Carnegie Mellon University

11-785 Introduction to Deep Learning

- Fall 2023 -

Recitation 8: RNN Basics

Shreyas Piplani & Meng Zhou

Objectives

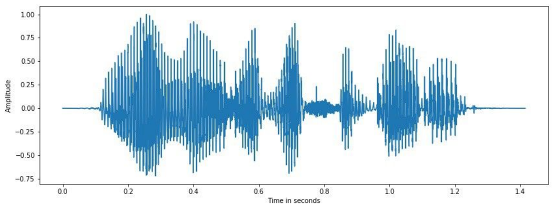
1. Understanding why we need RNNs
2. Understanding the working principles of RNNs
3. Implementation of LSTMs in PyTorch
4. Variants of RNNs
5. RNNs from a Perspective of Graphical Models
6. Normalization/Dropout in RNNs

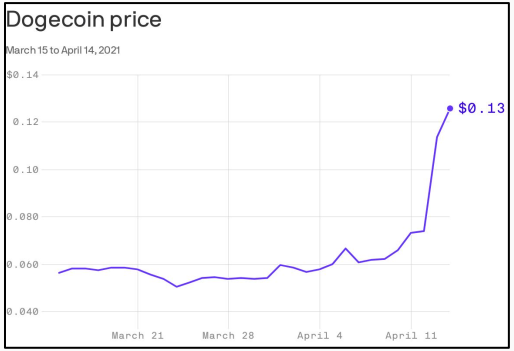
Why Recurrent Neural Network (RNNs)?

- RNNs learn the sequential characteristics in data inputs and makes predictions of the next possible outcomes
- RNNs have the ability to process temporal information present in sequential data
- RNNs have hidden states which act as the memory of the neural network which remembers information on data sequence

Sequential Data

- Consecutive data inputs which are dependent on each other
- Data input n is dependent on n-1 and n-1 dependent on n-2…
- Example:
    - Text: (Where are you off to?)
    - Audio/speech
    - Video
    - Time Series data (stock price, weather data)





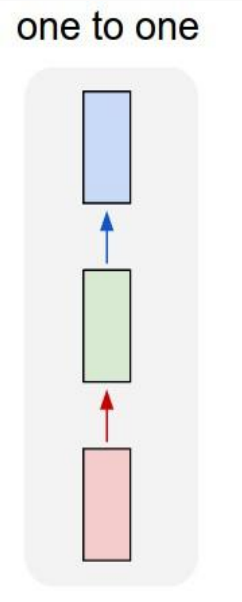
Application of RNN

- Intelligent stock prediction systems
- Machine translation
- Speech Recognition systems
- Language modeling and Text generation
- Video Tagging
- Image captioning
- Text Summarization

Software applications: Google translate, Trading bots, Siri, Cortana, voice search

Data Modeling

Types of Recurrent Neural Network



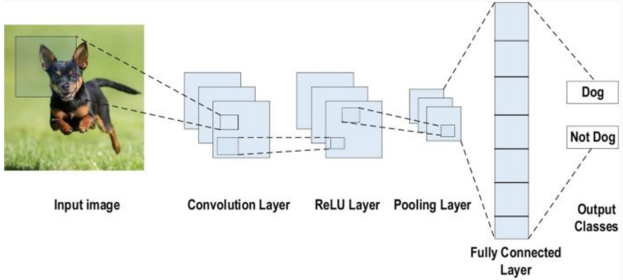
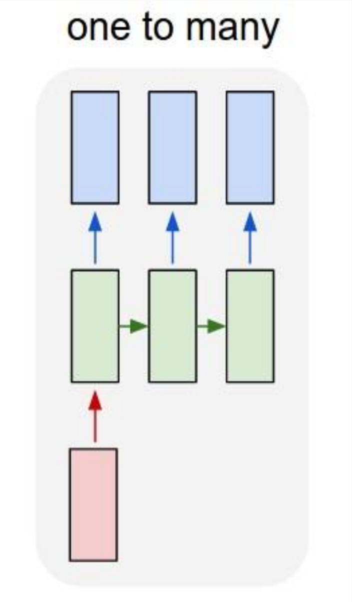


Image Classification [(ref)](ref)

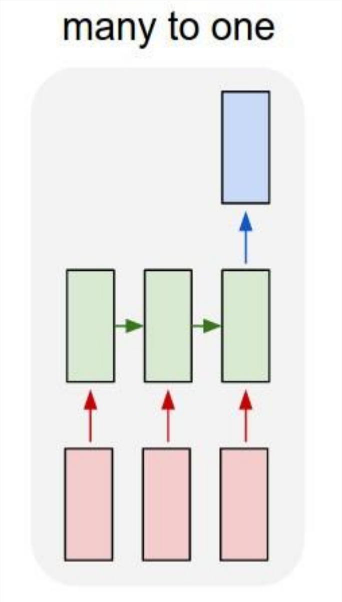


"man in black shirt is playing guitar."

Image Captioning [(ref)](ref)

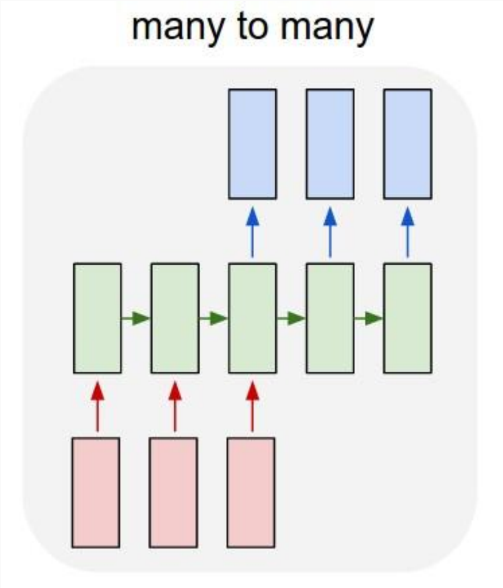
Data Modeling

Types of Recurrent Neural Network



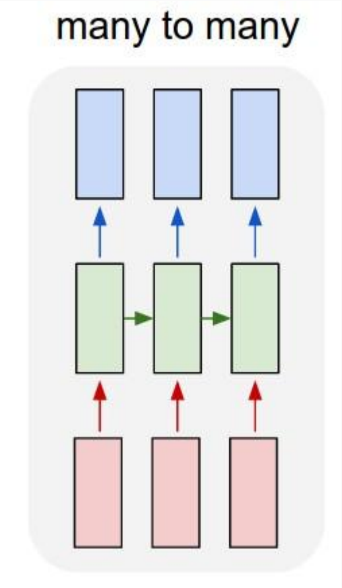
Sentiment Analysis (Movie Review)

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



Machine Translation

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

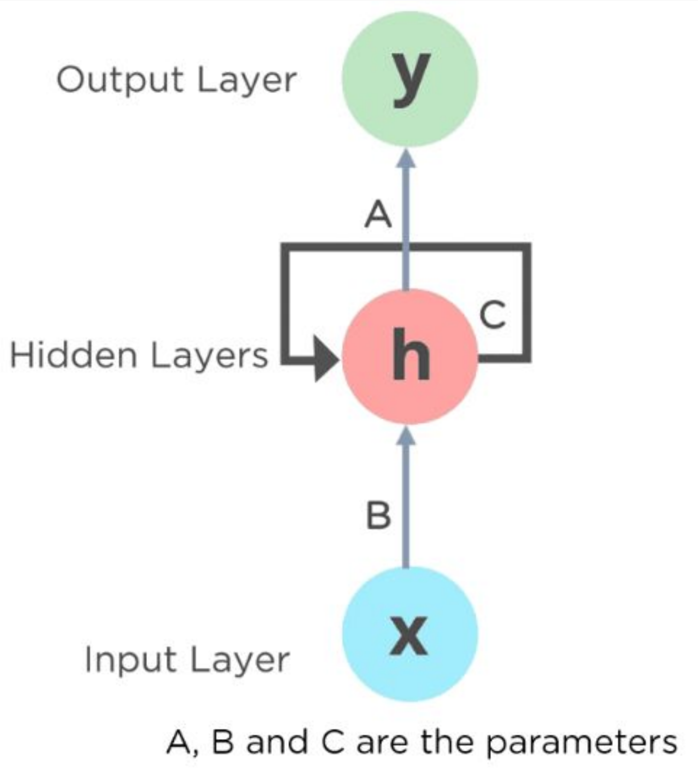


Object Tracking in videos

Video

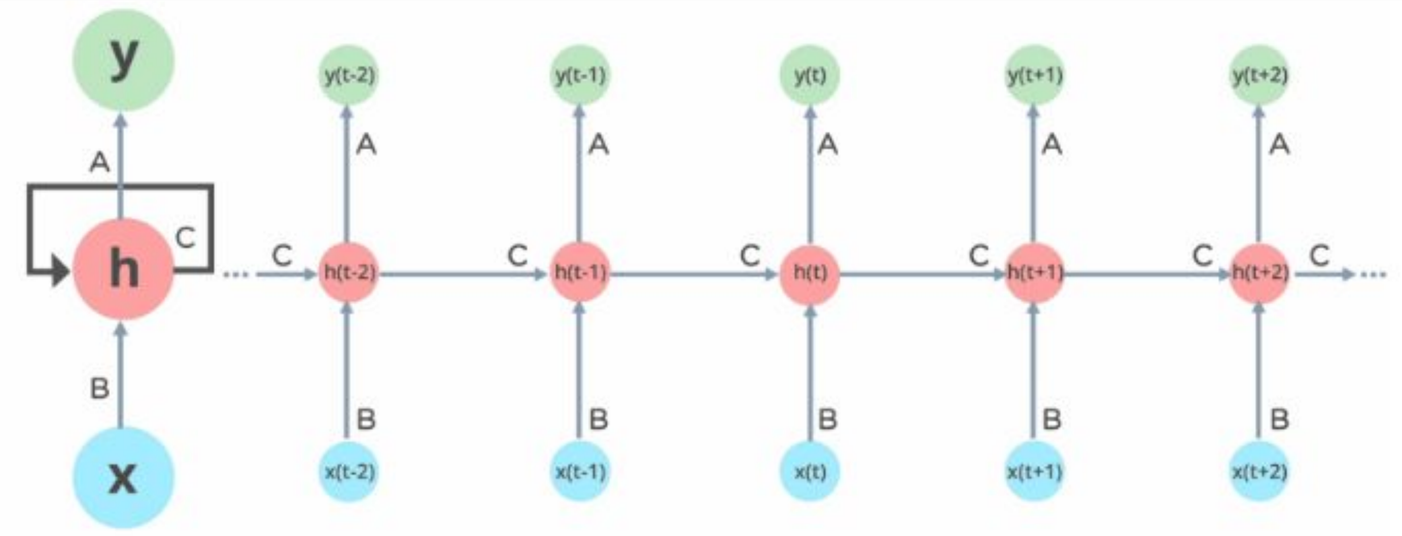
Recurrent Neural Networks

- Feeds the output of previous layer without activation to the input of next layer
- Looping Network (output of network depend on previous input of the sequence)
- Parameter sharing across timesteps
- Derivatives is aggregated across all the timesteps
- Backpropagation through time (BPTT)



Recurrent Neural Networks

At any given time t, the current input is a combination of input at x(t) and the output from the previous hidden layer h(t-1)



Problems with RNN

Vanishing gradient

Gradient back propageted through time becomes too small and loose information

Potential solutions:

- Weight initialization
- Choosing the right activation
- Long Short-Term Memory Networks (LSTMs)

Exploding gradient

Gradient tends to grow exponentially instead of decaying and cannot be contained

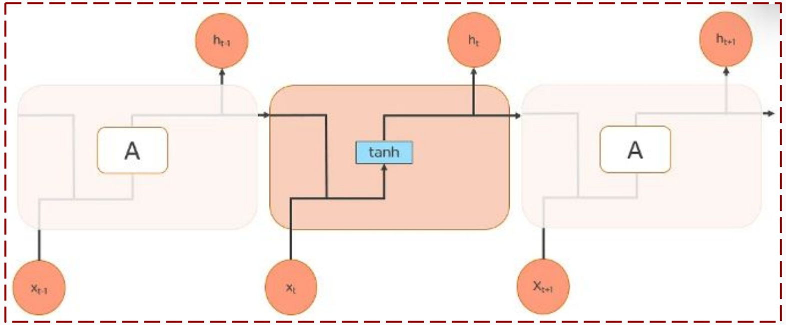
Potential solutions:

- Identity Initialization
- Truncated backpropagation
- Gradient clipping

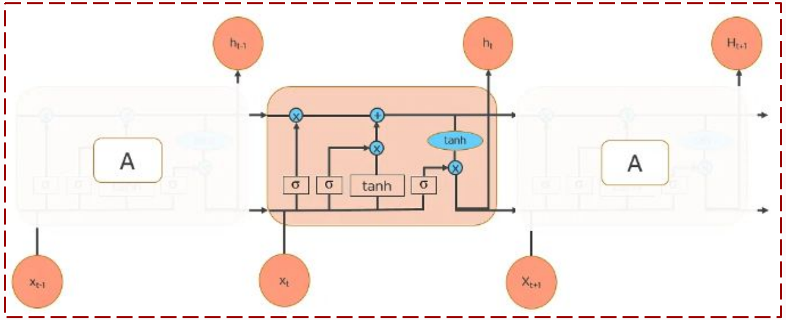
To avoid these gradient problems we need to decide the amount of information we would need to retain for prediction

Long Short-Term Memory (LSTMs)

- Capable of retaining information over a long period of time
- The most popular and efficient way of dealing with gradient problems
- LSTMs have chain-like structure repeating each cell
- Each LSTM cell have a defined structure depending on the variant
- LSTM Variants: +> LSTM Classic, LSTM Peephole connection
- Gated Recurrent Unit (GRU)

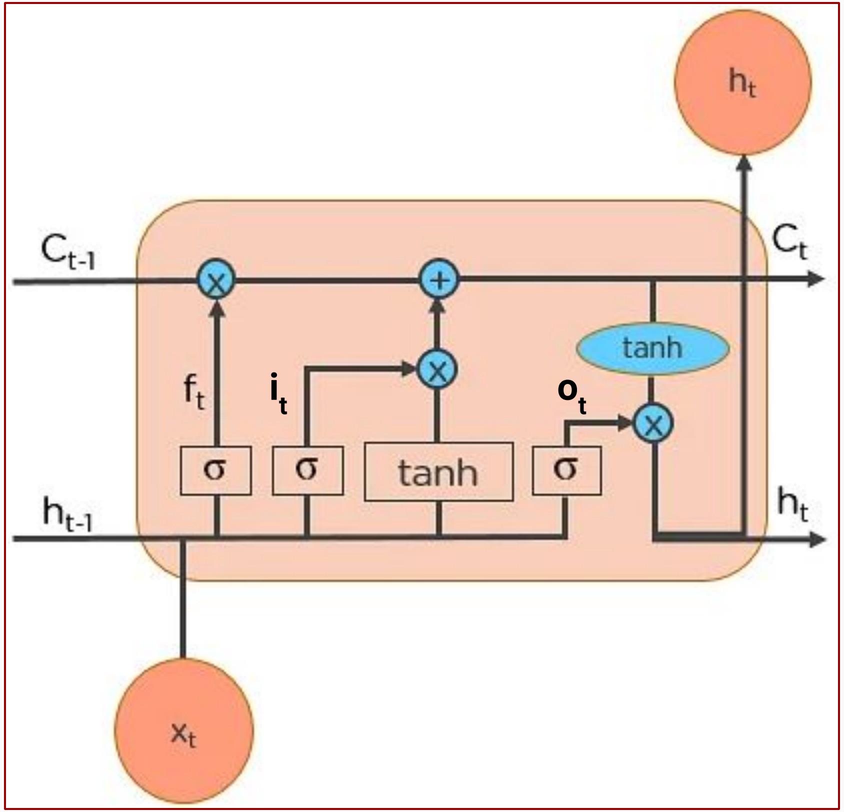


RNN



LSTM

Classic LSTM



Gates

Forget gate $f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$

Input gate $i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$

Output gate $o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$

Variables

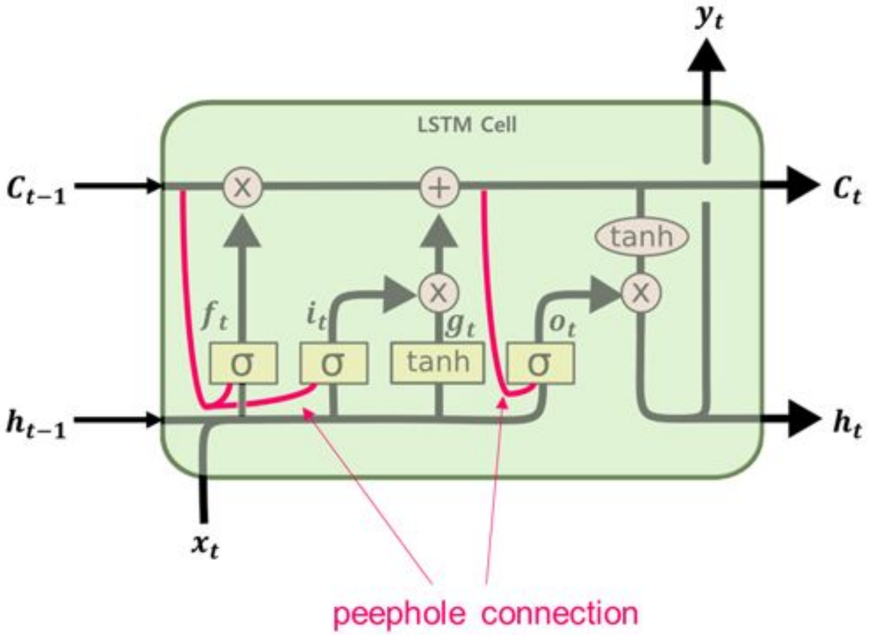
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

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

Peephole LSTM



Gates

Forget gate $f_t = \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right)$

Input gate $i_t = \sigma\left(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i\right)$

Output gate $o_t = \sigma\left(W_o \cdot [C_t, h_{t-1}, x_t] + b_o\right)$

Variables

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

$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$

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

LSTM PyTorch Implementation

```
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)
```

```
# 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)
```

Caution 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 PyTorch Implementation

```
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])
```

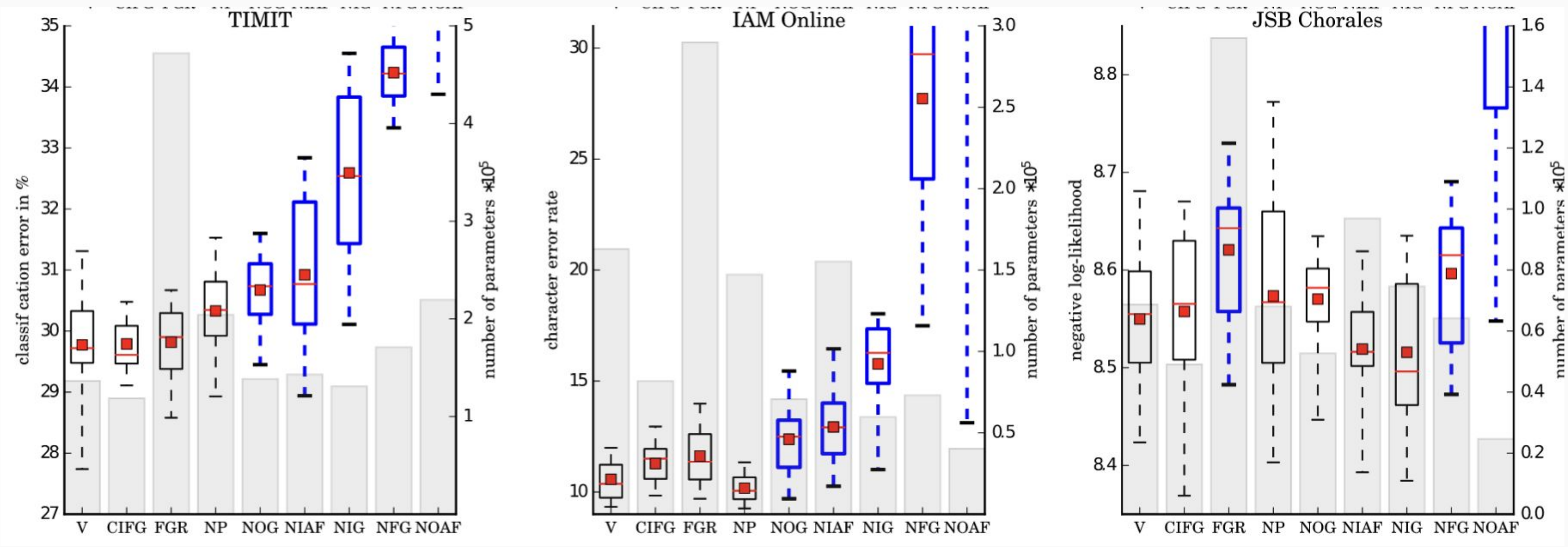
$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

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*)



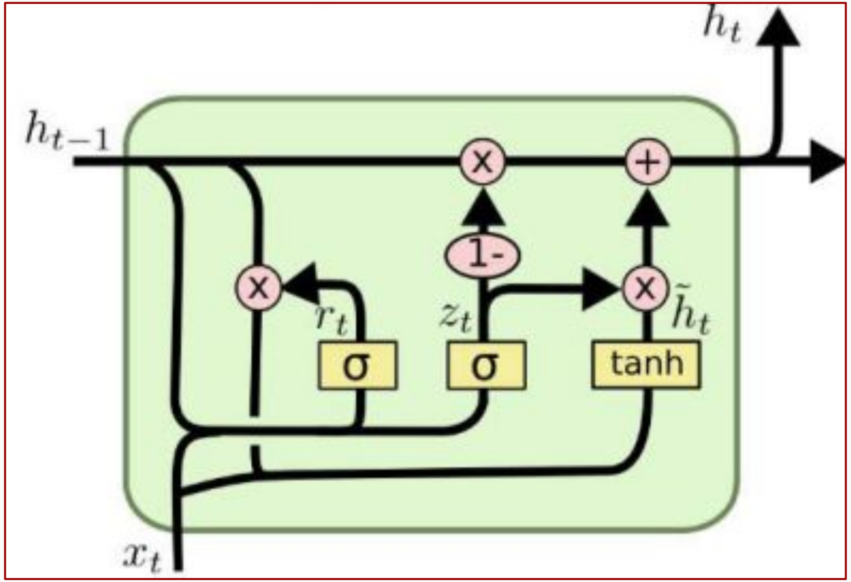
CIFG: GRU, NP: No peepholes, FGR: Full gate recurrence, NOG: No output gate, NIG: No input gate, **NFG: No forget gate**, NIAF: No input activation function, **NOAF: No output activation function**)

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*)

Gated Recurrent Unit (GRU)



Gates

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

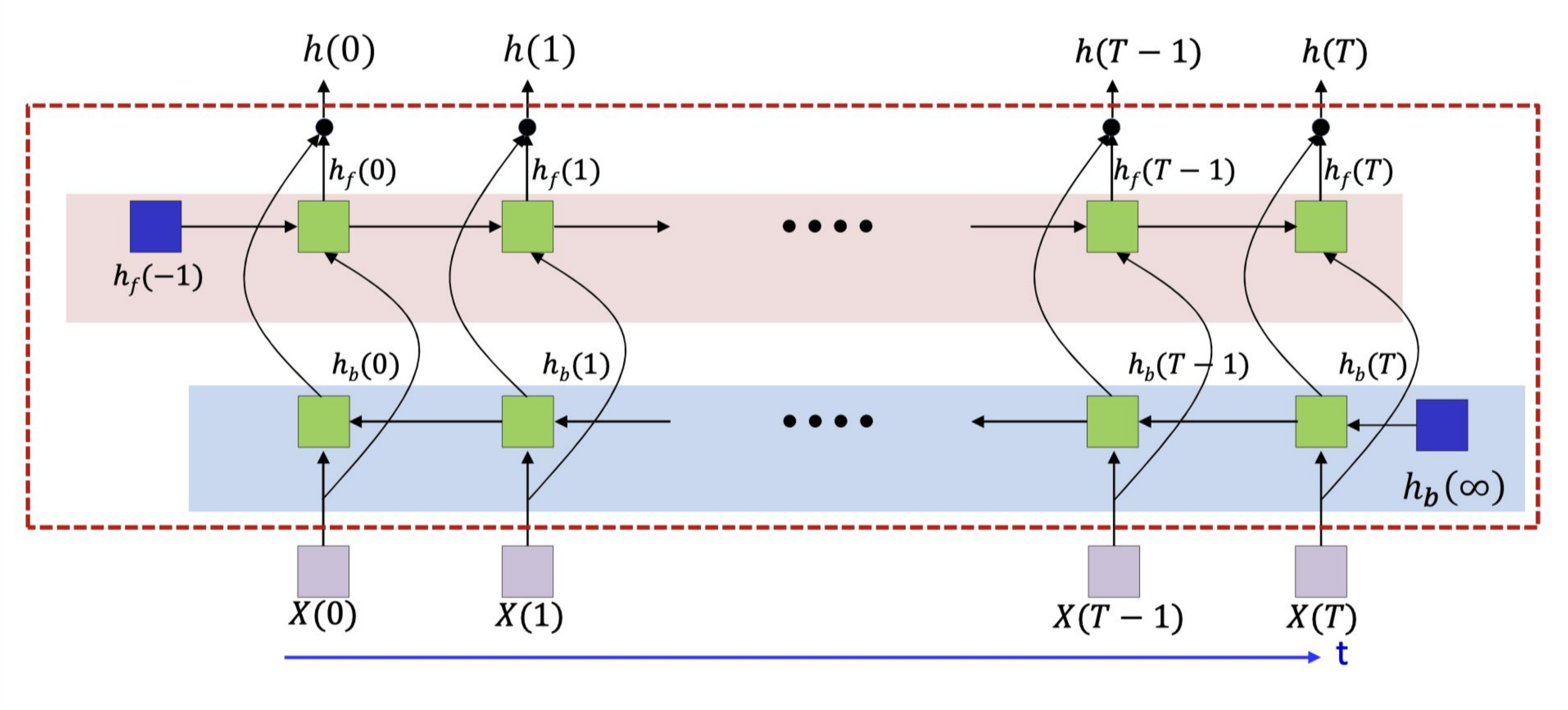
update gate $z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$

Variables

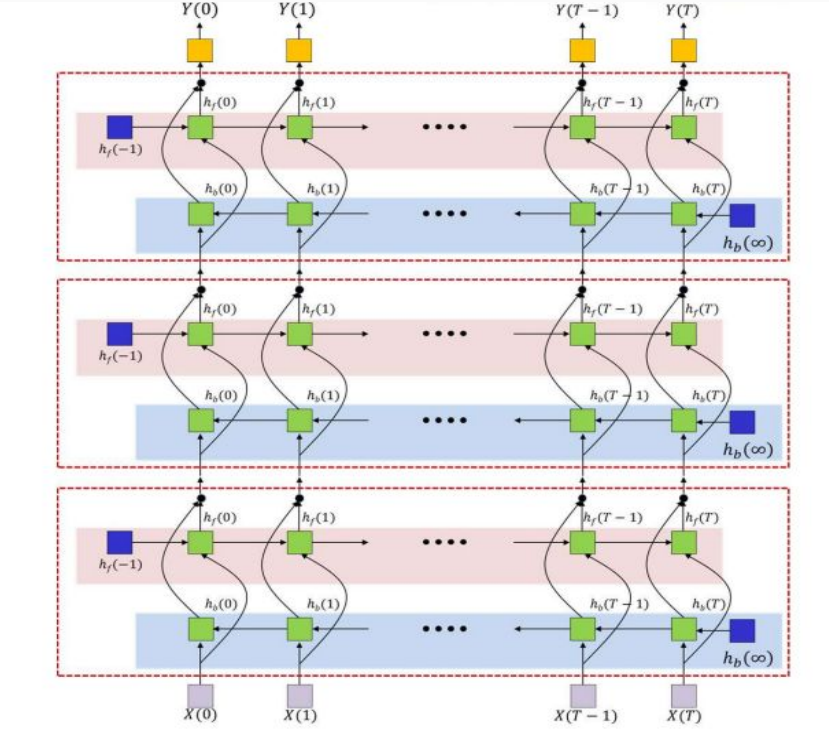
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

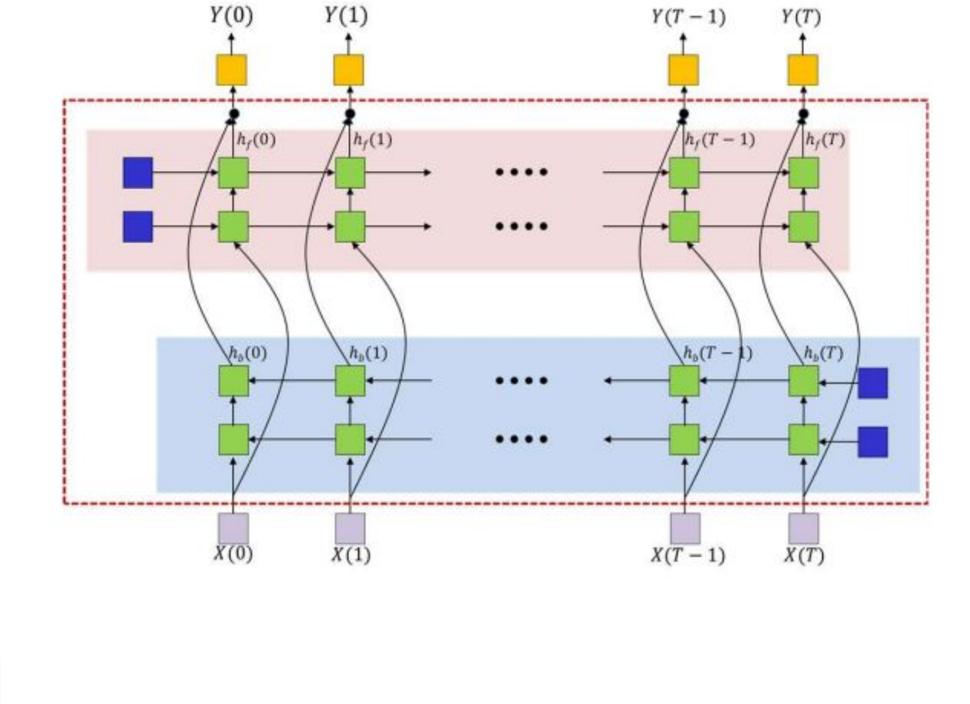
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Bidirectional RNN



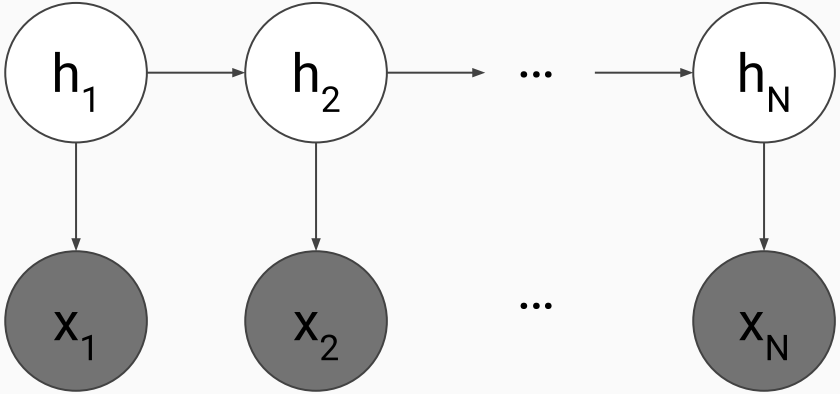
Actual Network with BRNNs





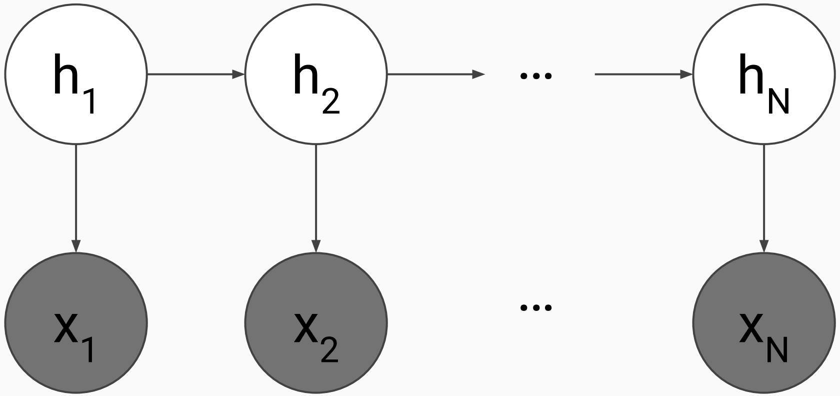
RNNs: A Perspective from Graphical Models

- Recall: Hidden Markov Models (HMMs)
  - A sequence of hidden variables h_i and a sequence of observations x_i
  - Conditional independence
  - The hidden variable h is discrete
- Was very popular for sequential tasks



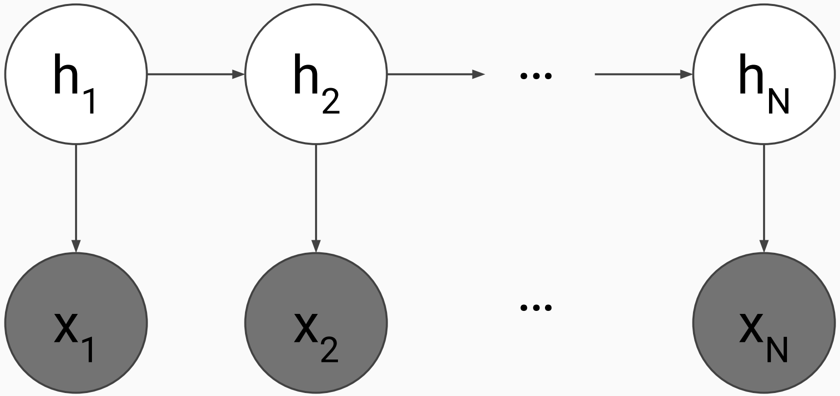
RNNs: A Perspective from Graphical Models

- State Space Model (SSMs): “continuous” version of HMMs
  - More complex than it sounds like
  - Implications: different inference algorithms
- A specific instance of SSMs: linear dynamical system
  - $h_t = Ah_{t-1} + Bw_{t-1}$ (Transition)
  - $X_t = Ch_t + v_t$ (Observation)
  - A and C are two matrices
  - w_t and v_t are two noise terms



RNNs: A Perspective from Graphical Models

- State Space Model
  - $h_t = Ah_{t-1} + Bw_t$ (Transition)
  - $X_t = Ch_t + v_t$ (Observation)
- An estimator of the true hidden variables (Kalman Filter)
  - $h_t = A'h_{t-1} + B'X_t$
- What about a non-linear estimator?
  - $h_t = f(h_{t-1}, X_t)$
  - We get RNN!



RNNs + Normalization?

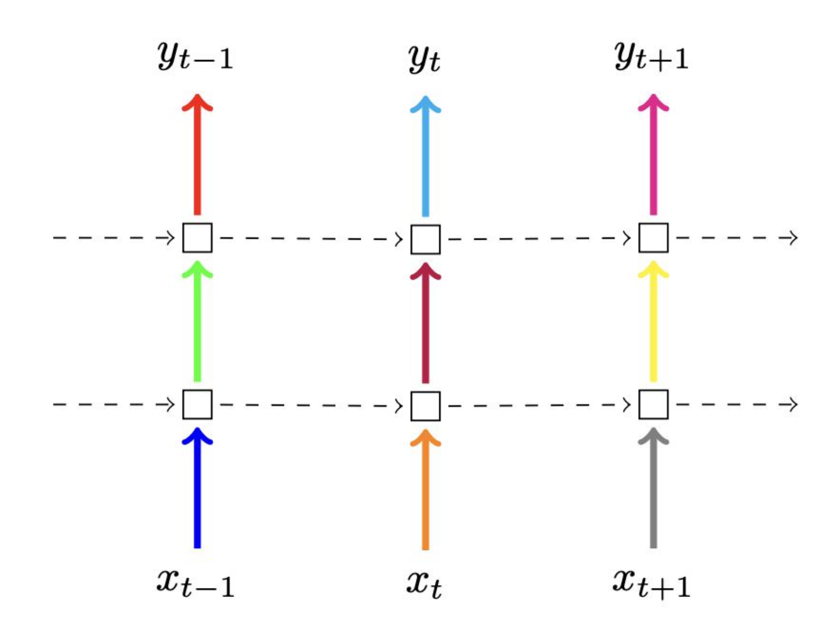
- Applying BN to RNNs
  - The statistics could be different for different time steps
  - Typically does not work well
- Recurrent Batch Normalization [1]
  - Separate BN for recurrent term and input term
  - Separate statistics for different time step
    - But what if the length of test data is longer than all of the training data?
  - Initialization of gain matters
- Layer Normalization [2]
  - Proposed for RNNs but still not very common for RNNs
  - Widely used in other scenarios, e.g. Transformers

[1] Cooijmans, Tim, et al. "Recurrent batch normalization." arXiv preprint arXiv:1603.09025 (2016).

[2] Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." arXiv preprint arXiv:1607.06450 (2016).

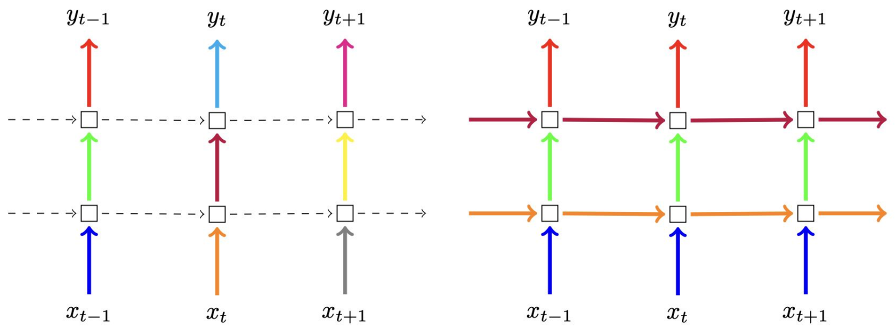
RNNs + Dropout?

- `dropout` in PyTorch nn.LSTM
    - No effect if num_layers=1
    - Most common
- Two Types of Connections in RNNs
    - Layer-to-Layer
    - Hidden-to-Hidden



Variational RNNs, a.k.a. Locked Dropout

- Locked Dropout [1]
  - Key idea: keep the dropout mask fixed
  - Note: the implementation in torchnlp is slightly different
    - Only applied to layer-to-layer connection



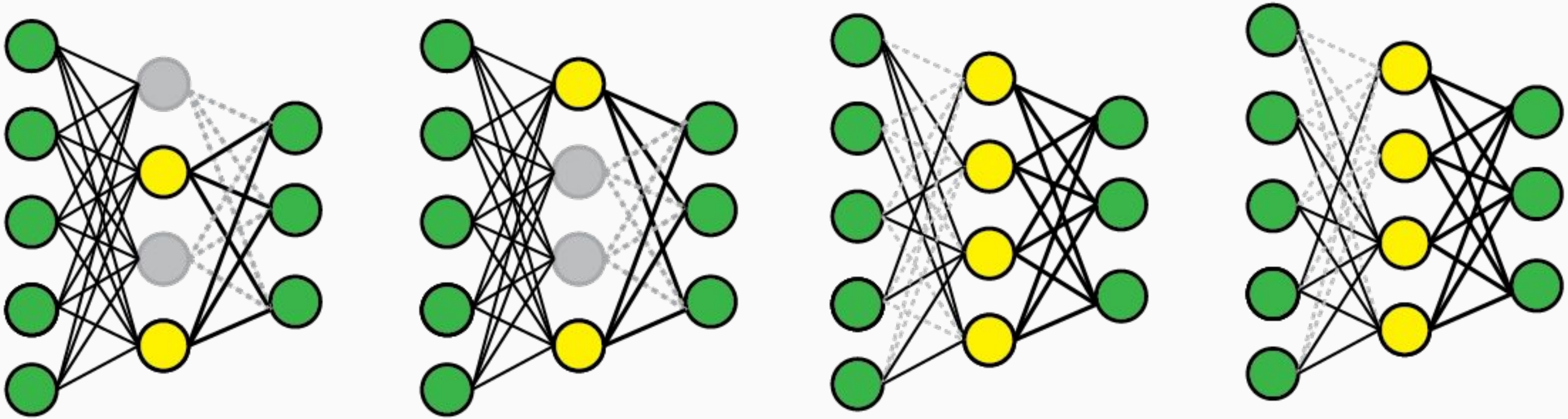
(a) Naive dropout RNN (b) Variational RNN

[1] Gal, Yarin, and Zoubin Ghahramani. "A theoretically grounded application of dropout in recurrent neural networks." Advances in neural information processing systems 29 (2016).

Weight-dropped LSTM

- Weight-dropped LSTM [1]
    - Essentially, “DropConnect”. Dropout applied to weight matrices

Dropout DropConnect



[1] Merity, Stephen, Nitish Shirish Keskar, and Richard Socher. "Regularizing and optimizing LSTM language models." arXiv preprint arXiv:1708.02182 (2017).

Language Model Demo: Shakespeare