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

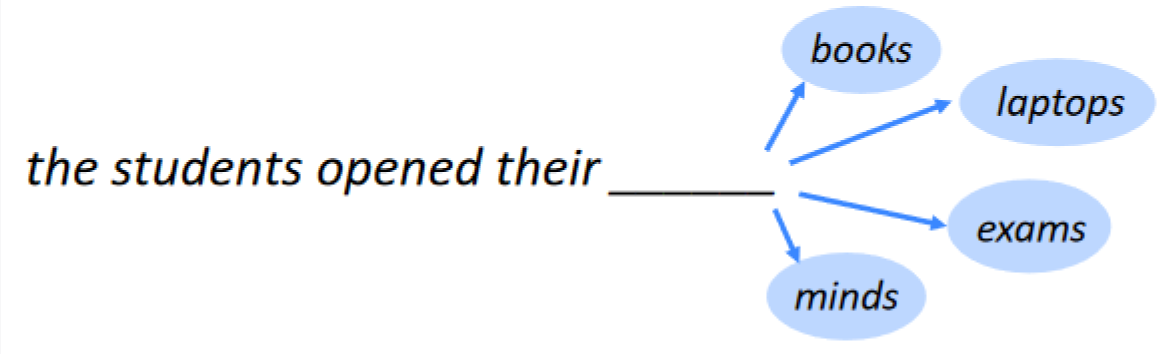
Informatics

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 $P(w_n \mid w_1, w_2 \ldots w_{n-1})$
- Probabilistic Language Models
  - Compare probabilities of sequence of words
  - Probability of upcoming word



What is a “Language Model”?

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

- How did you compute P?
  - Count and divide
  - Markov Assumption

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

$$P(\text{the} \mid \text{its water is so transparent that}) \text{ » } P(\text{the} \mid \text{that})$$

$$P(\text{the} \mid \text{its water is so transparent that}) \text{ » } P(\text{the} \mid \text{transparent that})$$

What is a "Language Model"?

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

- How did you compute P?
  - Count and divide
  - Markov Assumption

- Unigrams
- Bi-grams
- …
- N-grams

Language Model: A simple example

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

<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

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

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

Recall: Language Model

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

Perplexity

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

- How did you compute P?
  - Count and divide
  - Markov Assumption

- Unigrams
- Bi-grams
- …
- N-grams

- Issues: zero probabilities, smoothing, interpolation

Neural Language Model

- No smoothing
- Longer histories (compared to the fixed N in ”N-gram”)
- Generalize over contexts

- 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, …)
- Output: probability distribution over possible next words.

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

Neural Language Model - Input

- Standard Feed-Forward Network
- Input: a representation of previous words (w_1, w_2, …)
- Output: probability distribution over possible next words.

- N-grams used exact words! (P("cat"))

- Equi-distance!
- 1-of-N encoding (aka. one-hot vector)

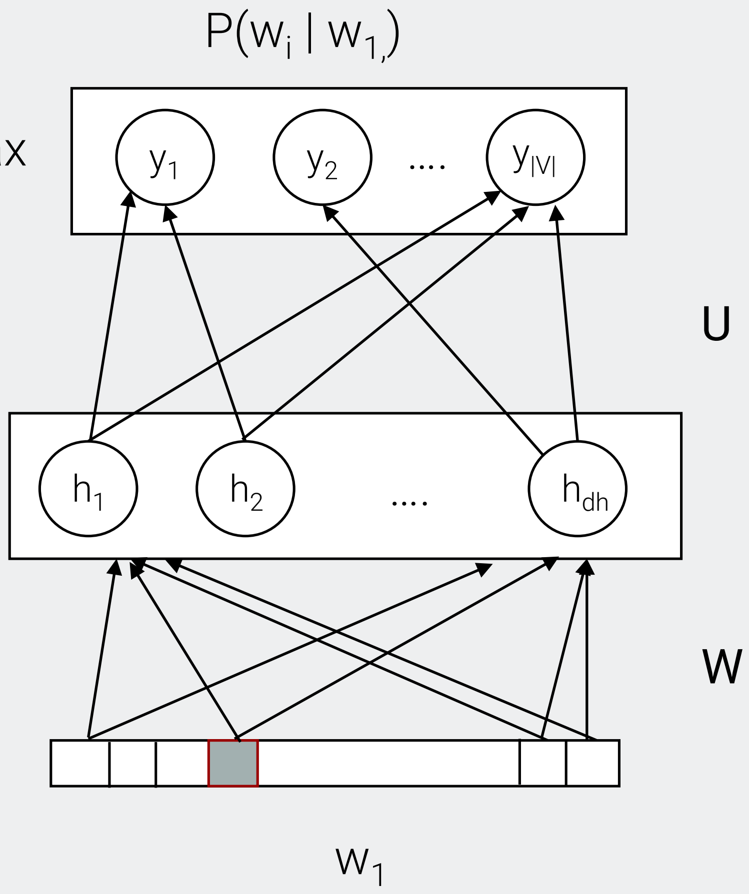
[1, 2, 3, 4, 5, 6, 7, …., …, | V |]
[0, 0, 0, 0, 0, 1, 0, …., 0, 0 , 0]

Feed Forward Net - Execution

Output layer - softmax
P(w | context)

Hidden Layer

Input Layer
one-hot vectors



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

Feed Forward Net - Training

Positive samples ($w_{3,}$ w_{402})
(metal jacket)

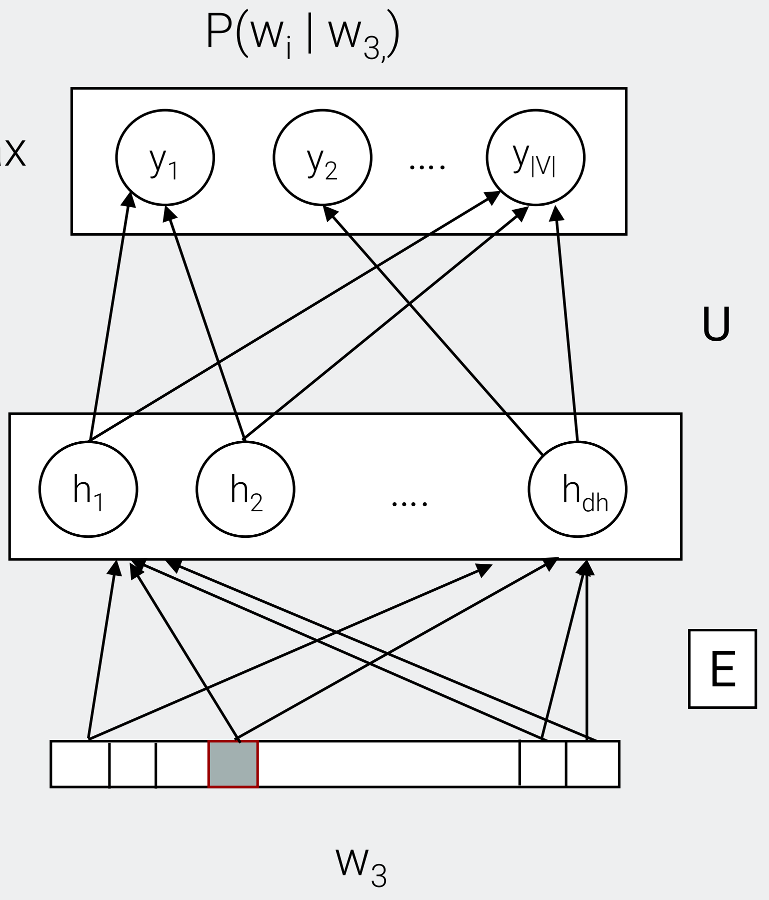
Negative samples ($w_{3,}$ w_{xx})
(metal heavy)
(metal towel)

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

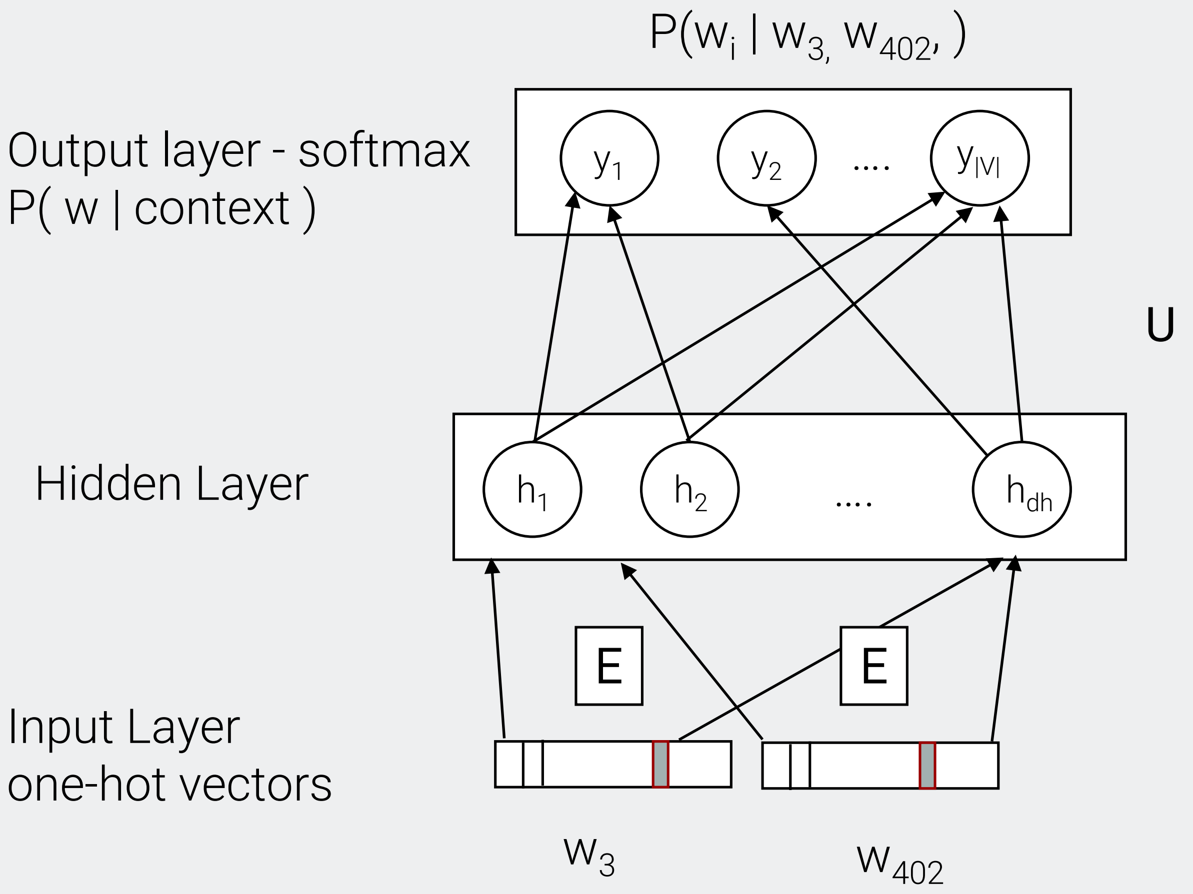
Output layer - softmax
P(w | context)

Hidden Layer

Input Layer
one-hot vectors



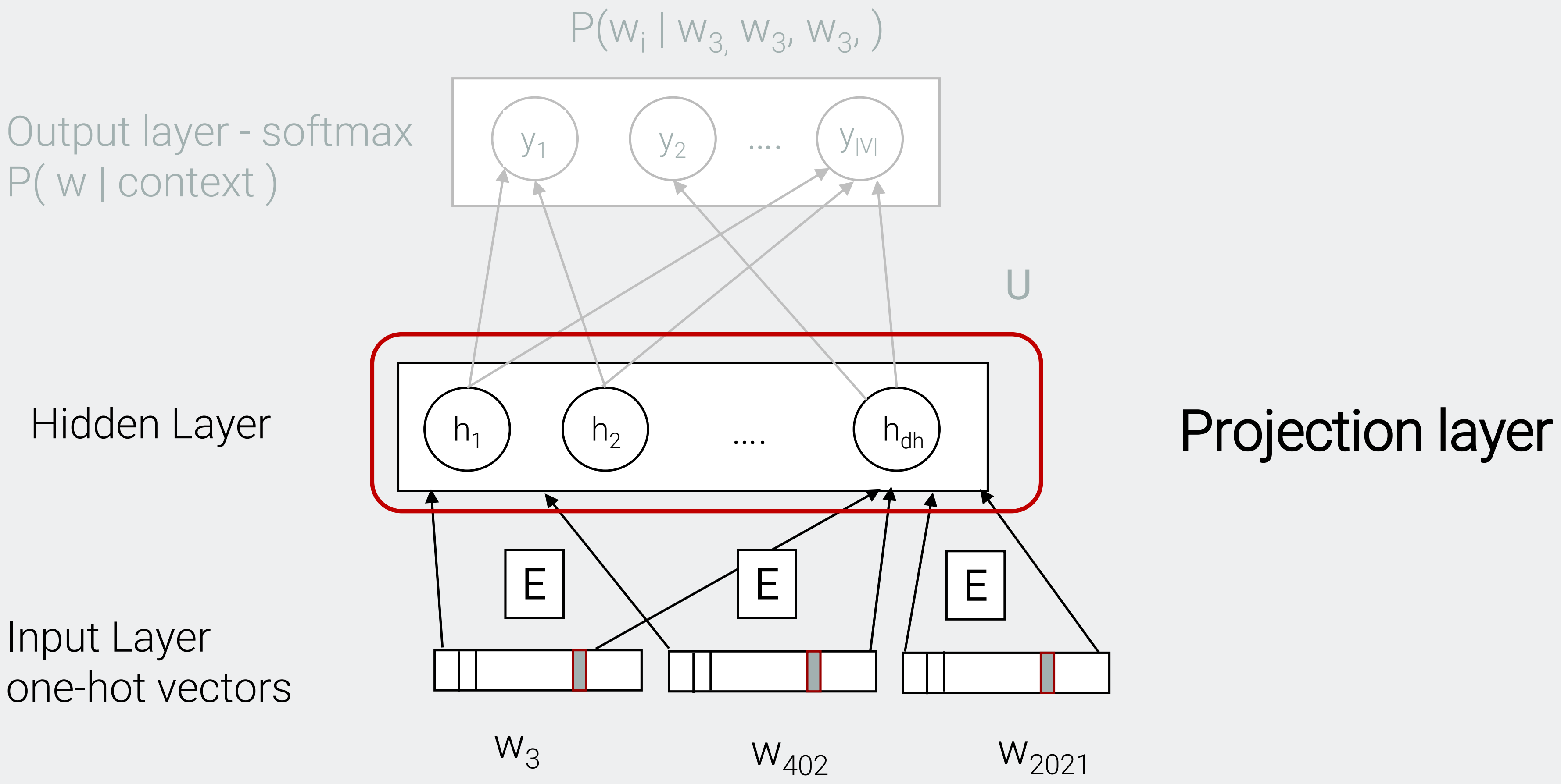
Feed Forward Net - Training

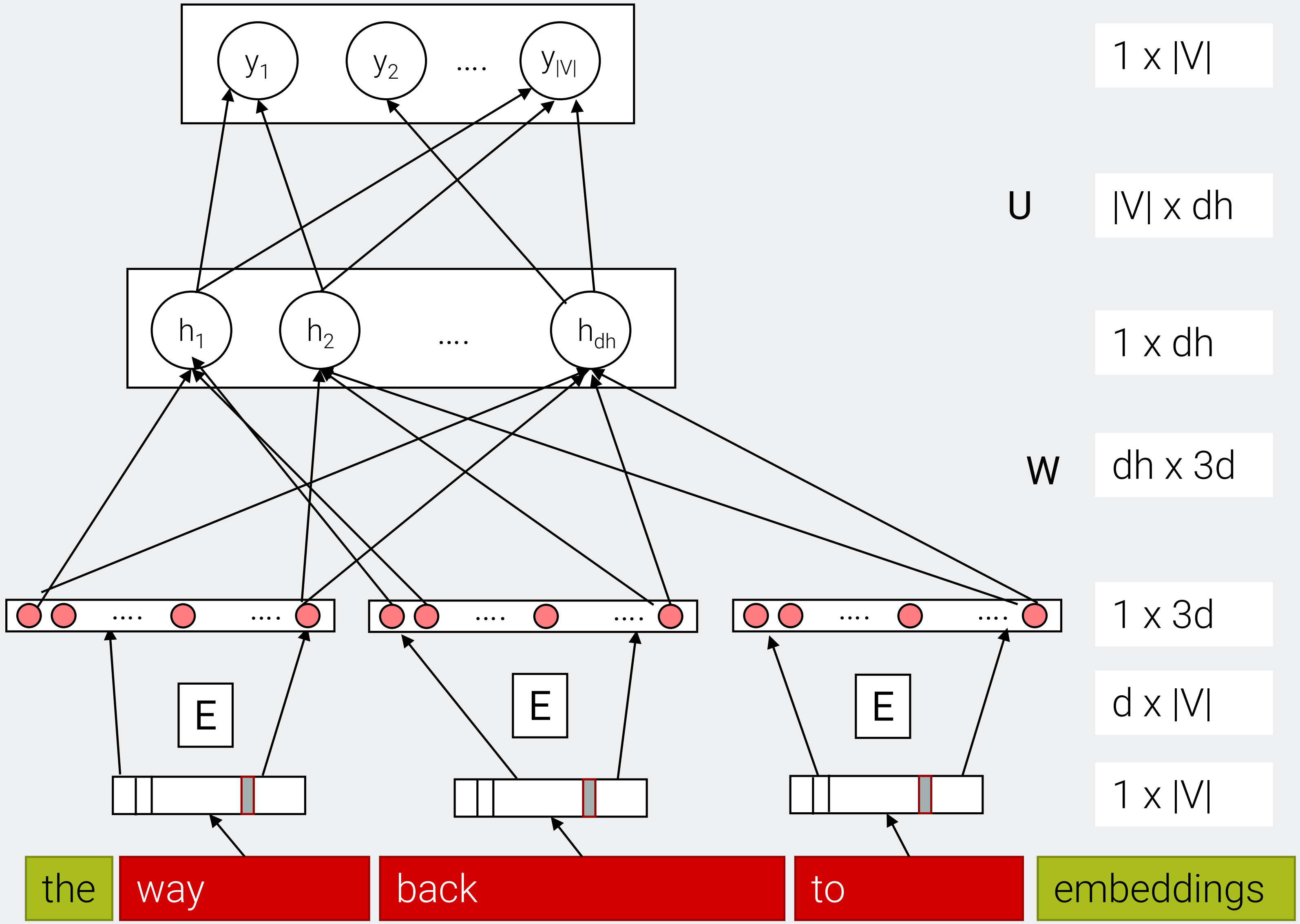


Positive samples ($w_{3,}$ w_{402}, w_{2021})
(**metal skirt hanger**)

Negative samples ($w_{3,}$ w_{402}, w_{xx})
(metal skirt mouse)
(metal skirt towel)

Feed Forward Net - Training





Output layer - softmax
P(w | context)

$$\boxed{y = \mathrm{softmax}(z)}$$

$$\boxed{z = Uh}$$

$$\boxed{h = \sigma(We+b)}$$

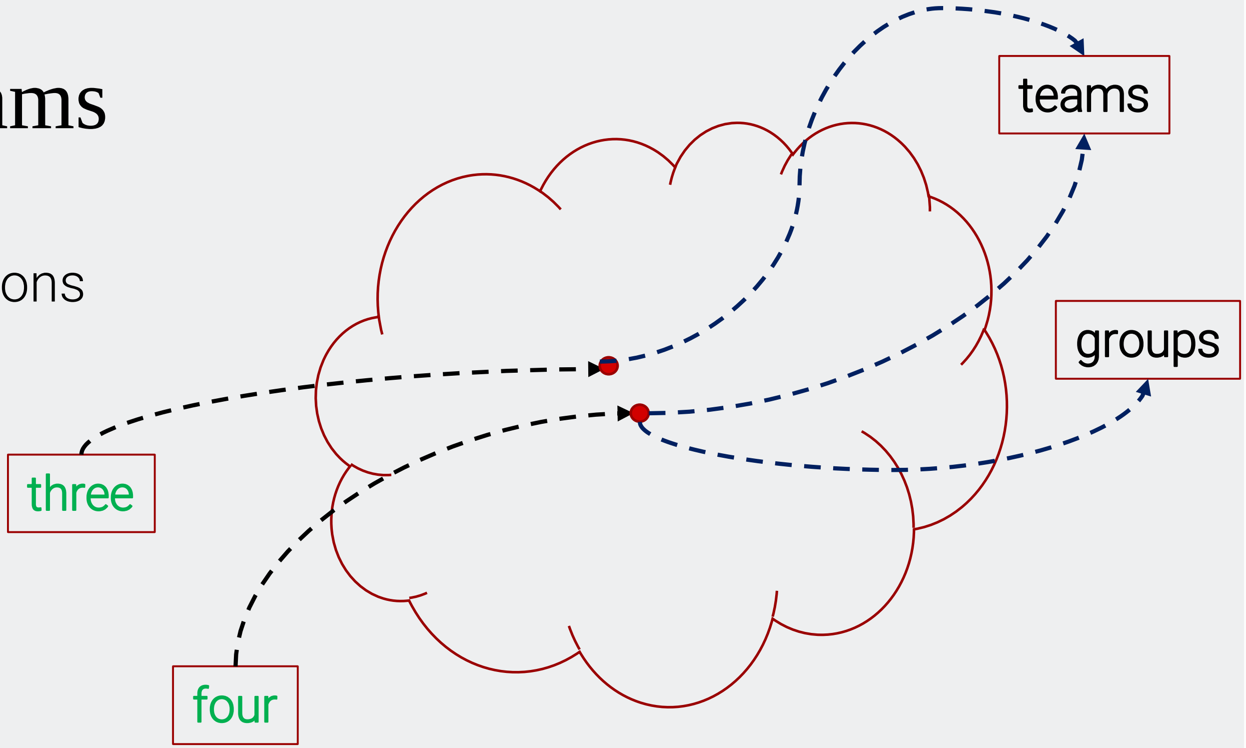
$$\boxed{e = (Ex_1, Ex_2, \ldots, Ex)}$$

Projection layer

Input Layer, one-hot vectors

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

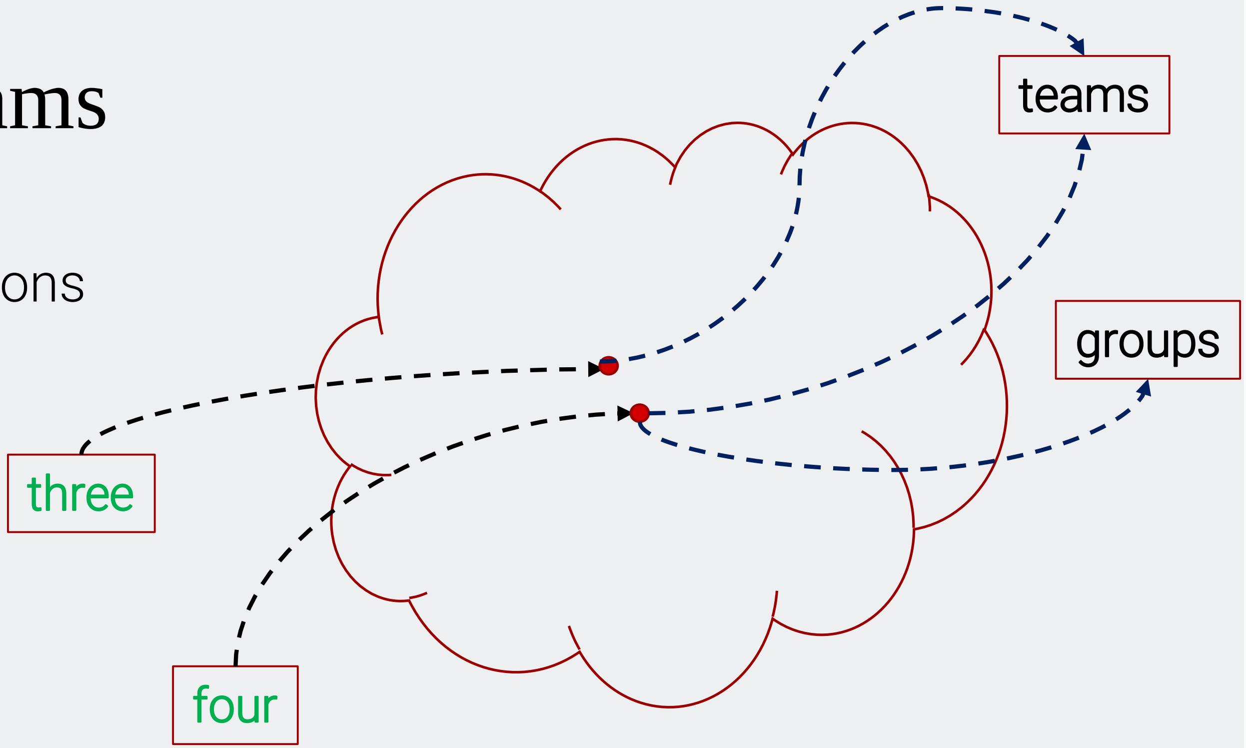


P(**teams** | four) ≈ P(**groups** | four)

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(**teams** | four) ≈ P(**groups** | four)

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
- Computationally high costs

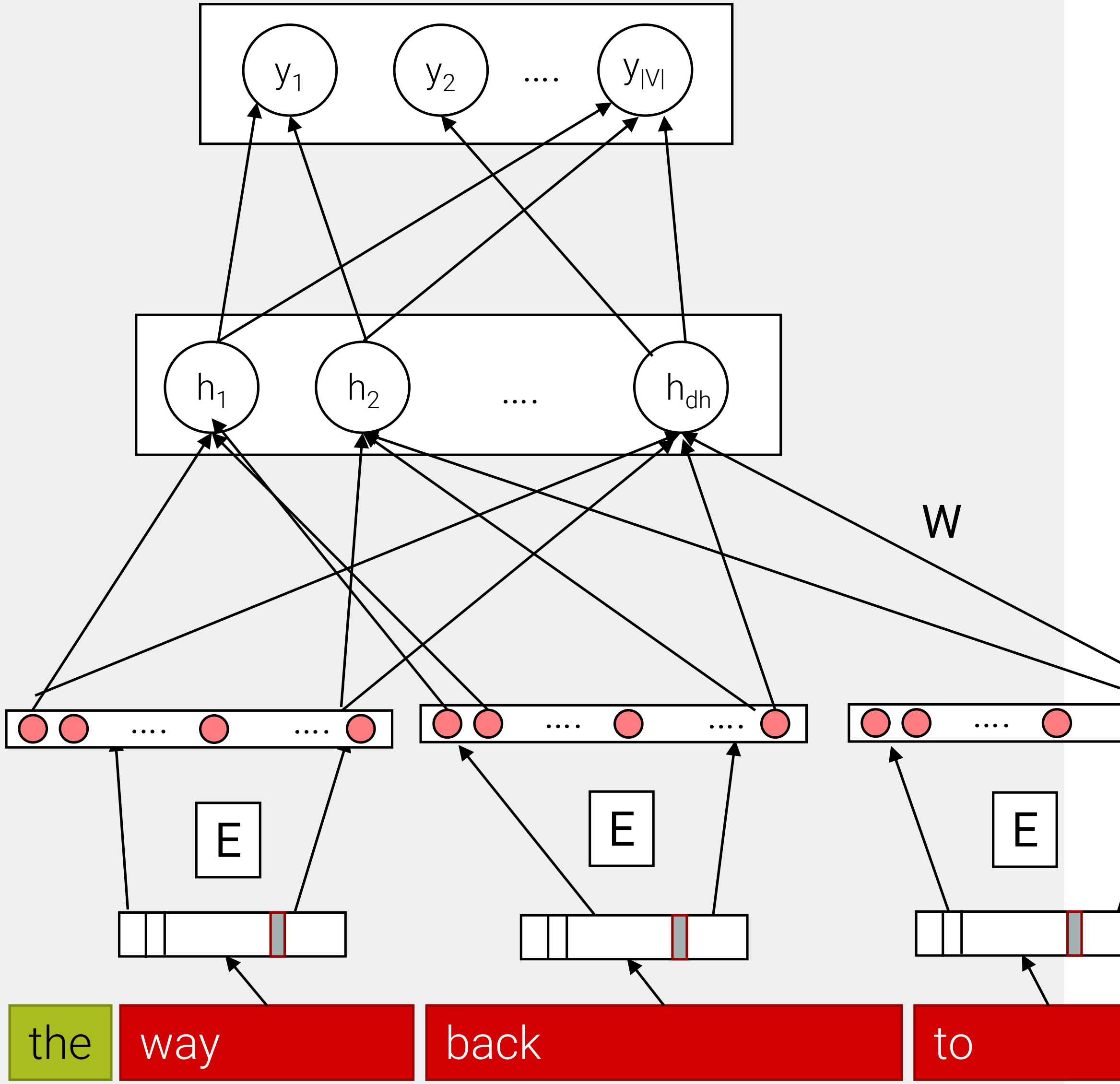
Content

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

(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

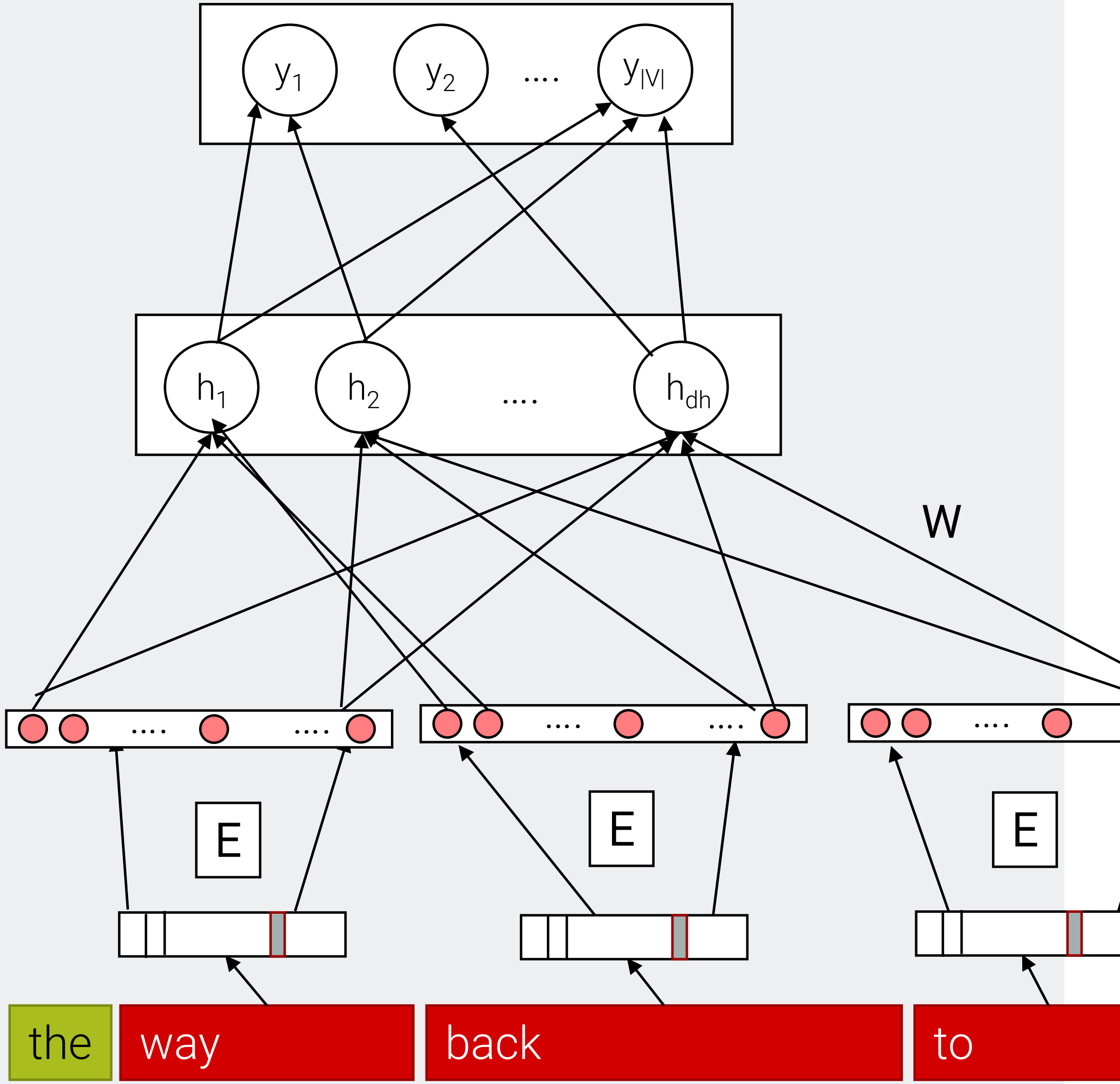
    (No symmetry in input processing)



(Simple) Neural Language Models

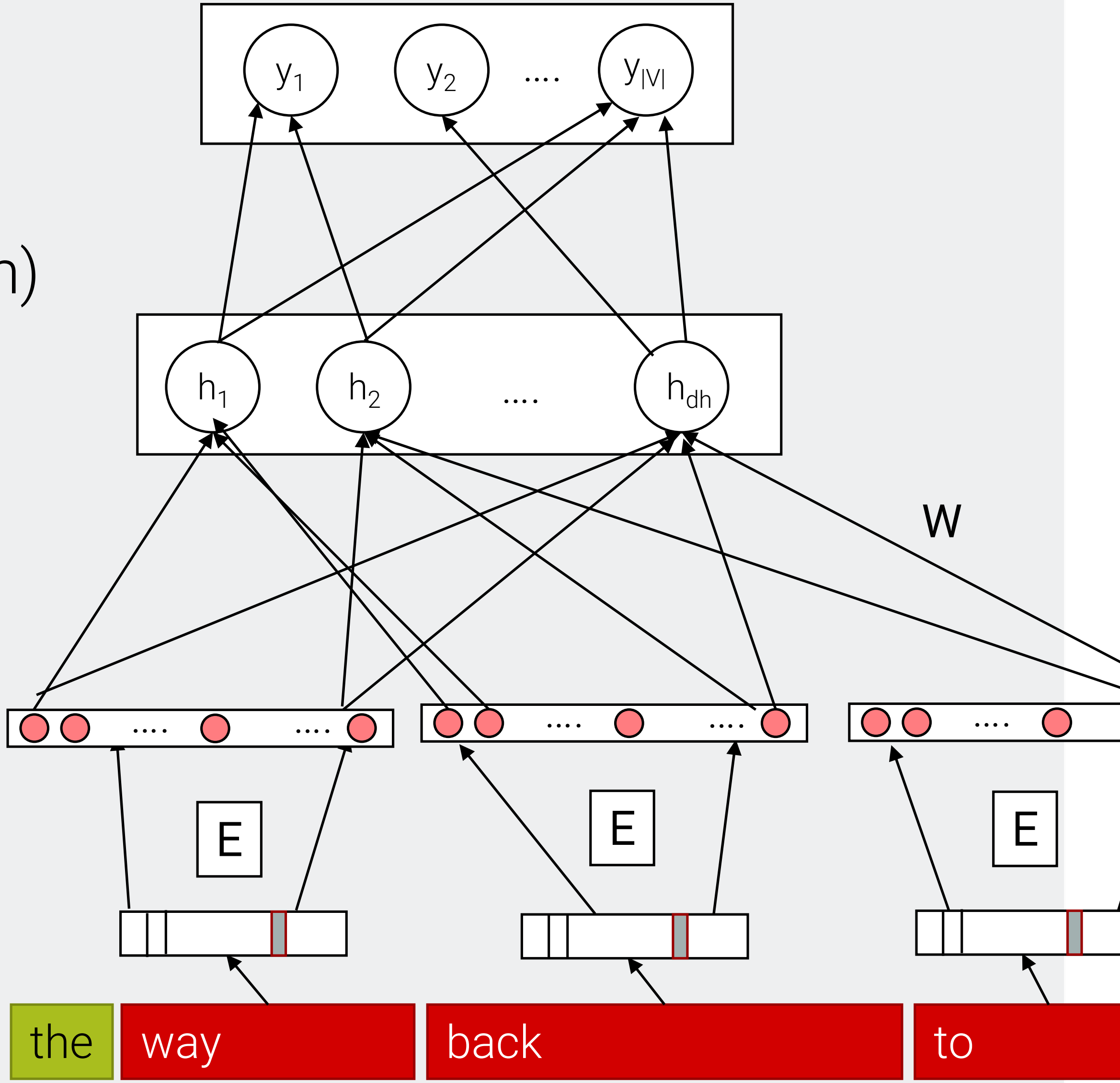
- How to deal with inputs of varying lengths (i.e. sequences)?

- 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
- Previous ML approaches have access to all input, simultaneously
- How to deal with *sequences of varying lengths*?



Sequences – Input of **Variable Lengths**

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

- 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?

- 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 $i \leftarrow 1, 2, \ldots, l$ **do** $s \leftarrow$ ADD1(x_i, s)
end for

Recursive Function for Natural Language Understanding

- ADD1 is hardcoded
- Parametrized recursive function

- Memory: $\mathbf{h} \in \mathbb{R}^{d_h}$

- Input x_1 and memory **h**, returns the new **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 ← 0
function ADD1(v,s)
    if v = 0 then return s
    else return s + 1
    end if
end function
```

Algorithm 2 A function ADD1

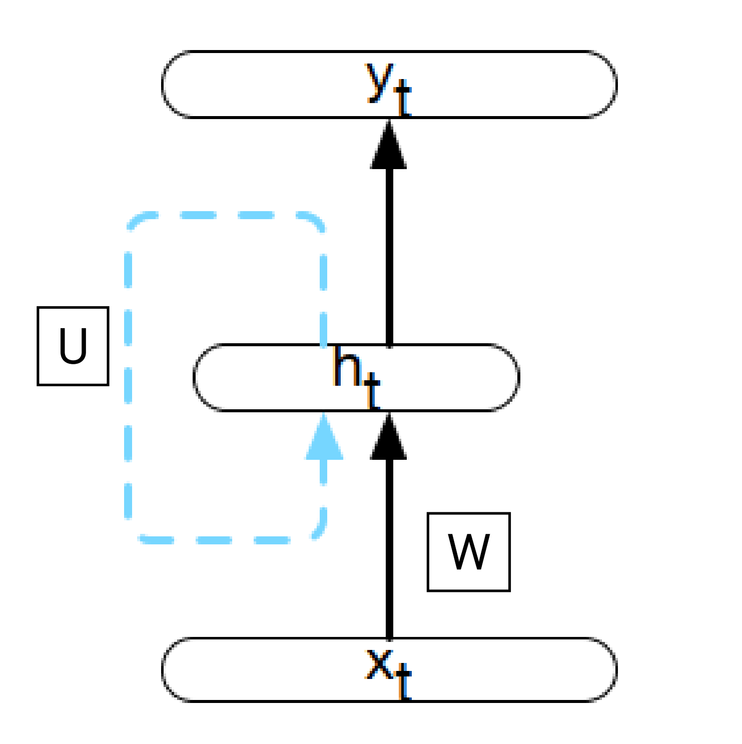
```
s ← 0
for i ← 1,2,...,l do s ← ADD1(x_i, s)
end for
```

Recursive Function for Natural Language Understanding

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

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

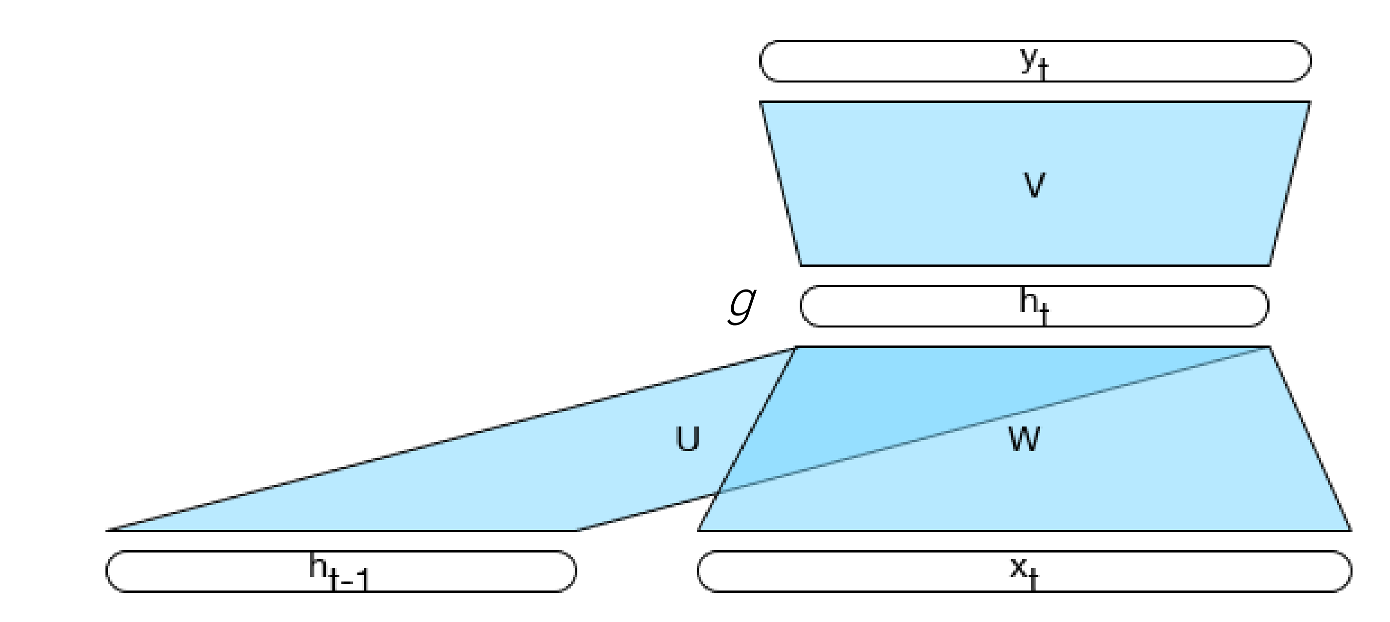
$$f(x_t, \mathbf{h}_{t-1}) = g(\mathbf{W}\phi(x_t) + \mathbf{U}\mathbf{h}_{t-1})$$



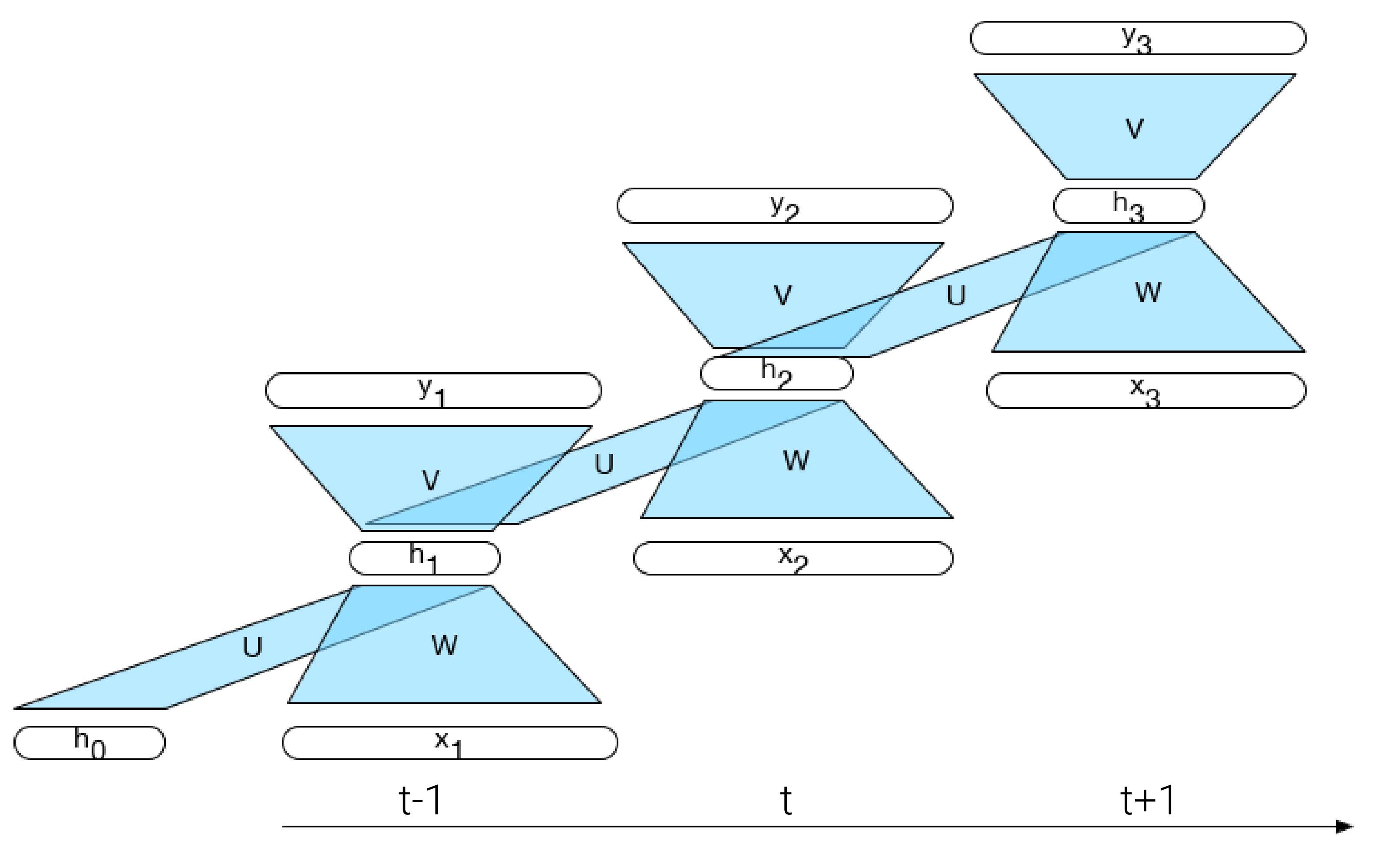
Recursive Neural Network – Unrolled

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

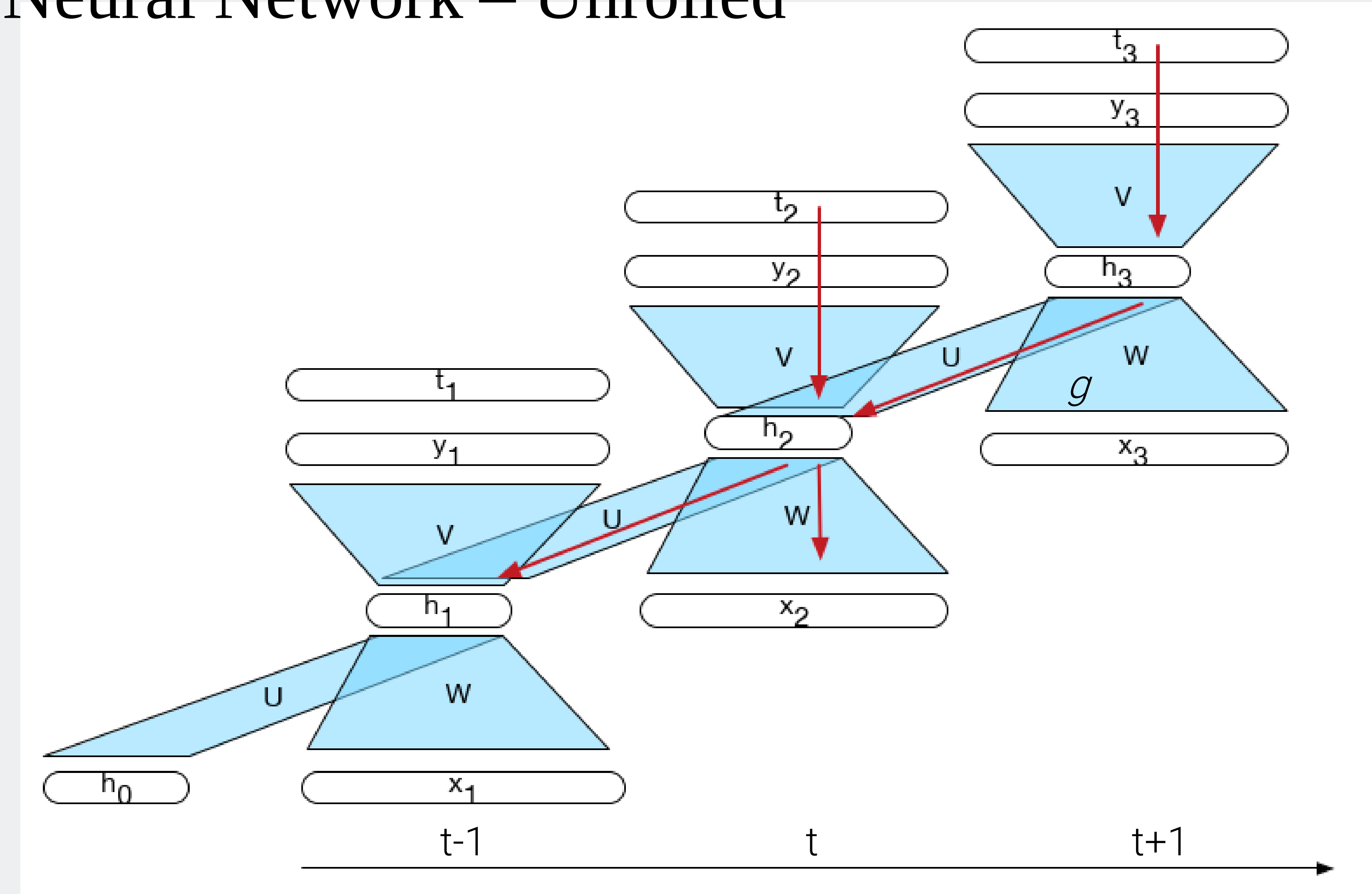
$$y_t = softmax(Vh_t)$$



Recursive Neural Network – Unrolled



Recursive Neural Network – Unrolled



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!

$$P(w_n|w_1^{n-1}) = y_n = softmax(Vh_n)$$

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

- Cross-entropy function for training

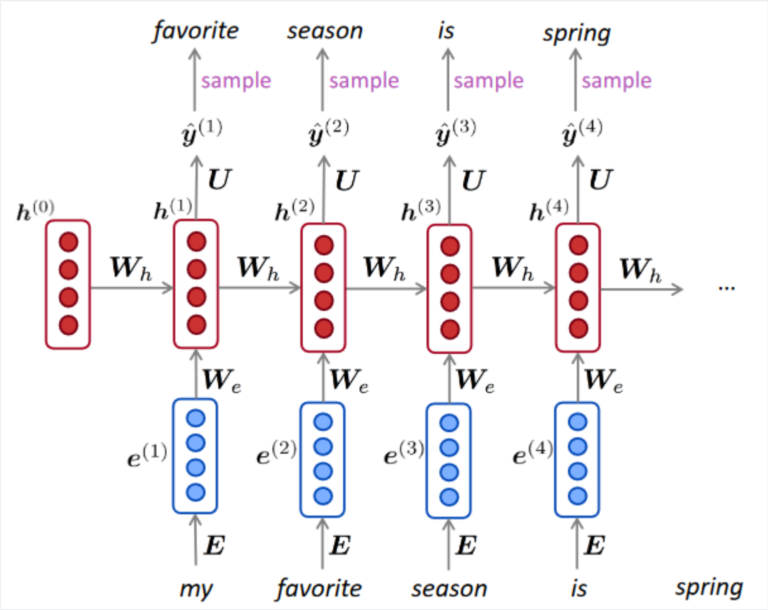
$$L_{CE}(\hat{y},y) = -\log \hat{y}_i = -\log \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- Perplexity for evaluation

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

RNN – Language Models

- Generate text by repeated sampling

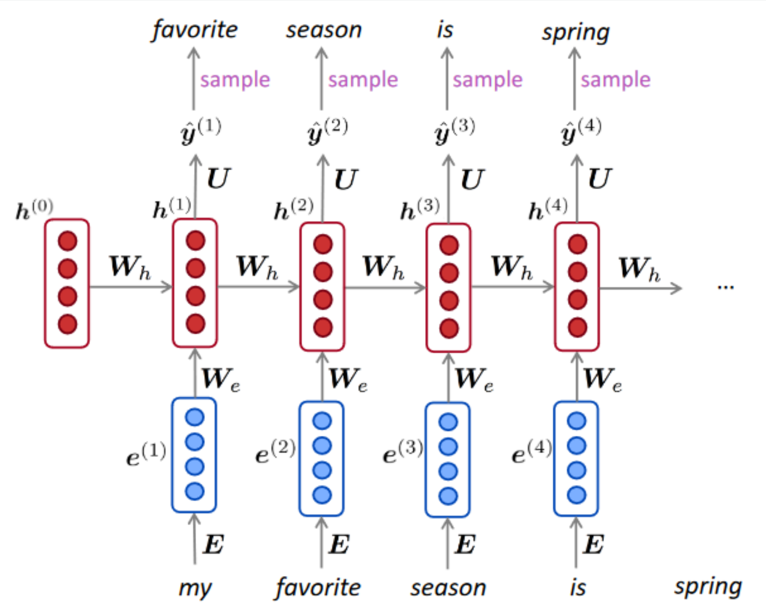


- RNN-LM trained on Obama speeches

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.

https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

RNN – Language Models



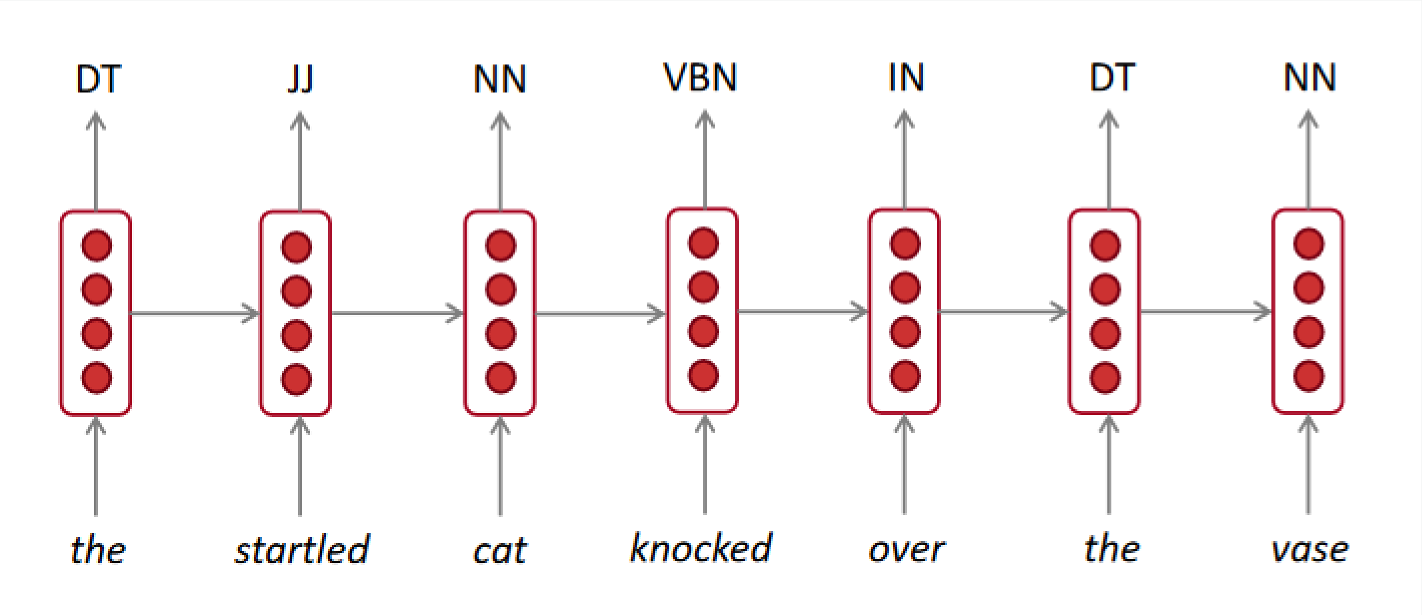
- Generate text by repeated sampling
  - On any kind of text!

  - Character level example

http://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-network

RNN – Applications

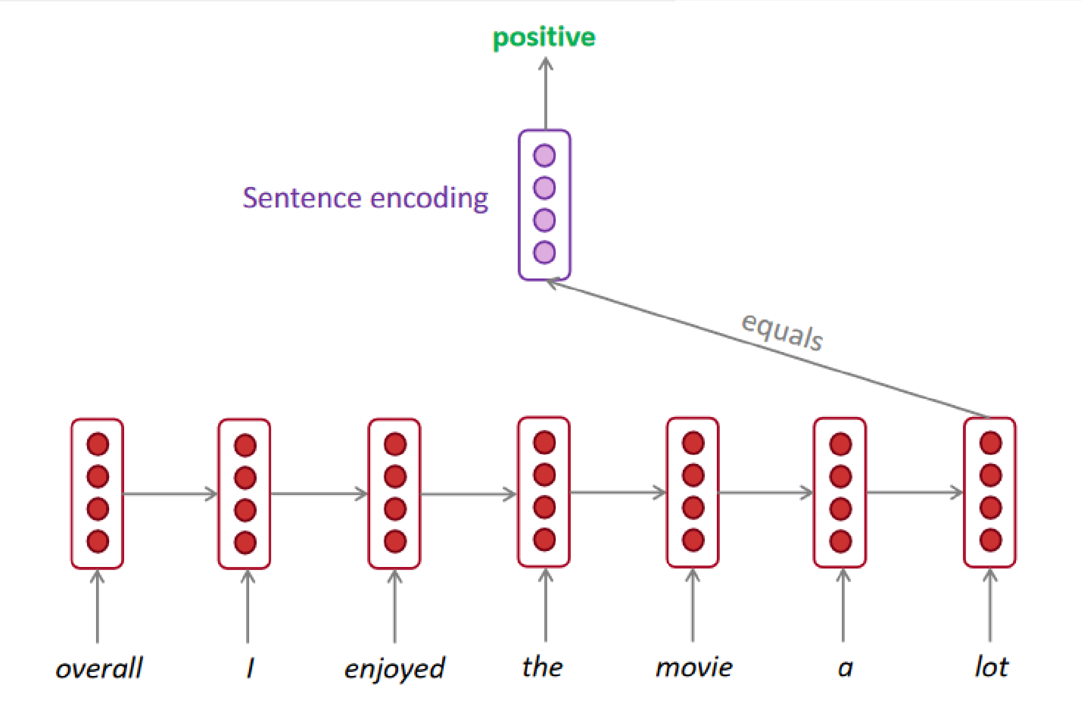
- Tagging (POS, named entity recognition, IOB encoding etc.)

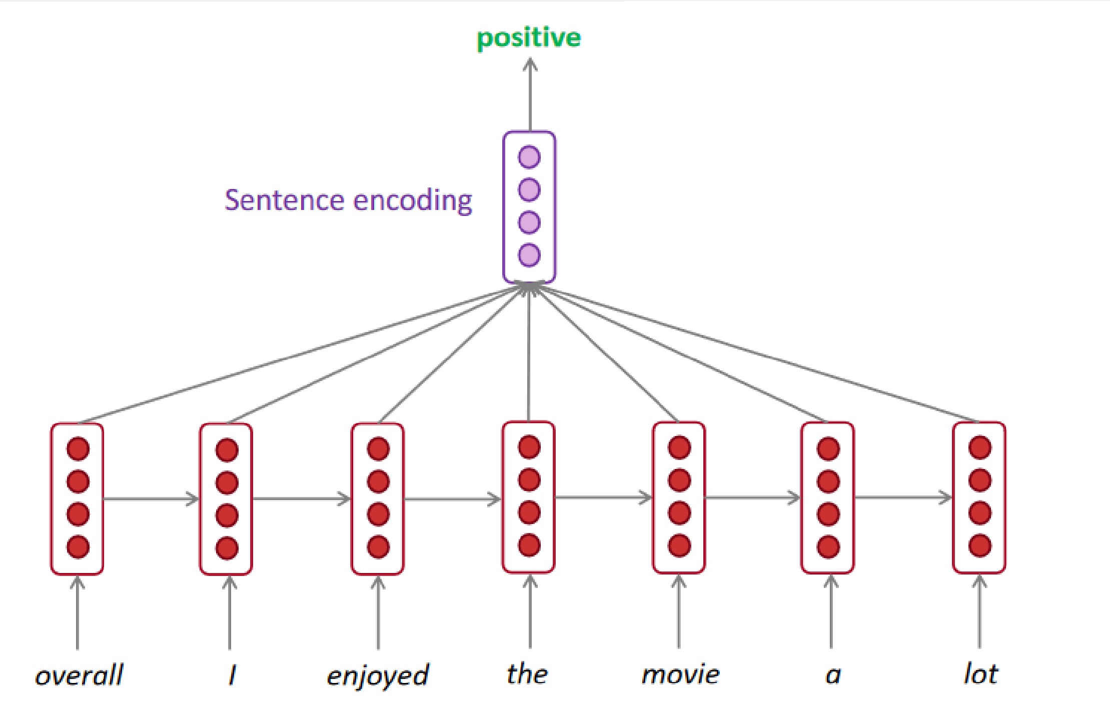


Jurafsky & Martin, SLP, 3rd Edition: *Chapter 8*

RNN – Applications

- Sentence Classification

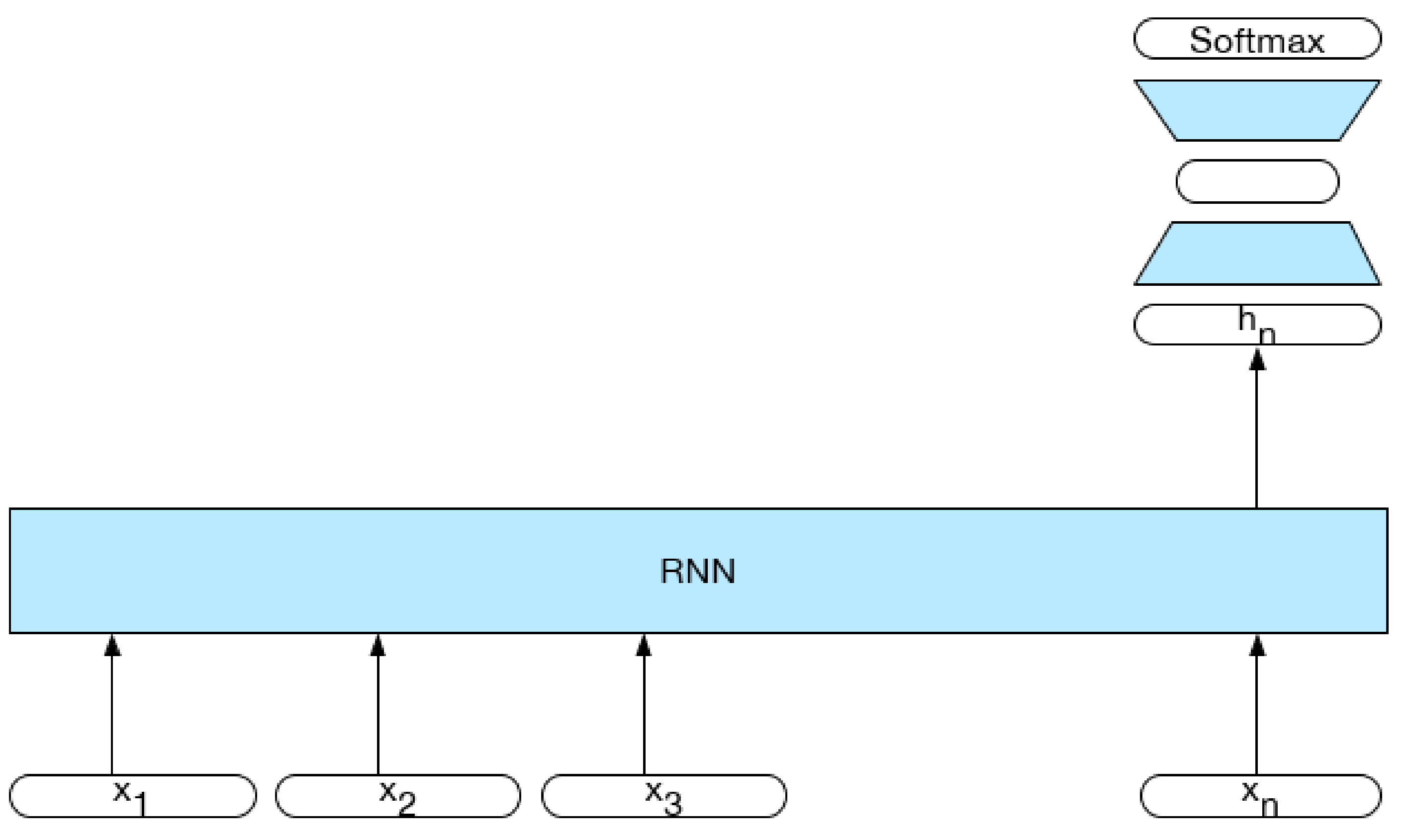




RNN – Deep Networks: Stacked and Bidirectional

- Sequence Classification

- end-to-end training

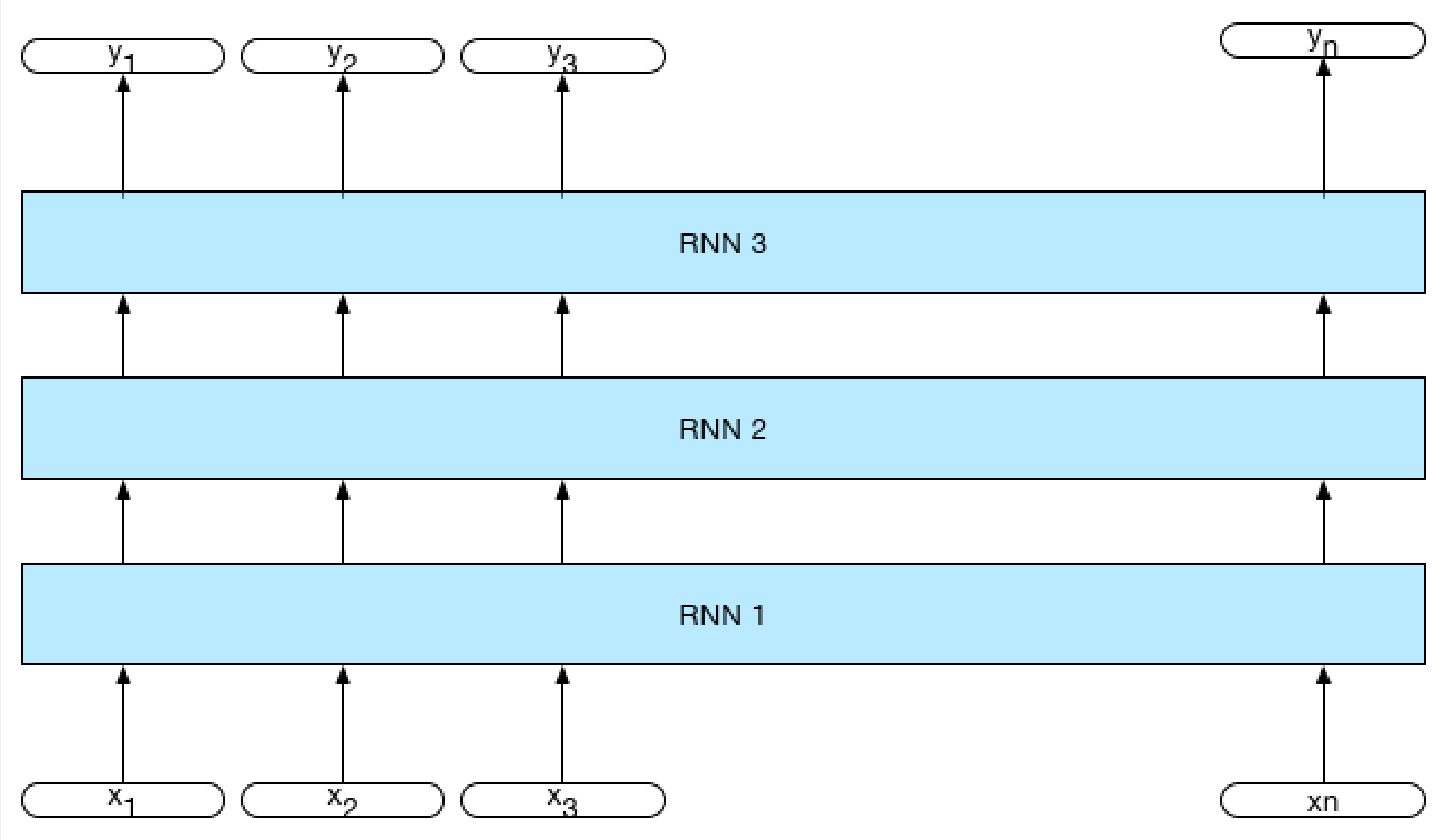


RNN – Deep Networks: Stacked

- Stacked

- Outperform single-layer
- Induce representations

- High training costs



RNN – Deep Networks: Bidirectional

- $\text{RNN}_{\text{forward}}$

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

- We have access to the entire input sequence

- $\text{RNN}_{\text{backward}}$

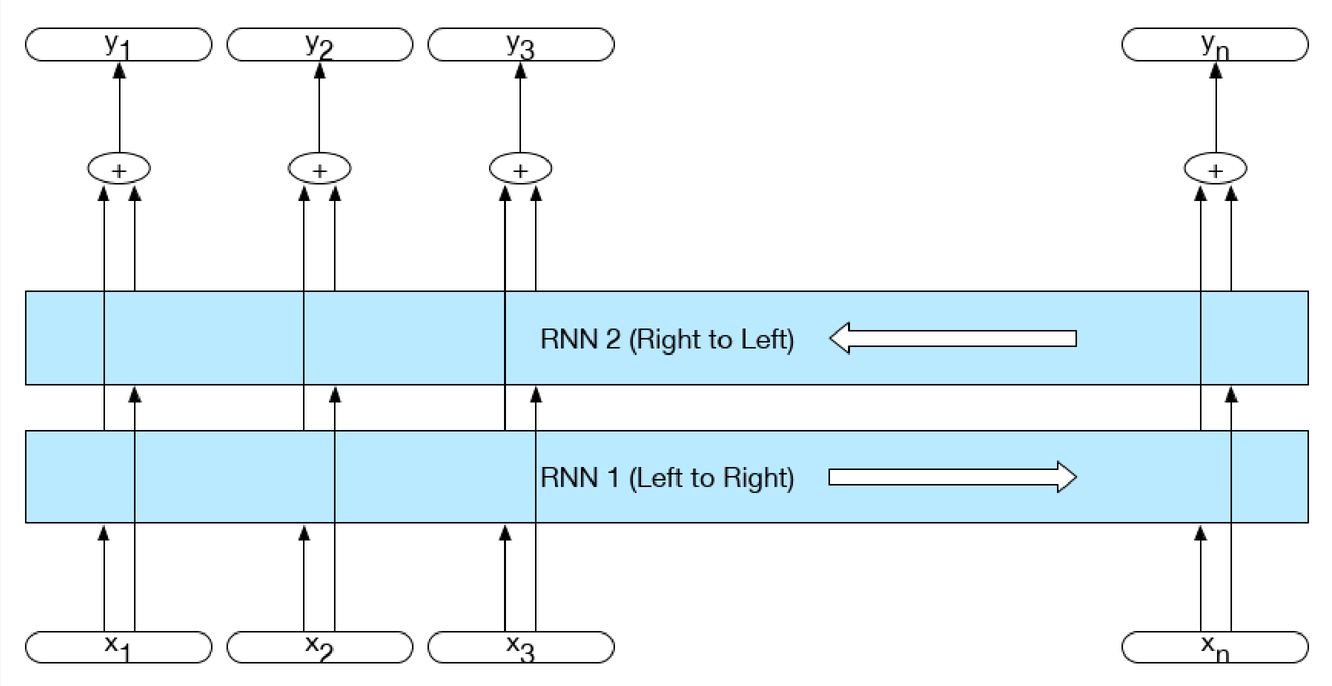
$$h_t^b = RNN_{backward}(x_t^n)$$

- Combine them -> Bi-RNN

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

RNN – Deep Networks: Bidirectional

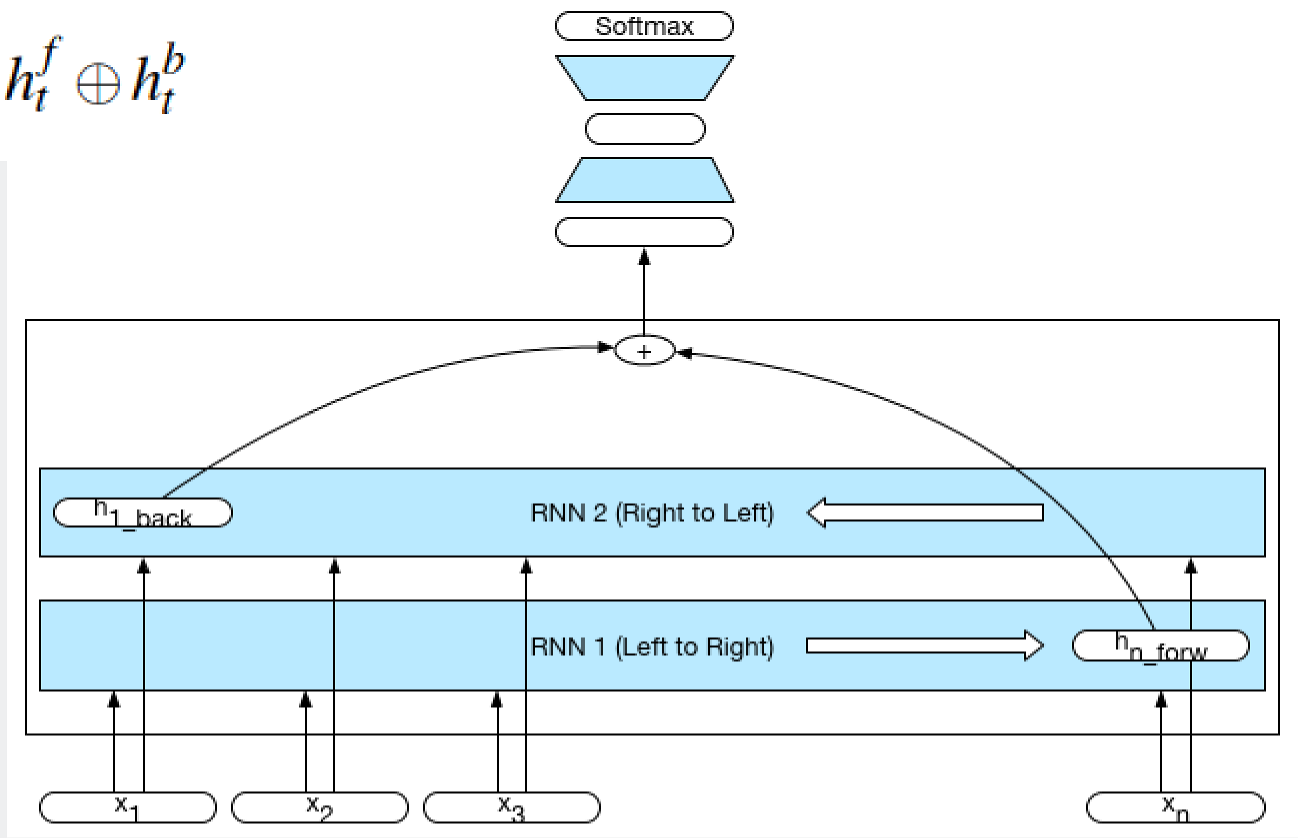
- Bi-RNN combines $h_t = h_t^f \oplus h_t^b$



RNN – Deep Networks: Bidirectional

- Bi-RNN combines $h_t = h_t^f \oplus h_t^b$

- Sequence classification



Content

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

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

RNN Shortcomings

- How to maintain relevant context over time?

The flights the airline was cancelling were full.

Recursive Function for Natural Language Understanding

- ADD1 is hardcoded
- Parametrized recursive function

- Memory: $\boxed{\mathbf{h} \in \mathbb{R}^{d_h}}$

- Input x_1 and memory **h**, returns the new **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 ← 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 ← 0
for i ← 1,2,...,l do s ← ADD1(x_i, s)
end for
```

Long Short-Term Memory Networks (LSTMs)

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

Long Short

gates

context ≈ memory

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

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

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

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

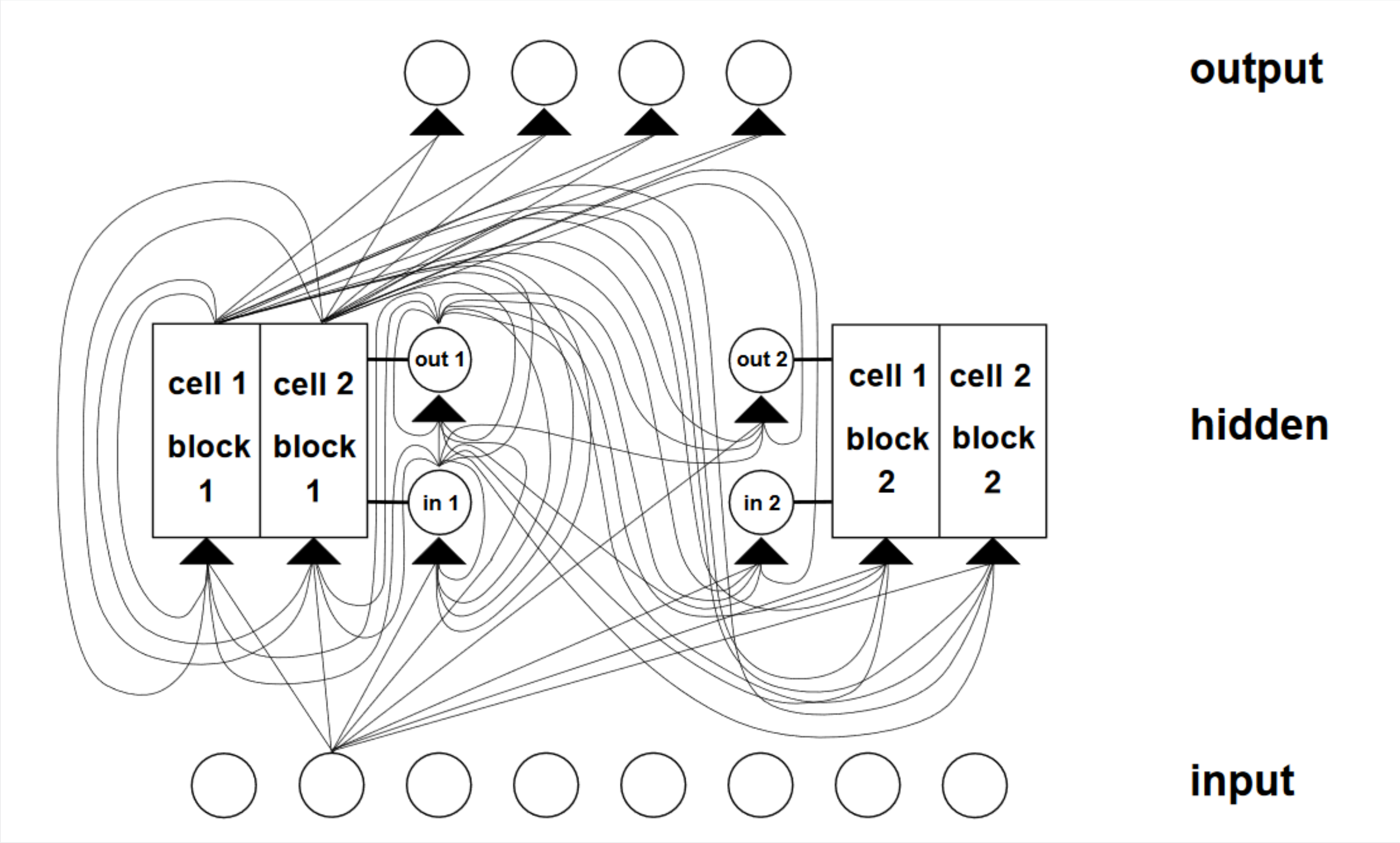
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length n

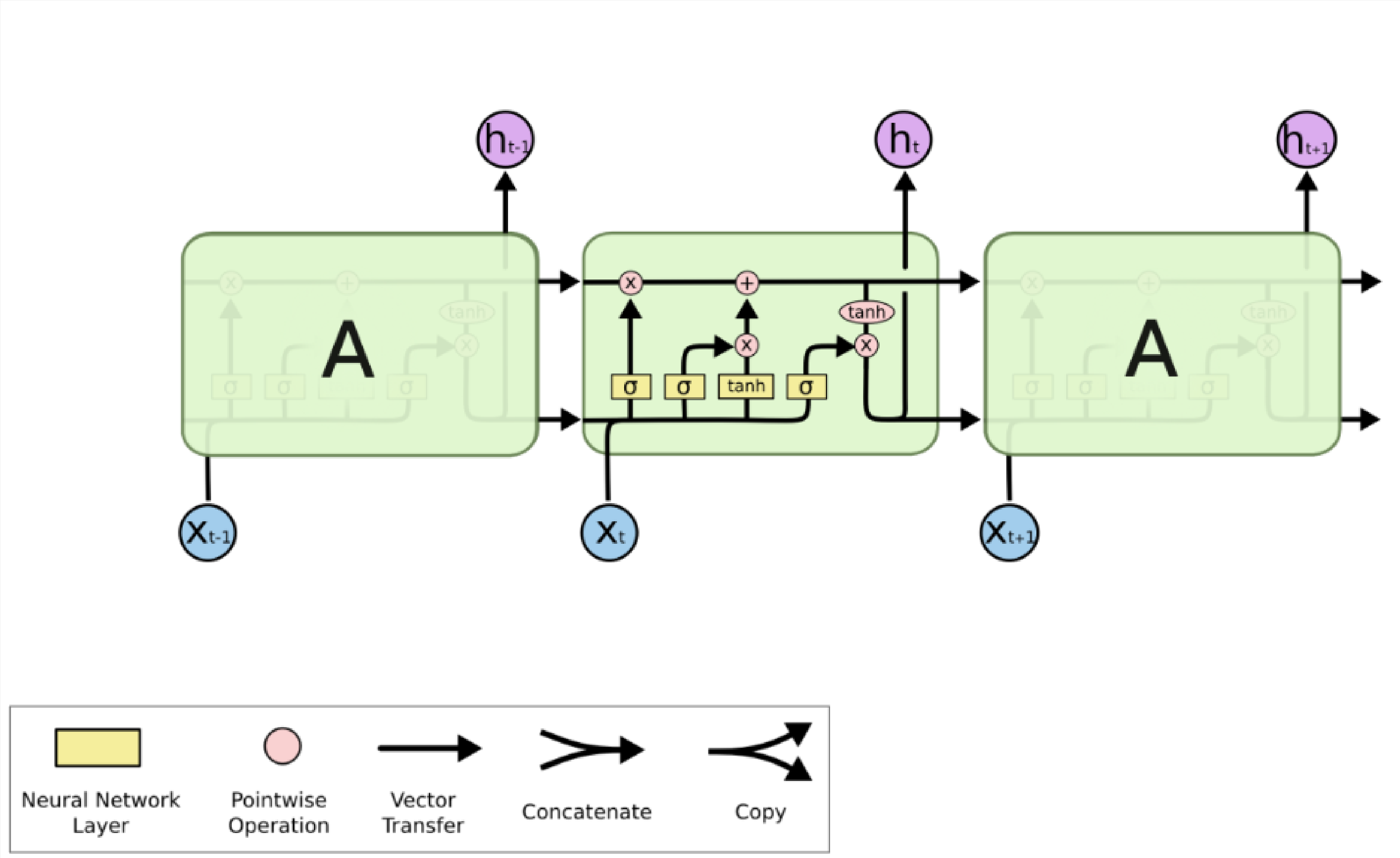
https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

Long Short-Term Memory Networks (LSTMs)



(Hochreiter, Schmidhuber, 1997) https://www.bioinf.jku.at/publications/older/2604.pdf

Long Short-Term Memory Networks (LSTMs)

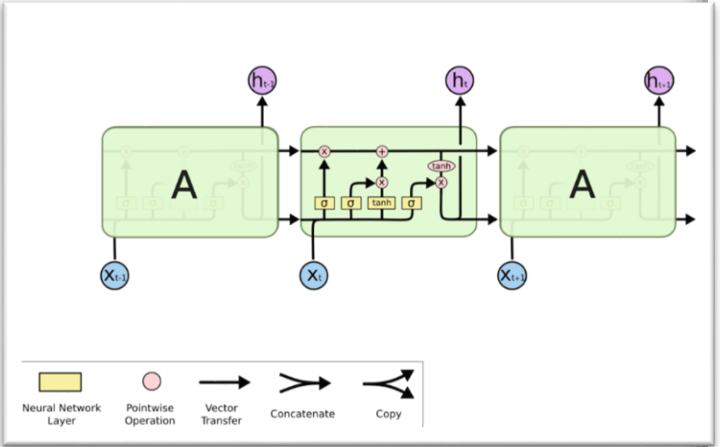


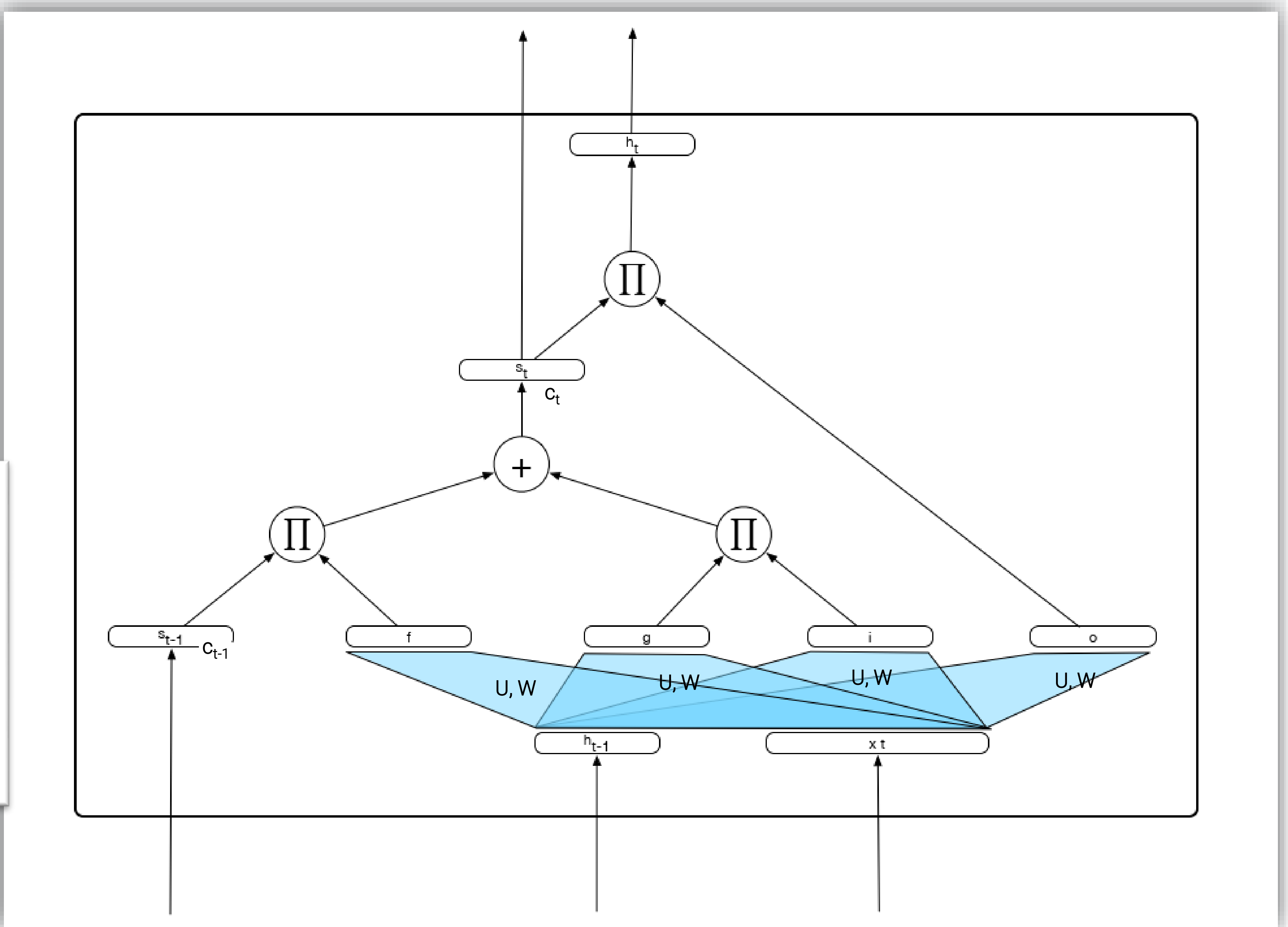
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs

- Forgetting (unnecessary info)
- Memorizing (new information)

- Learning 8 weight matrixes!

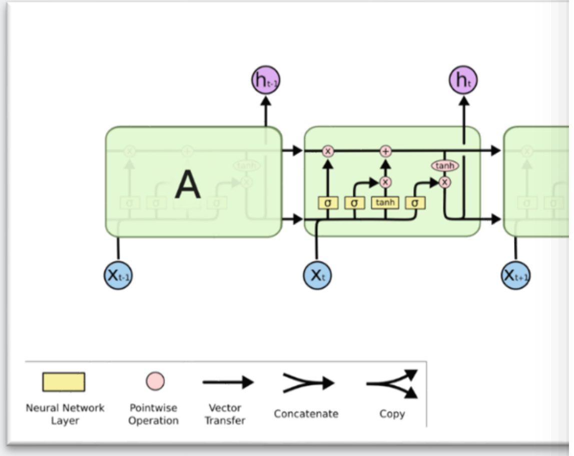


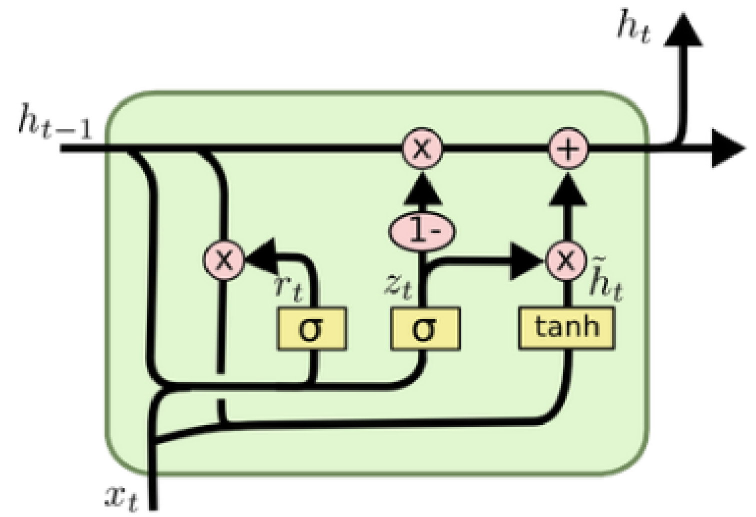


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit

- Uses only two gates: “reset”, *r*, and “update”, *z*
- Collapse “forget” and “input” gates into the “update” gate z





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

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

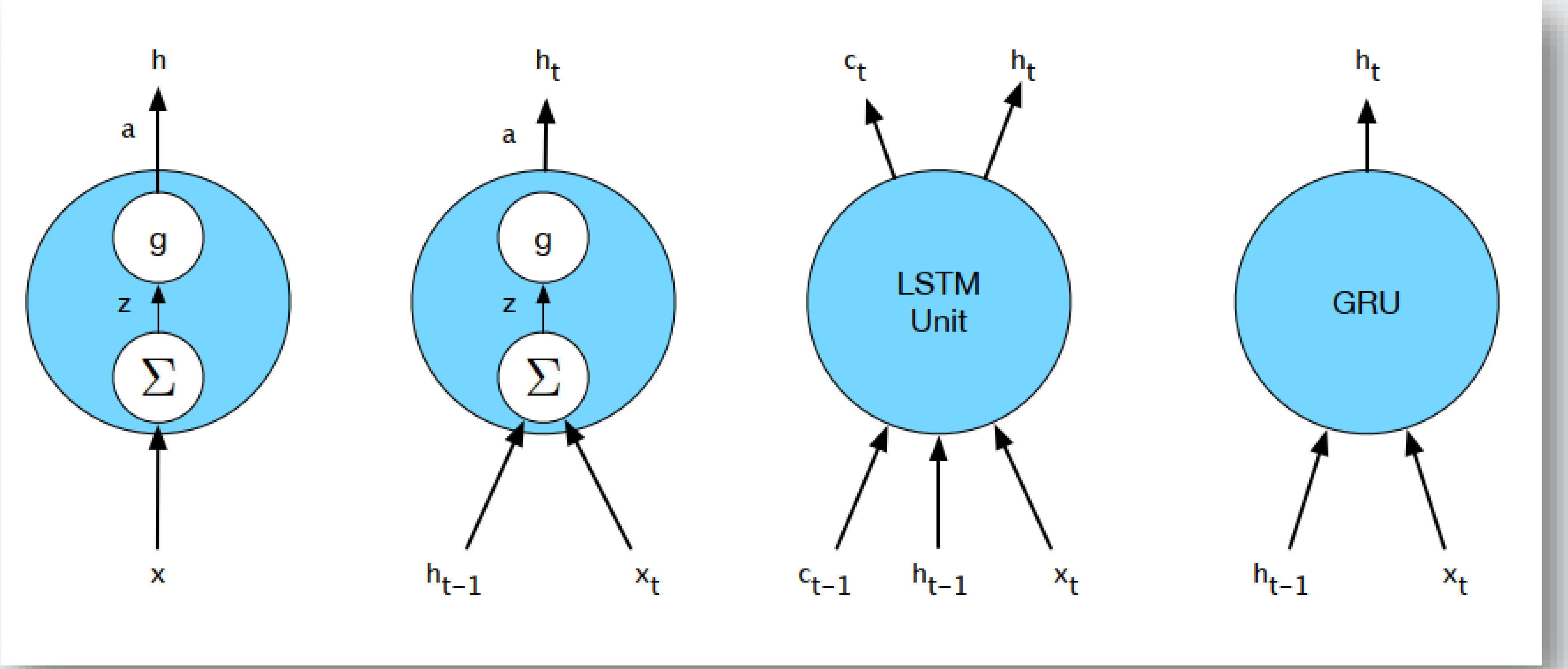
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Cho et al., 2014, http://arxiv.org/pdf/1406.1078v3.pdf
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural Units

- Complexity encapsulated in basic processing units



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)

TU WIEN Informatics

Sequence-to-Sequence aka. Encoder-decoder Models

- Neural Machine Translation

- **Source** sentence X in source language
- **Target** sentence Y in target language

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

- Translation: function application:

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

- More than one correct translation

$$\boxed{P(Y|X)}$$

$$C_k = \left\{ (t_0, \ldots, t_k) \in \mathbb{R}^{k+1} \middle| \sum_{i=1}^{k} t_k = 1 \text{ a} \ldots \text{for all } i \right\}$$

Following slides based on Cho, 2017, NLU with Distributional Representation, Chapter 6

Neural Machine Translation

- Conditional language modelling!

$$X = (x_1, x_2, \ldots, x_{T_x})$$
$$Y = (y_1, y_2, \ldots, y_{T_y})$$
$$f : V_x^+ \to C^+_{|V_y|-1}$$

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

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

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

Neural Machine Translation

- Conditional language modelling!

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

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

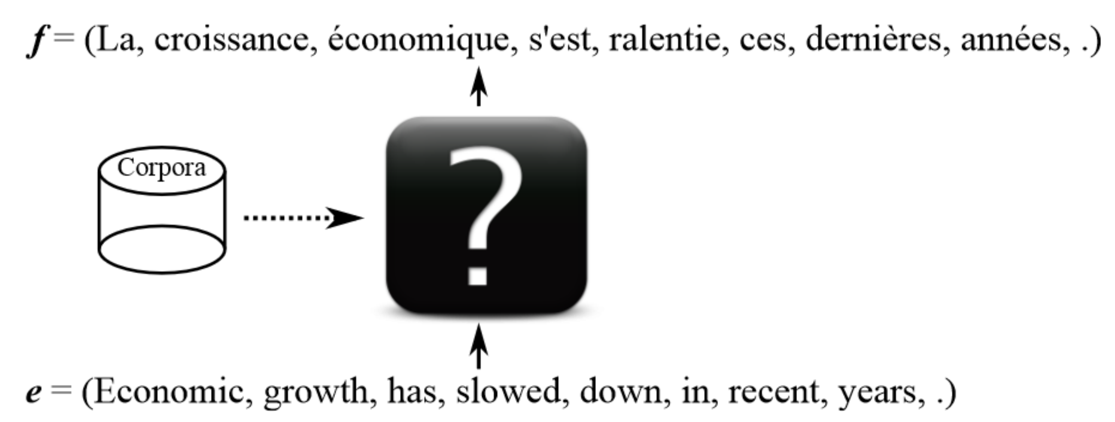
$$P(Y|X) = \underbrace{\prod_{t=1}^{T_y} P(y_t | y_1, \ldots, y_{t-1}, \underbrace{X}_{\text{conditional}})}_{\text{language modelling}}$$

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

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

$$\{(X^1, Y^1), (X^2, Y^2), \ldots, (X^N, Y^N)\}$$

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 lanugages
- 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

Brown et al, 1990, A statistical approach to machine translation
http://www.isi.edu/natural-language/download/hansard/
http://www.statmt.org/europarl/

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
  - Wikipedia
- Common Crawl Parallel Corpus (Smith et. Al, 2013)
  - http://www.statmt.org/wmt13/training-parallel-commoncrawl.tgz

Smith et al, 2013, Dirt cheap web-scale parallel text from the common crawl, ACL

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

Smith et al, 2013, Dirt cheap web-scale parallel text from the common crawl, ACL

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

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

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

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

Papineni et al. “Bleu: a method for automatic evaluation of machine translation.”

Evaluating Machine Translation – BLEU score

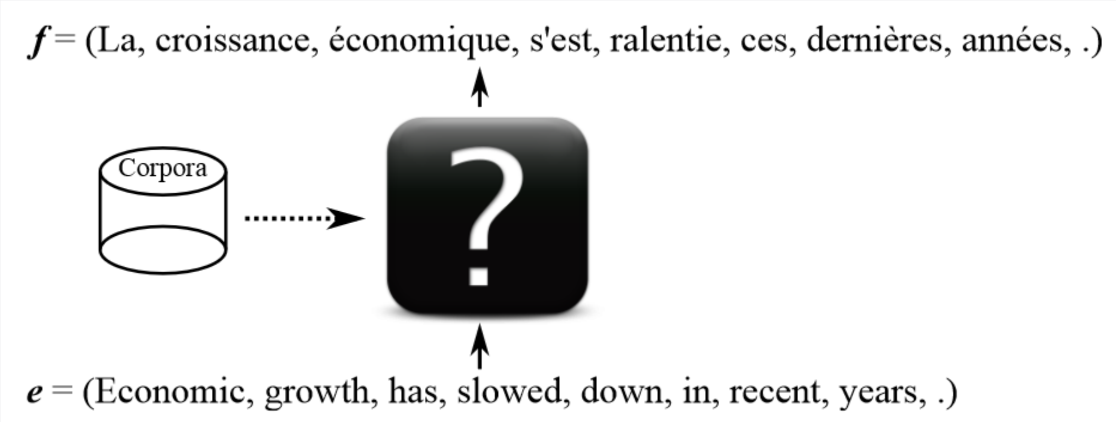
- The BLEU was shown to correlate well with human judgements
- But not perfect automatic evaluation metric

- METEOR (M. Denkowski and A. Lavie, 2014)
- TER (Translation Edit Rate, M. Snover, 2006)

Papineni et al. “Bleu: a method for automatic evaluation of machine translation.”

Neural Machine Translation

- The big picture:



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

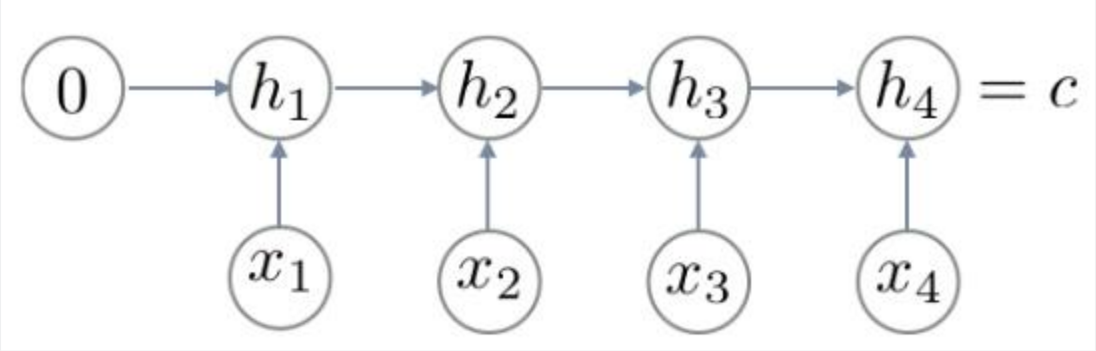
✓ Training data
✓ Evaluating MT

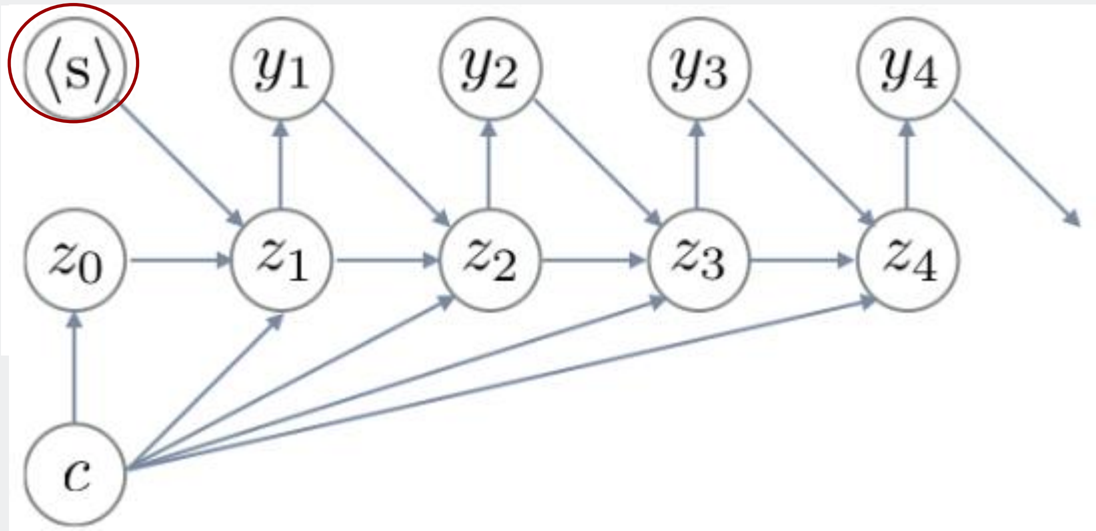
Neural Machine Translation: Encoder-Decoder Model

- Input: $Y = (y_1, \ldots, y_{t-1})$ $X = (x_1, \ldots, x_{T_x})$

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

- Start with *X*, how to handle it?
  - Variable-length sequence (RNN)
  - No explicit output/target → only the summary (vector)
  - RNN ~ *encoder*





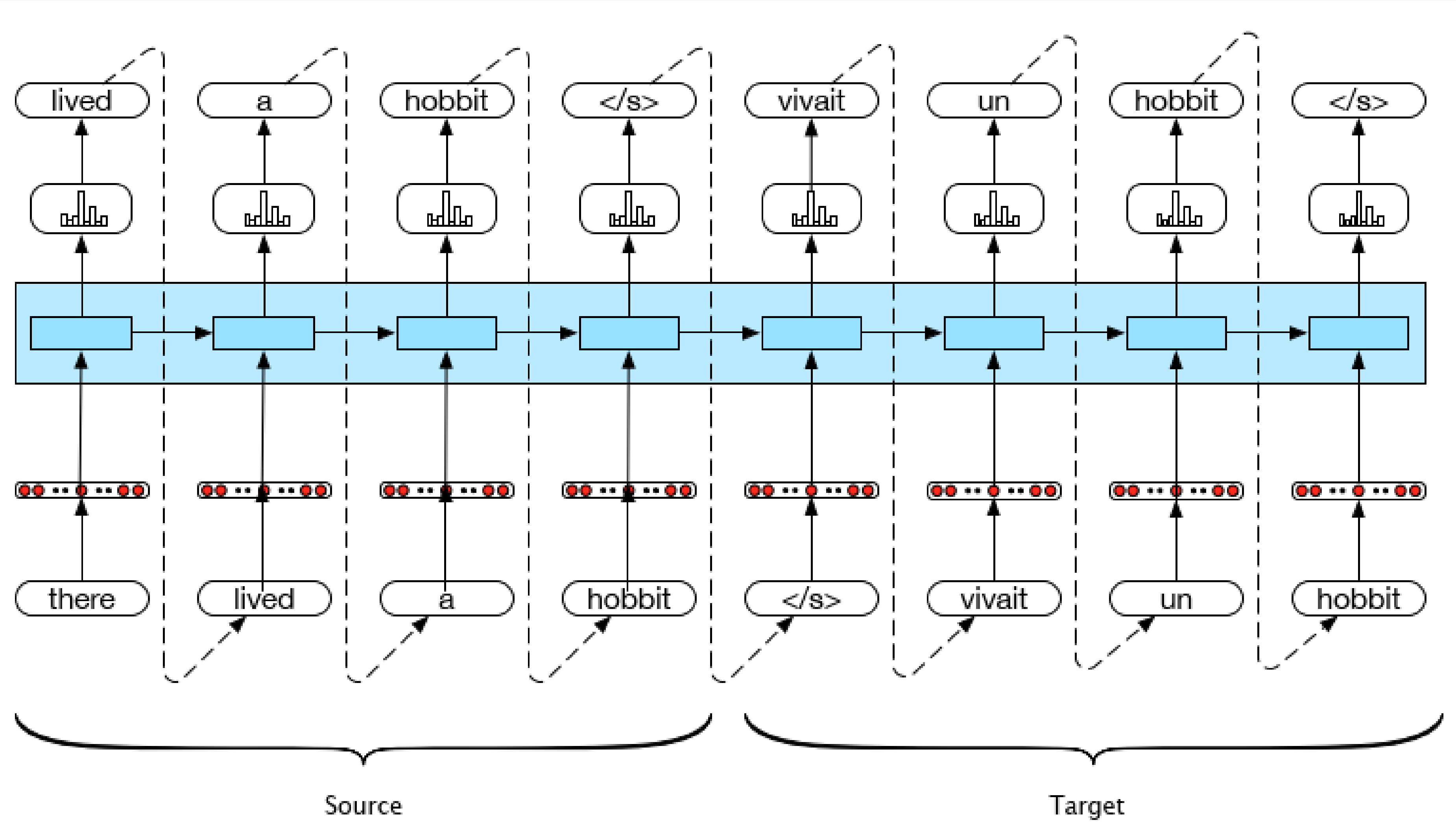
Neural Machine Translation: Encoder-Decoder Model

- 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, …



Informatics

Statistical Machine Translation

- 1990s-2010s: Statistical Machine Translation
- Core idea: Learn a probabilistic model from data
- We want to find *best* English sentence Y, given French sentence X $\text{argmax}_y P(y|x)$
- Use Bayes Rule to break this down into two components to be learnt separately:

$$= \text{argmax}_y \underbrace{P(x|y)}\,\underbrace{P(y)}$$

Translation Model

Models how words and phrases should be translated (*fidelity*). Learnt from parallel data.

Language Model

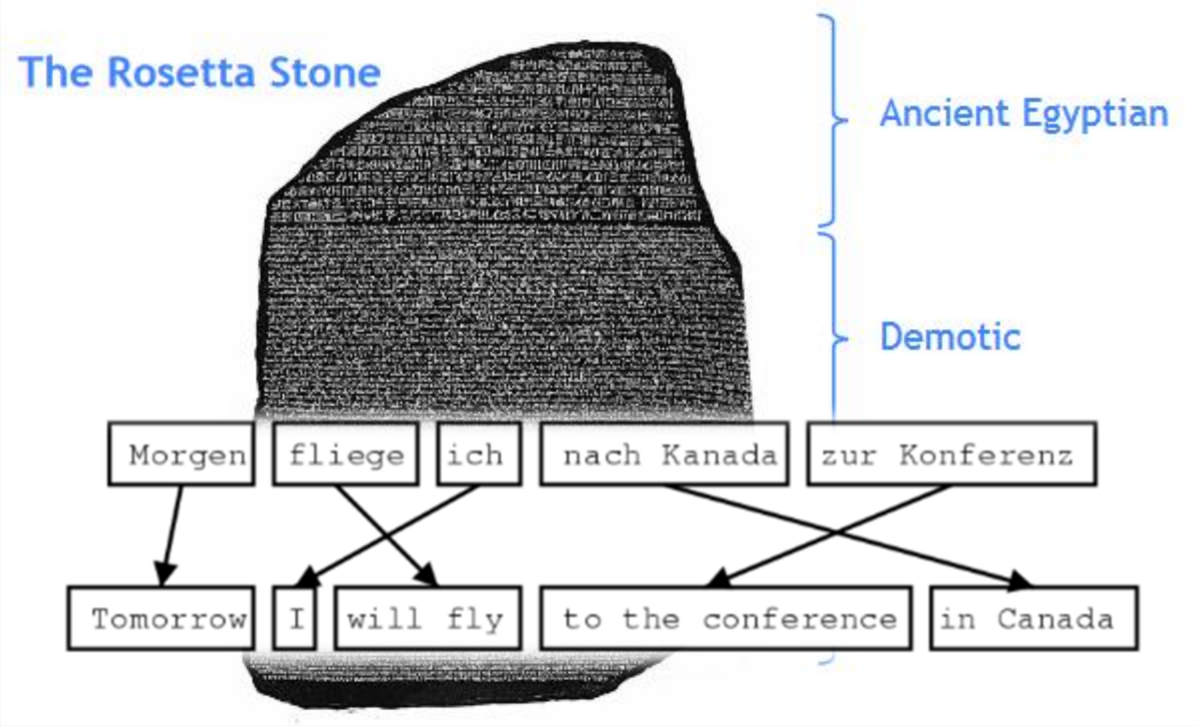
Models how to write good English (*fluency*). Learnt from monolingual data.

https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Statistical Machine Translation

- How to learn this translation model? $P(x|y)$
  - Parallel data

  - Smaller tasks:
    - Alignment $P(x, a|y)$



https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

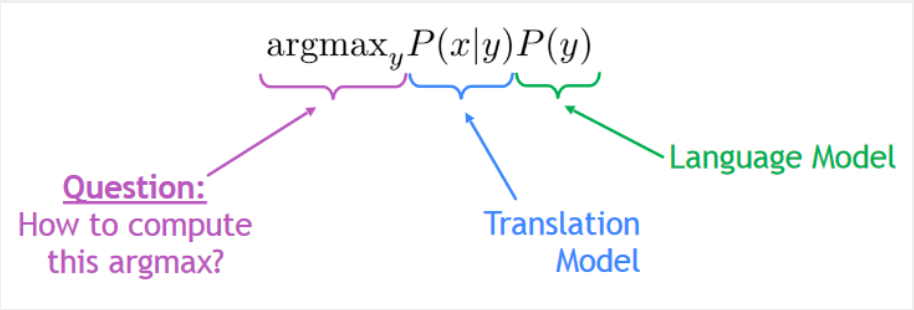
Statistical Machine Translation

- How to learn this translation model? $P(x|y)$
  - Parallel data
  - Smaller tasks:
    - Alignment $P(x, a|y)$
  - Decoding

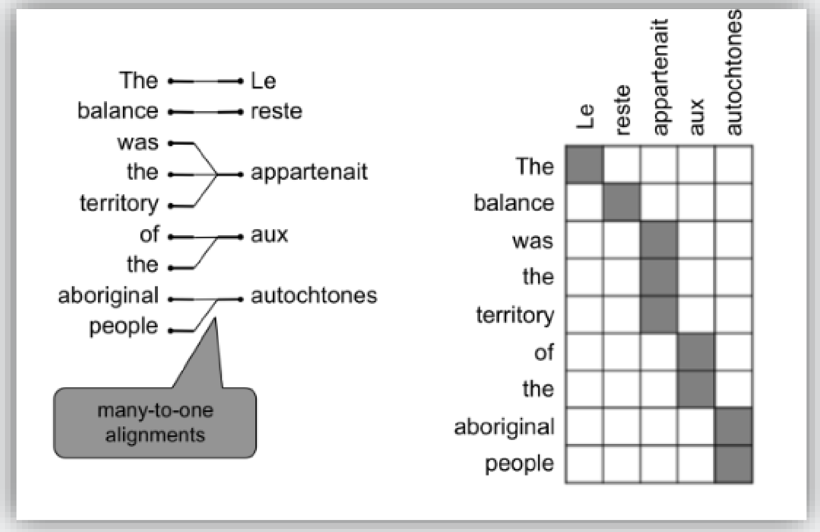
$$\underbrace{\text{argmax}_y}_{\text{Question: How to compute this argmax?}} \underbrace{P(x|y)}_{\text{Translation Model}} \underbrace{P(y)}_{\text{Language Model}}$$

https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Statistical Machine Translation



- Complex systems
- Hundreds of details
- Separately-designed subcomponents
- Lots of feature engineering!
- Compiling and maintaining extra resources
- Lots of human effort
- For each language pair!



- Now – how much of these did you have to do in the Neural Machine Translation shown before?

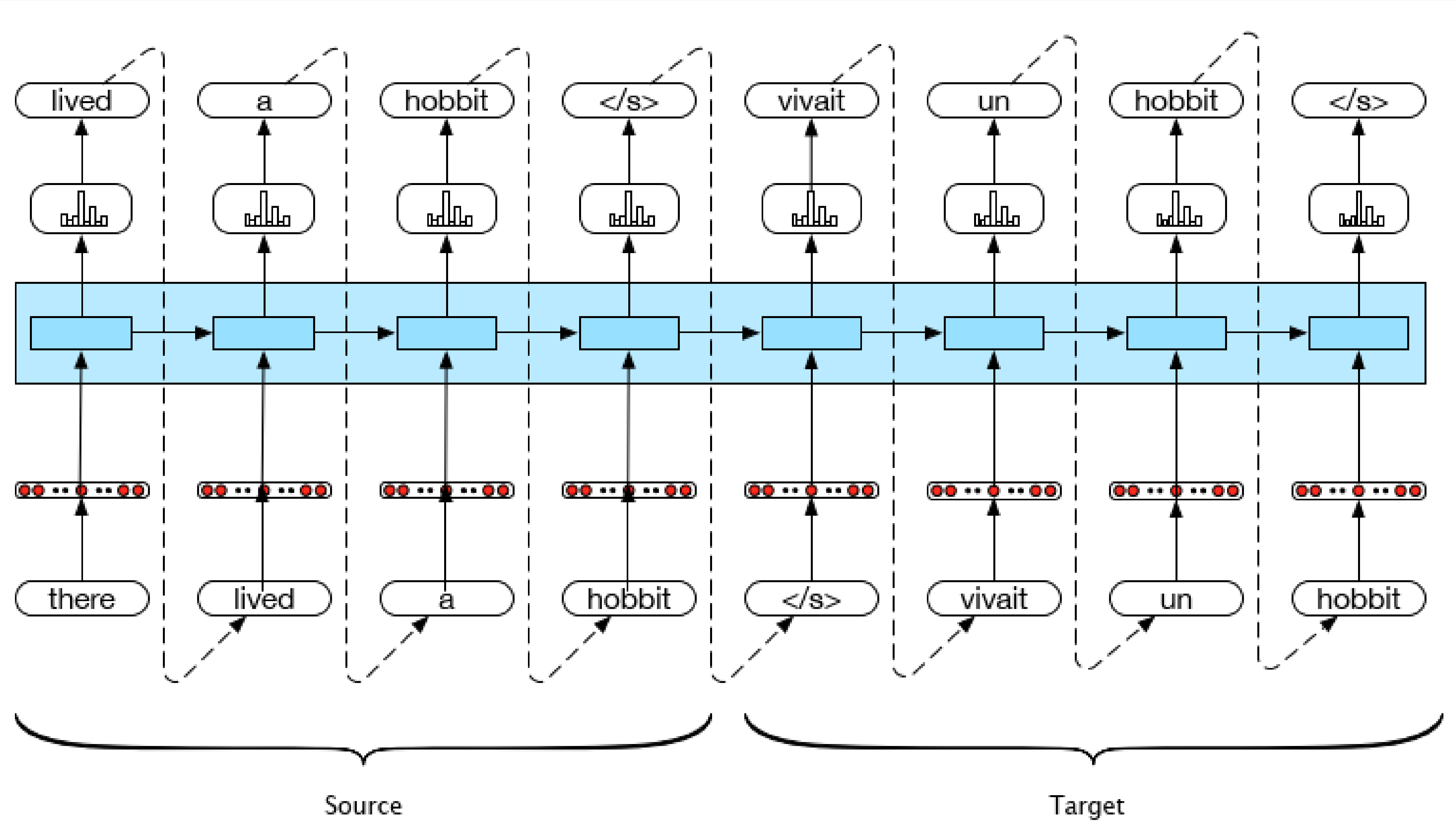
https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Encoder-Decoders

(aka. Sequence-to-sequence Models)

Neural Machine Translation: Encoder-Decoder Model

Simple RNN,
LSTM, GRU, …

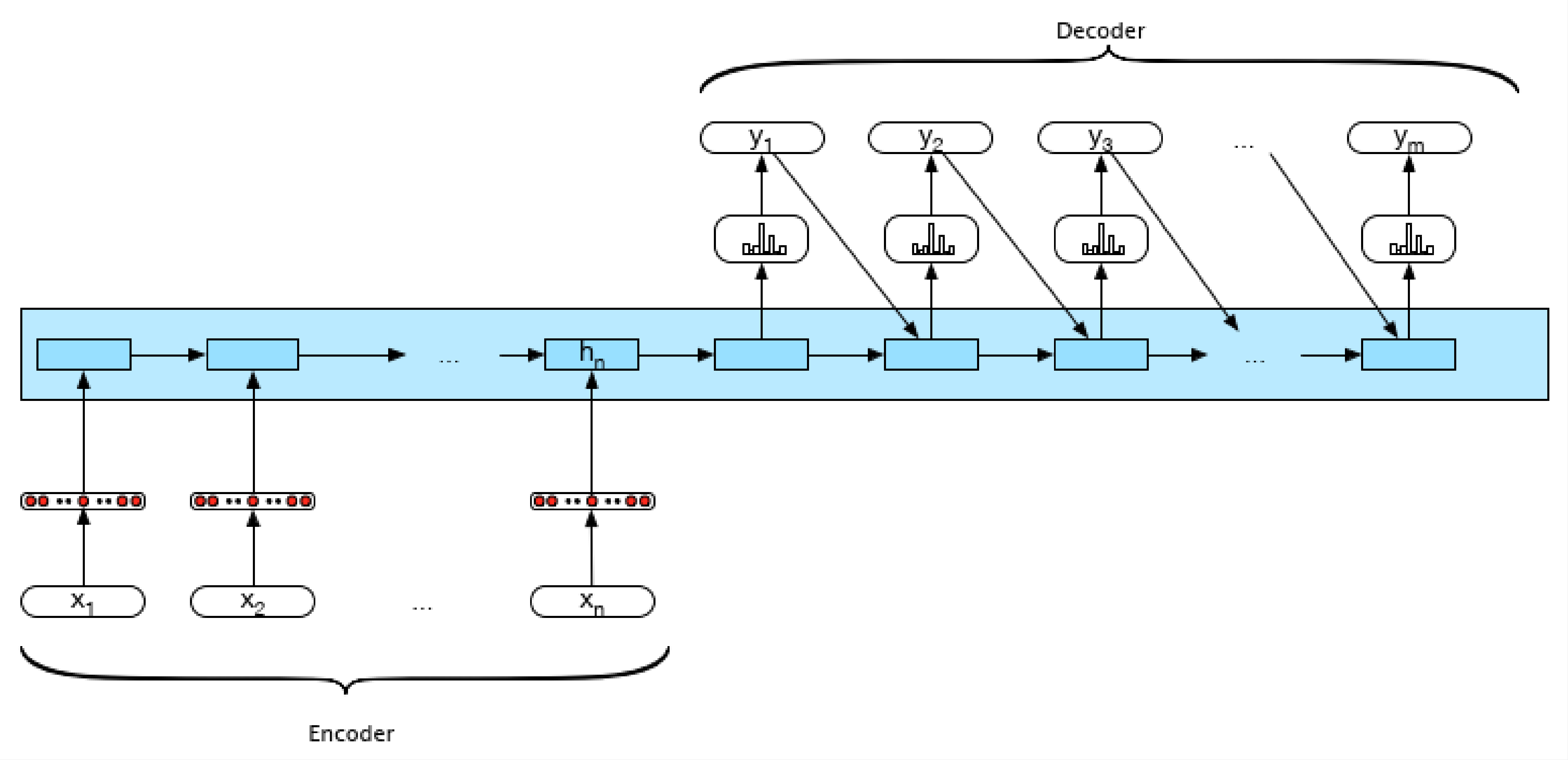


TU WIEN Informatics

Neural Machine Translation: Encoder-Decoder Model

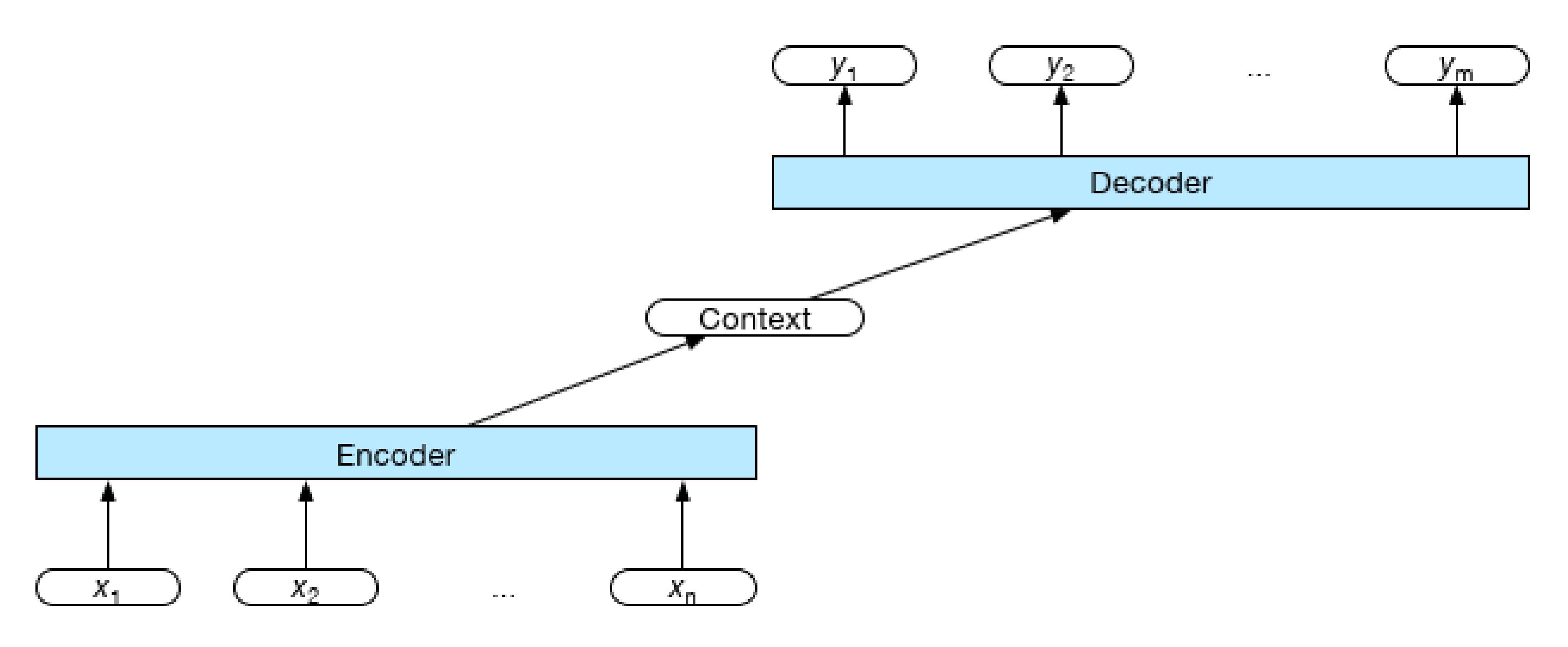
Simple RNN,

LSTM, GRU, ..



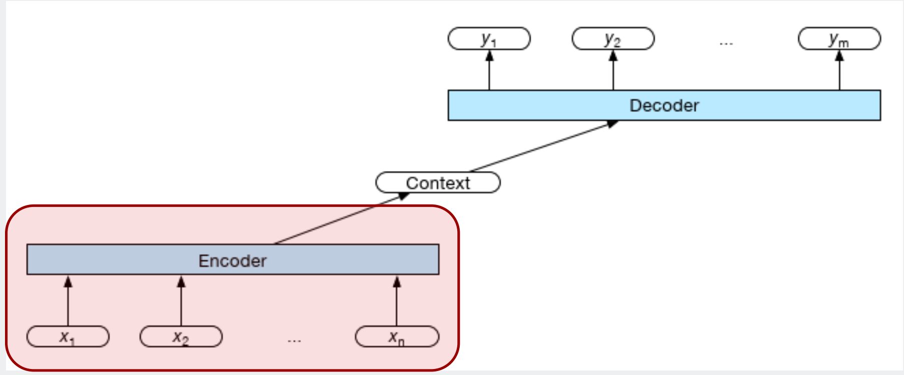
Neural Machine Translation: Encoder-Decoder Model

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



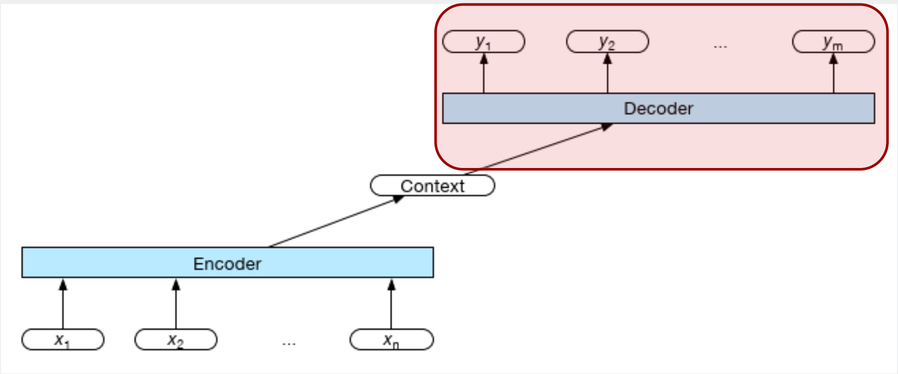
Encoder

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



Decoder

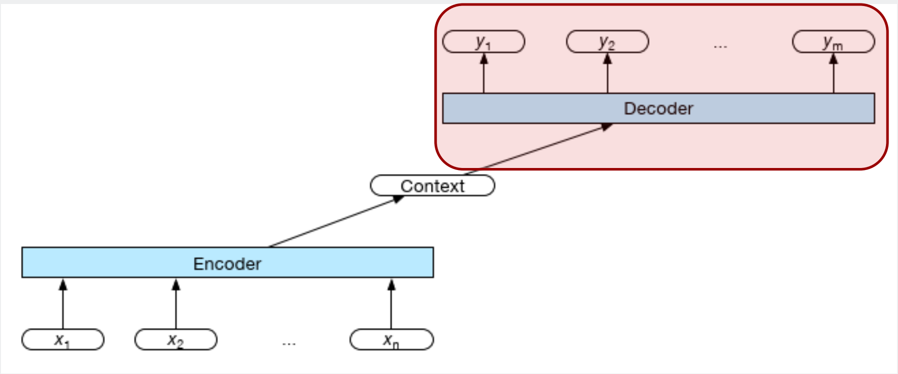
- Autoregressive generation
- Until </s> is generated
- LSTM, GRU



$$
\begin{aligned}
c &= h_n^e \\
h_0^d &= c
\end{aligned}
$$

$$
\begin{aligned}
h_t^d &= g(\hat{y}_{t-1}, h_{t-1}^d) \\
z_t &= f(h_t^d) \\
y_t &= \mathrm{softmax}(z_t)
\end{aligned}
$$

Decoder



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

$$
\begin{aligned}
c &= h_n^e \\
h_0^d &= c
\end{aligned}
$$

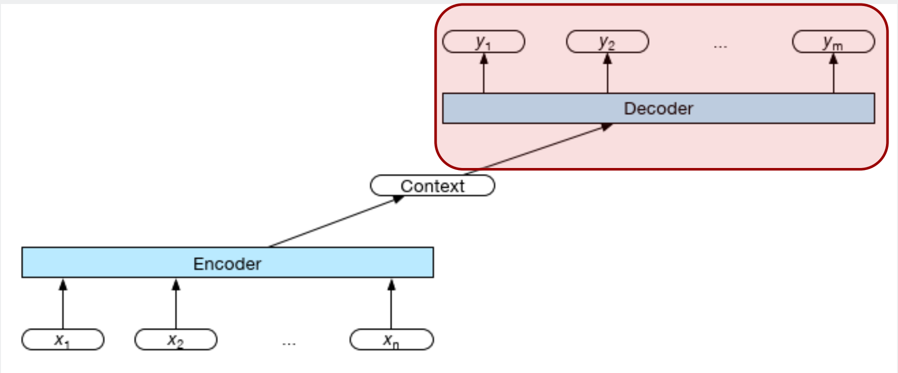
$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$

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

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

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

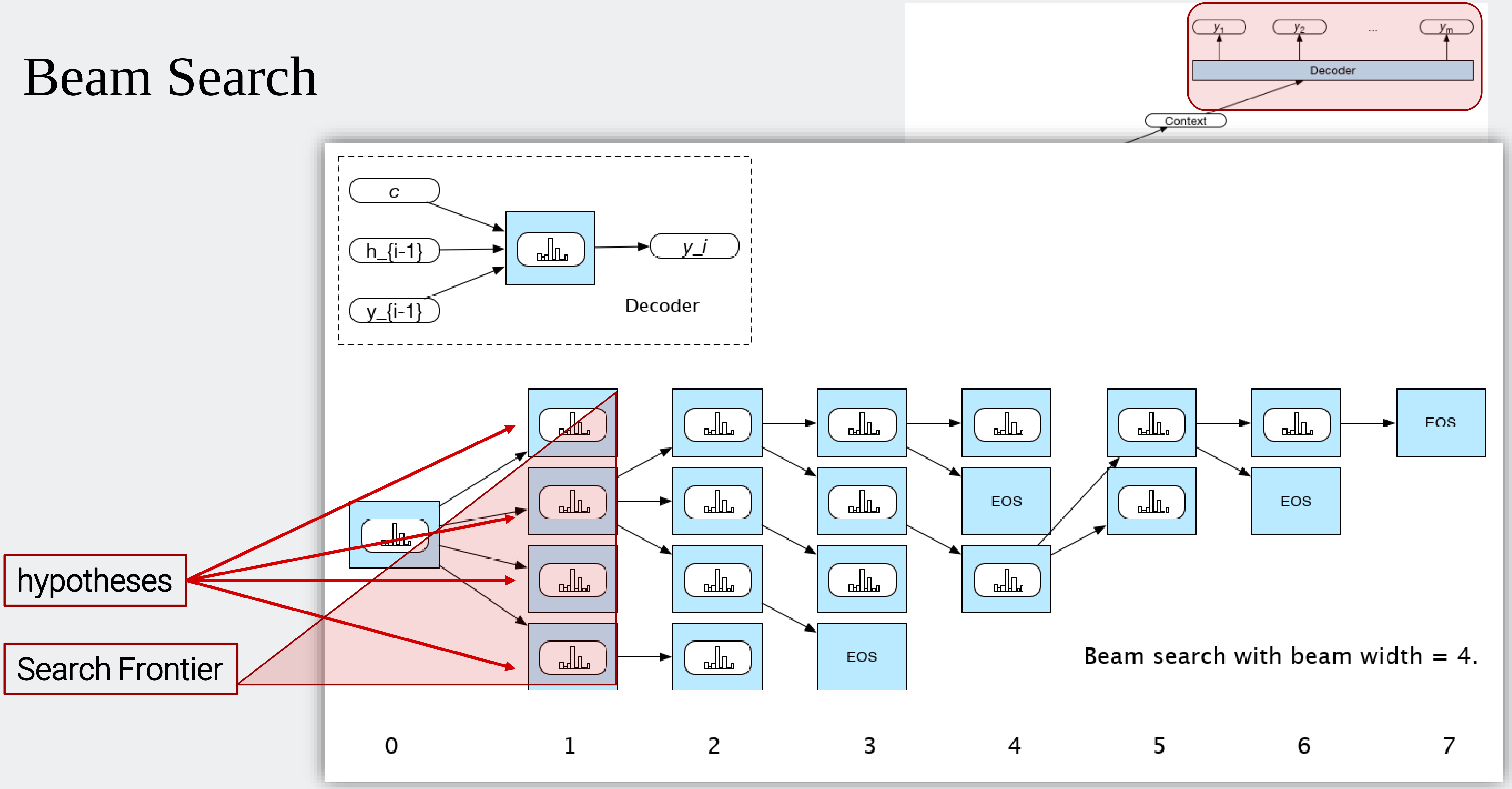
Beam Search



- Large search space
- Alternative: heuristic method, systematic exploration
- By controlling the exponential growth of the search space

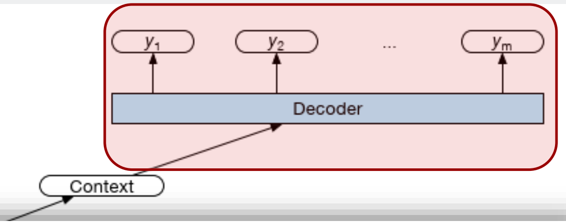
- How: combine breadth first with a heuristic filter
  - Score the options
  - Prune the search space

Beam Search



y_1
y_2
...
y_m
Decoder
Context
c
h_{i-1}
y_{i-1}
y_i
Decoder
hypotheses
Search Frontier
EOS
EOS
EOS
EOS
EOS
Beam search with beam width = 4.
0
1
2
3
4
5
6
7

Beam Search

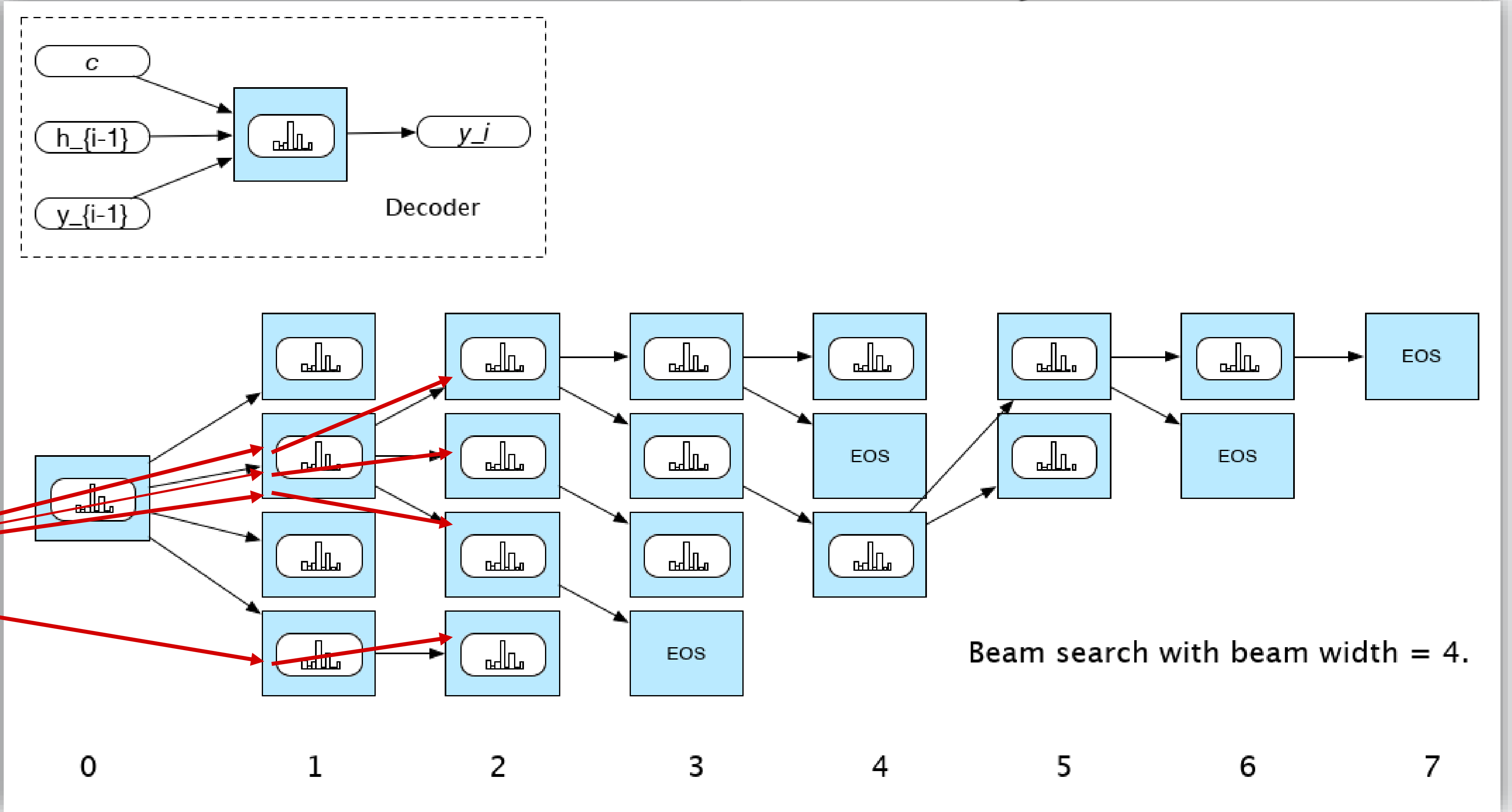


Scoring:

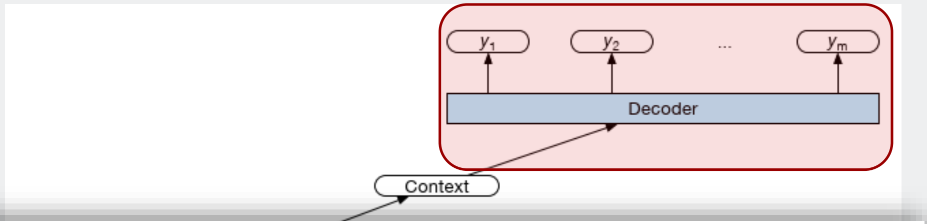
$$P(y_i|y_{<i})$$

hypotheses

Search Frontier



Beam Search

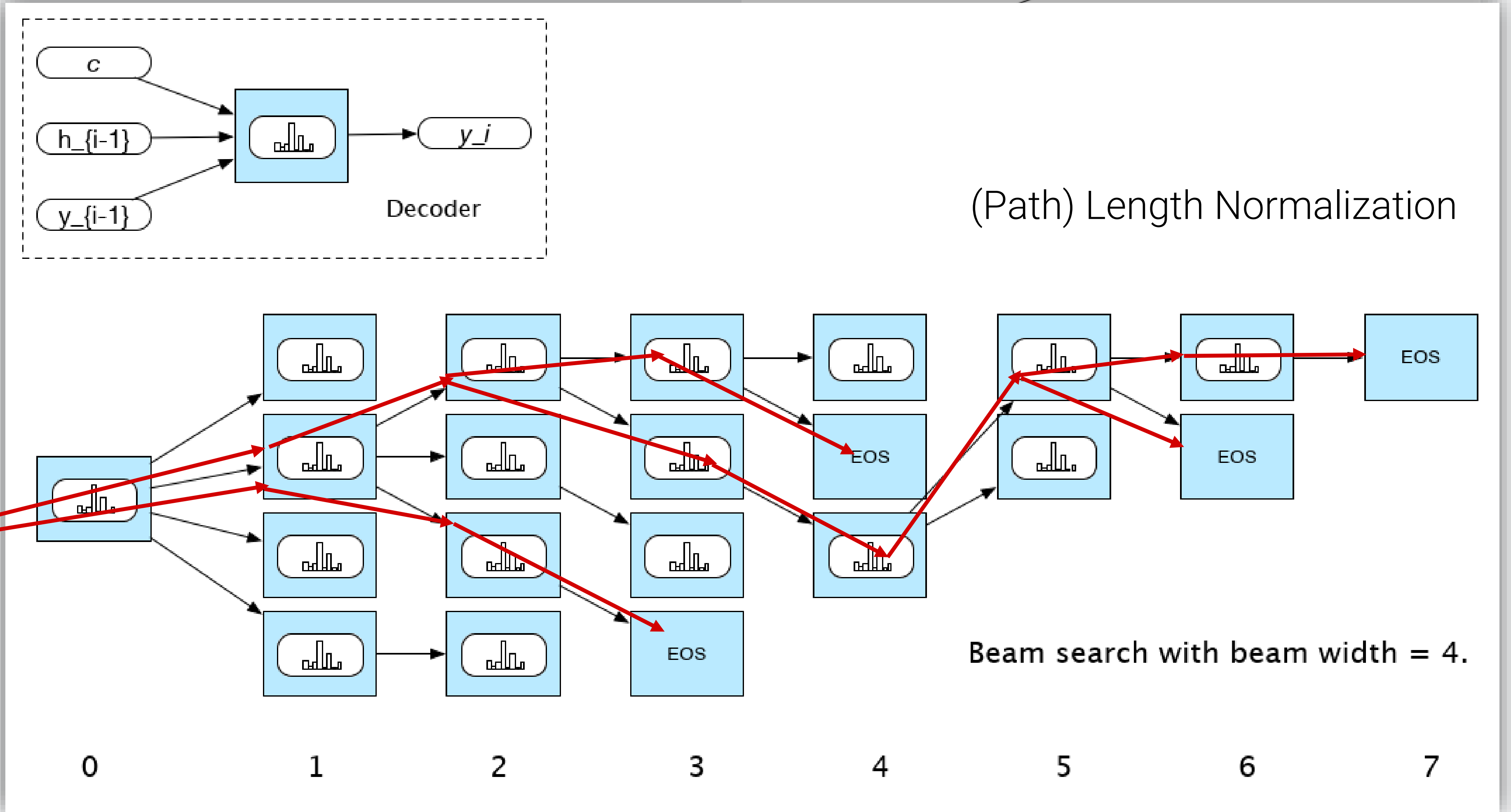


Scoring:

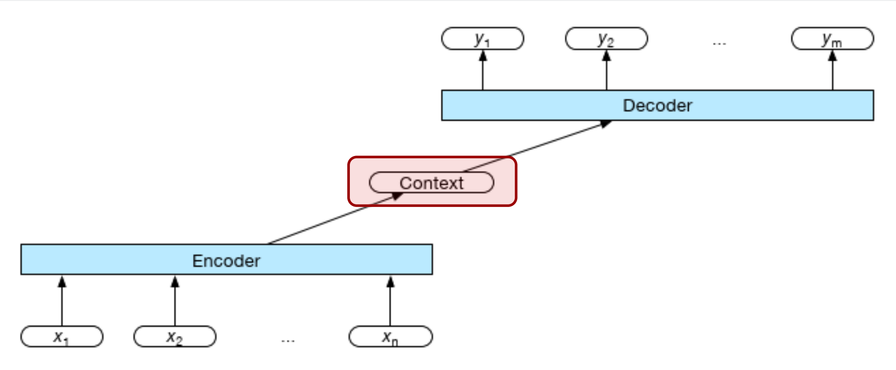
$$P(y_i|y_{<i})$$

hypotheses

Search Frontier



Context



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

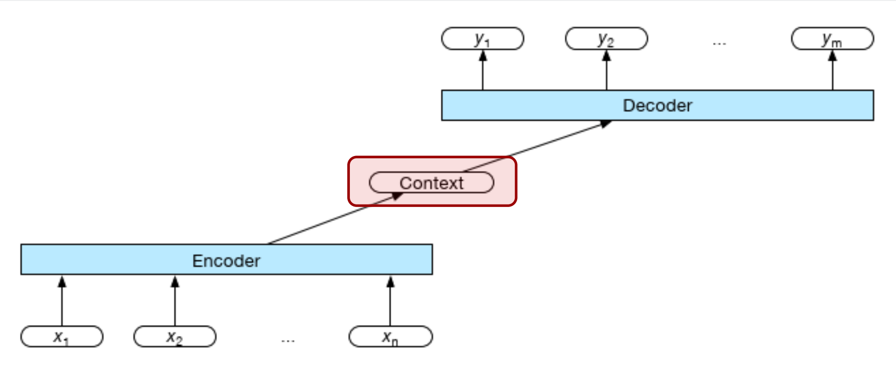
$$c = f(h_1^n)$$

- 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.
- 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

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

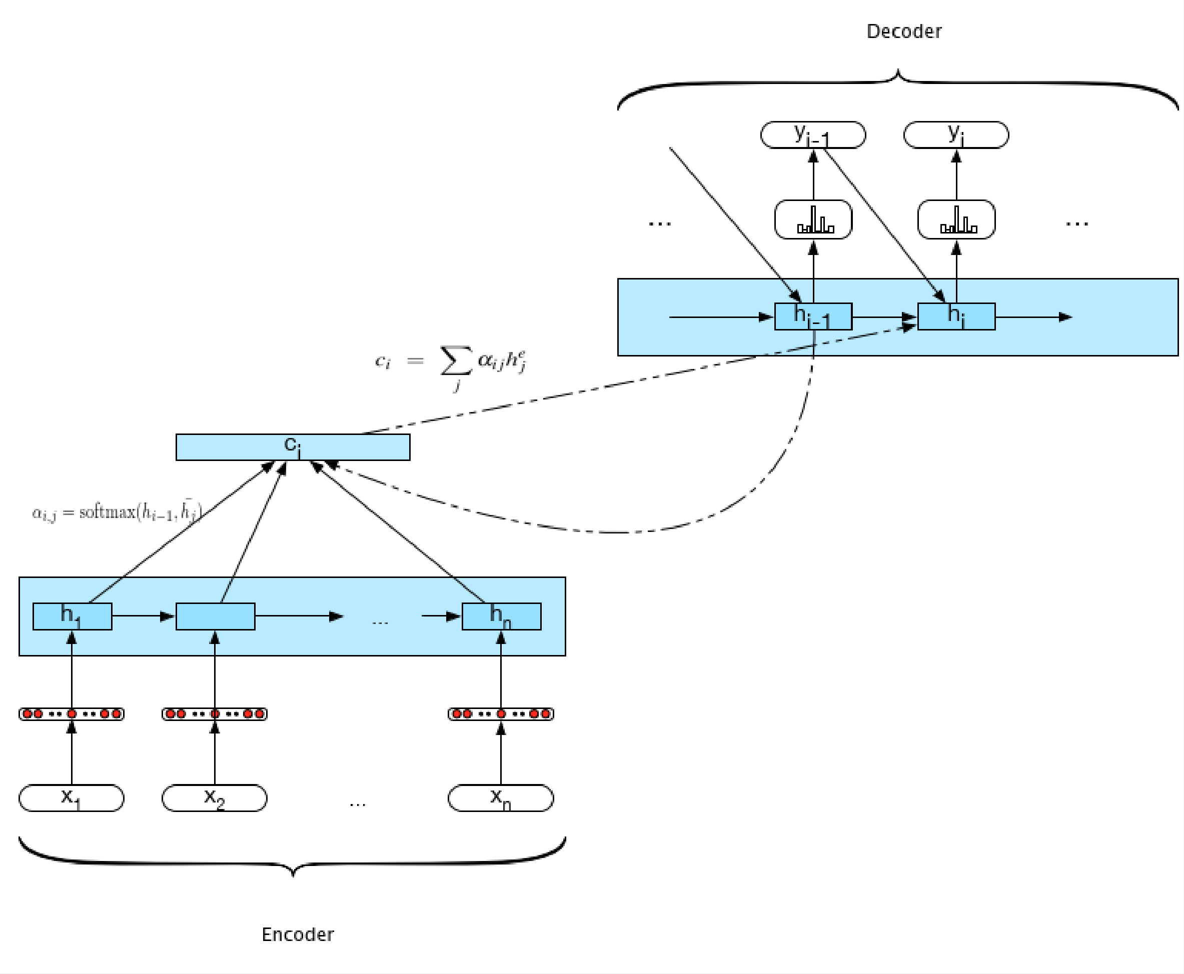
$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

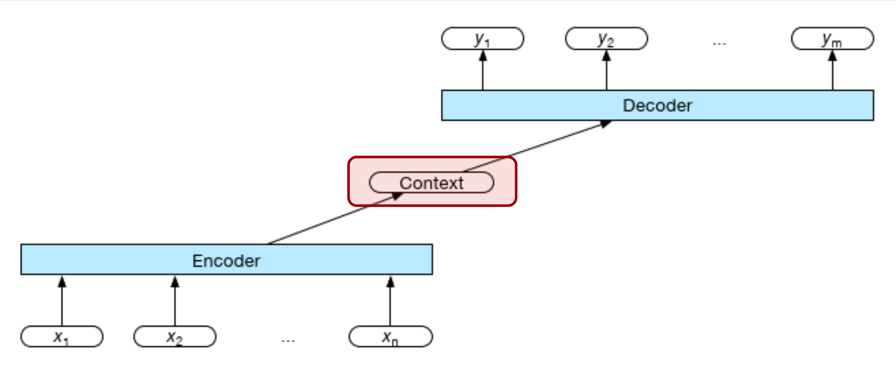
$$score(h_{i-1}^d, h_j^e) = h_{t-1}^d W_s h_j^e$$

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

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

Attention





$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c_i)$$

$$score(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

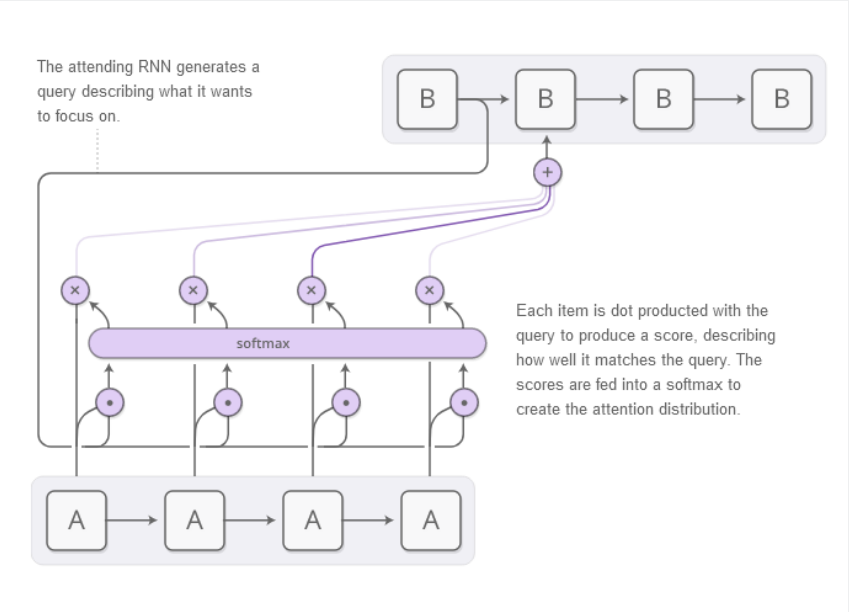
$$score(h^d_{i-1}, h^e_j) = h^d_{t-1} W_s h^e_j$$

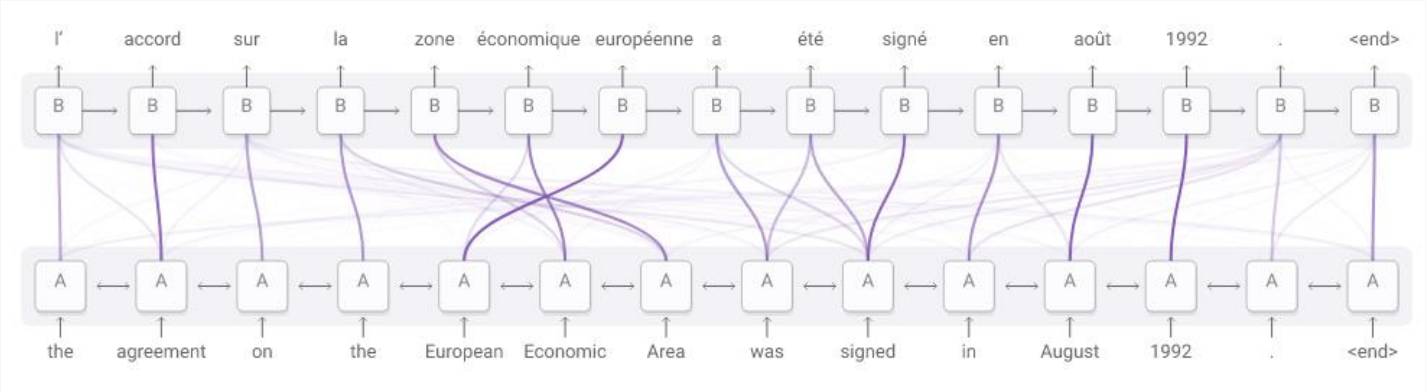
$$\alpha_{ij} = \mathrm{softmax}(score(h^d_{i-1}, h^e_j)\ \forall j \in e)$$

$$c_i = \sum_j \alpha_{ij} h^e_j$$

Attention

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





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/