## Neural Word Embeddings

Antoine Bosselut





### Announcements

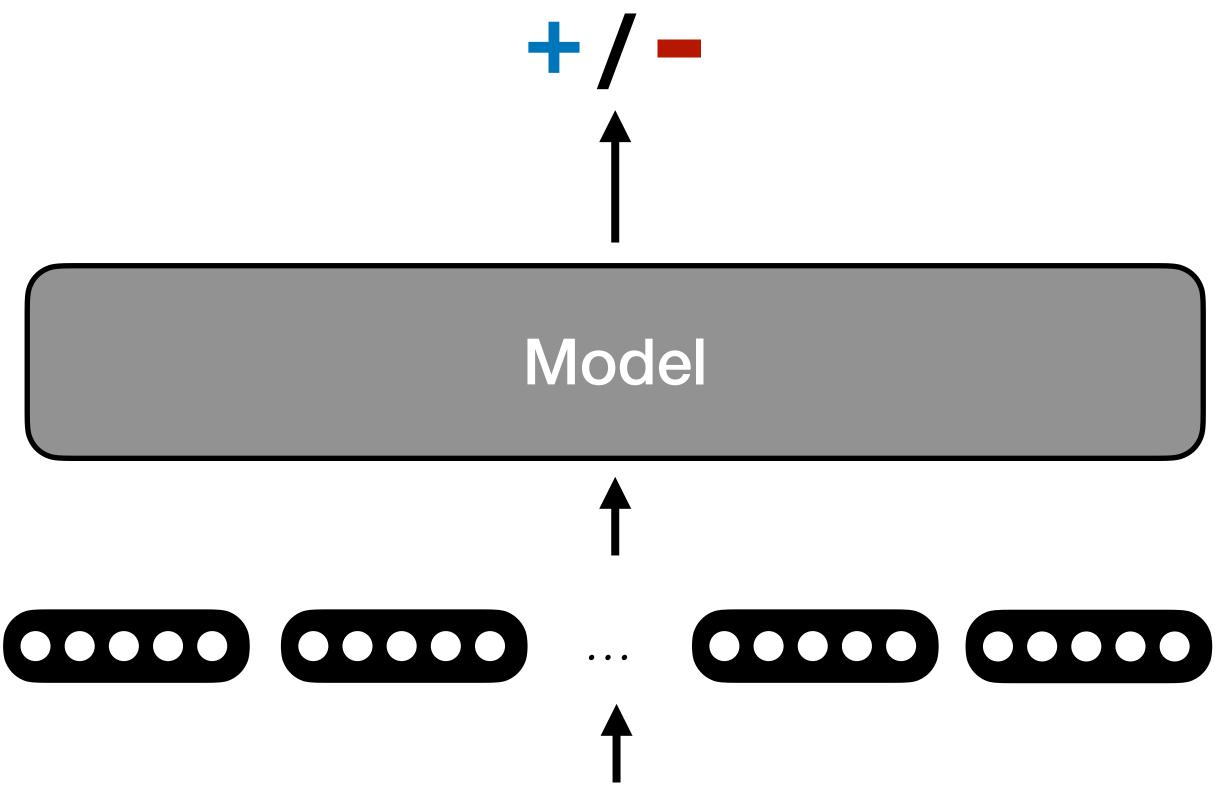
- Lectures are being recorded and MediaSpace channel will be posted to the course website
- Projects should be done in teams of 3
- Yes, you can audit the class, but assignments won't be graded and can't guarantee access to resources

## Today's Outline

- Recap: Words are vectors!
- New: Dense vector representations CBOW, Skipgram, GloVe, fastText

## Word Representations

How do we represent natural language sequences for NLP problems?



In neural natural language processing, words are vectors!

I really enjoyed the movie we watched on Saturday!

## Choosing a vocabulary

- Language contains many words (e.g., ~600,000 in English)
  - What about other tokens: Capitalisation? Accents ? Typos!? Words in other languages!? In other scripts!? Emojis !? Unicode !?
  - Millions of potential unique tokens! Most rarely appear in our training data (Zipfian distribution)
  - Model has limited capacity
- How should we select which tokens we want our model to process?
  - Week 13 tokenisation!
  - For now, initialize a vocabulary V of tokens that we can represent as a vector
  - Any token not in this vocabulary V is mapped to a special <UNK> token (e.g., unknown).

### One upon a time: sparse word representations

- Define a vocabulary V
- Each word in the vocabulary is represented by a sparse vector
- Dimensionality of sparse vector is size of vocabulary (e.g., thousands, possibly millions)

$$x_i \in \{0,1\}^V$$

## Problem

With sparse vectors, similarity is a function of common words!

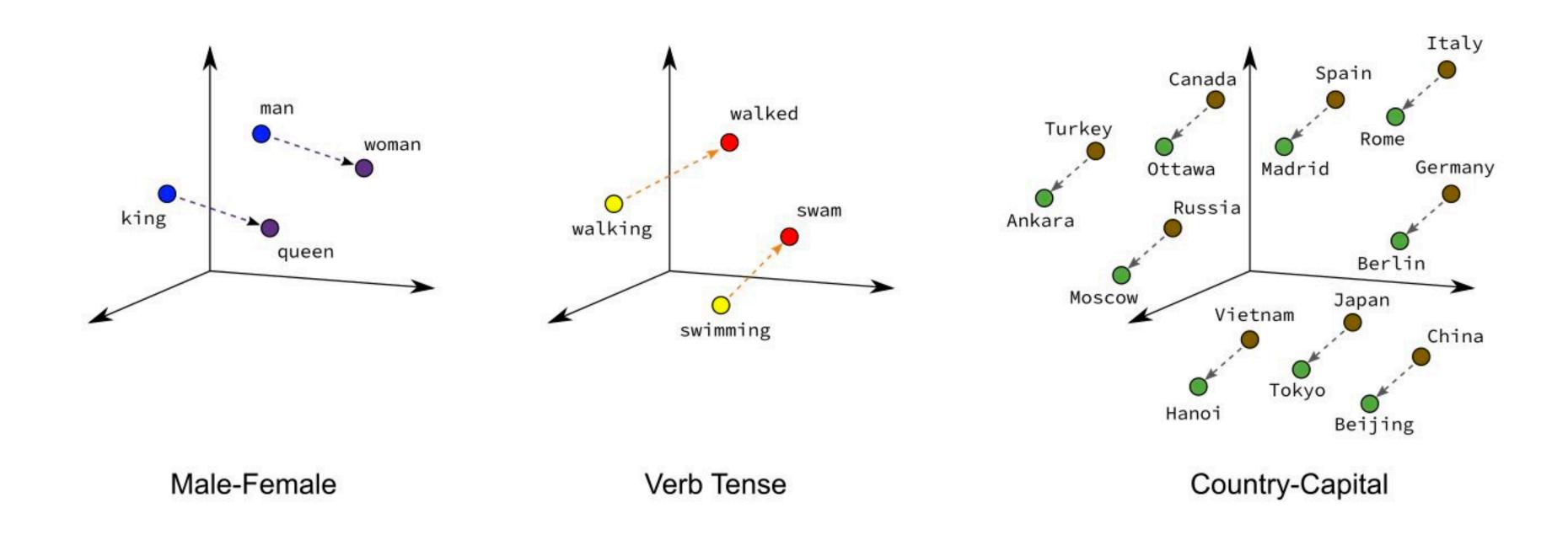
How do you learn learn similarity between words?

```
enjoyed → [0...001...00]

loved → [0...1...0000]
```

sim(enjoyed, loved) = 0

# Embeddings Goal



How do we train semantics-encoding embeddings of words?

## Dense Word Vectors

- Represent each word as a high-dimensional\*, real-valued vector
  - $*Low-dimensional compared to V-dimension sparse representations, but still usually <math>O(10^2 10^3)$

```
| → [0.113 -0.782 1.893 0.984 6.349 ...]
| really → [0.906 0.661 -0.214 -0.894 -0.880 ...]
| enjoyed → [-0.842 0.647 -0.882 0.045 0.029 ...]
| the → [0.100 0.765 -0.333 -0.538 -0.150 ...]
| movie → [0.104 -0.054 -0.268 -0.877 0.005 ...]
| ! → [0.439 -0.577 -0.727 0.261 0.699 ...]
```

word vectors

word embeddings

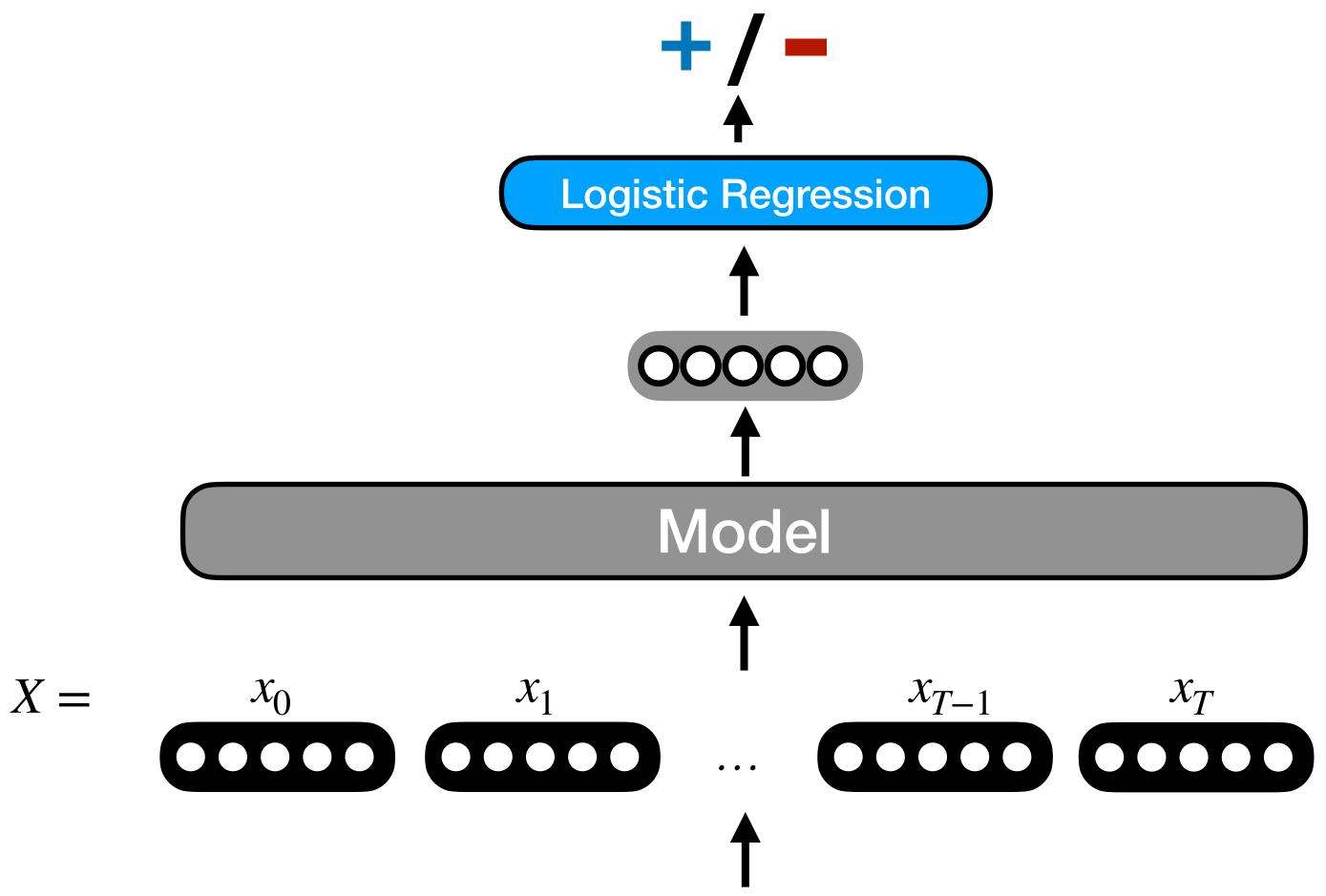
neural embeddings

dense embeddings

others...

• Similarity of vectors represents similarity of meaning for particular words

## Learn embeddings from the task!

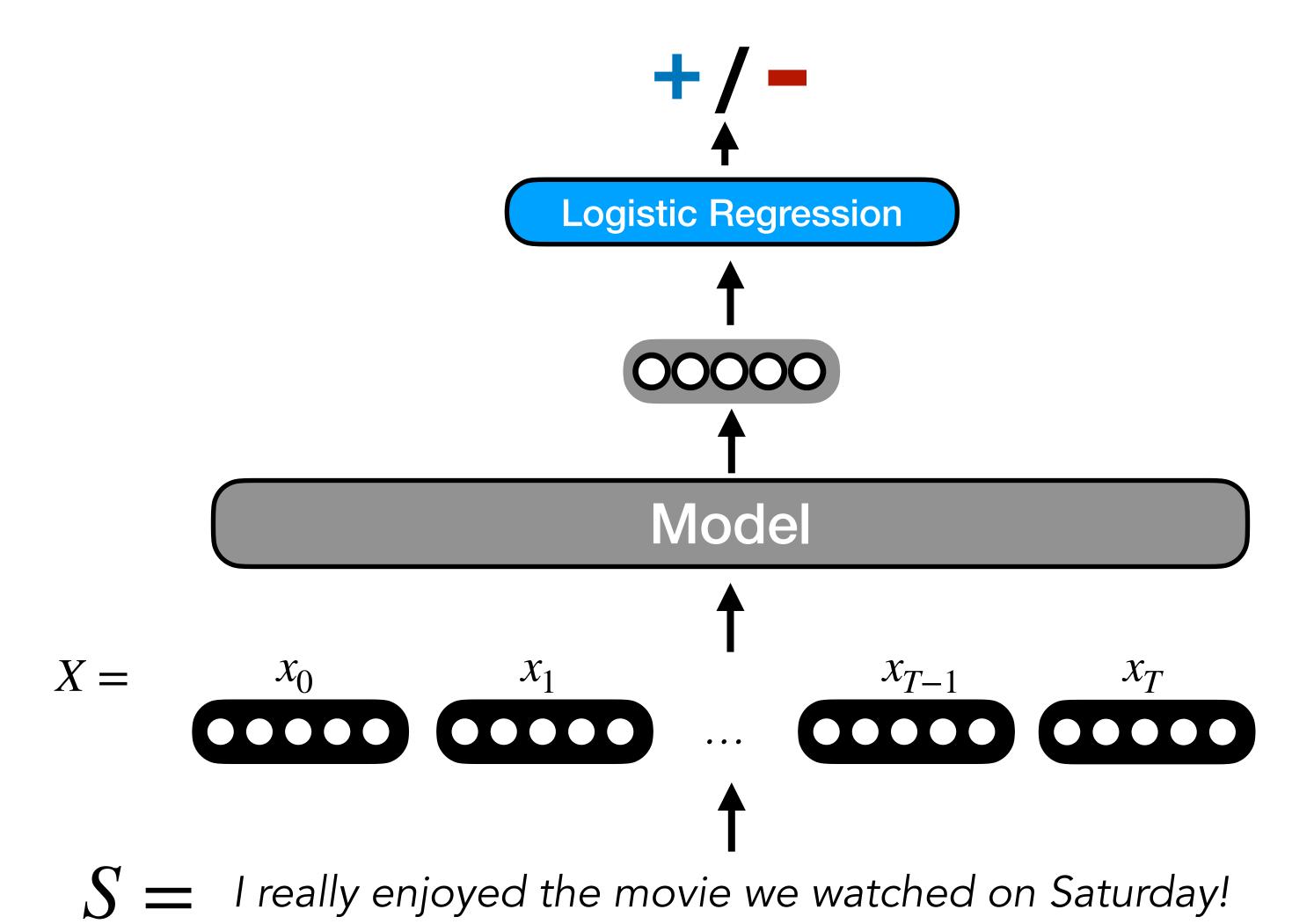


Learn using **backpropagation**: compute gradients of loss with respect to initial embeddings *X* 

Learn embeddings that allow you to do the task successfully!

S = I really enjoyed the movie we watched on Saturday!

## Supervised Learning



- Supervised learning with a task-specific objective
  - Learn word embeddings that help complete the task
- Q: Downsides of learning embeddings this way?
  - Data scarcity (clean labeled data is expensive to collect)
  - Embeddings are optimised for this task maybe not others!

## Question

What could be a better way to learn word embeddings?

## Self-supervised learning

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

-J.R. Firth, 1957

## Context Representations

### Solution:

Rely on the context in which words occur to learn their meaning

Context is the set of words that occur nearby

I really enjoyed the \_\_\_\_ we watched on Saturday!

The \_\_\_ growled at me, making me run away.

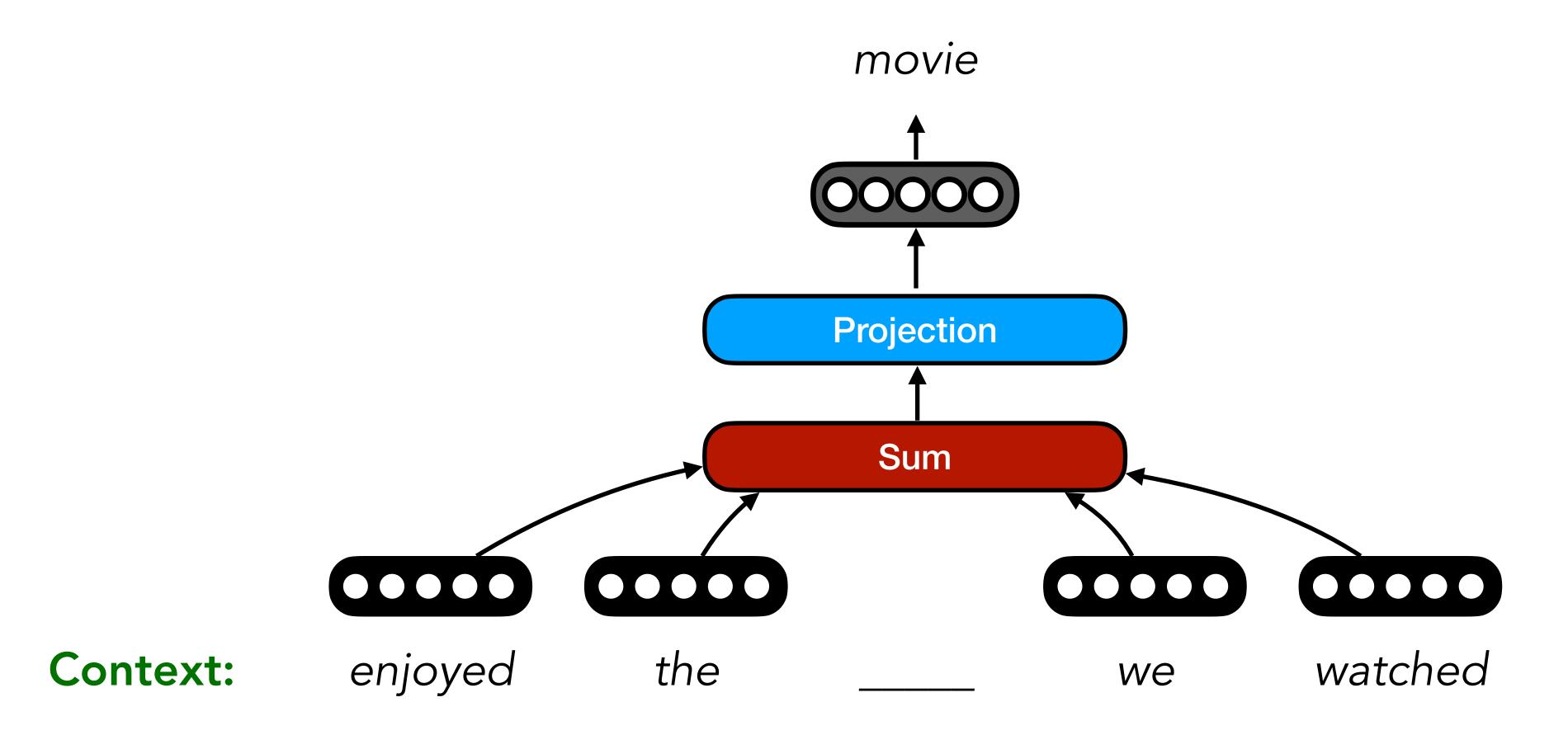
I need to go to the \_\_\_\_ to pick up some dinner.

Foundation of distributional semantics

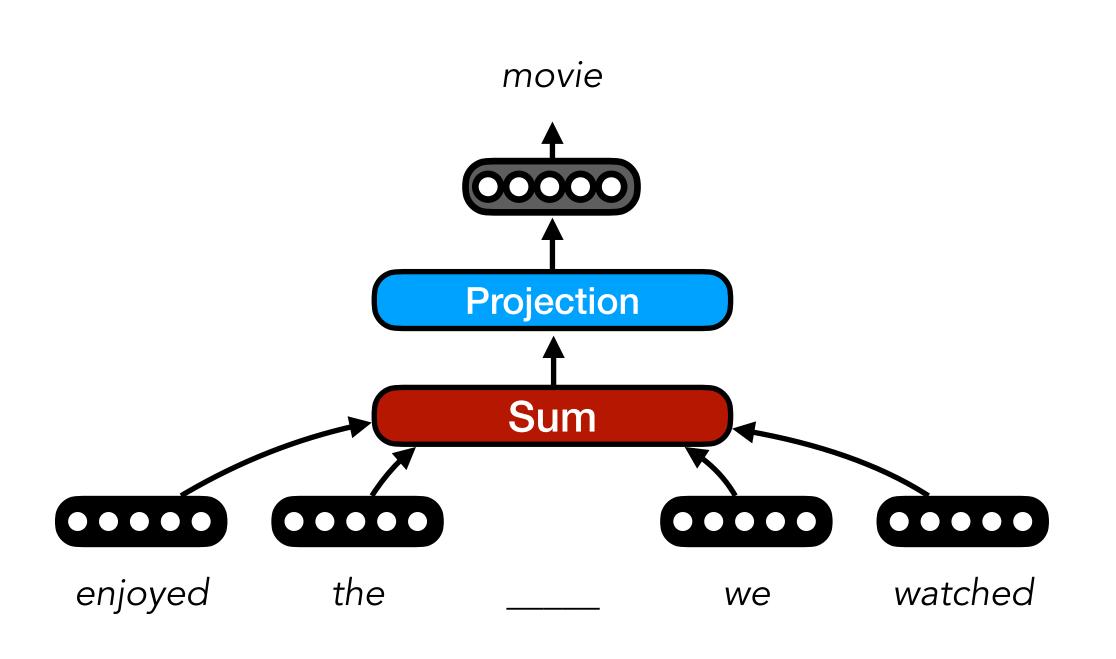
## Learning Word Embeddings

- Many options, huge area of research, but three standard approaches
- Word2vec Continuous Bag of Words (CBOW)
  - Learn to predict missing word from surrounding window of words
- Word2vec Skip-gram
  - Learn to predict surrounding window of words from given word
- GloVe
  - Not covered today

Predict the missing word from a window of surrounding words



Predict the missing word from a window of surrounding words

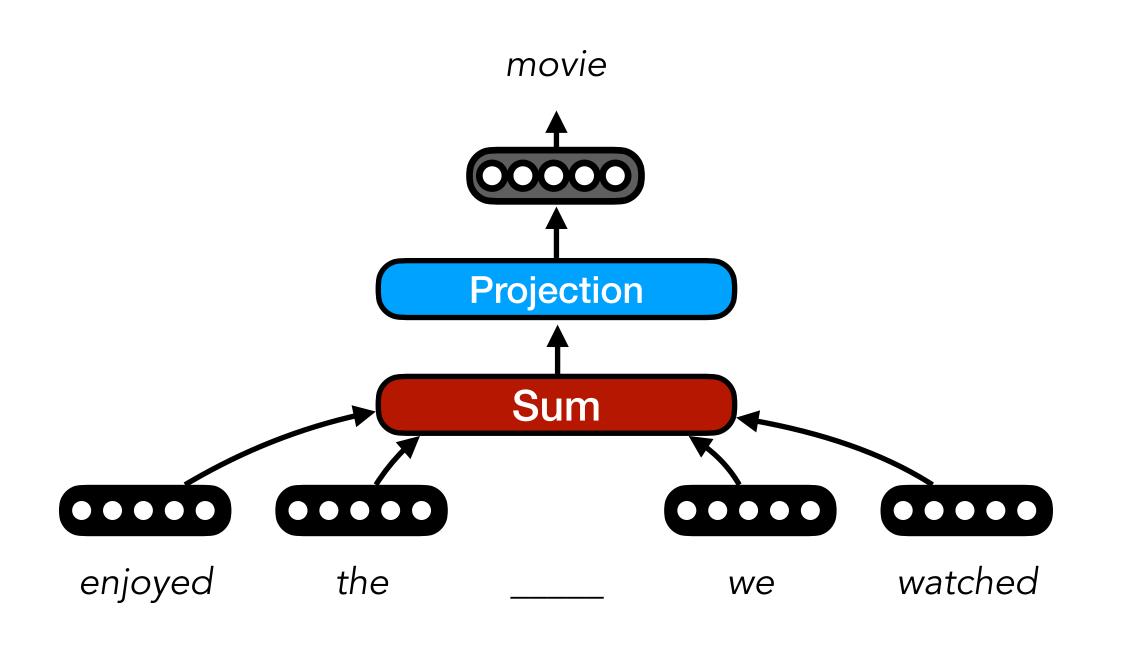


max *P*(*movie* | *enjoyed*, *the*, *we*, *watched*)

$$\max P(x_t | x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2})$$

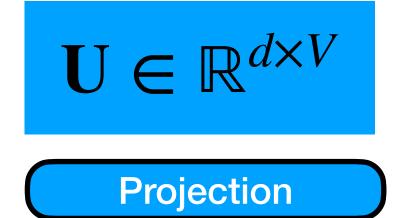
$$\max P(x_t | \{x_s\}_{s=t-2}^{s=t+2})$$

Predict the missing word from a window of surrounding words



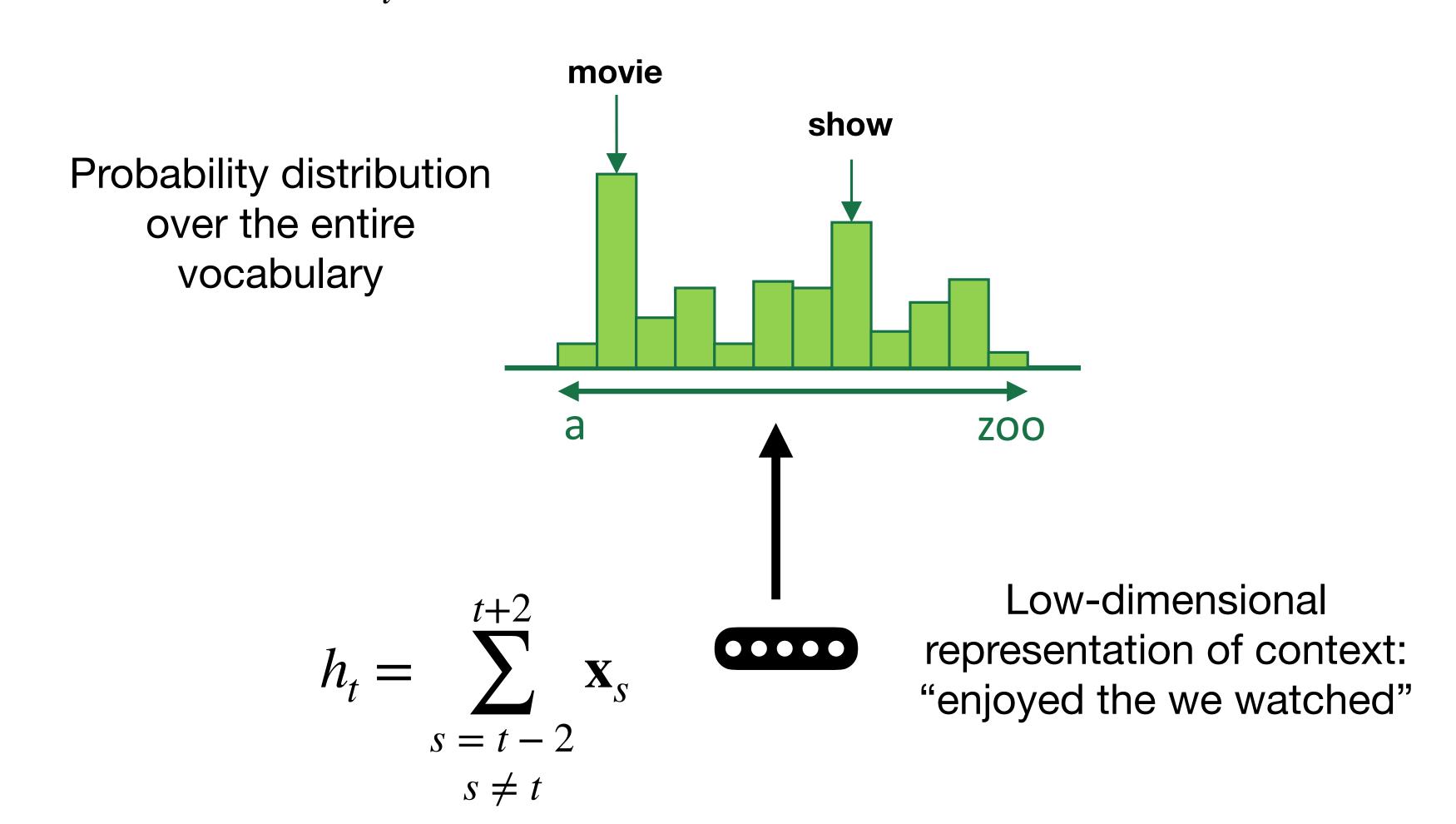
$$P(x_t | \{x_s\}_{s=t-2}^{s=t+2}) = \mathbf{softmax} \left( \mathbf{U} \sum_{s=t-2}^{t+2} \mathbf{x}_s \right)$$

$$\mathbf{x}_s \in \mathbb{R}^{1 \times d}$$



## Vocabulary Space Projection

 $P(w_i | \text{vector for "enjoyed the we watched"})$ 



Let's say our output vocabulary consists of just four words: "movie", "show", "book", and "shelf".

$$h_t = \sum_{S=t-2}^{t+2} \mathbf{x}_S$$

$$s = t-2$$

$$s \neq t$$

Low-dimensional representation of context: "enjoyed the we watched"

Let's say our output vocabulary consists of just four words: "movie", "show", "book", and "shelf".

movie show book shelf <0.6, 0.2, 0.1, 0.1>

We want to get a probability distribution over these four words



Low-dimensional representation of context: "enjoyed the we watched"

Let's say our output vocabulary consists of just four words: "movie", "show", "book", and "shelf".

$$\mathbf{U} = \left\{ \begin{array}{l} 1.2, -0.3, 0.9 \\ 0.2, 0.4, -2.2 \\ 8.9, -1.9, 6.5 \\ 4.5, 2.2, -0.1 \end{array} \right\}$$

first, we'll project our 3-d context representation to 4-d with a matrix-vector product

$$h_t = \langle -2.3, 0.9, 5.4 \rangle$$

Here's an example 3-d prefix vector



## How do we get there?

$$\mathbf{U} = \left\{ \begin{array}{l} 1.2, -0.3, 0.9 \\ 0.2, 0.4, -2.2 \\ 8.9, -1.9, 6.5 \\ 4.5, 2.2, -0.1 \end{array} \right\}$$

$$h_t = \langle -2.3, 0.9, 5.4 \rangle$$

intuition: each dimension of  $h_t$  corresponds to a feature of the context

# How do we get there?

 $h_t = \langle -2.3, 0.9, 5.4 \rangle$ 

intuition: each dimension of  $h_t$ corresponds to a feature of the context

$$\mathbf{U}h_t = \langle 1.8, -11.9, 12.9, -8.9 \rangle$$

How did we compute this? It's just the dot product of each row of  ${\bf U}$  with  $h_t$ 

$$h_t = \langle -2.3, 0.9, 5.4 \rangle$$

$$\mathbf{U}h_t = \langle 1.8, -11.9, 12.9, -8.9 \rangle$$

How did we compute this? It's just the dot product of each row of  ${\bf U}$  with  $h_t$ 

$$\mathbf{U} = \begin{cases} 1.2, -0.3, 0.9 \\ 0.2, 0.4, -2.2 \\ 8.9, -1.9, 6.5 \\ 4.5, 2.2, -0.1 \end{cases}$$

$$h_t = \langle -2.3, 0.9, 5.4 \rangle$$

$$\mathbf{U}h_t = \langle 1.8, -11.9, 12.9, -8.9 \rangle$$

How did we compute this? It's just the dot product of each row of  ${\bf U}$  with  $h_t$ 

$$\mathbf{U} = \begin{cases} 1.2, -0.3, 0.9 \\ 0.2, 0.4, -2.2 \\ 8.9, -1.9, 6.5 \\ 4.5, 2.2, -0.1 \end{cases}$$

$$h_t = \langle -2.3, 0.9, 5.4 \rangle$$

## Softmax

• The **softmax** function generates a probability distribution from the elements of the vector it is given

$$\mathbf{softmax(a)}_{i} = \frac{e^{a_{i}}}{\sum_{j=1}^{|\mathbf{a}|} e^{a_{j}}}$$

- a is a vector
- $a_i$  is dimension i of a
- each dimension *i* of the softmaxed output represents the probability of class *i*

## Softmax

• The **softmax** function generates a probability distribution from the elements of the vector it is given

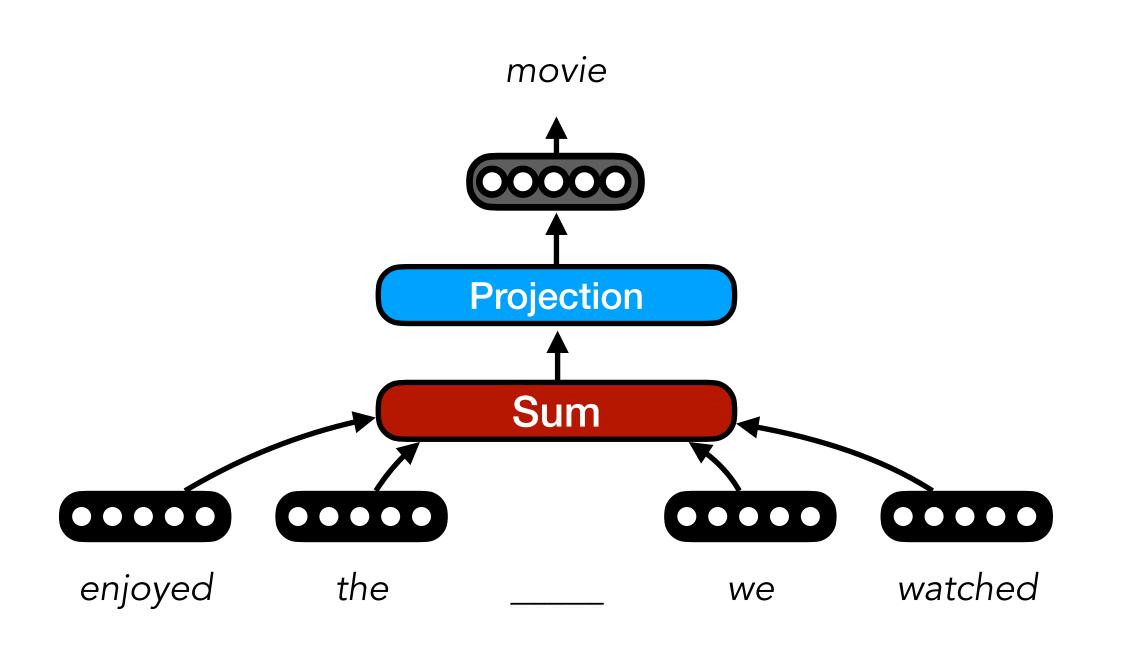
$$\mathbf{softmax(a)}_{i} = \frac{e^{a_{i}}}{\sum_{j=1}^{|\mathbf{a}|} e^{a_{j}}}$$

- a is a vector
- a<sub>i</sub> is dimension *i* of **a**
- each dimension *i* of the softmaxed output represents the probability of class *i*

$$Uh_t = \langle 1.8, -1.9, 2.9, -0.9 \rangle$$
  
**softmax(U** $h_t$ **)** =  $\langle 0.24, 0.006, 0.73, 0.02 \rangle$ 

Softmax will keep popping up, so be sure to understand it!

Predict the missing word from a window of surrounding words



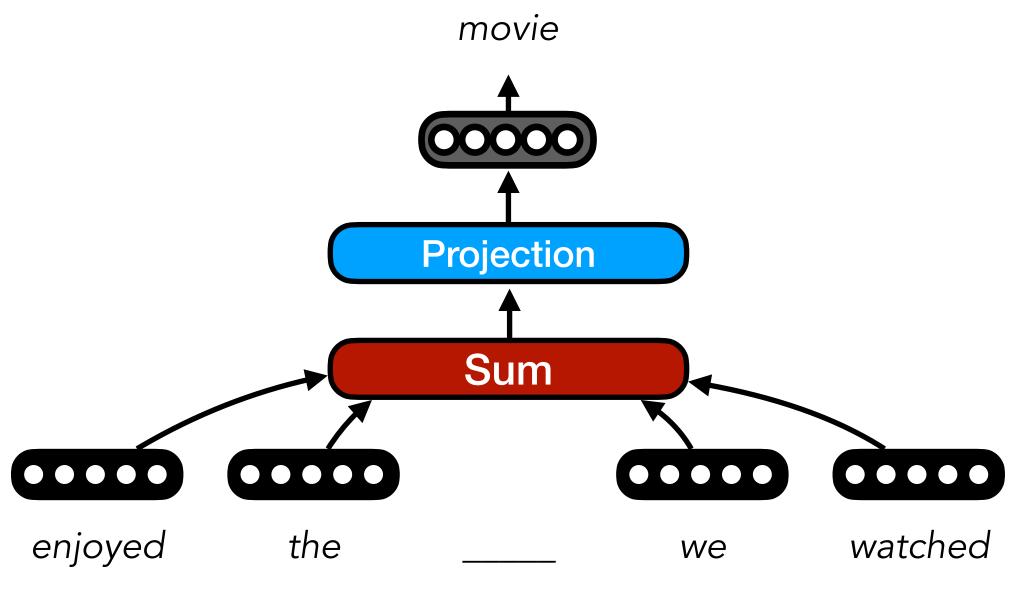
$$P(x_t | \{x_s\}_{s=t-2}^{s=t+2}) = \mathbf{softmax} \left( \mathbf{U} \sum_{s=t-2}^{t+2} \mathbf{x}_s \right)$$

$$\mathbf{x}_s \in \mathbb{R}^{1 \times d}$$



$$P(x_t | \{x_s\}_{s=t-2}^{s=t+2}) = \mathbf{softmax} \left( \mathbf{U} \sum_{s=t-2}^{t+2} \mathbf{x}_s \right)$$

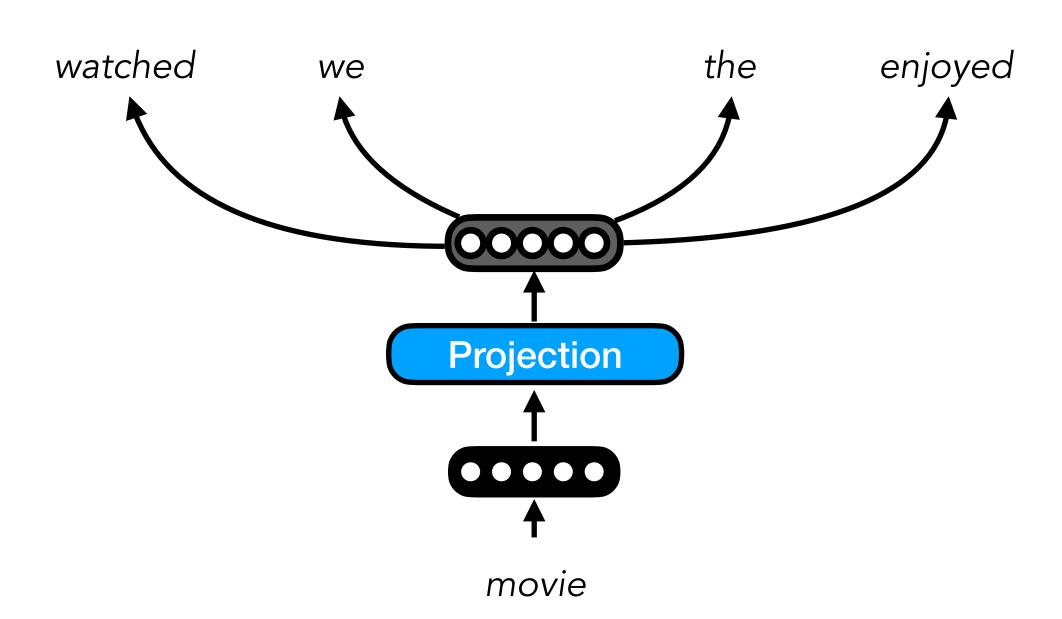
$$s \neq t$$



- Model is trained to maximise the probability of the missing word
  - For computational reasons, the model is typically trained to minimise the negative log probability of the missing word
- Here, we use a window of N=2, but the window size is a hyperparameter
- For computational reasons, a
   hierarchical softmax used to
   compute distribution (Eisenstein, 14.5.3)

• We can also learn embeddings by predicting the surrounding context from a single word

#### **Context:**

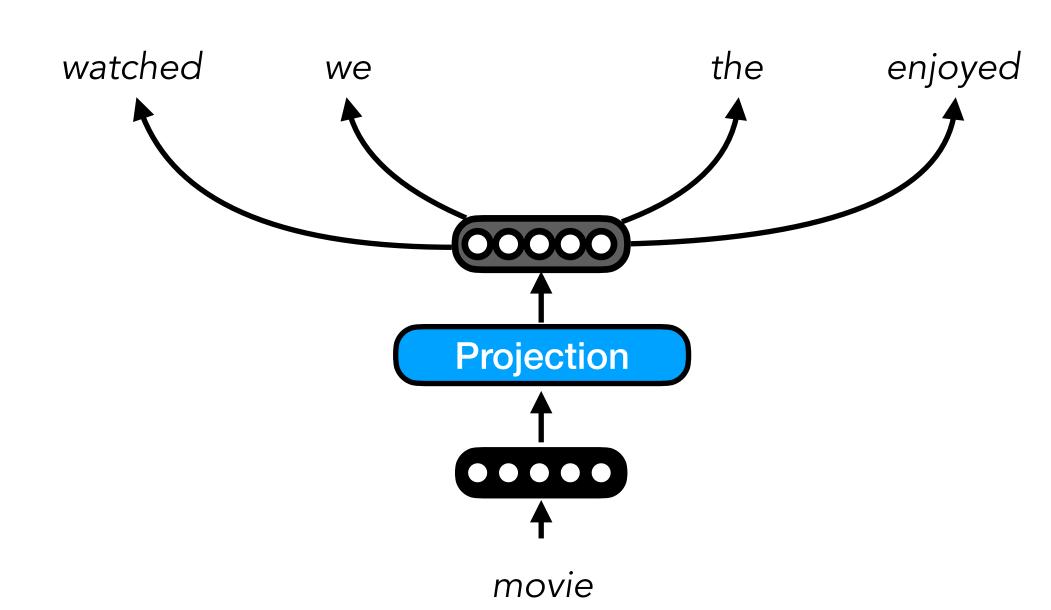


max *P*(*enjoyed*, *the*, *we*, *watched* | *movie*)

$$\max P(x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2} | x_t)$$

• We can also learn embeddings by predicting the surrounding context from a single word

#### **Context:**



max *P*(*enjoyed*, *the*, *we*, *watched* | *movie*)

$$= \max P(x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2} | x_t)$$

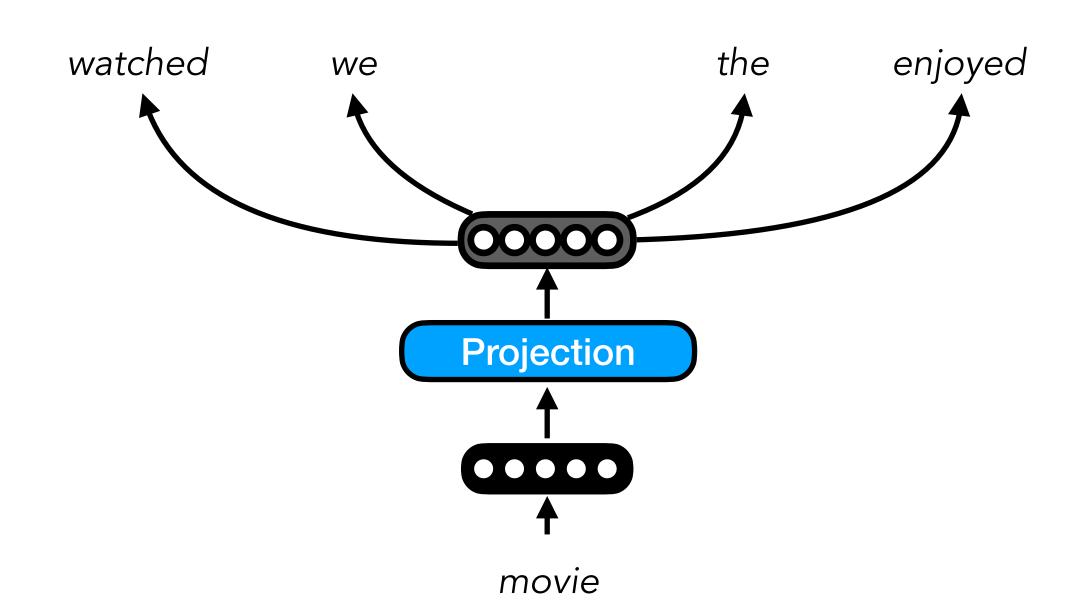
$$= \max \log P(x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2} | x_t)$$

$$= \max \left( \log P(x_{t-2} | x_t) + \log P(x_{t-1} | x_t) \right)$$

$$+\log P(x_{t+1} | x_t) + \log P(x_{t+2} | x_t)$$

• We can also learn embeddings by predicting the surrounding context from a single word

#### **Context:**



$$P(x_s | x_t) = \mathbf{softmax}(\mathbf{U}\mathbf{x}_t)$$

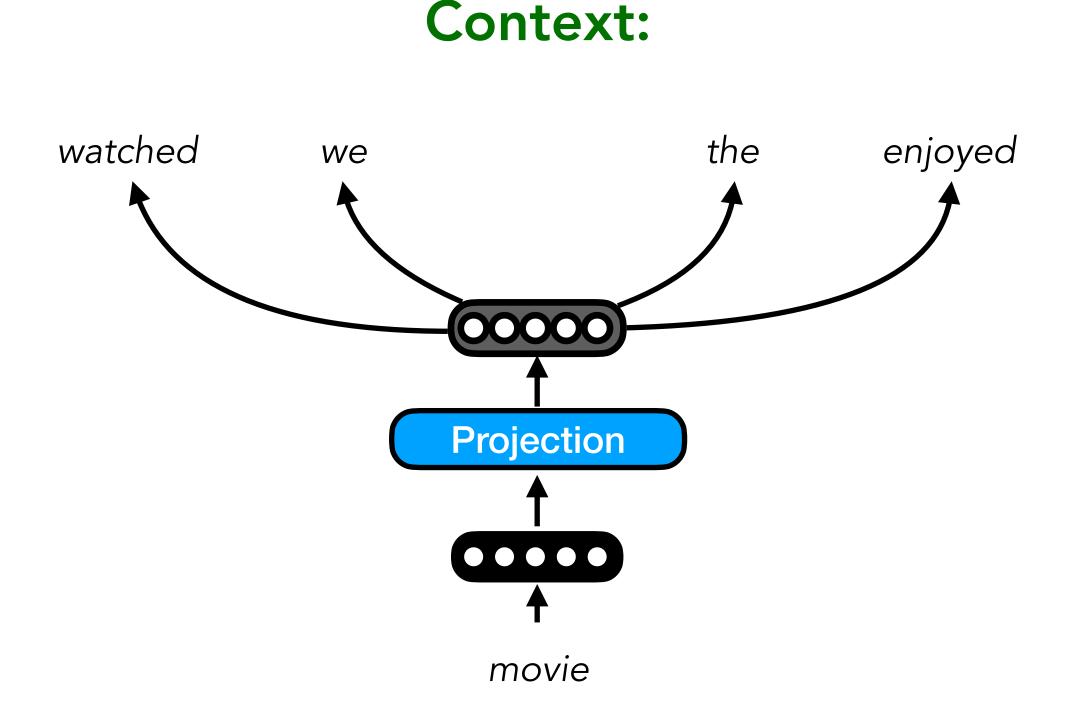
$$\mathbf{x}_t \in \mathbb{R}^{1 \times d}$$







We can also learn embeddings by predicting the surrounding context from a single word



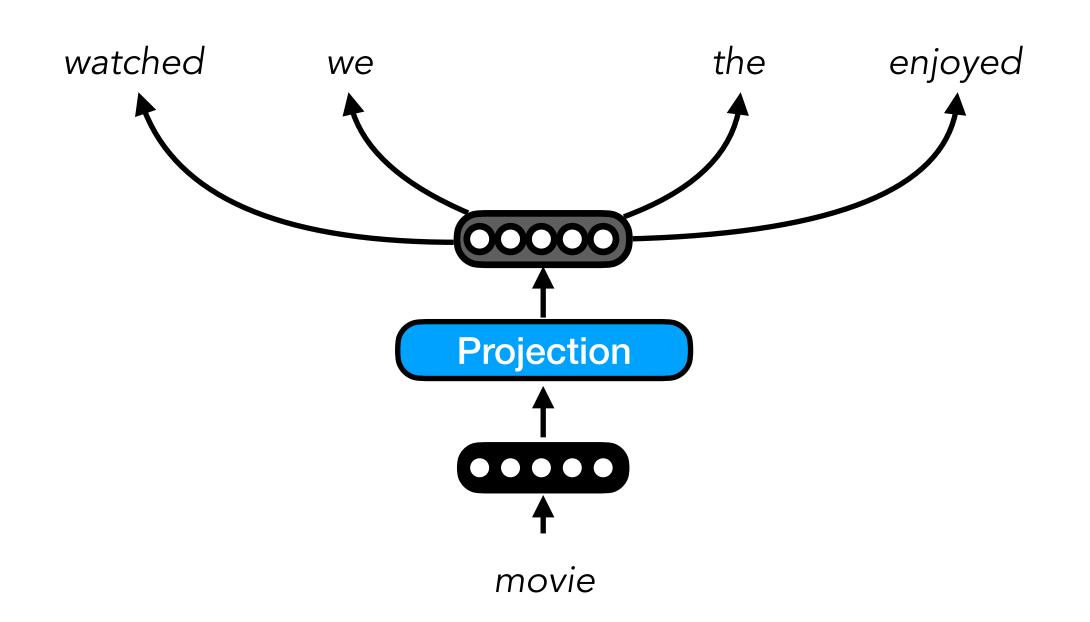
- Model is trained to minimise the negative log probability of the surrounding words
- Here, we use a window of N=2, but the window size is a hyperparameter.
  - Larger window = more information about related words in embedding
- Typically, set large window (N=10), but randomly select  $i \in [1,N]$  as dynamic window size so that closer words contribute more to learning

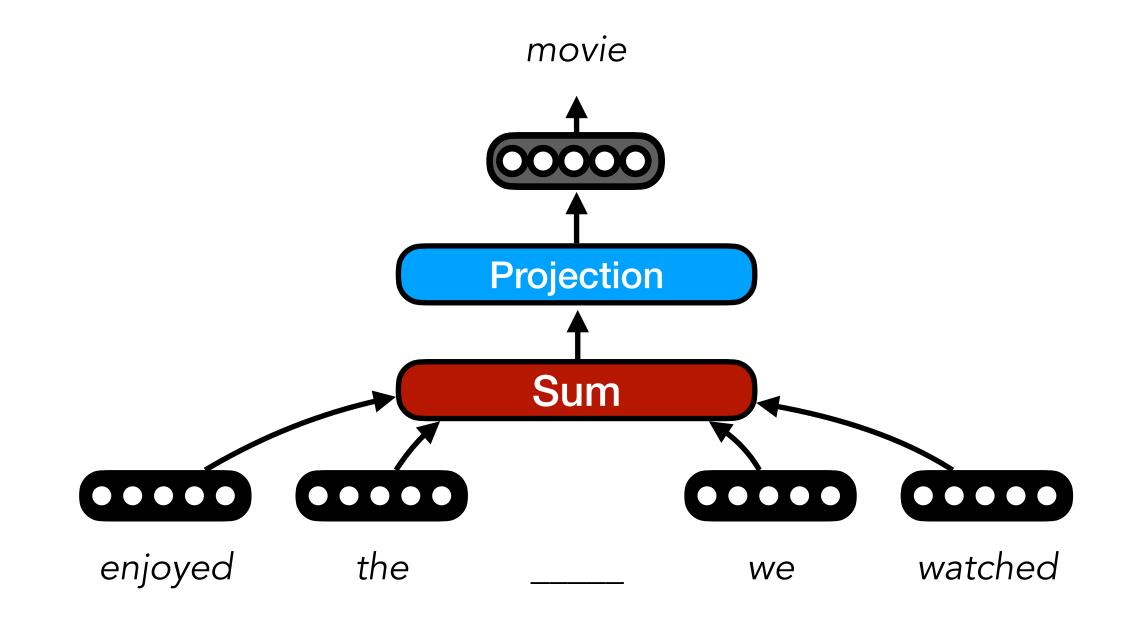
### Question

What is the major conceptual difference between the CBOW and Skipgram methods for training word embeddings?

# Skip-gram vs. CBOW

• **Question:** Do you expect a difference between what is learned by CBOW and Skipgram methods?





## Example

### **CBOW**

```
top_cbow = cbow.wv.most_similar('cut', topn=10)
print(tabulate(top_cbow, headers=["Word", "Simil
              Similarity
Word
slice
                0.662173
                0.650036
crosswise
                0.630569
score
                0.618827
tear
dice
                0.563946
lengthwise
                0.557231
cutting
                0.557228
break
                0.551517
                0.541566
chop
                0.537967
carve
```

### Skip-gram

```
top_sg = skipgram.wv.most_similar('cut', topn=10)
print(tabulate(top_sg, headers=["Word", "Similarity
                  Similarity
Word
                    0.72921
crosswise
                    0.702693
score
slice
                    0.696898
                    0.680091
crossways
1/2-inch-thick
                    0.678496
diamonds
                    0.671814
diagonally
                    0.670319
lengthwise
                    0.665378
cutting
                    0.66425
wise
                    0.656825
```

## Other Resources of Interest

- GloVe Embeddings (Pennington et al., 2014)
  - Use co-occurrence statistics to speed up training of skip-gram-like embeddings
  - Word pairs are training examples, rather than windows in a textual training corpus
- FastText Embeddings (Bojanowski et al., 2017; Mikolov et al., 2018)
  - Enhancement of Skip-gram model that handles morphology
  - Divide words into character n-grams of size n <where> = <wh, whe, her, ere, re>
- Retrofitting word vectors to semantic lexicons (Faruqui et al., 2014)
  - Training word vectors to encode relationships (e.g., synonymy) from high-level semantic resources: WordNet, PPDB, and FrameNet
- <u>S:</u> (n) **sofa**, <u>couch</u>, <u>lounge</u> (an upholstered seat for more than one person)
  - direct hyponym | full hyponym
  - direct hypernym / inherited hypernym / sister term
     S: (n) seat (furniture that is designed for sitting on)
  - derivationally related form

## Recap

- **Problem:** Learning word embeddings from scratch using labeled for a task is data-inefficient!
- **Solution:** Word embeddings can be learned in a self-supervised manner from large quantities of raw text
- Three main algorithms: Continuous Bag of Words (CBOW), Skip-gram, and GloVe

## Resources

- word2vec: https://code.google.com/archive/p/word2vec/
- GloVe: https://nlp.stanford.edu/projects/glove/
- FastText: https://fasttext.cc/
- Gensim: <a href="https://radimrehurek.com/gensim/">https://radimrehurek.com/gensim/</a>

#### Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the <u>Public Domain Dedication and License</u> v1.0 whose full text can be found at: <a href="http://www.opendatacommons.org/licenses/pddl/1.0/">http://www.opendatacommons.org/licenses/pddl/1.0/</a>.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby <u>script</u> for preprocessing Twitter data

## References

- Firth, J.R. (1957). A Synopsis of Linguistic Theory, 1930-1955.
- Mikolov, T., Chen, K., Corrado, G.S., & Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *International Conference on Learning Representations*.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., & Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. *ArXiv*, abs/1310.4546.
- Pennington, J., Socher, R., & Manning, C.D. (2014). GloVe: Global Vectors for Word Representation. Conference on Empirical Methods in Natural Language Processing.
- Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics.*
- Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2018). Advances in pre-training distributed word representations. International Conference on Language Resources and Evaluation.