Deep Learning

COMP 6721 Introduction of AI

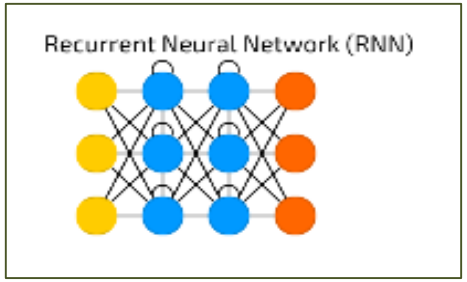
Topics

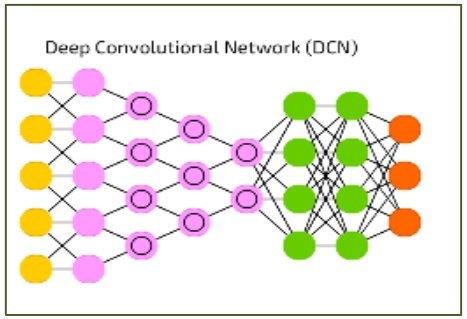
- Deep learning for NLP
- RNN
- Training RNNs
- Training Problems
- RNN Architectures
- Deep RNNs

Deep Learning for NLP

Deep learning models for NLP use:

- Vector representation of words
  - i.e., word embeddings
- Neural network structures
  - Recurrent Neural Networks (RNNs)
  - Recursive Neural Networks
  - …





Word Embeddings

- To do NLP with neural networks, words need to be represented as vectors
- Traditional approach: "one hot vector"
  - Binary vector
  - Length = | vocabulary |
  - 1 in the position of the word id, the rest are 0
  - [0, 0, 0, 1, 0, 0, 0, . . .]
- However, this does not represent word meaning ;-(
- Similar words such as *python* and *ruby* should have similar vector representations
- However, similarity/distance between all "one hot vectors" is the same

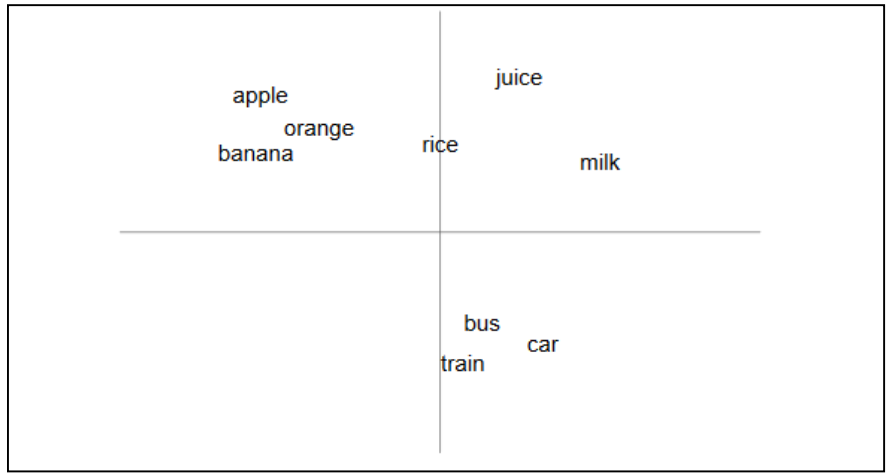
Word Embeddings

- We would like:
  - cat/kitty/dog ... to have similar representations
  - cat/orange/train/python… to have dissimilar representations
- Word embeddings:
  - Represent each word by a vector of fixed dimensions

  (eg, n= 50 to 300)
  - Like a point in n-dimensional space

50∼300 *dim*

python	0.52	0.21	0.37	…
ruby	0.48	0.21	0.33	…
word	0.05	0.23	0.06	…



Word2Vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
- Train a classifier on a binary **prediction** task:
  - Is *w* likely to show up near "*cat*"?
- In the end, we do not actually care about this task

  - But we will take the learned weights as the word embeddings

- Use as training set readily available texts, so no need for hand-labeled supervision !

Word2Vec Models

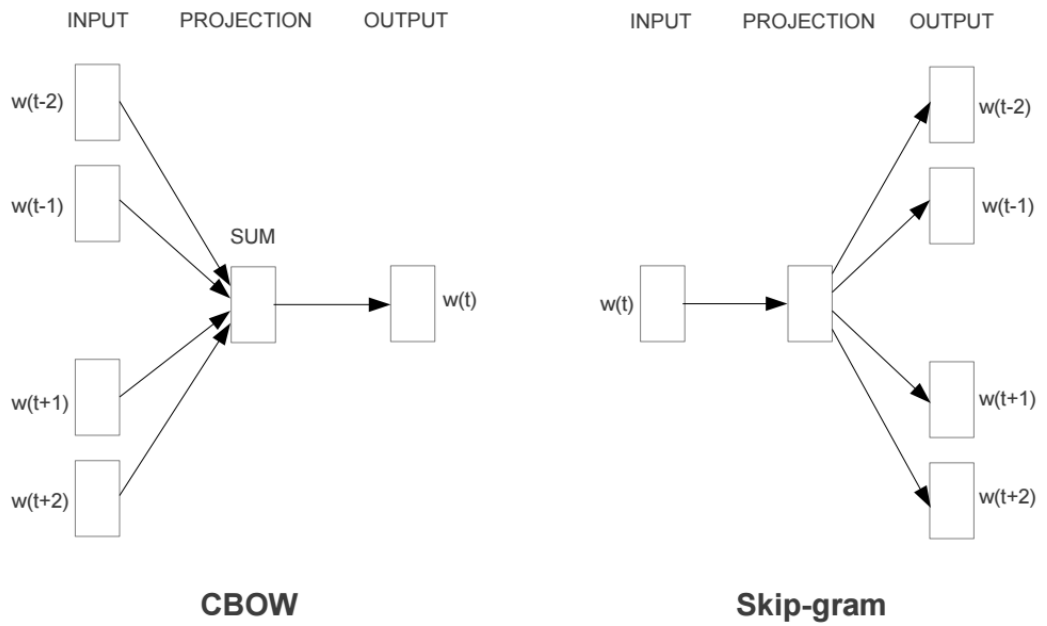
> *"A word is known by the company it keeps"* – J. R. Firth

- Basic Idea:
  1. Similar word should have similar contexts (surrounding words)
  2. So we can use the contexts to guess the word (or vice-versa).
     - The cat/kitty/dog hunts for mice.
     - The brown furry cat/kitty/dog is eating.
     - John's cat/kitty purrs.
  3. Train an ANN to guess a word given its context (or vice-versa)

Word2Vec Models

- Word2Vec has 2 models:

1. CBOW: given context words, guess the word
2. Skip-gram: given a word, guess one of its surrounding word

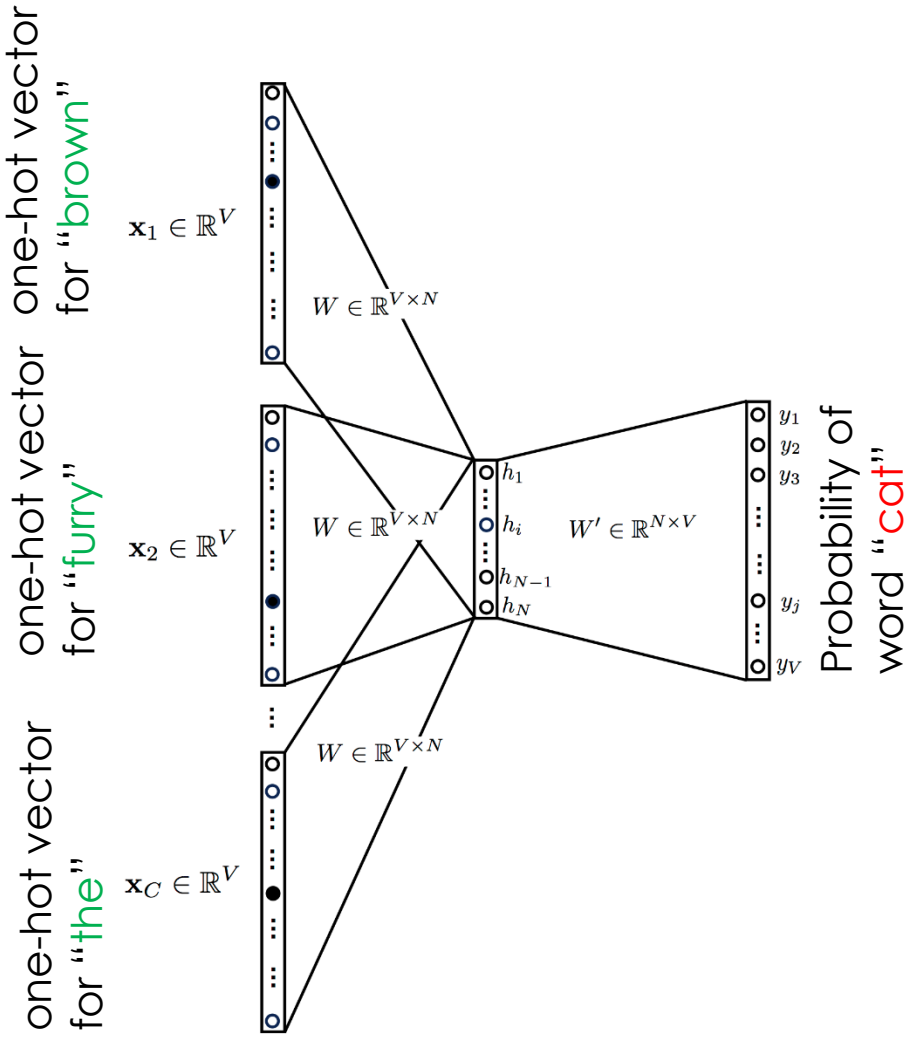


Word2Vec: CBOW Model

Uses a shallow neural network with only 3 layers:

1. One input layer
2. One hidden layer and
3. One output layer.

Goal: predict the probability of a target word (cat) given a context (brown furry chases the).



Word2Vec – Creating the Data Set

The brown furry cat chases the mouse inside the house.
…
…

Assume context words are those in +/- 2 word window

Instance	Context word -2	Context word -1	Context word +1	Context word +2	To Predict
1	the	brown	cat	chases	**furry**
2	brown	furry	chases	the	**cat**
3	furry	cat	the	mouse	**chases**
4	cat	chases	mouse	inside	**the**
5	chases	the	inside	the	**mouse**
6	the	mouse	the	house	**inside**

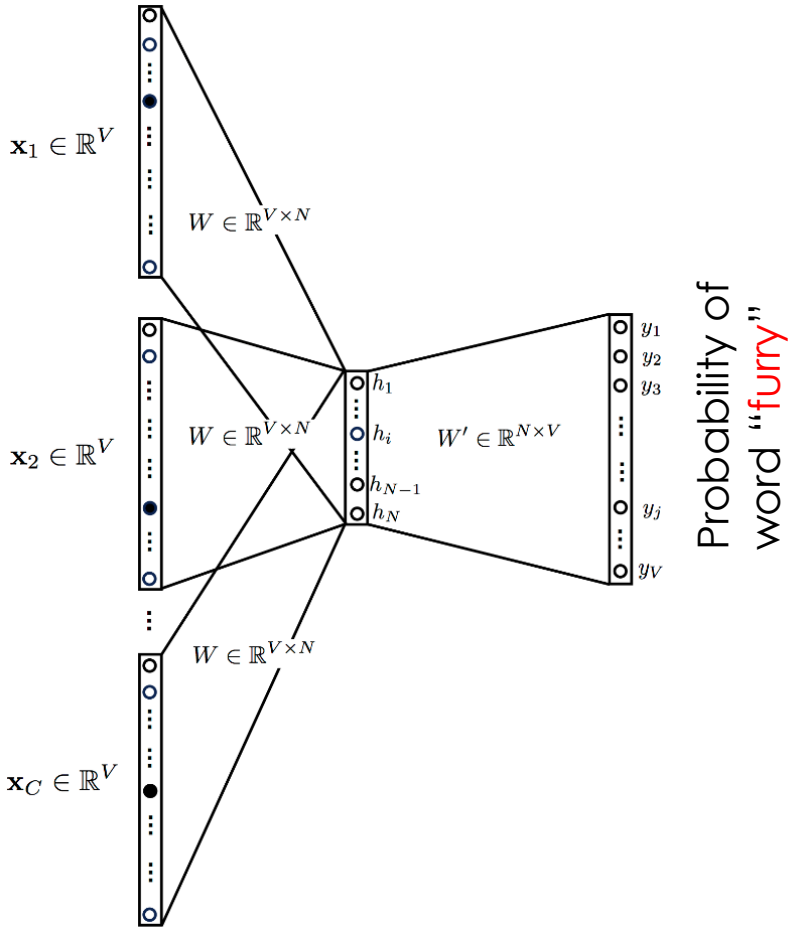
Word2Vec – Input to the Network

V = size of vocabulary
N = size of the embedding that we want
(i.e. number of neurons in the hidden layer)
C = size of context (2 words before + 2 after)

V

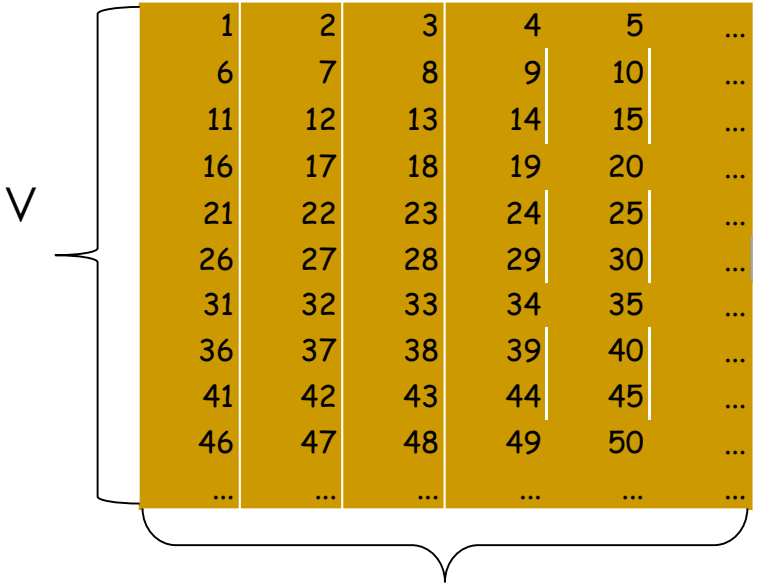
Instance	Input word											
1	Context -2	the	0	1	0	0	0	0	0	0	0	…
1	Context -1	brown	0	0	0	0	0	1	0	0	0	…
1	Context +1	cat	1	0	0	0	0	0	0	0	0	…
1	Context +2	chases	0	0	0	1	0	0	0	0	0	

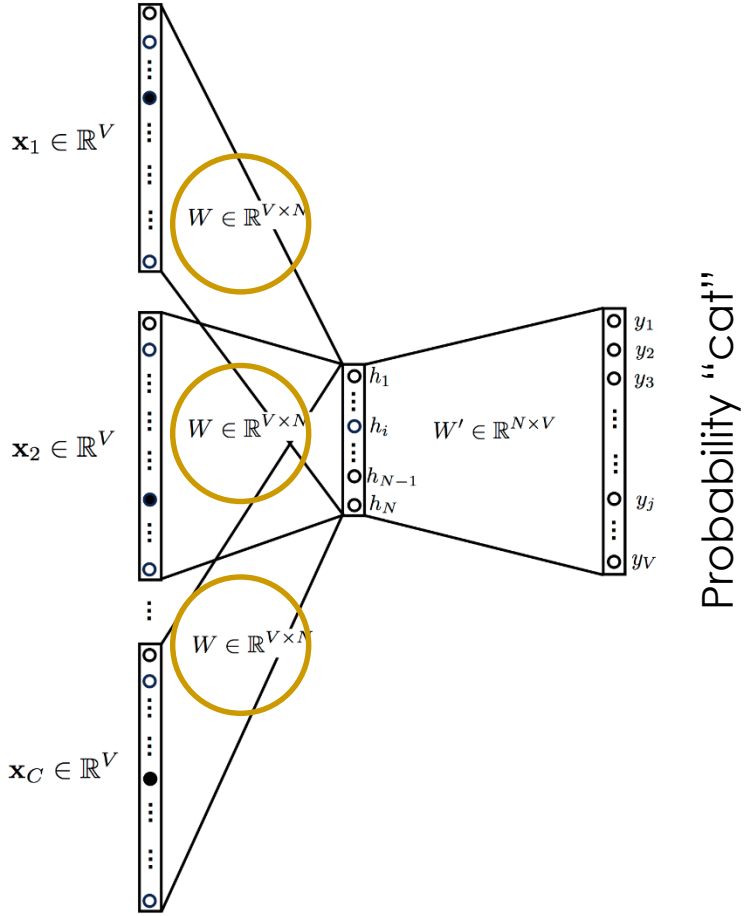
Instance	Input word											
2	Context -2	brown	0	0	0	0	0	1	0	0	0	…
2	Context -1	furry	0	0	0	0	1	0	0	0	0	…
2	Context +1	chases	0	0	0	0	0	0	0	0	1	…
	Context +2	the	0	1	0	0	0	0	0	0	0	…



Word2Vec – Weights W

- Weight Matrix W between input & hidden layer
- W is a VxN matrix…
- Initially random but modified via backpropagation





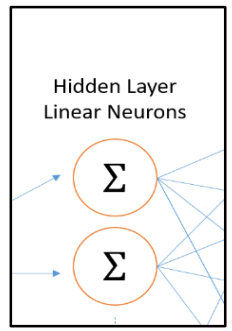
Word2Vec - Feedforward

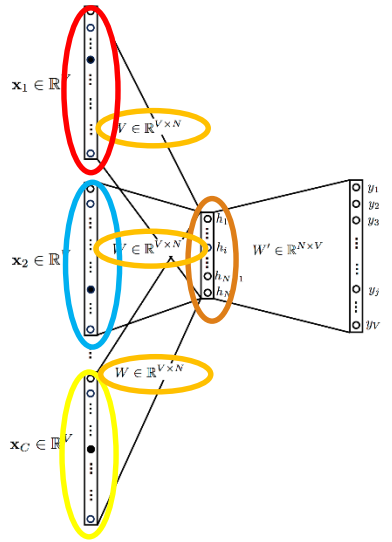
1. Calculate the output of each of the N hidden nodes for each context word

   Note: there is no activation function.

   Output is just the dot product

2. Then take the average





C	instance 1	Context -2	the	0	1	0	0	0	0	0	0	0	…
	instance 1	Context -1	brown	0	0	0	0	0	1	0	0	0	…
	instance 1	Context +1	cat	0	0	0	0	0	0	0	0	1	…
	instance 1	Context +2	chases	0	0	0	0	1	0	0	0	0	…

V

•

1	2	3	4	5	…
6	7	8	9	10	…
11	12	13	14	15	…
16	17	18	19	20	…
21	22	23	24	25	…
26	27	28	29	30	…
31	32	33	34	35	…
36	37	38	39	40	…
41	42	43	44	45	…
46	47	48	49	50	…
…	…	…	…	…	…

N

=

h1	h2	h3	h4	h5	…
6	7	8	9	10	…
26	27	28	29	30	…
41	42	43	44	45	…
21	22	23	24	25	…

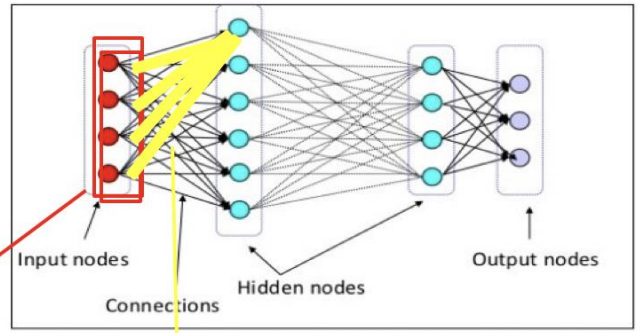
average of dot products

h1	h2	h3	h4	h5	
24	25	26	27	28	…

N

Remember this Slide?

Take dot product



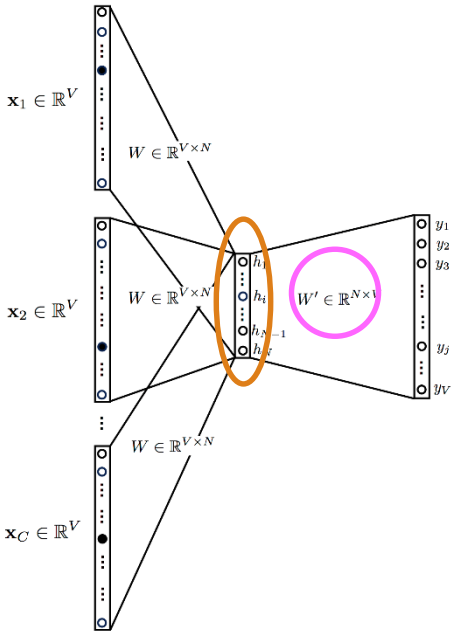
$$O_i = \text{sigmoid}\left(\sum_{j=1}^{n} w_{ji}x_j\right)$$

Assume:
n=4 input nodes
6 hidden nodes at layer 2

$$\begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} \cdot \begin{bmatrix} w_{x1,01} & w_{x2,01} & w_{x3,01} & w_{x4,01} \\ w_{x1,02} & w_{x2,02} & w_{x3,02} & w_{x4,02} \\ w_{x1,03} & w_{x2,03} & w_{x3,03} & w_{x4,03} \\ w_{x1,04} & w_{x2,04} & w_{x3,04} & w_{x4,04} \\ w_{x1,05} & w_{x2,05} & w_{x3,05} & w_{x4,05} \\ w_{x1,06} & w_{x2,06} & w_{x3,06} & w_{x4,06} \end{bmatrix} = \begin{bmatrix} (w_{x1,01})x_1 + (w_{x2,01})x_2 + (w_{x3,01})x_3 + (w_{x4,01})x_4 \\ (w_{x1,02})x_1 + (w_{x2,02})x_2 + (w_{x3,02})x_3 + (w_{x4,02})x_4 \\ (w_{x1,03})x_1 + (w_{x2,03})x_2 + (w_{x3,03})x_3 + (w_{x4,03})x_4 \\ (w_{x1,04})x_1 + (w_{x2,04})x_2 + (w_{x3,04})x_3 + (w_{x4,04})x_4 \\ (w_{x1,05})x_1 + (w_{x2,05})x_2 + (w_{x3,05})x_3 + (w_{x4,05})x_4 \\ (w_{x1,06})x_1 + (w_{x2,06})x_2 + (w_{x3,06})x_3 + (w_{x4,06})x_4 \end{bmatrix}$$

All incoming weights to hidden node O1

Word2Vec – Weights W'



- Weight Matrix W' between hidden & output layer
- W' is a NxV matrix…
- Initially random but modified via backpropagation
- Feed forward average of hidden neurons and do dot product with matrix W'

Matrix W'

average of dot products

h1	h2	h3	h4	h5	
24	25	26	27	28	…

N (size of embeddings)

•

0.70	0.13	0.14	0.60	0.16	0.17	0.60	0.19	0.20	0.21	…
0.12	0.50	0.14	0.15	0.50	0.30	0.18	0.20	0.20	0.21	…
0.17	0.40	0.60	0.25	0.26	0.50	0.35	0.37	0.30	0.55	…
0.32	0.33	0.34	0.38	0.36	0.37	0.25	0.55	0.56	0.41	…
0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.30	…
…	…	…	…	…	…	…	…	…	…	…

N

V

= output layer

y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	
43.7550	46.0750	43.1300	46.5950	44.8300	46.8250	47.2600	46.8500	45.8400	43.2200	…

V

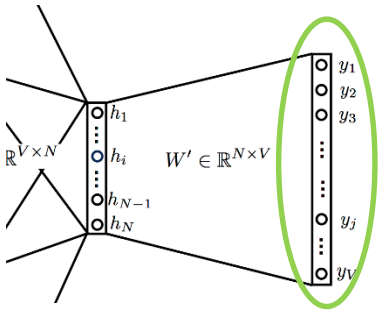
Turn dot product into probabilities

V

	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	
output layer	43.7550	46.0750	43.1300	46.5950	44.8300	46.8250	47.2600	46.8500	45.8400	43.2200	...

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

// use softmax function
// to turn the vector above into
// a vector of probabilities
// (that nicely add up to 1)



output layer	43.7550	46.0750	43.1300	46.5950	44.8300	46.8250	47.2600	46.8500	45.8400	43.2200	...	sum
e^y_i	1.01E+19	1.02E+20	5.38E+18	1.72E+20	2.95E+19	2.17E+20	3.35E+20	2.22E+20	8.09E+19	5.89E+18	...	1.180E+21
softmax(y_i)	0.0085	0.0868	0.0046	0.1459	0.0250	0.1837	0.2837	0.1883	0.0686	0.0050	...	1.0000

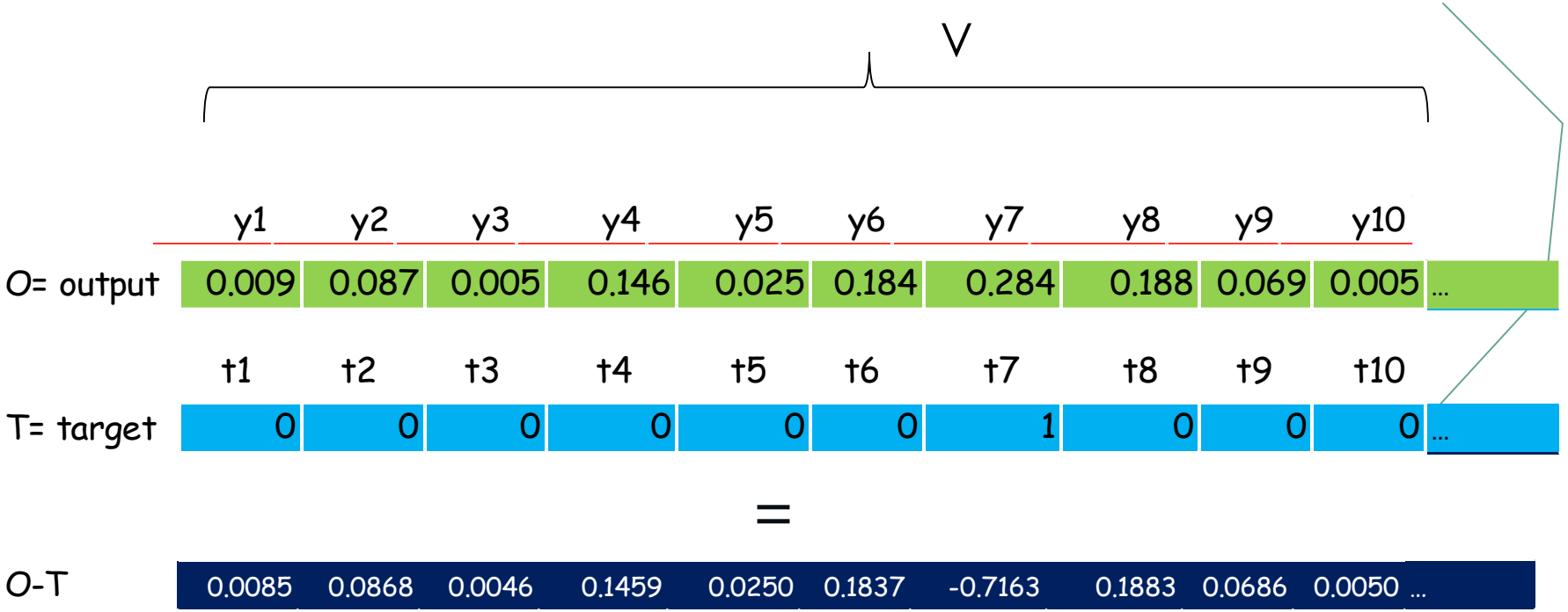
V

Compute Error of Output Layer

remember the training set:

Instance	Context word -2	Context word -1	Context word +1	Context word +2	To Predict
1	the	brown	cat	chases	**furry**
2	brown	furry	chases	the	**cat**
…	…	…	…	…	…

// target = 1 hot representation of the target (furry) in the training set

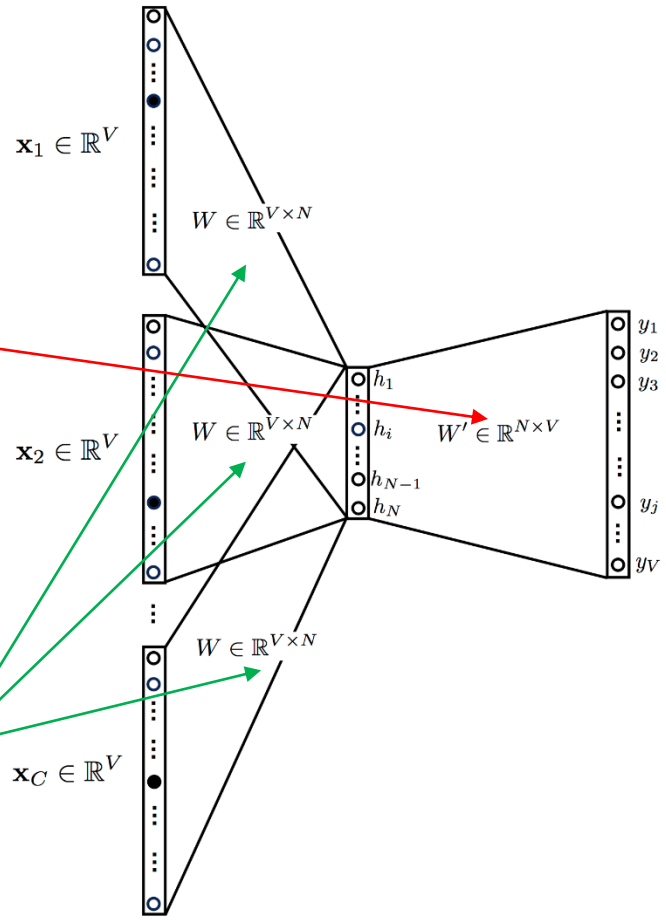


Backpropagate errors to adjust W and W'

- Adjust W' and W using backpropagation
- <after a bit of math>, we get:

$$w'_{ij} = w'_{ij} - \eta(O_j - t_j)h_i$$

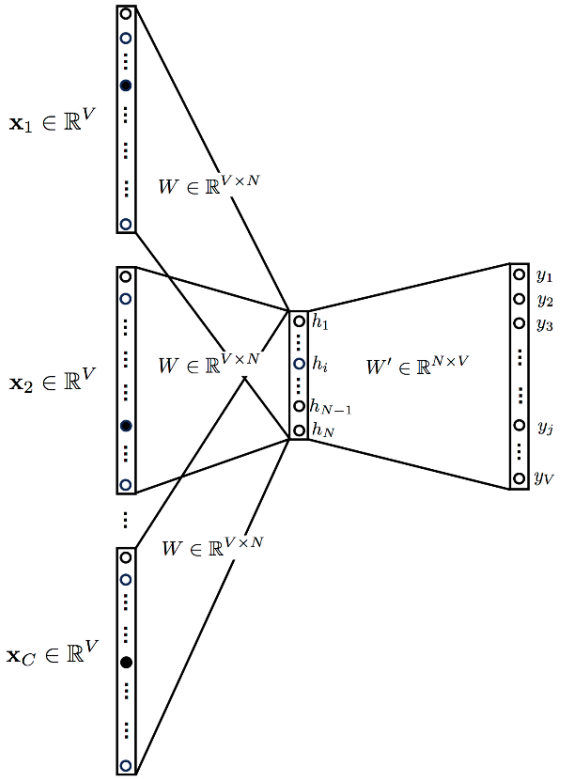
$$w_{ij} = w_{ij} - \eta\frac{1}{C}\sum_{j=1}^{V}(O_j - t_j)w'_{ij}$$



η: learning rate
C: size of context

Word2Vec – FeedForward Next Data

Instance	Context word -2	Context word -1	Context word +1	Context word +2	To Predict
~~1~~	~~the~~	~~brown~~	~~cat~~	~~chases~~	~~furry~~
2	brown	furry	chases	the	cat
3	furry	cat	the	mouse	chases
4	cat	chases	mouse	inside	the
5	chases	the	inside	the	mouse
6	the	mouse	the	house	inside



Instance	Input word										
2	Context -2	brown	0	0	0	0	0	1	0	0	0	…
2	Context -1	furry	0	0	0	0	1	0	0	0	0	…
2	Context +1	chases	0	0	0	0	0	0	0	0	1	…
1	Context +2	the	0	1	0	0	0	0	0	0	0	…

- Iterate feedforward/ backprop until error is minimized
- Trained on Google News dataset (about 100 billion words).
- See: https://code.google.com/archive/p/word2vec/

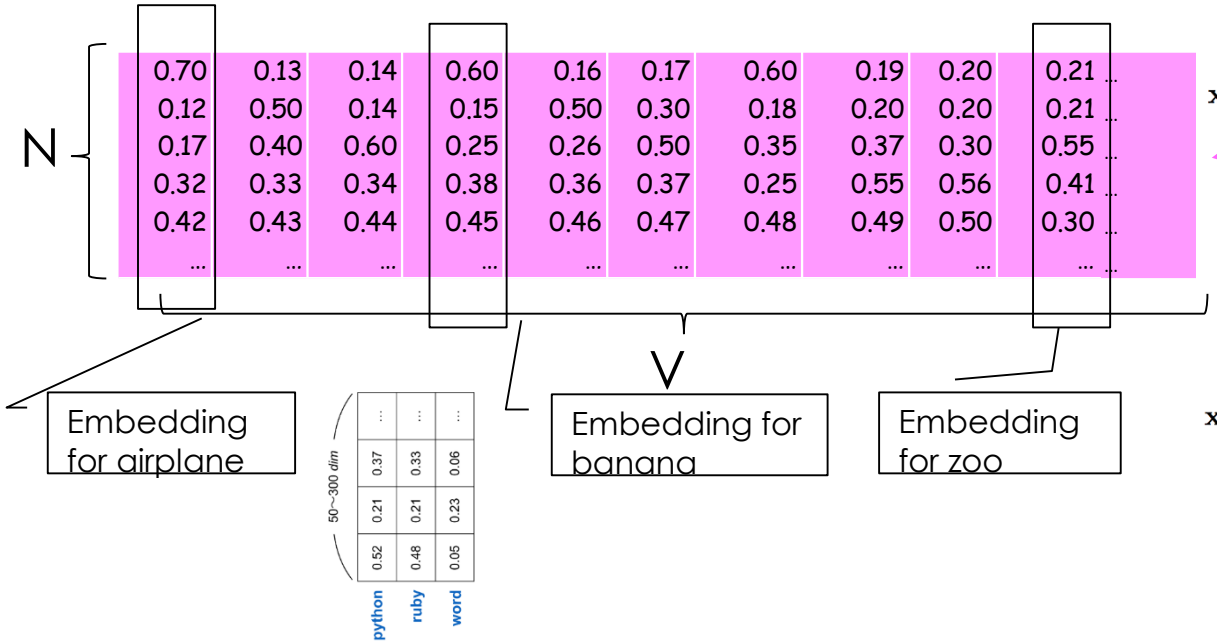
almost…

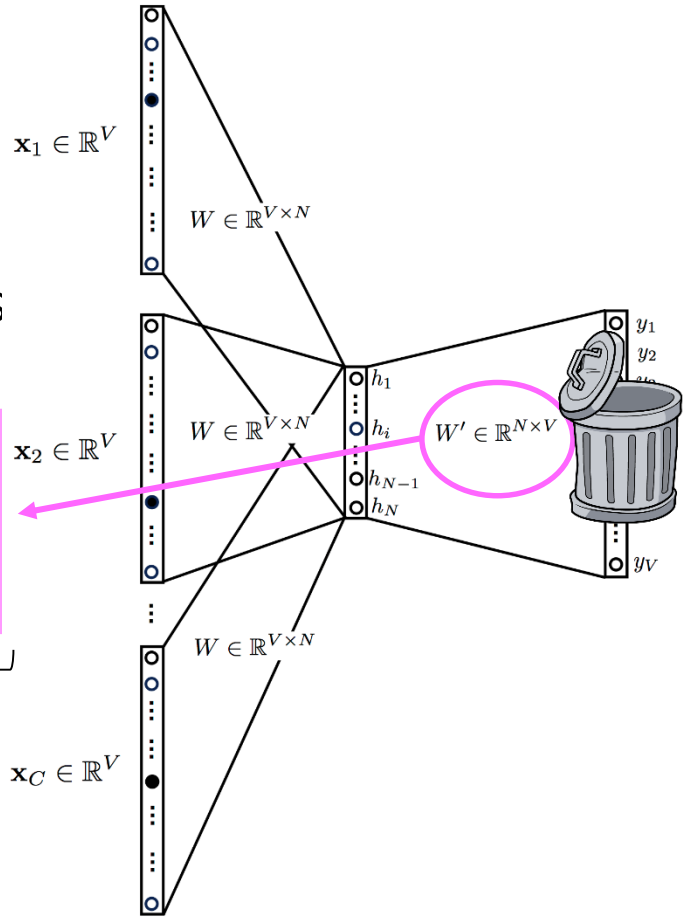
remember, we did all this to get embeddings…

I'm not leaving 'till I get my embeddings!

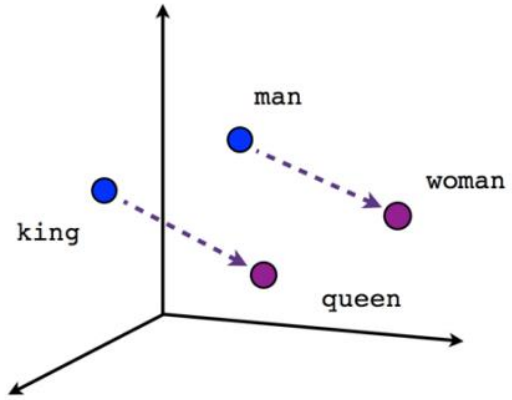
Word2Vec – Get the Embeddings

- After many iterations of feedforward, backpropagation on the entire training set …
- The classifier will be built!
- Then, we throw it away ! (yes, we do!)
- Only keep W', these are your word embeddings

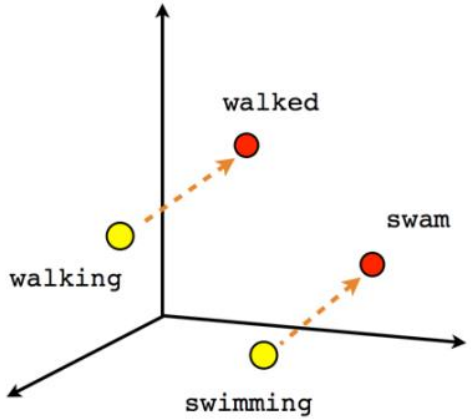




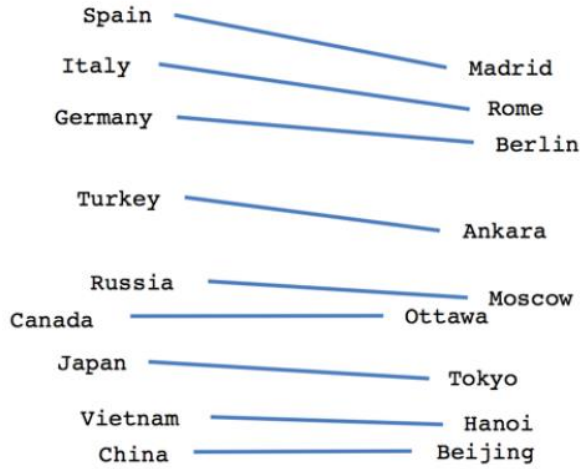
Results



Male-Female



Verb tense



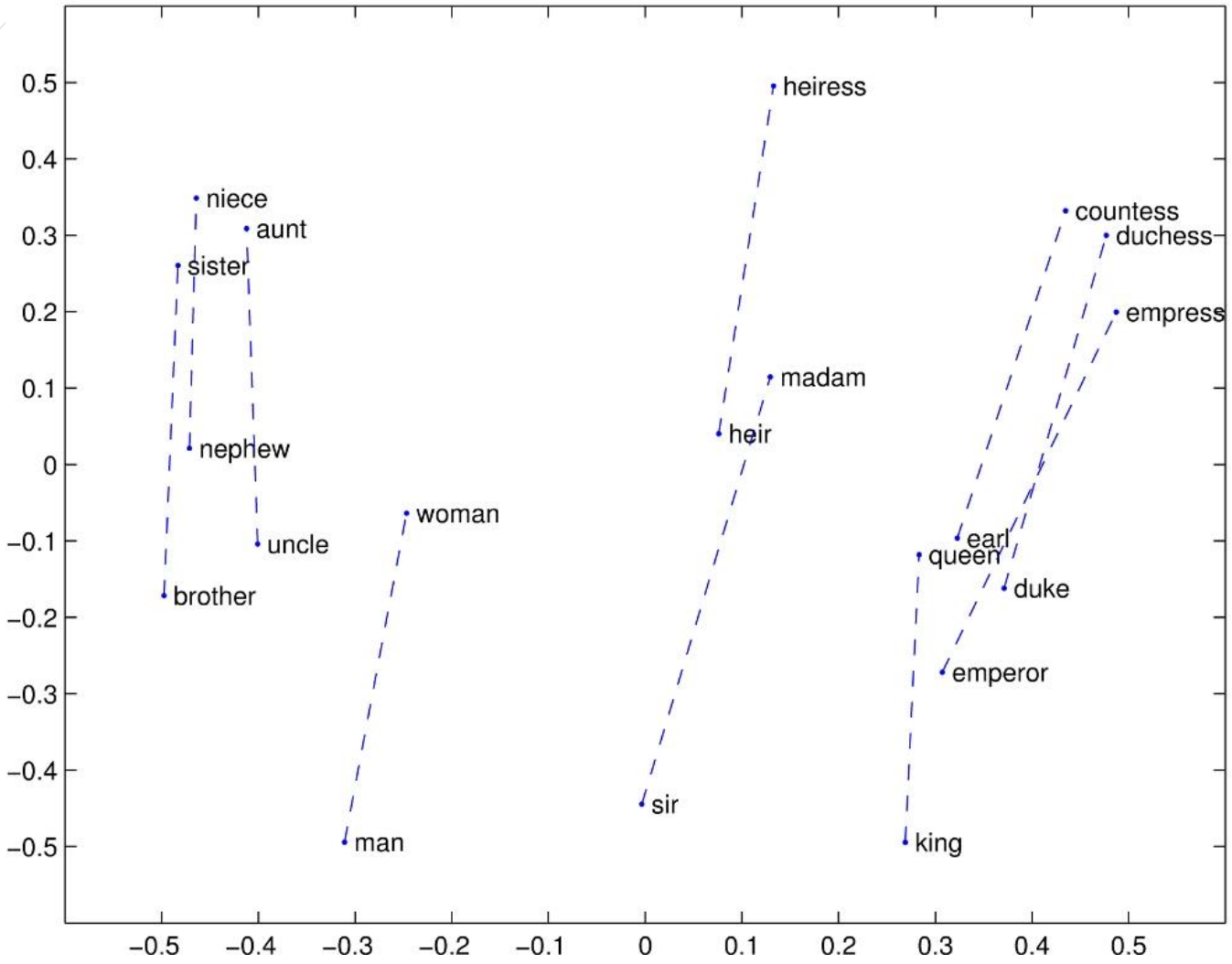
Country-Capital

vector[queen] ~ vector[king] - vector[man] + vector[woman]

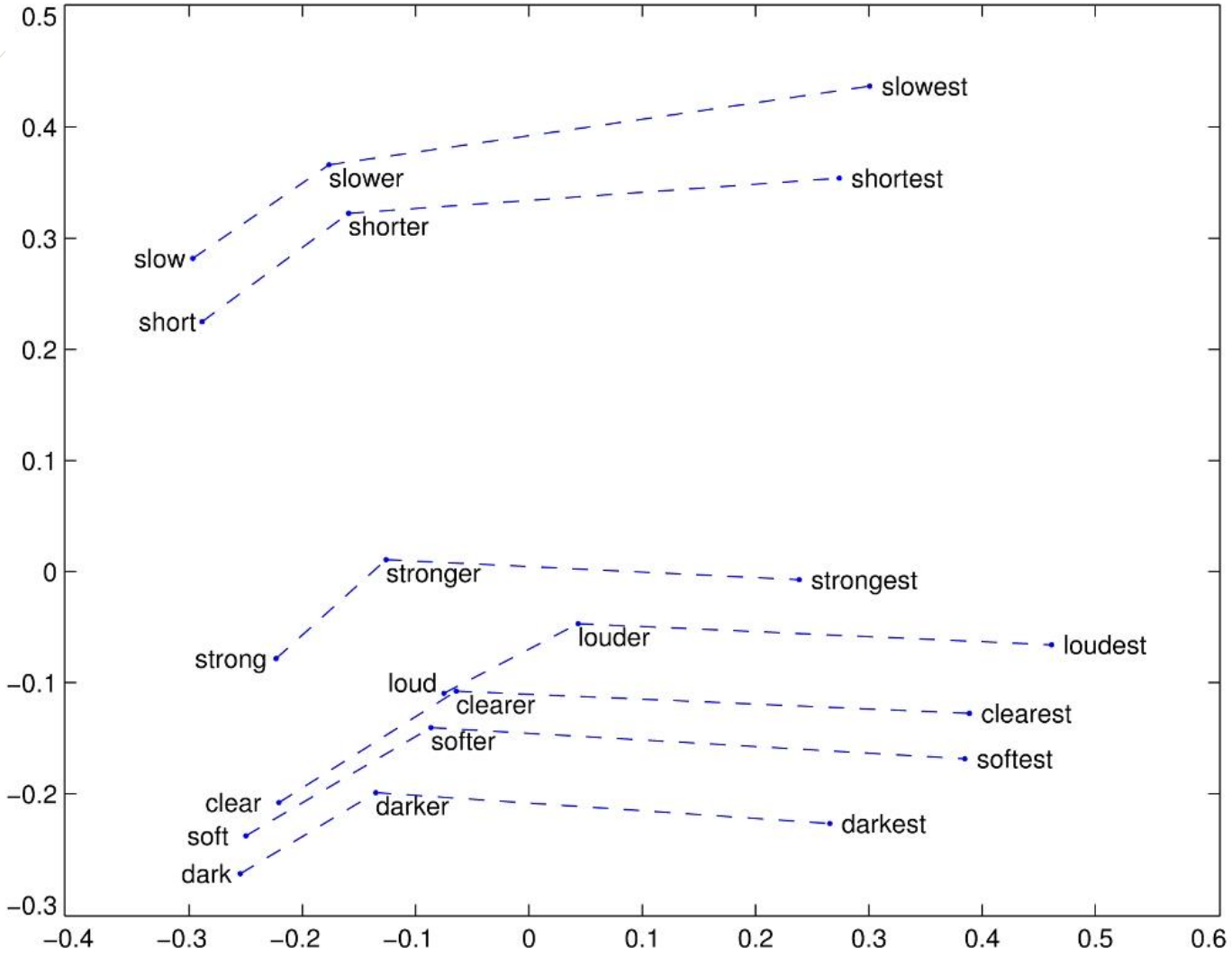
vector[swimming] ~ vector[swam] - vector[walked] + vector[walking]

vector[Rome] ~ vector[Madrid] – vector[Spain] + vector[Italy]

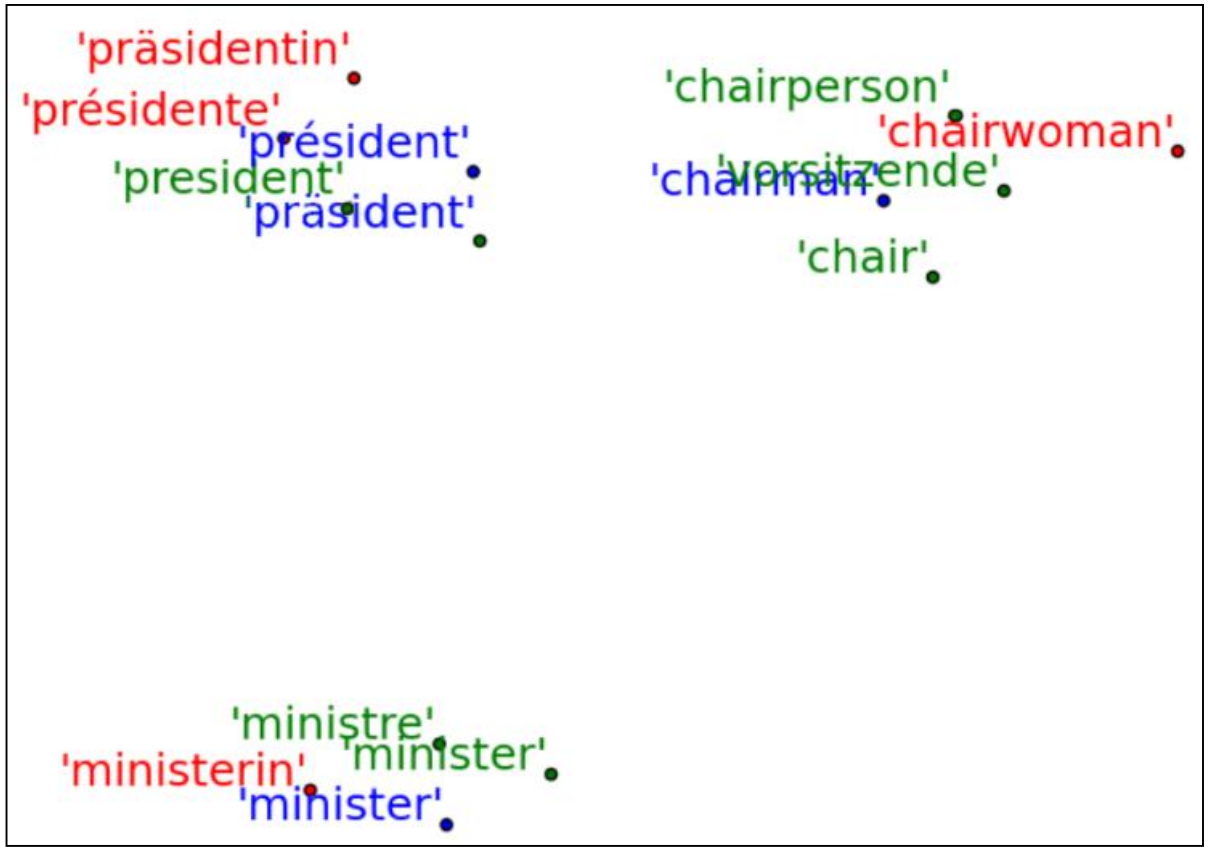
Results


heiress
niece
aunt
countess
duchess
sister
empress
madam
heir
nephew
woman
earl
queen
uncle
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5

Results


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark

Multilingual Word Embeddings

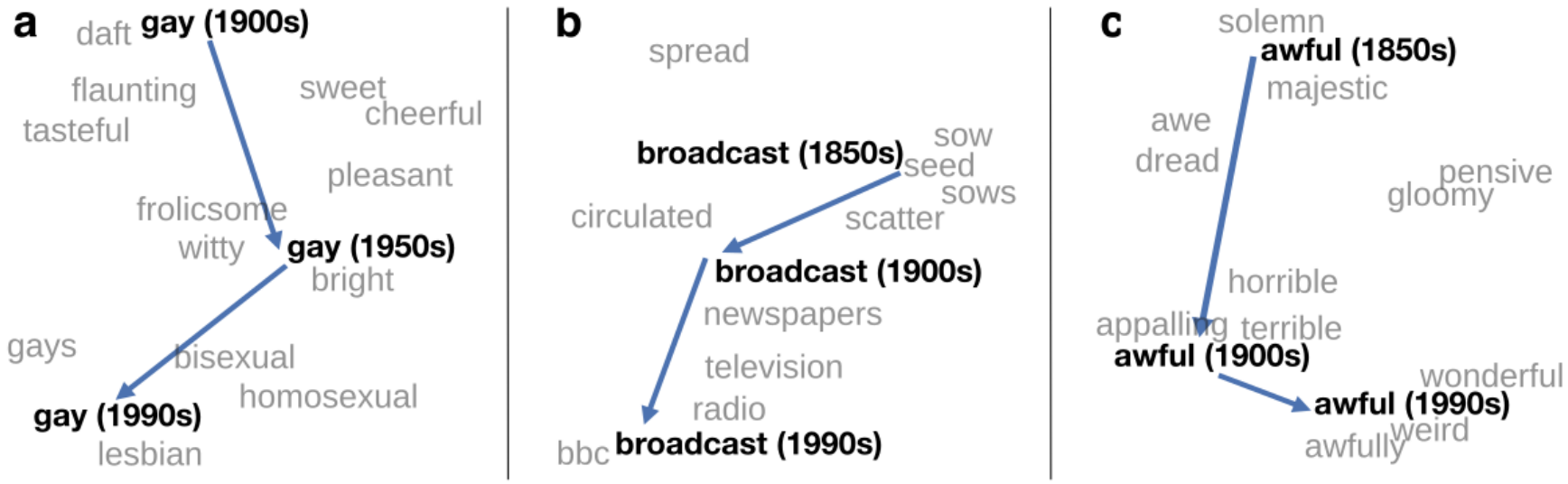


Words in different languages but with similar meanings (i.e. translations) are represented by similar vectors

Used in Machine Translation

Word History through Embeddings

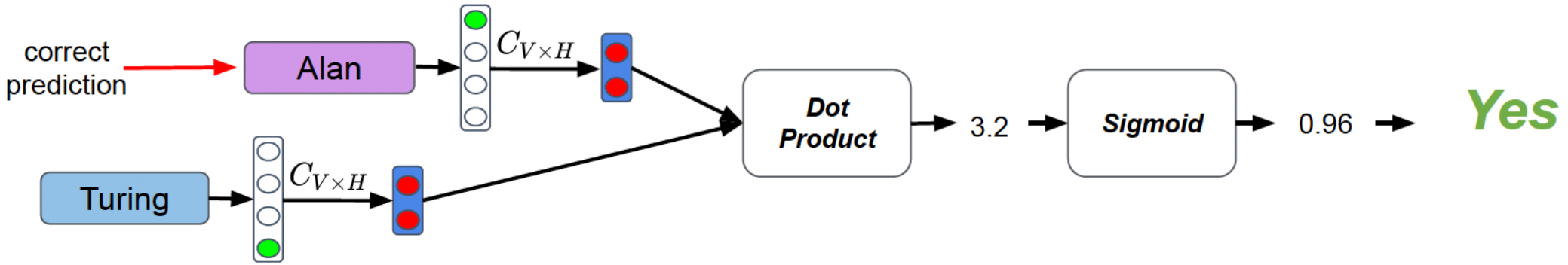
Train embeddings on old books to study changes in word meaning



Project 300 dimensions down into 2

Word2Vec – Negative Sampling

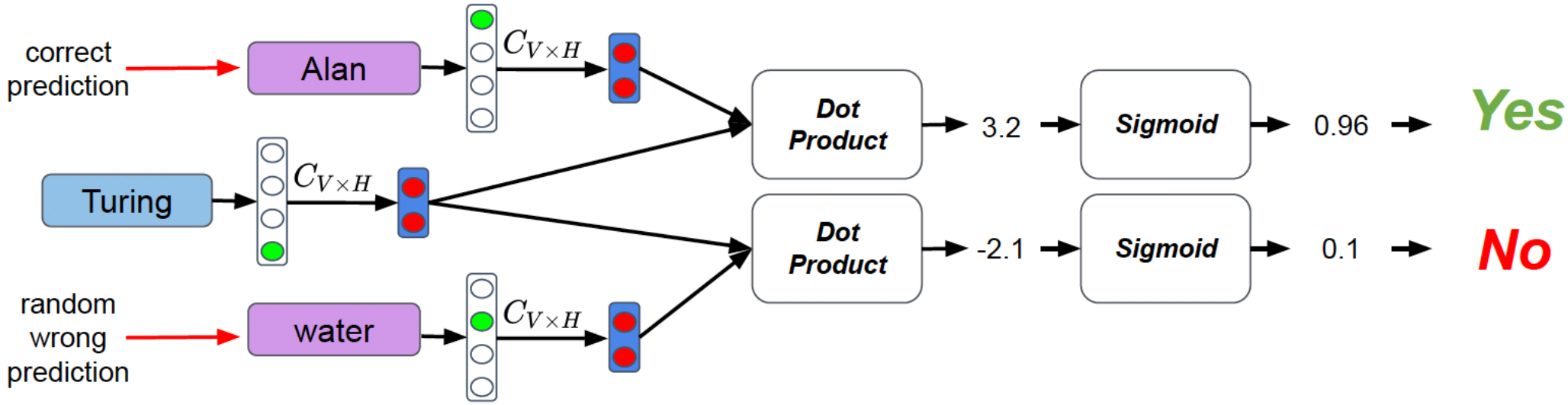
- Word2Vec may be expensive to train. Assuming V is the vocabulary size, the SoftMax is computed over V elements (which can be expensive if V is large).
- The SoftMax can be replaced with a Sigmoid computed only on K elements with K << V.
  - These elements include the correct prediction...



Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality."

Word2Vec – Negative Sampling

- Word2Vec may be expensive to train. Assuming V is the vocabulary size, the SoftMax is computed over V elements (which can be expensive if V is large).
- The SoftMax can be replaced with a Sigmoid computed only on K elements with K << V.
  - These elements include the correct prediction…
  - … plus K-1 wrong predictions (sampled proportionally to word frequencies).

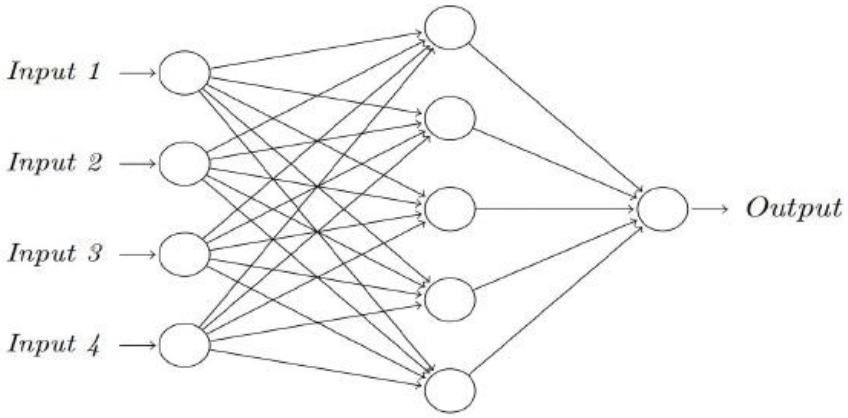


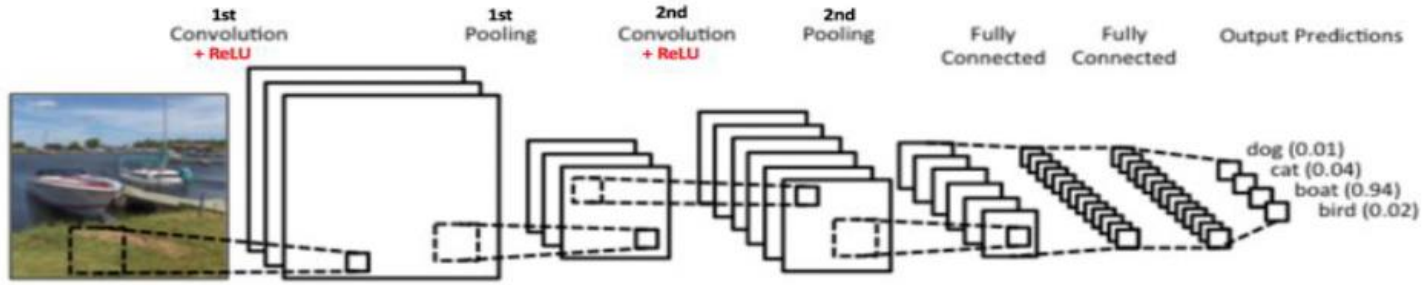
Topics

- ***Deep learning for NLP***
- RNN
- Training RNNs
- Training Problems
- RNN Architectures
- Deep RNNs

Motivation

- You have seen how to handle data with fixed size and how to handle images.

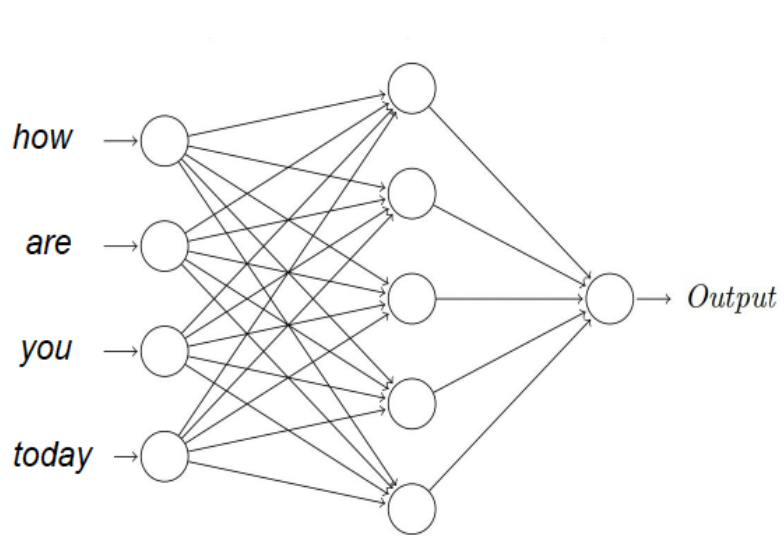


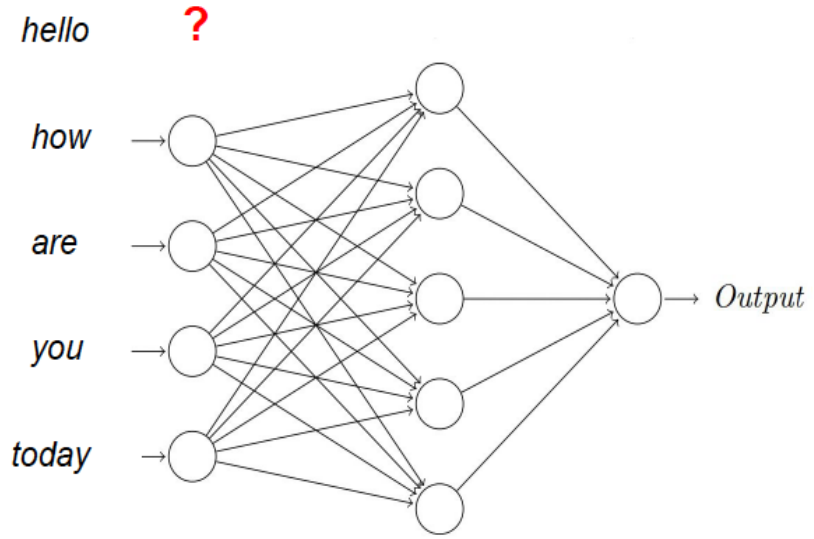


- How can we handle sequences of variable size?

Motivation

- MLPs **cannot** handle sequences of variable size.



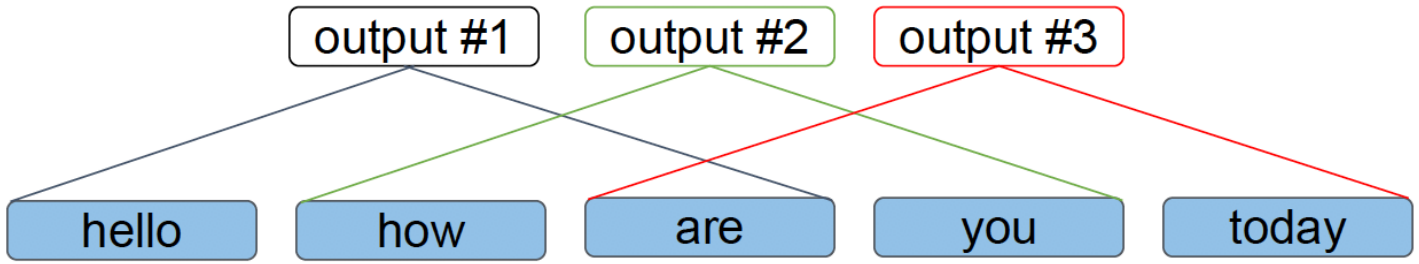


- Some techniques (such as bag-of-words for processing text input) allow an MLP to handle sequences of variable size; but those techniques ignore the order of the elements in the sequence.

Motivation

- A CNN can operate on sequences, but:
  - the receptive field is limited by the filter size.

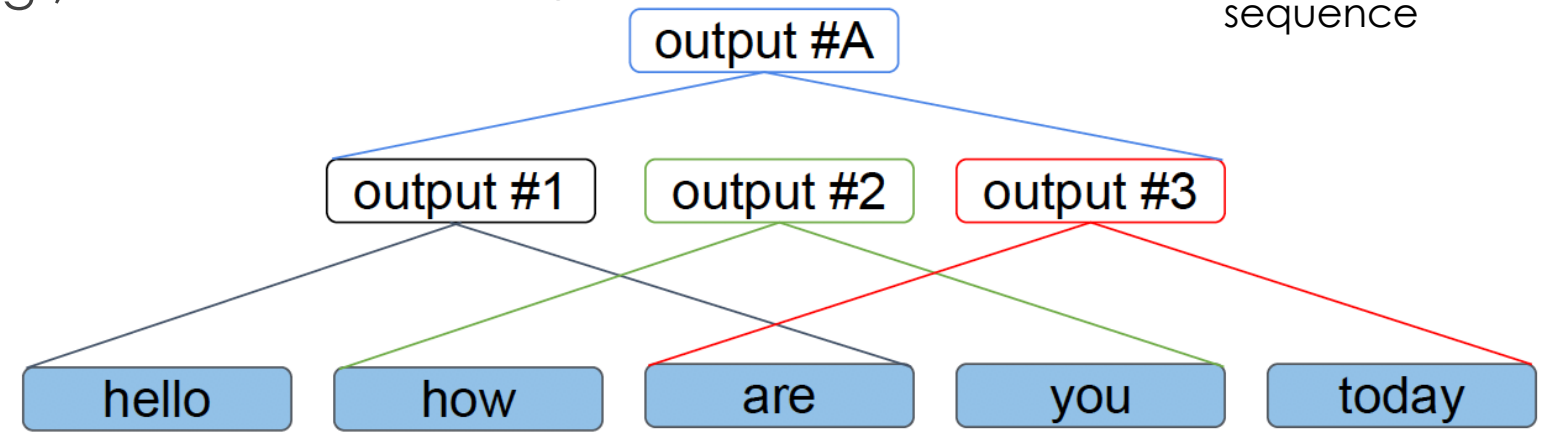
- E.g., with a filter of size 3:



Motivation

- A CNN can operate on sequences, but:
  - the receptive field is limited by the filter size.
  - more layers are needed to capture information from the entire sequence.

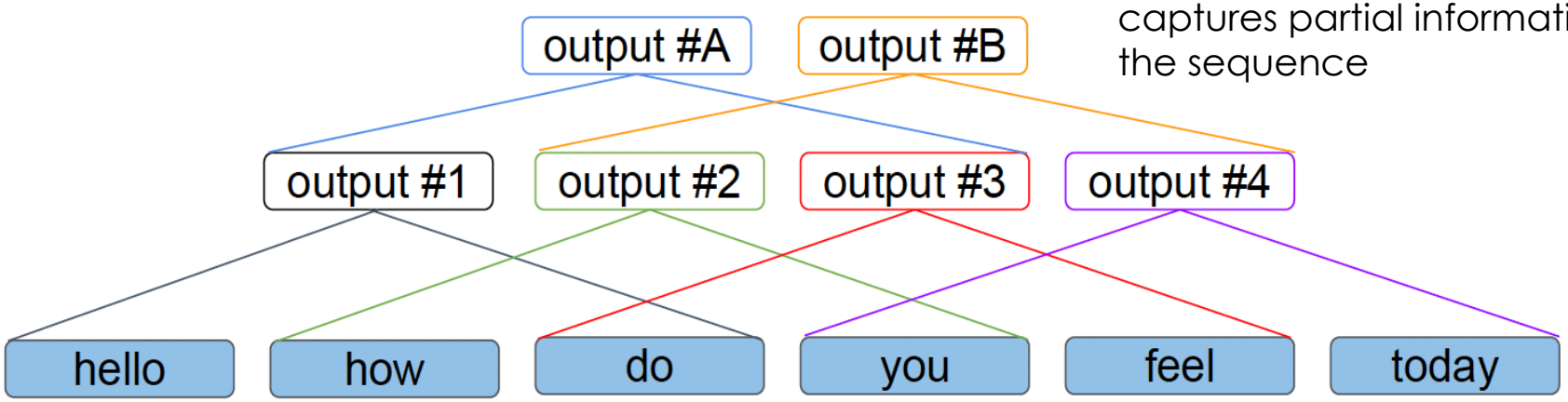
- E.g., with a filter of size 3:



output #A has access to information from the entire sequence

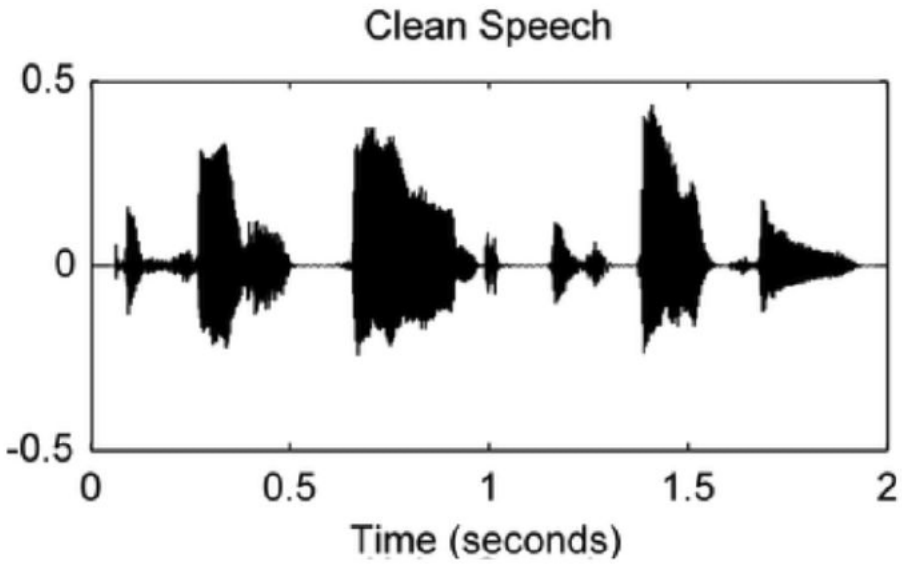
Motivation

- A CNN can operate on sequences, but:
  - the receptive field is limited by the filter size.
  - more layers are needed to capture information from the entire sequence.
  - longer sentences can still fall out of the receptive field.
- E.g., with a filter of size 3:



each of output #A and #B only captures partial information from the sequence

Examples

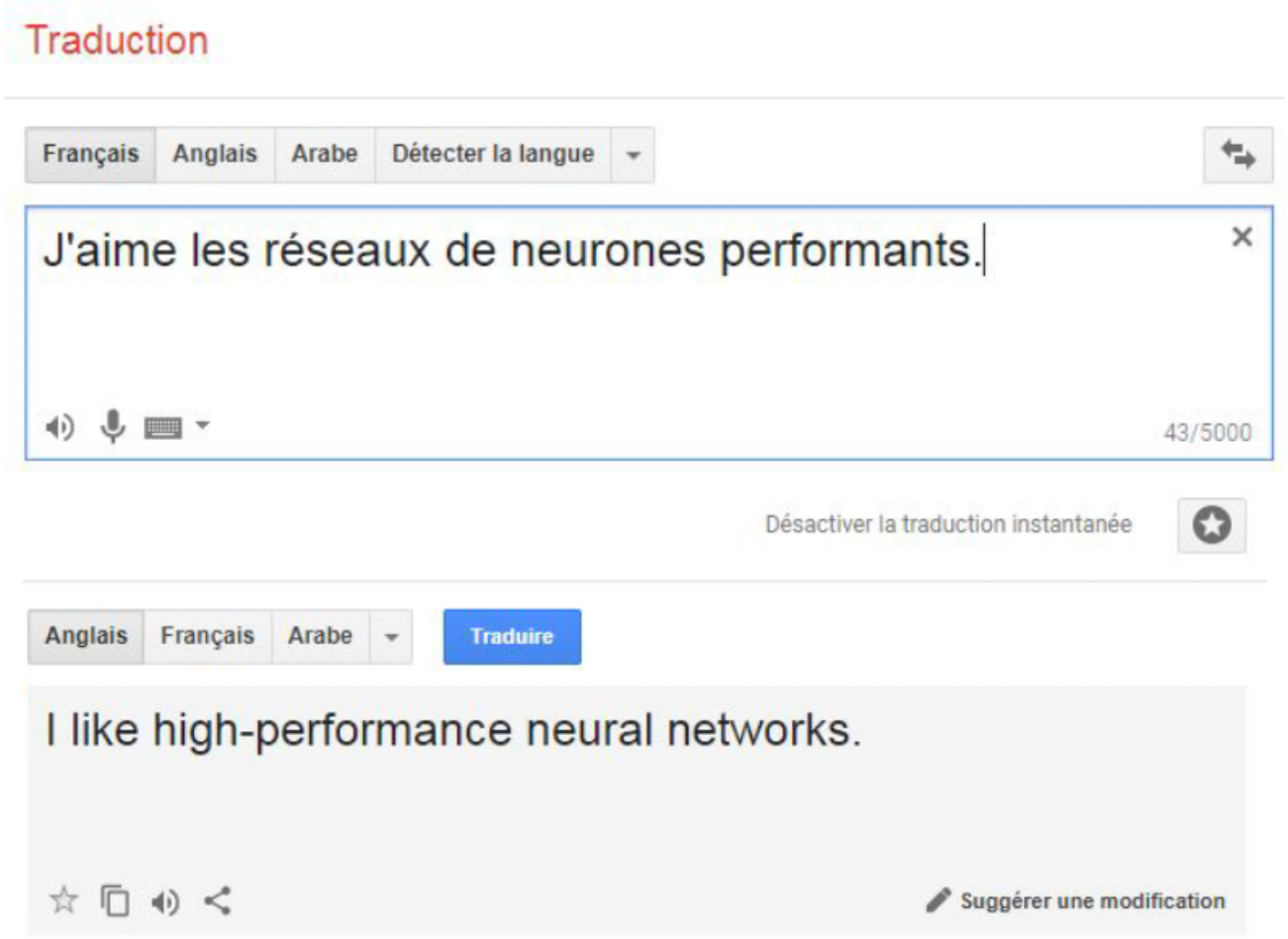


“The fresh bread is baking.”

Speech recognition: Audio sequence → Word sequence.

Images: Ye et al., A Wavelet-Based Noise Reduction Algorithm and Its Clinical Evaluation in Cochlear Implants

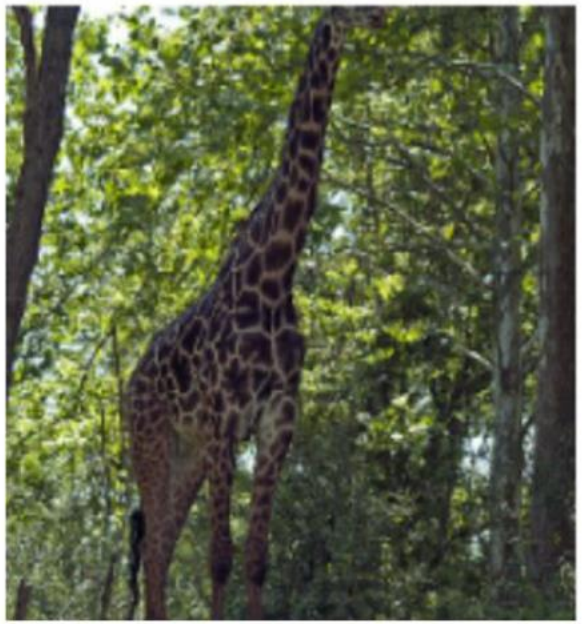
Examples



Translation: Word sequence → Word sequence.

Examples

A woman is throwing a frisbee in a park.



A giraffe standing in a forest with trees in the background.

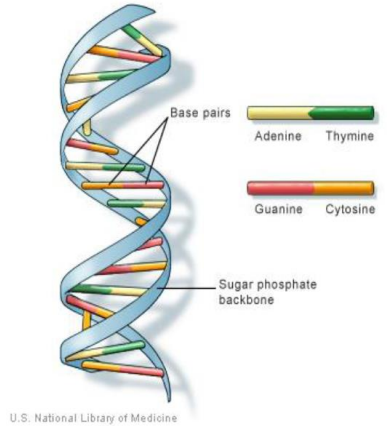
Caption generation: Image → Word sequence.

Images: Xu et al., Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

Examples

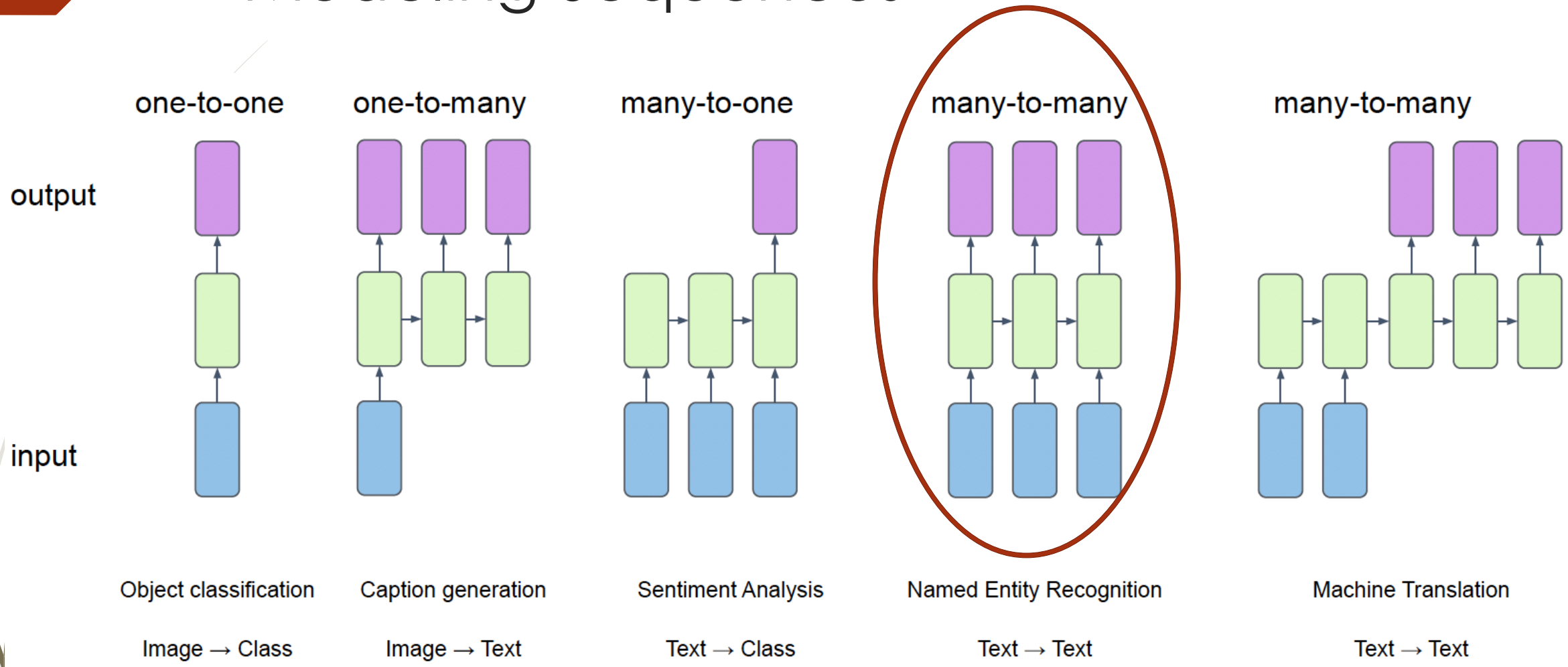
- Audio:
  - Speech recognition
  - Text to speech
- Video:
  - Caption generation
  - Movement detection
- Text:
  - Email classification
  - Machine translation

- Medical and Biological data:
  - DNA study
  - Electrocardiogram
- Time series (stocks, weather, ...)
- Etc…



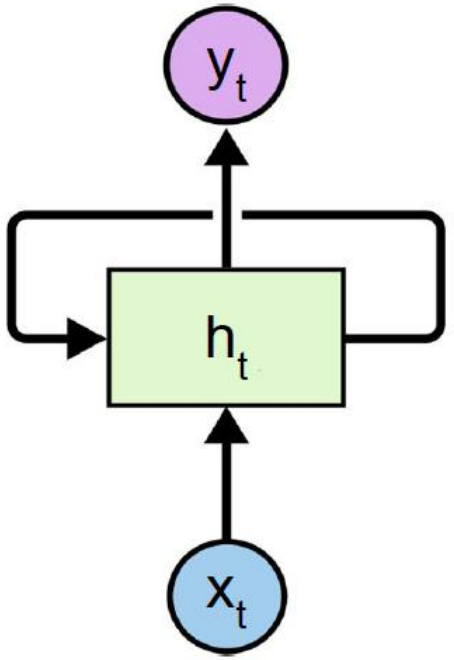
There is a lot of data with sequences!

Modeling Sequences



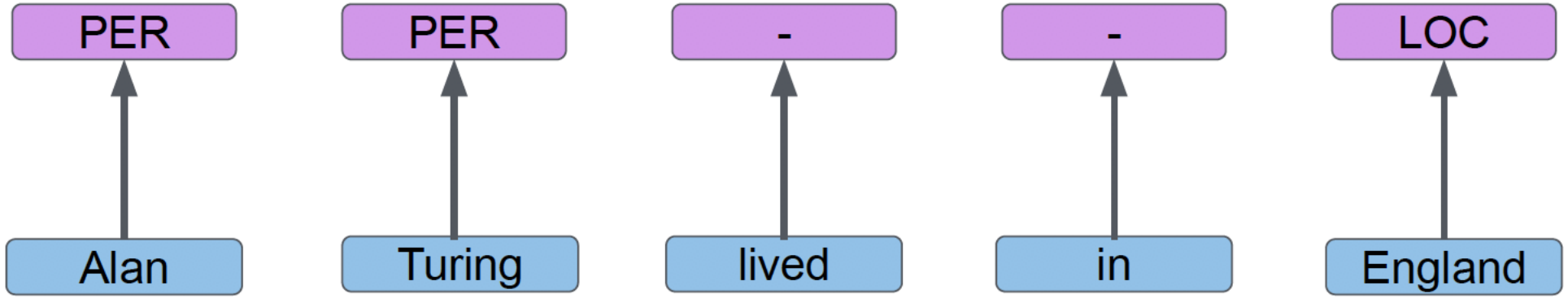
Recurrent Neural Networks

- A Recurrent Neural Network (RNN) applies a function to an input sequence x_t **one element at a time** - in order to generate an output sequence y_t while maintaining an internal state h_t.



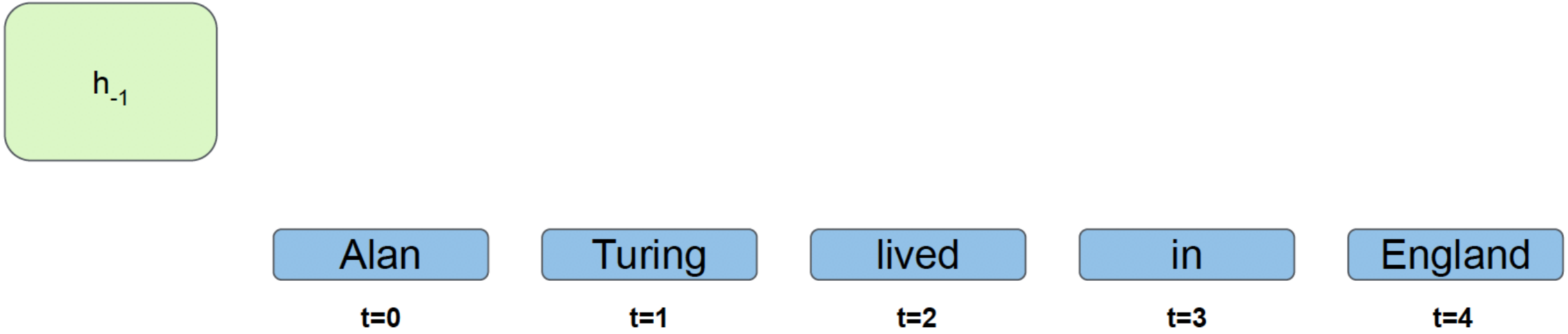
Recurrent Neural Networks - Example

- Before formalizing the architecture, let's see an example showing how a RNN works to solve a Named Entity Recognition (NER) problem:
  - assign a label that represents an entity class to every word in an input.
- In this example, possible labels are: "PER" (person), "LOC" (location), "-" (not a named entity).



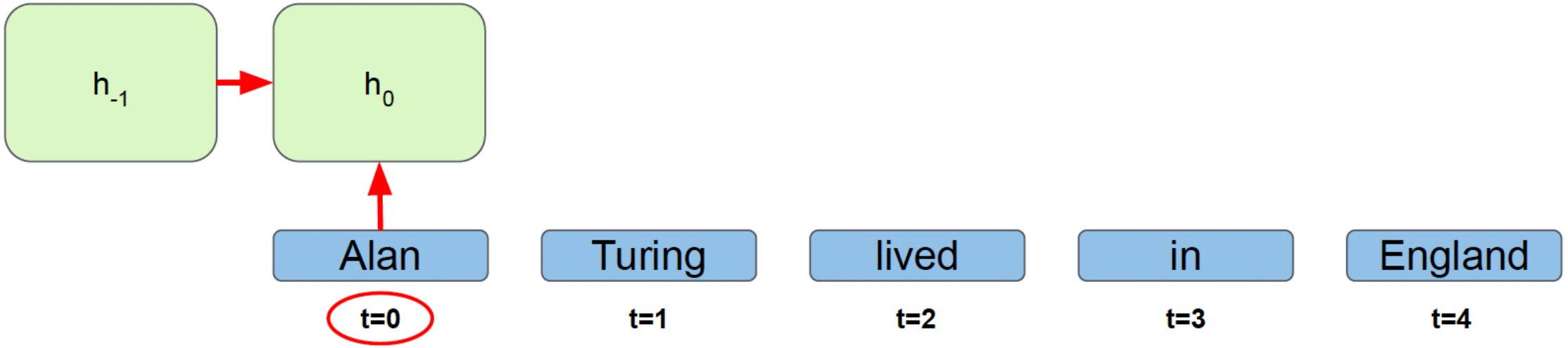
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



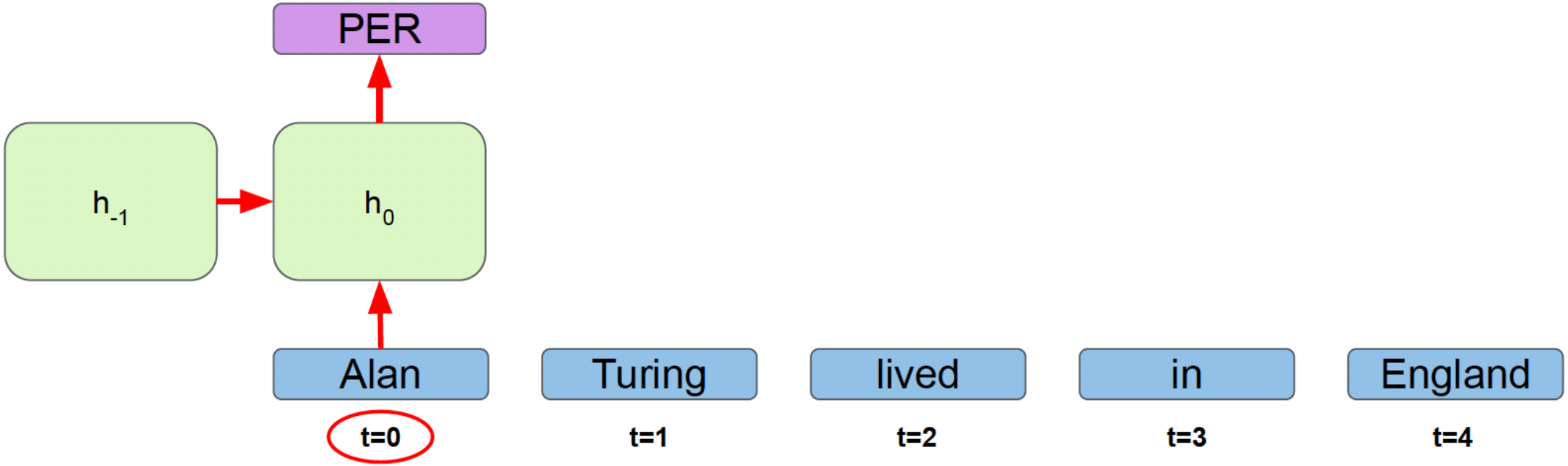
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



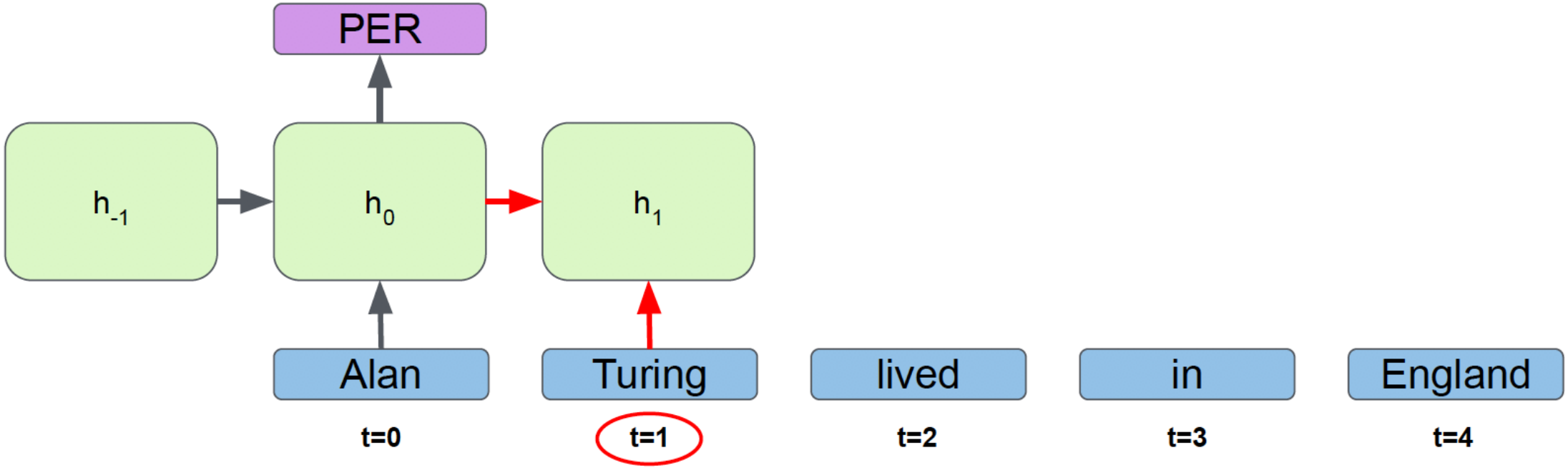
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



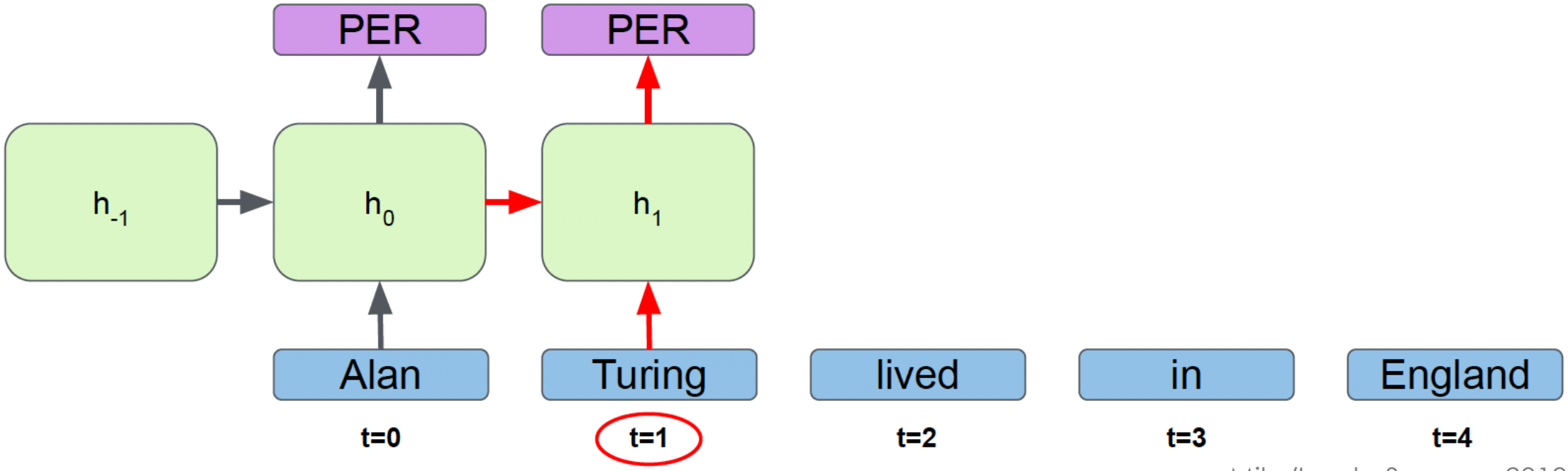
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



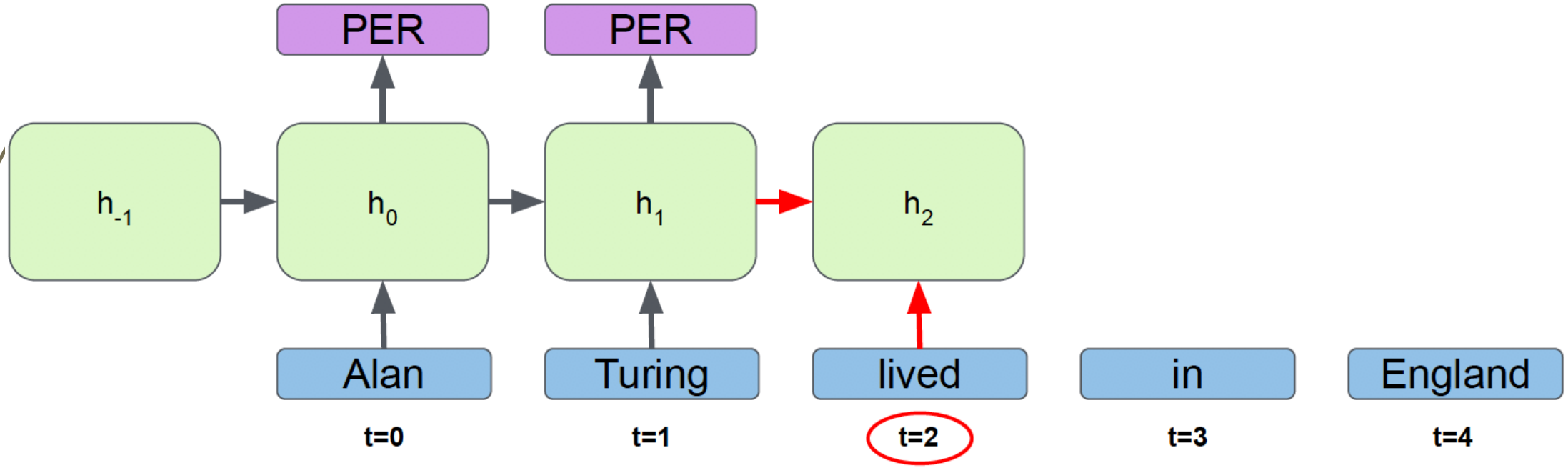
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



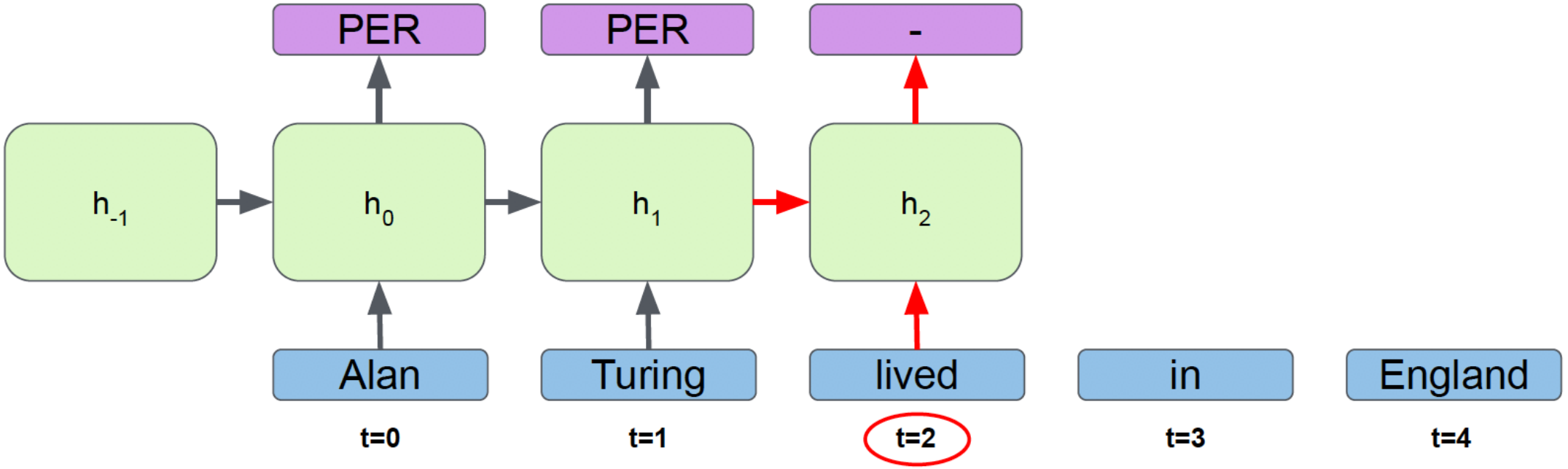
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



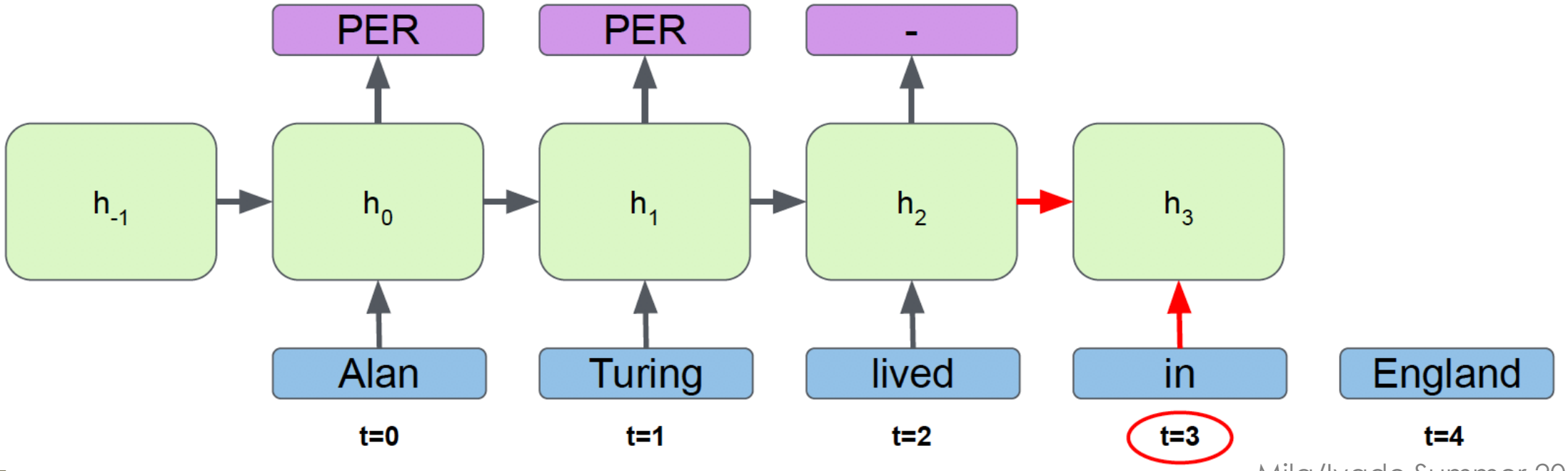
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



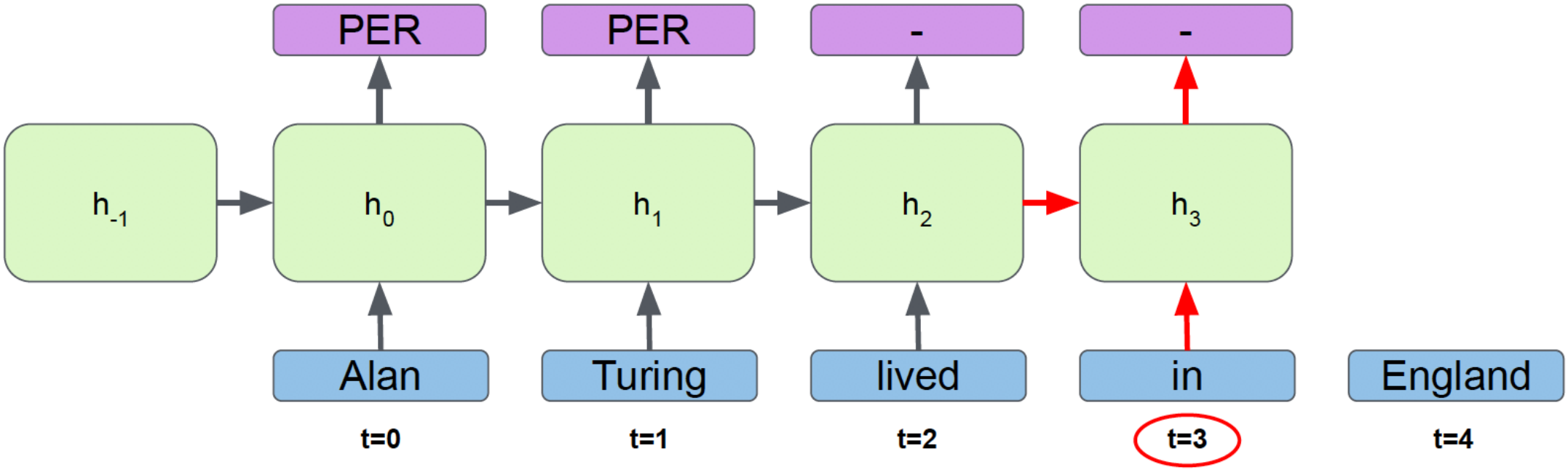
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



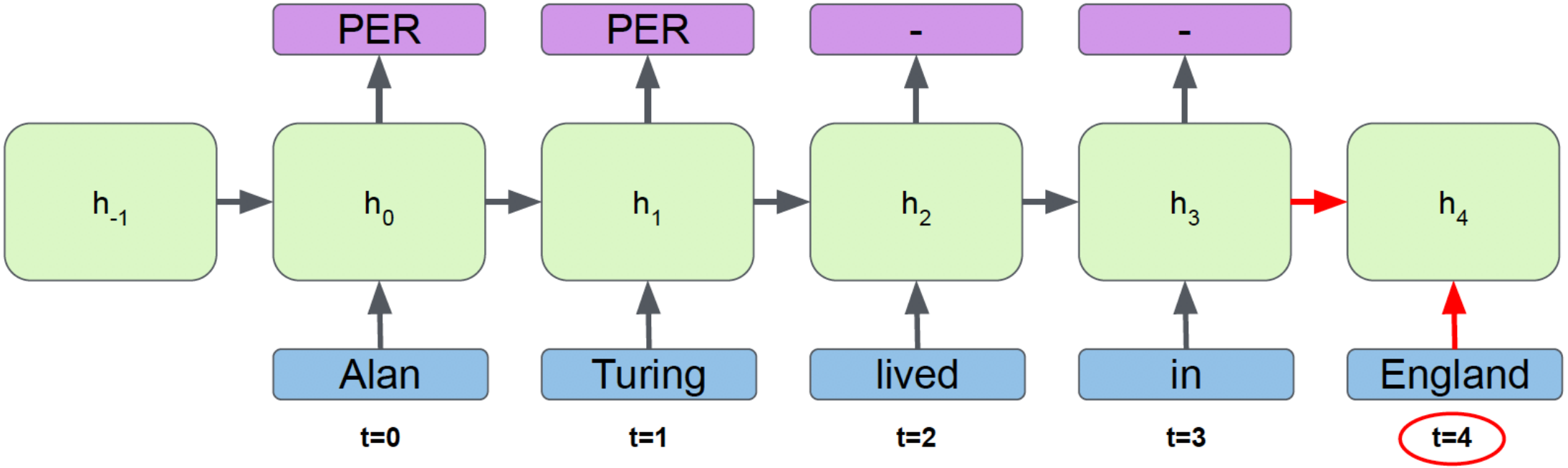
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



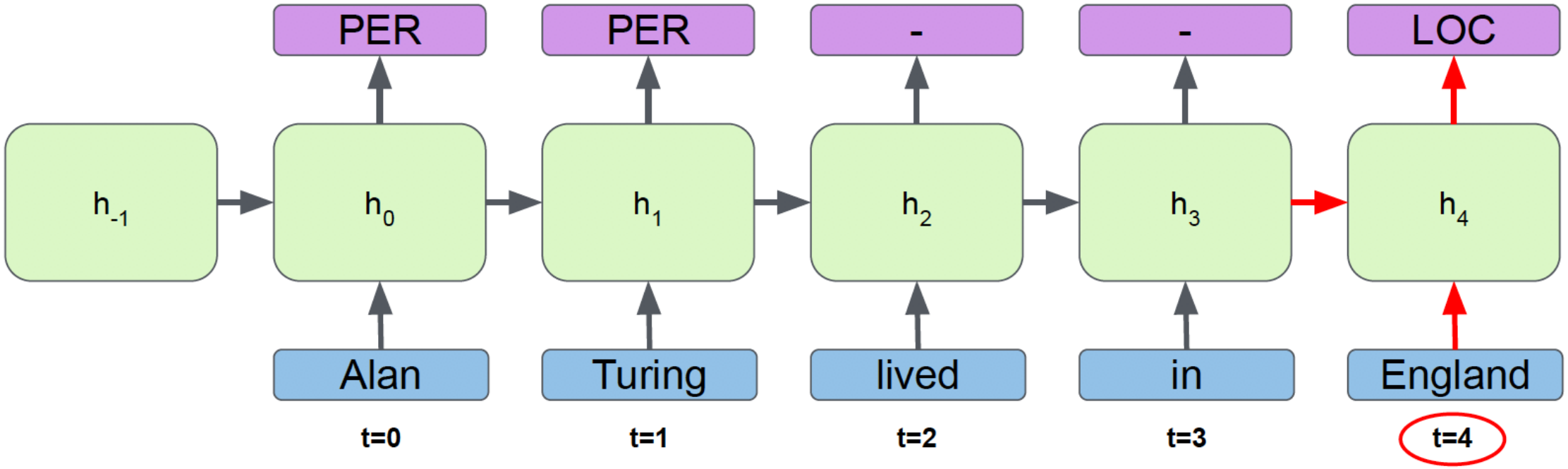
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



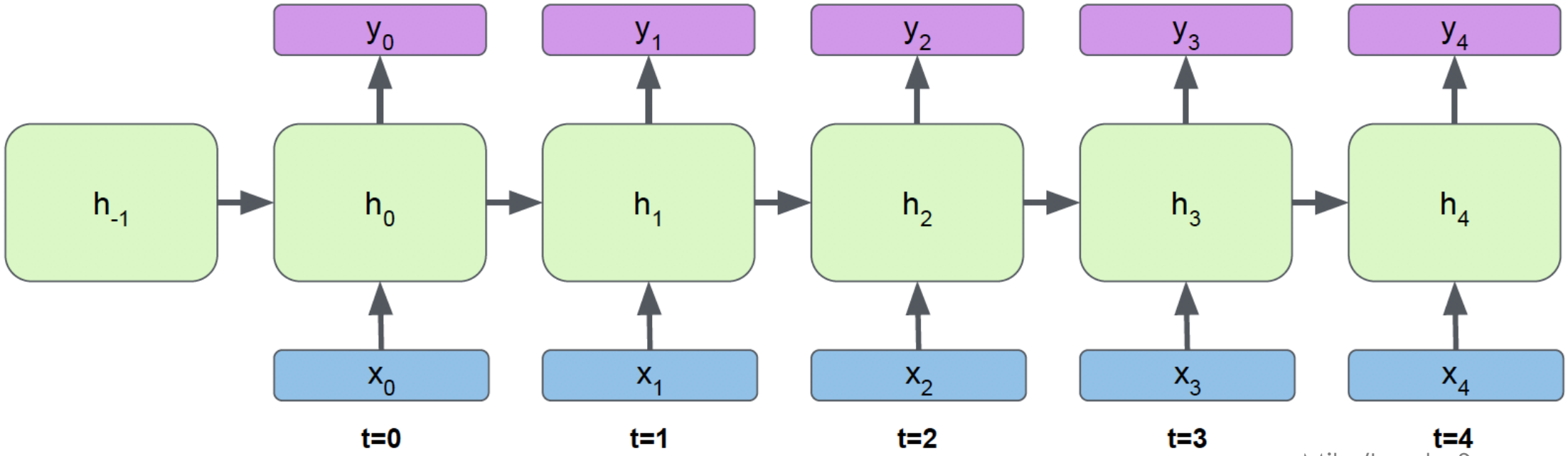
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence - **one element at a time** – in order to generate an output sequence while maintaining an internal state.



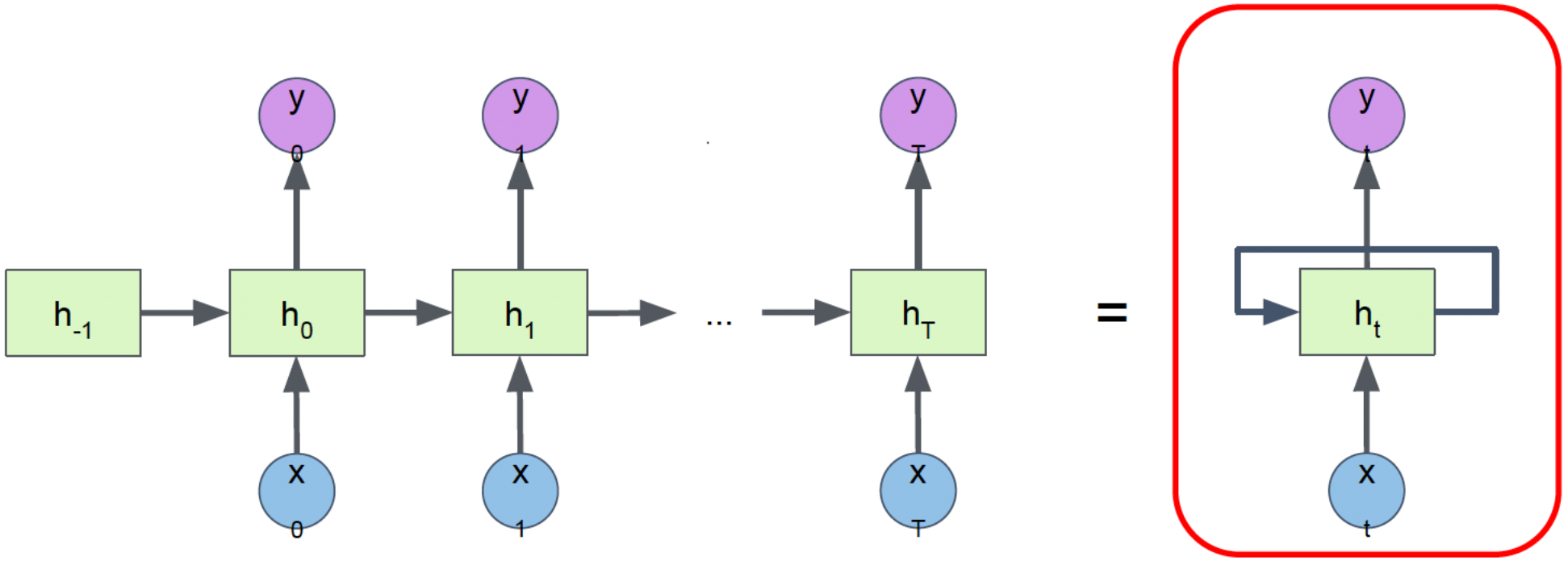
Recurrent Neural Networks - Example

- A RNN applies a function to an input sequence $[x_0, x_1, \ldots, x_T]$ - **one element at a time** – in order to generate an output sequence $[y_0, y_1, \ldots, y_T]$ while maintaining an internal state $[h_0, h_1, \ldots, h_T]$.



Recurrent Neural Networks - Formalization

- The previous example shows how a RNN "unfolds" over the input sequence.
- A RNN can also be described using a compact ("folded") representation:



Recurrent Neural Networks - Implementation

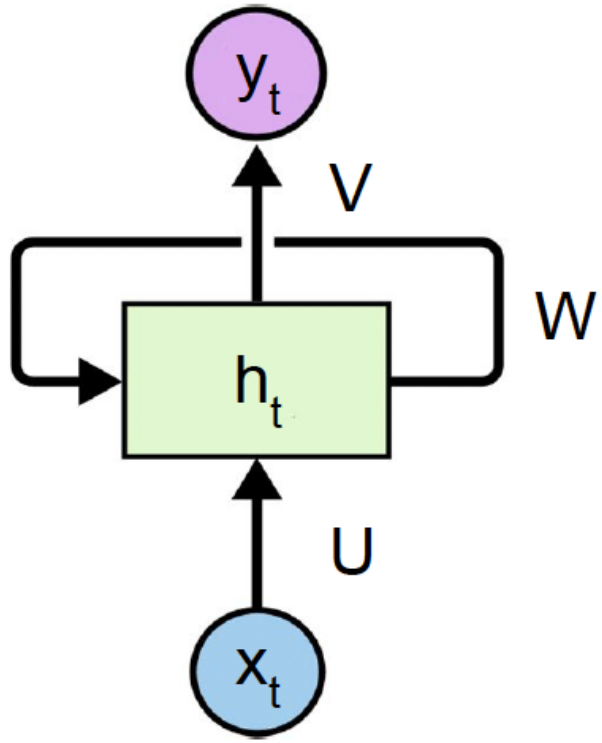
- Most simple implementation:

$$h_t = tanh(Ux_t + Wh_{t-1} + b_h)$$

$$y_t = g(Vh_t + b_y)$$

g = softmax, sigmoid, ...

- U, W, V, b_h, and b_y are the RNN parameters.
- They are **shared** over time.



Recurrent Neural Networks - Implementation

- The parameters are **shared** over time.
- The internal state (h_t) is updated at each time step.

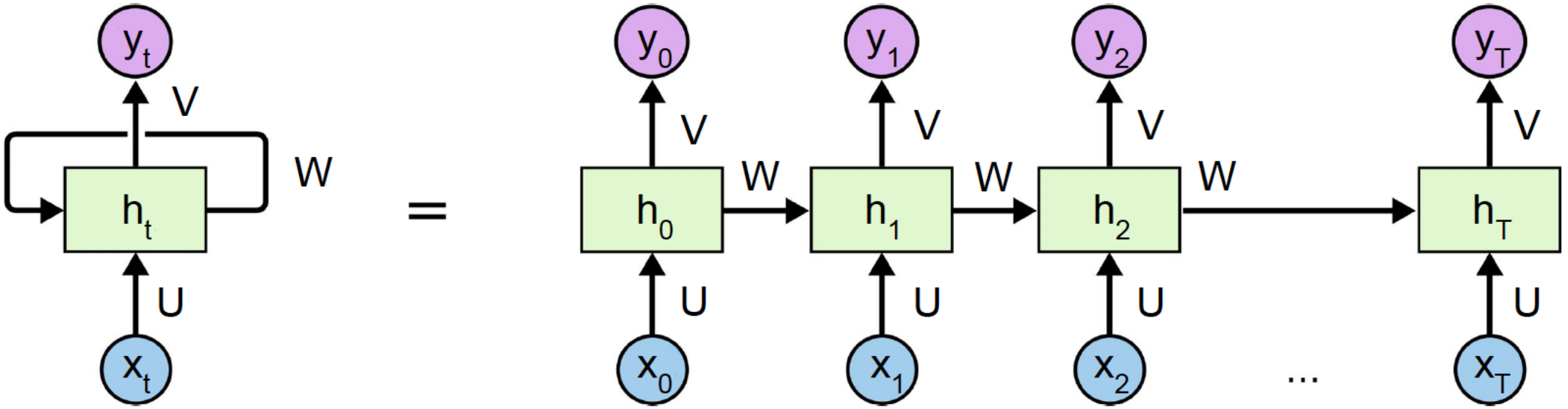
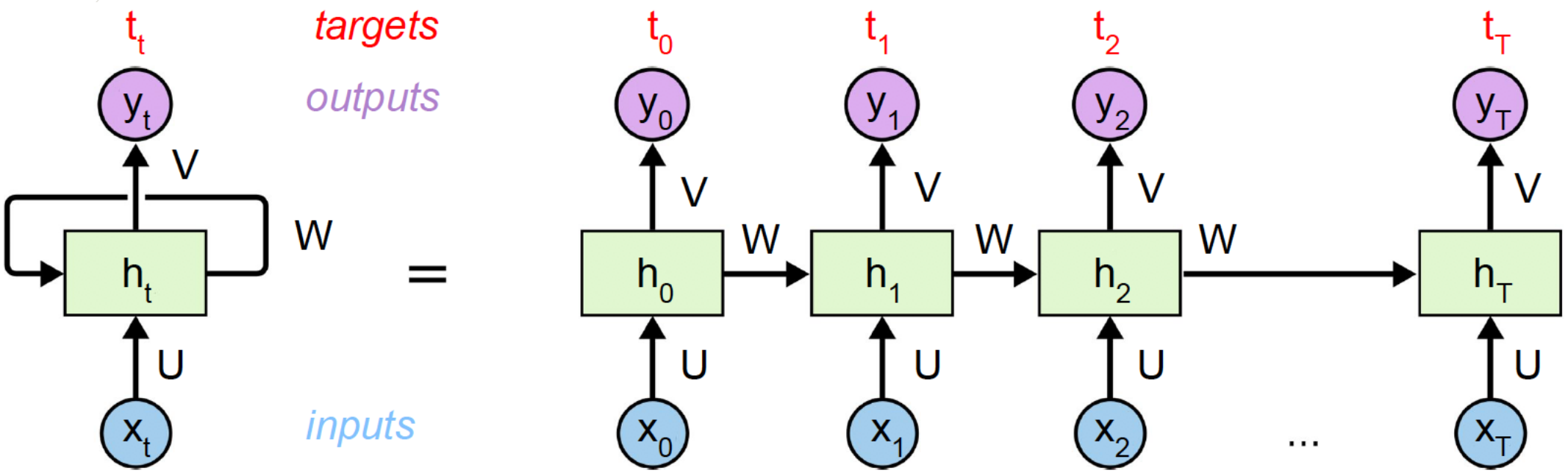


Image from Christopher Olah's blog

Topics

- ***Deep learning for NLP***
- ***RNN***
- Training RNNs
- Training Problems
- RNN Architectures
- Deep RNNs

Training Error



Global error E = sum of the error at every time step:

$$E = \sum_{t=0}^{T} E_t = \sum_{t=0}^{T} f(t_t, y_t)$$

f = loss function (cross-entropy, mean squared error, …)

Image from Christopher Olah's blog

Training Error

- The global error is:

$$E = \sum_{t=0}^{T} E_t$$

- To compute the gradient of the global error with respect to a parameter, we can compute the gradient of the individual error at each time step, and then sum all those values.
- For example, let's focus on the gradient of E over V.

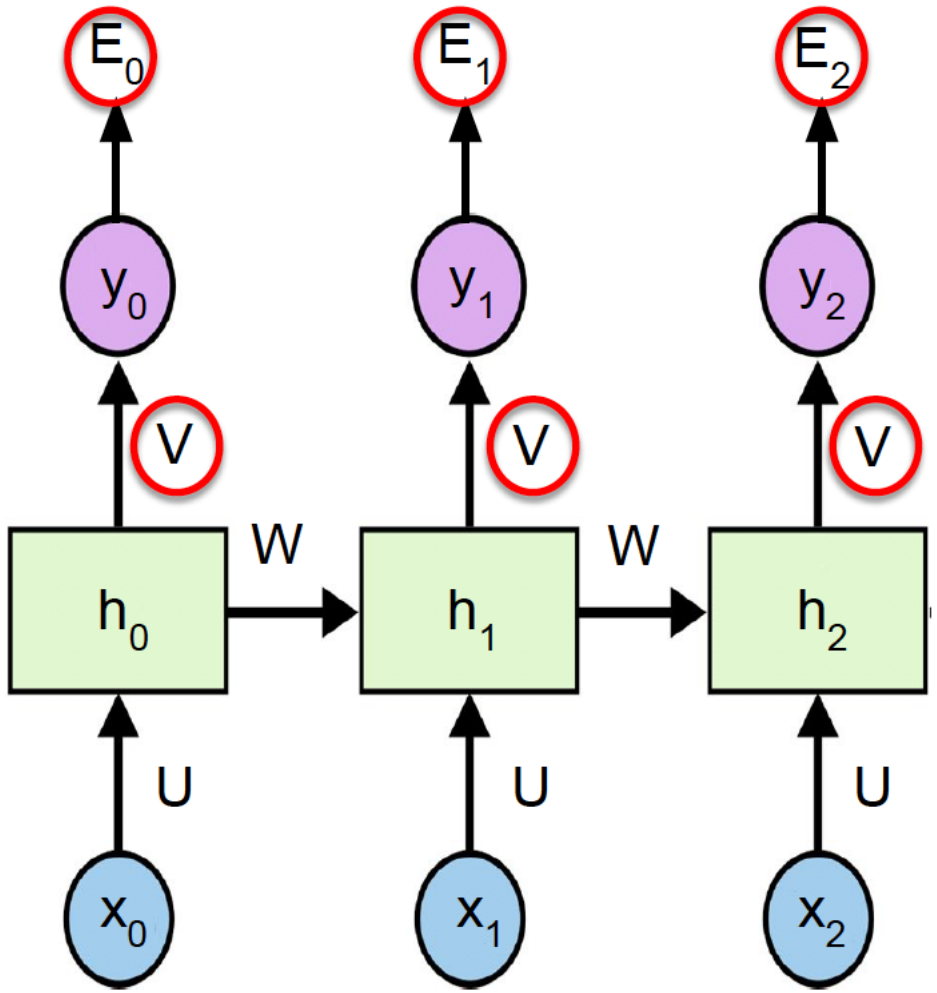


Image from Christopher Olah's blog

Backpropagation

- We start with E_2 ...

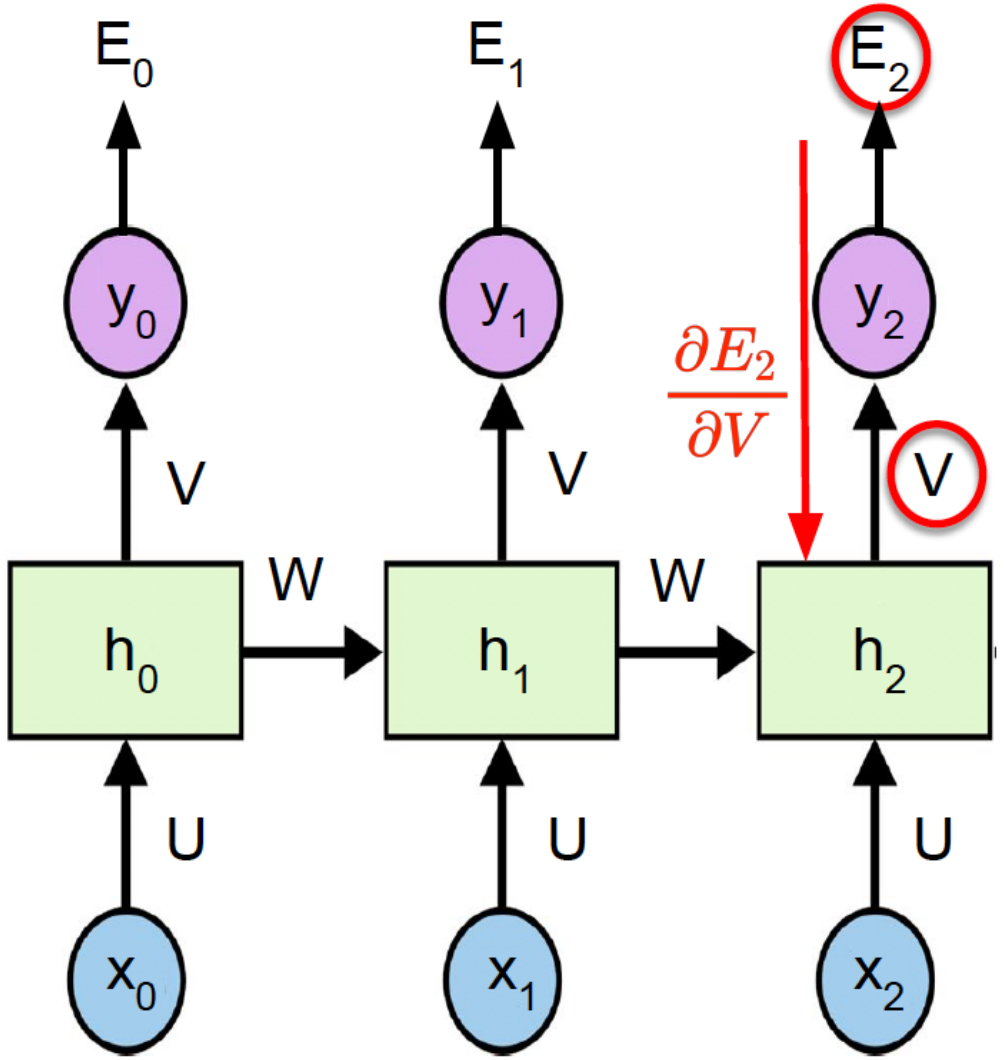


Image from Christopher Olah's blog

Backpropagation

- We start with E_2 ...
- ... then E_1...

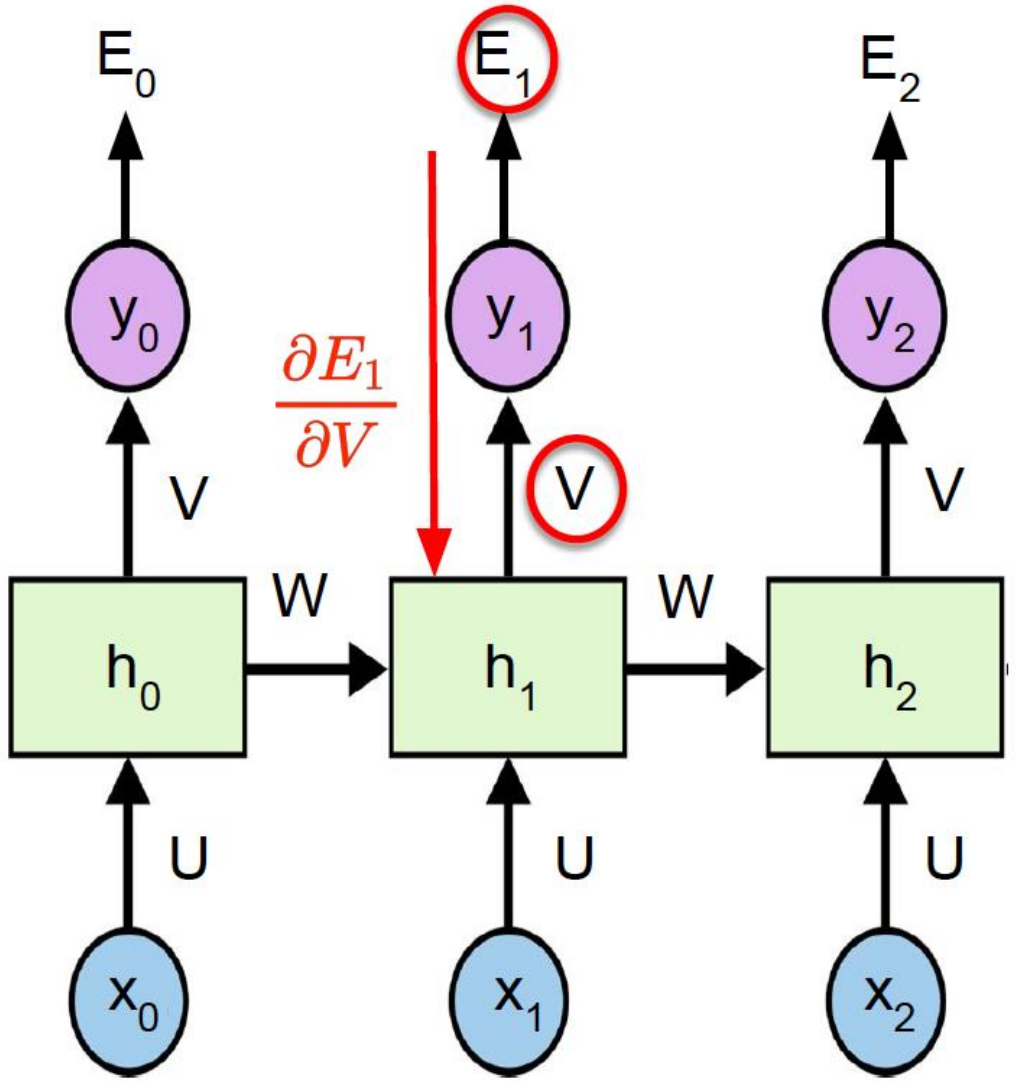


Image from Christopher Olah's blog

Backpropagation

- We start with E_2 ...
- ... then E_1...
- ... then E_0...

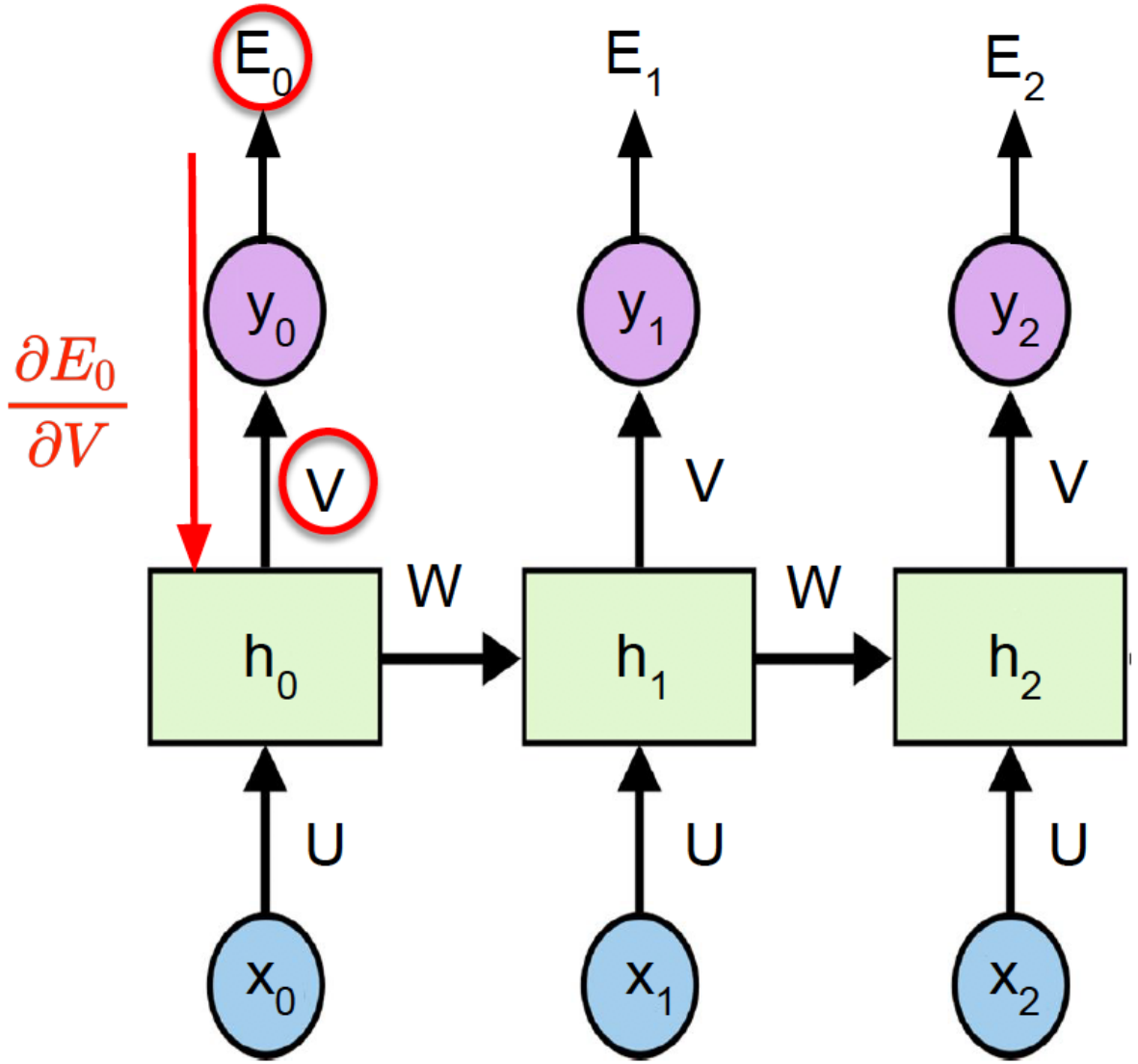


Image from Christopher Olah's blog

Backpropagation

- Now we can just sum the gradients:

$$\frac{\partial E}{\partial V} = \frac{\partial E_2}{\partial V} + \frac{\partial E_1}{\partial V} + \frac{\partial E_0}{\partial V}$$

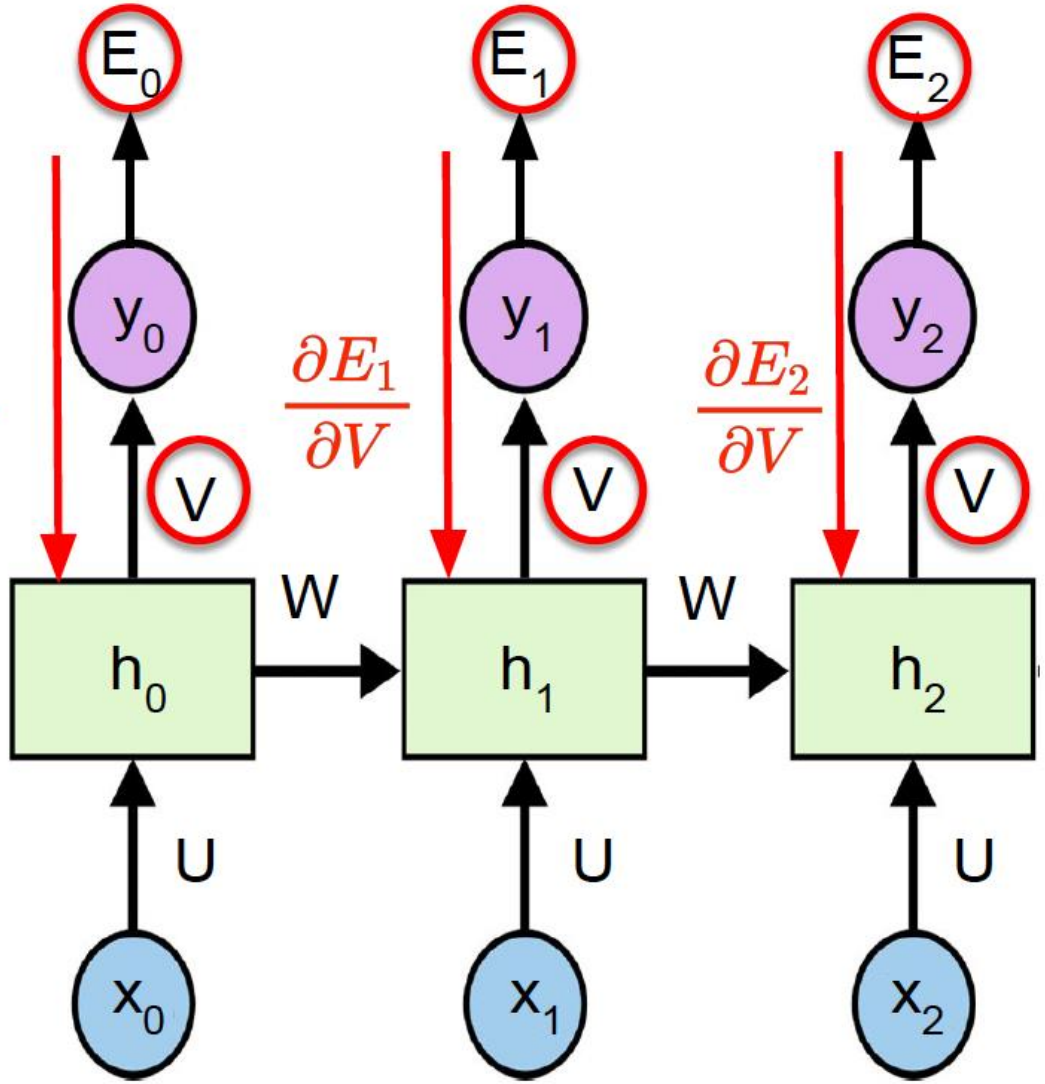


Image from Christopher Olah's blog

Backpropagation Through Time

- Some parameters are used more than once - even if we focus on a single error E_t.
- For example, U is used in three different places to generate y_2 (which is used to compute E_2).

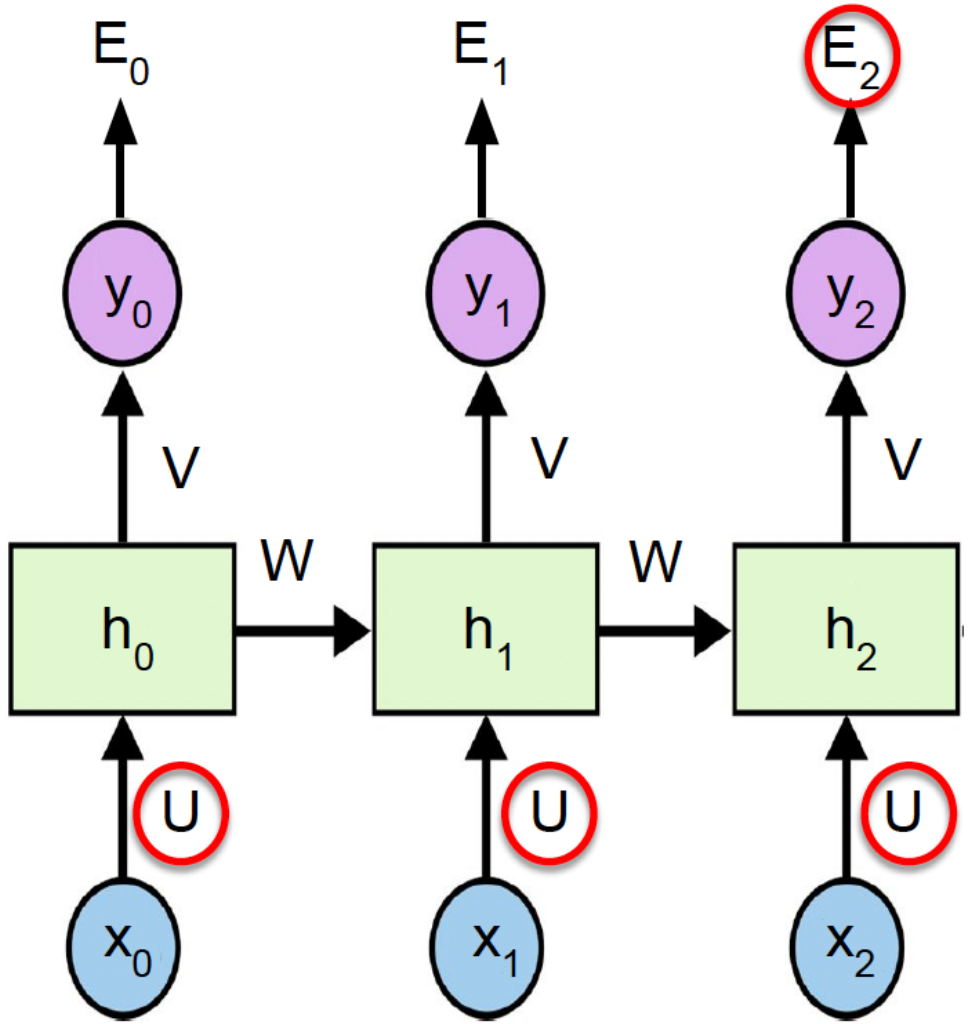


Image from Christopher Olah's blog

Backpropagation Through Time

- To perform the backpropagation, we need to consider all the places where U has been used.
- Given that we need to consider the "past" as well, we call this **backpropagation through time**.

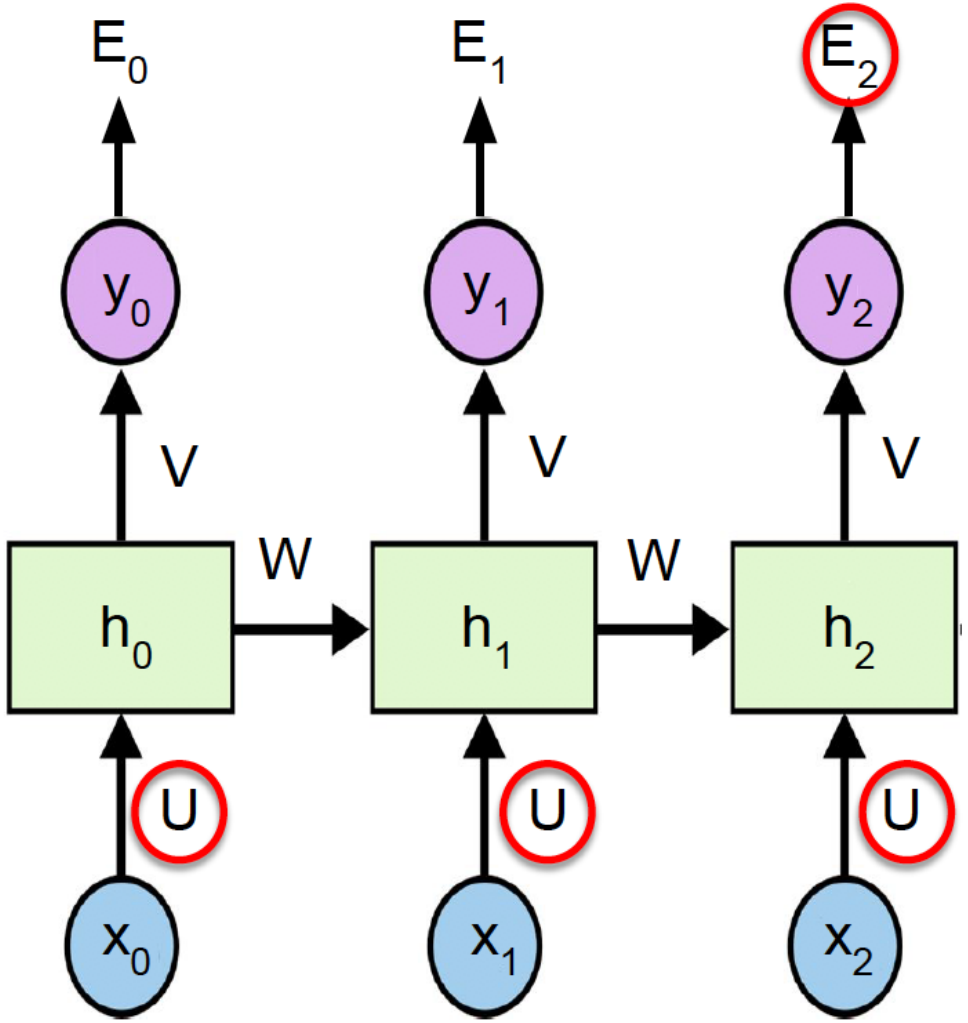


Image from Christopher Olah's blog

Backpropagation Through Time

- We apply the chain rule to compute:

$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial U} +$$

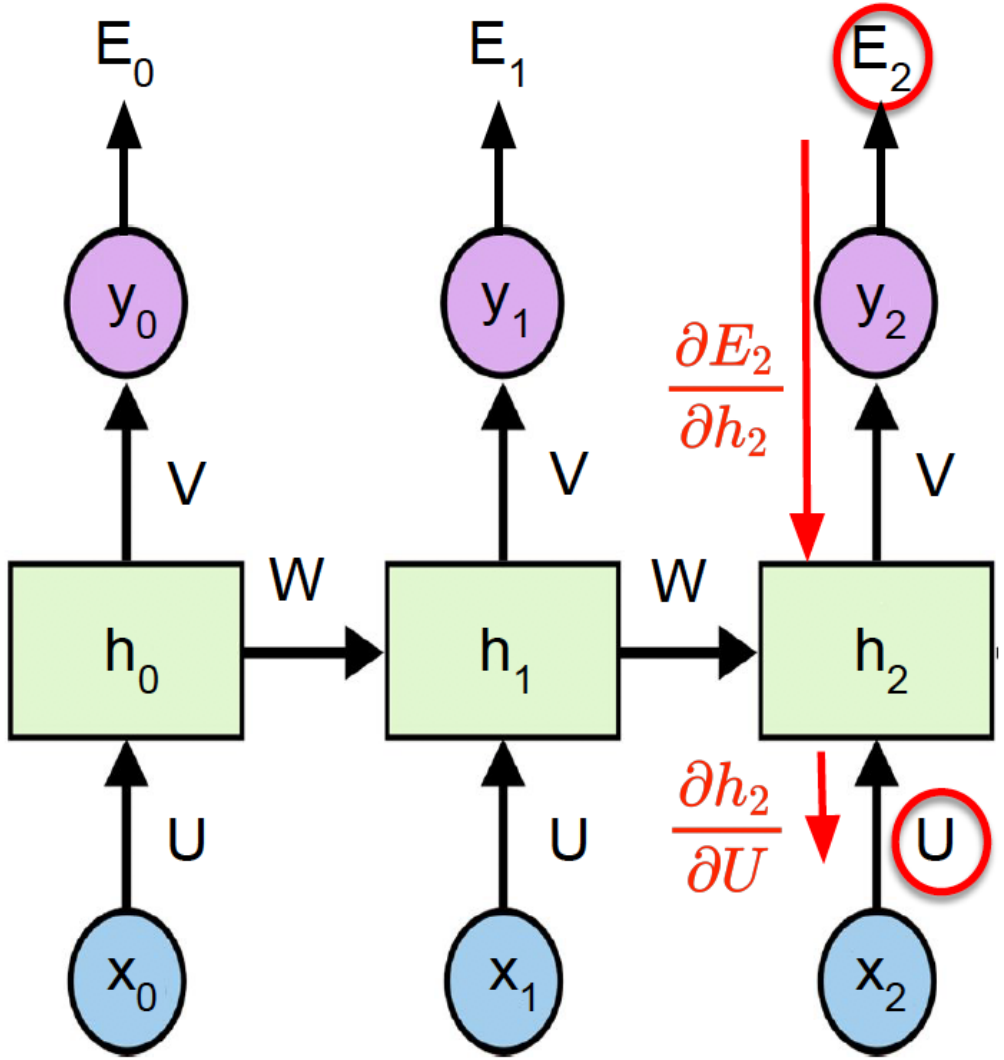


Image from Christopher Olah's blog

Backpropagation Through Time

- We apply the chain rule to compute:

$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial U} + \frac{\partial E_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial U} +$$

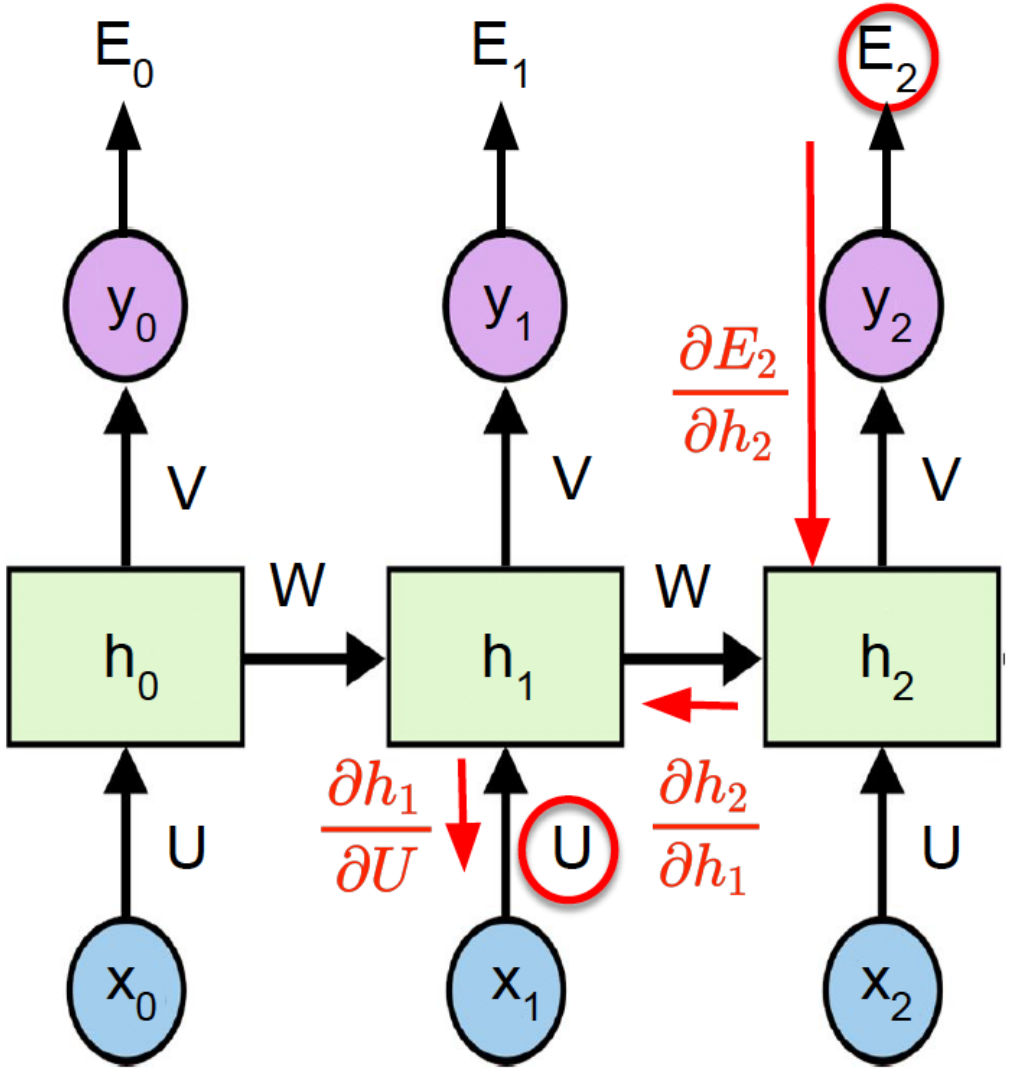


Image from Christopher Olah's blog

Backpropagation Through Time

- We apply the chain rule to compute:

$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial U} +$$

$$\frac{\partial E_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial U} +$$

$$\frac{\partial E_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0} \cdot \frac{\partial h_0}{\partial U}$$

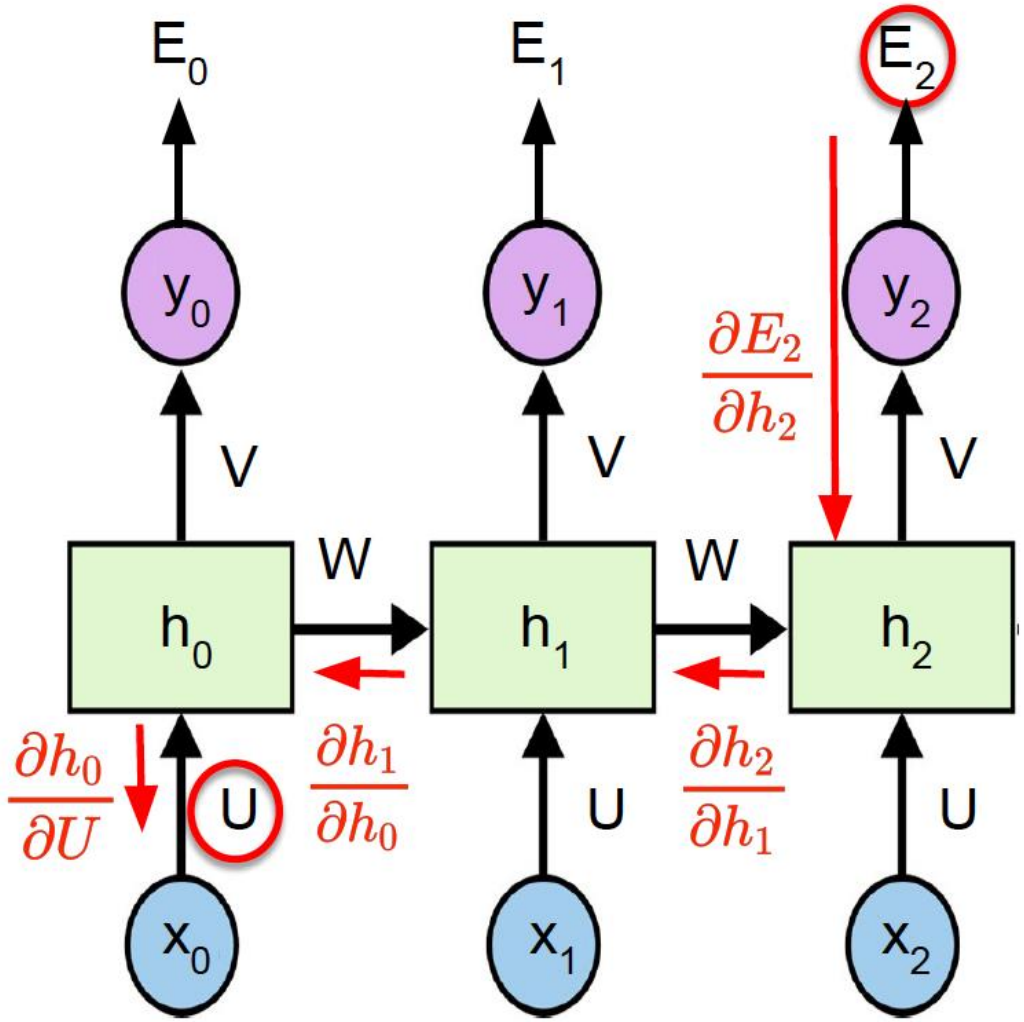


Image from Christopher Olah's blog

Backpropagation Through Time

- Once done with E_2 , we do the same for E_1…
- Note that we only need to consider the first two time steps.

$$\frac{\partial E_1}{\partial U} = \frac{\partial E_1}{\partial h_1} \cdot \frac{\partial h_1}{\partial U} + \frac{\partial E_1}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0} \cdot \frac{\partial h_0}{\partial U}$$

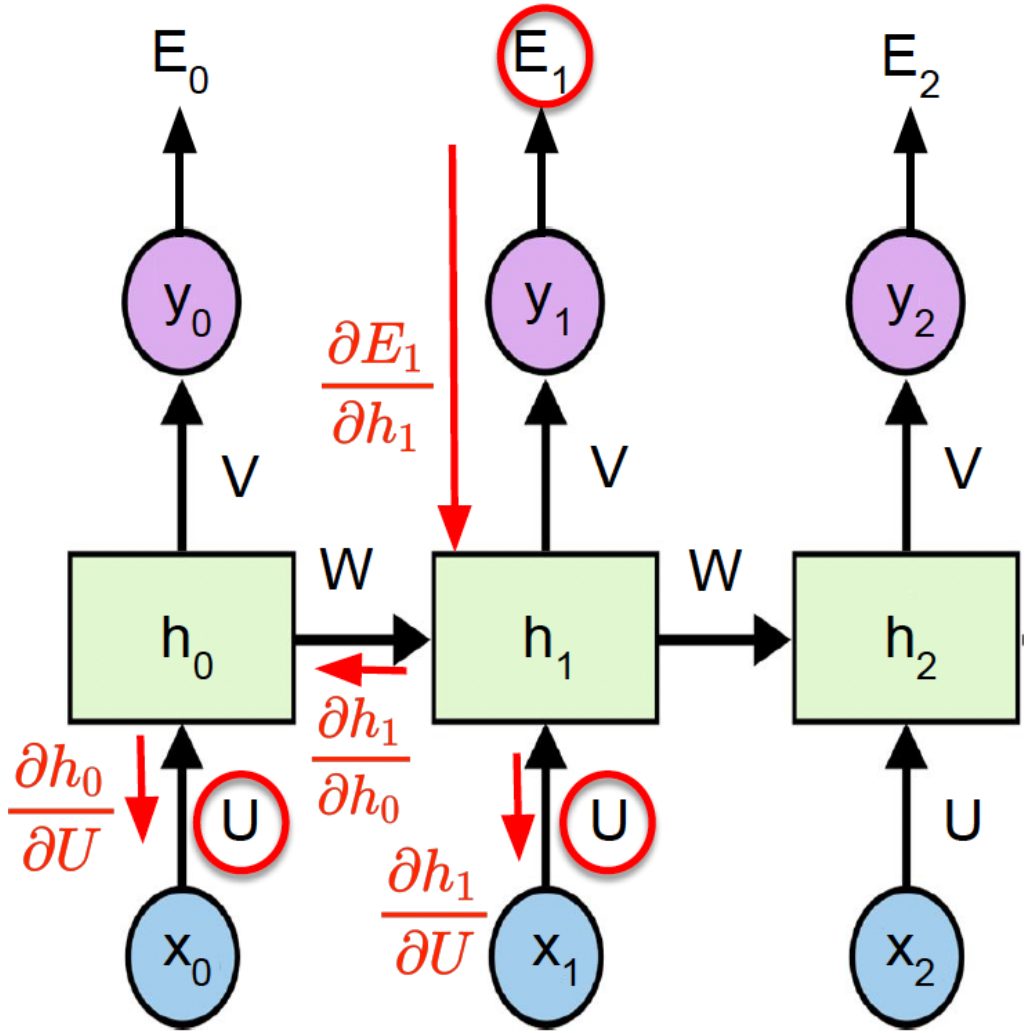


Image from Christopher Olah's blog

Backpropagation Through Time

- ...and for E_0.
- Note that we only need to consider the first time step.

$$\frac{\partial E_0}{\partial U} = \frac{\partial E_0}{\partial h_0} \cdot \frac{\partial h_0}{\partial U}$$

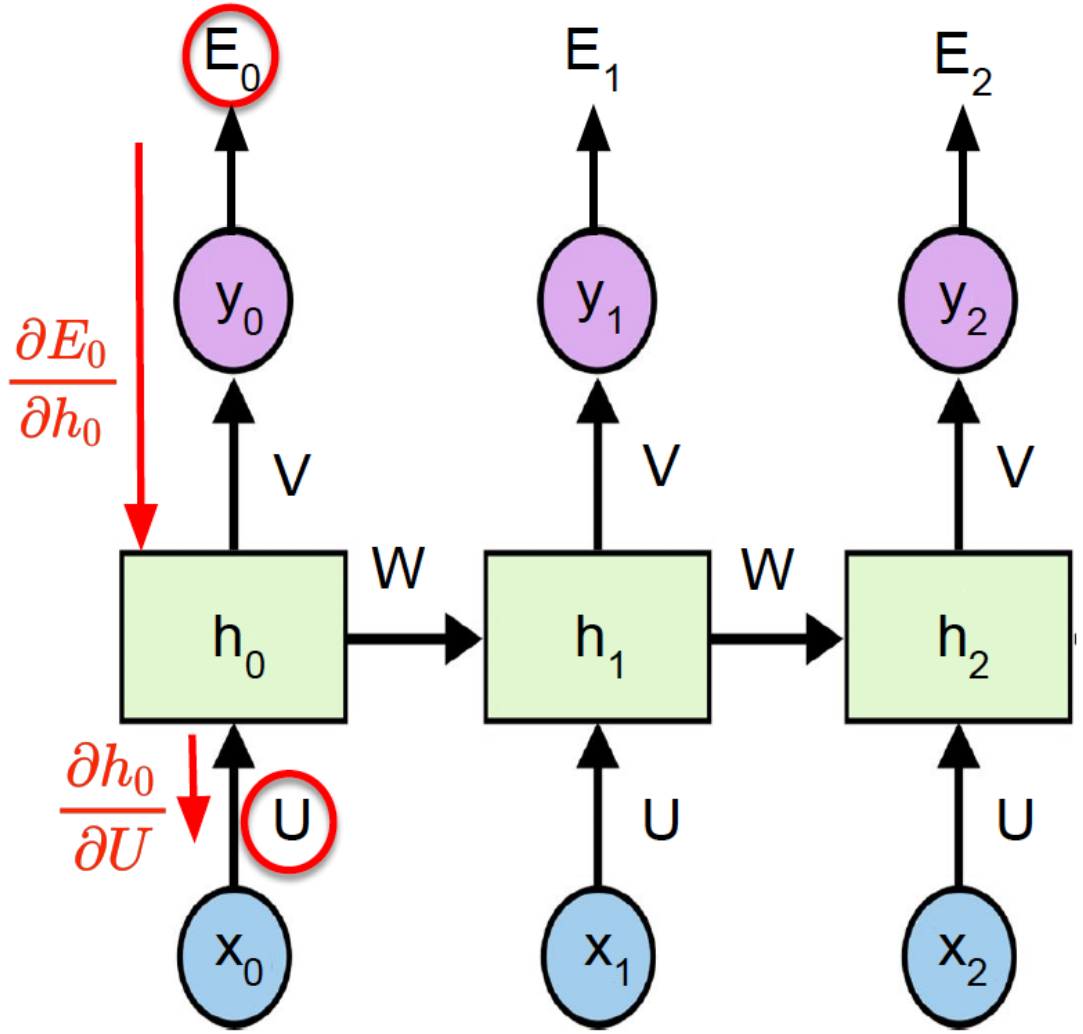


Image from Christopher Olah's blog

Backpropagation Through Time

- All the contributions are then summed to compute the gradient with respect to U:

$$\frac{\partial E}{\partial U} = \sum_{t=0}^{T} \frac{\partial E_t}{\partial U}$$

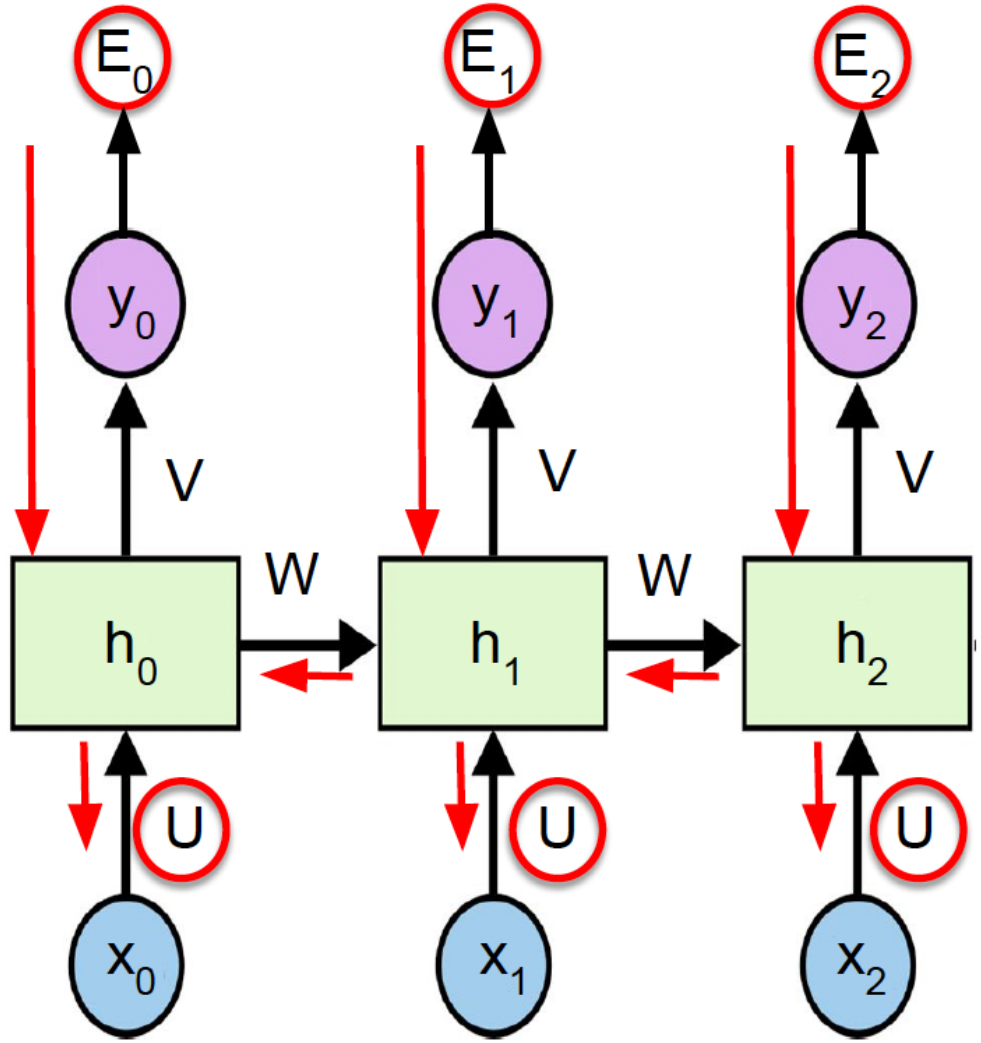


Image from Christopher Olah's blog

Backpropagation Through Time

- All the contributions are then summed to compute the gradient with respect to U:

$$\frac{\partial E}{\partial U} = \sum_{t=0}^{T} \frac{\partial E_t}{\partial U}$$

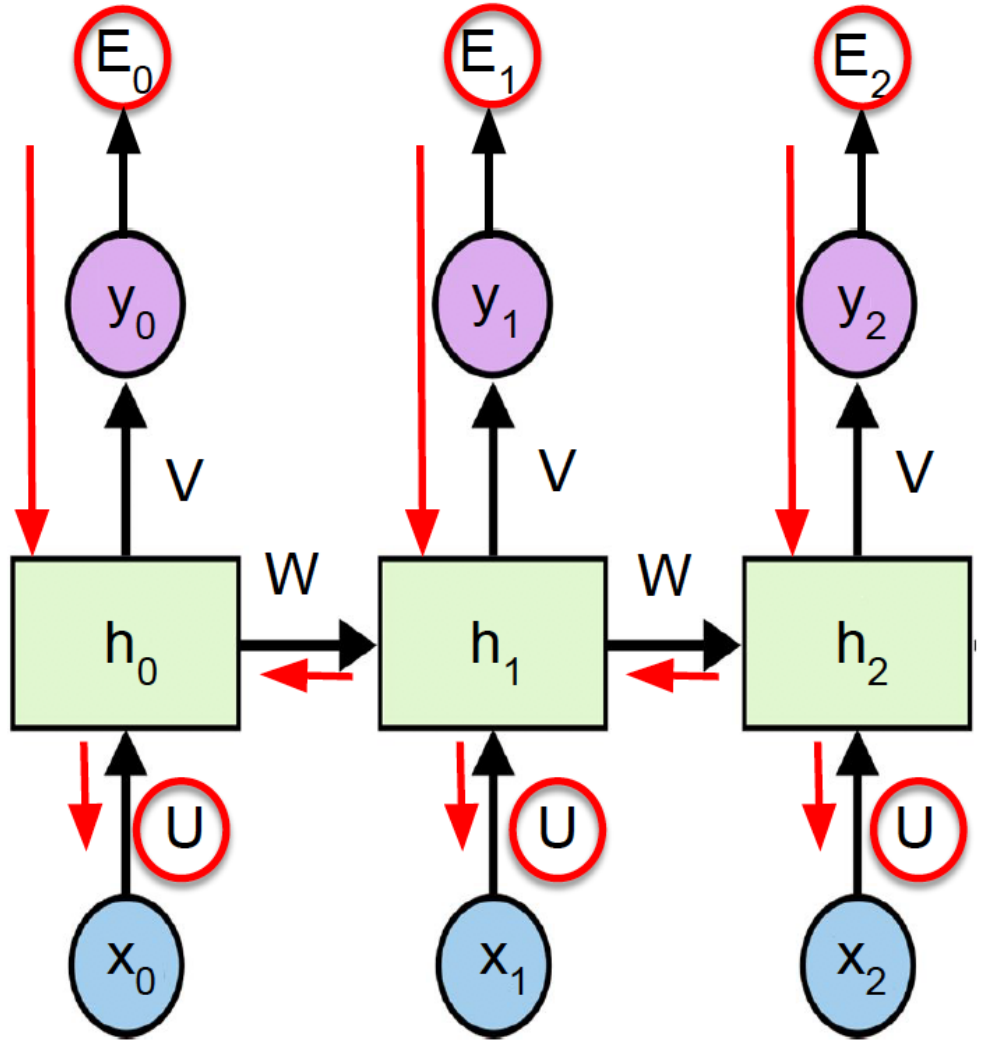


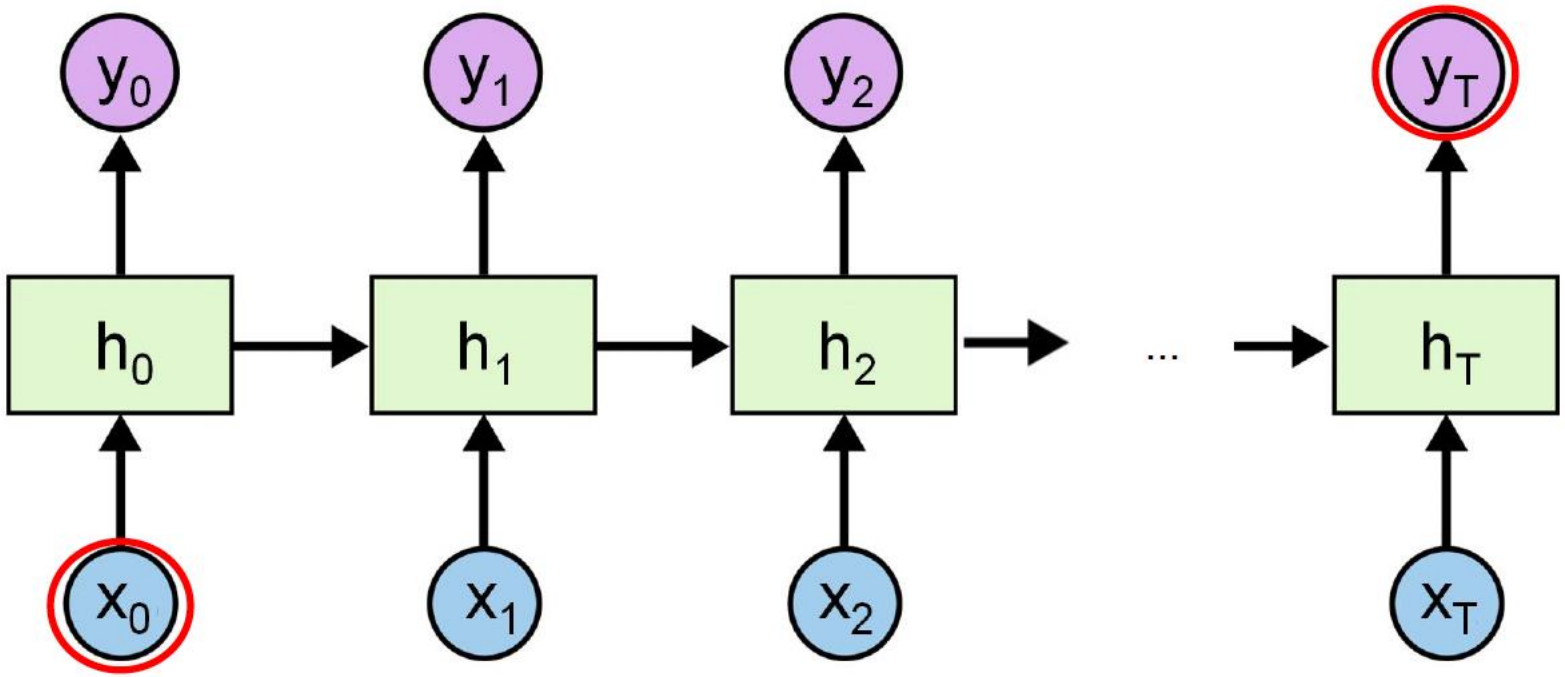
Image from Christopher Olah's blog

Topics

- ***Deep learning for NLP***
- ***RNN***
- ***Training RNNs***
- Training Problems
- RNN Architectures
- Deep RNNs

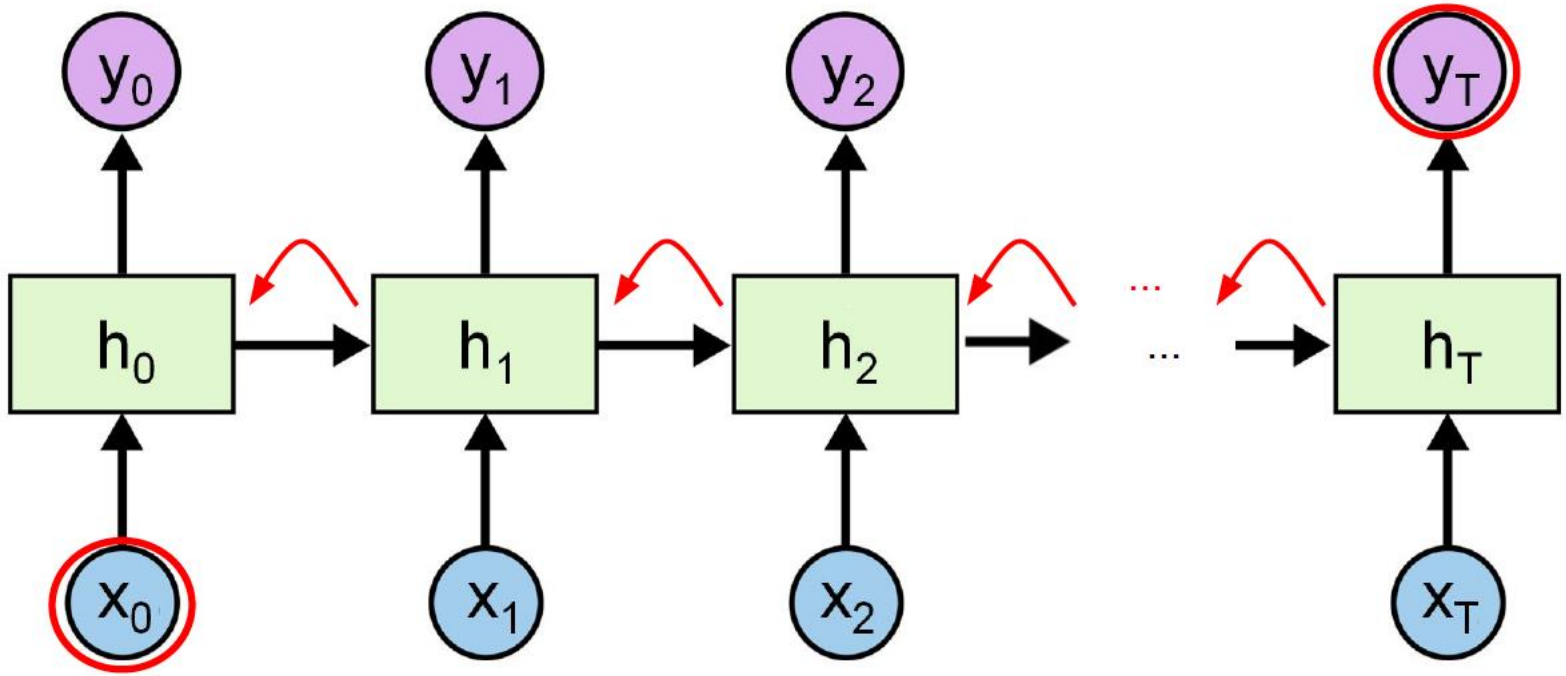
Long-Term Dependencies

- For long sequences, it may be important to capture long-term dependencies.



Long-Term Dependencies

- The problem is the long chain of gradients: $\frac{\partial h_T}{\partial h_{T-1}} \cdots \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0}$



Long-Term Dependencies

- Going back to the main equation for the internal state:

$$h_t = tanh(Ux_t + Wh_{t-1} + b_h)$$

- For a generic h_t, the gradient with respect to the internal state at the previous time step is:

$$\frac{\partial h_t}{\partial h_{t-1}} = W \frac{\partial\ tanh(Ux_t + Wh_{t-1} + b_h)}{\partial h_{t-1}}$$

- In particular, note the term W.

Long-Term Dependencies

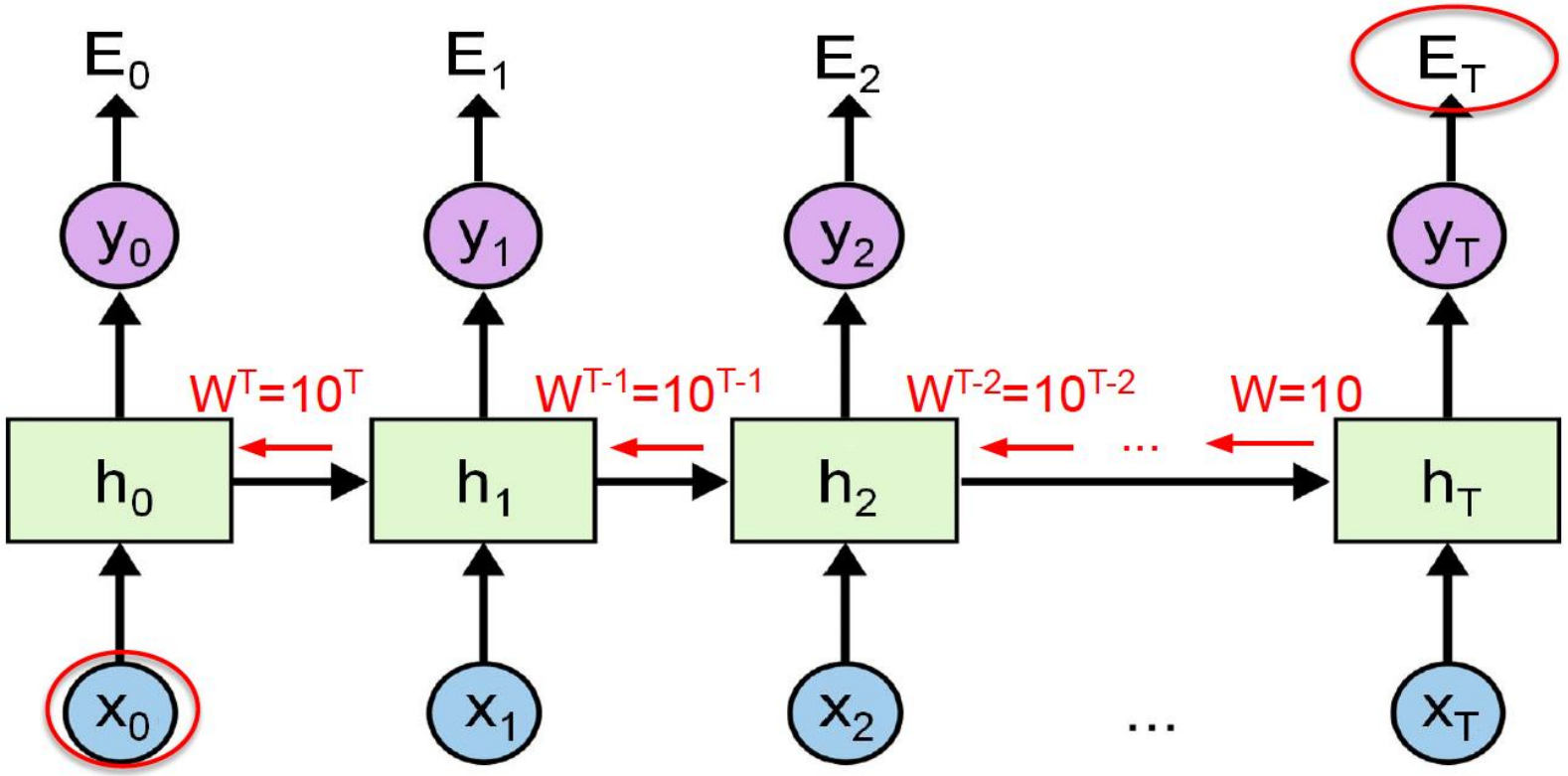
- Given we have a long chain of multiplication...

$$\frac{\partial h_T}{\partial h_{T-1}} \cdots \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0}$$

- ...we multiply by W several times.
- This can can make the system unstable.
- In particular, the result can “explode” or “vanish”.

Exploding Gradient

Simple 1-dimensional example with W = [10]



The gradient increases at every step = exploding gradient!

Exploding Gradient

- The gradient increases at every step ⇒ exploding gradient!
- Problem: the parameters will diverge.
  - Can lead to overflow problems.
- Simple solution: Gradient Clipping.

$$g = \frac{\partial E}{\partial W}$$

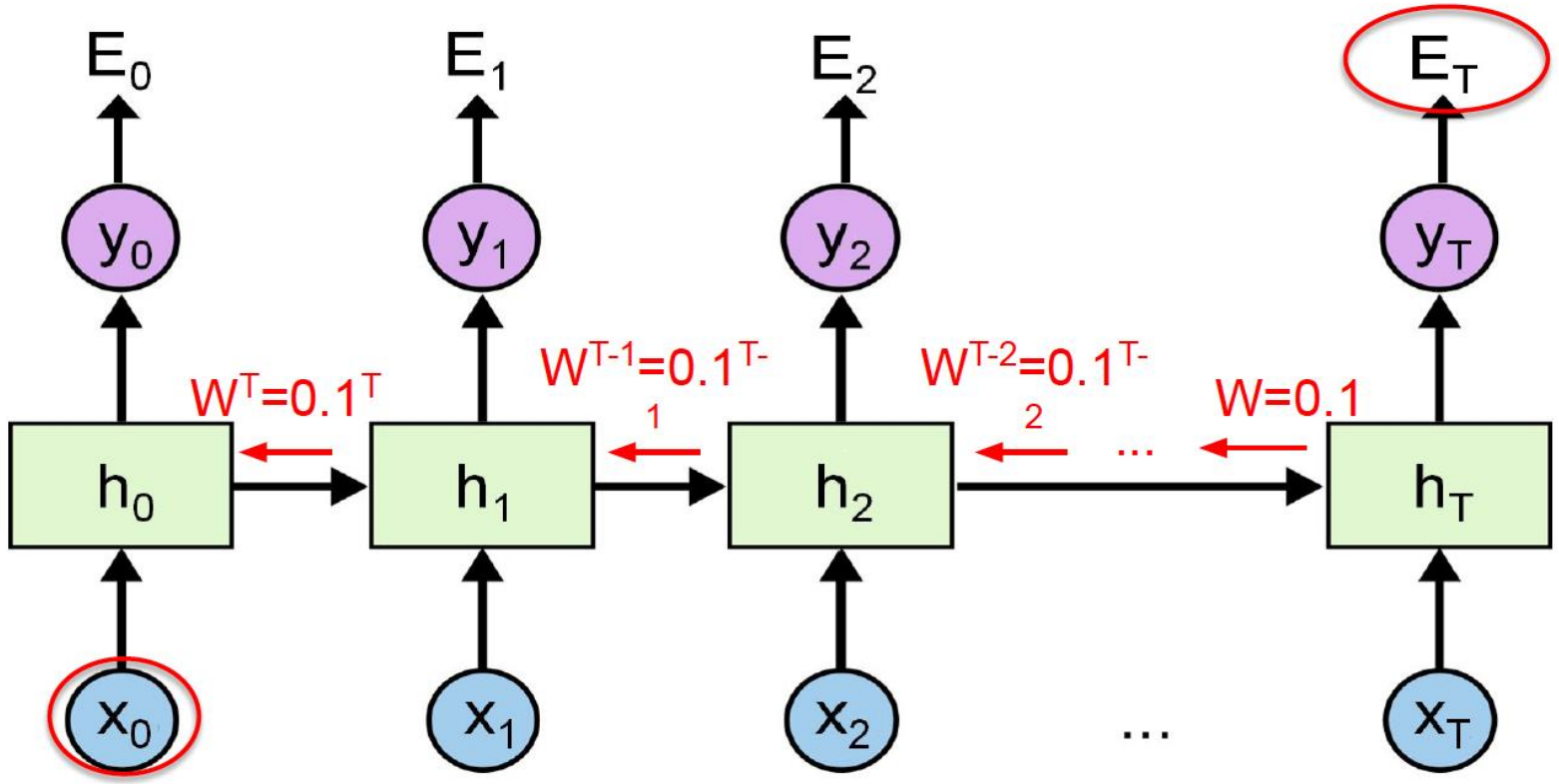
$$\textbf{if } \|g\| \geq threshold \textbf{ then}$$

$$g \leftarrow \frac{threshold}{\|g\|} g$$

- Where $\|\cdot\|$ = L2-norm.

Exploding Gradient

Simple 1-dimensional example with W = [0.1]



The gradient increases at every step = exploding gradient!

Exploding Gradient

- The gradient diminishes at every step ⇒ vanishing gradient!
- Problem: very slow learning (or no learning at all).
  - It affects long-term dependency learning.
- There is no easy solution.
- We need to use more complex RNN architectures.

The End