

# Deep Learning for NLP

The lecture  
starts at 13:15

Florina Piroi

# What we did last week

- Vector Semantics & Embeddings
  - Lexical and Vector Semantics
  - Words as Vectors
  - Measuring similarity & tf-idf
  - Word2Vec
- Neural Networks
  - Perceptron, units, activation functions
  - Feed forward
  - Training
- Neural Language Models

# Contents

- Neural Language Models
- Recurrent Neural Networks
- LSTMs (Long Short-Term Memory Networks)

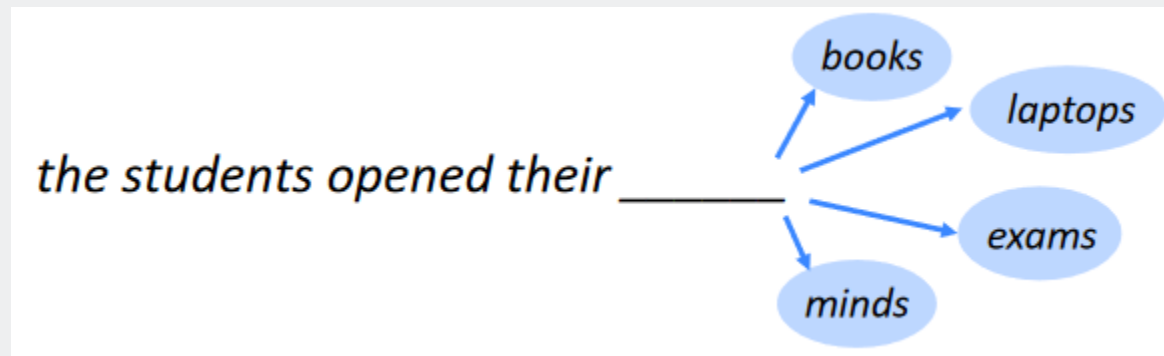
# Neural Language Models

# Relevant Literature

- Jurafsky & Martin, SLP, 3rd Edition: *Chapters 7, 9*
  - (including slides), references therein
- Cho, 2017, NLU with Distributional Representation, Chapters 4, 5
- Other material listed on individual slides

# What is a “Language Model”?

- A model that predicts  $P(W)$  or  $P(w_n | w_1, w_2 \dots w_{n-1})$
- Probabilistic Language Models
  - Compare probabilities of sequences of words
  - Probability of upcoming word



# What is a “Language Model”?

- A model that predicts  $P(W)$  or  $P(w_n | w_1, w_2 \dots w_{n-1})$

*“A language model is a function that puts a probability measure on strings drawn from some vocabulary.”*

(Manning, Raghvan, Schütze – An Introduction to Information Retrieval, 2009, Cambridge UP)

$$\begin{aligned} P(\text{frog said that toad likes frog}) &= (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01) \\ &\quad \times (0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.2) \\ &\approx 0.0000000000001573 \end{aligned}$$

Model $M_1$	
the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
dog	0.005
cat	0.003
monkey	0.001
...	...

$$\sum_{t \in V} P(t) = 1$$

This is a unigram model, aka. “Bag of Words” model

# What is a “Language Model”?

- A model that predicts  $P(W)$  or  $P(w_n | w_1, w_2 \dots w_{n-1})$
- How did you compute  $P$ ?
  - Count and divide
  - Markov Assumption

$$P(\text{the} | \text{its water is so transparent that}) = \frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}$$

$$P(\text{the} | \text{its water is so transparent that}) \gg P(\text{the} | \text{that})$$

Works well in practice but theoretically not correct

$$P(\text{the} | \text{its water is so transparent that}) \gg P(\text{the} | \text{transparent that})$$

- Unigrams:  $P(w_n)$
- Bi-grams:  $P(w_n | w_{n-1})$
- ...
- N-grams:  $P(w_n | w_1, w_2 \dots w_{n-1})$



# Language Model: A simple (bi-gram) example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

Symbols for the start and end  
of a sentence

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

We also need a way to handle the 0 probabilities to avoid computations vanishing  
Some smoothing or assigning very small probabilities are often assigned

$$P(\text{like} | \text{I}) = 0$$

# Language Model

- A model that predicts  $P(W)$  or  $P(w_n | w_1, w_2 \dots w_{n-1})$
- Many types of LMs
  - N-gram based
  - Grammar-based
  - Context-free grammars
  - ...
- Less complex in IR (Information Retrieval)
- Performance measurement
- Issues: zero probabilities, smoothing, interpolation

Perplexity measure used to assess how well we are doing

Perplexity

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$P(\text{like} | \mathbf{I}) = 0$$

# Neural Language Model

- No smoothing
  - Longer histories (compared to the fixed N in "N-gram")
  - Generalize over contexts
    - "chases a dog" vs. "chases a cat" vs. "chases a rabbit"
  - Higher predictive accuracy!
  - Further models are based on NLMs.
- Slower to train!

# Neural Language Model - Definition

- Standard Feed-Forward Network
- Input: a representation of previous words ( $w_1, w_2, \dots$ ) We are looking at the whole vocabulary
- Output: probability distribution over possible next words.

$$P(w_n \mid w_1, w_2 \dots w_{n-1}) = f_{\theta}^{w_n}(w_1, w_2 \dots w_{n-1})$$

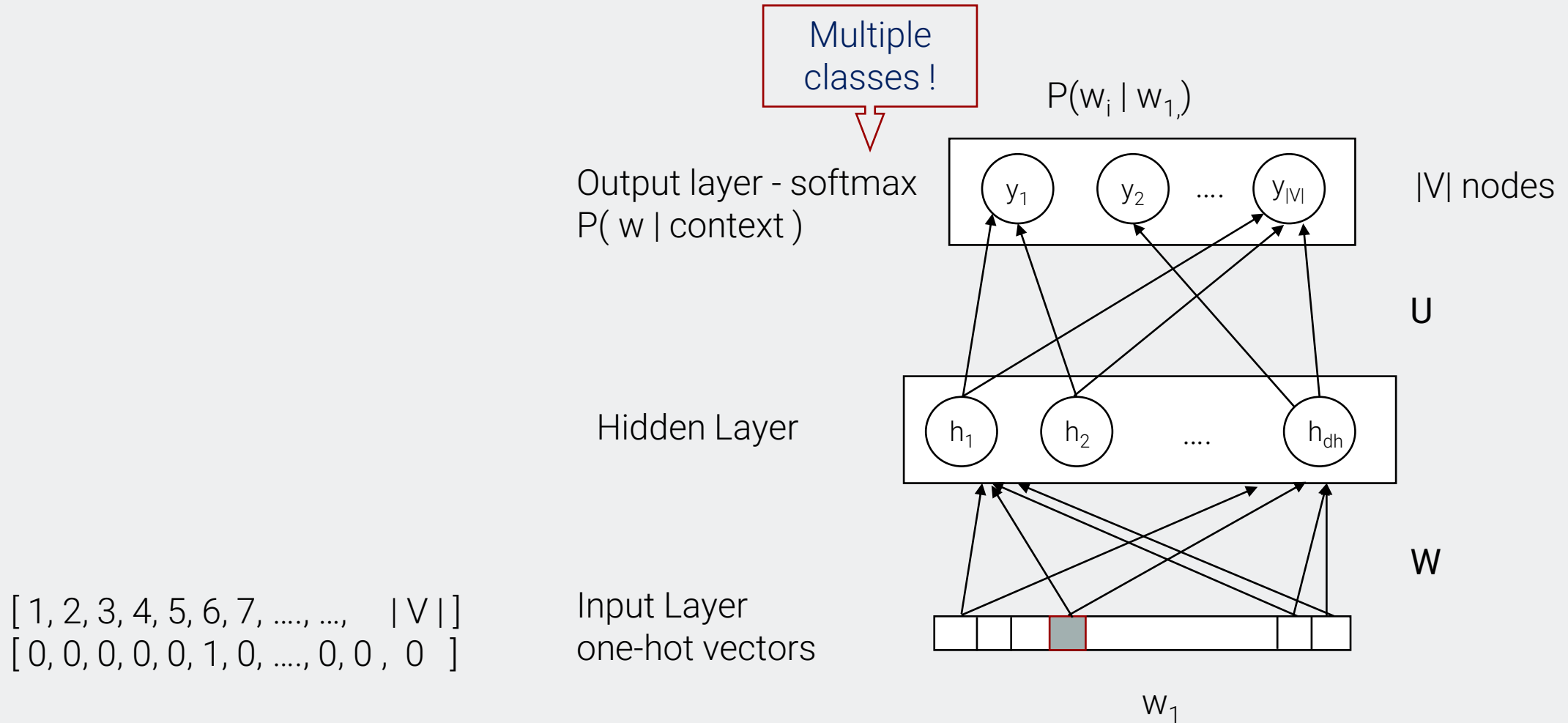
i.e.: find the function

# Neural Language Model - Input

- Standard Feed-Forward Network
  - **Input:** a representation of previous words ( $w_1, w_2, \dots$ )
  - Output: probability distribution over possible next words.
  - N-grams used exact words! (  $P(\text{"cat"})$  )
  - Equi-distance! (lowest prior knowledge)
  - 1-of-N encoding (aka. one-hot vector)
- I have  $V$  words and use one hot encoding to make words encoding orthogonal between each other so they carry no information among them

	a	hair		cat		zombie				
vocabulary index:	1	2	3	4	5	6	7	....	...	$ V $
	0	0	0	0	0	1	0	....	...	0

# Feed Forward Net - Execution



# Feed Forward Net - Training

Take negative and positive samples from training text  
and update parameters accordingly.

The good thing is that in the end you also end up with embeddings for your data in E

Positive samples ( $w_3, w_{402}$ )  
(metal jacket)

Negative samples ( $w_3, w_{xx}$ )  
(metal heavy)  
(metal towel)

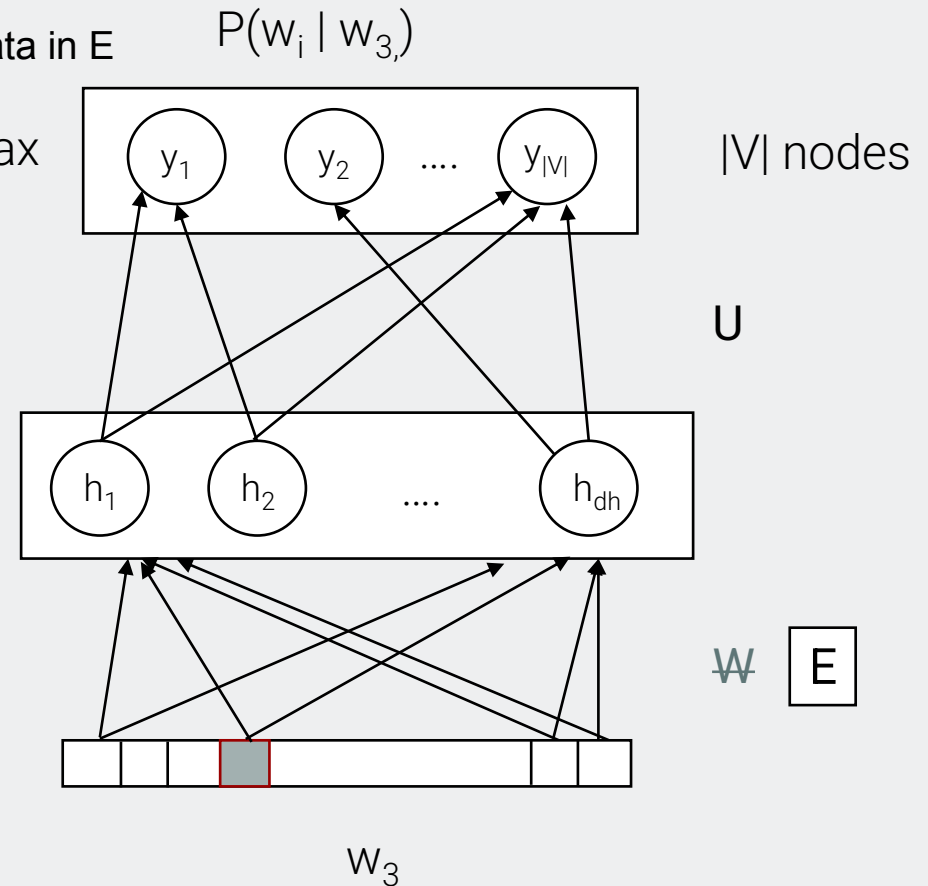
Here very simple example  
only looking at bigrams

[ 1, 2, 3, 4, 5, 6, 7, ..., ..., |V| ]  
[ 0, 0, 0, 0, 0, 1, 0, ..., 0, 0, 0 ]

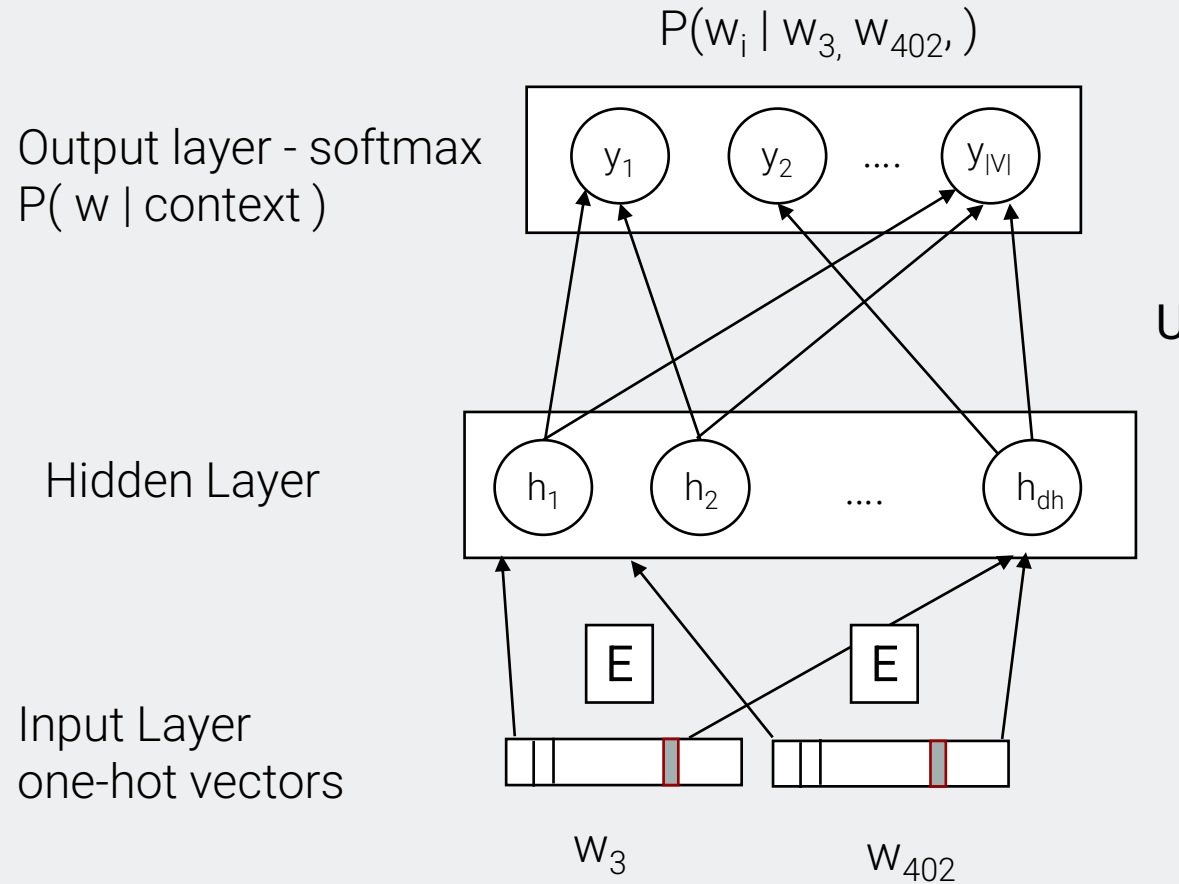
Output layer - softmax  
 $P(w | \text{context})$

Hidden Layer

Input Layer  
one-hot vectors

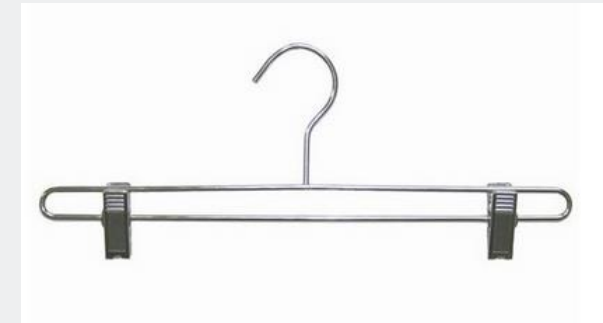


# Feed Forward Net - Training



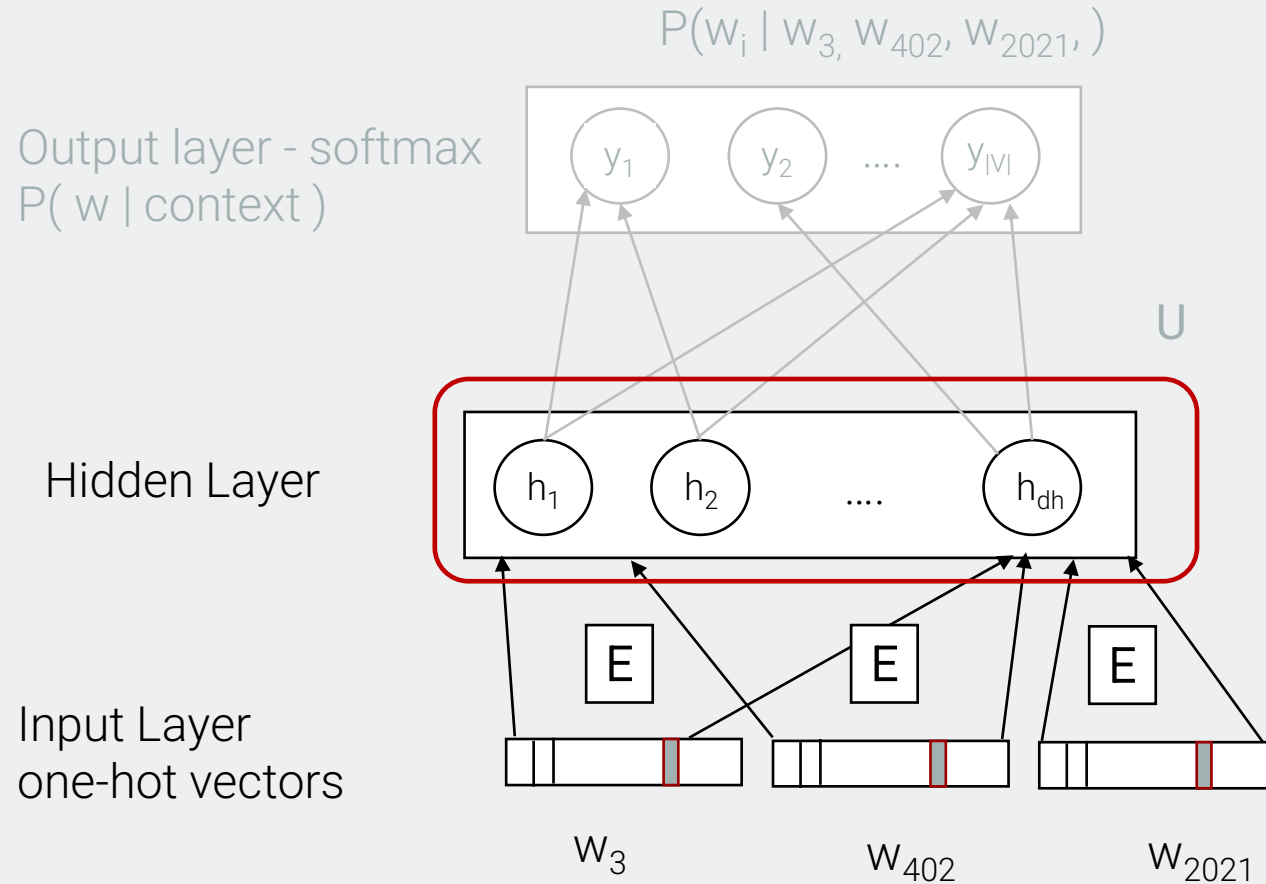
Positive samples ( $w_3, w_{402}, w_{2021}$ )  
(metal skirt hanger)

Negative samples ( $w_3, w_{402}, w_{xx}$ )  
(metal skirt mouse)  
(metal skirt towel)



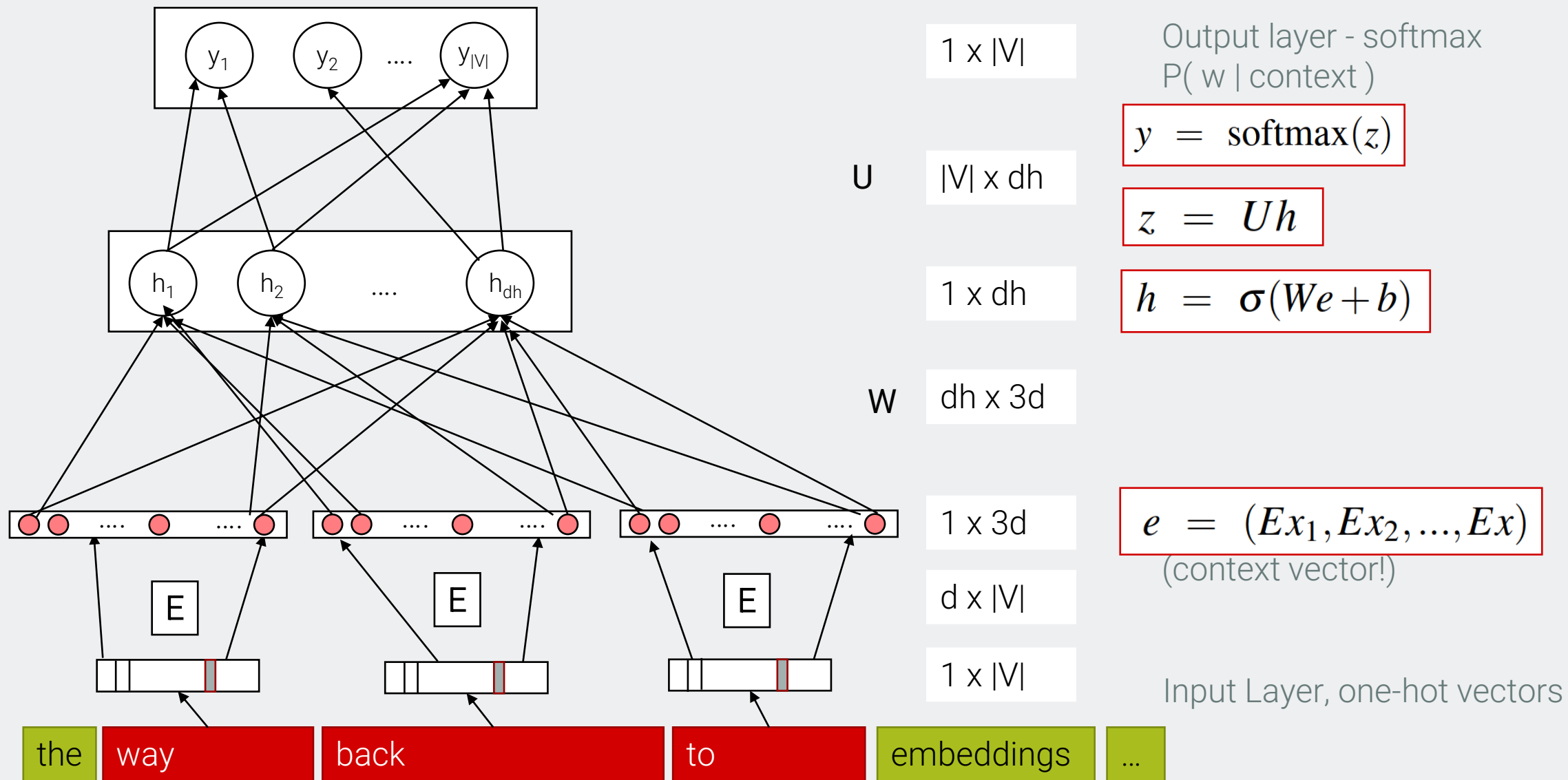


# Feed Forward Net - Training



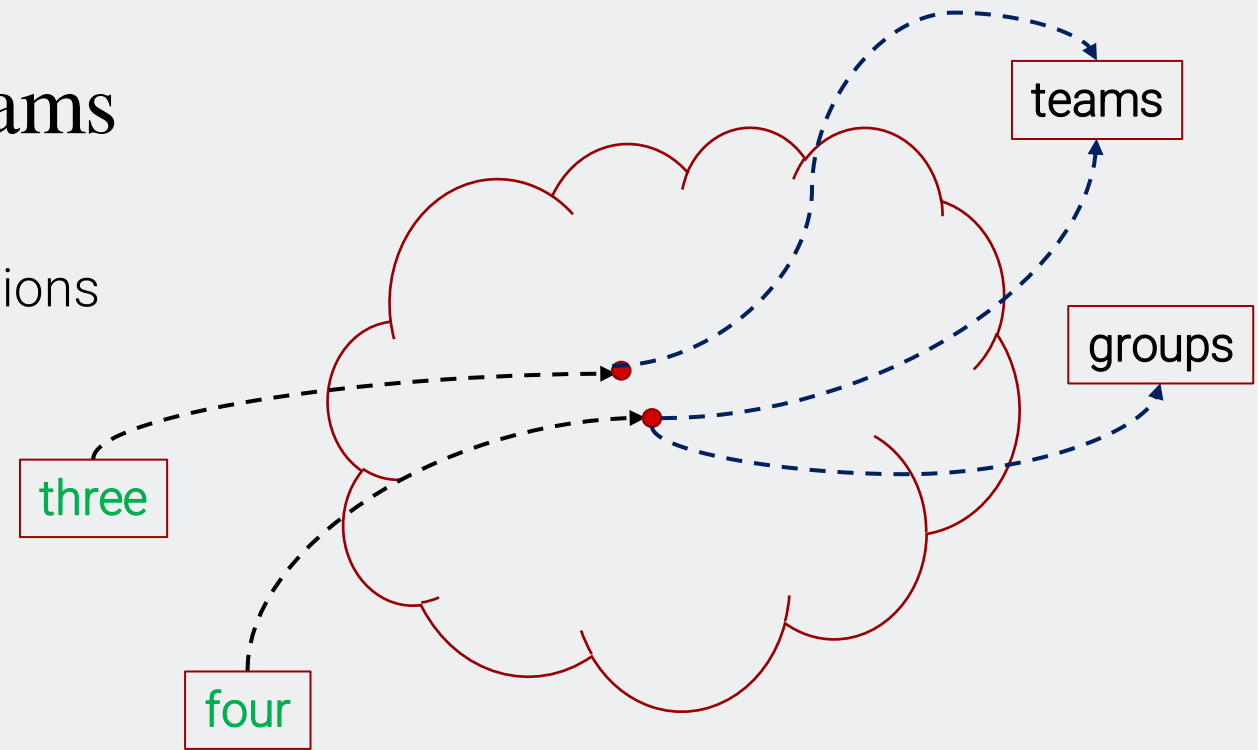
Projection layer

E gives us at the end the word embeddings for this specific data set



# Generalization to Unseen n-grams

- There are **three** teams left for the qualifications
- **four** teams have passed the first round
- **four** groups are playing in the field

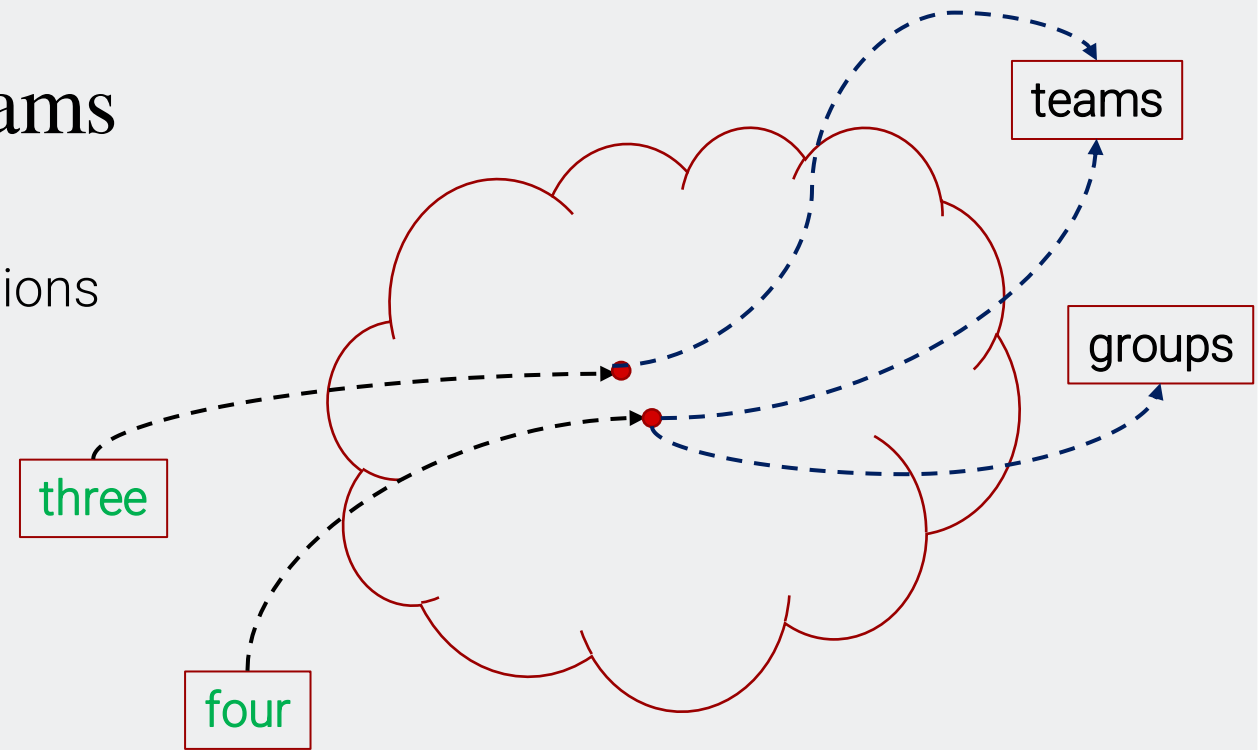


We have three sentences and I am looking at the bigrams (green-bold black)  $P(\text{teams} \mid \text{four}) \approx P(\text{groups} \mid \text{four})$   
During training the neural network will push the vectors three and four together

- (during training) and so the probability of teams to follow them.
- **context** If I ask the model for the probability of phrase “three groups”.  
It will see that three and 4 are close so it will be able to compute the similarity so that  
also with  
3 it will be quite similar

# Generalization to Unseen n-grams

- There are **three** teams left for the qualifications
- **four** teams have passed the first round
- **four** groups are playing in the field
- Assign probability to “three groups”



$$P(\text{teams} \mid \text{four}) \approx P(\text{groups} \mid \text{four})$$

- (during inference)
- **context**

# Neural Language Models – In a small nutshell

- pattern recognition problems
- Data-driven
- High performance in many problems
- No domain knowledge needed
- Generalization
  
- Data-hungry (bad for small data sets)
- Cannot handle symbols very well For example formulas, molecules, ...
- Computationally high costs

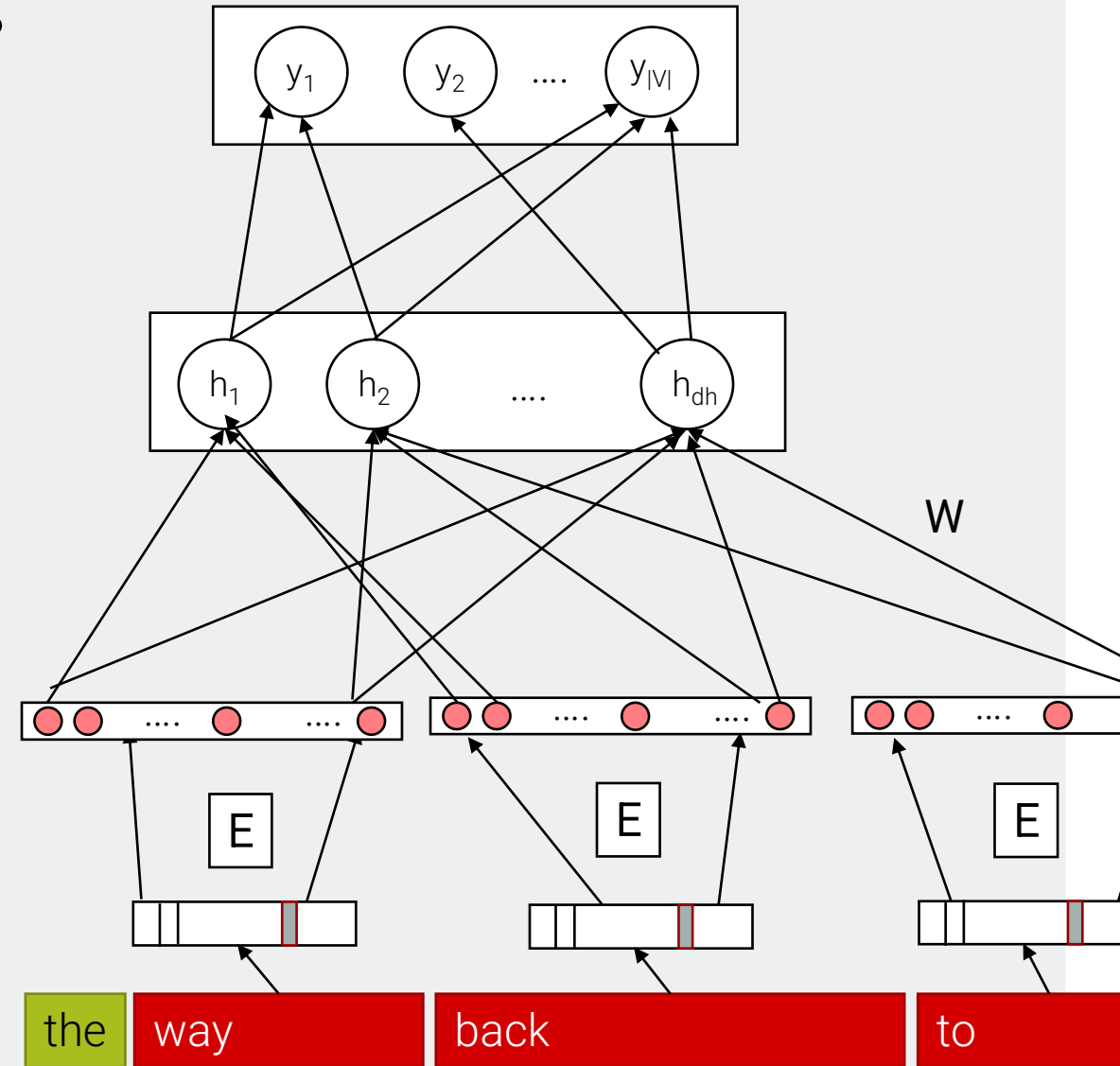
# Content

- Neural Language Models
- Recurrent Neural Networks
- LSTMs (Long Short-Term Memory Networks)

You always have a fixed number of inputs, if you increase it so will the weights  
Moreover, you would like to just hand it the whole data not sets of words one at a time

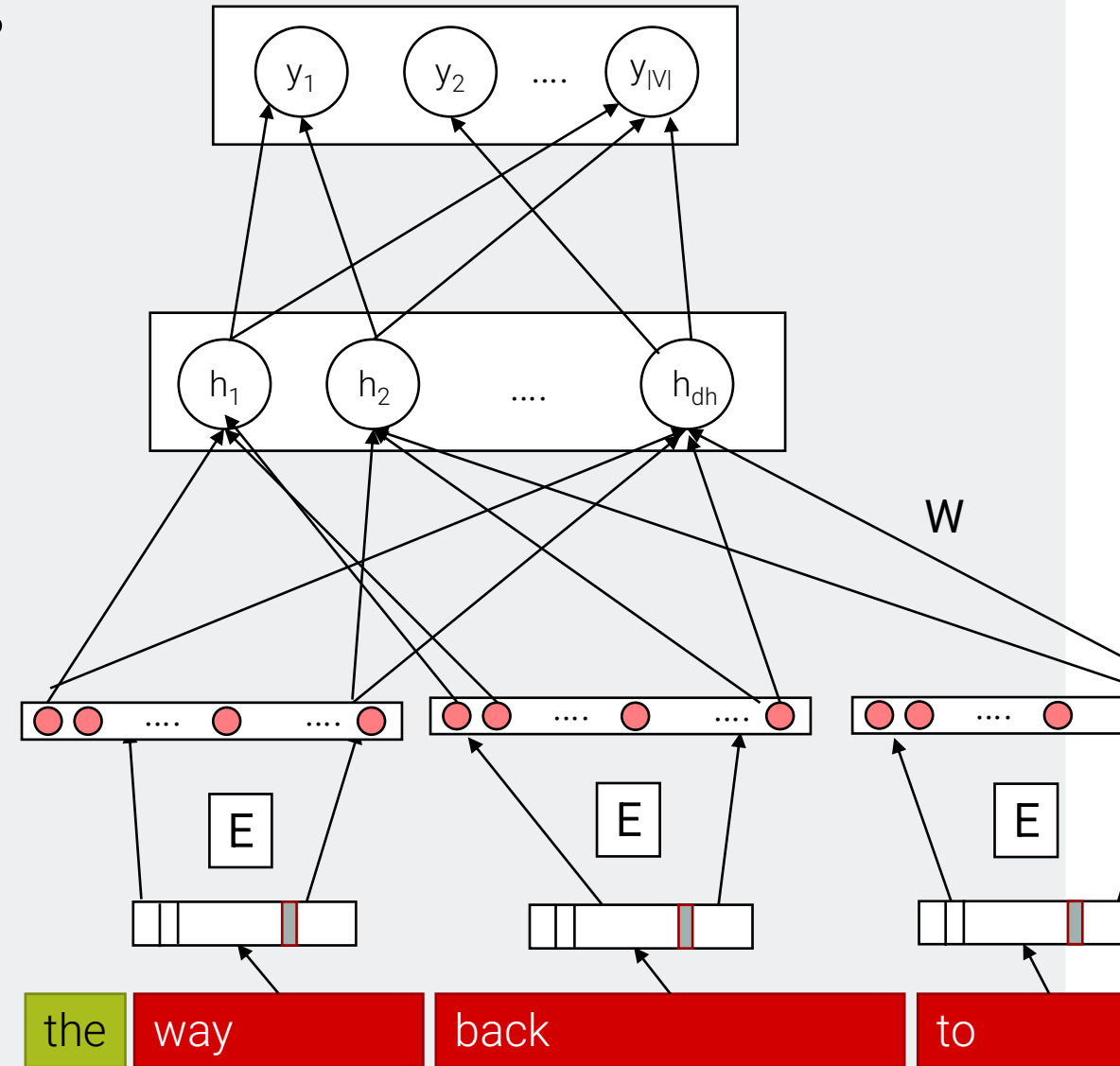
## (Simple) Neural Language Models

- Improvements over n-gram LM
  - No sparsity problem
  - Don't need to store all observed n-grams
- Remaining problems:
  - Fixed window is too small
  - Enlarging window enlarges  $W$
  - Window can never be large enough!
  - (embedded) words are multiplied by completely different weights in  $W$   
(No symmetry in input processing)  
Cause at every iteration you update the weights



# (Simple) Neural Language Models

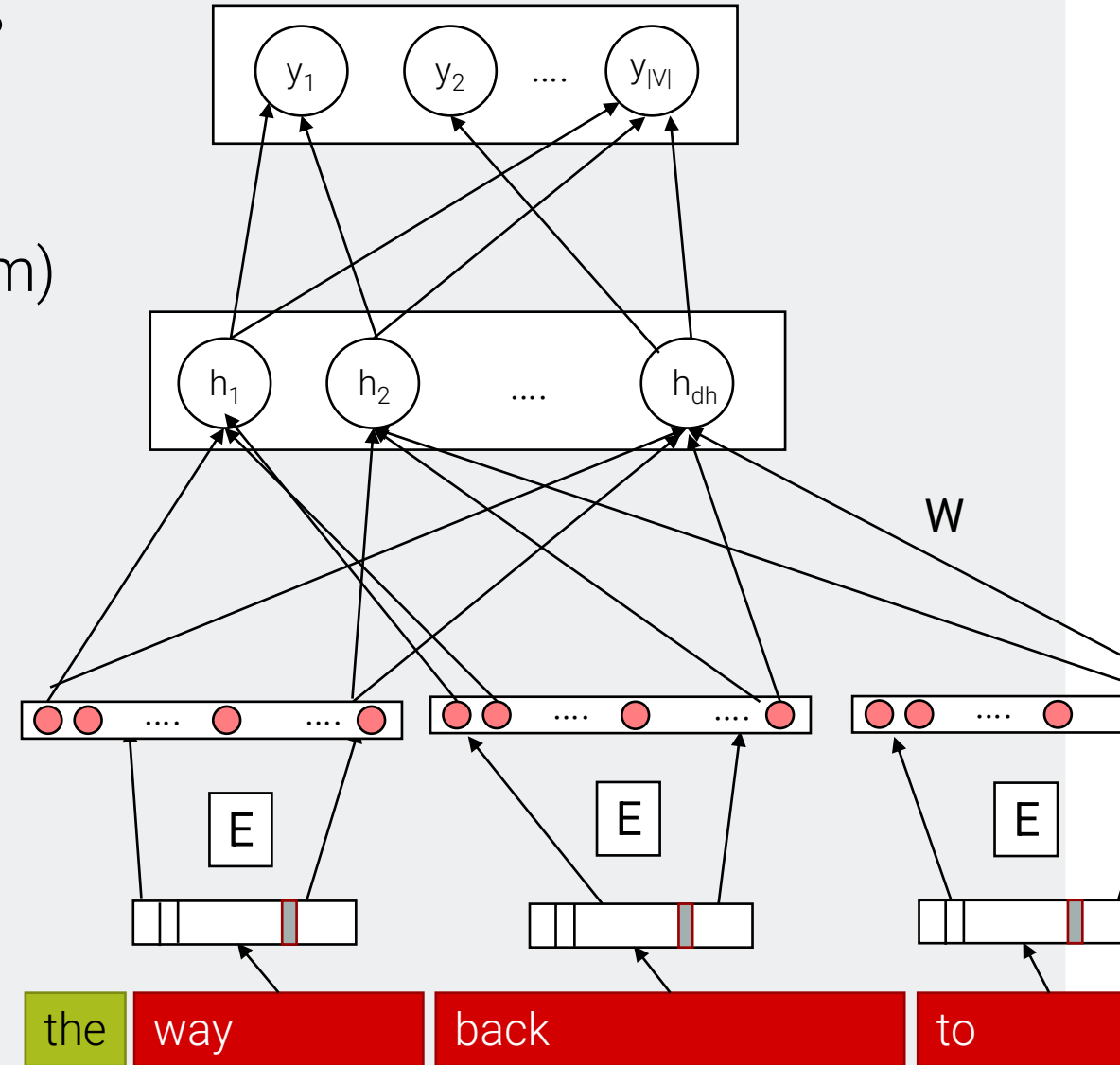
- How to deal with inputs of varying lengths (i.e. sequences)?
- Slide the input window
- Still, decision on one window does not influence decision on other window.
- Cannot learn systematic patterns (e.g. Constituency)





# (Simple) Neural Language Models

- Language is temporal (continuous stream)
  - “Sequence that unfolds in time”
- Algorithms use this
  - Viterbi (see SLP3 book)
- Previous ML approaches have access to all input, simultaneously
- How to deal with *sequences of varying lengths*?



# Sequences – Input of **Variable Lengths**

$$x^1 = (x_1^1, x_2^1, \dots, x_{l^1}^1)$$

- Each input has a variable number of elements:
- Simplification: binary elements (0 or 1 values)
- How many 1s in this sequence? How can we implement that?

To handle unknown length of sentences we need to use recursive functions

- ADD1, Recursive function
- Call it for each element of the input.

---

**Algorithm 1** A function ADD1

---

```
 $s \leftarrow 0$   
function ADD1( $v, s$ )  
  if  $v = 0$  then return  $s$   
  else return  $s + 1$   
  end if  
end function
```

---

---

**Algorithm 2** A function ADD1

---

```
 $s \leftarrow 0$   
for  $i \leftarrow 1, 2, \dots, l$  do  $s \leftarrow \text{ADD1}(x_i, s)$   
end for
```

---

# Recursive Function for Natural Language Understanding

- ADD1 is hardcoded
- Parametrized recursive function
- Memory:  $\mathbf{h} \in \mathbb{R}^{d_h}$  The memory now is the hidden state  $\mathbf{h}$
- Input  $x_t$  and memory  $\mathbf{h}$ , returns the new  $\mathbf{h}$
- Time index!

$$h_t = f(x_t, \mathbf{h}_{t-1})$$

$$f(x_t, \mathbf{h}_{t-1}) = g(\mathbf{W}\phi(x_t) + \mathbf{U}\mathbf{h}_{t-1})$$

---

**Algorithm 1** A function ADD1

---

```
 $s \leftarrow 0$   
function ADD1( $v, s$ )  
  if  $v = 0$  then return  $s$   
  else return  $s + 1$   
  end if  
end function
```

---

---

**Algorithm 2** A function ADD1

---

```
 $s \leftarrow 0$   
for  $i \leftarrow 1, 2, \dots, l$  do  $s \leftarrow \text{ADD1}(x_i, s)$   
end for
```

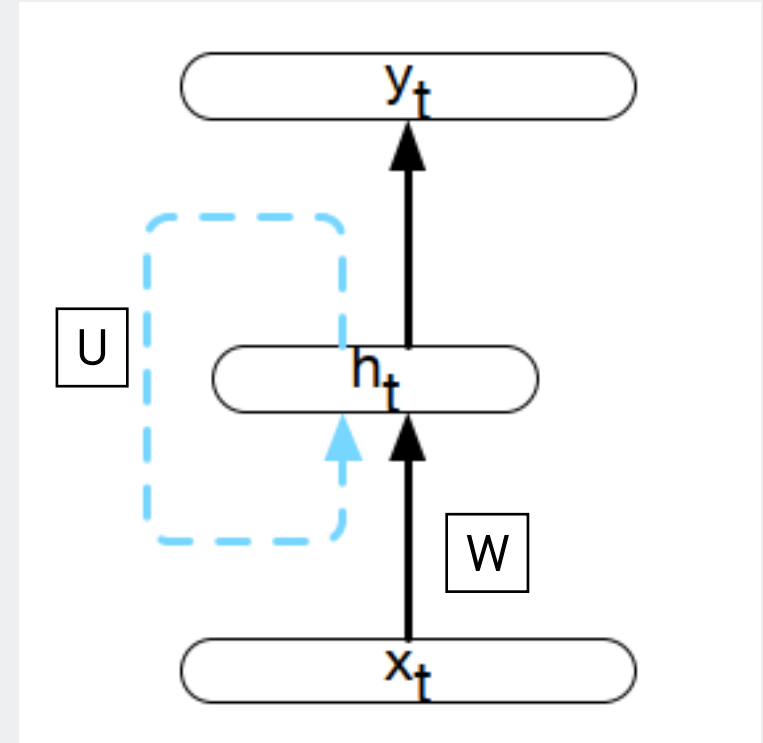
---

# Recursive Function for Natural Language Understanding

$$\mathbf{h} \in \mathbb{R}^{d_h}$$

$$h_t = f(x_t, \mathbf{h}_{t-1})$$

$$f(x_t, \mathbf{h}_{t-1}) = g(\mathbf{W}\phi(x_t) + \mathbf{U}\mathbf{h}_{t-1})$$



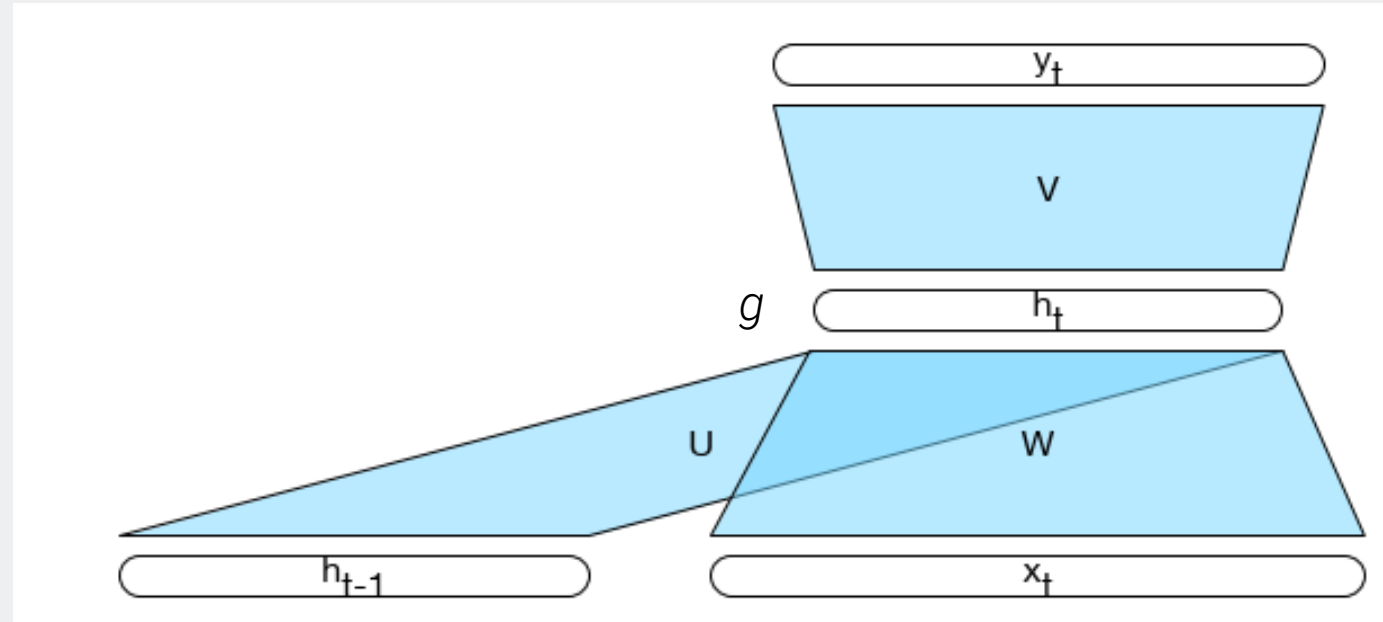
# Recursive Neural Network – Unrolled

Inferencing:

$$h_t = g(Uh_{t-1} + Wx_t)$$

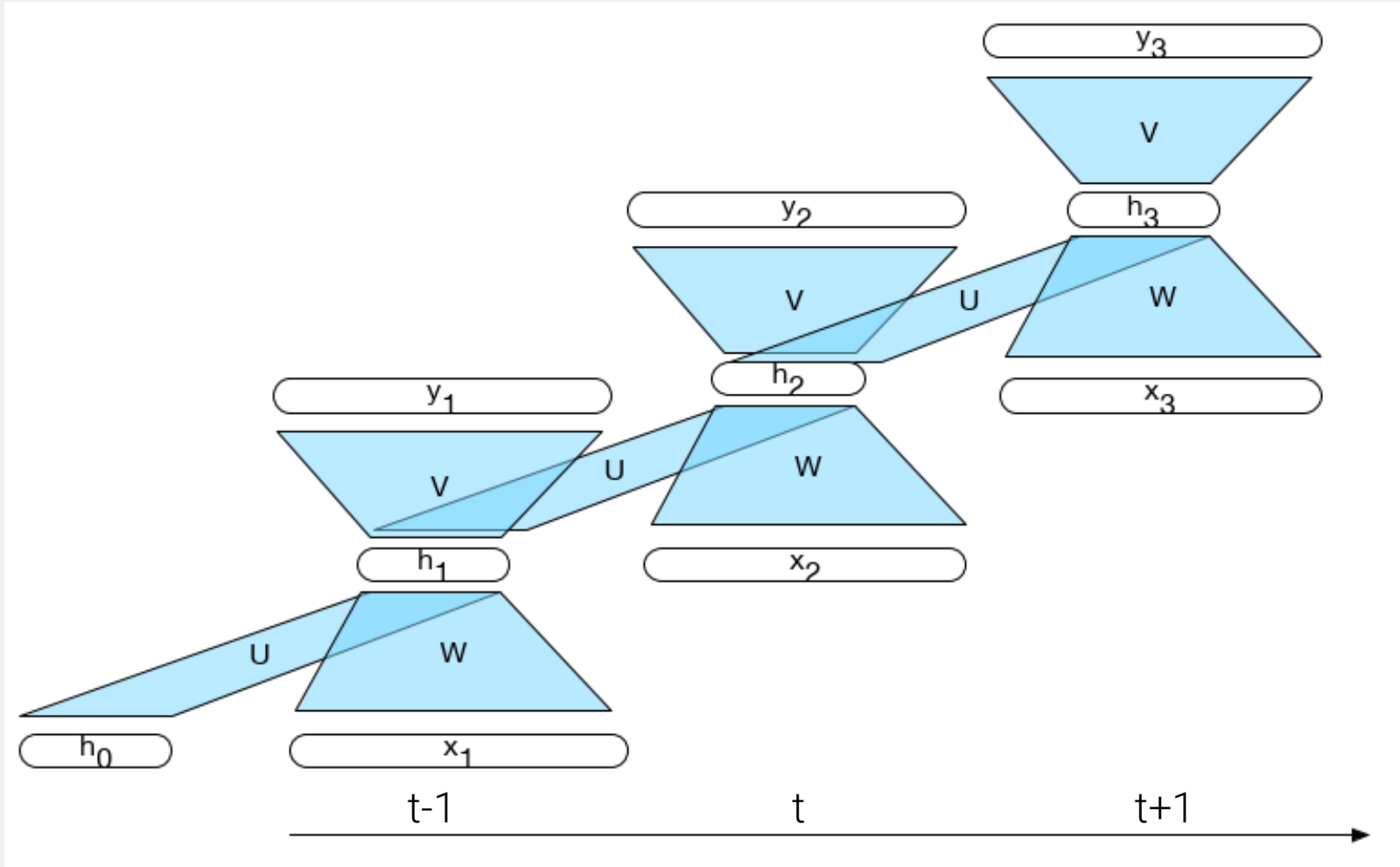
$$y_t = f(Vh_t)$$

$$y_t = \text{softmax}(Vh_t)$$



- Time dimension makes them look exotic (they aren't)
- Difference to FNN -> additional set of weights ( $U$ )

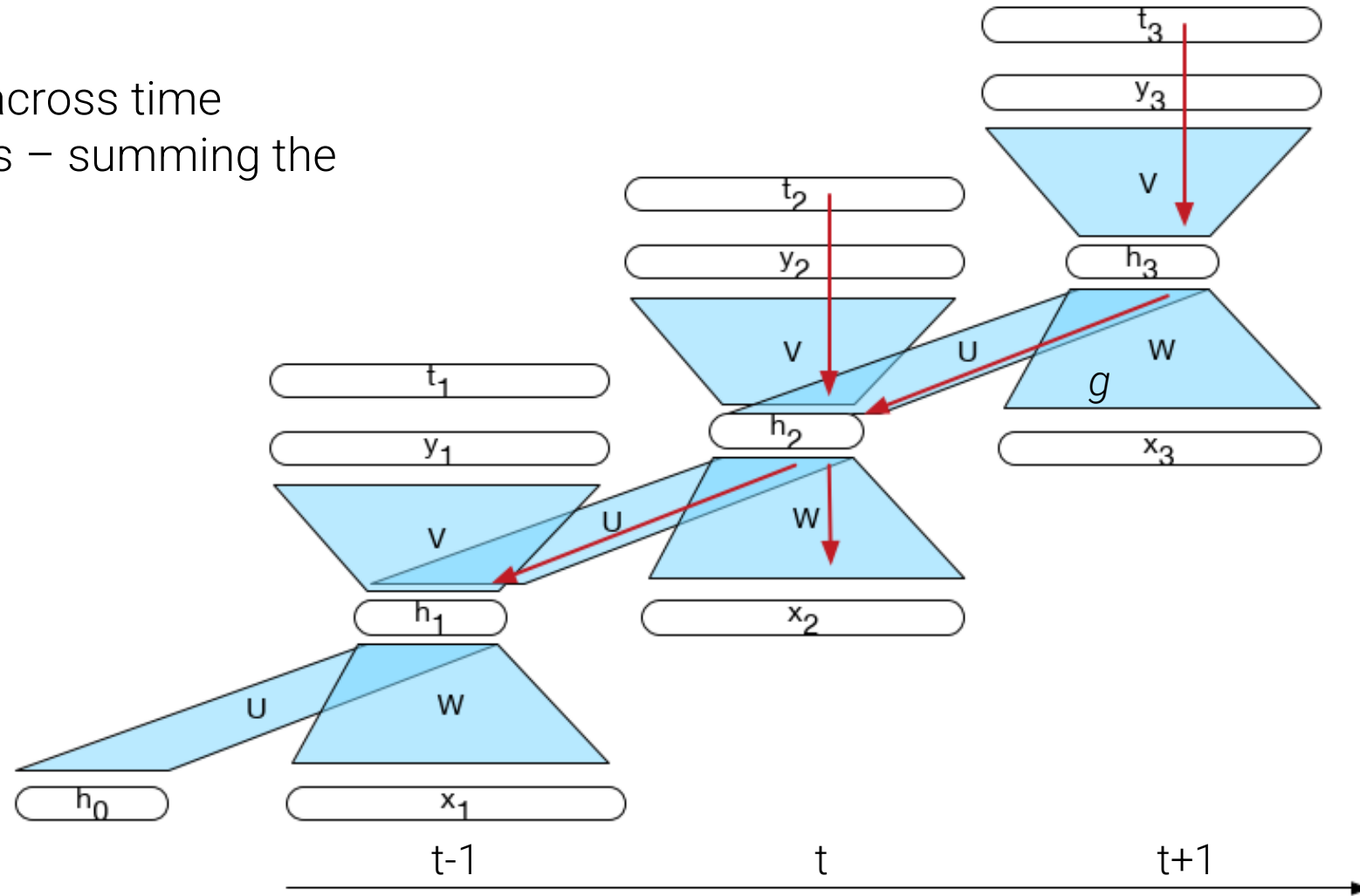
# Recursive Neural Network – Unrolled



$U, W, V$  – shared across time!

# Recursive Neural Network – Unrolled

backpropagation – across time  
two incoming arrows – summing the  
values / errors



# RNN – Applications

- RNN Language Models
  - (Autoregressive) generation
- Sequence labelling
- Sequence classification
- ...



# RNN – Language Models

- N-gram and FF models
  - Fixed sliding window, i.e. fixed context.

$$P(w_n | w_1^{n-1})$$

- Quality of prediction largely dependent on the size of the window
- Constrained by the Markov assumption

$$P(w_n | w_1^{n-1}) \approx P(w_n | w_{n-N+1}^{n-1})$$

- Limitation is avoided in RNN!

# RNN – Language Models

- Limitation is avoided in RNN!

$$\begin{aligned} P(w_n | w_1^{n-1}) &= y_n \\ &= \text{softmax}(V h_n) \end{aligned}$$

$$\begin{aligned} P(w_1^n) &= \prod_{k=1}^n P(w_k | w_1^{k-1}) \\ &= \prod_{k=1}^n y_k \end{aligned}$$

At each step:

- get embedding for  $w_n$
  - combine with previous steps (hidden layer)
  - pass through *softmax* (probability distribution over all vocabulary)
- 
- Probability of the complete sequence is product of probabilities

# RNN – Language Models

- Limitation is avoided in RNN!

$$\begin{aligned} P(w_n | w_1^{n-1}) &= y_n \\ &= \text{softmax}(Vh_n) \end{aligned}$$

$$\begin{aligned} P(w_1^n) &= \prod_{k=1}^n P(w_k | w_1^{k-1}) \\ &= \prod_{k=1}^n y_k \end{aligned}$$

- Cross-entropy function for training

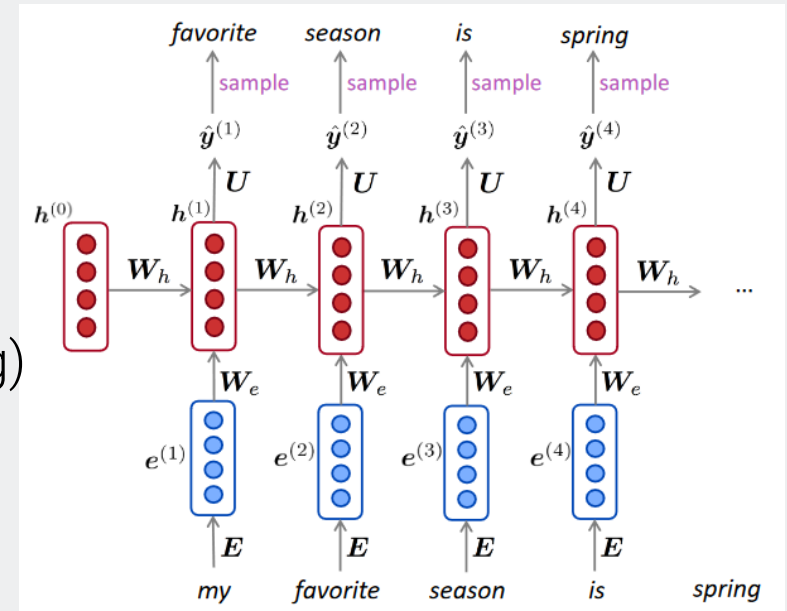
$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\log \hat{y}_i \\ &= -\log \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \end{aligned}$$

- Perplexity for evaluation

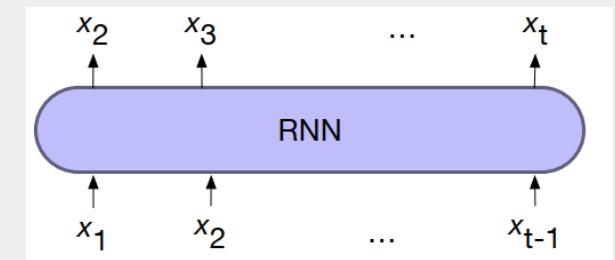
$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

# RNN – Language Models

- Generate text by repeated sampling (during training)
- RNN-LM trained on Obama speeches

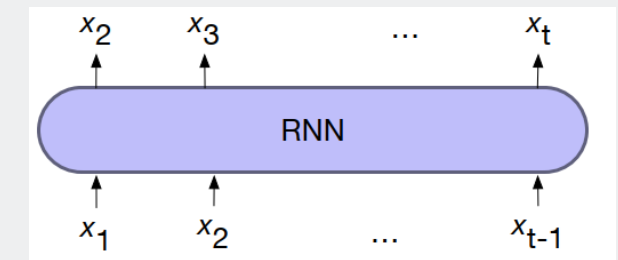
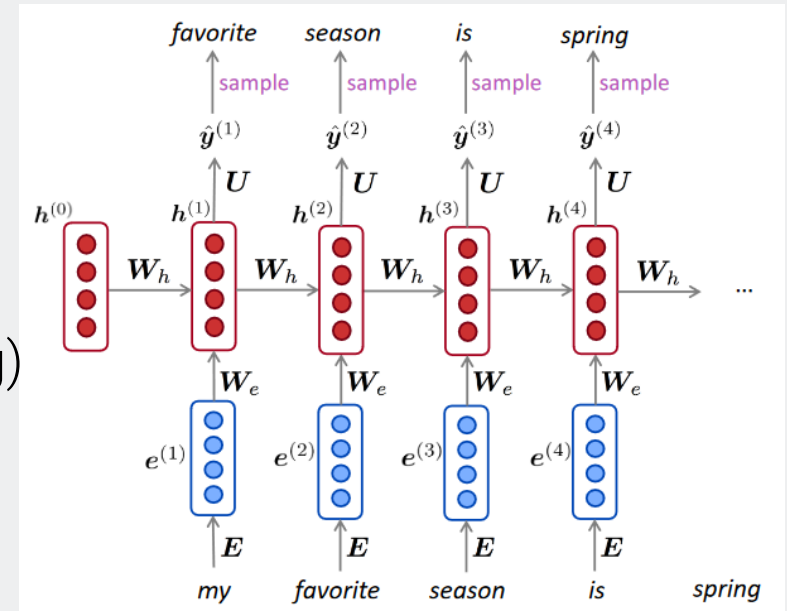



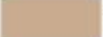





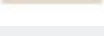
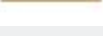
*The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.*



# RNN – Language Models

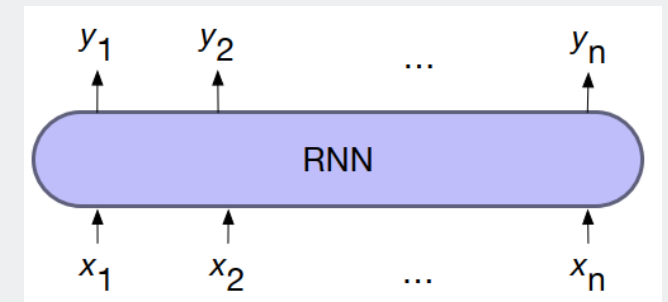
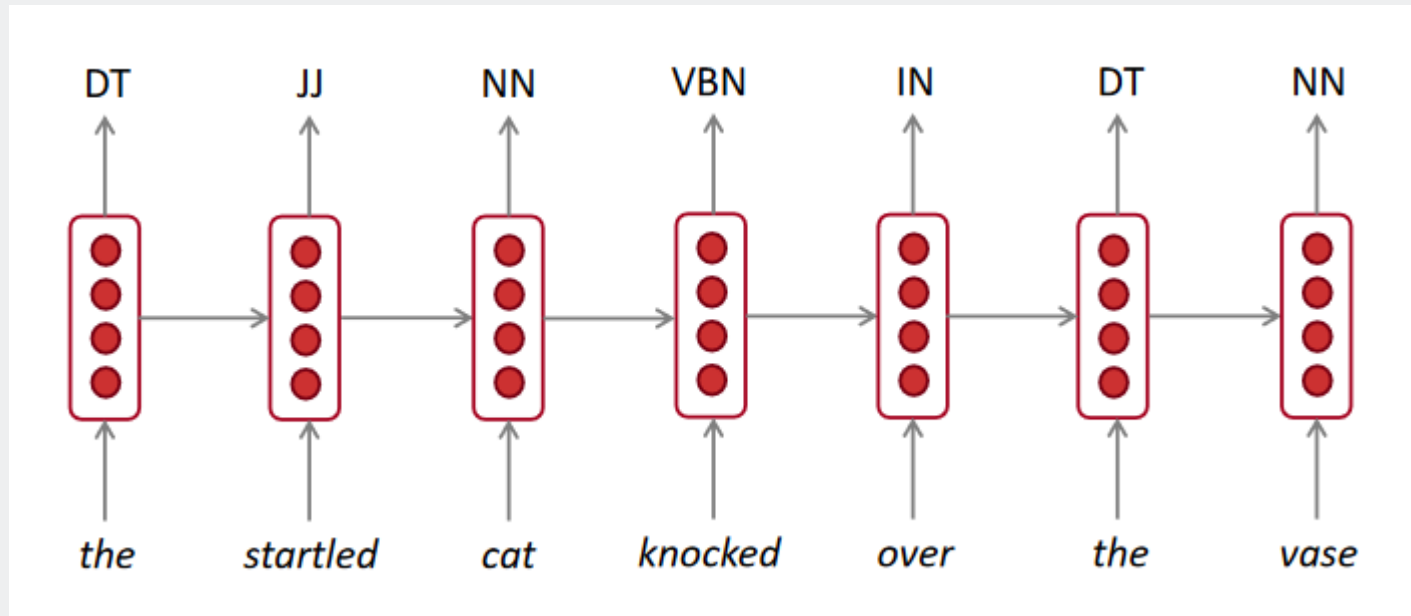
- Generate text by repeated sampling (during training)
  - On any kind of text!
  - Character level example



 Ghasty Pink 231 137 165	 Sand Dan 201 172 143
 Power Gray 151 124 112	 Grade Bat 48 94 83
 Navel Tan 199 173 140	 Light Of Blast 175 150 147
 Bock Coe White 221 215 236	 Grass Bat 176 99 108
 Horble Gray 178 181 196	 Sindis Poop 204 205 194
 Homestar Brown 133 104 85	 Dope 219 209 179
 Snader Brown 144 106 74	 Testing 156 101 106
 Golder Craam 237 217 177	 Stoner Blue 152 165 159
 Hurky White 232 223 215	 Burble Simp 226 181 132
 Burf Pink 223 173 179	 Stanky Bean 197 162 171
 Rose Hork 230 215 198	 Turdly 190 164 116

# RNN – Applications

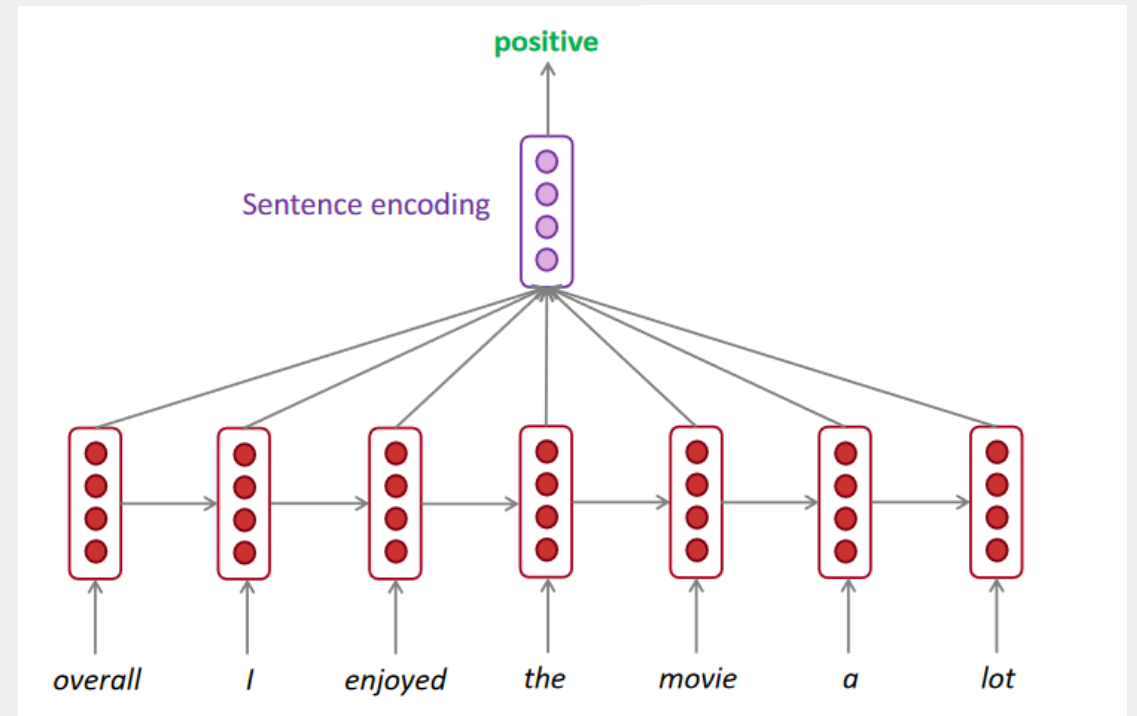
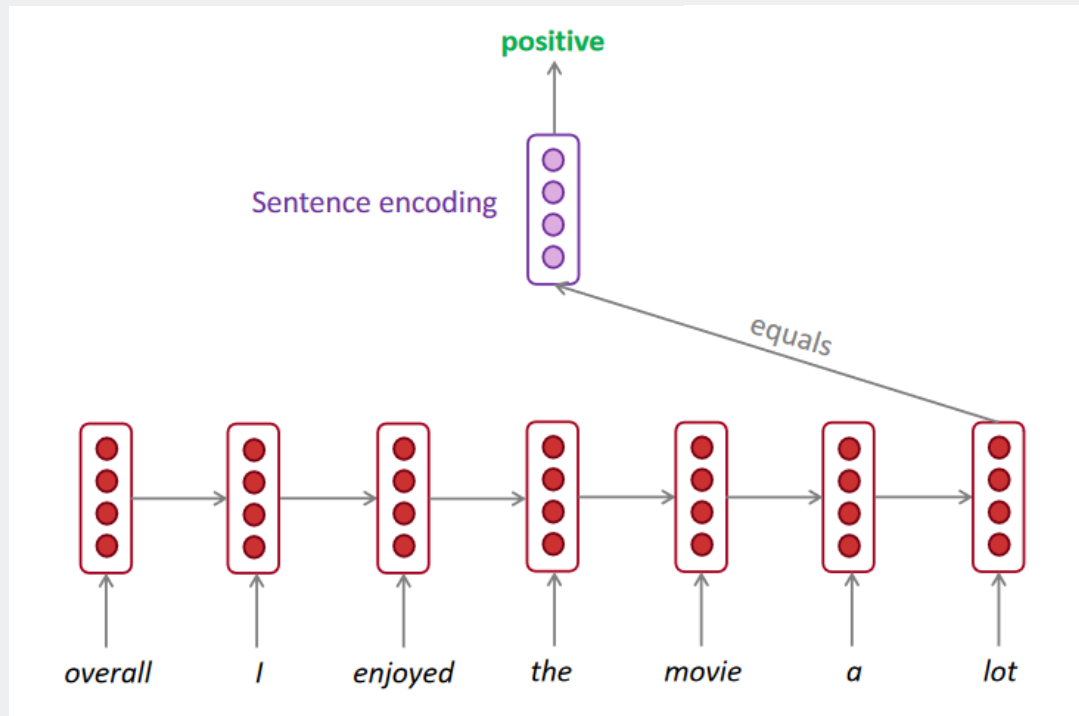
- Tagging (POS, named entity recognition, IOB encoding etc.)
- (sequence labelling)



Jurafsky & Martin, SLP, 3rd Edition: *Chapter 8*

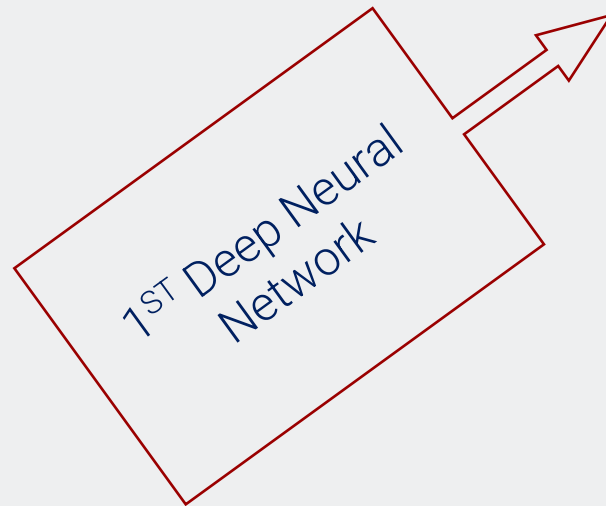
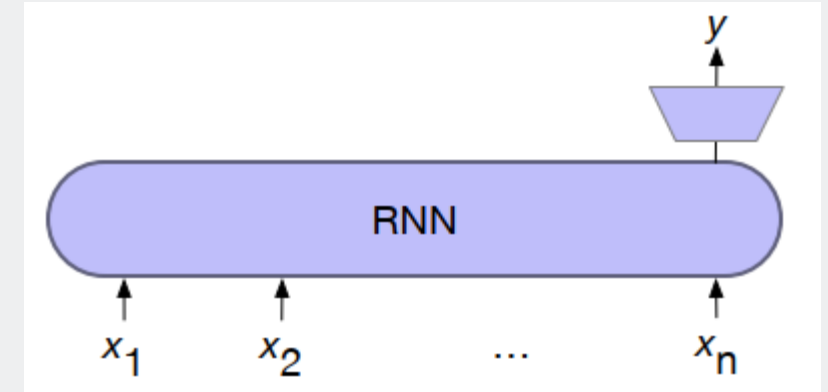
# RNN – Applications

- Sentence (sequence) Classification



# RNN – Deep Networks: Stacked and Bidirectional

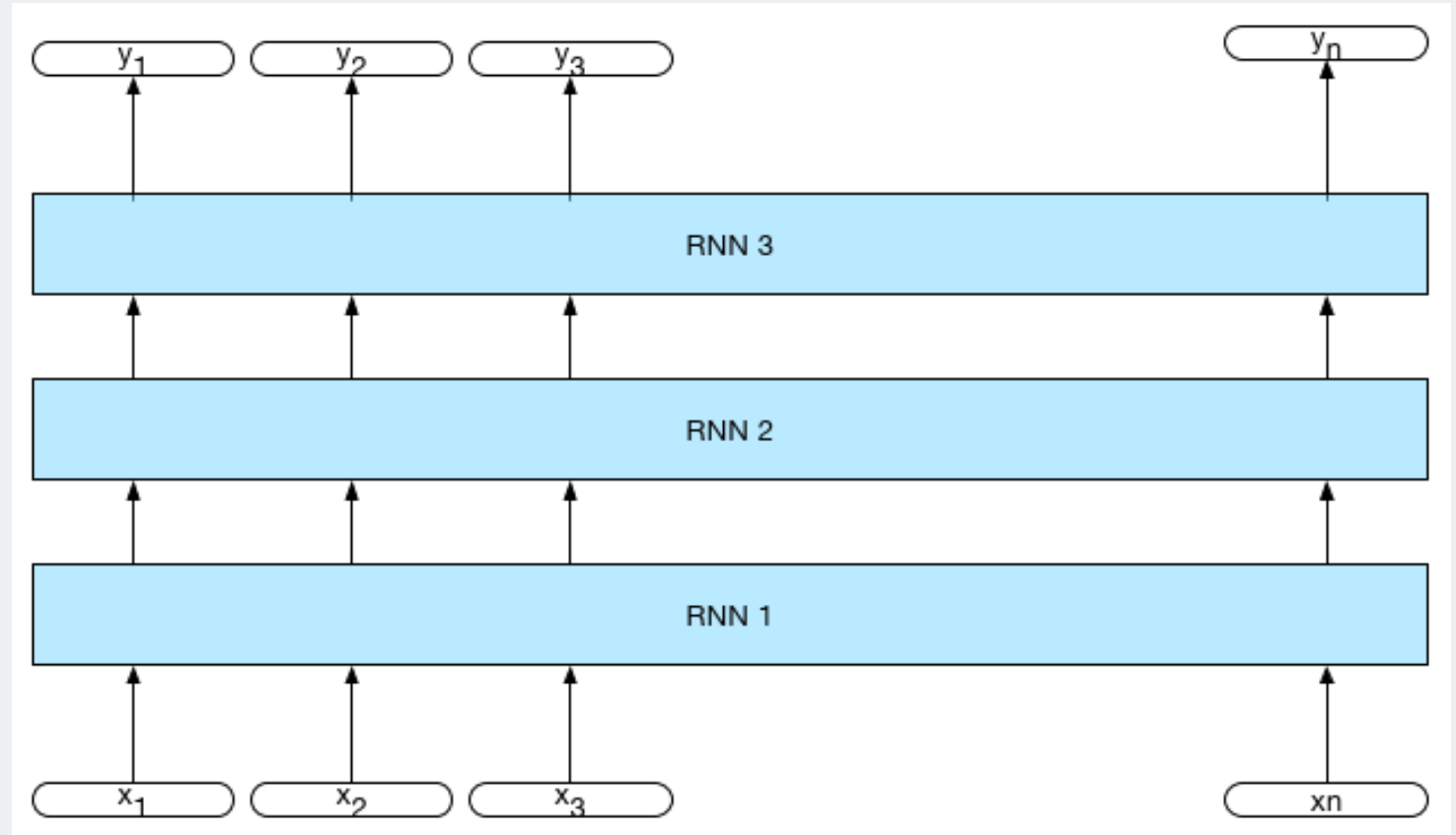
- Sequence Classification
  - Usually RNN combined with a FF
- end-to-end training





# RNN – Deep Networks: Stacked

- Stacked
- Outperform single-layer
- Induce representations
- High training costs



# RNN – Deep Networks: Bidirectional

- $RNN_{\text{forward}}$

$$h_t^f = RNN_{\text{forward}}(x_1^t)$$

- We have access to the entire input sequence

- $RNN_{\text{backward}}$

$$h_t^b = RNN_{\text{backward}}(x_t^n)$$

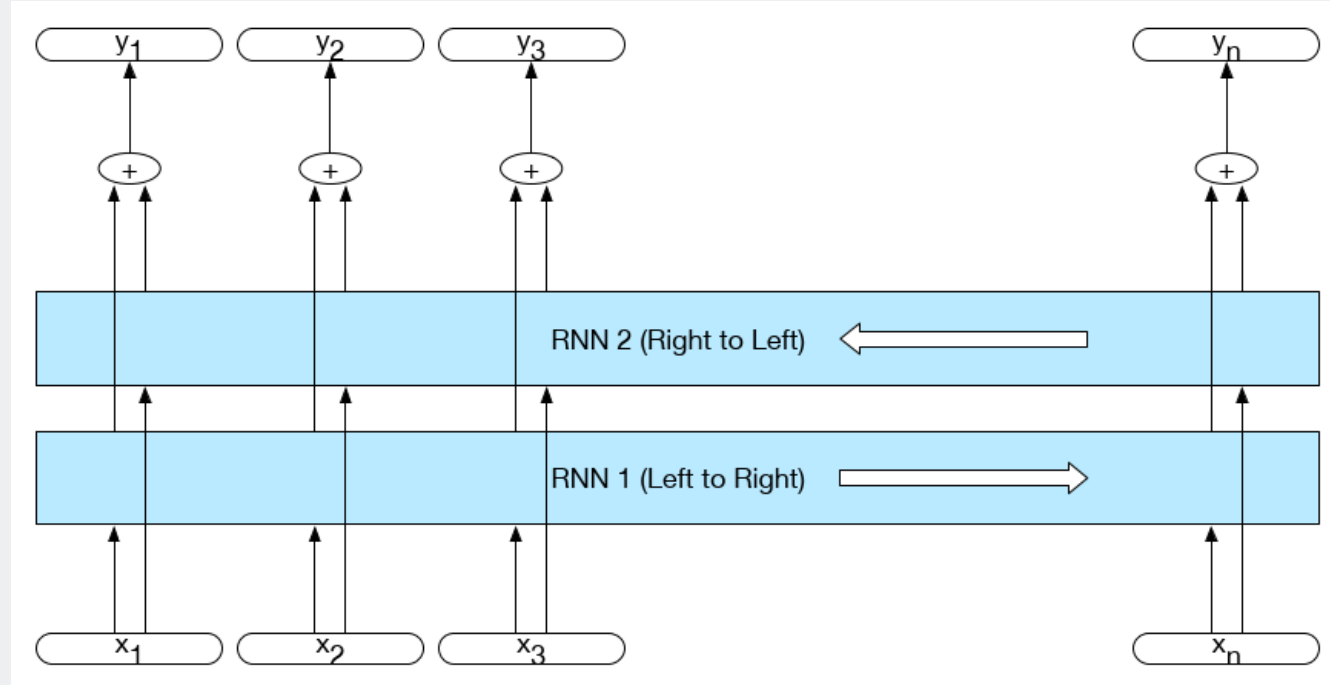
- Combine them -> Bi-RNN

$$h_t = h_t^f \oplus h_t^b$$

# RNN – Deep Networks: Bidirectional

- Bi-RNN combines

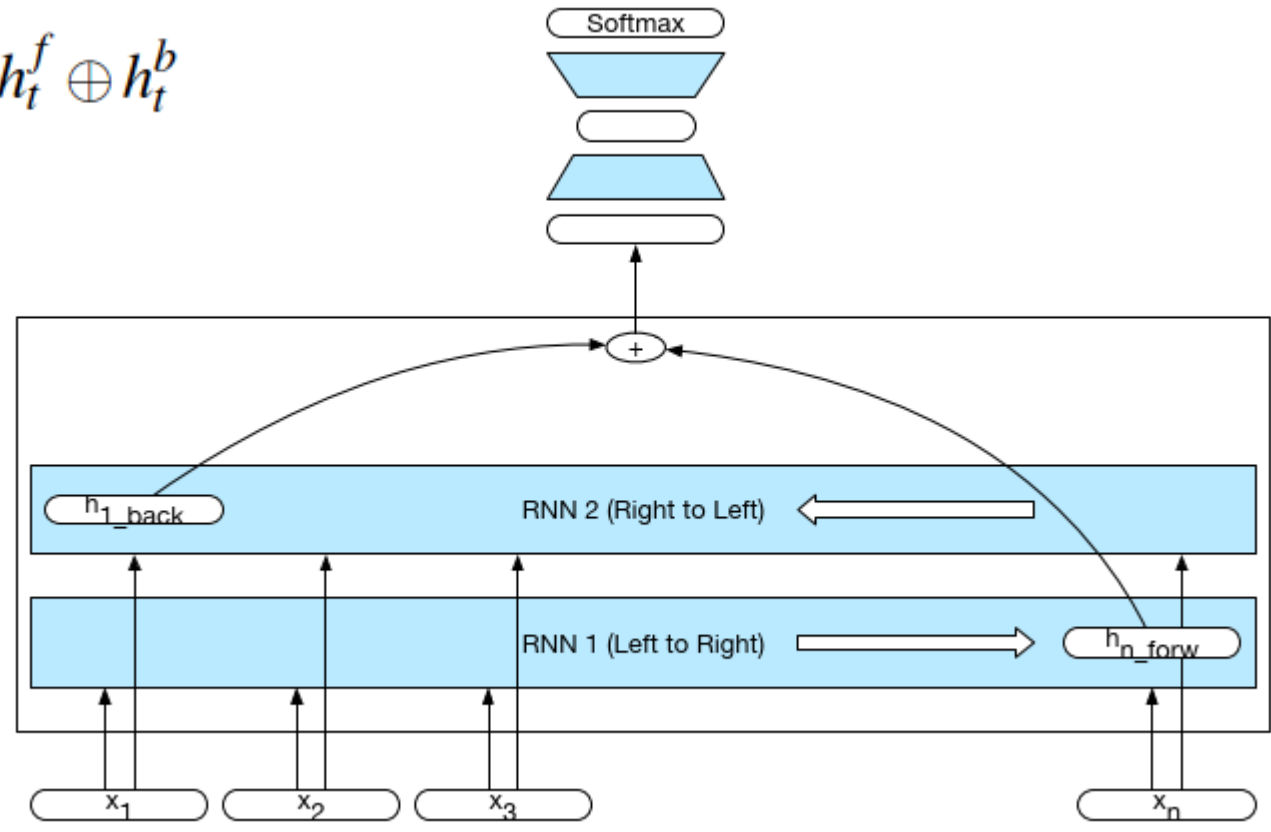
$$h_t = h_t^f \oplus h_t^b$$



# RNN – Deep Networks: Bidirectional

- Bi-RNN combines
- Sequence classification

$$h_t = h_t^f \oplus h_t^b$$



# Content

- Neural Language Models
- Recurrent Neural Networks
- LSTMs (Long Short-Term Memory Networks)

## Quick Recap

- Simple RNNs process sequences naturally one element at a time
- Neural unit output at time  $t$  is based both on the current input and value of the hidden layer from the previous  $t-s$
- RNNs trained backpropagation through time (BPTT – extension of the usual BP)
- Common language-based applications include:
  - Probabilistic language modelling (assigns a probabilities to sequences or to the next element of a sequence)
  - Auto-regressive generation using a trained language model.
  - Sequence labelling
  - Sequence classification (e.g. spam detection, sentiment analysis).

# Long Short-Term Memory Networks

# RNN Shortcomings

- Cannot use information distant from the current time
- Information encoded in the current hidden layer is local

The flights the airline was cancelling were full.

- Hidden layers and weights:
  - useful information for *current* decision
  - Update information for *future* decisions
- Vanishing gradients

$P(\text{was} \mid \text{airline})$  – OK  
 $P(\text{were} \mid \text{flights})$  – ?

# RNN Shortcomings

- How to maintain relevant context over time?

The flights the airline was cancelling were full.

- Learn to forget
- Learn what to keep

$P(\text{was} \mid \text{airline})$  – OK

$P(\text{were} \mid \text{flights})$  – ?

Using RNN we would not be able to understand the second one because the words are quite far and the memory is not as strong

Our aim is to learn a network that learns also what to forget and what to keep



# Recursive Function for Natural Language Understanding

- ADD1 is hardcoded
- Parametrized recursive function
- Memory:  $\mathbf{h} \in \mathbb{R}^{d_h}$
- Input  $x_t$  and memory  $\mathbf{h}$ , returns the new  $\mathbf{h}$
- Time index!

$$h_t = f(x_t, \mathbf{h}_{t-1})$$

Remember this?

$$f(x_t, \mathbf{h}_{t-1}) = g(\mathbf{W}\phi(x_t) + \mathbf{U}\mathbf{h}_{t-1})$$

---

**Algorithm 1** A function ADD1

---

```
 $s \leftarrow 0$   
function ADD1( $v, s$ )  
  if  $v = 0$  then return  $s$   
  else return  $s + 1$   
  end if  
end function
```

---

---

**Algorithm 2** A function ADD1

---

```
 $s \leftarrow 0$   
for  $i \leftarrow 1, 2, \dots, l$  do  $s \leftarrow \text{ADD1}(x_i, s)$   
end for
```

---

# Long Short-Term Memory Networks (LSTMs)

- Memory (aka. context):  $\mathbf{h} \in \mathbb{R}^{d_h}$
- Want: divide context management into:
  - Forgetting (old/unnecessary information)
  - memorizing (new information/context)
- If possible without hard-coding into the architecture!
- Solution:
  - add an explicit context layer
  - gates to control the forgetting/memorizing

# Long Short

gates

context  $\approx$  memory

$U$  is  $W$  and  $W$  is  $U$

We have a sequence of inputs  $x^{(t)}$ , and we will compute a sequence of hidden states  $h^{(t)}$  and cell states  $c^{(t)}$ . On timestep  $t$ :

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

**New cell content:** this is the new content to be written to the cell

**Cell state:** erase ("forget") some content from last cell state, and write ("input") some new cell content

**Hidden state:** read ("output") some content from the cell

**Sigmoid function:** all gate values are between 0 and 1

$$f^{(t)} = \sigma \left( W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left( W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left( W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

$$\tilde{c}^{(t)} = \tanh \left( W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

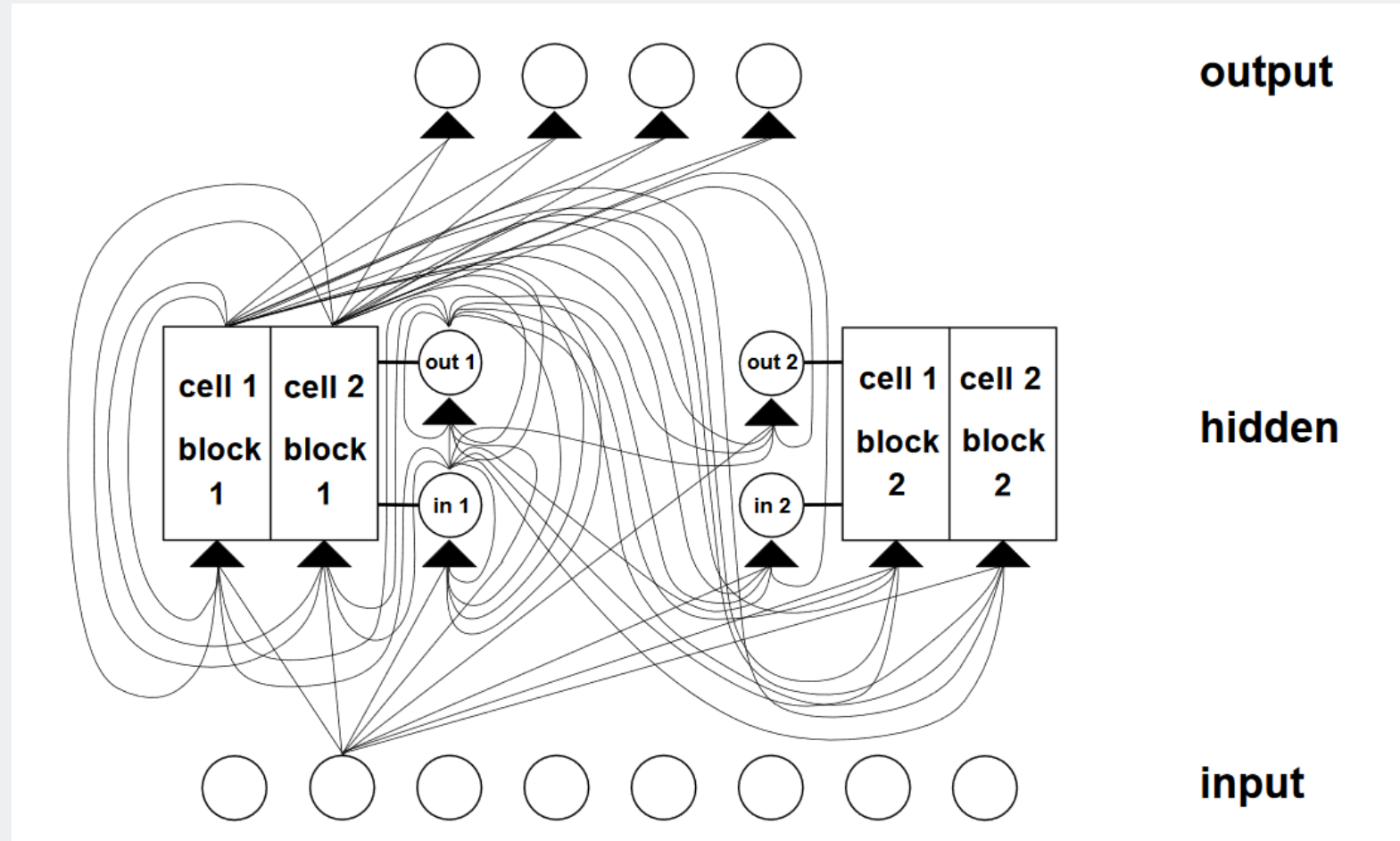
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

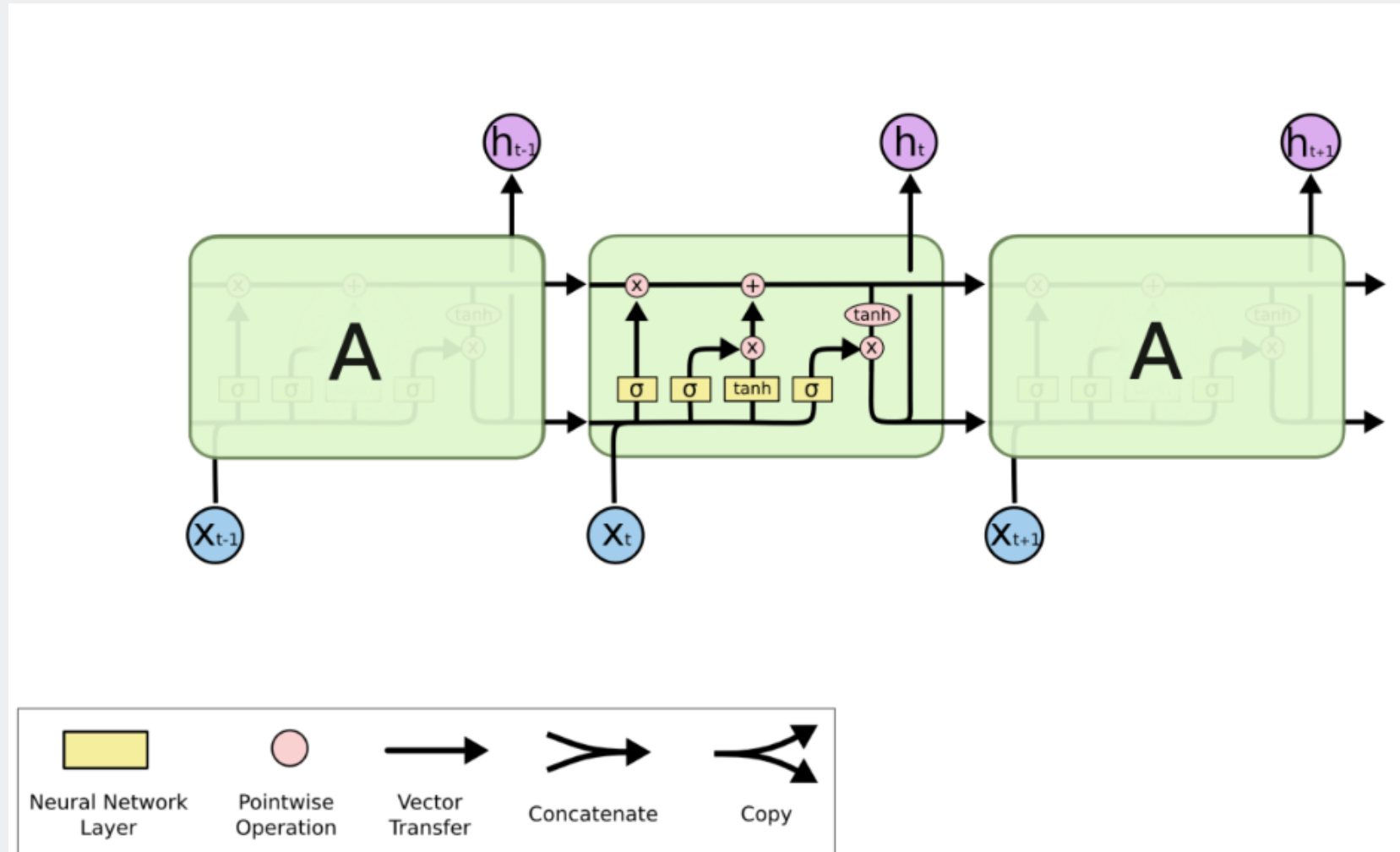
Gates are applied using element-wise product

All these are vectors of same length  $n$

# Long Short-Term Memory Networks (LSTMs)



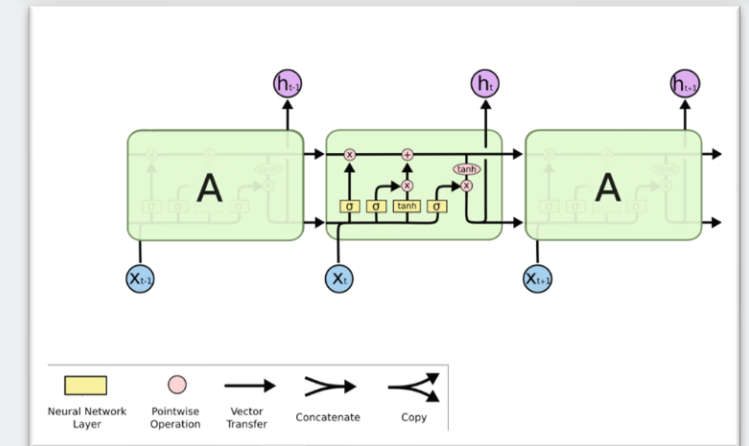
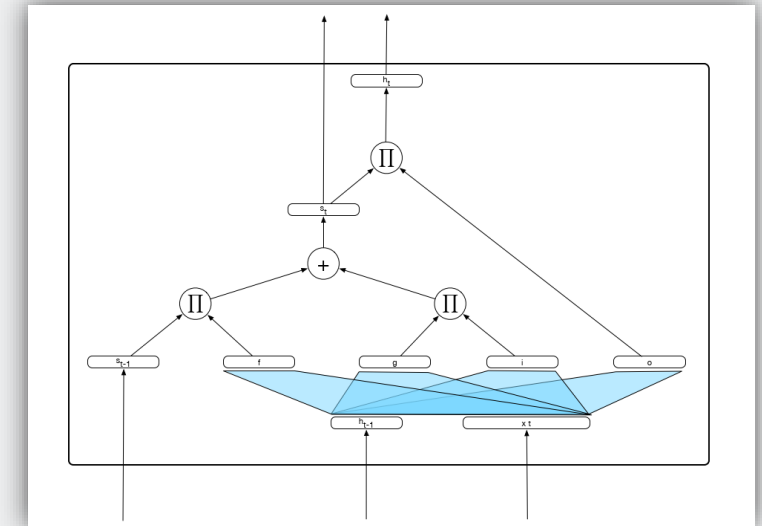
# Long Short-Term Memory Networks (LSTMs)



# LSTMs – recap

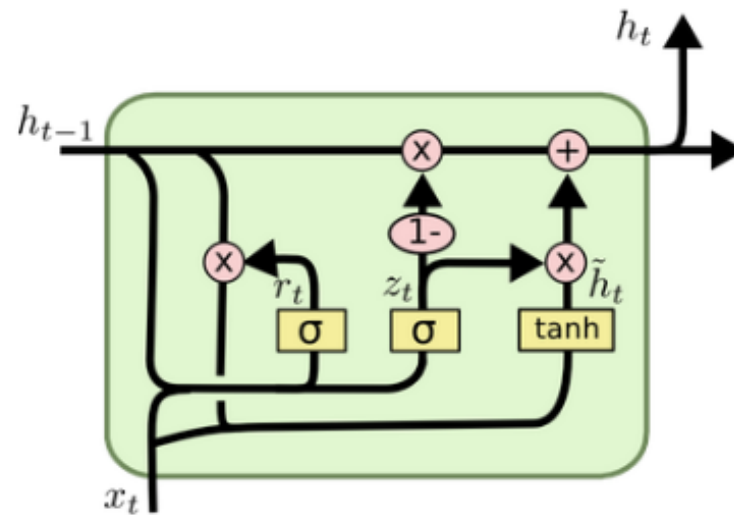
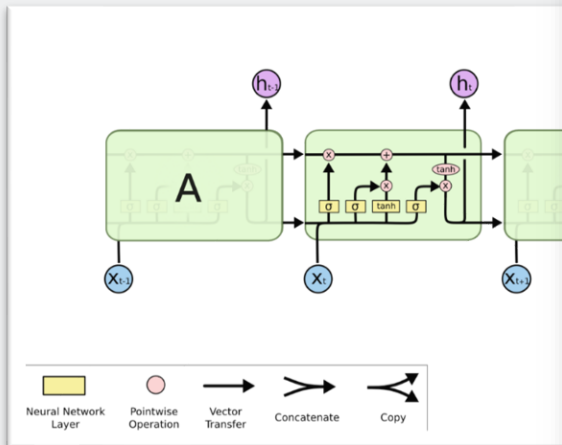
At each step:

- Hidden state  $h$
- Cell state  $c$
- Gates to control the cell state  $c$  (read, write, erase)
  - Forgetting (unnecessary info)
  - Memorizing (new information)
  - Dynamic! (we didn't hard code them)
- But lots of new parameters  $\rightarrow$  higher training costs
- Learning 8 weight matrixes!



# Gated Recurrent Unit

- Uses only two gates: “reset”,  $r$ , and “update”,  $z$
- Collapse “forget” and “input” gates into the “update” gate  $z$
- (less training effort)



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

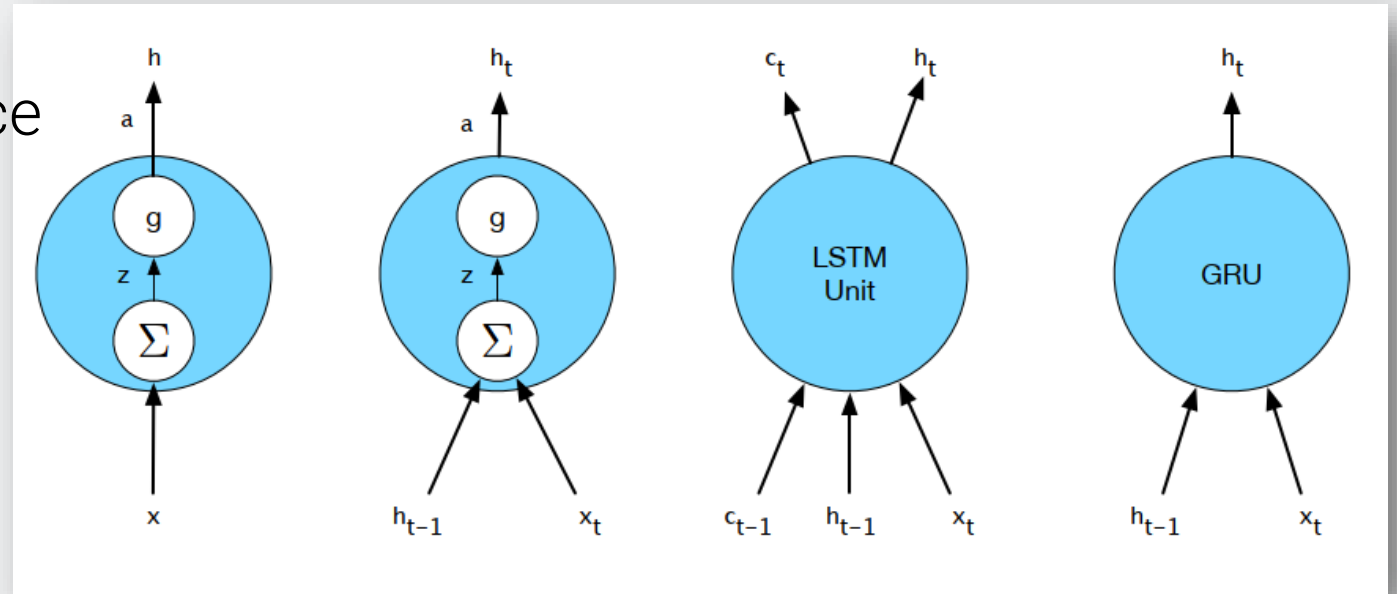
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Neural Units

- Complexity encapsulated in basic processing units
- Easy modularity maintenance
- “wild” architectures easy to understand



Unrolling!

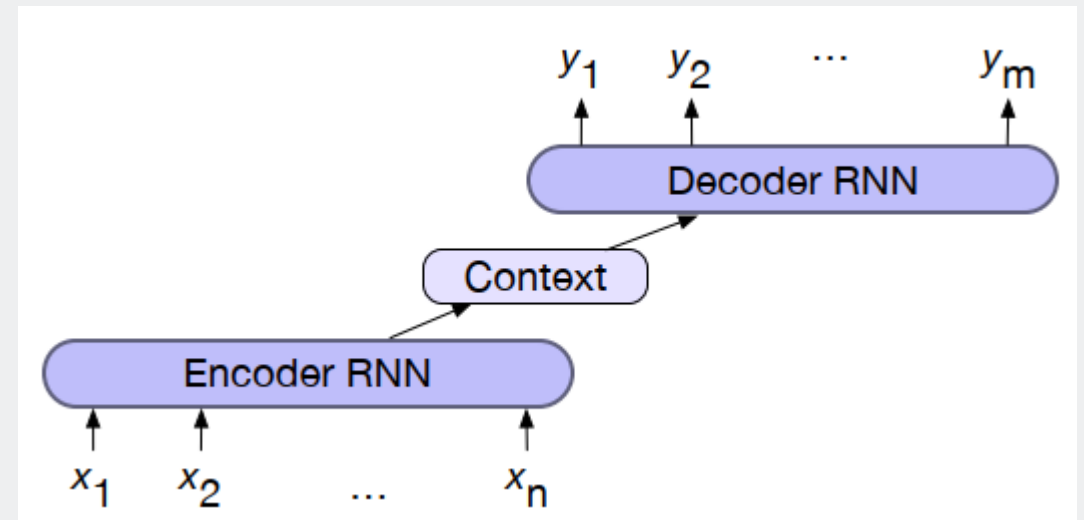


# Content

- Neural Language Models
- Recurrent Neural Networks
- LSTMs (Long Short-Term Memory Networks)
- **Encoder-Decoder**
- **Attention**
- **Very active research area – not all details are included**

# Machine Translation

(sequence-to-sequence processing)



# Sequence-to-Sequence aka. Encoder-decoder Models

- Neural Machine Translation
- **Source** sentence  $X$  in source language
- **Target** sentence  $Y$  in target language
- Translation: function application:
- More than one correct translation

$$X = (x_1, x_2, \dots, x_{T_x})$$
$$Y = (y_1, y_2, \dots, y_{T_y})$$

$$f : V_x^+ \rightarrow C_{|V_y|-1}^+$$

$$C_k = \left\{ (t_0, \dots, t_k) \in \mathbb{R}^{k+1} \mid \sum_{i=1}^k t_i = 1 \text{ and } t_i \geq 0 \text{ for all } i \right\}$$

$$P(Y|X)$$

# Neural Machine Translation

- Conditional language modelling!

$$X = (x_1, x_2, \dots, x_{T_x})$$

$$Y = (y_1, y_2, \dots, y_{T_y})$$

$$f : V_x^+ \rightarrow C_{|V_y|-1}^+$$

$$P(Y|X) = \underbrace{\prod_{t=1}^{T_y} P(y_t | y_1, \dots, y_{t-1}, \underbrace{X}_{\text{conditional}})}_{\text{language modelling}}$$

$$C_k = \left\{ (t_0, \dots, t_k) \in \mathbb{R}^{k+1} \left| \sum_{i=1}^k t_k = 1 \text{ and } t_i \geq 0 \text{ for all } i \right. \right\}$$

- Use what we learned to compute these!
  - N-grams
  - Embeddings
  - ...

# Neural Machine Translation

- Conditional language modelling!

$$X = (x_1, x_2, \dots, x_{T_x})$$

$$Y = (y_1, y_2, \dots, y_{T_y})$$

$$P(Y|X) = \underbrace{\prod_{t=1}^{T_y} P(y_t | y_1, \dots, y_{t-1}, \underbrace{X}_{\text{conditional}})}_{\text{language modelling}}$$

- Training:
  - Maximizing the log-likelihood cost function for a given training set

$$-\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_y} \log p(y_t^n | y_{<t}^n, X^n)$$
$$\{(X^1, Y^1), (X^2, Y^2), \dots, (X^N, Y^N)\}$$

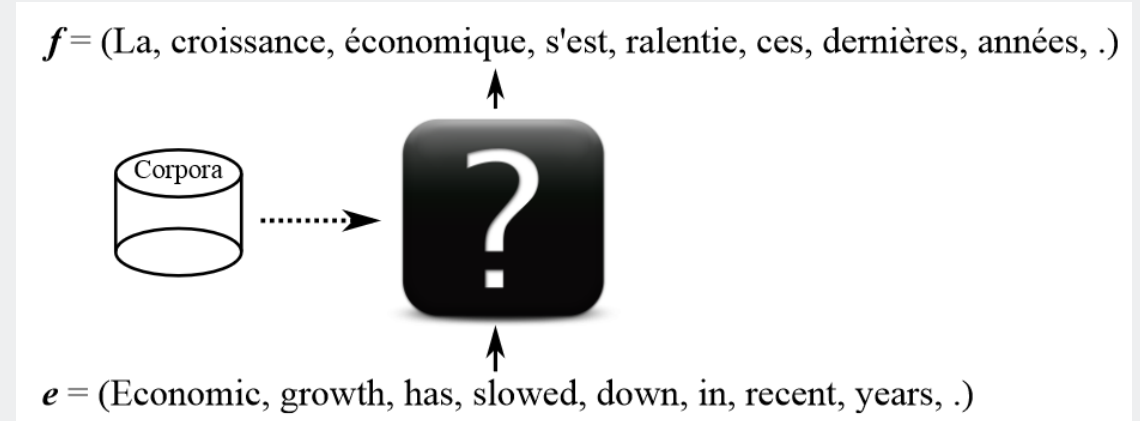
# Neural Machine Translation

- The big picture:

- 1) Assign probabilities to sentences
- 2) Handle variable length sequences (RNNs)
- 3) Train with costs functions & gradient descent

? Training data

? Evaluating MT



# Training Data for Machine Translation

- Sequence-to-sequence
- Sentence pairs (source\_language, target\_language)
- *parallel- corpus*
  - where to get it?
- International news agencies (AFP)
- Books published in multiple languages
- Ebay/Amazon/... (product descriptions)

Copyright issues!

# Training Data for Machine Translation

- Sequence-to-sequence
- Sentence pairs (source\_language, target\_language)
- *parallel- corpus*
  - where to get it?
- proceedings from the Canadian parliament (Brown et al, 1990)
  - French – English, curated (professional translators)
- EU parliament - more than 20 languages



# Training Data for Machine Translation

- translated subtitle of the TED talks, (WIT, <https://wit3.fbk.eu/>)
  - 104 languages
- Russian-English: Yandex (<https://translate.yandex.ru/corpus?lang=en>)
- SWRC English-Korean multilingual corpus: 60,000 sentence pairs
- <https://github.com/jungyeul/korean-parallel-corpora> (~94K sentence pairs)
- Crawl the internet for pairs of pages (but check the small print!)
  - Wikipedia
- Common Crawl Parallel Corpus (Smith et. Al, 2013)
  - <http://www.statmt.org/wmt13/training-parallel-commoncrawl.tgz>

# Evaluating Machine Translation

- There may be many correct translations for one sentence
  - It is a guide to action that ensures that the military will forever heed Party commands.
  - It is the guiding principle which guarantees the military forces always being under the command of the Party.
  - It is the practical guide for the army always to heed the directions of the party.
- Quality is not success or failure

# Evaluating Machine Translation

- Quality is not success or failure:
  - French: “J’aime un llama, qui est un animal mignon qui vit en Amérique du Sud”
  - “I like a llama which is a cute animal living in South America”. 100
  - “I like a llama, a cute animal that lives in South America”. 90
  - “I like a llama from South America”? 50
  - “I do not like a llama which is an animal from South America”?
- We want automated evaluation!

# Evaluating Machine Translation – BLEU score

- geometric mean of the modified N-gram precision scores multiplied by brevity penalty.

- N-gram precision:

$$p_n = \frac{\sum_{S \in C} \sum_{\text{ngram} \in S} \hat{c}(\text{ngram})}{\sum_{S \in C} \sum_{\text{ngram} \in S} c(\text{ngram})}$$

$$\hat{c}(\text{ngram}) = \min(c(\text{ngram}), c_{\text{ref}}(\text{ngram})).$$

- Geometric mean
- But: “cute animal that lives”  $P = 1$
- Brevity Penalty (BP)

$$P_1^4 = \exp \left( \frac{1}{4} \sum_{n=1}^4 \log p_n \right)$$

$$\text{BP} = \begin{cases} 1 & , \text{ if } l \geq r \\ \exp \left( 1 - \frac{r}{l} \right) & , \text{ if } l < r \end{cases}$$

# Evaluating Machine Translation – BLEU score

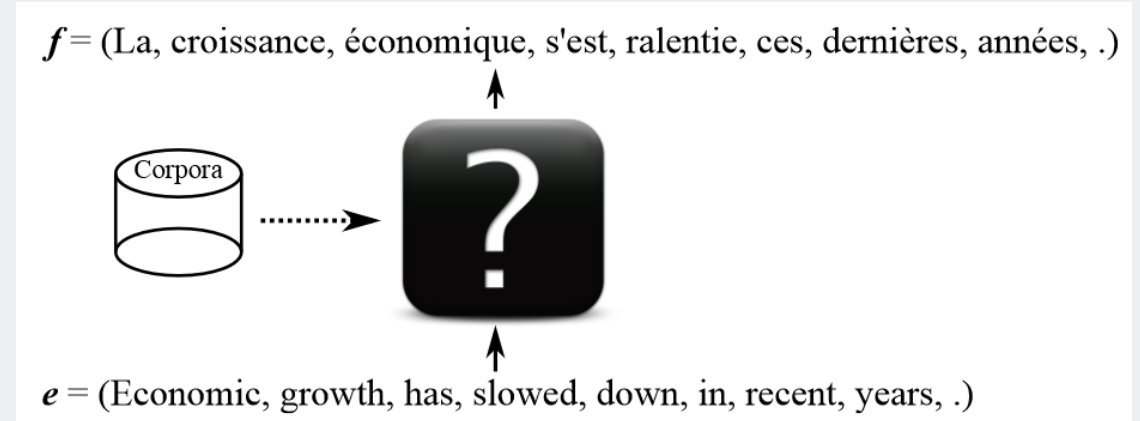
- The BLEU was shown to correlate well with human judgements
- But not perfect automatic evaluation metric
- METEOR (M. Denkowski and A. Lavie, 2014)
- TER (Translation Edit Rate, M. Snover, 2006)

# Neural Machine Translation

- The big picture:

- 1) Assign probabilities to sentences
- 2) Handle variable length sequences (RNNs)
- 3) Train with costs functions & gradient descent

- ✓ Training data
- ✓ Evaluating MT

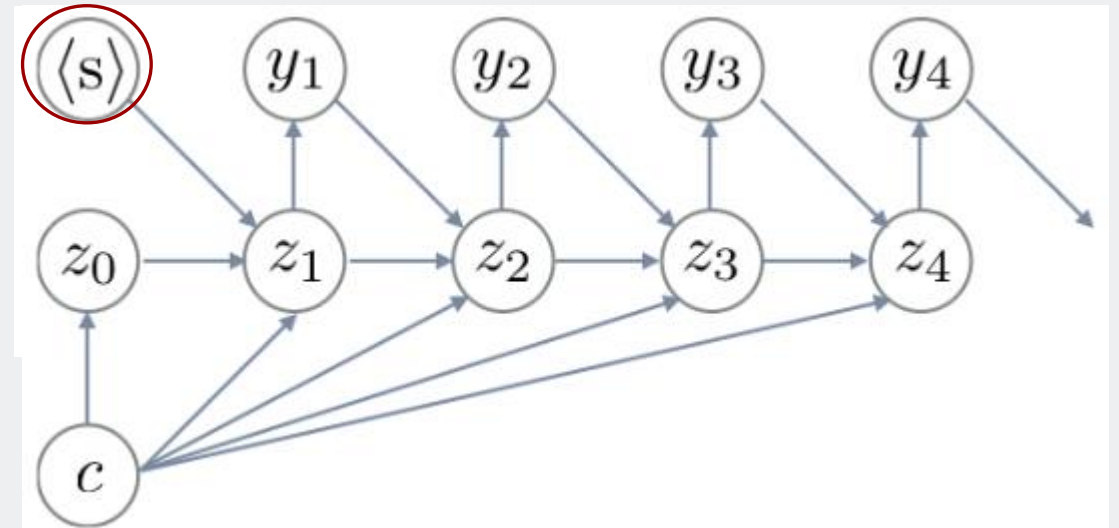
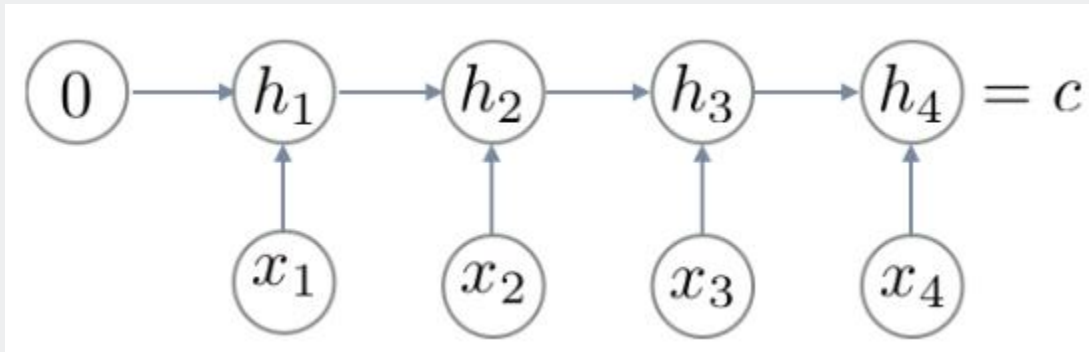


# Neural Machine Translation: Encoder-Decoder Model

- Input:  $Y = (y_1, \dots, y_{t-1})$   $X = (x_1, \dots, x_{T_x})$
- Start with  $X$ , how to handle it?
  - Variable-length sequence (RNN)
  - No explicit output/target  $\rightarrow$  only the summary (vector)
  - RNN  $\sim$  *encoder*

$$P(Y|X) = \prod_{t=1}^{T_y} P(y_t | y_1, \dots, y_{t-1}, \underbrace{X}_{\text{conditional}})$$

language modelling



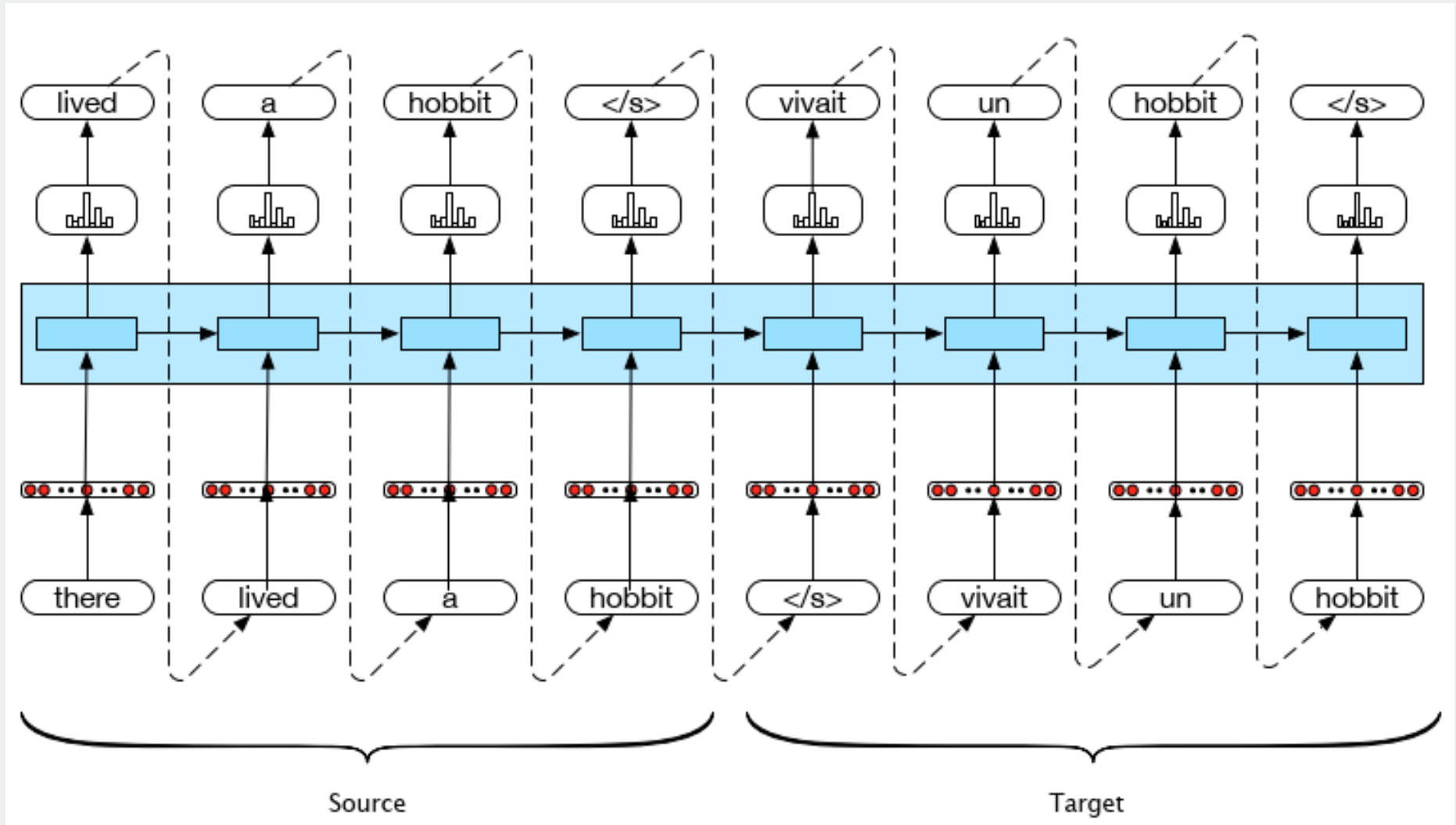
# Neural Machine Translation: Encoder-Decoder Model

- Task: automatically translate from one language to another
- Source language/sentence/sequence
- Target language/sentence/sequence
- Parallel Corpus or **bitexts**
- Language Models & Autoregressive Generation extended to Machine Translation
  - End-of-sentence marker between **bitexts** (source</s>target)
  - Use them as training data (RNN-based LM)
  - Predict next word in the sentence



# Neural Machine Translation: Encoder-Decoder Model

Simple RNN,  
LSTM, GRU, ...

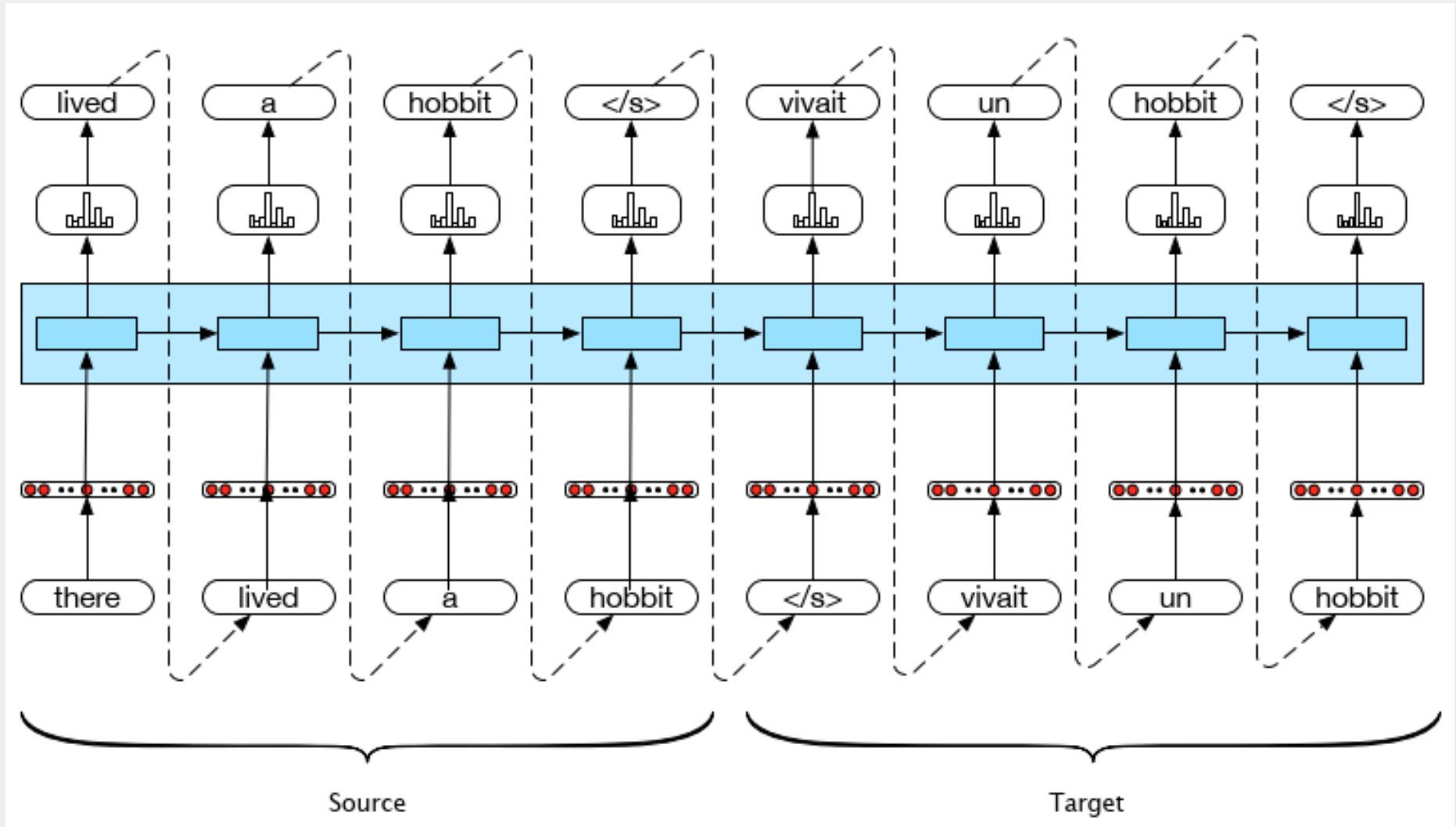


# Encoder-Decoders

(aka. Sequence-to-sequence Models)

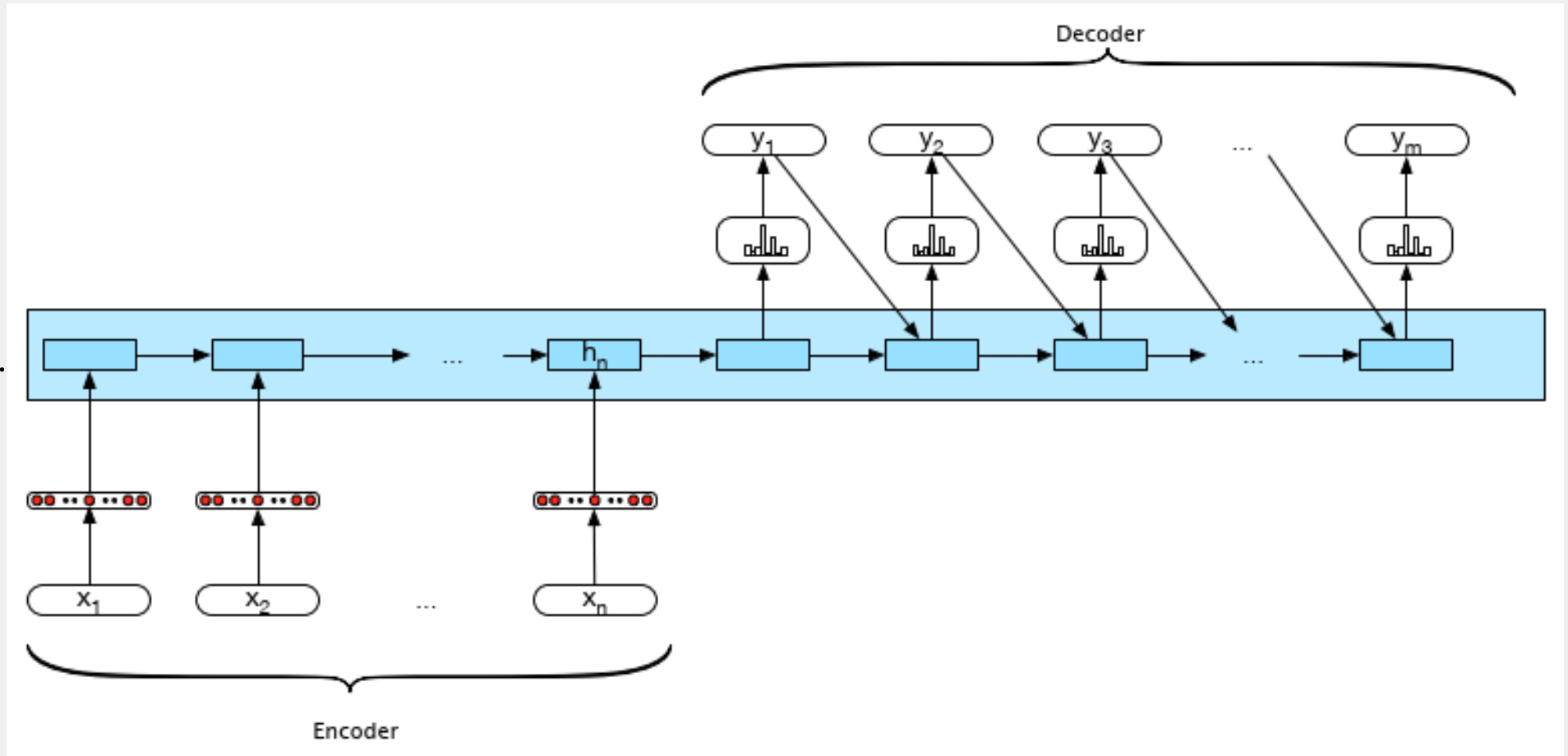
# Neural Machine Translation: Encoder-Decoder Model

Simple RNN,  
LSTM, GRU, ...



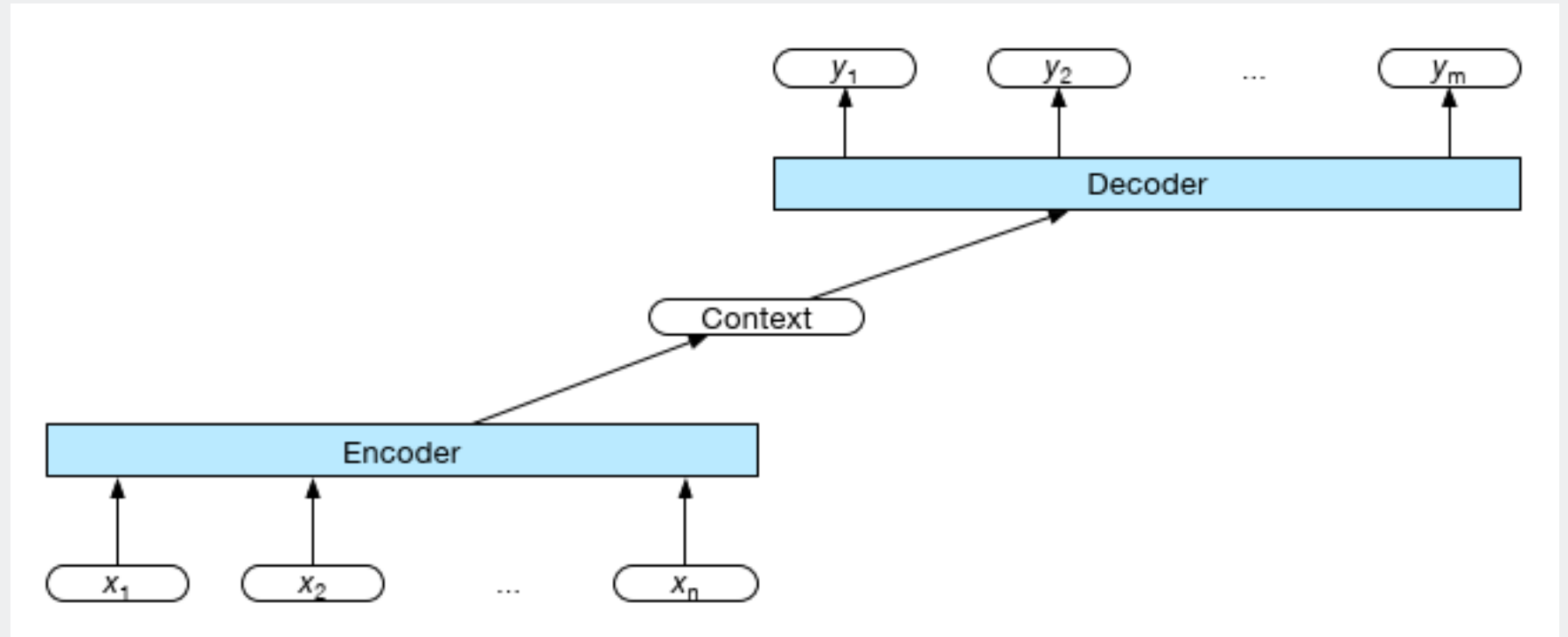
# Neural Machine Translation: Encoder-Decoder Model

Simple RNN,  
LSTM, GRU, ...



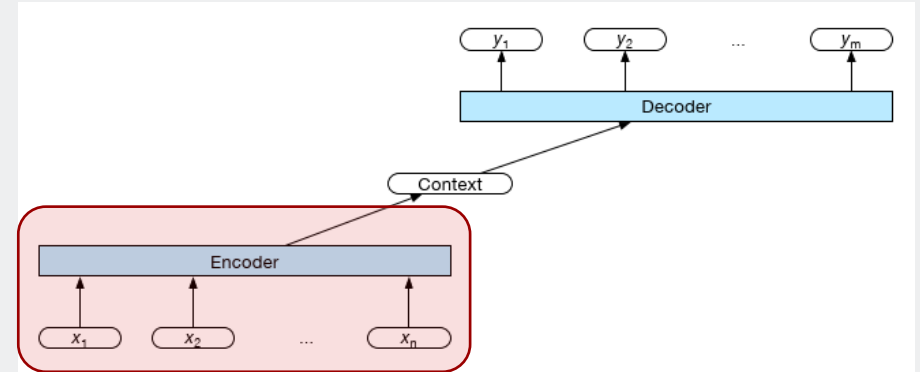
# Neural Machine Translation: Encoder-Decoder Model

- Three main components:
  - Encoder
  - Context vector
  - decoder



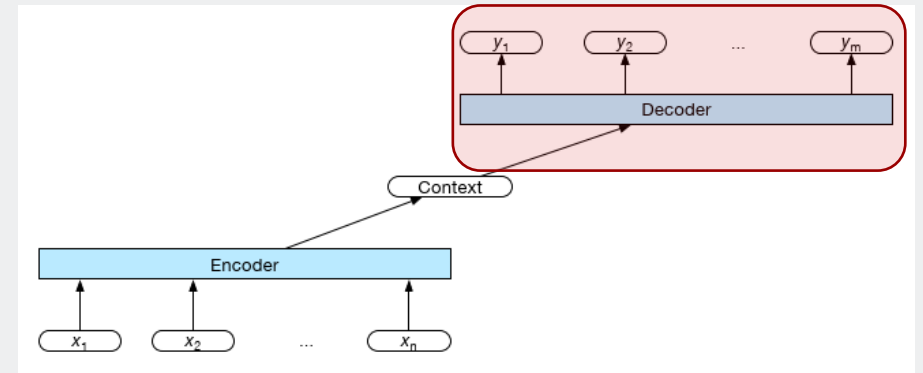
# Encoder

- Simple RNNs, LSTM, GRU
- Stacked
- Bi-LSTMs are the norm



# Decoder

- Autoregressive generation
- Until `</s>` is generated
- LSTM, GRU

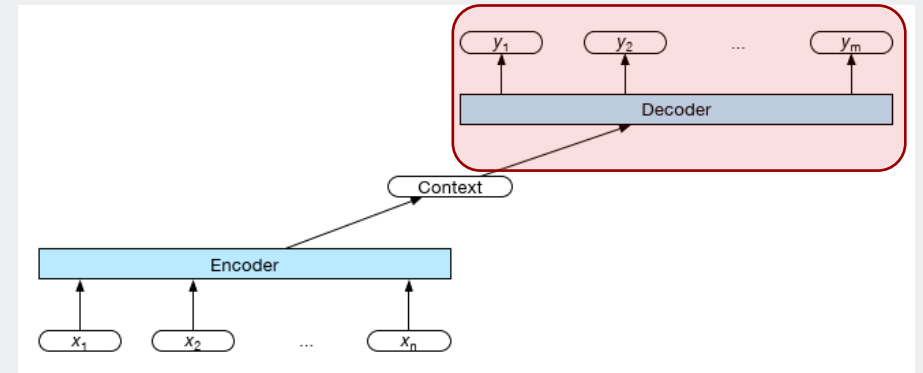


$$c = h_n^e$$
$$h_0^d = c$$

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$z_t = f(h_t^d)$$
$$y_t = \text{softmax}(z_t)$$

# Decoder

- Context available only once.
- How to choose, from the output space the right “next” decoded sequence element?
  - Large search space!
  - Algorithm: Beam Search



$$c = h_n^e$$
$$h_0^d = c$$

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c) \quad h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$

$$z_t = f(h_t^d)$$

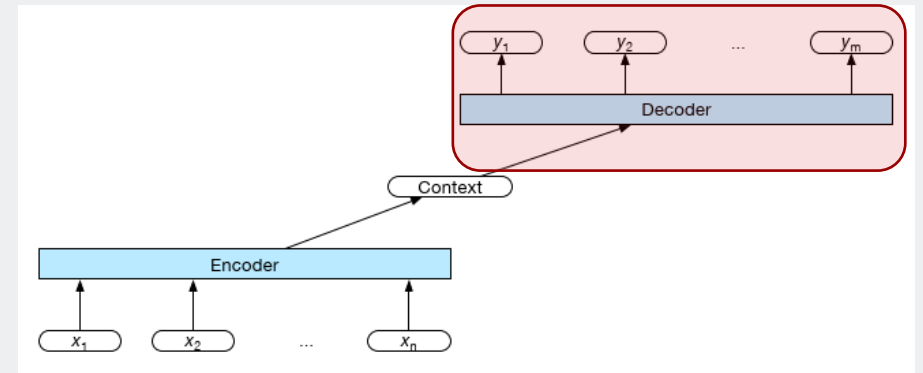
$$y_t = \text{softmax}(\hat{y}_{t-1}, z_t, c) \quad y_t = \text{softmax}(z_t)$$

$$\hat{y} = \text{argmax} P(y_i | y_{< i})$$



# Beam Search

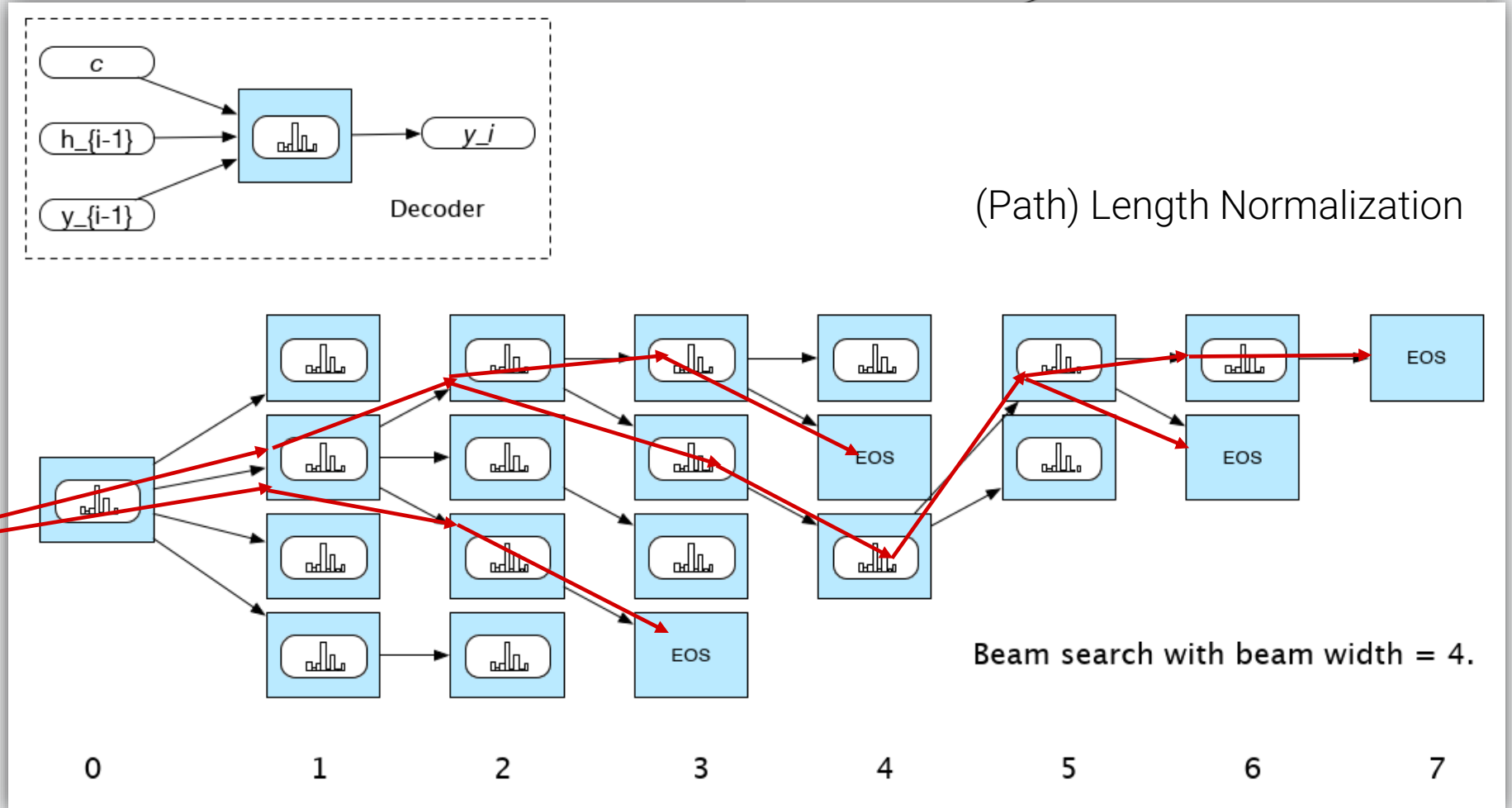
- Large search space
- Alternative: heuristic method, systematic exploration
- By controlling the exponential growth of the search space
- How: combine breadth first with a heuristic filter
  - Score the options
  - Prune the search space



# Beam Search

Scoring:

$$P(y_i | y_{<i})$$

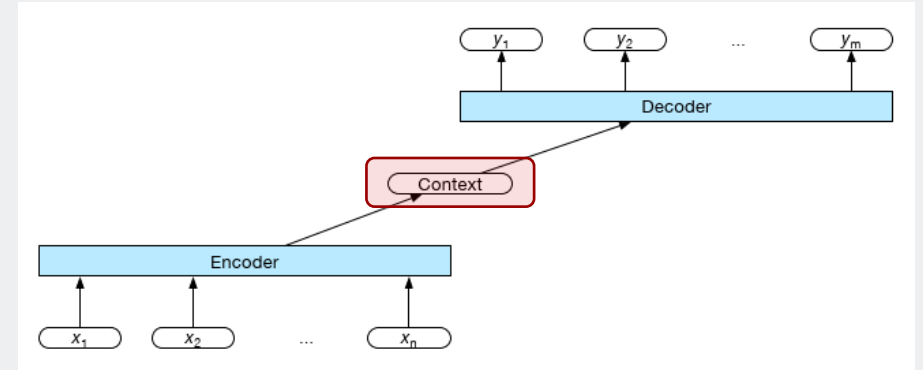


# Context

- Context available only once.
- Function of the hidden encoder states

$$c = f(h_1^n)$$

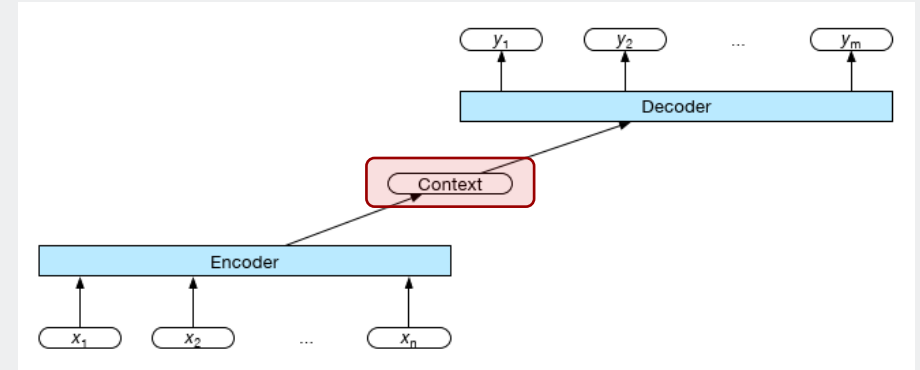
- Variable number of hidden states!
- Bi-RNNs (end states of forward & backward passes, separate or concatenated)
- Average over encoder hidden states



# Attention

# Attention

- Take all encoder context
- Dynamically update during decoding  $\rightarrow c_i$
- Function of the hidden encoder states
- Condition the decoding on the dynamic context
  - Relevance of **encoder** hidden states to the current decoder state
  - Use softmax to normalize these scores
    - Vector of weights



$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

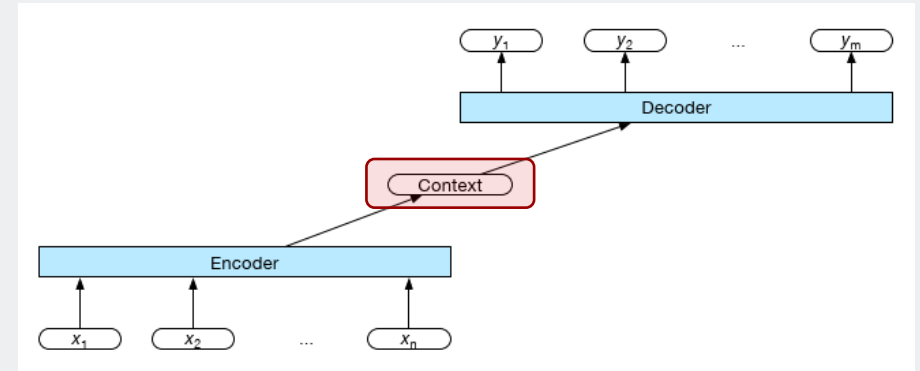
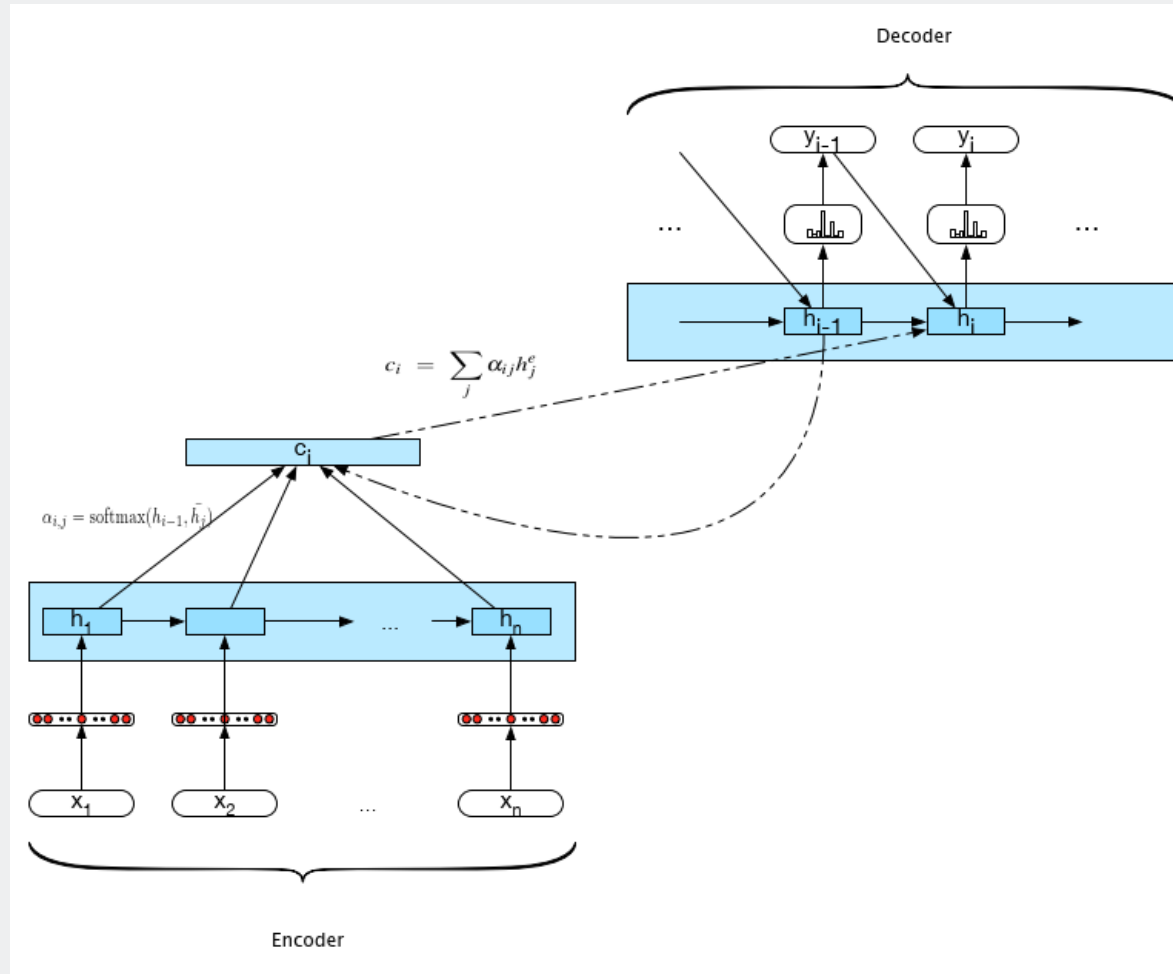
$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d W_s h_j^e$$

$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \quad \forall j \in e)$$

$$c_i = \sum_j \alpha_{ij} h_j^e$$

# Attention



$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d W_s h_j^e$$

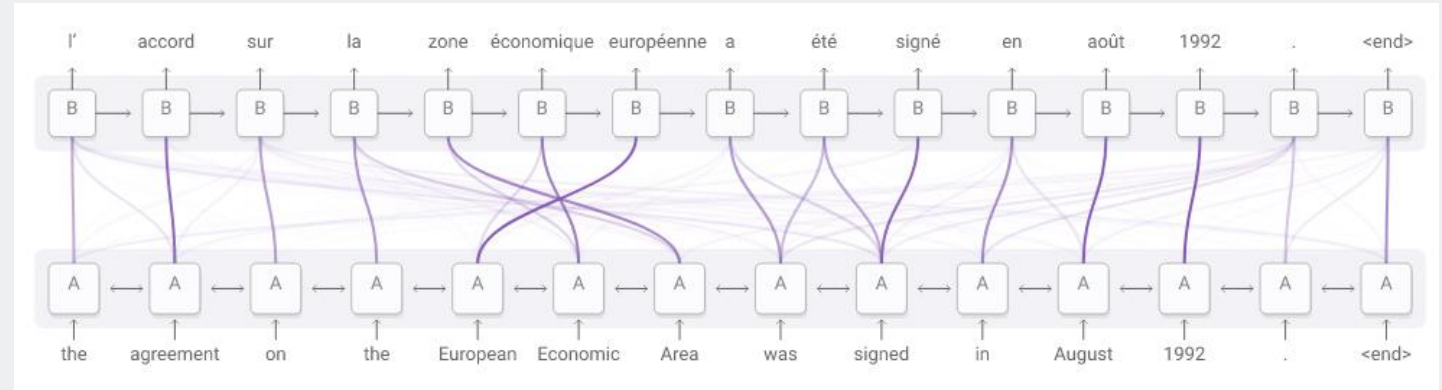
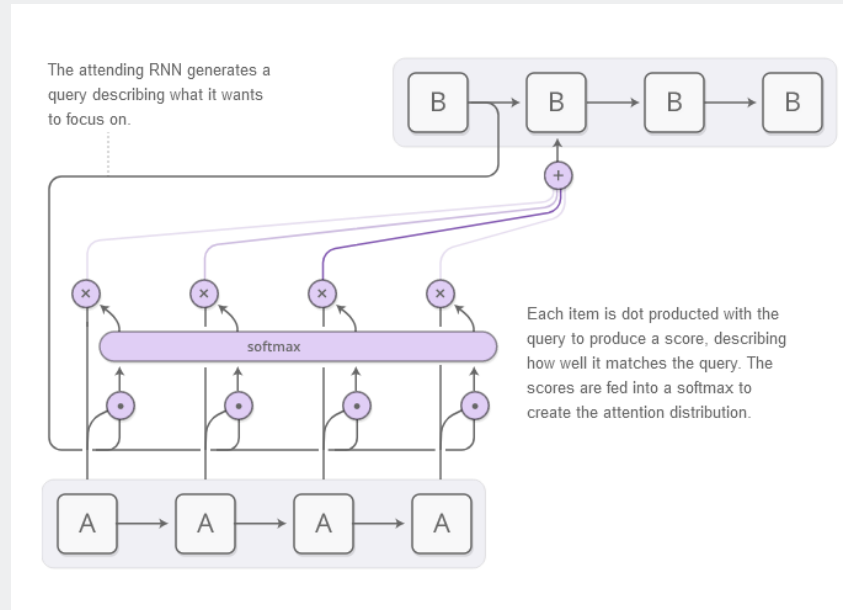
$$\alpha_{ij} = \text{softmax}(\text{score}(h_{i-1}^d, h_j^e) \quad \forall j \in e)$$

$$c_i = \sum_j \alpha_{ij} h_j^e$$

# Attention

<https://distill.pub/2016/augmented-rnns/>

Live tool to observe which words in the input sequence affect which part of the output sequence



# Content

- Sequence-to-sequence (Encoder-Decoder)
- Attention

“Attention is All You Need” <https://arxiv.org/pdf/1706.03762.pdf>  
<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>  
<https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/>  
<https://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/>