ISO 29990:2010

NATIONAL INFOCOMM COMPETENCY FRAMEWORK

NUS National University of Singapore

ISS INSTITUTE OF SYSTEMS SCIENCE

Text Processing using Machine Learning

Memory Network and Conditioned Generation

Liling Tan

04 Dec 2019

OVER 5,500 GRADUATE ALUMNI

OFFERING OVER 120 ENTERPRISE IT, INNOVATION & LEADERSHIP PROGRAMMES

TRAINING OVER 120,000 DIGITAL LEADERS & PROFESSIONALS

Overview

Lecture

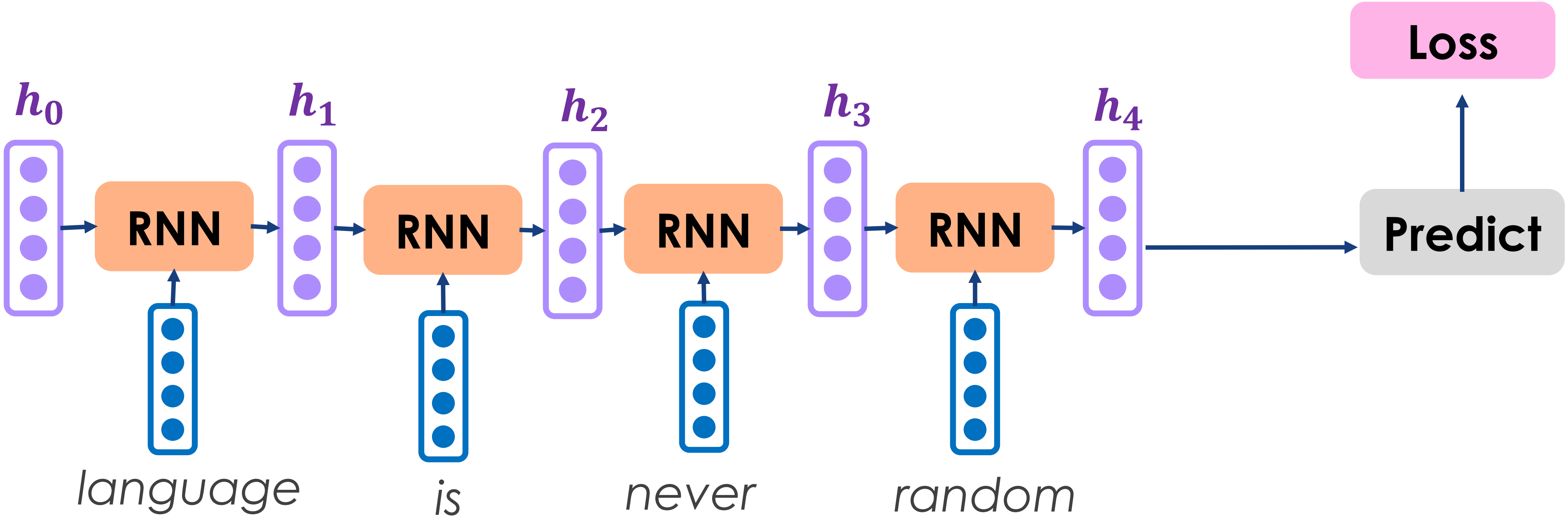
- Gated Memory RNNs
- Conditioned Generation

Hands-on

- Sequence Generation

Gated Memory RNNs

Recurrent Neural Nets



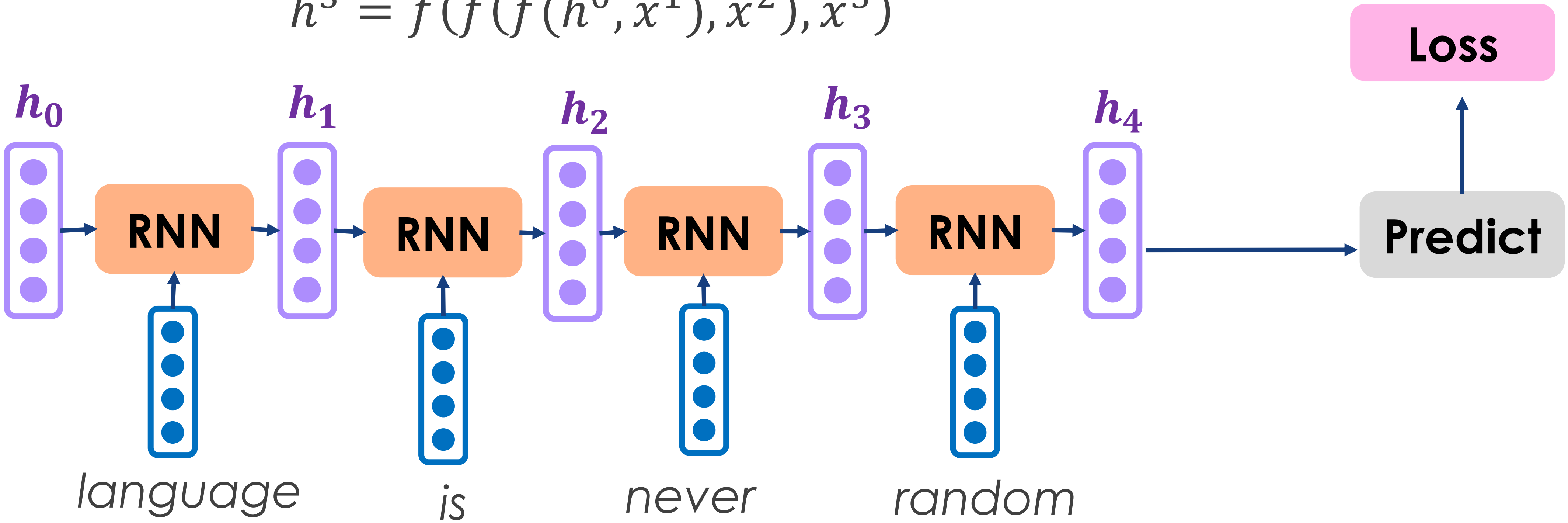
RNN Exploding gradient

Lets look at the **"curled up" RNN:**

$$h^t = f(h^{t-1}, x^t)$$

If we **unroll the RNN** that has 3 time-step, we get:

$$h^3 = f(f(f(h^0, x^1), x^2), x^3)$$



Exploding gradient

Lets look at the **“curled up” RNN:**

$$h^t = f(h^{t-1}, x^t)$$

If we **unroll the RNN** that has 3 time-step, we get:

$$h^3 = f(f(f(h^0, x^1), x^2), x^3)$$

And if consider $f(x)$ as a “harmless-looking”

$$f(x) = 3.5x(1-x)$$

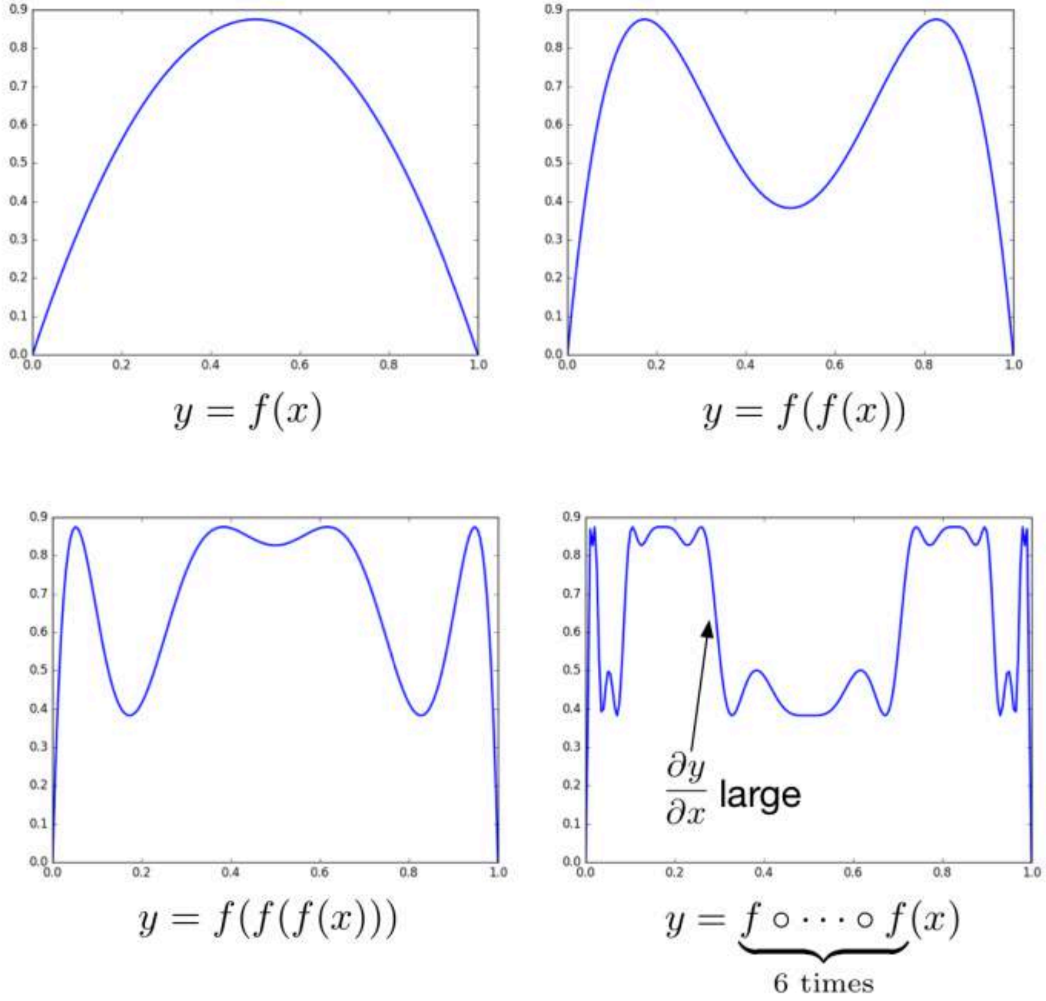


Figure and example from (Grosse 2017)

Solution: Gradient Clipping

- **General idea:** if we don't like the big numbers, just put a max to the gradient values

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$

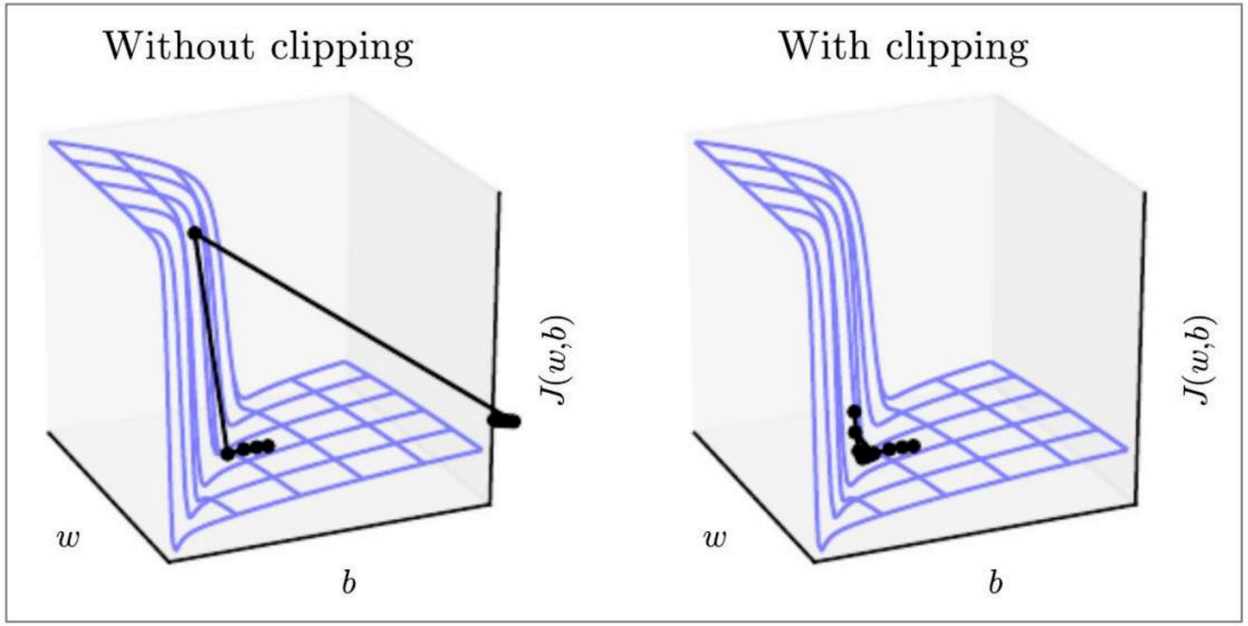
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**

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

end if

Source: (Pascanu et al, 2013)

Solution: Gradient Clipping



- Figure above shows a loss surface of an RNN

- **(Left) without clipping**, weights values increases and so does gradient causing loss to "jump off the clip"

- **(Right) with clipping**, weights are kept to a max and loss doesn't inflate

Source: (Goodfellow et al. 2016)

Partial Gradients

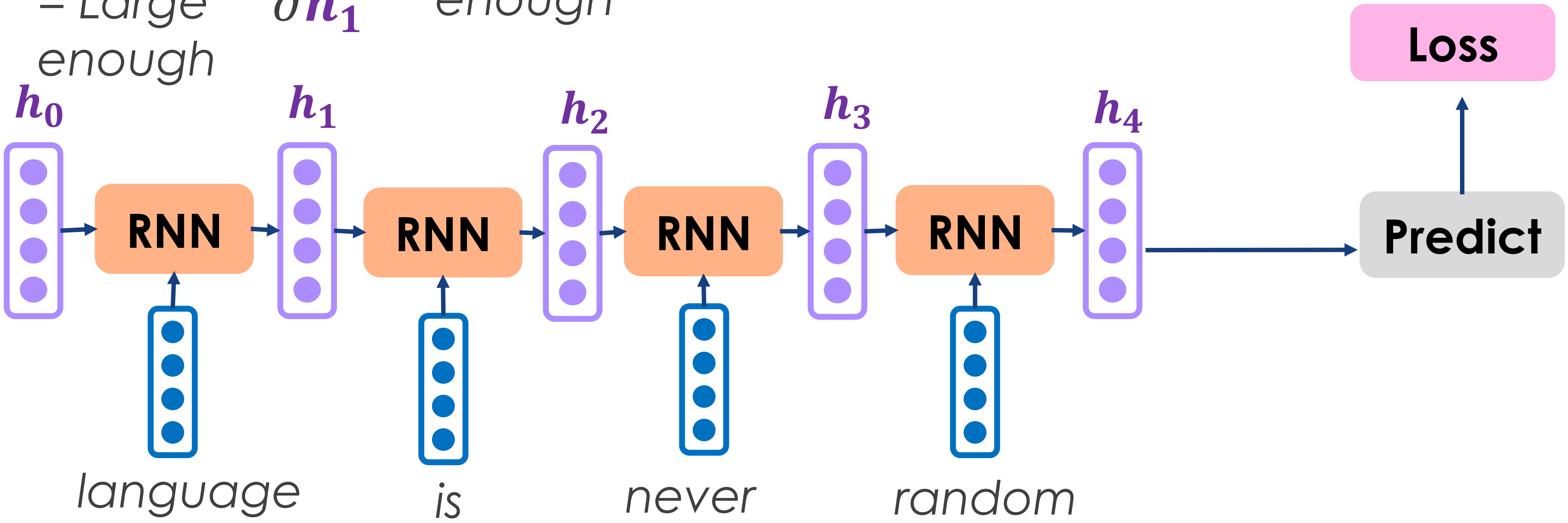
$$\frac{\partial(loss)}{\partial h_4}$$ = Large enough

$$\frac{\partial(h_4)}{\partial h_3}$$ = Large enough

$$\frac{\partial(h_3)}{\partial h_2}$$ = Large enough

$$\frac{\partial(h_2)}{\partial h_1}$$ = Large enough

$$\frac{\partial(h_1)}{\partial h_0}$$ = Large enough



RNN Vanishing Gradients

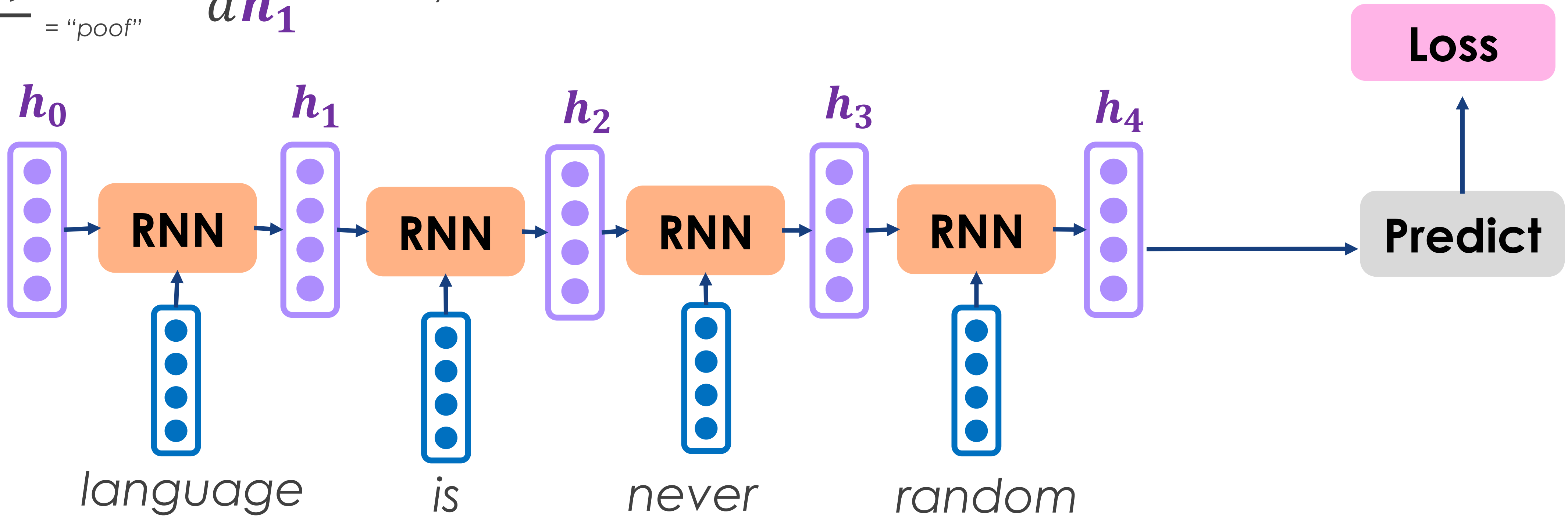
$$\frac{d(loss)}{d\boldsymbol{h}_4} = \text{Large enough}$$

$$\frac{d(loss)}{d\boldsymbol{h}_3} = \text{Okay}$$

$$\frac{d(loss)}{d\boldsymbol{h}_2} = \text{small}$$

$$\frac{d(loss)}{d\boldsymbol{h}_1} = \text{tiny}$$

$$\frac{d(loss)}{d\boldsymbol{h}_0} = \text{“poof”}$$



Why is vanishing gradient a problem?

- When moving along the timestep of RNN, gradient can be thought as "***the effects of the past on the future***" **(See, 2019)**

Why is vanishing gradient a problem?

- When moving along the timestep of RNN, gradient can be thought as "***the effects of the past on the future***" **(See, 2019)**

- Vanishing gradient means info from the ***previous words are less influential to predict words in the future***

"**Trump** haz **cheezburger**"

"**Trump** haz the **kat** tat haz **cheezburger**"

"**Trump** haz the kat buy tat **cheezburger**"

Why is vanishing gradient a problem?

- When moving along the timestep of RNN, gradient can be thought as "***the effects of the past on the future***" ***(See, 2019)***
- Vanishing gradient means info from the ***previous words are less influential to predict words in the future***
- When gradients are small over longer distance, it's hard to know whether
  - **Trump** has no relation to the **cheezburger** or
  - the model learns to **wrong parameter** that doesn't capture relation between **Trump** and **cheezburger**

Solution: Information Controlling RNN

- **General idea:** make additive connections between time steps

- ***Keeping a memory of past time steps and add them to the current one***

- ***Addition don't change gradient***, no vanishing

- ***Gates control different information strengths*** from the past

Long Short Term Memory (LSTM)

- At each time step, perform the following operations

Input: controls how much new cell info is written to cell

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$

Forget: controls how much previous cell info should be forgotten

$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$

Output: controls how much info is written to hidden state

$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$

New cell info: new content to write to cell

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

Cell state (Memory): forget some of the previous cell info and write some new cell info

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Hidden state: output some bits of info from the cell state for next cell to "remember"

$$h_t = o_t \circ \tanh(c_t)$$

Long Short Term Memory (LSTM)

- At each time step, perform the following operations

Sigmoid to make sure all outputs are (0,1)

Elementwise product

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$

$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$

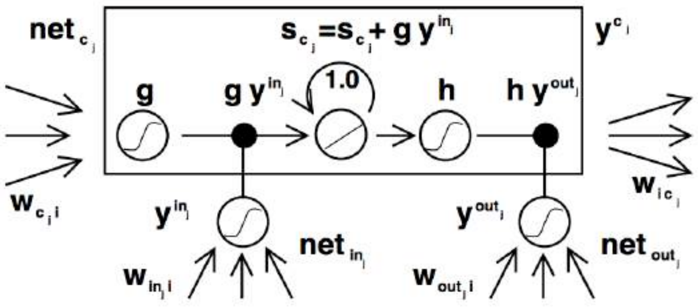
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

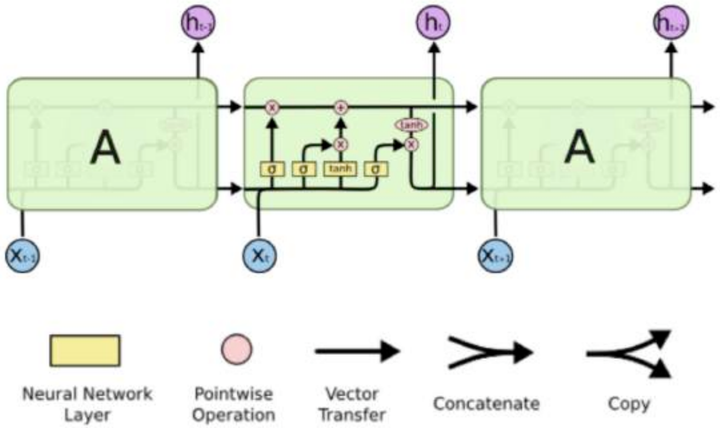
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tanh(c_t)$$

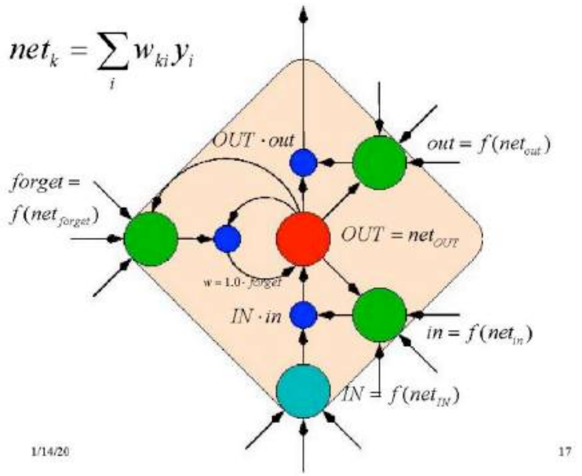
LSTM Galore



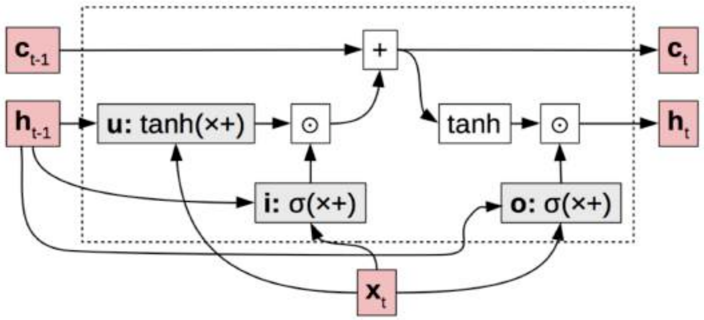
From Hochreiter and Schmidhuber (1997)



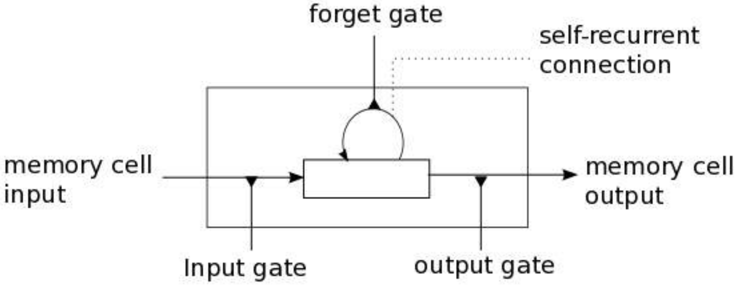
From Olah (2015) blogpost



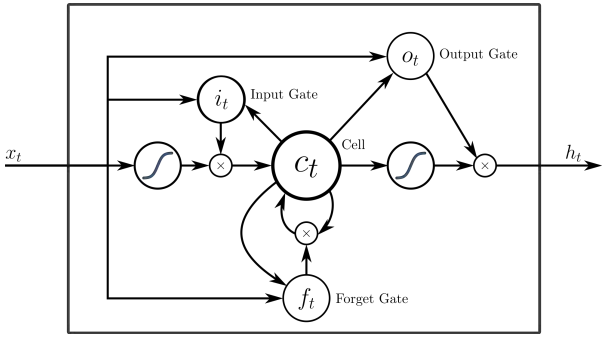
From Schmidhuber (2017) page



From Neubig (2019) CMU NN4NLP Course

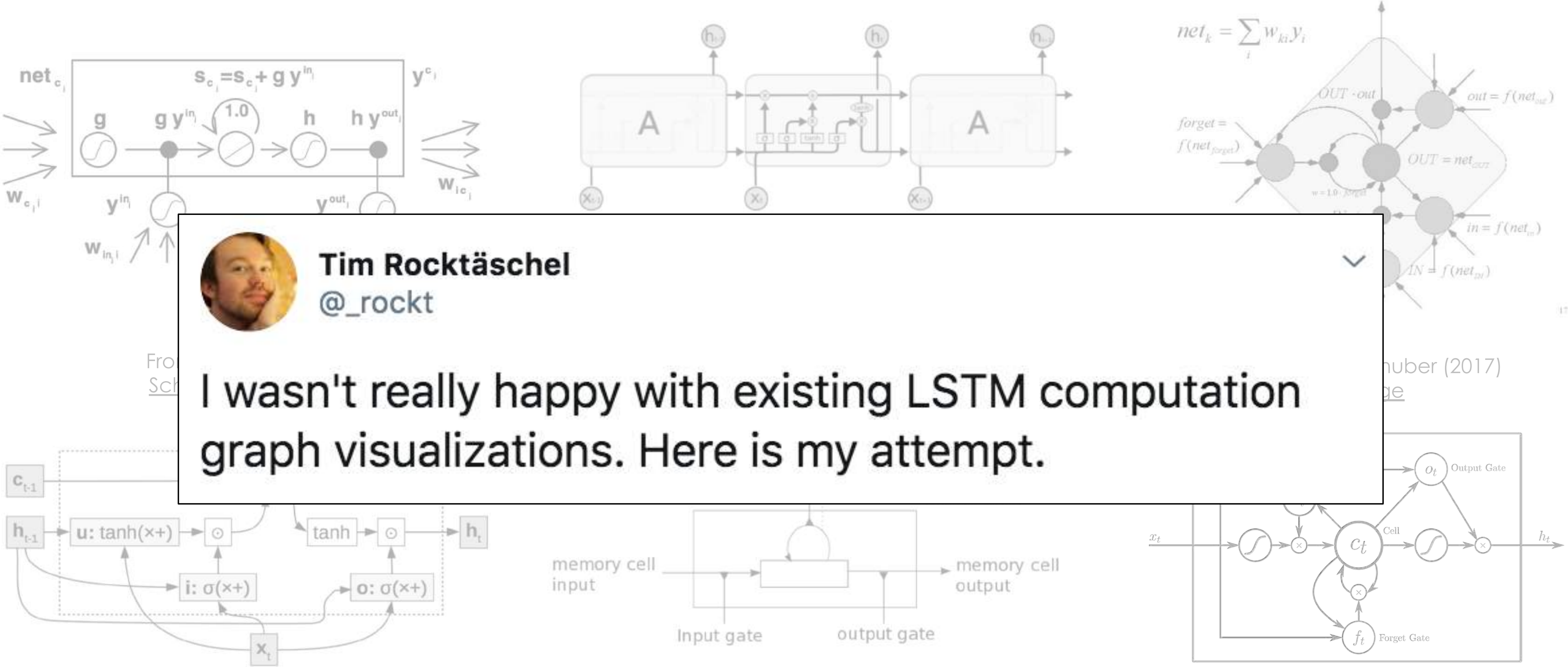


From http://deeplearning.net



From Wikipedia

LSTM Galore



From Neubig (2019) CMU NN4NLP Course

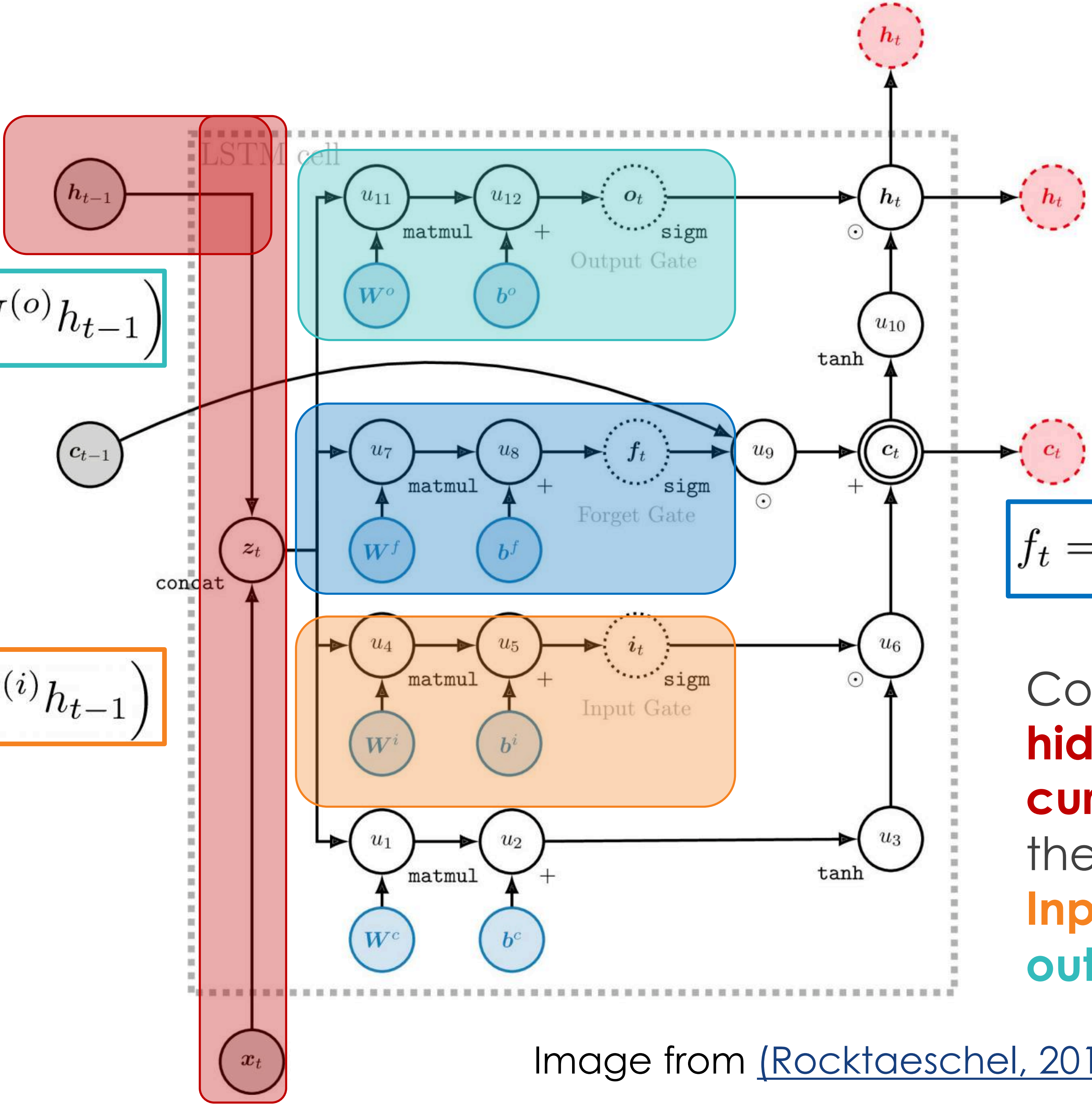
From http://deeplearning.net

From Wikipedia

LSTM

$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$



$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$

Concat the previous **hidden state** and **current input** and put them through the **Input**, **forget** and **output** gates

Image from [(Rocktaeschel, 2017)]

LSTM

Compute the **non-linearity** like a vanilla RNN **for this time step**

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

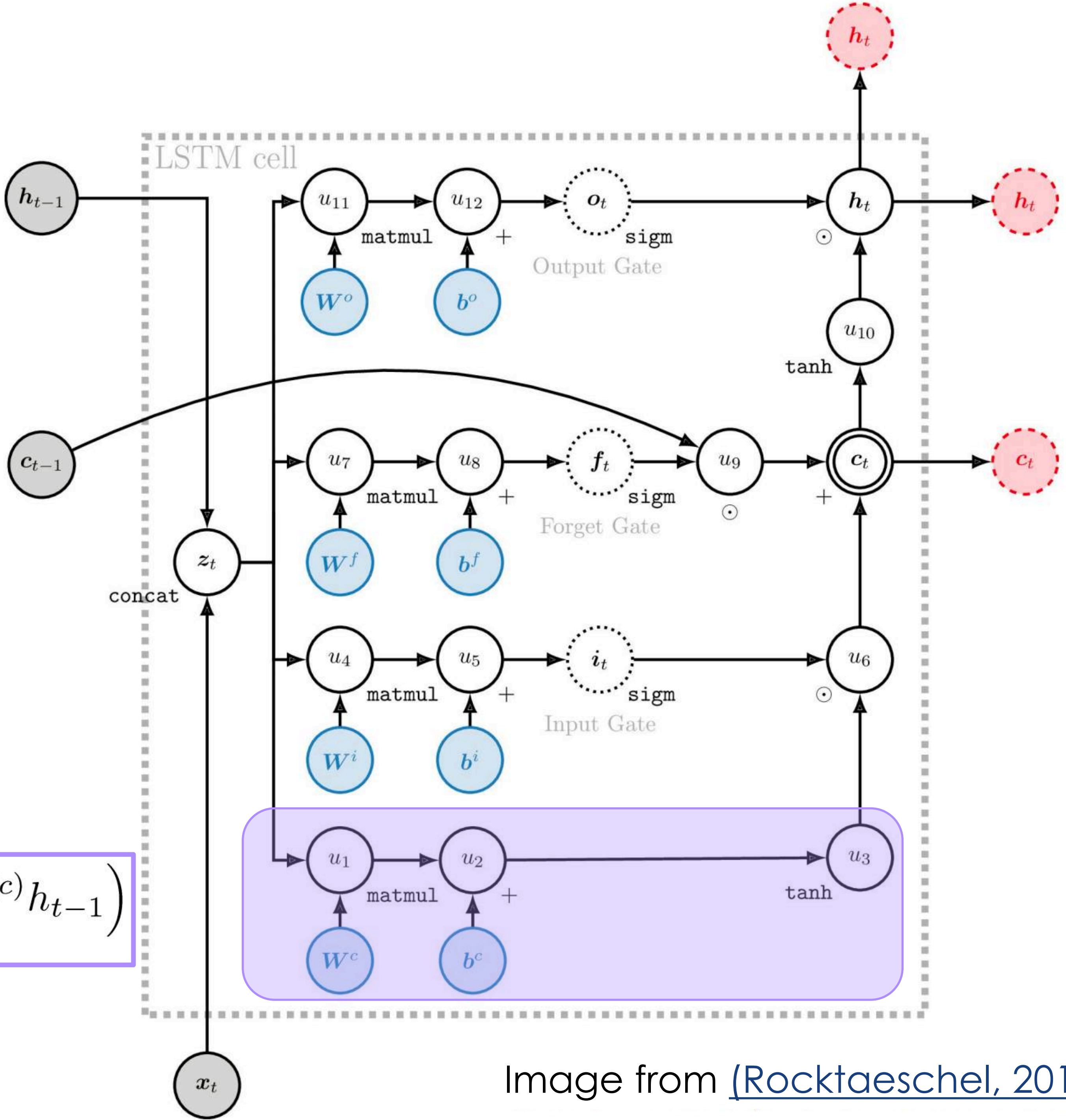
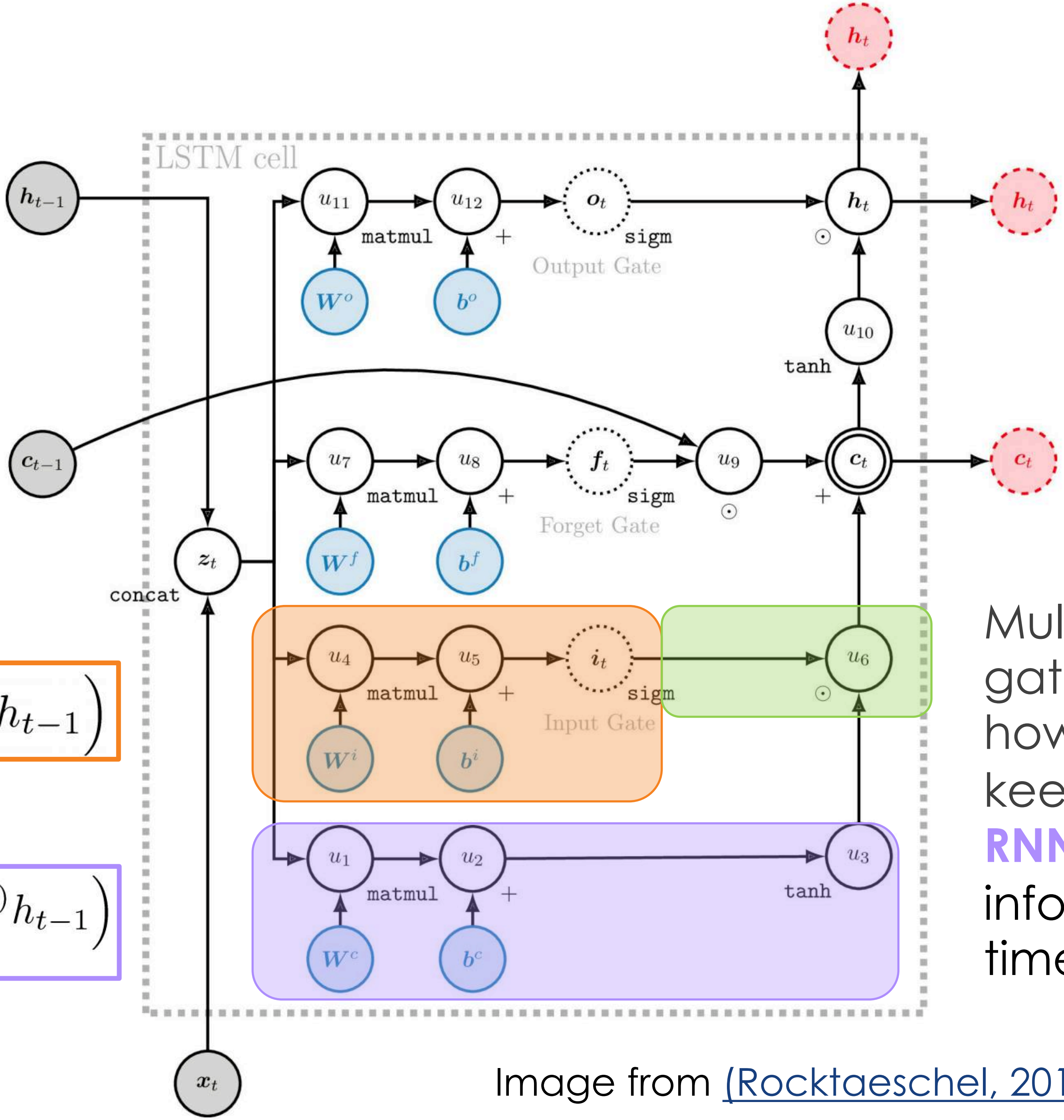


Image from (Rocktaeschel, 2017)

LSTM



$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

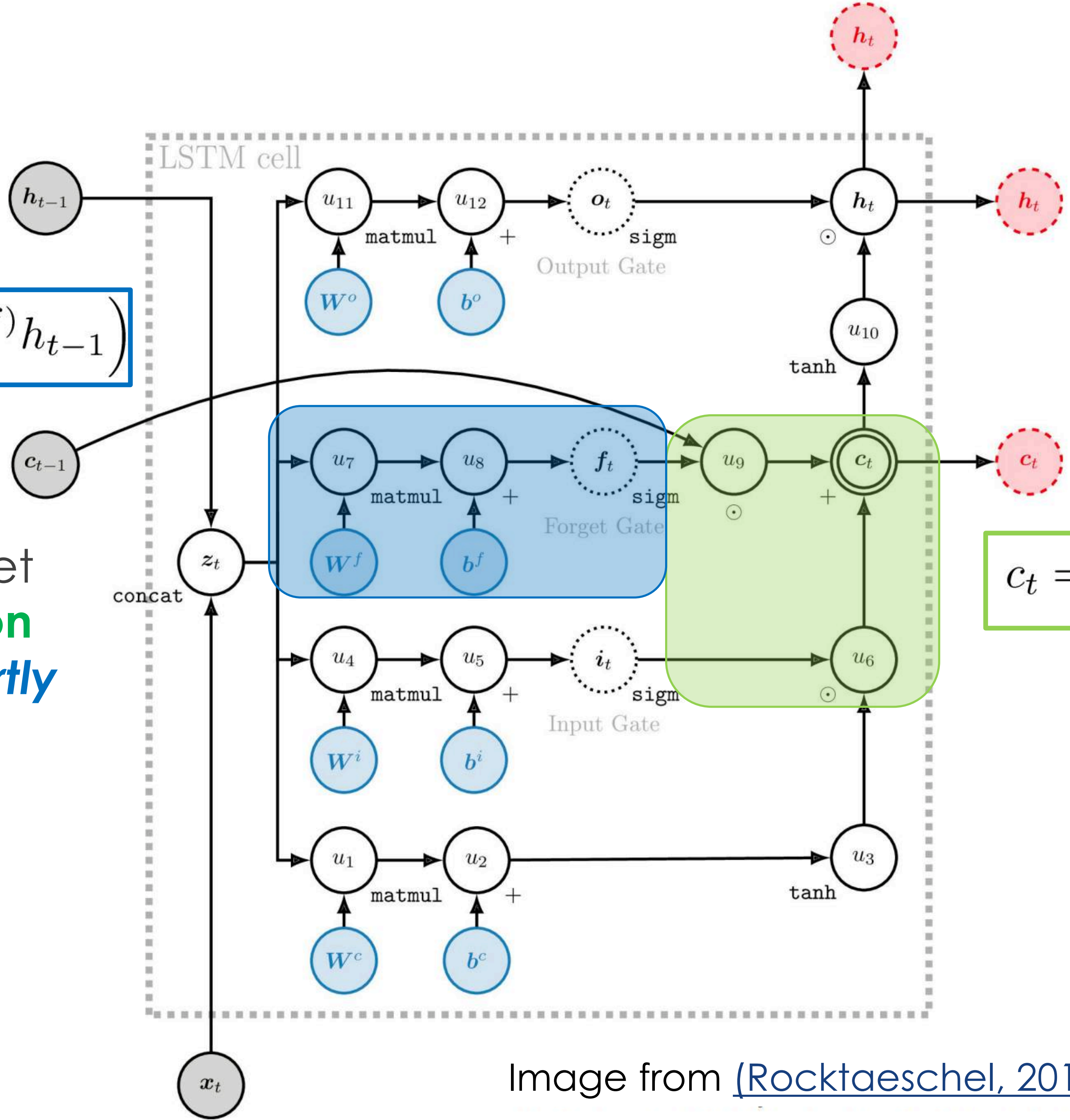
Multiply the **Input** gate output to control how much info to keep from the **vanilla RNN** using the kept info for the current timestep **prediction**

Image from (Rocktaeschel, 2017)

LSTM

$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$

Rather than multiplying, we get the **final prediction** c_t by adding "***partly forgotten***" c_{t-1}



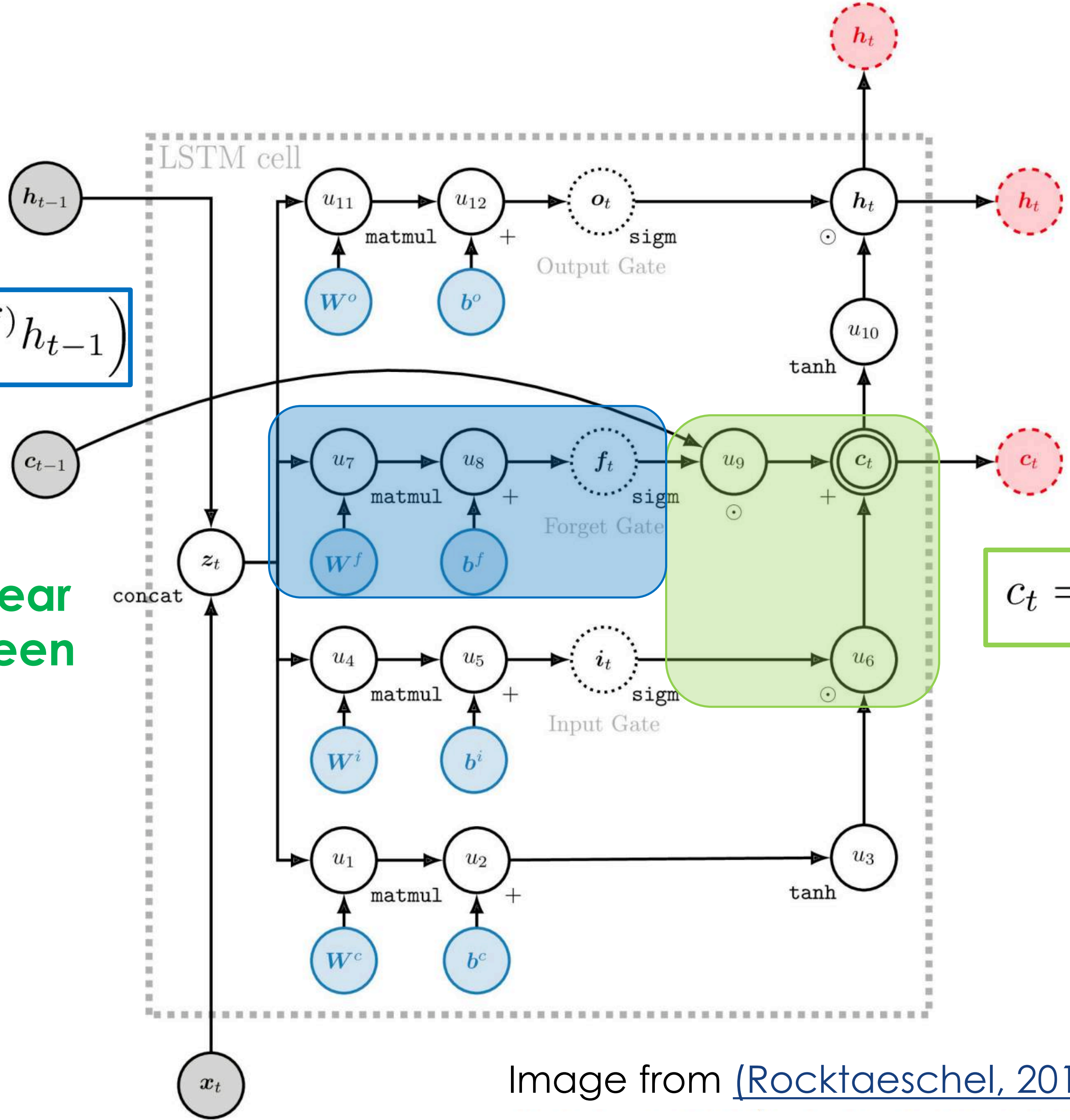
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Image from [(Rocktaeschel, 2017)]

LSTM

$$f_t = \sigma\left(W^{(f)} x_t + U^{(f)} h_{t-1}\right)$$

This creates an **additive direct linear connection between c_t and c_{t-1}**

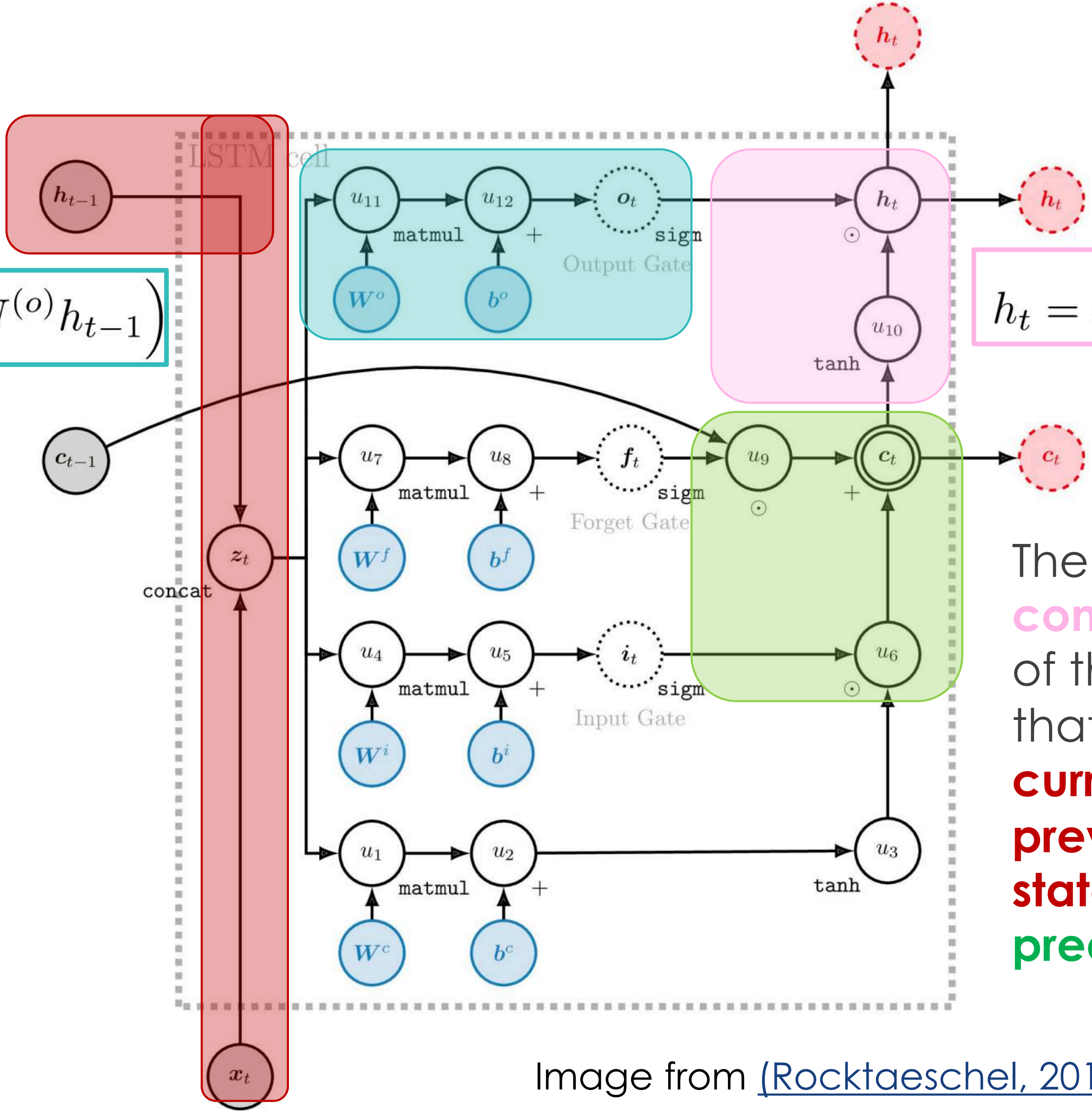


$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Image from [(Rocktaeschel, 2017)]

LSTM

$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$



$$h_t = o_t \circ \tanh(c_t)$$

The final cell combines the output of the output gates (that controls info from **current input** and **previous hidden state**) and the current **prediction**

Image from (Rocktaeschel, 2017)

Bruce Lee on LSTM

"*Absorb* ***what is useful***,
discard what is not,
add what is uniquely
your own."
– Bruce Lee

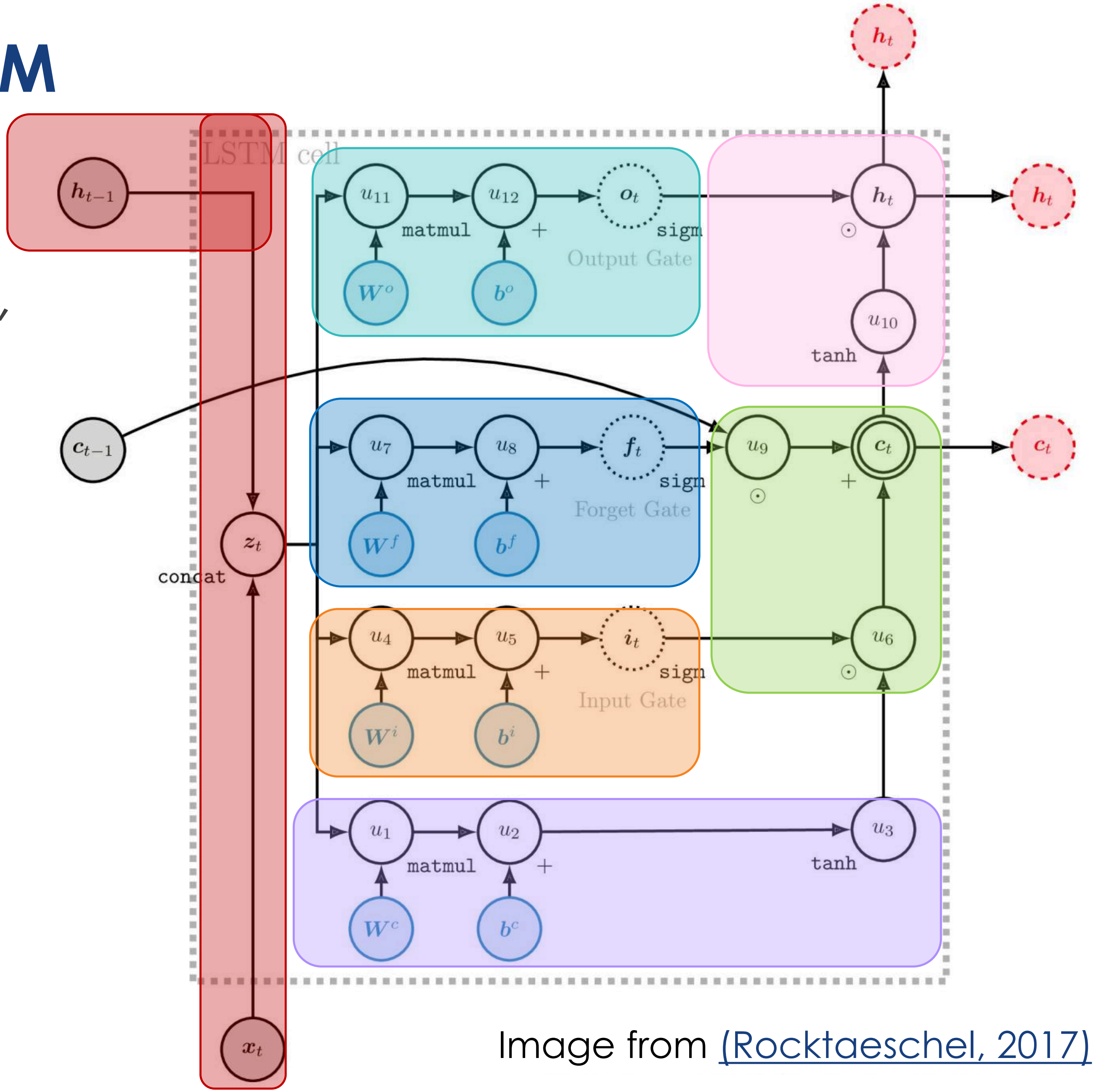


Image from [(Rocktaeschel, 2017)]

Gated Recurrent Neural Nets

- At each time step, perform the following operations

Update: controls how much info in the new hidden states are kept or updated

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

Reset: controls how much info in the previous hidden states are kept

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

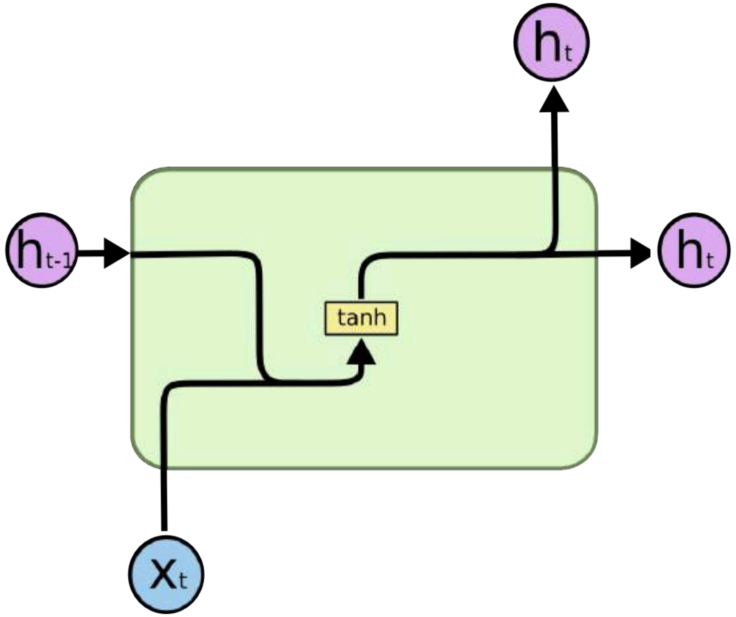
New Hidden state: Reset selects info from previous hidden state and combines it with the current input

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

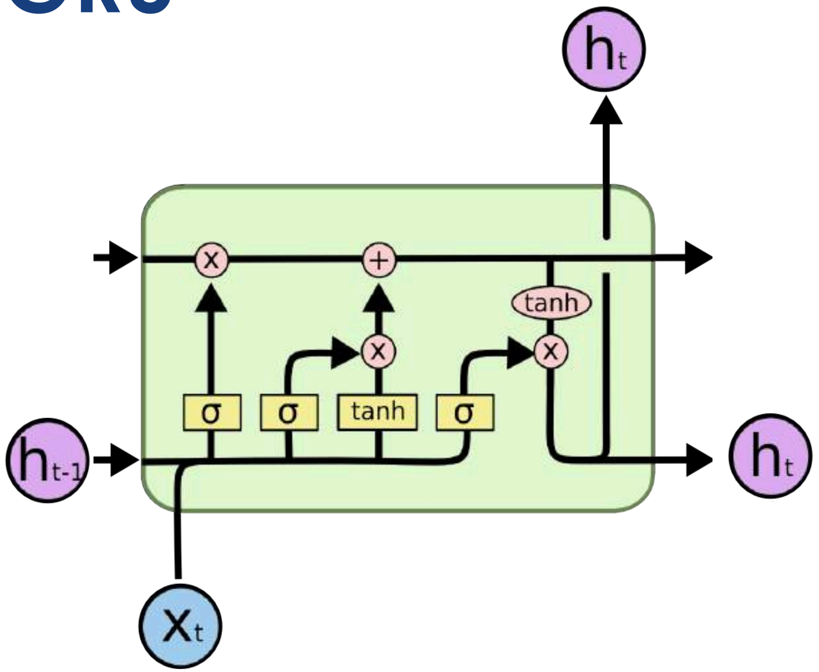
Hidden state: Update selects info from previous hidden state and combines it with the current input

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

RNN vs LSTM vs GRU



$$h_t = \tanh(W x_t + U h_{t-1})$$



$$i_t = \sigma\left(W^{(i)} x_t + U^{(i)} h_{t-1}\right)$$

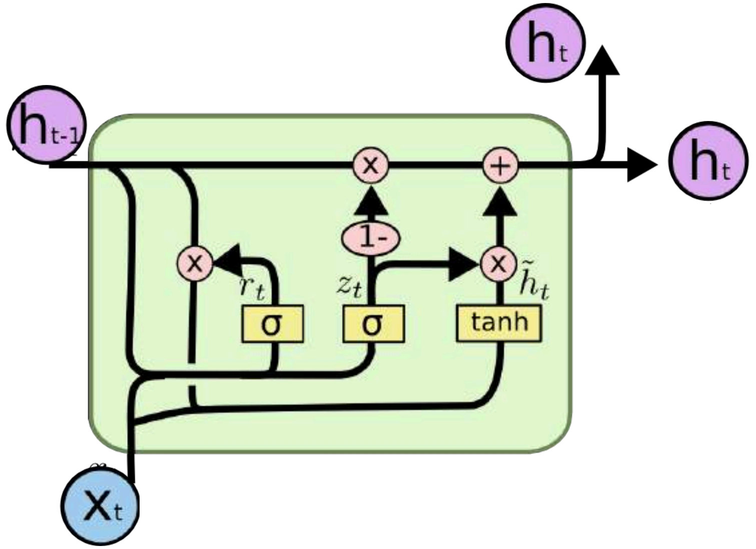
$$f_t = \sigma\left(W^{(f)} x_t + U^{(f)} h_{t-1}\right)$$

$$o_t = \sigma\left(W^{(o)} x_t + U^{(o)} h_{t-1}\right)$$

$$\tilde{c}_t = \tanh\left(W^{(c)} x_t + U^{(c)} h_{t-1}\right)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tanh(c_t)$$



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

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

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

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

Images from (Colah 2015)

LSTM vs GRU

- **No conclusive evidence on which performs better**

- ***"I am impatient"*** -> Use **GRU** (fewer parameters)

- ***"I want good results easily"*** -> Use **LSTM** (more papers used it)

- In both cases, enjoy the grind of tuning hyperparameters…

Throw away your RNN

- *"We fell for RNN, LSTM, and all their variants.* ***Now it is time to drop them!***" – Eugenio Culurciello
- RNNs are **not hardware friendly**
- **Attention-base models** (e.g. Transformer) outperforms RNN (Vaswani et al. 2017)
- **Hierarchical Attention** (Yang et al. 2016) or **Casual Convolution Networks** (Elbayad et al. 2018) outperforms LSTMs and Transformers

LSTM: A Search Space Odyssey

- Greff et al. (2017) explored 8 variants of LSTMs

- No Peepholes == GRU

- Original LSTM works well

- CIFG and GRU simplifies LSTM but no drop in performance

NIG: No Input Gate: $\mathbf{i}^t = \mathbf{1}$
NFG: No Forget Gate: $\mathbf{f}^t = \mathbf{1}$
NOG: No Output Gate: $\mathbf{o}^t = \mathbf{1}$
NIAF: No Input Activation Function: $g(\mathbf{x}) = \mathbf{x}$
NOAF: No Output Activation Function: $h(\mathbf{x}) = \mathbf{x}$
CIFG: Coupled Input and Forget Gate: $\mathbf{f}^t = \mathbf{1} - \mathbf{i}^t$
NP: No Peepholes:

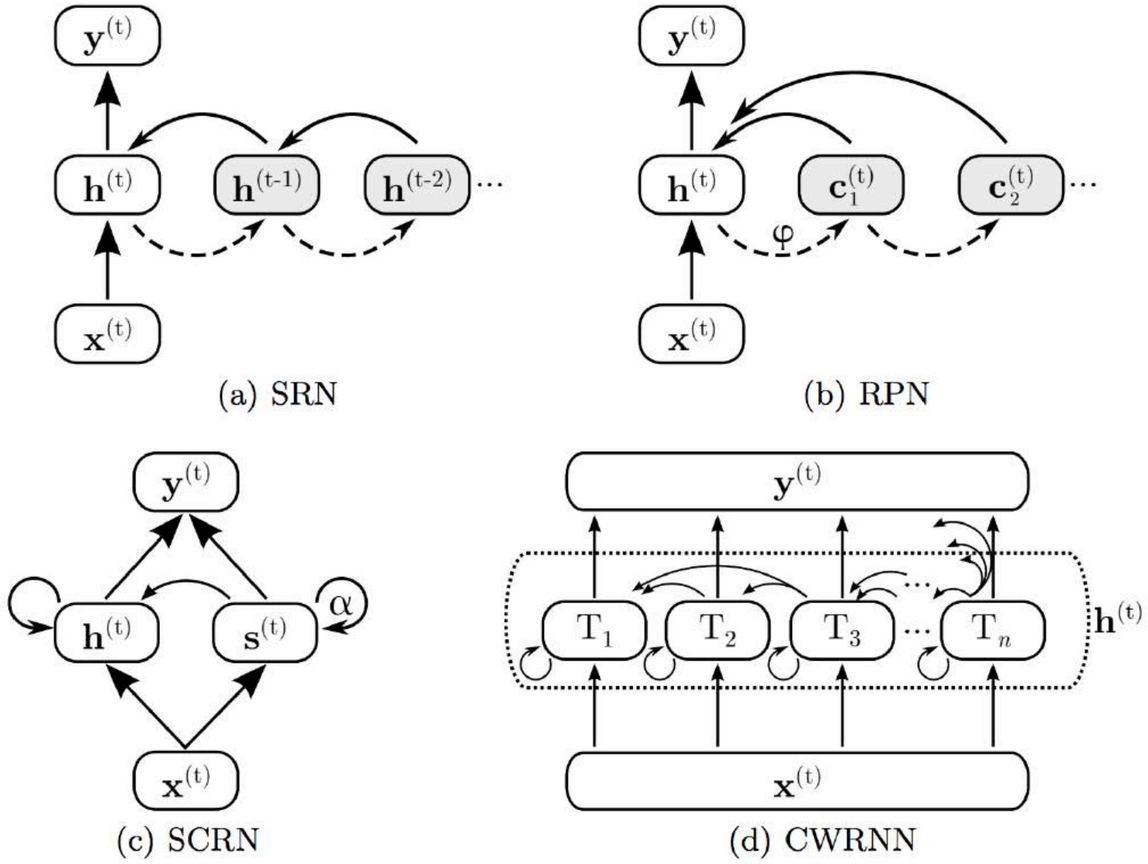
$$\bar{\mathbf{i}}^t = \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{b}_i$$
$$\bar{\mathbf{f}}^t = \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{b}_f$$
$$\bar{\mathbf{o}}^t = \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{b}_o$$

FGR: Full Gate Recurrence:

$$\begin{aligned}
\bar{\mathbf{i}}^t &= \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i \\
&\quad + \mathbf{R}_{ii}\mathbf{i}^{t-1} + \mathbf{R}_{fi}\mathbf{f}^{t-1} + \mathbf{R}_{oi}\mathbf{o}^{t-1} \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f \\
&\quad + \mathbf{R}_{if}\mathbf{i}^{t-1} + \mathbf{R}_{ff}\mathbf{f}^{t-1} + \mathbf{R}_{of}\mathbf{o}^{t-1} \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^{t-1} + \mathbf{b}_o \\
&\quad + \mathbf{R}_{io}\mathbf{i}^{t-1} + \mathbf{R}_{fo}\mathbf{f}^{t-1} + \mathbf{R}_{oo}\mathbf{o}^{t-1}
\end{aligned}$$

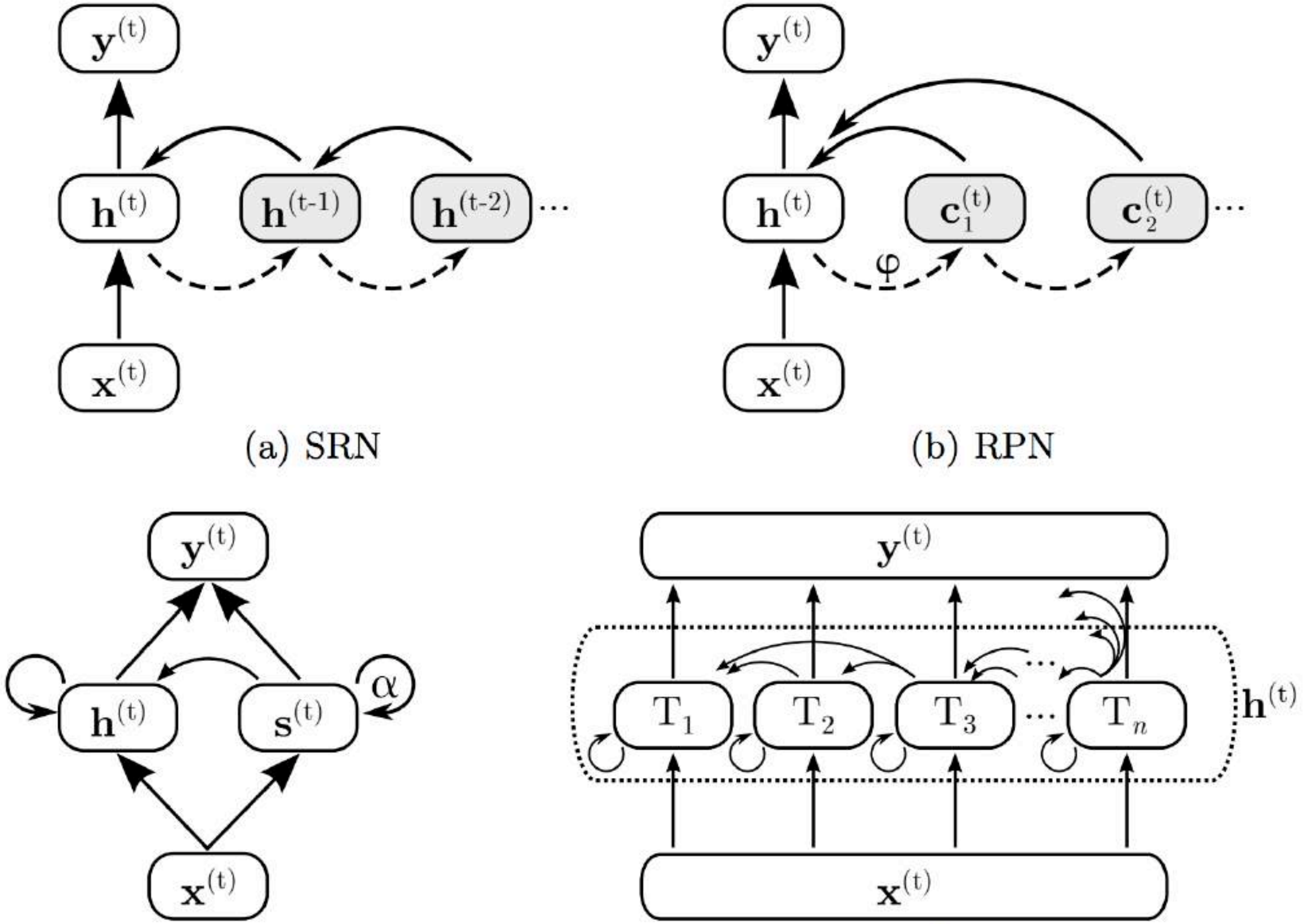
Learning Multiple Timescales in RNNs

- Alpay (2016) investigated different ways to manipulate the time-steps in RNNs



Learning Multiple Timescales in RNNs

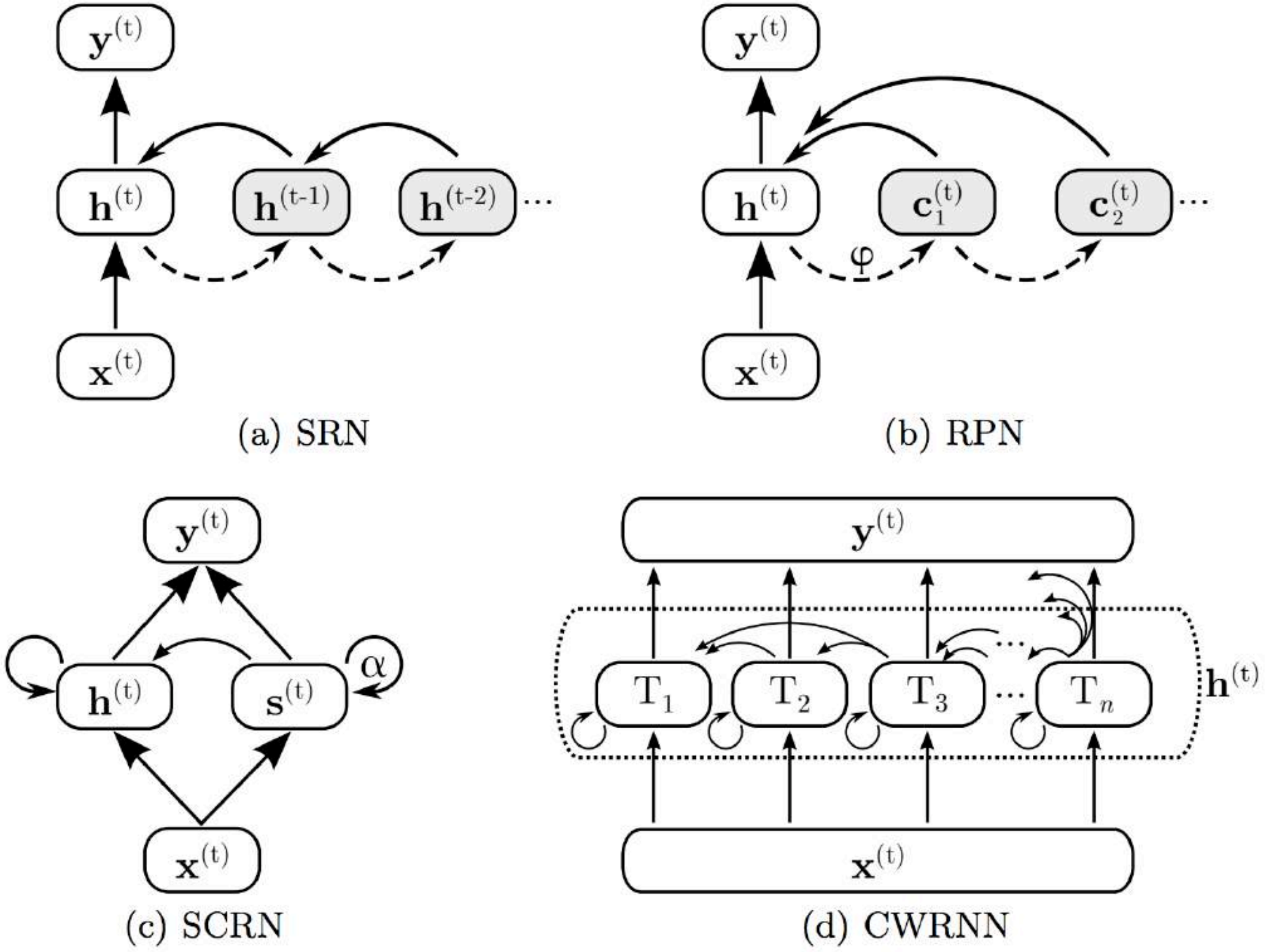
- Simple Recurrent Network (SRN) == Vanilla RNN
- Next prediction conditioned on previous timestep
- Backpropagation through time sequentially



(a) SRN (b) RPN (c) SCRN (d) CWRNN

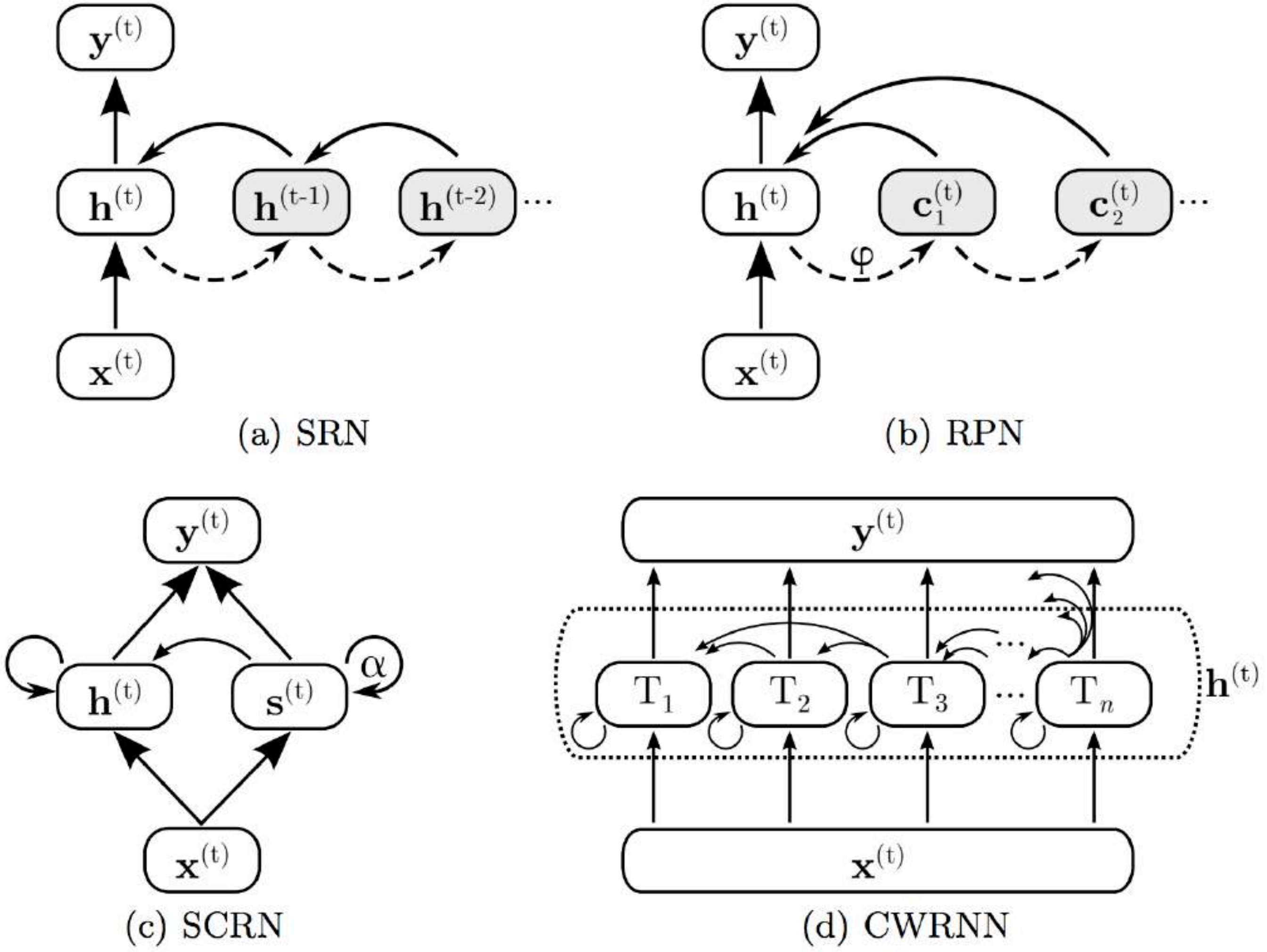
Learning Multiple Timescales in RNNs

- Recurrent Plausible Network (RPN) (Wermter et al. 1995)
- Short cutting the connections between timesteps
- Needs to account for a timelag factor φ



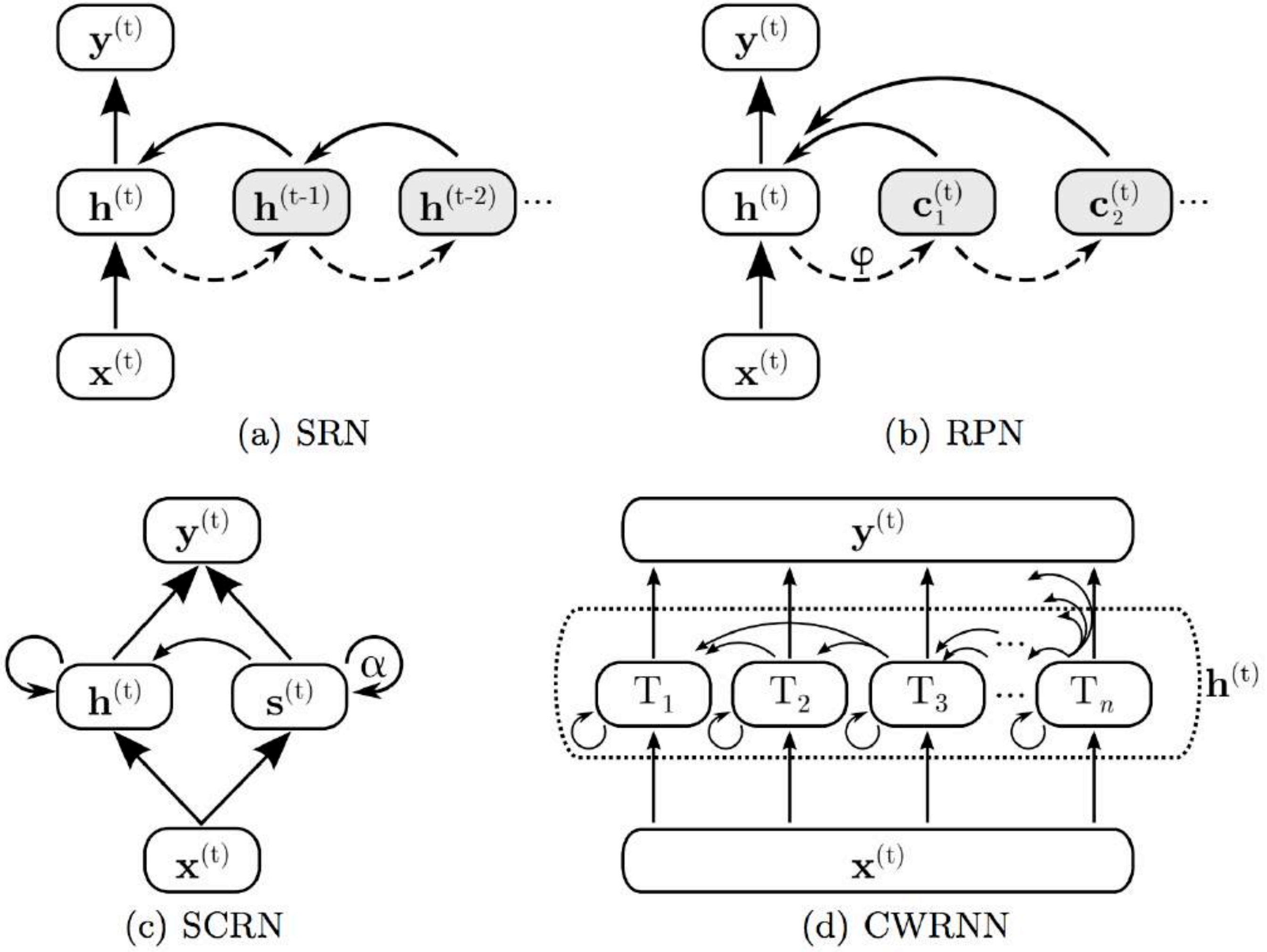
Learning Multiple Timescales in RNNs

- Recurrent Plausible Network (RPN) (Wermter et al. 1995)
- Short cutting the connections between timesteps
- Needs to account for a timelag factor φ



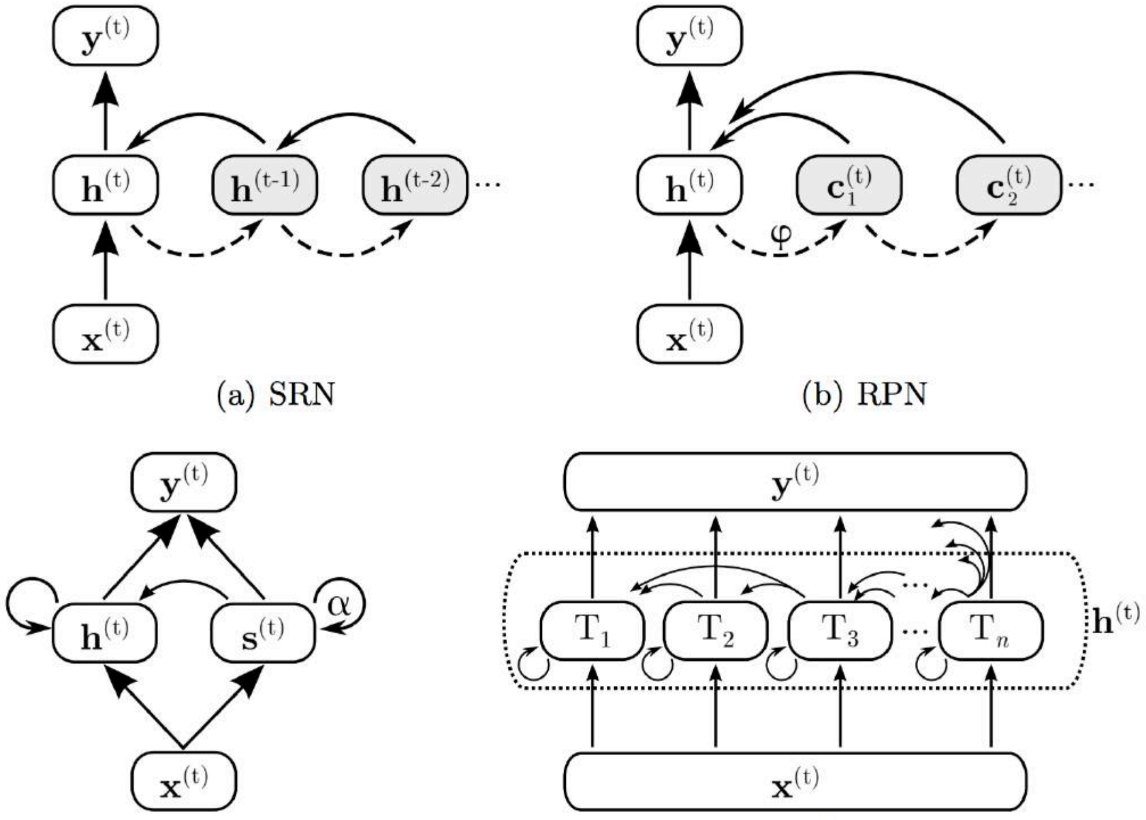
Learning Multiple Timescales in RNNs

- Clockwork RNN (CWRNN) (Koutnik et al. 2014)
- No time lag parameter
- An external module controls how much temporal information to propagate



Learning Multiple Timescales in RNNs

- Alpay (2016) investigated different ways to manipulate the time-steps in RNNs
- Manipulating timesteps connection allows units to self-organize long vs short term contexts
- Not sure which is really the best for NLP =(



(a) SRN (b) RPN (c) SCRN (d) CWRNN

LSTM/GRU solves the vanishing gradients?

- The additive memory of the "**forget gate**" prevents small partial gradients from disappearing

- It remembers information over multiple timesteps so the **hidden states** don't disappear across time

- But **LSTM/GRU doesn't guarantee no gradient vanishing/exploding**, it's just better than the vanilla RNN

Is vanishing/exploding gradient just an RNN problem?

- Nope, as long as ***functions keeps getting nested*** in the network and the ***partial derivations needs to be multiplied*** causing unstable gradients

- As long as there are many layers, the functions and gradients gets multiplied in a nested manner

- Without using gates, we can just "***short-circuit" the network and make previous layers interact directly with the current layers***

Non-Gate Connections

- He et al. (2015) proposed Residual Networks, that skip connections for alternate layers

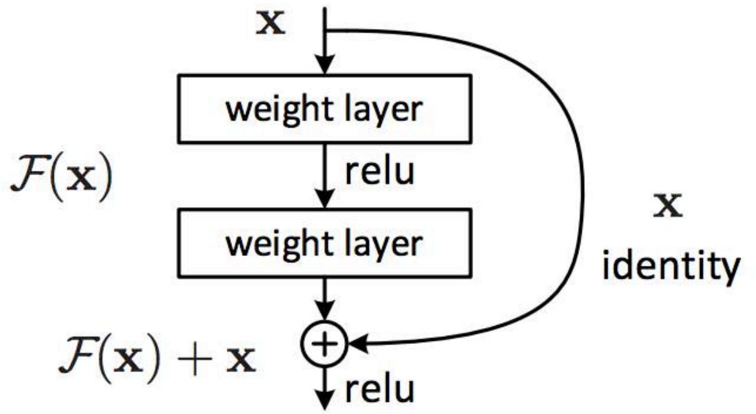


Figure 2. Residual learning: a building block.

Non-Gate Connections

- He et al. (2015) proposed Residual Networks, that skip connections for alternate layers

- Huang et al. (2018) proposed connects every previous layer to every layer down the network

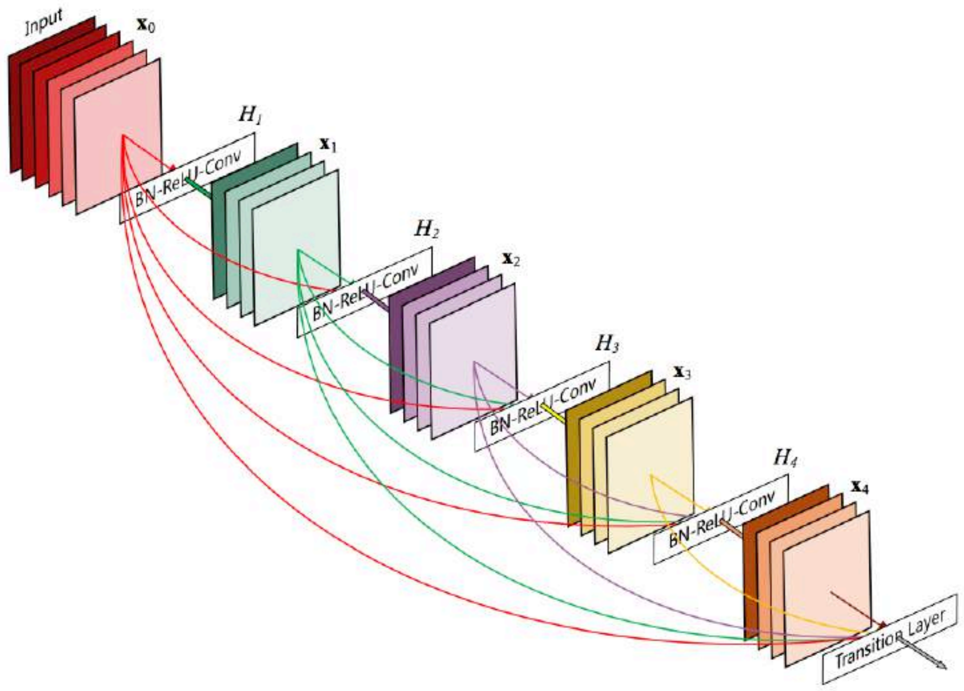


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

LSTMs: Real-Word Success (Briefly)

- In **2013-2015**, LSTMs started achieving state-of-the-art results
  - Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning
  - **LSTM** became the **dominant approach**

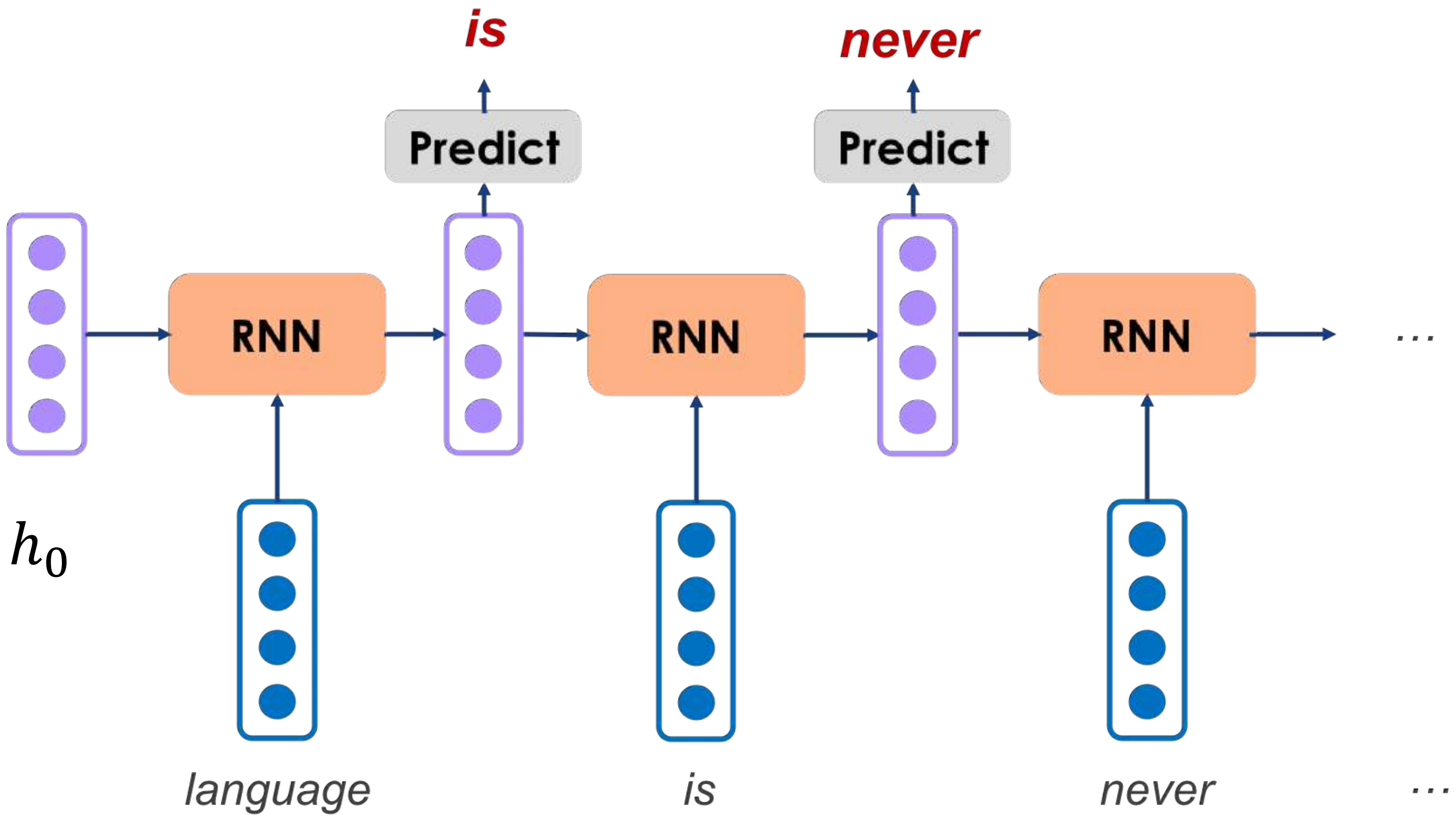
- **Now (2019),** other approaches (e.g. **Transformers**) have become more dominant for certain tasks.
  - For example in WMT (a MT conference + competition):
  - In WMT **2016**, the summary report contains **"RNN" 44 times**
  - In WMT **2018**, the report contains **"RNN" 9 times** and **"Transformer" 63 times**

Almost entirely cite from [(See, 2019), Stanford CS244N course]

Conditioned Generation

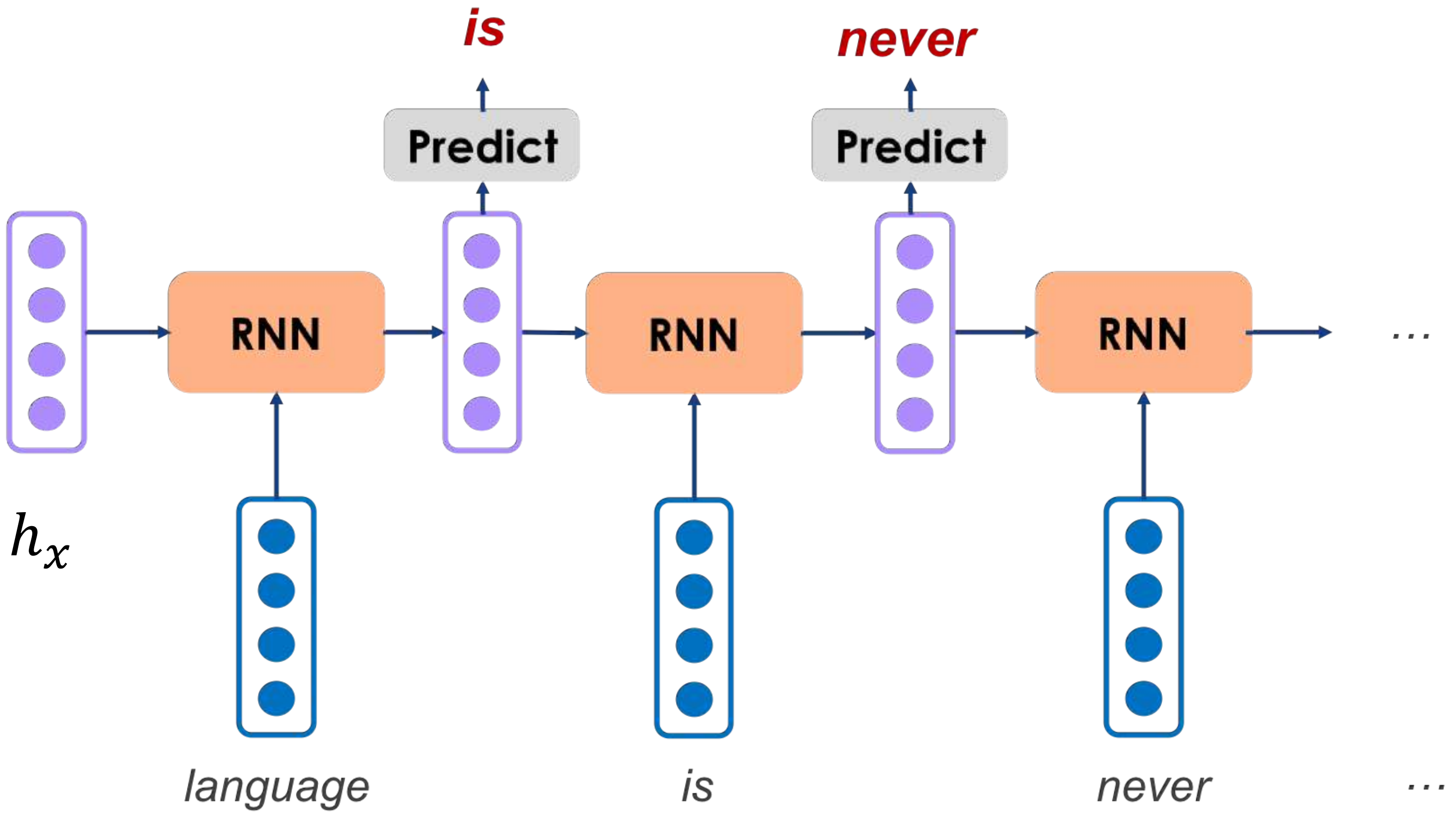
RNN Generation

- RNN generation starts with a random hidden state h_0



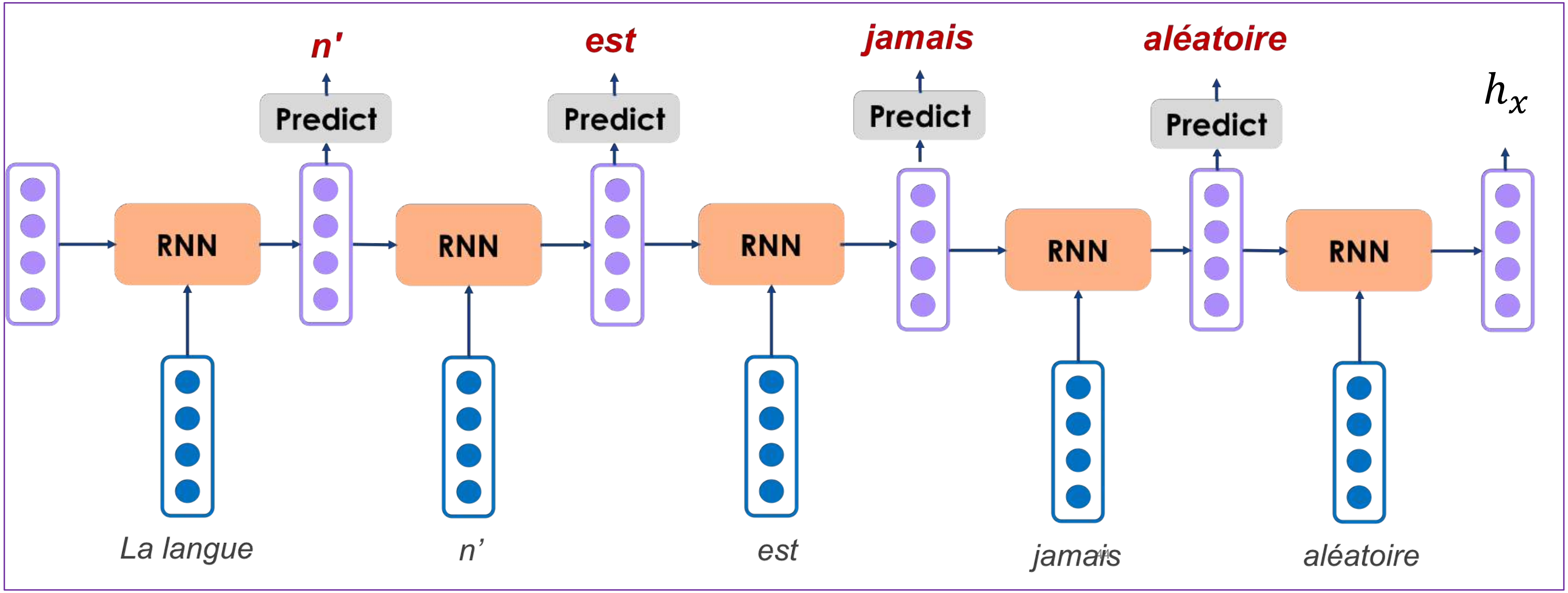
RNN Generation

- What if h_0 is something non-random?



RNN Generation

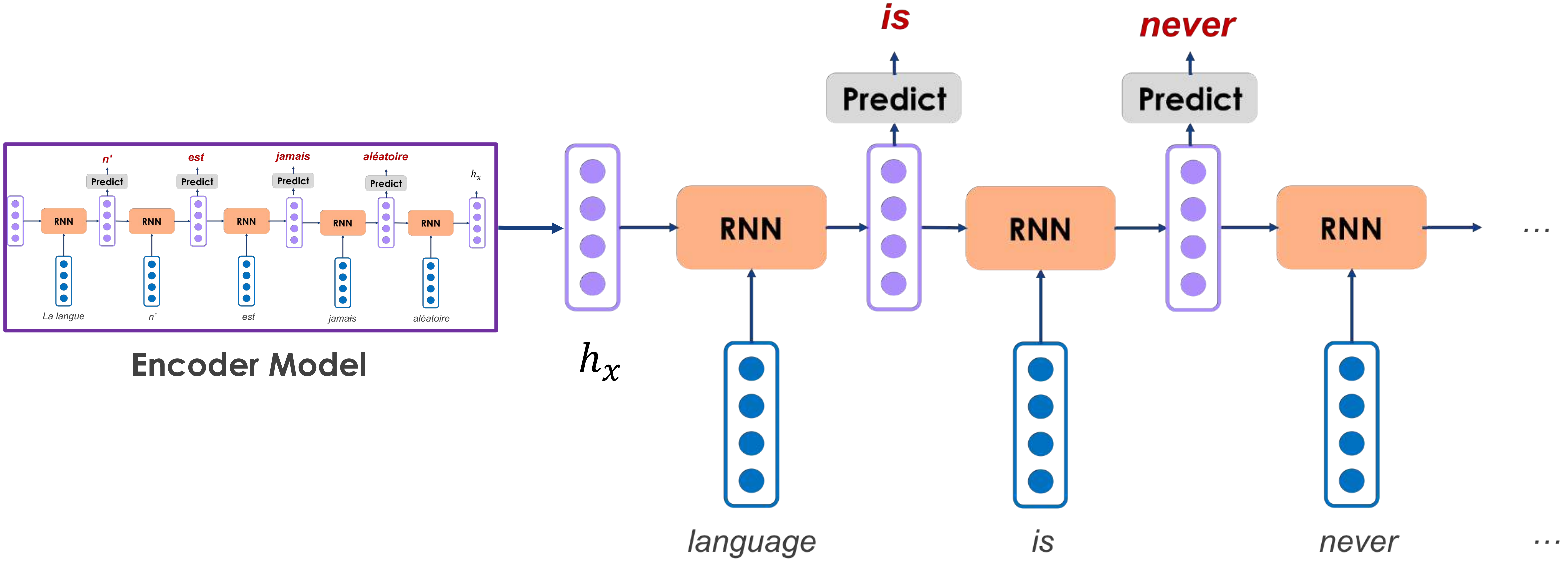
- What if h_0 is something non-random?



Encoder Model

RNN Generation

- What if h_0 is something non-random?



RNN Generation

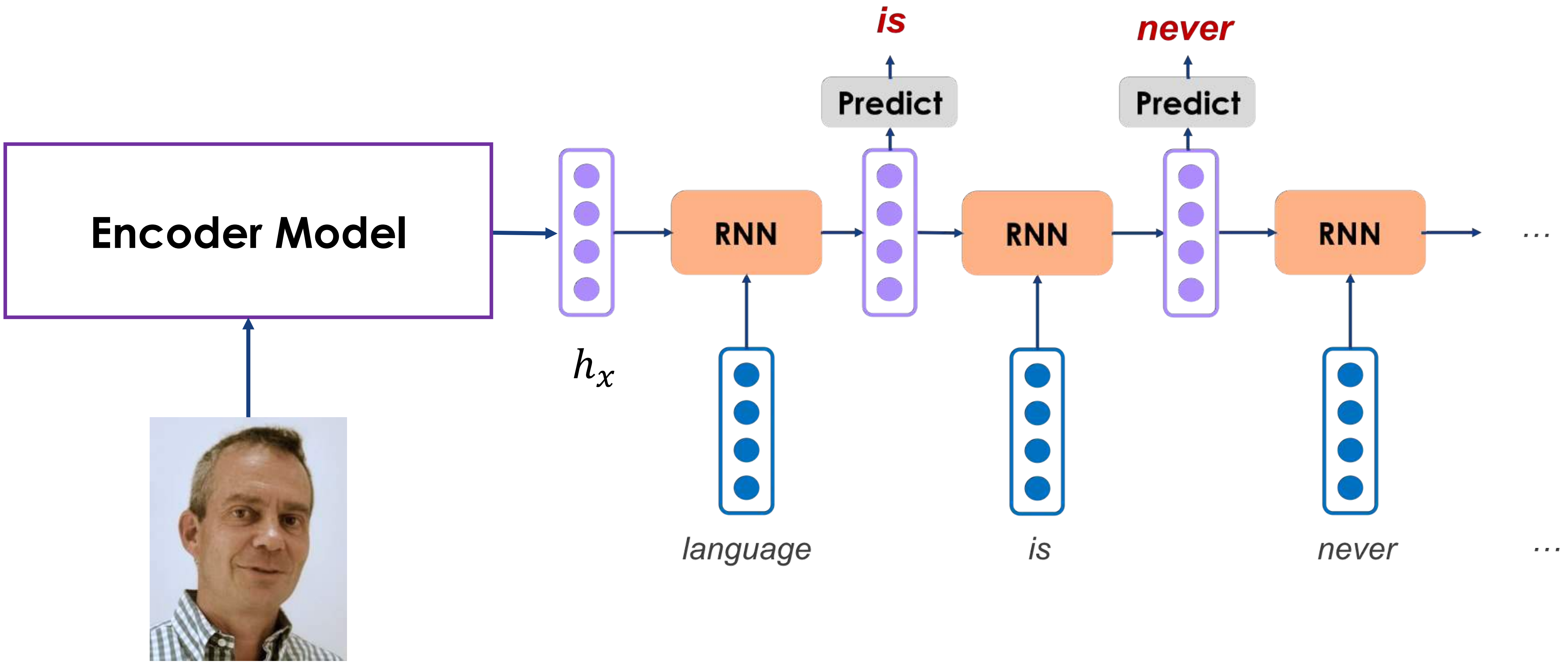
- RNN generation starts with a random hidden state h_0

- What if h_0 is something non-random?

- **FR->EN Translation:** Initialize English model h_0 inputs with h_x outputs from a French model

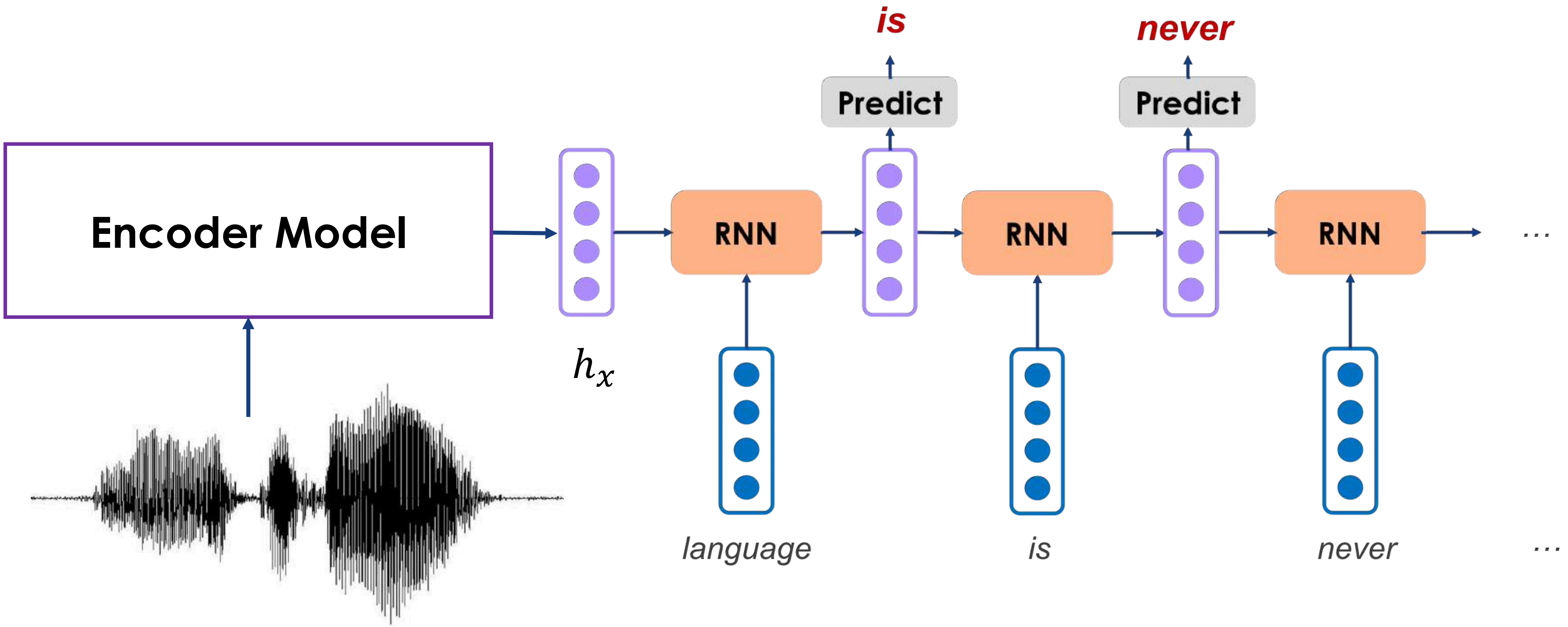
RNN Generation

- What if input of model that produces h_x isn't text?



RNN Generation

- What if input of model that produces h_x isn't text?



RNN Generation

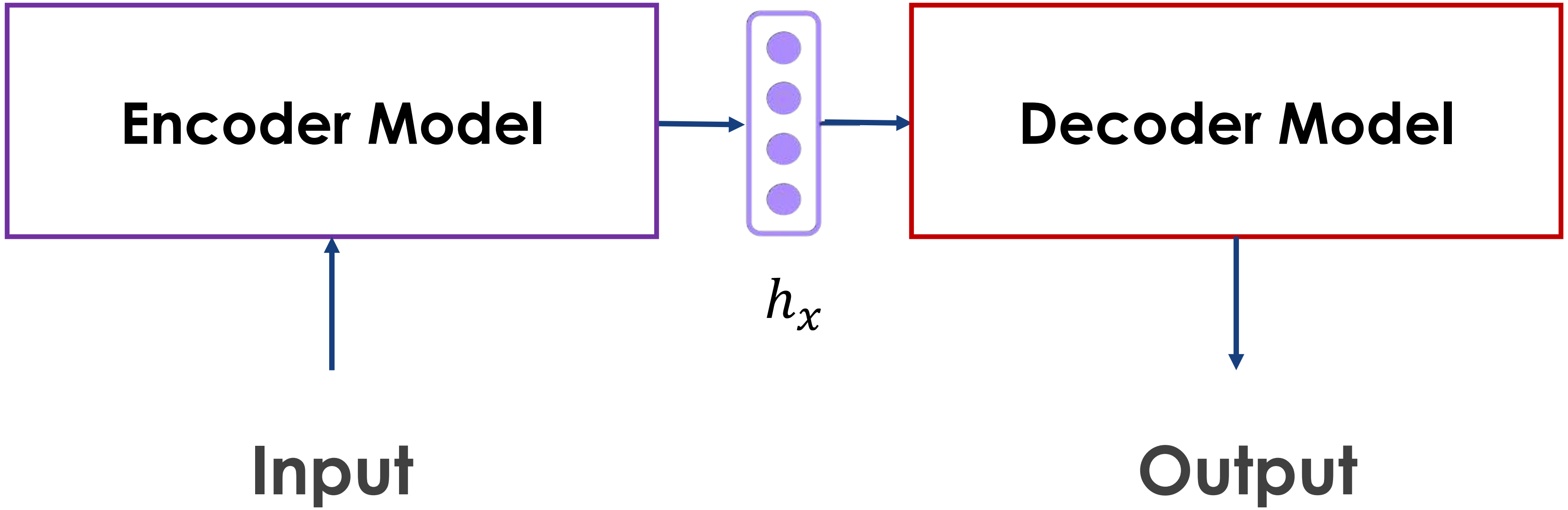
- RNN generation starts with a random hidden state h_0

- What if h_0 is something non-random?

- **Translation:** French -> English
- **Image Captioning:** Image -> Text
- **Speech Recognition:** Audio -> Text

RNN Generation

- Generalized Encode-Decoder Framework



Conditional RNN Generation

- RNN generation starts with a random hidden state h_0

- What if h_0 is something non-random?

- **Translation:** French -> English
- **Image Captioning:** Image -> Text
- **Speech Recognition:** Audio -> Text
- **Summarization:** Document -> Summary
- **Chatbots:** Utterance -> Response

Conditioned RNN Generation

- **Endless possibilities** of what to condition on and what to generate

- But training requires the **paired condition and target generation**

- And relatively **large amount of data is needed** for model to train well

Language Model Probability

$$P(X) = \prod_{i=1}^{I} P(\boldsymbol{x_i} | \boldsymbol{x_1}, \dots, \boldsymbol{x_{i-1}})$$

Next word **Context**

Conditioned Language Model Probability

Encoder hidden context

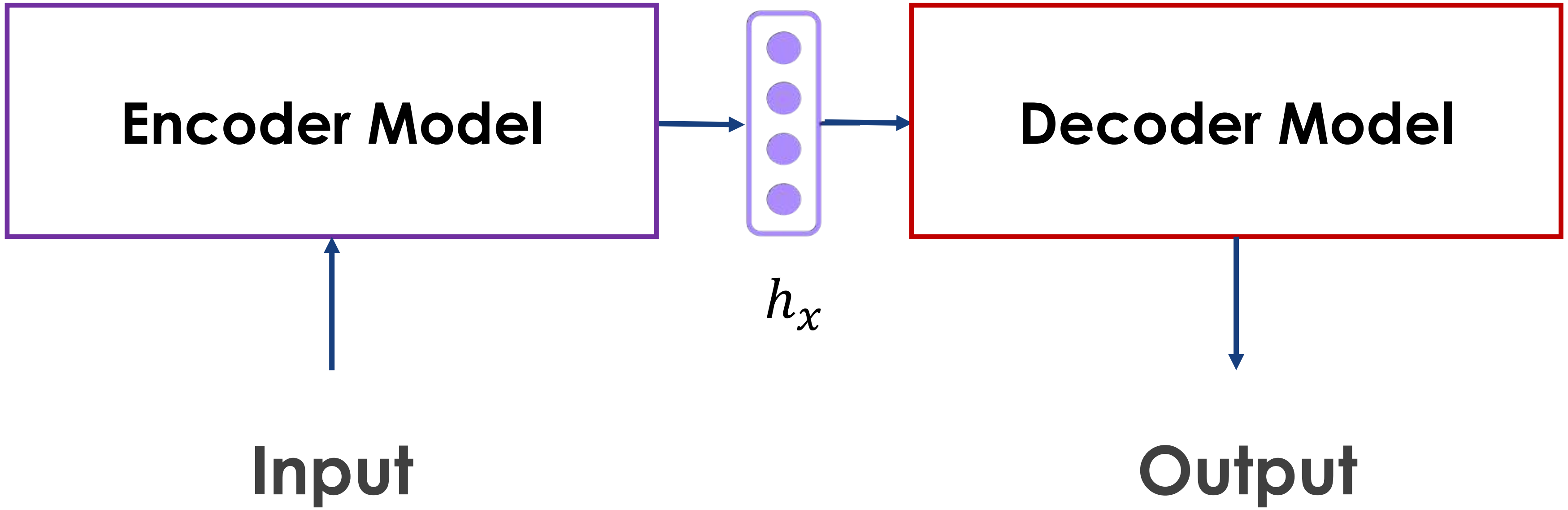
$$P(Y|X) = \prod_{j=1}^{J} P(\boldsymbol{y_j}|\boldsymbol{X}, \boldsymbol{y_1}, \dots, \boldsymbol{y_{i-1}})$$

Next decoder word

Previous decoder context

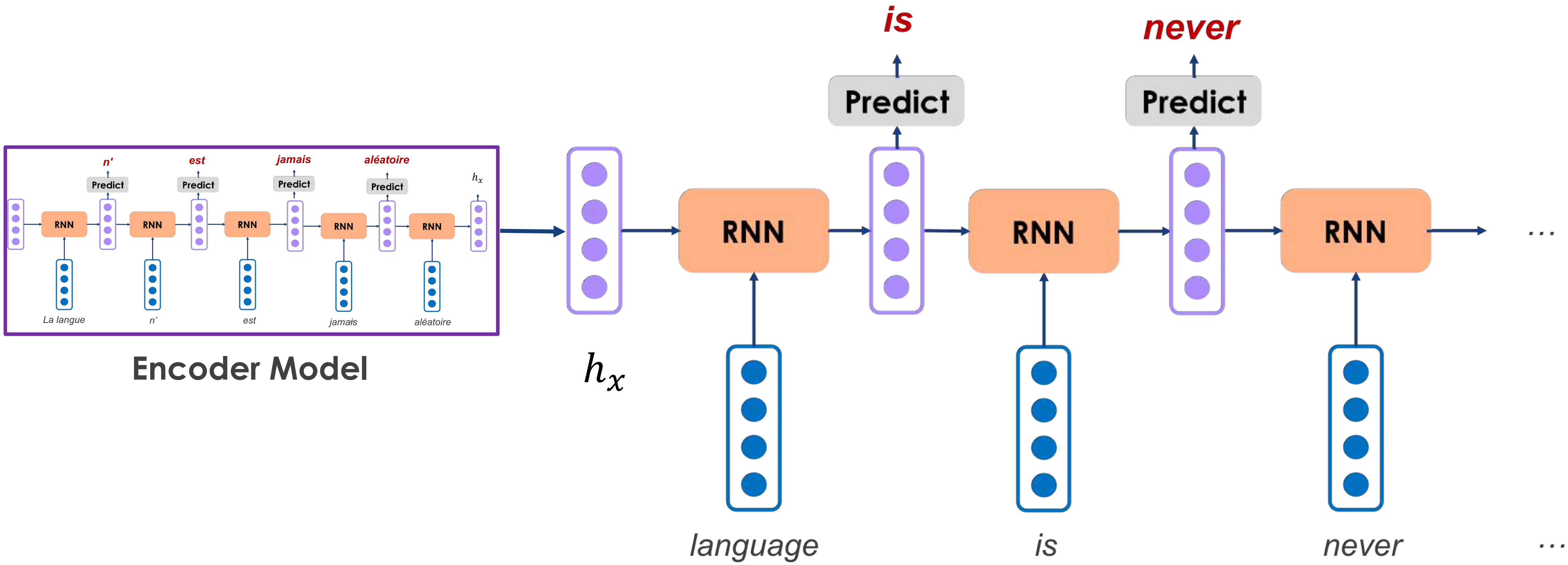
RNN Generation

- Generalized Encode-Decoder Framework



Sequence to Sequence Learning with NN

- Take final hidden state of an encoder model, feed it as the start state of a decoder model (Sutskever et al. 2014)



Sequence to Sequence Learning with NN

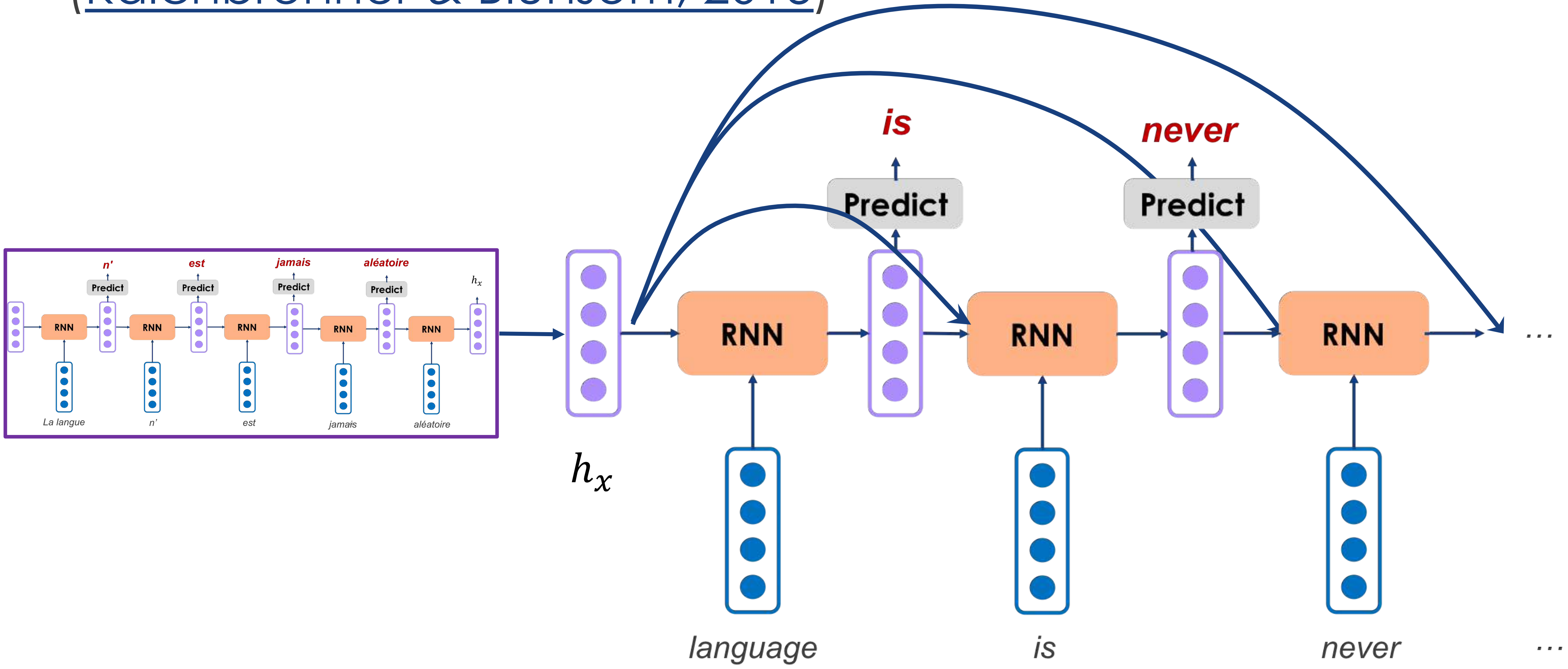
- Take final hidden state of an encoder model, feed it as the start state of a decoder model (Sutskever et al. 2014)

- RNNs deal naturally with undefined encoding input lengths

- But we’re depending on a single hidden state to squeeze information of the inputs to feed to the decoder

Recurrent Continuous Translation Models

- Add the encoded hidden state at every decoder time step ([Kalchbrenner & Blunsom, 2013](#))



Recurrent Continuous Translation Models

- Add the encoded hidden state at every decoder time step (Kalchbrenner & Blunsom, 2013)
- Having the encoded hidden state force fed to every decoder timestep is too overwhelming
- Still, too much depend on that one hidden encoder state

Greedy Decoding

- Generally, we want to find the most probable output given the input
- Simple approximation is to pick the highest probability word at each time step, `torch.max(predictions, 1)`
- Simple causes problem…
  - Often generate “easy” words first
  - Prefers common phrases to one rare word

Beam Search

- Better approximation is to predict k-best outputs at each time step

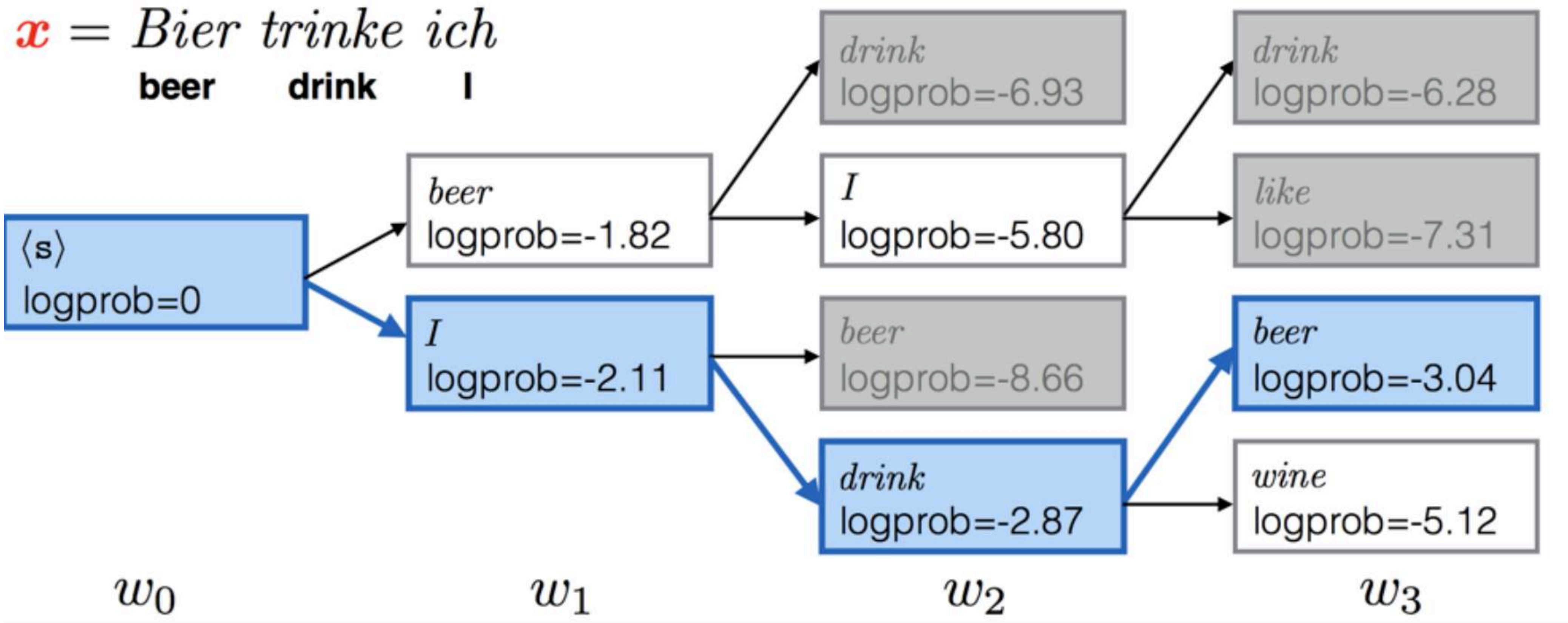


Figure and example from (Dyer, 2018)

Environment Setup

Open Anaconda Navigator.

Go to the PyTorch installation page, copy the command as per configuration: https://pytorch.org/get-started/locally/

Fire up the terminal in Anaconda Navigator.

Start a Jupyter Notebook.

Download http://bit.ly/ANLP-Session6Gen

Import the .ipynb to the Jupyter Notebook

Summary

Gated RNN + Generation Knowledge Checklist

- **Gated Recurrent Neural Net**
  - Always use GRU or LSTM, never vanilla RNN
  - When gradient explodes, clip it, even if it doesn't still clip it
  - Bi-directional GRU/LSTM produces reasonable results

- **Conditioned Generation**
  - Encode inputs into a hidden state vector/matrix, then generating outputs stepwise in an RNN
  - Pairs of conditions and target generations are necessary
  - When possible, always do beam search

Fin