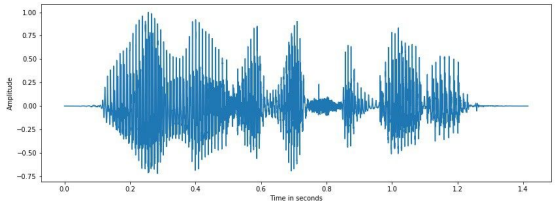
11-785: Recitation 8 (Spring 22)

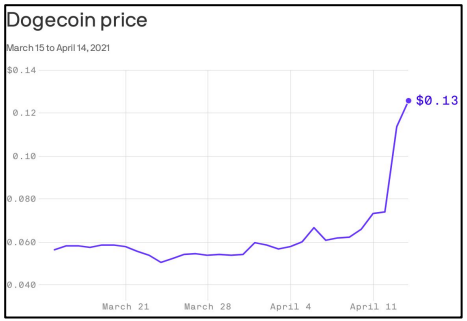
RNN Basics

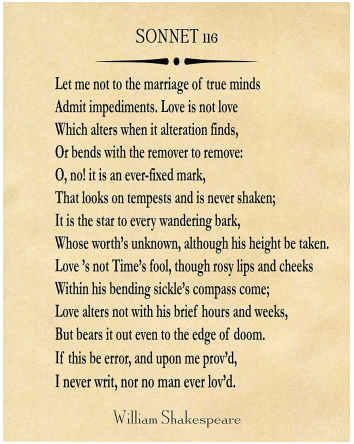
Aparajith, Soumya, Shreyas, Lavanya

Sequential Data

- Data from which various inputs are dependent
- Examples:
  - Text: *“Hi. How are you doing today?”*
  - Audio/speech
  - Video
  - Any other time series data like stock price, daily temperature, etc.






SONNET 116

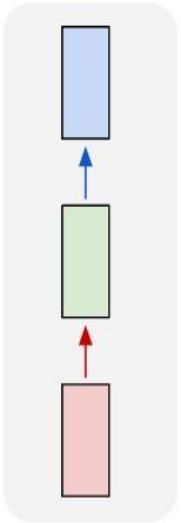
Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O, no! it is an ever-fixed mark,
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth’s unknown, although his height be taken.
Love ’s not Time’s fool, though rosy lips and cheeks
Within his bending sickle’s compass come;
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error, and upon me prov’d,
I never writ, nor no man ever lov’d.

William Shakespeare

Reference: Audio, Stock, Text, Video

Data Modeling

one to one



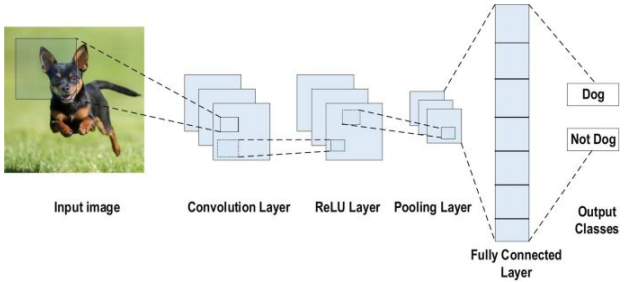
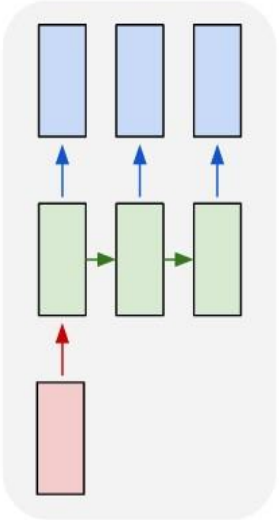


Image Classification (ref)

one to many

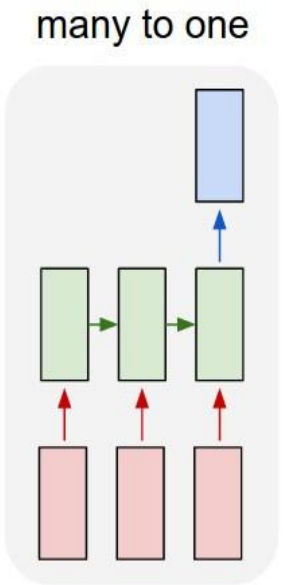


"man in black shirt is playing guitar."

Image Captioning (ref)

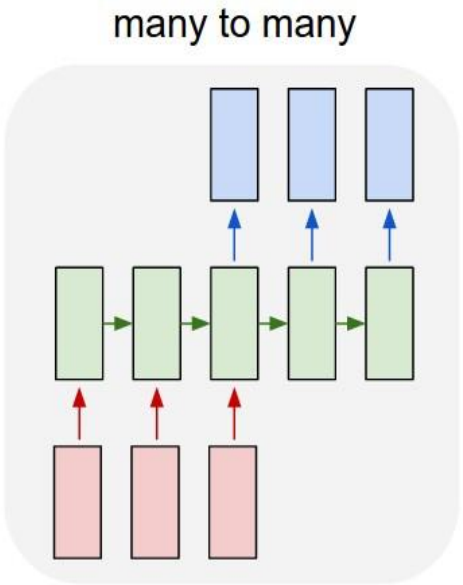
(https://i.stack.imgur.com/b4sus.jpg)

Data Modeling



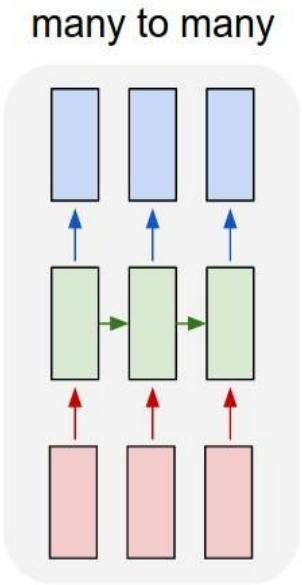
Sentiment Analysis (Movie Review)

The Batman (2022) is everything a superhero movie should be. **(Positive)**



Machine Translation

“How are you?” -> “எப்படி இருக்கிறீர்கள்?"

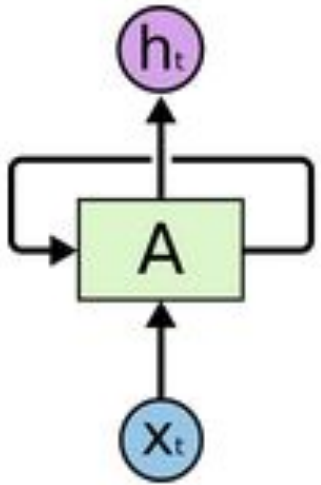


Object Tracking in videos

Video

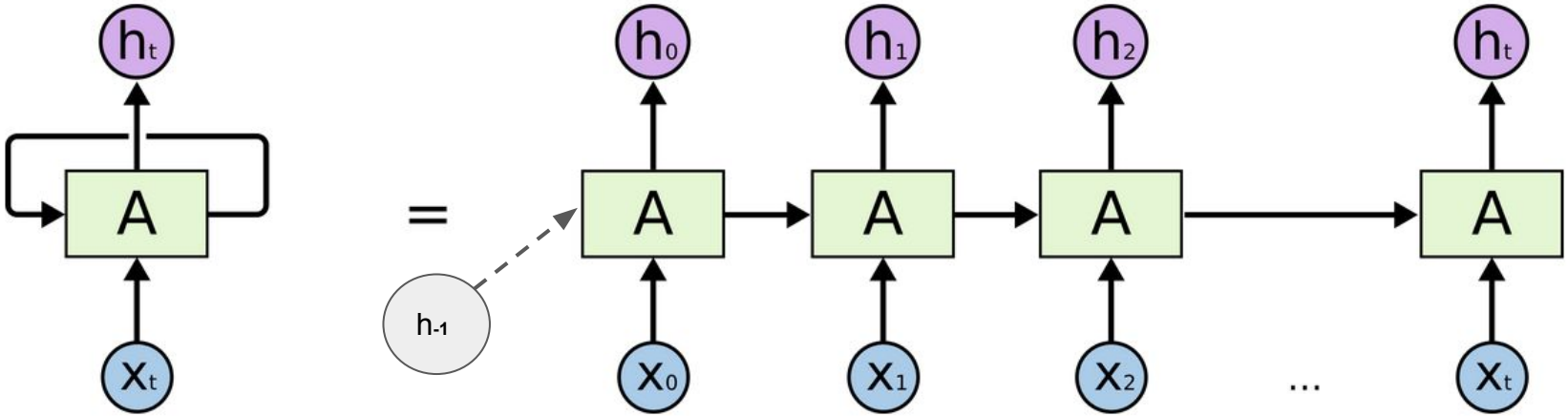
Recurrent Neural Networks

- Looping network
- Parameter sharing across timesteps
- Derivatives aggregated across all time steps
- “Backpropagation through time (BPTT)”



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

RNN Unrolled

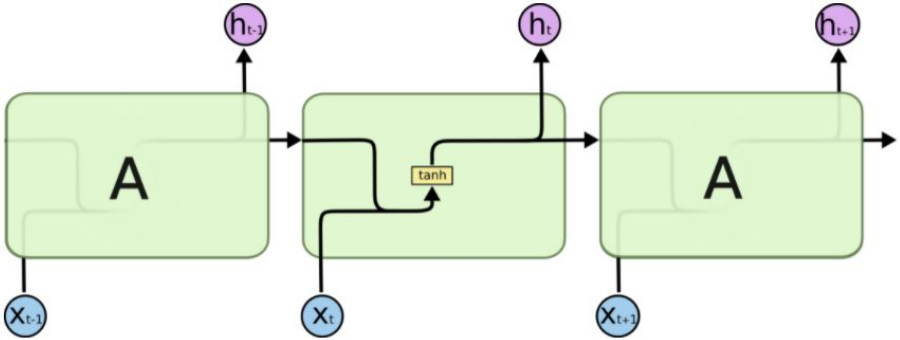


An unrolled recurrent neural network.

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

Problems with RNN

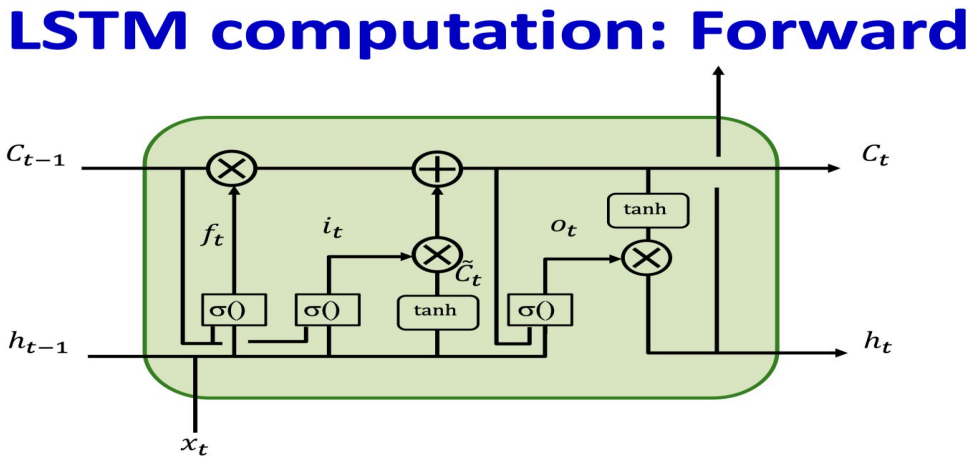
- Short term memory
- Exploding gradients
- Vanishing gradients



Factors governing the retention of memory in RNN:

- Weights of the recurrent layers
- Bias of the recurrent layers
- Activation function used in recurrent layers

LSTM Cell (Variation 1)



- Forward rules:

Gates

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

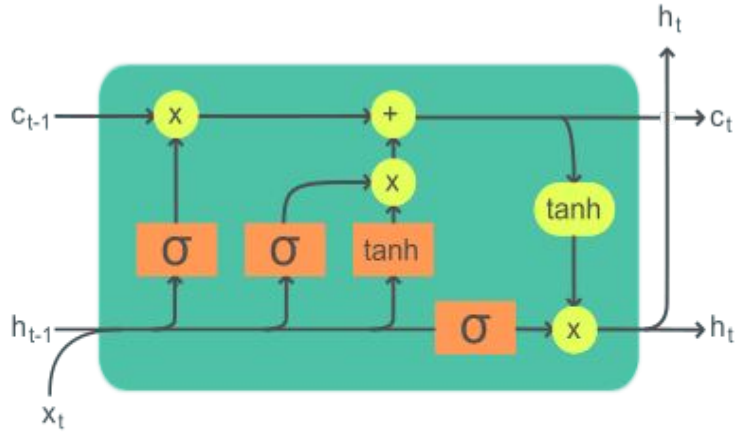
Variables

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh\left(C_t\right)$$

(Gers and Schmidhuber 2000: *Recurrent Nets that Time and Count*)

LSTM Cell (Variation 2)

Key components: Cell state, Forget gate, Input gate and output gate



$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Torch example to understand parameters and shapes

```
input_size = 1    # The number of variables in your sequence data.
n_hidden   = 100  # The number of hidden nodes in the LSTM layer.
n_layers   = 2    # The total number of LSTM layers to stack.
out_size   = 1    # The size of the output you desire from your RNN.

lstm   = nn.LSTM(input_size, n_hidden, n_layers, batch_first=True)
linear = nn.Linear(n_hidden, 1)
```

Input output shapes

```
# 1. network input shape: (batch_size, seq_length, num_features)
# 2. LSTM output shape: (batch_size, seq_length, hidden_size)
# 3. Linear input shape:  (batch_size * seq_length, hidden_size)
# 4. Linear output: (batch_size * seq_length, out_size)
```

https://towardsdatascience.com/lstms-in-pytorch-528b0440244

Caution in PyTorch Implementation

```python
import torch

lstm = torch.nn.LSTM(input_size = 1, hidden_size = 4, num_layers = 1)
for name, param in lstm.named_parameters():
    print(name, param.shape)
```

```
weight_ih_l0 torch.Size([16, 1])
weight_hh_l0 torch.Size([16, 4])
bias_ih_l0 torch.Size([16])
bias_hh_l0 torch.Size([16])
```

Questions:

1. What are weight_ih and weight_hh?
2. How to interpret the dimensions?
3. Which version of LSTM is this?
4. How should you use initialization (e.g. Xavier, Kaiming)?

Caution in PyTorch Implementation

```python
import torch

lstm = torch.nn.LSTM(input_size = 1, hidden_size = 4, num_layers = 1)
for name, param in lstm.named_parameters():
    print(name, param.shape)
```

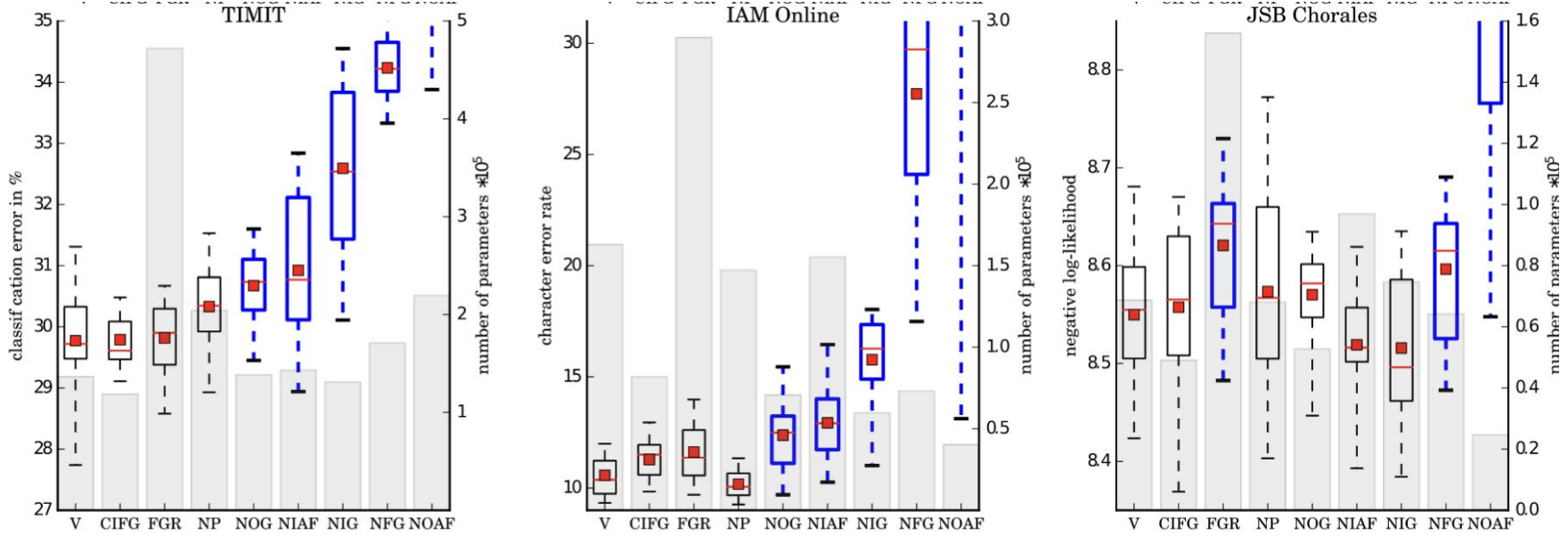
```
weight_ih_l0 torch.Size([16, 1])
weight_hh_l0 torch.Size([16, 4])
bias_ih_l0 torch.Size([16])
bias_hh_l0 torch.Size([16])
```

Questions:

1. What are weight_ih and weight_hh? Input weights and hidden weights
2. How to interpret the dimensions? Input, forget, cell, and output weights stacked (reference)
3. Which version of LSTM is this? Wikipedia version (no peephole connection)
4. How should you use initialization (e.g. Xavier, Kaiming)? We initialize each one of four (three if GRU) matrices separately

Performance per LSTM Component

(Greff et al. 2017: *LSTM: A Search Space Odyssey*)



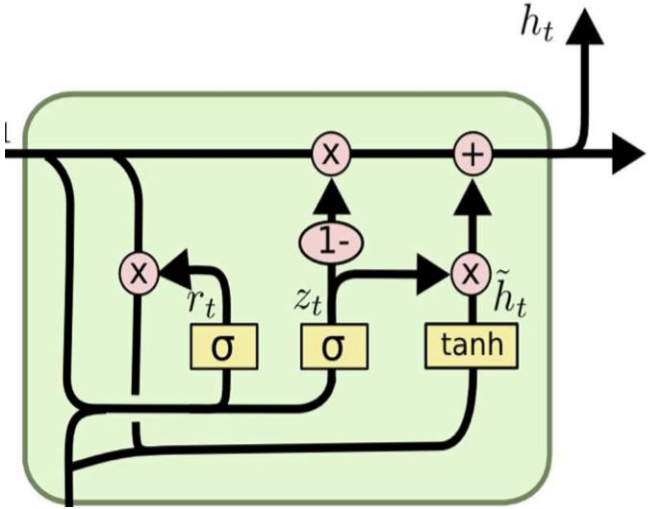
CIFG: GRU, NP: No peepholes, FGR: Full gate recurrence, NOG: No output gate, NIG: No input gate, **NFG: No forget gate**, NIAF: No input activation function, **NOAF: No output activation function**)

Performance per LSTM Component

Arch.	Arith.	XML	PTB
Tanh	0.29493	0.32050	0.08782
LSTM	0.89228	0.42470	0.08912
LSTM-f	0.29292	0.23356	0.08808
LSTM-i	0.75109	0.41371	0.08662
LSTM-o	0.86747	0.42117	0.08933
LSTM-b	0.90163	0.44434	0.08952
GRU	0.89565	0.45963	0.09069
MUT1	**0.92135**	**0.47483**	0.08968
MUT2	0.89735	**0.47324**	0.09036
MUT3	0.90728	0.46478	**0.09161**

(Jozefowicz et al. 2015: *An Empirical Exploration of Recurrent Network Architectures*)

GRU Cell



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

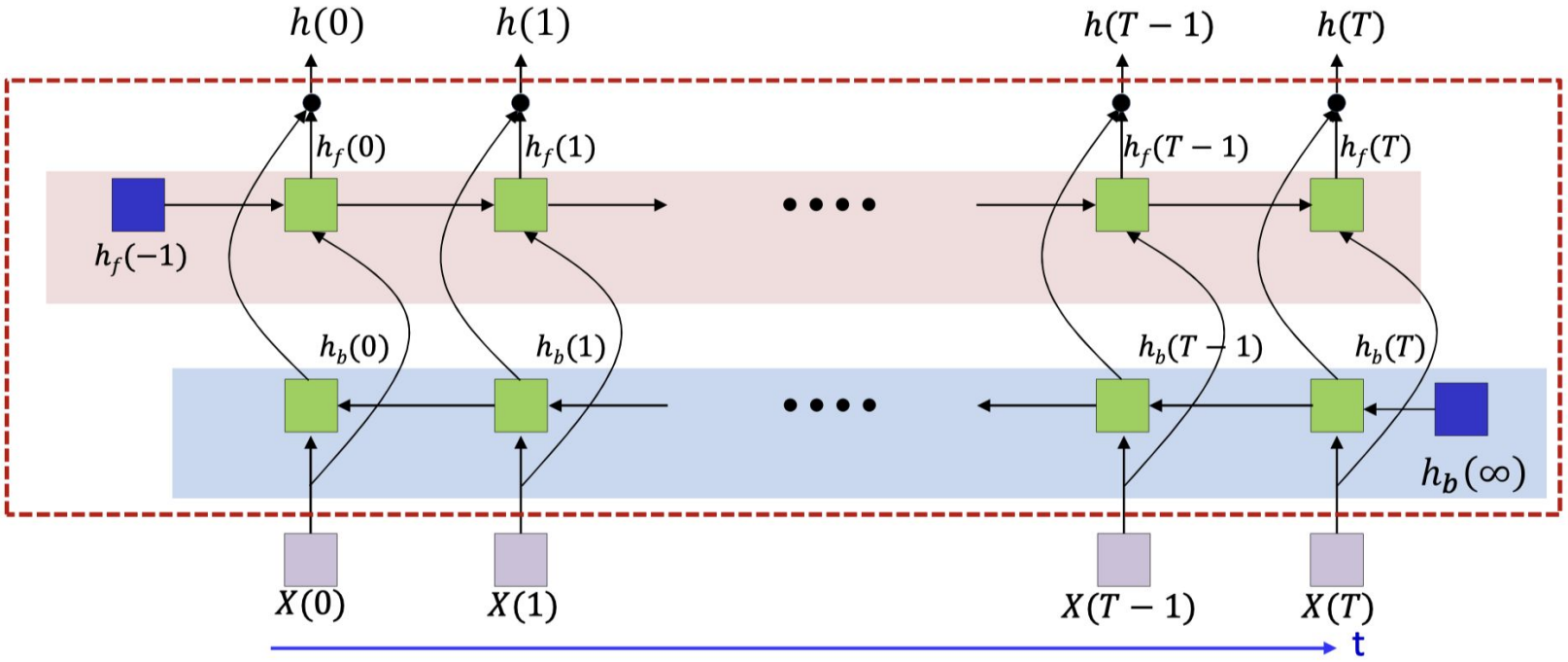
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

GRUs can't count! (Weiss et al. 2018: *On the Practical Computational Power of Finite Precision RNNs for Language Recognition*)

Bidirectional RNN



Actual Network with BRNNs: