Recurrent Neural Networks

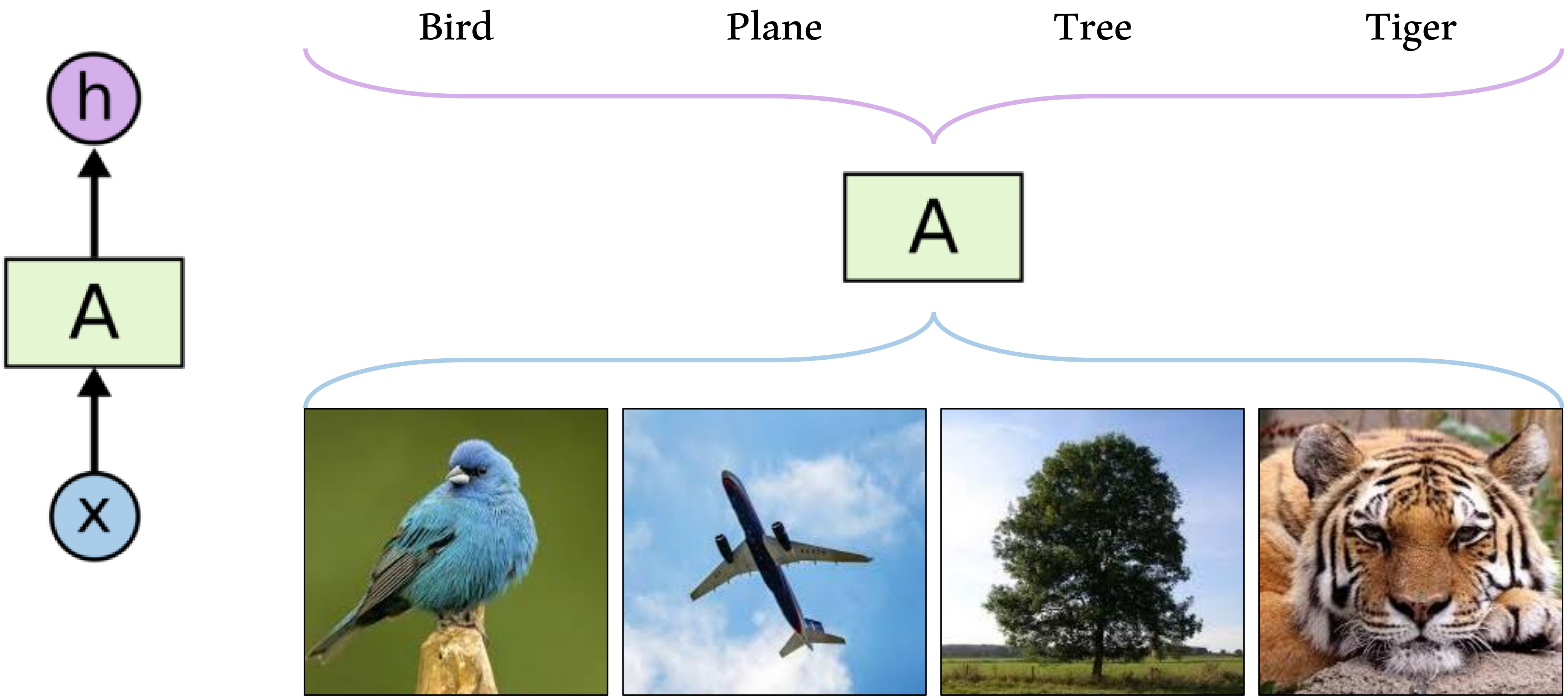
Paola Ruiz & Natalia Valderrama

Tutor: Laura Daza

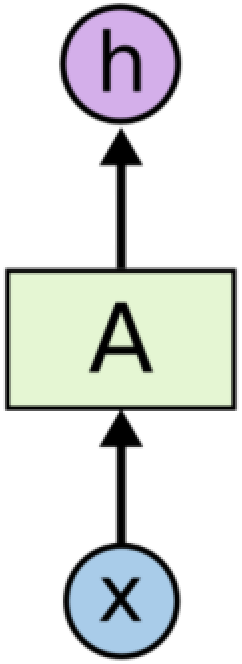
Content List

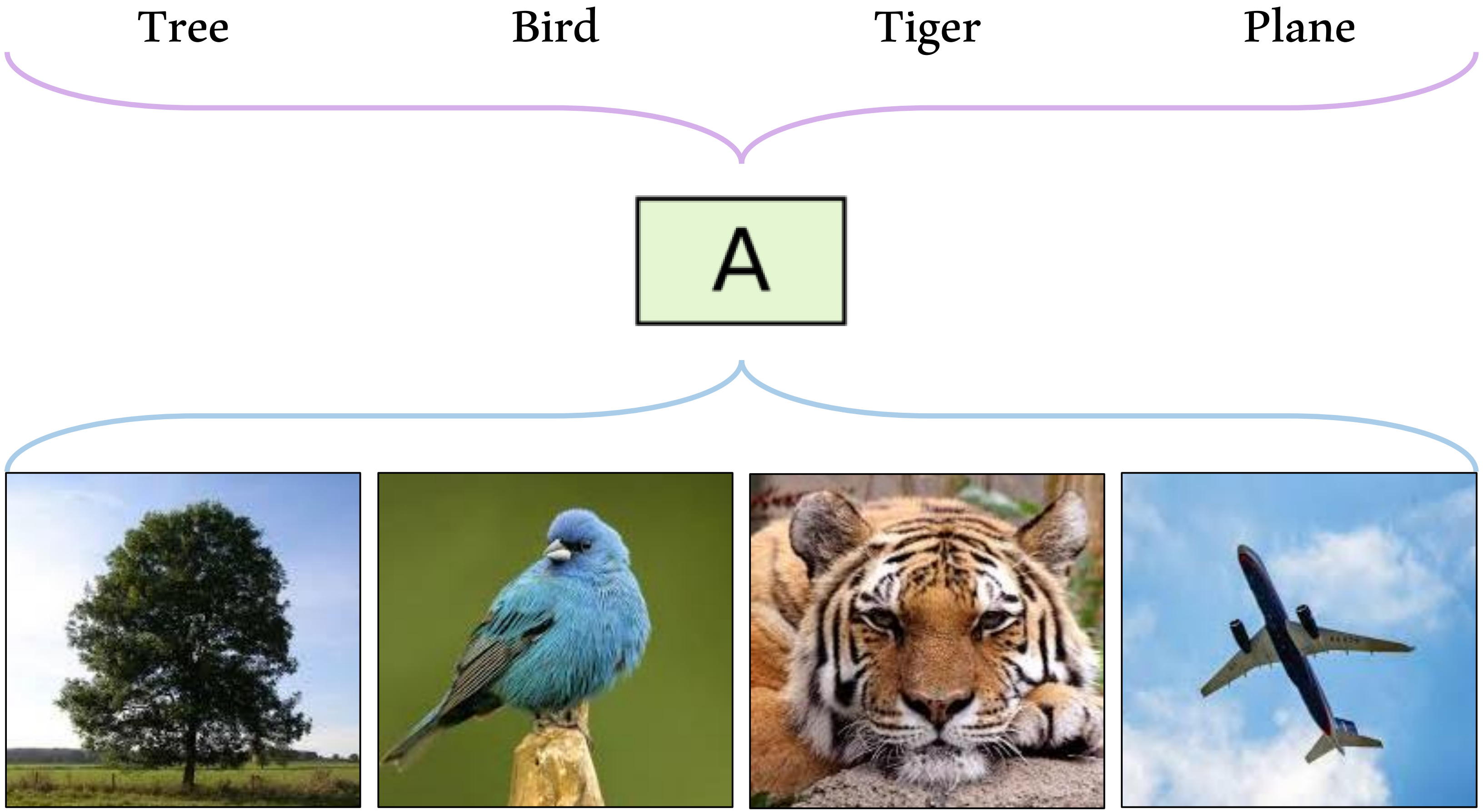
- ✓ Introduction
- ✓ RNN structure
- ✓ Types
- ✓ Backpropagation through time
- ✓ Vanishing and exploding Gradient Problem
- ✓ LSTM
- ✓ GRU
- ✓ SRU
- ✓ Variation diagrams:
  - ✓ Multilayer
  - ✓ Bidirectional
  - ✓ Encoder-Decoder
  - ✓ Attention Layer
- ✓ Papers:
  - ✓ SMT
  - ✓ Biometrics
- ✓ Tutorial
- ✓ Homework

Introduction



Introduction



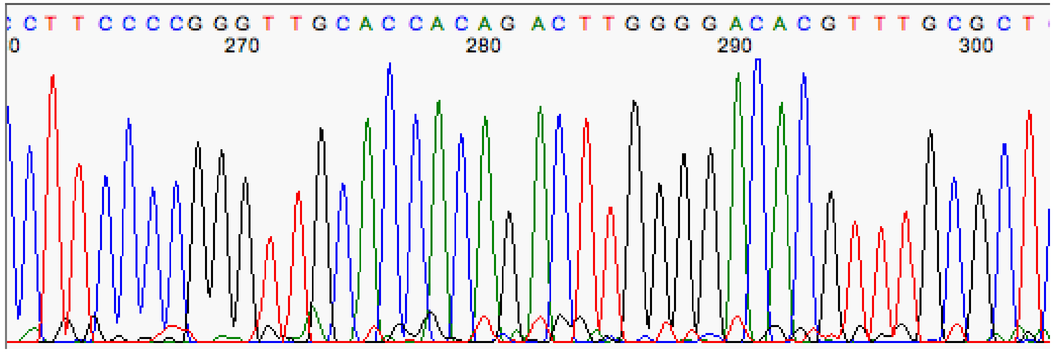


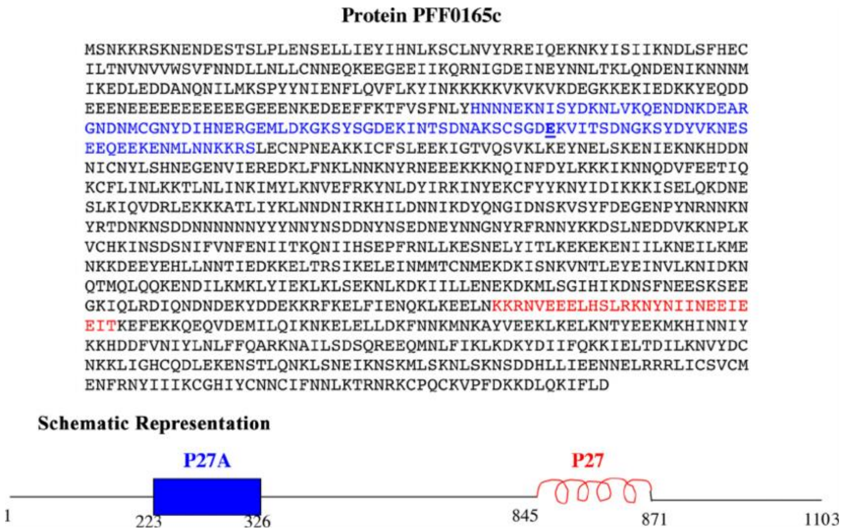
Introduction

“I've got to think that that was unethical," Joshua said.
"Josh, faking demonic possession is like a mustard seed."
"How is it like a mustard seed?"
"You don't know, do you? Doesn't seem at all like a mustard seed, does it? Now you see how we all feel when you liken things unto a mustard seed? Huh?”
― Christopher Moore, Lamb: The Gospel According to Biff, Christ's Childhood Pal

“I also felt guilty about the three pens I'd stolen, but only for a second. And since there was no convenient way to give them back, I stole a bottle of ink before I left.”
― Patrick Rothfuss, The Name of the Wind

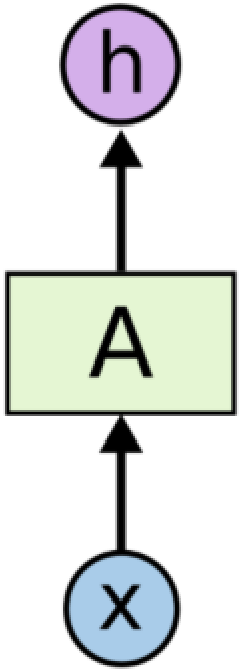
“It wasn't even a good note. 'If you are reading this I am probably dead.' What sort of a note is that?”
― Patrick Rothfuss, The Name of the Wind



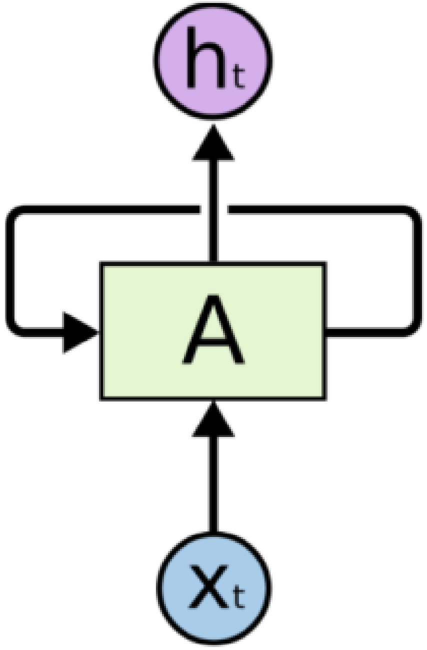


Introduction

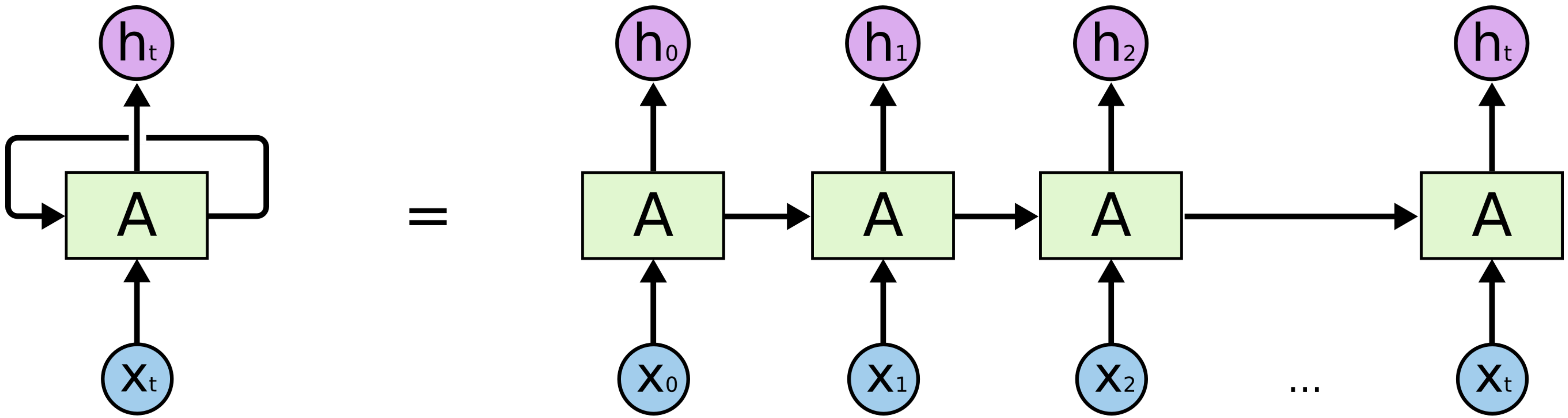
Vanilla Neural Network



Recurrent Neural Network (RNN)



Introduction



Unrolled Recurrent Neural Network (RNN)

Introduction

RNNs are nice and everything, but…

- How far should our "memory" go?

"I've got to think that that was unethical," Joshua said.

– Christopher Moore, Lamb: The Gospel According to Biff, Christ's Childhood Pal

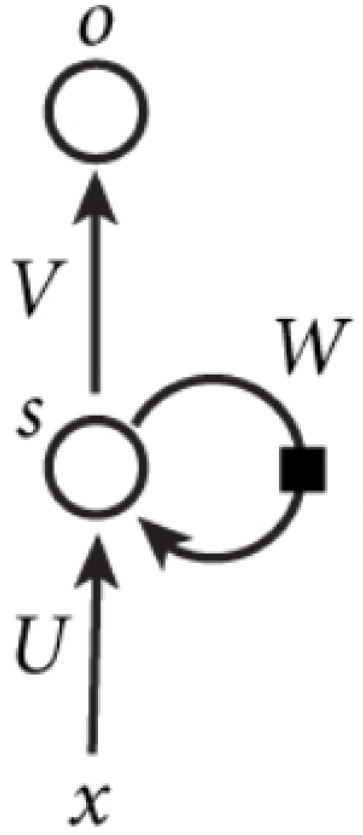
  - If the sequence is too long, wouldn't it be too heavy to store all that information?
- How do we train them?

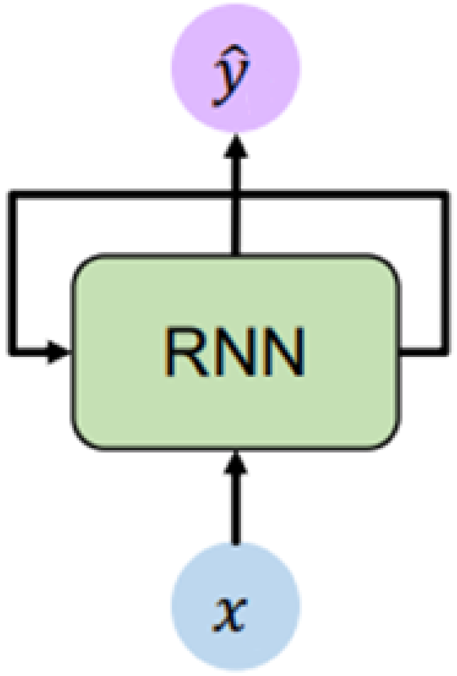
Recurrent Neural Networks

Paola Ruiz &Natalia Valderrama

Tutor: Laura Daza

Recurrent Neural Networks (RNN)





Graphic Representation

Recurrent Neural Networks (RNN)

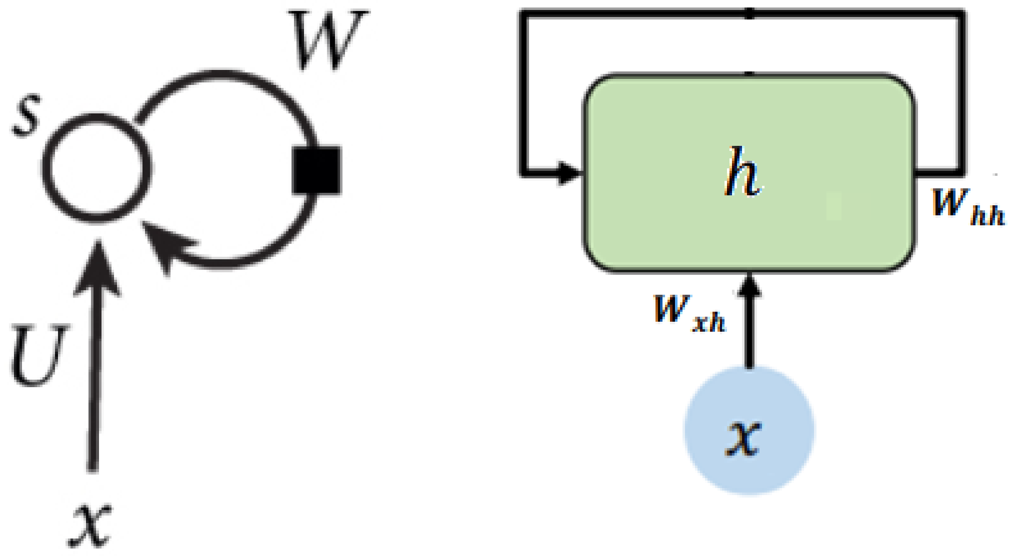
Recurrent Core Cell

Recurrent Neural Networks (RNN)

- Hidden State
- Compute recurrent relation with a function f_W

$$h_t = f_W(h_{t-1}, x_t)$$

Recurrent Neural Networks (RNN)

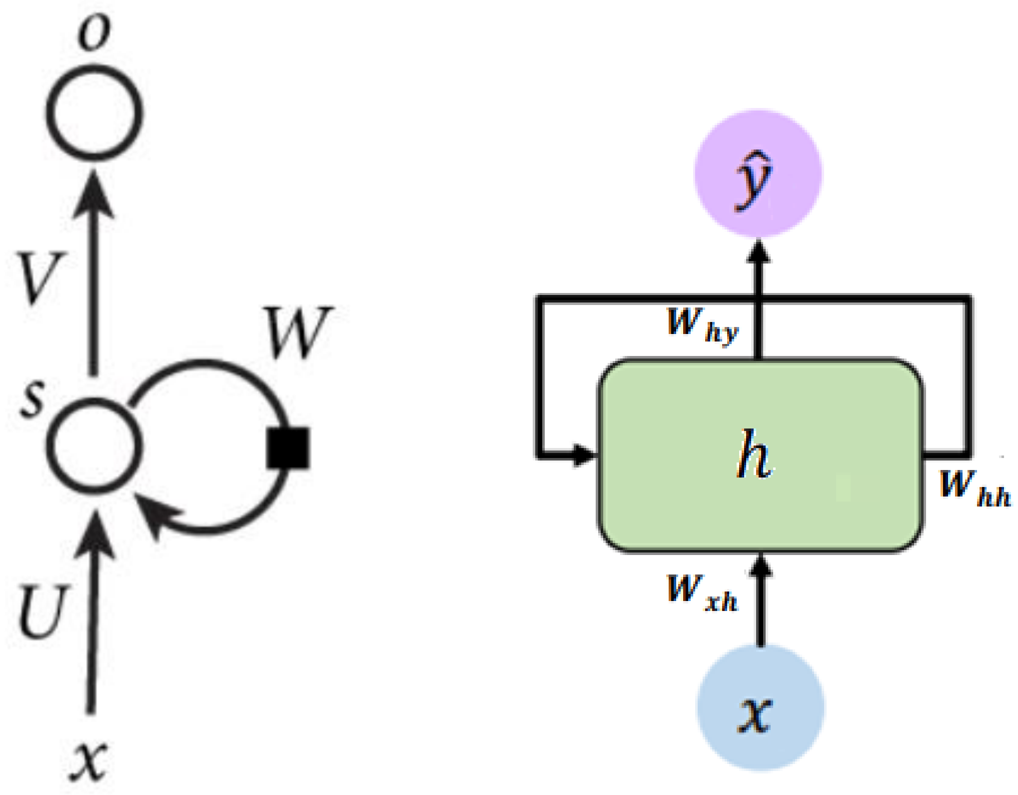


- Input X with weights U (W_{xh})
- Computes h with weights W (W_{hh})

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = tanh\,(W_{hh}h_{t-1} + W_{xh}x_t)$$

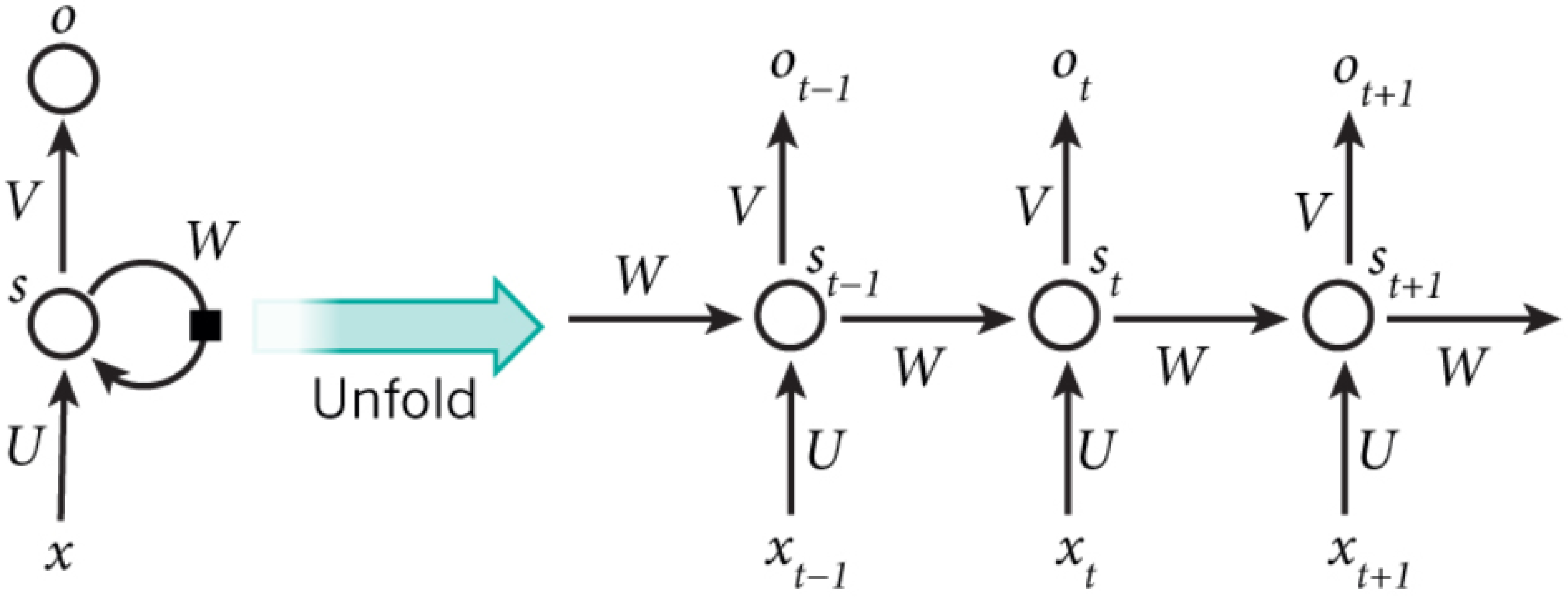
Recurrent Neural Networks (RNN)



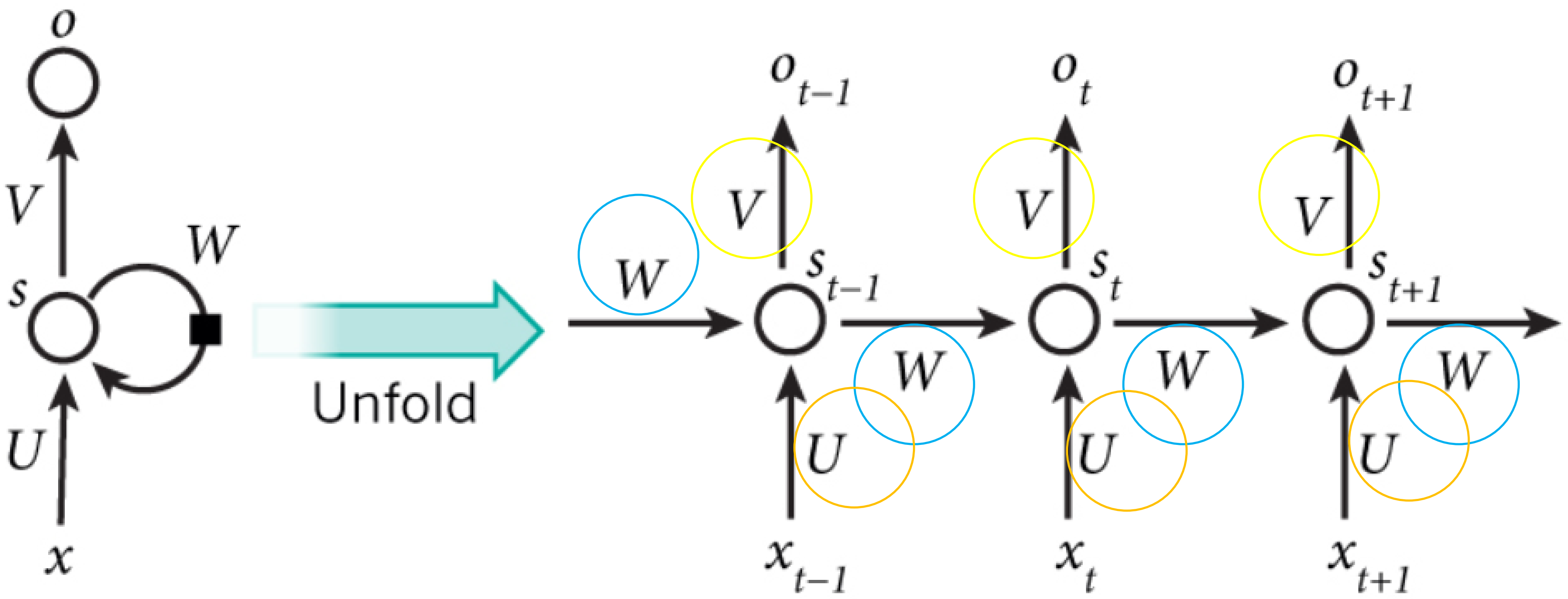
- Output o $(\hat{y})$ with weights V (W_{hy})
- Additional fully connected layers that read h to produce an output.

$$y_t = W_{hy} h_t$$

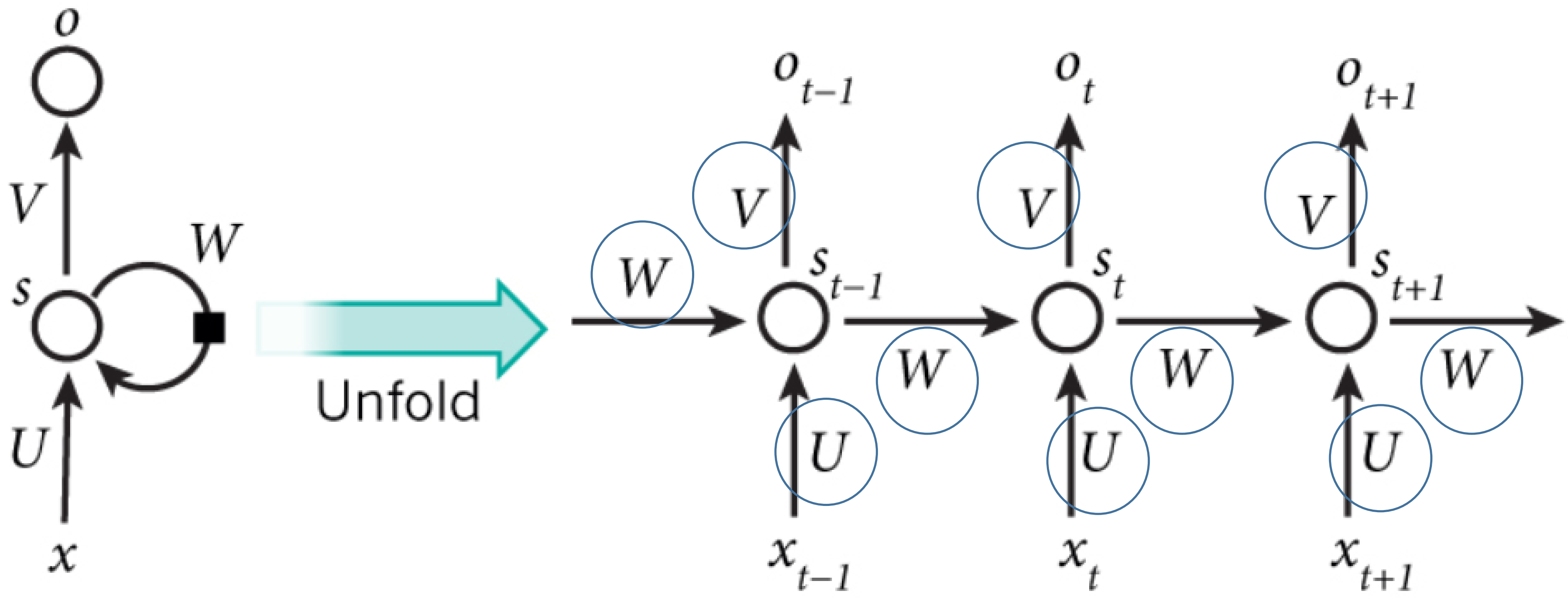
Recurrent Neural Networks (RNN)



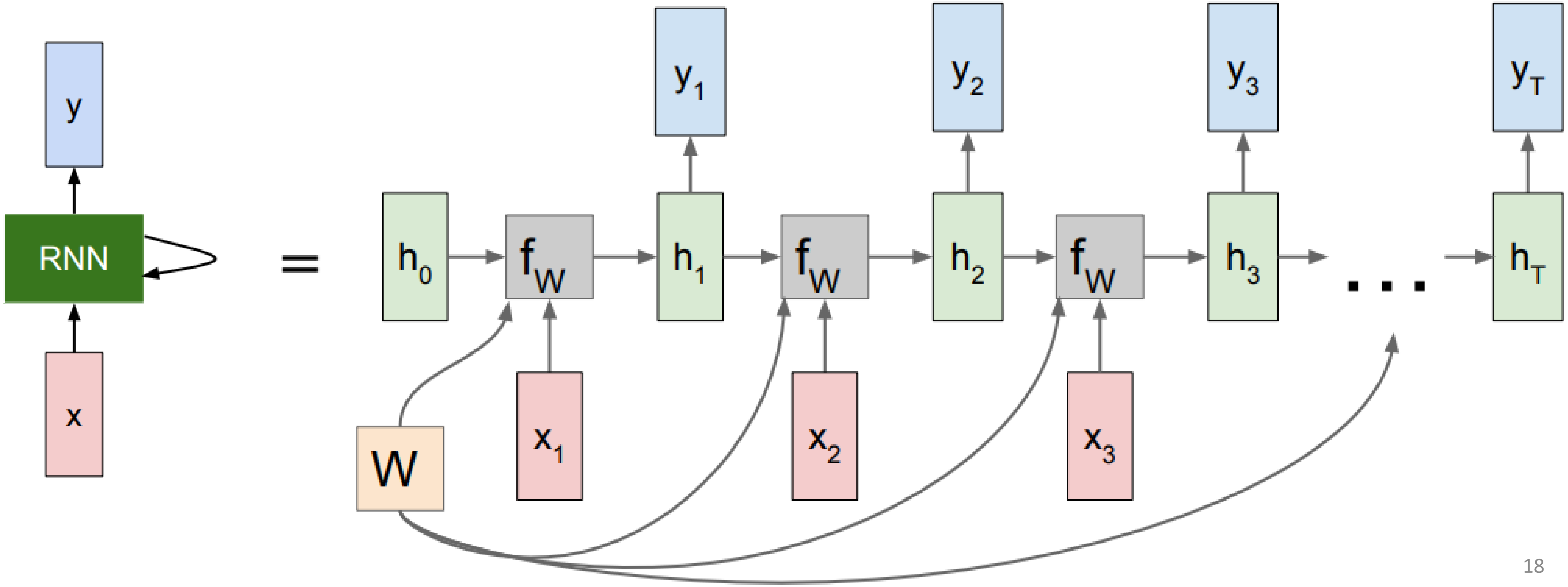
Recurrent Neural Networks (RNN)



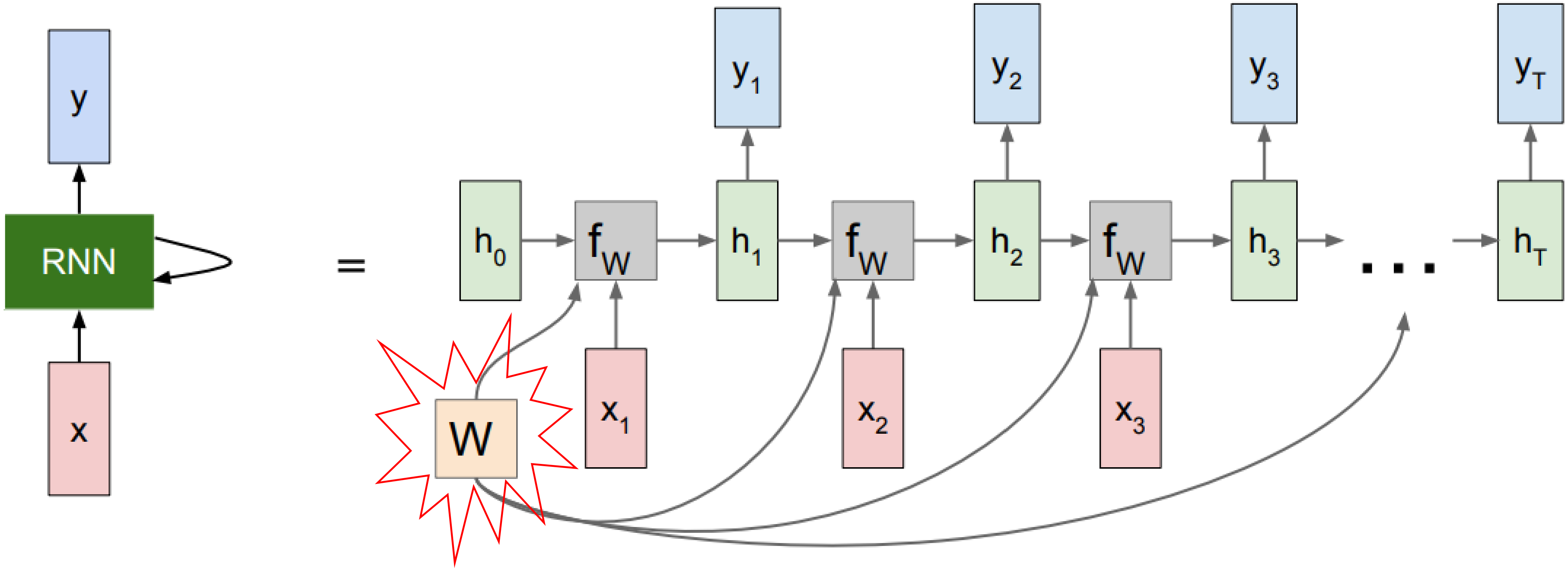
Recurrent Neural Networks (RNN)



RNN Computational Graph

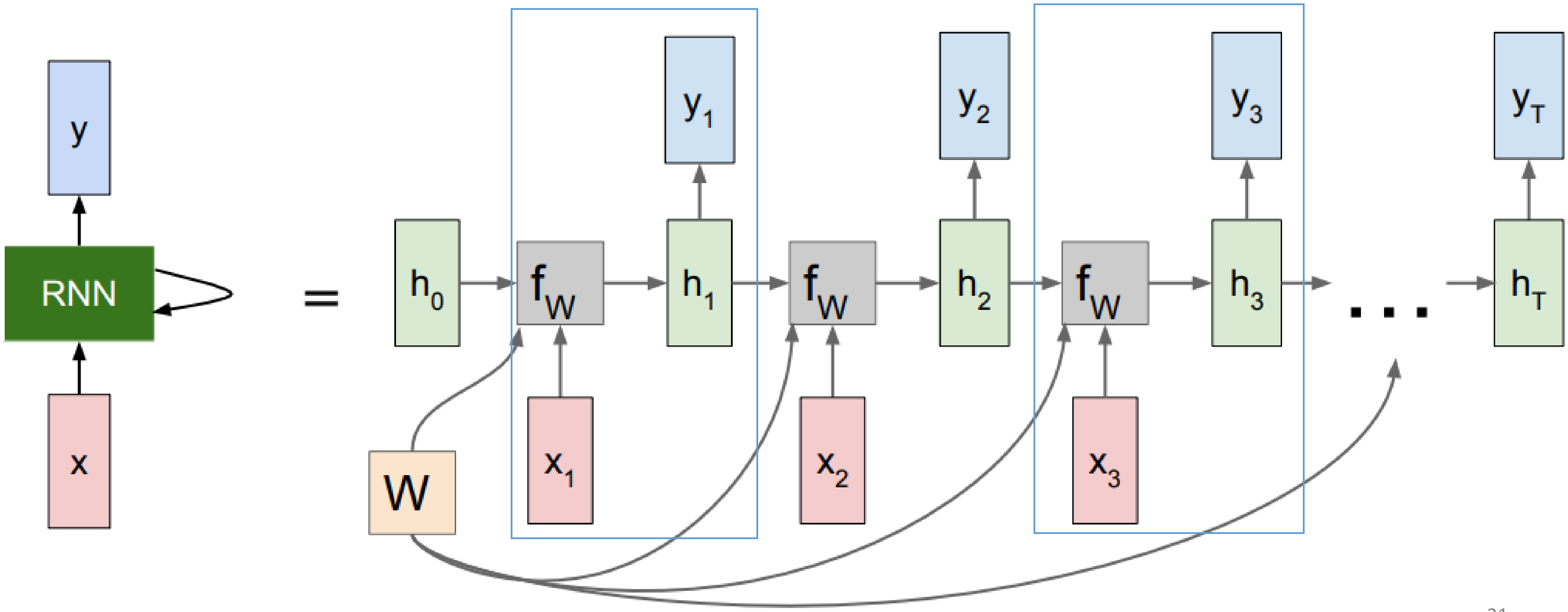


RNN Computational Graph

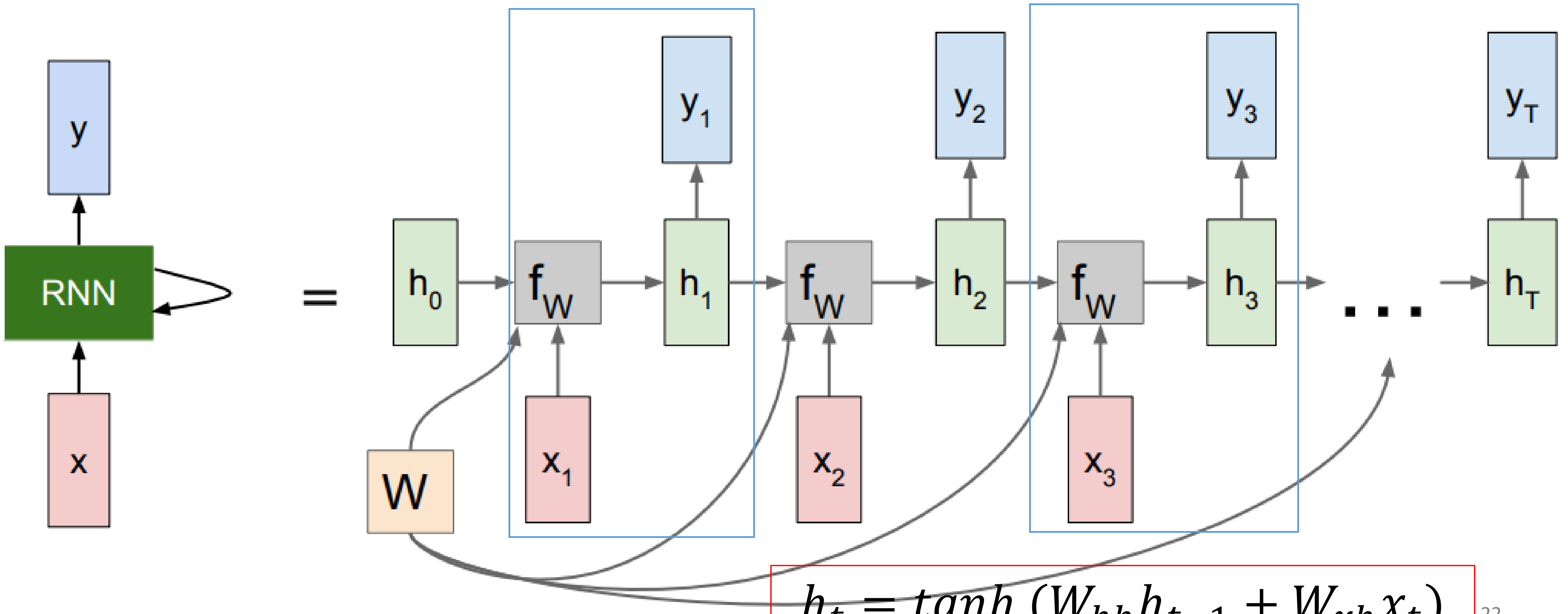


Do you notice any problem?

RNN Computational Graph



RNN Computational Graph



$$h_t = tanh\,(W_{hh}h_{t-1} + W_{xh}x_t)$$

Is it parallelizable?

- No! Note that for having h_t we first need h_{t-1}.

$$\boxed{h_t = tanh\,(W_{hh}h_{t-1} + W_{xh}x_t)}$$

Is it parallelizable?

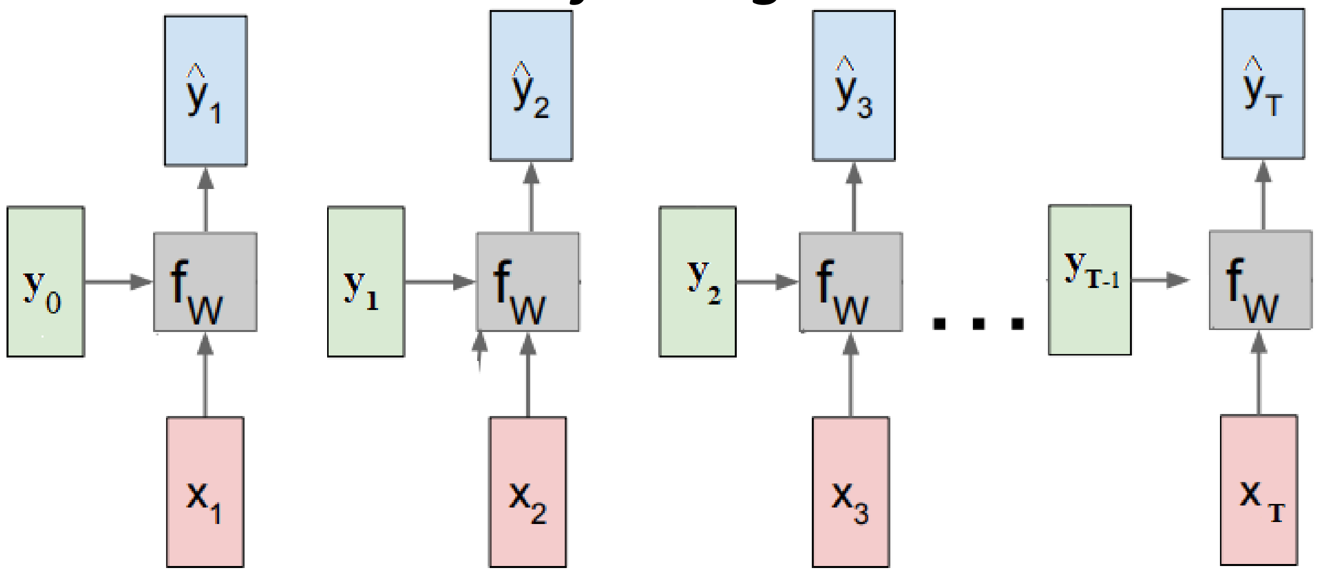
- No! Note that for having h_t we first need h_{t-1}.
- How to solve it??

Is it parallelizable?

- No! Note that for having h_t we first need h_{t-1}.
- How to solve it??
  - For some architectures we have *teacher forcing*

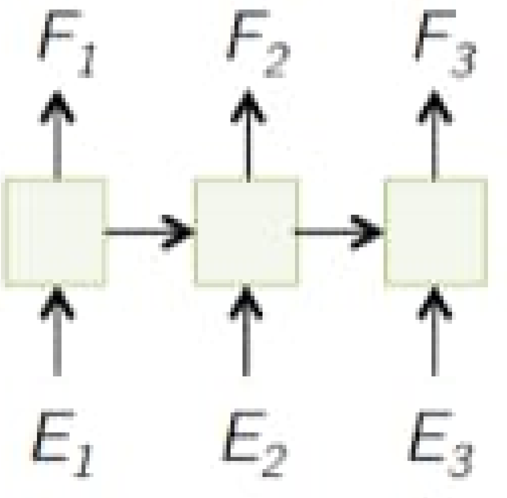
Is it parallelizable?

- No! Note that for having h_t we first need h_{t-1}.
- How to solve it??
  - For some architectures we have *teacher forcing*



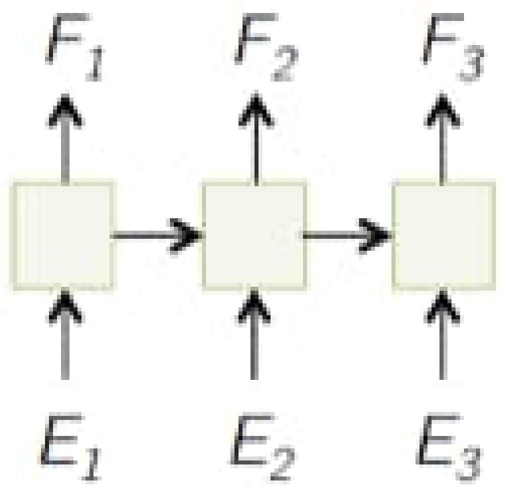
Before continuing...

- Translate a phrase from English to French.
- Is this architecture suitable for this problem?



Before continuing…

- Translate a phrase from English to French.
- Is this architecture suitable for this problem?



- Ans: No! Sentences might have different amount of words. We need to know the entire sentence before translating!

Types

One to one

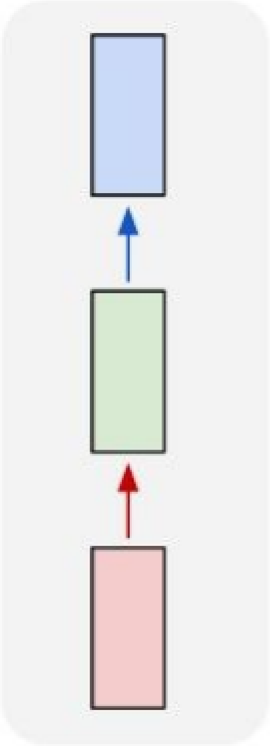


Image Classification

One to many

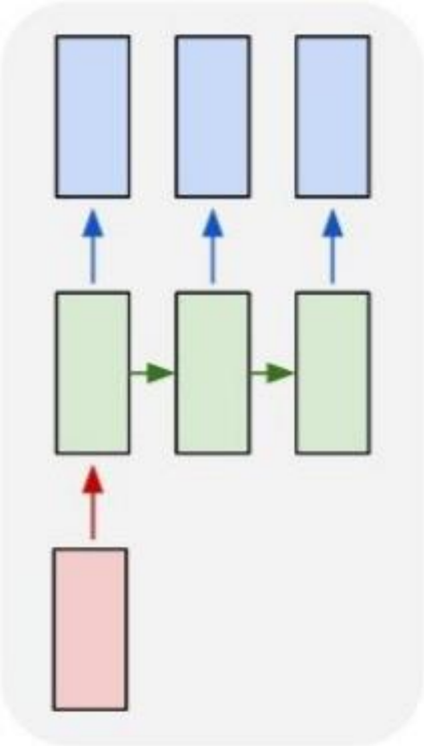
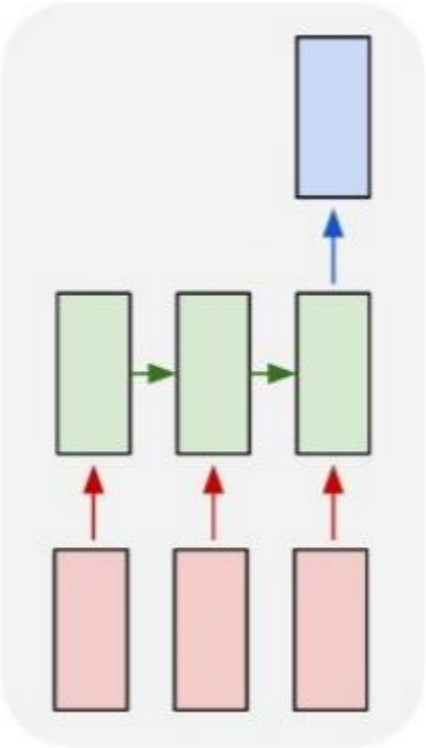


Image Captioning

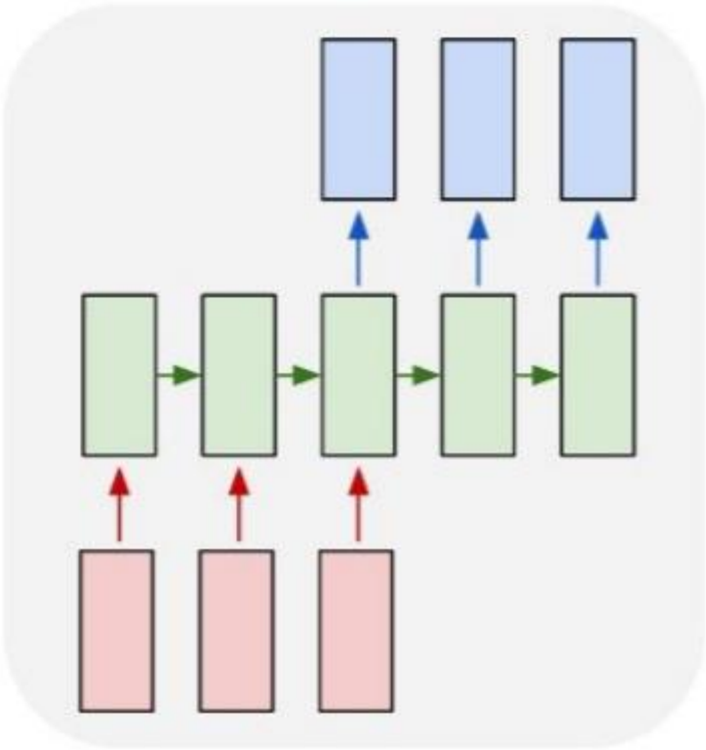
Many to one



Sentiment Classification

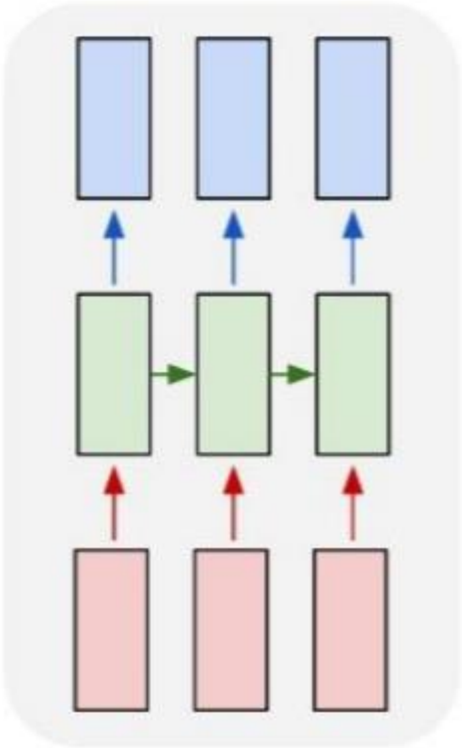
Types

Many to many



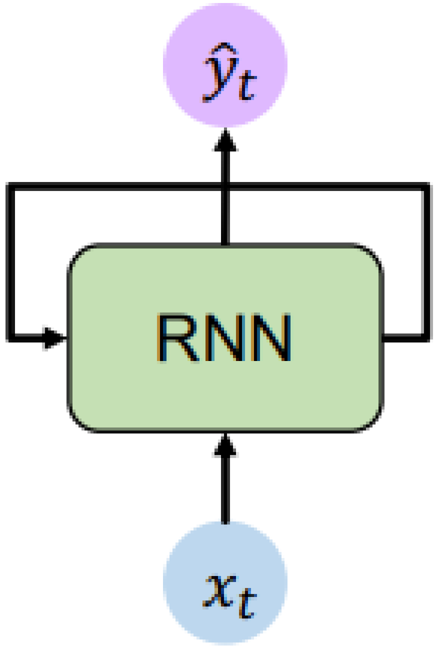
Machine Translation

Many to many

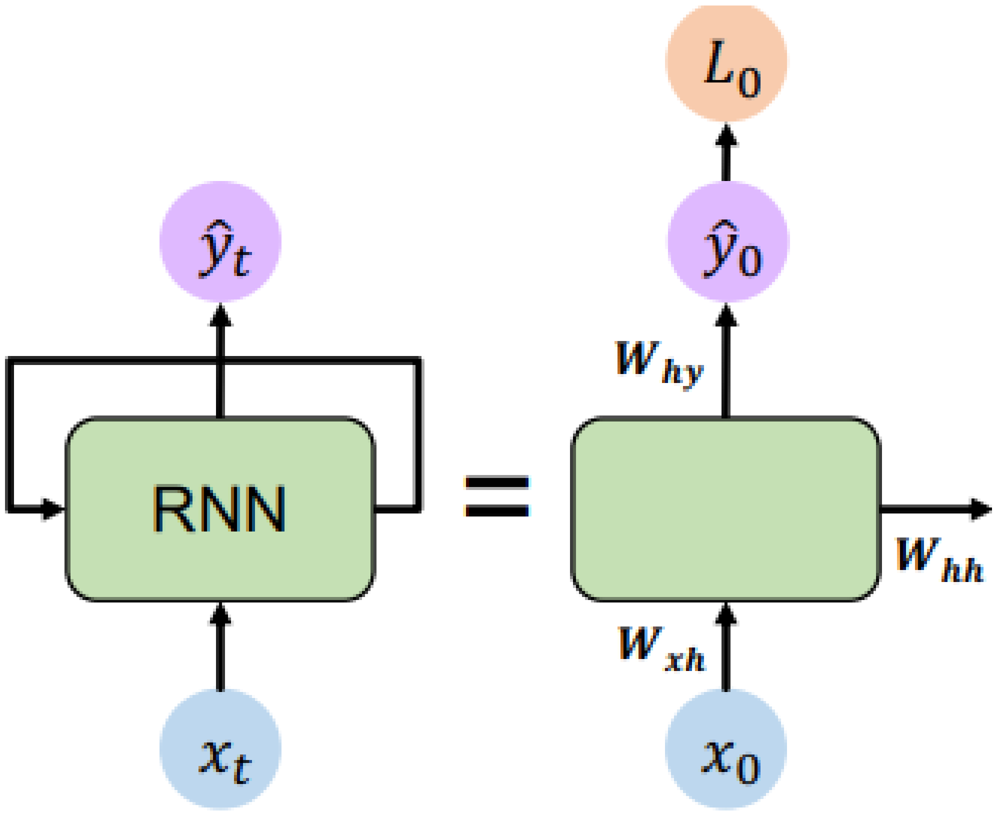


Video Classification on Frame Level

For each output...

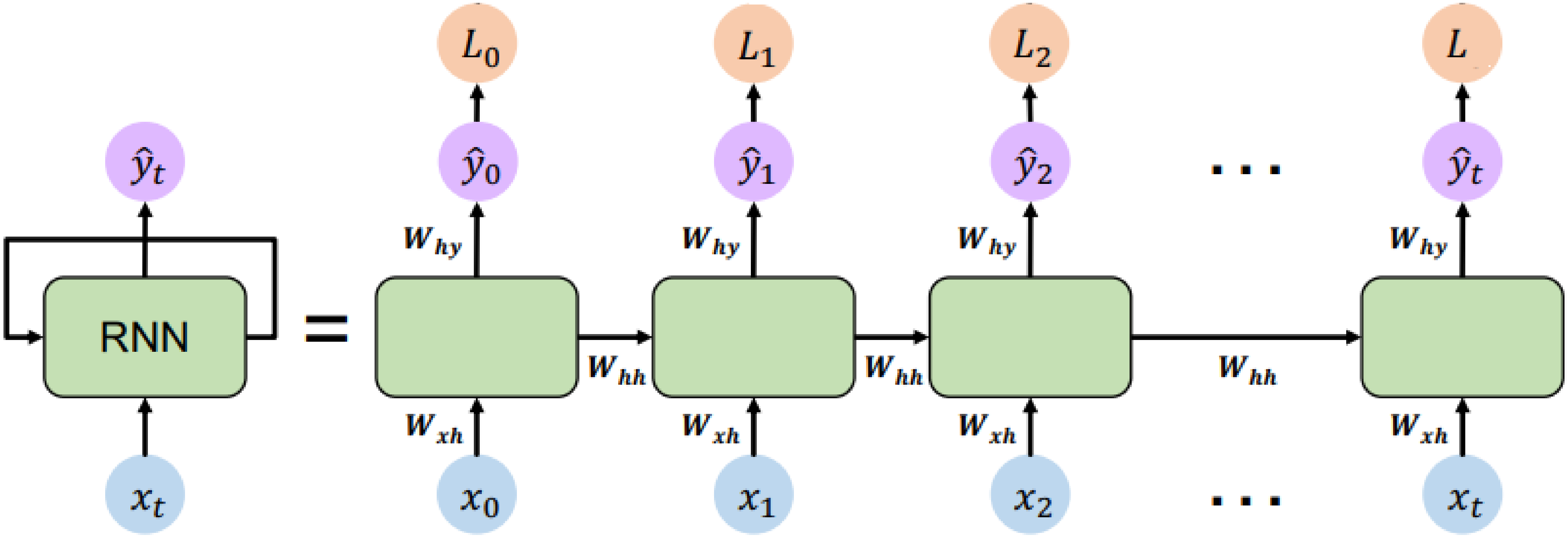


For each output...



$$\boxed{E_0(y_0, \hat{y}_0) = -y_0 \log(\hat{y}_0)}$$

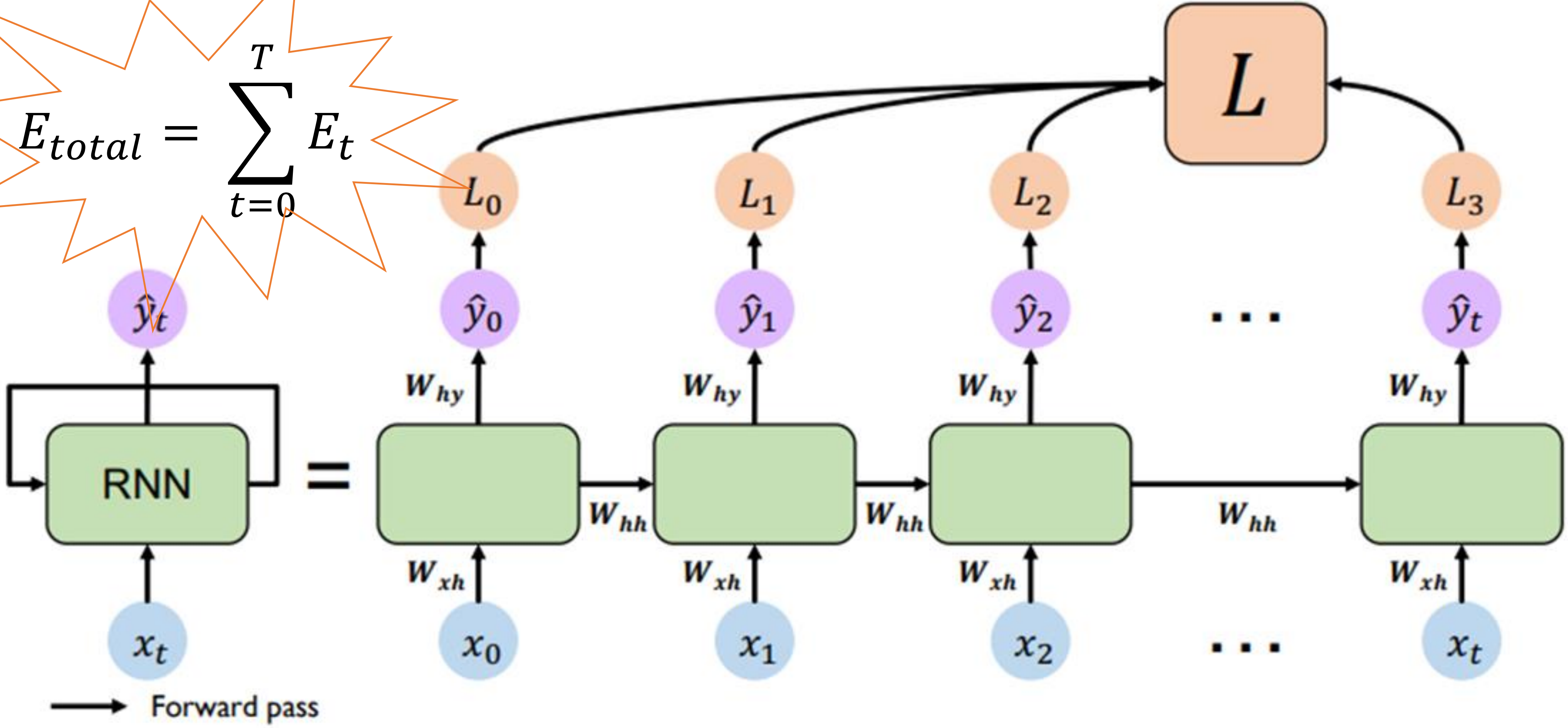
For each output...



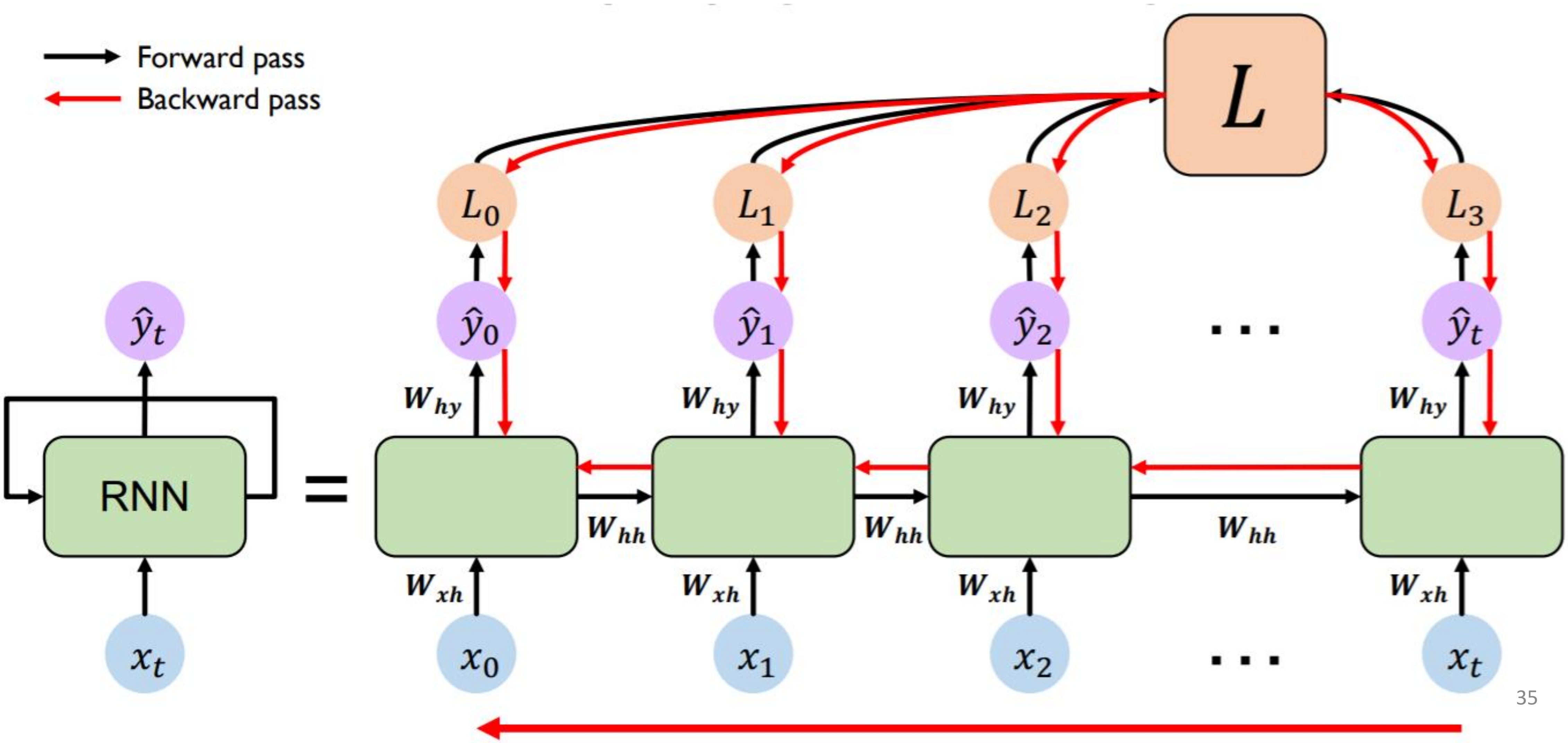
$$\boxed{E_t(y_t, \hat{y}_t) = -y_t \log(\hat{y}_t)}$$

For each output...

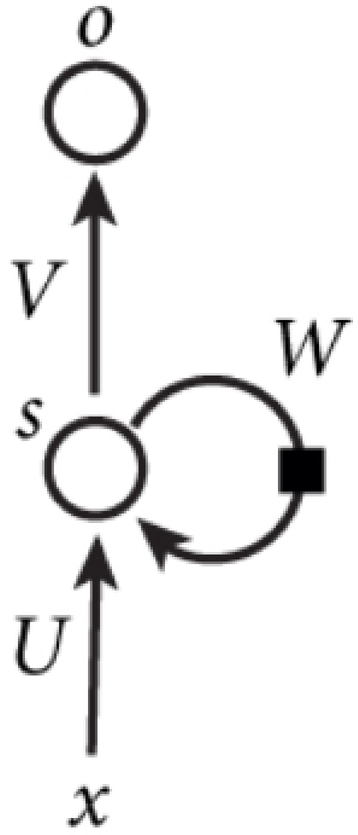
$$E_{total} = \sum_{t=0}^{T} E_t$$



Backpropagation through time (BPTT)



Backpropagation through time (BPTT)



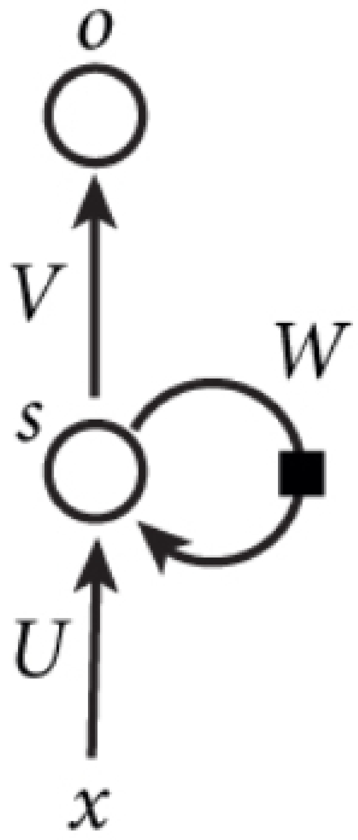
- We must backpropagate through W, V and U
- Let's assume we are in the third cell…

$$\begin{aligned}\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} \\ &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} \\ &= (\hat{y}_3 - y_3) \otimes s_3\end{aligned}$$

$z_3 = V s_3 \rightarrow s_3 = \tanh(U x_t + W s_2)$

For V it depends only on the values in cell number 3! But now let's see W and U.

Backpropagation through time (BPTT)



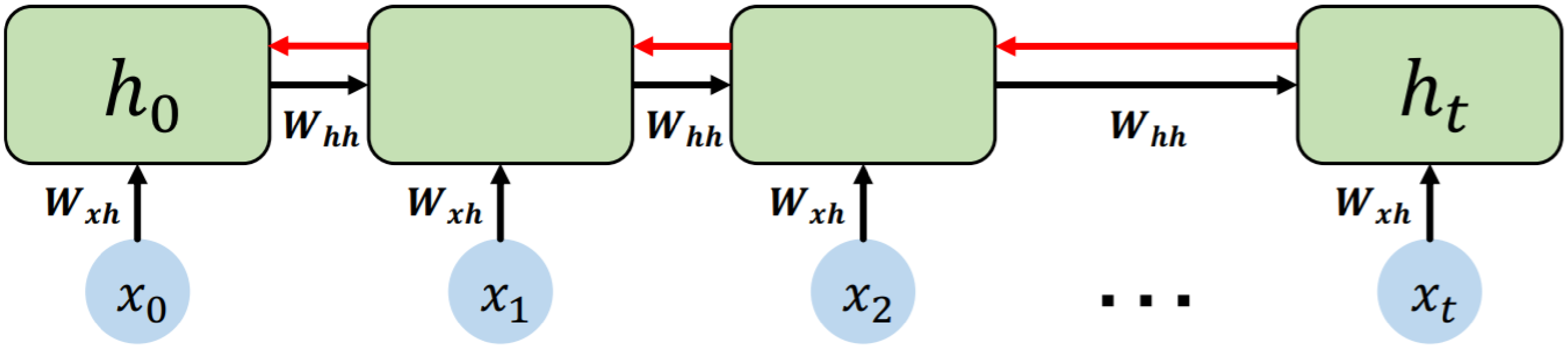
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

$$s_3 = \tanh(Ux_t + Ws_2)$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

The gradient will depend in all the previous cells!
It's the same situation for U!

Vanishing/Exploding Gradients Problem



Computing the BPTT algorithm involves the multiplication of many factors of the derivate of W (Whh) and the activation function (tanh). Therefore its possible to have some gradient problems.

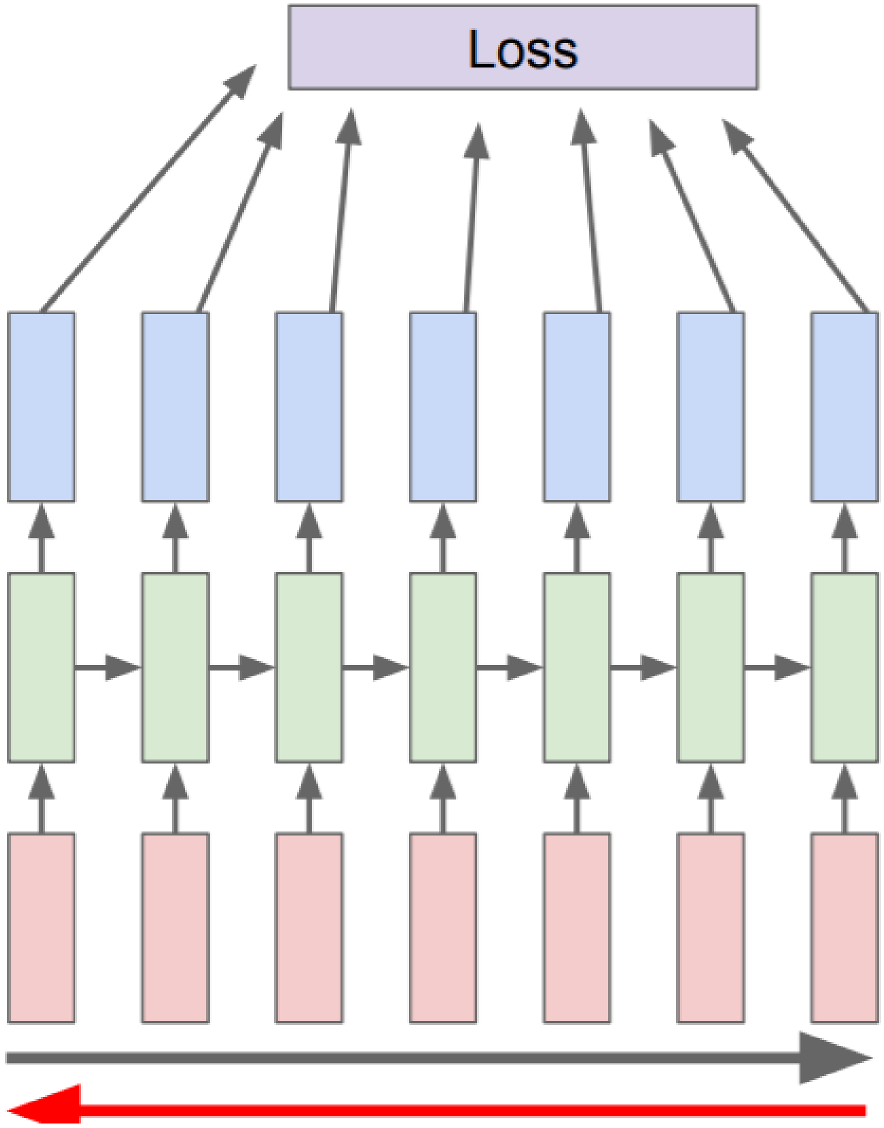
Vanishing/Exploding Gradients Problem

Many of the values > 1

Exploding gradients

1. Gradient clippling: Scale the gradient if it's bigger than a threshold.

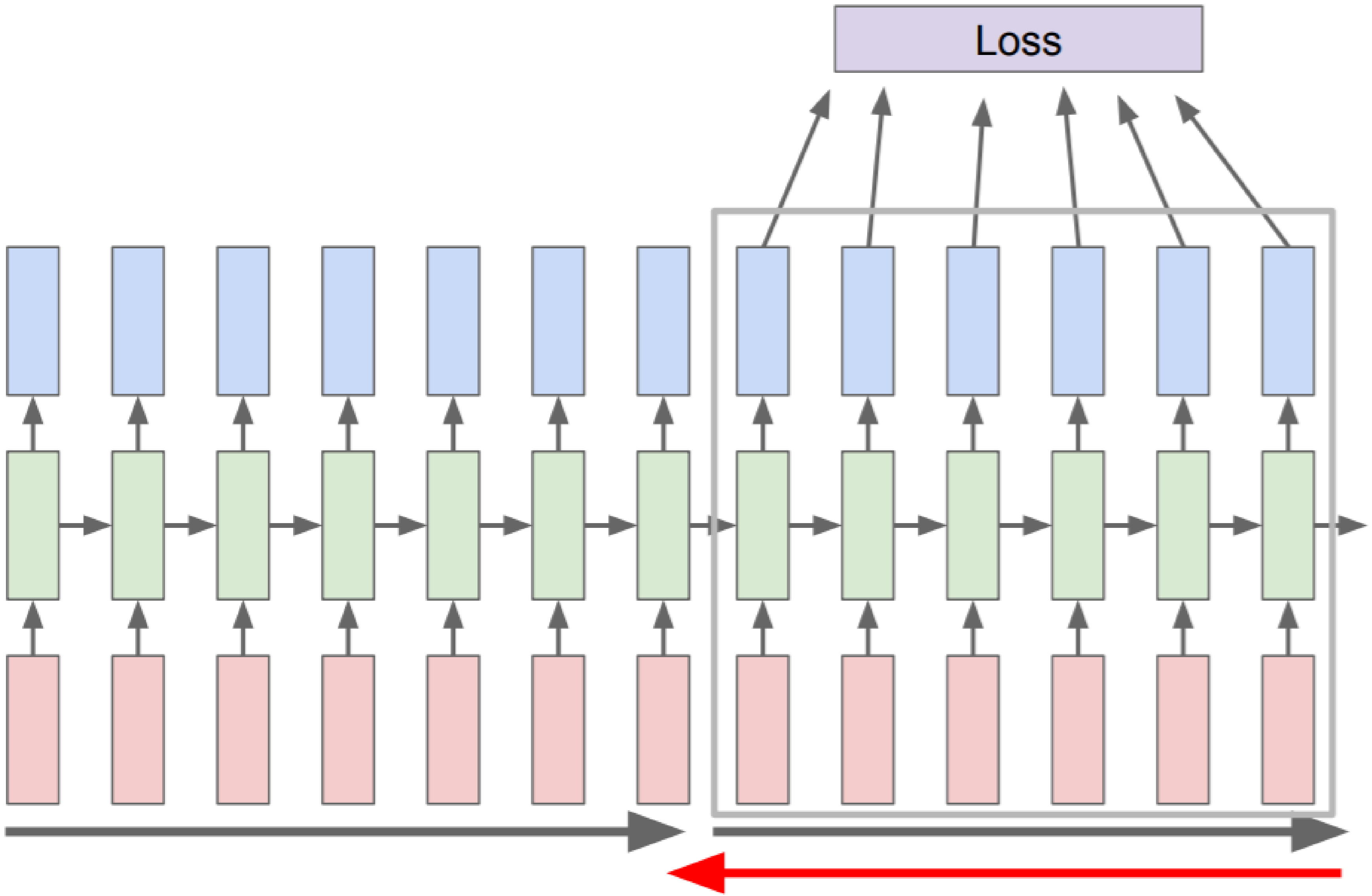
2. Truncated BPTT



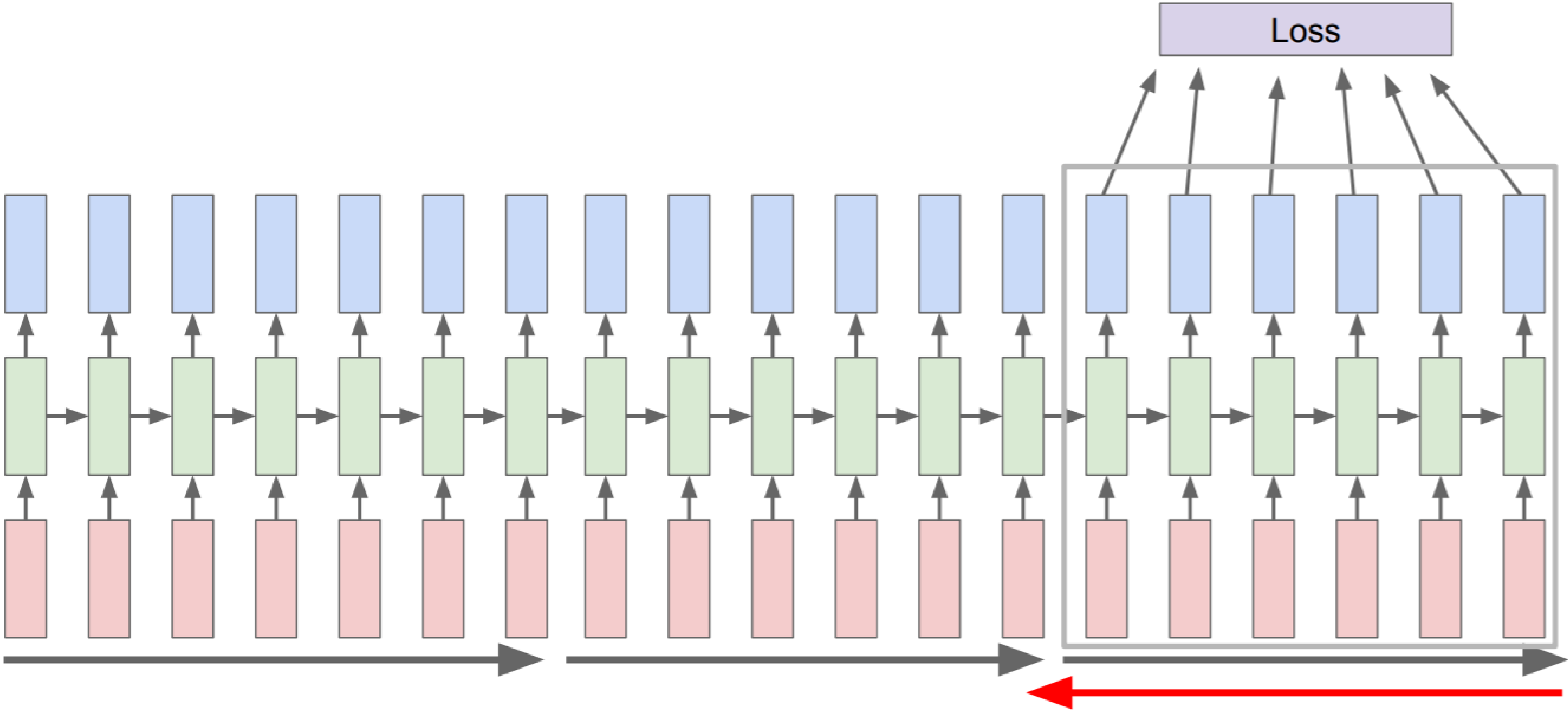
Truncated backpropagation through time (TBTT)

In long RNN's BPTT can be time consuming because it has to go through a lot of cells. So some researches are just backpropagating through a pre-stablished number of cells.

TBTT



TBTT



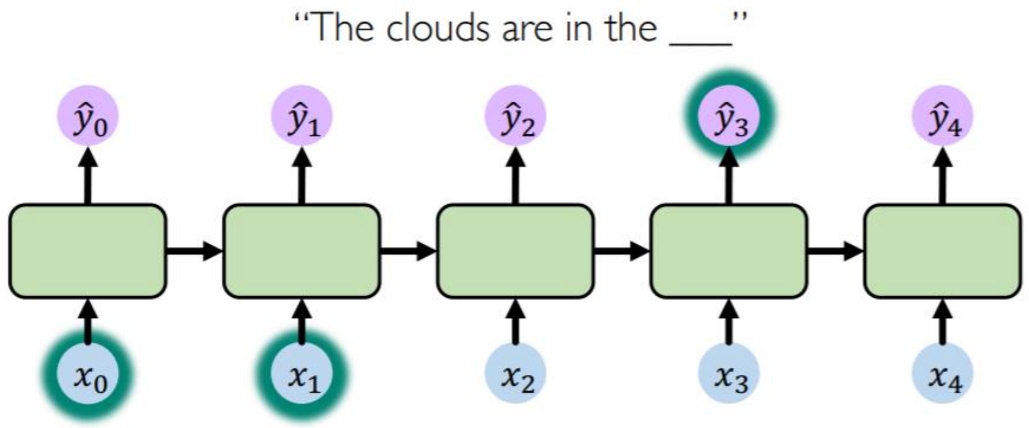
Vanishing/Exploding Gradients Problem

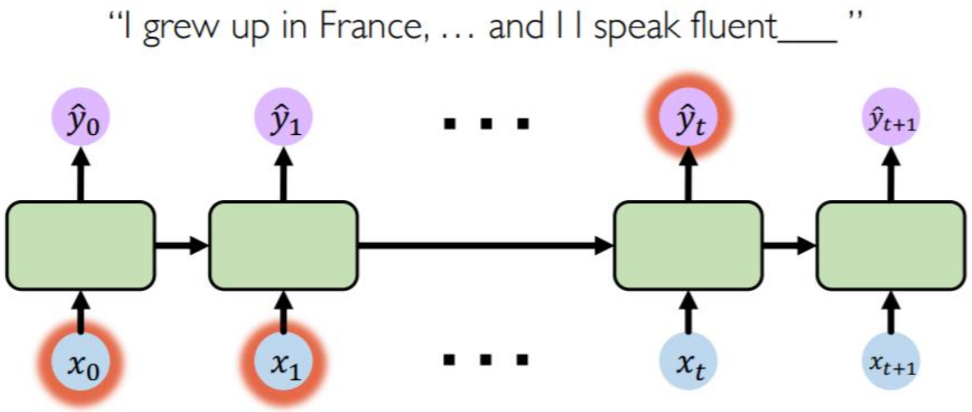
Many of the values < 1

Vanishing gradients

Bias network to short-term dependencies.

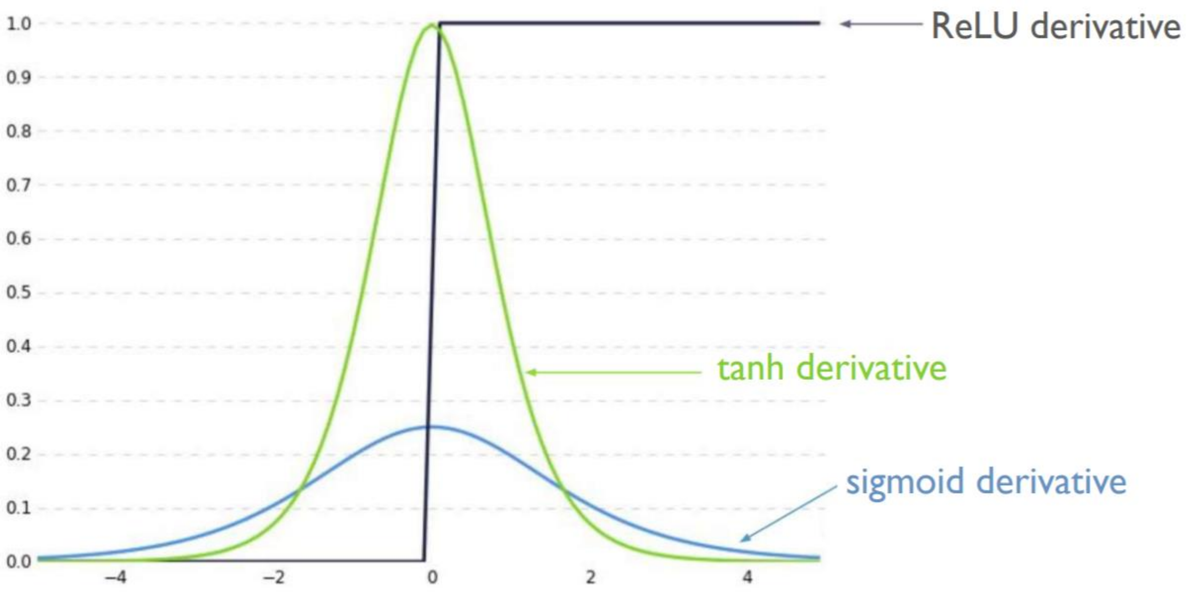
1. Change Activation function.
2. Weight Initialization
3. Change RNN architecture





Vanishing/Exploding Gradients Problem

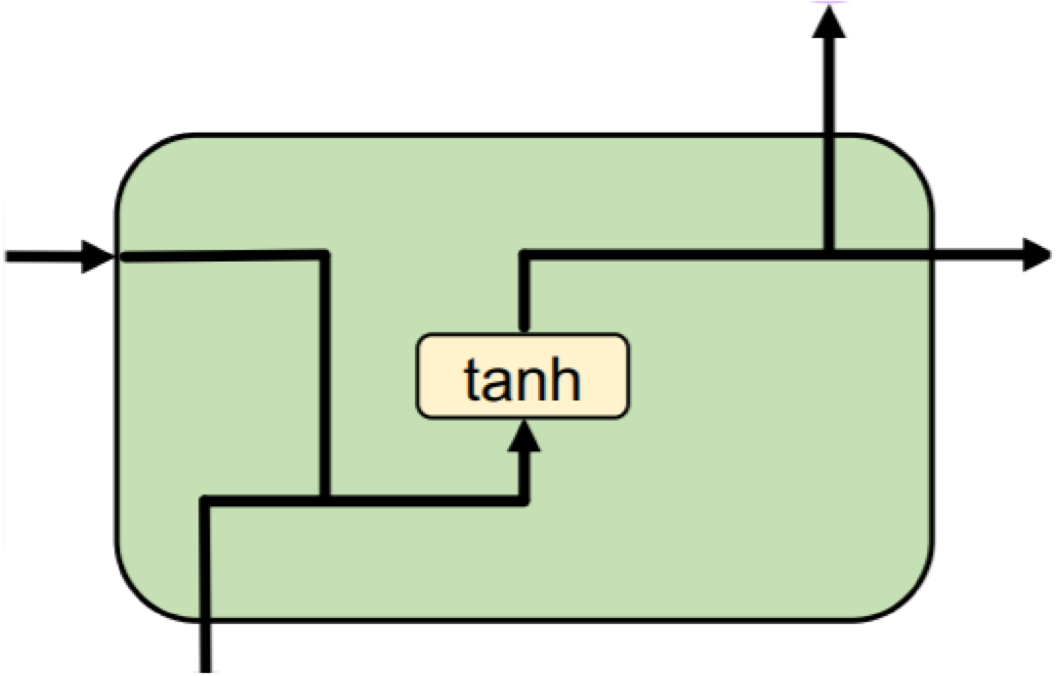
Activation Function

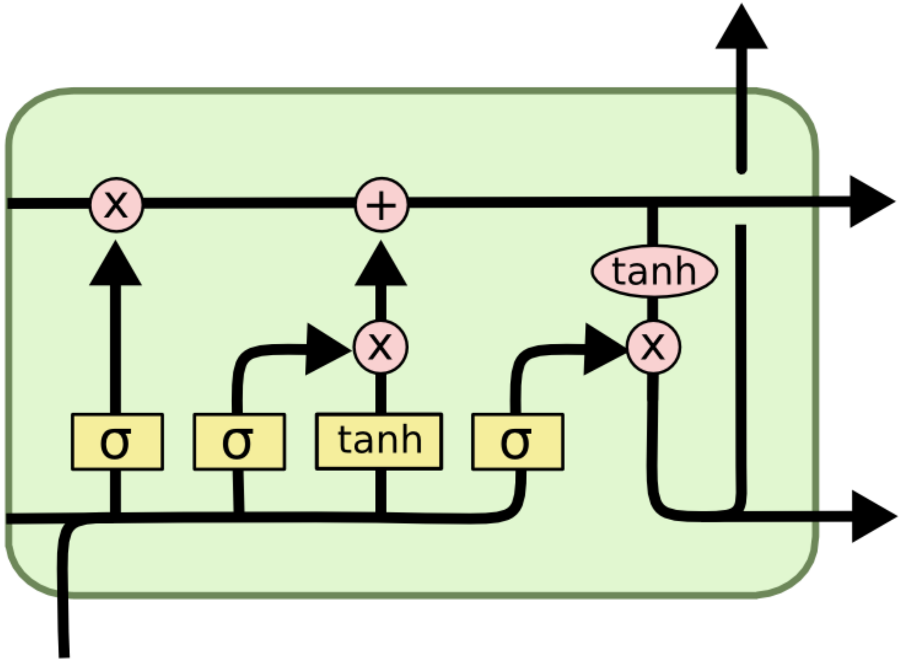


Weight Initialization

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

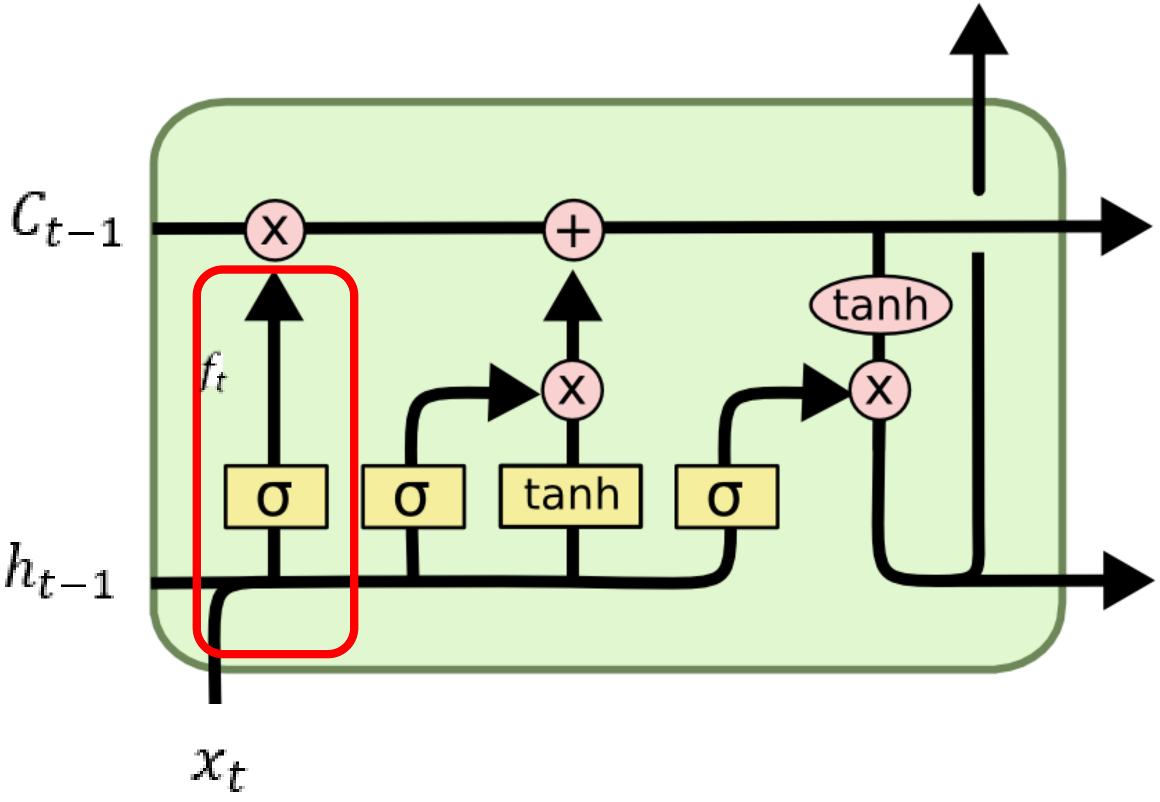
Long-Short Term Memory (LSTM)





Long-Short Term Memory (LSTM)

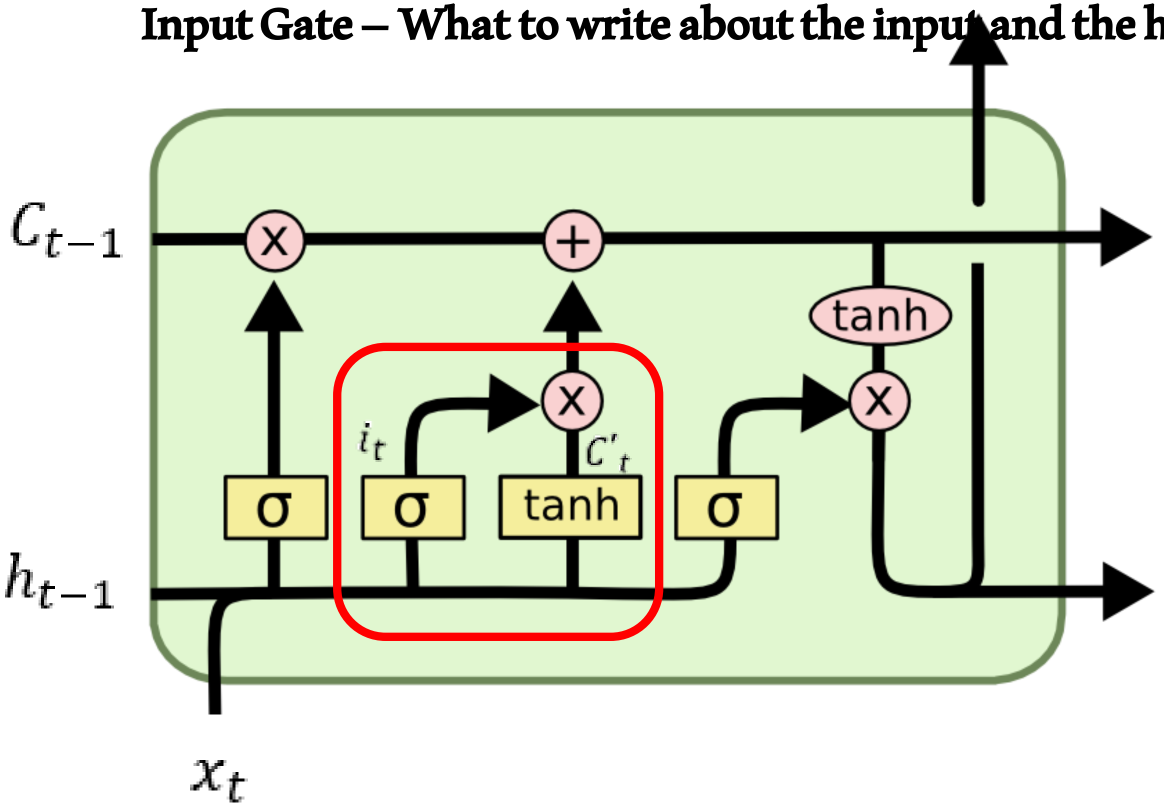
Forget Gate – What to forget about the input and the hidden state of the previous cell.



$$f_t = \sigma(\boldsymbol{W}_i[\, h_{t-1}, x_t \,] + b_f)$$

Long-Short Term Memory (LSTM)

Input Gate – What to write about the input and the hidden state of the previous cell into the cell state

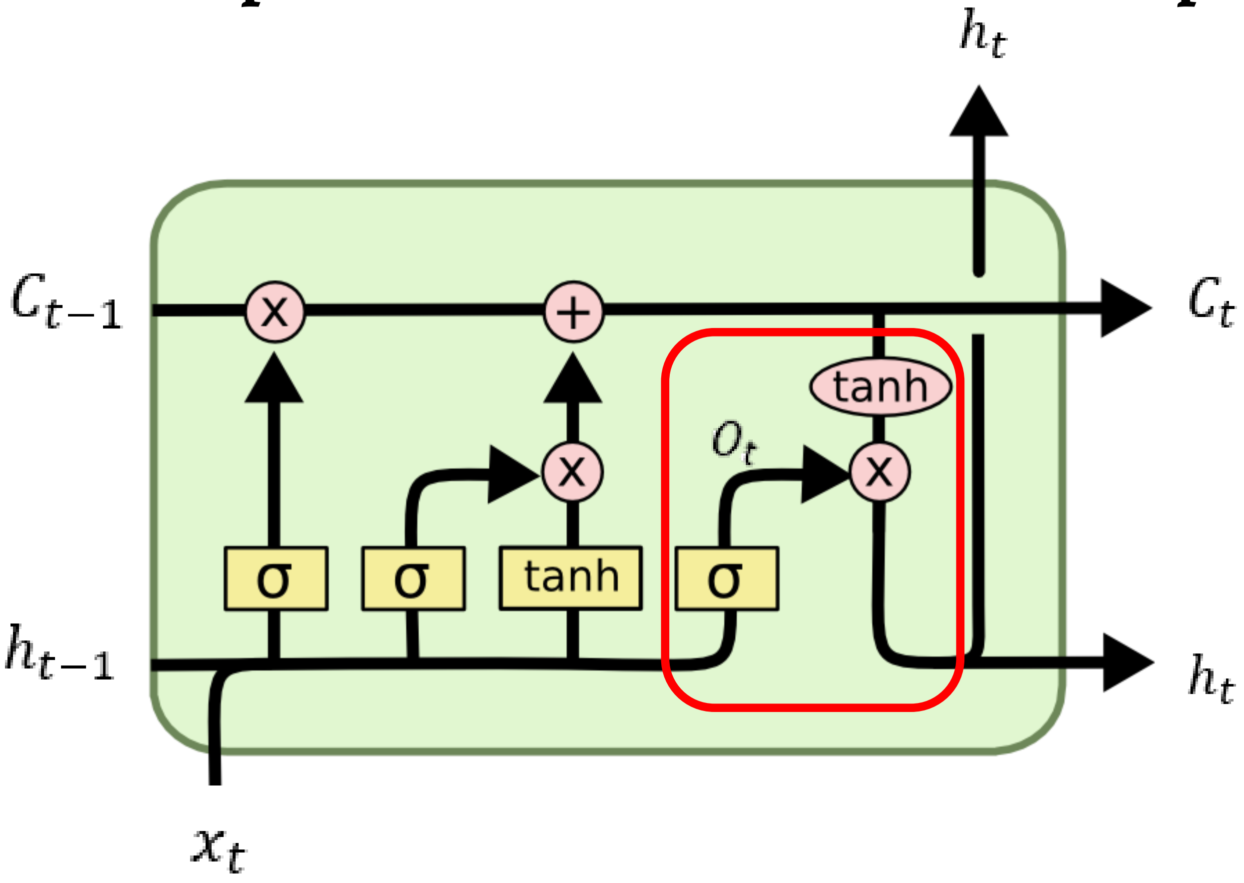


$$i_t = \sigma(\boldsymbol{W}_i[\, h_{t-1}, x_t \,] + b_i)$$

$$\tilde{C}_t = \tanh(\boldsymbol{W}_C[\, h_{t-1}, x_t \,] + b_C)$$

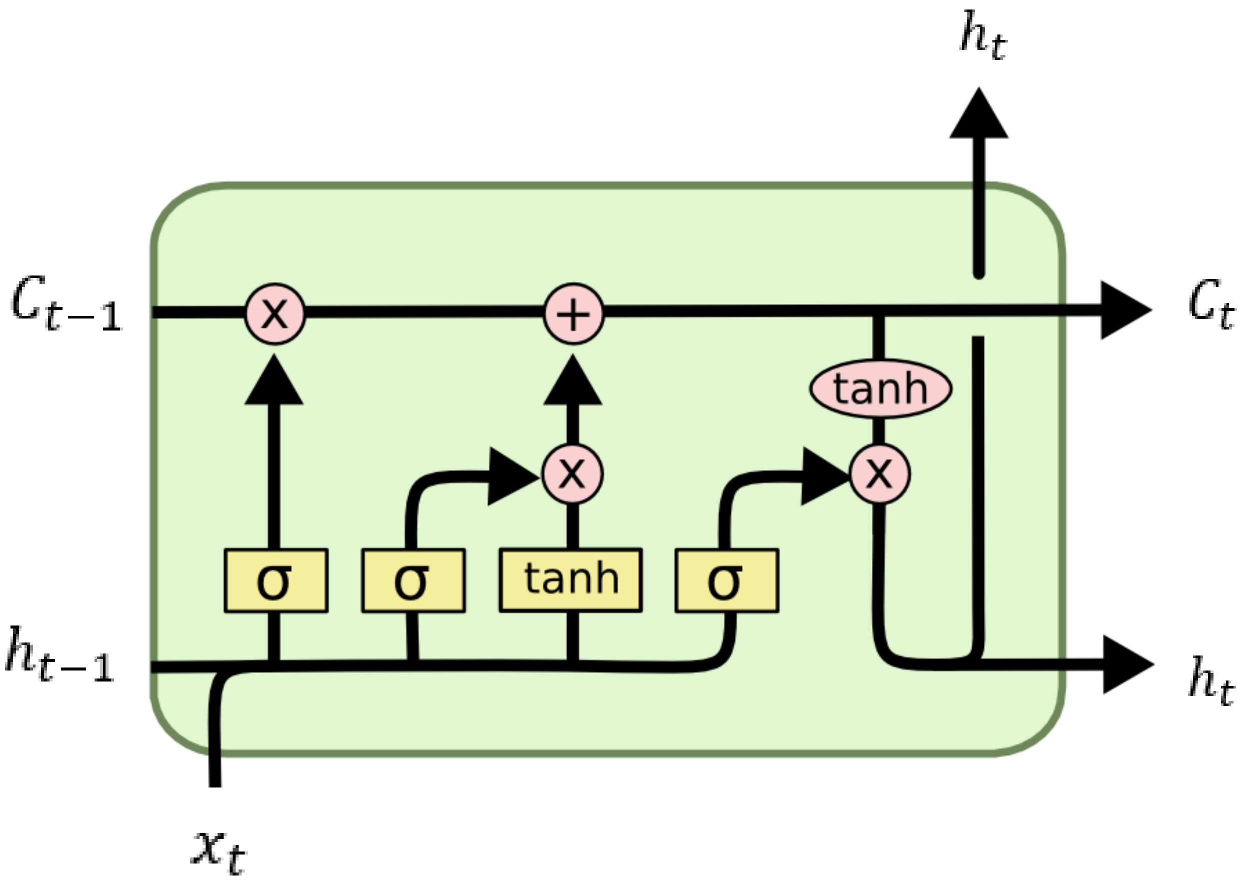
Long-Short Term Memory (LSTM)

Output Gate – What to show about the input and the cell state to the next cell.



$$o_t = \sigma(\boldsymbol{W_o}[\, h_{t-1}, x_t \,] + b_o)$$

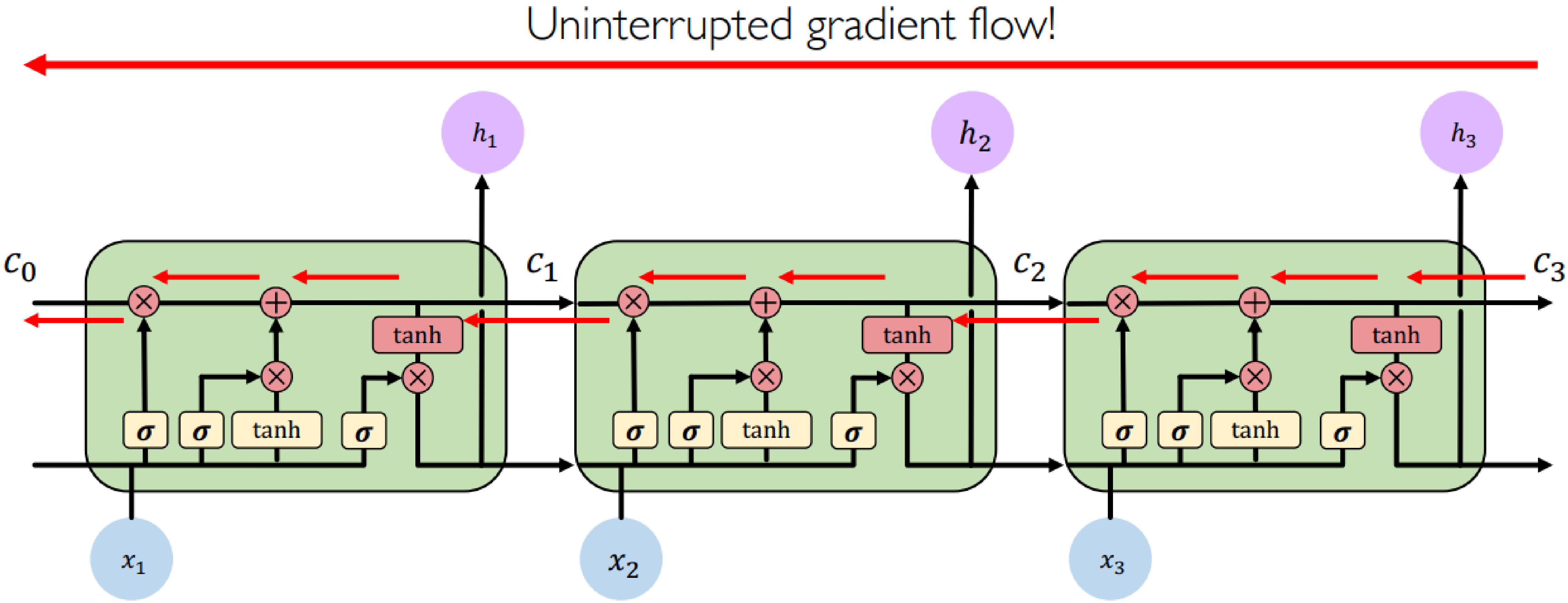
Long-Short Term Memory (LSTM)



$$c_t = f \odot c_{t-1} + i \odot C'_t$$

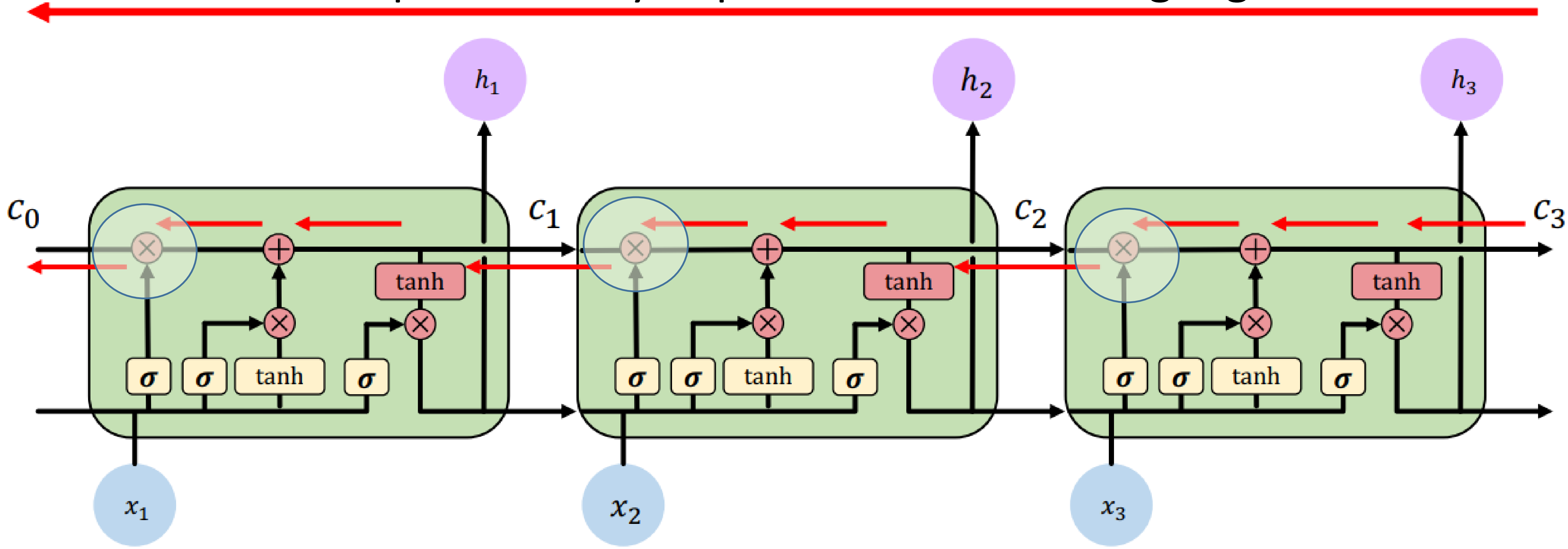
$$h_t = o \odot \tanh(c_t)$$

Long-Short Term Memory (LSTM)

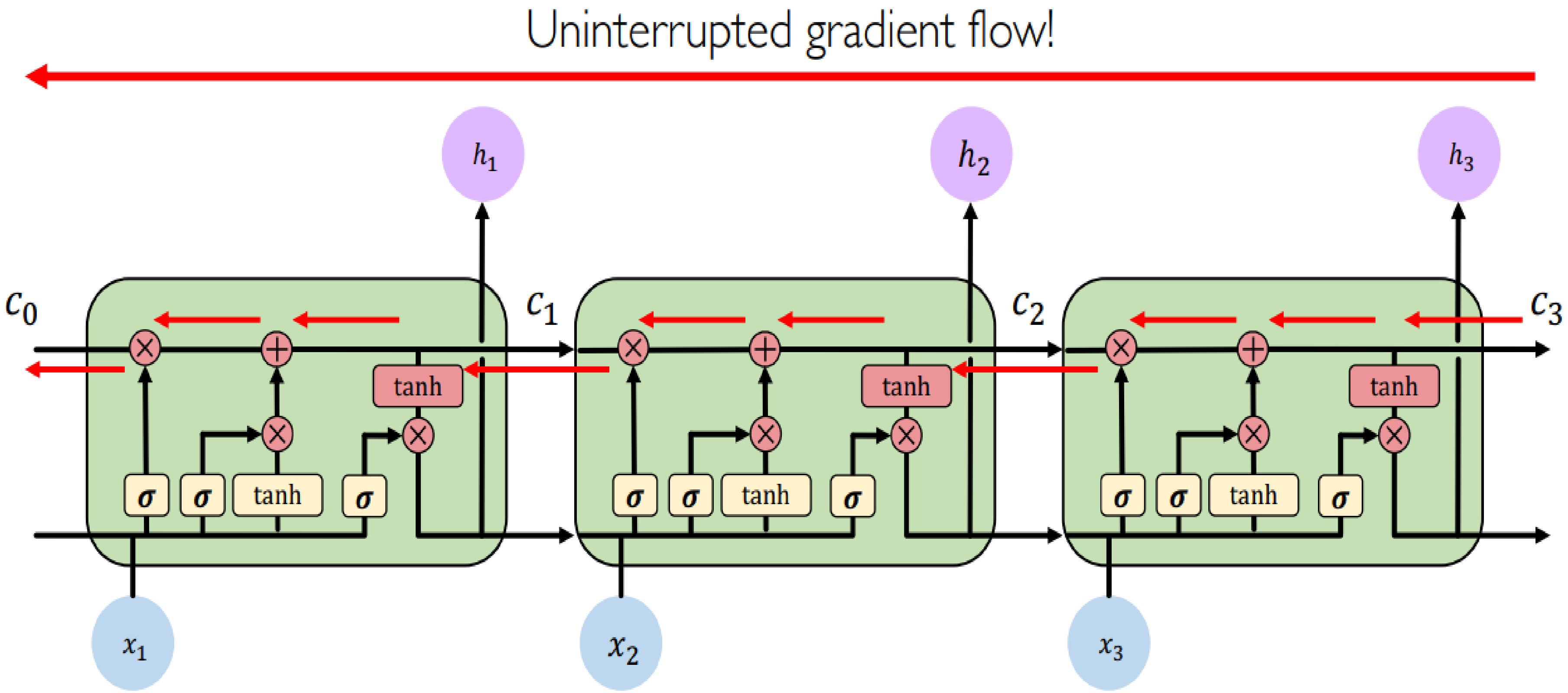


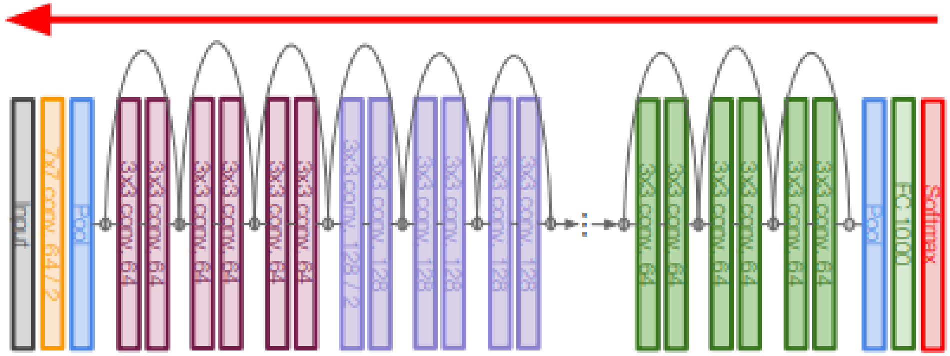
Long-Short Term Memory (LSTM)

Backward process only dependent with the forget gate!!!



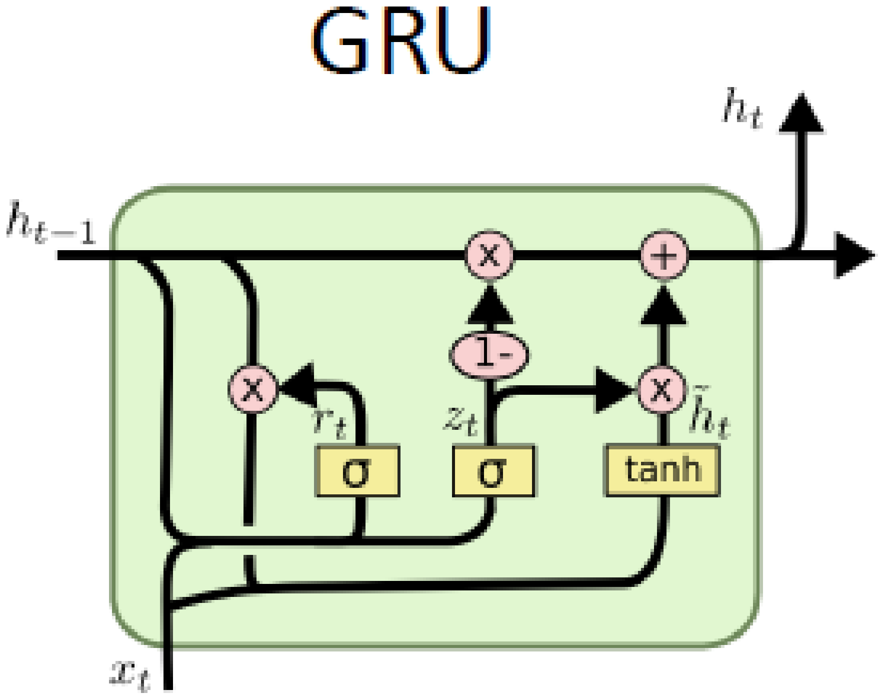
Long-Short Term Memory (LSTM)





Similar to RESNET!

Gradient Recurrent Unit(GRU)



$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

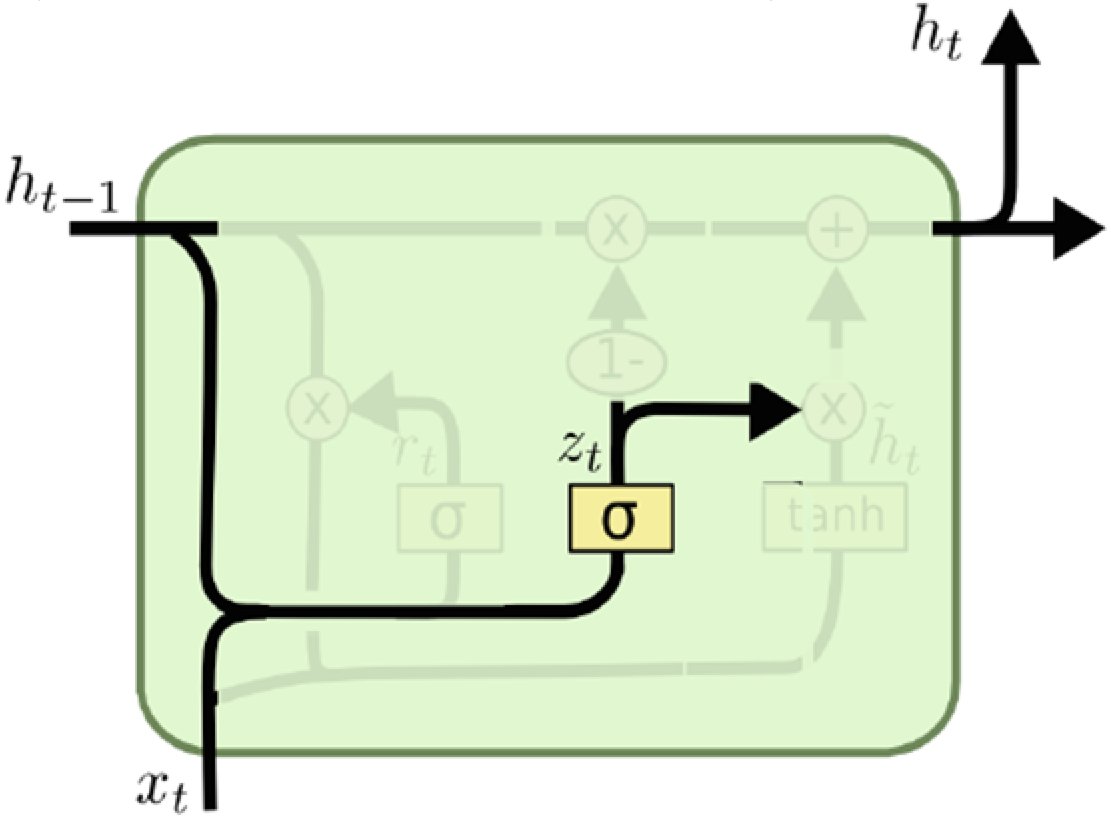
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

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

Gradient Recurrent Unit(GRU)

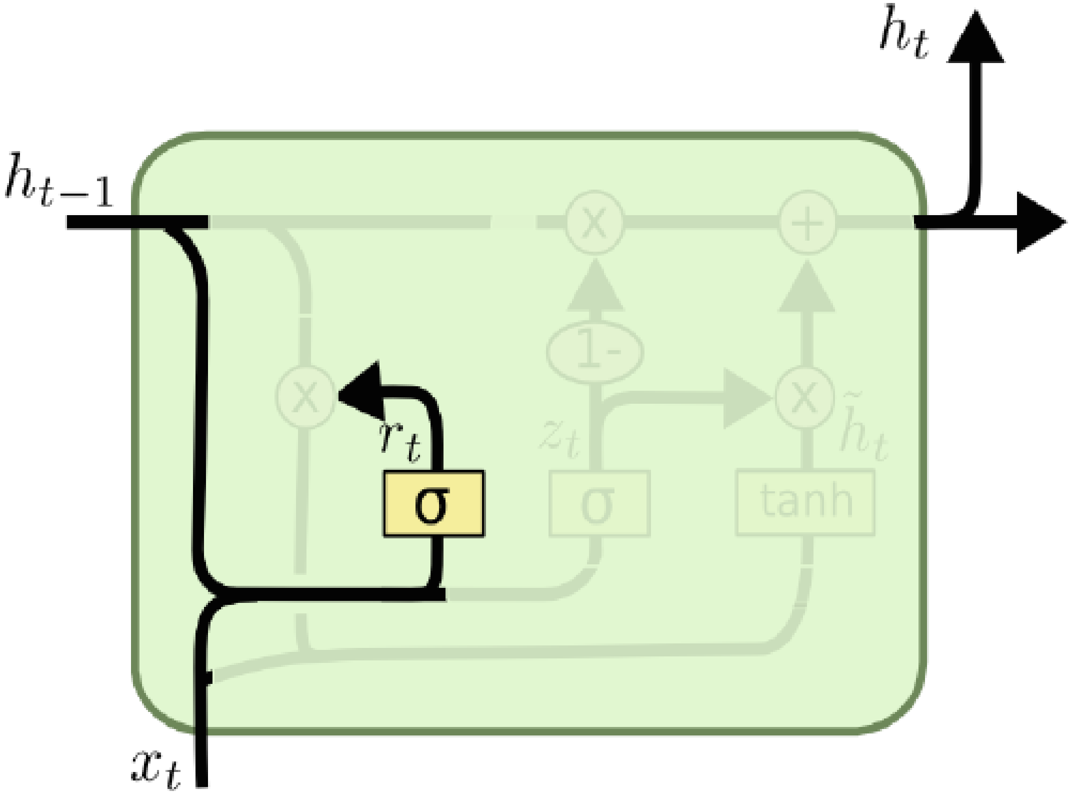
Update Gate – Determine how much of past information needs to pass to next step



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

Gradient Recurrent Unit(GRU)

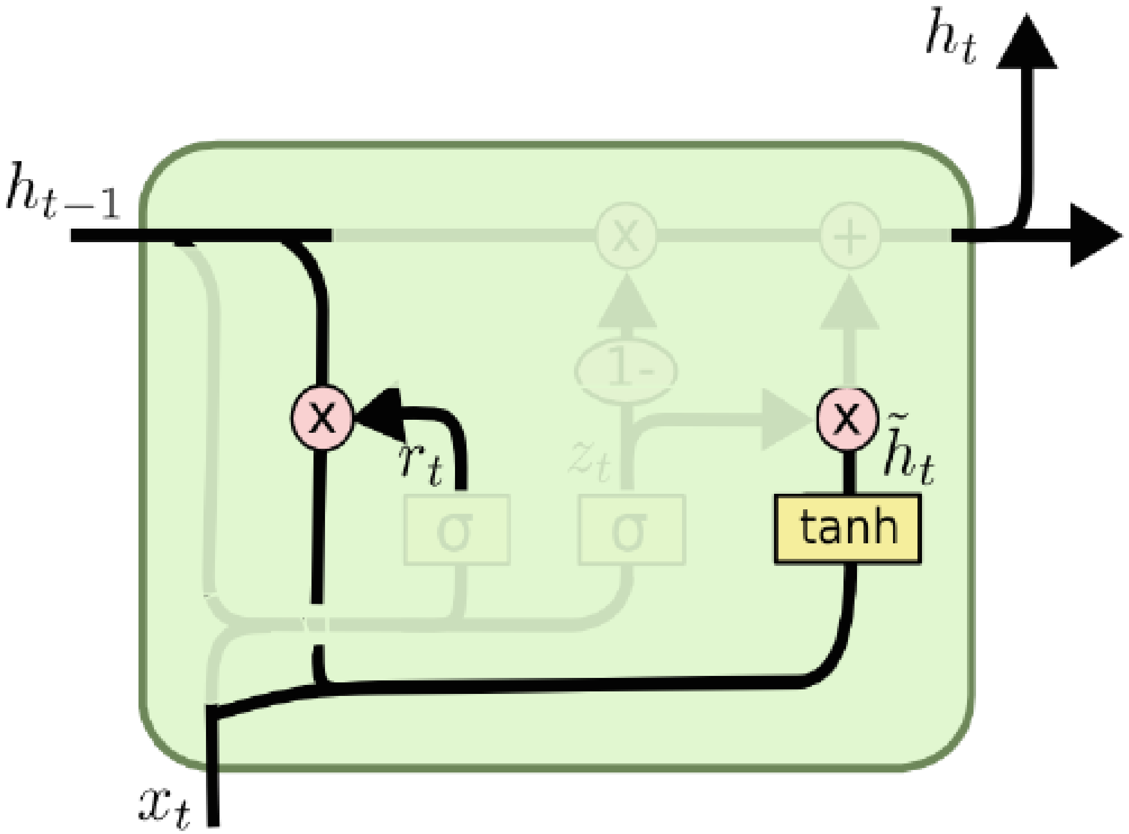
Reset Gate – Decide how much of past information forget



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

Gradient Recurrent Unit(GRU)

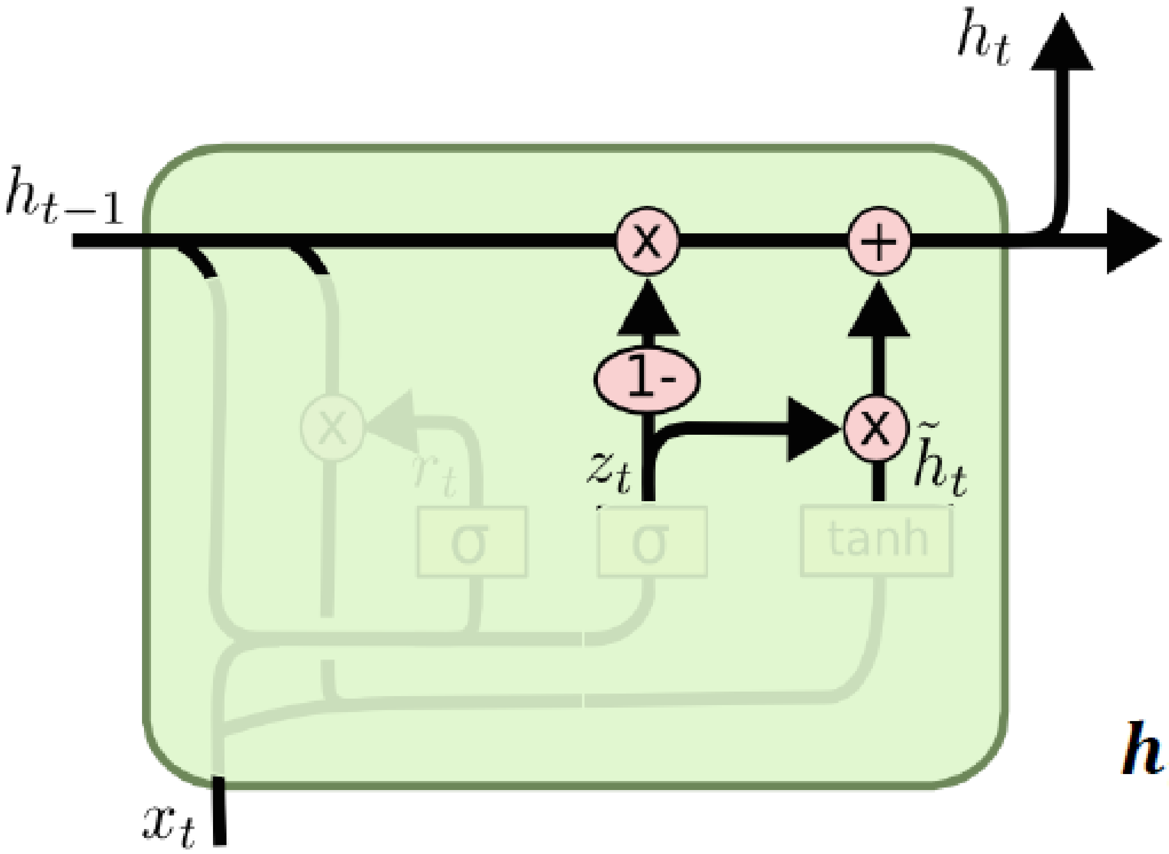
Current Memory Content– Use the reset gate to store relevant information from the past



$$\widetilde{h_t} = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

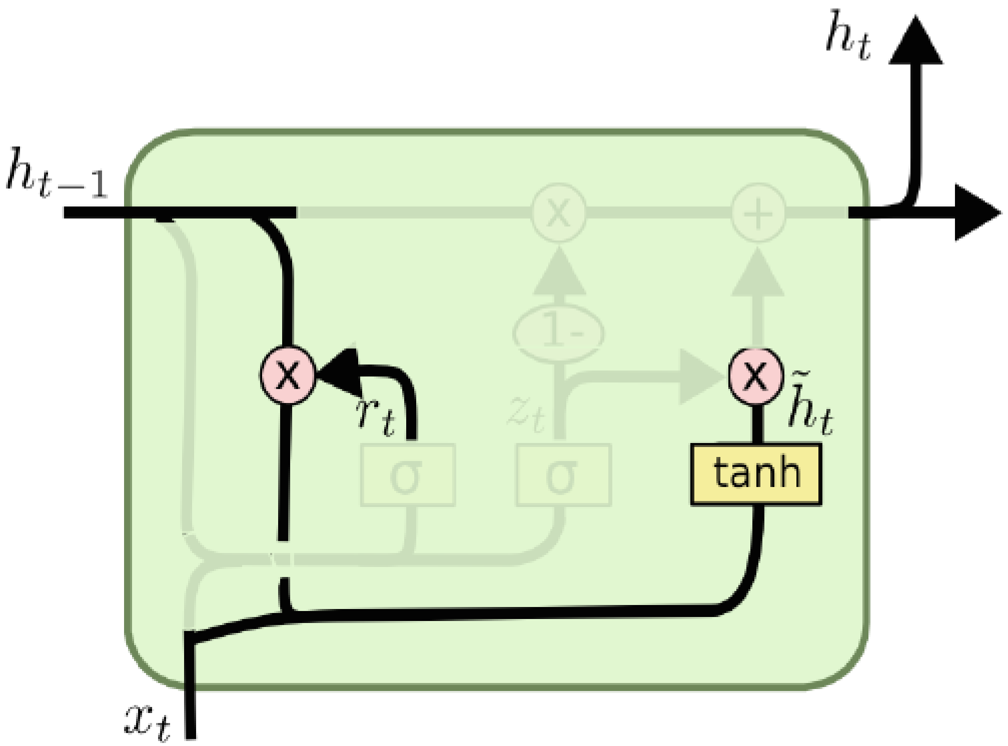
Gradient Recurrent Unit(GRU)

Final Current memory at time step – Holds information for the current unit.



$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \widetilde{h_t}$$

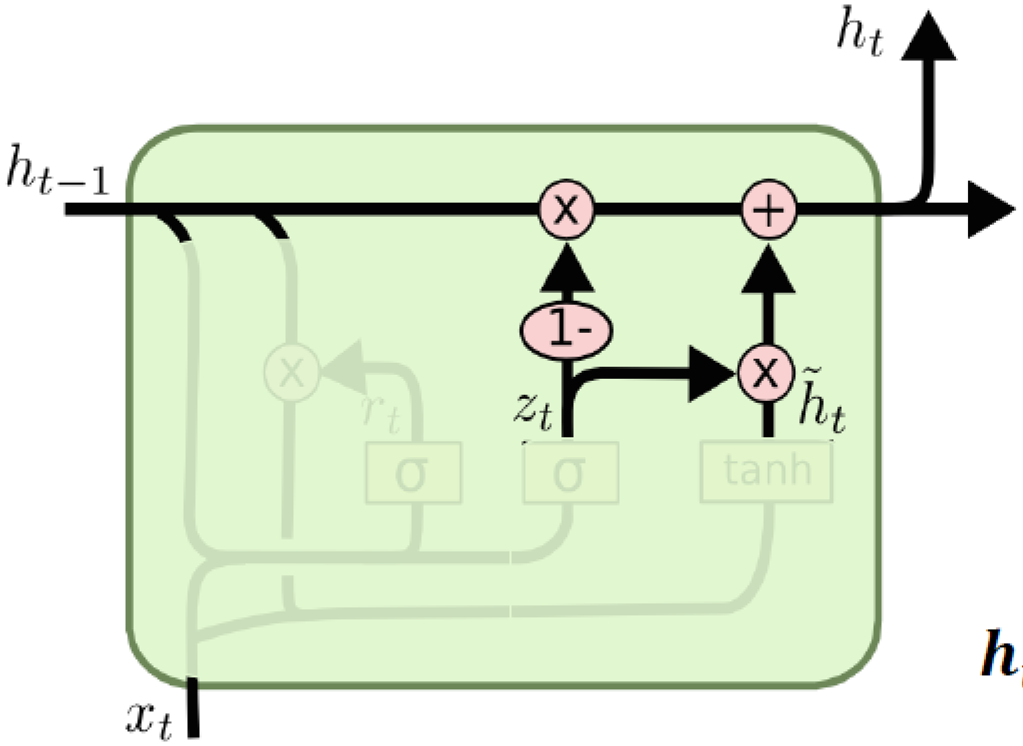
GRU:Current Memory Content



The new memory content will use the reset gate to store the relevant information from the past.

$$\tilde{h}_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

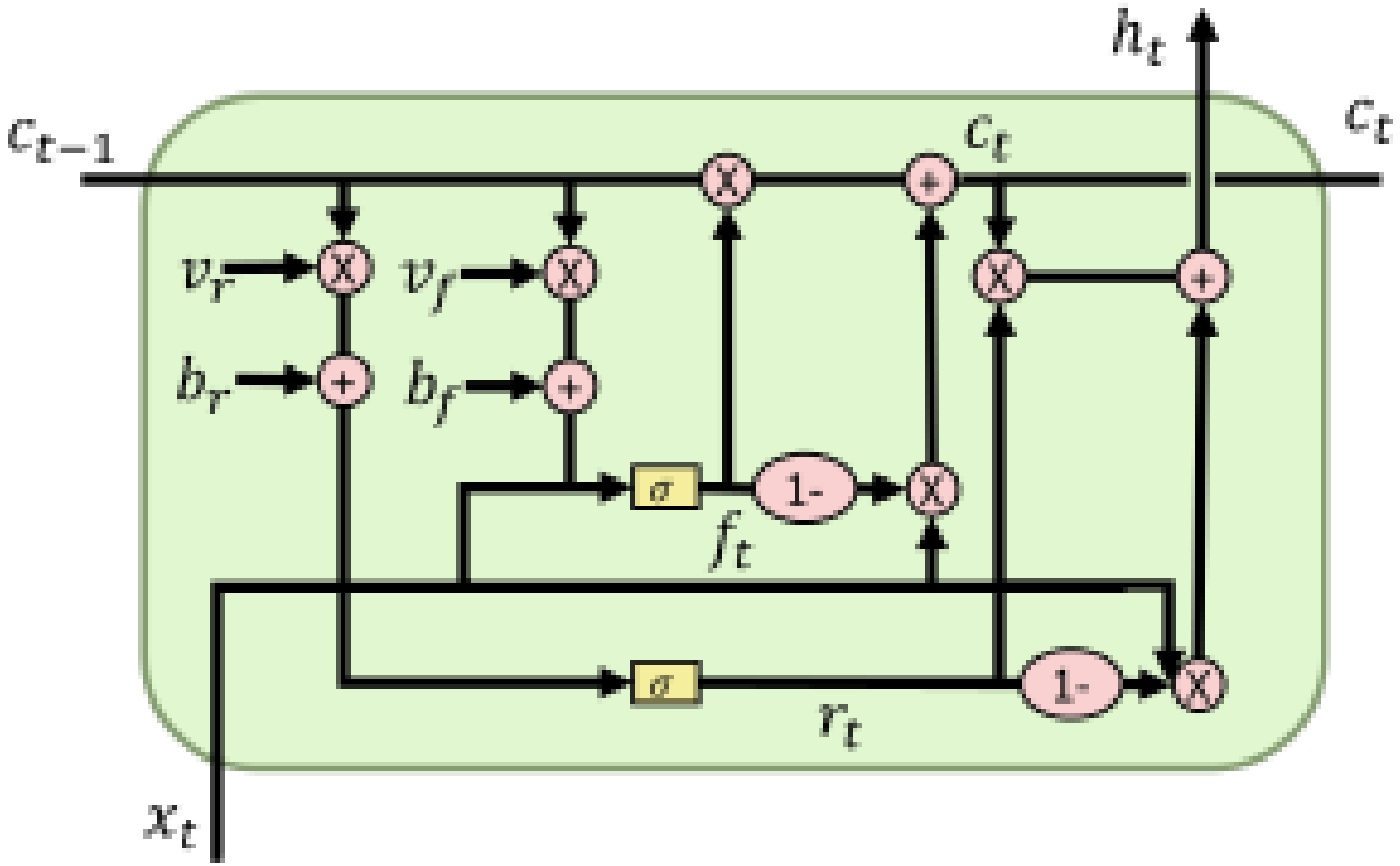
GRU:Final Current memory at time step



Holds information for the current unit and passes it down to the network.

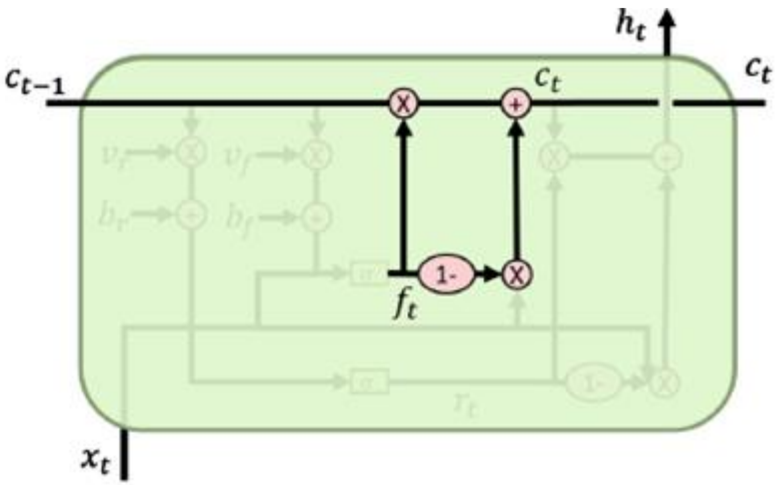
$$\boldsymbol{h_t} = \boldsymbol{z_t} \odot \boldsymbol{h_{t-1}} + (\boldsymbol{1} - \boldsymbol{z_t}) \odot \widetilde{\boldsymbol{h_t}}$$

Simple Recurrent Unit SRU)



$$
\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{W}_f \mathbf{x}_t + \mathbf{v}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot (\mathbf{W}\mathbf{x}_t) \\
\mathbf{r}_t &= \sigma\left(\mathbf{W}_r \mathbf{x}_t + \mathbf{v}_r \odot \mathbf{c}_{t-1} + \mathbf{b}_r\right) \\
\mathbf{h}_t &= \mathbf{r}_t \odot \mathbf{c}_t + (1 - \mathbf{r}_t) \odot \mathbf{x}_t
\end{aligned}
$$

Simple Recurrent Unit(SRU)



$$f_t = \sigma\left(W_f x_t + v_f \odot c_{t-1} + b_f\right)$$

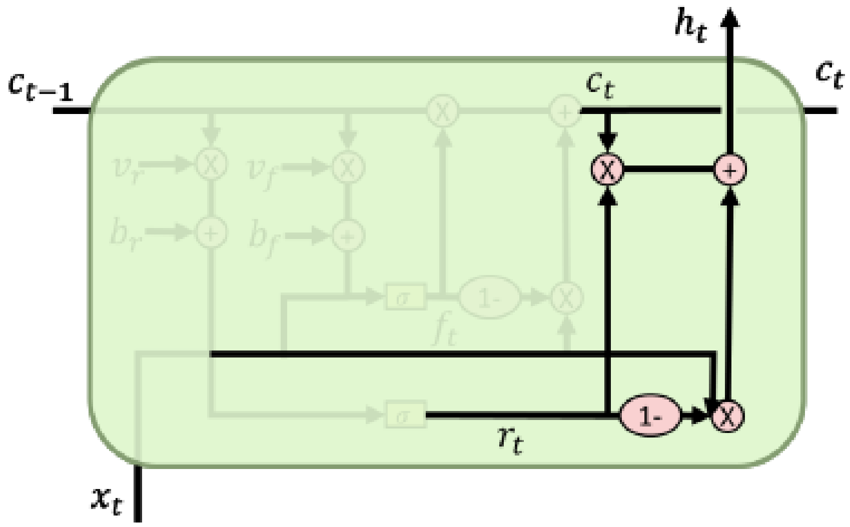
$$c_t = f_t \odot c_{t-1} + (1 - f_t) \odot (W x_t)$$

Weighted average according to the forgot gate

Light recurrence:

1. **Forget gate: Controls information flow**
2. **State vector: Adaptively average the previous state and the current c**

Simple Recurrent Unit(SRU)



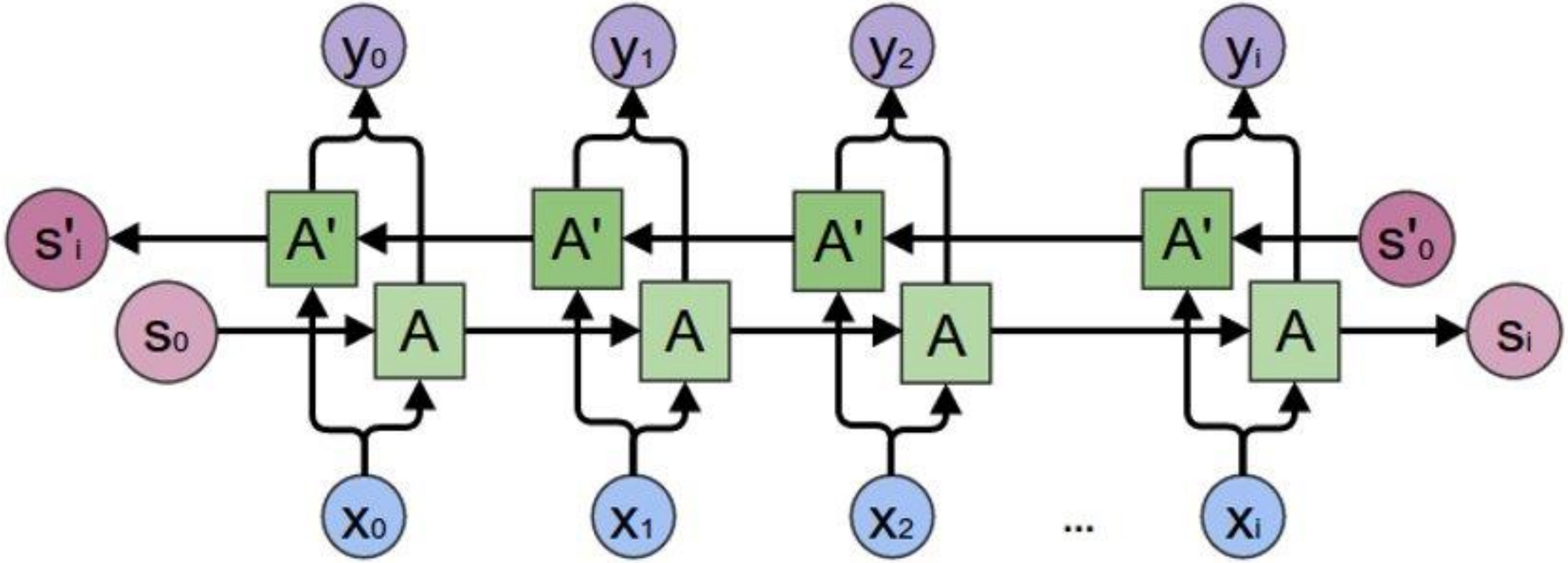
$$r_t = \sigma(W_r x_t + v_r \odot c_{t-1} + b_r)$$

$$h_t = r_t \odot c_t + (1 - r_t) \odot x_t$$

Highway network:

1. **Reset gate: how much current information to be passed.**
2. **Output vector: Adaptively combine input and the state vector**

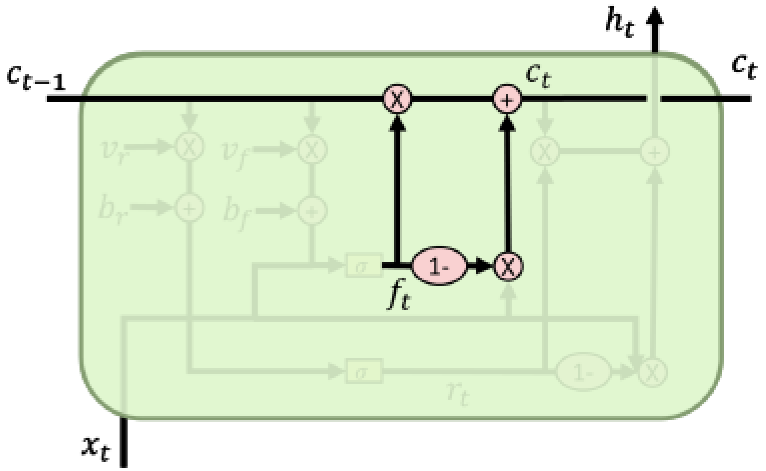
Variations – Bidirectional



Simple recurrent unite (SRU)

f — **Forget gated:** how much of the past information to be passed.

c — **State vector:** adaptively averaging the previous state and the current observation.



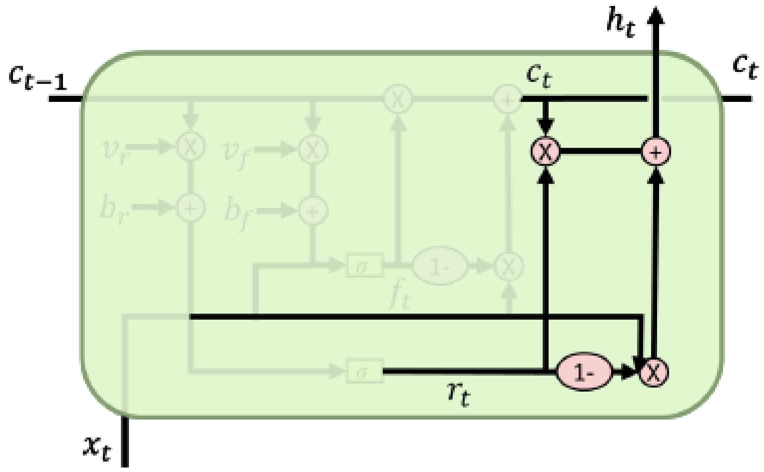
$$f_t = \sigma\left(W_f x_t + v_f \odot c_{t-1} + b_f\right)$$

$$c_t = f_t \odot c_{t-1} + (1 - f_t) \odot (W x_t)$$

Weighted average according to the forgot gate

Simple recurrent unite (SRU)

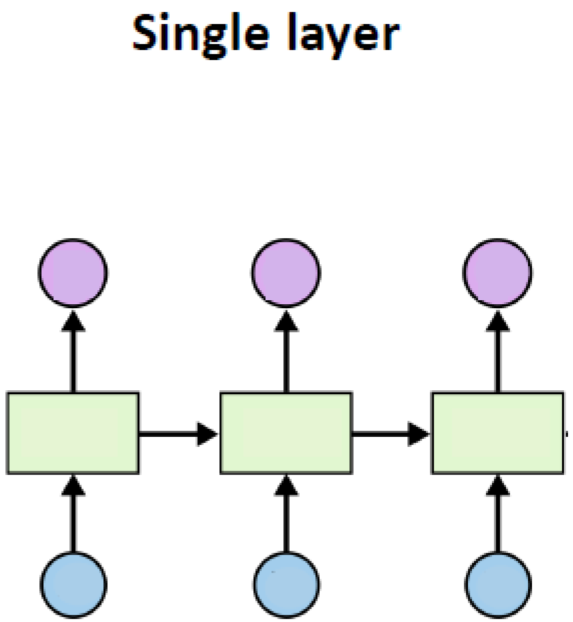
f	**Forget gated:** how much of the past information to be passed.
c	**State vector:** adaptively averaging the previous state and the current observation.
r	**Reset gated:** how much of the current information to be passed.
h	**Output vector:** adaptively combine the input and the state vector produced from the light recurrence.

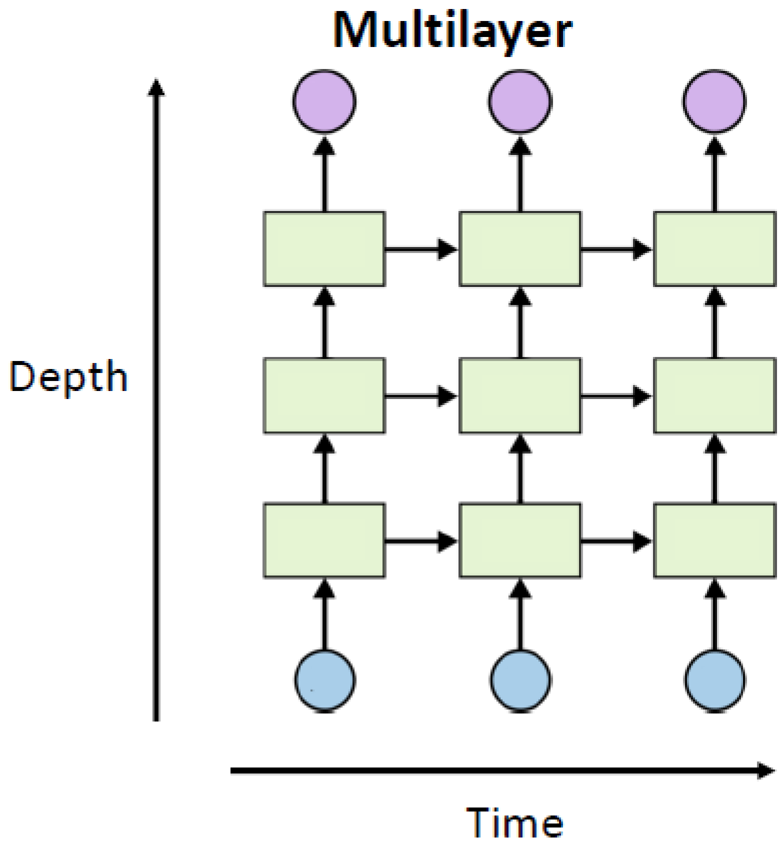


$$r_t = \sigma(W_r x_t + v_r \odot c_{t-1} + b_r)$$

$$h_t = r_t \odot c_t + (1 - r_t) \odot x_t$$

Variations – Multilayer

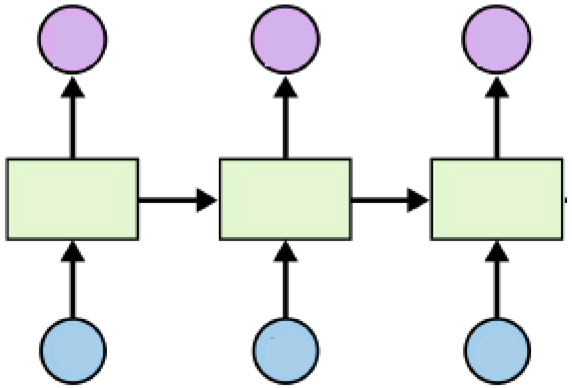




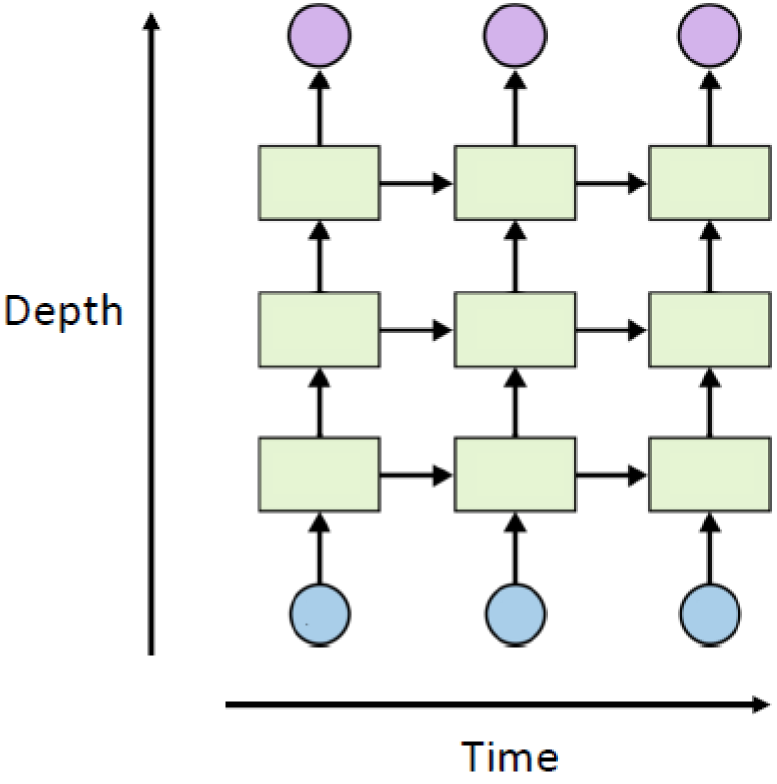
$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

Variations – LSTM Multilayer

Single layer



Multilayer



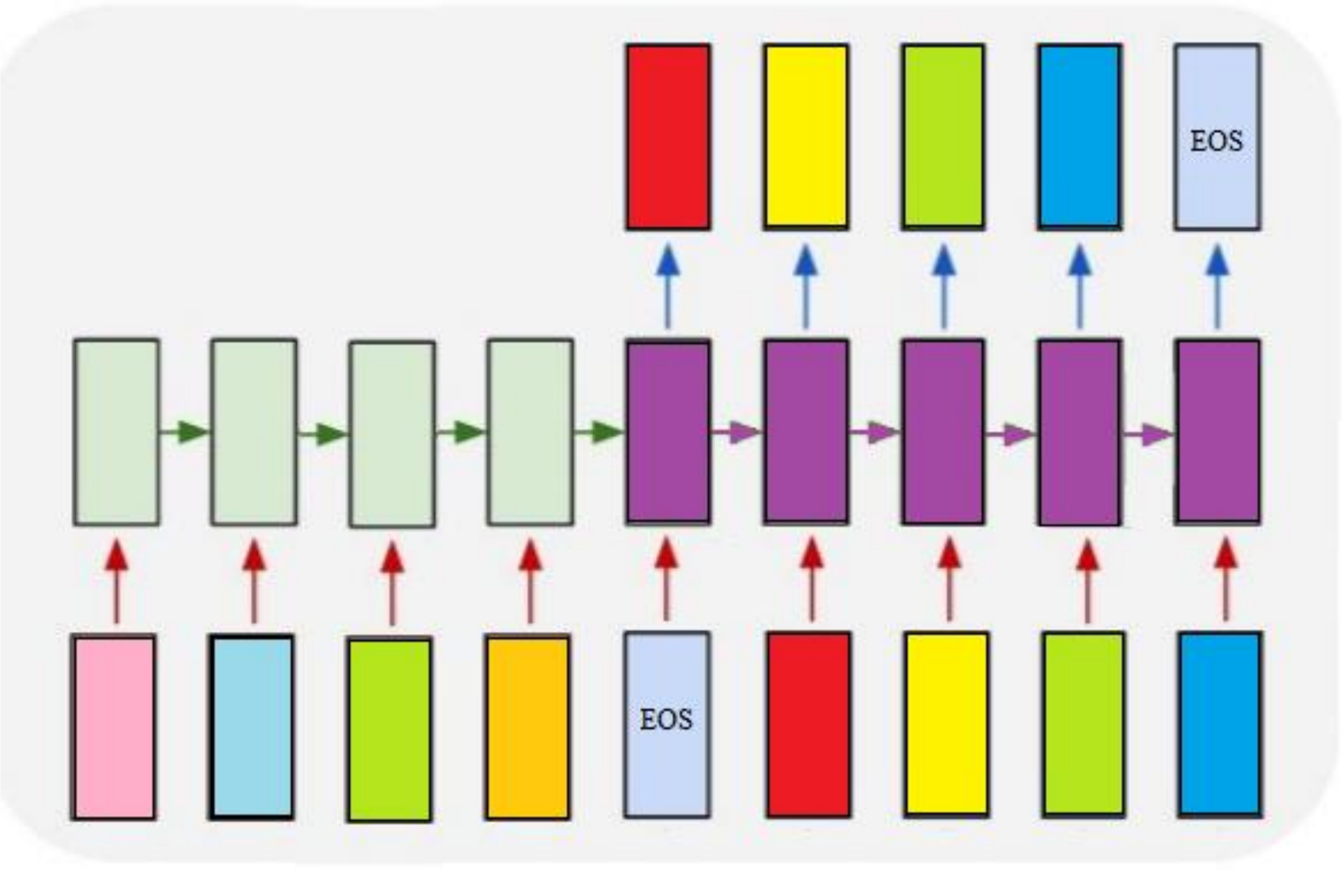
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

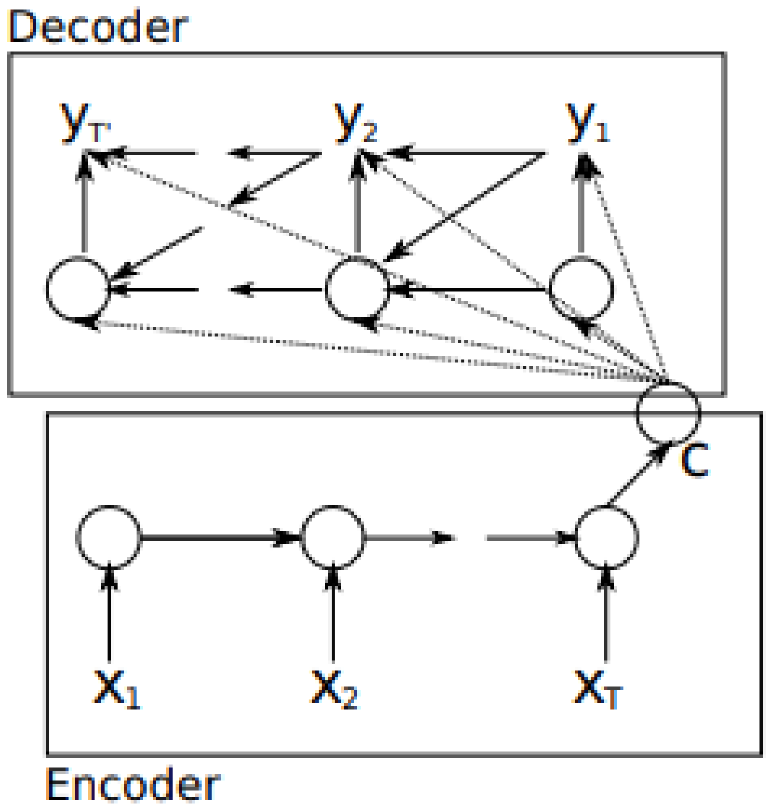
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

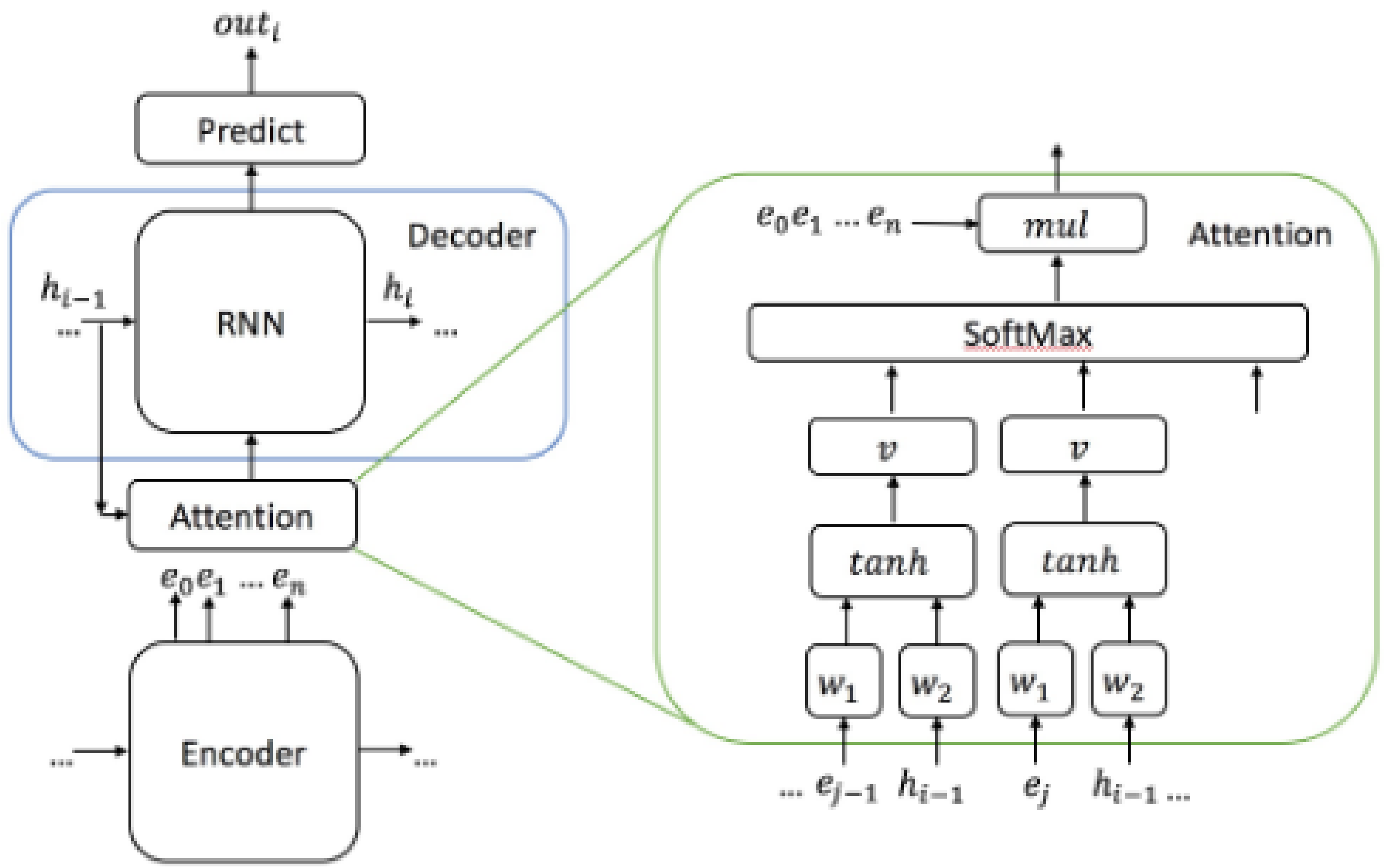
Variations-

Encoder-Decoder





Variations – Attention



Papers

Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation

Kyunghyun Cho

Dzmitry Bahdanau

Fethi Bougares

Yoshua Bengio

Holger Schwenk

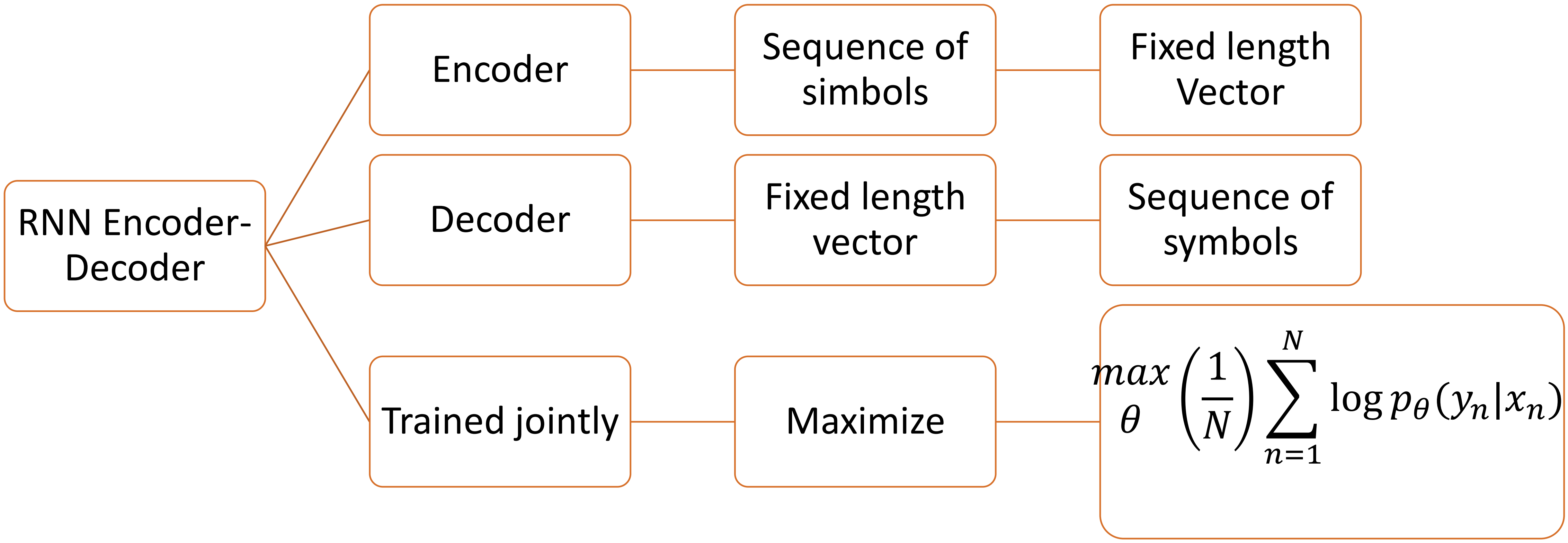
Introduction

- NN used in natural language processing (NLP)
- Encoder – Decoder RNN for statistical Machine Translation (SMT)
- Sophisticated hidden unit
- Evaluation Context: translating English to French

Main Contributions

1. Encoder-Decoder structure using two RNN
2. GRU
3. Continuous Space Representation

Proposed Methods



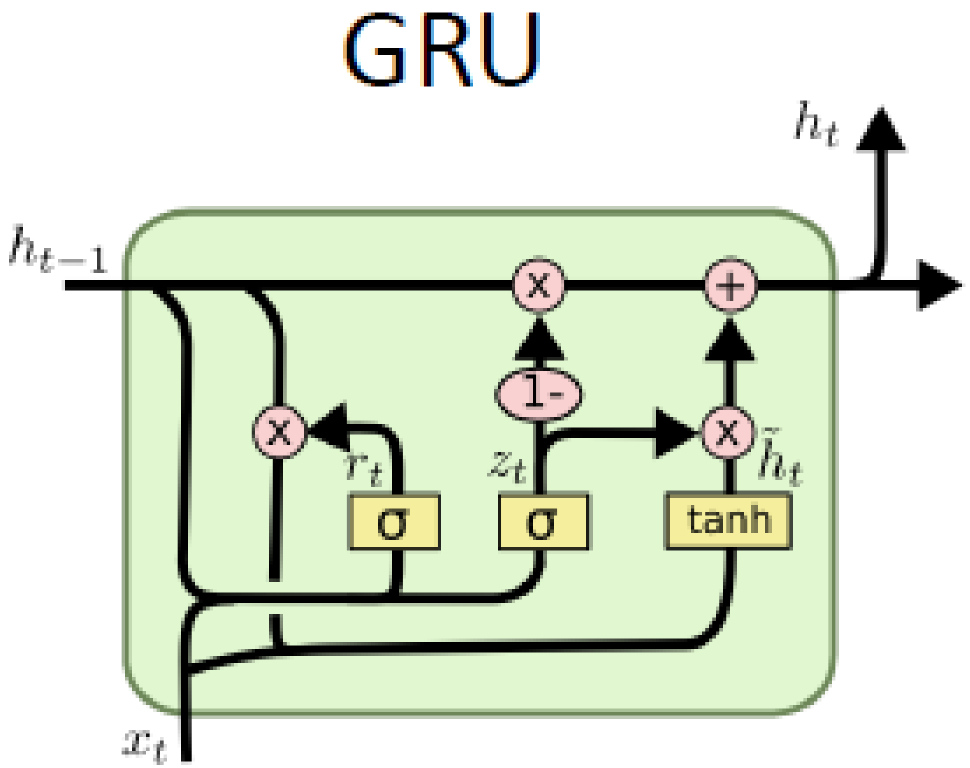
Proposed Methods

- Hidden Unit
  - Reset gate: $r_j = \theta([W_r x]_j + [U_r h_{t-1}]_j$
  - Update gate: $z_j = \theta([W_z x]_j + [U_z h_{t-1}]_j$
  - Actual activation: $h_j^{<t>} = z_j h_j^{<t-1>} + (1 - z_j)\tilde{h}_j^{<t>}$

$$\tilde{h}_j^{(t)} = \phi\left([\mathbf{W}\mathbf{x}]_j + \left[\mathbf{U}\left(\mathbf{r} \odot \mathbf{h}_{(t-1)}\right)\right]_j\right)$$

Proposed Methods

Hidden Unit



$]_j + [U_r h_{t-1}]_j$

$]_j + [U_z h_{t-1}]_j$

$> + (1 - z_j)\tilde{h}_j^{<t>}$

$$\tilde{h}_j^{\langle t \rangle} = \phi\left([\mathbf{W}\mathbf{x}]_j + \left[\mathbf{U}\left(\mathbf{r} \odot \mathbf{h}_{\langle t-1 \rangle}\right)\right]_j\right)$$

Proposed Methods

- STM
  - Goal
    - Optimize weights in 1
    - Maximize BLEU
  - Train
    - Table of phrase pairs
    - Ignore frecuencies

BLEU = Bilingual Evaluation Understudy

1. $$\log p(\mathbf{f} \mid \mathbf{e}) = \sum_{n=1}^{N} w_n f_n(\mathbf{f}, \mathbf{e}) + \log Z(\mathbf{e}),$$

Dataset

ACL 2014 NINTH WORKSHOP
ON STATISTICAL MACHINE TRANSLATION

Shared Task: Machine Translation

26-27 June 2014
Baltimore, USA

[HOME] | [TRANSLATION TASK] | [METRICS TASK] | [QUALITY ESTIMATION TASK] | [MEDICAL TRANSLATION TASK] | [SCHEDULE] | [PAPERS] | [AUTHORS] | [RESULTS]

The recurring translation task of the WMT workshops focuses mainly on European language pairs, but this year we have introduced English-Hindi as an experimental, low resource language pair. Translation quality will be evaluated on a shared, unseen test set of news stories. We provide a parallel corpus as training data, a baseline system, and additional resources for download. Participants may augment the baseline system or use their own system.

WMT’ 14 workshop

- Europarl
- News commentary
- UN
- Two crawled corpora

- Train set
  - Most frequent 15000 words

- Test set
  - Data selection (newstest2012 and 2013)
    - Weight tuning with MERT
  - newstest2014

Time function representations of data

TABLE 1. Set of time functions considered in this work.

#	Feature
1	x-coordinate: x_n
2	y-coordinate: y_n
3	Pen-pressure: z_n
4	Path-tangent angle: θ_n
5	Path velocity magnitude: v_n
6	Log curvature radius: ρ_n
7	Total acceleration magnitude: a_n
8-14	First-order derivate of features 1-7: $\dot{x_n}, \dot{y_n}, \dot{z_n}, \dot{\theta_n}, \dot{v_n}, \dot{\rho_n}, \dot{a_n}$
15-16	Second-order derivate of features 1-2: $\ddot{x_n}, \ddot{y_n}$
17	Ratio of the minimum over the maximum speed over a 5-samples window: v_n^r
18-19	Angle of consecutive samples and first order difference: α_n, $\dot{\alpha_n}$
20	Sine: s_n
21	Cosine: c_n
22	Stroke length to width ratio over a 5-samples window: r_n^5
23	Stroke length to width ratio over a 7-samples window: r_n^7

Quantitative Results

Models	BLEU dev	BLEU test
Baseline	30.64	33.30
RNN	31.20	33.87
CSLM + RNN	31.48	34.64
CSLM + RNN + WP	31.50	34.54

Table 1: BLEU scores computed on the development and test sets using different combinations of approaches. WP denotes a *word penalty*, where we penalizes the number of unknown words to neural networks.

Qualitative Results

Source	Translation Model	RNN Encoder–Decoder
at the end of the	[a la fin de la] [ŕ la fin des années] [être supprimés à la fin de la]	[à la fin du] [à la fin des] [à la fin de la]
for the first time	[г © pour la premirère fois] [été donnés pour la première fois] [été commémorée pour la première fois]	[pour la première fois] [pour la première fois ,] [pour la première fois que]
in the United States and	[? aux ?tats-Unis et] [été ouvertes aux États-Unis et] [été constatées aux États-Unis et]	[aux Etats-Unis et] [des Etats-Unis et] [des États-Unis et]
, as well as	[?s , qu'] [?s , ainsi que] [?re aussi bien que]	[, ainsi qu'] [, ainsi que] [, ainsi que les]
one of the most	[?t ?l' un des plus] [?l' un des plus] [être retenue comme un de ses plus]	[l' un des] [le] [un des]

(a) Long, frequent source phrases

Source	Translation Model	RNN Encoder–Decoder
, Minister of Communications and Transport	[Secrétaire aux communications et aux transports :] [Secrétaire aux communications et aux transports]	[Secrétaire aux communications et aux transports] [Secrétaire aux communications et aux transports :]
did not comply with the	[vestimentaire , ne correspondaient pas à des] [susmentionnée n' était pas conforme aux] [présentées n' étaient pas conformes à la]	[n' ont pas respecté les] [n' était pas conforme aux] [n' ont pas respecté la]
parts of the world .	[© gions du monde .] [régions du monde considérées .] [région du monde considérée .]	[parties du monde .] [les parties du monde .] [des parties du monde .]
the past few days .	[le petit texte .] [cours des tout derniers jours .] [les tout derniers jours .]	[ces derniers jours .] [les derniers jours .] [cours des derniers jours .]
on Friday and Saturday	[vendredi et samedi à la] [vendredi et samedi à] [se déroulera vendredi et samedi ,]	[le vendredi et le samedi] [le vendredi et samedi] [vendredi et samedi]

(b) Long, rare source phrases

Table 2: The top scoring target phrases for a small set of source phrases according to the translation model (direct translation probability) and by the RNN Encoder–Decoder. Source phrases were randomly selected from phrases with 4 or more words. ? denotes an incomplete (partial) character. г is a Cyrillic letter *ghe*.

Qualitative Results

Source	Samples from RNN Encoder–Decoder
at the end of the	[à la fin de la] (×11)
for the first time	[pour la première fois] (×24) [pour la première fois que] (×2)
in the United States and	[aux États-Unis et] (×6) [dans les États-Unis et] (×4)
, as well as	[, ainsi que] [,] [ainsi que] [, ainsi qu'] [et UNK]
one of the most	[l' un des plus] (×9) [l' un des] (×5) [l' une des plus] (×2)

(a) Long, frequent source phrases

Source	Samples from RNN Encoder–Decoder
, Minister of Communications and Transport	[, ministre des communications et le transport] (×13)
did not comply with the	[n' tait pas conforme aux] [n' a pas respect l'] (×2) [n' a pas respect la] (×3)
parts of the world .	[arts du monde .] (×11) [des arts du monde .] (×7)
the past few days .	[quelques jours .] (×5) [les derniers jours .] (×5) [ces derniers jours .] (×2)
on Friday and Saturday	[vendredi et samedi] (×5) [le vendredi et samedi] (×7) [le vendredi et le samedi] (×4)

(b) Long, rare source phrases

Table 3: Samples generated from the RNN Encoder–Decoder for each source phrase used in Table 2. We show the top-5 target phrases out of 50 samples. They are sorted by the RNN Encoder–Decoder scores.

Word and Phrase Representation

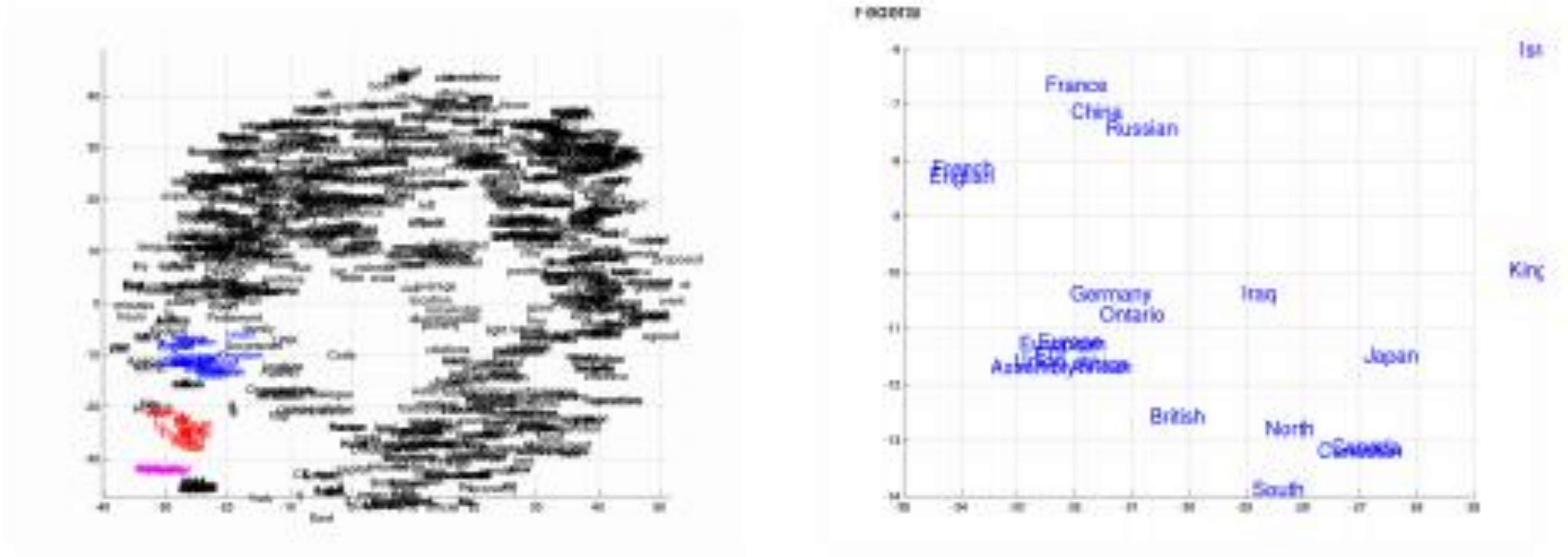


Figure 4: 2–D embedding of the learned word representation. The left one shows the full embedding space, while the right one shows a zoomed-in view of one region (color–coded). For more plots, see the supplementary material.

Word and Phrase Representation

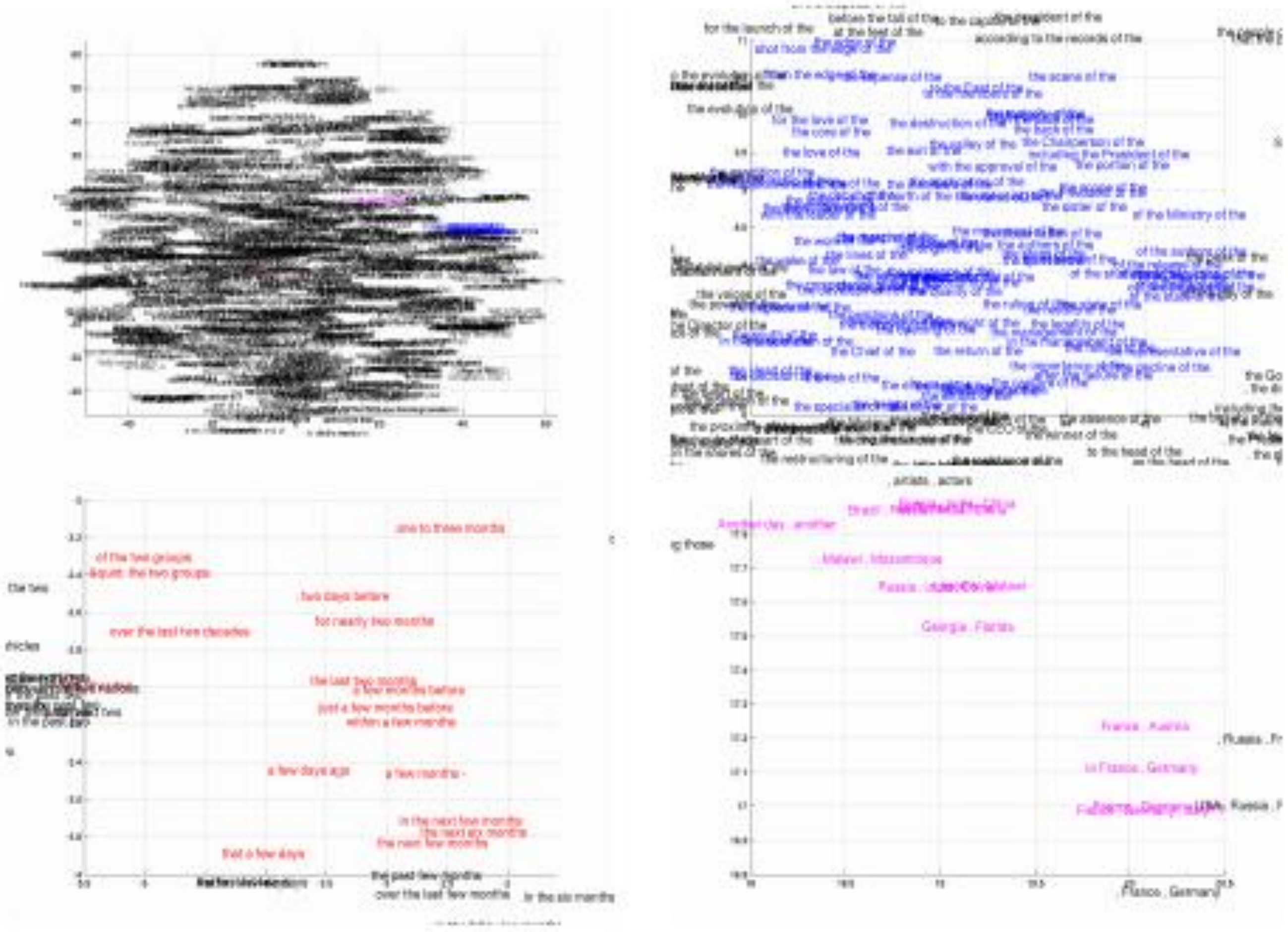


Figure 5: 2–D embedding of the learned phrase representation. The top left one shows the full representation space (5000 randomly selected points), while the other three figures show the zoomed-in view of specific regions (color–coded).

4 vs. 1 – Average pairs scores

TABLE 3. 4vs1 Evaluation Results: System performance in terms of EER(%) for the three different training scenarios considered, i.e., "skilled", "random" and "skilled + random".

	Train: "skilled"		Train: "random"		Train: "skilled + random"	
	Skilled	Random	Skilled	Random	Skilled	Random
LSTM	5.58	24.03	15.17	4.08	6.17	3.67
GRU	6.25	28.69	13.92	4.25	5.58	3.63
BLSTM	**4.75**	24.03	15.58	3.89	**5.50**	3.00
BGRU	4.92	19.69	12.33	**3.25**	5.92	**2.92**

TABLE 4. 1vs1 and 4vs1 DTW-based Evaluation Results: System performance in terms of EER(%).

	1vs1	4vs1
Skilled	10.17	7.75
Random	0.94	0.50

Detection Error Tradeoff curve

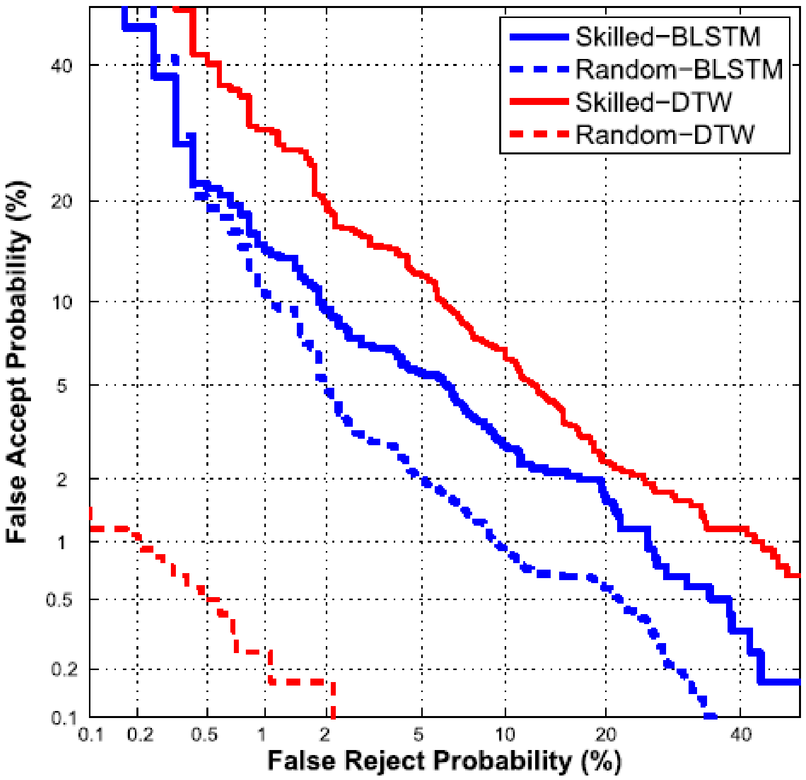


FIGURE 7. System performance results obtained using our Proposed BLSTM System for the 4vs1 case and "skilled + random" train scenario over the BiosecurID evaluation dataset.

To achieve a state-of-the-art performance of the model for both skilled and random forgeries, a possible solution is to perform two consecutive stages:

1. Stage based on DTW optimized for rejecting random forgeries.
2. Proposed RNNs Systems in order to reject the remaining skilled forgeries.

Conclusions

- RNN Encoder-Decoder is able to learn mapping from a sequence to another.
- Also, is able to score a pair of sequences or generate a target given a source sequence.
- The hidden unit is able to adaptively control how much it remembers or forget while reading or generating a sequence
- The model is able to capture linguistic regularities.
- The RNN Encoder–Decoder is able to propose well-formed target phrases
- The RNN Encoder–Decoder improves BLEU.
- Potential for improvement and analysis!

Exploring Recurrent Neural Networks for On-Line Handwritten Signature Biometrics

Ruben tolosana

Ruben vera-rodriguez

Julian fierrez

Javier ortega-garcia

Introduction

- Bidirectional LSTMs and GRUs RNNs caused great impact in handwriting recognition due to the relationship that exists between current inputs and past and future contexts.
- Off-line vs. On-line considerations
- Despite the good results obtained in the field of handwriting recognition, very few studies have successfully RNN architectures to handwritten signature verification.

Introduction

- Until now LSTM RNN systems trained with standard mechanisms are not appropriate for the task of signature verification as the amount of available data for this task is scarce.

Main Contributions

1. RNNs with a Siamese architecture
2. Writer-independent scenario
3. Strict experimental protocol
4. First analysis of RNNs for the two types of forgeries considered in on-line signature verification (i.e. skilled and random or zero-effort forgeries).
5. Bidirectional Scheme

Proposed Methods

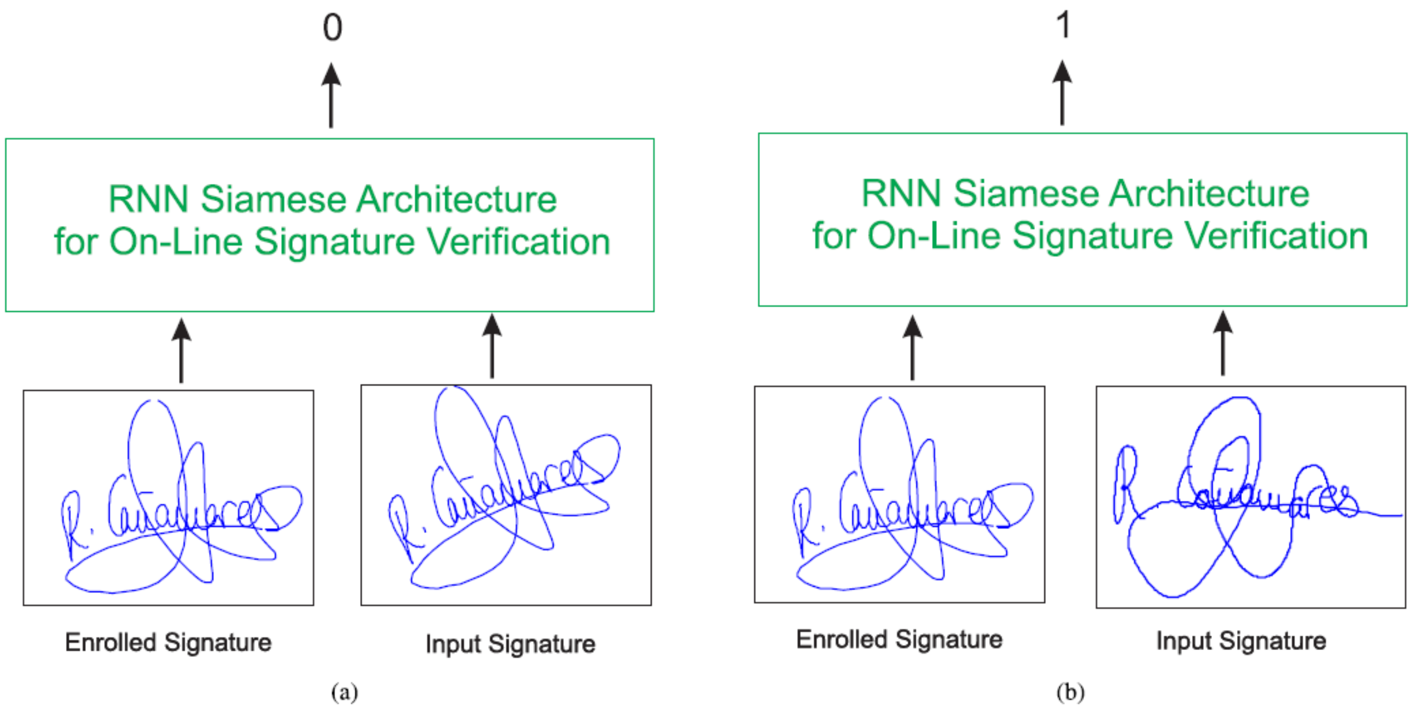


FIGURE 1. Examples of our proposed LSTM and GRU RNN systems based on a Siamese architecture for minimizing a discriminative cost function. (a) Genuine case. (b) Impostor case.

- Siamese Architecture
- LSTMs
- GRUs
- Bidirectional RNNs

Dataset

BiosecurID

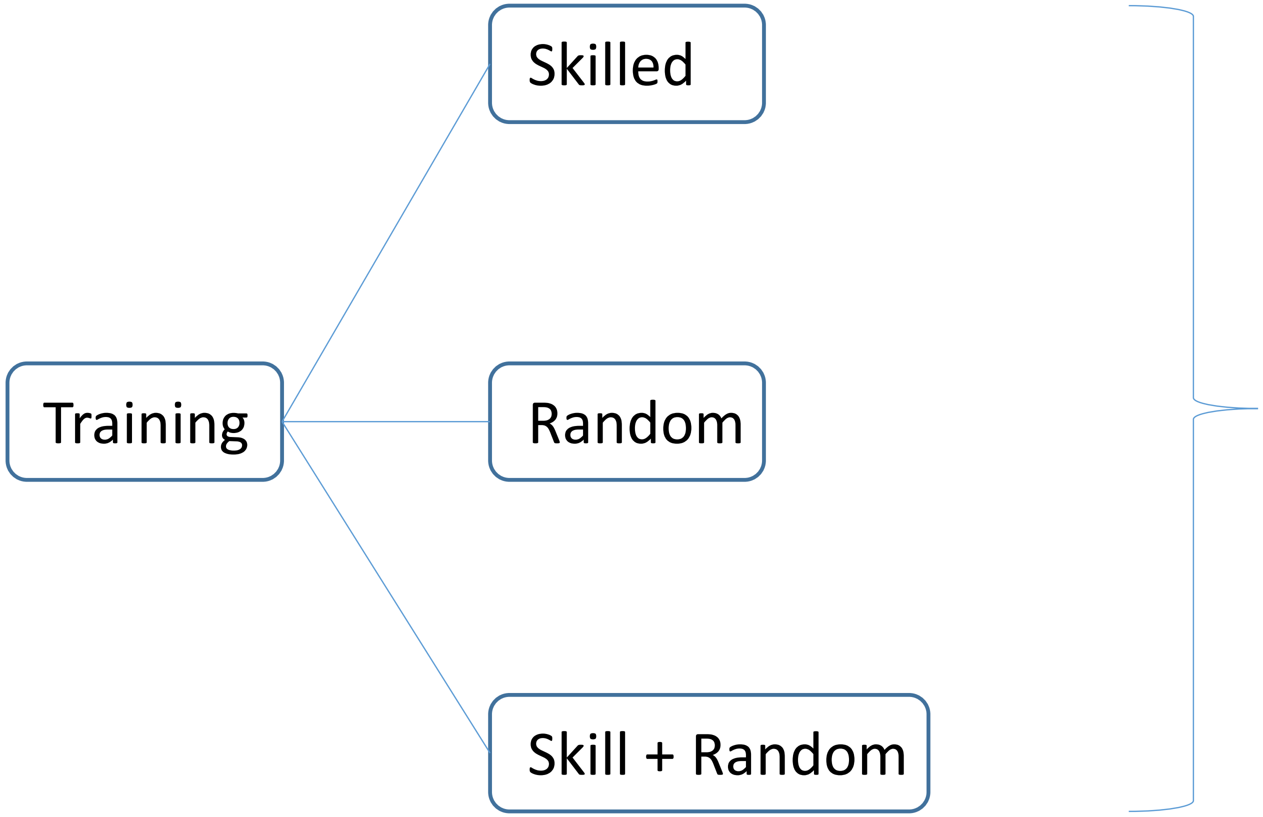
- 400 Users
  - 16 original signature
  - 12 skilled forgeries signatures
  - 4 acquisition sessions
- Data of each signature
  - X and Y coordinates: 0.25 mm resolution
  - Pressure: 1024 levels
  - Timestamp: 100Hz

Time function representations of data

TABLE 1. Set of time functions considered in this work.

#	Feature
1	x-coordinate: x_n
2	y-coordinate: y_n
3	Pen-pressure: z_n
4	Path-tangent angle: θ_n
5	Path velocity magnitude: v_n
6	Log curvature radius: ρ_n
7	Total acceleration magnitude: a_n
8-14	First-order derivate of features 1-7: $\dot{x_n}, \dot{y_n}, \dot{z_n}, \dot{\theta_n}, \dot{v_n}, \dot{\rho_n}, \dot{a_n}$
15-16	Second-order derivate of features 1-2: $\ddot{x_n}, \ddot{y_n}$
17	Ratio of the minimum over the maximum speed over a 5-samples window: v_n^r
18-19	Angle of consecutive samples and first order difference: α_n, $\dot{\alpha_n}$
20	Sine: s_n
21	Cosine: c_n
22	Stroke length to width ratio over a 5-samples window: r_n^5
23	Stroke length to width ratio over a 7-samples window: r_n^7

Experimental Protocol



1. LSTM
2. BLSTM
3. GRU
4. BGRU

Final topology

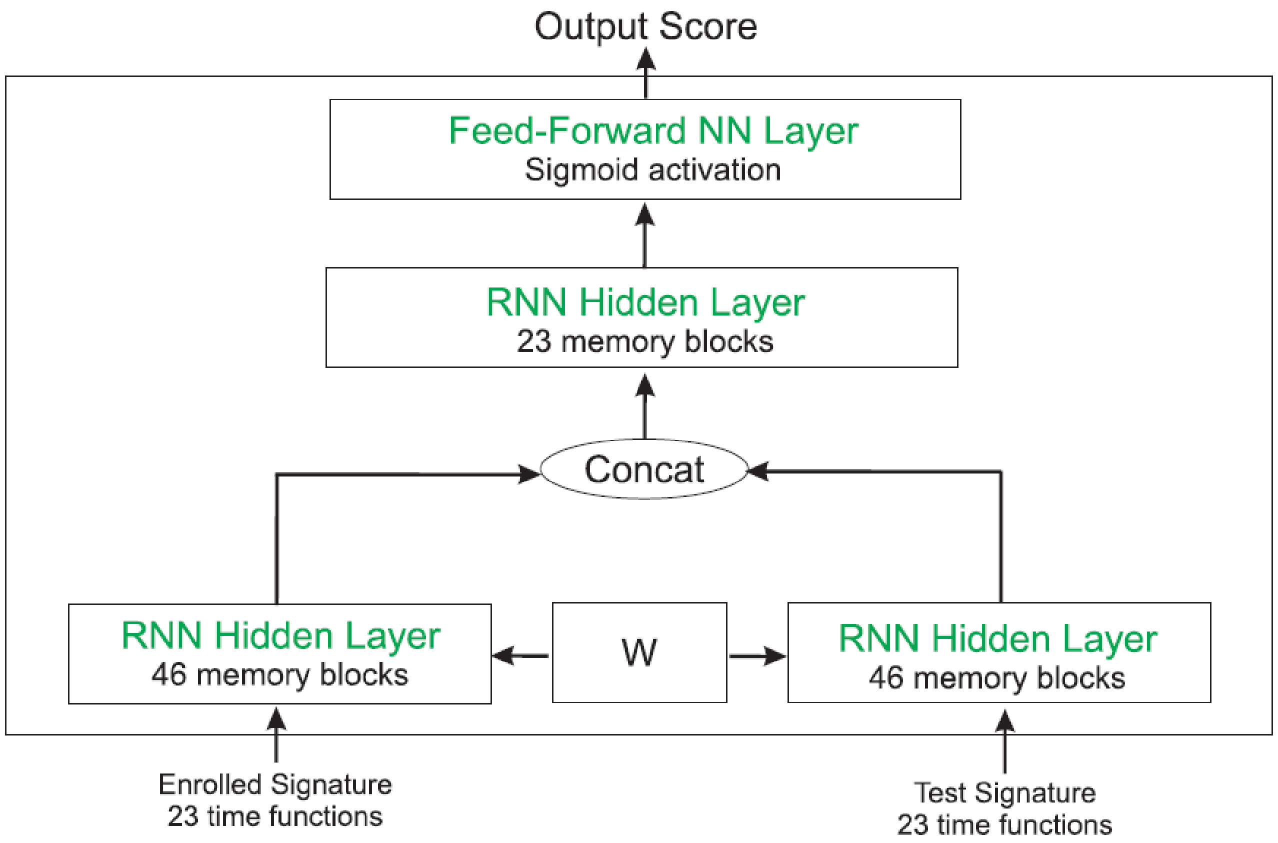


FIGURE 5. End-to-end on-line signature verification system proposed in this work and based on the use of LSTM and GRU RNNs with a Siamese architecture.

Training Cost

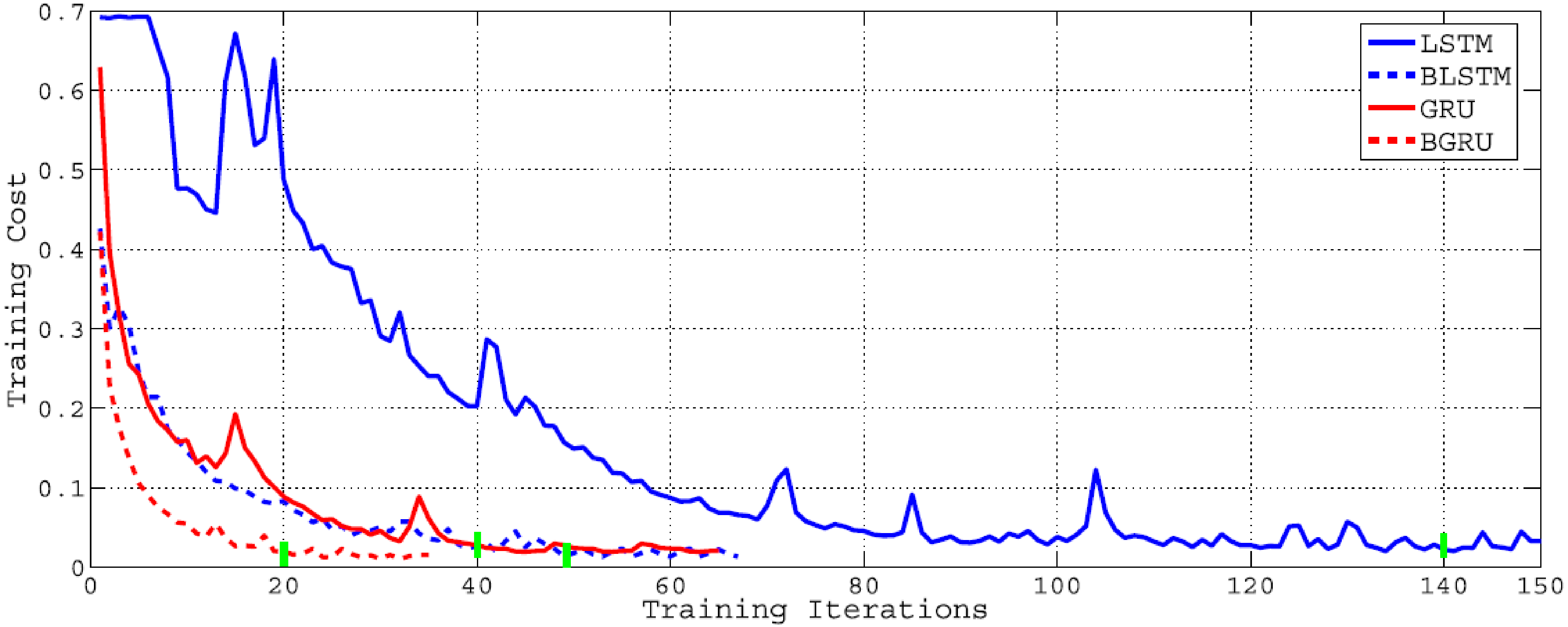


FIGURE 6. Considered RNNs cost during training for the "skilled" scenario. A small green vertical line indicates for each proposed RNN system the training iteration which provides the best system performance over the evaluation dataset.

1 vs. 1 – All pair scores

TABLE 2. 1vs1 Evaluation Results: System performance in terms of EER(%) for the three different training scenarios considered, i.e., "skilled", "random" and "skilled + random".

	Train: "skilled"		Train: "random"		Train: "skilled + random"	
	Skilled	Random	Skilled	Random	Skilled	Random
LSTM	6.44	24.48	13.31	5.38	7.94	6.22
GRU	7.69	29.42	15.63	6.92	7.67	5.98
BLSTM	**5.60**	24.48	15.31	**5.28**	**6.83**	**5.38**
BGRU	6.31	19.14	12.56	5.33	7.88	5.52

TABLE 4. 1vs1 and 4vs1 DTW-based Evaluation Results: System performance in terms of EER(%).

	1vs1	4vs1
Skilled	10.17	7.75
Random	0.94	0.50

4 vs. 1 – Average pairs scores

TABLE 3. 4vs1 Evaluation Results: System performance in terms of EER(%) for the three different training scenarios considered, i.e., "skilled", "random" and "skilled + random".

	Train: "skilled"		Train: "random"		Train: "skilled + random"	
	Skilled	Random	Skilled	Random	Skilled	Random
LSTM	5.58	24.03	15.17	4.08	6.17	3.67
GRU	6.25	28.69	13.92	4.25	5.58	3.63
BLSTM	**4.75**	24.03	15.58	3.89	**5.50**	3.00
BGRU	4.92	19.69	12.33	**3.25**	5.92	**2.92**

TABLE 4. 1vs1 and 4vs1 DTW-based Evaluation Results: System performance in terms of EER(%).

	1vs1	4vs1
Skilled	10.17	7.75
Random	0.94	0.50

DET curve

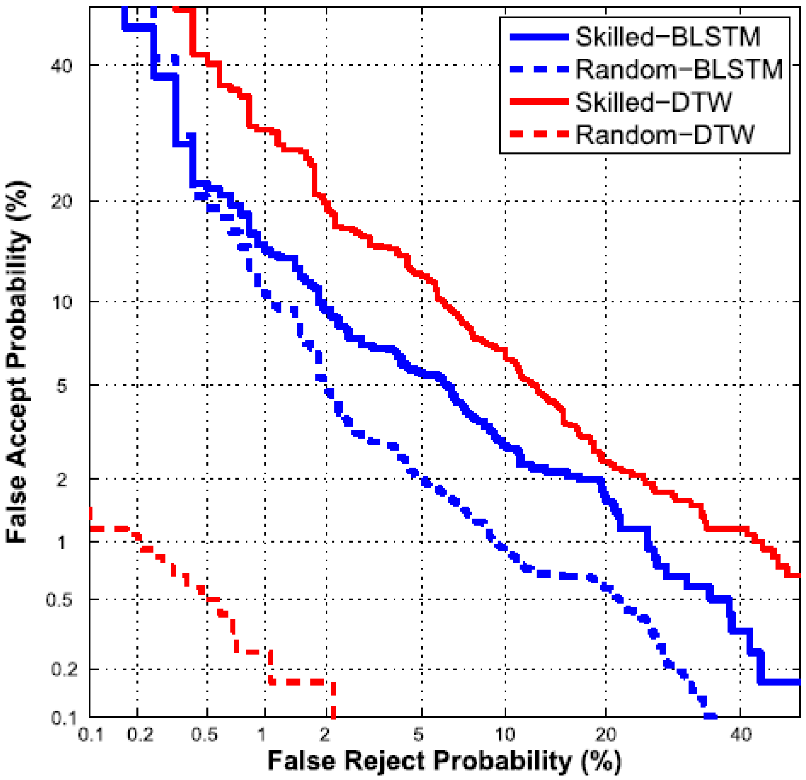


FIGURE 7. System performance results obtained using our Proposed BLSTM System for the 4vs1 case and "skilled + random" train scenario over the BiosecurID evaluation dataset.

To achieve a state-of-the-art performance of the model for both skilled and random forgeries, a possible solution is to perform two consecutive stages:

1. Stage based on DTW optimized for rejecting random forgeries.
2. Proposed RNNs Systems in order to reject the remaining skilled forgeries.

Conclusions

- The main contribution is to assess the feasibility of different RNNs systems in combination with a Siamese architecture for on-line handwritten signature verification.
- First complete and successful framework on the use of multiple RNN systems (i.e. LSTM and GRU) for on-line handwritten signature verication considering both skilled and random types of forgeries.
- Difference in the number of training iterations needed between normal and bidirectional schemes.
- Difference in the number of training iterations between both LSTM and GRU RNNs.
- High ability of our proposed approach for learning even with small amounts of signatures.

Tutorial

```python
from __future__ import unicode_literals, print_function, division
from io import open
import unicodedata
import string
import re
import random

import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
SOS_token = 0
EOS_token = 1


class Lang:
    def __init__(self, name):
        self.name = name
        self.word2index = {}
        self.word2count = {}
        self.index2word = {0: "SOS", 1: "EOS"}
        self.n_words = 2  # Count SOS and EOS

    def addSentence(self, sentence):
        for word in sentence.split(' '):
            self.addWord(word)

    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.word2count[word] = 1
            self.index2word[self.n_words] = word
            self.n_words += 1
        else:
            self.word2count[word] += 1
```

1. We will be representing each word in a language as a one-hot vector.
2. We will however use a few thousand words per language.
3. We'll need a unique index per word to use as the inputs and targets of the networks later.
4. To keep track of all this we will use a helper class called Lang

```python
# Turn a Unicode string to plain ASCII, thanks to
# https://stackoverflow.com/a/518232/2809427
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

# Lowercase, trim, and remove non-letter characters


def normalizeString(s):
    s = unicodeToAscii(s.lower().strip())
    s = re.sub(r"([.!?])", r" \1", s)
    s = re.sub(r"[^a-zA-Z.!?]+", r" ", s)
    return s
```

1. Convert Unicode to ASCII.
2. Make all words Lowercase, trim non-letter characters.

```python
def readLangs(lang1, lang2, reverse=False):
    print("Reading lines...")

    # Read the file and split into lines
    lines = open('data/%s-%s.txt' % (lang1, lang2), encoding='utf-8').\
        read().strip().split('\n')

    # Split every line into pairs and normalize
    pairs = [[normalizeString(s) for s in l.split('\t')] for l in lines]

    # Reverse pairs, make Lang instances
    if reverse:
        pairs = [list(reversed(p)) for p in pairs]
        input_lang = Lang(lang2)
        output_lang = Lang(lang1)
    else:
        input_lang = Lang(lang1)
        output_lang = Lang(lang2)

    return input_lang, output_lang, pairs
```

1. Read the file.
2. Split it in pairs [ENG-FRE].

```python
MAX_LENGTH = 10

eng_prefixes = (
    "i am ", "i m ",
    "he is", "he s ",
    "she is", "she s ",
    "you are", "you re ",
    "we are", "we re ",
    "they are", "they re "
)


def filterPair(p):
    return len(p[0].split(' ')) < MAX_LENGTH and \
        len(p[1].split(' ')) < MAX_LENGTH and \
        p[1].startswith(eng_prefixes)


def filterPairs(pairs):
    return [pair for pair in pairs if filterPair(pair)]
```

1. Filter phrases by size.
2. Filter phrases by prefixes.

```python
def prepareData(lang1, lang2, reverse=False):
    input_lang, output_lang, pairs = readLangs(lang1, lang2, reverse)
    print("Read %s sentence pairs" % len(pairs))
    pairs = filterPairs(pairs)
    print("Trimmed to %s sentence pairs" % len(pairs))
    print("Counting words...")
    for pair in pairs:
        input_lang.addSentence(pair[0])
        output_lang.addSentence(pair[1])
    print("Counted words:")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)
    return input_lang, output_lang, pairs


input_lang, output_lang, pairs = prepareData('eng', 'fra', True)
print(random.choice(pairs))
```

1. Prepare all the data using the previous functions.

```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

1. Definition of the Encoder.

- From a phrase to a vector representing the meaning of it.

```python
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

1. Definition of the Decoder.

- From Hidden State of the Encoder to the translation.

```python
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

        attn_weights = F.softmax(
            self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                                 encoder_outputs.unsqueeze(0))

        output = torch.cat((embedded[0], attn_applied[0]), 1)
        output = self.attn_combine(output).unsqueeze(0)

        output = F.relu(output)
        output, hidden = self.gru(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

1. Definition of the an Attention Decoder.

- “Focus” on a different part of the encoder’s outputs for every step of the decoder’s own outputs.

```python
def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')]


def tensorFromSentence(lang, sentence):
    indexes = indexesFromSentence(lang, sentence)
    indexes.append(EOS_token)
    return torch.tensor(indexes, dtype=torch.long, device=device).view(-1, 1)


def tensorsFromPair(pair):
    input_tensor = tensorFromSentence(input_lang, pair[0])
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return (input_tensor, target_tensor)
```

1. Get the indices that represents a sentence.
2. Get a tensor that represent the sentence.
3. Get a tensors that represent the input phrase and another that represents the target phrase.

```python
teacher_forcing_ratio = 0.5


def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer,
criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # Teacher forcing

    else:
        # Without teacher forcing: use its own predictions as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()  # detach from history as input

            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break

    loss.backward()

    encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / target_length
```

1. Train for one pair of inputs.

```python
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [tensorsFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_tensor = training_pair[0]
        target_tensor = training_pair[1]

        loss = train(input_tensor, target_tensor, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss

        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            print('%s (%d %d%%) %.4f' % (timeSince(start, iter / n_iters),
                                         iter, iter / n_iters * 100, print_loss_avg))

        if iter % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_loss_total = 0

    showPlot(plot_losses)
```

1. Training the model.

```
hidden_size = 256
encoder1 = EncoderRNN(input_lang.n_words, hidden_size).to(device)
attn_decoder1 = AttnDecoderRNN(hidden_size, output_lang.n_words, dropout_p=0.1).to(device)

trainIters(encoder1, attn_decoder1, 75000, print_every=5000)
```

```
output_words, attentions = evaluate(
    encoder1, attn_decoder1, "je suis trop froid .")
```

1. Define the size of the hidden state.
2. Create the encoder.
3. Create the decoder.
4. Train a model.
5. Evaluate the model with a phrase

Homework

- Implement LSTM and SRU instead of GRU. Compare and discuss the results.
- Modify the architecture adding more layers. Show your results and discuss.
- Modify the RNN direction into a bidirectional network. Explain how this affect your initial results.