UNSW
THE UNIVERSITY OF NEW SOUTH WALES

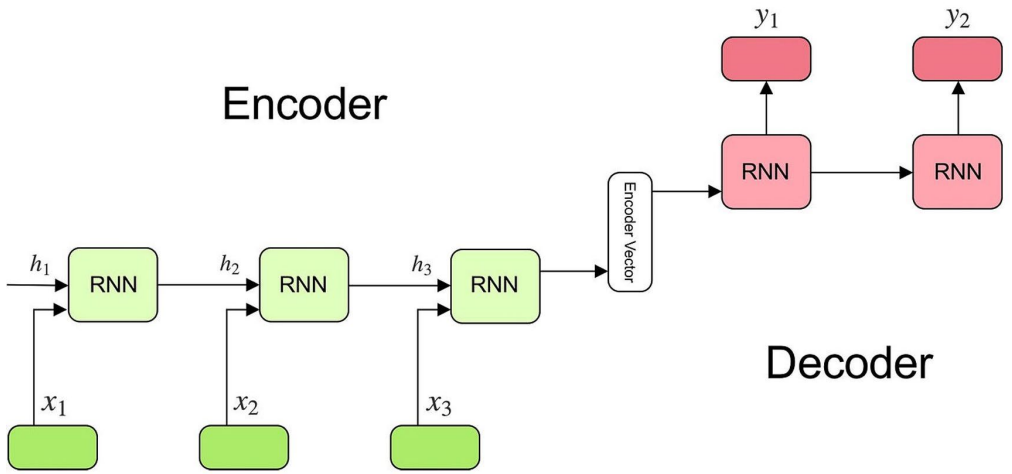
5b. Word Vectors

Never Stand Still | Faculty of Engineering | COMP9444 Week 5

Hao Xue
School of Computer Science and Engineering
Faculty of Engineering
The University of New South Wales, Sydney, Australia
cs9444@cse.unsw.edu.au

Previously: RNNs

- Previously: RNNs for sequence modelling
- Challenge: RNNs require numerical inputs, but words are symbolic
- What we need for language processing: compact, meaningful word representations for neural models
- In this lecture: how to turn words into vectors that RNNs and other models can use



Outline

- Statistical Language Processing
- n-gram models
- co-occurence matrix
- word representation
- Word2Vec
- GloVe
- word relationships

How do we represent the meaning of a word

- Dictionary definitions
  - e.g., Webster dictionary
- Relations
  - Synonyms: have the same meaning in some or all contexts
    - big / large; automobile / car
  - Antonyms: Senses that are opposites with respect to only one feature of meaning
    - hot / cold; fast / slow; dark / light
  - Taxonomy
    - e.g., WordNet
    - Rose is-a Flower is-a Plant is-a Living Thing is-a Thing

Statistical Language Processing

- Problems
  - A useful resource but missing nuance
    - “proficient” is listed as a synonym for “good”
    - This is only correct in some contexts
  - Missing new meanings of words
  - Subjective
  - Requires human labor to create and adapt
  - Can’t be used to accurately compute word similarity (e.g., king and queen)

- Could we instead extract some statistical properties automatically, without human involvement?

Example: There was a Crooked Man

There was a crooked man,
who walked a crooked mile
And found a crooked sixpence
upon a crooked stile.
He bought a crooked cat,
who caught a crooked mouse
And they all lived together
in a little crooked house.

www.kearley.co.uk/images/uploads/JohnPatiencePJ03.gif

Counting Frequencies

- some words occur frequently in all (or most) documents
- some words occur frequently in a particular document, but not generally
- this information can be useful for document classification

word	frequency
a	7
all	1
and	2
bought	1
cat	1
caught	1
crooked	7
found	1
he	1
house	1
in	1
little	1
lived	1
man	1
mile	1
mouse	1
sixpence	1
stile	1
there	1
they	1
together	1
upon	1
walked	1
was	1
who	2

Document Classification

word	doc 1	doc 2	doc X
a	.	.	7
all	.	.	1
and	.	.	2
bought	.	.	1
cat	.	.	1
caught	.	.	1
crooked	.	.	7
found	.	.	1
he	.	.	1
house	.	.	1
in	.	.	1
little	.	.	1
lived	.	.	1
man	.	.	1
mile	.	.	1
mouse	.	.	1
sixpence	.	.	1
stile	.	.	1
there	.	.	1
they	.	.	1
together	.	.	1
upon	.	.	1
walked	.	.	1
was	.	.	1
who	.	.	2

each column of the matrix becomes a vector representing the corresponding document

N-gram Models

- How to predict the next word
- Probability of an upcoming word $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$
- How to compute? (Hint: Chain Rule of Probability)

$$
\begin{aligned}
P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\
&= \prod_{k=1}^{n} P(w_k|w_{1:k-1})
\end{aligned}
$$

N-gram Models

- Simplifying assumption: Markov Assumption

$$P(w_n \mid w_1, \ldots, w_{n-1}) \approx P(w_n \mid w_{n-k+1}, \ldots, w_{n-1})$$

- That is, the probability of w_n depends only on the previous $k-1$ words.
  - 1-gram (Unigram): $P(w_n)$
  - 2-gram (Bigram): $P(w_n \mid w_{n-1})$
  - 3-gram (Trigram): $P(w_n \mid w_{n-2}, w_{n-1})$

- n-gram is a contiguous sequence of 'n' items (words/tokens) from a given sample of text

1-Gram (Unigram) Model

Word	Frequency	Probability
a	7	0.1794871794871795
crooked	7	0.1794871794871795
who	2	0.05128205128205128
And	2	0.05128205128205128
There	1	0.02564102564102564
cat	1	0.02564102564102564
little	1	0.02564102564102564
in	1	0.02564102564102564
together	1	0.02564102564102564
lived	1	0.02564102564102564

- Collect a large corpus of text
- Count the frequency of every unique word
- Calculate the probability of each word

- Prediction: When asked to predict the "next word," the model simply outputs the word that has the highest overall probability of occurring in the training corpus.

2-Gram (Bigram) Model

Bigram	Frequency	Probability
a crooked	6	0.15789473684210
There was	1	0.02631578947368
they all	1	0.02631578947368
cat who	1	0.02631578947368
who caught	1	0.02631578947368
caught a	1	0.02631578947368
crooked mouse	1	0.02631578947368
mouse And	1	0.02631578947368
And they	1	0.02631578947368
all lived	1	0.02631578947368

- Suppose we want to predict the next word after “a”
  - “crooked” 6 times VS. “little” 1 time

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a							$\frac{6}{7}$					$\frac{1}{7}$													
all													1												
and								$\frac{1}{2}$												$\frac{1}{2}$					
bought	1																								
cat																									1
caught	1																								
crooked					$\frac{1}{7}$					$\frac{1}{7}$				$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$							
found	1																								
he				1																					
house																									
in	1																								
little							1																		
lived																					1				
man																									1
mile			1																						
mouse			1																						
sixpence																						1			
stile									1																
there																								1	
they		1																							
together											1														
upon	1																								
walked	1																								
was	1																								
who						$\frac{1}{2}$																		$\frac{1}{2}$	

Co-occurrence Matrix

- Sometimes, we don’t necessarily predict the next , but simply a “nearby word”
- We can build a matrix in which each row represents a word, and each column a nearby word
- A co-occurrence matrix is a count-based representation of how often words appear together in a corpus, typically within a fixed context window. It's one of the earliest methods for learning word semantics from raw text.

Co-occurrence Matrix

- Rows represent target words (words we want vectors for)
- Columns represent context words (words that appear near the target, context window)
- Entries M[i][j] = number of times word j appears near word I
- Used for extracting word meaning and building semantic representations (e.g., Words that occur in similar contexts tend to have similar meanings)

Co-occurrence Matrix (2-word window)

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a				1		1	6	1			1	1										1	1	1	
all													1							1					
and								1							1	1				1					
bought	1								1																
cat							1																		1
caught	1																								1
crooked	6				1					1		1		1	1	1	1	1							
found	1		1																						
he				1														1							
house							1																		
in	1																				1				
little	1						1																		
lived		1																			1				
man							1																		1
mile			1				1																		
mouse			1				1																		
sixpence							1															1			
stile							1		1																
there																								1	
they		1	1																						
together											1		1												
upon	1																1								
walked	1																								1
was	1																		1						
who					1	1								1										1	

There was a crooked man,
who walked a crooked mile
And found a crooked sixpence
upon a crooked stile.
He bought a crooked cat,
who caught a crooked mouse
And they all lived together
in a little crooked house.

Co-occurrence Matrix (10-word window)

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a	10	2	3	2	2	2	13	3	2	1	1	1	1	2	2	1	2	2	1	2	1	2	2	1	4
all	2		1				1				1	1	1			1				1	1				
and	3	1				1	3	1			1		1		1	1	1			1	1	1	1		2
bought	2				1	1	2		1									1				1			1
cat	2			1		1	2		1							1		1							1
caught	2		1	1	1		2									1				1					1
crooked	13	1	3	2	2	2	10	2	2	1	1	1	2	2	2	1	2	3	1	1	1	2	2	1	4
found	3		1				2								1		1					1	1		
he	2			1	1		2										1	1				1			1
house	1						1				1	1									1				
in	1	1	1				1			1		1	1							1	1				
little	1	1					1			1	1		1								1				
lived	1	1	1				2				1	1				1				1	1				
man	2						2								1				1				1	1	1
mile	2		1				2	1						1			1						1		1
mouse	1	1	1		1	1	1						1							1	1				1
sixpence	2		1				2	1	1						1			1				1			
stile	2			1	1		3		1								1					1			
there	1						1							1										1	1
they	2	1	1			1	1				1		1			1					1				
together	1	1	1				1			1	1	1	1			1				1					
upon	2		1	1			2	1	1								1	1							
walked	2		1				2	1						1	1									1	1
was	1						1							1					1				1		1
who	4		2	1	1	1	4		1					1	1	1			1				1	1	

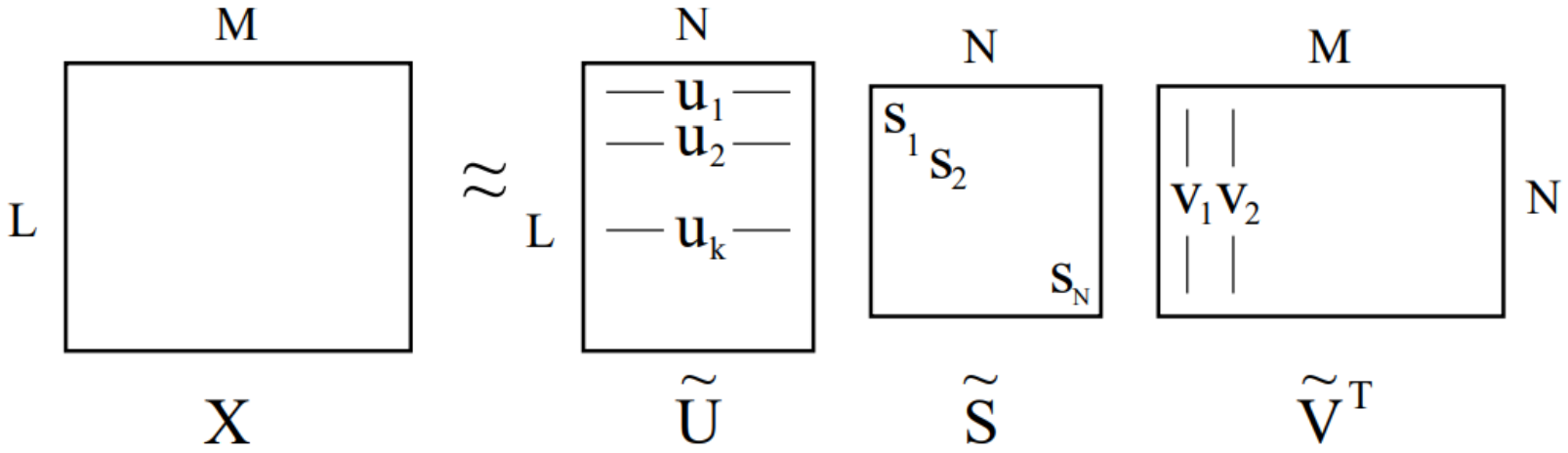
Co-occurrence Matrix

- Each row of this matrix could be considered as a vector representation for the corresponding word

- But the number of dimensions is equal to the size of the vocabulary, which could be very large
  - High-dimensional and sparse
  - Need to conduct dimensionality reduction for NNs

Recall: Singular Value Decomposition (SVD)

Co-occurrence matrix $X_{(L\times M)}$ can be decomposed as $\mathrm{X} = \mathrm{U}\,\mathrm{S}\,\mathrm{V}^{\mathrm{T}}$ where $U_{(L\times L)}$, $V_{(M\times M)}$ are unitary (all columns have unit length) and $S_{(L\times M)}$ is diagonal, with diagonal entries $s_1 \geq s_2 \geq \ldots \geq s_M \geq 0$



We can obtain an approximation for X of rank $N < M$ by truncating U to $\tilde{\mathrm{U}}_{(L\times N)}$, S to $\tilde{\mathrm{S}}_{(N\times N)}$ and V to $\tilde{\mathrm{V}}_{(N\times M)}$. The kth row of $\tilde{\mathrm{U}}$ then provides an N-dimensional vector representing the k^{th} word in the vocabulary.

Word2Vec and GloVe

For language processing tasks, typically, L is the number of words in the vocabulary (about $60,000$) and M is either equal to L or, in the case of document classification, the number of documents in the collection. SVD is computationally expensive, proportional to $L \times M^2$ if $L \geq M$. Can we generate word vectors in a similar way but with less computation, and incrementally?

- Word2Vec
  - predictive model
  - maximize the probability of a word based on surrounding words
- GloVe
  - count-based model
  - reconstruct a close approximation to the co-occurrence matrix X

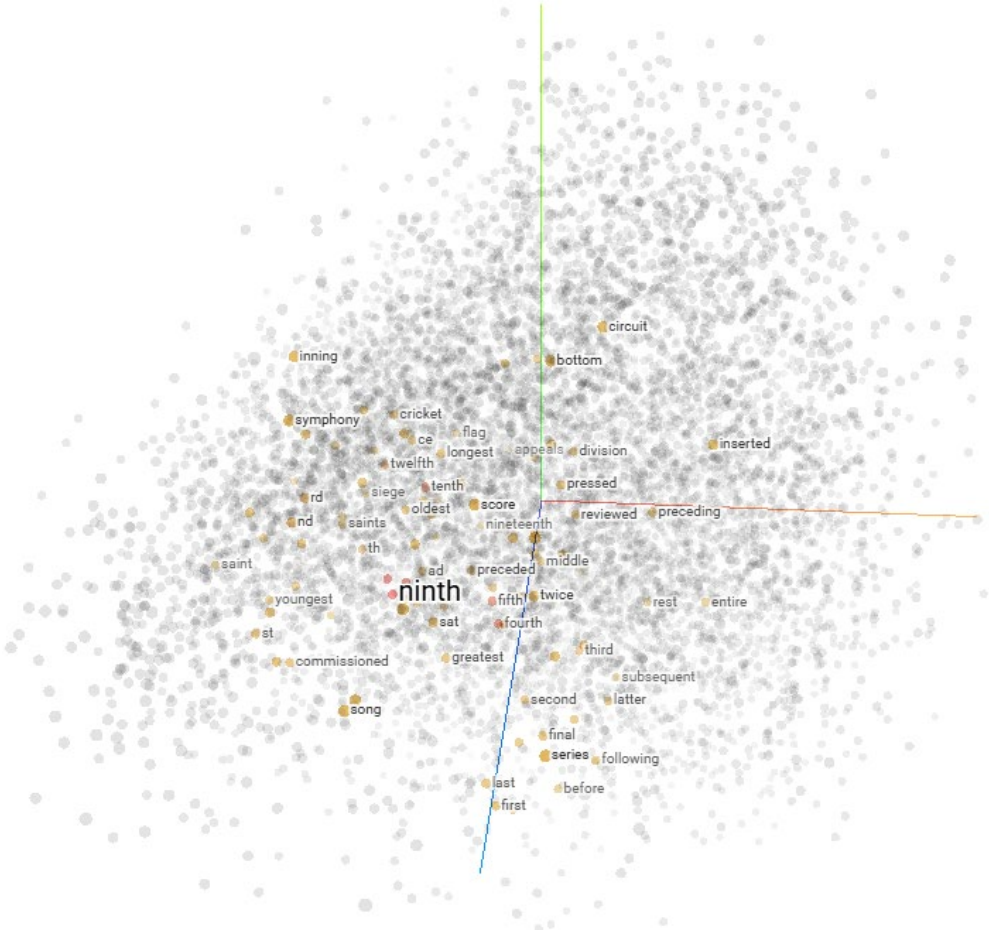
Word Embeddings

"Words that are used and occur in the same contexts tend to purport similar meanings."

Z. Harris (1954)

"You shall know a word by the company it keeps."

J.R. Firth (1957)



https://projector.tensorflow.org/

History of Word Embeddings

- Structuralist Linguistics (Firth, 1957)
- Recurrent Networks (Rumelhart, Hinton & Williams, 1986)
- Latent Semantic Analysis (Deerwester et al., 1990)
- Hyperspace Analogue to Language (Lund, Burgess & Atchley, 1995)
- Neural Probabilistic Language Models (Bengio, 2000)
- NLP (almost) from Scratch (Collobert et al., 2008)
- word2vec (Mikolov et al., 2013)
- GloVe (Pennington, Socher & Manning, 2014)
- BERT (J Devlin et al. 2018) & LLMs

Word Embeddings

- Each word = a vector
- Similar words are "nearby in semantic space“
- Aim of Word Embeddings:
  - *Find a vector representation of each word, such that words with nearby representations are likely to occur in similar contexts.*
- Called an "embedding" because it's embedded into a space
- Every modern NLP algorithm uses embeddings as the representation of word meaning

Intuition: why vectors?

- Using Words:
  - A feature is a word identity
  - E.g., Feature X: The previous word was "terrible"
  - requires exact same word to be in training and test

- Using Embeddings:
  - Feature is a word vector
  - The previous word was vector [35,22,17…]
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to similar but unseen words!!!

Word Embeddings: Word2Vec and GloVe

For language processing tasks, typically, L is the number of words in the vocabulary (about $60,000$) and M is either equal to L or, in the case of document classification, the number of documents in the collection. SVD is computationally expensive, proportional to $L \times M^2$ if $L \geq M$. Can we generate word vectors in a similar way but with less computation, and incrementally?

- Word2Vec
  - predictive model
  - maximize the probability of a word based on surrounding words
- GloVe
  - count-based model
  - reconstruct a close approximation to the co-occurrence matrix X

Word2vec

- Idea: predict rather than count

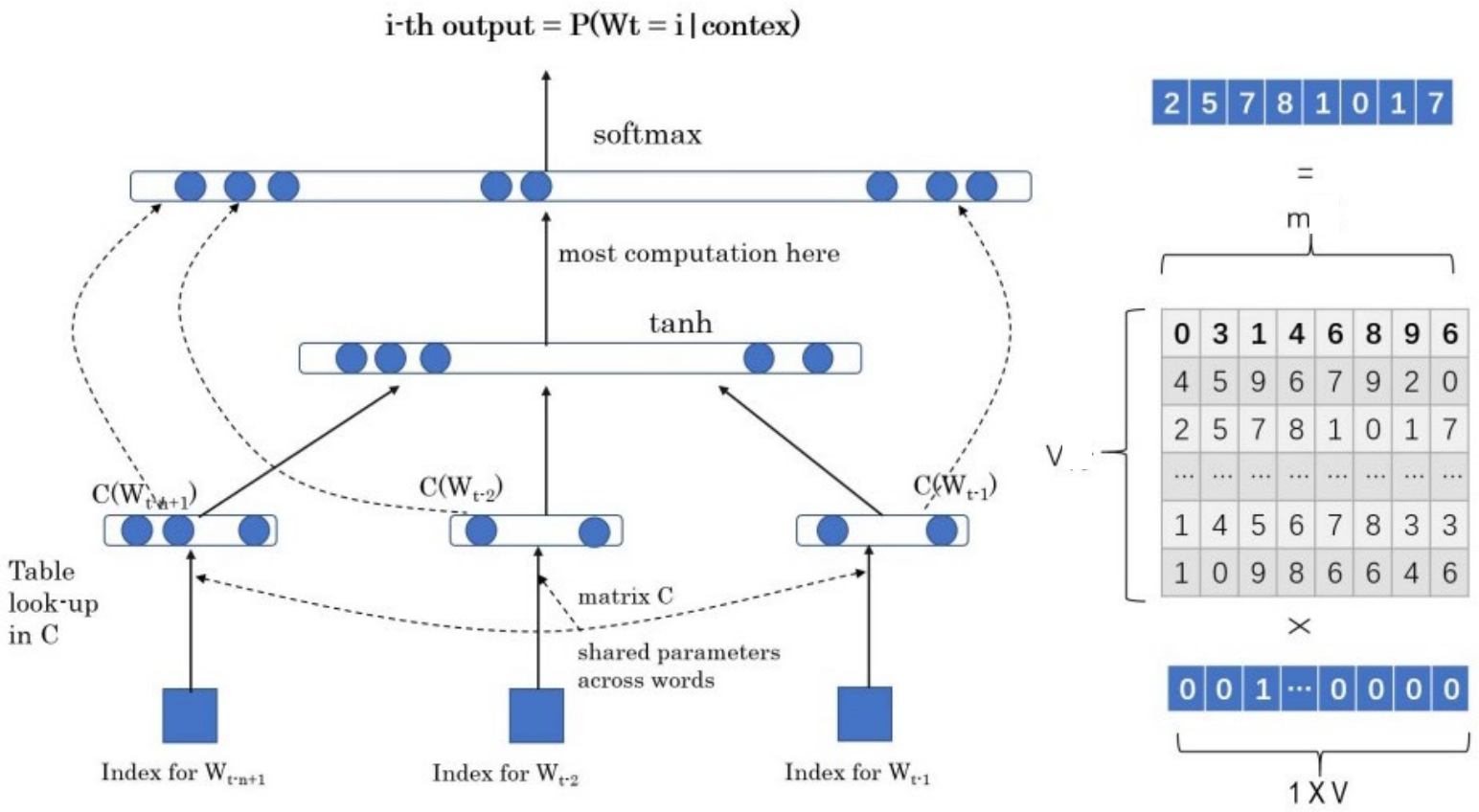
- Instead of counting how often each word w occurs near “university” train a classifier on a binary prediction task:
  - Is w likely to show up near “university”?

- We don’t actually care about this task
  - But we’ll take the learned classifier weights as the word embeddings

- Use running text as implicitly supervised training data

- No need for hand-labeled supervision

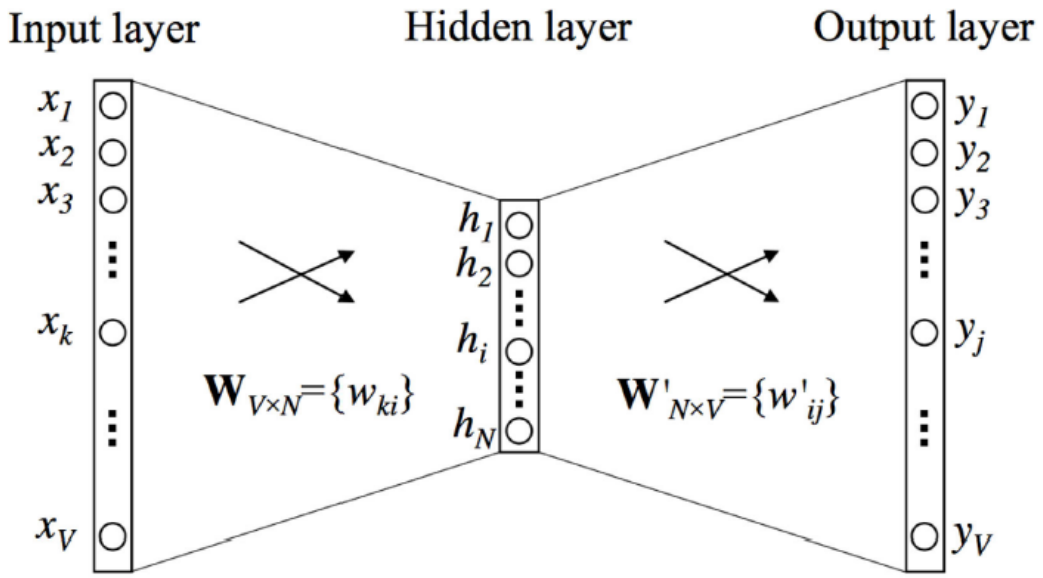
Neural Net Language Model (NNLM)



- Use one-hot embedding
- Matrix C to get the word feature vector
- Predict the next word

Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137-1155, 2003.

Word2Vec 1-word Context Model



The k^{th} row $\mathbf{v}_k$ of $\mathbf{W}$ is a representation of word k.
The j^{th} column $\mathbf{v}'_j$ of $\mathbf{W}'$ is an (alternative) representation of word j.
If the (1-hot) input is k, the linear sum at each output will be $u_j = \mathbf{v}'^{\mathrm{T}}_j \mathbf{v}_k$

Cost Function

Softmax can be used to turn these linear sums u_j into a probability distribution estimating the probability of word j occurring in the context of word k

$$\text{prob}(j|k) = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} = \frac{\exp(\mathbf{v}_j'^{\mathrm{T}} \mathbf{v}_k)}{\sum_{j'=1}^{V} \exp(\mathbf{v}_{j'}'^{\mathrm{T}} \mathbf{v}_k)}$$

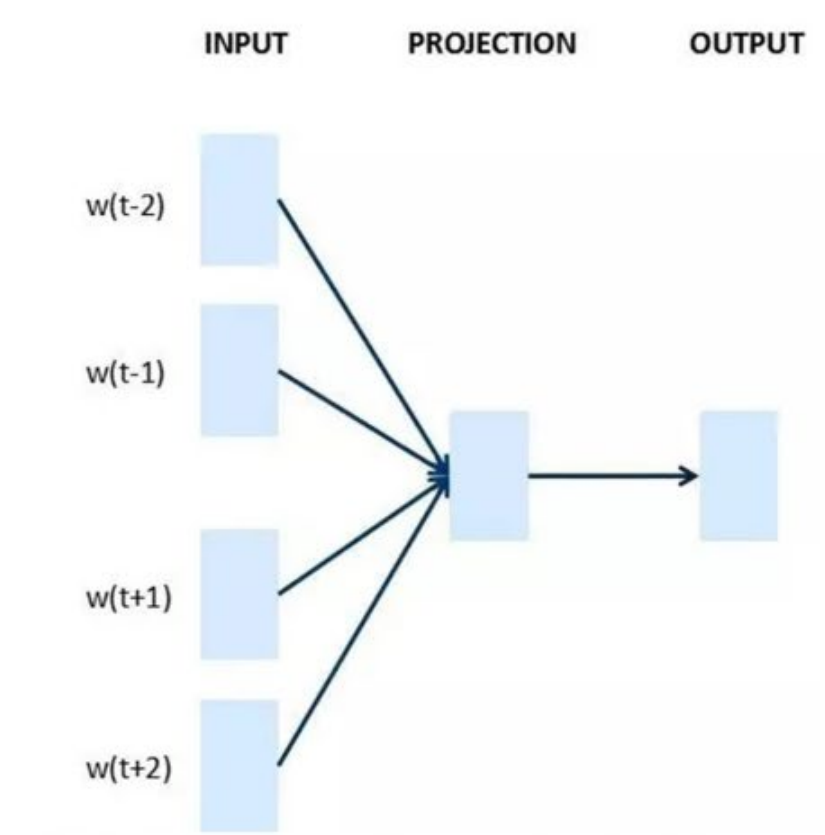
We can treat the text as a sequence of numbers $w_1, w_2, \ldots, w_T$ where $w_i = j$ means that the i^{th} word in the text is the j^{th} word in the vocabulary.

We then seek to maximize the log probability

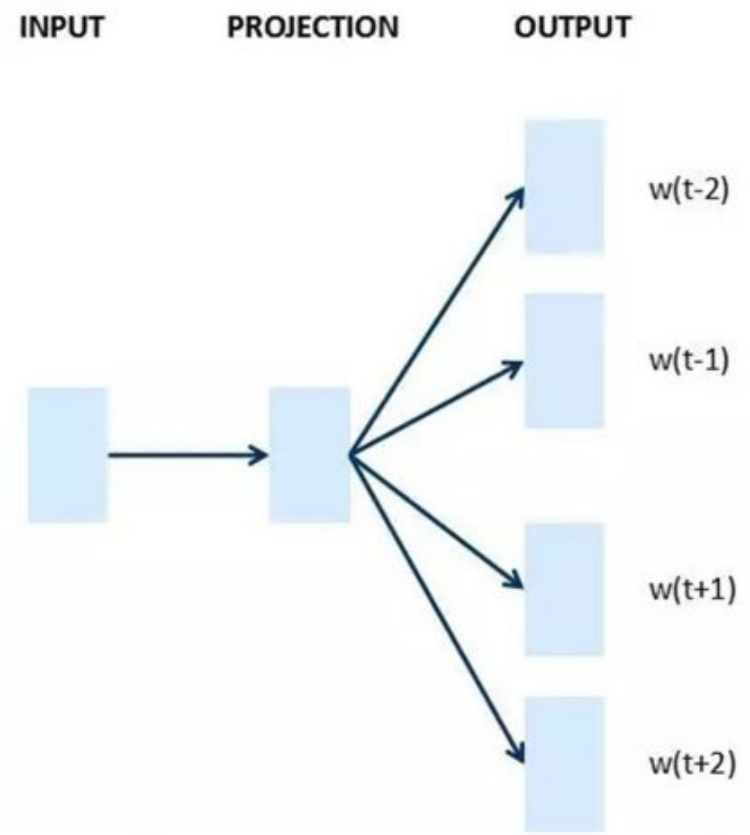
$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le r \le c, r \neq 0} \log \ \text{prob}(w_{t+r}|w_t)$$

where c is the size of training context (which may depend on w_t)

Word2vec



CBOW (Continuous Bag-of-Words Model)



Skip-gram (Continuous Skip-gram Model)

CBOW vs. Skip-gram

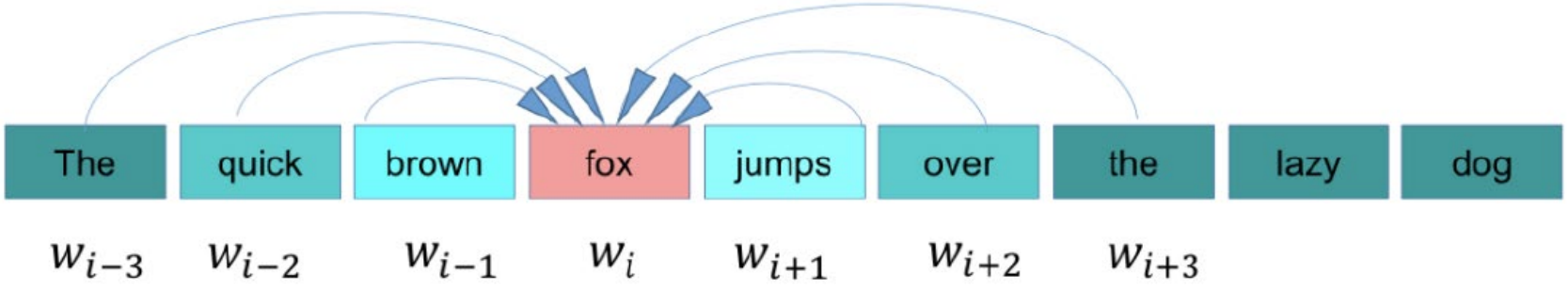


Figure: Continous Bag of Words (CBOW)

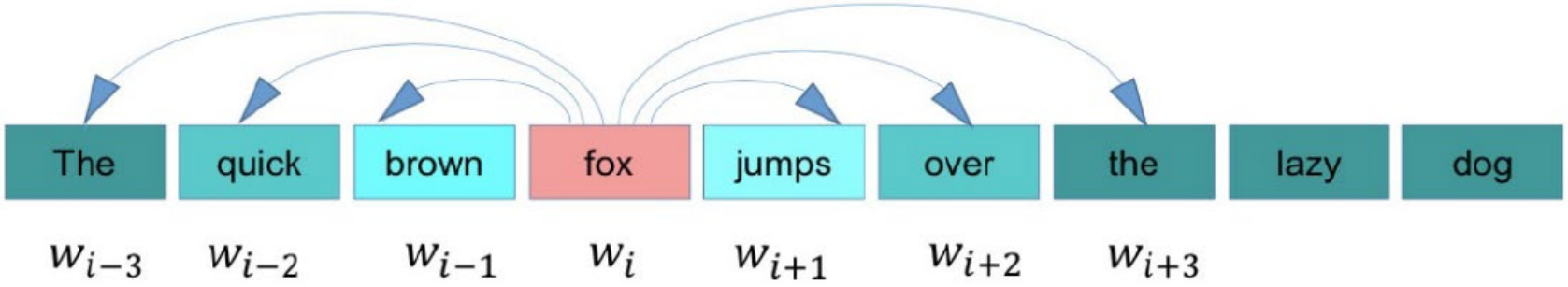
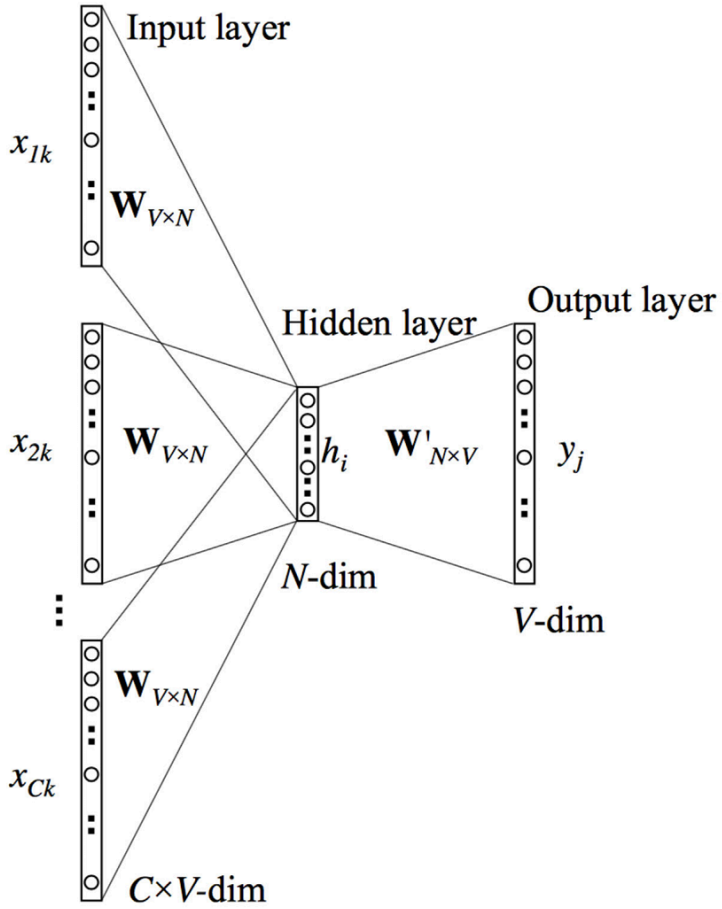


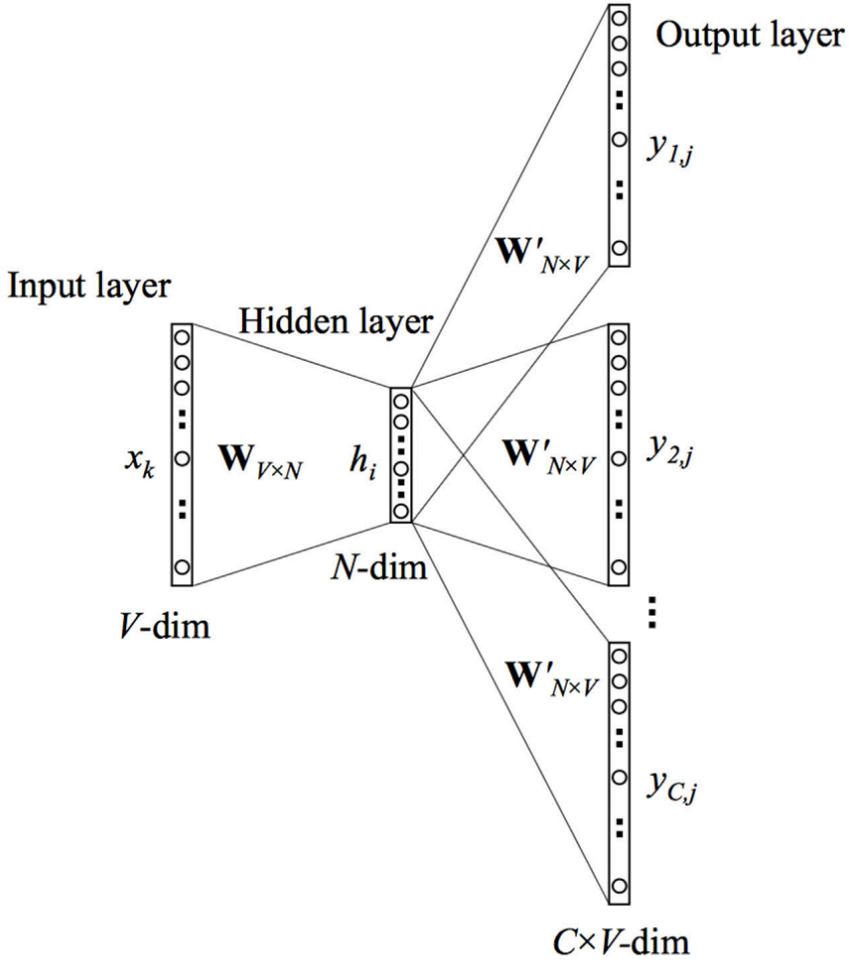
Figure: Skip-Gram model

Continuous Bag of Words (CBOW)



- If several context words are each used independently to predict the center word, the hidden activation becomes a sum (or average) over all the context words
- All context words share the same input-to-hidden weights

Word2vec Skip-Gram Model



- Try to predict the context words, given the center word
- This skip-gram model is similar to CBOW, except that in this case a single input word is used to predict multiple context words
- All context words share the same hidden-to-output weights

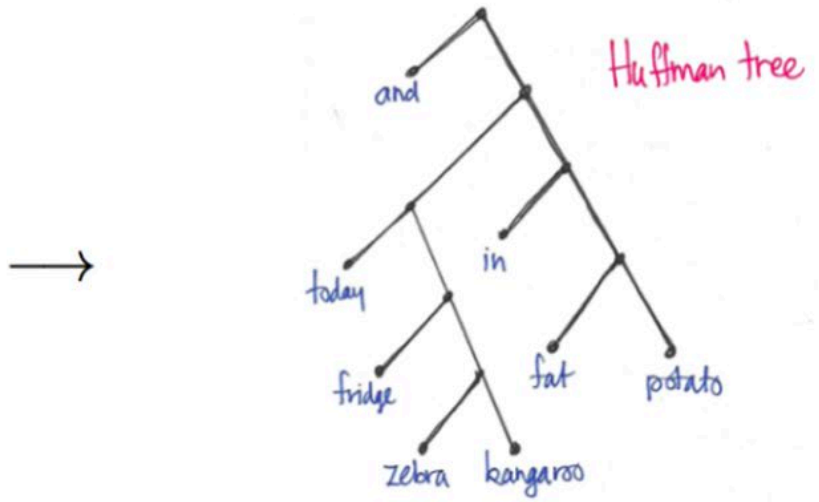
So far

- Word2Vec is a linear model in the sense that there is no activation function at the hidden nodes (the entire training consists of linear projections and dot products between vectors)
- The 1-word prediction model can be extended to multi-word prediction in two different ways:
  - Continuous Bag of Words
  - Skip-Gram
- Need a computationally efficient alternative to Softmax
  - Hierarchical Softmax
  - Negative Sampling
- Need to sample frequent words less often

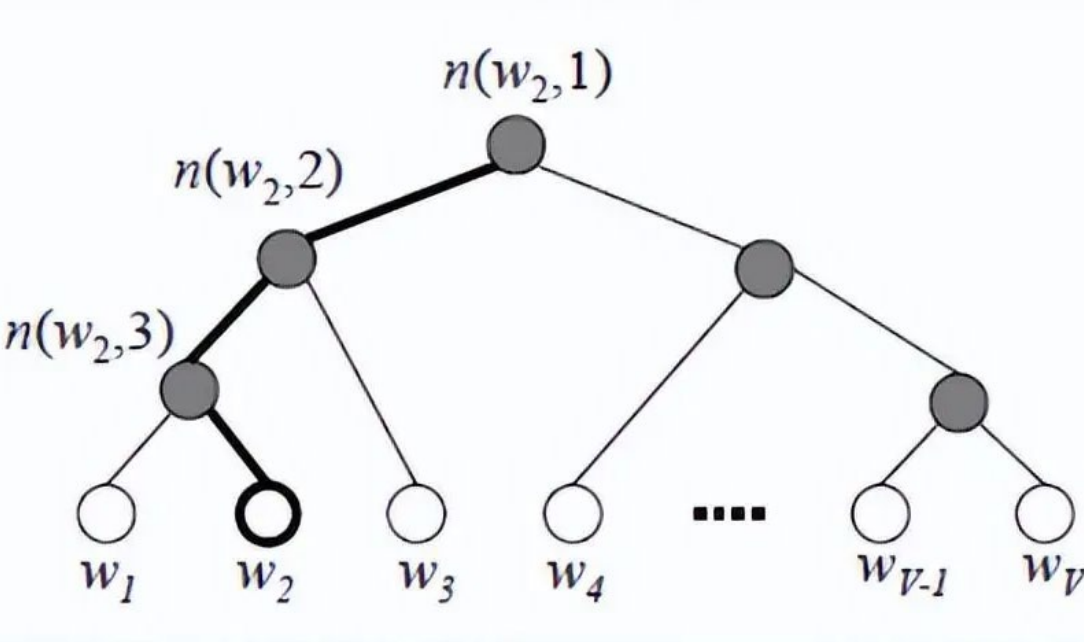
Hierarchical Softmax

- Motivation: Computing softmax over all V vocabulary words is very expensive when V is large
- Target words are organized in a Huffman-coded Binary Tree
  - target words are organized in a Huffman-coded Binary Tree
  - only those nodes that are visited along the path to the target word are evaluated
  - O(V)→O(log V)

word	count
fat	3
fridge	2
zebra	1
potato	3
and	14
in	7
today	4
kangaroo	2



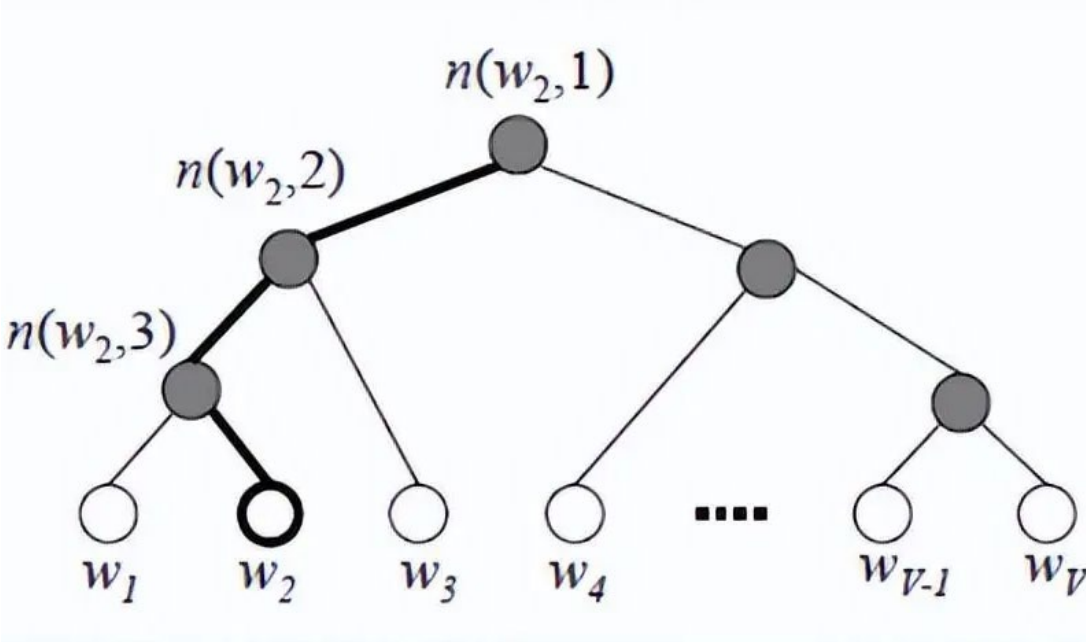
Hierarchical Softmax



- Hierarchical softmax turns a **multi-class classification** problem (over a vocabulary of size V) into a **binary classification** problem at each node in a binary tree.

- The probability of predicting a specific word is decomposed into a product of probabilities of making binary decisions at each internal node along the path from the root to that word's leaf node.

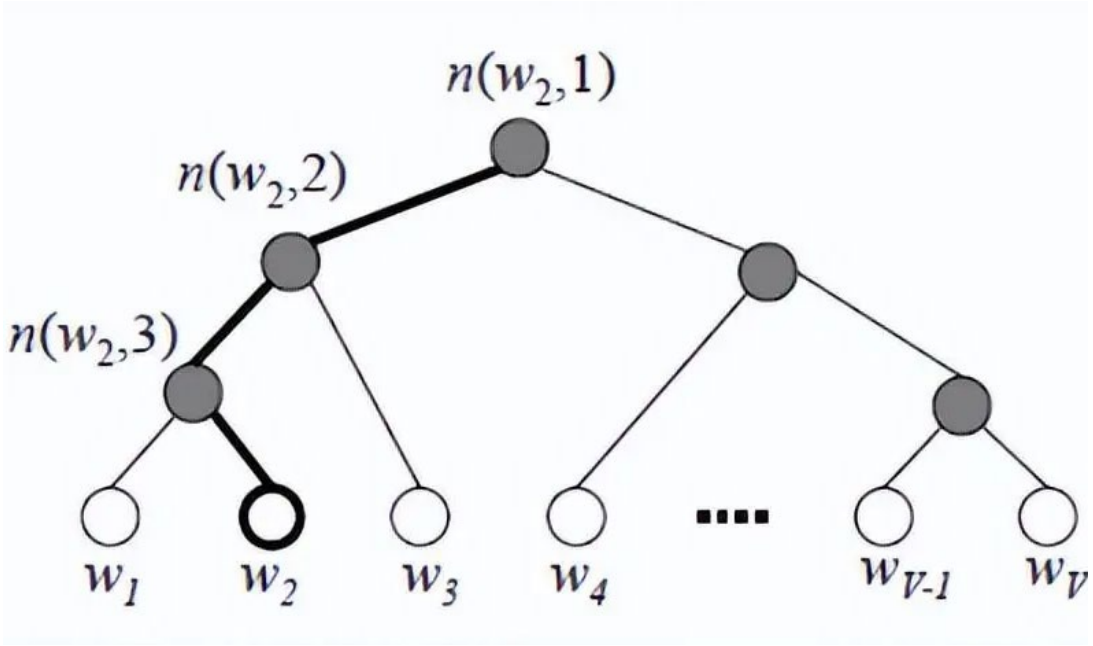
Hierarchical Softmax



- Each word in the vocabulary is a leaf node
- To compute the probability of a word w_j
  - Start from the root and
  - Traverse down to leaf w_j by making a sequence of binary decisions: left or right.
- Each internal node is a binary classifier (sigmoid)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Hierarchical Softmax



$$[n' = \text{child}(n)] = \begin{cases} +1, & \text{if } n' \text{ is left child of node } n, \\ -1, & \text{otherwise.} \end{cases}$$

$$\text{prob}(w = w_t) = \prod_{j=1}^{L(w)-1} \sigma([n(w, j+1) = \text{child}(n(w, j))]\mathbf{v'}_{n(w,j)}{}^{\mathrm{T}}\,\mathbf{h})$$

Negative Sampling

$$\text{prob}(j|k) = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} = \frac{\exp(\mathbf{v}_j'^{\mathrm{T}}\mathbf{v}_k)}{\sum_{j'=1}^{V} \exp(\mathbf{v}_{j'}'^{\mathrm{T}}\mathbf{v}_k)}$$

- The denominator sums over all words in the vocabulary → computationally expensive
- Instead of computing the full softmax, negative sampling reformulates the task as a binary classification problem

Negative Sampling

- Consider for each word *w* binary classifier: if given word *C* is good context for *w*, or not
- The idea of negative sampling is that we train the network to increase its estimation of the target word j^* and reduce its estimate not of all the words in the vocabulary but just a subset of them $\mathcal{W}_{\text{neg}}$, drawn from an appropriate distribution.

$$E = -\log \sigma({\mathbf{v}'_{j*}}^{\mathrm{T}}\mathbf{h}) - \sum_{j \in \mathcal{W}_{\text{neg}}} \log \sigma(-\mathbf{v}'^{\mathrm{T}}_{j}\mathbf{h})$$

- Given a sentence "**I love neural networks and deep learning**"
- Let's assume **networks** as the centre word, the *positive* words (that are in the correct context words for the given centre word).
  Positive words: neural, love, deep learning
- **Negative words**: bus, auto, food, Japan, etc. ...

Negative Sampling

$$E = -\log \sigma({\mathbf{v}'_{j^*}}^\top \mathbf{h}) - \sum_{j \in \mathcal{W}_{\text{neg}}} \log \sigma(-{\mathbf{v}'_j}^\top \mathbf{h})$$

- maximize the probability of the positive term
- minimize the probability of the negative terms

Negative Sampling

- For small datasets, use 5–20 negative samples per positive pair. For large datasets (like Google News corpus), 2–5 is usually enough.
  - More data → less need for many negative samples (the model already sees enough variation).
- Empirically, a good choice of the distribution from which to draw the negative samples is:

$$P(w) = \frac{U(w)^{3/4}}{Z}$$

  - U(w) : unigram distribution, the frequency distribution of words in the corpus
  - Z: normalization constant
  - The power of 3/4 empirically improves performance. It reduces over-sampling of frequent but uninformative words like "the", "is", "and".

Sub-sampling of Frequent Words

In order to diminish the influence of more frequent words, each word in the corpus is discarded with probability

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where $f(w_i)$ is the frequency of word w_i and $t \sim 10^{-5}$ is an empirically determined threshold.

Word2vec Summary

- Go through each word in the whole corpus and
  - predict surrounding words for eac hword (skip-gram)
  - predict centre word, based on surrounding words (CBOW)

- This captures co-occurrence of words one at a time.

Why not capture co-occurrence counts directly?

Summary

- Word vectors, also sometimes called word embeddings or word representations are distributed representations of words.
- Two kinds of embeddings
  - Sparse vectors: Words are represented by a simple function of the counts of nearby words.
  - Dense vectors: Representation is created by training a classifier to distinguish nearby and far-away words
- The contexts in which a word appears tells us a lot about what it means. Distributional similarities use the set of contexts in which words appear to measure their similarity
- Word2Vec and GloVe are two important dense representations of words.