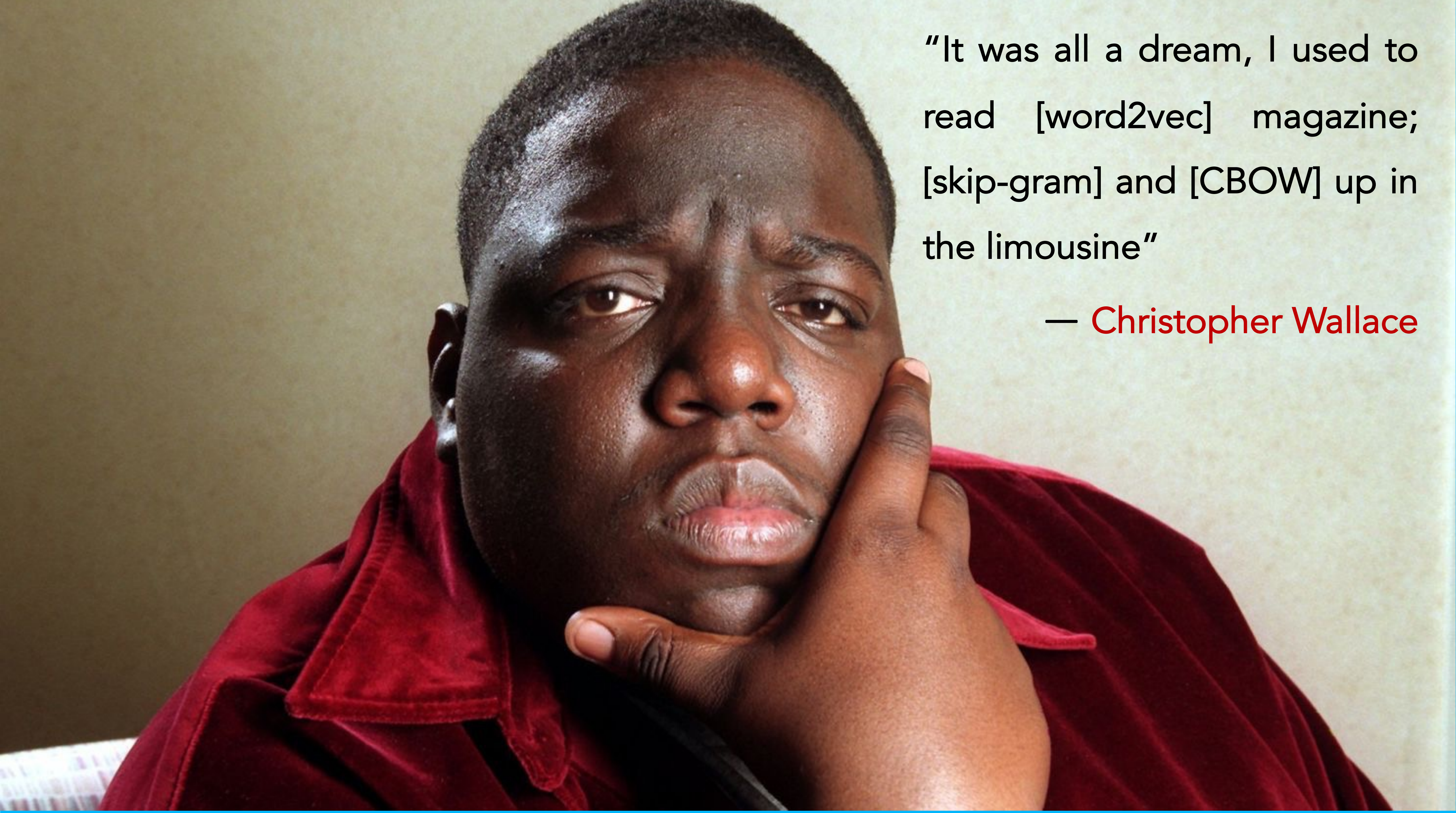
Lecture 4: Neural Language Models

An introduction with word2vec

Harvard
AC295/CS287r/CSCI E-115B
Chris Tanner



"It was all a dream, I used to read [word2vec] magazine; [skip-gram] and [CBOW] up in the limousine"

— Christopher Wallace

ANNOUNCEMENTS

- Keep an eye on the **HW1 Errata**, posted on Ed. HW1 is due in 1 week!
- We now have a Slack workspace, mostly for lively discussion and research work
- Ed is still our main forum for all official communication, questions you may have, and remote Quizzes.
- Office Hours change a little:
  - ~~Mon @ 1pm – 3pm~~ Mon @ 1:30pm – 3:30pm
  - ~~Sat @ 10am – 11am~~ Sun @ 10am – 11am

ANNOUNCEMENTS

Research Project:

- Add your ideas and name/info to Research Brainstorming spreadsheet
- **Phase 1** is due Sept 30. Write a **1-page proposal** for one of your ideas listed on Research Brainstorming. Afterwards, we'll refine the list to ~25 projects and determine teams.
- **Phase 2** is due Oct 14. This will lock-in all 20 projects and teammates.

RECAP: L2

a t e

61	74	65

- Default character-level representations aren't useful
- Simple document-level representations (e.g., BoW and TFIDF) can be useful but have weaknesses:
  - Context-insensitive ("the horse ate" = "ate the horse")
  - Curse of Dimensionality (vocab could be over 100k)
  - Orthogonality: no concept of semantic similarity at the word-level

$$\text{TFIDF} = f_{w_i} * log\left(\frac{\text{\# docs in corpus}}{\text{\# docs containing } w_i}\right)$$

RECAP: L3

- Language Modelling is a core NLP task and highly useful for many other tasks.
- n-gram models (count-based) can be surprisingly useful but have weaknesses:
  - Must handle OOV words (all LMs must do this)
  - Unsustainable approach to handling increasingly larger contexts
  - No semantic information is conveyed by the counts (e.g., vehicle vs car)
- Perplexity is the canonical evaluation metric for LMs

Bi-gram model with alpha-beta smoothing

$$P(\text{"w w}'\text{"}) = \frac{n_{w,w'}(\boldsymbol{d}) + \beta * P(\text{w}')}{n_{w,w*}(\boldsymbol{d}) + \beta} \quad \text{, where } P(\text{w}') = \frac{n_{w'}(\boldsymbol{d}) + \alpha}{n_{w_*} + \alpha|V|}$$

Outline

- Featurized, Linear Model
- Neural Models
    - Bengio (2003)
    - word2vec (2013)
- Evaluation
- Remaining challenges

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Remaining Issues

1. More context while avoiding sparsity, storage, and compute issues

2. No semantic information conveyed by counts (e.g., vehicle vs car)

New goals!

3. Cannot leverage non-consecutive patterns

Dr. West ____ — Occurred 25 times

Dr. Cornell West ____ — Occurred 3 times

4. Cannot capture combinatorial signals (i.e., non-linear prediction)

P(Chef cooked food) high

P(Chef ate food) low

P(Customer cooked food) low

P(Customer ate food) high

Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

Featurized Model

Instead of counts, let's move toward having **words** represented as <u>features</u>, where # features ≪ # of words in vocab

We can develop a very simple linear model that calculates word probabilities

Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

Featurized Model

Specifically, let's have:

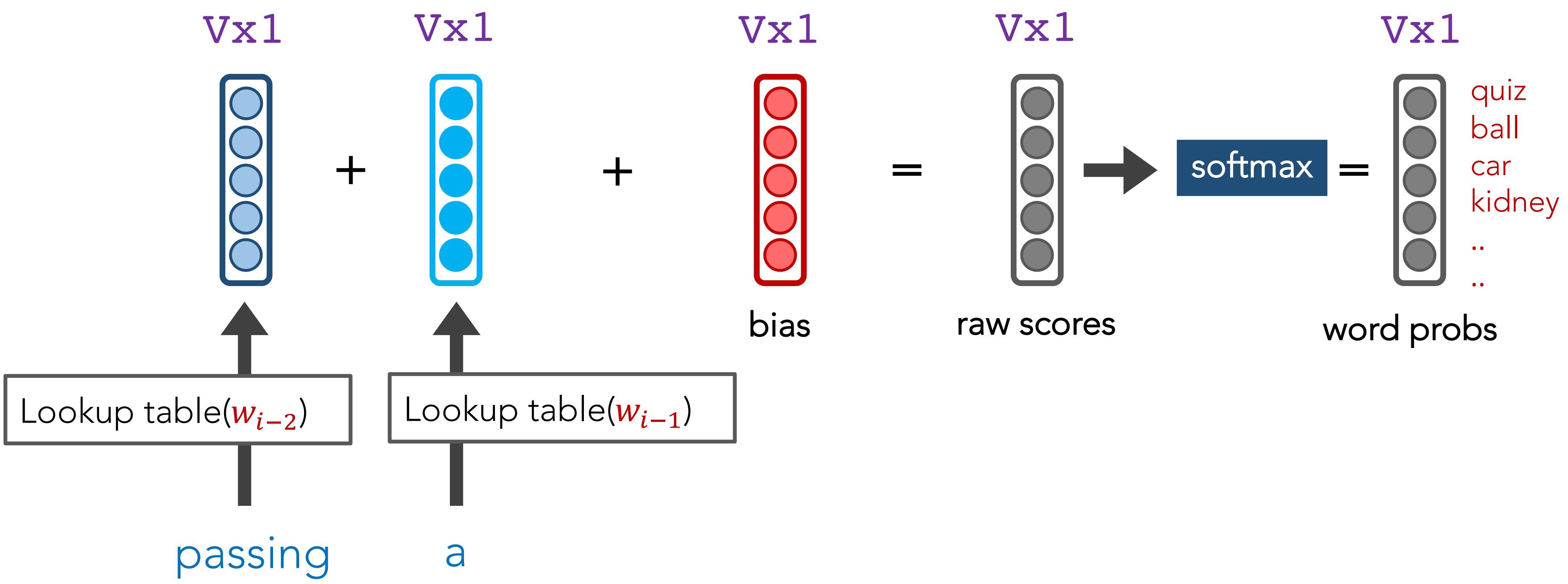
- 1 featurized representation for word w_{i-2}
- A <u>separate</u> representation for word w_{i-1}

Combine them w/ a bias, and predict the *next* word

Featurized Model

"passing a _____"

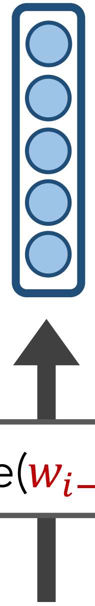
w_{i-2} w_{i-1} w_i



Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

Featurized Model

Vx1



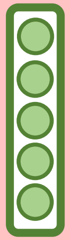
Lookup table(w_{i-2})

passing a

A "lookup table" is trivial.

It simply converts each unique word $\boldsymbol{w}$ to an index $i \in V$, where V is the size of our vocabulary.

We often work with the one-hot version of it, $\boldsymbol{x_i}$:

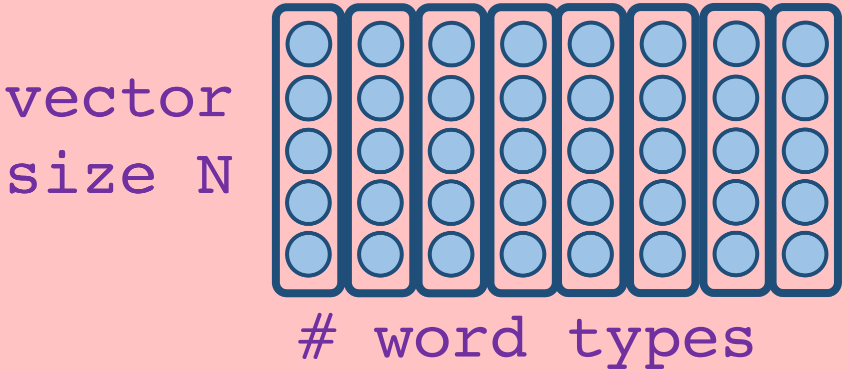


Vx1

$\boldsymbol{x_i}$

Featurized Model

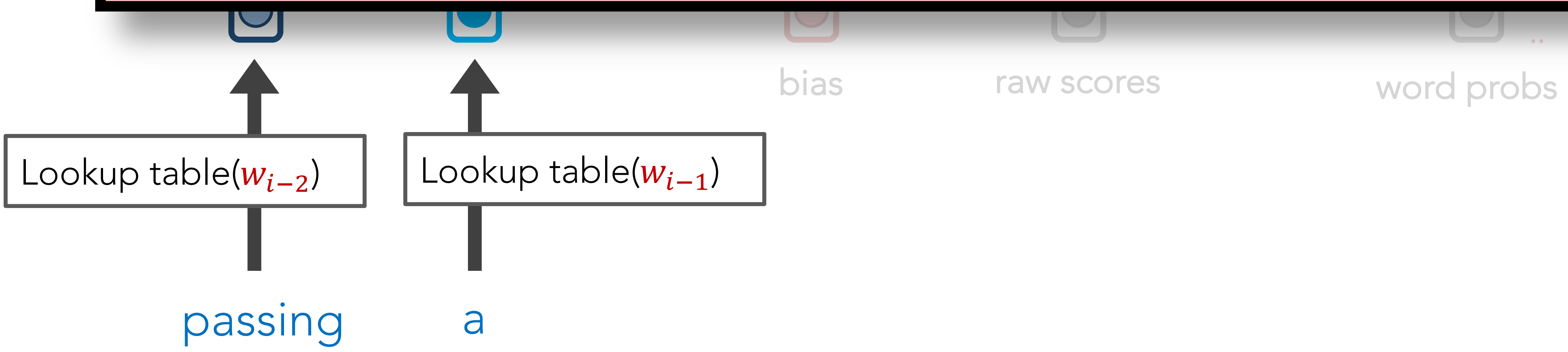
Embedding/ feature matrix $\boldsymbol{v}$ is an "input word matrix".
Each column of $\boldsymbol{v}$ corresponds to each unique word type $\boldsymbol{w}$



Retrieve a word $\boldsymbol{w}$'s Embedding via:

- Slicing the index $\boldsymbol{i}$, or
- Matrix multiply

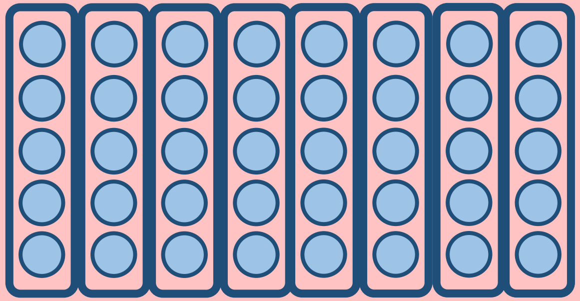
$$v_i = \boldsymbol{v}\boldsymbol{x}_{\boldsymbol{i}}$$



Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

Featurized Model

Embedding/ feature matrix $\boldsymbol{v}$ is an "input word matrix".
Each column of $\boldsymbol{v}$ corresponds to each unique word type $\boldsymbol{w}$



Retrieve a word $\boldsymbol{w}$'s Embedding via:

- Slicing the index $\boldsymbol{i}$, or
- Matrix multiply

$$v_i = \boldsymbol{v}\boldsymbol{x}_i$$

`Nx1 = NxV * Vx1`

Look

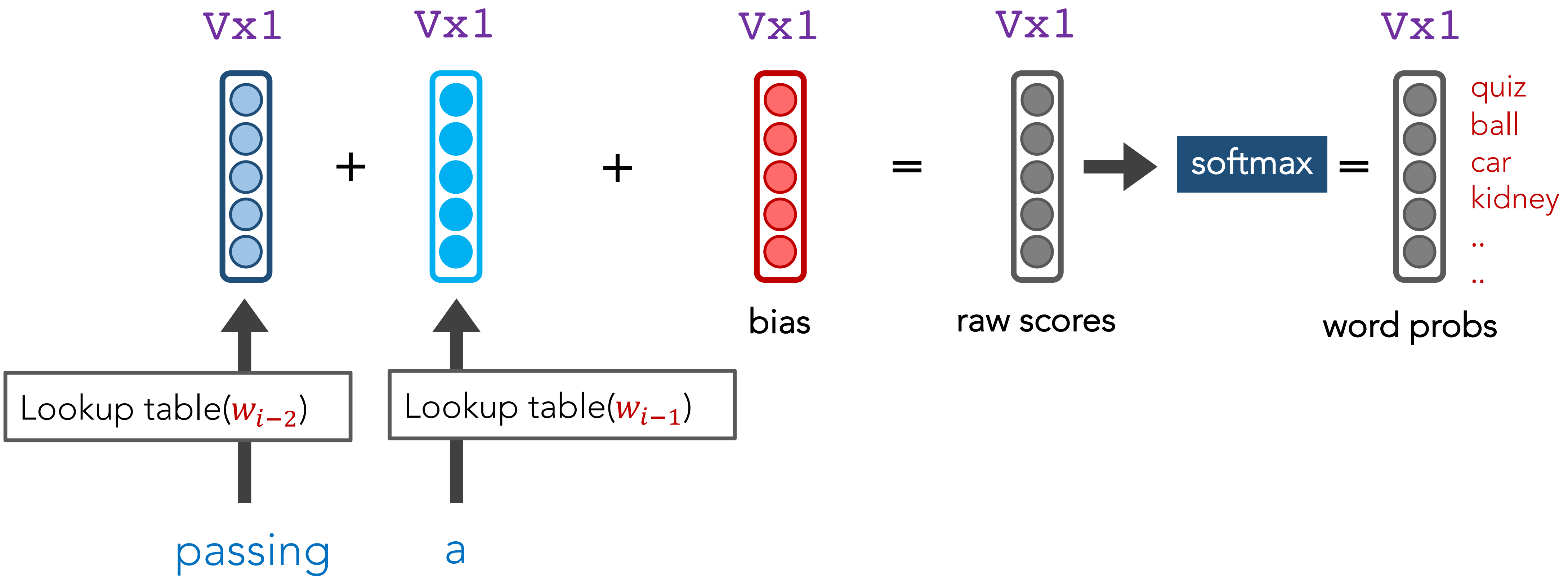
passing a

Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

Featurized Model

"passing a ______"

w_{i-2} w_{i-1} w_i



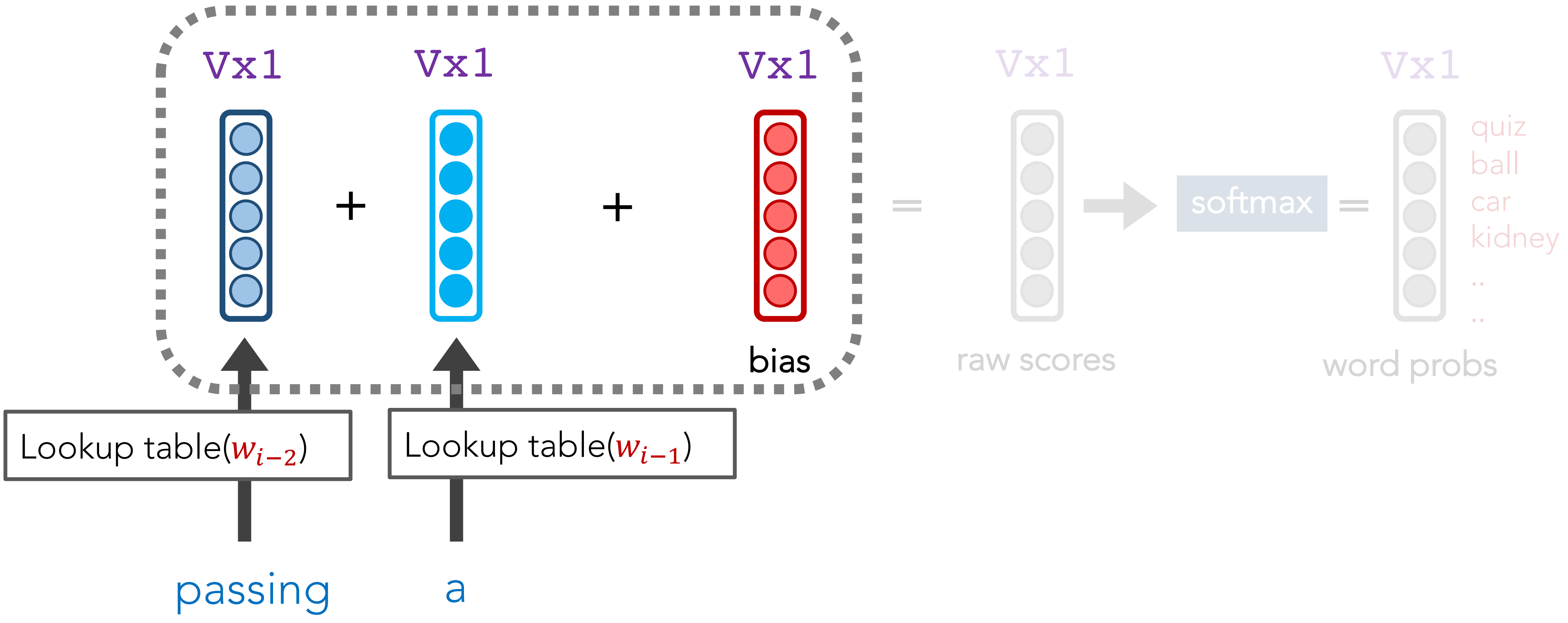
Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

Featurized Model

"passing a ______"

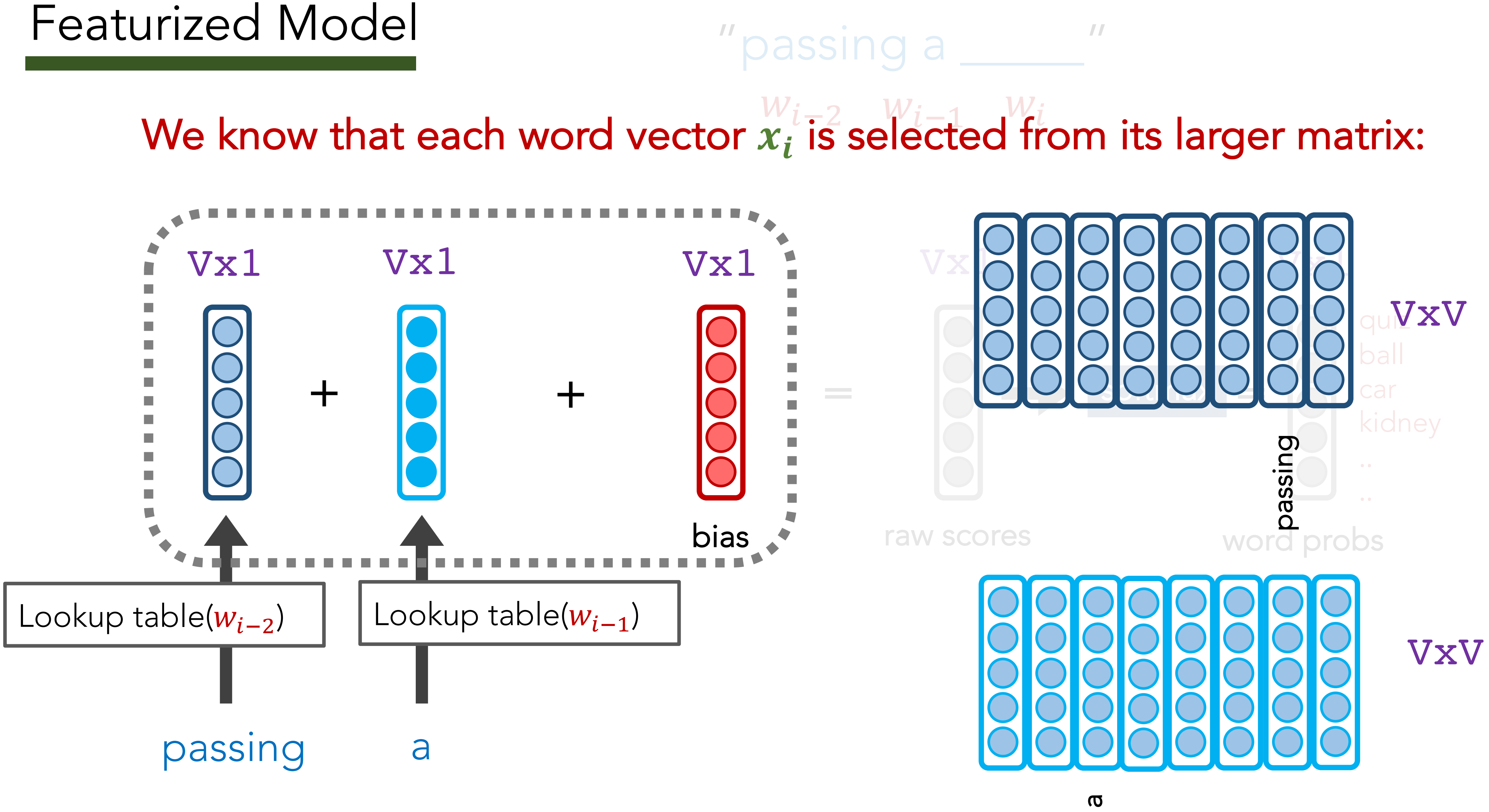
w_{i-2} w_{i-1} w_i

These are the only 3 components of our model.



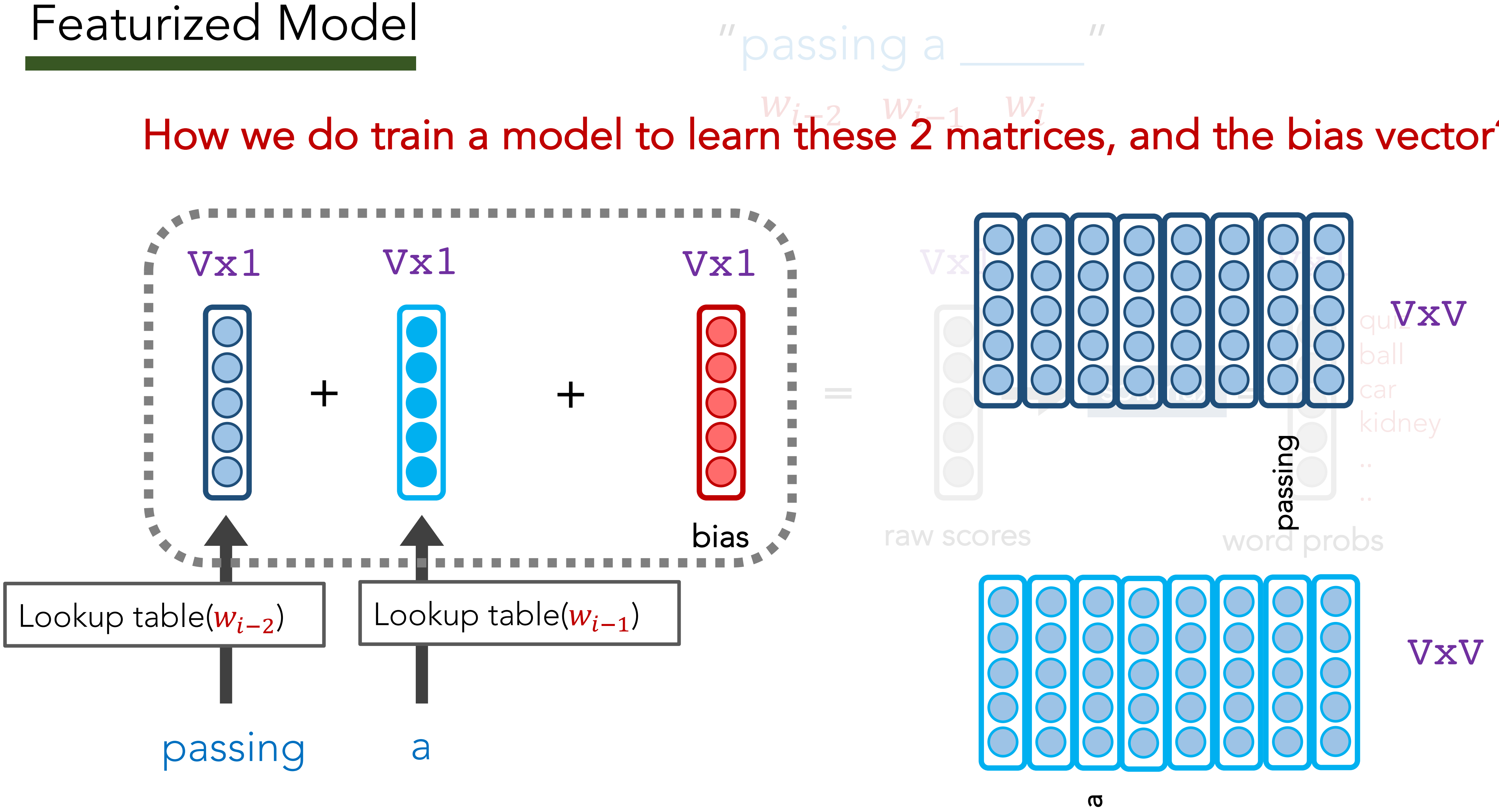
Featurized Model

We know that each word vector x_i is selected from its larger matrix:



Featurized Model

How we do train a model to learn these 2 matrices, and the bias vector?



Featurized Model

Train the model using gradient descent:

- Use our output probabilities
- Calculate the cross-entropy loss
- Use backprop to calculate gradients
- Update the 2 embedding matrices and bias via GD

Featurized Model

What does it mean to “update” these Embedding matrices? How? Where are the weights?!

See chalkboard for details.

Unknown Words

- We still need to handle UNK words. Always.
- Language is always evolving
- Zipfian distribution
- Larger vocabularies require more memory and compute time

How can we handle UNK words in a neural model?

Unknown Words

Common ways to amend the data:

- Frequency threshold (e.g., UNK <= 2)
- Remove bottom N%
- Represent each word as sub-words (e.g., byte-pair encodings)

Common neural modelling approaches:

- Add an UNK token to your vocabulary (just like for n-grams)

Evaluation

Very Important:

- Any given LM must be able to generate the **test set (**at least**)**. Otherwise, it cannot be fairly evaluated (OOV problem).
- When comparing multiple LMs to each other, their vocabularies must be the same (e.g., words, sub-words, characters).

Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

Remaining Issues

1. More context while avoiding sparsity, storage, and compute issues

2. No semantic information conveyed by counts (e.g., vehicle vs car)

New goals!

3. Cannot leverage non-consecutive patterns

Dr. West ____

Occurred 25 times

Dr. Cornell West ____

Occurred 3 times

4. Cannot capture combinatorial signals (i.e., non-linear prediction)

P(Chef cooked food)	high	P(Customer cooked food)	low
P(Chef ate food)	low	P(Customer ate food)	high

Slide adapted from or inspired by Graham Neubig's CMU NLP 2021

We clearly need:

- dense representations (i.e., $< |\boldsymbol{V}|$)
- leverage semantic information
- non-linear power

Neural models, here we come!

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Neural Network Motivation

Non-linear power: using non-linear activation functions can allow us to capture rich, combinatorial attributes of language

Neural Network Motivation

Curse of dimensionality:

- Say our vocab $|V| = 100{,}000$
- Naively modelling the joint probability of 10 consecutive, discrete random variables (e.g., words in a sentence) yields $100{,}000^{10} - 1 = 10^{50}$ free parameters.
- Word embeddings reduce the # of parameters and hopefully improve the model's ability to generalize

Bengio (2003)

1.1 Fighting the Curse of Dimensionality with Distributed Representations

In a nutshell, the idea of the proposed approach can be summarized as follows:

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

A Neural Probabilistic Language Model. Bengio et al. JMLR (2003)

Bengio (2003)

1.1 Fighting the Curse of Dimensionality with Distributed Representations

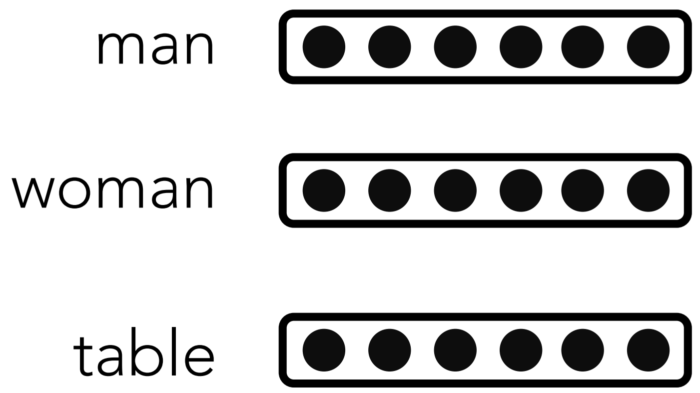
In a nutshell, the idea of the proposed approach can be summarized as follows:

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

A Neural Probabilistic Language Model. Bengio et al. JMLR (2003)

Bengio (2003)

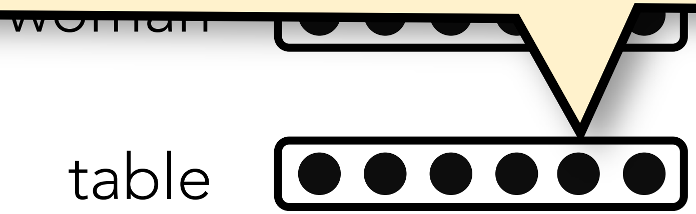
Simultaneously learn **the representation** and do the **modelling**!



Bengio (2003)

Simultaneously learn the representation and do the modelling!

- Each circle is a specific floating point scalar
- Words that are more semantically similar to one another will have embeddings that are proportionally similar, too

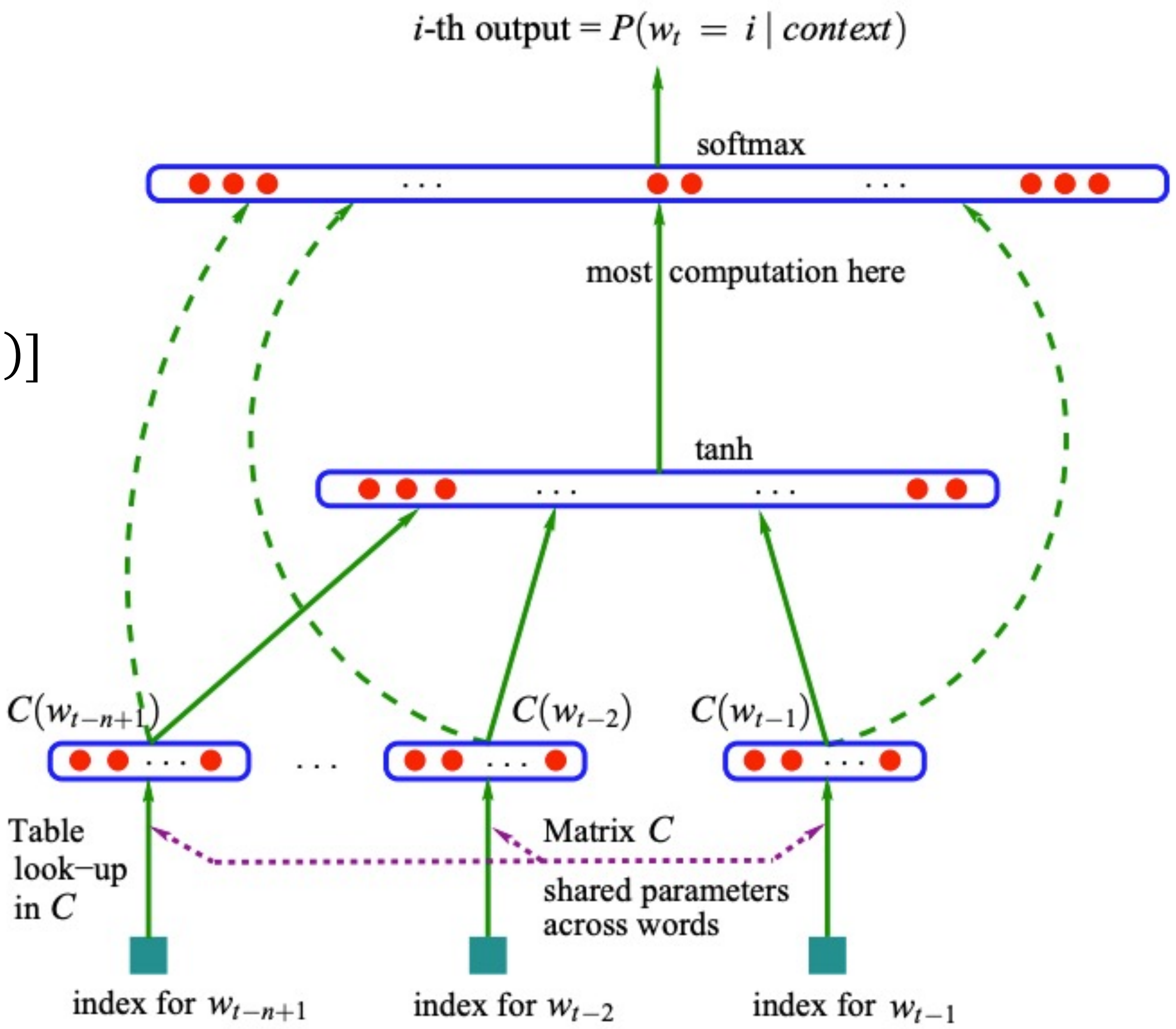


Bengio (2003)

$$y = b + Wx + U\tanh(d + Hx)$$

$$x = [C(w_{t-3}), C(w_{t-2}), C(w_{t-1})]$$

$$\theta = \{b, W, U, d, H, C\}$$



Bengio (2003)

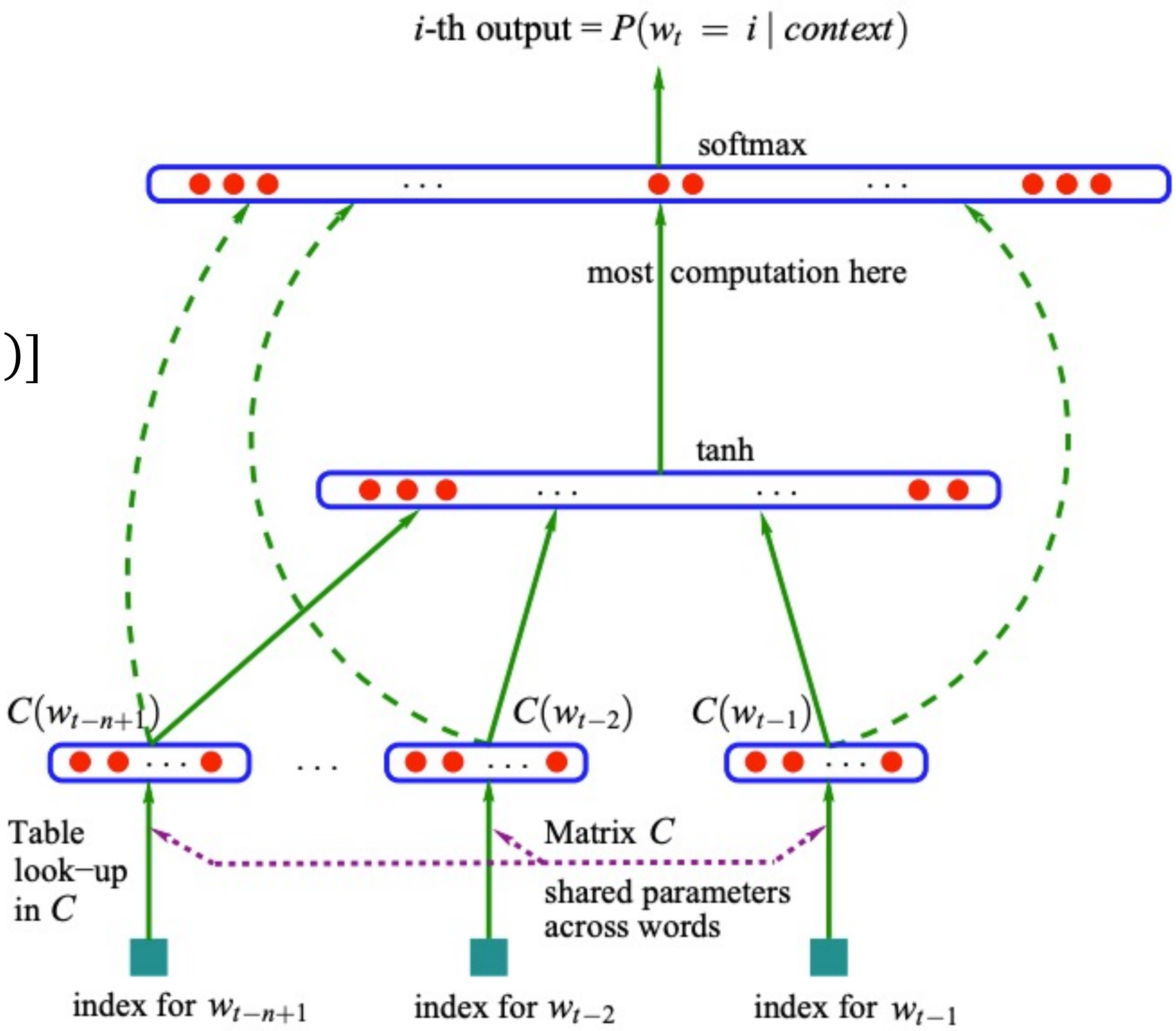
$$y = b + Wx + U\tanh(d + Hx)$$

$$x = [C(w_{t-3}), C(w_{t-2}), C(w_{t-1})]$$

$$\theta = \{b, W, U, d, H, C\}$$

$$\hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}.$$

predict the most likely word w, via softmax



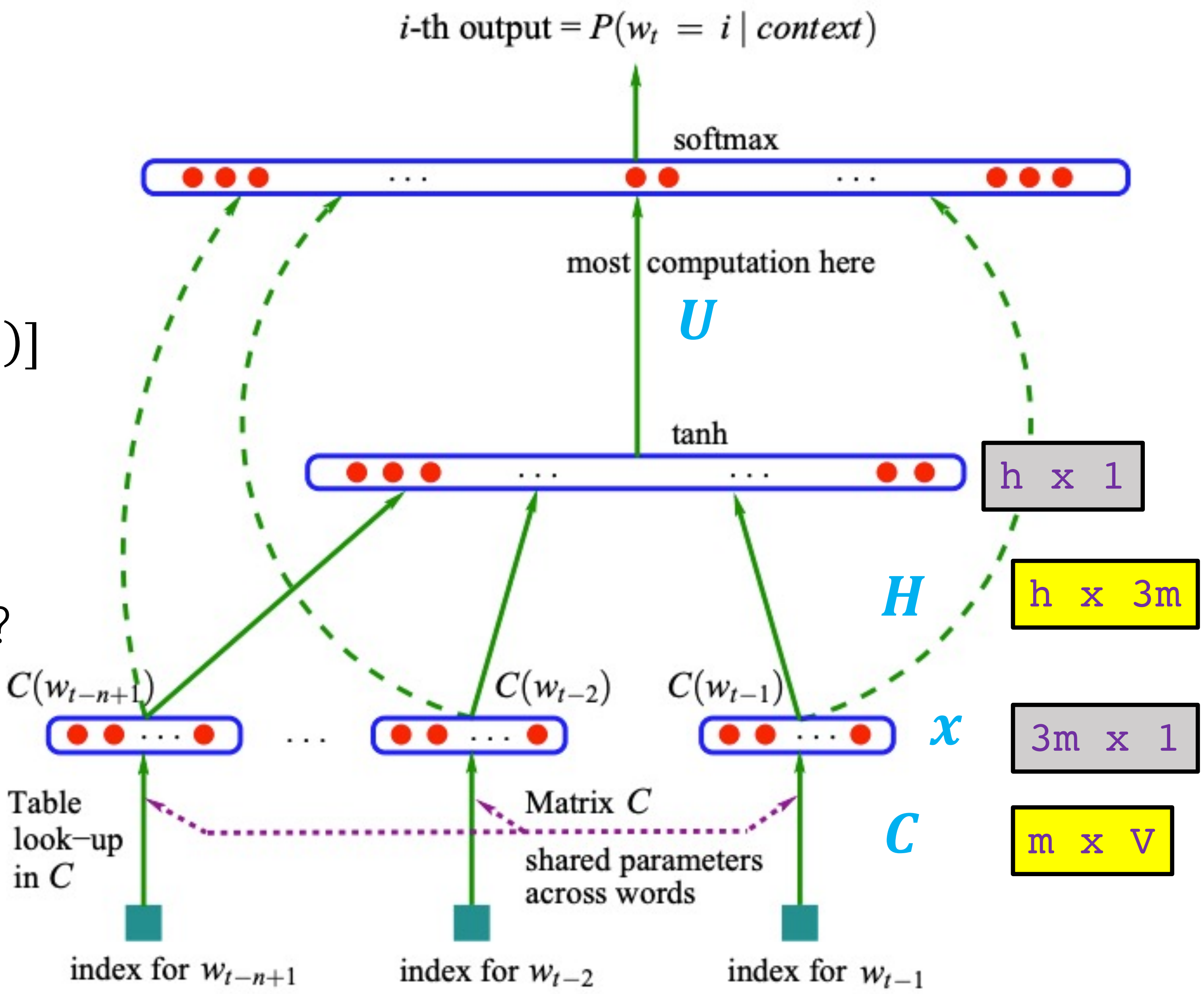
Bengio (2003)

$$y = b + Wx + U\tanh(d + Hx)$$

$$x = [C(w_{t-3}), C(w_{t-2}), C(w_{t-1})]$$

$$\theta = \{b, W, U, d, H, C\}$$

What are the dimensions of U?

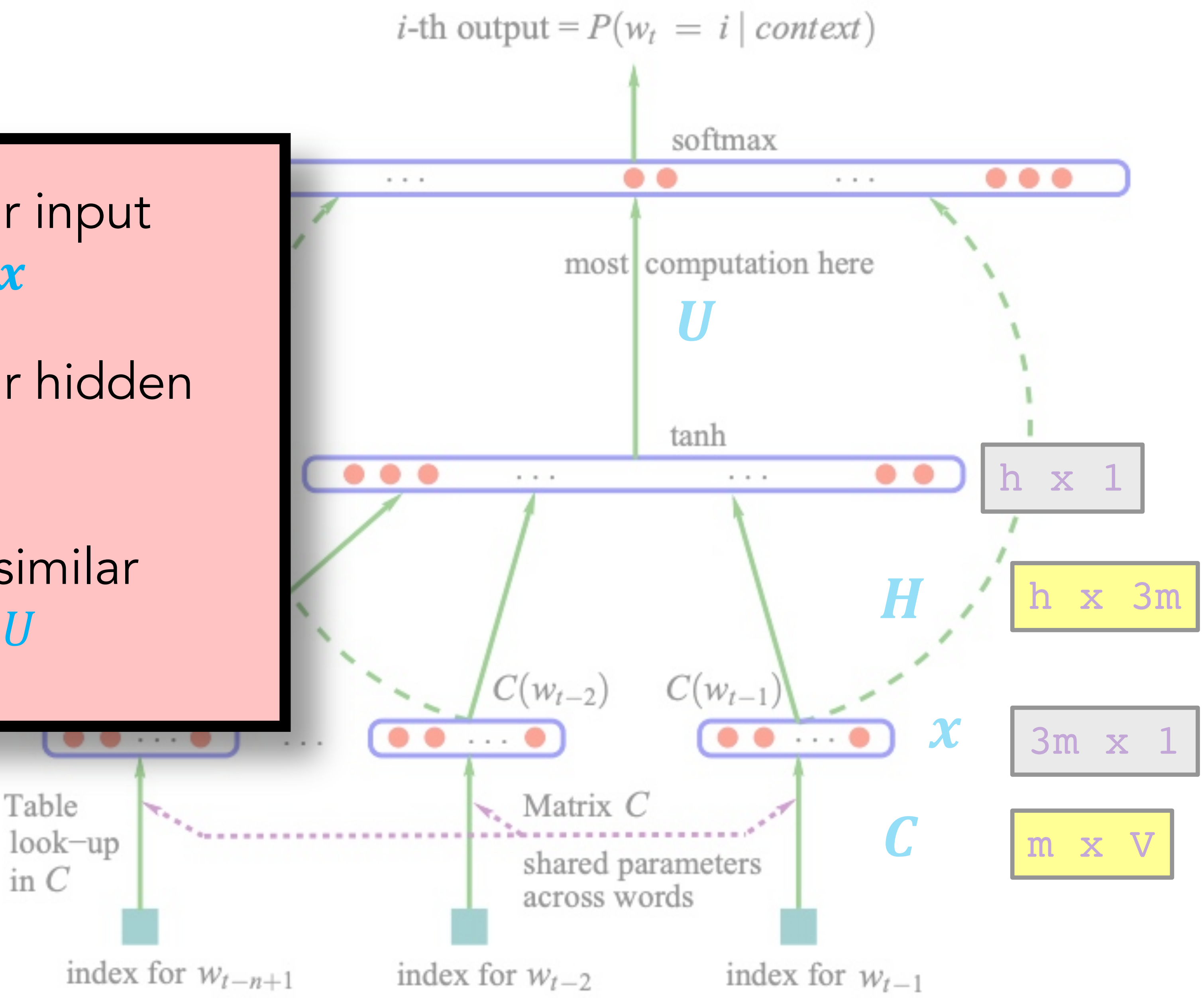


Bengio (2003)

Word embeddings: similar input words get similar vectors x

Similar **contexts** get similar hidden states

Similar **output words** get similar rows in the output matrix U



Bengio (2003)

FORWARD PASS

(a) Perform forward computation for the word features layer:
$x(k) \leftarrow C(w_{t-k})$,
$x = (x(1), x(2), \cdots, x(n-1))$

(b) Perform forward computation for the hidden layer:
$o \leftarrow d + Hx$
$a \leftarrow \tanh(o)$

(c) Perform forward computation for output units in the i-th block:
$s_i \leftarrow 0$
Loop over j in the i-th block

i. $y_j \leftarrow b_j + a.U_j$
ii. If (direct connections) $y_j \leftarrow y_j + x.W_j$
iii. $p_j \leftarrow e^{y_j}$
iv. $s_i \leftarrow s_i + p_j$

(d) Compute and share $S = \sum_i s_i$ among the processors. This can easily be achieved with an MPI `Allreduce` operation, which can efficiently compute and share this sum.

(e) Normalize the probabilities:
Loop over j in the i-th block, $p_j \leftarrow p_j / S$.

(f) Update the log-likelihood. If w_t falls in the block of CPU $i > 0$, then CPU i sends p_{w_t} to CPU 0. CPU 0 computes $L = \log p_{w_t}$ and keeps track of the total log-likelihood.

Bengio (2003)

BACKWARD PASS

BACKWARD/UPDATE PHASE, with learning rate ε.

(a) Perform backward gradient computation for output units in the i-th block:
clear gradient vectors $\frac{\partial L}{\partial a}$ and $\frac{\partial L}{\partial x}$.
Loop over j in the i-th block

i. $\frac{\partial L}{\partial y_j} \leftarrow 1_{j==w_t} - p_j$

ii. $b_j \leftarrow b_j + \varepsilon \frac{\partial L}{\partial y_j}$

If (direct connections) $\frac{\partial L}{\partial x} \leftarrow \frac{\partial L}{\partial x} + \frac{\partial L}{\partial y_j} W_j$

$\frac{\partial L}{\partial a} \leftarrow \frac{\partial L}{\partial a} + \frac{\partial L}{\partial y_j} U_j$

If (direct connections) $W_j \leftarrow W_j + \varepsilon \frac{\partial L}{\partial y_j} x$

$U_j \leftarrow U_j + \varepsilon \frac{\partial L}{\partial y_j} a$

(b) Sum and share $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial a}$ across processors. This can easily be achieved with an MPI `Allreduce` operation.

(c) Back-propagate through and update hidden layer weights:
Loop over k between 1 and h,

$\frac{\partial L}{\partial o_k} \leftarrow (1 - a_k^2) \frac{\partial L}{\partial a_k}$

$\frac{\partial L}{\partial x} \leftarrow \frac{\partial L}{\partial x} + H' \frac{\partial L}{\partial o}$

$d \leftarrow d + \varepsilon \frac{\partial L}{\partial o}$

$H \leftarrow H + \varepsilon \frac{\partial L}{\partial o} x'$

(d) Update word feature vectors for the input words:
Loop over k between 1 and $n-1$

$C(w_{t-k}) \leftarrow C(w_{t-k}) + \varepsilon \frac{\partial L}{\partial x(k)}$

where $\frac{\partial L}{\partial x(k)}$ is the k-th block (of length m) of the vector $\frac{\partial L}{\partial x}$.

Bengio (2003)

Train the model using gradient descent:

- Use our output probabilities
- Calculate the cross-entropy loss
- Use backprop to calculate gradients
- Update all weight matrices and bias via GD

SAME AS WE DO FOR ALL OF OUR NEURAL NETS

Bengio (2003)

RESULTS

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Bengio (2003) Remaining Issues

1. More context while avoiding sparsity, storage, and compute issues

2. No semantic information conveyed by counts (e.g., vehicle vs car)

New goals!

3. Cannot leverage non-consecutive patterns

Dr. West ____	Dr. Cornell West ____
Occurred 25 times	Occurred 3 times

4. Cannot capture combinatorial signals (i.e., non-linear prediction)

P(Chef cooked food)	high	P(Customer cooked food)	low
P(Chef ate food)	low	P(Customer ate food)	high

Bengio (2003) Remaining Issues

This was not the first neural language model, but it was the first, highly compelling model with great results (e.g., beating n-grams)

The softmax output layer is annoyingly slow

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Distributional Semantics

Distributional: meaning is represented by the contexts in which its used

"Distributional statements can cover all of the material of a language without requiring support from other types of information"

-- Zellig Harris. *Distributional Structure.* (1954)

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

-- John Rupert Firth. *A Synopsis of Linguistics Theory.* (1957)

Auto-regressive language models

I bought a _____

Good morning, _____

I got my ______

Masked language models

I bought a _____ from the bakery

Good morning, _____. Rise and shine!

I got my ______ license last week

word2vec

Two approaches:

1. Continuous Bag-of-Words (CBOW)
2. Skip-gram w/ negative sampling

word2vec: CBOW

Step 1: Iterate through your entire corpus, with sliding context windows of size $\boldsymbol{N}$ and step size $\mathbf{1}$

Step 2: Using all $2\mathrm{N}$ context words, except the center word, try to predict the center word.

Step 3: Calculate your loss and update parameters (like always)

word2vec: CBOW

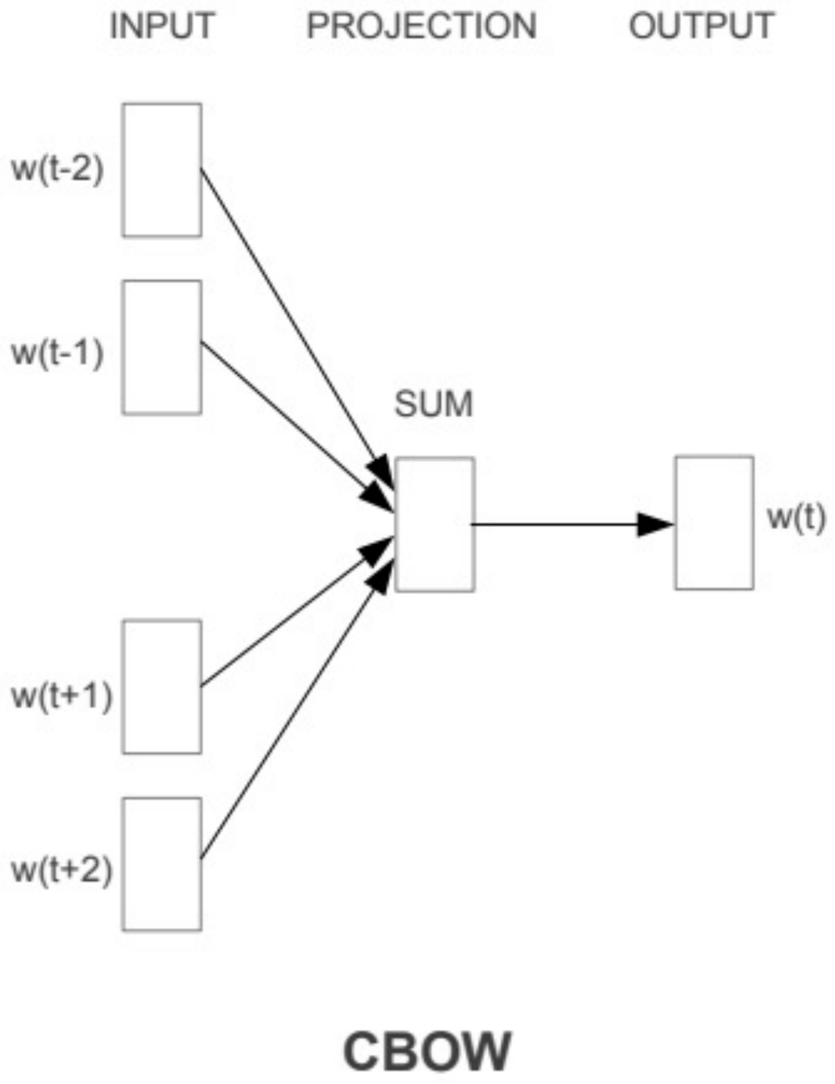


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

word2vec: CBOW

$y = U * \mathrm{sum}(Hx)$

$x = [w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$

N = # total context words

D = embedding size

V = # word types

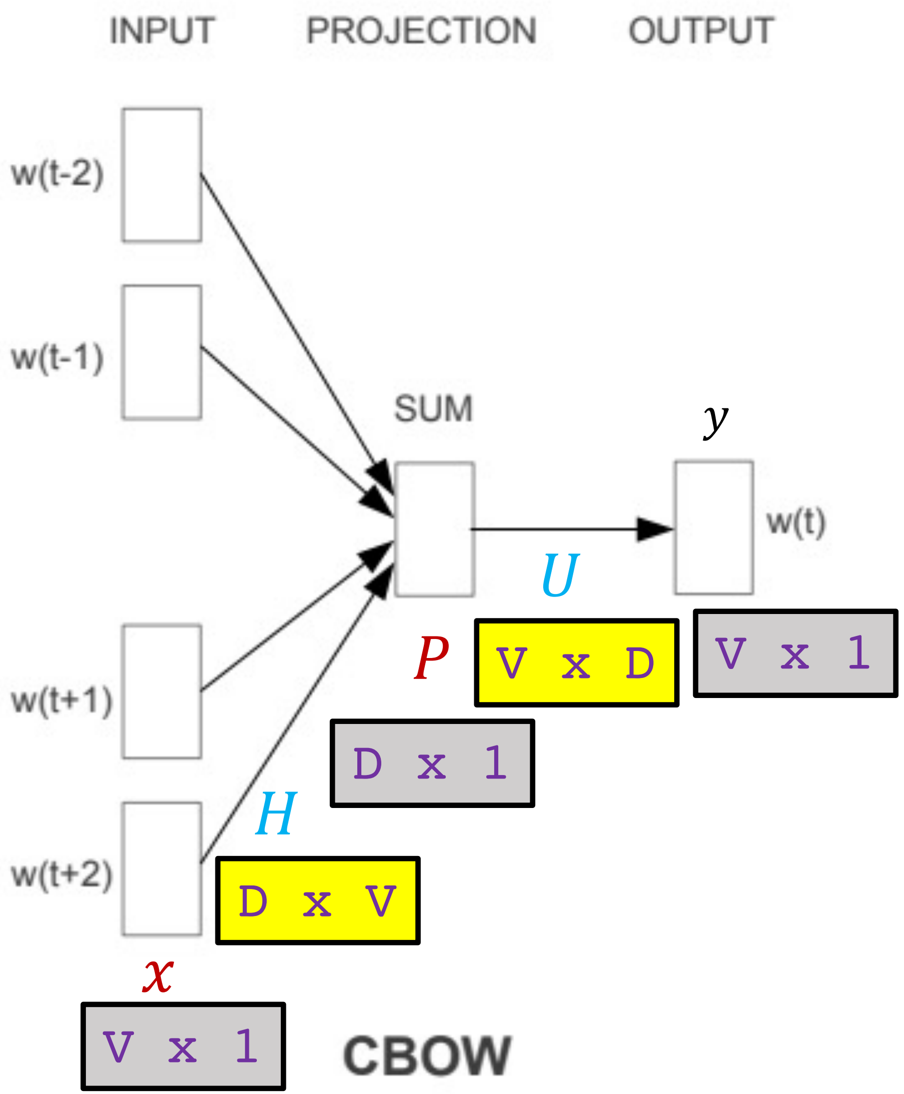


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

word2vec: CBOW

- Linear projection layer
- Non-linear output layer (softmax)
- Training in batches helps a lot

word2vec: skip-gram

Step 1: Iterate through your entire corpus, with sliding context windows of size $\boldsymbol{N}$ and step size $\mathbf{1}$

Step 2: Using the masked center word, try to predict all 2N center words.

Step 3: Calculate your loss and update parameters (like always)

word2vec: skip-gram

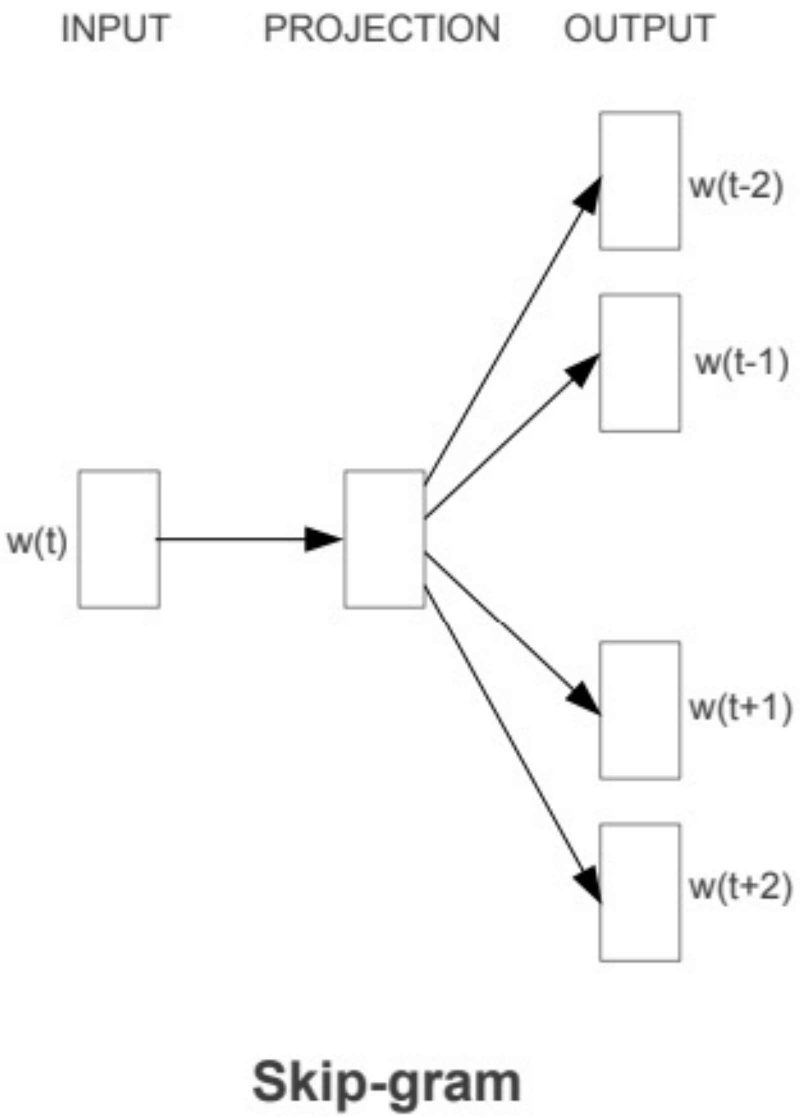


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

word2vec: skip-gram

In practice, this softmax is painfully slow.

Instead, flip the modelling to be pairs of words:

e.g., (center word, context word$_i$)

It would learn to always predict 1. So, probabilistically sample negative examples based on their frequencies

word2vec: results

- Smaller window sizes yield embeddings such that high similarity scores indicates that the words are *interchangeable*
- Larger window sizes (e.g., 15+) yield embeddings such that high similarity is more indicative of *relatedness* of the words.

word2vec: results

- Words that appear in the same contexts are forced to gravitate toward having the same embeddings as one another

- Imagine two words, w_1 and w_2, that never appear together, but they each, individually have the the exact same contexts with *other* words. w_1 and w_2 will have ~identical embeddings!

- "The" appears the most. What do you imagine its embedding is like?

word2vec results

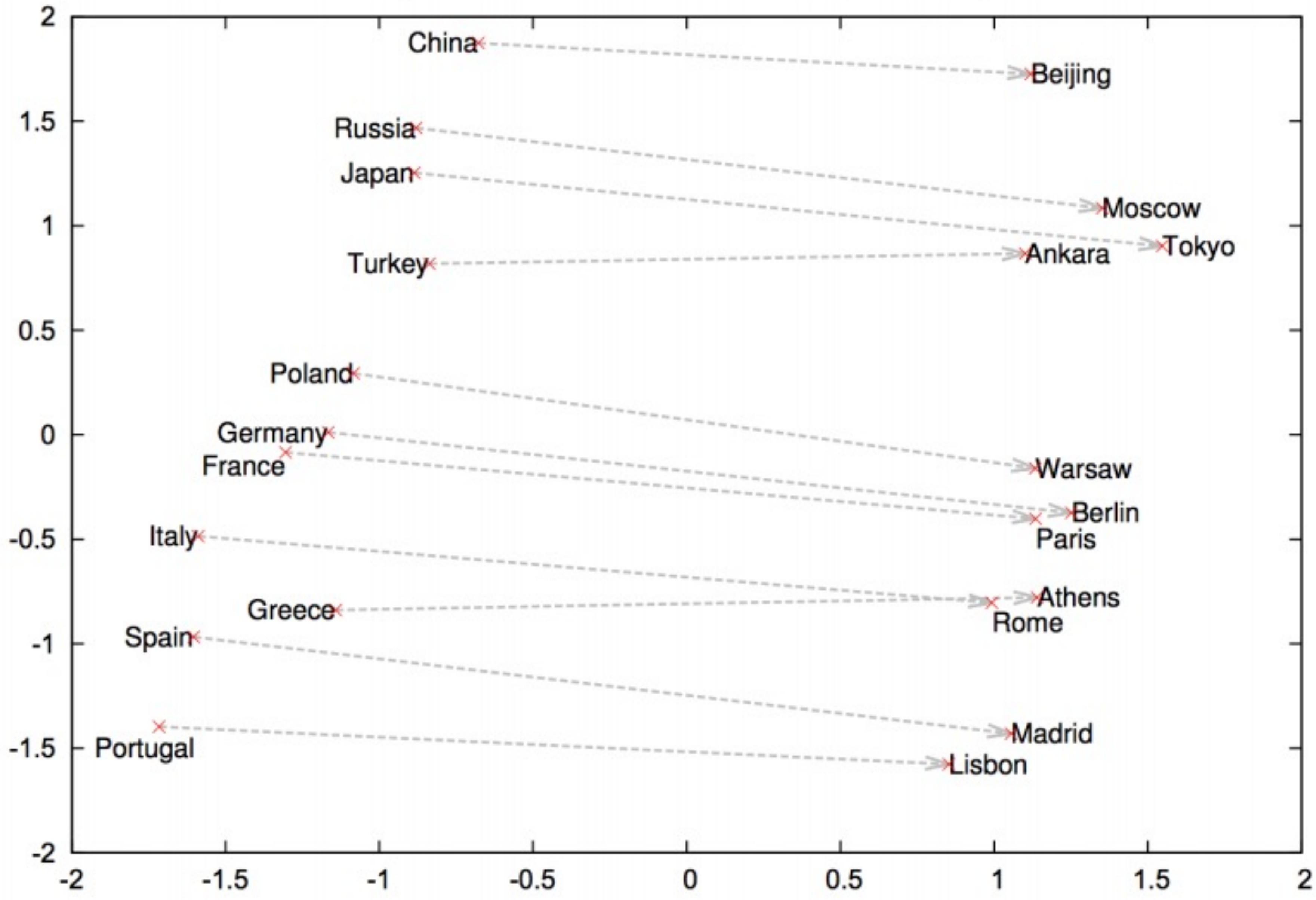


Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

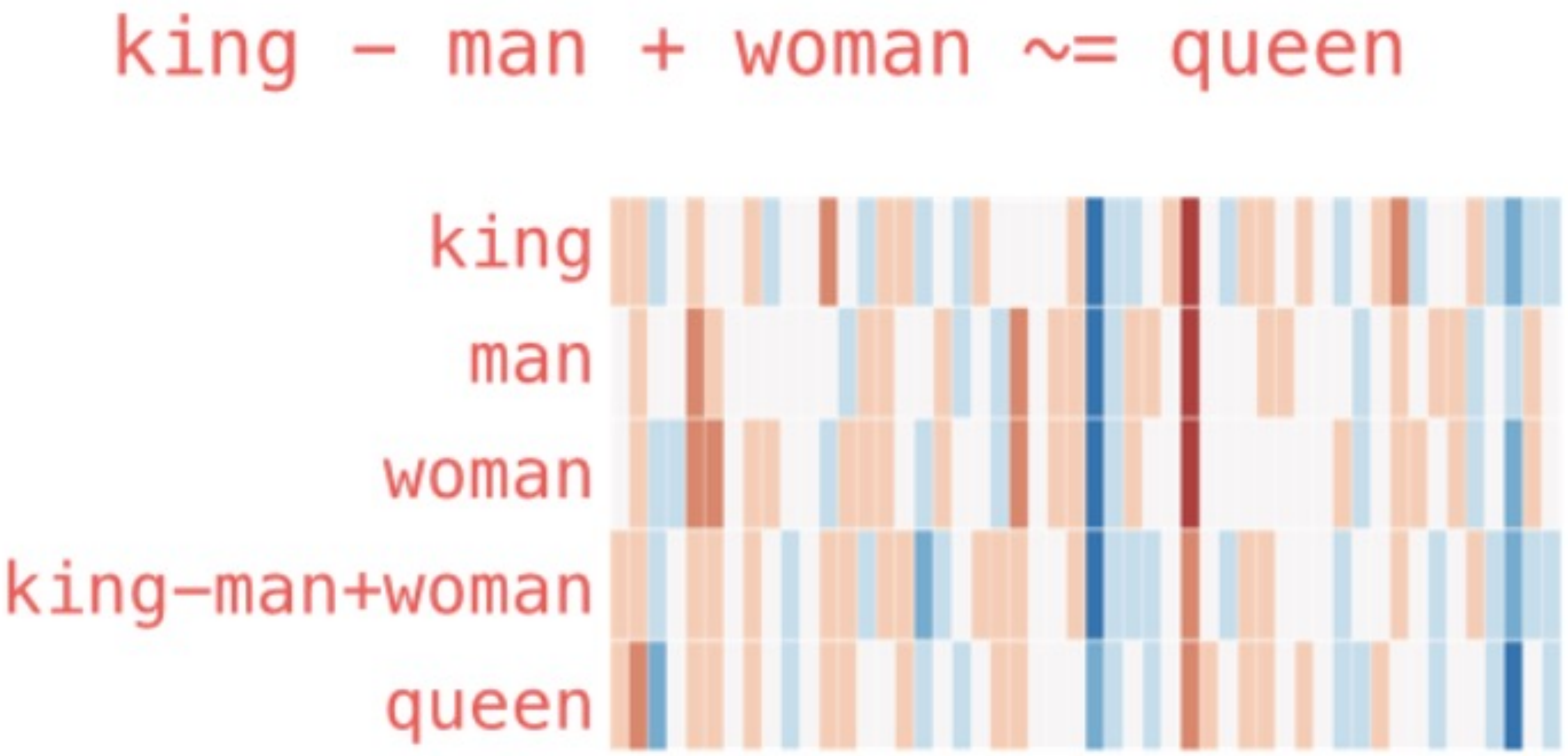
Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

word2vec results

Incredible finding!!



word2vec results

Disclaimer: As a heads-up, no models create embeddings such that the dimensions actually correspond to linguistic or real-world phenomenon.

The embeddings are often really great and useful, but no single embedding (in the absence of others) is interpretable.

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Evaluation

We cheated by looking ahead, so it's unfair to measure perplexity against n-gram or other auto-regressive LM

Intrinsic evaluation:

- Word similarity tasks
- Word analogy tasks

Extrinsic evaluation:

- Apply to downstream tasks (e.g., Natural language inference, entailment, question answering, information retrieval)

Evaluation

Word Similarity

SimLex-999

word1	word2	SimLex999
absence	presence	0.4
absorb	learn	5.48
absorb	possess	5
absorb	withdraw	2.97
abundance	plenty	8.97
accept	reject	0.83
accept	acknowled	6.88
accept	believe	6.75
accept	deny	1.75
accept	forgive	3.73

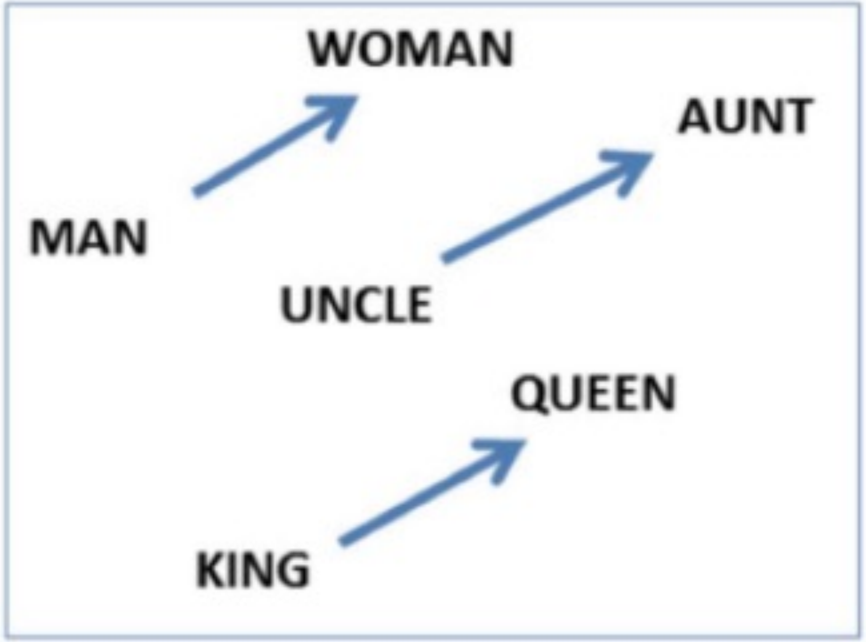
Slide adapted from or inspired by Sam Bowman's NYU NLP 2021

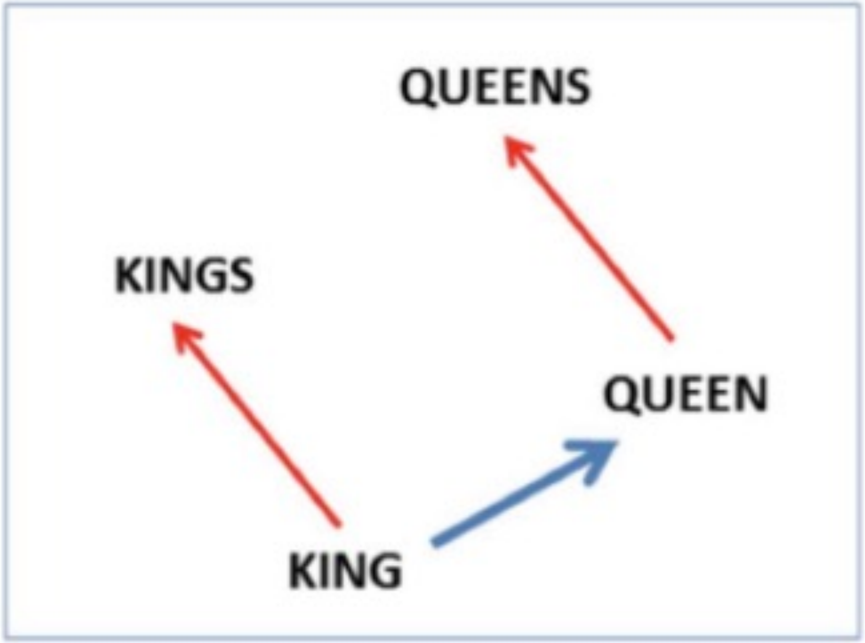
Evaluation

Word Analogy

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Outline

- Featurized, Linear Model
- Neural Models
  - Bengio (2003)
  - word2vec (2013)
- Evaluation
- Remaining challenges

Remaining Challenges

- Still can't handle long-range dependencies.
- Each decision is independent of the previous!
- Having a small, fixed window that repeats is a bit forced and awkward