

CS769 Advanced NLP

Language Modeling

Junjie Hu



Slides adapted from Graham

<https://junjiehu.github.io/cs769-fall23/>

Goals for Today

- Problem definition
- **N-gram Language Model**
- **Log-Linear Language Model**
- **Neural Language Model**
- Evaluation

Are These Sentences OK?

- Jane went to the store.
- store to Jane went the.
- Jane went store.
- Jane goed to the store.
- The store went to Jane.
- The food truck went to Jane.

Engineering Solutions

- Jane went to the store.
 - store to Jane went the.
 - Jane went store.
 - Jane goed to the store.
 - The store went to Jane.
 - The food truck went to Jane.
- } Create a grammar of the language
- } Consider morphology and exceptions
- } Semantic categories, preferences
- } And their exceptions

Quick Review of Probability

- Event space (e.g., \mathcal{X}, \mathcal{Y})—in this class, usually discrete
- Random variables (e.g., X, Y)
- Typical statement: “random variable X takes value $x \in \mathcal{X}$ with probability $P(X = x)$, or in shorthand, $P(x)$ ”
- Joint probability: $P(X = x, Y = y)$
- Conditional probability:
$$P(X = x | Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)}$$
- Bayes rule: $P(X, Y) = P(X | Y)P(Y) = P(Y | X)P(X)$
- Independent variables X, Y : $P(X, Y) = P(X)P(Y)$
- The difference between *true* and *estimated* probability distributions

Notation and Definitions

- \mathcal{V} is a finite set of (discrete) symbols (e.g., words or characters);
 $V = |\mathcal{V}|$
- \mathcal{V}^* is the (infinite) set of sequences of symbols from \mathcal{V}
- In language modeling, we imagine a sequence of random variables $X = \langle x_1, x_2, \dots, x_n \rangle$ that continues until $x_n = \text{“[EOS]”}$
- \mathcal{V}^+ is the (infinite) set of sequences of \mathcal{V} symbols, with the last token $x_n = \text{“[EOS]”}$
- LM problem: Estimate the probability of a sequence $P(X)$, $X \in \mathcal{V}^+$

Language Modeling Problem

- Input: training data a sequence $X = \langle x_1, x_2, \dots, x_n \rangle \in \mathcal{V}^+$
 - Sometimes it's useful to consider a collection of training sentences, each in \mathcal{V}^+ , but it complicates notation.
- Output: $P : \mathcal{V}^+ \rightarrow \mathbb{R}$

$$P(X) = \prod_{i=1}^I P(x_i \mid x_1, \dots, x_{i-1})$$

Next Word Context

The big problem: How do we predict

$$P(x_i \mid x_1, \dots, x_{i-1})$$

?!?

Shannon's Language Model

(Shannon 1950)

- First seriously formulated study on LM
- Information theory: how much information does English as a language convey?



One method of calculating the entropy H is by a series of approximations F_1, F_2, \dots , which successively take more and more of the statistics of the language into account and approach H as a limit. F_N may be called the N -gram entropy; it measures the amount of information or entropy due to statistics extending over N adjacent letters of text. F_N is given by¹

$$\begin{aligned} F_N &= -\sum_{i,j} p(b_i, j) \log_2 p_{b_i}(j) \\ &= -\sum_{i,j} p(b_i, j) \log_2 p(b_i, j) + \sum_i p(b_i) \log p(b_i) \end{aligned} \quad (1)$$

in which: b_i is a block of $N-1$ letters [($N-1$)-gram]

j is an arbitrary letter following b_i

$p(b_i, j)$ is the probability of the N -gram b_i, j

$p_{b_i}(j)$ is the conditional probability of letter j after the block b_i ,

and is given by $p(b_i, j)/p(b_i)$.

The equation (1) can be interpreted as measuring the average uncertainty (conditional entropy) of the next letter j when the preceding $N-1$ letters are known. As N is increased, F_N includes longer and longer range statistics and the entropy, H , is given by the limiting value of F_N as $N \rightarrow \infty$:

$$H = \lim_{N \rightarrow \infty} F_N. \quad (2)$$

What Can we Do w/ LMs?

- Score sentences, e.g., $P(X = \text{"Jane went to the store"})$:

Jane went to the store . \rightarrow high

store to Jane went the . \rightarrow low

(same as calculating loss for training)

- Generate sentences:

while didn't choose end-of-sentence symbol, i.e., [EOS]:

calculate probability $P(\text{Next Word} \mid \text{Context})$

sample a new word from the probability distribution

N-gram Language Models

Review: Count-based Unigram Model

- **Independence assumption:** $P(x_i | x_1, \dots, x_{i-1}) \approx P(x_i)$
- **Maximum-likelihood estimation (MLE): counting how likely each word appearing in a corpus**

$$P_{\text{MLE}}(x_i) = \frac{c_{\text{train}}(x_i)}{\sum_{\tilde{x}} c_{\text{train}}(\tilde{x})}$$

- **Interpolation w/ UNK model:**

$$P(x_i) = (1 - \lambda_{\text{unk}}) * P_{\text{MLE}}(x_i) + \lambda_{\text{unk}} * P_{\text{unk}}(x_i)$$

From Unigram to Bigram LM

- Next word prediction only depends on the previous word.

$$P(x_i | x_{i-n+1}, \dots, x_1) \approx P(x_i | x_{i-1})$$

- Given a training corpus of 3 sentences:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

- Compute the bigram probability by counting (MLE):

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67 \quad P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33 \quad P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5 \quad P(\text{Sam} | \text{am}) = \frac{1}{2} = .5 \quad P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

- Probability of a sentence is just the product of all bigram probabilities in this sentence.

Higher-order n -gram Models

- Limit context length to n

$$P(x_i | x_{i-n+1}, \dots, x_1) \approx P(x_i | x_{i-n+1}, \dots, x_{i-1})$$

- Maximum likelihood estimation: count, and divide

$$P_{ML}(x_i | x_{i-n+1}, \dots, x_{i-1}) := \frac{c(x_{i-n+1}, \dots, x_i)}{c(x_{i-n+1}, \dots, x_{i-1})}$$

$$P(\text{example} | \text{this is an}) = \frac{c(\text{this is an example})}{c(\text{this is an})}$$

- Add smoothing by linear interpolation with $(n-1)$ -gram LM, to deal with zero counts:

$$P(x_i | x_{i-n+1}, \dots, x_{i-1}) = \lambda P_{ML}(x_i | x_{i-n+1}, \dots, x_{i-1}) \\ + (1 - \lambda) P(x_i | x_{1-n+2}, \dots, x_{i-1})$$

Smoothing for words in unknown context

- Add-one smoothing, i.e., every word adds 1 count.

$$P(x_i) = \frac{\text{count}(x_i)}{N}$$

$$P(x_i) = \frac{\text{count}(x_i) + 1}{N + V}$$

- Add smoothing by linear interpolation with (n-1)-gram LM, to deal with zero counts:

$$P(x_i \mid x_{i-n+1}, \dots, x_{i-1}) = \lambda P_{ML}(x_i \mid x_{i-n+1}, \dots, x_{i-1}) \\ + (1 - \lambda) P(x_i \mid x_{1-n+2}, \dots, x_{i-1})$$

More Smoothing Methods

(e.g. Goodman 1998)

- **Additive/Dirichlet:**

fallback distribution

$$P(x_i | x_{i-n+1}, \dots, x_{i-1}) := \frac{c(x_{i-n+1}, \dots, x_i) + \alpha P(x_i | x_{i-n+2}, \dots, x_{i-1})}{c(x_{i-n+1}, \dots, x_{i-1}) + \alpha}$$

interpolation hyperparameter

- **Discounting:**

discount hyperparameter

$$P(x_i | x_{i-n+1}, \dots, x_{i-1}) := \frac{c(x_{i-n+1}, \dots, x_i) - d + \alpha P(x_i | x_{i-n+2}, \dots, x_{i-1})}{c(x_{i-n+1}, \dots, x_{i-1})}$$

interpolation calculated by sum of discounts $\alpha = \sum_{\{\tilde{x}; c(x_{i-n+1}, \dots, \tilde{x}) > 0\}} d$

- **Kneser-Ney:** discounting w/ modification of the lower-order distribution

Problems and Solutions?

- Cannot share strength among **similar words**

she bought a car she bought a bicycle
she purchased a car she purchased a bicycle

→ solution: class based language models

- Cannot condition on context with **intervening words**

Dr. Jane Smith Dr. Gertrude Smith

→ solution: skip-gram language models

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet
for programming class he wanted to buy his own computer

→ solution: cache, trigger, topic, syntactic models, etc.

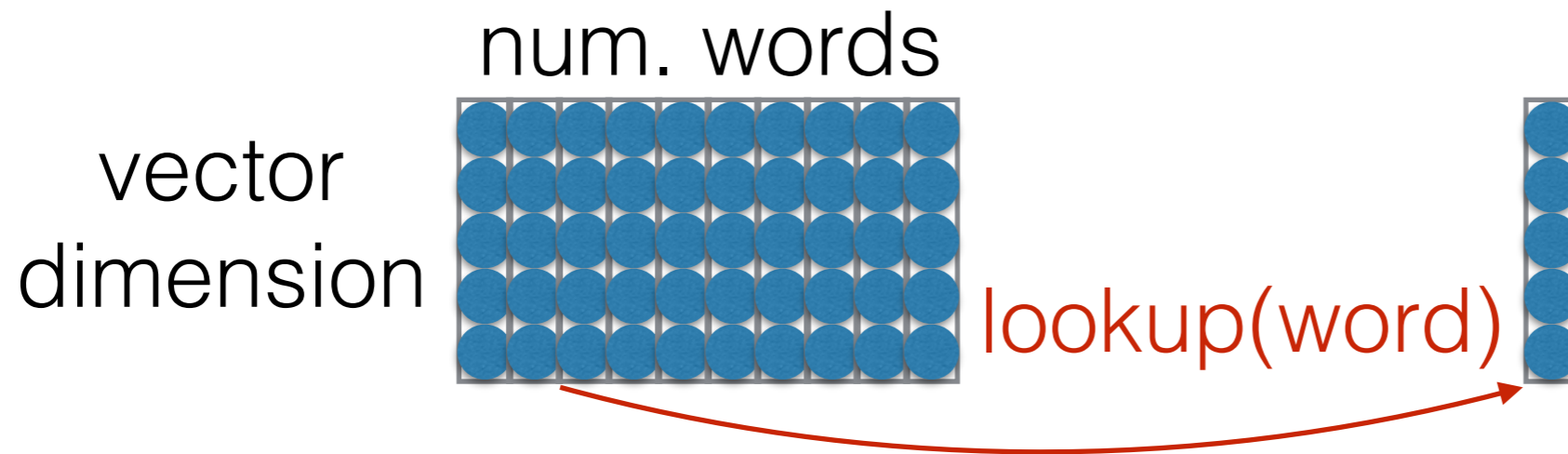
An Alternative:
Featurized Log-Linear Models
(Rosenfeld 1996)

An Alternative: Featurized Models

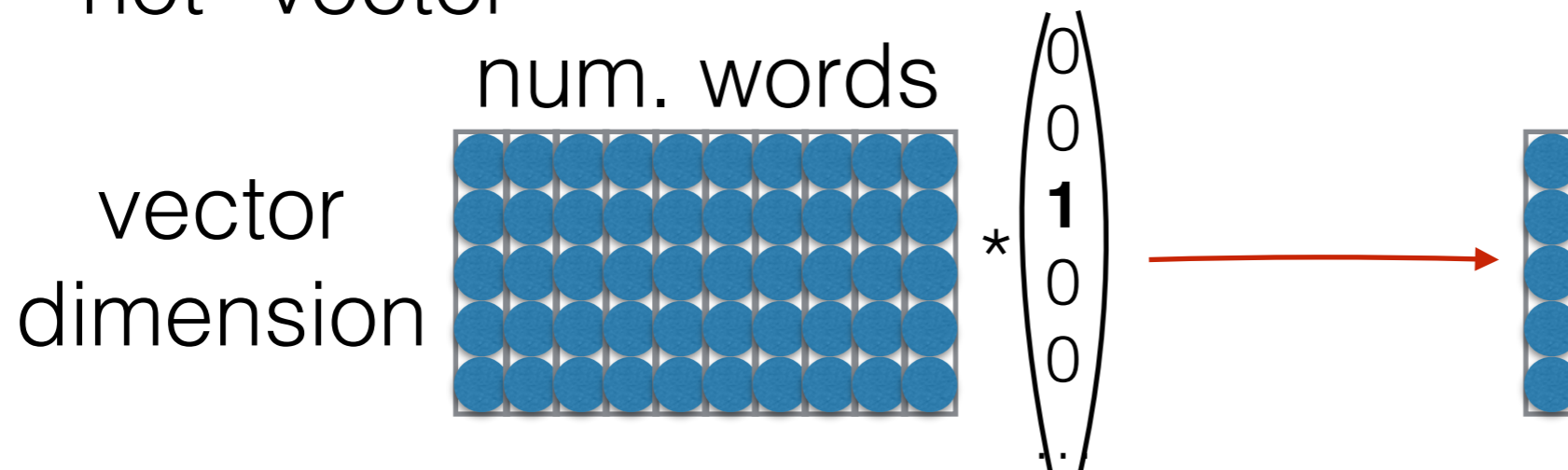
- Calculate features of the context
- Based on the features, calculate probabilities
- Optimize feature weights using gradient descent, etc.

A Note: “Lookup”

- Lookup can be viewed as “grabbing” a single vector from a big matrix of word embeddings



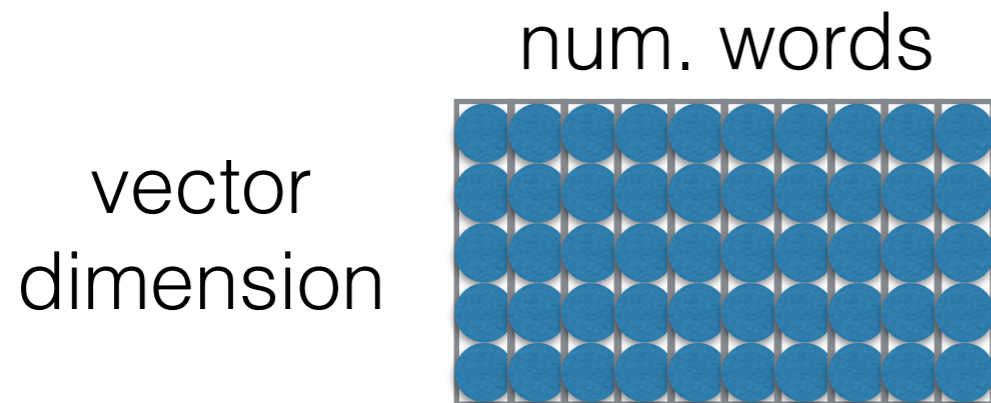
- Similarly, can be viewed as multiplying by a “one-hot” vector



- Former tends to be faster

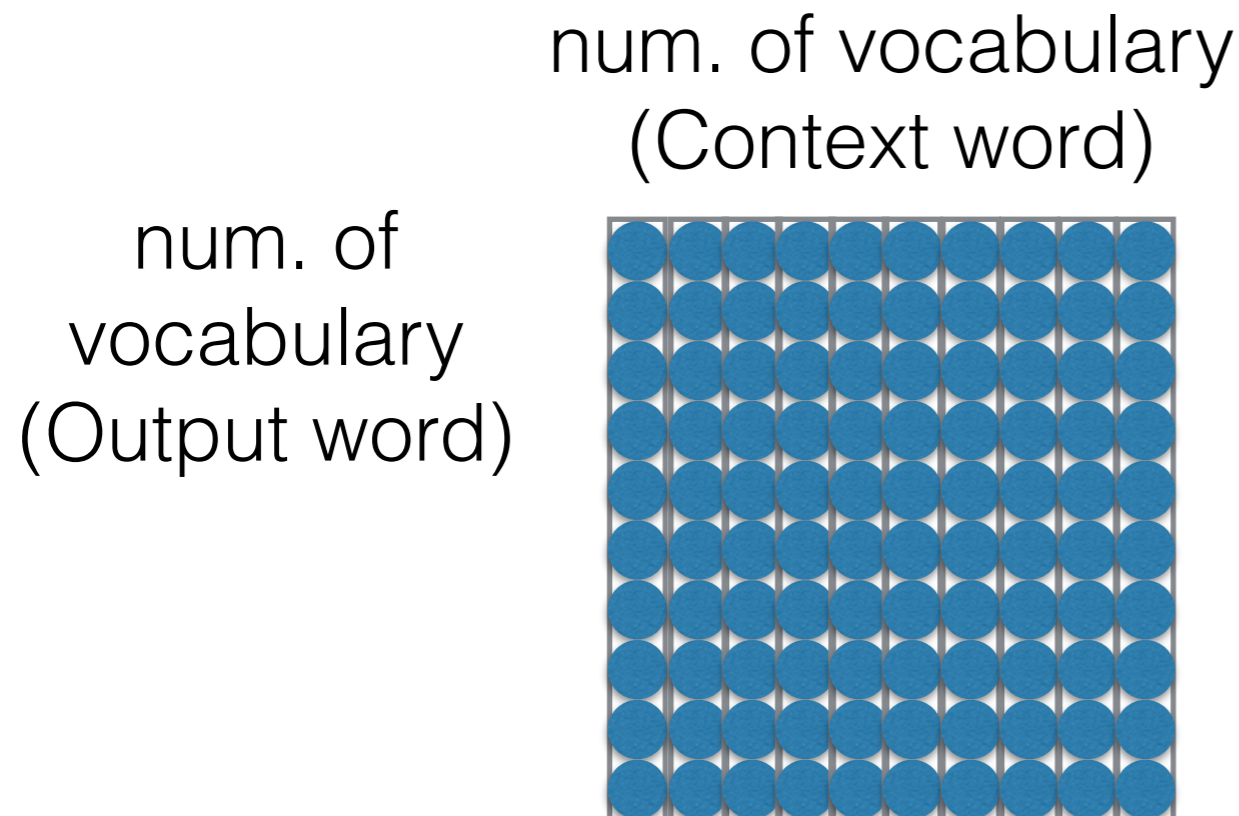
Word-to-Word Similarity

- If we have word embeddings, we can compute the word-to-word similarity matrix



$$X \in \mathbb{R}^{d \times V}$$

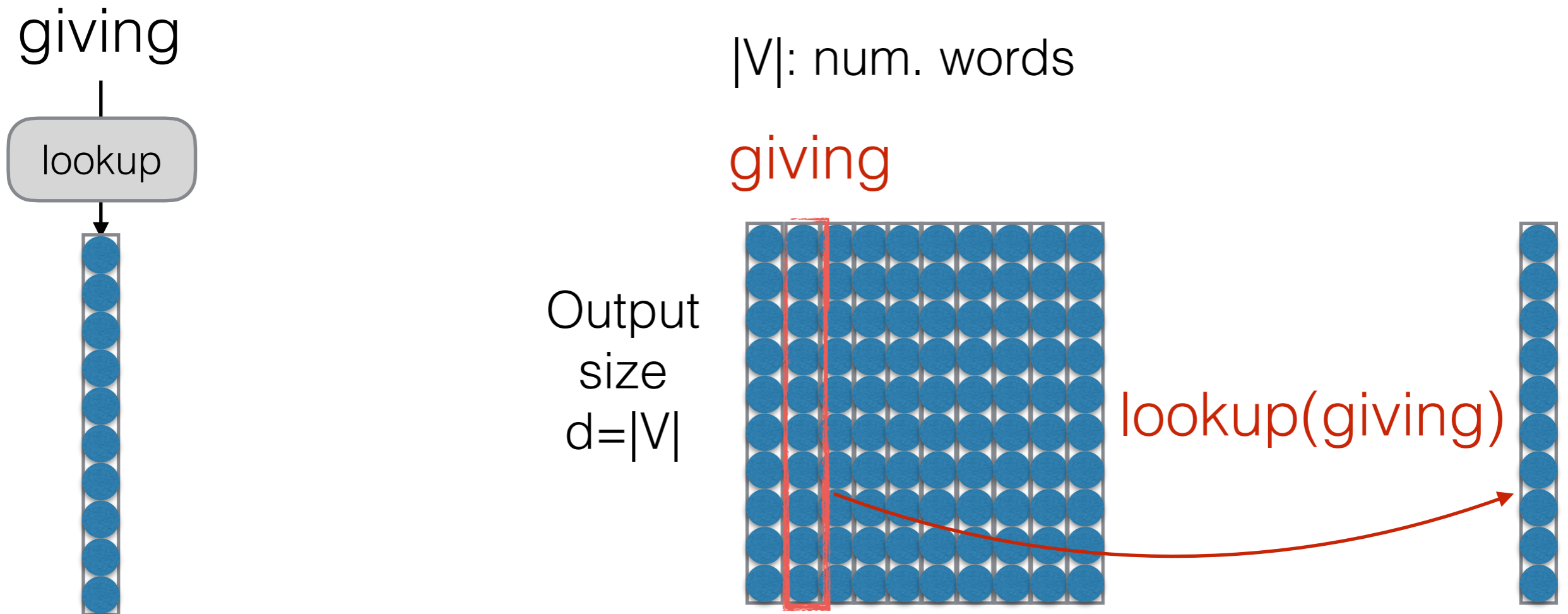
- If no word embeddings, just directly learn the similarity matrix



$$S = X^T X \in \mathbb{R}^{V \times V}$$

An Alternative: Featurized Models

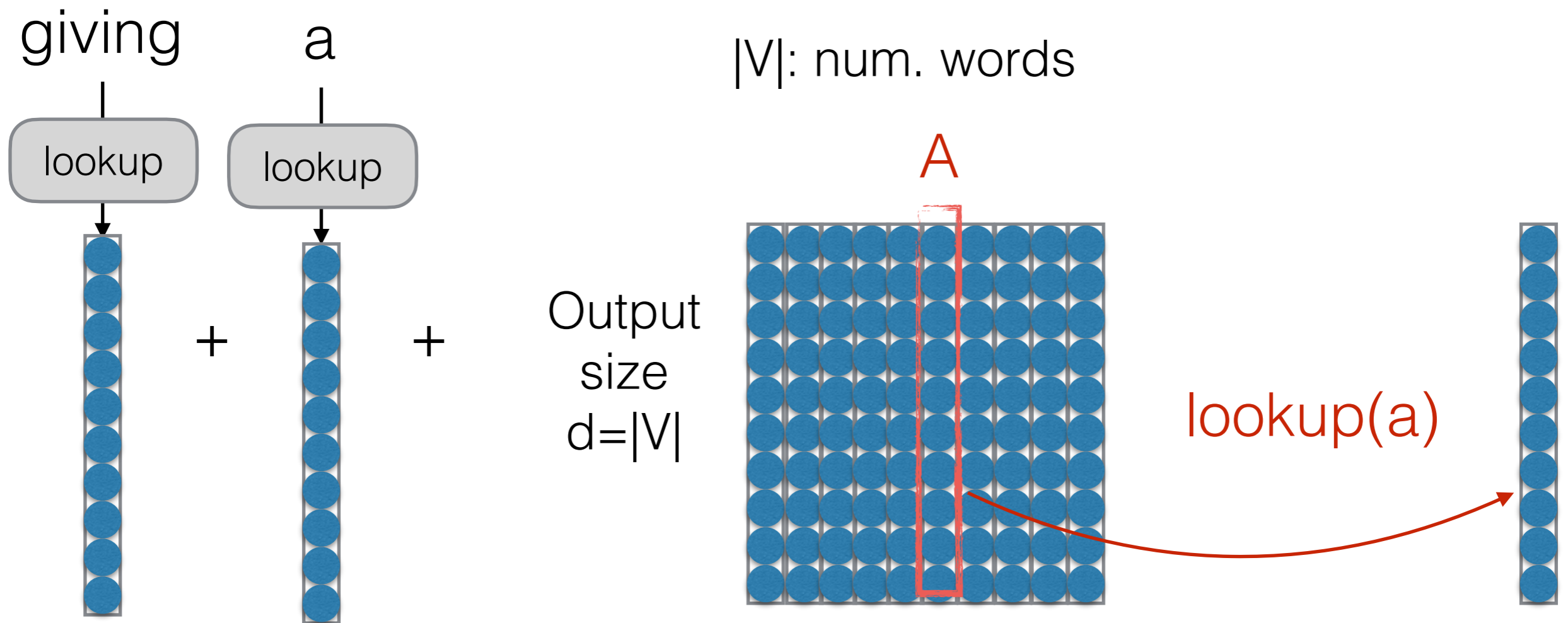
- Assume that we aim to learn a feature matrix W_0 where each column corresponds to a feature vector for each word.



- The word vector learns the similarity (coexistence) between the selected word (i.e., “giving”) and the other words, i.e., the likelihood of the next word coexisting with “giving” in the context

An Alternative: Featurized Models

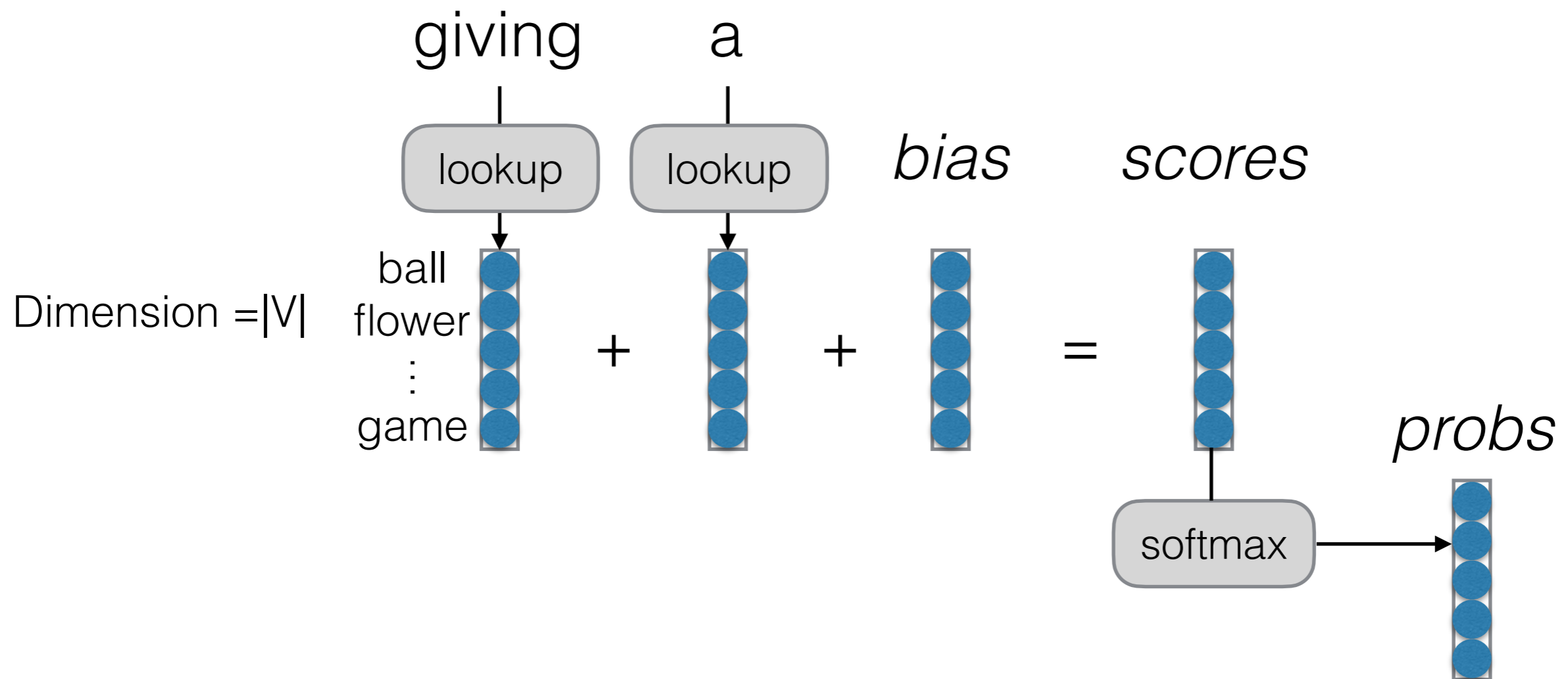
- Assume that we aim to learn a feature matrix W_0 where each column corresponds to a feature vector for each word.



- The word vector learns the similarity (coexistence) between the selected word (i.e., "giving") and the other words, i.e., the likelihood of the next word coexisting with "giving" in the context

An Alternative: Featurized Models

- Combine with the bias vector (model parameter), compute the probability over the output vocabulary V
- Each word vector has the size of output vocabulary, where each element represents **the probability of the output word given the current word in the context**



- Feature weights optimized by SGD, etc.
- What are similarities/differences w/ BOW classifier?

Example:

Previous words: "giving a"

a
the
talk
gift
hat
...

$$b = \begin{pmatrix} 3.0 \\ 2.5 \\ -0.2 \\ 0.1 \\ 1.2 \\ \dots \end{pmatrix}$$

$$w_a = \begin{pmatrix} -6.0 \\ -5.1 \\ 0.2 \\ 0.1 \\ 0.5 \\ \dots \end{pmatrix}$$

$$w_{\text{giving}} = \begin{pmatrix} -0.2 \\ -0.3 \\ 1.0 \\ 2.0 \\ -1.2 \\ \dots \end{pmatrix}$$

$$s = \begin{pmatrix} -3.2 \\ -2.9 \\ 1.0 \\ 2.2 \\ 0.6 \\ \dots \end{pmatrix}$$

Vocabulary:
Words we're
predicting

How likely
are they?

How likely
are they
given prev.
word is "a"?

How likely
are they
given 2nd prev.
word is "giving"?

Total
score

Reminder: Training Algorithm

- Calculate the **gradient of the loss function** with respect to the parameters

$$\frac{\partial \mathcal{L}_{\text{train}}(\theta)}{\partial \theta}$$

- How? Use the chain rule / back-propagation.
More in a second
- **Update** to move in a direction that decreases the loss

$$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}_{\text{train}}(\theta)}{\partial \theta}$$

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car she bought a bicycle
she purchased a car she purchased a bicycle

→ not solved yet 😞

- Cannot condition on context with **intervening words**

Dr. Jane Smith Dr. Gertrude Smith

→ solved! 😊

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet
for programming class he wanted to buy his own computer

→ not solved yet 😞

Neural Language Model

—Beyond Linear Models

Linear Models can't Learn Feature Combinations

students take tests → **high** teachers take tests → **low**
students write tests → **low** teachers write tests → **high**

- These can't be expressed by linear features
- What can we do?
 - Remember combinations as features (individual scores for “students take”, “teachers write”)
→ Feature space explosion!
 - Neural networks!

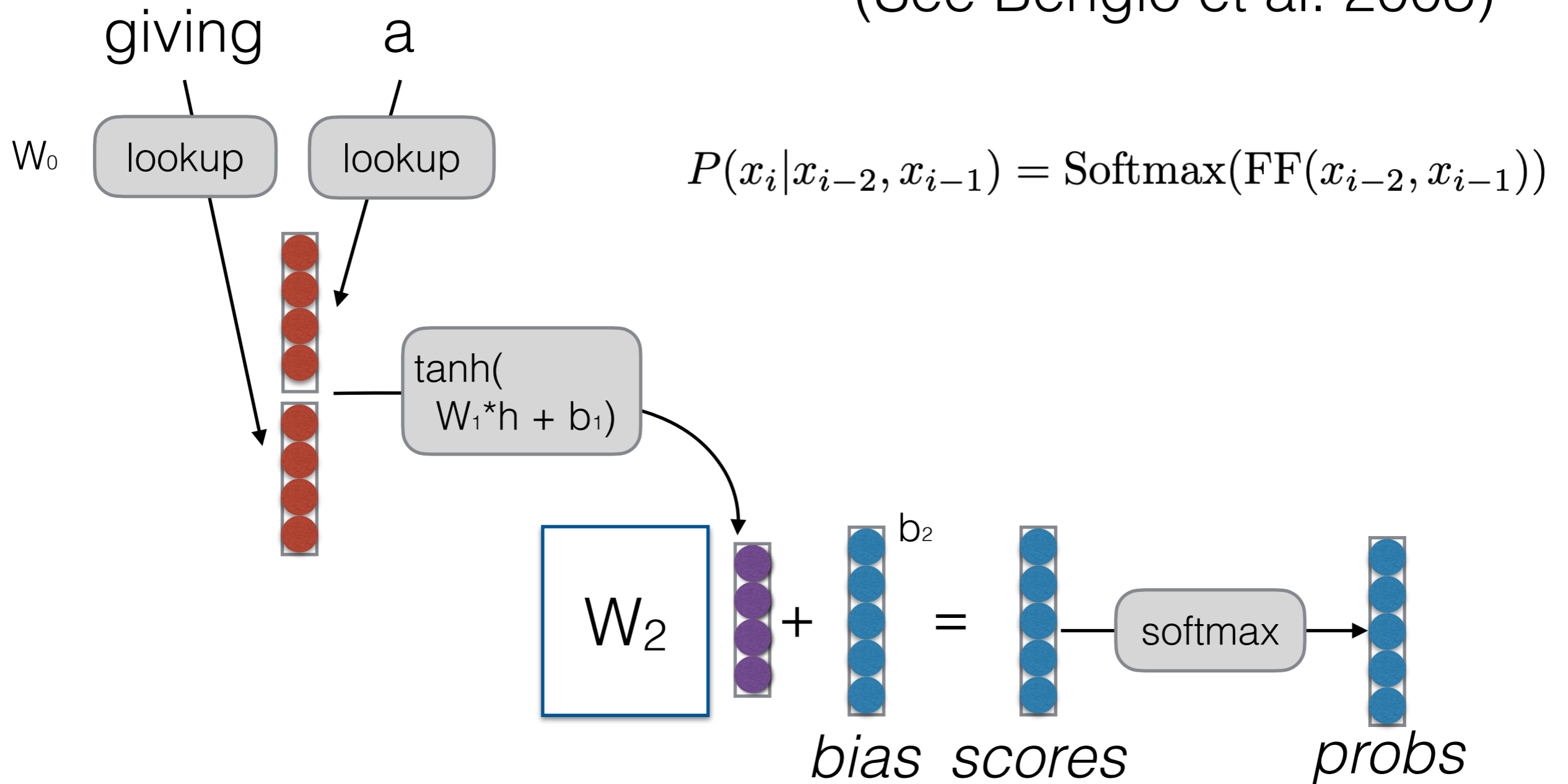
Neural Language Models

- Convert the word prediction problem to discriminative text classification
- The input is the $n-1$ previous words (context)
- The output is a word in the vocabulary (V-class classification)

$$P(x_i | x_{i-n+1}, \dots, x_{i-1}) = \text{Softmax}(\text{NeuralNet}(x_{i-n+1}, \dots, x_{i-1}))$$

Feed-forward Neural Language Models

- (See Bengio et al. 2003)



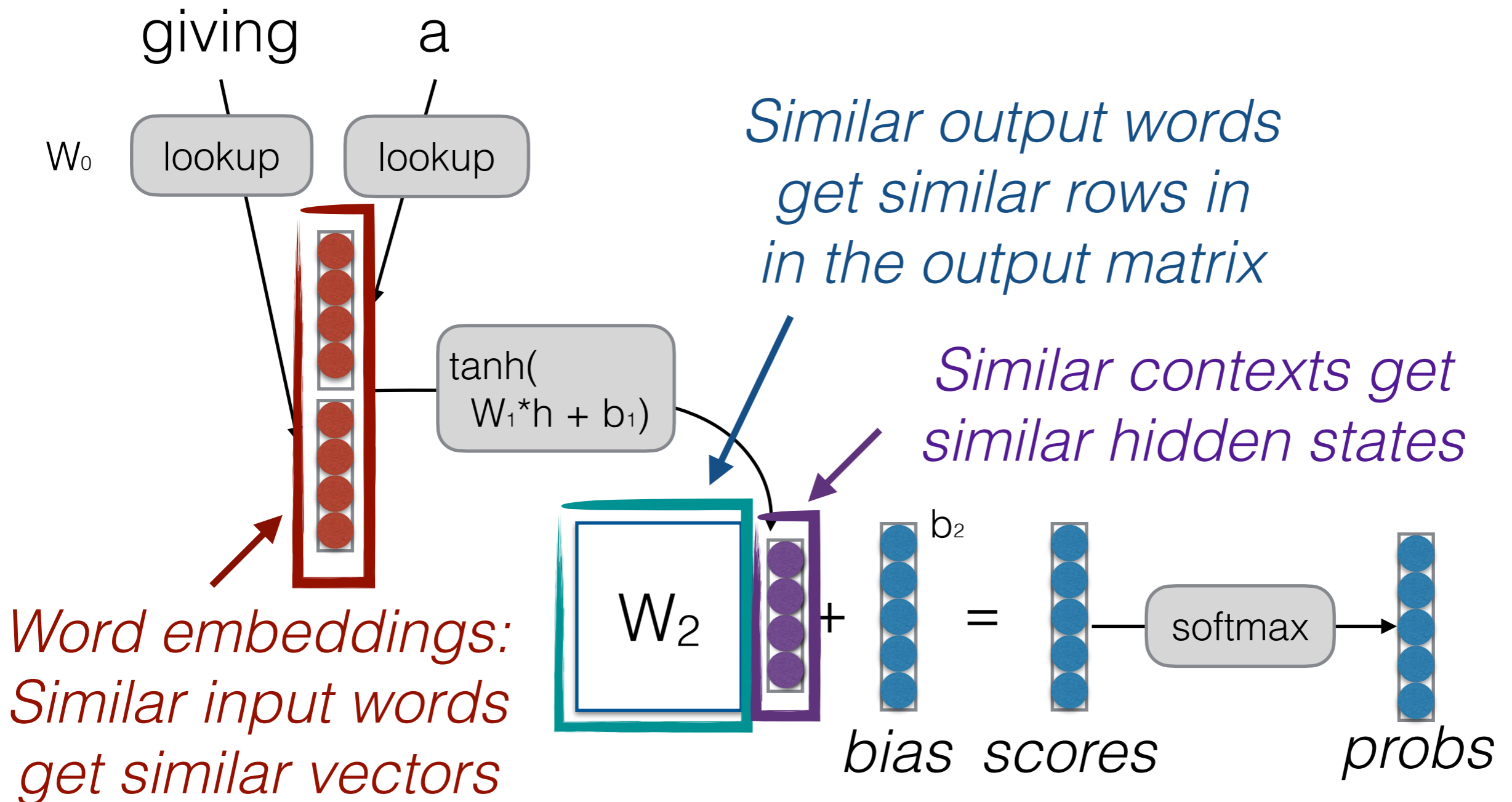
Example of Combination Features

- Word embeddings capture features of words
 - e.g. feature 1 indicates verbs, feature 2 indicates determiners
- A row in the weight matrix (together with the bias) can capture particular *combinations* of these features
 - e.g. the 34th row in the weight matrix looks at feature 1 in the second-to-previous word, and feature 2 in the previous word

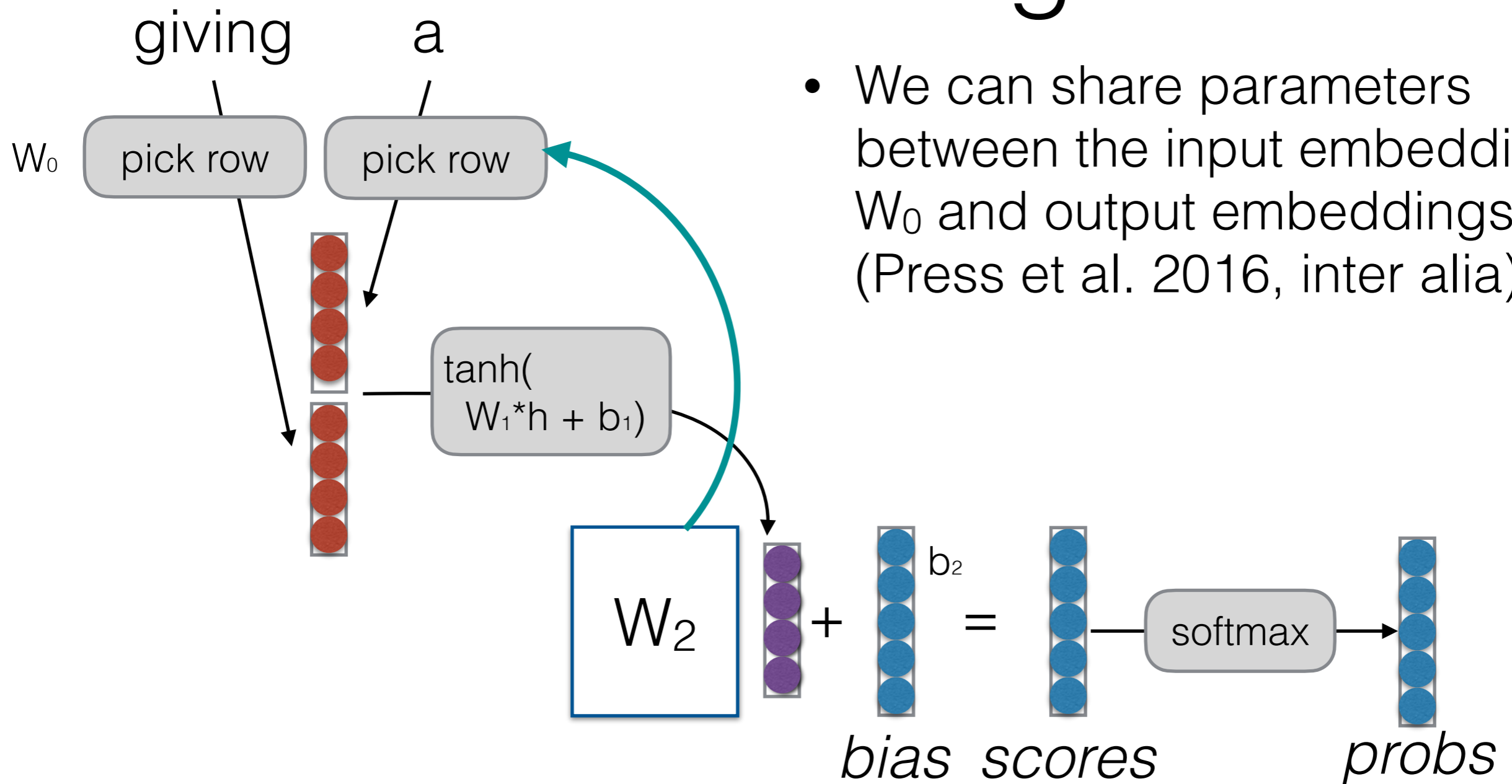
$$\begin{array}{l} \text{giving} \\ \text{a} \end{array} \begin{array}{|c|} \hline 1.2 \\ -0.1 \\ 0.7 \\ -2.1 \\ 0.5 \\ \hline \end{array} * \begin{array}{|c|} \hline 1.5 \\ 0 \\ 0 \\ 0 \\ 0 \\ \hline \end{array} + \begin{array}{|c|} \hline -2 \\ \hline \end{array} = \begin{array}{l} \text{positive number if} \\ \text{the previous word is a} \\ \text{determiner and} \\ \text{second-to-previous} \\ \text{word is a verb} \end{array}$$

The diagram illustrates the calculation of a combination feature for the word "giving". It shows the dot product of the word embedding for "giving" (a 5x1 vector with values 1.2, -0.1, 0.7, -2.1, 0.5) and the 34th row of the weight matrix W_{34} (a 5x1 vector with values 1.5, 0, 0, 0, 0). This result is then added to the bias b_{34} (a scalar value of -2). The final result is a positive number, which indicates that the previous word is a determiner and the second-to-previous word is a verb.

Where is Strength Shared?



Tying Input/Output Embeddings



- We can share parameters between the input embeddings W_0 and output embeddings W_2 (Press et al. 2016, inter alia)

Want to try? Delete the input embeddings W_0 , and instead pick a row from the output matrix W_2 .

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car she bought a bicycle
she purchased a car she purchased a bicycle

→ solved, and similar contexts as well! 😊

- Cannot condition on context with **intervening words**

Dr. Jane Smith Dr. Gertrude Smith

→ solved! 😊

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet
for programming class he wanted to buy his own computer

→ not solved yet 😞

Many Other Potential Designs!

- Neural networks allow design of arbitrarily complex functions!
- In future classes, we can replace Feedforward models by more powerful sequential NNs:
 - **Recurrent neural network LMs**
 - **Transformer LMs**

When to Use n-gram Models?

- Neural language models (next) achieve better performance, but
- n-gram models are extremely fast to estimate/apply
- n-gram models can be better at modeling low-frequency phenomena
- **Toolkit:** kenlm

<https://github.com/kpu/kenlm>

Comparison Between N-gram LM and Neural LMs

N-gram LM

(Nonparametric Modeling)

- **Count:** Build a table of all possible **prefix contexts** and their **counts** from the training corpus \mathcal{D}

$O(|\mathcal{D}|)$
Corpus size

Prefix Context	Next Word	Count
$x_{i-n+1}, \dots, x_{i-2}, x_{i-1}$	x_i	$c(x_{i-n+1}, \dots, x_i)$

N-gram LM

(Nonparametric Modeling)

- **Retrieve:** How to compute $P(x_i | x_{i-n+1}, \dots, x_{i-1})$?
 - Retrieve all rows that match the prefix
 - Divide the count of $(x_{i-n+1}, \dots, x_{i-1}, x_i)$ by the sum of counts of all retrieved rows

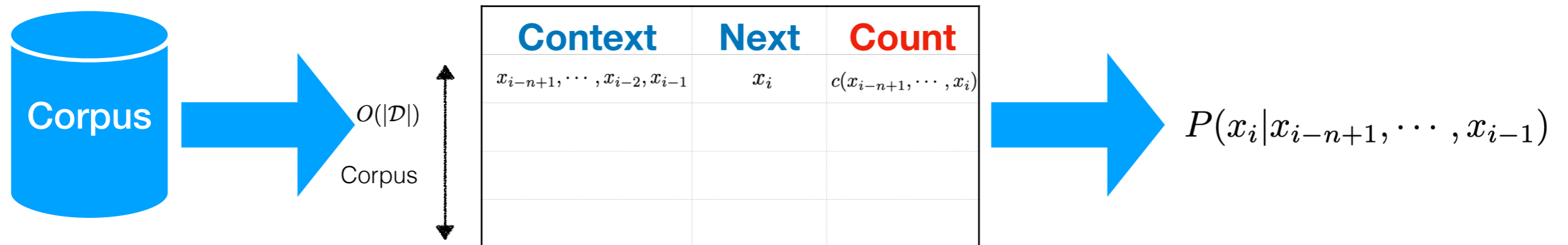
$$P(x_i | x_{i-n+1}, \dots, x_{i-1}) \approx \frac{c(x_{i-n+1}, \dots, x_{i-1}, x_i)}{\sum_{x' \in \mathcal{V}} c(x_{i-n+1}, \dots, x_{i-1}, x')}$$

$O(|\mathcal{D}|)$
Corpus

Prefix Context	Next Word	Count
$x_{i-n+1}, \dots, x_{i-2}, x_{i-1}$	x_i	$c(x_{i-n+1}, \dots, x_i)$

Count-based Language Modeling

- Count & Retrieve!



Neural Language Model

(Bengio et al. 2003)

- **Non-parametric to parametric:** We replace a count table with a neural network to approximate the next-word prediction probability:

$$P_{\theta}(x_i|x_{i-n+1}, \dots, x_{i-1}) = \text{Softmax}(\text{NeuralNet}(x_{i-n+1}, \dots, x_{i-1}))$$

θ is the LM parameter

- **Training:** Use SGD to optimize the network by minimizing the **negative log-likelihood** of next word prediction

$$\mathcal{L}_{\text{nll}}(\theta) = -\mathbb{E}_{x_i|x_{<i}} \log P_{\theta}(x_i|x_{i-n+1}, \dots, x_{i-1})$$

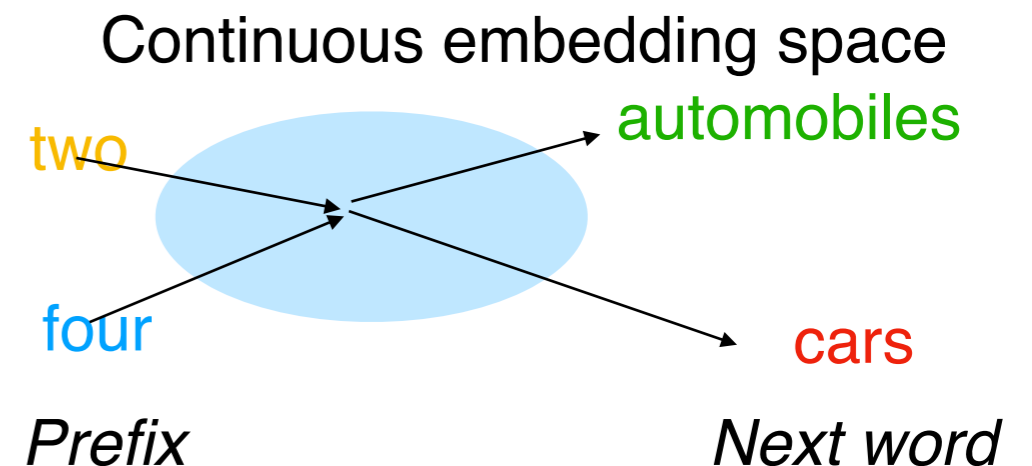
Sample a pair of (prefix context, next word) from a large corpus

Neural LM v.s. Count LM

- A neural LM *learns to **count***, but also *learns to **compress***
 - Learning is **not** simply counting
 - Neural nets **memorize** texts but also **generalize**
 - Similar text inputs are clustered to save **no. of bits**
 - Similarity is implicitly imposed by similarity between **prefix context** and **next word**

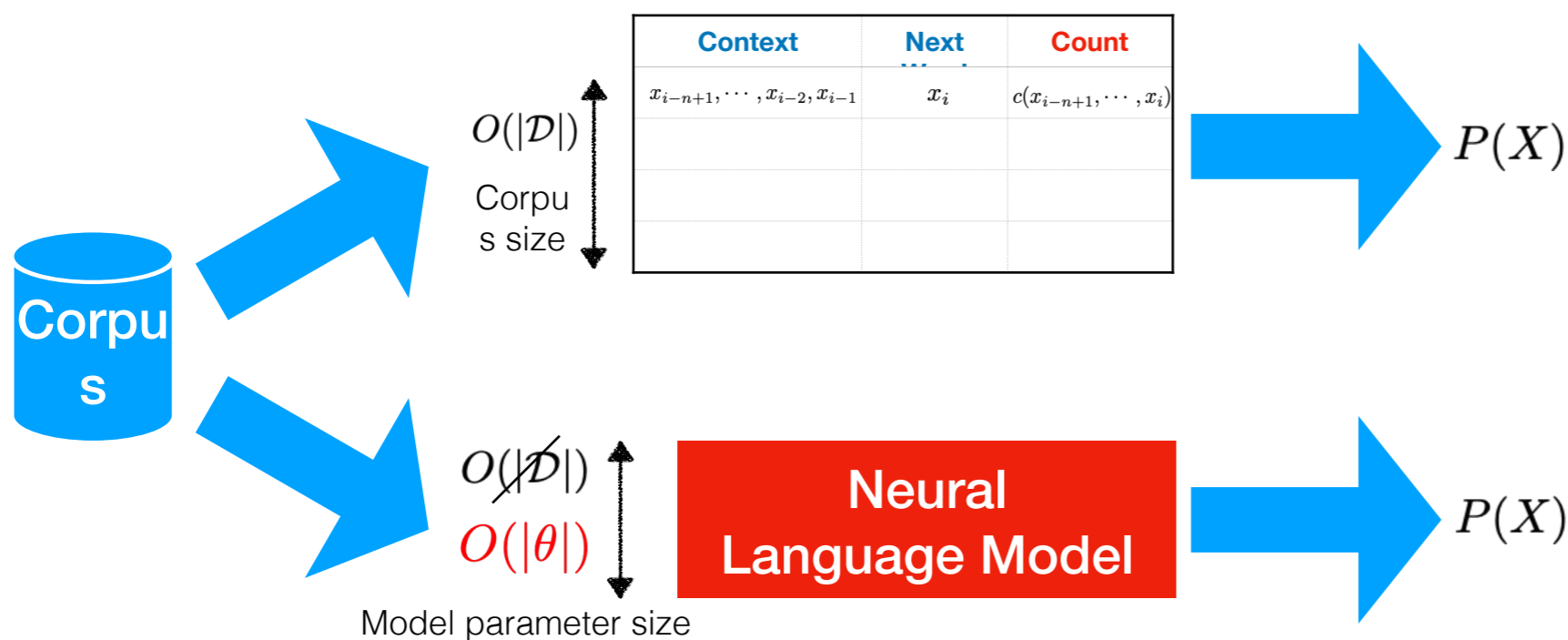
Example:

- Training examples
 - Peter and Mary have **two cars**.
 - There are **four cars** on the road.
 - The parking lots have **four automobiles**.
- Q: How likely is “**automobiles**” followed by “**two**”?



Neural LM learns to count & compress

- Neural LM generalizes by compressing the no. of rows



Non-parametric

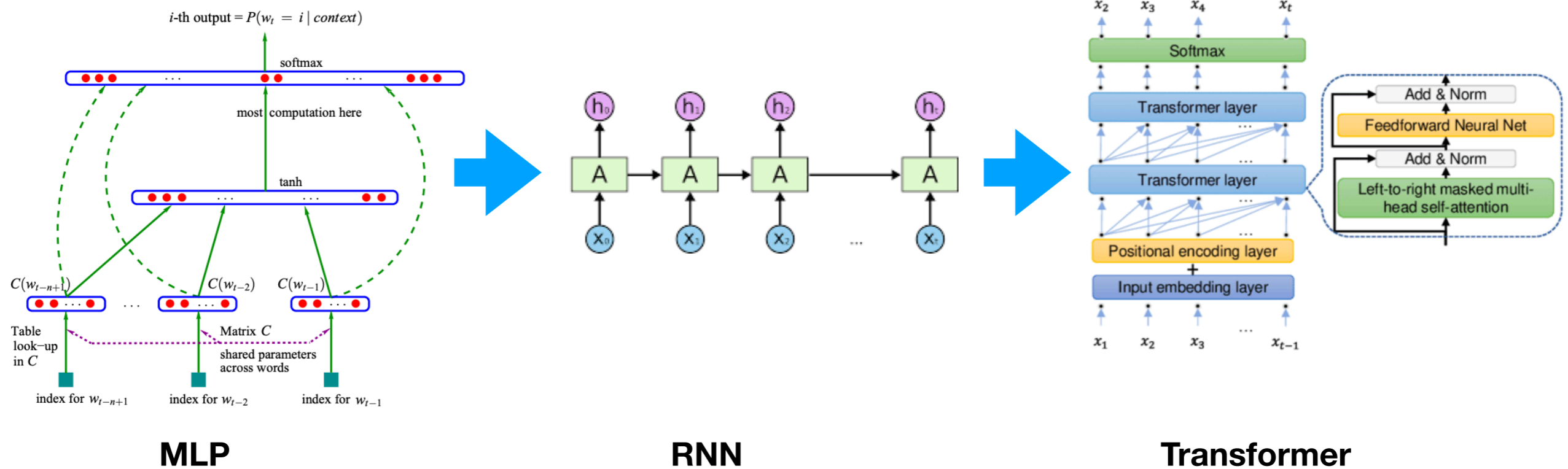
- Memorize low-frequency pairs of (prefix, next word)
- Cannot generalize to unseen pairs
- Interpretable & easy for incremental updates (insert/delete/replace)

Parametric

- Learning from data-driven training
- Good generalization to unseen pairs
- Black-box & expensive for parameter updates with forgetting issues

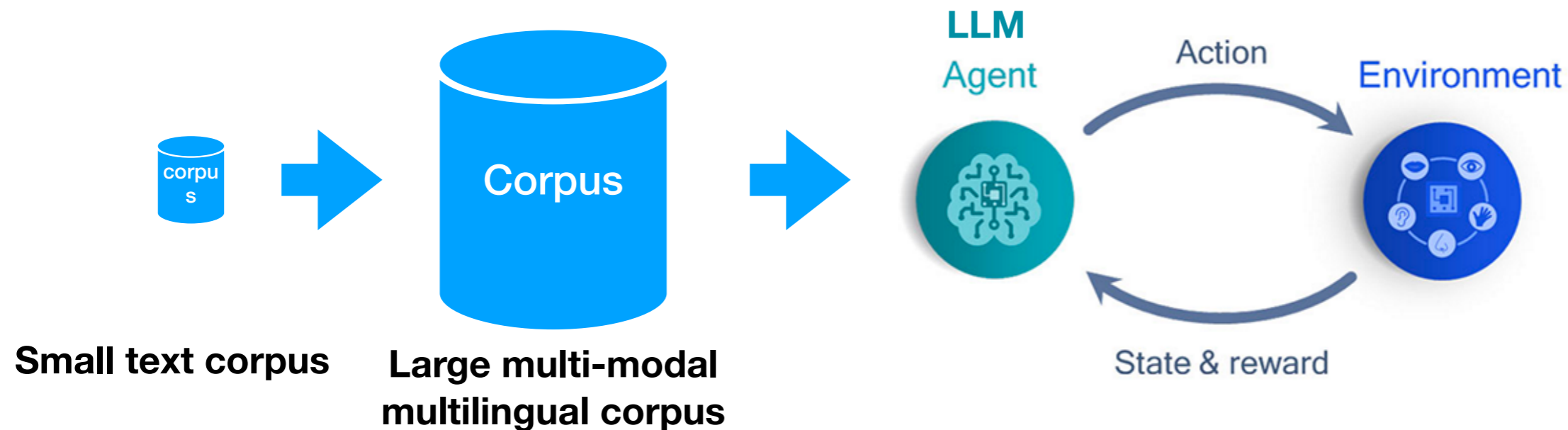
What makes LLM so powerful?

- **Advances from neural network architecture:**
 - Scale up the size of model parameters to 100B+



What makes LLM so powerful?

- **Pre-training** on massive raw **texts** (and even **images**) in **100+ languages**
 - BERT, GPT-4, PaLM, T5, LLaMA, ...
- **Fine-tuning** on language **instructions** with **supervised learning** or **RL with human feedback**
 - ChatGPT, Bard, FlanT5, Alpaca, ...



LM Evaluation

Evaluation of LMs

- **Log-likelihood:**

$$LL(\mathcal{D}_{\text{test}}) = \sum_{X \in \mathcal{D}_{\text{test}}} \log P(X)$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{D}_{\text{test}}) = \frac{1}{\sum_{X \in \mathcal{D}_{\text{test}}} |X|} \sum_{X \in \mathcal{D}_{\text{test}}} \log P(X)$$

- **Per-word (Cross) Entropy:**

$$H(\mathcal{D}_{\text{test}}) = \frac{1}{\sum_{X \in \mathcal{D}_{\text{test}}} |X|} \sum_{X \in \mathcal{D}_{\text{test}}} -\log_2 P(X)$$

- **Perplexity:**

$$ppl(\mathcal{D}_{\text{test}}) = 2^{H(\mathcal{D}_{\text{test}})} = e^{-WLL(\mathcal{D}_{\text{test}})}$$

Unknown Words

- Necessity for UNK words
 - We won't have all the words in the world in training data
 - Larger vocabularies require more memory and computation time
- Common ways:
 - Limit vocabulary by frequency threshold (usually UNK ≤ 1) or rank threshold
 - Model characters or subwords

Evaluation and Vocabulary

- **Important:** the vocabulary must be the same over models you compare
- Or more accurately, all models must be able to generate the test set (it's OK if they can generate *more* than the test set, but not less)
- e.g. Comparing a character-based model to a word-based model is fair, but not vice-versa

LM Problem Definition

Count-based LMs

Evaluating LMs

Log-linear LMs

Neural Net Basics

Feed-forward NN LMs

Questions?