Quebec
Artificial
Intelligence
Institute

Mila

Introduction to Recurrent Neural Networks

Mirko Bronzi
Applied Research Scientist, Mila
mirko.bronzi@mila.quebec

Plan

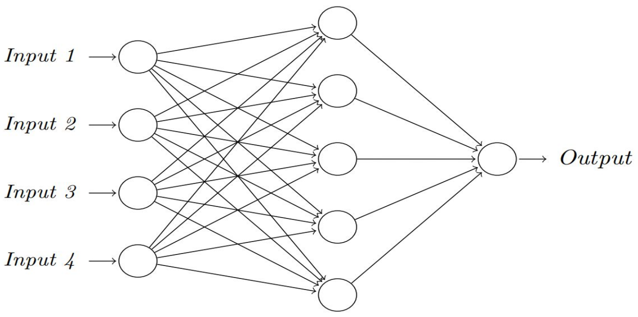
- Motivation
- Introduction to Recurrent Neural Networks (RNNs)
- Training RNNs
- Training problems
- RNN architectures
- Deep RNNs

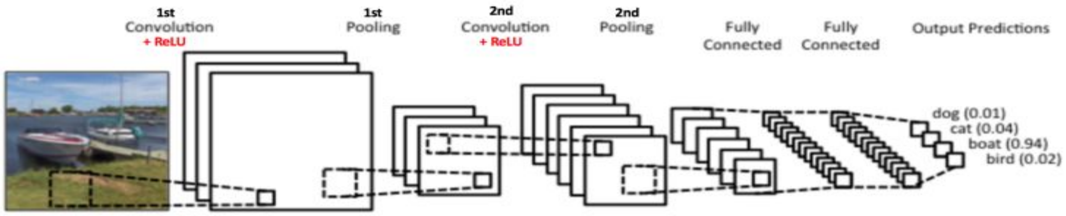
Plan

- **Motivation**
- Introduction to Recurrent Neural Networks (RNNs)
- Training RNNs
- Training problems
- RNN architectures
- Deep RNNs

Motivation

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



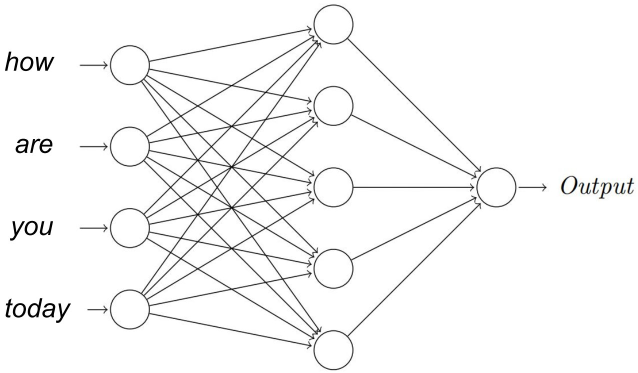


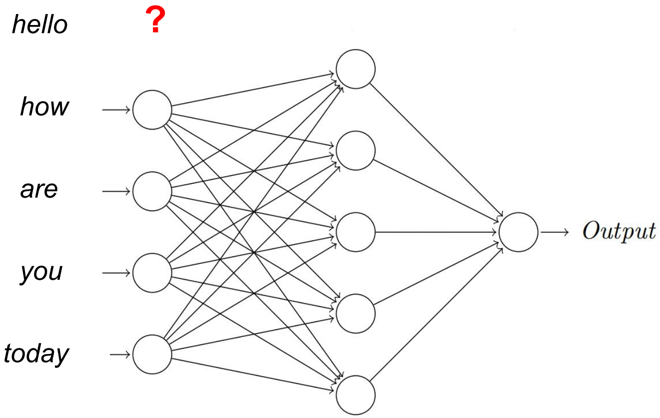
- How can we handle sequences of variable size?

Image: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets

Motivation

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



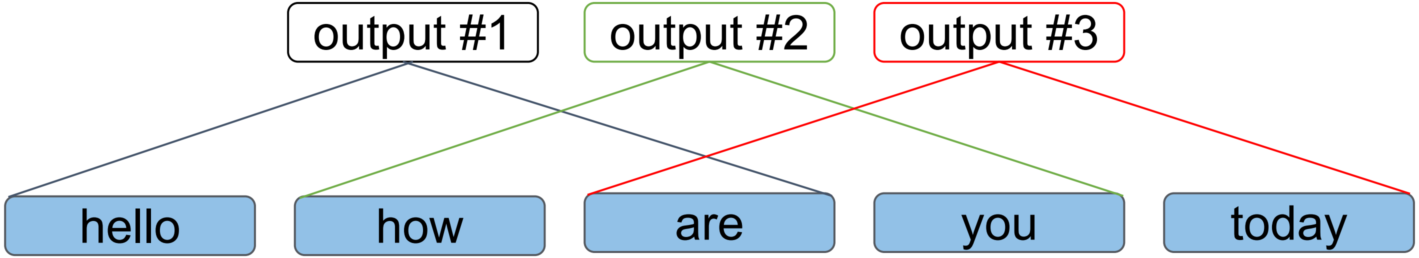


- Some techniques (such as bag-of-words for processing text input) allow an MLP to handle sequences of variable size; but those techniques ignore the order of the elements in the sequence.

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:

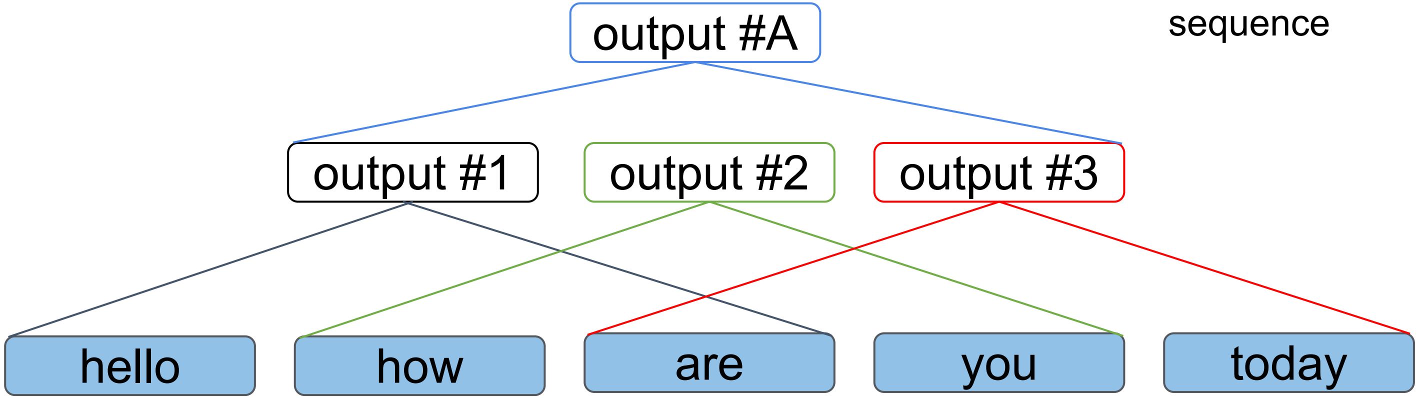


Motivation

- 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:

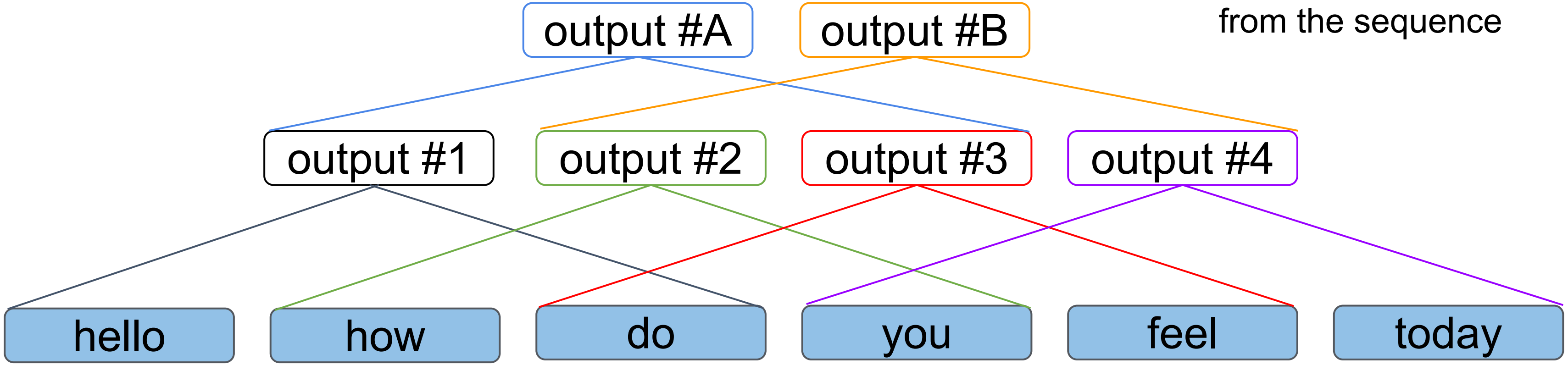
output #A has access to information from the entire sequence



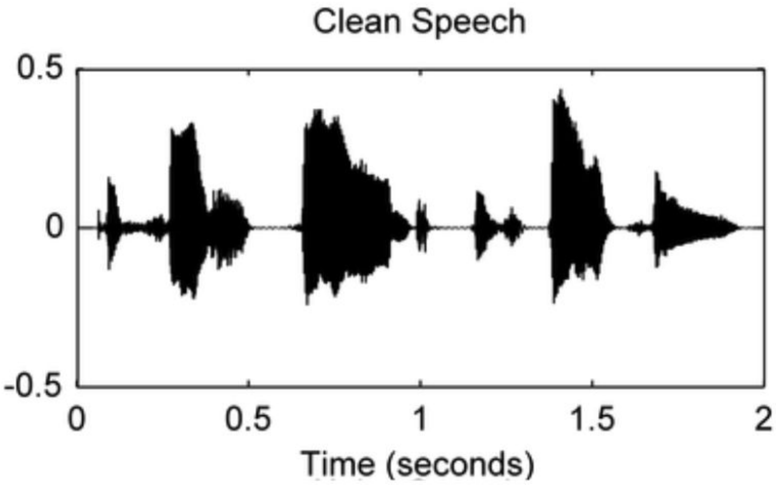
Motivation

- 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

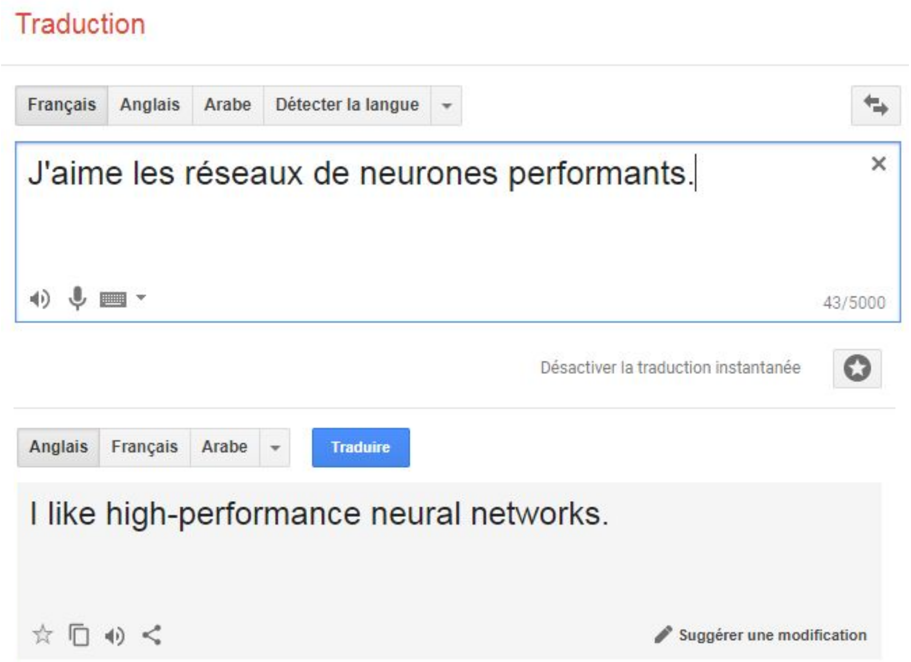


“The fresh bread is baking.”

Speech recognition: Audio sequence → Word sequence.

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

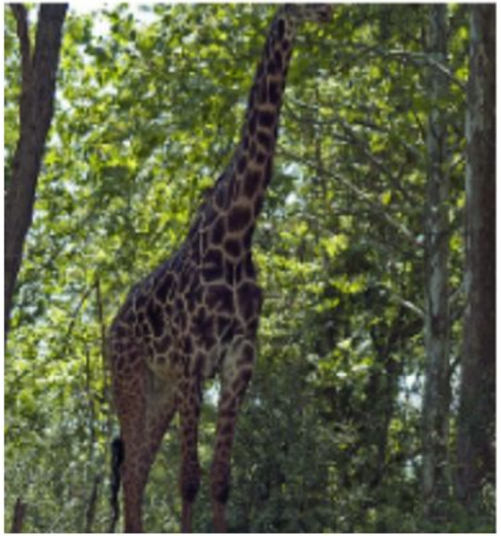
Examples



Translation: Word sequence → Word sequence.

Examples

A woman is throwing a frisbee in a park.



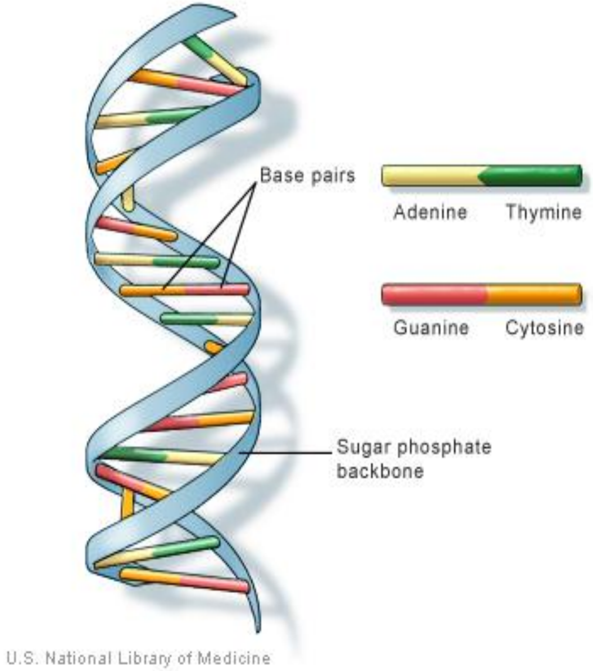
A giraffe standing in a forest with trees in the background.

Caption generation: Image → Word sequence.

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

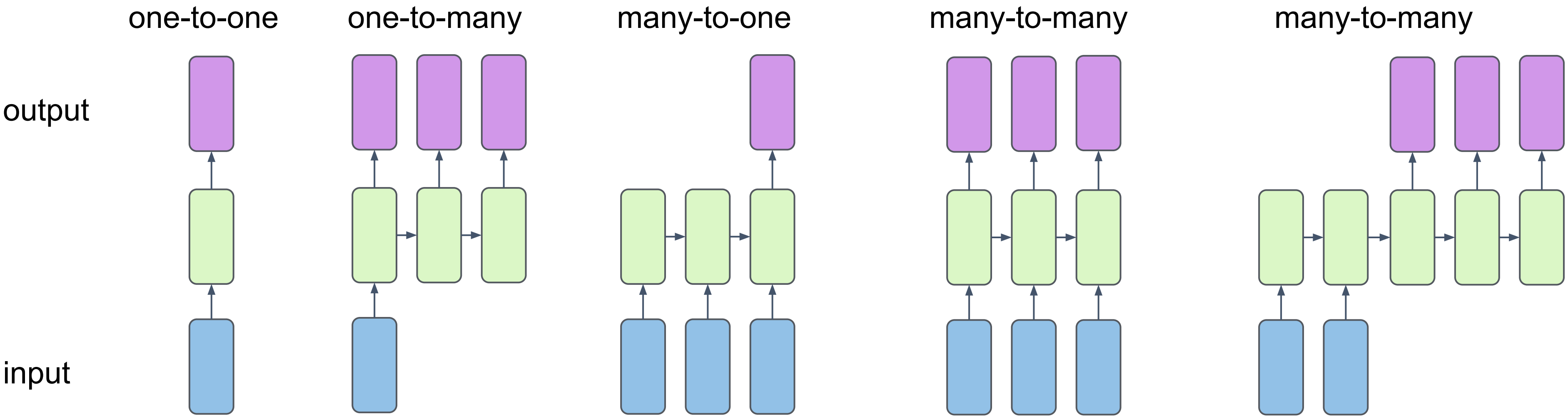
Examples

- 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…

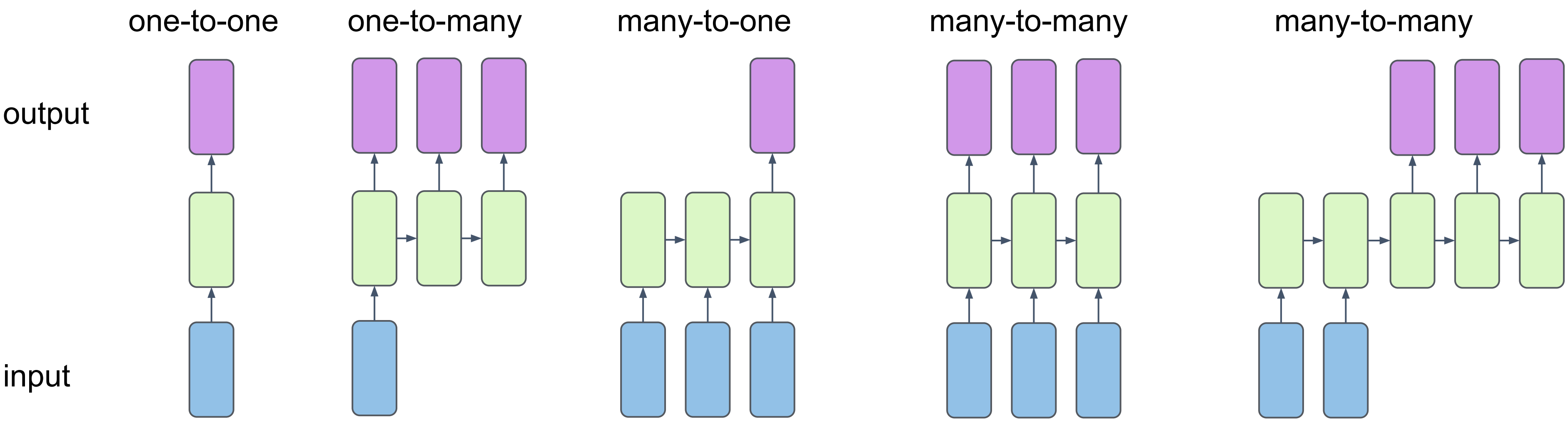


There is a lot of data with sequences!

Modeling Sequences



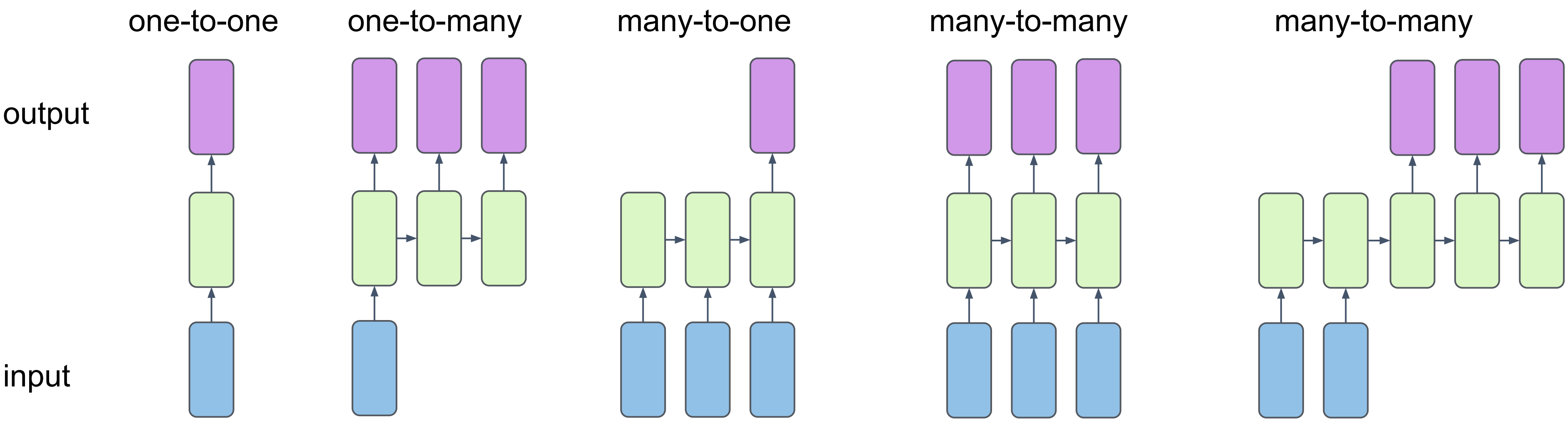
Modeling Sequences



Object classification

Image → Class

Modeling Sequences



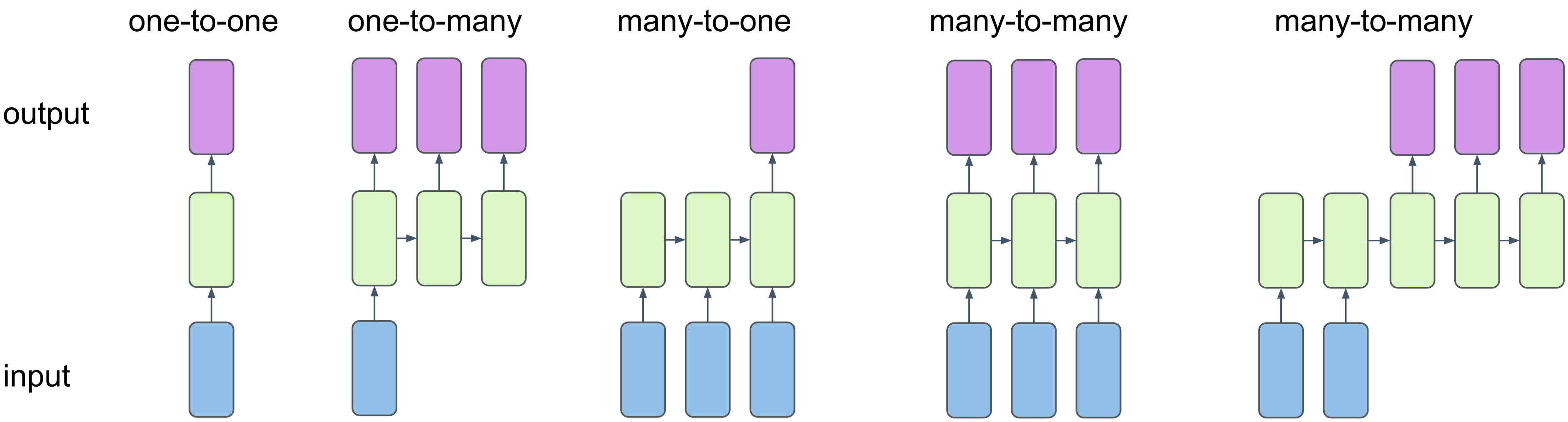
Object classification

Image → Class

Caption generation

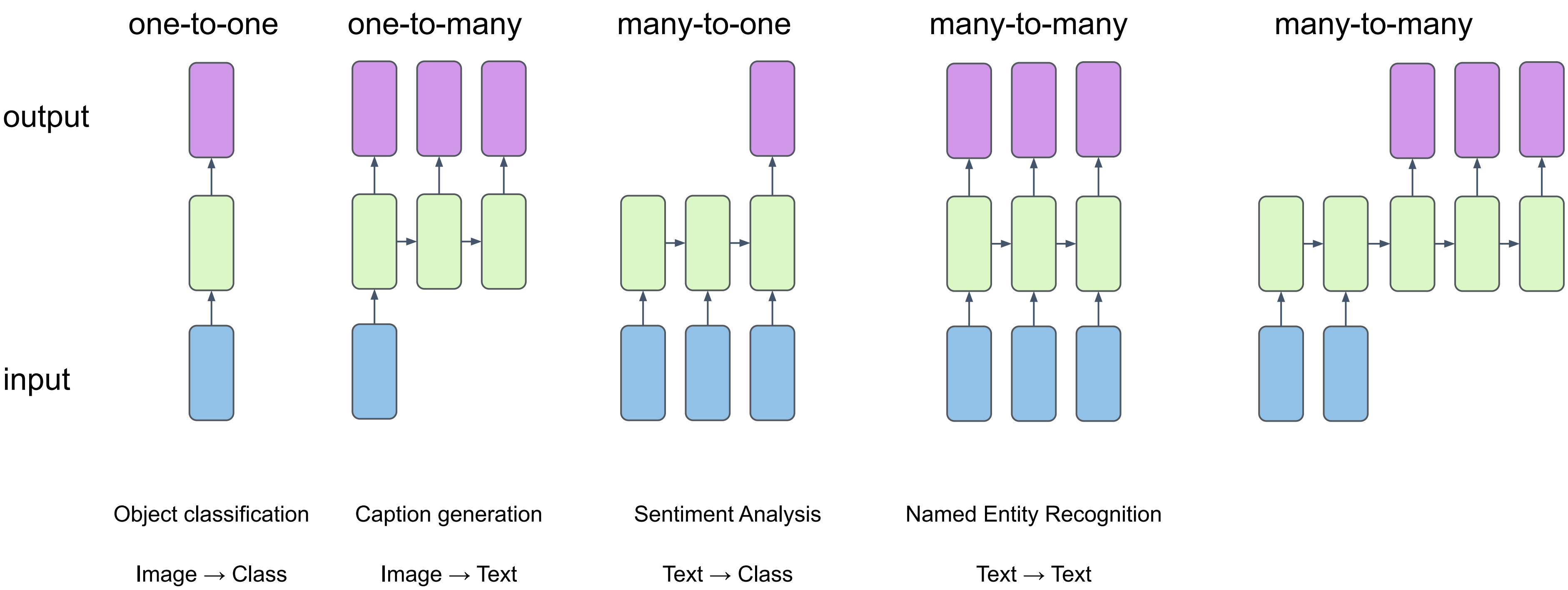
Image → Text

Modeling Sequences

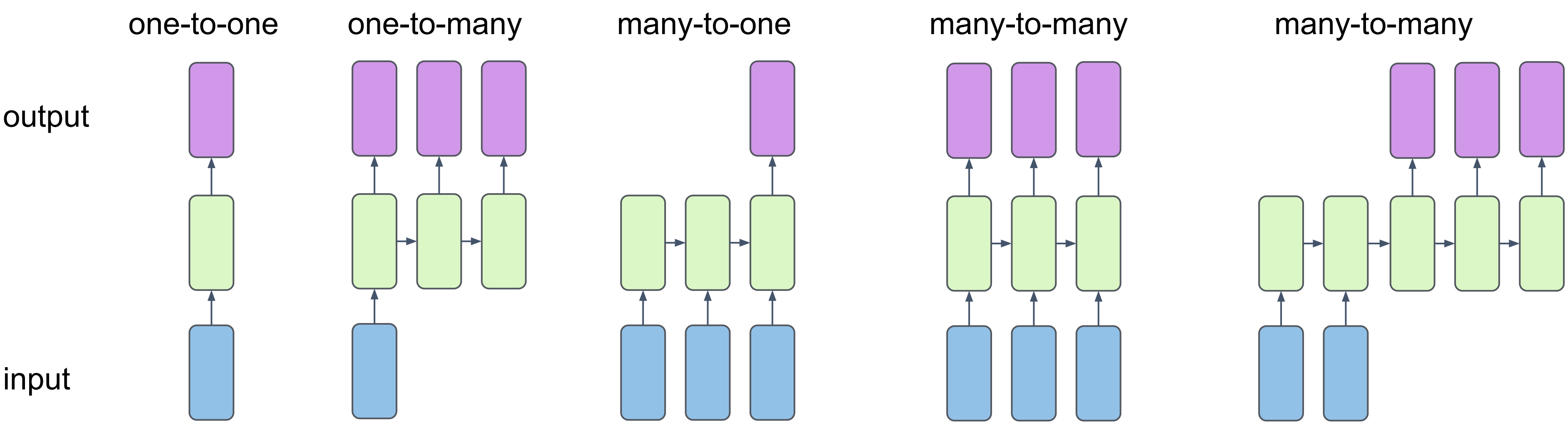


Object classification	Caption generation	Sentiment Analysis
Image → Class	Image → Text	Text → Class

Modeling Sequences



Modeling Sequences



Object classification — Image → Class

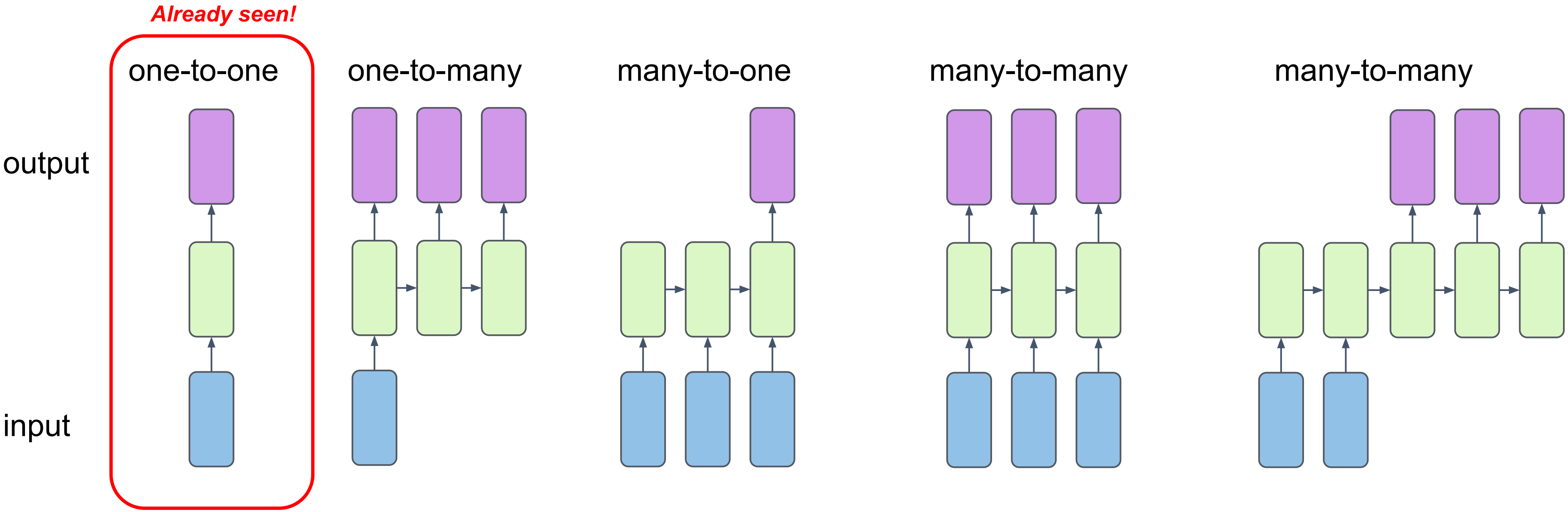
Caption generation — Image → Text

Sentiment Analysis — Text → Class

Named Entity Recognition — Text → Text

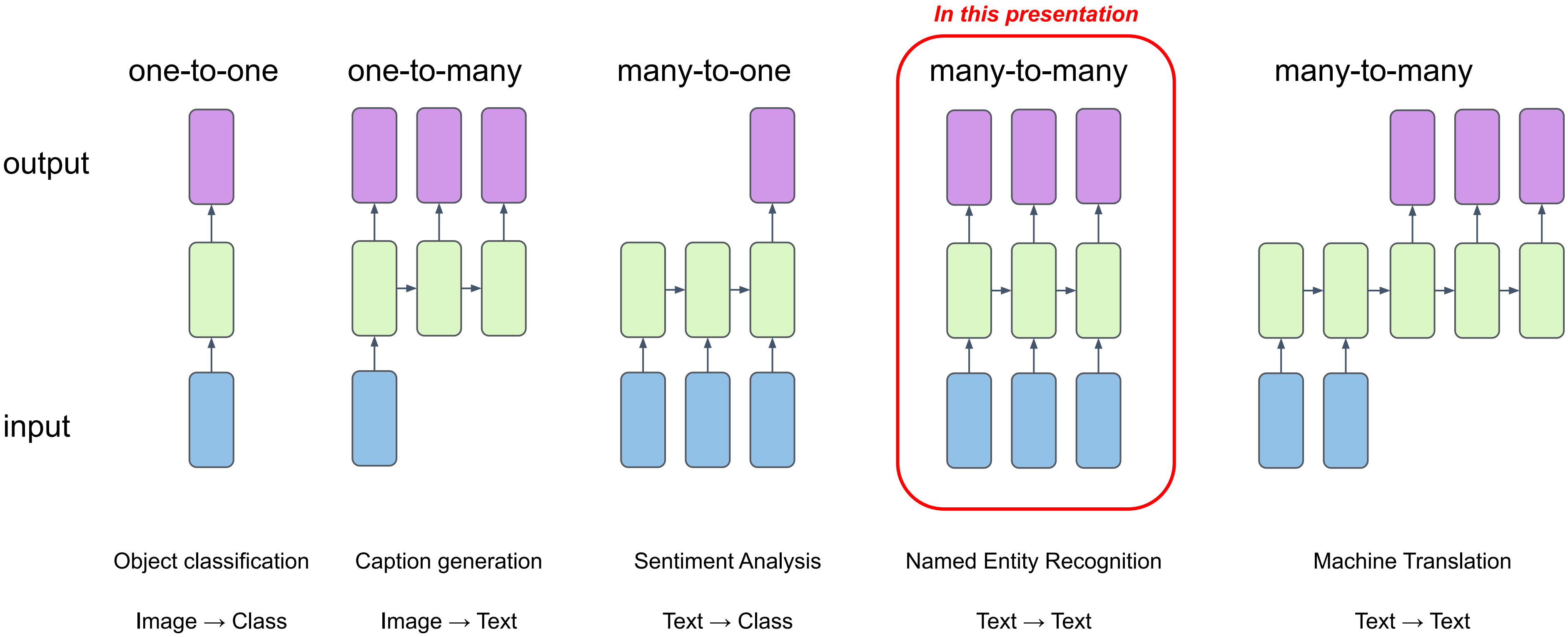
Machine Translation — Text → Text

Modeling Sequences



Object classification	Caption generation	Sentiment Analysis	Named Entity Recognition	Machine Translation
Image → Class	Image → Text	Text → Class	Text → Text	Text → Text

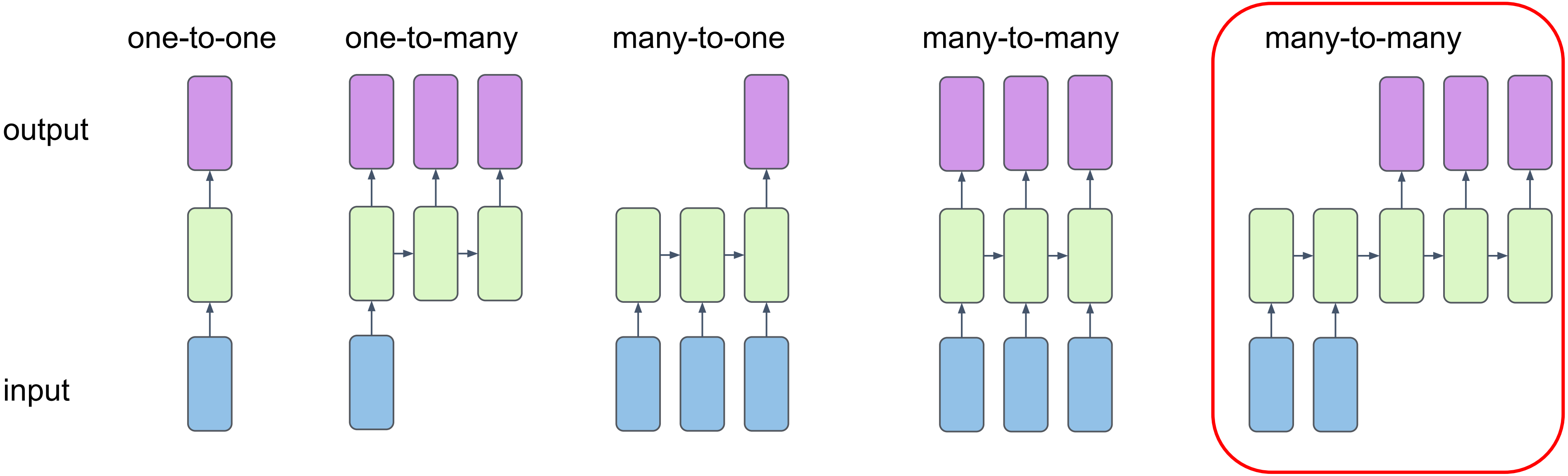
Modeling Sequences



Mila

Modeling Sequences

In the next presentation



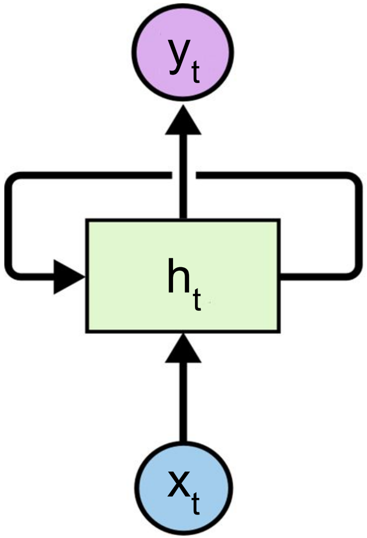
Object classification	Caption generation	Sentiment Analysis	Named Entity Recognition	Machine Translation
Image → Class	Image → Text	Text → Class	Text → Text	Text → Text

Plan

- Motivation
- **Introduction to Recurrent Neural Networks (RNNs)**
- Training RNNs
- Training problems
- RNN architectures
- Deep RNNs

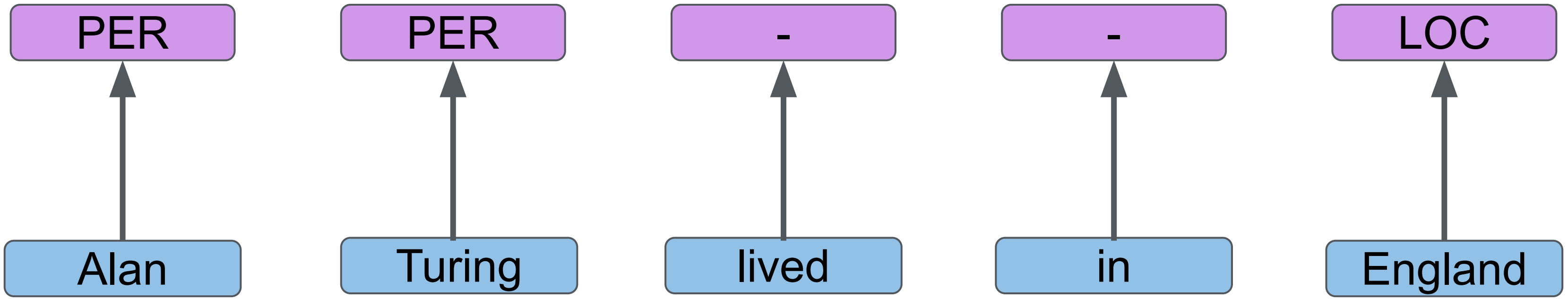
Recurrent Neural Networks

- A Recurrent Neural Network (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.



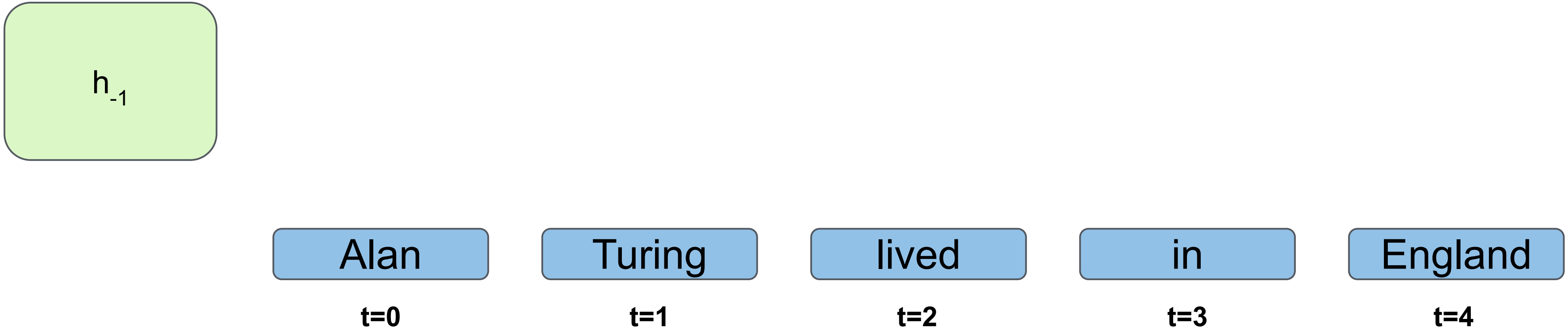
Recurrent Neural Networks - Example

- 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).



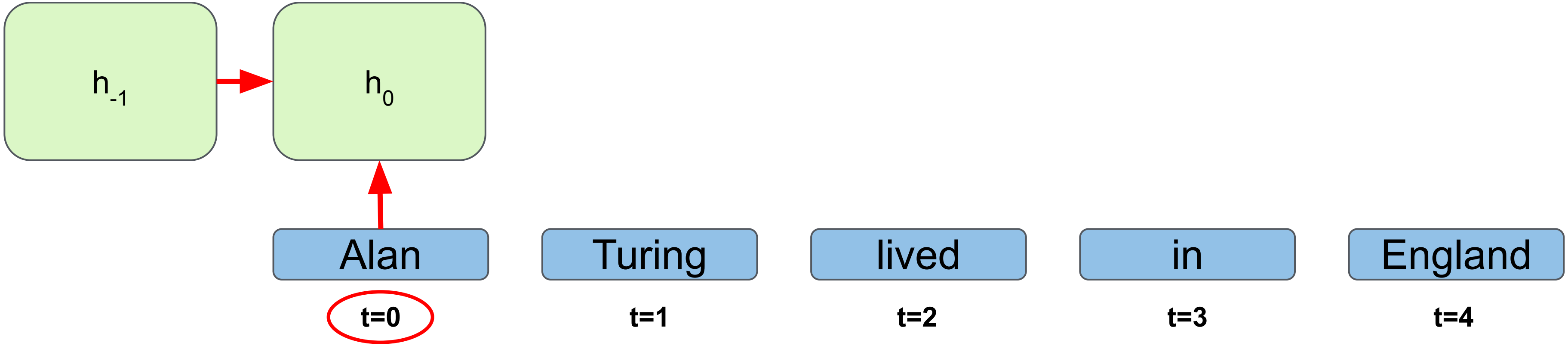
Recurrent Neural Networks - Example

- 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.



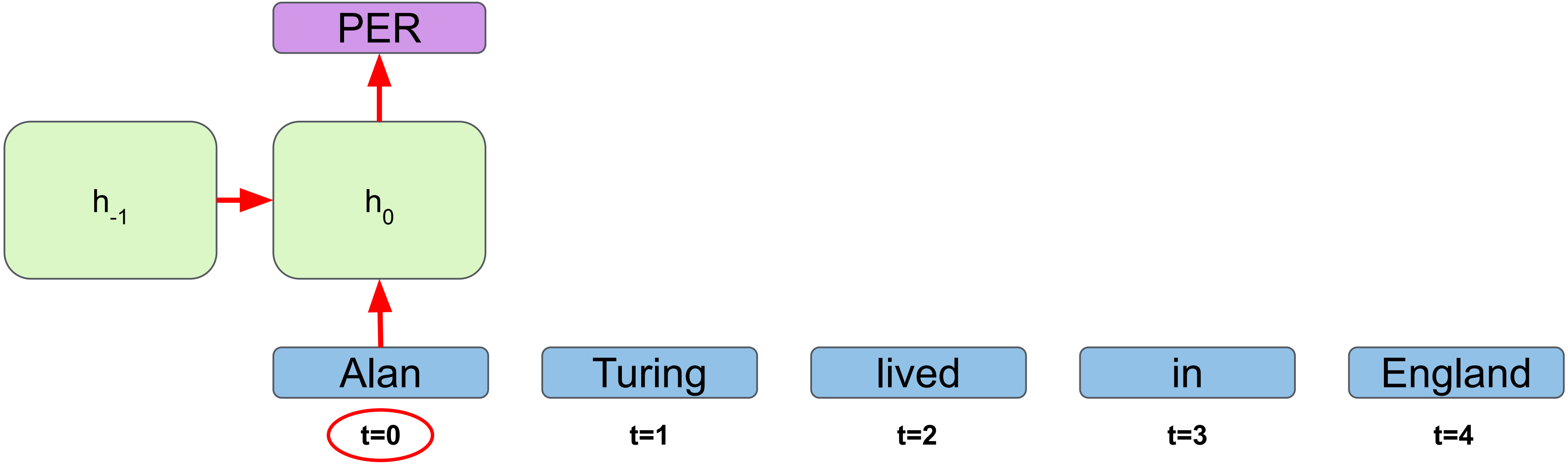
Recurrent Neural Networks - Example

- 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.



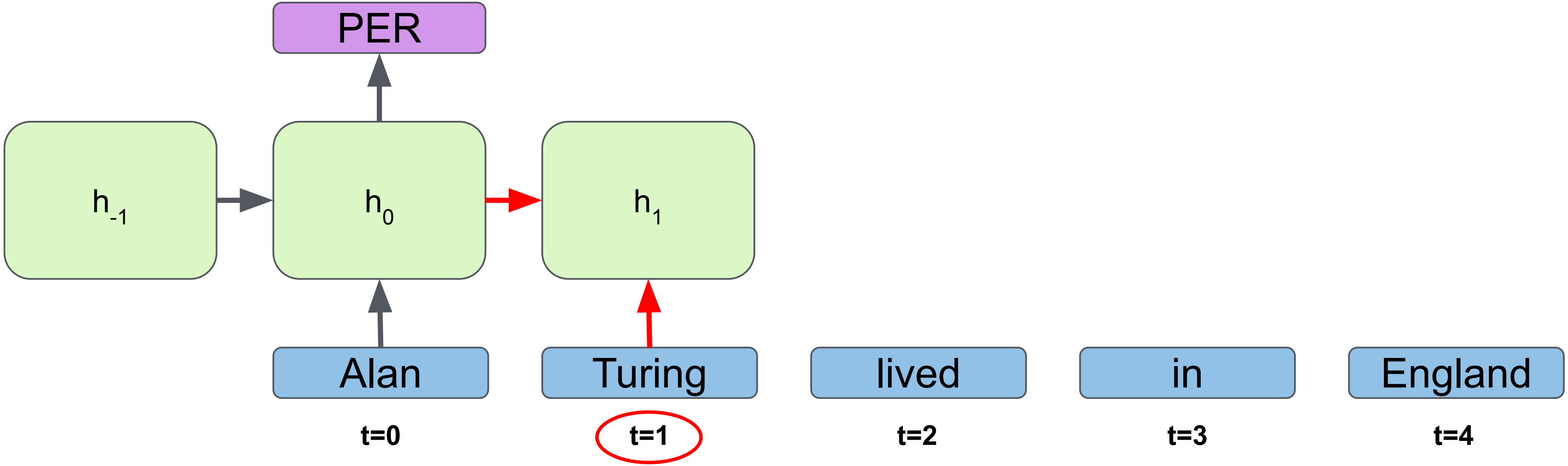
Recurrent Neural Networks - Example

- 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

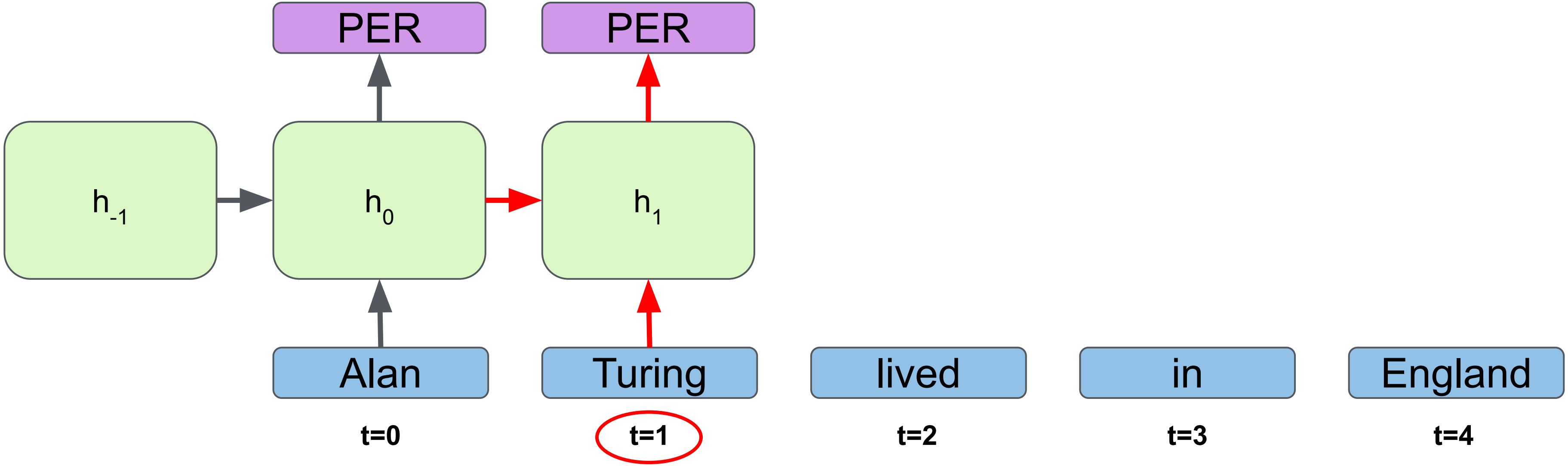
- A RNN applies a function to an input sequence - **one element at a time** - in order to generate an output sequence while maintaining an internal state.



Mila

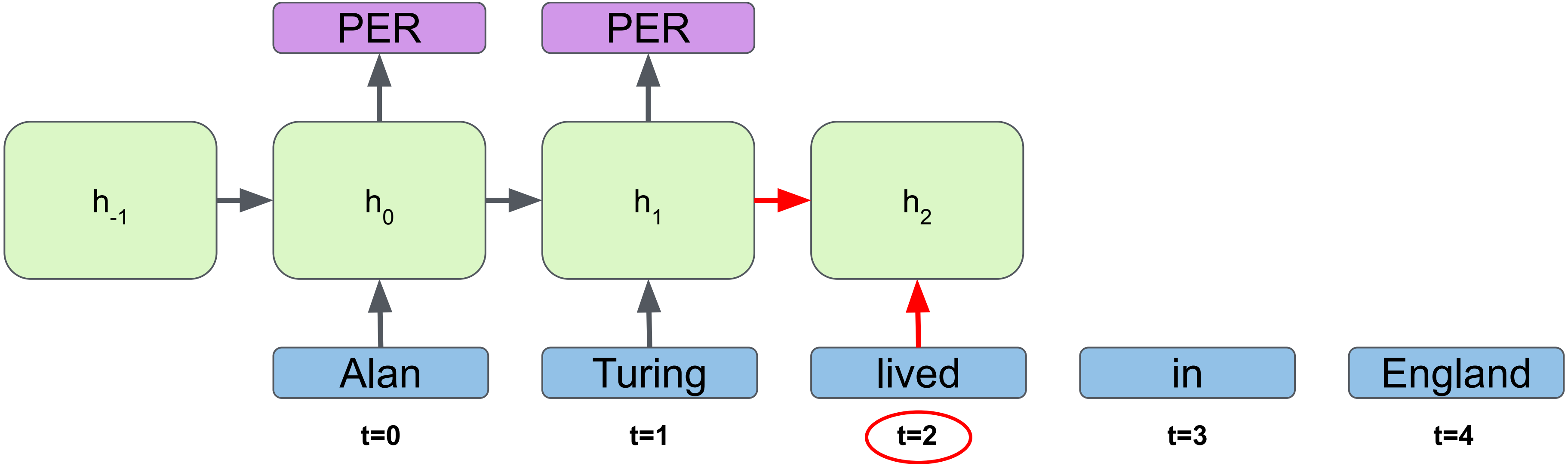
Recurrent Neural Networks - Example

- 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.



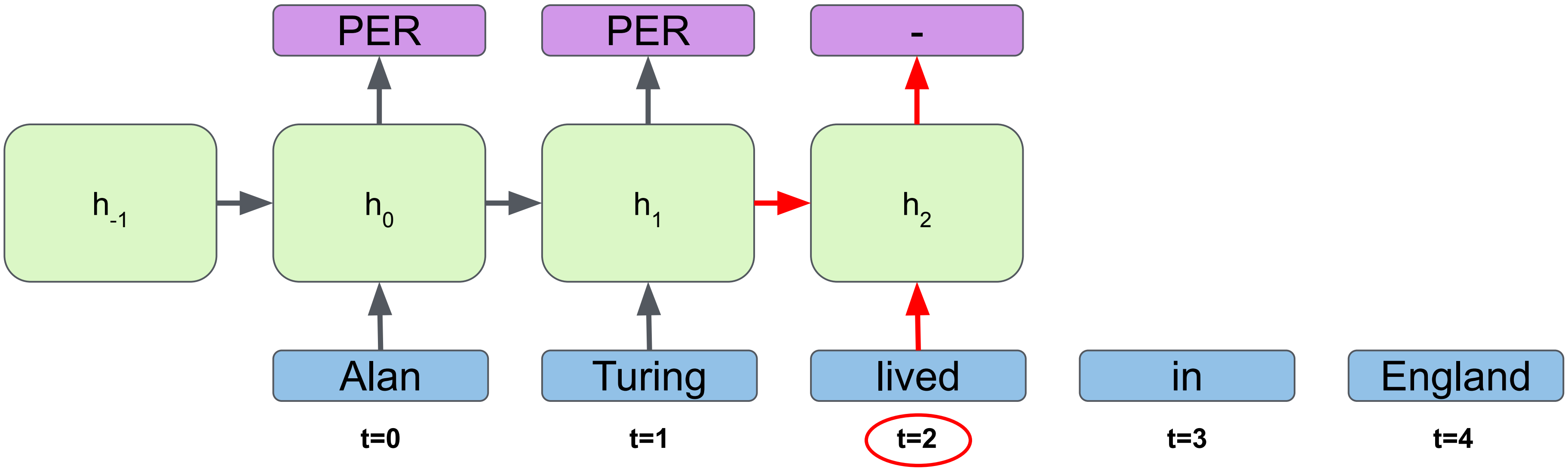
Recurrent Neural Networks - Example

- 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.



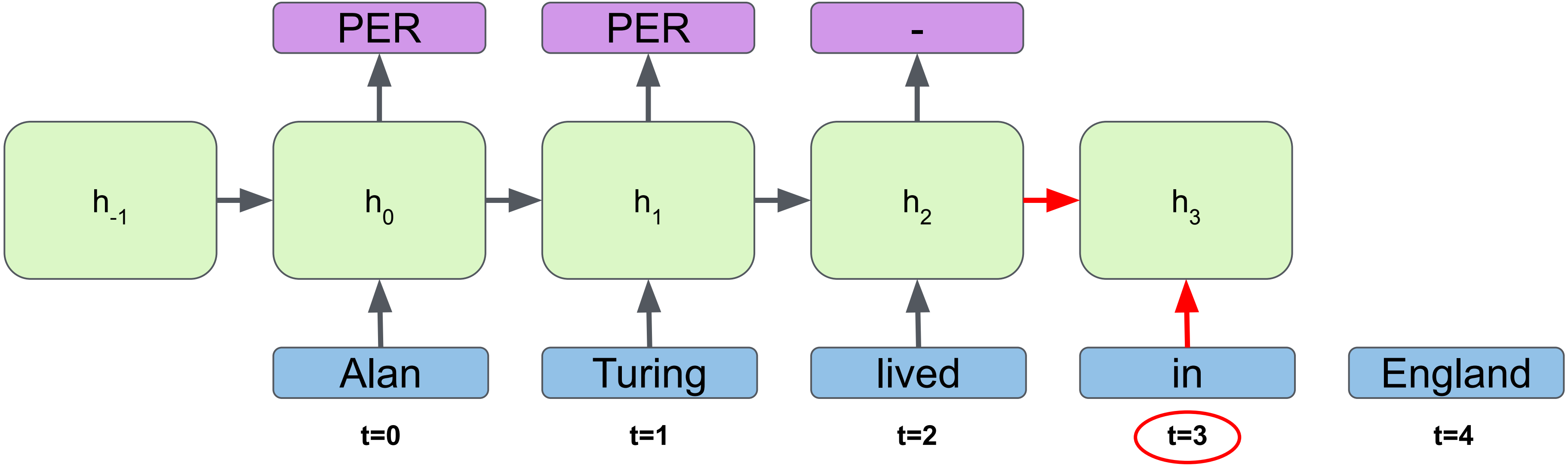
Recurrent Neural Networks - Example

- 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.



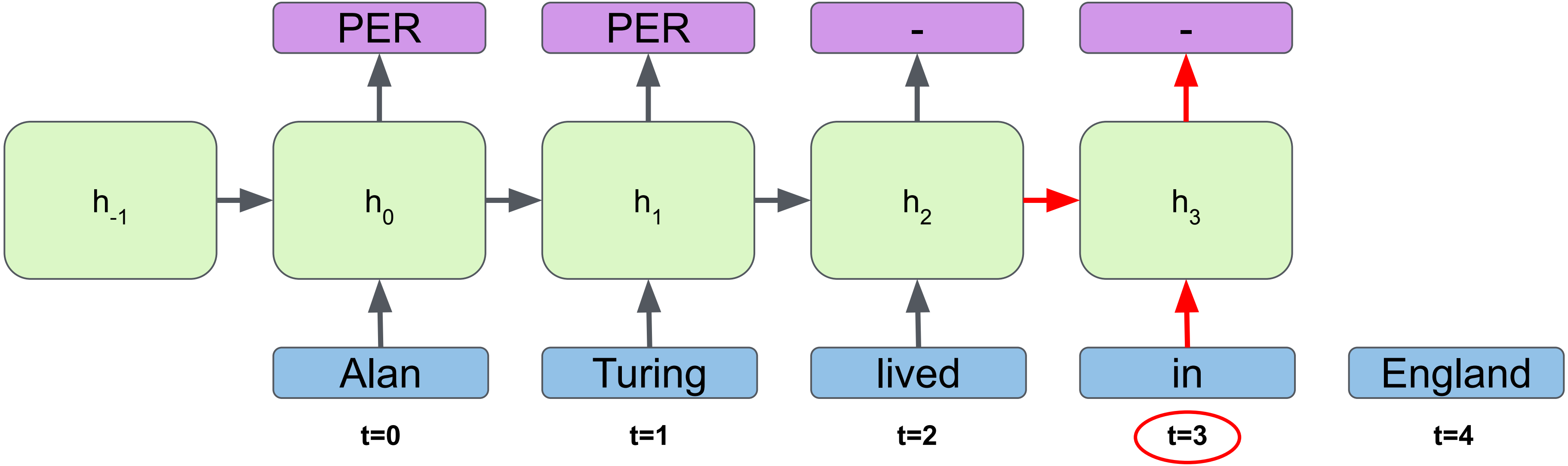
Recurrent Neural Networks - Example

- 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

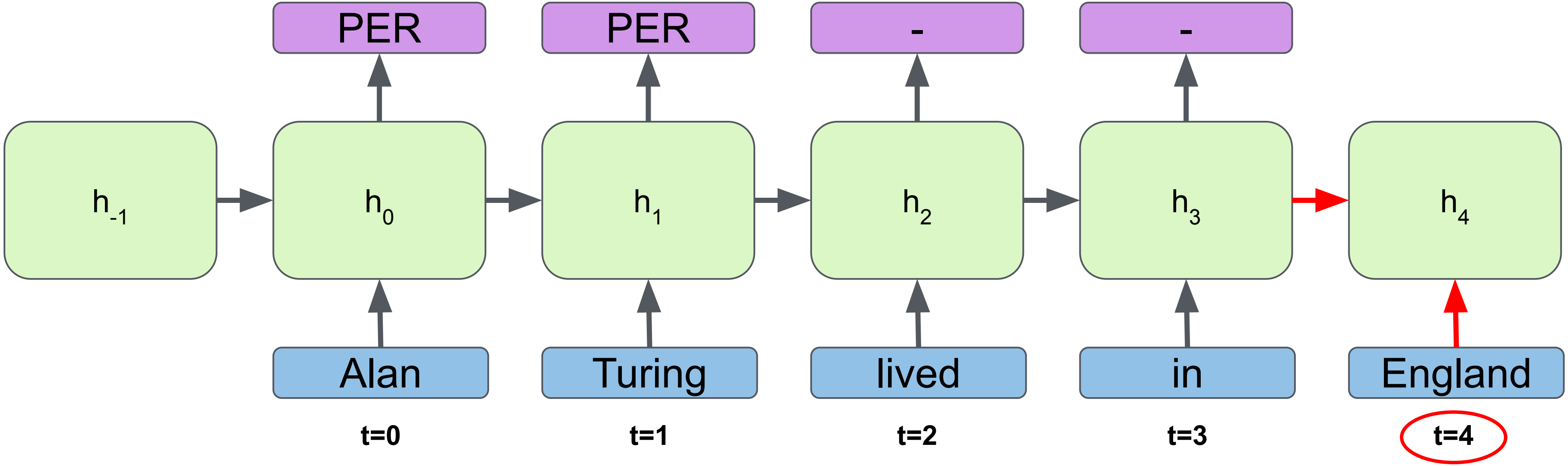
- A RNN applies a function to an input sequence - **one element at a time** - in order to generate an output sequence while maintaining an internal state.



Mila

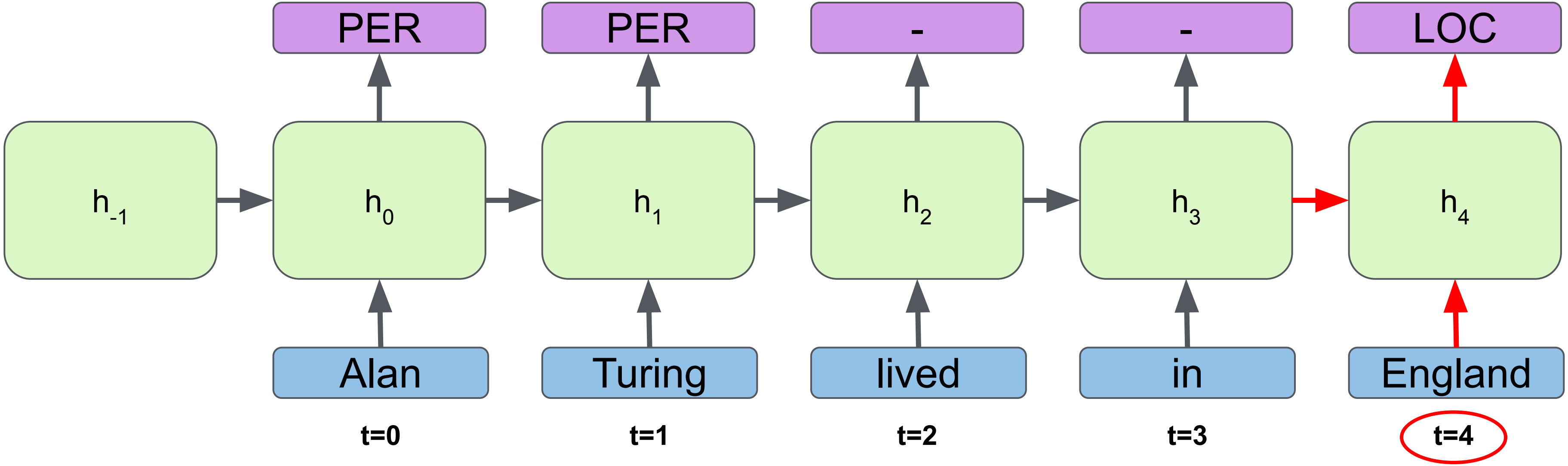
Recurrent Neural Networks - Example

- 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

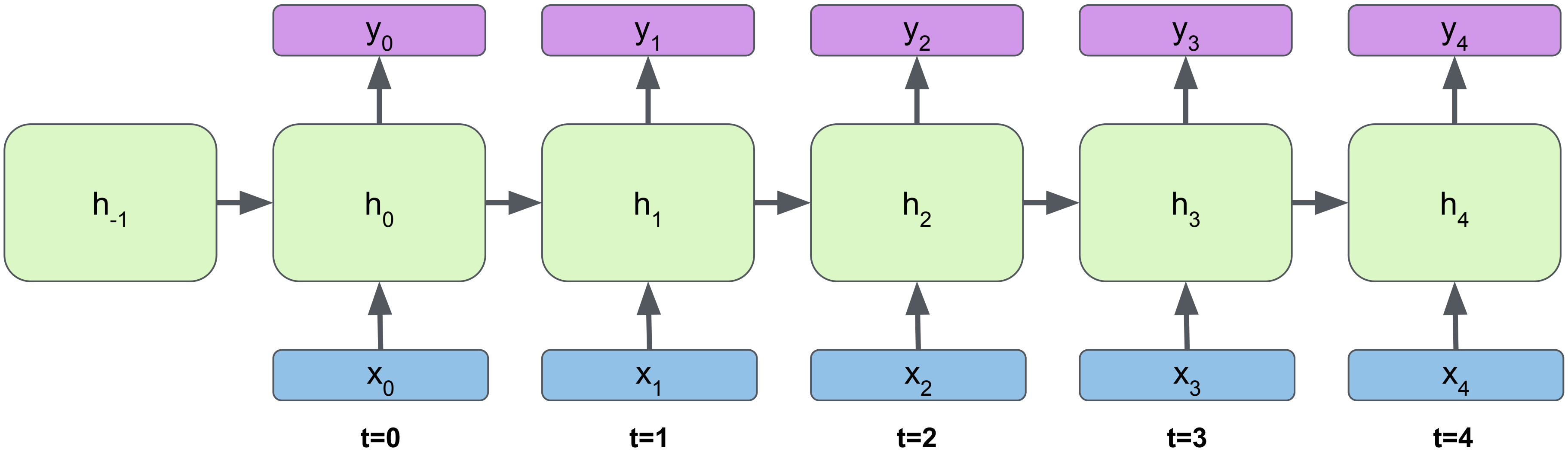
- A RNN applies a function to an input sequence - **one element at a time** - in order to generate an output sequence while maintaining an internal state.



Mila

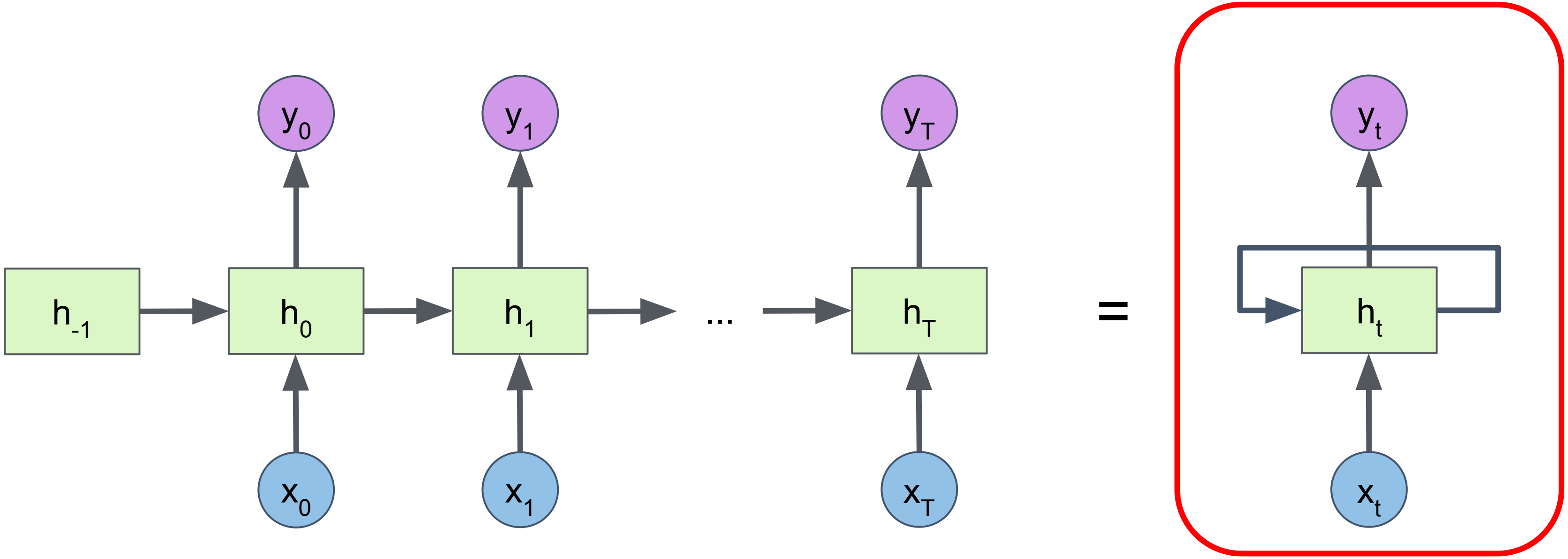
Recurrent Neural Networks - Formalization

- 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:



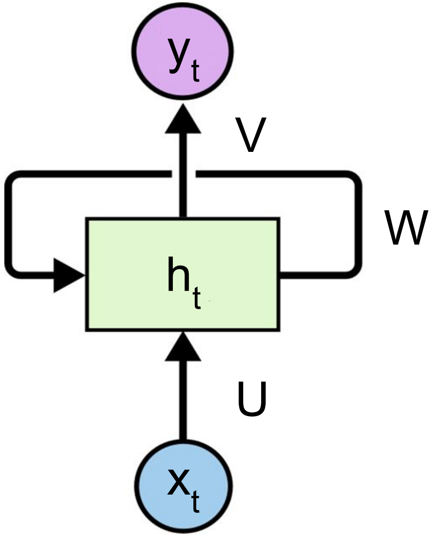
Recurrent Neural Networks - Implementation

- Most simple implementation:

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

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

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



g=softmax, sigmoid, ...

Image from Christopher Olah's blog

Recurrent Neural Networks - Implementation

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

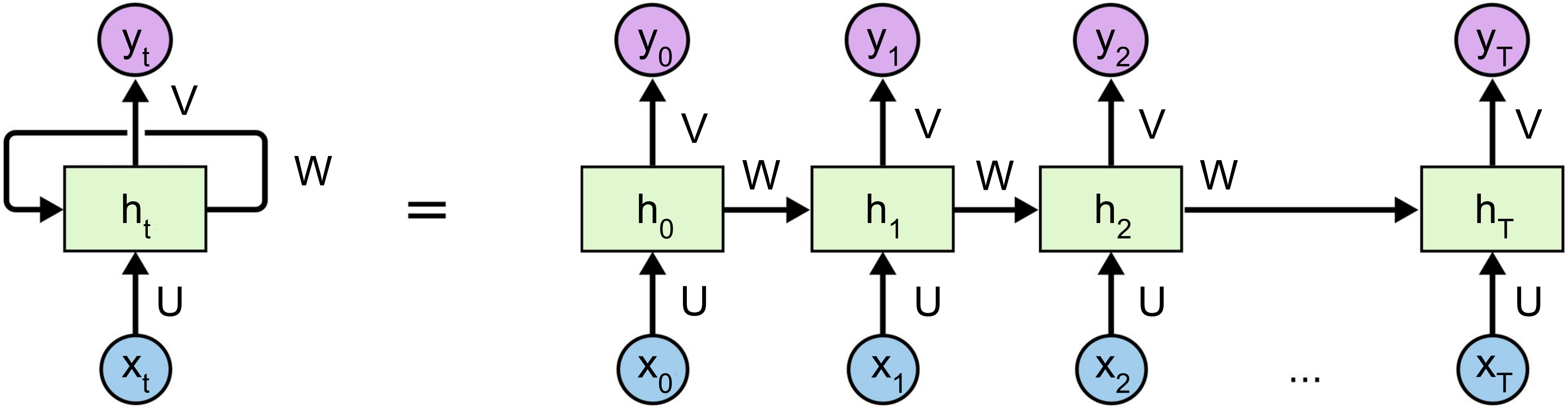


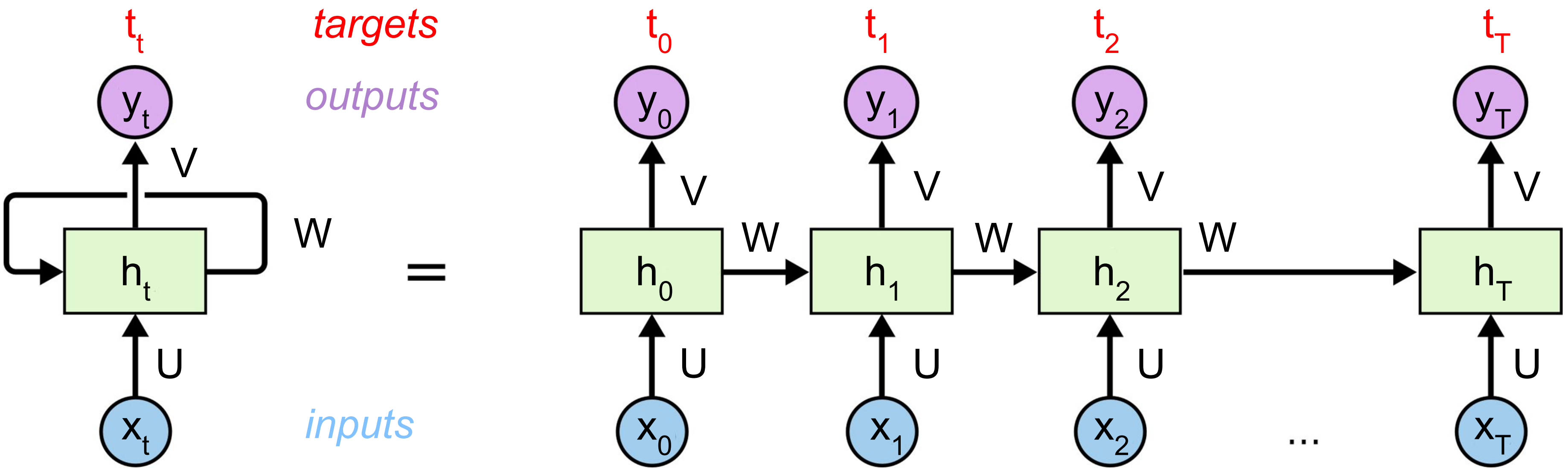
Image from Christopher Olah's blog

The initial internal state (h_{-1}) is dropped for simplicity

Plan

- Motivation
- Introduction to Recurrent Neural Networks (RNNs)
- **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.

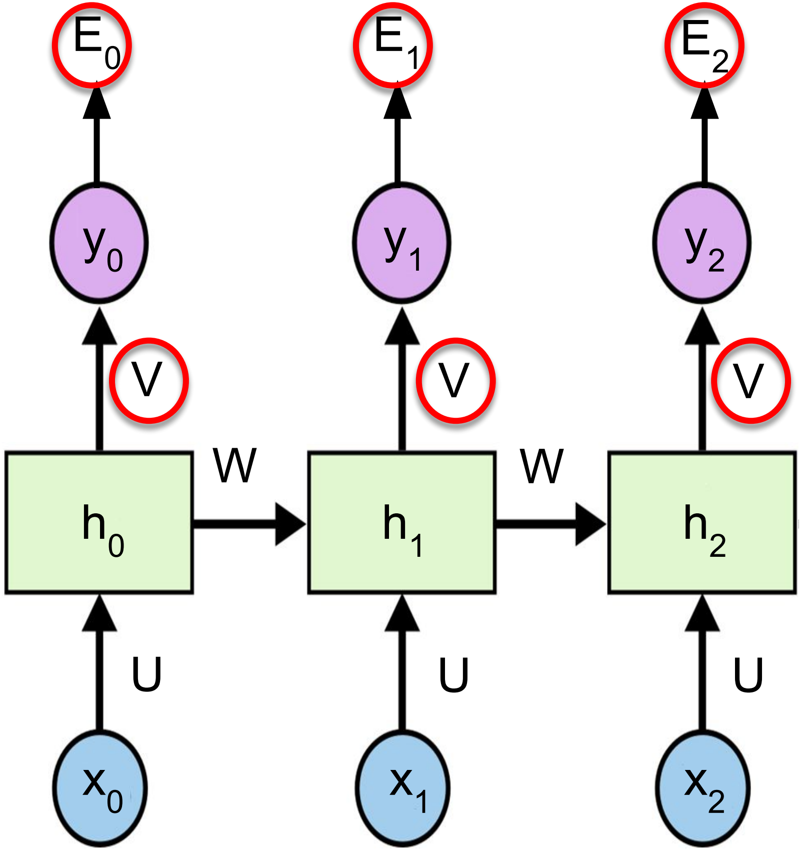


Image from Christopher Olah's blog

Backpropagation

- We start with E_2...

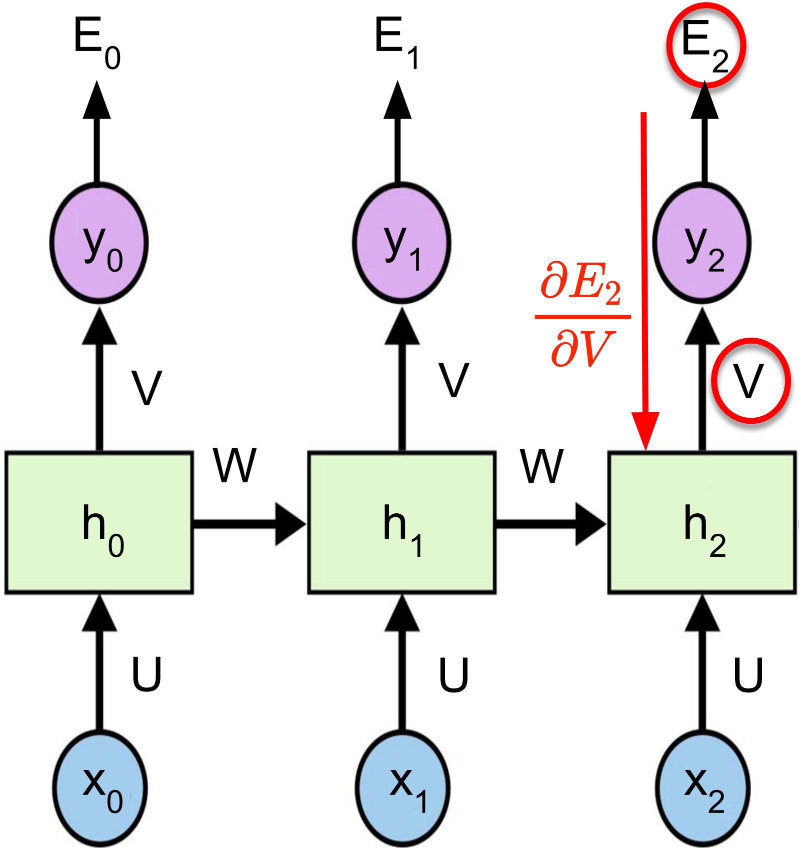


Image from Christopher Olah's blog

Backpropagation

- ... then E_1...

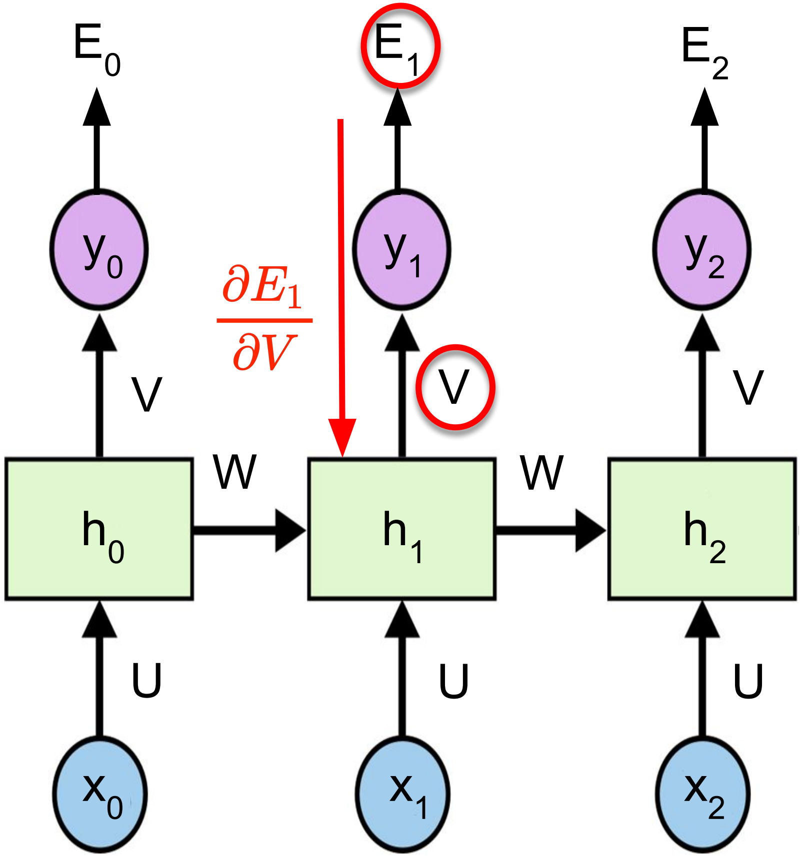


Image from Christopher Olah's blog

Mila

Backpropagation

- ... then E_0.

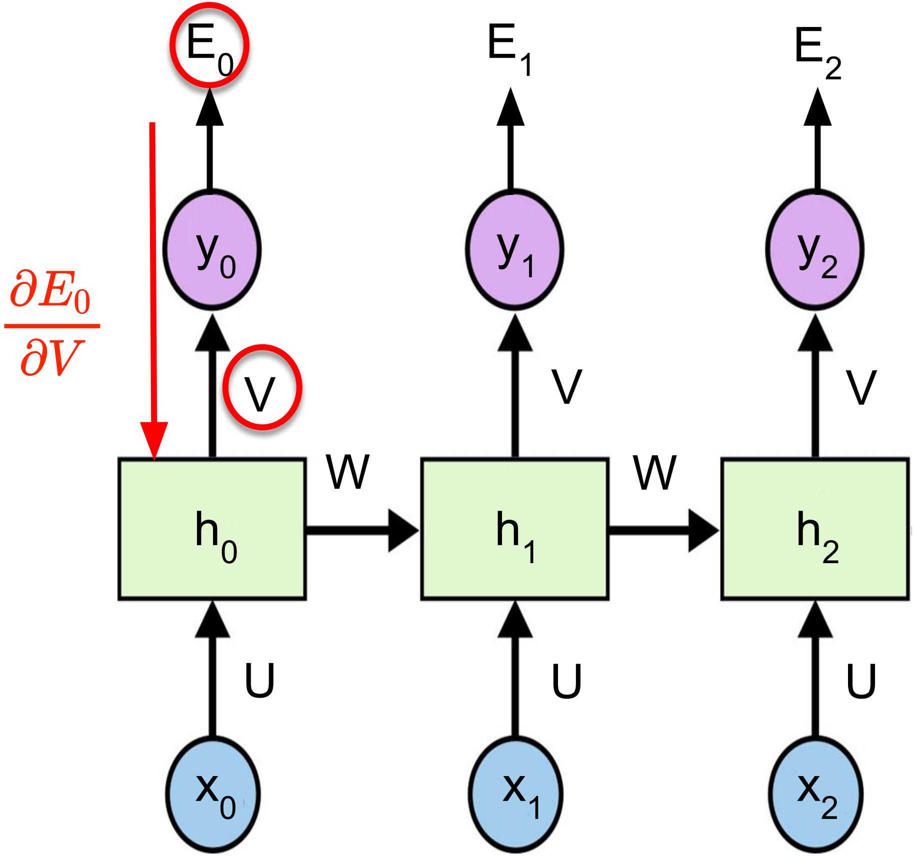


Image from Christopher Olah's blog

Backpropagation

- Now we can just sum the gradients:

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

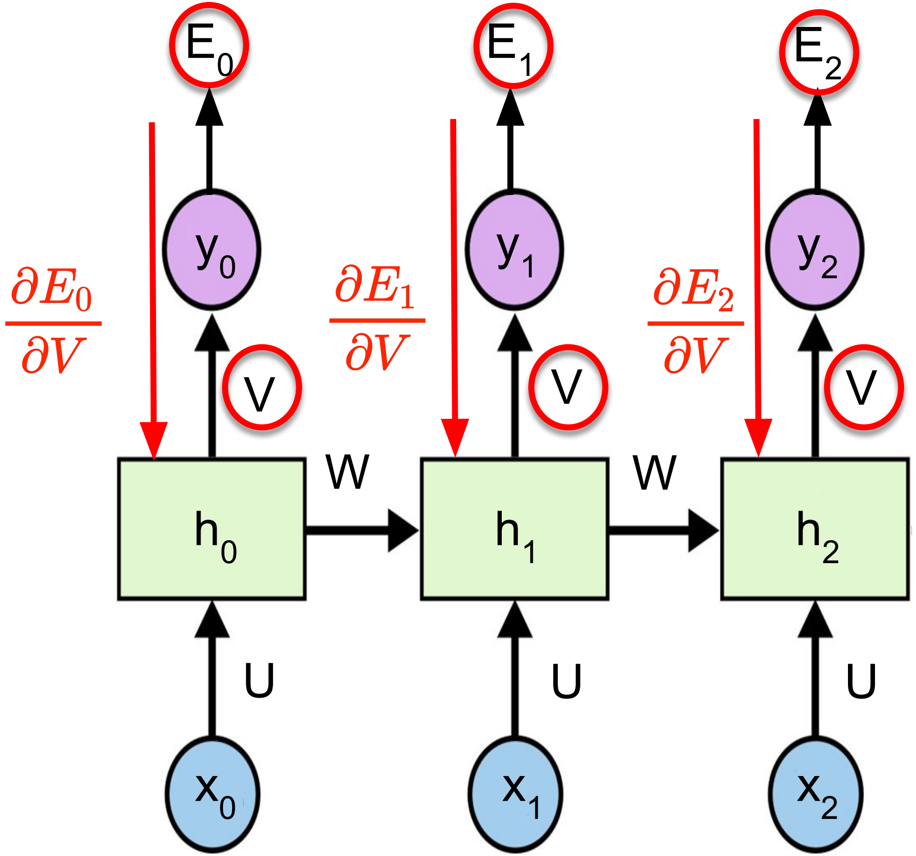


Image from Christopher Olah's blog

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).

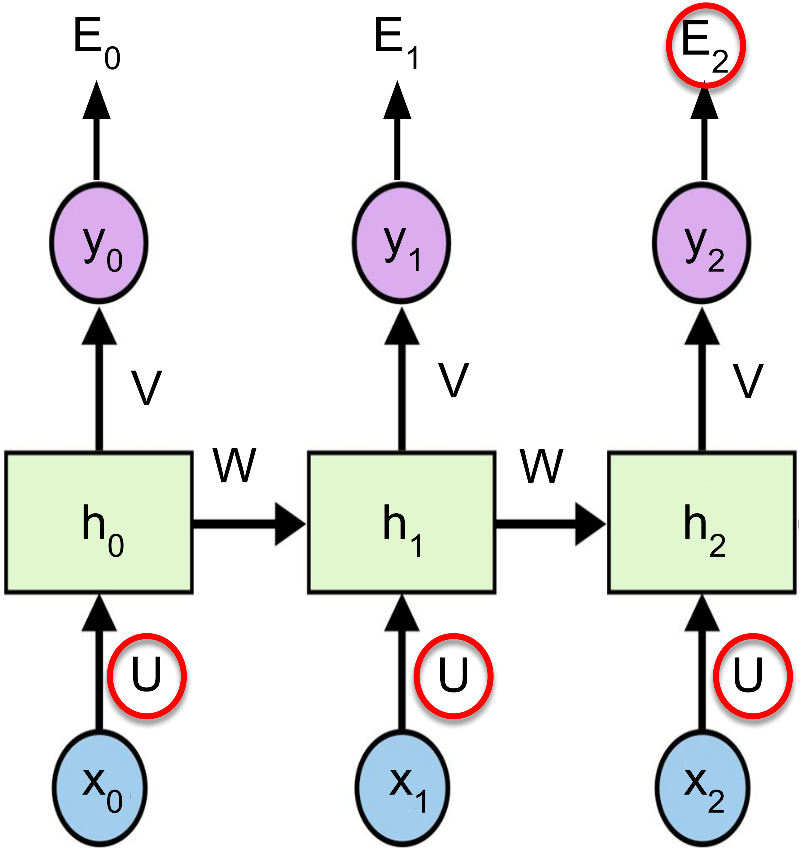


Image from Christopher Olah's blog

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**.

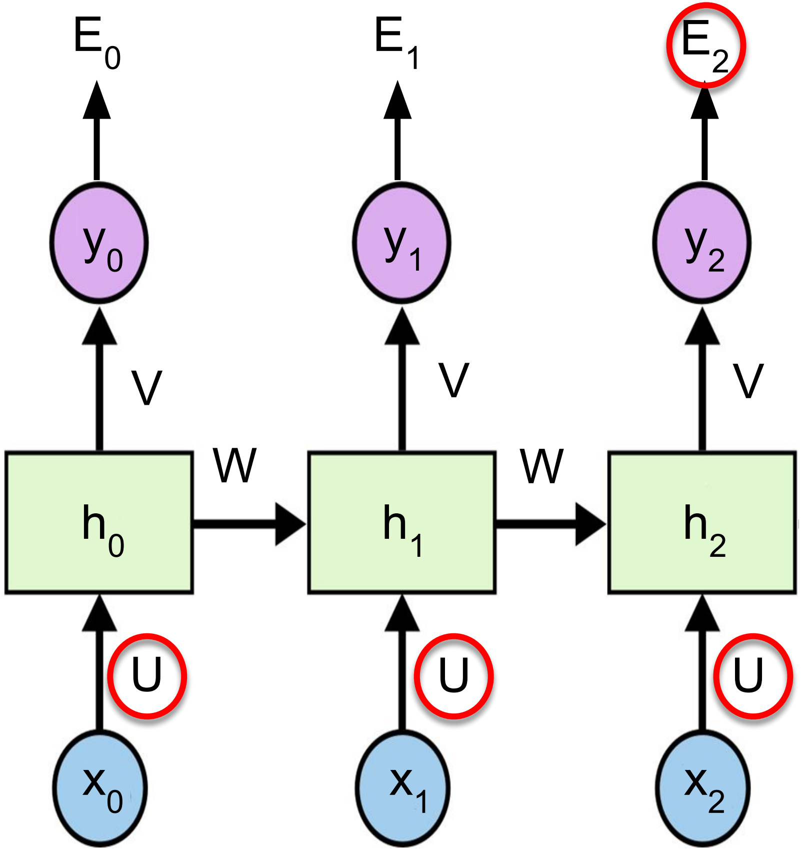


Image from Christopher Olah's blog

Backpropagation Through Time

- We apply the chain rule to compute:

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

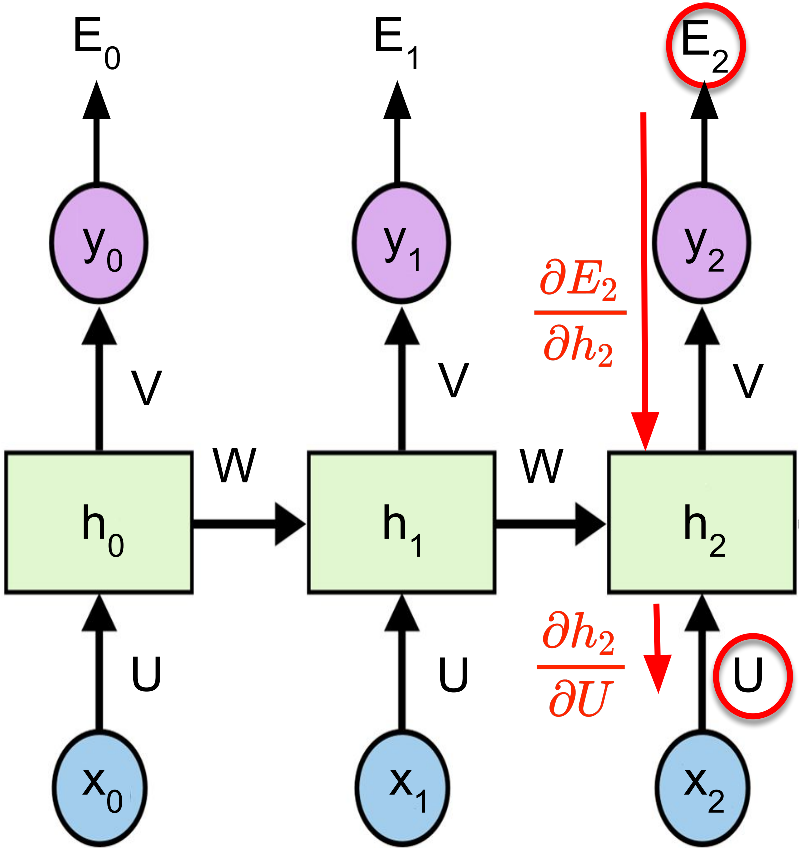


Image from Christopher Olah's blog

Backpropagation Through Time

- We apply the chain rule to compute:

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

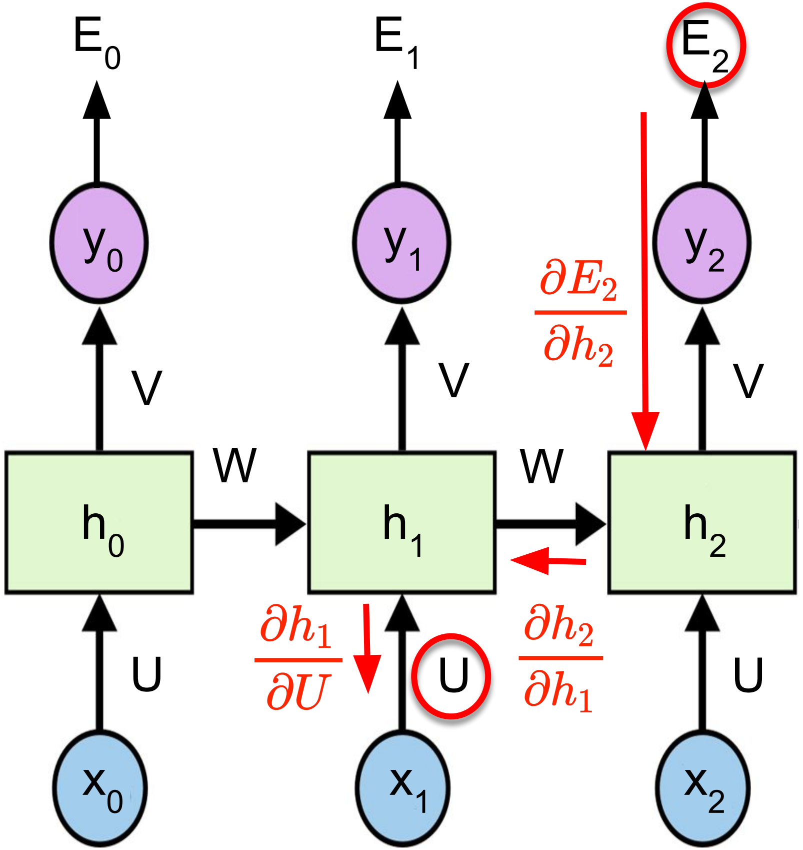


Image from Christopher Olah's blog

Backpropagation Through Time

- We apply the chain rule to compute:

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

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

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

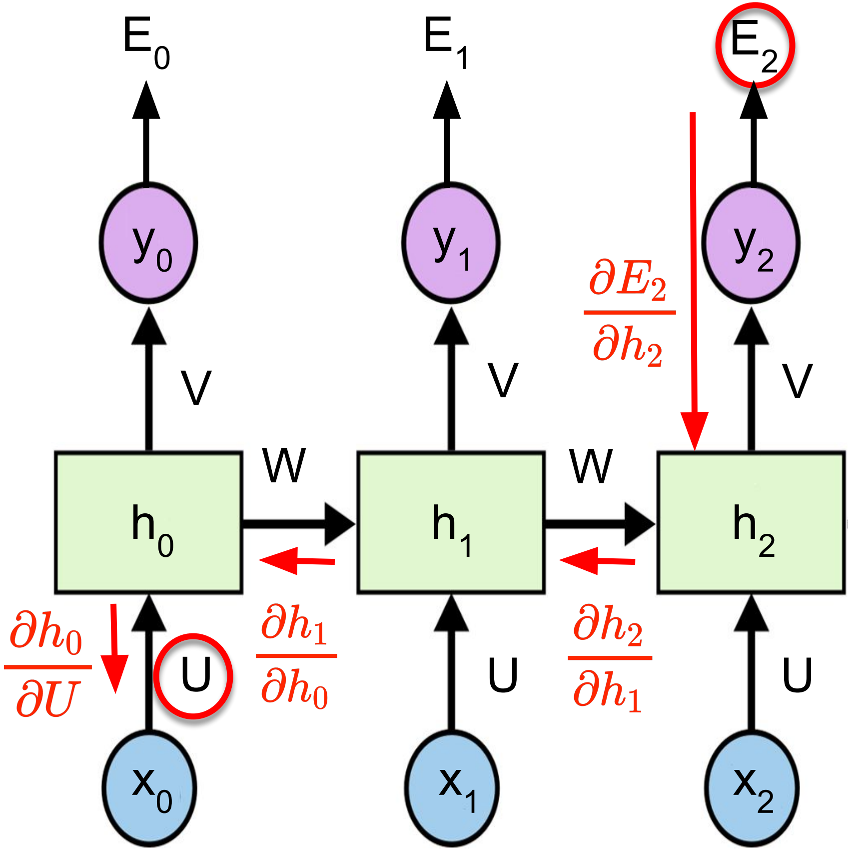


Image from Christopher Olah's blog

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.

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

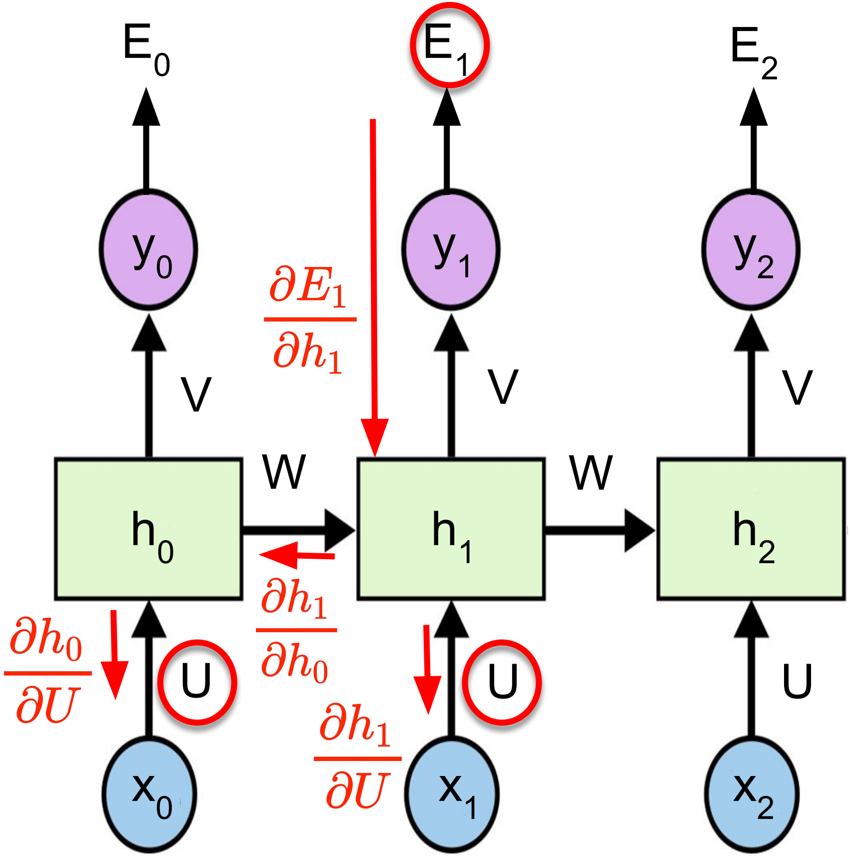


Image from Christopher Olah's blog

Backpropagation Through Time

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

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

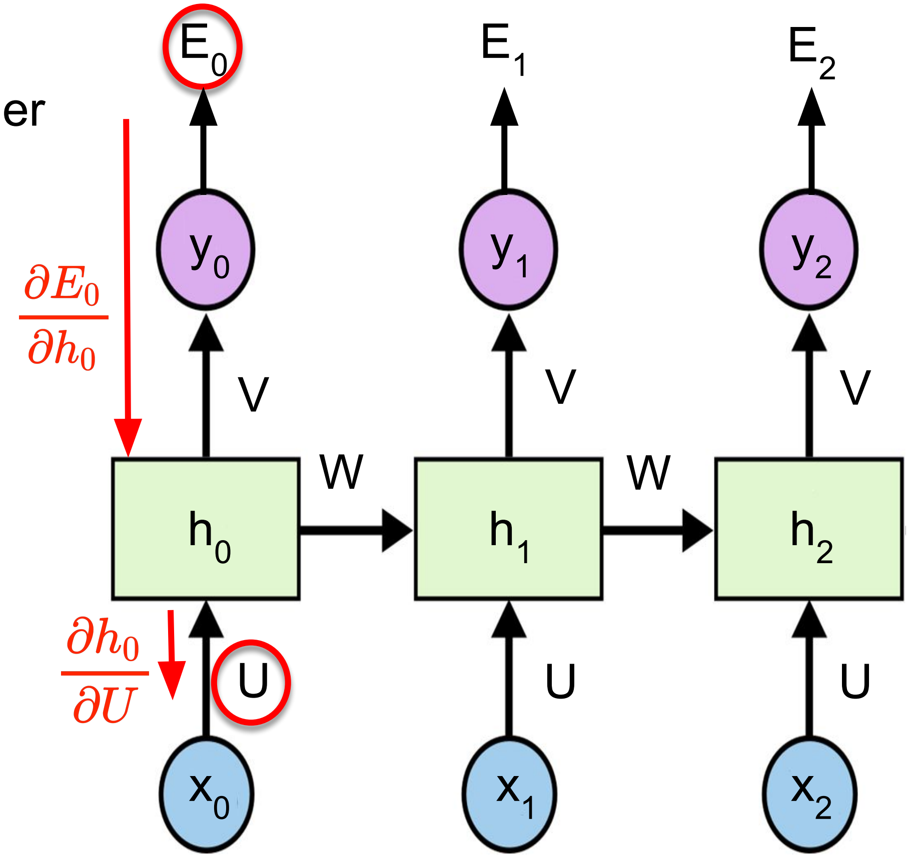


Image from Christopher Olah's blog

Backpropagation Through Time

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

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

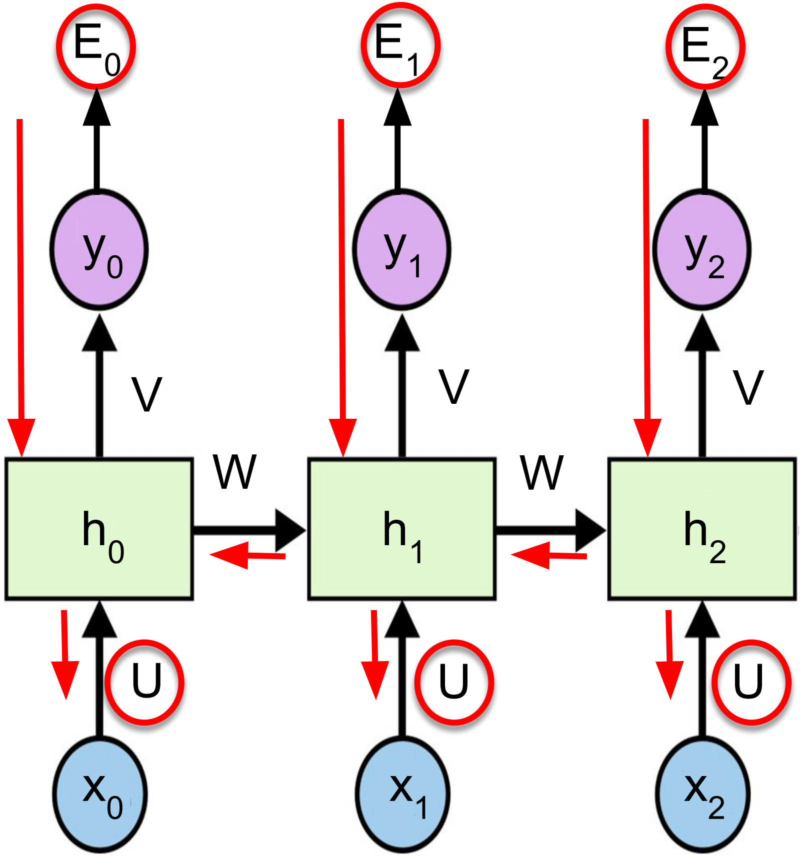


Image from Christopher Olah's blog

Backpropagation Through Time

- If you haven't understood all the steps:
  deep learning frameworks will compute the gradient for you!

- Note that the recurrent nature of RNNs may lead to some problems during training.
- We will inspect those in the next slides.

Plan

- Motivation
- Introduction to Recurrent Neural Networks (RNNs)
- Training RNNs
- **Training problems**
- RNN architectures
- Deep RNNs

Long-Term Dependencies

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

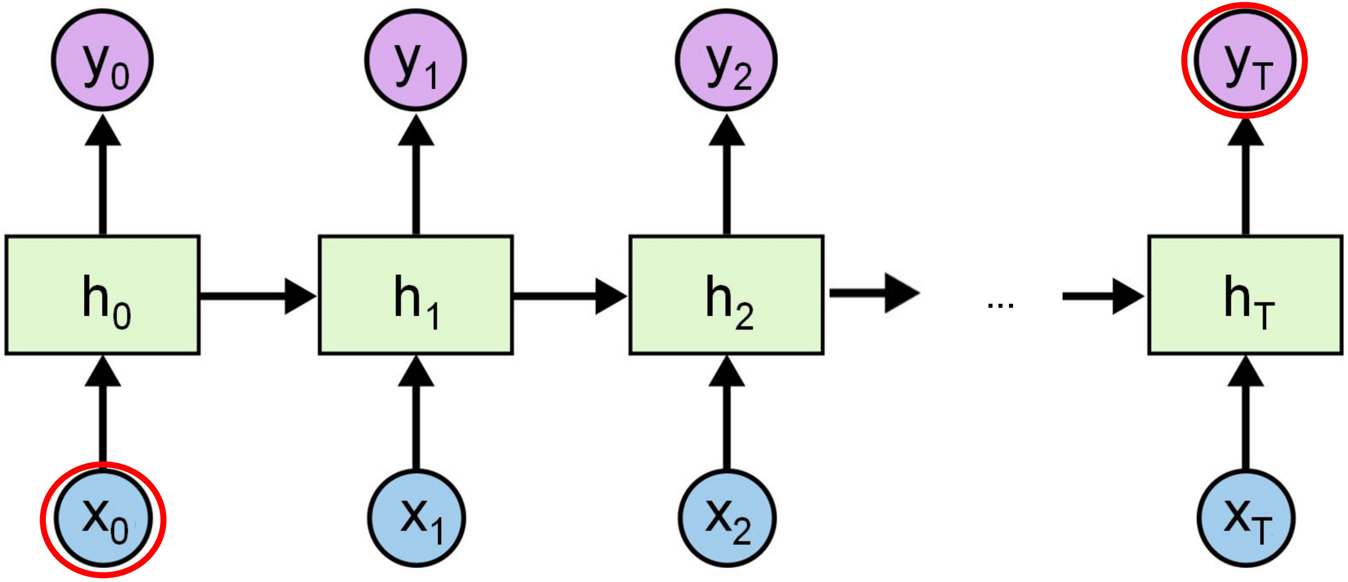


Image from Christopher Olah’s blog

Mila

Long-Term Dependencies

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

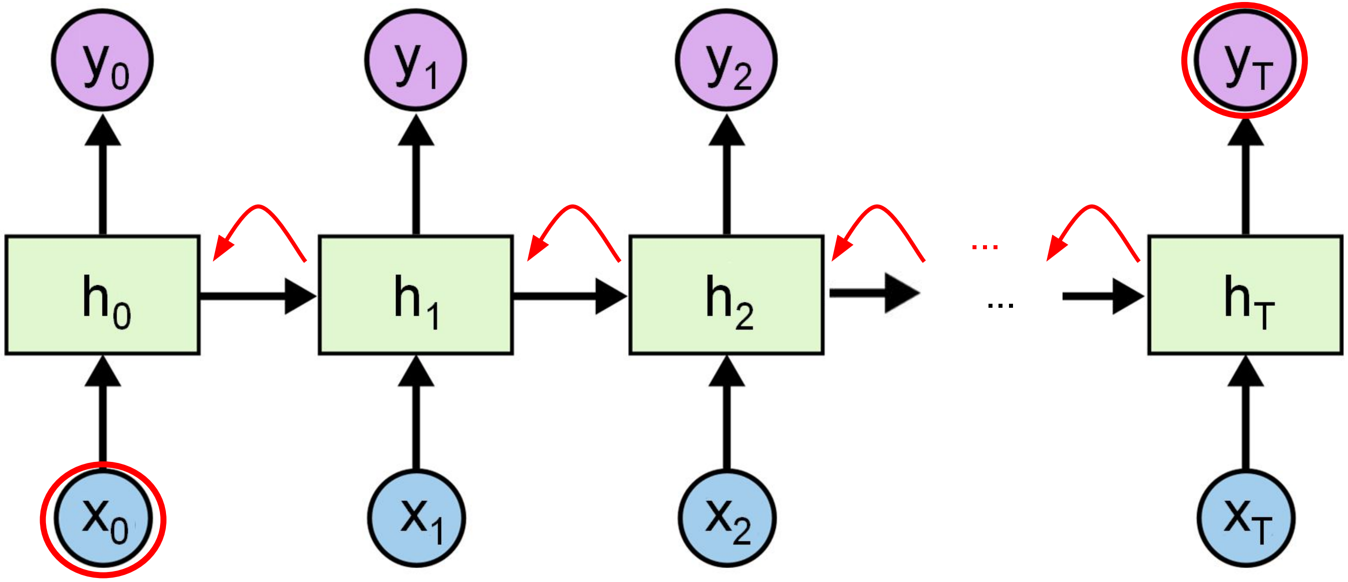


Image from Christopher Olah's blog

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...

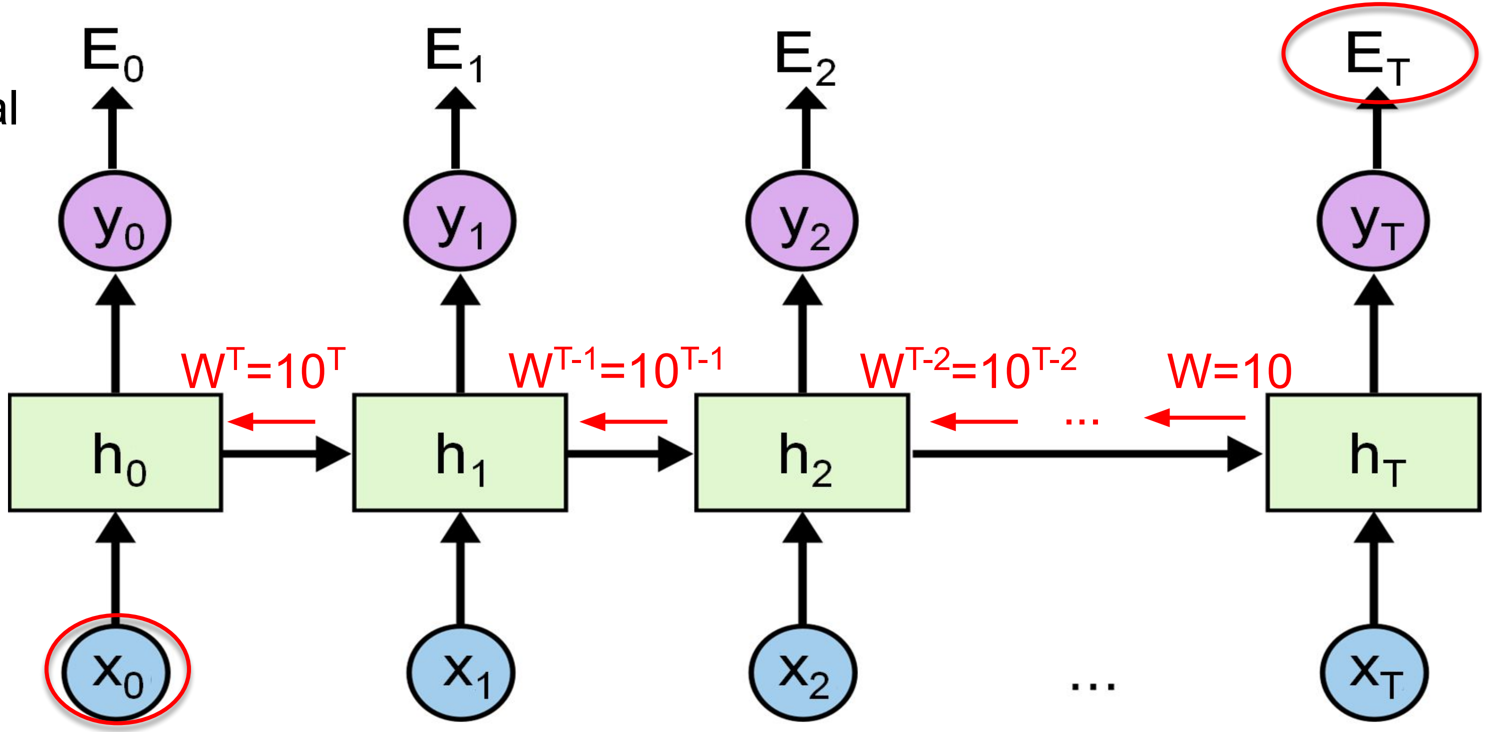
$$\frac{\partial h_T}{\partial h_{T-1}} \cdots \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0}$$

- ...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!

Image from Christopher Olah's blog

Exploding Gradient

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

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

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

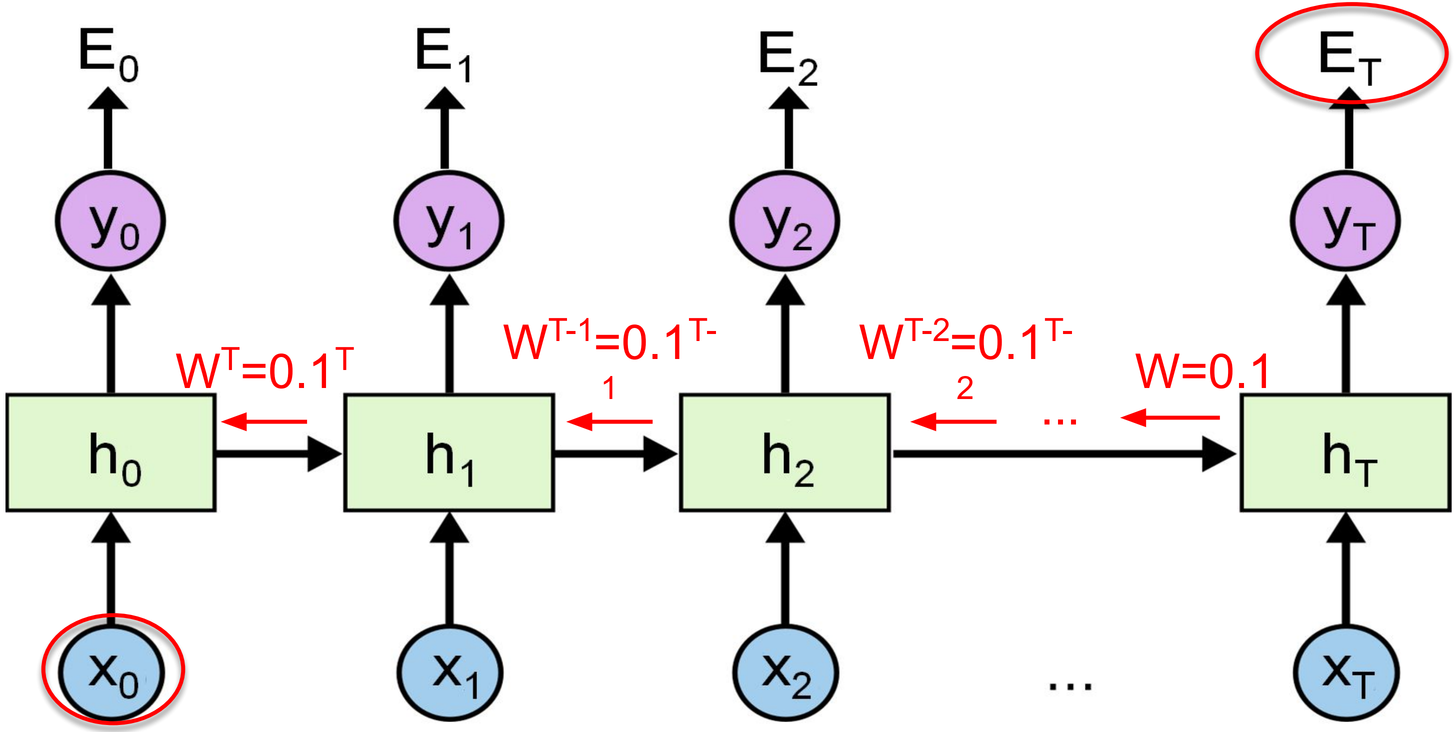
$$\quad g \leftarrow \frac{threshold}{\|g\|} g$$

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

Mila

Vanishing Gradient

Simple 1-dimensional example with W = [0.1]



The gradient decreases at every step = vanishing gradient!

Image from Christopher Olah's blog

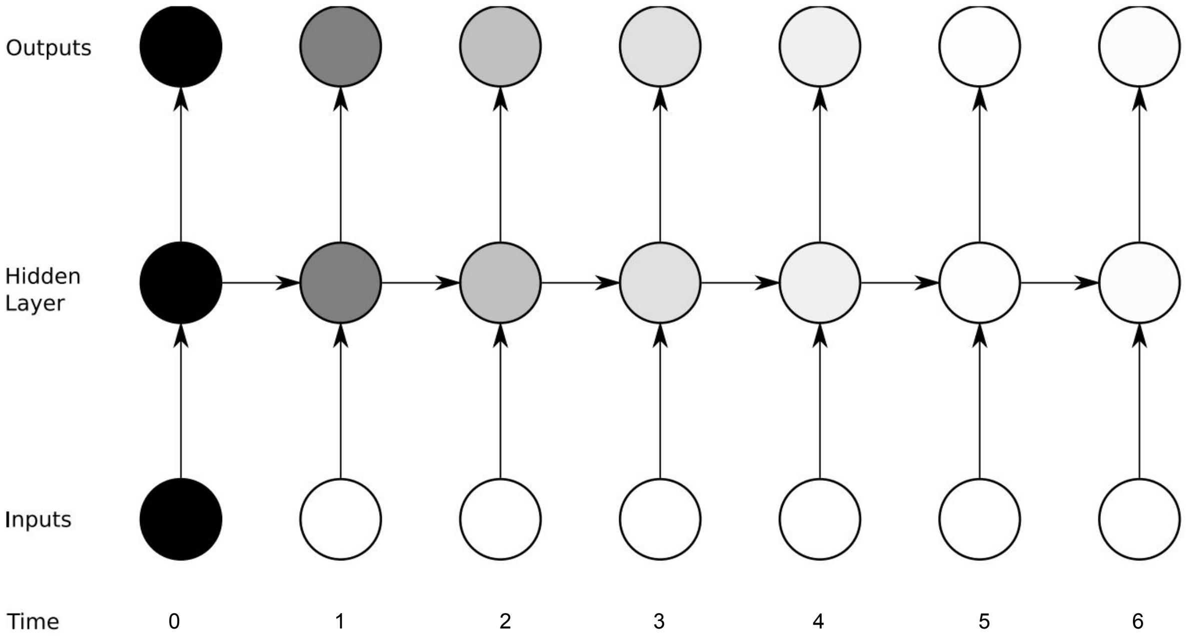
Vanishing Gradient

- 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.

Plan

- Motivation
- Introduction to Recurrent Neural Networks (RNNs)
- Training RNNs
- Training problems
- **RNN architectures**
- Deep RNNs

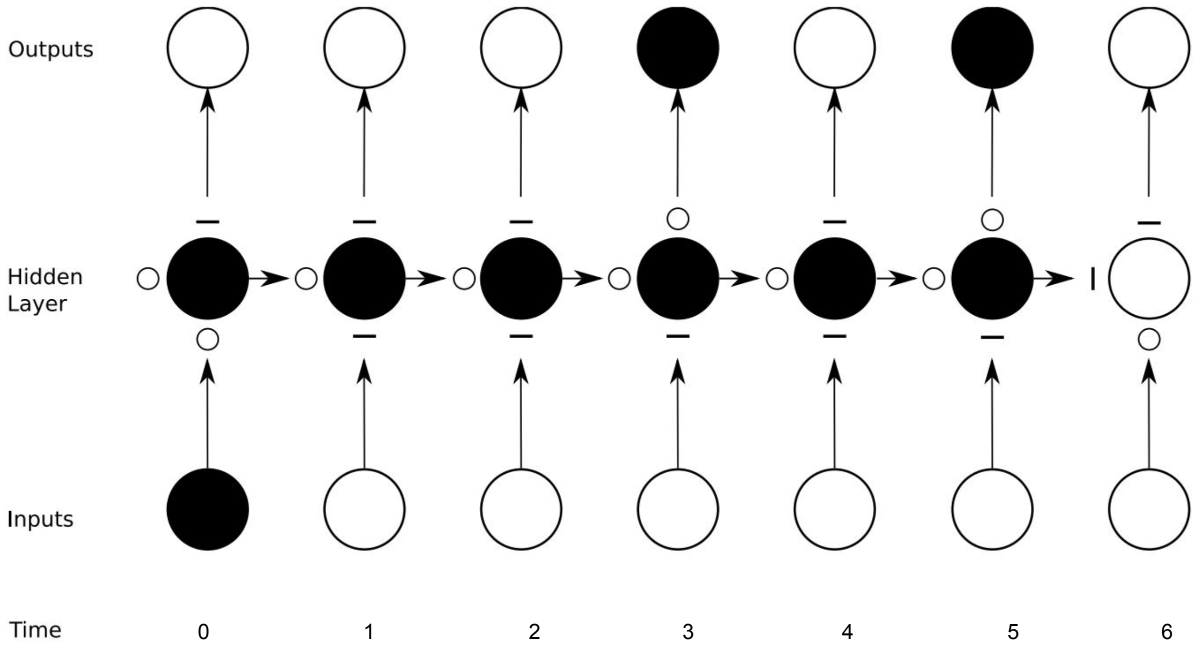
Memory Problems



The colors show the influence of the input at time 0 which decreases over time as the RNN gradually forgets that particular input.

Graves, Supervised Sequence Labelling with Recurrent Neural Networks

Memory Problems

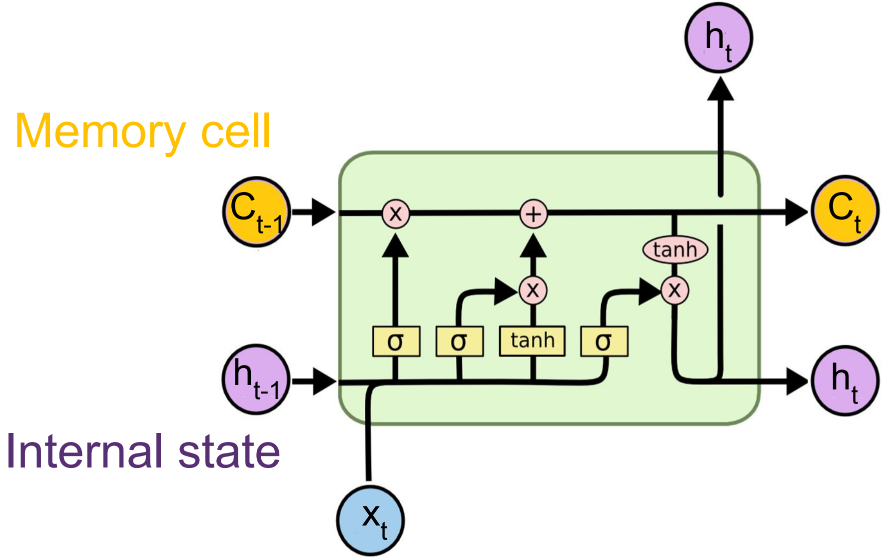


By adding gates (o open; - closed), the RNN can selectively control the flow of information (and greatly minimize the vanishing gradient problem).

Graves, Supervised Sequence Labelling with Recurrent Neural Networks

Long Short-Term Memory (LSTM)

- Reduce the vanishing gradient problem using a gate mechanism and adding a memory cell.



$$i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$$

$$f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$$

$$o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$$

$$\tilde{C}_t = tanh(U_g x_t + W_g h_{t-1} + b_g)$$

$$C_t = i_t \times \tilde{C}_t + f_t \times C_{t-1}$$

$$h_t = o_t \times tanh(C_t)$$

Image from Christopher Olah's blog

Hochreiter et al., Long short-term memory, Neural Computation 1997

LSTM - Step-by-Step

- The key idea introduced in the LSTM is the **Memory cell**.
  - Few operations happen there.
  - Information can flow more easily.

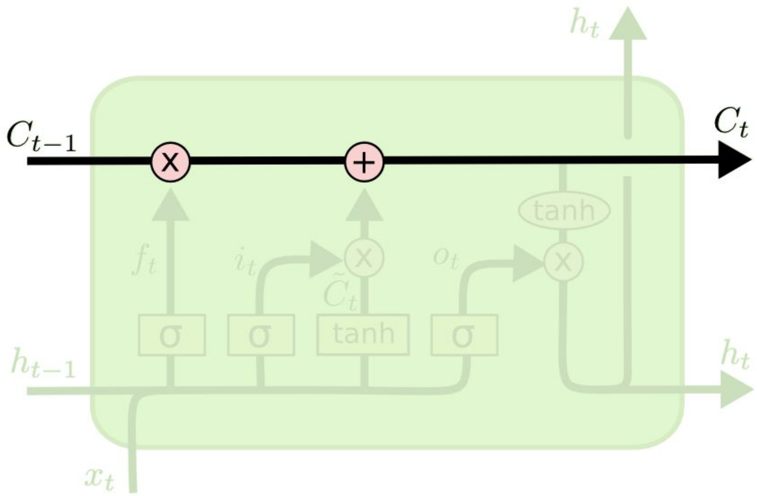


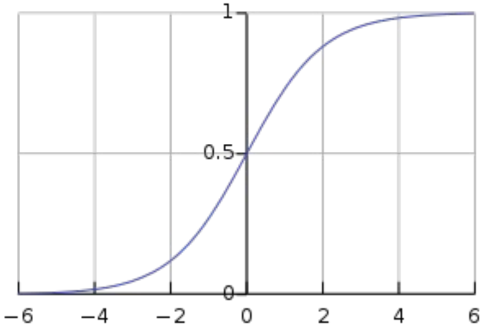
Image from Christopher Olah's blog

LSTM - Step-by-Step

- The **Forget gate** is computed from x_t and h_{t-1}:

$$f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$$

- σ is the sigmoid function (bounded between 0 and 1).



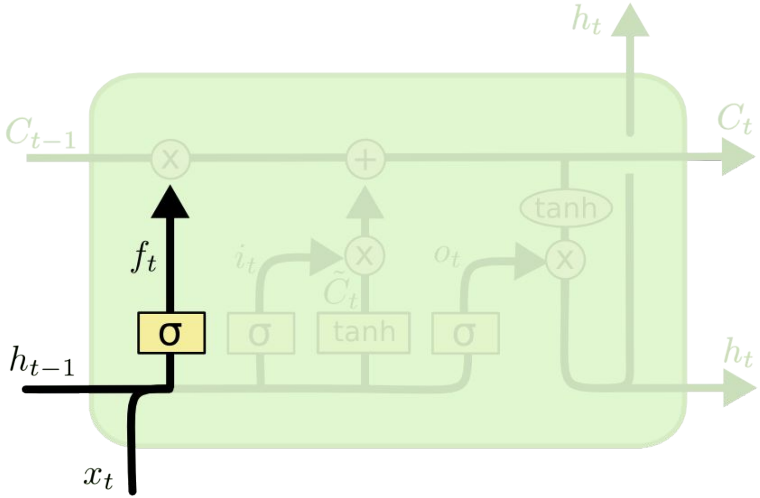


Image from Christopher Olah's blog

LSTM - Step-by-Step

- The **Forget gate** is computed from x_t and h_{t-1}:

$$f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$$

- σ is the sigmoid function (bounded between 0 and 1).
- The Forget gate allows the LSTM to delete information from its memory.

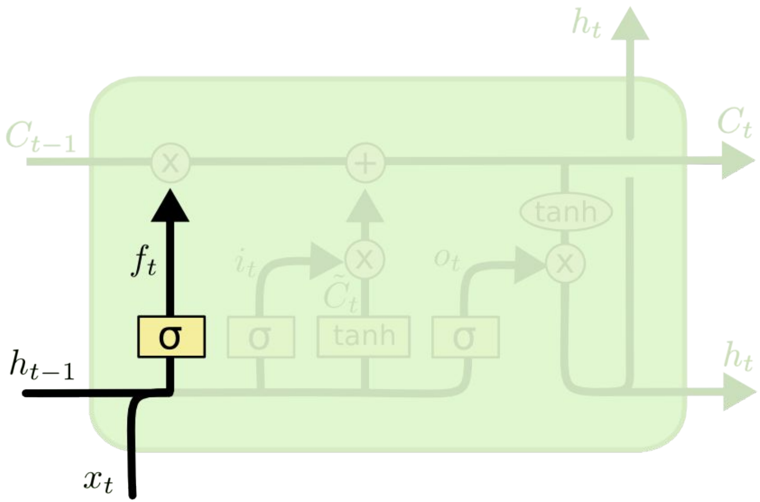


Image from Christopher Olah's blog

LSTM - Step-by-Step

- The **Input gate** is computed from x_t and h_{t-1}:

$$i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$$

- The Input gate controls how much is added to the memory cell.
- The **candidate value** controls what is added to the memory.

$$\tilde{C}_t = tanh(U_g x_t + W_g h_{t-1} + b_g)$$

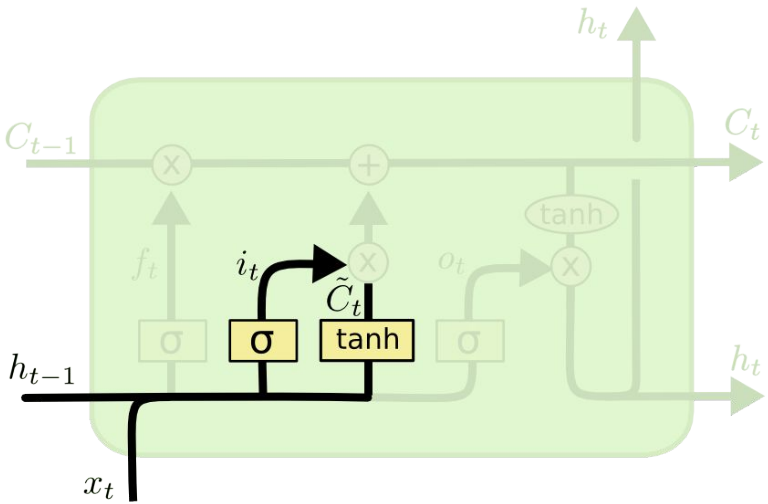


Image from Christopher Olah's blog

LSTM - Step-by-Step

- The **Memory cell** is updated using the Input gate and the Forget gate:

$$C_t = i_t \times \tilde{C}_t + f_t \times C_{t-1}$$

- $\times$= element-wise multiplication.
- The Input gate controls the amount of information added to the cell, and the Forget gate controls the amount of information deleted from the cell.

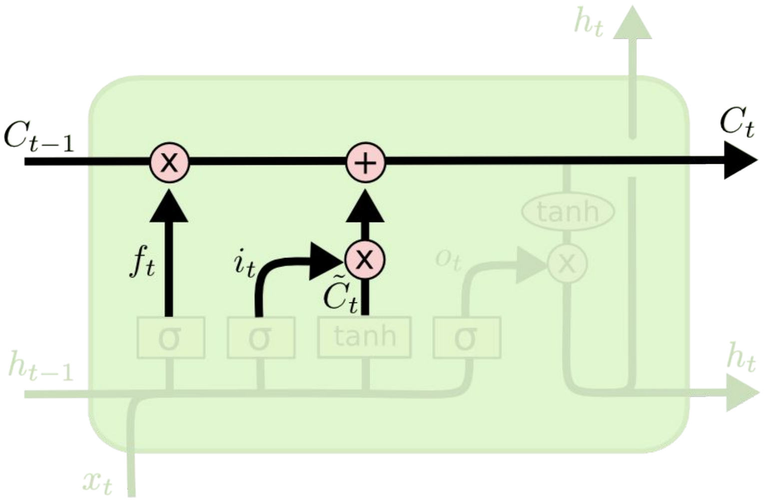


Image from Christopher Olah’s blog

LSTM - Step-by-Step

- The **Output gate** is computed from x_t and h_{t-1}:

$$o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$$

- The Output gate controls the output of the memory cell.
- The **new internal state** is computed as:

$$h_t = o_t \times tanh(C_t)$$

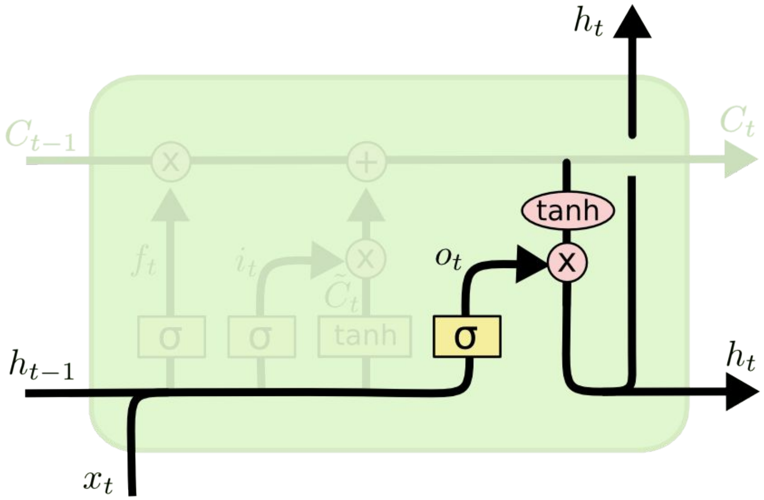
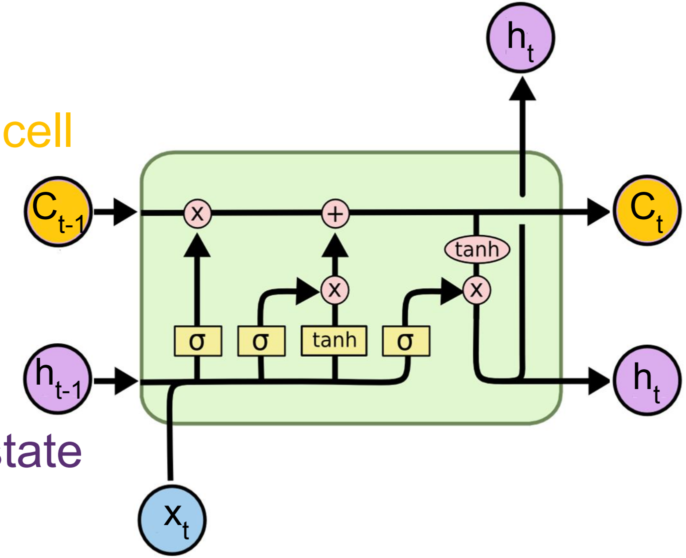


Image from Christopher Olah's blog

Long Short-Term Memory (LSTM)

- Reducing the vanishing problem using a gate mechanism and adding a memory cell.

Memory cell



Internal state

$$i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$$
$$f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$$
$$o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$$
$$\tilde{C}_t = tanh(U_g x_t + W_g h_{t-1} + b_g)$$
$$C_t = i_t \times \tilde{C}_t + f_t \times C_{t-1}$$
$$h_t = o_t \times tanh(C_t)$$

Image from Christopher Olah's blog

Hochreiter et al., Long short-term memory, Neural Computation 1997

Gated Recurrent Unit (GRU)

- A popular variant of the LSTM.
- No dedicated memory cell.
- Input and Forget gates are combined.
- In practice, it provides results similar to an LSTM.
- Faster to compute.

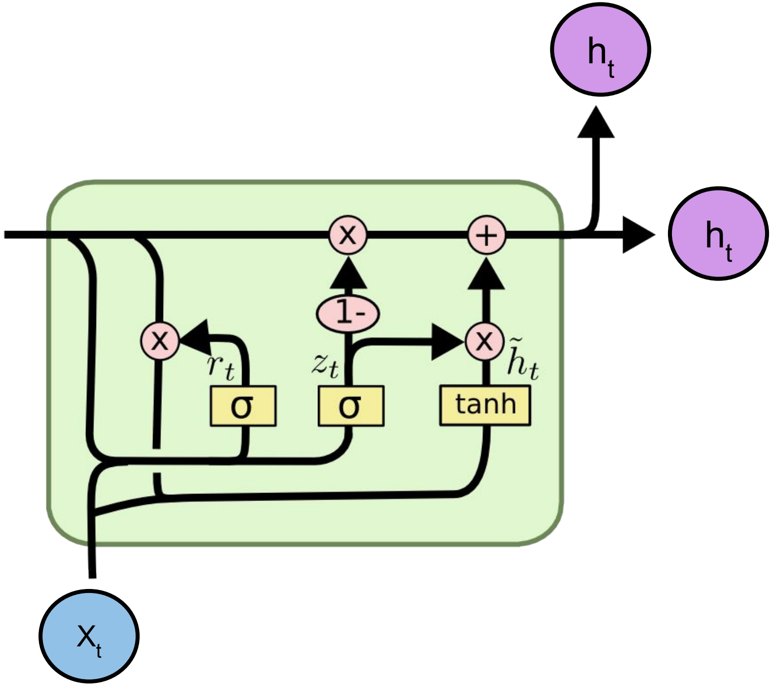


Image from Christopher Olah’s blog

Chung et al.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Gated Recurrent Unit (GRU)

$$z_t = \sigma(U_z x_t + W_z h_{t-1} + b_z)$$ ← Update gate

$$r_t = \sigma(U_r x_t + W_r h_{t-1} + b_r)$$ ← Reset gate

$$\tilde{h}_t = tanh(U_g x_t + W_g(r_t \times h_{t-1}) + b_g)$$

$$h_t = z_t \times \tilde{h}_t + (1 - z_t) \times h_{t-1}$$ ← Internal state

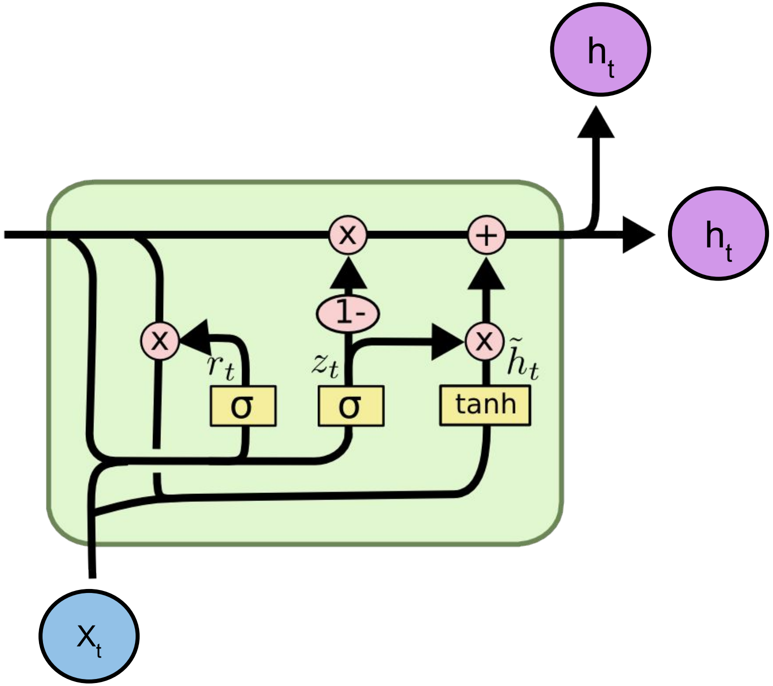


Image from Christopher Olah’s blog

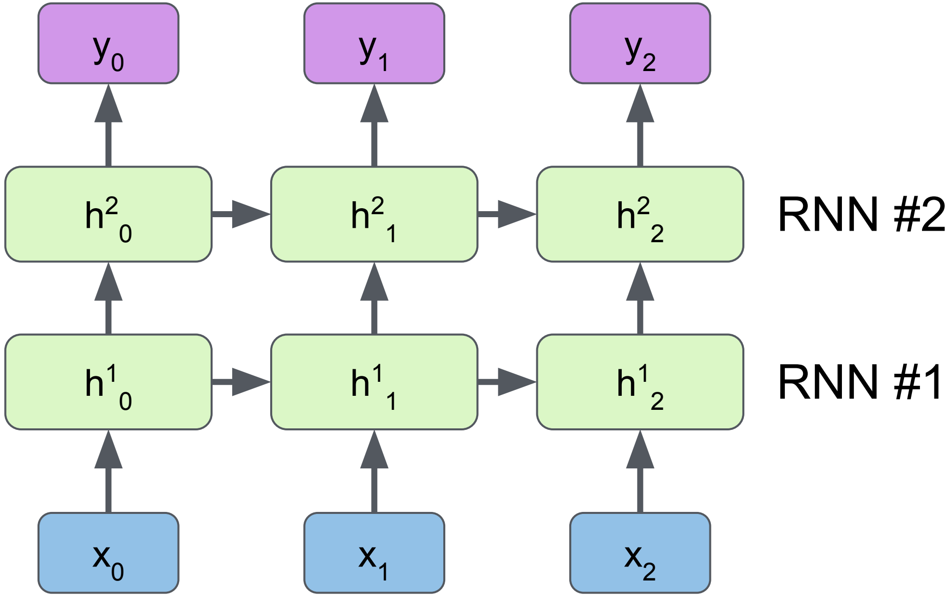
Chung et al.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Plan

- Motivation
- Introduction to Recurrent Neural Networks (RNNs)
- Training RNNs
- Training problems
- RNN architectures
- **Deep RNNs**

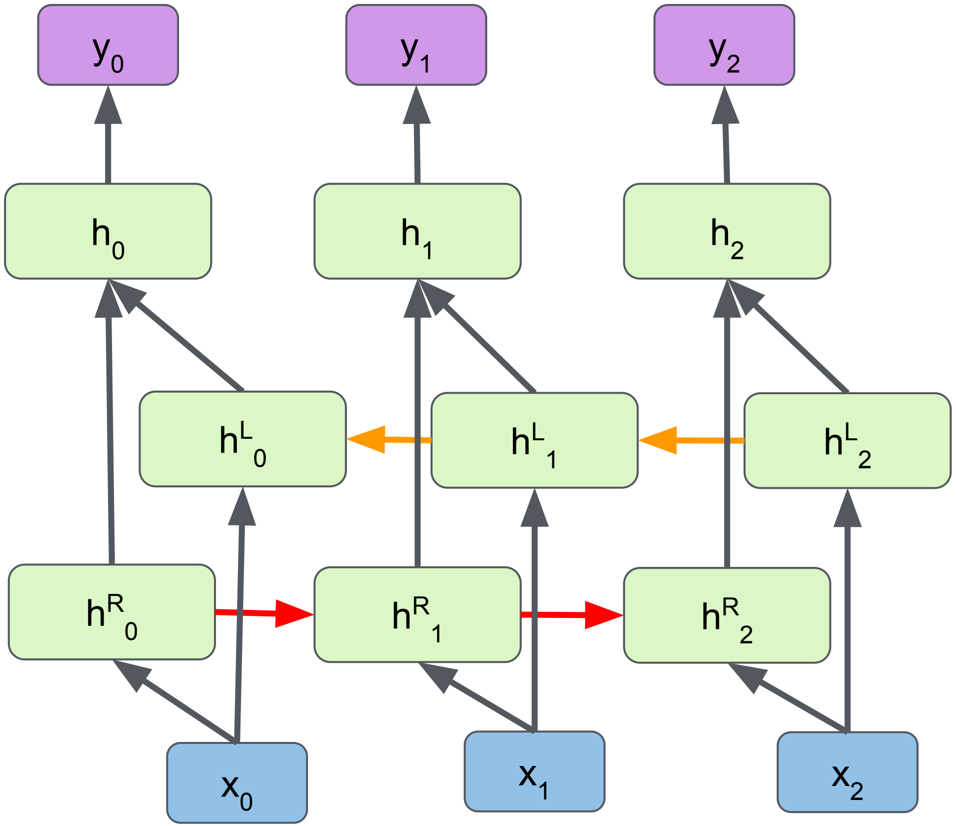
Deep RNNs

- To create deep RNNs, one can stack several RNN layers.
- The output of the first layer is the input to the second layer, etc.
- Every layer has a different set of parameters.



Bidirectional RNNs

- Use two RNNs: one operating left-to-right, one operating right-to-left.
- This allows to look at information coming from the "past" and "future".
- The two RNNs are different (different parameters).
- The two RNN outputs (h^R_t,h^L_t) can be "merged" (h_t) in various ways: concatenation, sum, ...



Quebec
Artificial
Intelligence
Institute

Mila

Questions?