https://github.com/bandpooja/Assignment_1

bandpooja Update README.md — 6cf47c2 20 days ago — 24 commits

Data	Update README.md	2 months ago
cache	Update README.md	
utils	Add files via upload	
.gitignore	Add files via upload	
README.md	Update README.md	20 days ago
assignment1.ipynb	Update assignment1.ipynb	2 months ago
assignment1.py	Update assignment1.py	2 months ago
requirements.txt	Add files via upload	2 months ago

Readme: make a connection between text and code
Notebook: Sample setup and run
Notebook: NOT the main codebase

README.md

Assignment 1

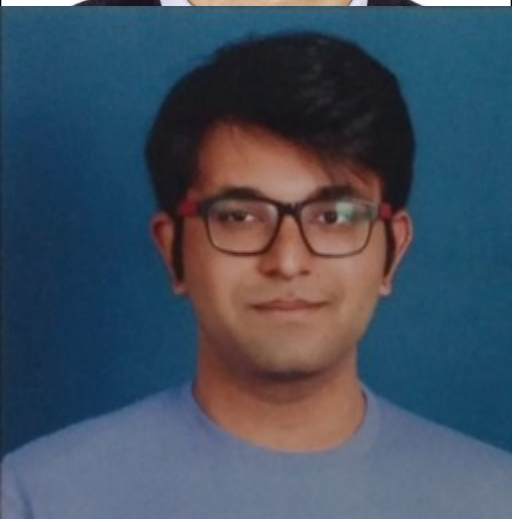
For Assignment 1 I have used the *Levenshtein distance* of all the words in the **Brikbeck** corpus with every word in **WordNet** corpus. The task aims to find correct spelling of mis-spelled words in the Brikbeck corpus.

Setup

To setup the environment which I used to create this project. Just run

```
pip install -r requirements.txt
```

Assumptions



https://github.com/EhsanMohd/COMP8730_Proposed_Solution/blob/main/COMP8730_Proposed_Solution.ipynb

Performing singular-valued decomposition (SVD)

We now perform feature reduction on the processed dataset using the singular-valued decomposition (SVD) feature reduction approach. Here, we use the TruncatedSVD class to do so.

```
start_time = time.time()

x_reshaped = model.svd_process(count_processed_data, n=100)

end_time = time.time()
svd_process_time = end_time - start_time
print("Code block took " + str(svd_process_time) + " sec")
times.append(("svd_process", svd_process_time))
```

```
print("Shape of the reshaped data: ", x_reshaped.shape)
```

Notebook: Sample setup and run
Notebook: Make a connection between text and code

Notebook: NOT the main codebase

Building the long short-term memory (LSTM) neural network model

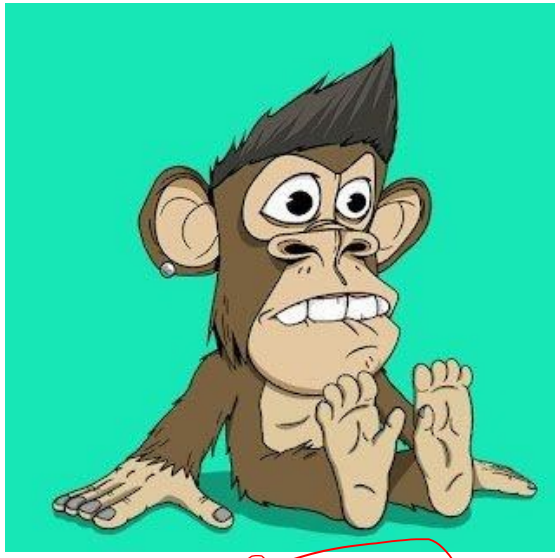
We used the data now processed using truncated SVD in a neural network model comprising an LSTM layer.

```
# Setting units and dropout parameters
units = 100
dropout = 0.2
```

Checking the time taken to construct and build a model over the entire dataset.

```
start_time = time.time()

lstm_model = model.build_model(units=units, dropout=dropout)
lstm_model_fit = lstm_model.fit(x_reshaped, np.array(processed_labels), epochs = 1)

end_time = time.time()
lstm_model_build_fit_time = end_time - start_time
print("Code block took " + str(lstm_model_build_fit_time) + " sec")
times.append(("lstm_build_model_fit", lstm_model_build_fit_time))
```

src
cmn
main.py



https://github.com/parmarsuraj99/COMP8730_research_project

requirements.txt

Colab

2483 lines (2483 sloc) | 84.8 KB

Raw Blame

Notebook: Sample setup and run
Notebook: NOT the main codebase
Notebook: Successful execution

Notebook: Make a connection between text and code

Proposed Solution

```
In [ ]: !pip install transformers
        !pip install datasets
        !pip install apache_beam
        !pip install sentencepiece
```

```
In [ ]:
```

```
In [1]: import random
        import re

        import gc

        random.seed(0)
        print(random.random())
```

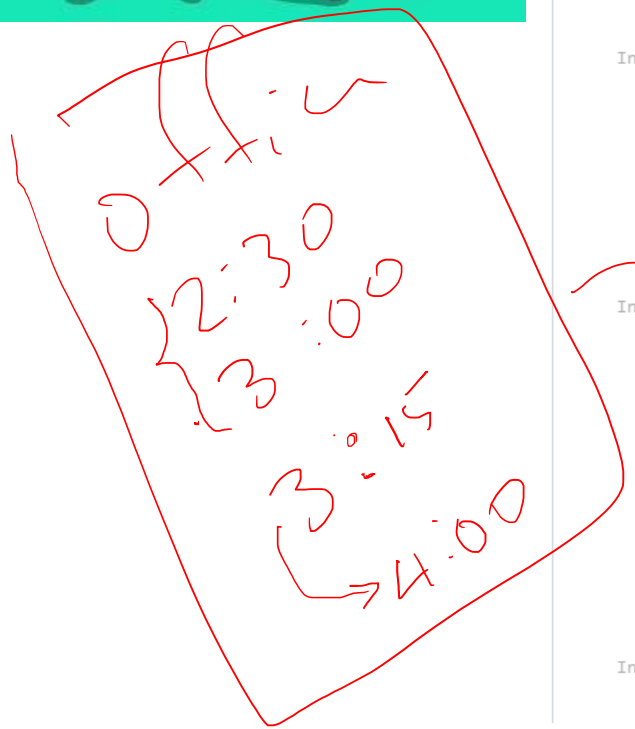
```
In [2]: from datasets import load_dataset

        dataset = load_dataset("oscar", "unshuffled_deduplicated_sa")
```

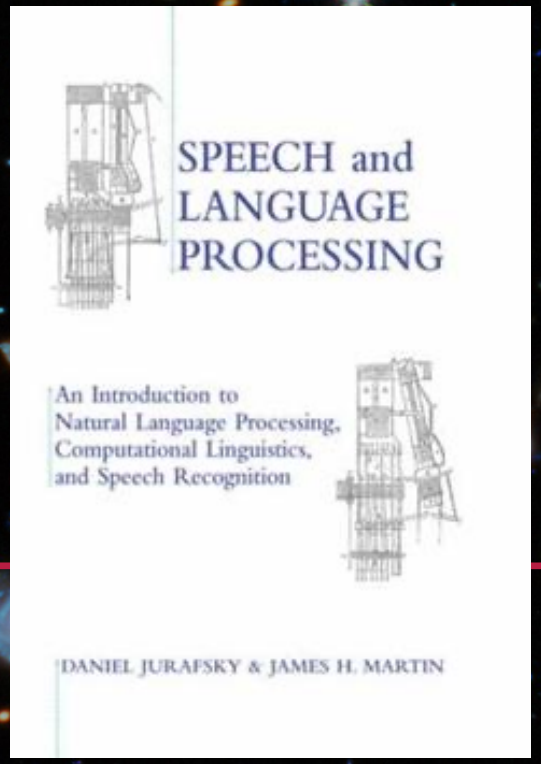
```
Downloading builder script: 14.8kB [00:00, 7.38MB/s]
Downloading metadata: 3.07MB [00:00, 85.9MB/s]
```

```
Downloading data: 100%|██████████| 81.0/81.0 [00:00<00:00, 77.9kB/s]
Downloading data: 100%|██████████| 7.27M/7.27M [00:00<00:00, 13.6MB/s]
Downloading data files: 100%|██████████| 1/1 [00:00<00:00,  1.05it/s]
```

```
100%|██████████| 1/1 [00:00<00:00, 90.93it/s]
```

```
In [ ]: text = dataset["train"]["text"]
```





Neural Networks and Neural LM

CH07

Neural Language Model

Neural Language Model

Bigram Language Model

$$P(w_i | w_{i-1}) = \frac{\#(w_i, w_{i-1})}{\#(w_i)}$$

$$f: (\{0,1\}^{|V|})^2 \rightarrow R^{[0,1]}; \; f(w_i, w_{i-1}) = P(w_i | w_{i-1})$$

Input: one-hot vector

w_{i-1} = [0 0 0 … 1 0… 0 0]

w_i = [0 0 0 … 0 1 … 0 0]

Output:

[y]

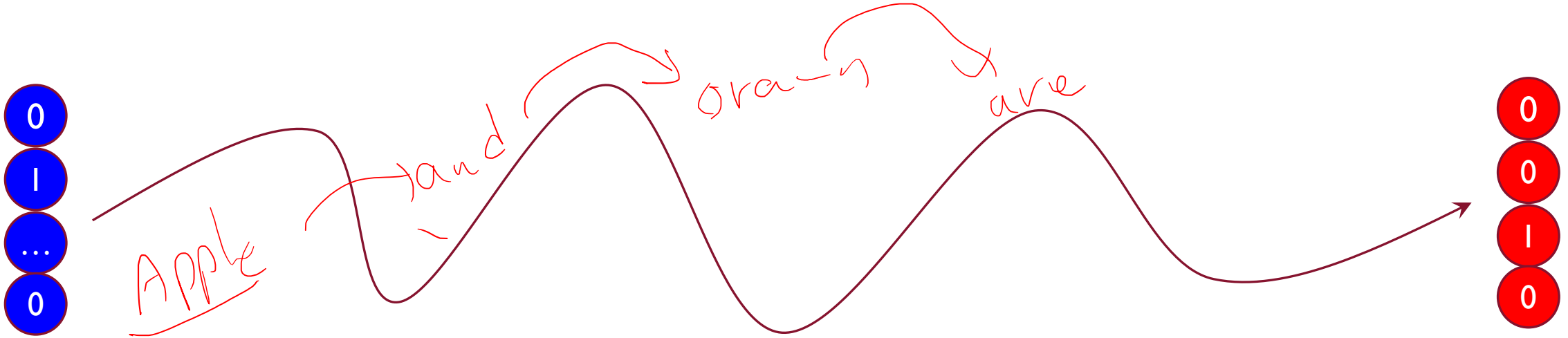
Neural Language Model

Bigram Language Model

No use! Why?

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1})$ = ? Unknown Initially

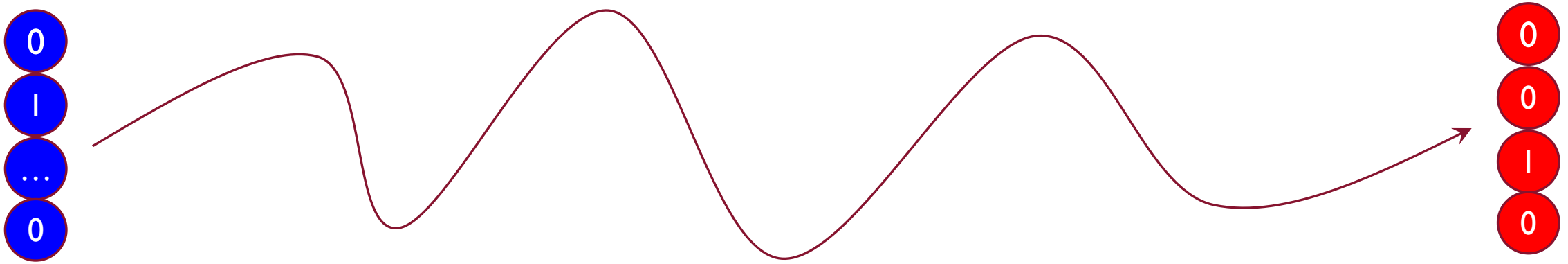
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$f: \{0,1\}^{|V|} \rightarrow \{0,1\}^{|V|}$; $f(w_{i-1}) \rightarrow w_i$

$[0\ 0\ 0\ \ldots\ 1\ 0\ \ldots\ 0\ 0\ 0] \rightarrow [0\ 0\ 0\ \ldots\ 0\ 1\ \ldots\ 0\ 0\ 0]$

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1})$ = ? Unknown Initially

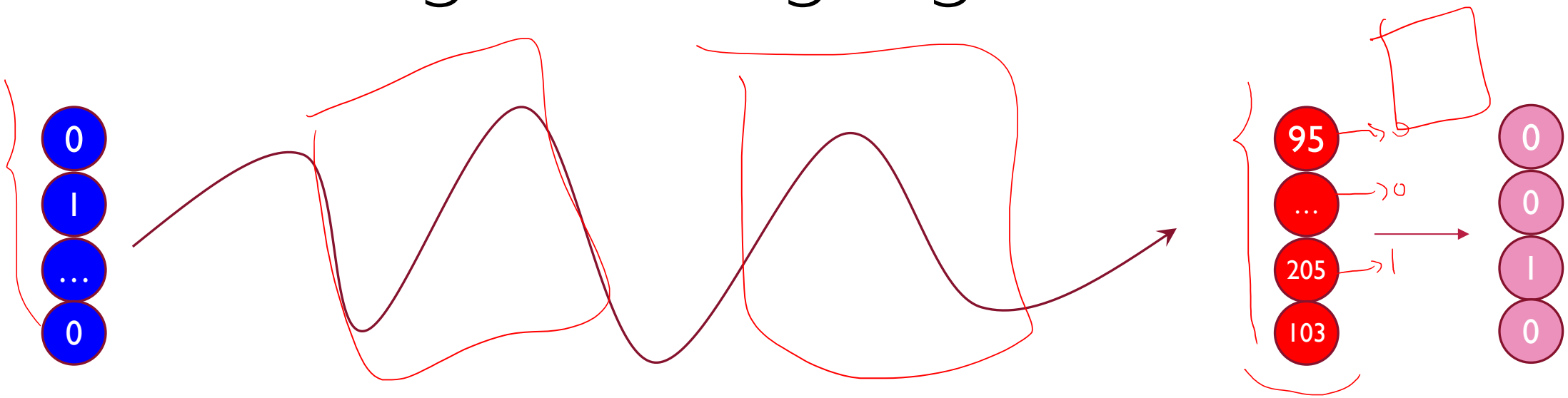
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$f: \{0,1\}^{|V|} \rightarrow \{0,1\}^{|V|}$; $f(w_{i-1}) \rightarrow$ class of w_i

[0 0 0 … 1 0 … 0 0 0] → [0 0 0 … 0 1 … 0 0 0]

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1}) = ?$ Unknown Initially

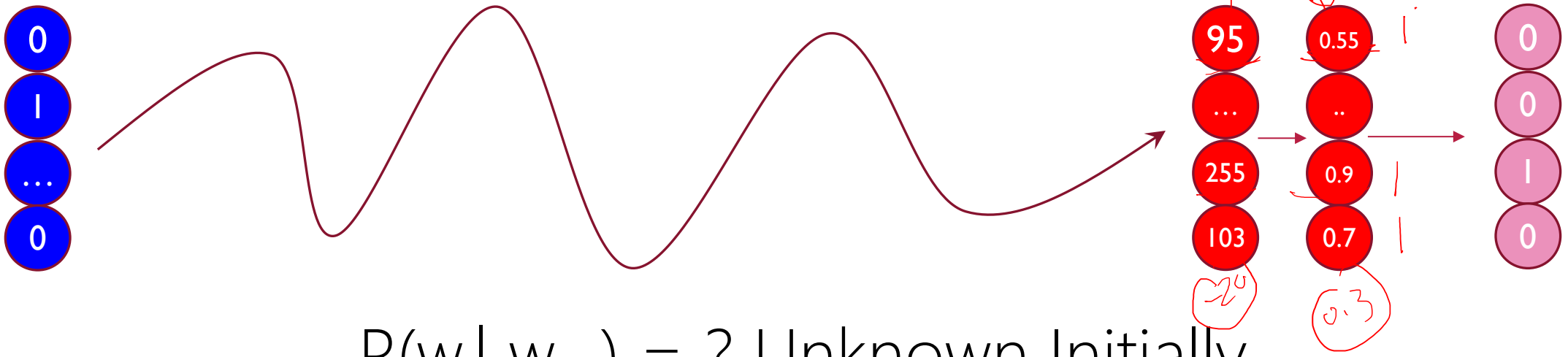
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$f: \{0,1\}^{|V|} \rightarrow R^{|V|}$; $f(w_{i-1}) \rightarrow$ Normalized $\rightarrow$ class of w_i

[0 0 0 … 1 0 … 0 0 0] $\rightarrow$ Multiplications $\rightarrow$ [95 … 205 … 0 103]

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1})$ = ? Unknown Initially

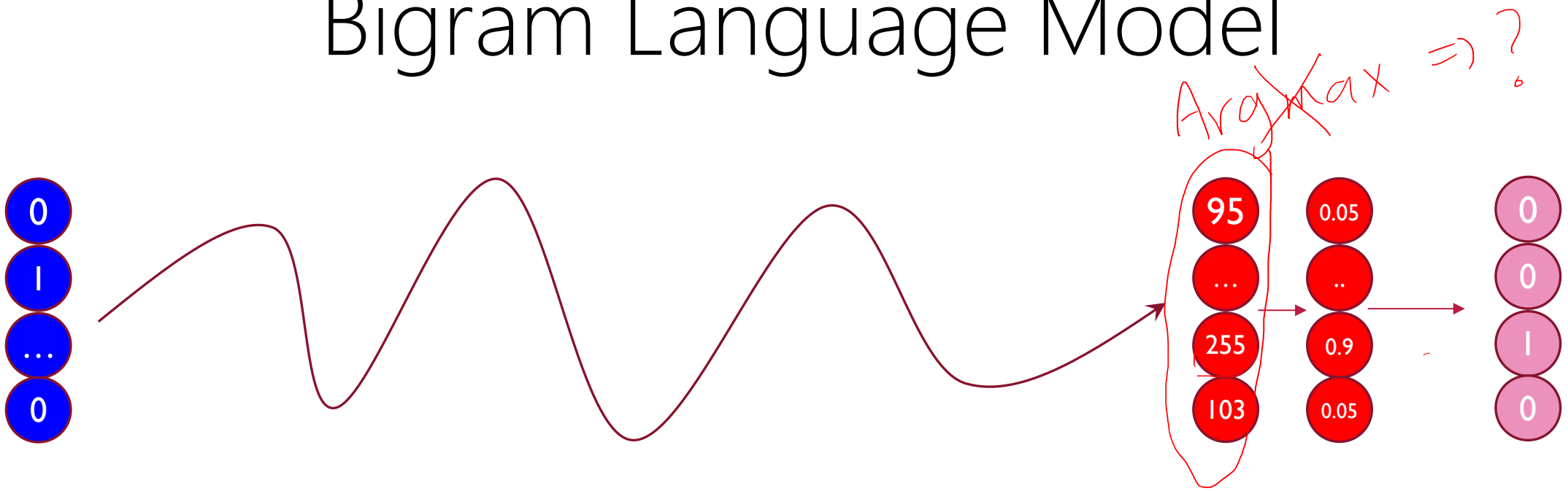
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$f: \{0,1\}^{|V|} \rightarrow R^{|V|}$; $f(w_{i-1}) \rightarrow$ ~~Sigmoid~~ $\rightarrow$ class of w_i

[0 0 0 … 1 0 … 0 0 0] $\rightarrow$ [0.55 … 0.9 … 0 0.7]

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1}) = ?$ Unknown Initially

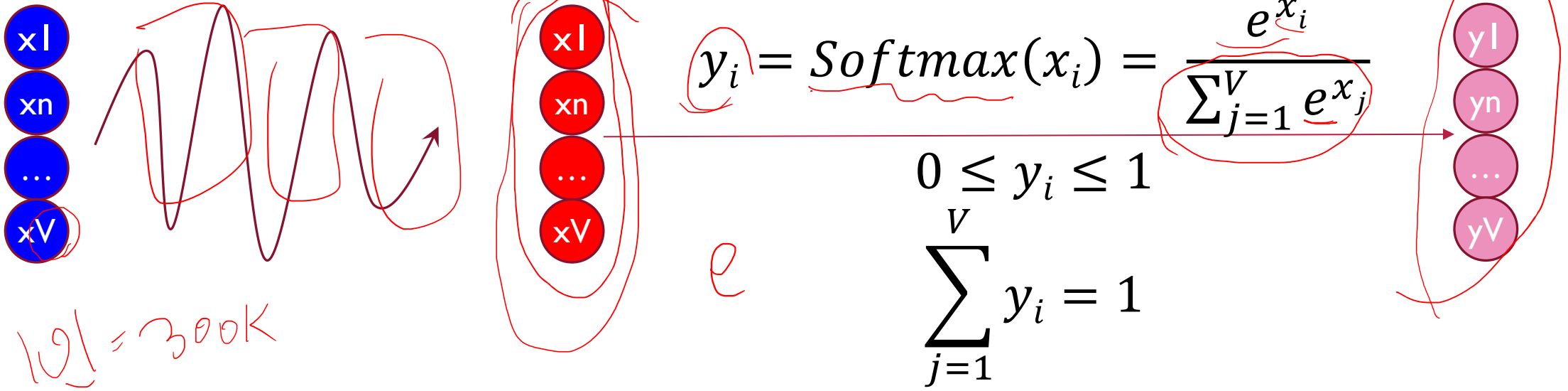
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$f: \{0,1\}^{|V|} \rightarrow R^{|V|}$; $f(w_{i-1}) \rightarrow$ Softmax $\rightarrow$ class of w_i

[0 0 0 … 1 0 … 0 0 0] $\rightarrow$ [0.05 … 0.9 … 0 0.05]

Neural Language Model

Bigram Language Model



$$y_i = Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{V} e^{x_j}}$$

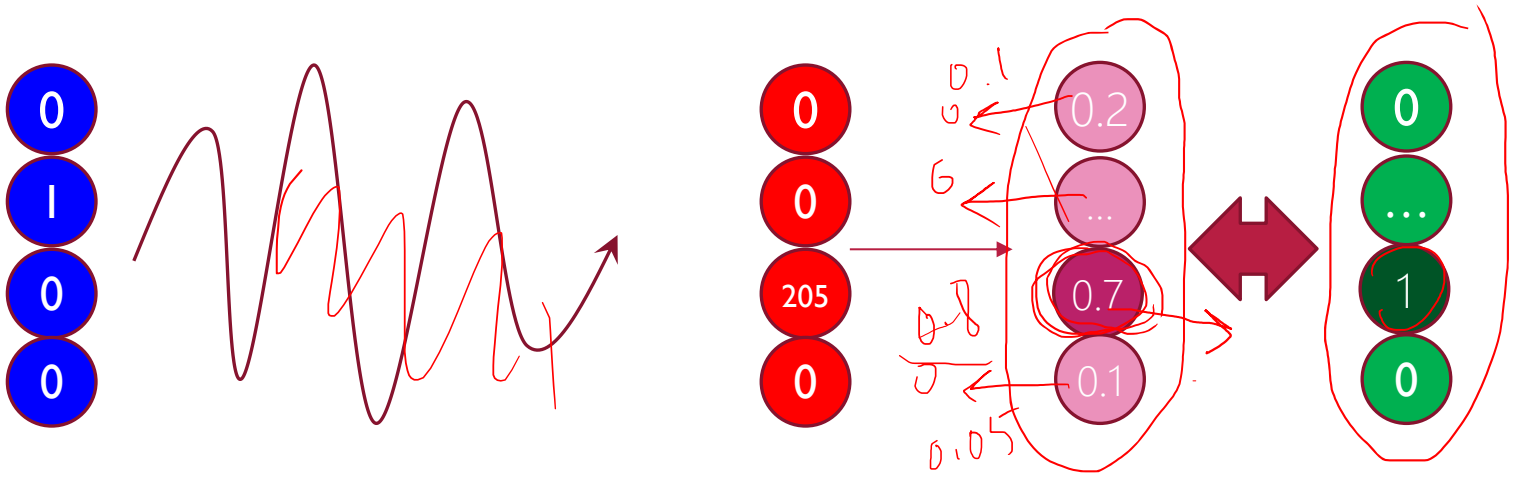
$$0 \le y_i \le 1$$

$$\sum_{j=1}^{V} y_i = 1$$

f: f: $\{0,1\}^{|V|} \longrightarrow [0,1]^{|V|}$; $f(w_{i-1}) = P(w_i | w_{i-1}) = y_i$

Neural Language Model

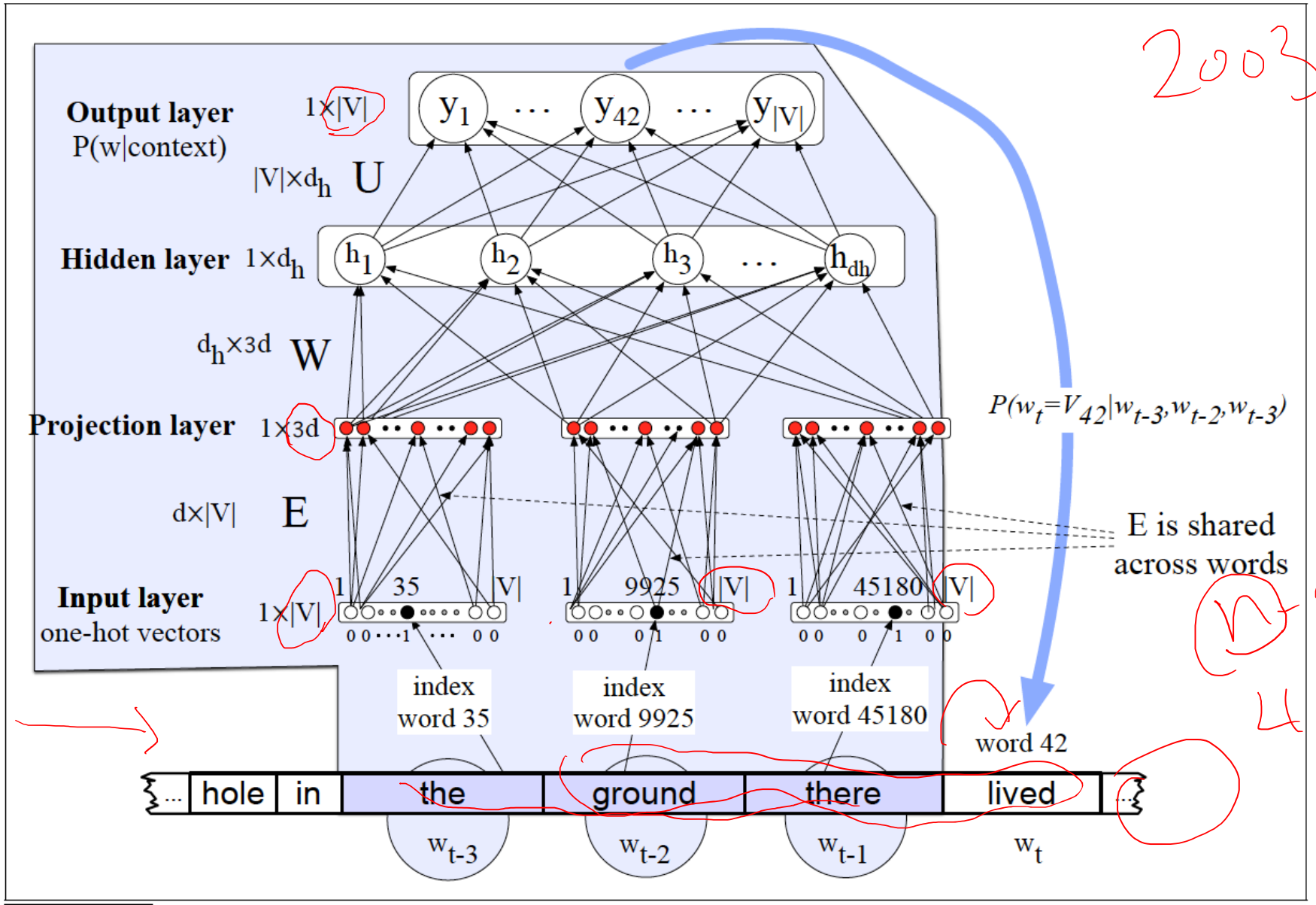
Bigram Language Model



Error = Loss = - Log $P(w_i | w_{i-1})$ = - log y_i

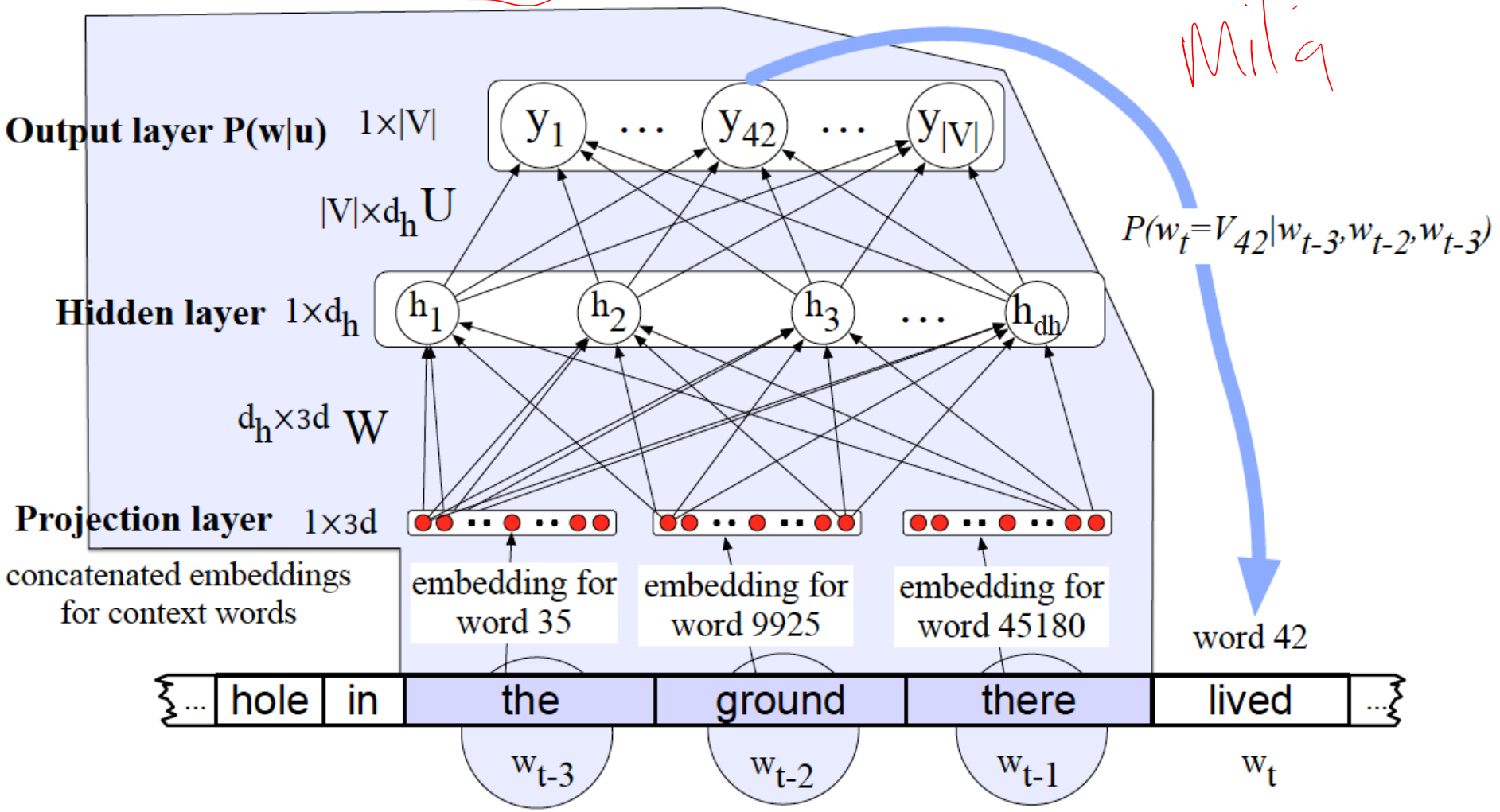
$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{V} \mathbb{1}\{\mathbf{y}_k = 1\} \log \hat{\mathbf{y}}_k$$

Feedforward Neural Network Language Model



Neural Language Model

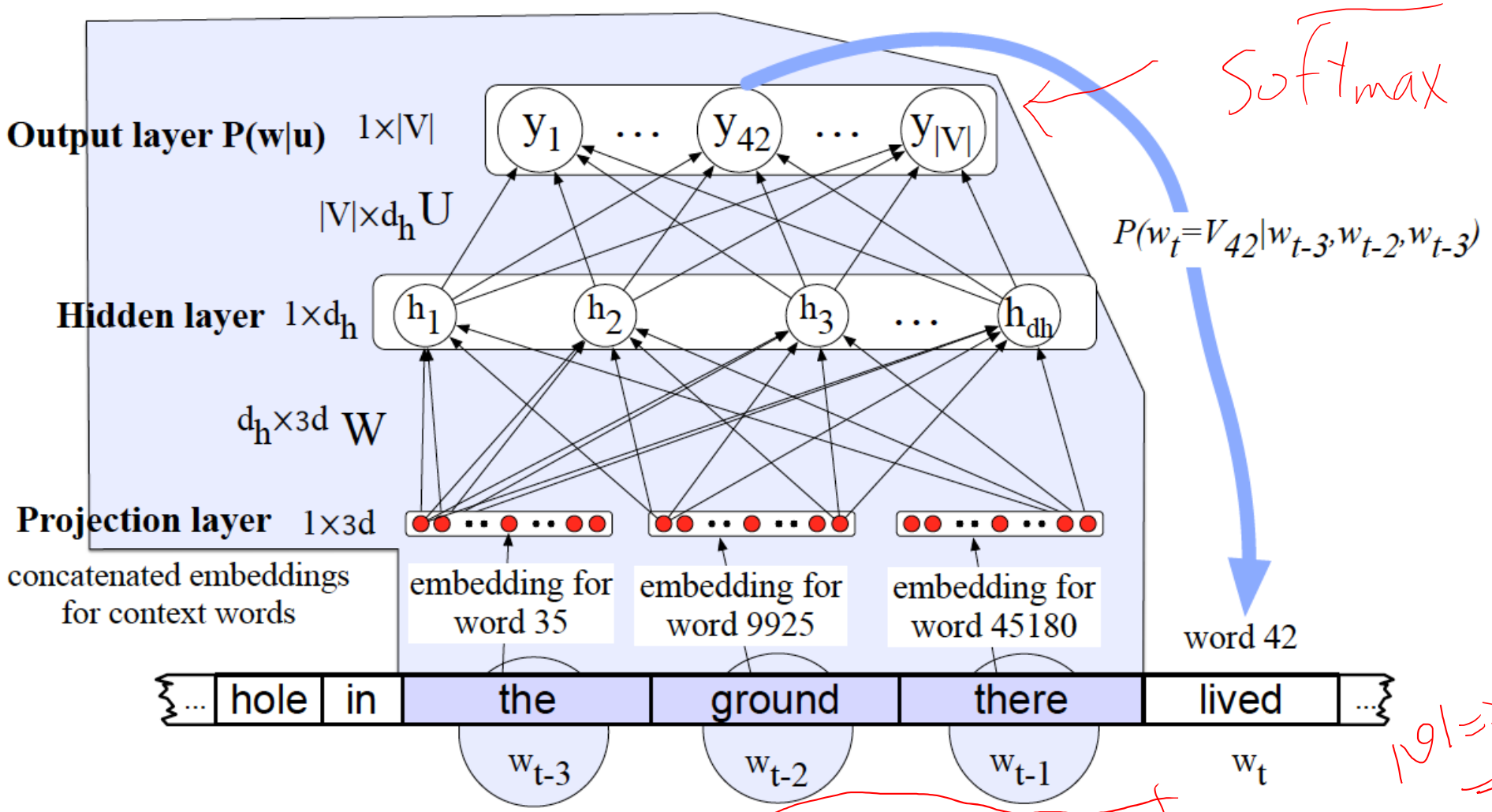
Bengio, Yoshua, et al. "A neural probabilistic language model." The journal of machine learning research 3 (2003): 1137-1155.



N-gram or Bag-of-Word?

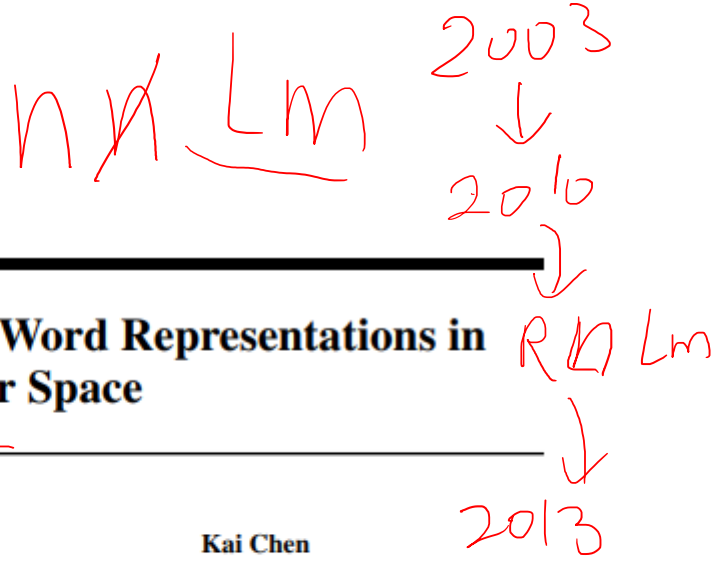
Neural Language Model

Bengio, Yoshua, et al. "A neural probabilistic language model." The journal of machine learning research 3 (2003): 1137-1155.



Bottleneck → Softmax → Hierarchical Softmax

Word2Vec (revisited)



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Efficient Estimation of Word Representations in Vector Space

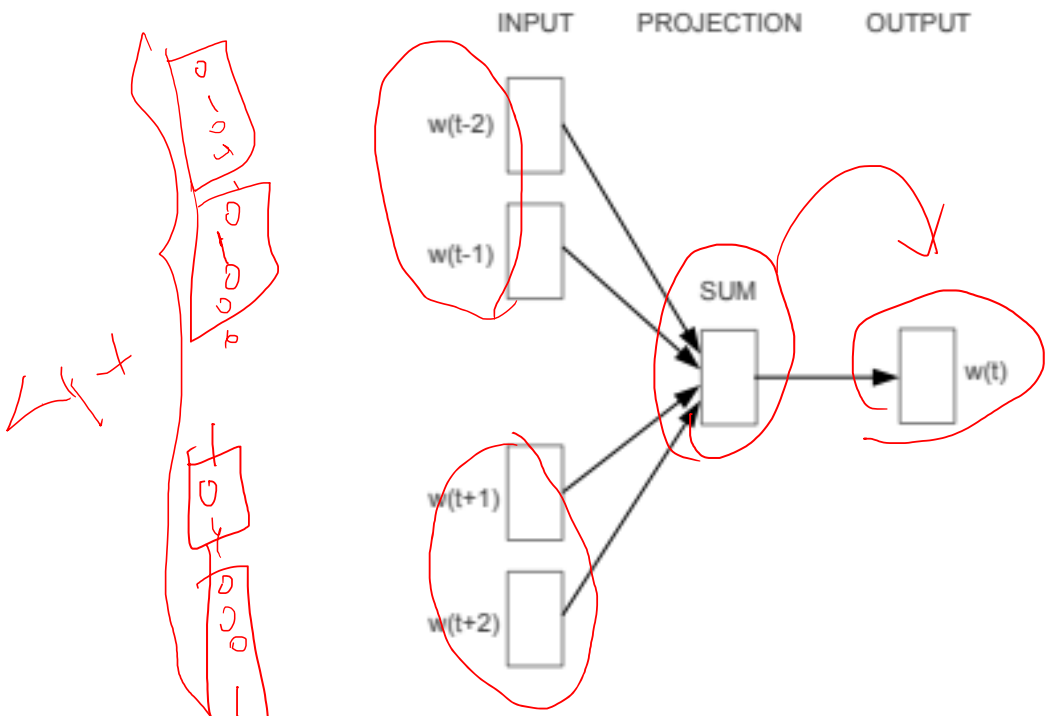
Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

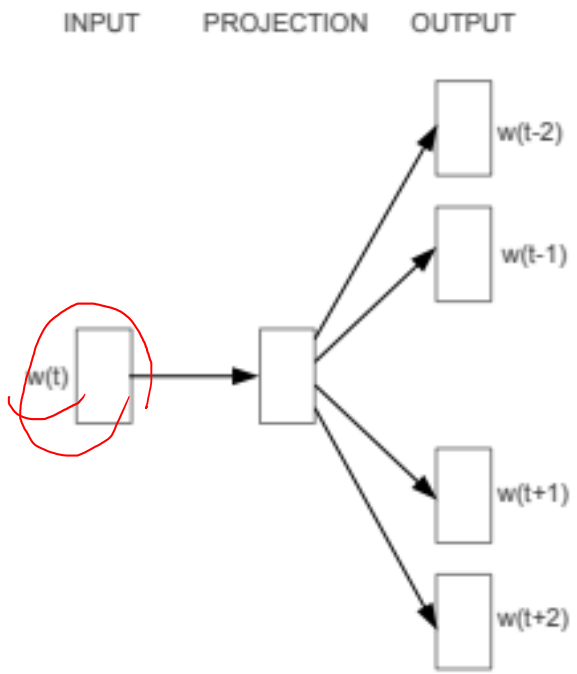
Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Word2Vec (revisited)



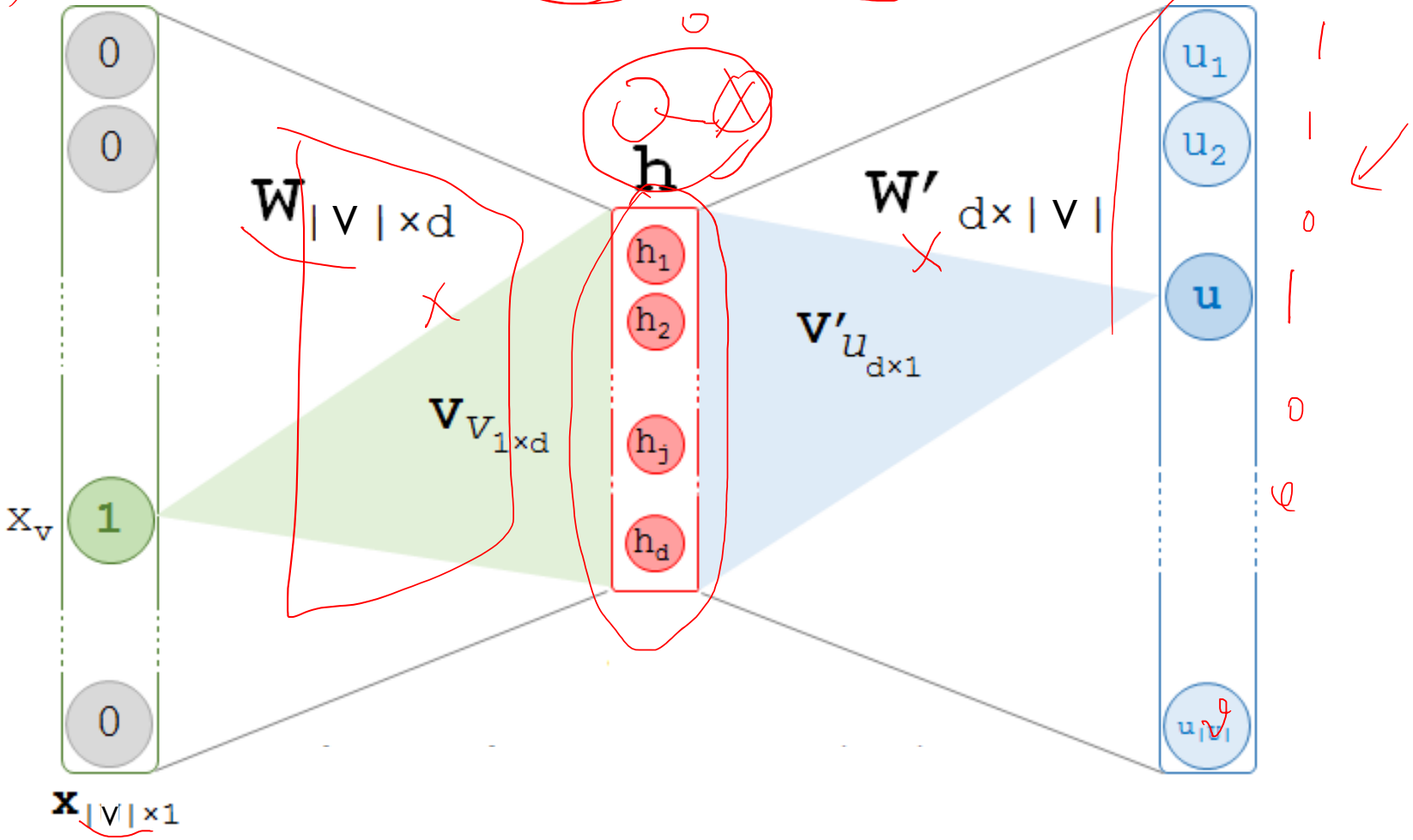
CBOW



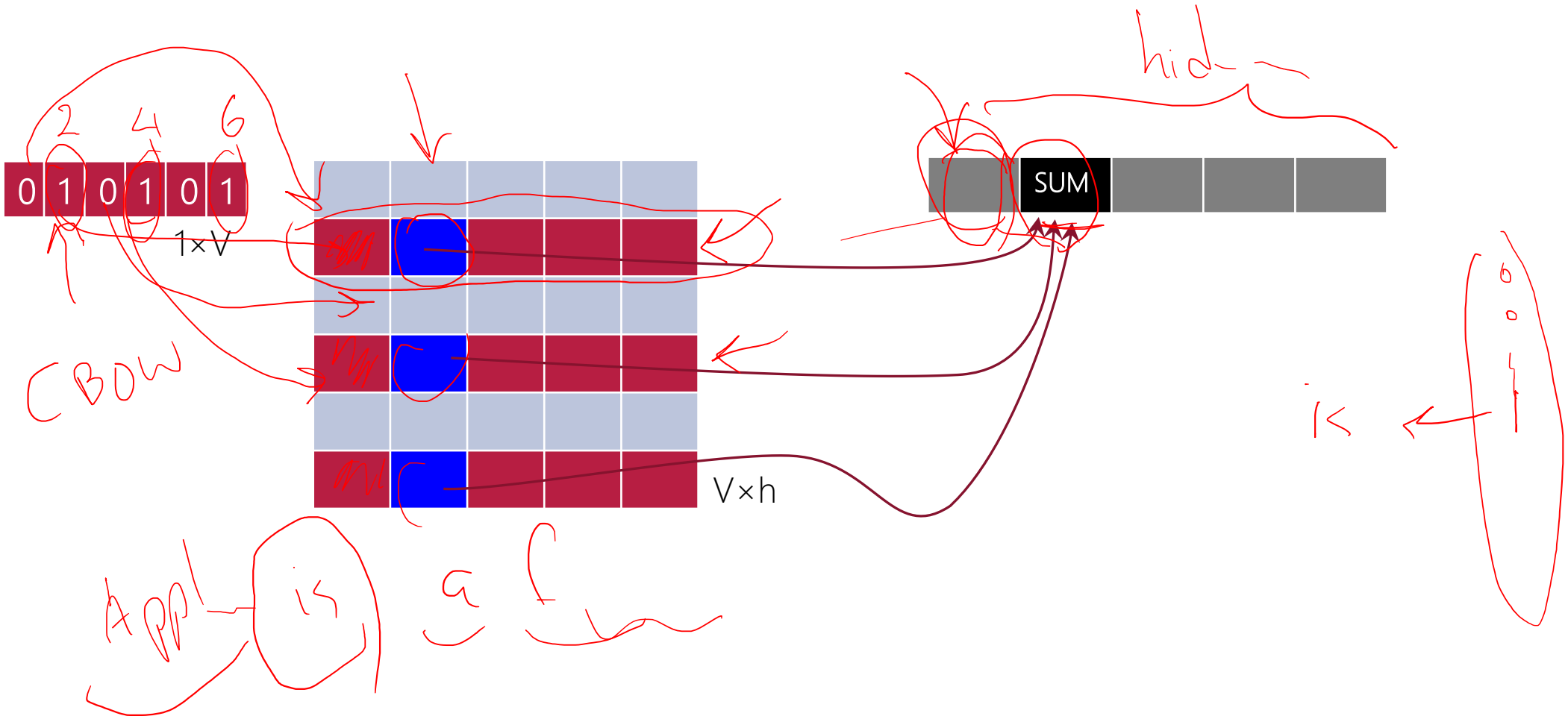
Skip-gram

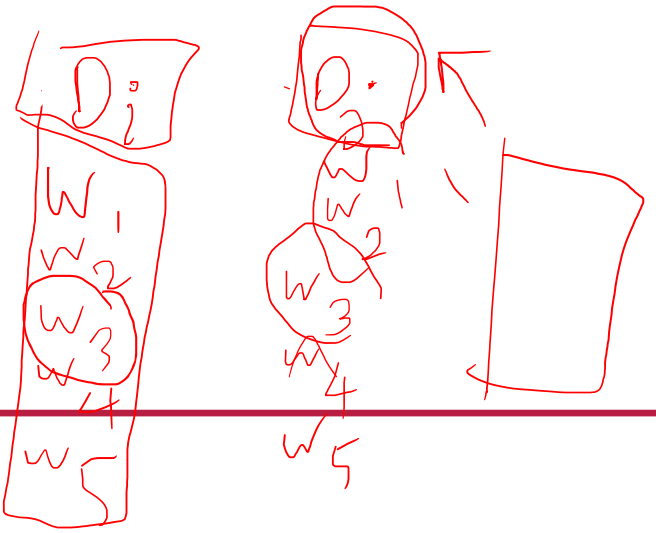
Word2Vec

$$\sigma\left((h = x^T W + \cancel{b})\, W' + \cancel{b}\right)$$



Word2Vec





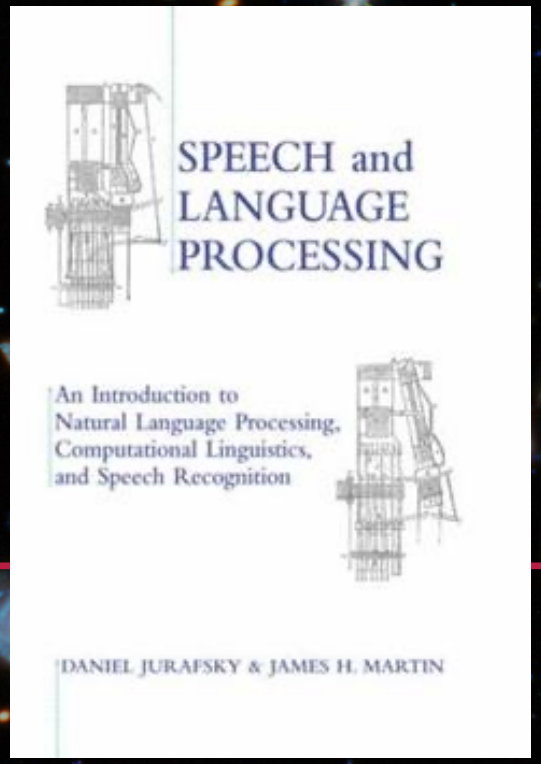
Word2Vec Predicts within a Context Window

Doc 2 Vec

The Context Window moves over the single stream of words.

No further context such as sentence, paragraph, or document is considered!

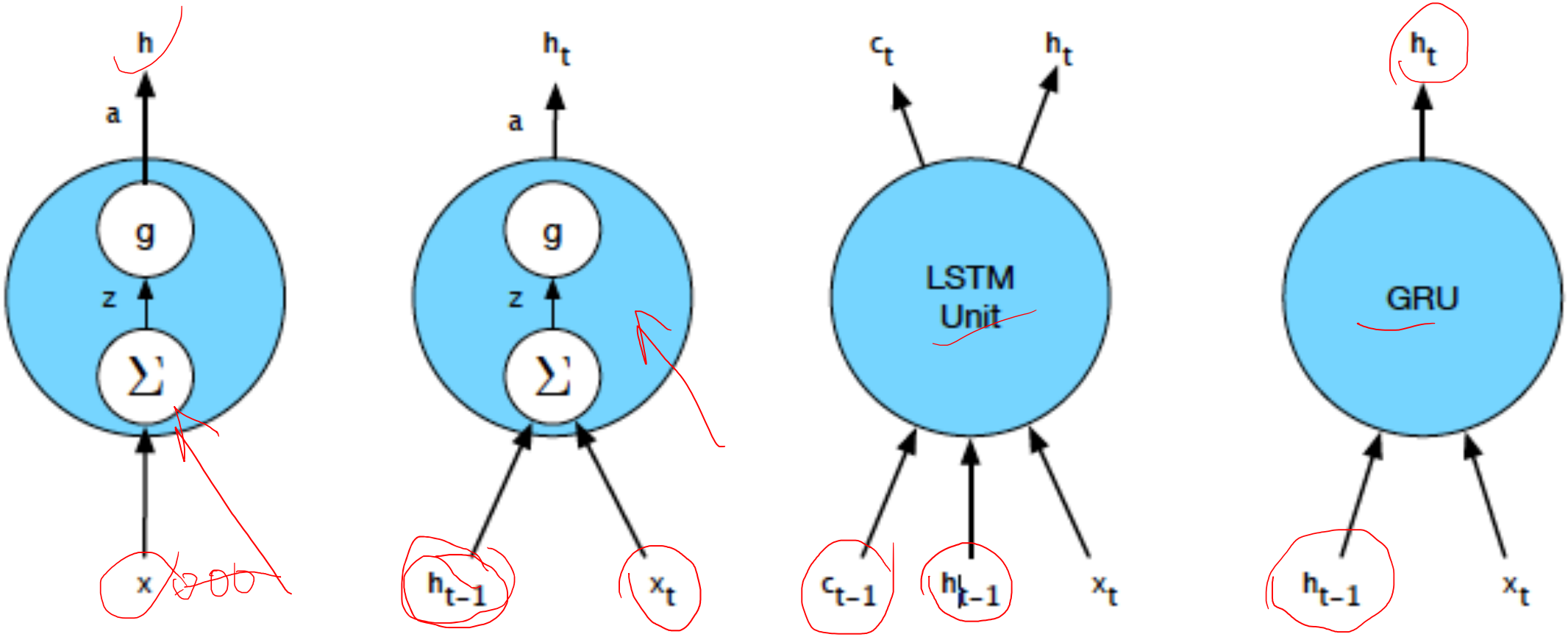
What if we say that the Window Context is moving within what document?



Deep Learning Architectures for Sequence Processing

CH09

RECURRENT LANGUAGE MODELS



Digital Design

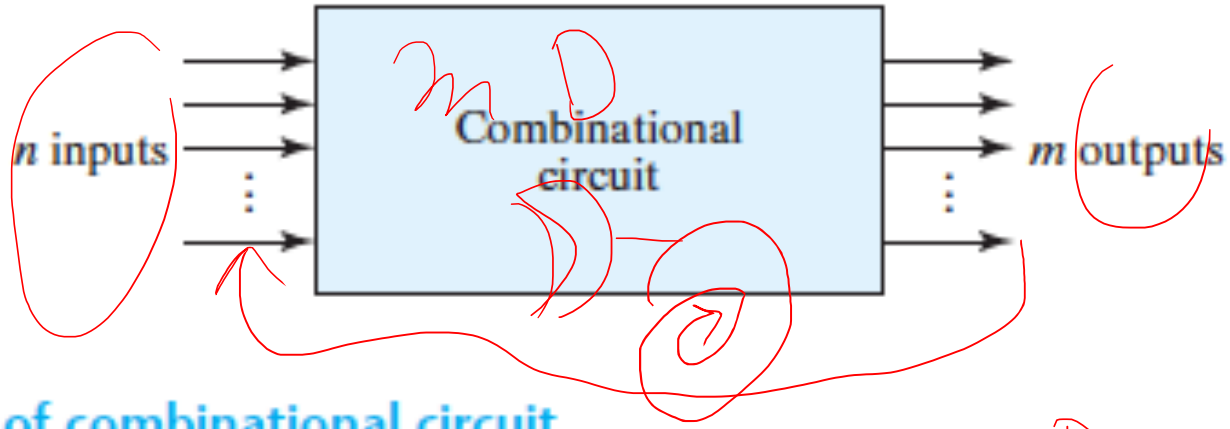


FIGURE 4.1
Block diagram of combinational circuit

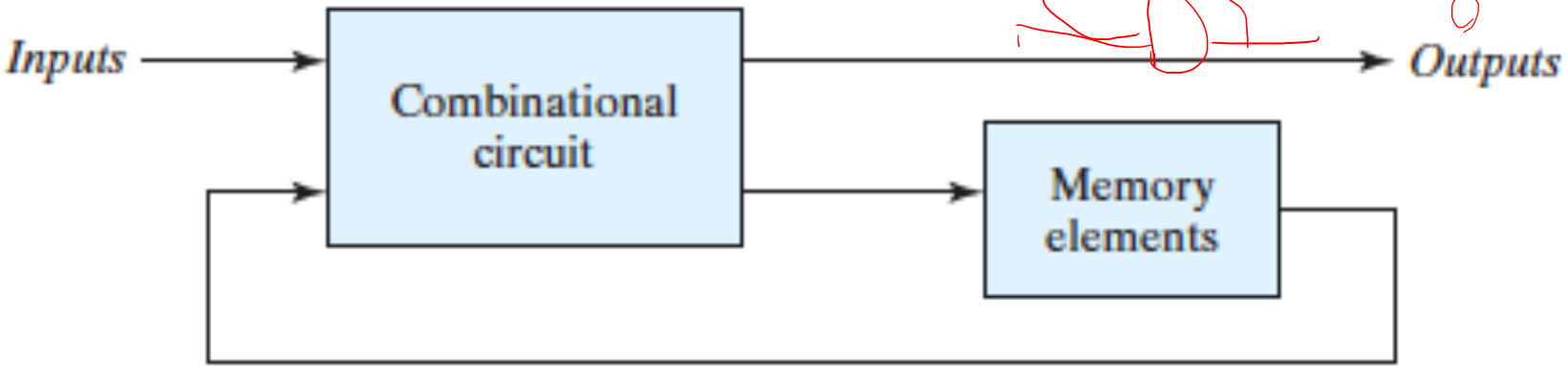
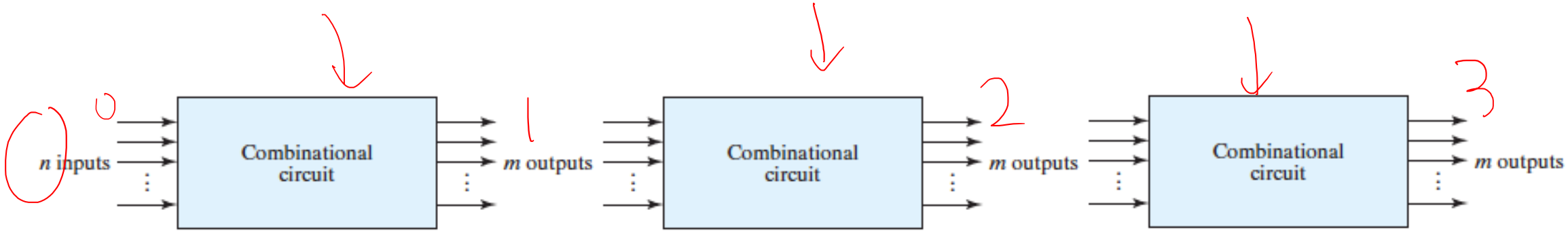
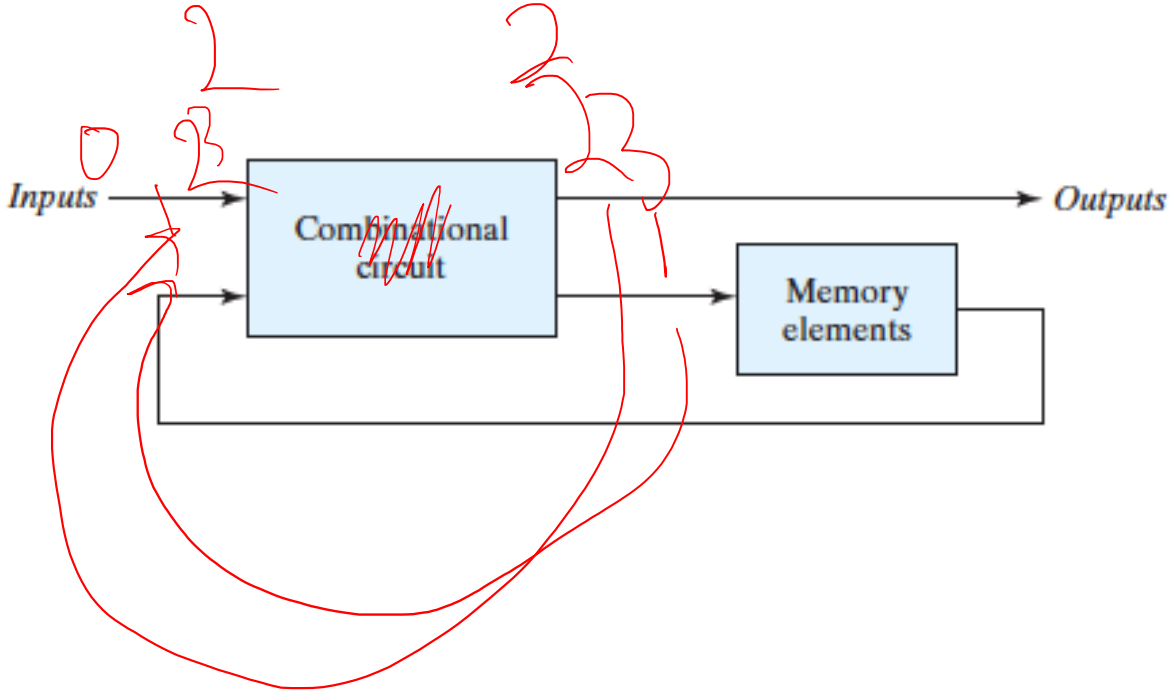


FIGURE 5.1
Block diagram of sequential circuit

Counter: 0 → 1 → 2 → …


n inputs
Combinational circuit
m outputs
Combinational circuit
m outputs
Combinational circuit
m outputs

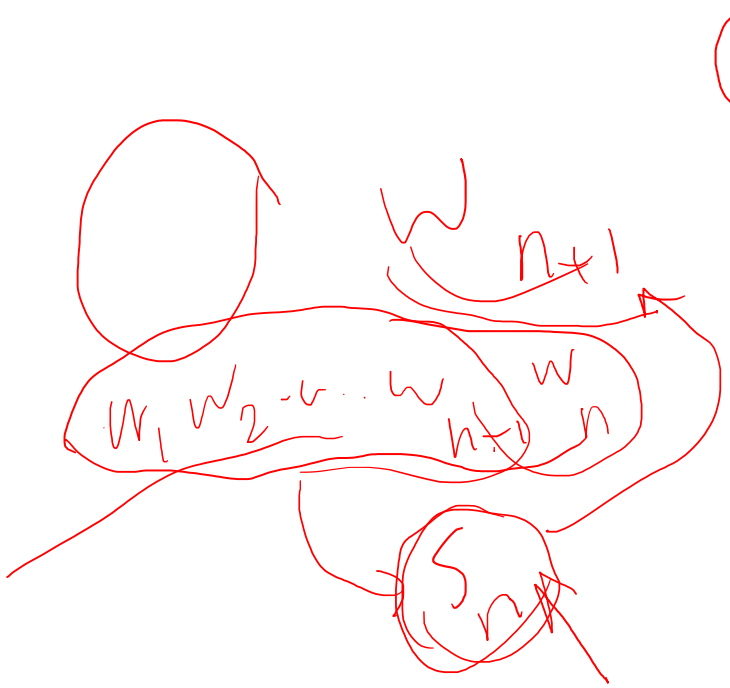

Inputs
Combinational circuit
Outputs
Memory elements

Language is Temporal

We already saw that previous context matters!

Unigram → Bigram → Trigram → …

Conversations
Social Timelines
Social Feeds



Language is Temporal

We already saw that previous context matters!

Unigram → Bigram → Trigram → …

Neural LM

The same in this respect!

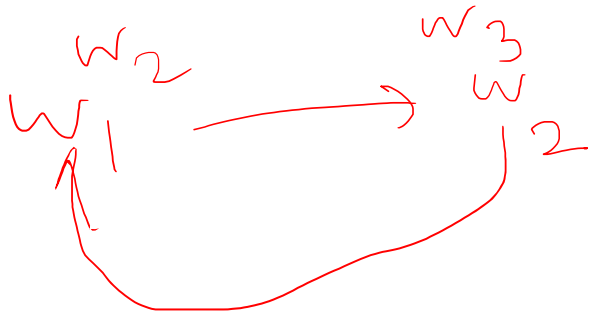
Only different method (approach)

But …

Language is Temporal

We already saw that previous context matters!

Unigram → Bigram → Trigram → …



Neural LM

The same in this respect!

Only different method (approach)

But it offers a modular structure!

Neural LM

Neural LM

- But it offers a modular structure!
- We can
  - Stack them
  - Connect different parts
  - Create loopback connection

Recurrent Neural LM

Bengio's Neural LM

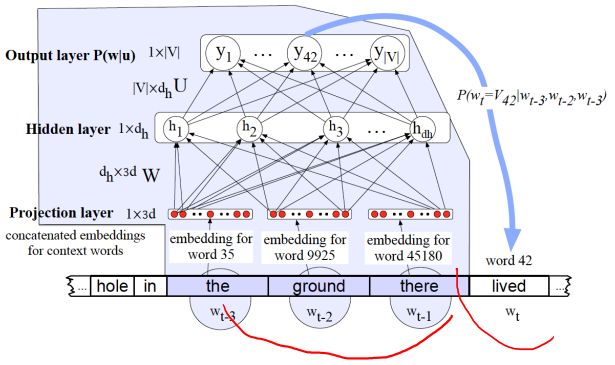
Fixed Length Context Window (n-gram)

Longer Stream

Sliding the Context Window

"we are in natural language processing class"

[we are in]
[are in natural]
[in natural language]
[natural language processing]
[language processing class]



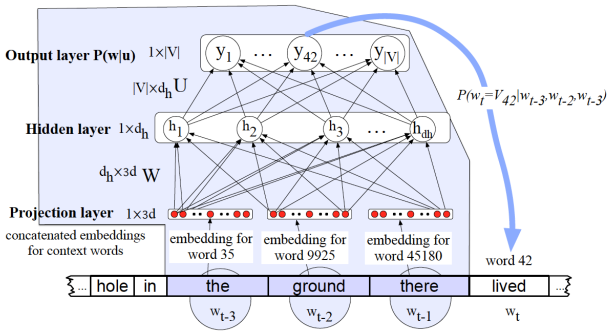
Recurrent Neural LM

Bengio's Neural LM

- Fixed Length Context Window (n-gram)
- Longer Stream
  - Sliding the Context Window

"we are in natural language processing class"

[we are in]
[are in natural]
[in natural language]
[natural language processing]
[language processing class]



Overlapping sliding carries some context from history

Recurrent Neural LM

Bengio's Neural LM

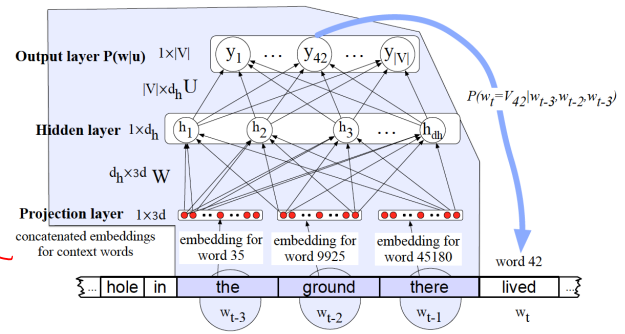
Fixed Length Context Window (n-gram)

Longer Stream

Sliding the Context Window

"we are in natural language processing class"

[we are in]
[are in natural]
[in natural language]
[natural language processing]
[language processing class]



But not from far distances!

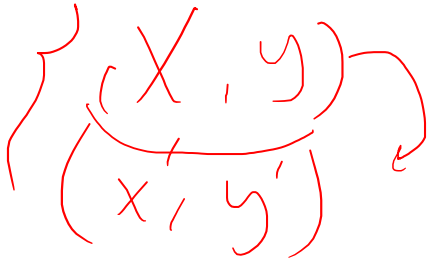
Recurrent Neural LM

Bengio's Neural LM

Fixed Length Context Window (n-gram)

Longer Stream

Sliding the Context Window



"we are in natural language processing class"

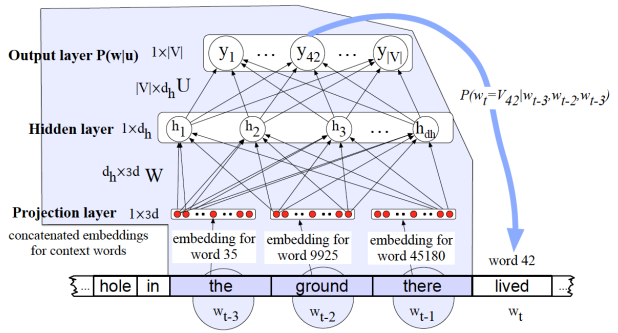
[language processing class]

[are in natural]

[we are in]

[in natural language]

[natural language processing]



Also, inputs are independent!

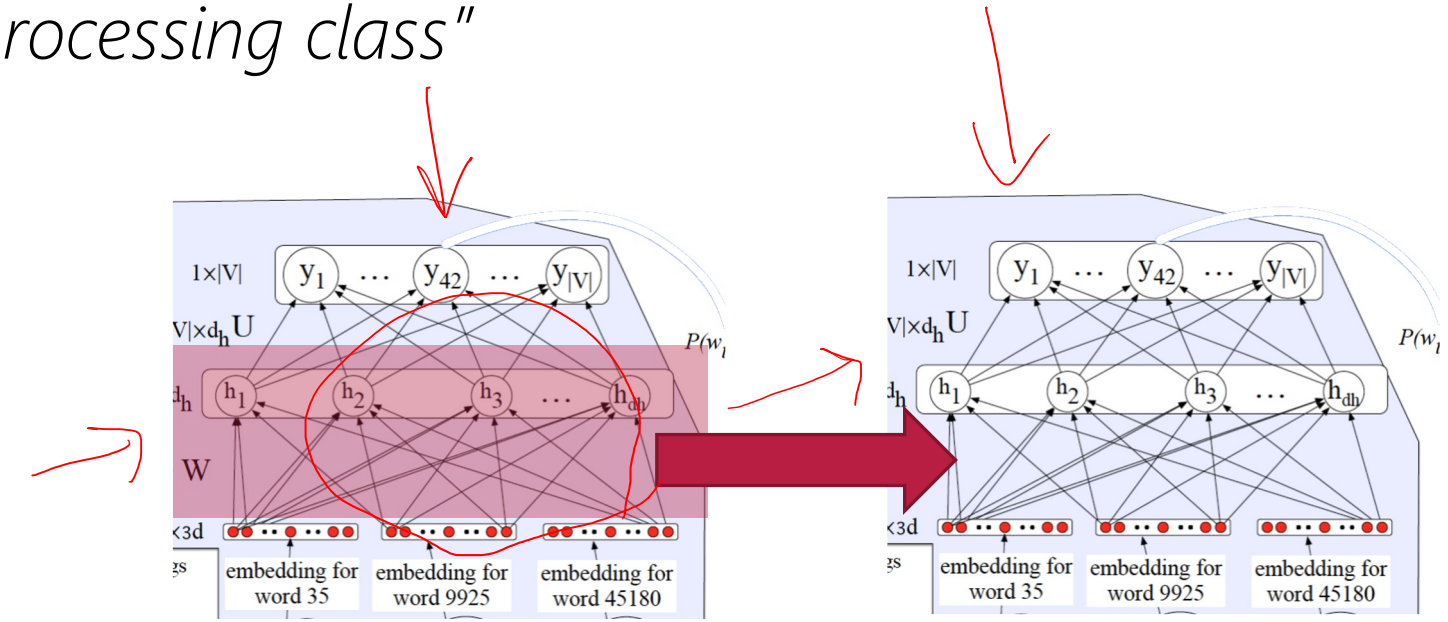
Recurrent Neural LM

Bengio's Neural LM

- Fixed Length Context Window (n-gram)
- Longer Stream
- Sliding the Context Window

"we are in natural language processing class"

[we are in]
[are in natural]
[in natural language]
[natural language processing]
[language processing class]



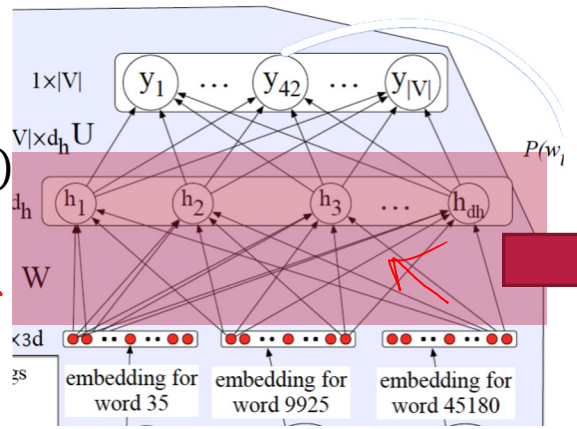
Recurrent Neural LM

$$Y = softmax(hU)$$

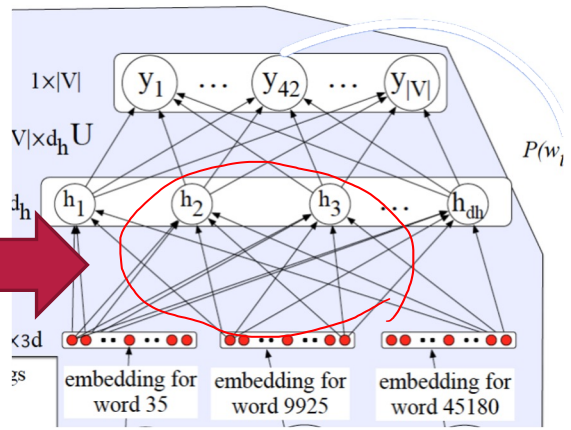
$$h = \tanh(X\,W + b)$$

$$X\,W + b$$

X



+ hV



$$Y' = softmax(h'U)$$

$$h' = \tanh(X'W + b + hV)$$

$$X'\,W + b$$

X'

$f(h)$: *what part of history and how to transfer to future*!

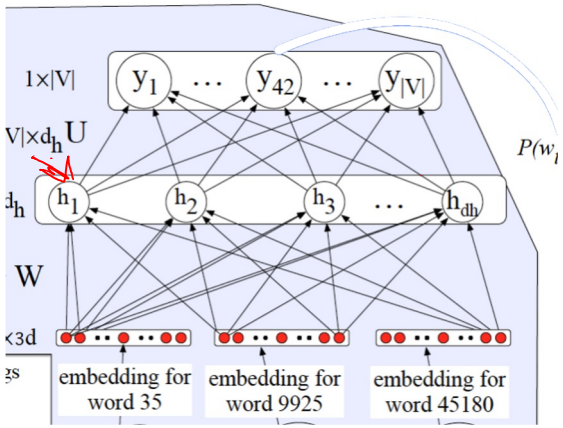
Recurrent Neural LM

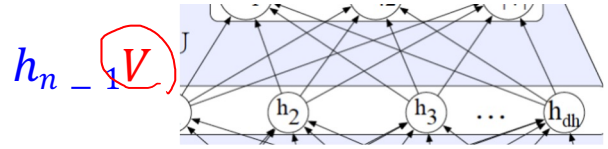
$$Y_n = softmax(hnU)$$

$$\mathrm{h_n} = \tanh(Xn\,W + b + h_{n\_1}V)$$

$$X_n\,W + b$$

$$X_n$$





Recurrent Neural LM

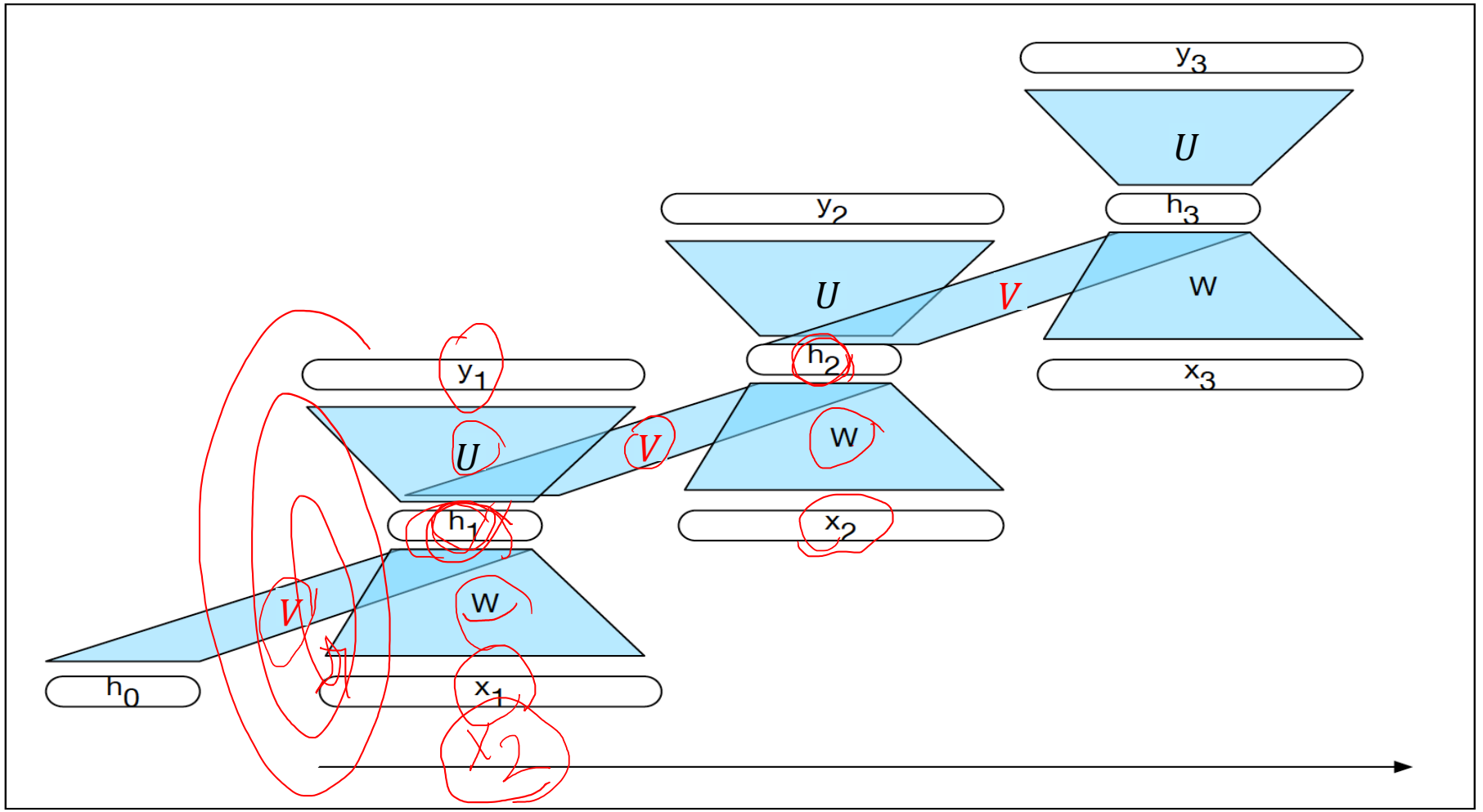


Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.

Recurrent Neural LM

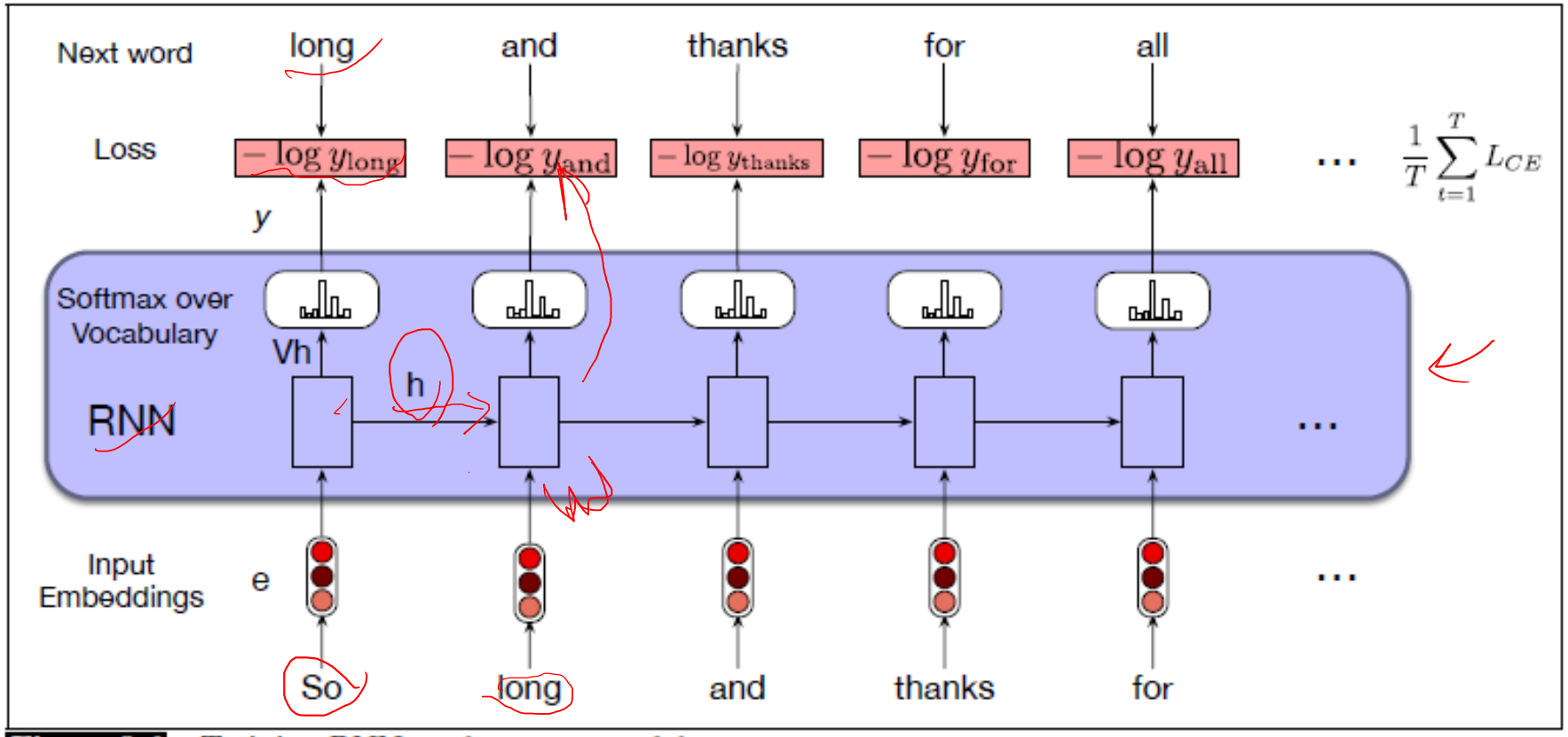
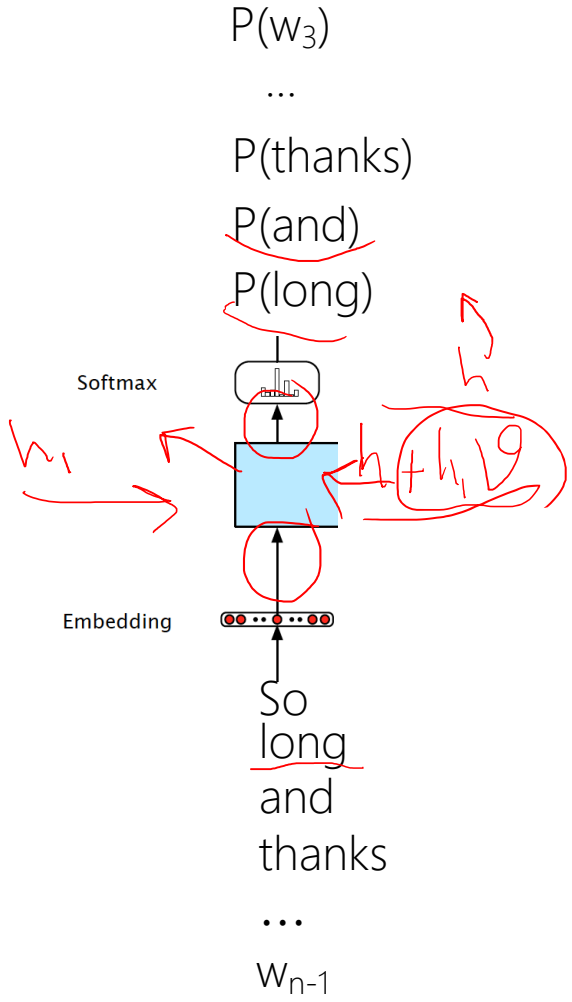


Figure 9.6 Training RNNs as language models.



Recurrent Neural LM: Training

$$Y_n = \sigma'(h_n U + b); \quad hn = \sigma(x_n W + b + hn\_1 V)$$

L = distance (Y_n – Y*)

$$\frac{\partial L}{\partial v \in V} = ?$$

$$\frac{\partial L}{\partial w \in W} = ?$$

$$\frac{\partial L}{\partial u \in U} = ?$$

Backpropagation Through Time (Werbos 1974, Rumelhart et al. 1986, Werbos 1990).

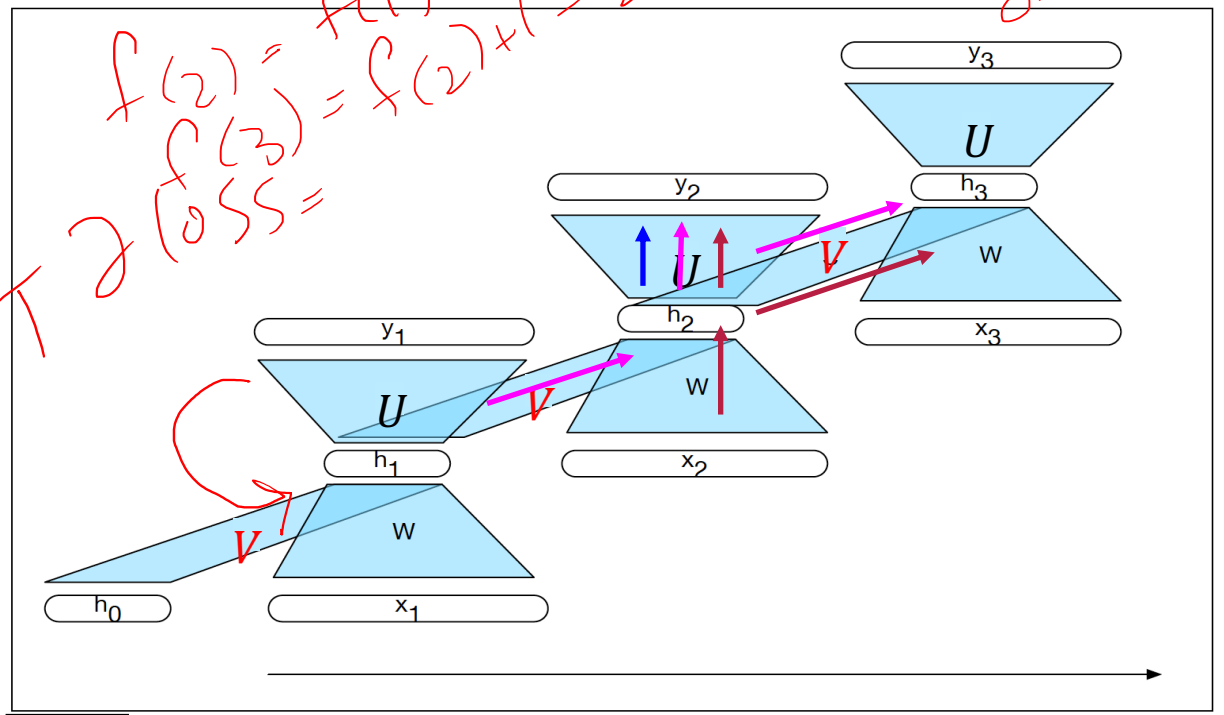


Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.

Recurrent Neural LM: Training

Mikolov, T. et al (2010). Recurrent neural network based language model. In INTERSPEECH 2010, 1045–1048.

The limited context constraint inherent in both N-gram models and sliding window approaches is avoided since the hidden state embodies information about all of the preceding words all the way back to the beginning of the sequence.

True Markovian!

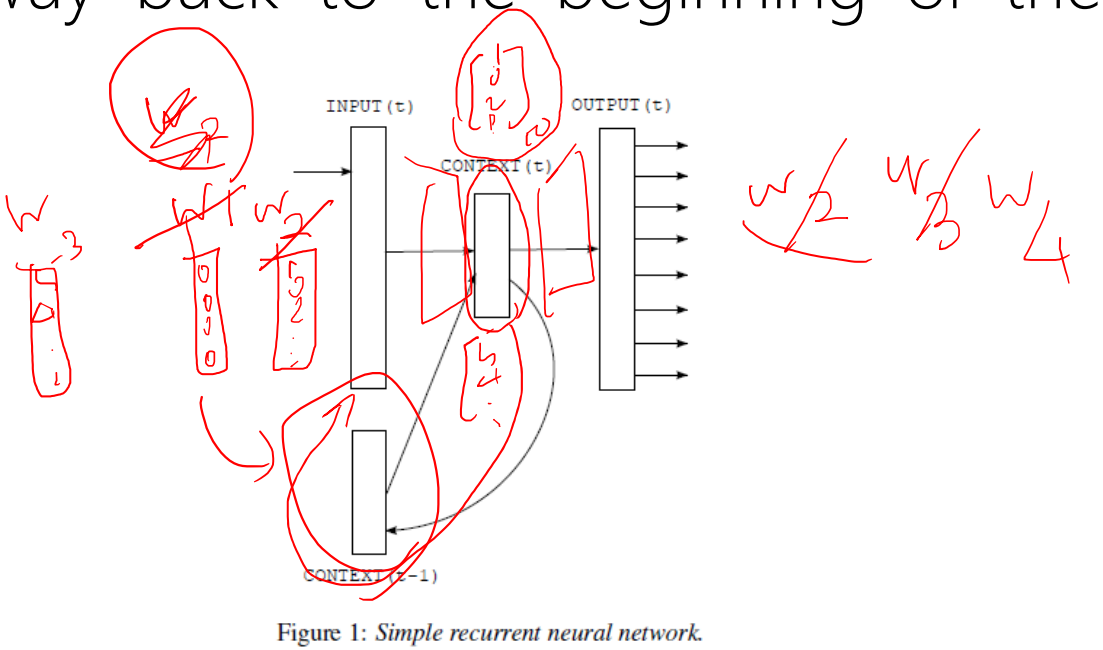


Figure 1: *Simple recurrent neural network.*

Recurrent Neural LM: Autoregression

Generate a sentence by the trained language model:

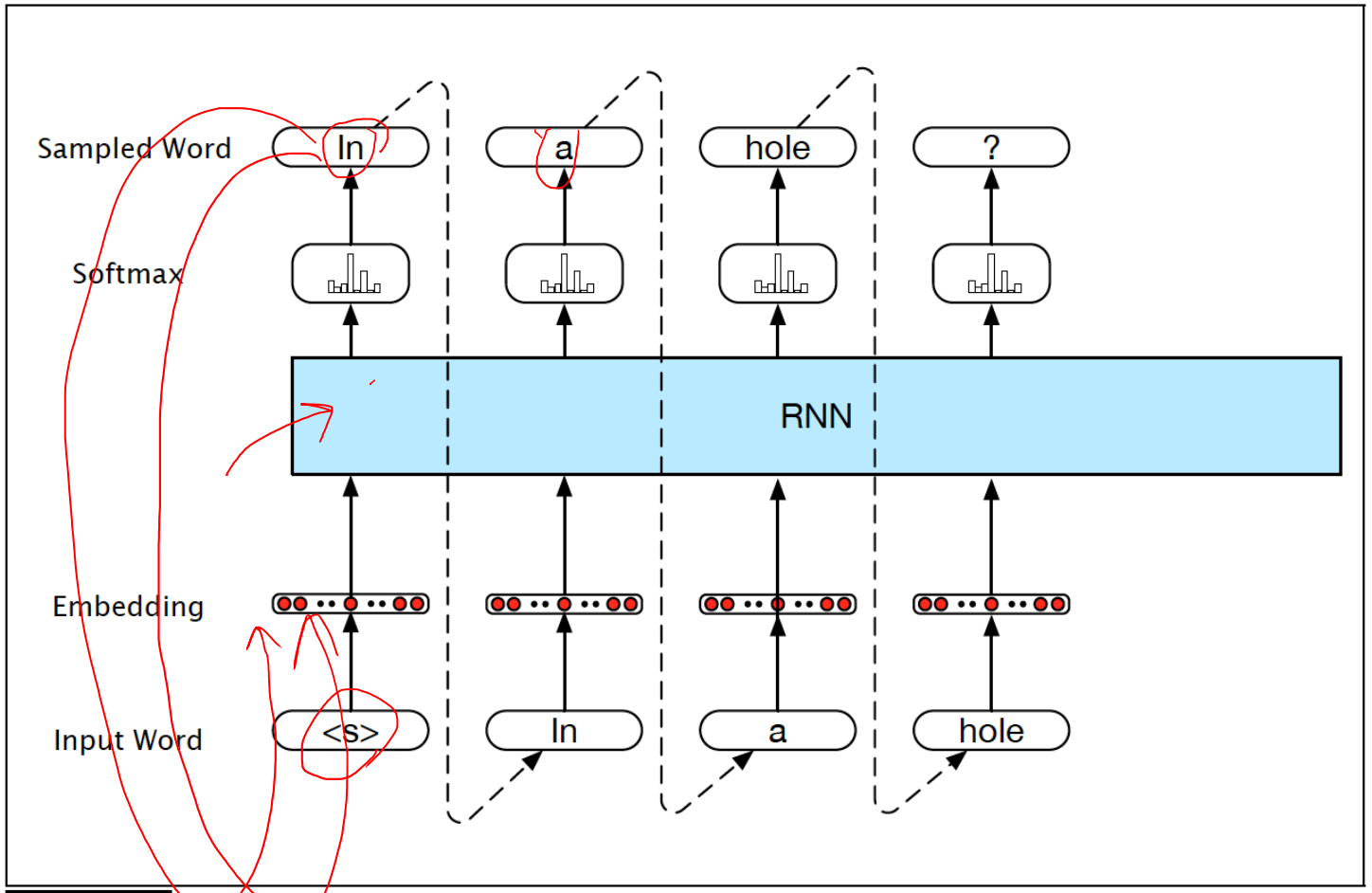


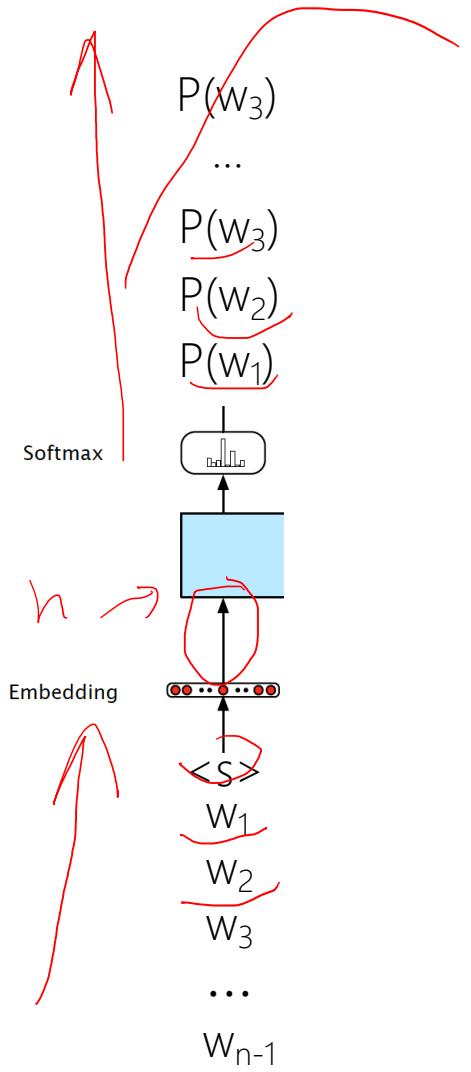
Figure 9.7 Autoregressive generation with an RNN-based neural language model.

Recurrent Neural LM: Evaluation

Likelihood of the test text stream:

$P(w_1w_2w_3..w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2) \ldots P(w_n|w_1w_2\ldots w_{n-1})$

$$\text{Perplexity}(w_1w_2w_3..w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_1w_2w_3..wn)}}$$



Recurrent Neural LM: Application

- Part-of-Speech Tagging
- Sequence Classification

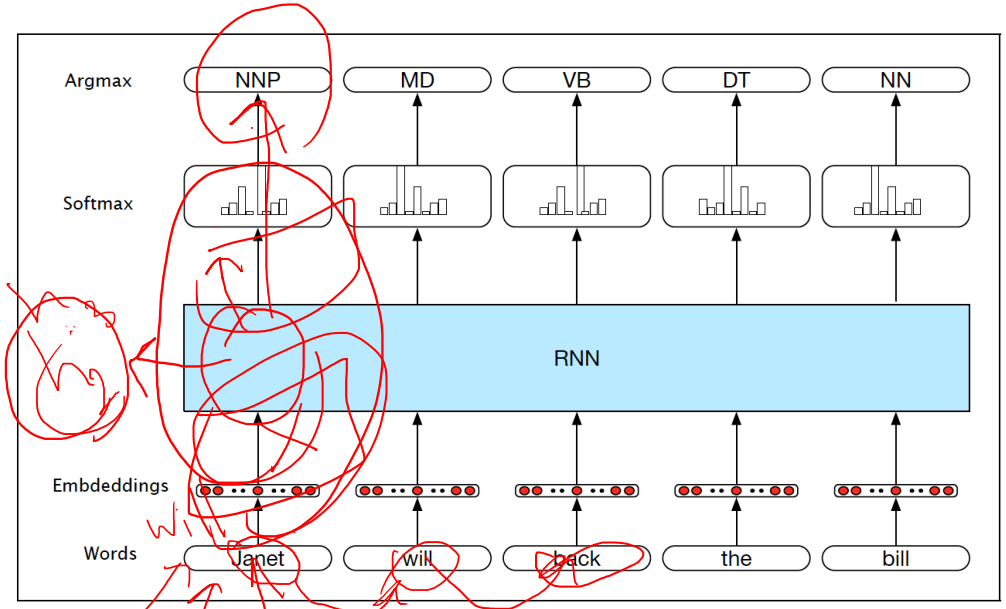


Figure 9.8 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

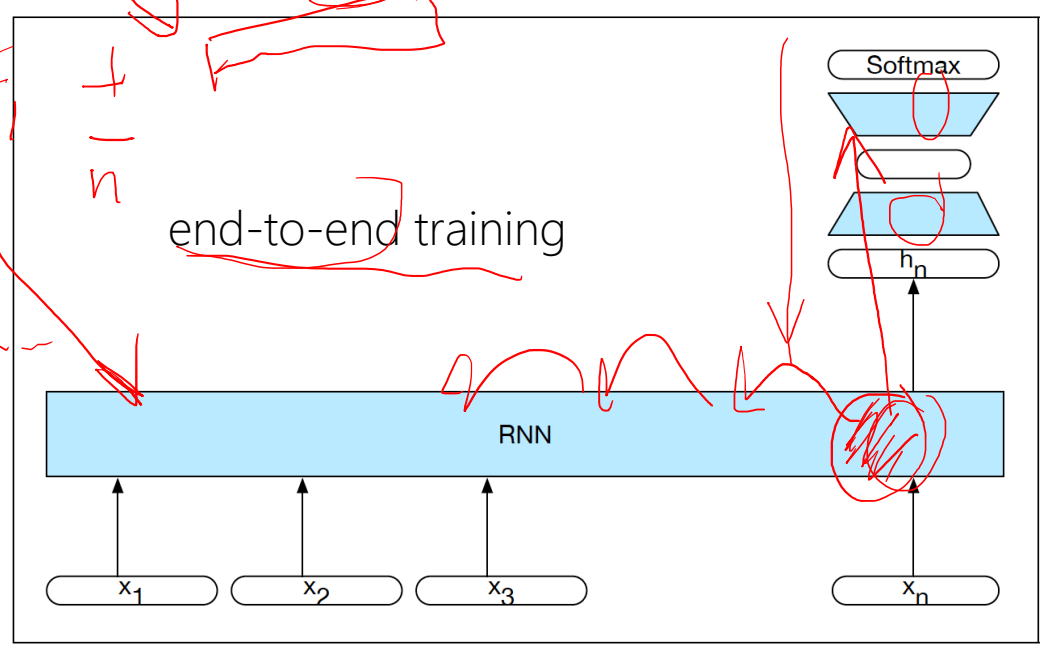


Figure 9.9 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Recurrent Neural LM

Bidirectional

- We have the whole context
  (e.g., a sentence, short dialog, …)

- We have the future!
  From future to the past (backward)

Recurrent Neural LM: Bidirectional

$$P(w_i | w_1 w_2 .. w_{i-1} \; w_{i+1} \ldots w_n)$$

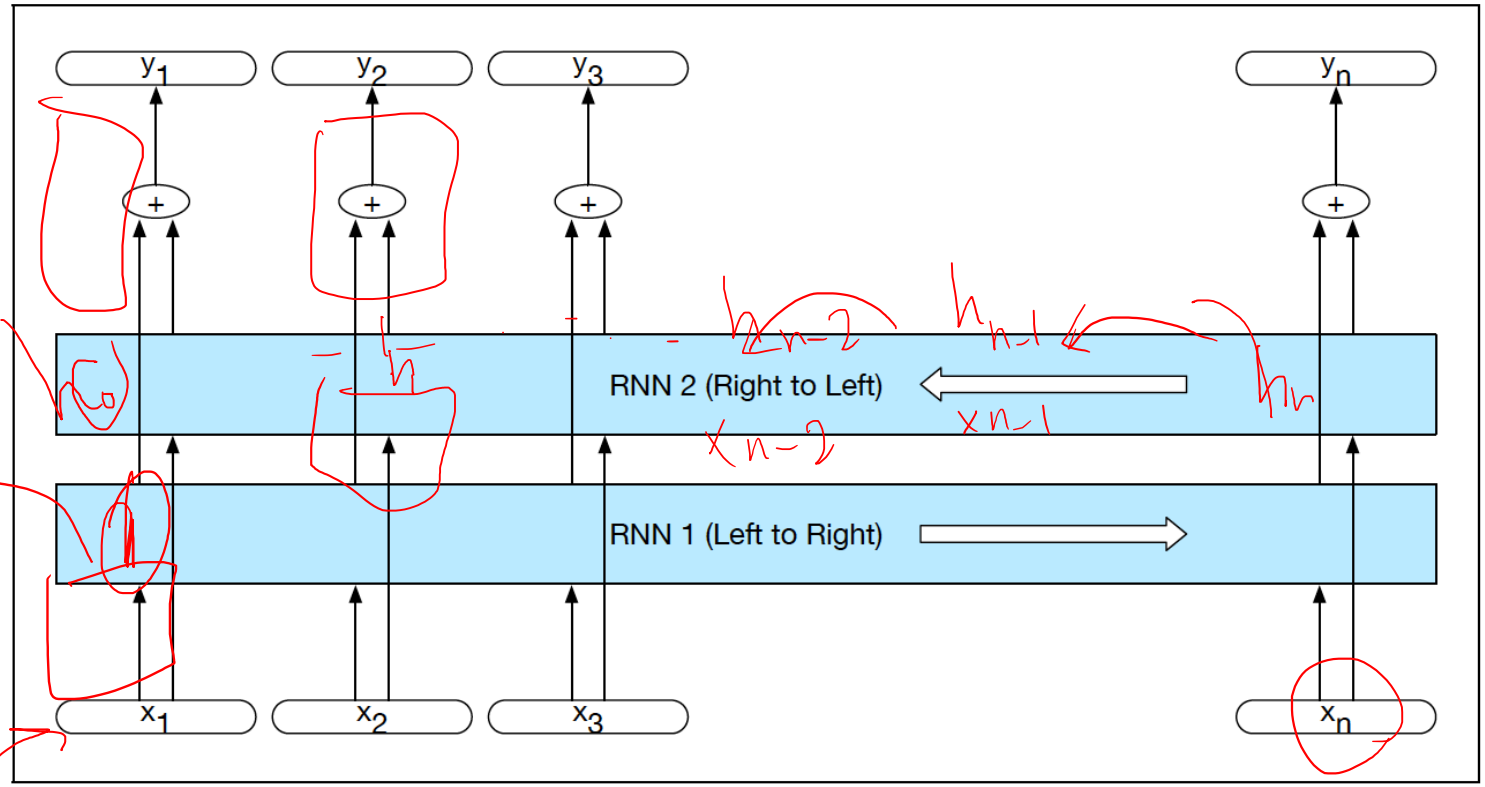


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions with the output of each model at each time point concatenated to represent the state of affairs at that point in time. The box wrapped around the forward and backward network emphasizes the modular nature of this architecture.

Recurrent Neural LM: Bidirectional

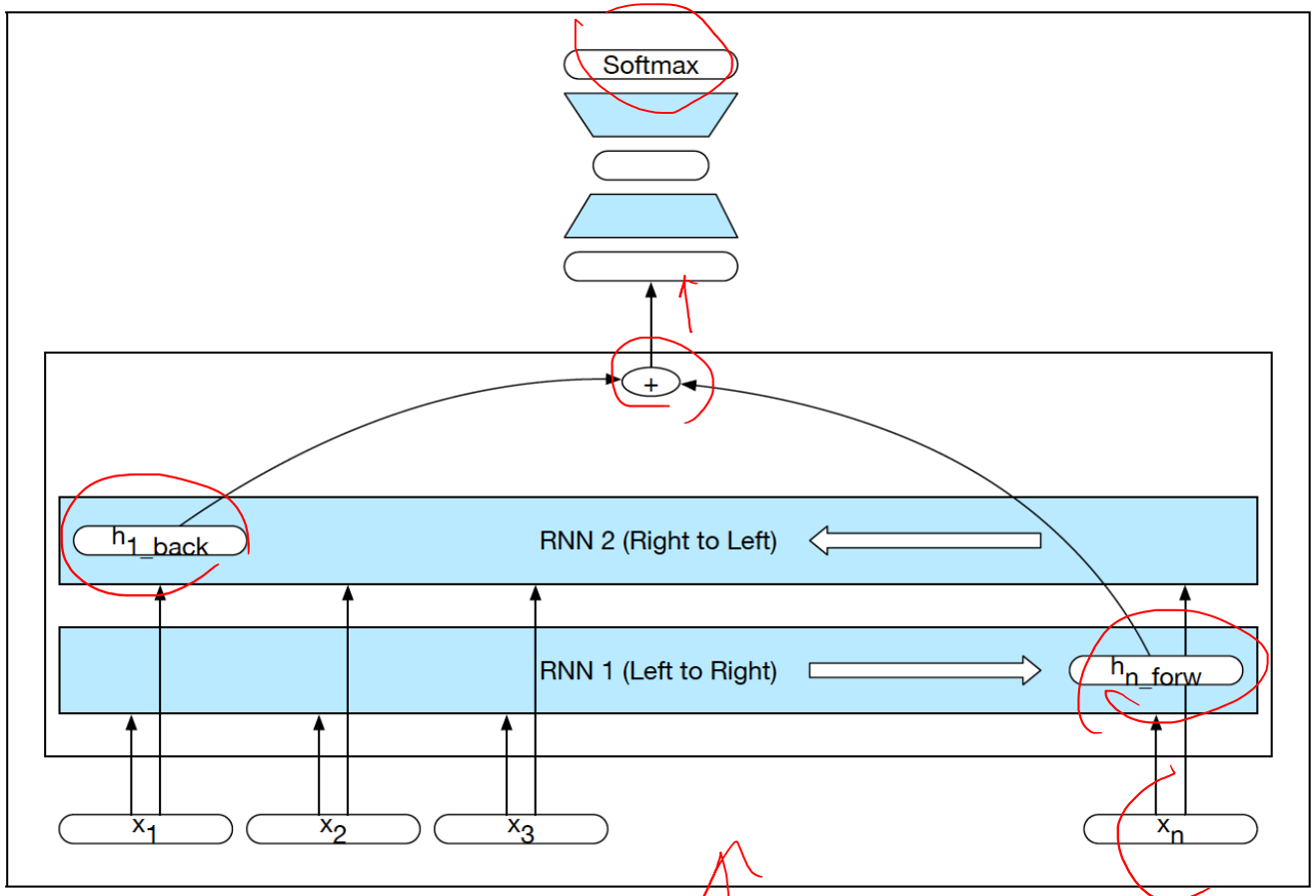


Figure 9.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

Recurrent Neural LM: Bidirectional

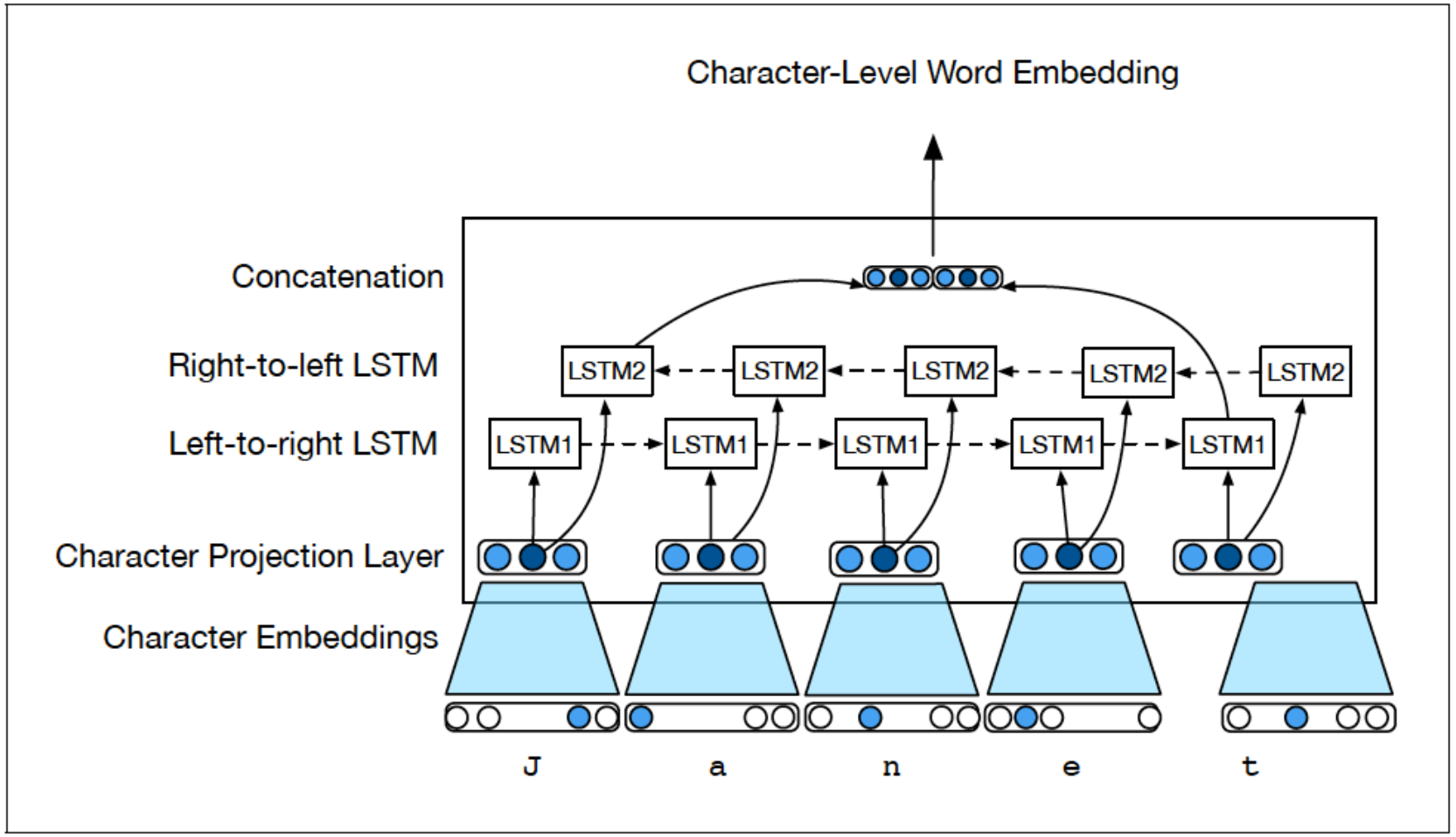


Figure 9.16 Bi-RNN accepts word character sequences and emits embeddings derived from a forward and backward pass over the sequence. The network itself is trained in the context of a larger end-application where the loss is propagated all the way through to the character vector embeddings.

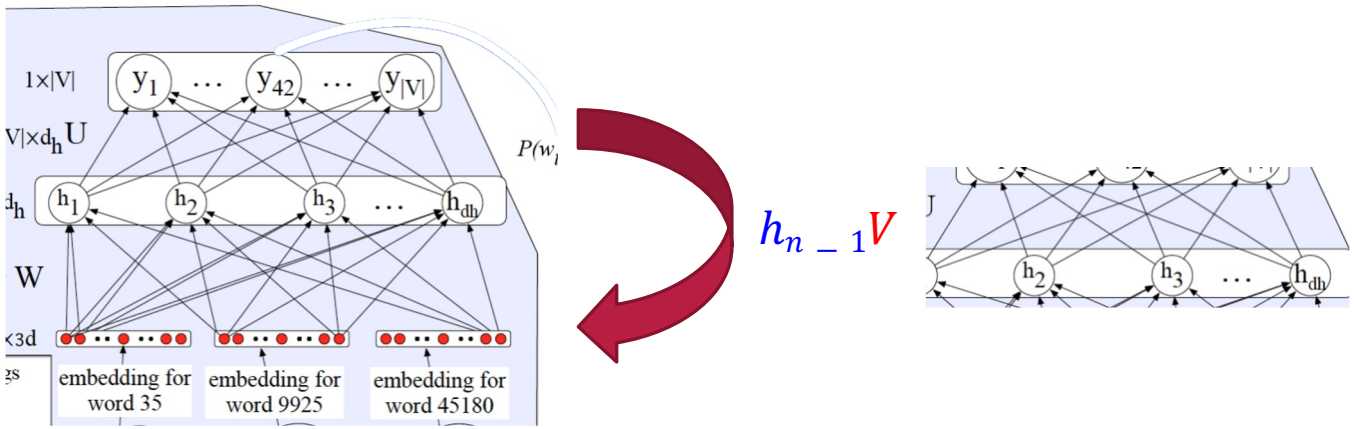
Managing Context in RNNs

$$Y_n = softmax(hnU)$$

$$\mathrm{h_n} = \tanh(Xn\,W + b + h_{n\_1}V)$$

$$X_n\,W + b$$

$$X_n$$



fully consider the whole history so far

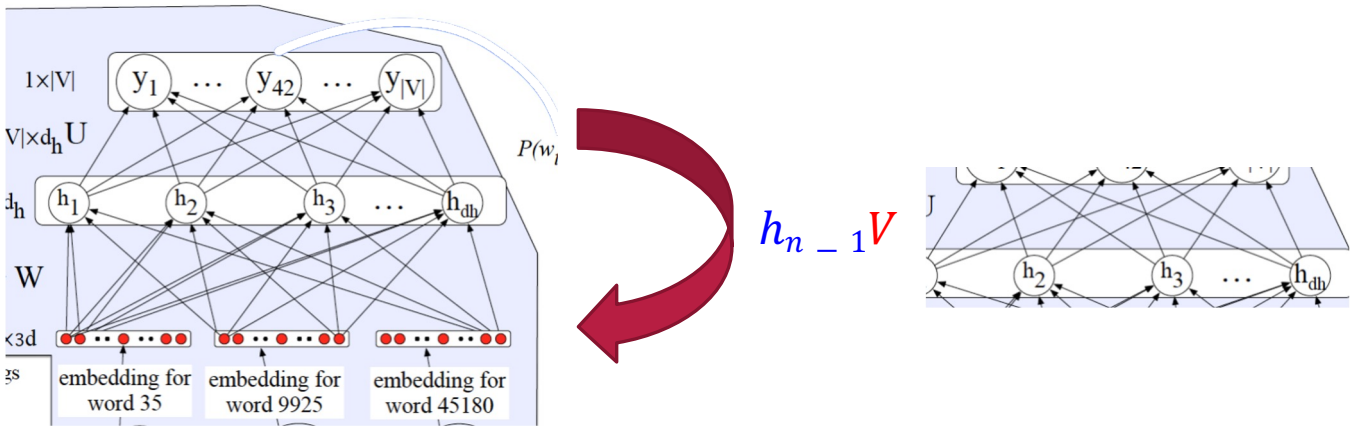
Managing Context in RNNs

$$Y_n = softmax(hnU)$$

$$\mathrm{h_n} = \tanh(Xn\,W + b + h_{n\_1}V)$$

$$X_n\,W + b$$

$$X_n$$

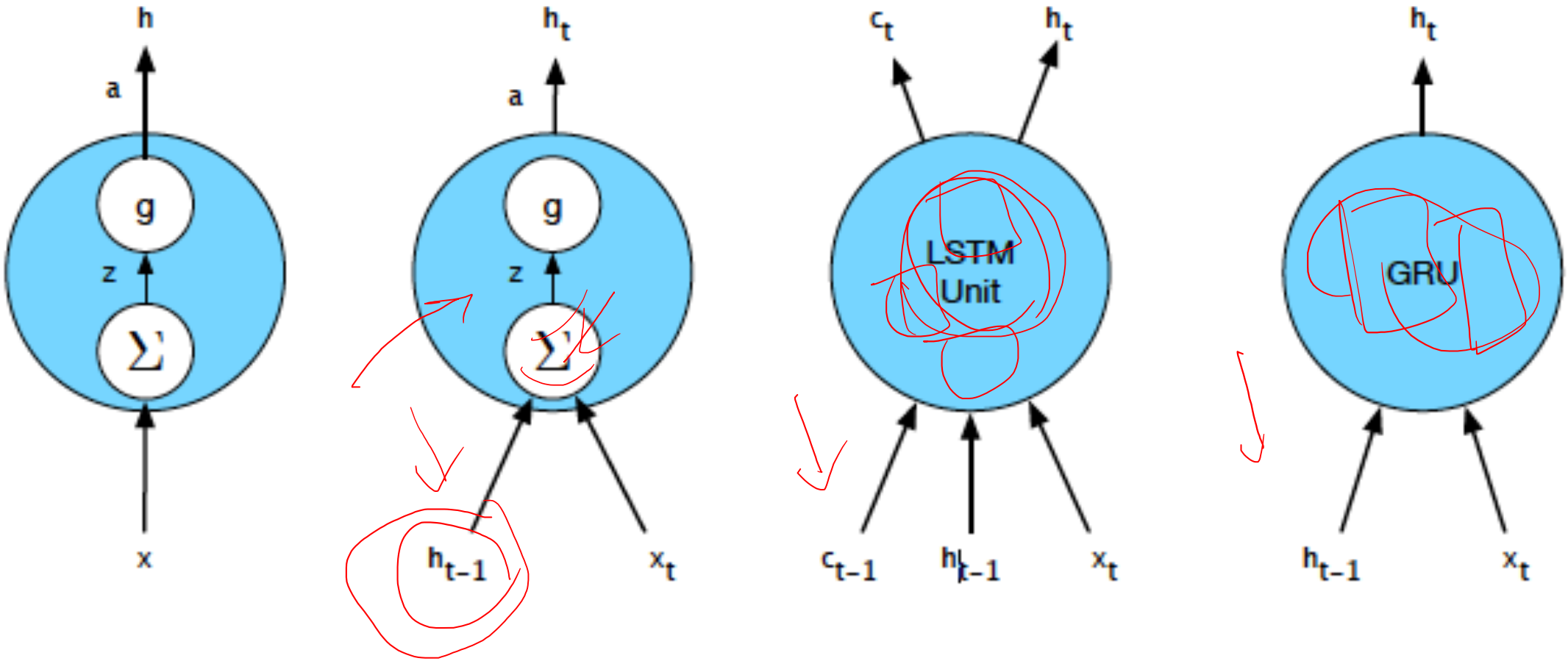


Forget: drop some part of history

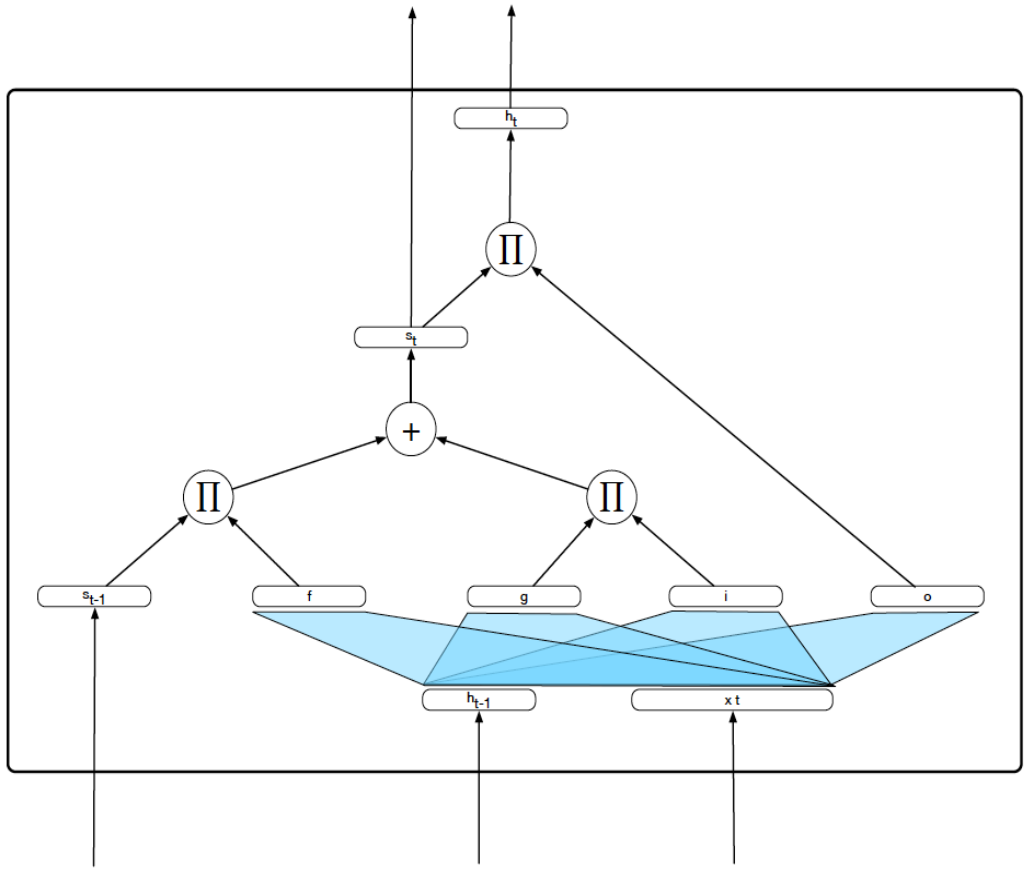
Remember: only consider the important ones

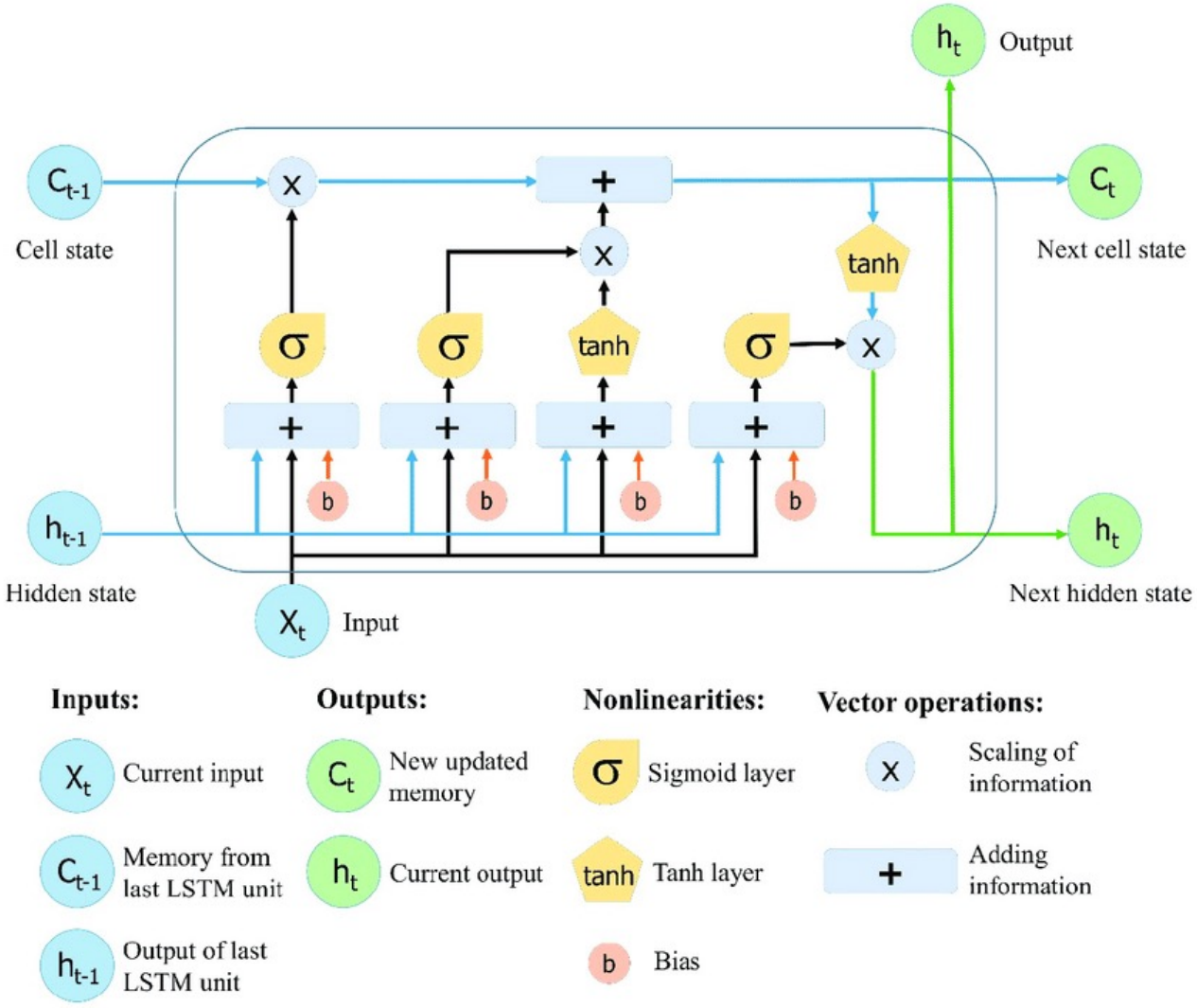
Recall: try to remember those that forgotten!

Managing Context in RNNs



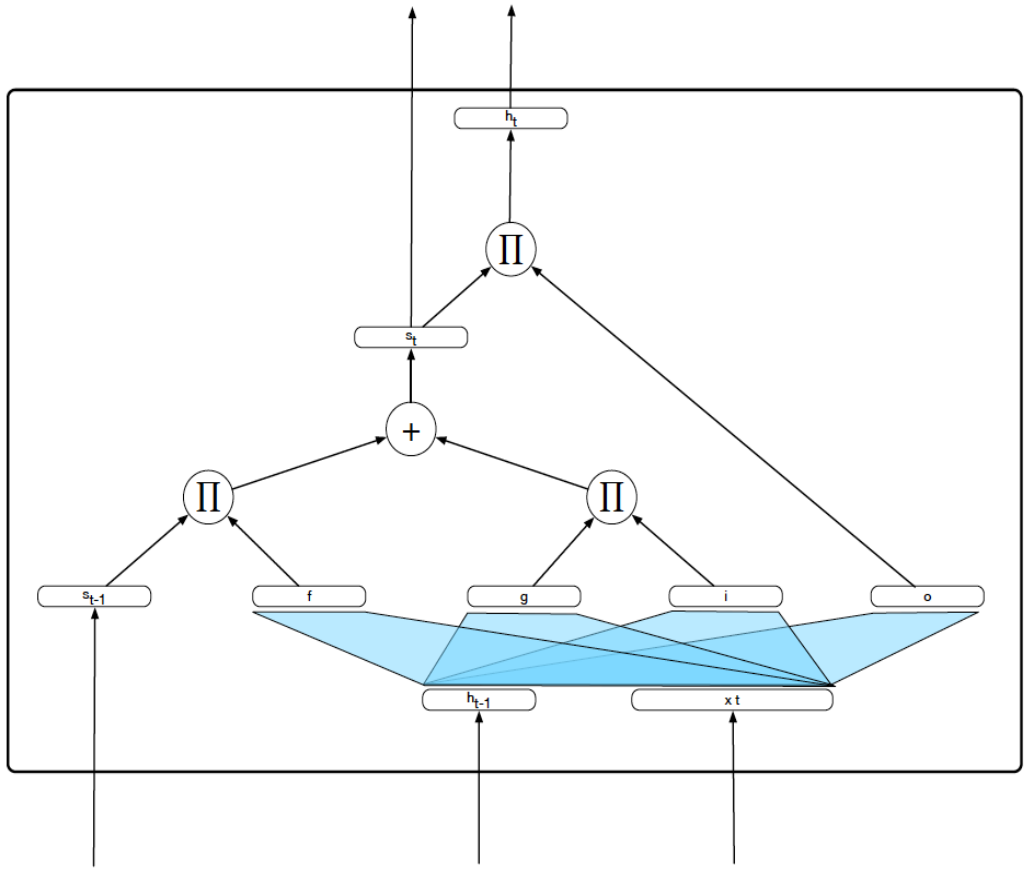
Long Short-Term Memory

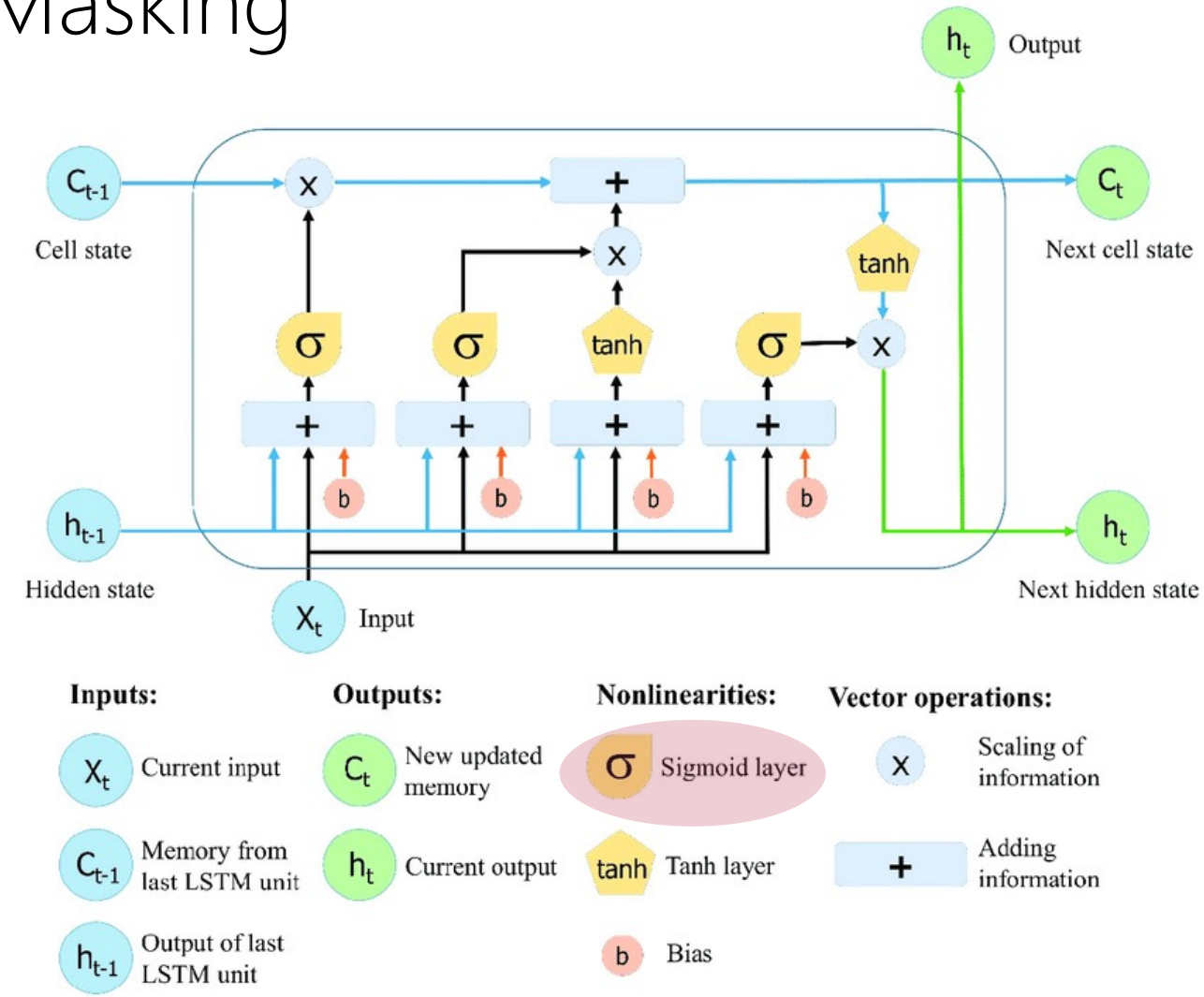




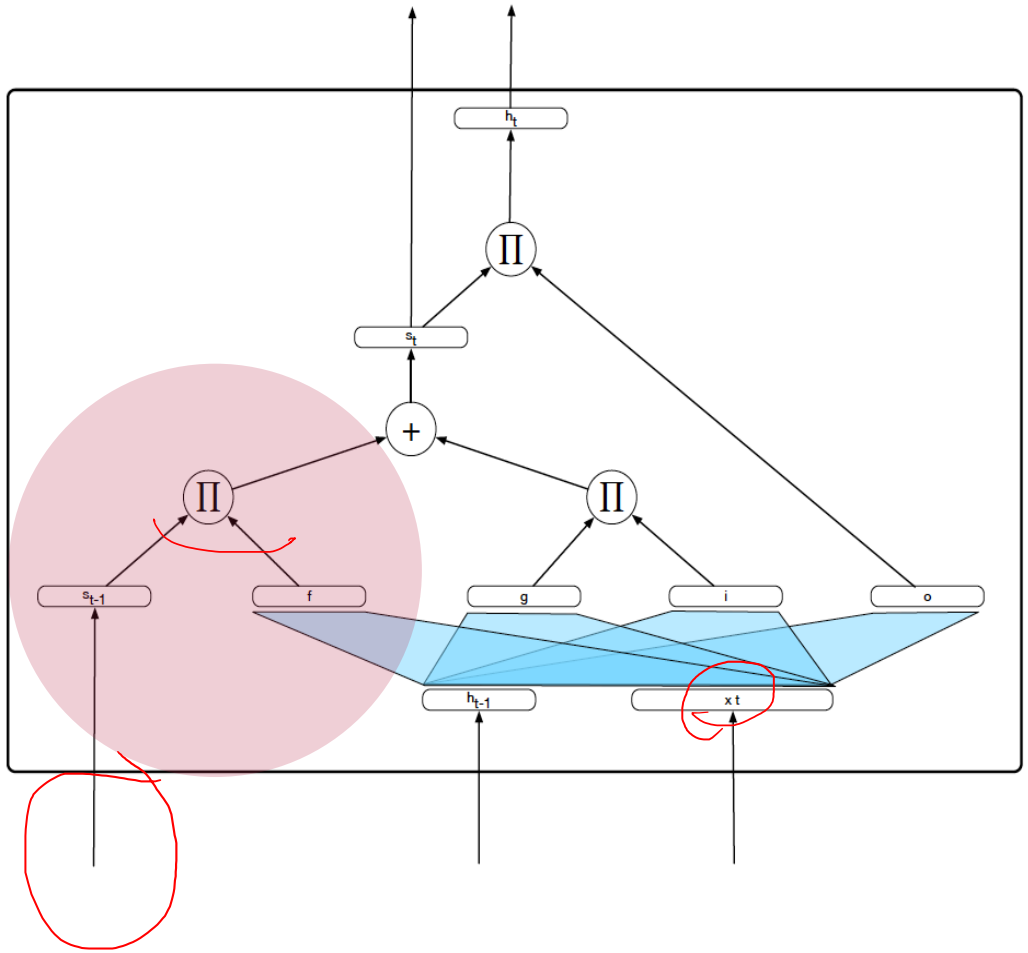
Long Short-Term Memory

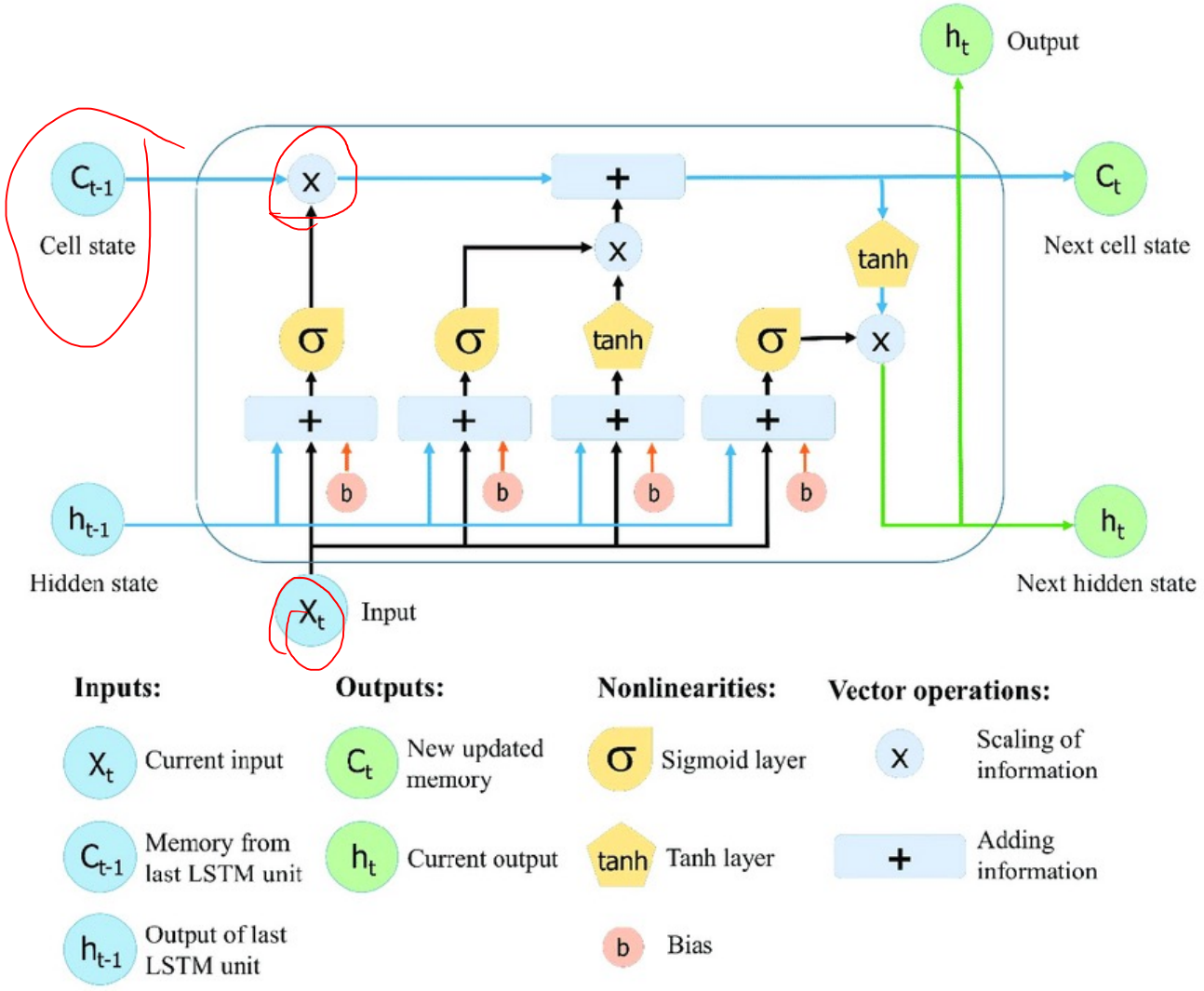
Sigmoid: Probabilistic Binary Masking



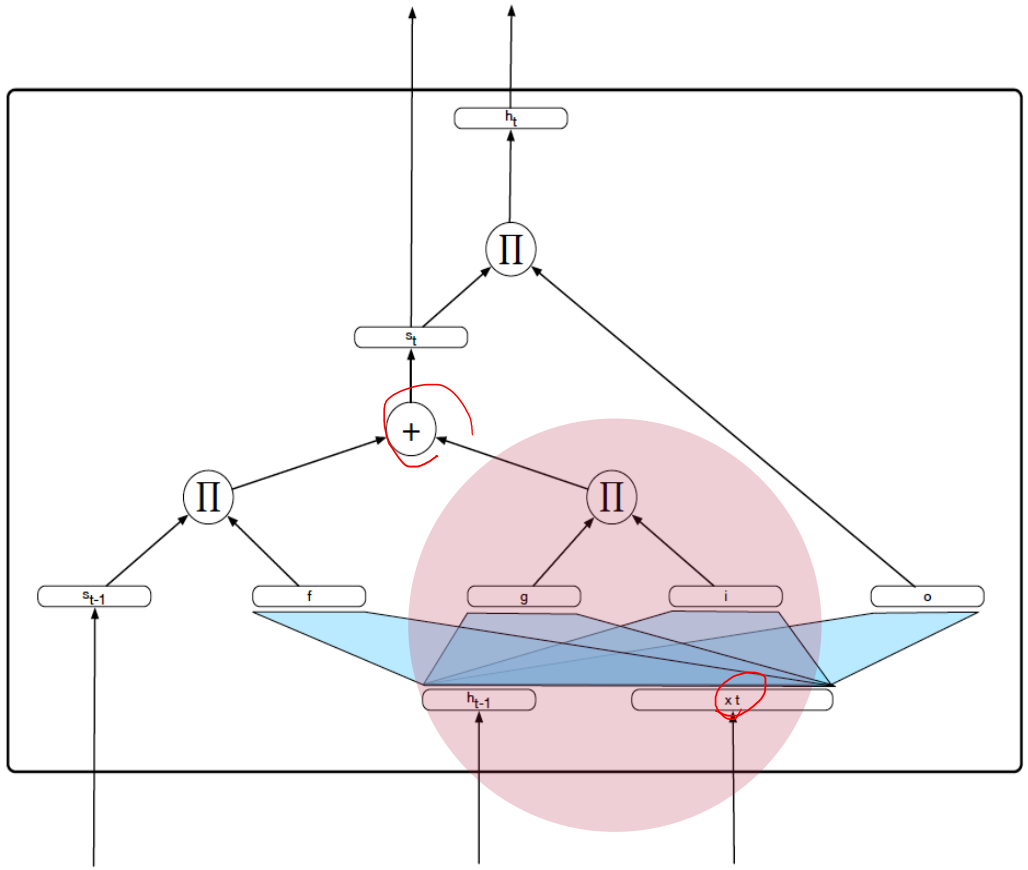


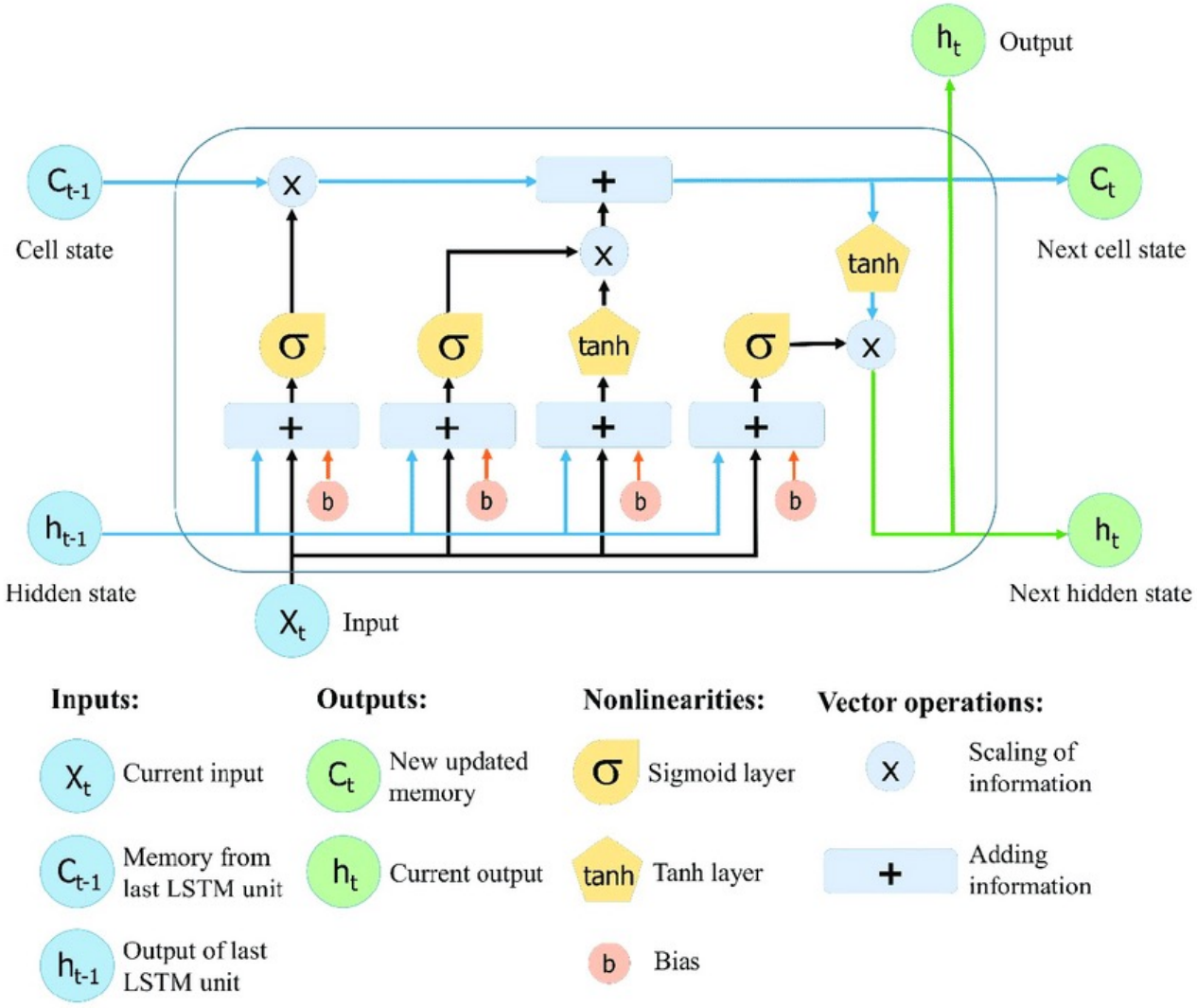
LSTM: Forget from the long memory



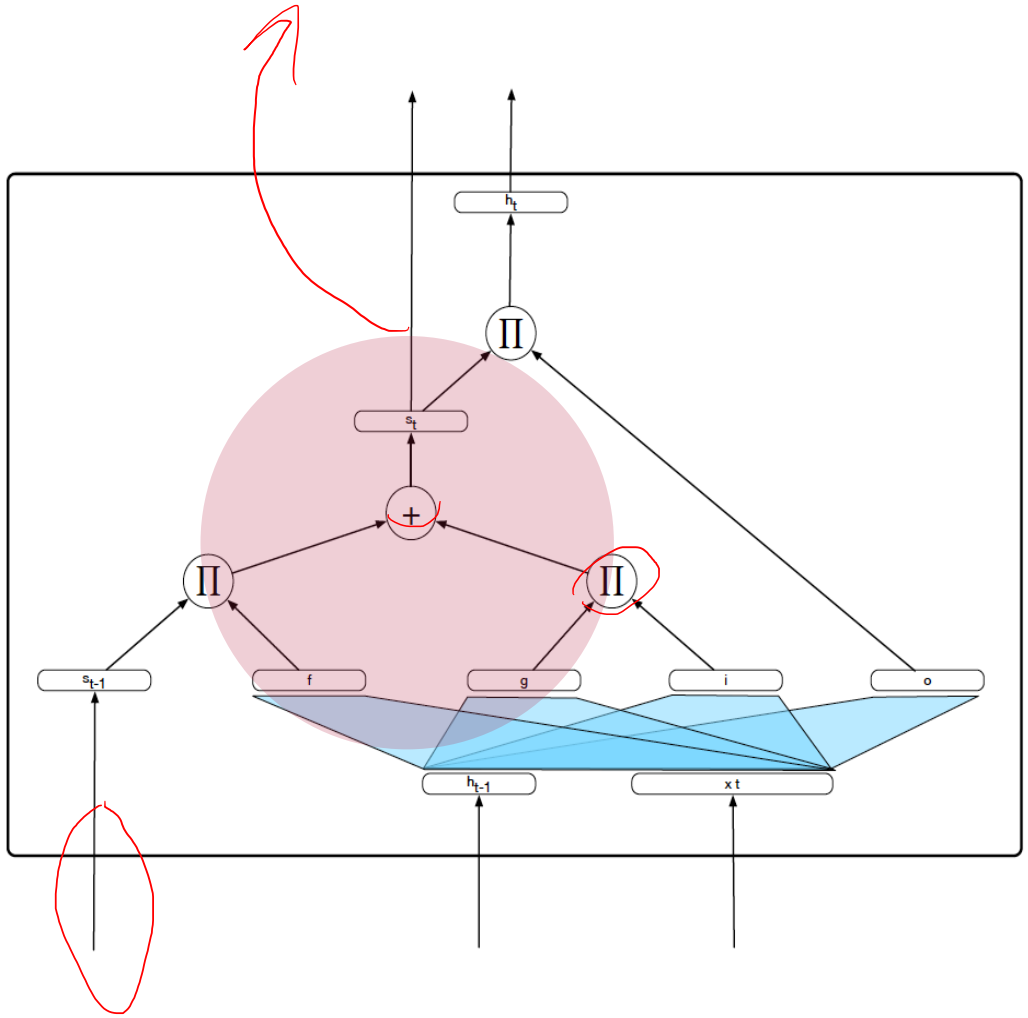


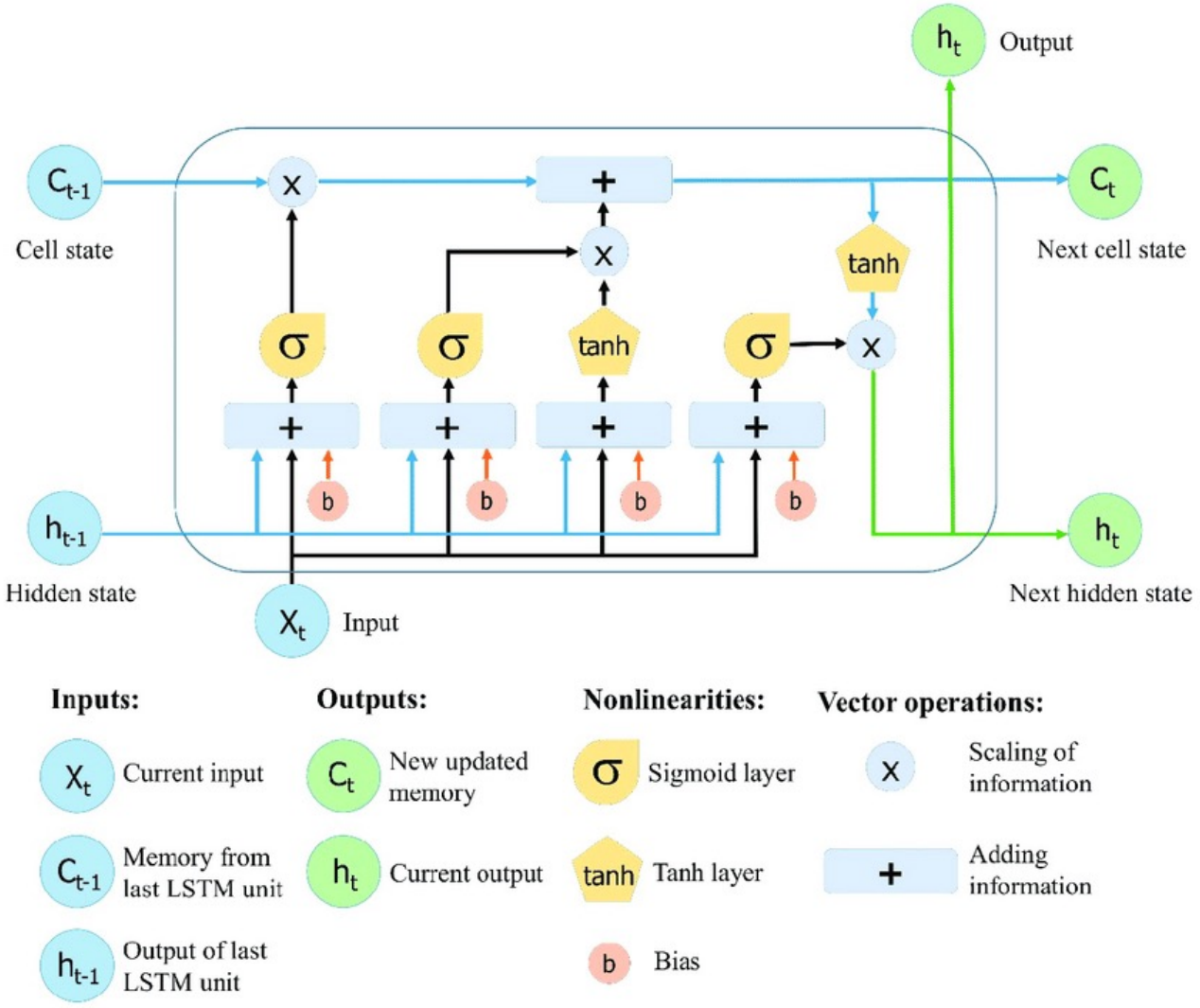
LSTM: Add to the long memory



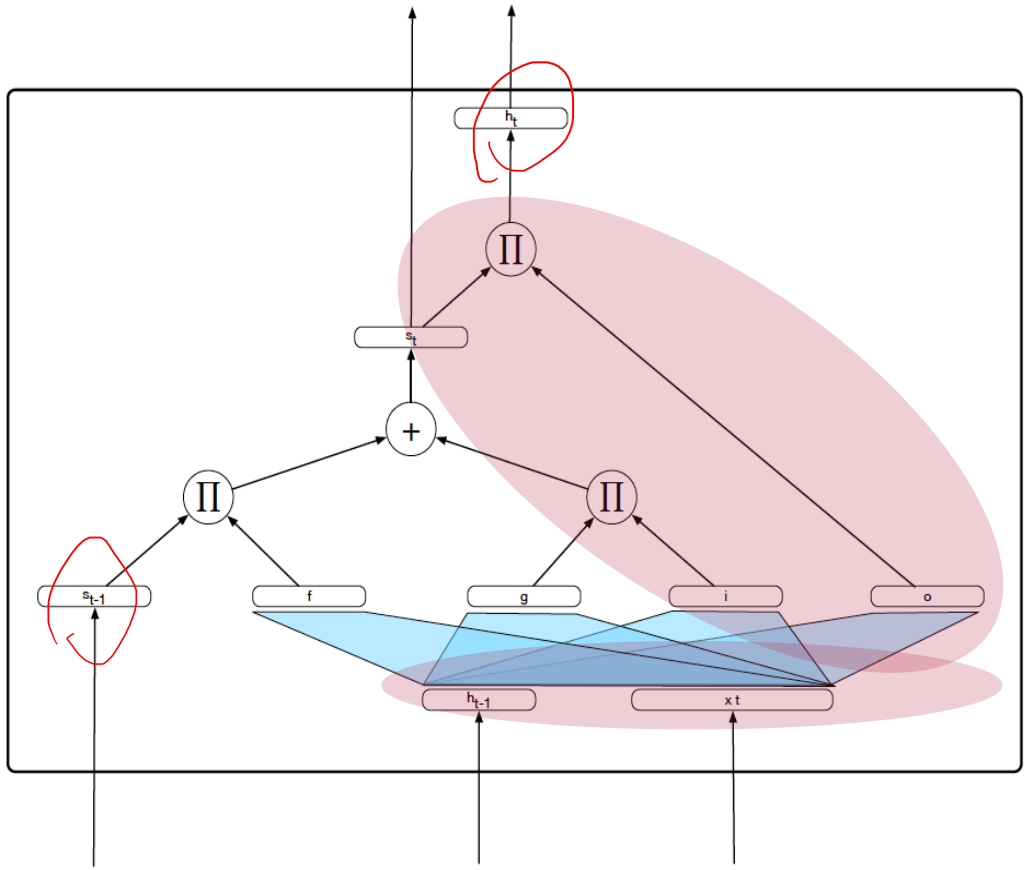


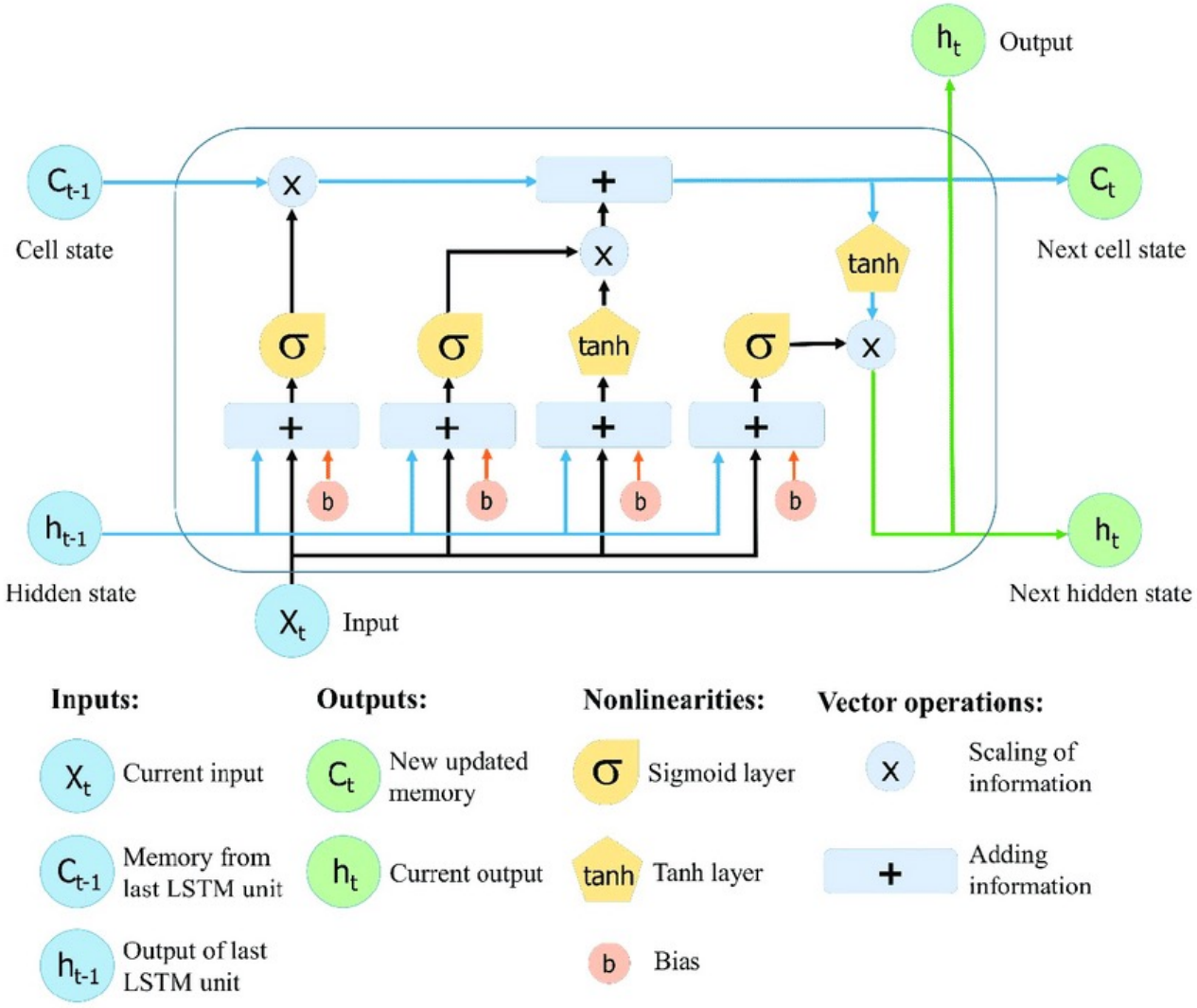
LSTM: Updated long memory





LSTM: Short memory





Gated Recurrent Unit (GRU)

LSTM: too many weights!