An Introduction to Natural Language Processing

Lawrence Carin

Duke University

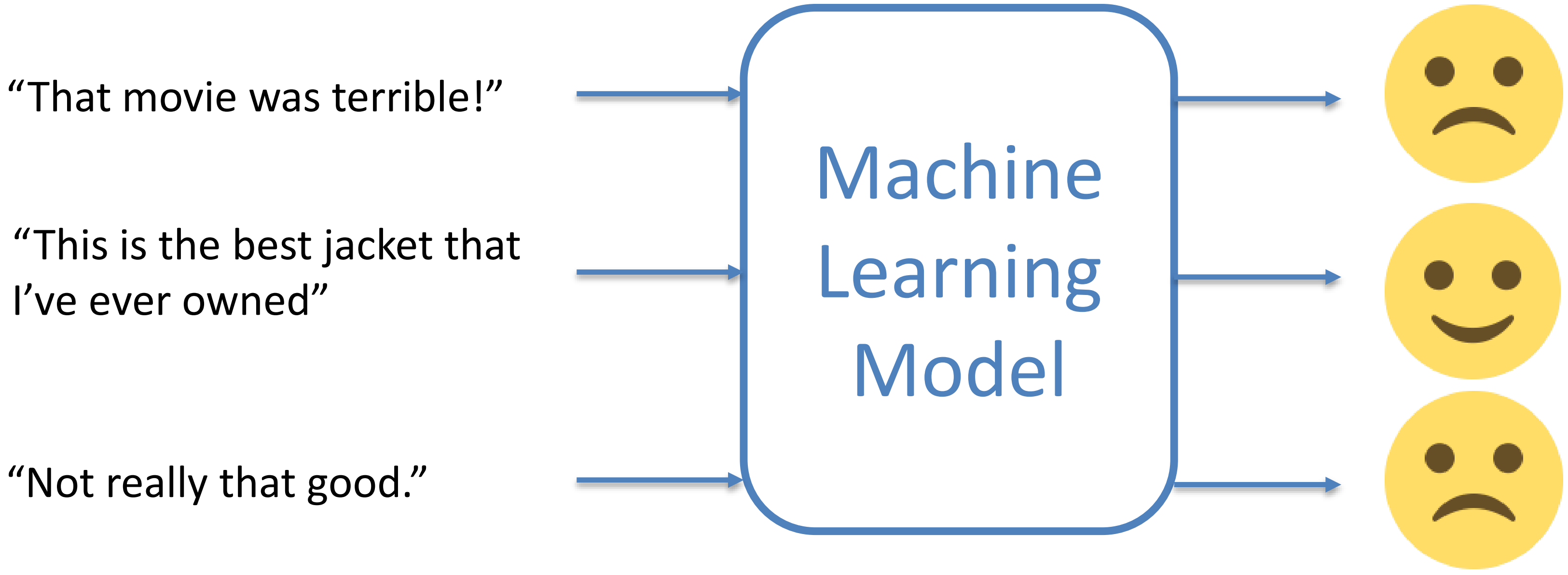
Electrical & Computer Engineering Department

Lecture 1

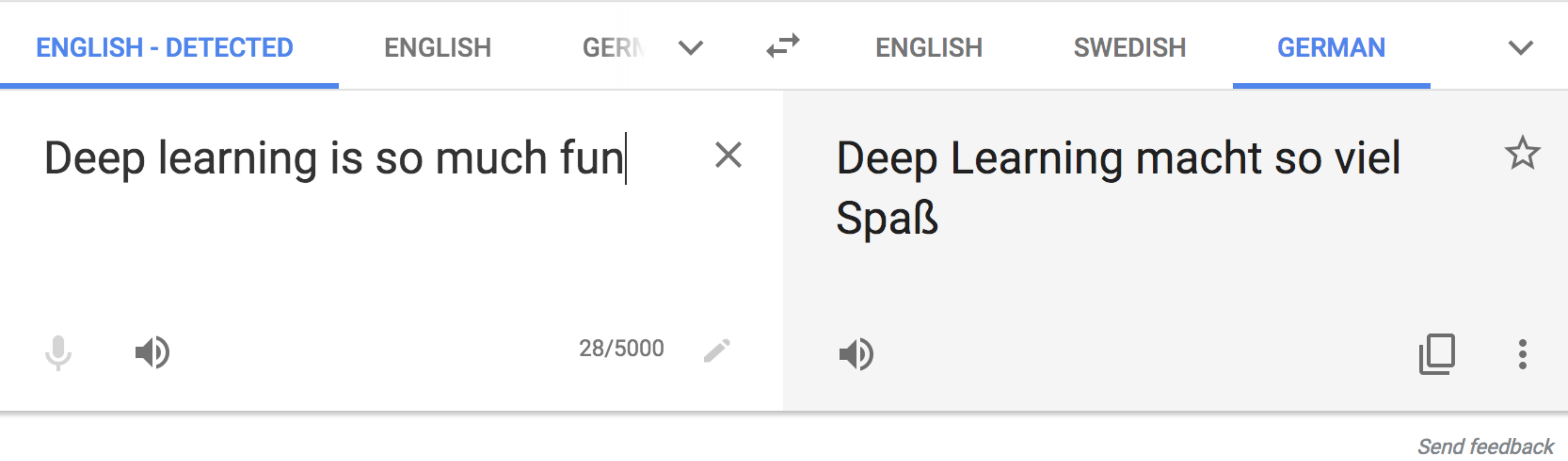
What is happening today?

- We are first going over the basics of “natural language processing”
- Want to be able to automatically make sense of written language (whether “natural” or not!)
- Later sessions will go through specific case studies
- First, what can we do with natural language processing?

Sentiment Analysis



Text Translation



translate.google.com

Automatic Text Synthesis

A	the service was great, the receptionist was very friendly and the place was clean, we waited for a while, and then our room was ready .
●	same with all the other reviews, this place is a good place to eat, i came here with a group of friends for a birthday dinner, we were hungry and decided to try it, we were seated promptly.
●	this place is a little bit of a drive from the strip, my husband and i were looking for a place to eat, all the food was good, the only thing i didn t like was the sweet potato fries.
●	this is not a good place to go, the guy at the front desk was rude and unprofessional, it s a very small room, and the place was not clean.
●	service was poor, the food is terrible, when i asked for a refill on my drink, no one even acknowledged me, they are so rude and unprofessional.
B	how is this place still in business, the staff is rude, no one knows what they are doing, they lost my business .

Automatic Image Captioning

a cow is standing in front of a store

a group of elephants standing next to each other

a table that has wooden spoons on it

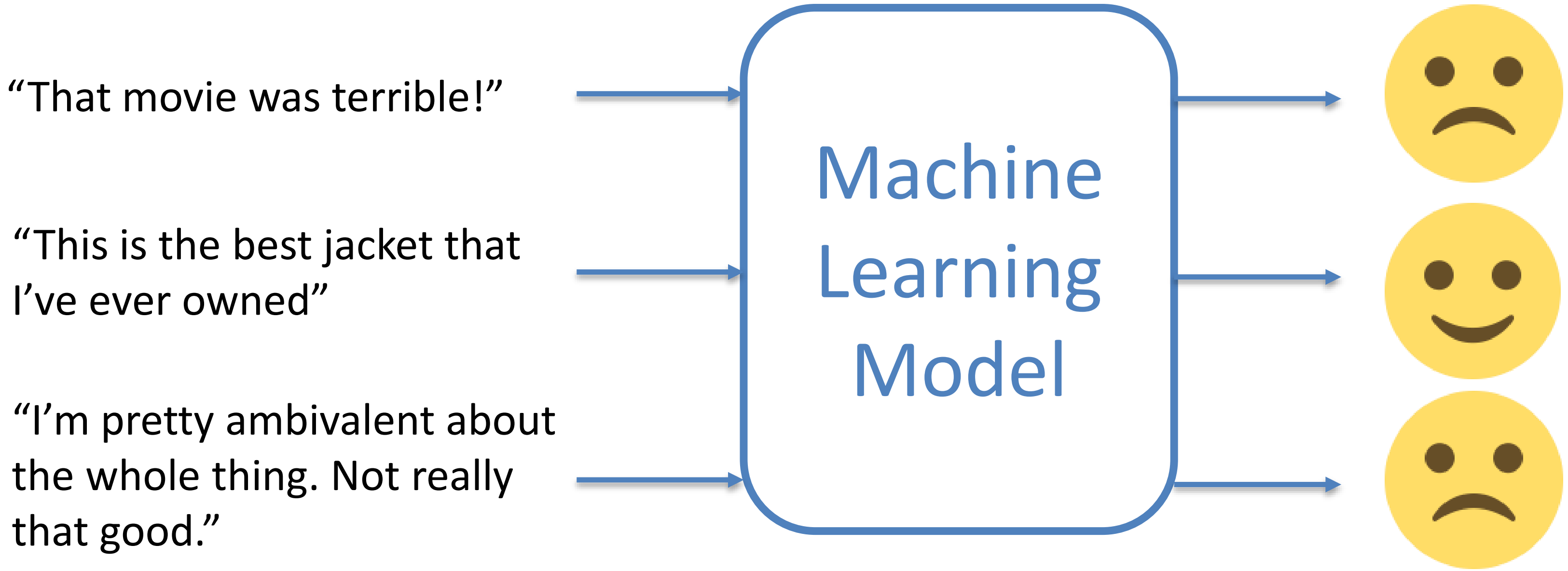
a cat is eating some kind of food

a bunch of bananas are sitting on a table

a motorcycle is parked next to a window

HOW DOES THIS FIT IN OUR PREVIOUS FRAMEWORK?

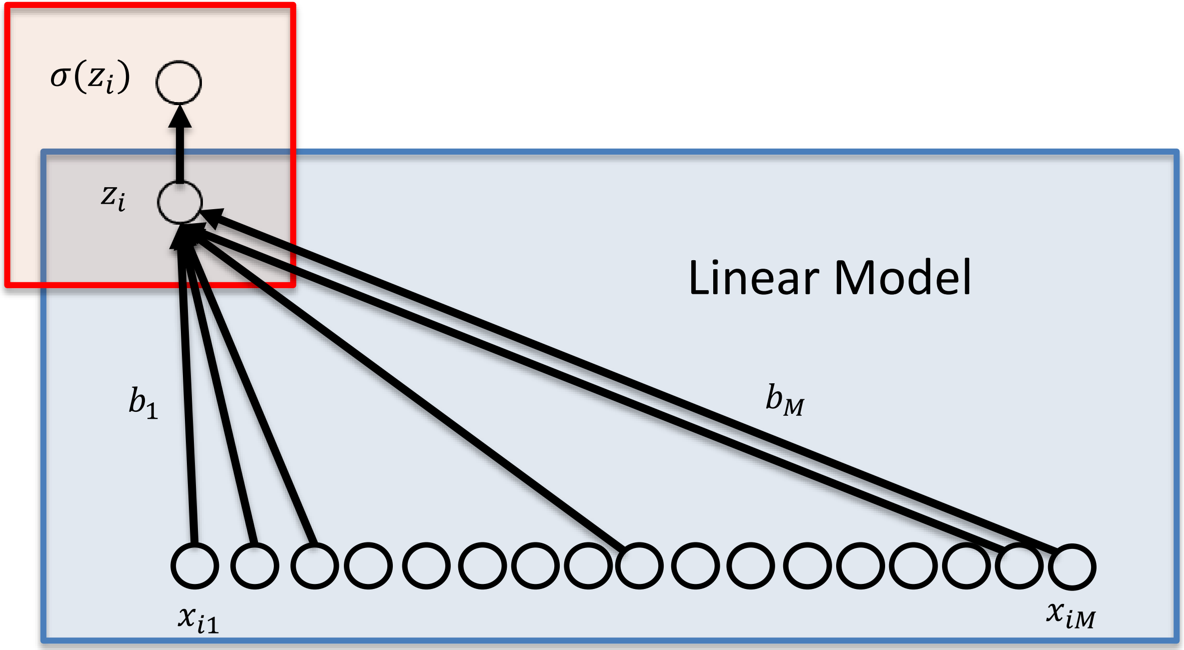
Sentiment Analysis



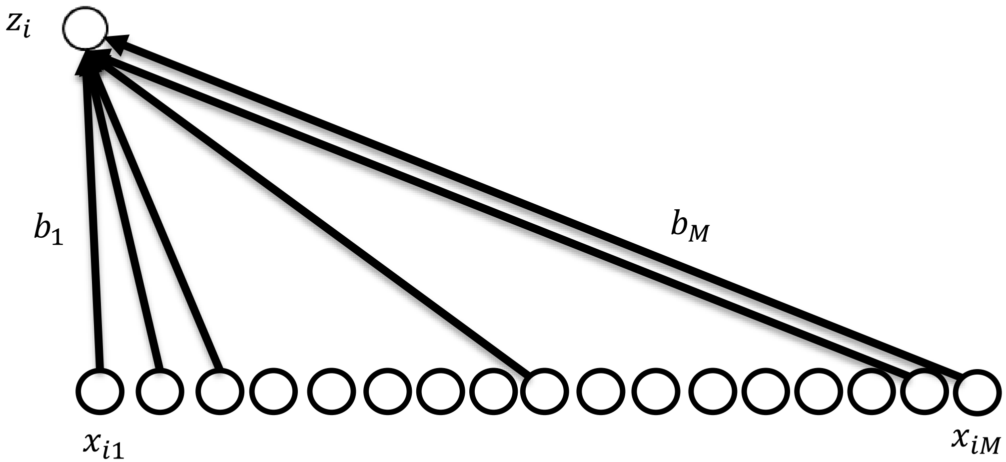
This is a **binary classification**, just like logistic regression!

Revisiting Logistic Regression

Convert to Probability



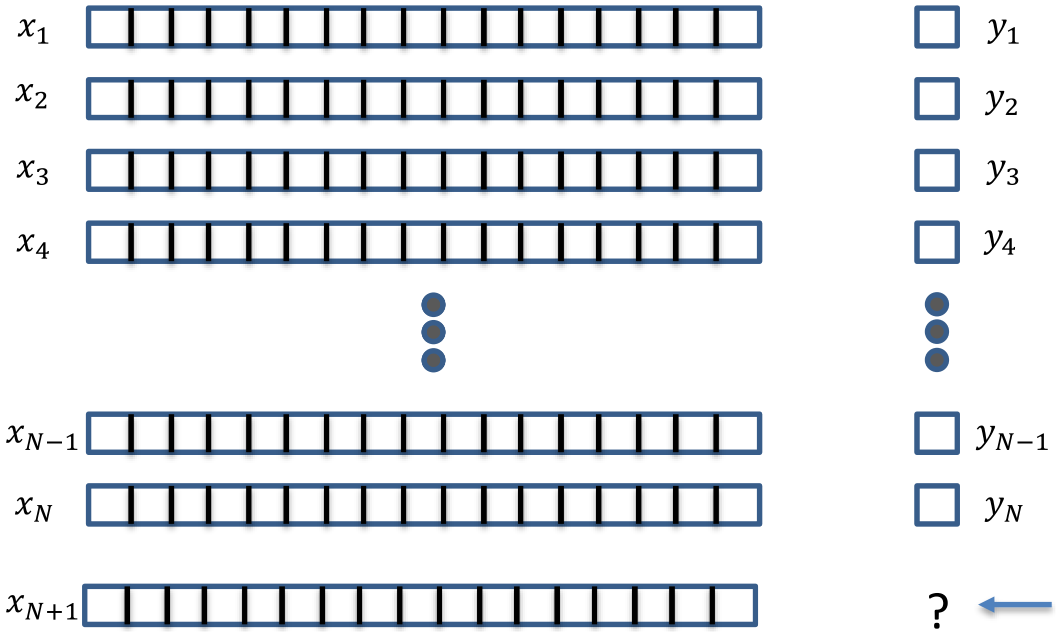
Revisiting our Linear Predictive Model



$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

Big issue: we do not have a vector of features…

Our Previous Training Set Setup



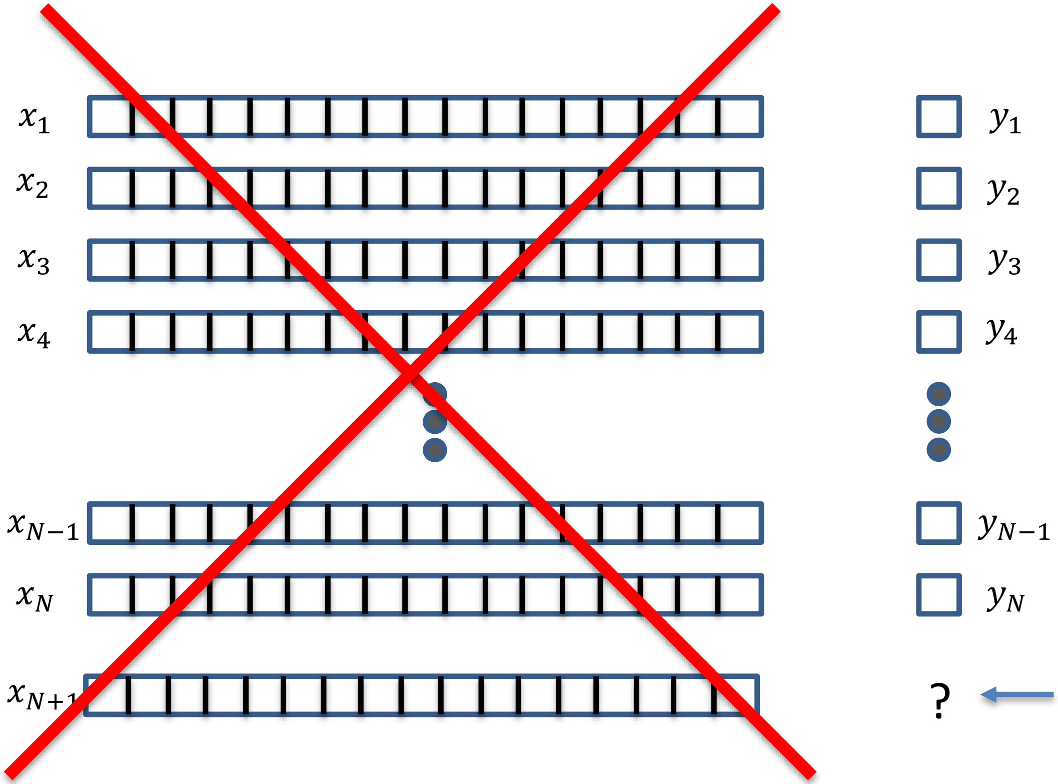
Start by limiting y to a binary outcome:

- False/True
- 0/1

New Data

Want to learn how to *predict* outcome

Our Previous Training Set Setup



Start by limiting y to a binary outcome:

- False/True
- 0/1

New Data

Want to learn how to *predict* outcome

Our Training Set is Text

x_1	"That movie was terrible!"	☐	y_1
x_2	"This is the best jacket…"	☐	y_2
x_3	"Most awful thing ever…"	☐	y_3
x_4	"One of the most amazing…"	☐	y_4
	⋮	⋮	
x_{N-1}	"Will never buy again…"	☐	y_{N-1}
x_N	"I'm rejuvenated…"	☐	y_N

Start by limiting y to a binary outcome:

- False/True
- 0/1

New Data → x_{N+1} "This class is great!" ? ← Want to learn how to *predict* outcome

Algorithms work on numbers

- A simple question:

“Can we convert our text to a vector or sequence of numbers?”

- If yes, we can start using our previous methodology!

First approach: Counting

- Let's define a dictionary of vocabulary words
  - Each word is assigned an index
- Count the number of times that each word appears in each text example

"That was a good, really good, game last night"

↓ Counting word occurrences

good:2,
that:1,
was:1,
a:1,
really:1,
game:1,
last:1,
night:1

Counting the number of occurrences

Original documents

perspective identifying tumor suppressor genes in human...
letters global warming report leslie roberts article global....
research news a small revolution gets under way the 1990s....
a continuing series the reign of trial and error draws to a close...
making deep earthquakes in the laboratory lab experimenters...
quick fix for freeways thanks to a team of fast working...
feathers fly in grouse population dispute researchers...

. . .

Word index and counts

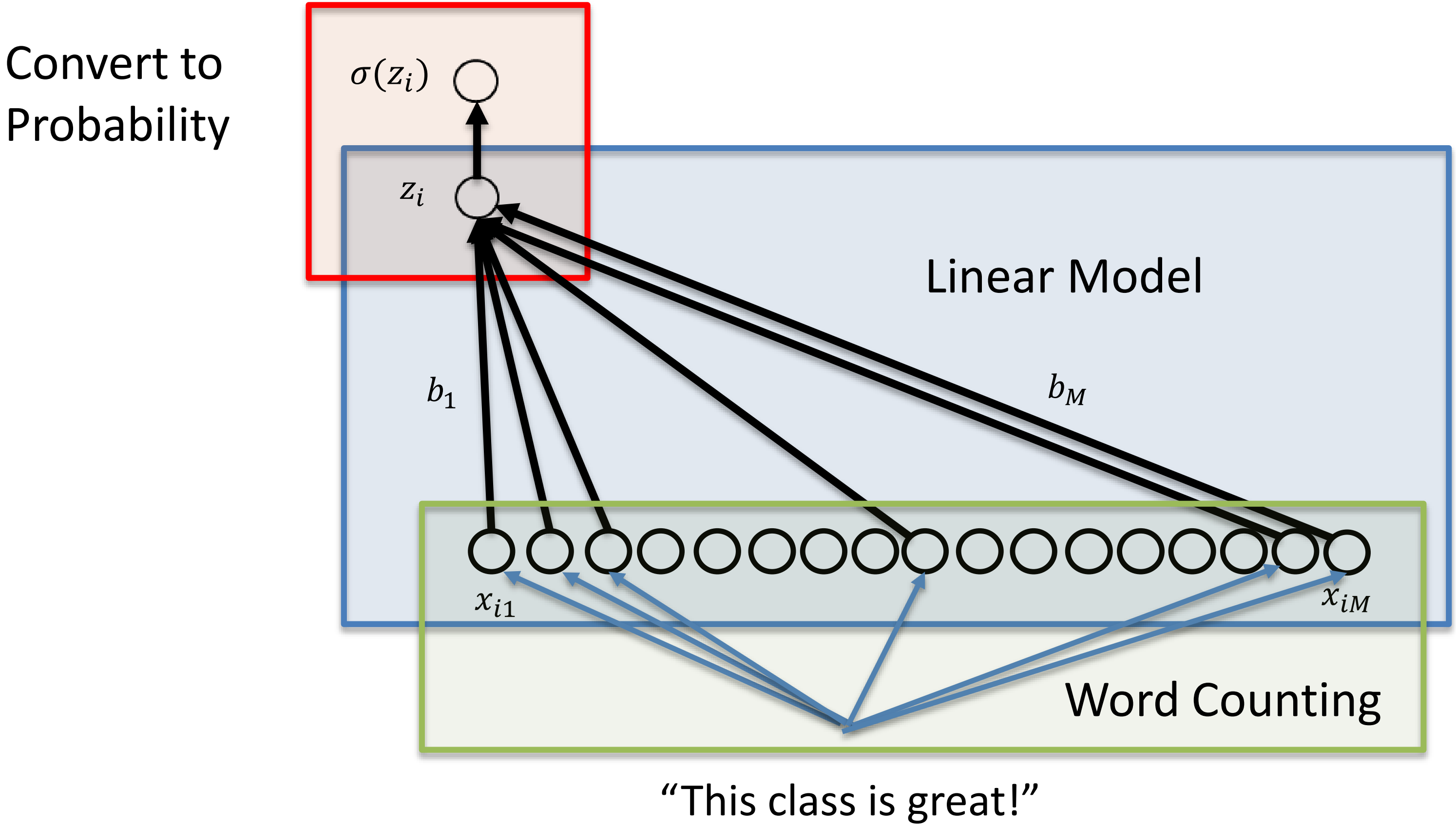
1897:1 1467:1 1351:1 731:2 800:5 682:1 315:6 3668:1 14:1
4261:2 518:1 271:6 2734:1 2662:1 2432:1 683:2 1631:7
2724:1 107:3 518:1 141:3 3208:1 32:1 2444:1 182:1 250:1
2552:1 1993:1 116:1 539:1 1630:1 855:1 1422:1 182:3 2432:1
1372:1 1351:1 261:1 501:1 1938:1 32:1 14:1 4067:1 98:2
4384:1 1339:1 32:1 4107:1 2300:1 229:1 529:1 521:1 2231:1
569:1 3617:1 3781:2 14:1 98:1 3596:1 3037:1 1482:12 665:2

. . .

The format is:
index:count

Each section of text is now a vector, where the m^{th} entry is the number of times the m^{th} dictionary word occurred in that example.

Revisiting Logistic Regression

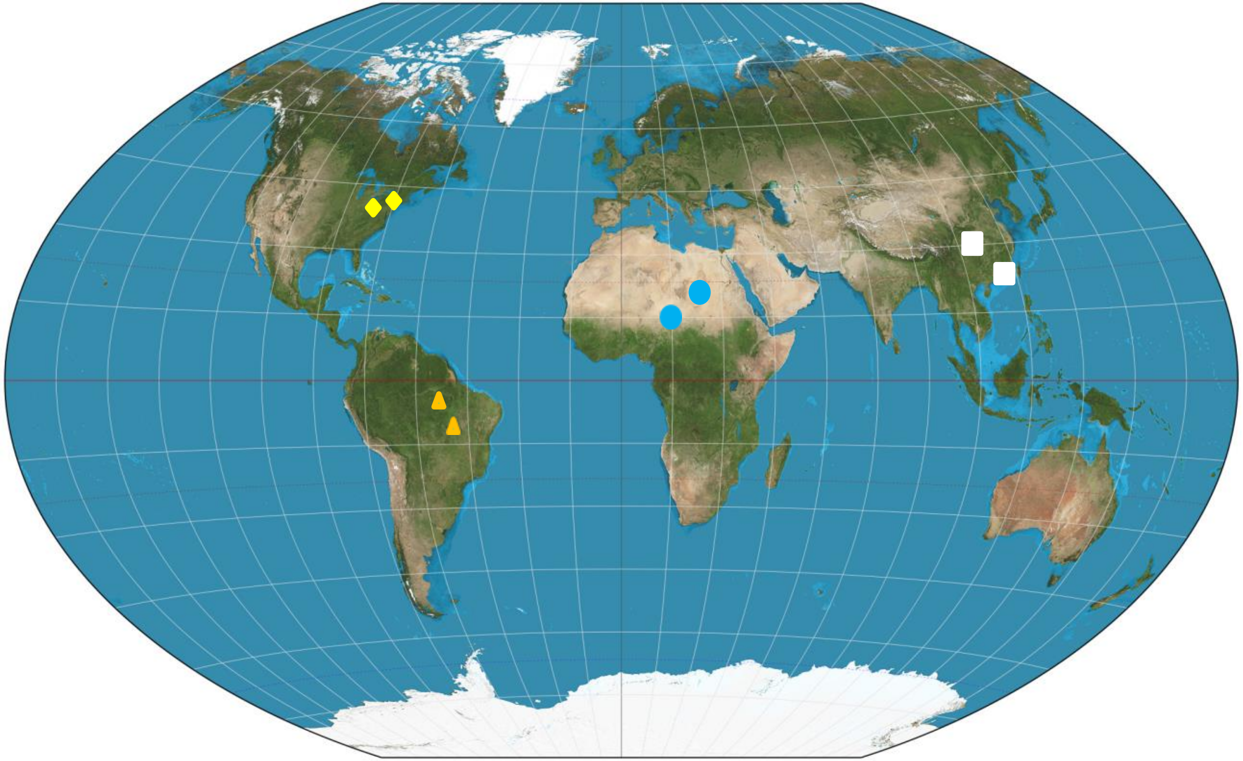


Quick Comments

- This word counting strategy can work decently well in practice
- Can easily put in our deep models instead
- But we’re not really processing “natural” language
  - Don’t capture word relationships (e.g. “not bad”=“good”)
  - Don’t capture word similarity (“cat” and “dog” are both pets, “lawyer” and “attorney” are synonyms)
- Can we capture these types of relationships?

WORD EMBEDDINGS

Similarities Based on Proximity



- Points that are nearby spatially are likely to have similar attributes
- May think of each location is a point in a 2D space (lat,long)
- Measure similarity between places by their distance in 2D space

Map Each Word in Vocabulary to a Point in Space

- Each word in a vocabulary is mapped to a point in a 2D space
- The closer words are in the mapping, the more related/synonymous they are
- The algorithm will learn the 2D point at which to place each word

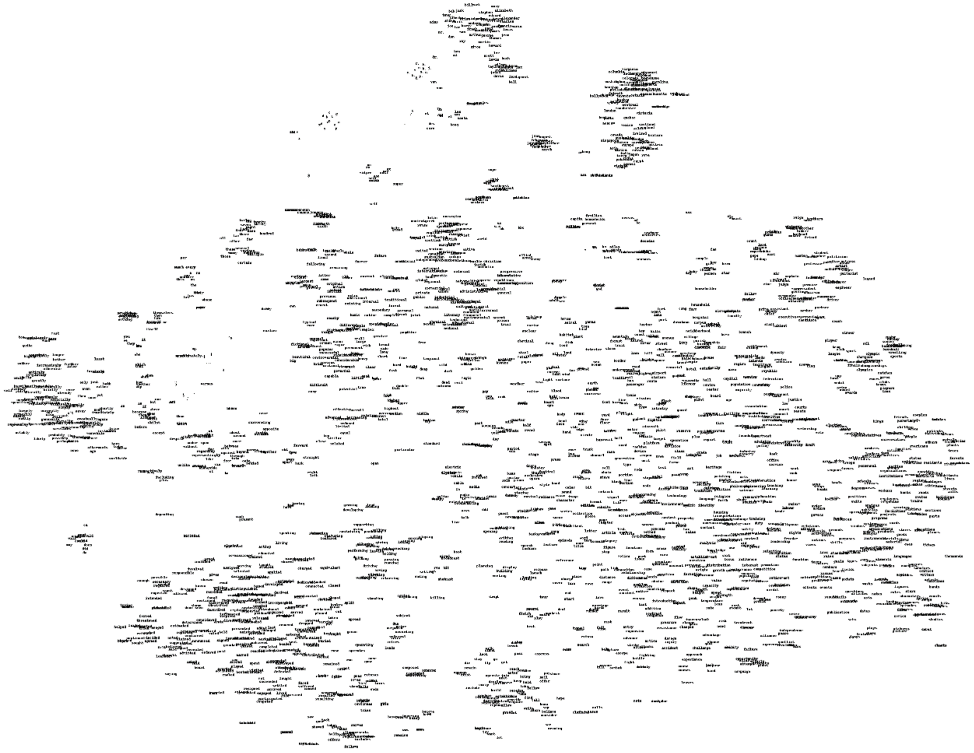
	lawyer 1	attorney 2	penguin 3	apple 4	●	●	●	V
longitude	78.8986	79.0558	135.0000	74.0060				
latitude	35.9940	35.9132	82.8628	40.7128				

vocabulary words

Example Word Geography

Here we show the learned geography of many different vocabulary words.

Too many words here to see! Let's zoom in on a smaller section.

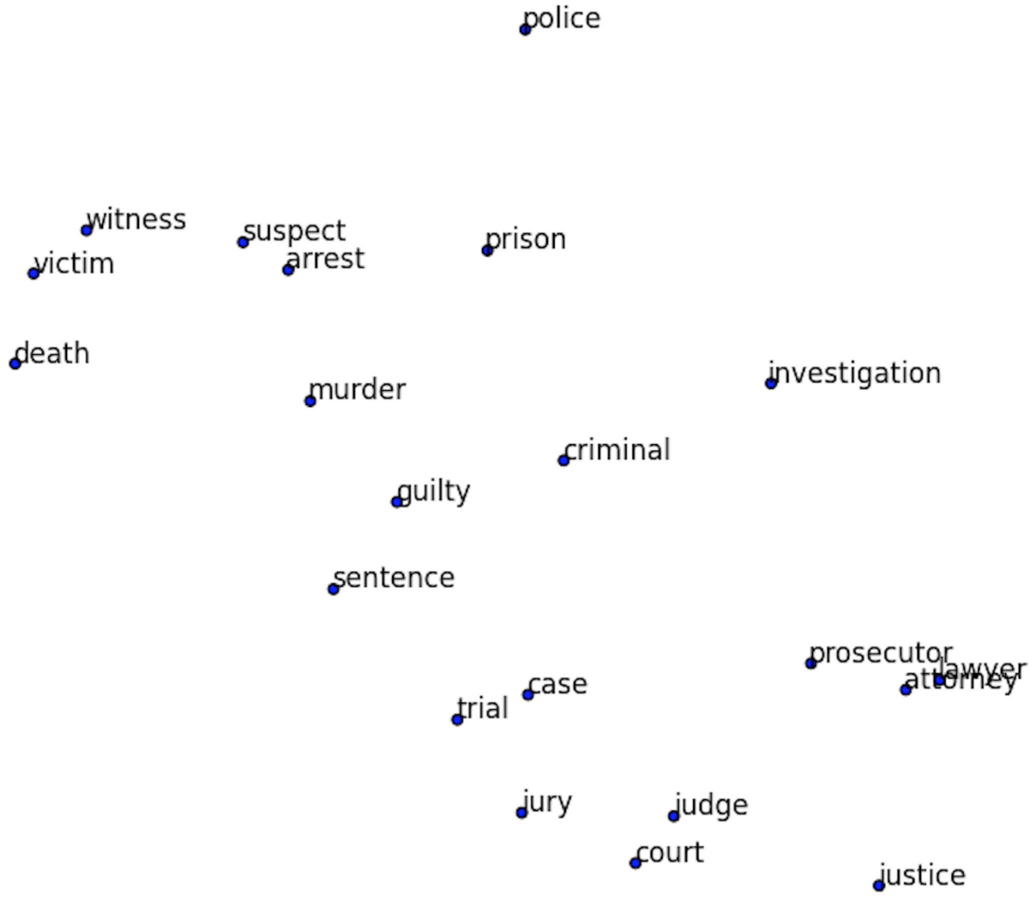


Example Word Geography

If we zoom in on a small region of our word map, it’s all related words.

Note the similarity of all the words as a whole, but also of the individual neighbors.

“Lawyer” and “attorney” are nearly identical in space!

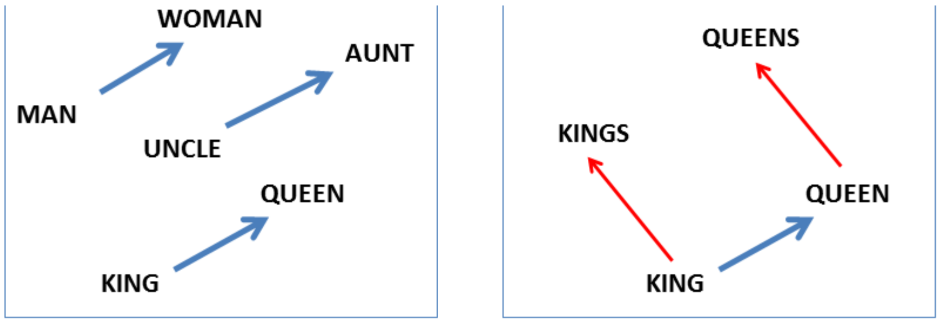


Learn Geographical Relationships

The relationship between words can be maintained, we can do mathematical operations on these word vectors.

Add the same vector distance between man and woman will convert uncle to aunt and king to queen.

Plural relationships are also maintained.



Word to Vector (Word2Vec)

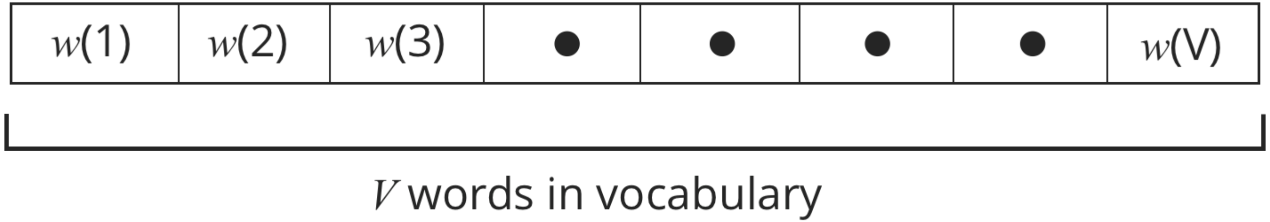
- The key to neural language models involves mapping each word to a vector
- Such word vectors are also called word "embeddings"
- Let $w(i)$ represent the ith word in the vocabulary

	lawyer 1	attorney 2	penguin 3	apple 4	•	•	•	V
longitude	78.8986	79.0558	135.0000	74.0060				
latitude	35.9940	35.9132	82.8628	40.7128				

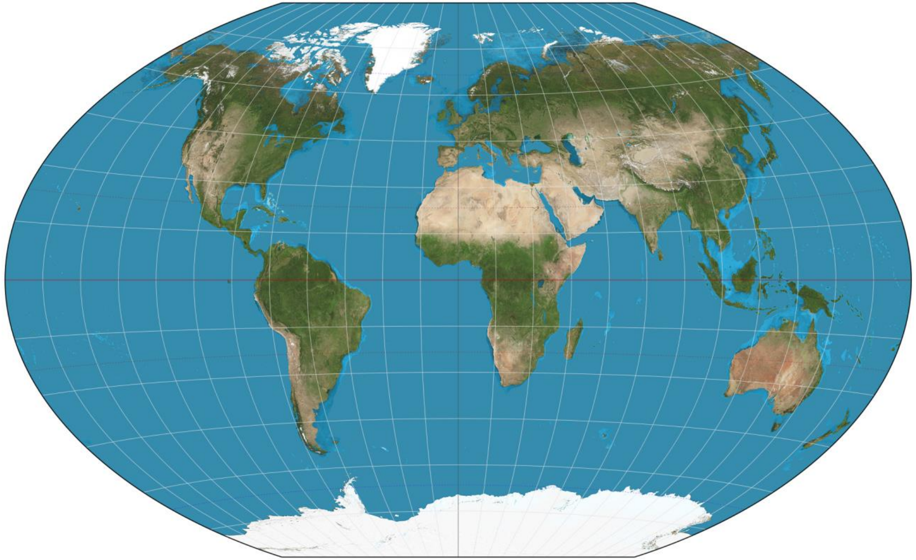
vocabulary words

Word to Vector (Word2Vec)

- Can think Word2Vec as a dictionary
  - Look up each word in the dictionary
  - If the word is in the dictionary, then we can look up the "definition" or location for it



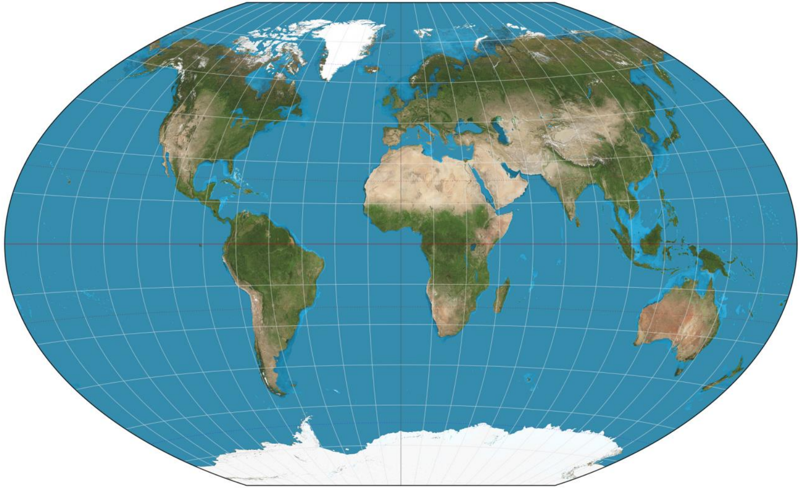
Do we use 2-dimensional embeddings?



	lawyer 1	attorney 2	penguin 3	apple 4	●	●	●	V
longitude	78.8986	79.0558	135.0000	74.0060				
latitude	35.9940	35.9132	82.8628	40.7128				

vocabulary words

Do we use 2-dimensional embeddings?



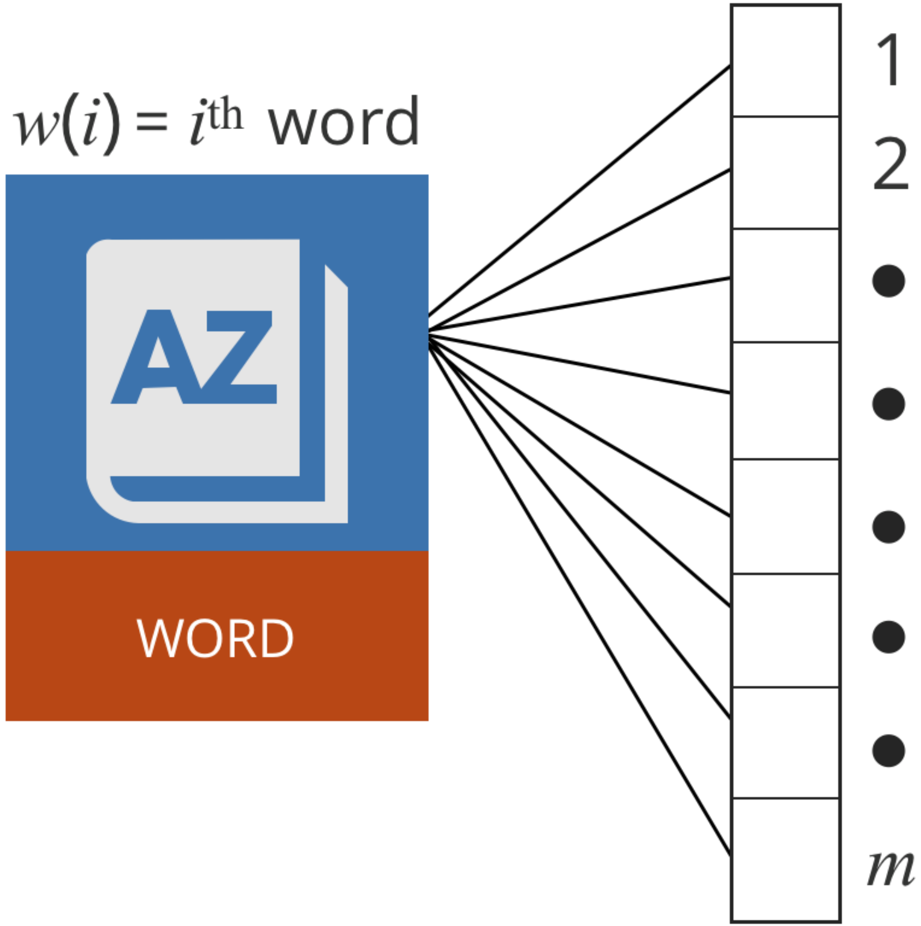
	lawyer 1	attorney 2	penguin 3	apple 4	●	●	●	V
longitude	78.8986	79.0558	135.0000	74.0060				
latitude	35.9940	35.9132	82.8628	40.7128				

vocabulary words

- 2-dimensional locations are useful to illustrate similarity
- *Computationally,* it turns out to be much easier to give each word a much longer address
  - Also representation is easier:
    - 1 direction for plural
    - 1 direction for past tense
    - 1 direction for gender
    - etc…
- The location vector will have *M* different entries
- This is just like specifying a longer address (e.g. state, city, zip, street, house color, car color, etc…)

Word to Vector (Word2Vec)

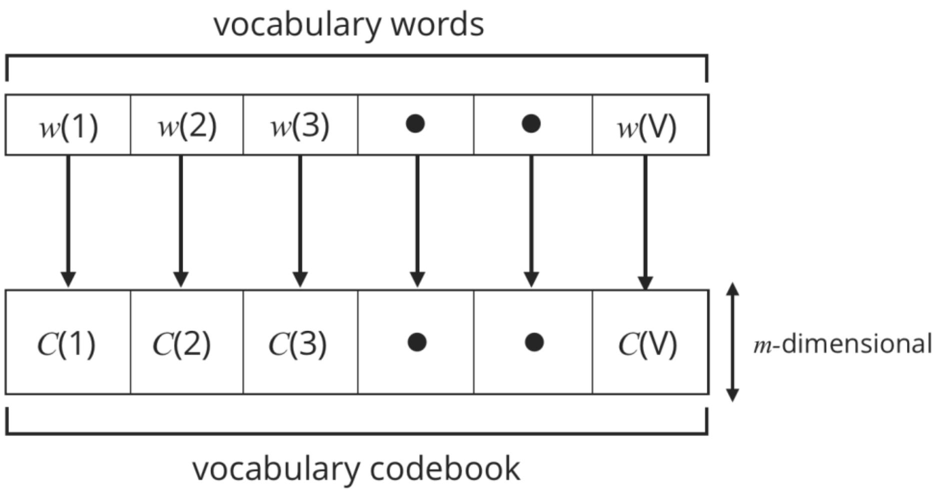
- The key to neural language models involves mapping each word to a vector
- Such word vectors are also called word "embeddings"
- Let $w(i)$ represent the ith word in the vocabulary



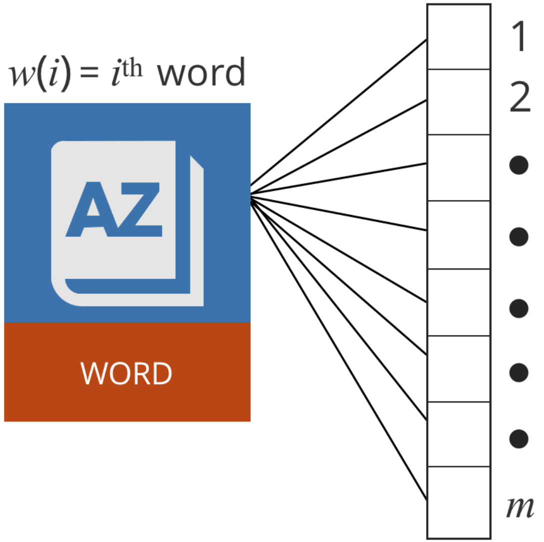
m

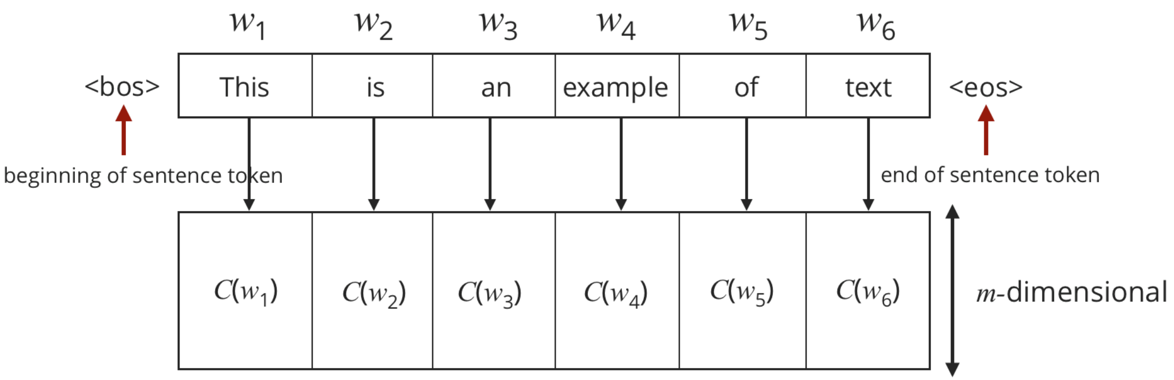
Word to Vector (Word2Vec)

- The set of words in the vocabulary mapped to a codebook of vectors



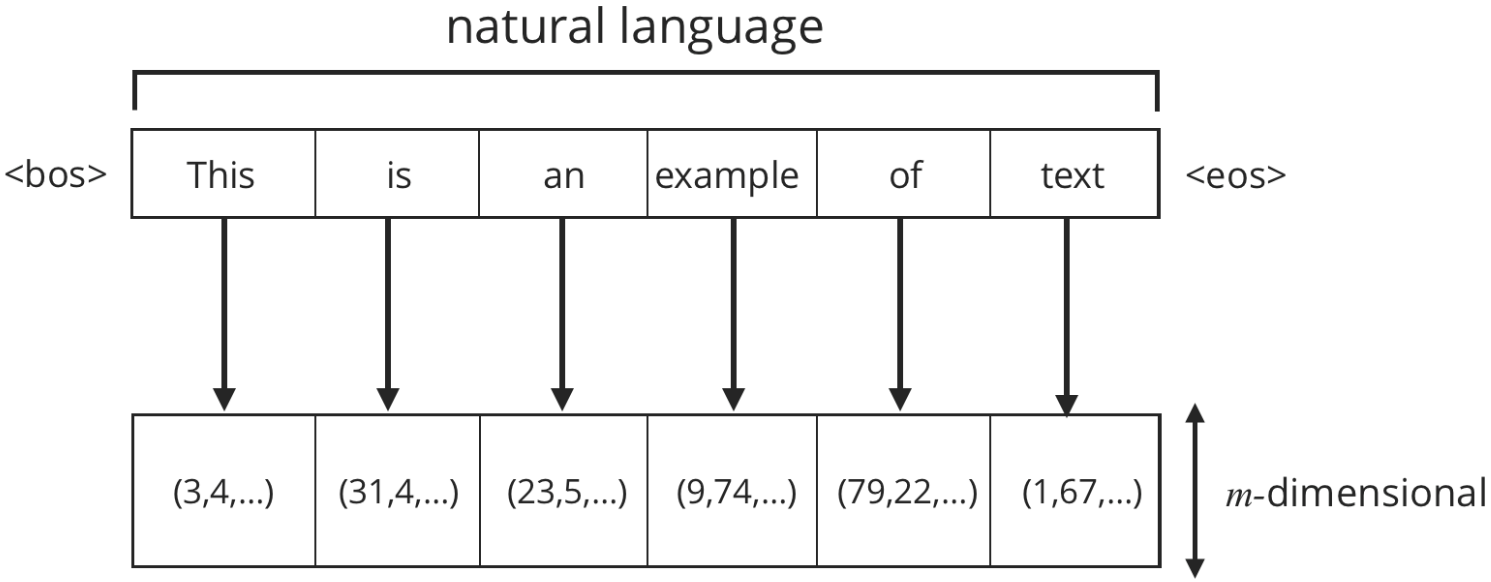
Looking up locations in a dictionary



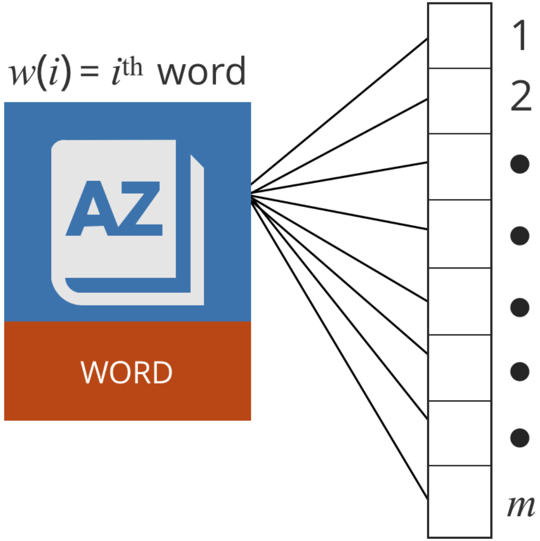


Converting a sentence to embeddings

- We can construct a sequence of word embeddings to represent each sentence.
  - Each word is looked up in our dictionary individually
  - Specific tokens to mark “beginning of sentence” (<bos>) and “end of sentence” (<eos>)



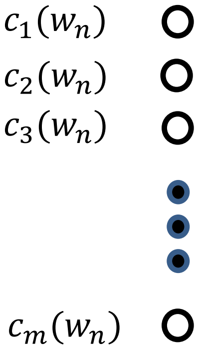
How can we learn the dictionary?



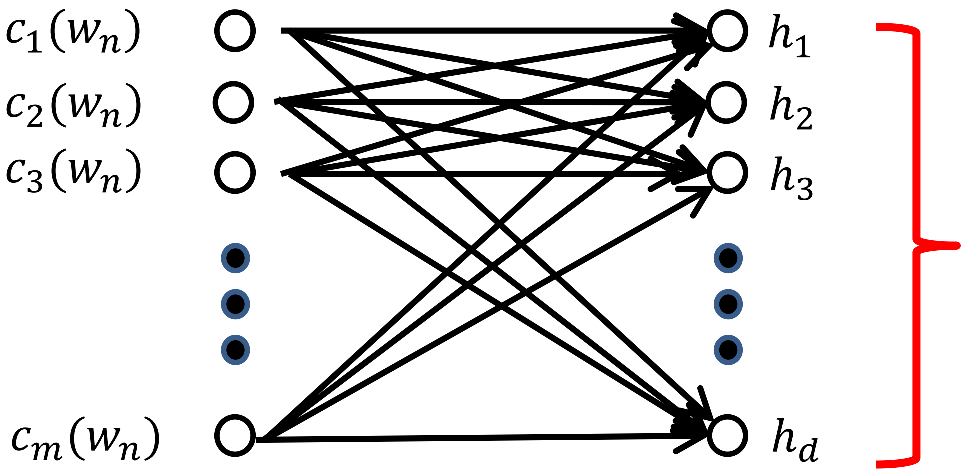
- We don’t start knowing this dictionary—must be learned from the data

Unsupervised Learning of Word Vectors

- Seek to predict the next (w_{n+1}) in a sequence
- Let $c(w_n)$ represent the m-dimensional code for word w_n
- m-dimensional vector $c(w_n)$ is the input to a network



Model Construction

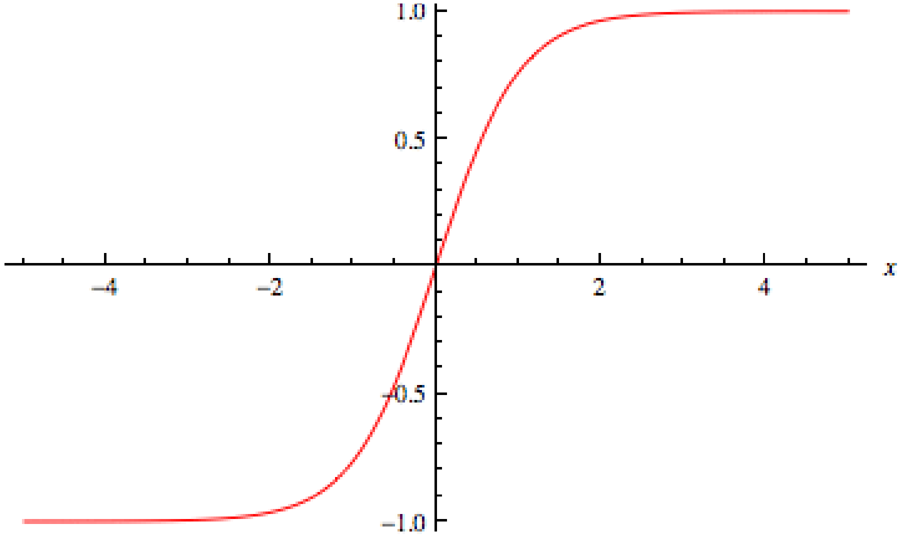


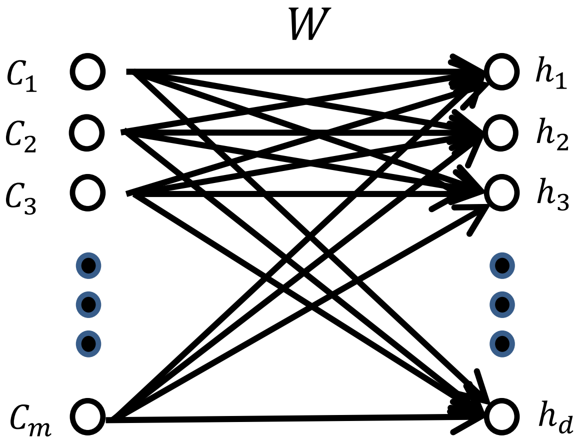
d-dimensional "hidden" unit vector h

Model Construction

$$h_j = g\left[\, W_{1j} \cdot c_1 + W_{2j} \cdot\ c_2 + \cdots + W_{mj} \cdot c_m + b_j \right]$$

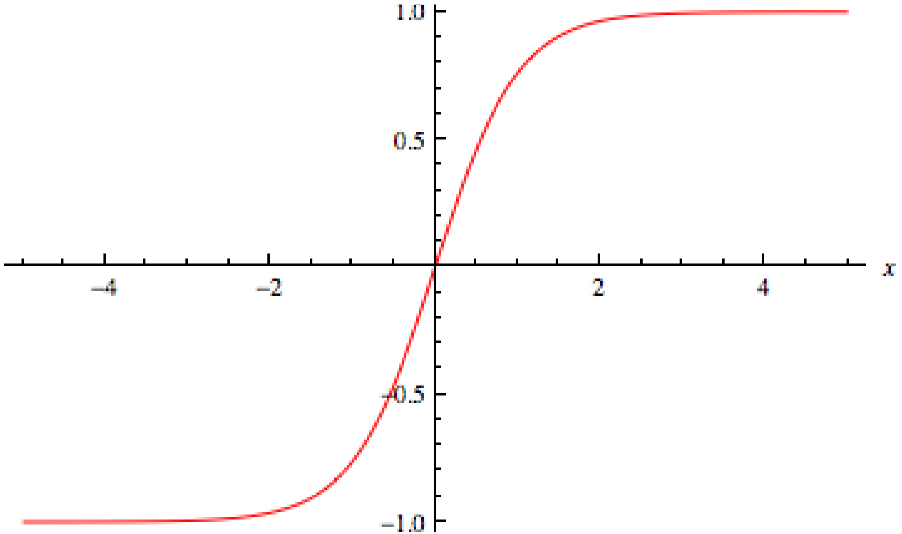
$g[x]$=tanh(x)

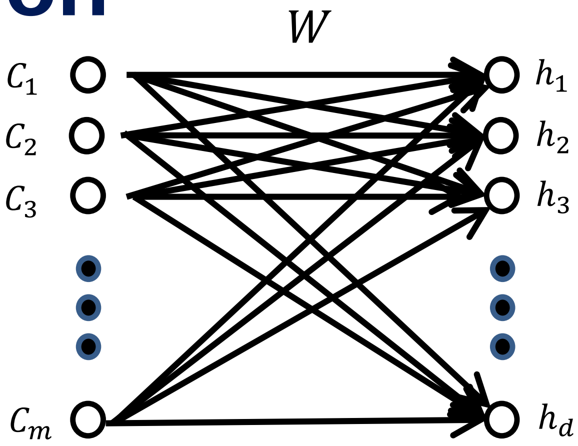




Concise Description

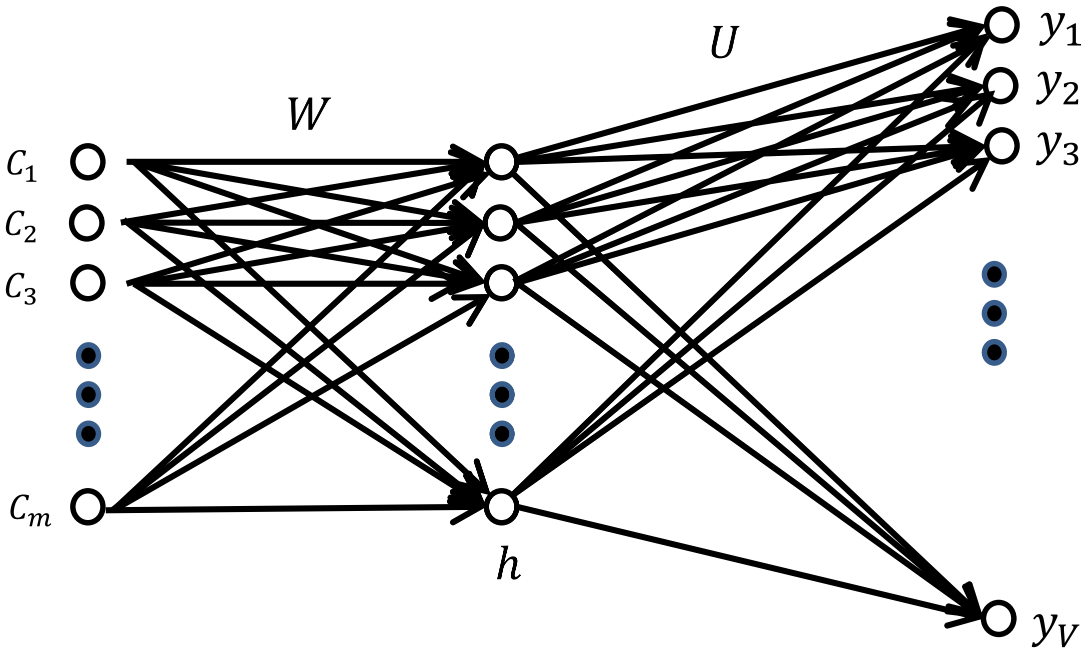
$g[x]$=tanh(x)



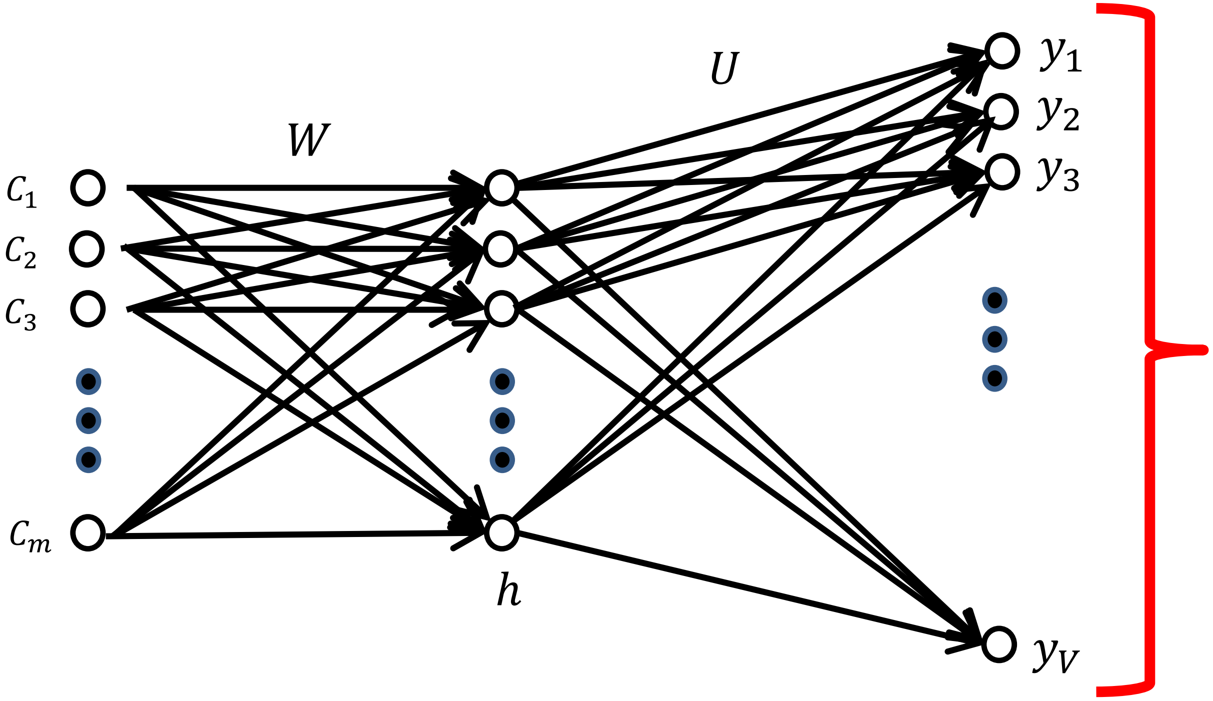


$$h = \tanh[\underbrace{W \cdot c + b}_{\text{Vector}}]$$

Predicting Words

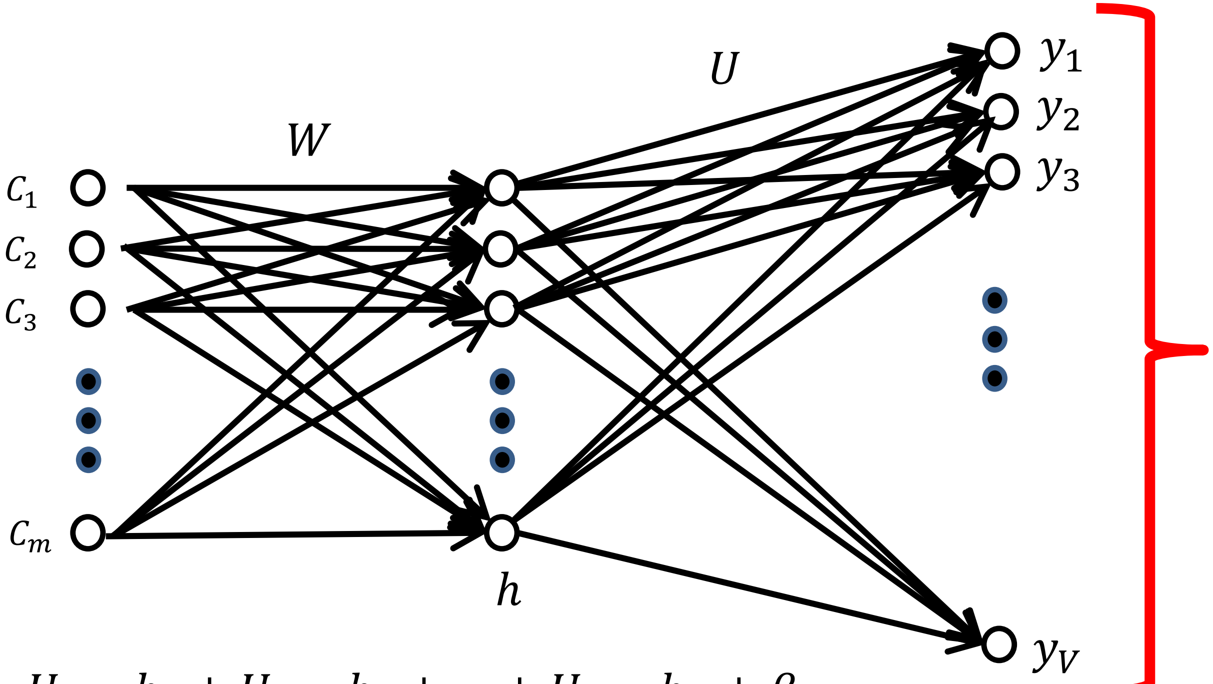


Predicting Words



Output Dimension is V, our vocabulary size

Predicting Words

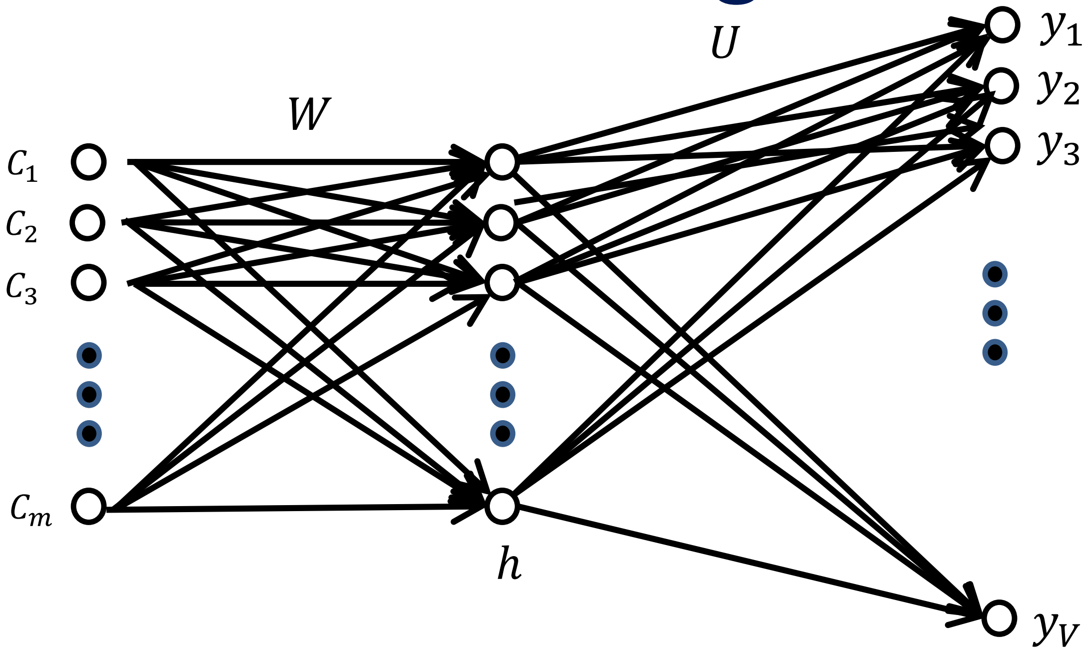


Output Dimension is V, our vocabulary size

$$y_v = U_{1v} \cdot h_1 + U_{2v} \cdot h_2 + \cdots + U_{dv} \cdot h_d + \beta_v$$

Concisely: $y = U \cdot h + \beta$

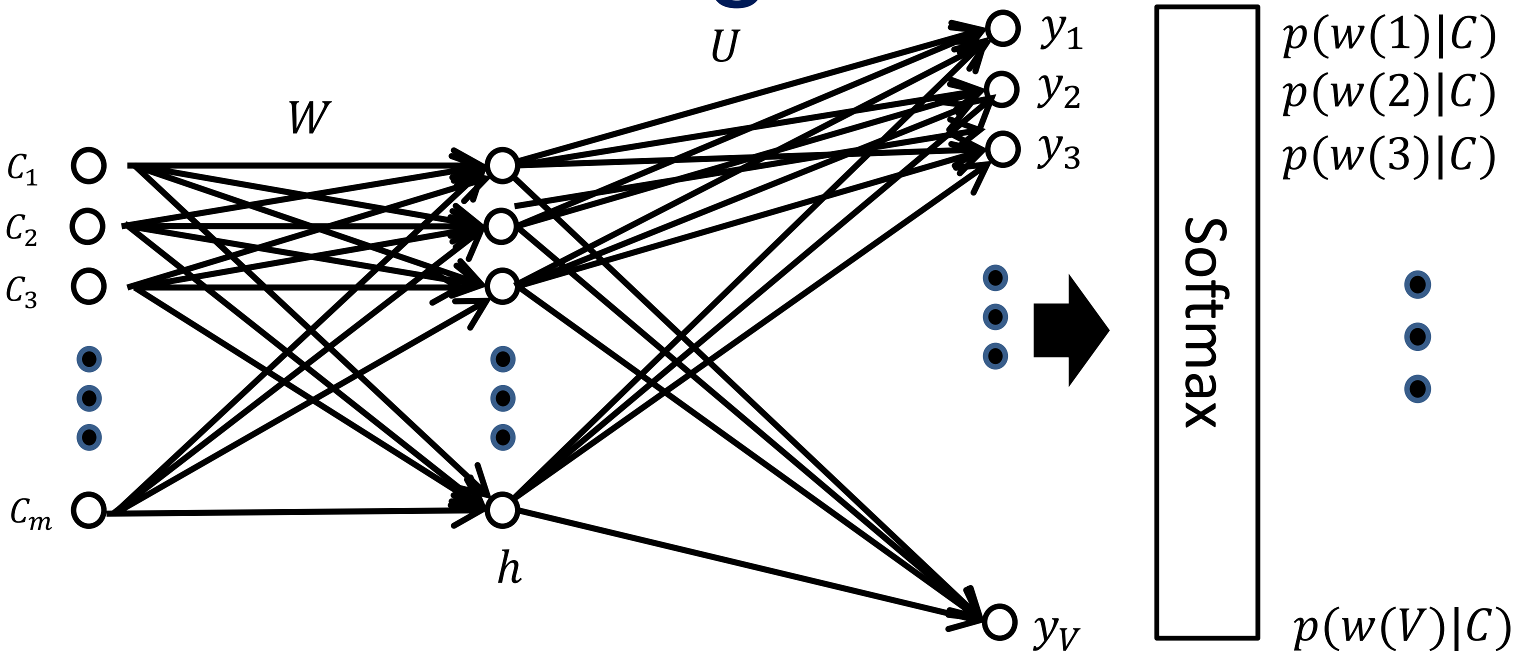
Predicting Words



Convert to probabilities:

$$p\left(w_{n+1} = i^{th} \text{ word} \middle| C(w_n)\right) = \frac{\exp(y_i)}{\exp(y_1) + \exp(y_2) + \cdots + \exp(y_V)}$$

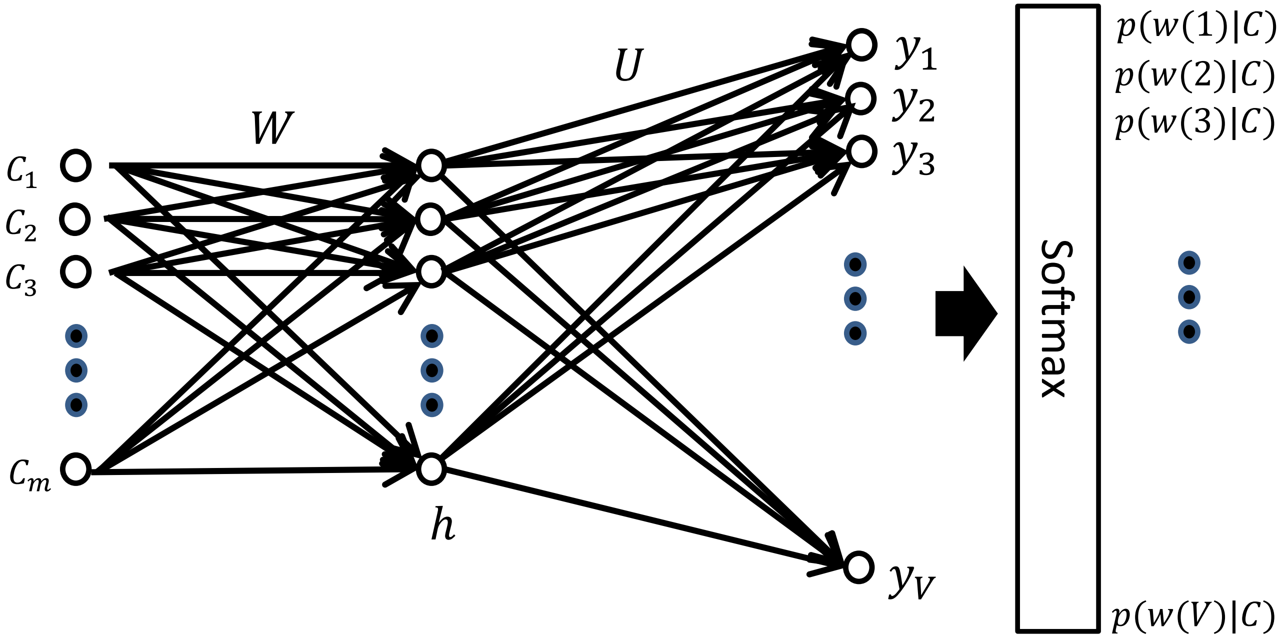
Predicting Words



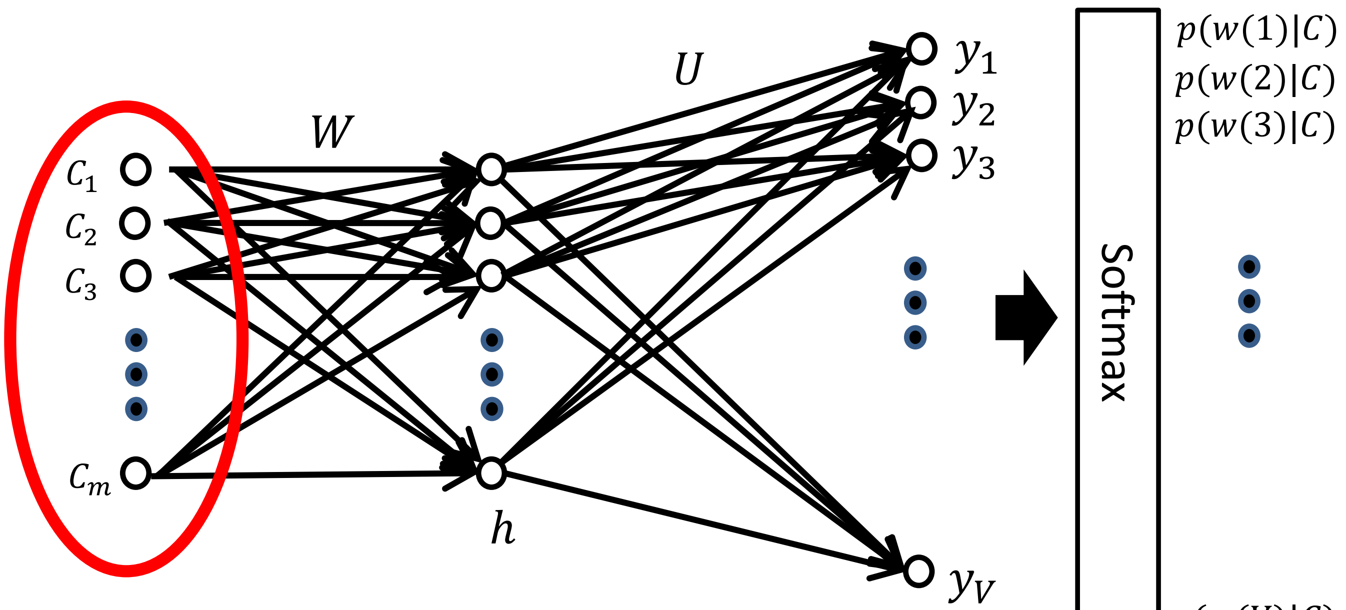
Convert to probabilities:

$$p\left(w_{n+1} = i^{th}\ \text{word} \middle| C(w_n)\right) = \frac{\exp(y_i)}{\exp(y_1) + \exp(y_2) + \cdots + \exp(y_V)}$$

Neural Text Model

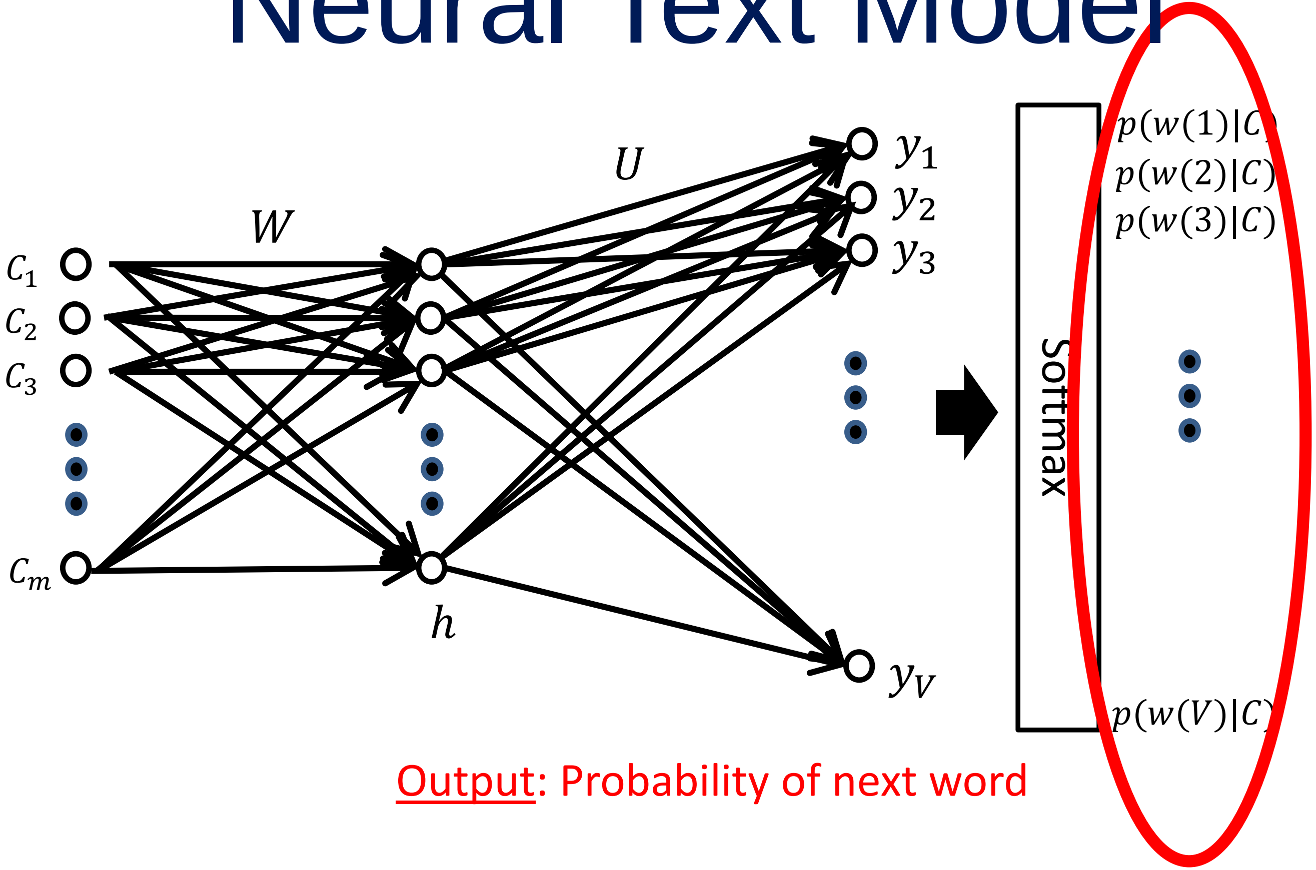


Neural Text Model



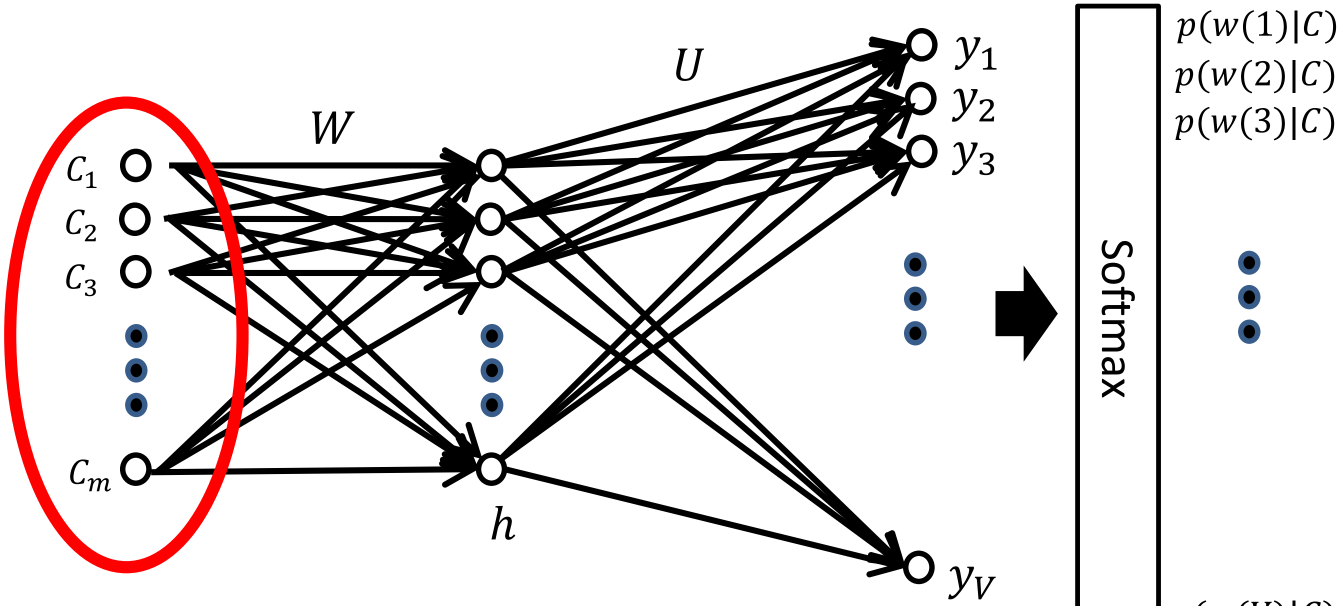
Input: Word vector for particular word in document

Neural Text Model



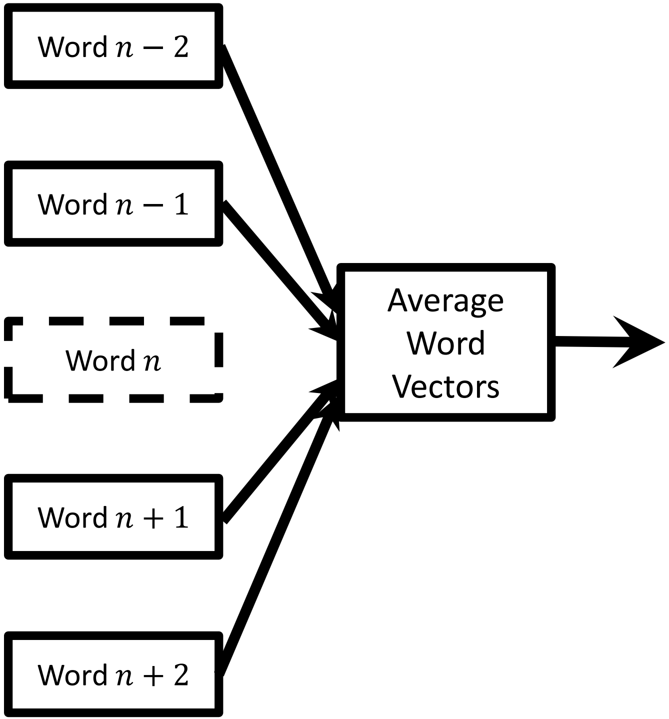
Output: Probability of next word

Neural Text Model

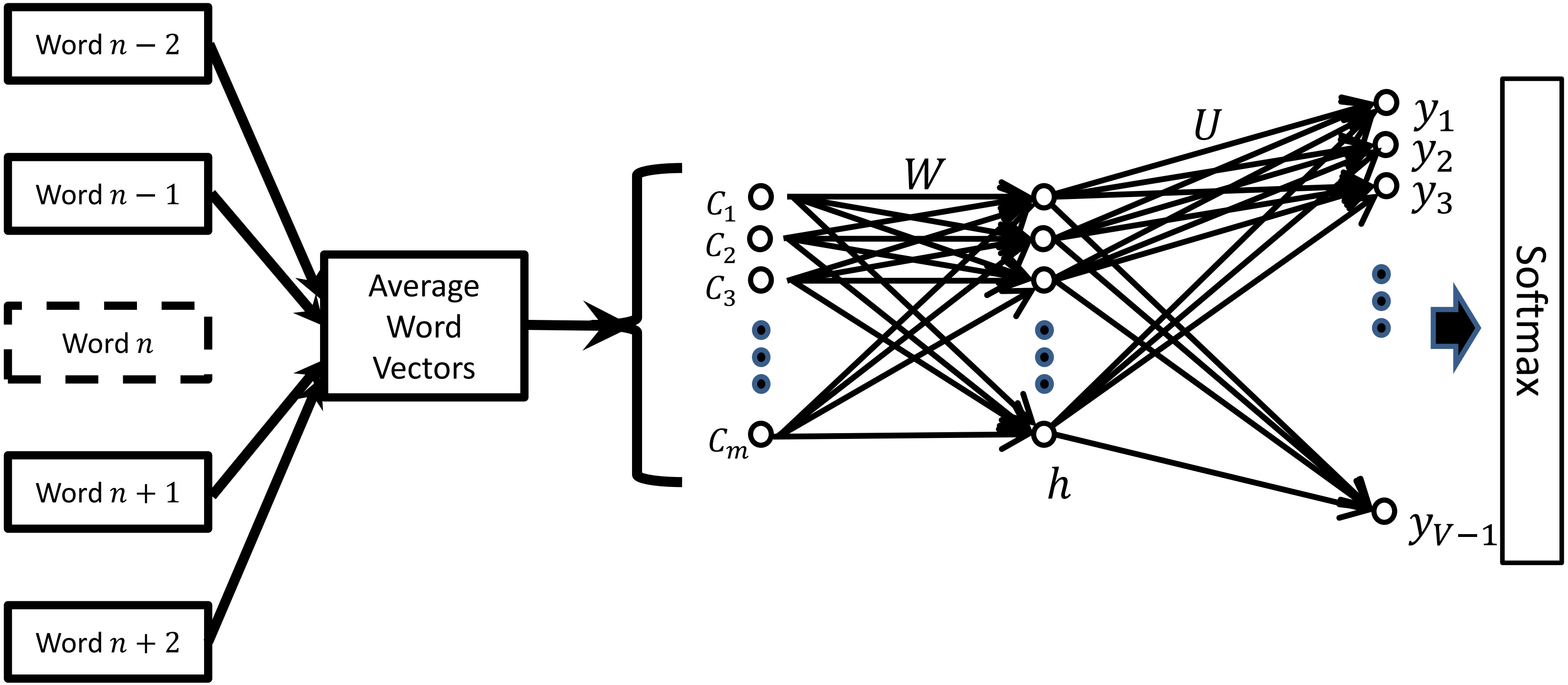


Will learn the word embedding to improved prediction of the next word.

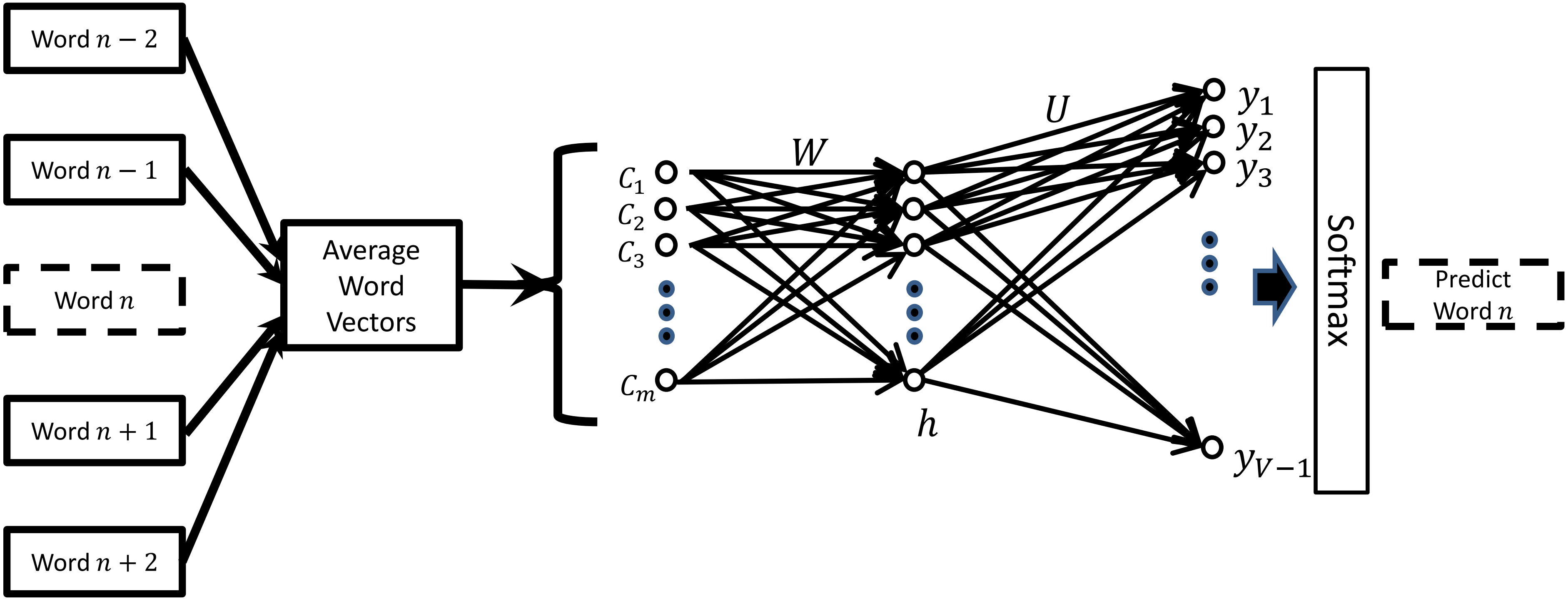
Continuous Bag of Words (CBOW) Model



Continuous Bag of Words (CBOW) Model

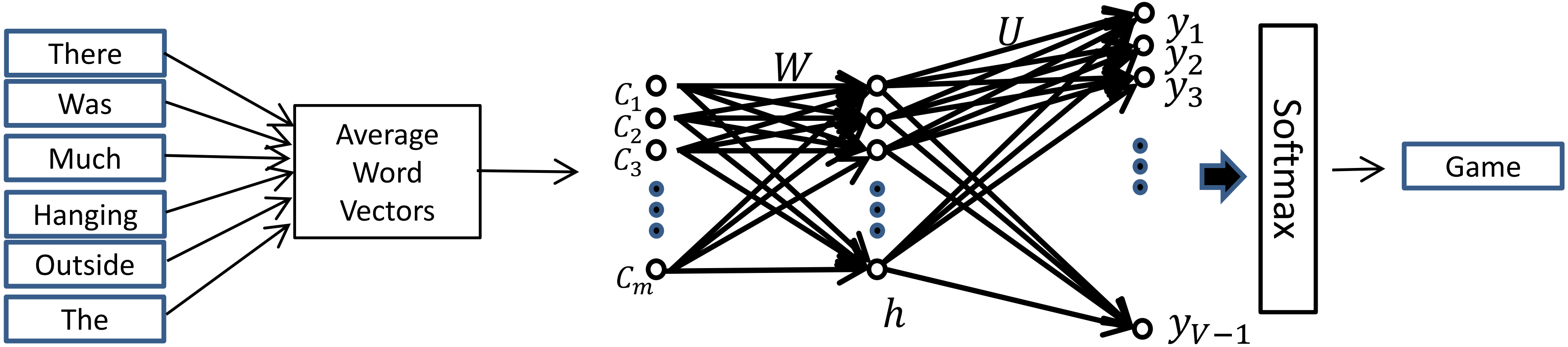


Continuous Bag of Words (CBOW) Model



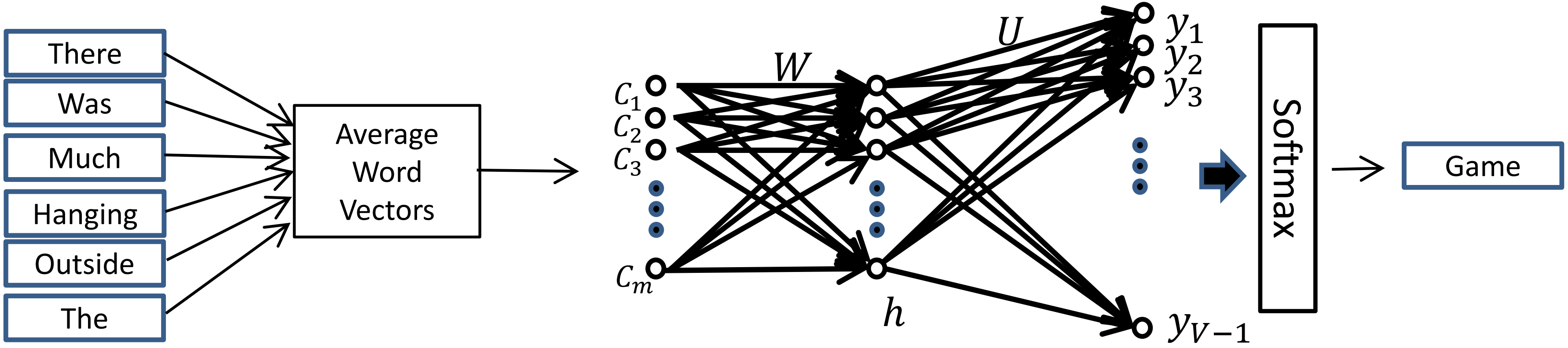
In the fall the war was always there, but we did not go to it any more. It was cold in the fall in Milan and the dark came very early. Then the electric lights came on, and it was pleasant along the streets looking in the windows. There was much game hanging outside the shops, and the snow powdered in the fur of the foxes and the wind blew their tails. The deer hung stiff and heavy and empty, and small birds blew in the wind and the wind turned their feathers. It was a cold fall and the wind came down from the mountains.

In the fall the war was always there, but we did not go to it any more. It was cold in the fall in Milan and the dark came very early. Then the electric lights came on, and it was pleasant along the streets looking in the windows. There was much game hanging outside the shops, and the snow powdered in the fur of the foxes and the wind blew their tails. The deer hung stiff and heavy and empty, and small birds blew in the wind and the wind turned their feathers. It was a cold fall and the wind came down from the mountains.



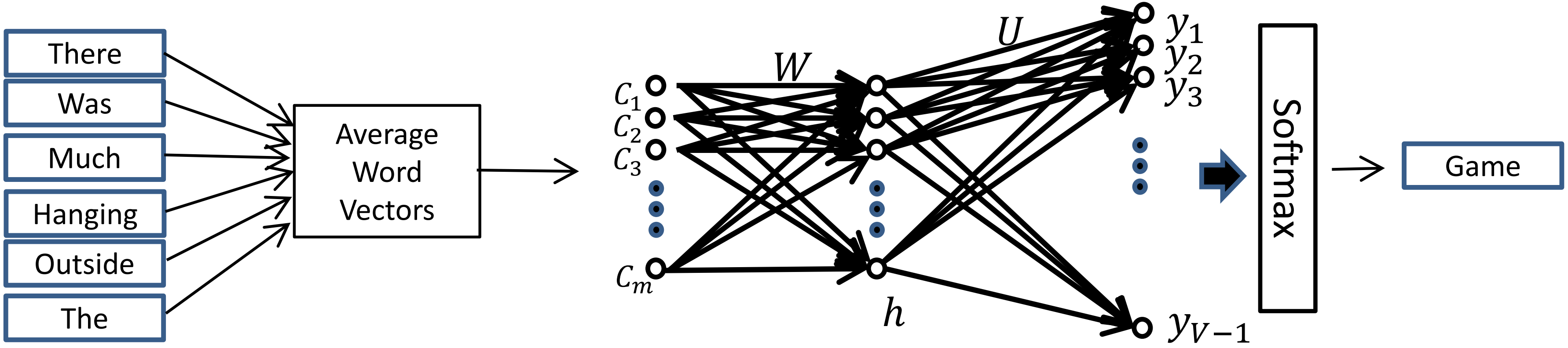
Learning

- For the CBOW setup, there is a true word, and a prediction



Learning

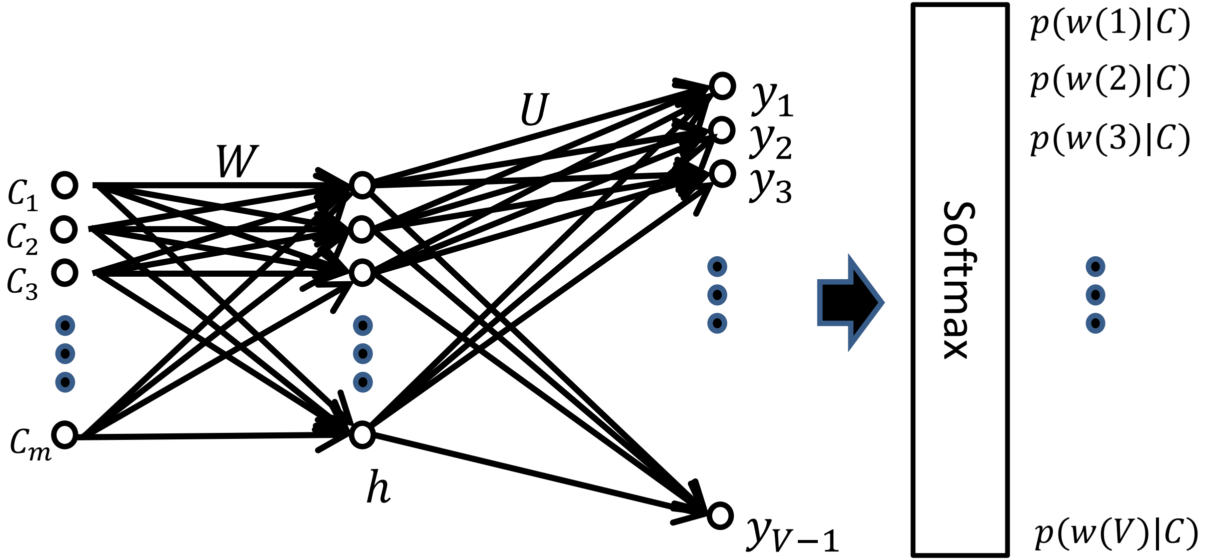
- For the CBOW setup, there is a true word, and a prediction



- For either form of model, learn the model parameters C, W, U, b and β such that the model yields high probabilities for the true word output

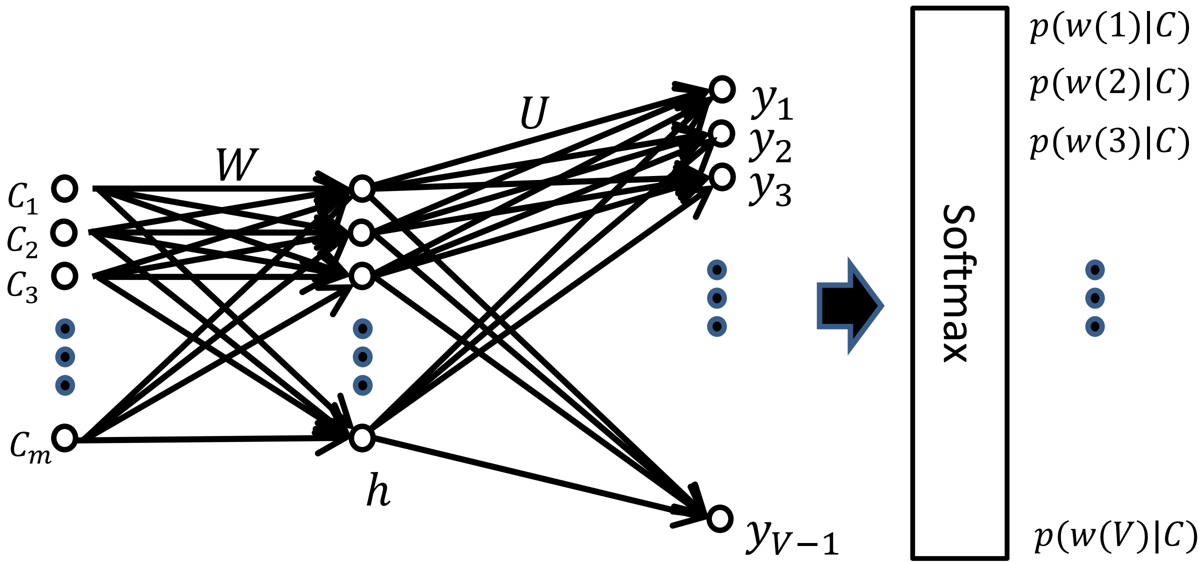
- Do such model fitting across an entire, massive database of text

Model For Unsupervised Learning of Word Vectors



Model parameters to be learned: C, W, U, b, β

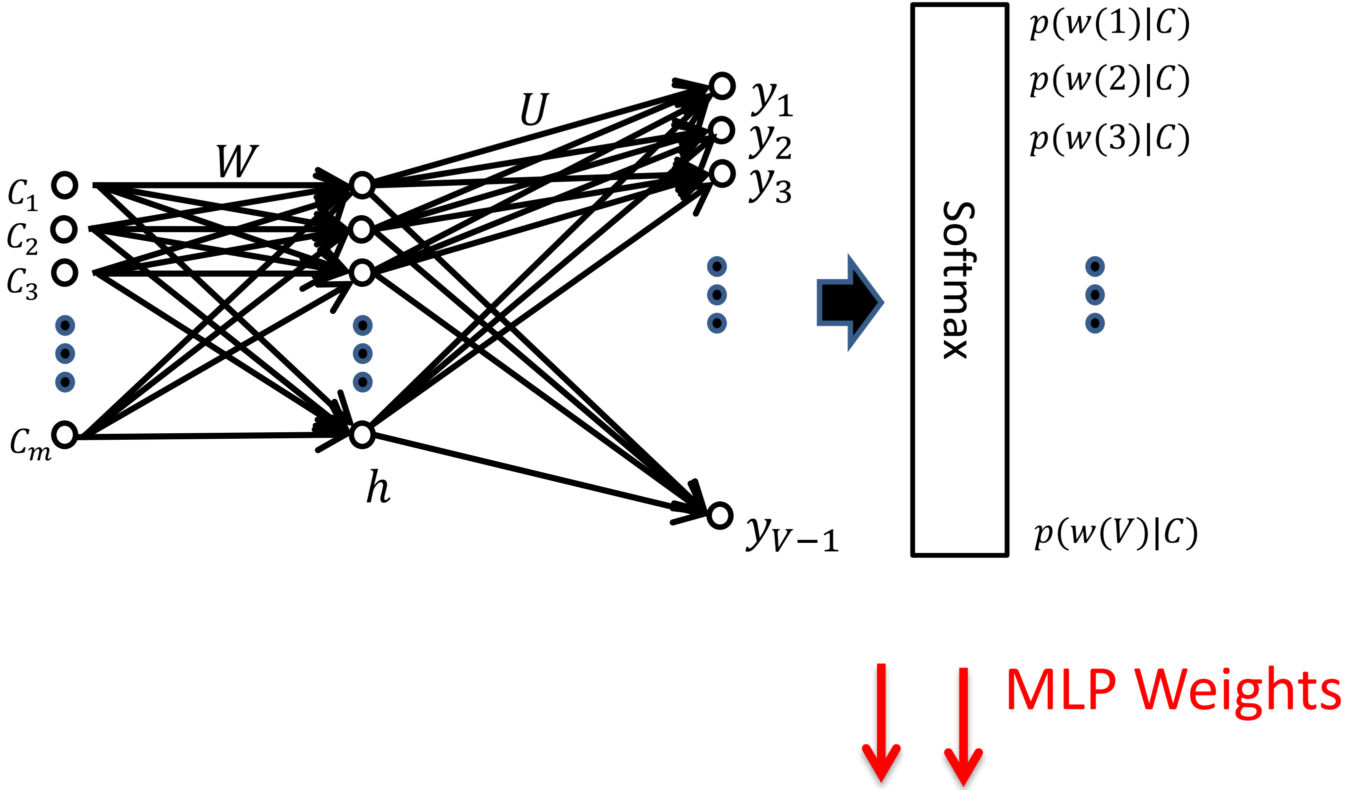
Model For Unsupervised Learning of Word Vectors



Word Vectors

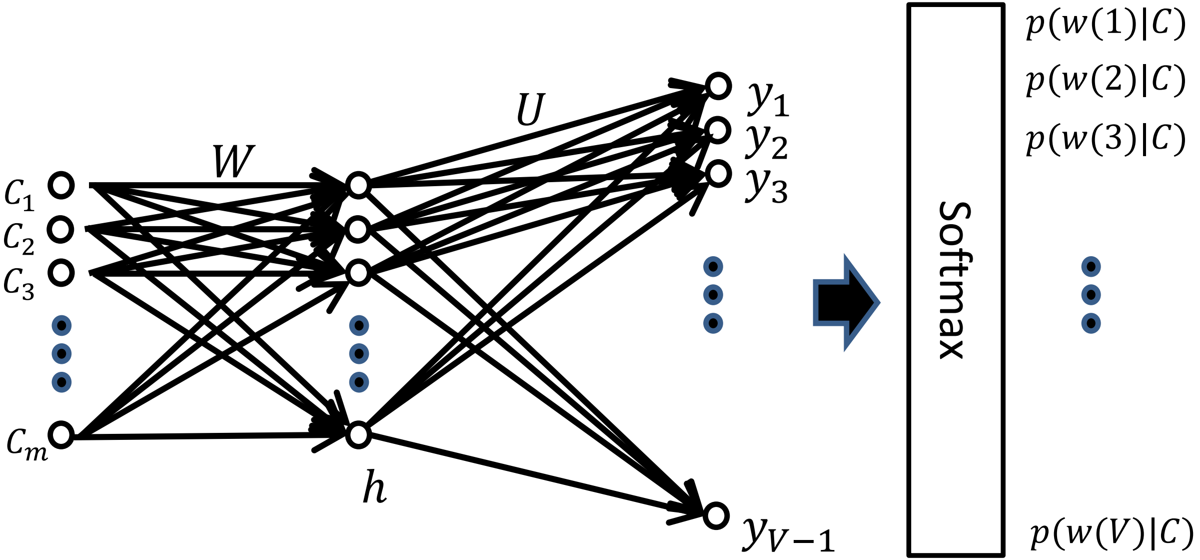
Model parameters to be learned: C, W, U, b, β

Model For Unsupervised Learning of Word Vectors



Model parameters to be learned: C, W, U, b, β

Model For Unsupervised Learning of Word Vectors



MLP Biases

Model parameters to be learned: C, W, U, b, β

Some Comments

- This forms the basis of a text model
- Can learn the dictionary to minimize the associated loss (i.e. predict next words as well as possible)
- Can incorporate a larger history (more words)? Can we do this automatically?
- How do we make this useful for predictions?

Lecture 2

LEARNING MODEL PARAMETERS

Learning

- Assume that we employ such learning for M word examples from our corpus

- Represent the model as

$$p(w_{out}|w_{in}\,;\theta)$$

Learning

- Assume that we employ such learning for M word examples from our corpus

- Represent the model as

$$p(w_{out}|w_{in}\,;\theta)$$

Probability of Output Word
From Neural Network (via Softmax)

Learning

- Assume that we employ such learning for M word examples from our corpus

- Represent the model as

Contextual Word(s)
Defining Input to Neural Network

$$p(w_{out}|w_{in}\,;\theta)$$

Learning

- Assume that we employ such learning for M word examples from our corpus

- Represent the model as

$$p(w_{out}|w_{in}\,;\theta)$$

Neural Network Model Parameters
C, W, U, b, β

Learning

- Assume that we employ such learning for M word examples from our corpus

- Represent the model as $p(w_{out}|w_{in};\theta)$

- Via M examples from our corpus, $\{(w_{in}^i, w_{out}^i)\}_{i=1,M}$, we constitute

$$f(\theta; Data) = \log p\left(w_{out}^1 \middle| w_{in}^1; \theta\right) + \log p\left(w_{out}^2 \middle| w_{in}^2; \theta\right) + \cdots + \log p\left(w_{out}^M \middle| w_{in}^M; \theta\right)$$

Learning

- Assume that we employ such learning for M word examples from our corpus

- Represent the model as $p(w_{out}|w_{in}\,;\theta)$

- Via M examples from our corpus, $\{(w_{in}^i, w_{out}^i)\}_{i=1,M}$, we constitute

$$f(\theta; Data) = \log p\left(w_{out}^1 \middle| w_{in}^1; \theta\right) + \log p\left(w_{out}^2 \middle| w_{in}^2; \theta\right) + \cdots + \log p\left(w_{out}^M \middle| w_{in}^M; \theta\right)$$

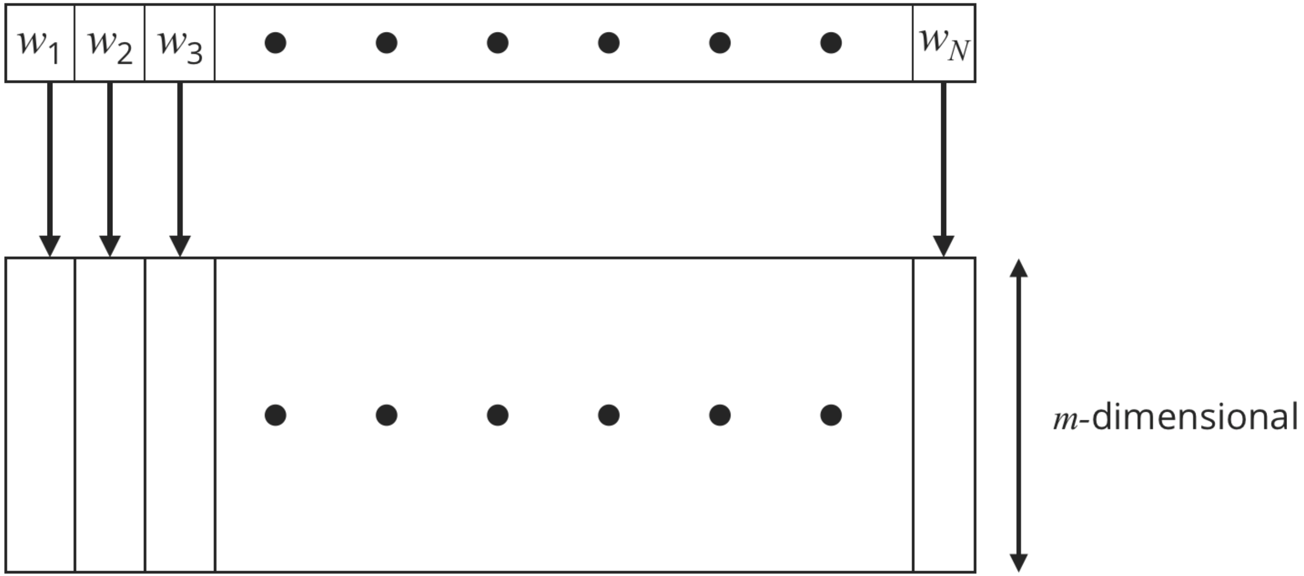
- Seek model parameters θ that maximize $f(\theta; Data)$

RECURRENT NEURAL NETWORKS

Generating Text

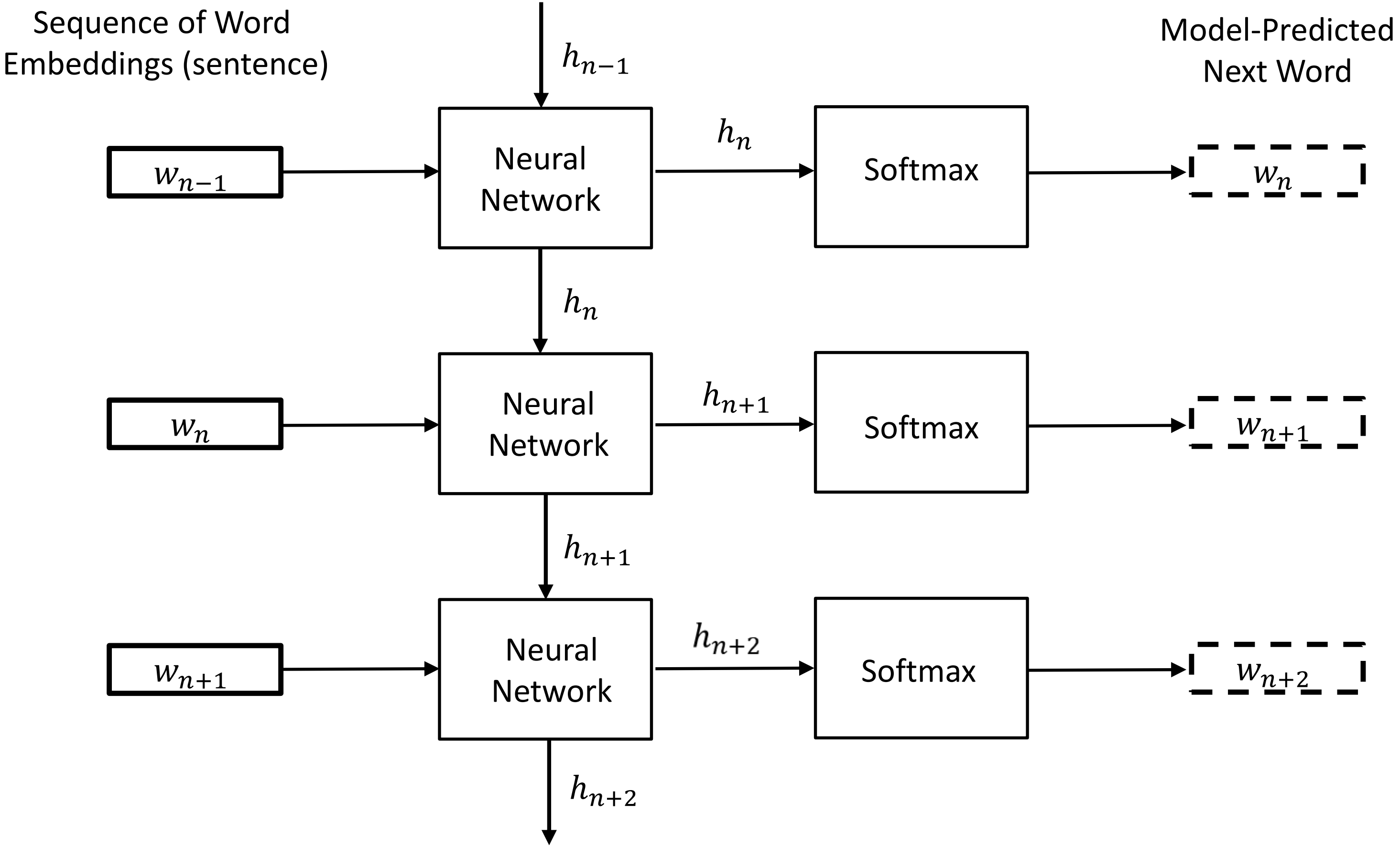
- Assume we have learned word embeddings (vectors) Want to use these embeddings in applications
- Text *synthesis* may be of interest for automatic captioning of images, and for translation from one language to another
- These tools are quite useful generally (will discuss in case studies)
- We require additional tools for text synthesis: The recurrent neural network (RNN)

Using Word Embeddings

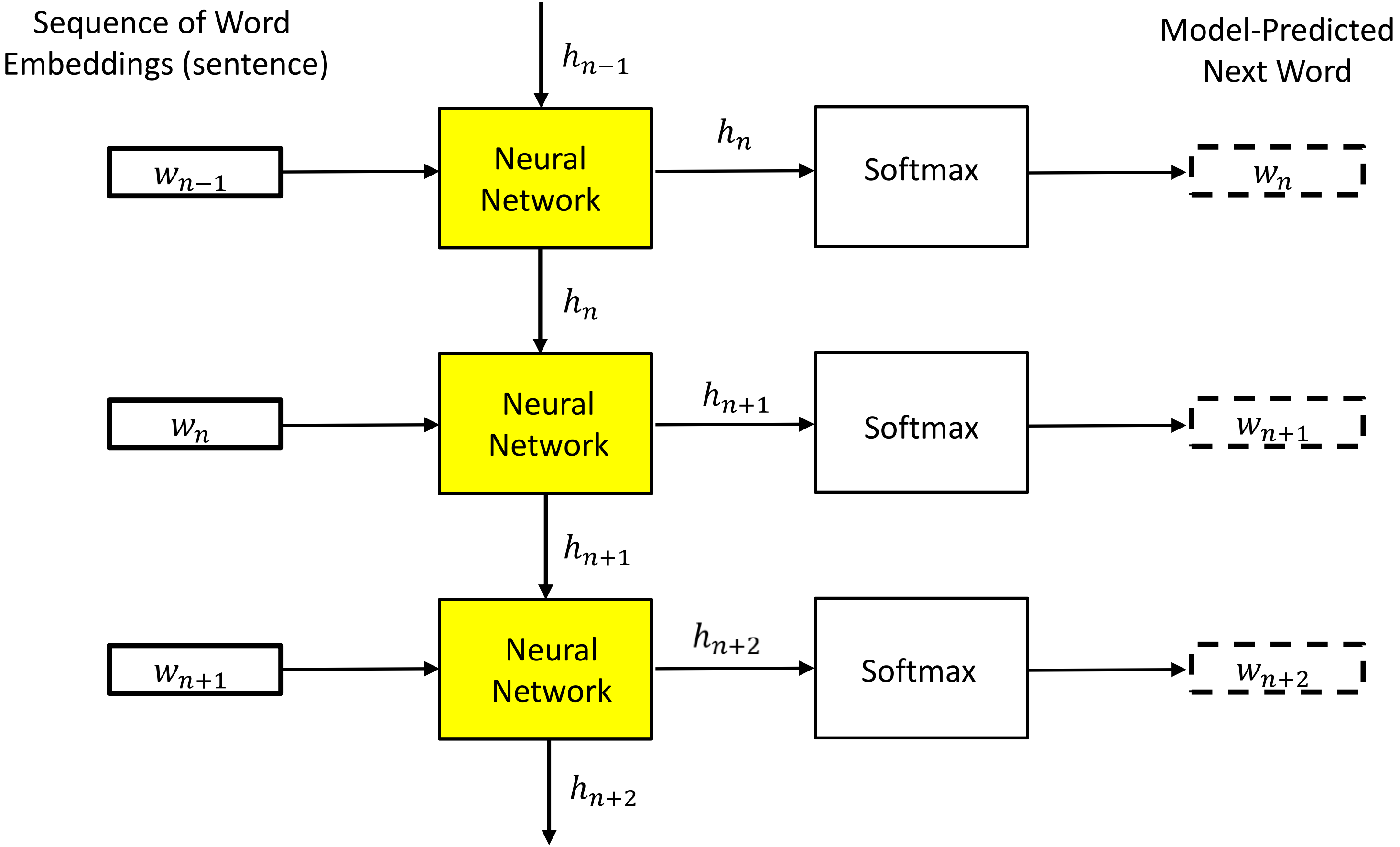


- Our representation depends on the number of words
  - Not a constant number of features!

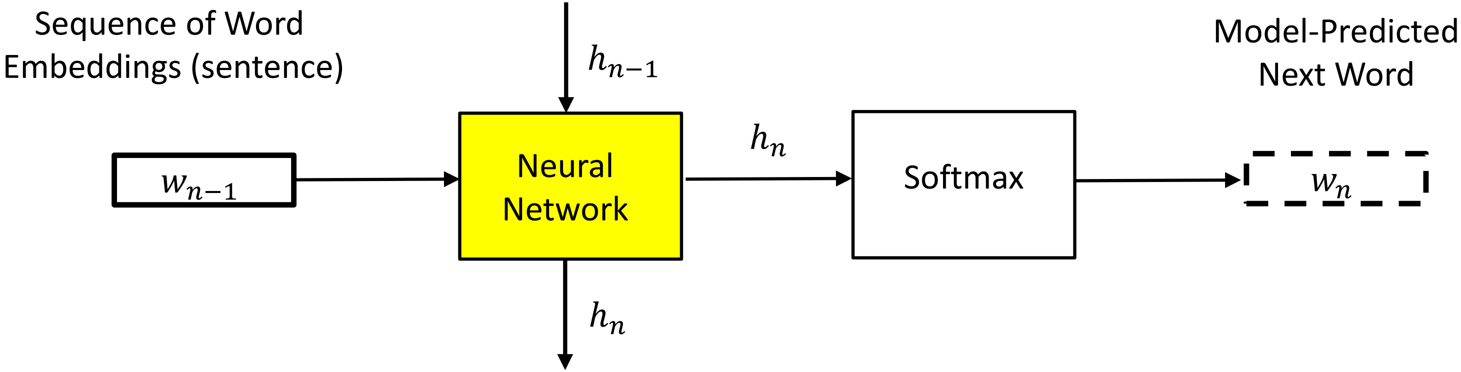
Recurrent Neural Network



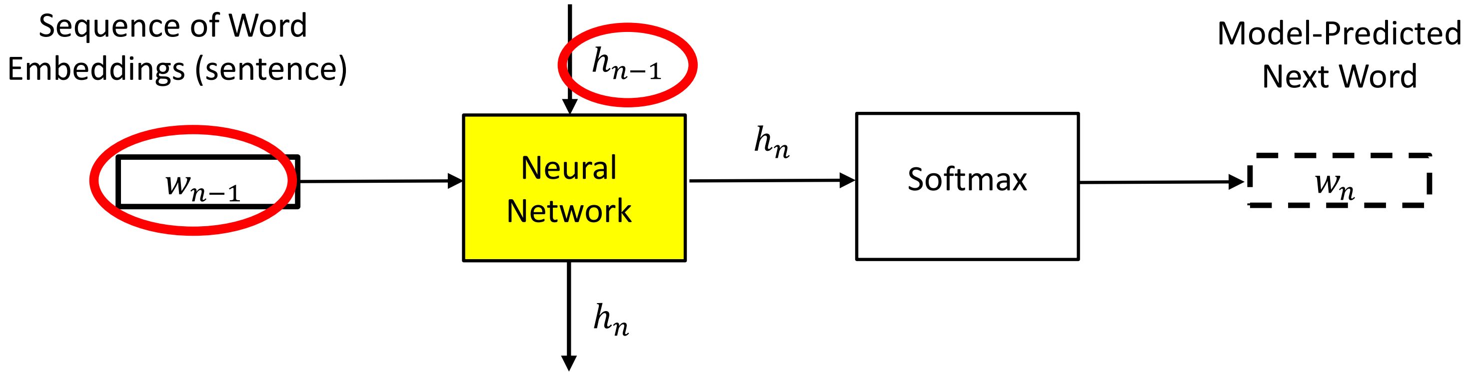
Recurrent Neural Network



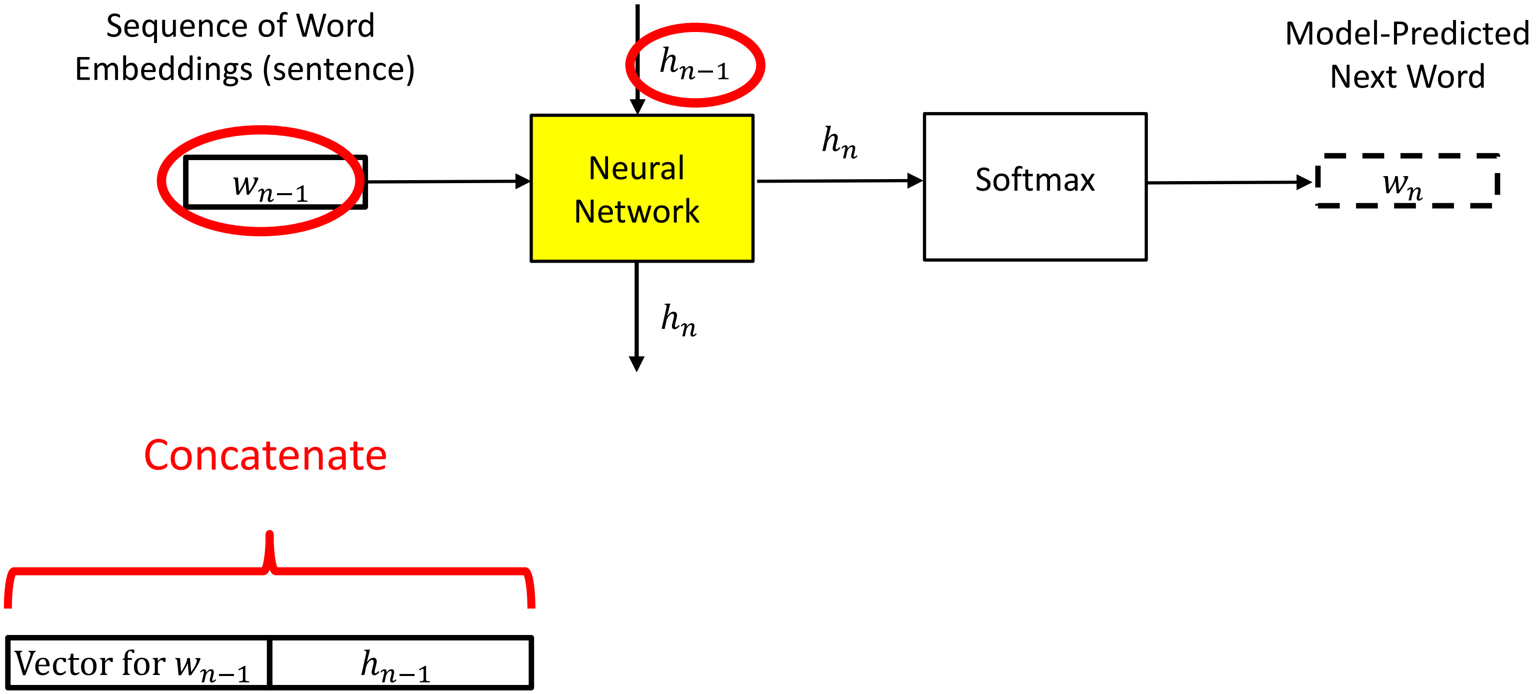
Recurrent Neural Network



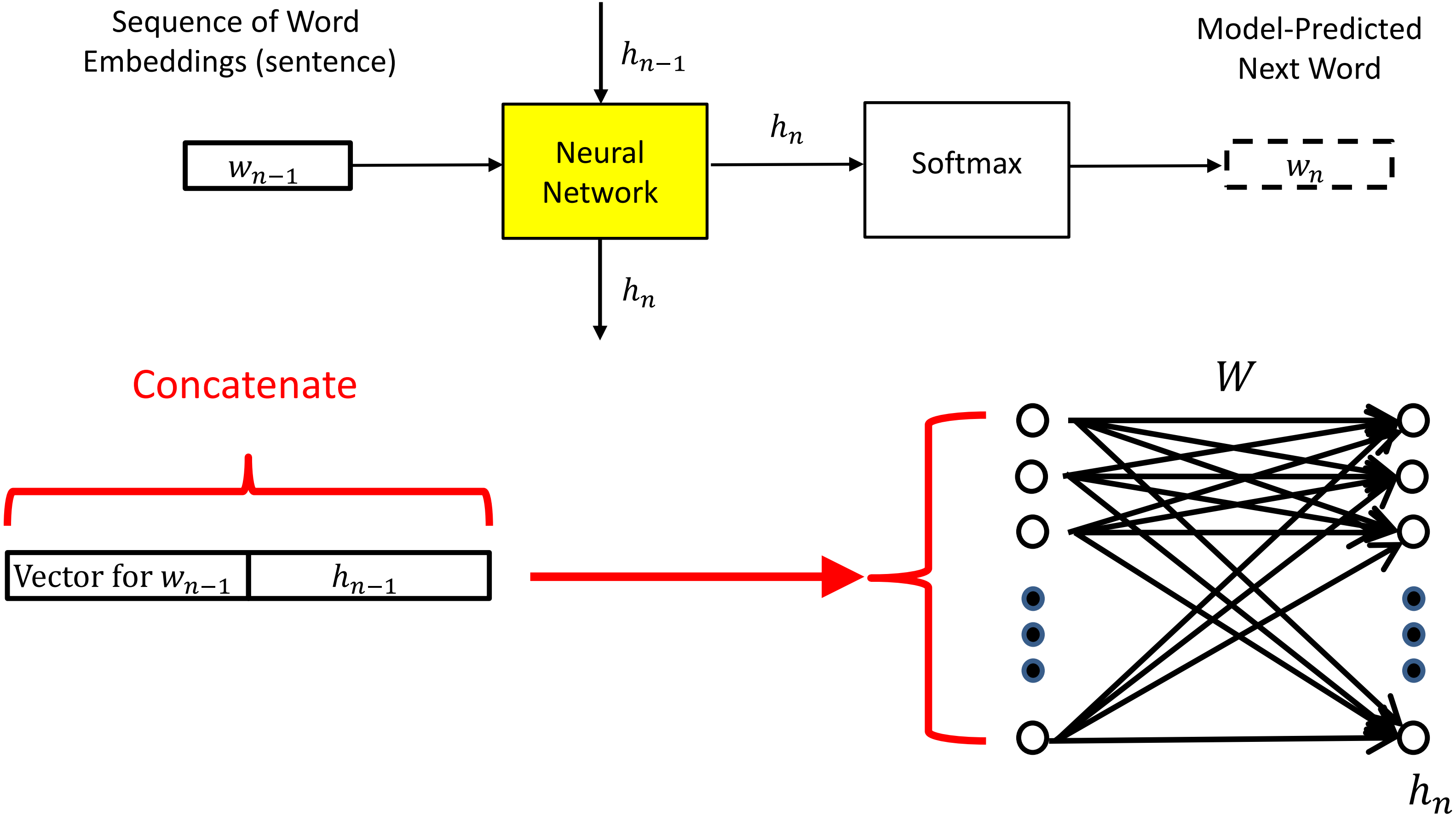
Recurrent Neural Network



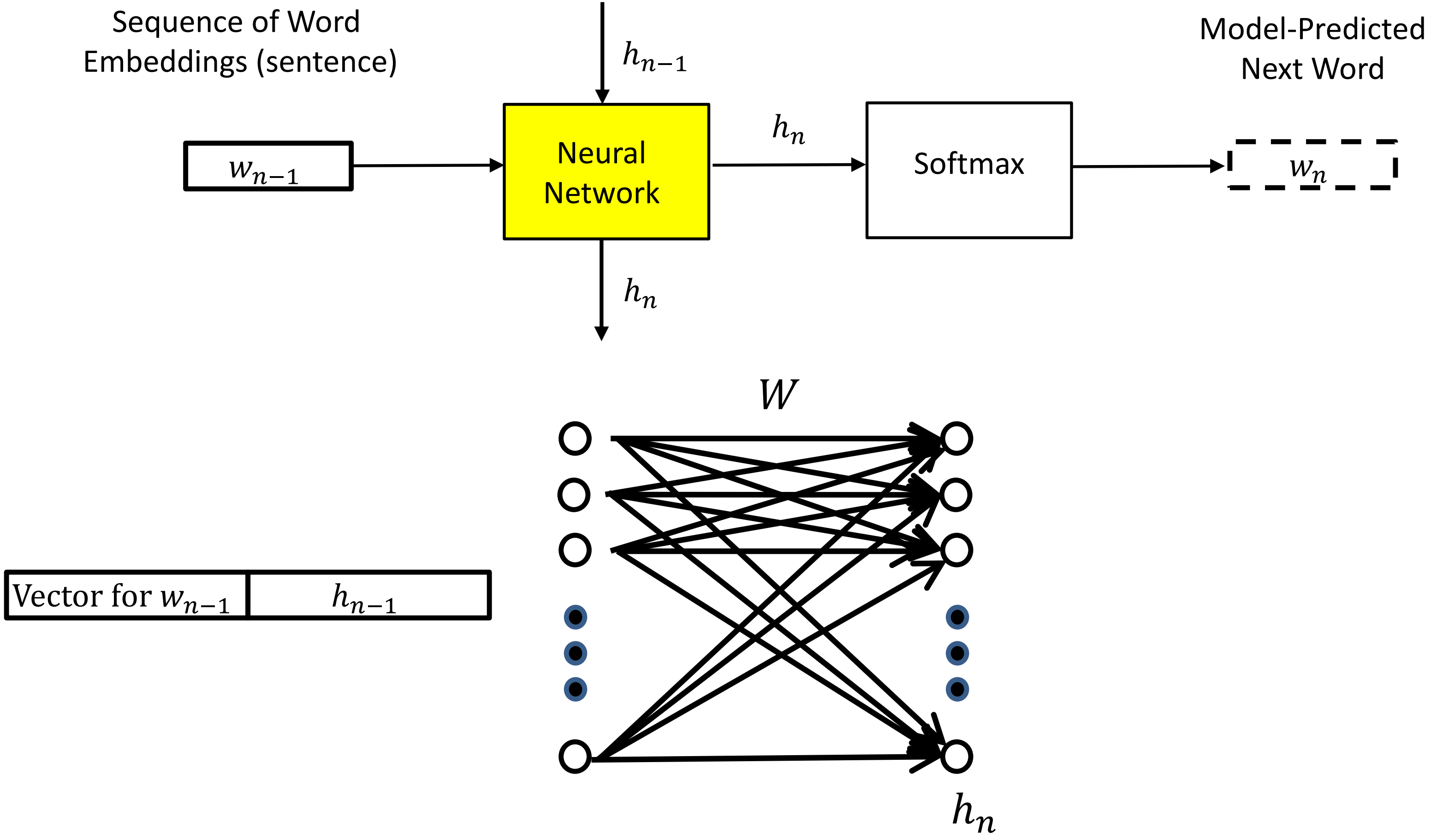
Recurrent Neural Network



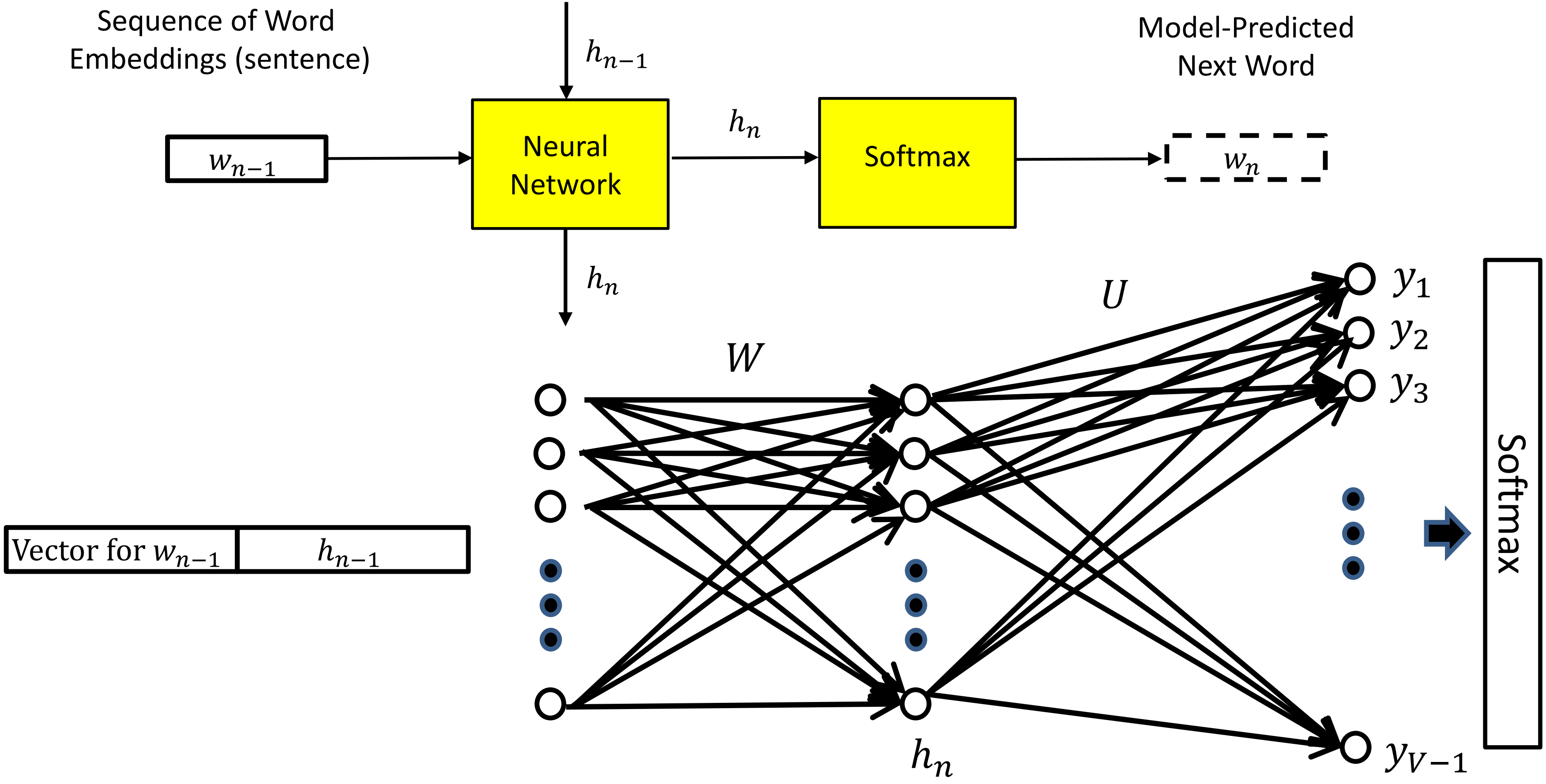
Recurrent Neural Network



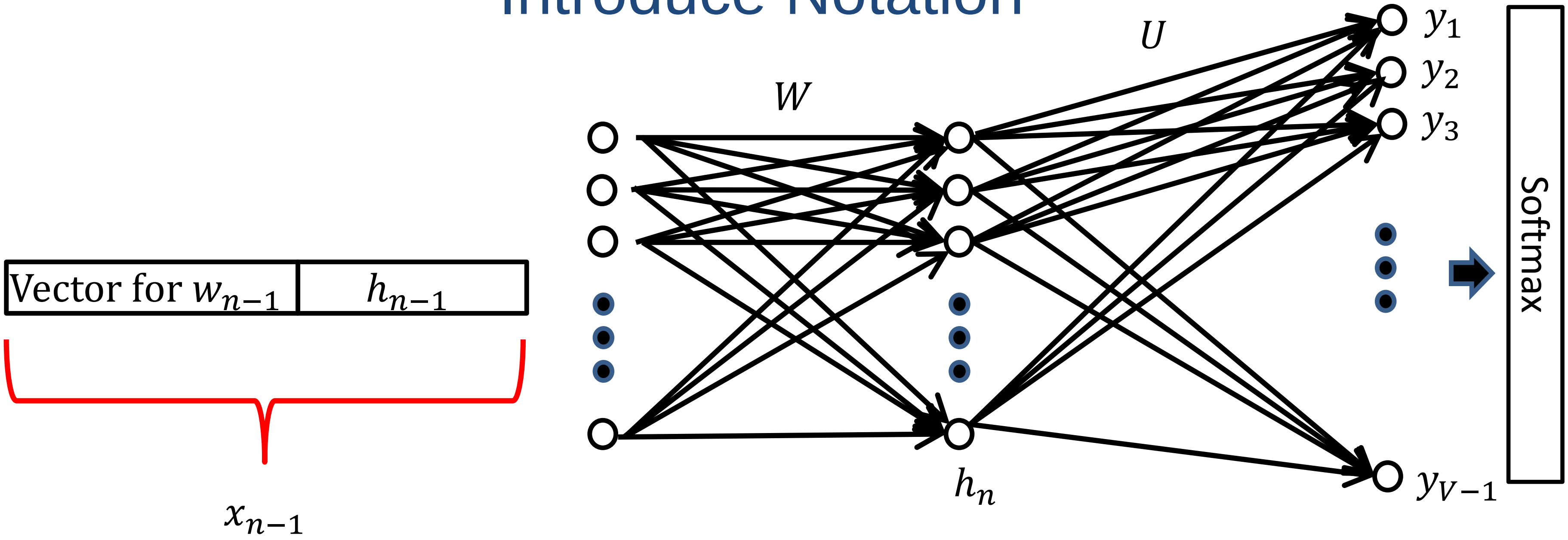
Recurrent Neural Network



Recurrent Neural Network



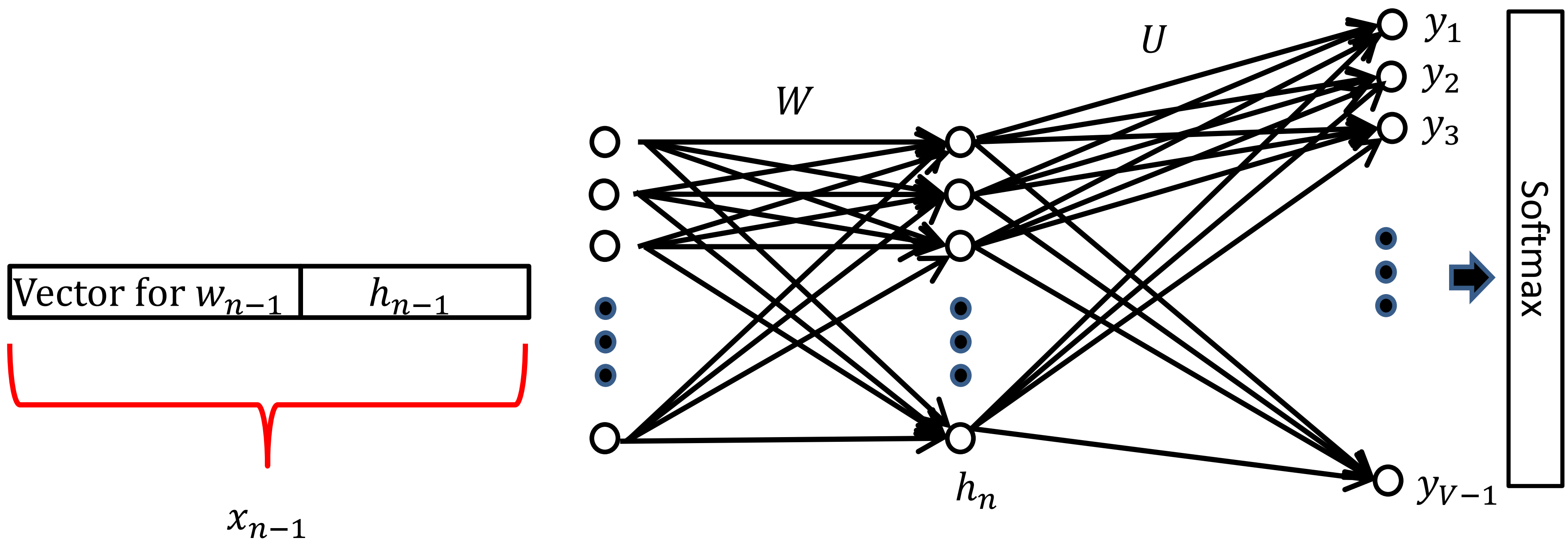
Introduce Notation



$$h_n = \tanh(W \cdot x_{n-1} + b)$$

$$p(w_n | w_{n-1}, h_{n-1}) = \text{softmax}(U \cdot h_n + \beta)$$

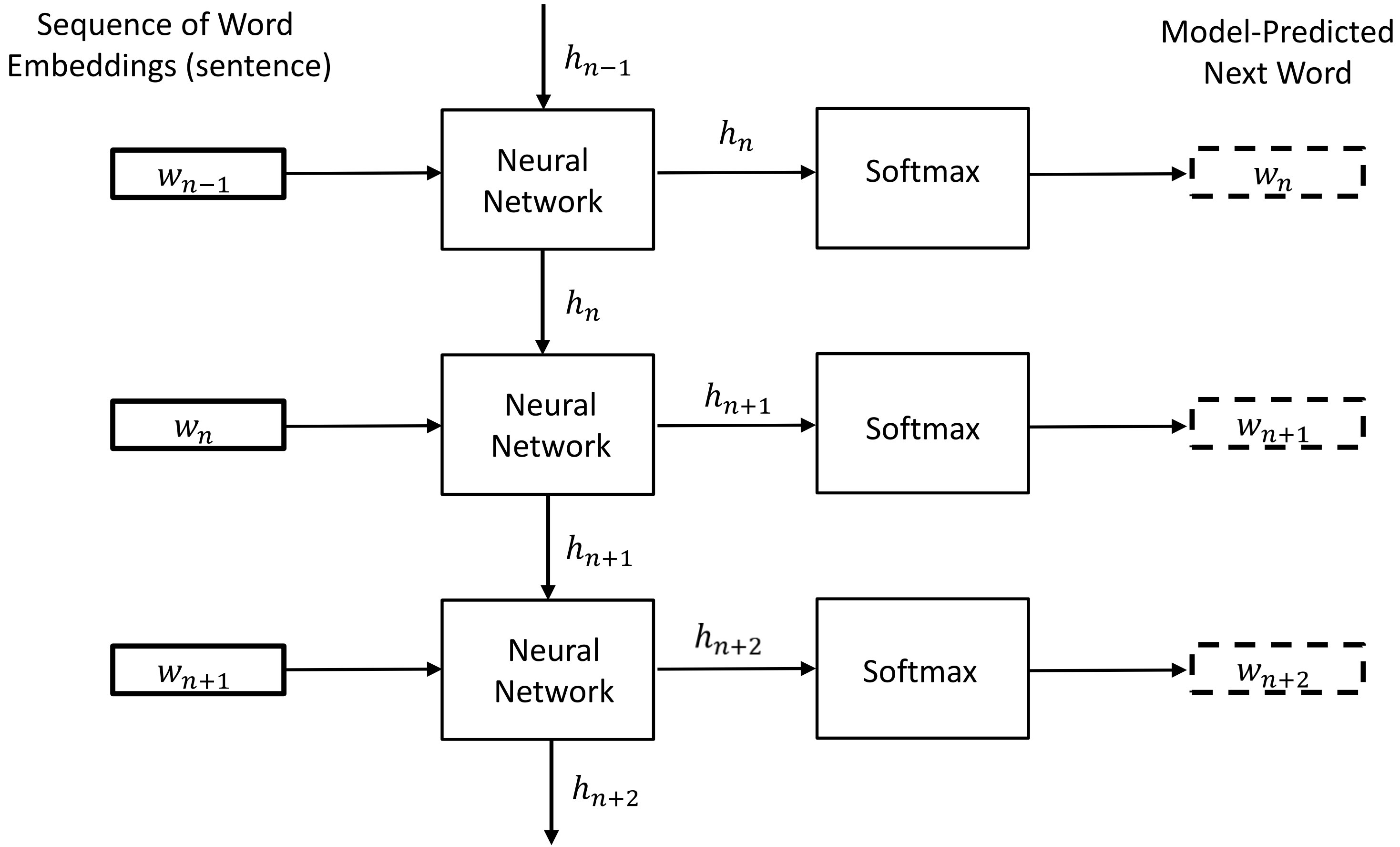
Intuition on Model for Predicting *n*th Word



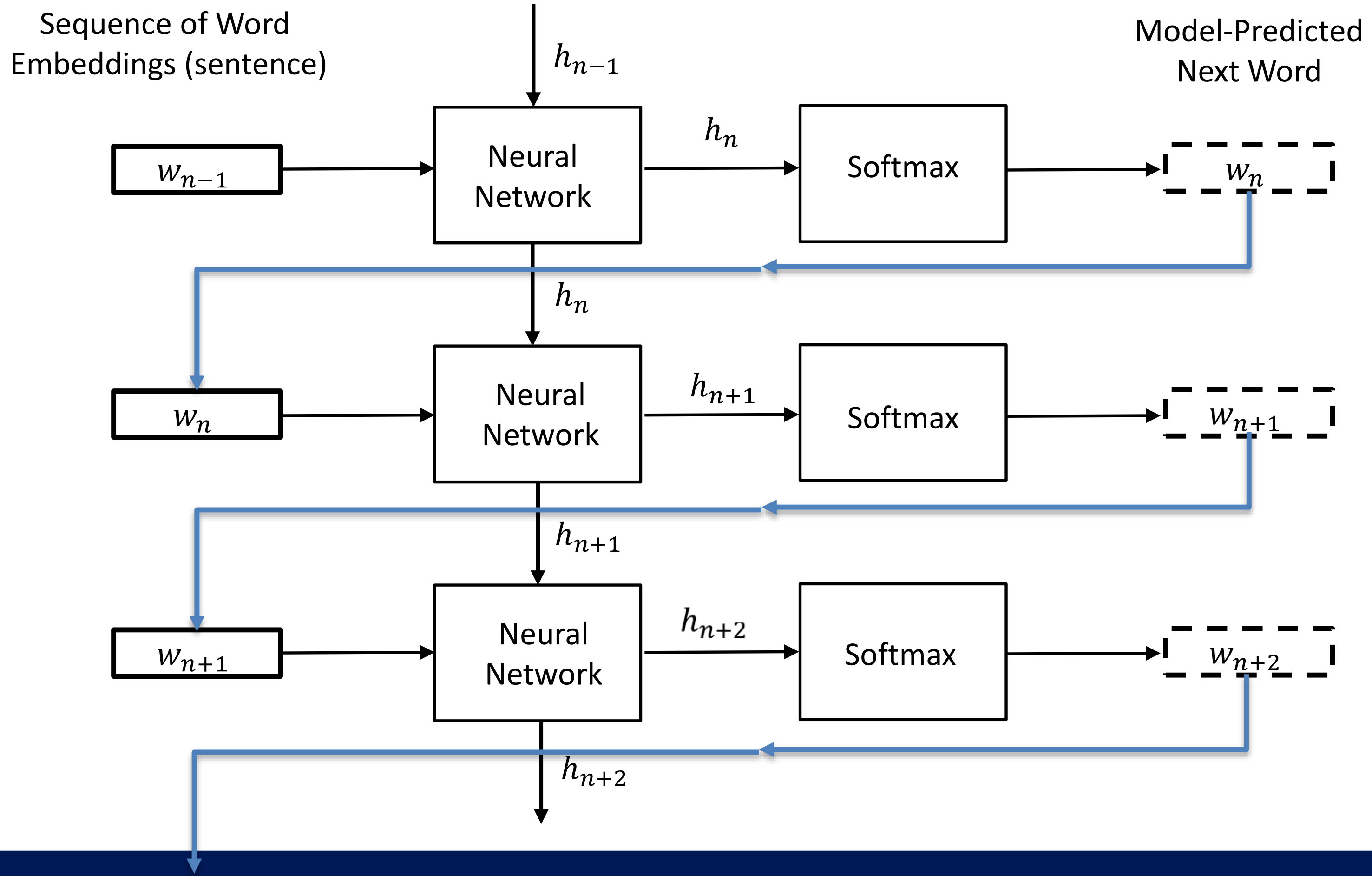
h_{n-1}: Tells us which words were likely prior to selection of previous word (context)

w_{n-1}: Tells us which word was used/selected at point $n-1$ in text, as we predict the nth word

Recurrent Neural Network

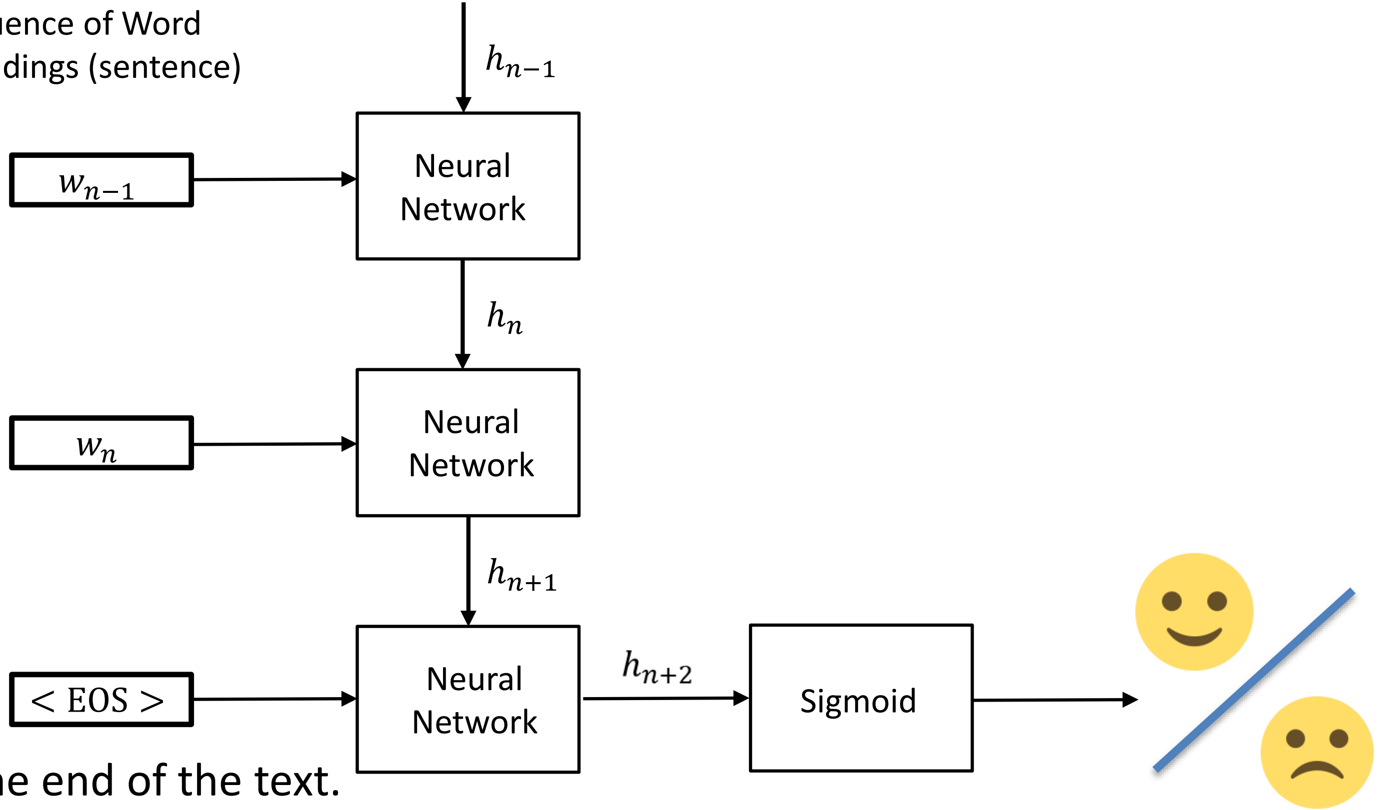


Generating Text



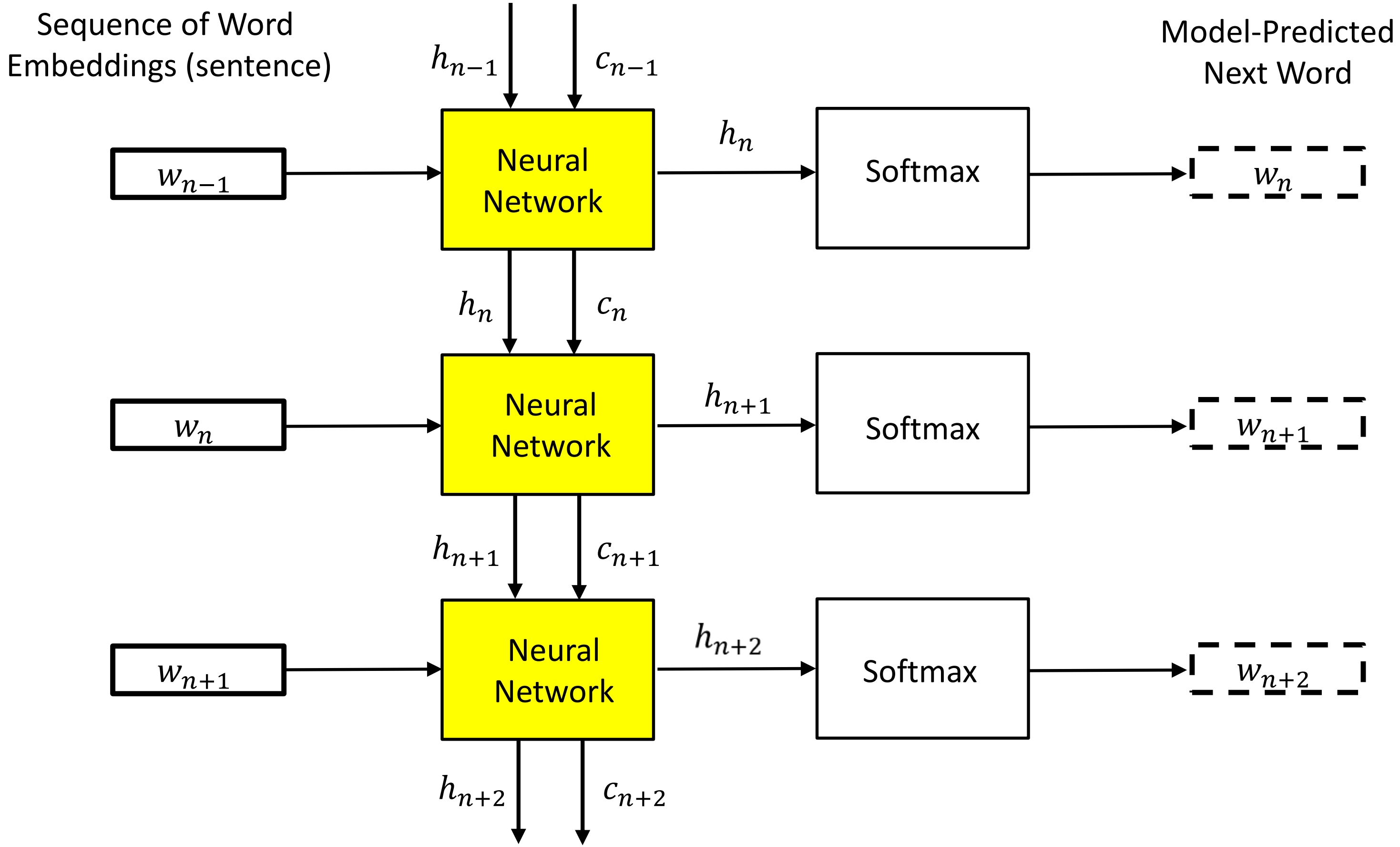
Predicting a Single Output

Sequence of Word Embeddings (sentence)

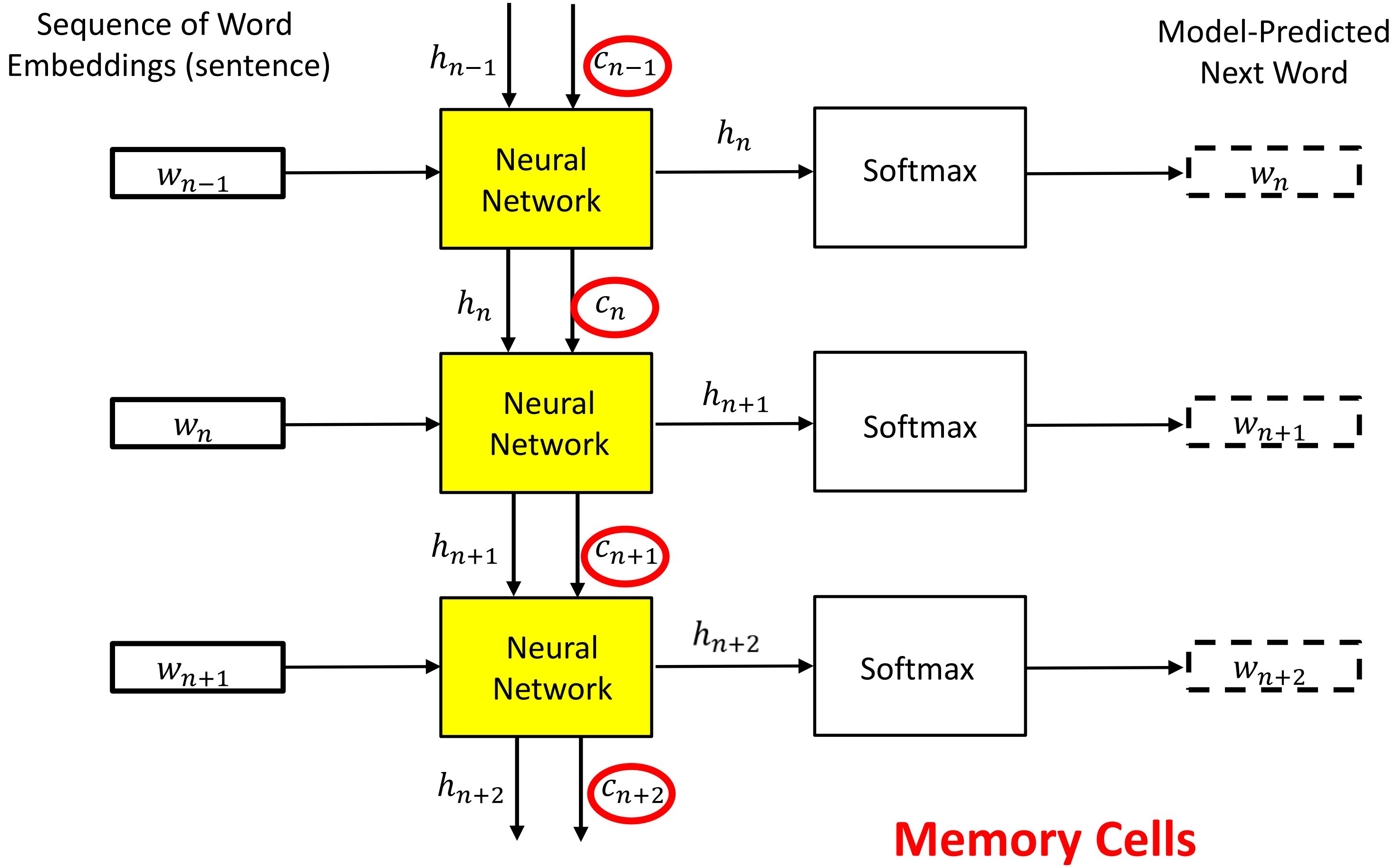


Predict at the end of the text.

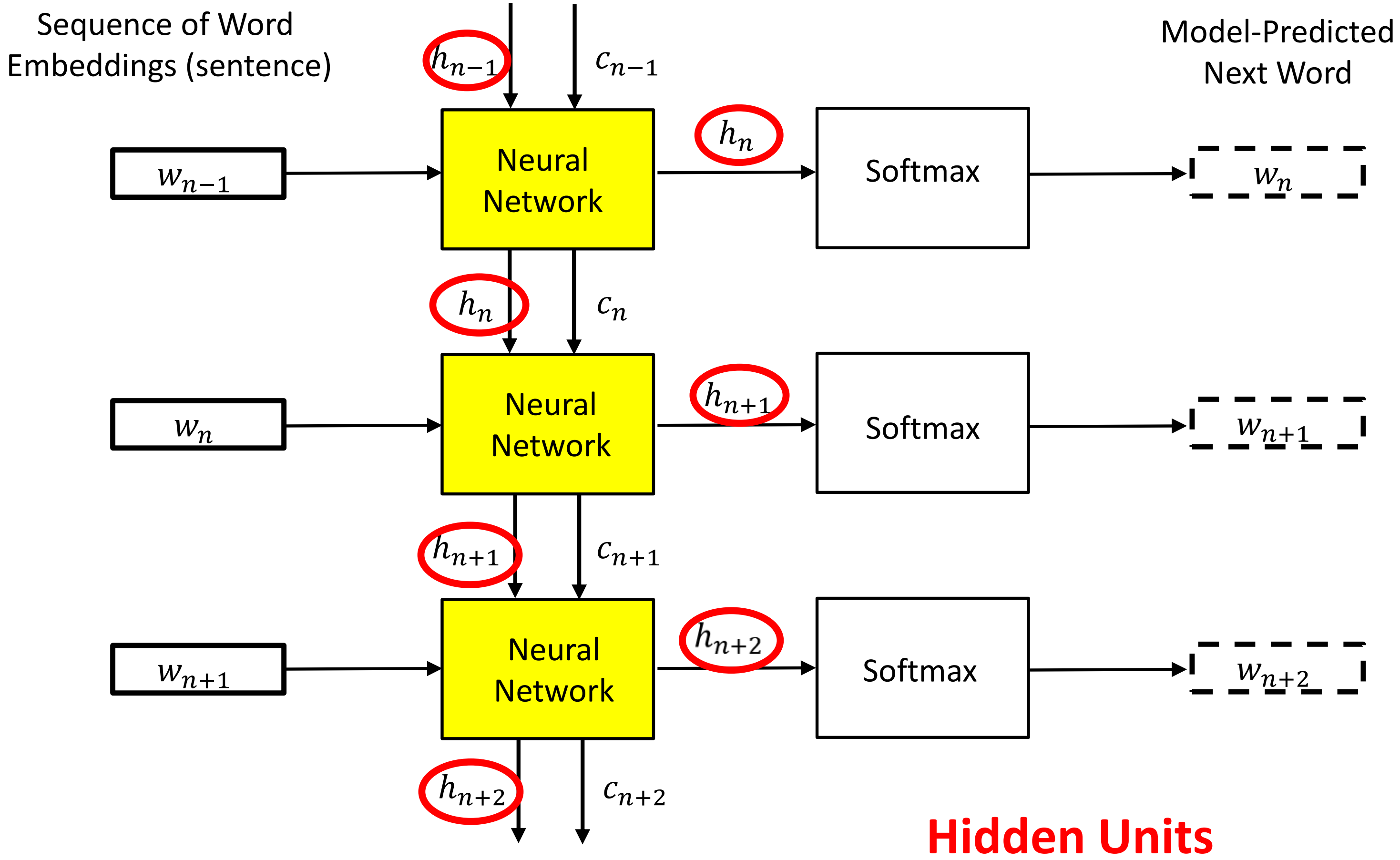
RNN with Memory Cells & Outputs



RNN with Memory Cells & Outputs

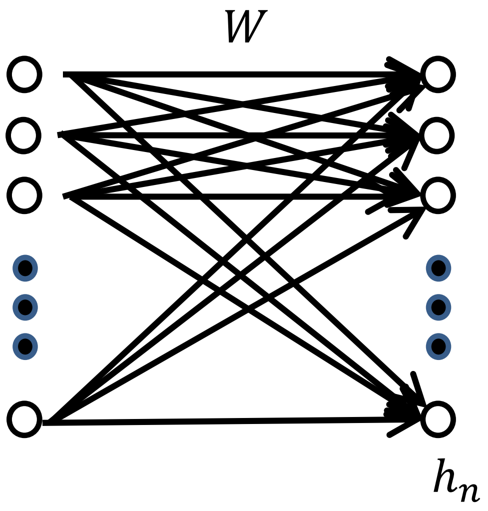


RNN with Memory Cells & Outputs



Recall Notation

$$\underbrace{\boxed{\text{Vector for } w_{n-1}} \boxed{\; h_{n-1} \;}}_{x_{n-1}}$$

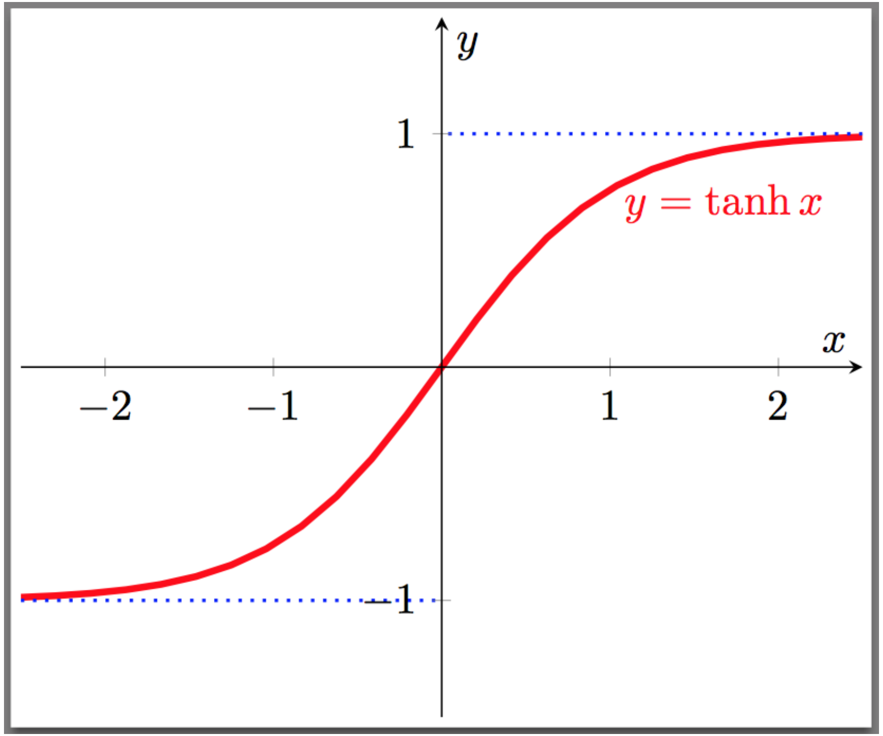


Nonlinear function

$$h_n = f\left(W \cdot x_{n-1} + b\right)$$

Tanh Nonlinear Function

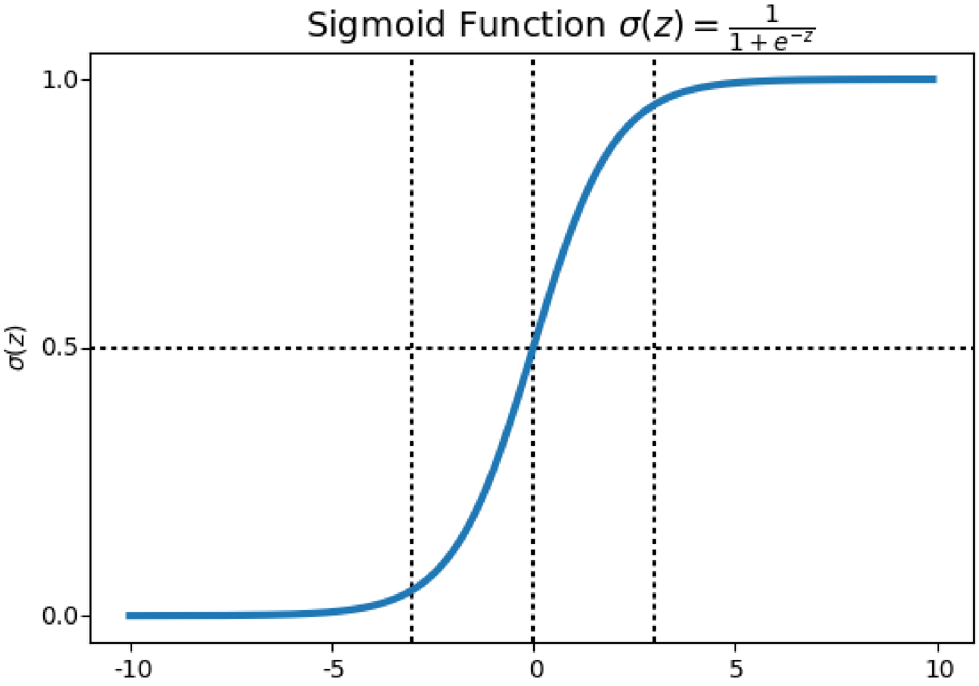
$$h_n = \tanh(W \cdot x_{n-1} + b)$$



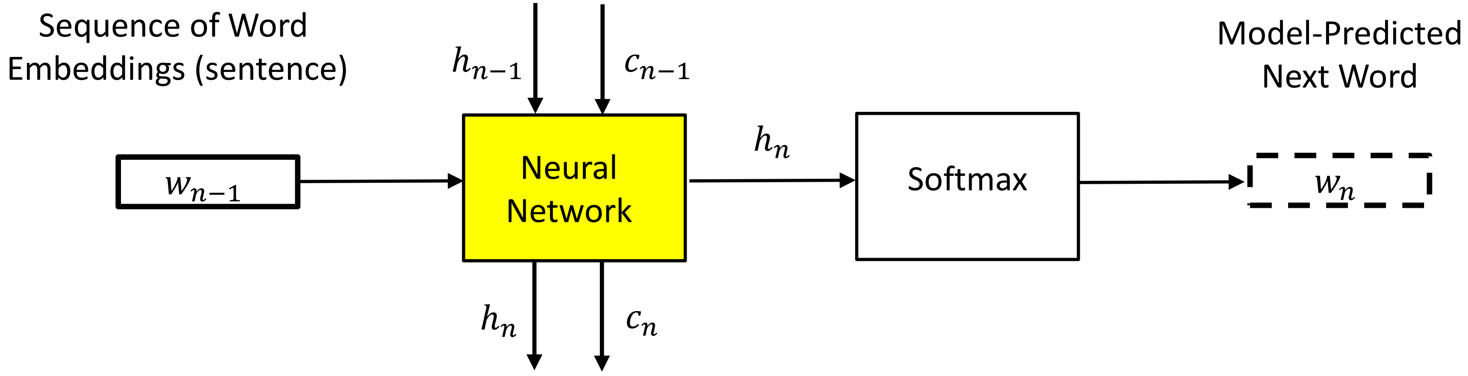
Sigmoid Nonlinear Function

Nonlinear function

$$h_n = \sigma(W \cdot x_{n-1} + b)$$



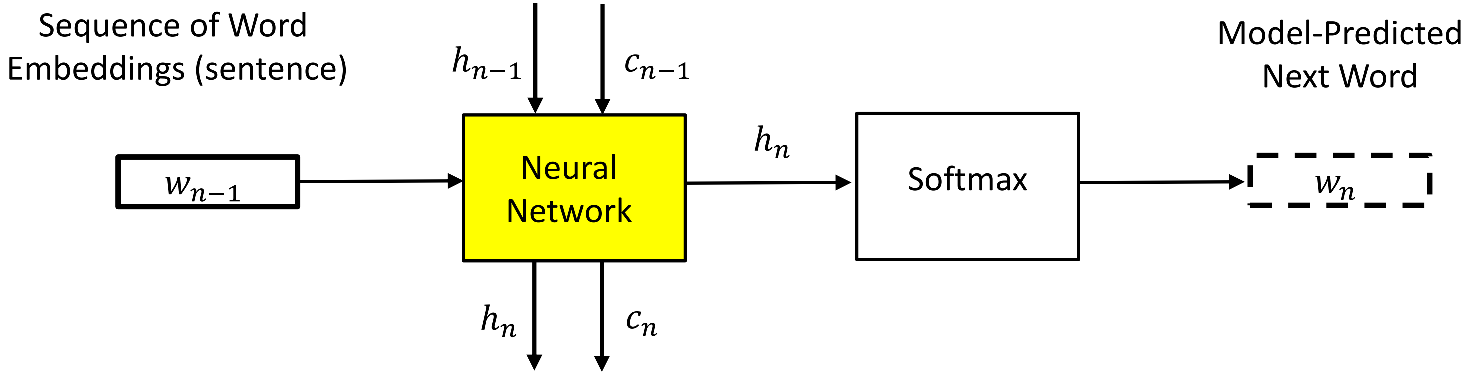
RNN with Memory Cells & Outputs



- As before, concatenate w_{n-1} and h_{n-1} to constitute

$$x_{n-1} = [w_{n-1}, h_{n-1}]$$

RNN with Memory Cells & Outputs



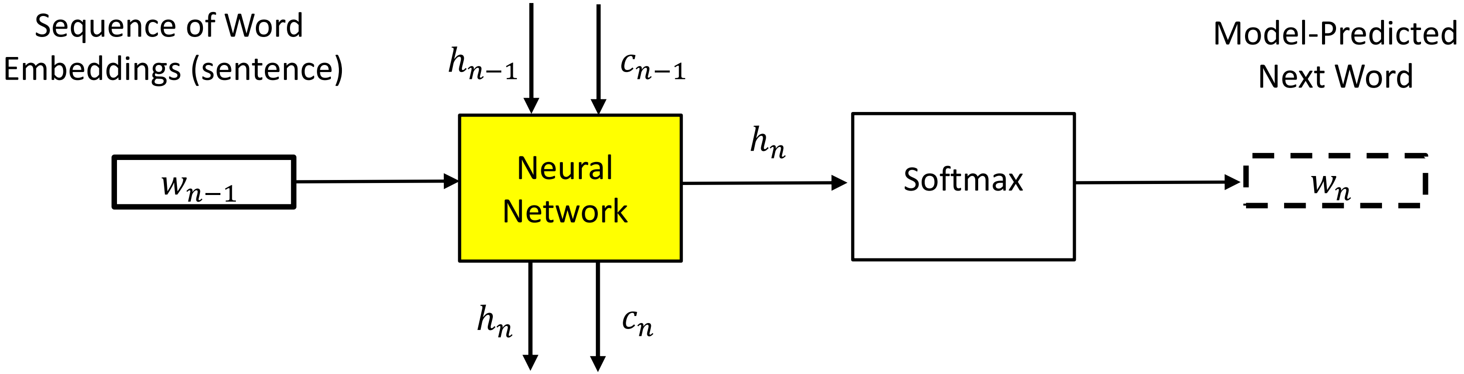
- ❑ As before, concatenate w_{n-1} and h_{n-1} to constitute

$$x_{n-1} = [w_{n-1}, h_{n-1}]$$

- ❑ Introduce three control-gate vectors, $i_n, f_n,$ and o_n:

$$i_n = \sigma(W_i \cdot x_{n-1} + b_i) \qquad f_n = \sigma(W_f \cdot x_{n-1} + b_f) \qquad o_n = \sigma(W_o \cdot x_{n-1} + b_o)$$

RNN with Memory Cells & Outputs



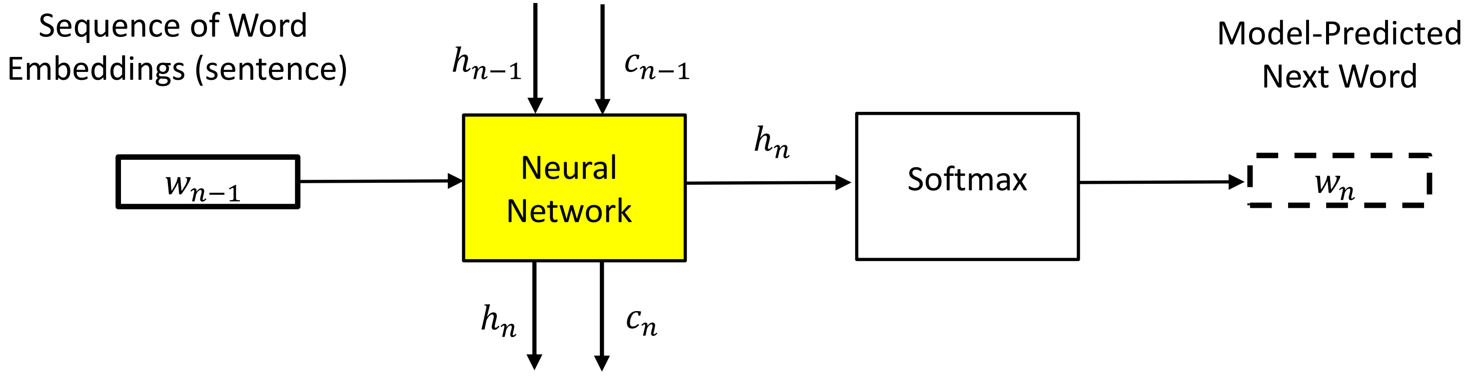
- As before, concatenate w_{n-1} and h_{n-1} to constitute

$$x_{n-1} = [w_{n-1}, h_{n-1}]$$

- Introduce three control-gate vectors, $i_n, f_n,$ and o_n:

$$i_n = \sigma(W_i \cdot x_{n-1} + b_i) \qquad f_n = \sigma(W_f \cdot x_{n-1} + b_f) \qquad o_n = \sigma(W_o \cdot x_{n-1} + b_o)$$

RNN with Memory Cells & Outputs

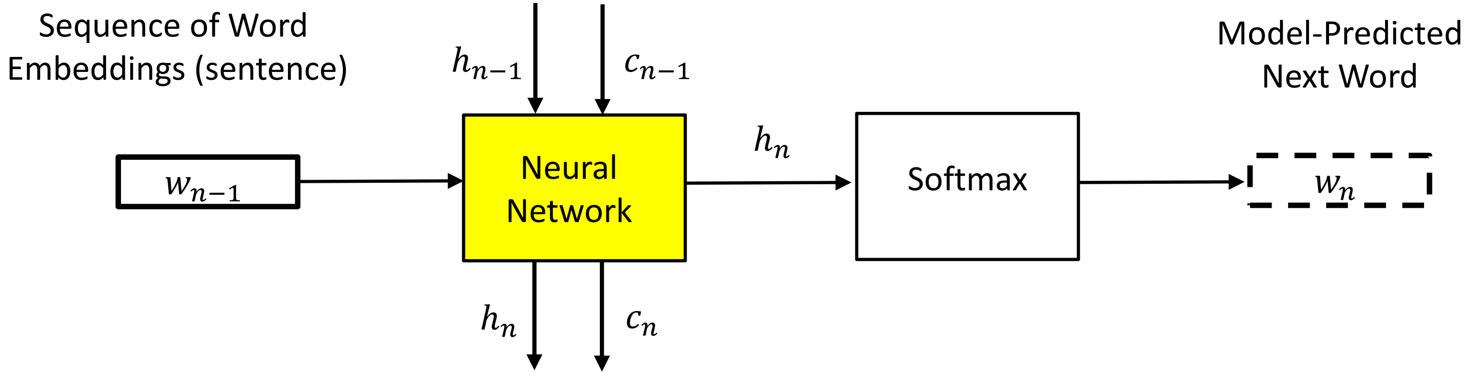


$$o_n = \sigma(W_o \cdot x_{n-1} + b_o) \qquad f_n = \sigma(W_f \cdot x_{n-1} + b_f) \qquad i_n = \sigma(W_i \cdot x_{n-1} + b_i)$$

- ❑ We introduce three distinct control neural networks, with respective weights and biases W_o, W_f, W_i, b_o, b_f, and b_i

- ❑ The outputs of these three neural networks, o, f, i, are each vectors, the components of which are each between zero and one

RNN with Memory Cells & Outputs



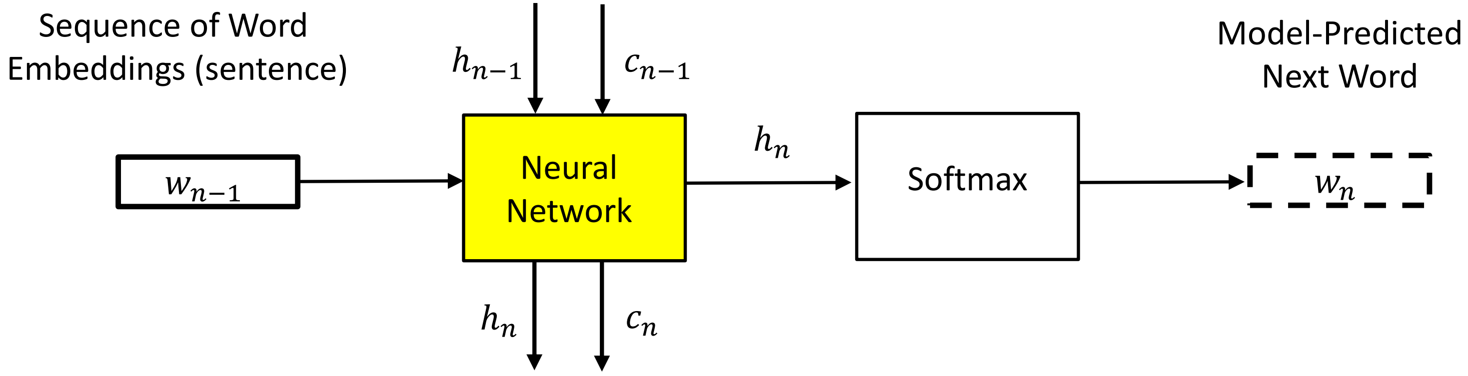
- New estimate of memory-cell values

$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c)$$

- Update the memory cell as

$$c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$

RNN with Memory Cells & Outputs



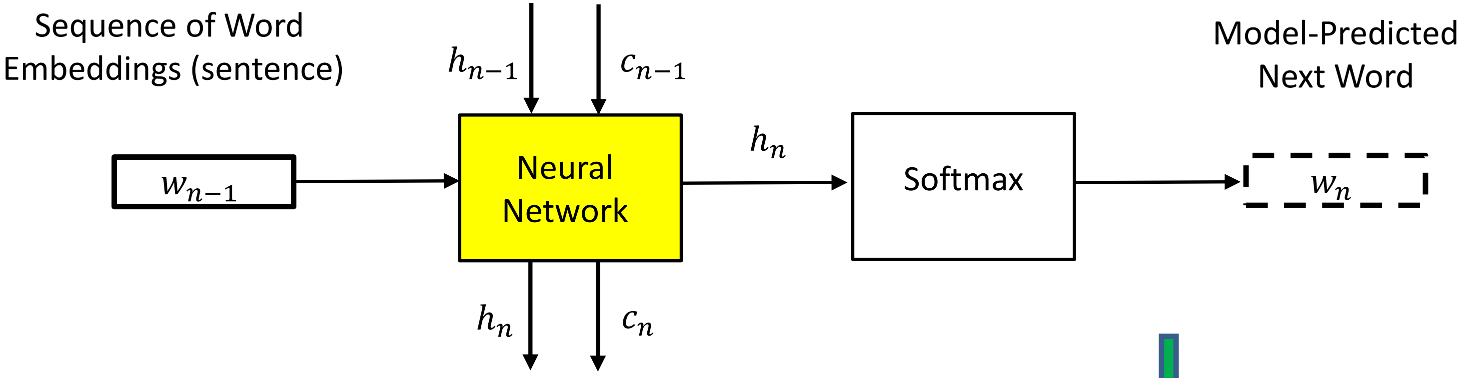
- New estimate of memory-cell values

$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c)$$

- Update the memory cell as

$$c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$

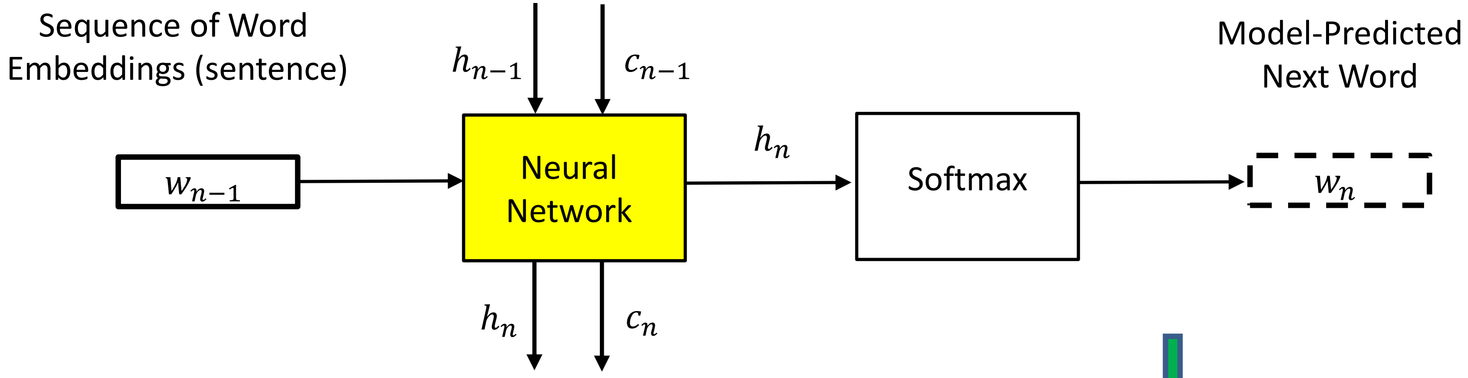
RNN with Memory Cells & Outputs



$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c)$$

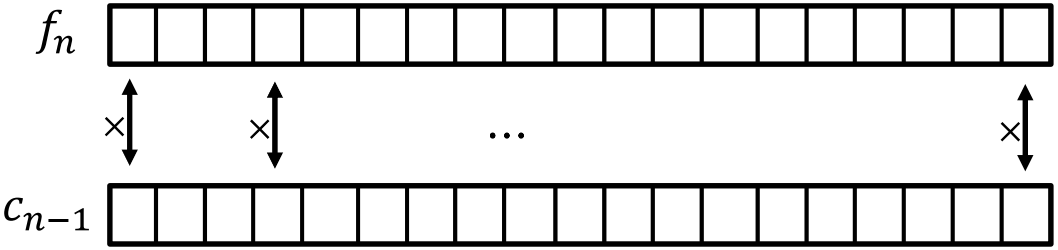
$$c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$

RNN with Memory Cells & Outputs

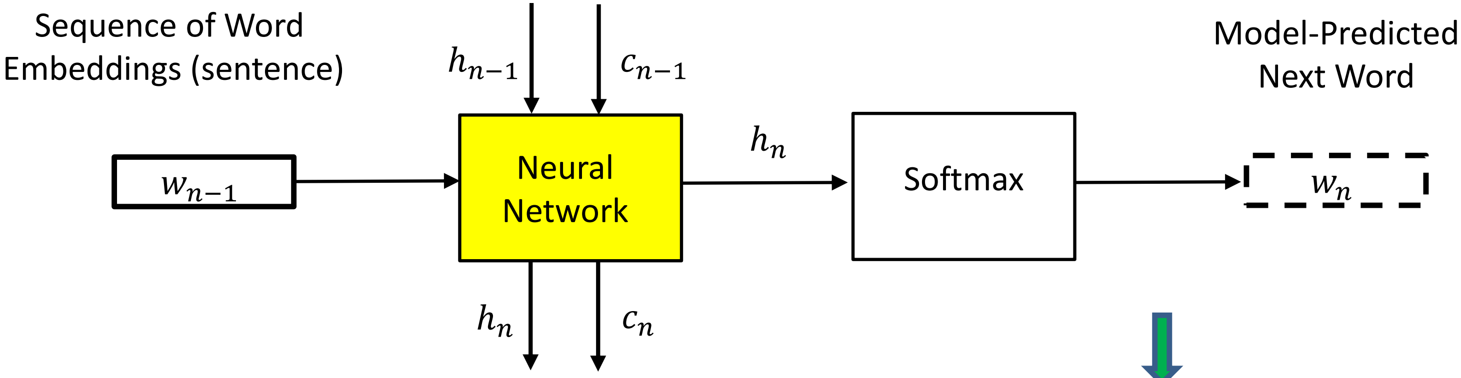


$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c)$$

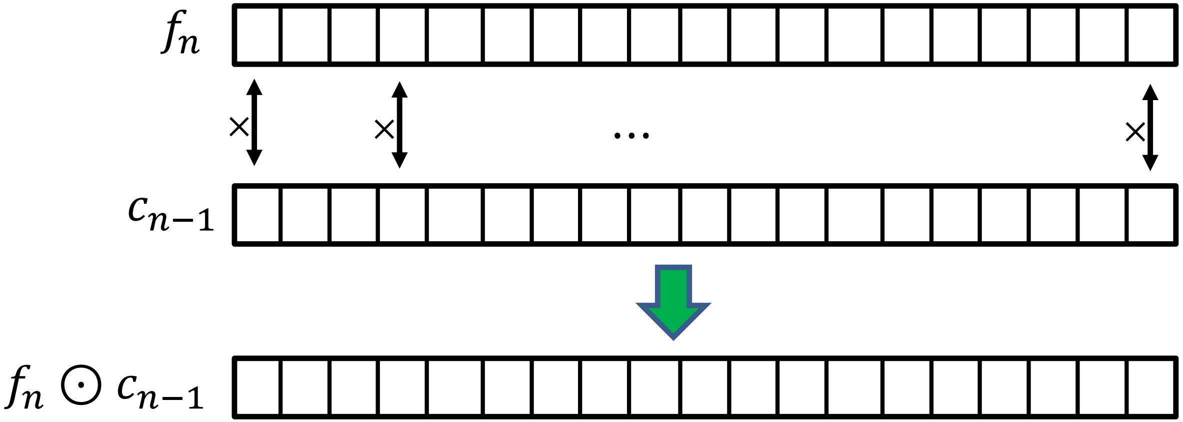
$$c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$



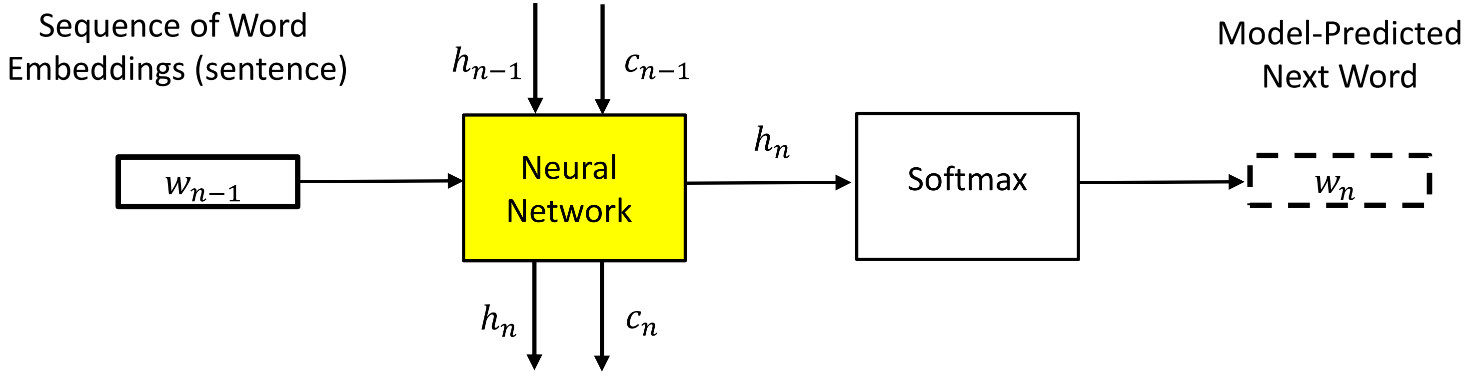
RNN with Memory Cells & Outputs



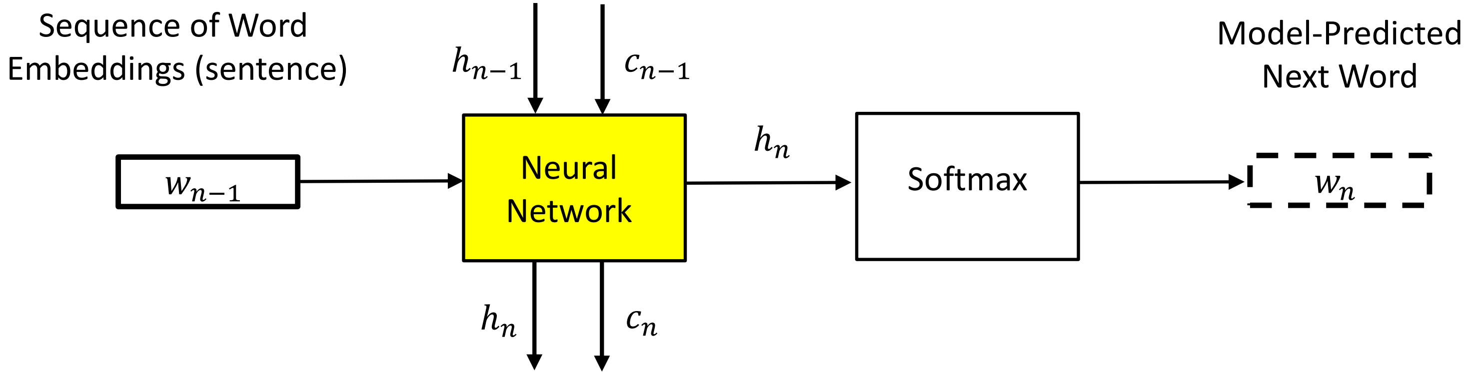
$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c) \qquad c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$



RNN with Memory Cells & Outputs

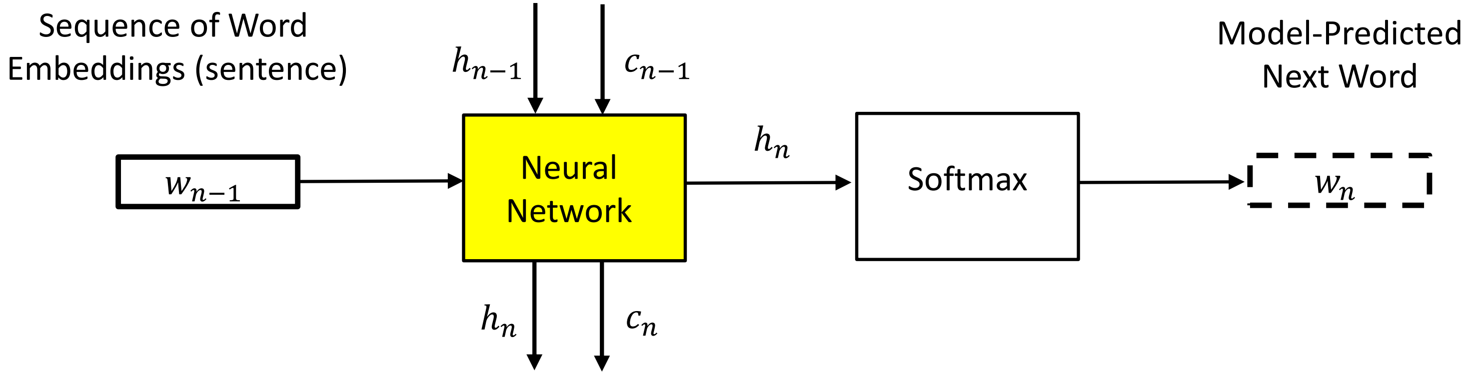


RNN with Memory Cells & Outputs



$$x_{n-1} = [w_{n-1}, h_{n-1}]$$

RNN with Memory Cells & Outputs

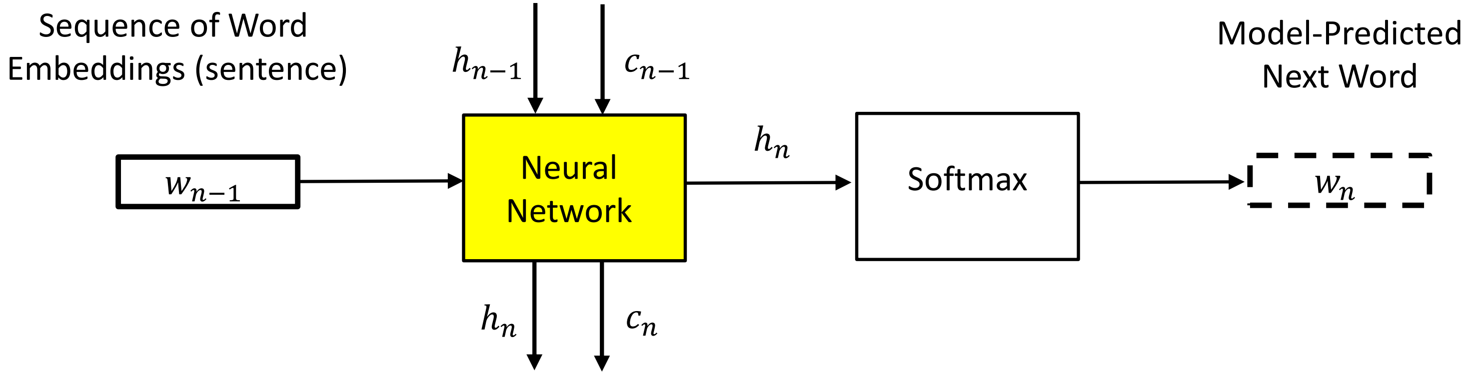


$$o_n = \sigma(W_o \cdot x_{n-1} + b_o) \qquad f_n = \sigma(W_f \cdot x_{n-1} + b_f) \qquad i_n = \sigma(W_i \cdot x_{n-1} + b_i)$$

$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c)$$

Four Neural Networks

RNN with Memory Cells & Outputs



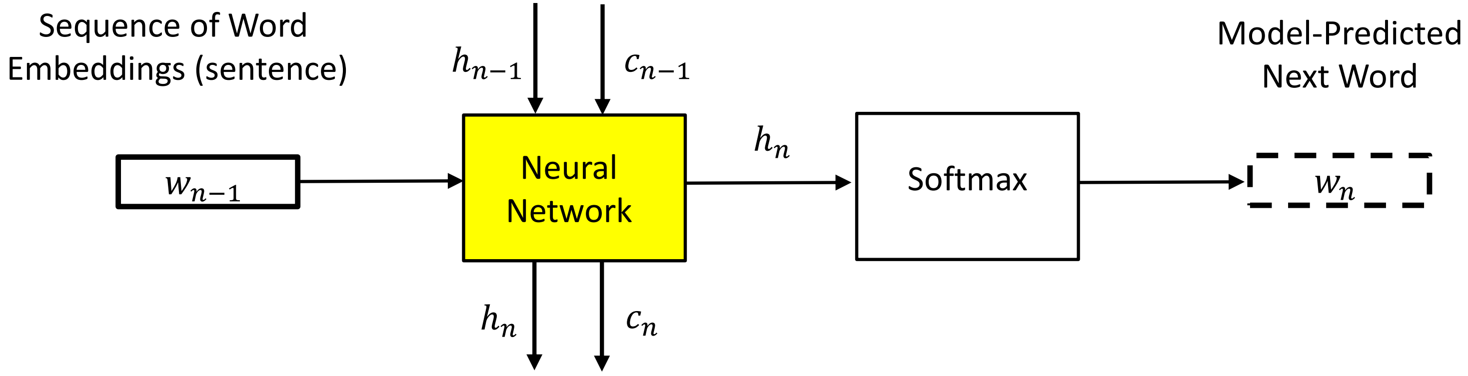
$$o_n = \sigma(W_o \cdot x_{n-1} + b_o) \qquad f_n = \sigma(W_f \cdot x_{n-1} + b_f) \qquad i_n = \sigma(W_i \cdot x_{n-1} + b_i)$$

$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c)$$

$$c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$

Update Memory Cell

RNN with Memory Cells & Outputs



$$o_n = \sigma(W_o \cdot x_{n-1} + b_o) \qquad f_n = \sigma(W_f \cdot x_{n-1} + b_f) \qquad i_n = \sigma(W_i \cdot x_{n-1} + b_i)$$

$$\tilde{c}_n = \tanh(W_c \cdot x_{n-1} + b_c)$$

$$c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$

Update Memory Cell

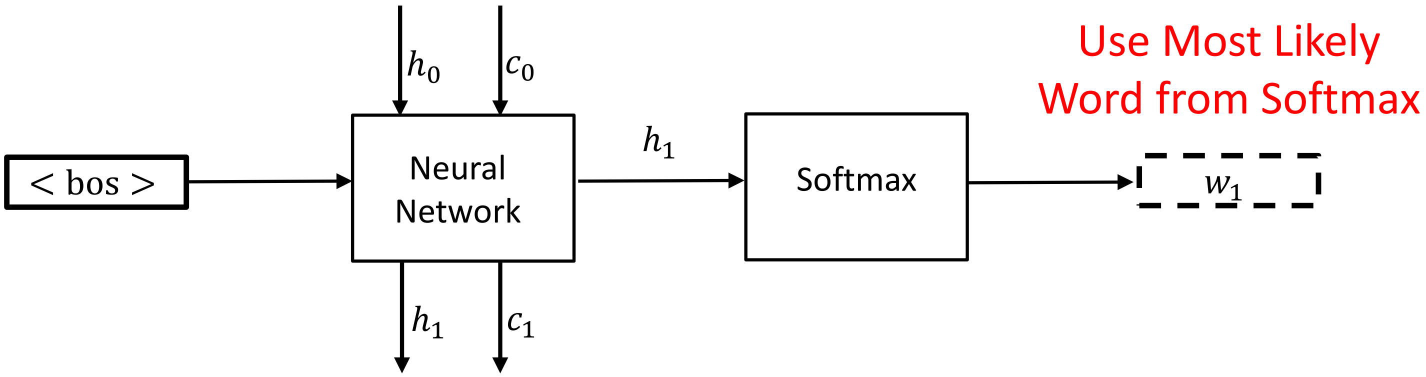
$$h_n = o_n \odot \tanh(c_n)$$

Output Hidden Vector

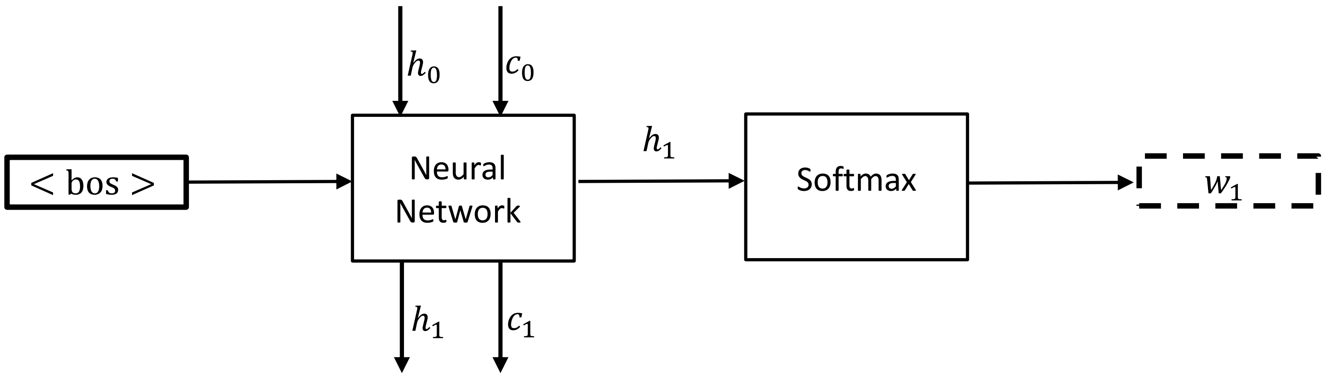
LSTM-Based Text Synthesis

- Assume we have trained the parameters of an LSTM using a large document corpus

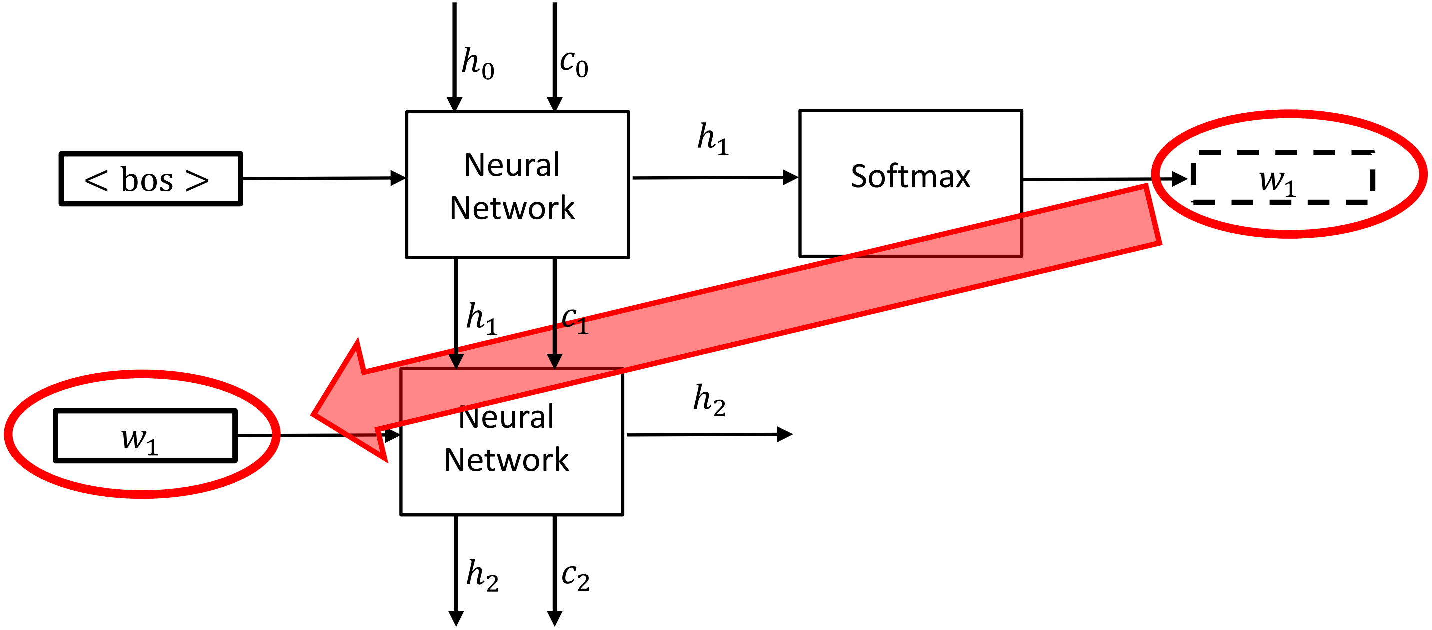
- How may we use the LSTM to synthesize text?



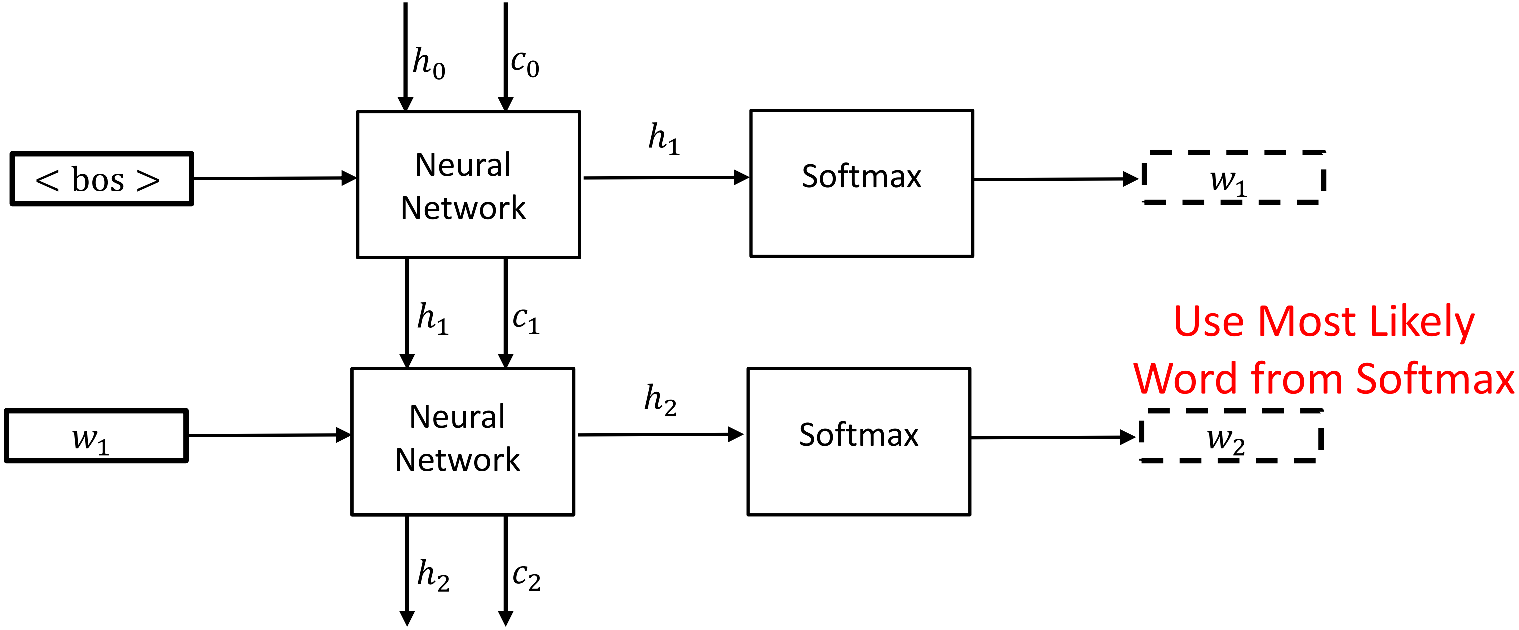
LSTM-Based Text Synthesis



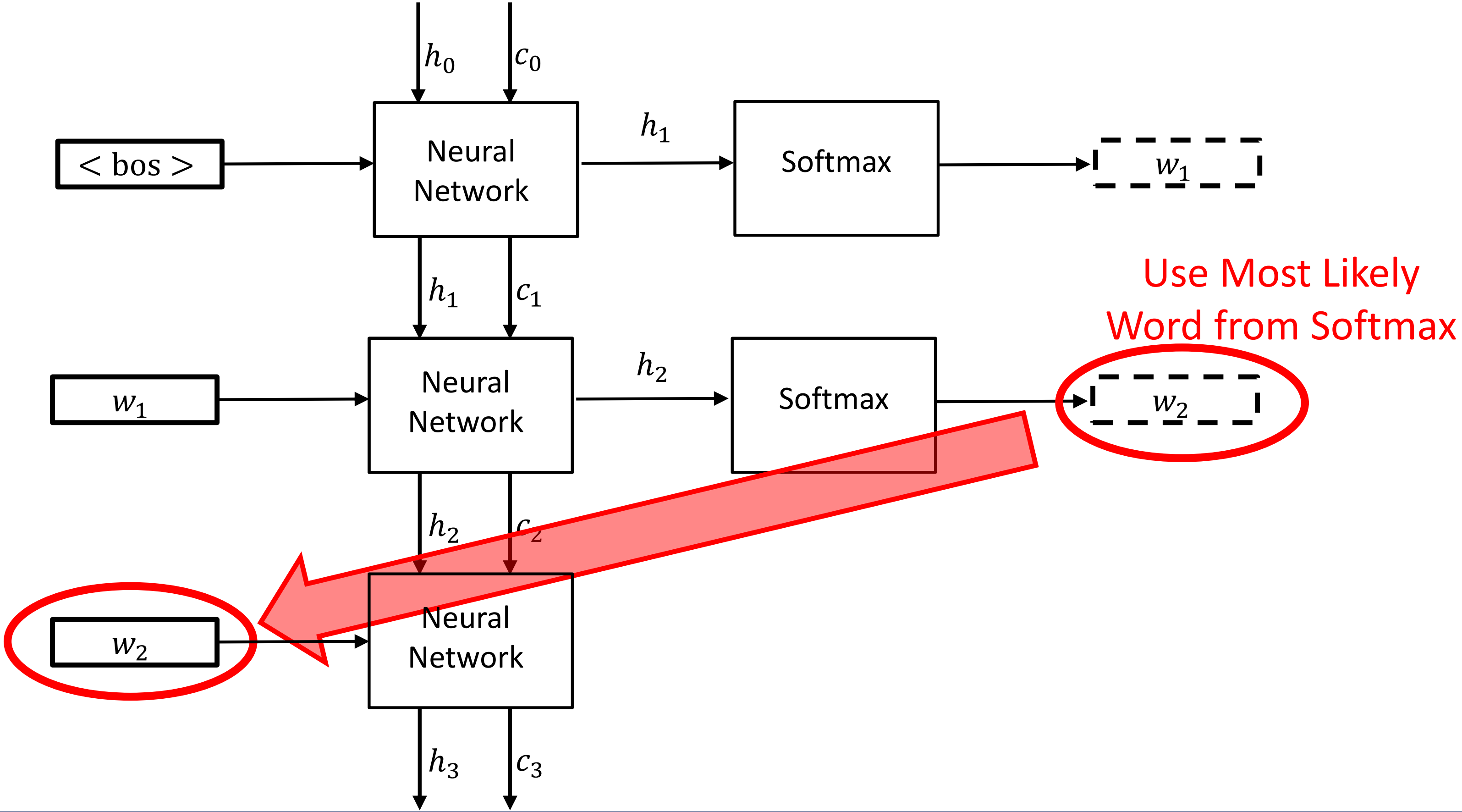
LSTM-Based Text Synthesis



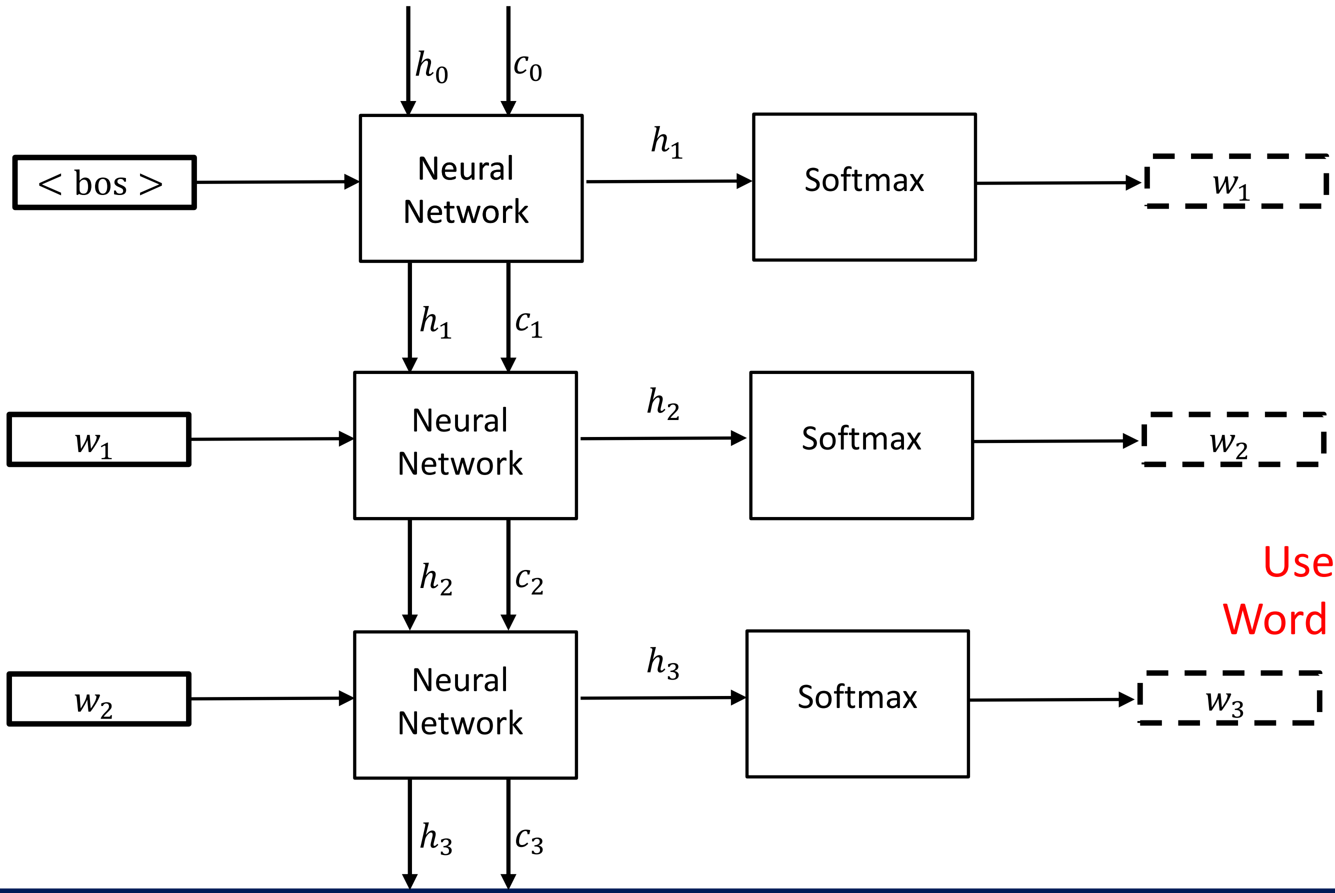
LSTM-Based Text Synthesis



LSTM-Based Text Synthesis

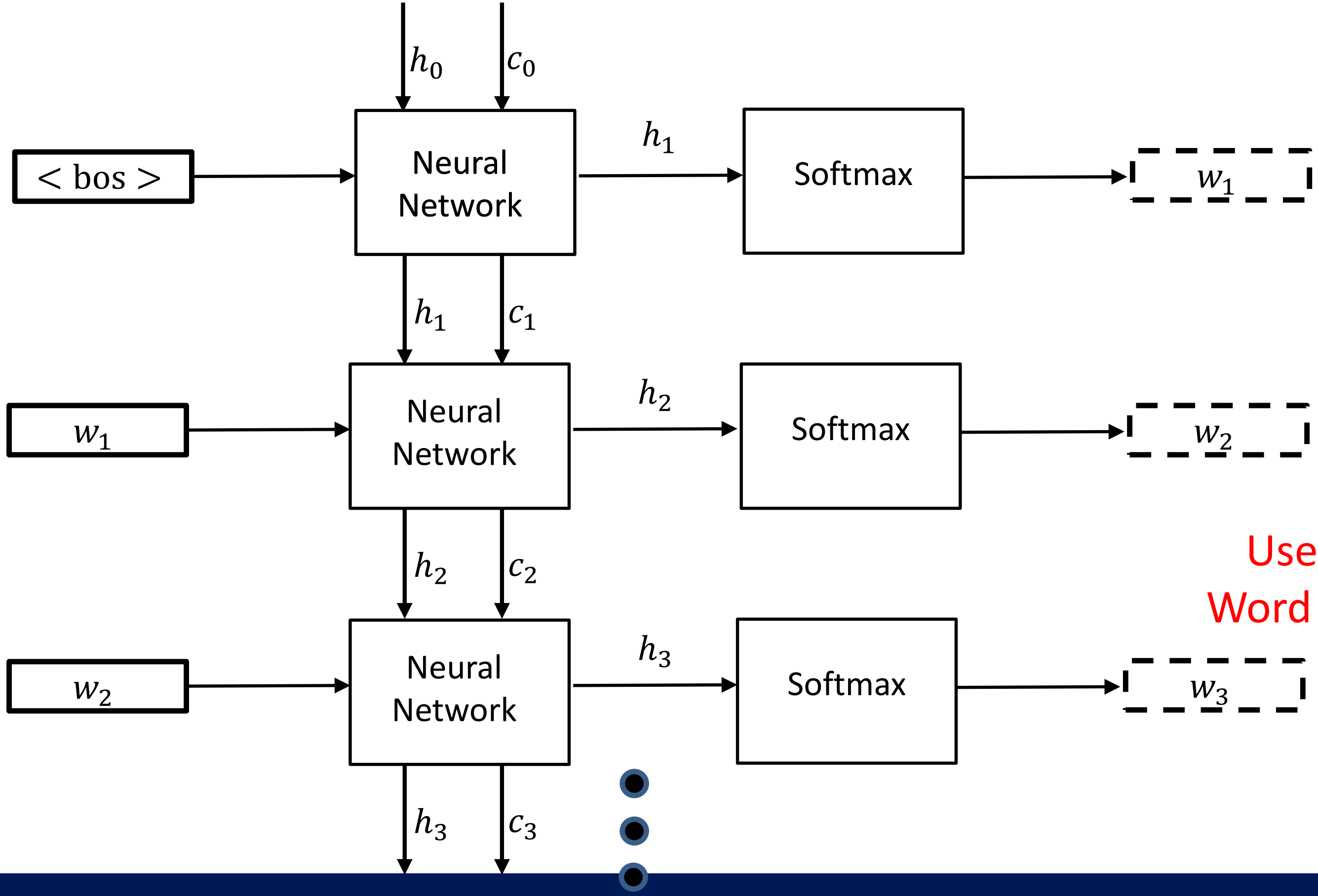


LSTM-Based Text Synthesis



Use Most Likely Word from Softmax

LSTM-Based Text Synthesis



Use Most Likely Word from Softmax

Image-to-Caption


a cow is standing in front of a store


a group of elephants standing next to each other

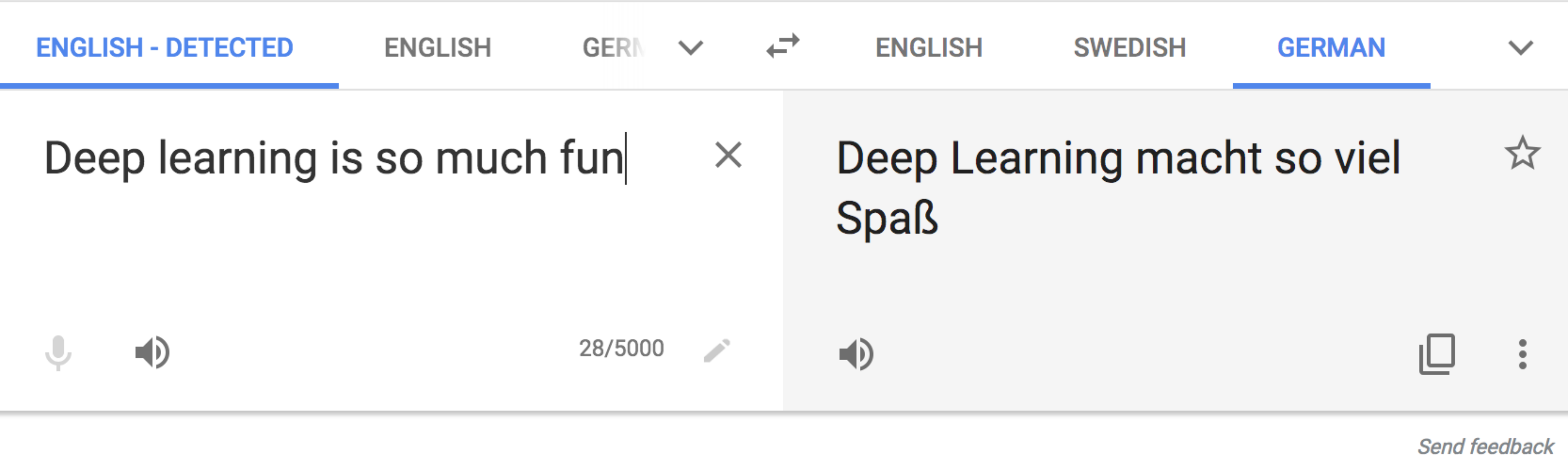

a table that has wooden spoons on it


a cat is eating some kind of food


a bunch of bananas are sitting on a table


a motorcycle is parked next to a window

Text Translation



translate.google.com

Conclusions

- Word embeddings and recurrent neural networks are currently the cornerstones of natural language processing
- Nearly all text systems are based on these techniques (e.g. google translate, chatbots, etc.)
- Many more versions of Recurrent Neural Networks
  - Long-Short Term Memory (LSTM) to build a “memory”
  - Stacking recurrent units to make deep recurrent networks
- The Transformer Network is a new variant that achieves similar goals