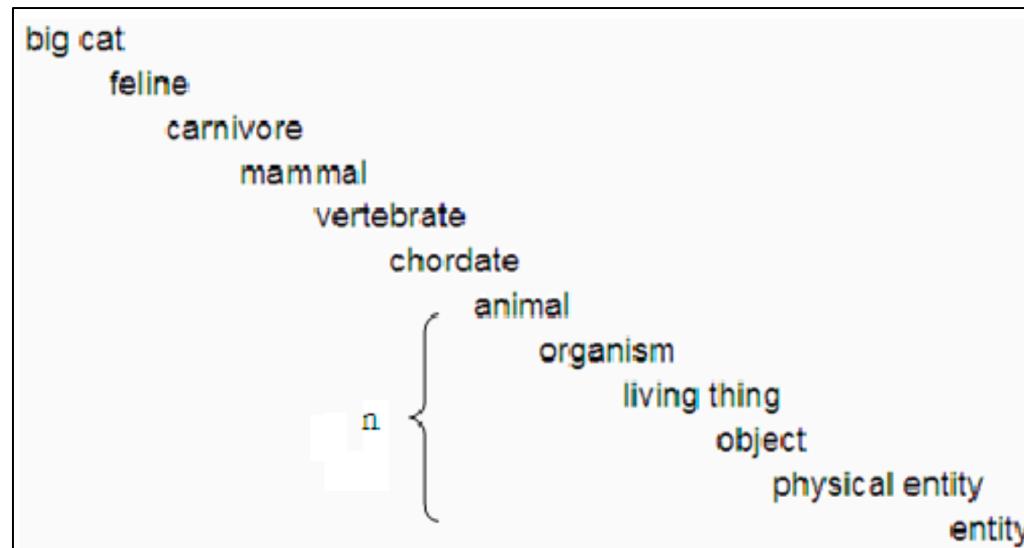
Computing node probabilities

- Multiple paths identify a single word
 - Captures words with multiple meanings
 - Probability of a word is the sum over all paths
- Conditional probability at each node
- Logistic regression for all with same context C
(i.e., word appearing after context C)
 - Supervised learning-- correct output in training set
 - Cross-entropy loss-- maximizing log-likelihood of sequence of decisions

Defining Word hierarchy

1. Use existing hierarchies

- E.g., Wordnet hierarchy for “tiger”



2. Learn hierarchy

- Jointly with the neural language model
- Discrete optimization to partition words into classes

Advantage/disadvantage of Hierarchy

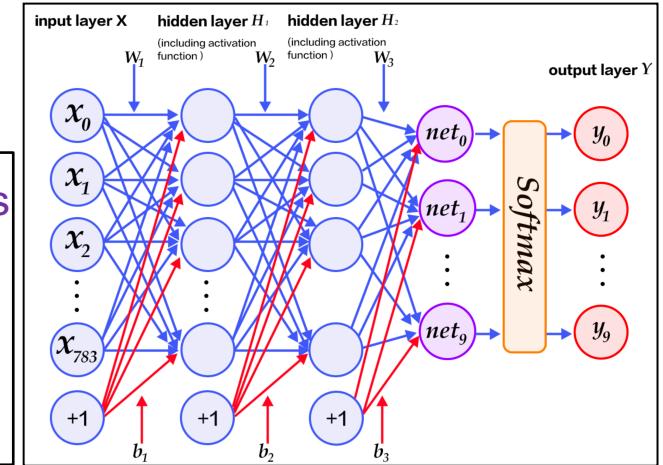
- **Advantage:** Computation time
 - For both training and test time, if at test time we want to compute the probability of specific words
- **Disadvantage:** Worse results
 - Than sampling-based methods described next
 - This may be due to a poor choice of word classes

Speeding-up Gradient during training

- Model with flat output list V

$$a_i = b_i + \sum_j W_{ij} h_j \quad \forall i \in \{1, \dots, |V|\} \text{ Pre-Softmax Activations}$$

$$\hat{y}_i = \frac{e^{a_i}}{\sum_{i'=1}^{|V|} e^{a_{i'}}} \quad \text{Softmax Output Probabilities}$$



$$\text{softmax}(a)_i = \frac{e^{a_i}}{\sum_{j=1}^{|V|} e^{a_j}}$$

- Where y is the output vector of $|V|$ probabilities
- Log-likelihood is the logarithm of softmax output
- The gradient of the log-likelihood is

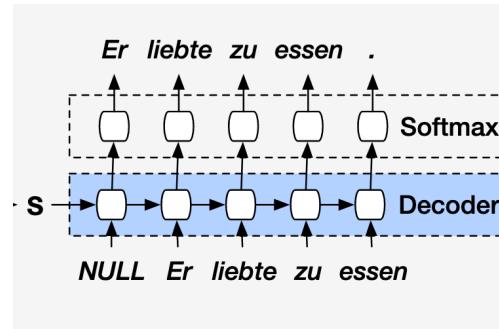
$$\frac{\partial}{\partial \theta} \log \frac{e^{a_y}}{\sum_i e^{a_i}}$$

- Which has contributions from all words i in V

Sampling a subset of words

- Training speeded up by avoiding contribution to gradient from words not in the next position

$$\frac{\partial}{\partial \theta} \log \frac{e^{a_y}}{\sum_i e^{a_i}}$$



e.g., *liebte* is the second word
Other words are incorrect

- Incorrect words should have low probabilities
- Instead of enumerating all words, it is possible to sample only a subset of words
 - As seen next

Decomposing the Output Gradient

Using notation:

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

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

where $\mathbf{a} = [a_1, \dots, a_{|\mathbb{V}|}]$ is the vector of pre-softmax activations (or scores) with one element per word.

the gradient written as follows:

$$\begin{aligned} \frac{\partial \log P(y | C)}{\partial \theta} &= \frac{\partial \log \text{softmax}_y(\mathbf{a})}{\partial \theta} \\ &= \frac{\partial}{\partial \theta} \log \frac{e^{a_y}}{\sum_i e^{a_i}} \\ &= \frac{\partial}{\partial \theta} (a_y - \log \sum_i e^{a_i}) \\ &= \frac{\partial a_y}{\partial \theta} - \sum_i P(y = i | C) \frac{\partial a_i}{\partial \theta} \end{aligned}$$

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 $P(i|C)$)

Note: chain rule

$$\frac{\partial}{\partial \theta} \log_e x = \frac{1}{x} \frac{\partial x}{\partial \theta}$$

$$\frac{\partial}{\partial \theta} e^a = e^a \frac{\partial a}{\partial \theta}$$

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

Importance Sampling

- Gradient method based on sampling would require sampling from the model itself
 - Sampling from 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 model, sample from a 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
 - Because it requires computing weights p_i/q_i
 $p_i = P(i|C)$ can be computed only if all a_i are computed
- Solution is biased importance sampling
 - Where importance weights normalize to sum to 1
 - When negative word n_i is sampled, associated gradient is weighted by

$$w_i = \frac{p_{n_i}/q_{n_i}}{\sum_{j=1}^N p_{n_j}/q_{n_j}}$$

- Which give importance to m negative samples from q used to form the negative phase contribution

$$\sum_{i=1}^{|\mathbb{V}|} P(i | C) \frac{\partial a_i}{\partial \theta} \approx \frac{1}{m} \sum_{i=1}^m w_i \frac{\partial a_{n_i}}{\partial \theta}$$

Choice of Proposal Distribution

- Unigram or a bigram distribution works well for proposal q
 - It is easy to estimate parameters of such a distribution from data
 - After estimating parameters, it is also possible to sample from such a distribution very efficiently

Noise-Contrastive Estimation & Ranking Loss

- Other sampling approaches reduce cost of training with large vocabularies

1. Ranking Loss

$$L = \sum_i \max(0, 1 - a_y + a_i)$$

- Output for each word is a score
- Correct word a_y ranked high over other scores a_y
 - The gradient is zero for the i -th term if the score of the observed word, a_y , is greater than the score of the negative word a_i by a margin of 1

2. Noise contrastive estimation

- A training objective for a neural language model