Natural Language Processing – Recurrent Neural Network

Instructor: Dr. Muhammad Fahim

INNOPOLIS UNIVERSITY

Contents

- Natural Language Processing (NLP)
- Recurrent Neural Network (RNN)
- RNN – Architecture
- RNN – Training
- Use cases of RNN
- Variants of RNN
- Recursive Neural Network
- Summary

Natural Language Processing (NLP)

Natural Language Processing focuses on the interactions between human language and computers.

NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way.

Natural Language Processing (NLP)

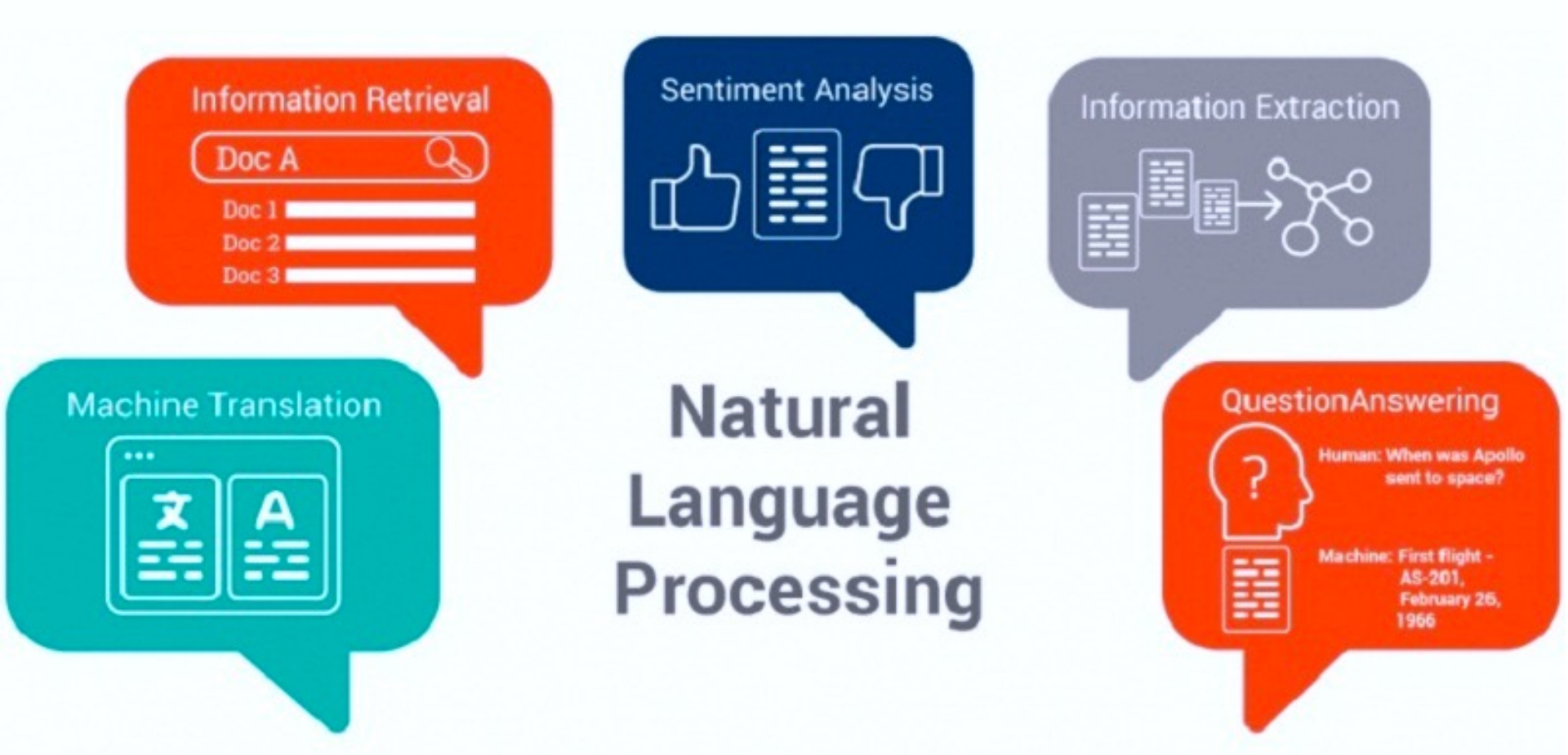


Image Source: http://m.el-dosuky.com/course.php?c=natural-language-processing

Recurrent Neural Network (RNN)

- The idea behind RNNs is to make use of sequential information.
- If the next word in a sentence is going to be predicted, there is the need know which words came before it.
- RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations.
- Another way to think about RNNs is that it has a memory which captures information about what has been calculated so far.

Recurrent Neural Network (RNN)

- Technically, an RNN models sequences

- What is a sequence?
  - Time series
  - Languages
  - Speech
  - Many more…

- RNN successfully generate
  - TED Talks https://www.youtube.com/watch?v=-OodHtJ1saY&feature=youtu.be&t=31s
  - Eminem rapper https://soundcloud.com/mrchrisjohnson/recurrent-neural-shady
  - Music https://web.stanford.edu/class/cs224n/reports/2762076.pdf

Model Sequences

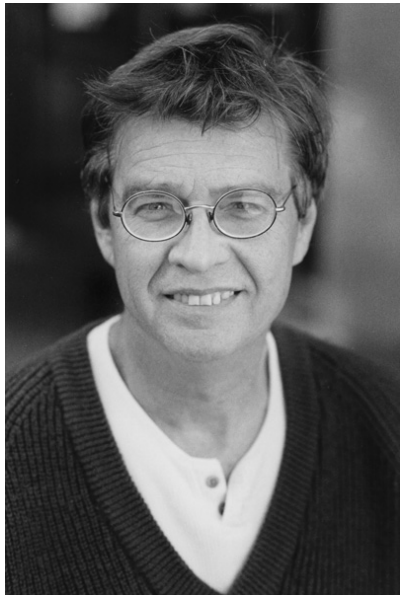
- To model sequences, we need:
  1. To deal with variable-length sequences
  2. To maintain sequence order
  3. To keep track of long-term dependencies
  4. To share parameters across the sequence

Recurrent Neural Network

RNN Background – 1986

- A psychology professor who studied how people think and learn complex skills such as reading and the use of language

"David was interested in how we're able to bring thoughts together in our minds”



David Rumelhart

Recurrent Neural Network (RNN)

- RNN is specialized for processing a sequence of values

$$\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$$

- It would be practical for networks without sequence-based specialization.

- Most recurrent networks can also process sequences of variable length

Sequence of Variable Length

- One of the early ideas found in machine learning and statistical models (1980s)

Sharing parameters across different parts of a model

- **Parameter sharing**
  - Makes it possible to extend and apply the model to examples of different forms (different lengths, here) and generalize across them.

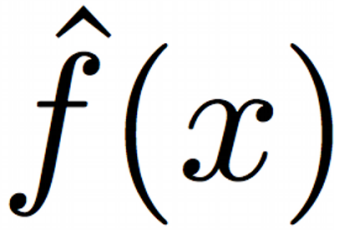
Importance of Parameter Sharing!

- **For Example:**

"I went to Nepal in 2009"

"In 2009, I went to Nepal."

Which year narrator went to Nepal?



Machine Learning Model

2009

- We would like it to recognize the year as the relevant piece of information, whether it appears in the sixth word or in the second word of the sentence

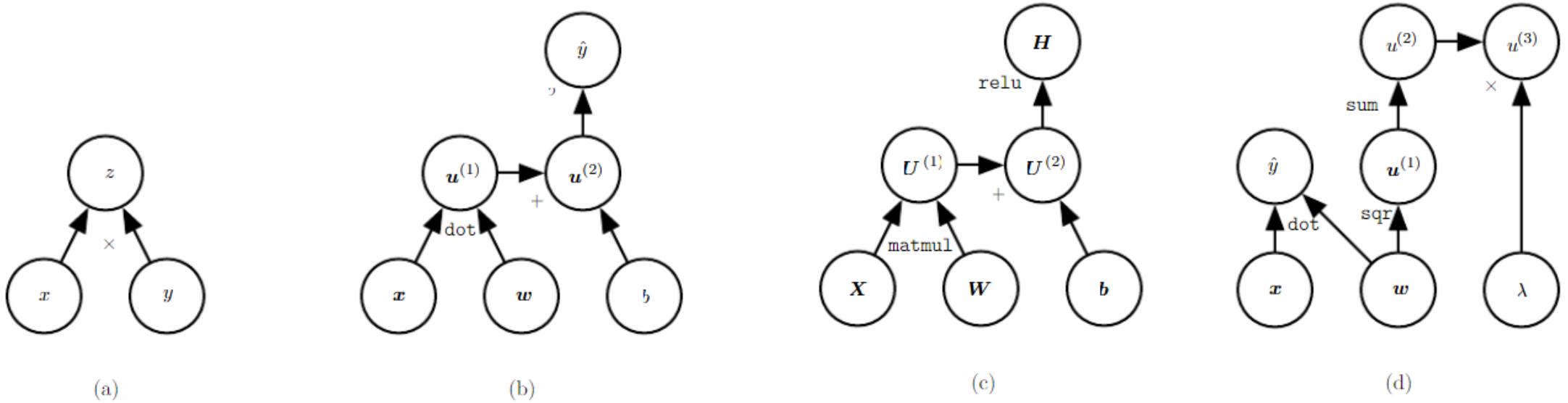
- What happens when you trained a feedforward neural network?

Parameter Sharing in RNN

- Each member of the output is a function of the previous members of the output.
- Each member of the output is produced using the same update rule applied to the previous outputs.
- This recurrent formulation results in the sharing of parameters through a very deep computational graph

Computational Graph (CG)

- CG is a way to formalize the structure of a set of computations – those involved in mapping inputs and parameters to outputs and loss.



$$z = xy$$

$$\hat{y} = \sigma\left(\boldsymbol{x}^\top \boldsymbol{w} + b\right)$$

$$\boldsymbol{H} = \max\{0, \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}\}$$

A computation graph that applies more than one operation to the weights w

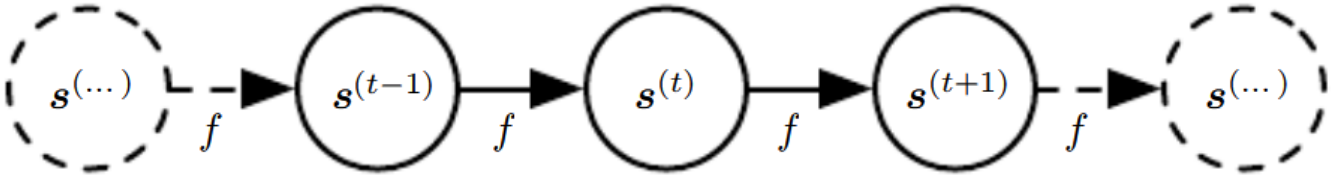
Familiarization with Recurrent Process

- Let's consider a dynamical system

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

- For Example τ = 3 time steps

$$\begin{aligned} \boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}) \end{aligned}$$



Another Example

- Let's consider another dynamical system driven by an external signal $x^{(t)}$

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

- Recurrent neural networks use above or a similar equation to define the values of their hidden units

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

  - **Where**
    - $h^{(t)}$ is the hidden state at time t (a vector)
    - $x^{(t)}$ is input vector at time t
    - $\boldsymbol{\theta}$ is the parameters of f

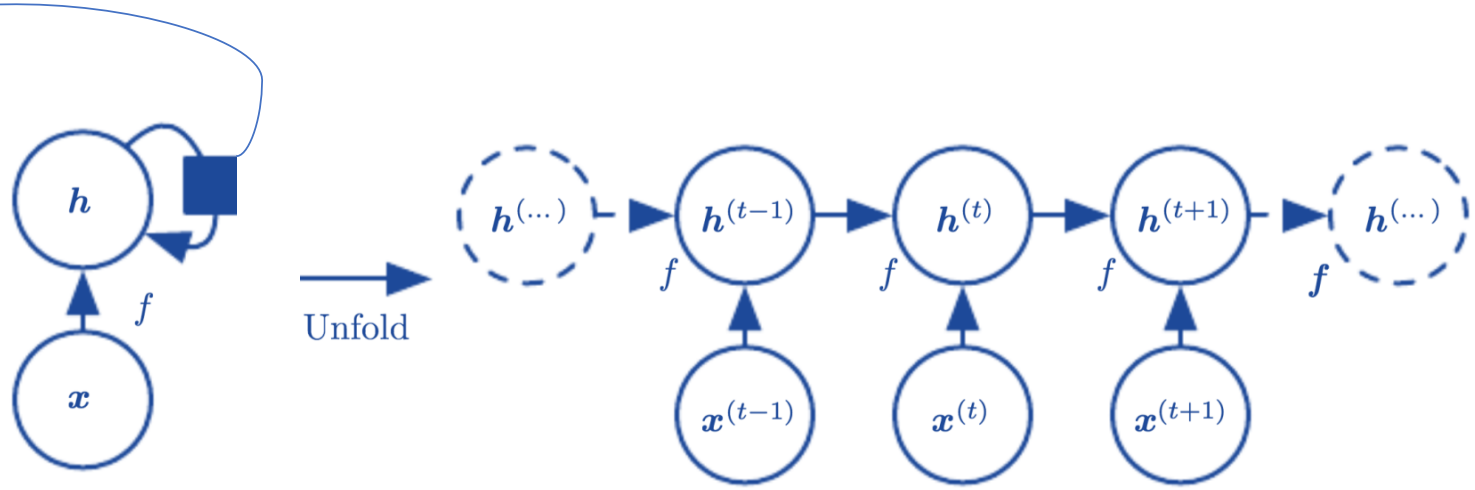
  - **$\boldsymbol{\theta}$ remains constant as t changes. (why?)**

Reason: Apply the **same function** with the same **parameter values** at **each iteration**.

RNN with No Outputs

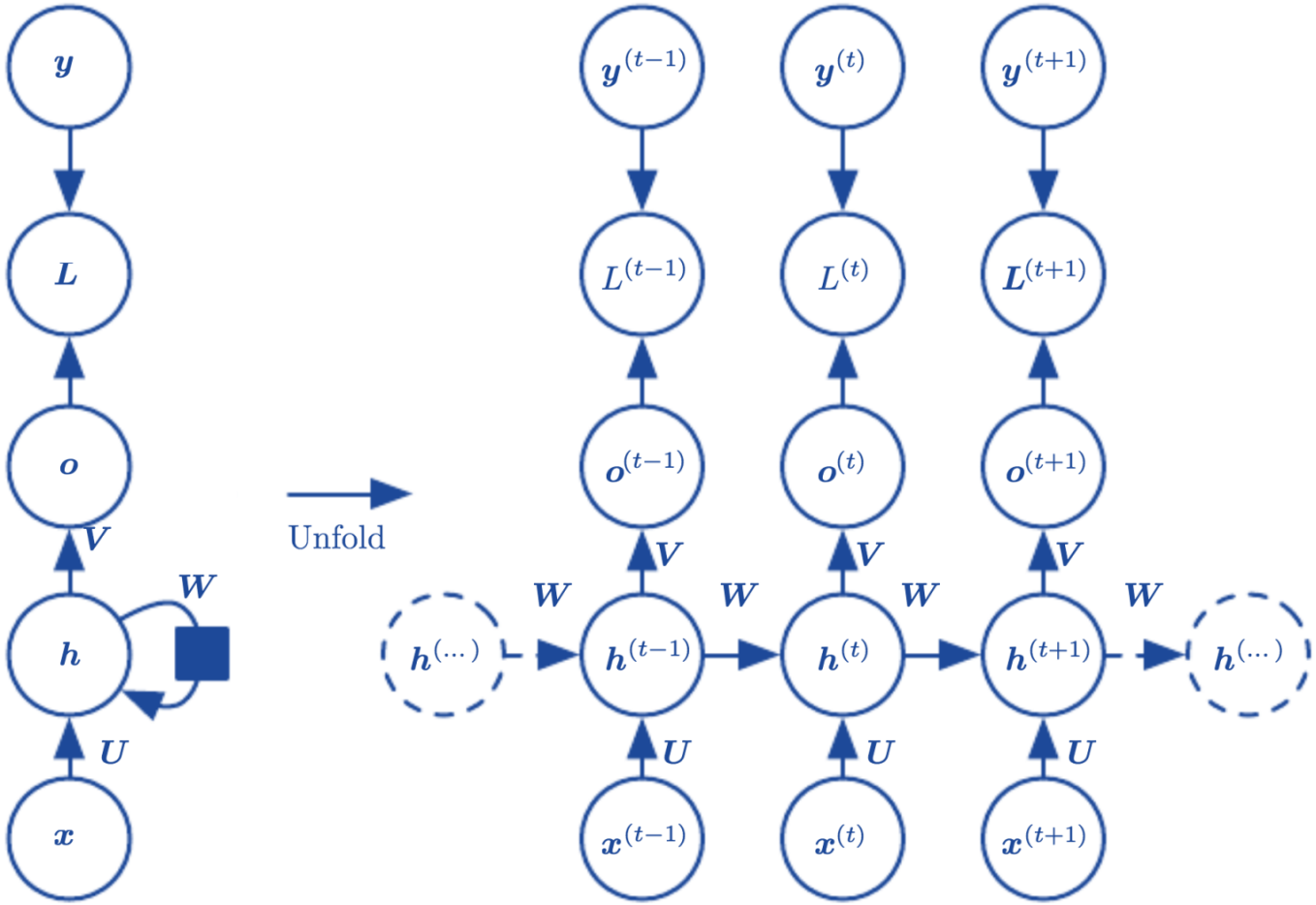
$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

Delay of time step 1.



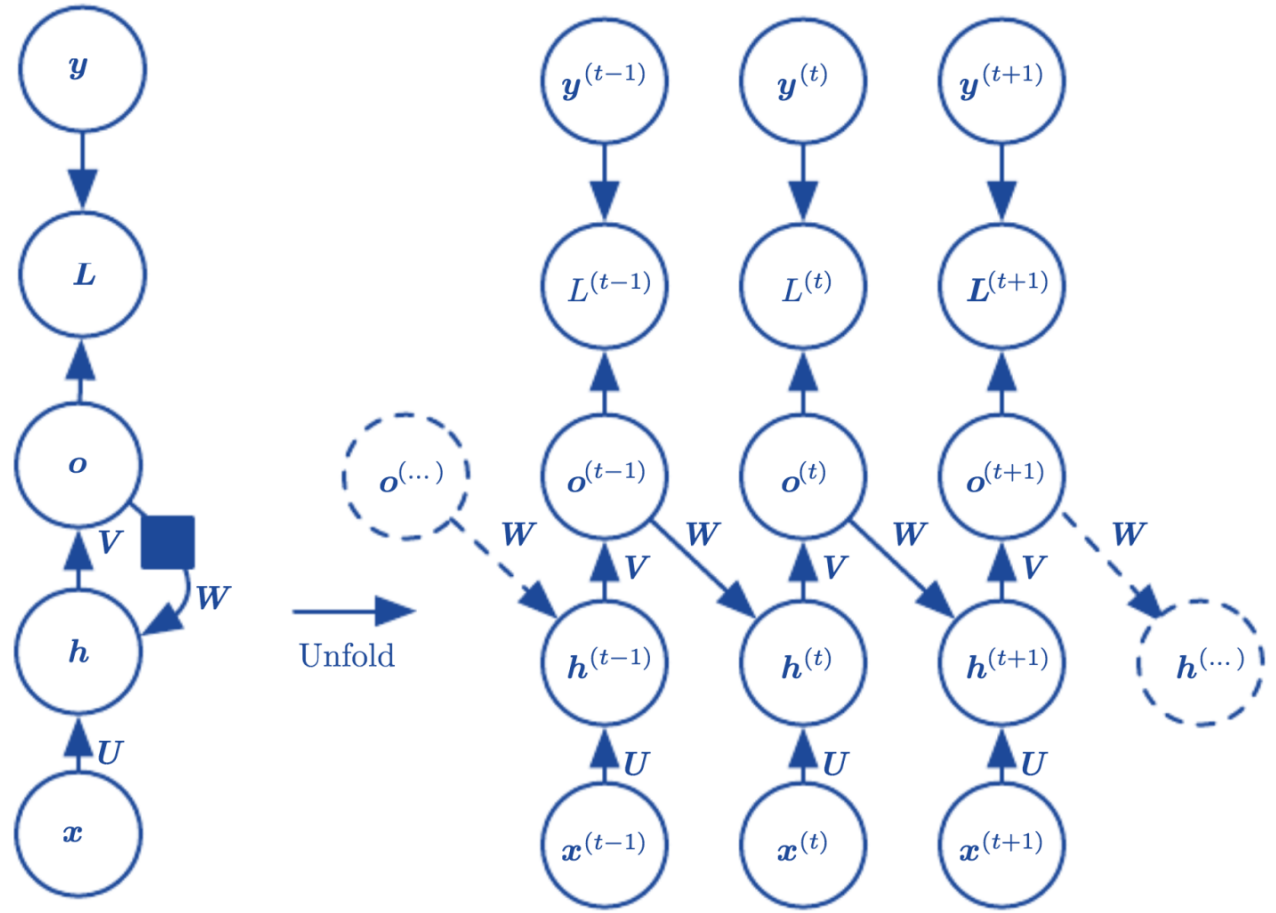
A recurrent network with no outputs

Design Patterns for RNN – 1/3



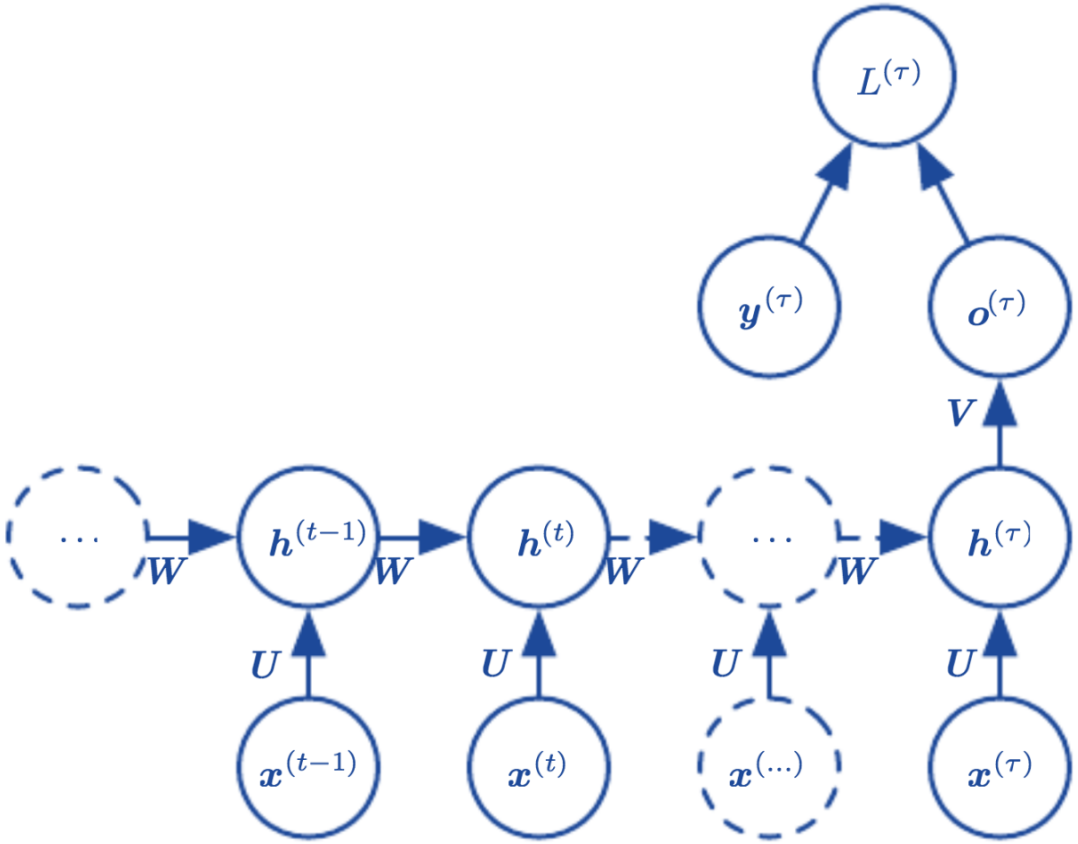
Recurrent networks that produce an output at each time step and have recurrent connections between hidden units

Design Patterns for RNN – 2/3



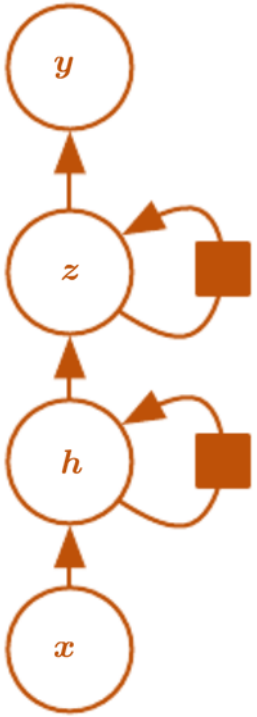
Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step

Design Patterns for RNN – 3/3



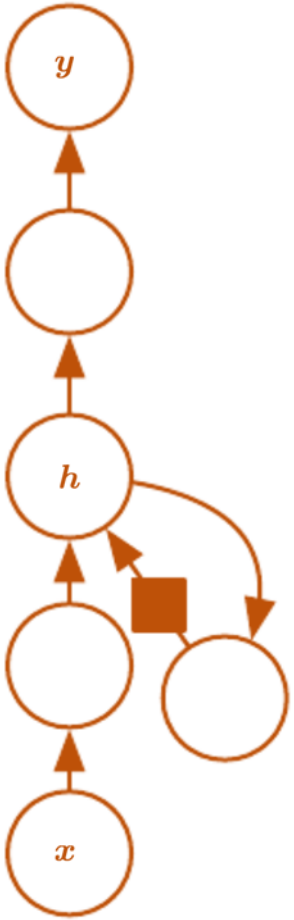
Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output

Deep RNN – 1/3



The hidden recurrent state can be broken down into groups organized hierarchically.

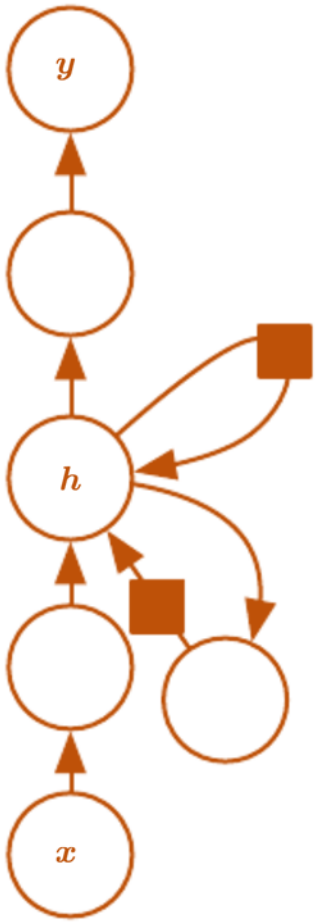
Deep RNN



Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden, and hidden-to-output parts.

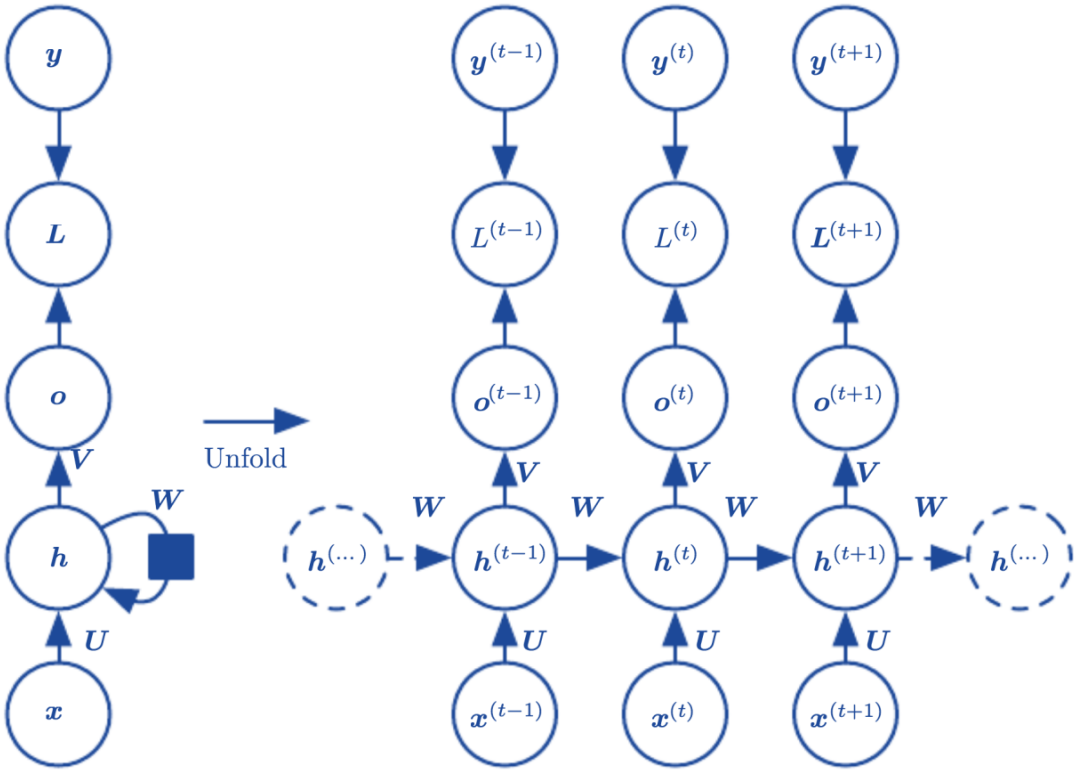
This may lengthen the shortest path linking different time steps.

Deep RNN



The path-lengthening effect can be mitigated by introducing skip connections.

RNN – Equations!

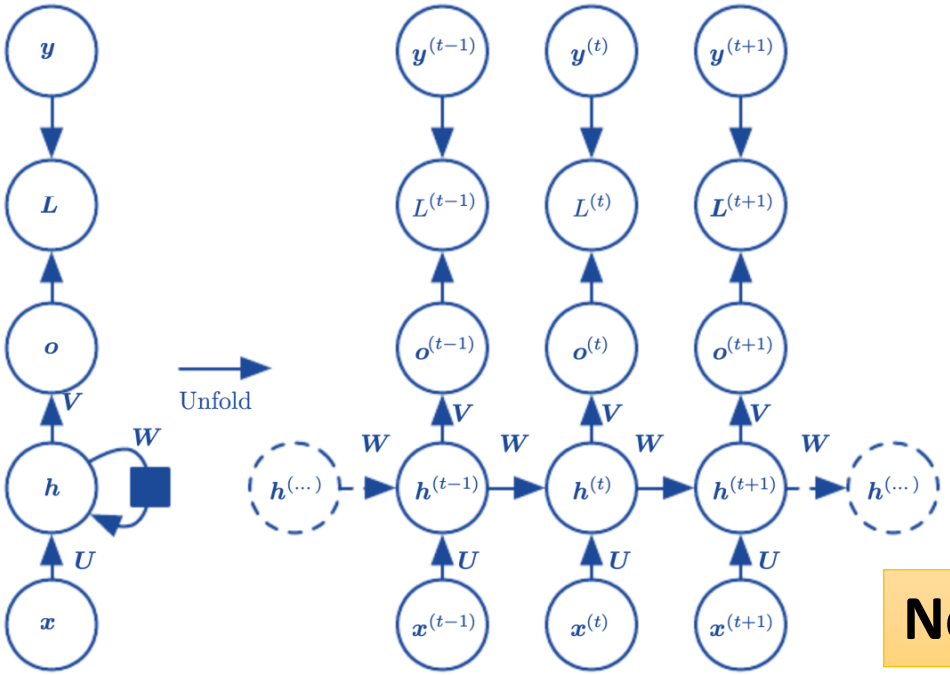


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

Network's input

- h^0 initial hidden state has size **m × 1** (assumed given)
- x^t input vector at time t has size **d × 1**

Network's Output and Hidden vectors



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

Network's Output and Hidden vectors

- a^t hidden state at time t of size **m x 1** before non-linearity
- h^t hidden state at time t of size **m x 1**
- o^t output vector at time t of size **C x 1**
- p^t output porbability vector at time t of size **C X 1**

Parameters of the Network

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

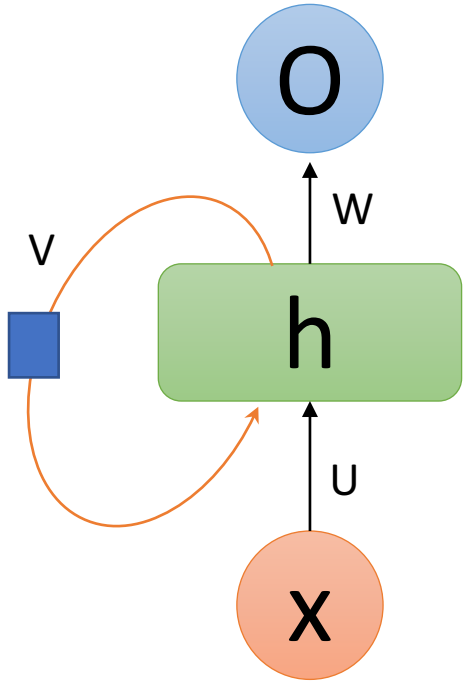
Parameters of the Network

- ***W*** weight matrix of size **m x m** applied to $\mathbf{h}^{t-1}$ (hidden-to-hidden connection)
- ***U*** weight matrix of size **m x d** applied to $\mathbf{x}^{t}$ (input-to-hidden connection)
- ***b*** bias vector of size **m x 1** in equation for $\mathbf{a}^{t}$
- ***V*** weight matrix of size **C x m** applied to $\mathbf{a}^{t}$ (hidden-to-output connection)
- ***c*** bias vector of size **C x 1** in equation for $\mathbf{o}^{t}$

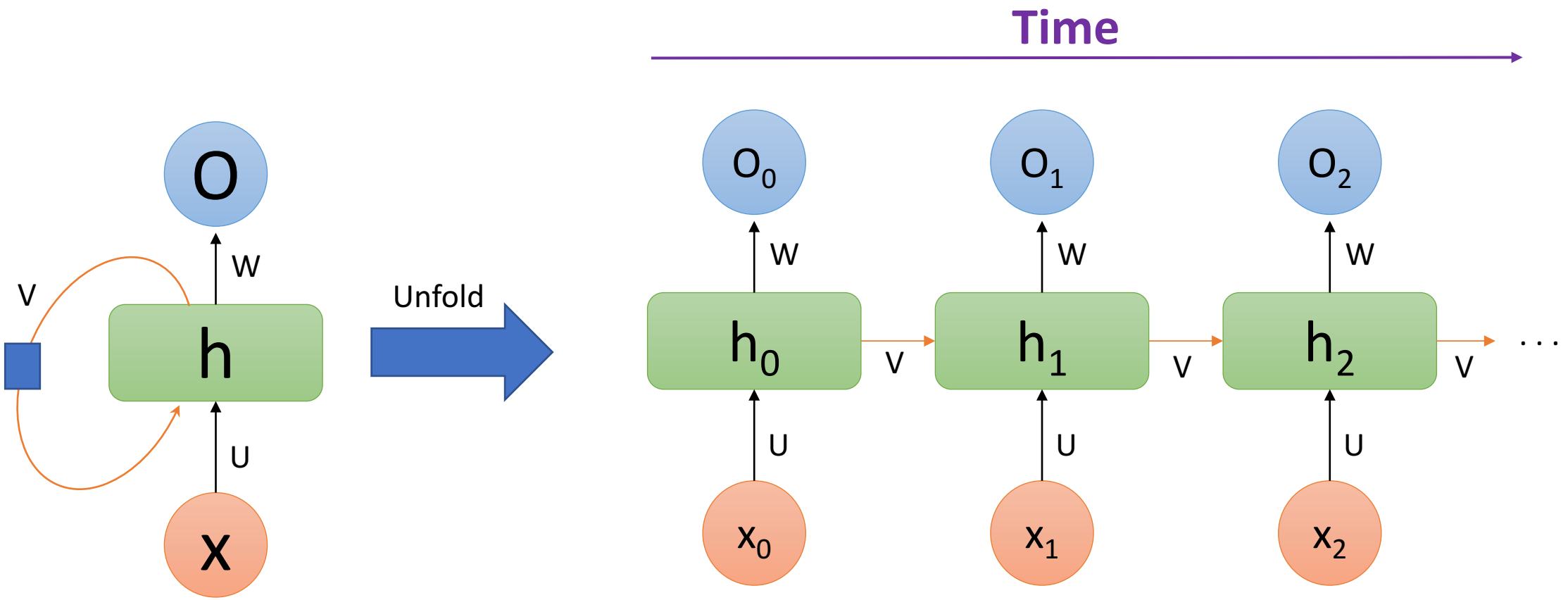
How to train RNN?

Backpropagation
(through time)

RNN with Outputs



RNN with Outputs



Unfolding this graph results in the sharing of parameters across a deep network structure

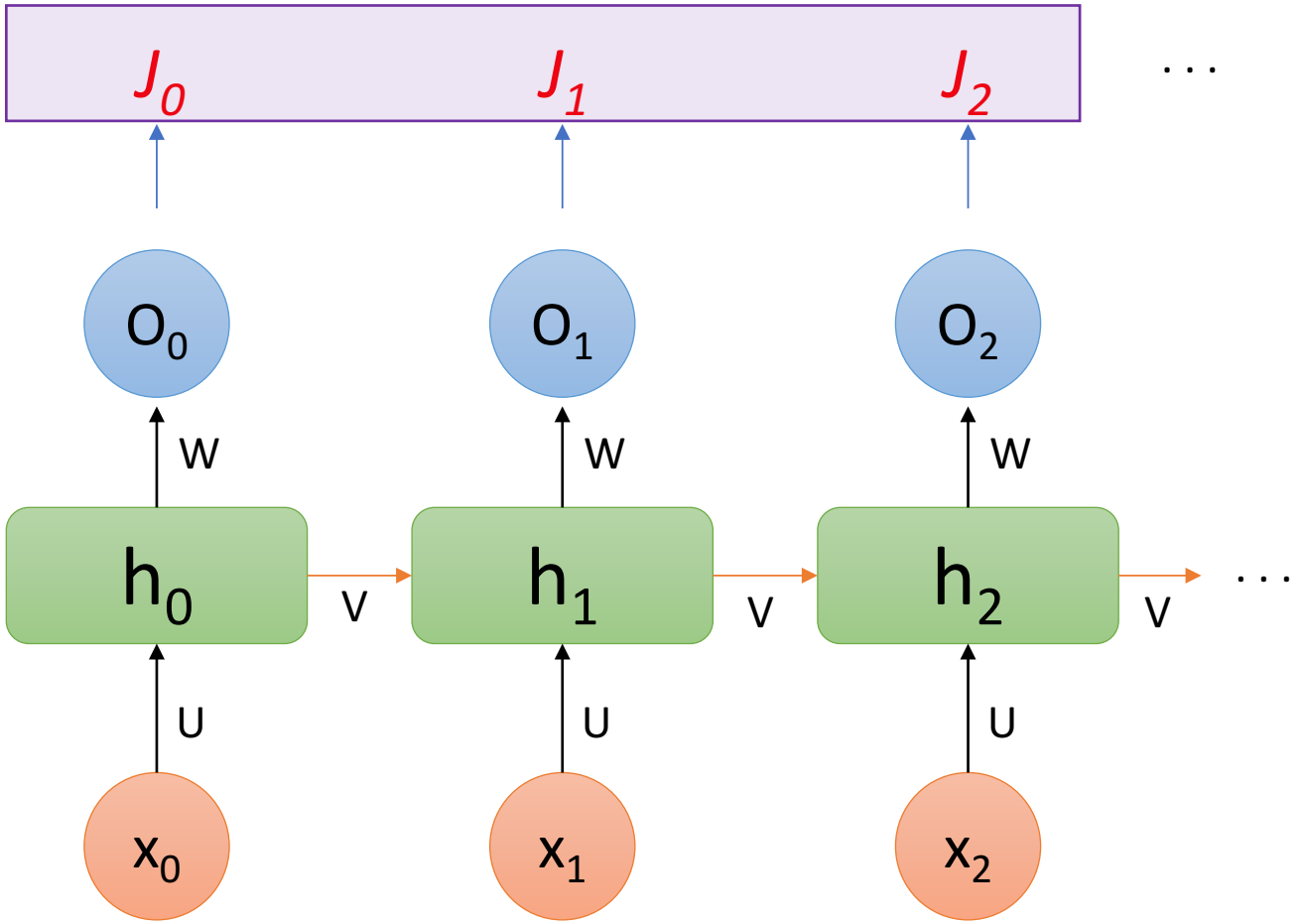
Recap!!! Backpropagation

1. Take the derivative (gradient) of the loss with respect to each parameter

2. Shift parameters in the opposite direction in order to minimize loss

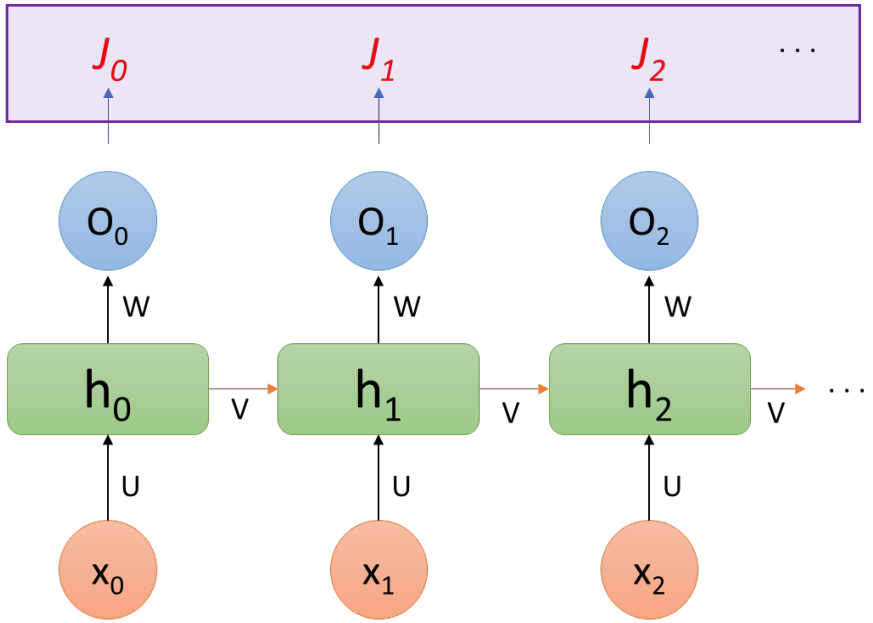
Backpropagation

- Considering loss (J) at each timestamp



Backpropagation

- We sum the losses across time

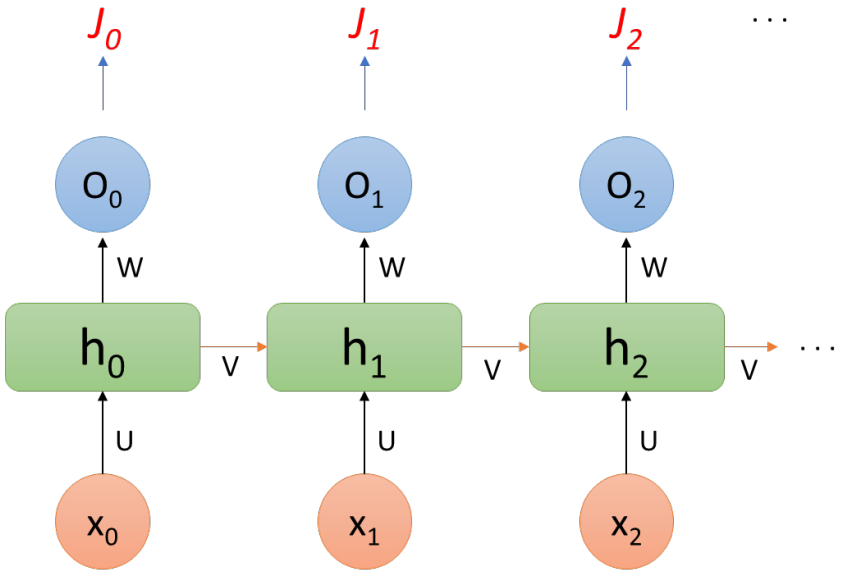


$$Loss\ at\ time\ t = J_t(\Theta)$$

$$Total\ loss = J(\Theta) = \sum_t J_t(\Theta)$$

Backpropagation

- Let's try it out for U with the chain rule:



$$\frac{\delta \mathrm{J}_2}{\delta U} = \frac{\delta J_2}{\delta \mathrm{O}_2} \frac{\delta \mathrm{O}_2}{\delta h_2} \boxed{\frac{\delta h_2}{\delta U}}$$

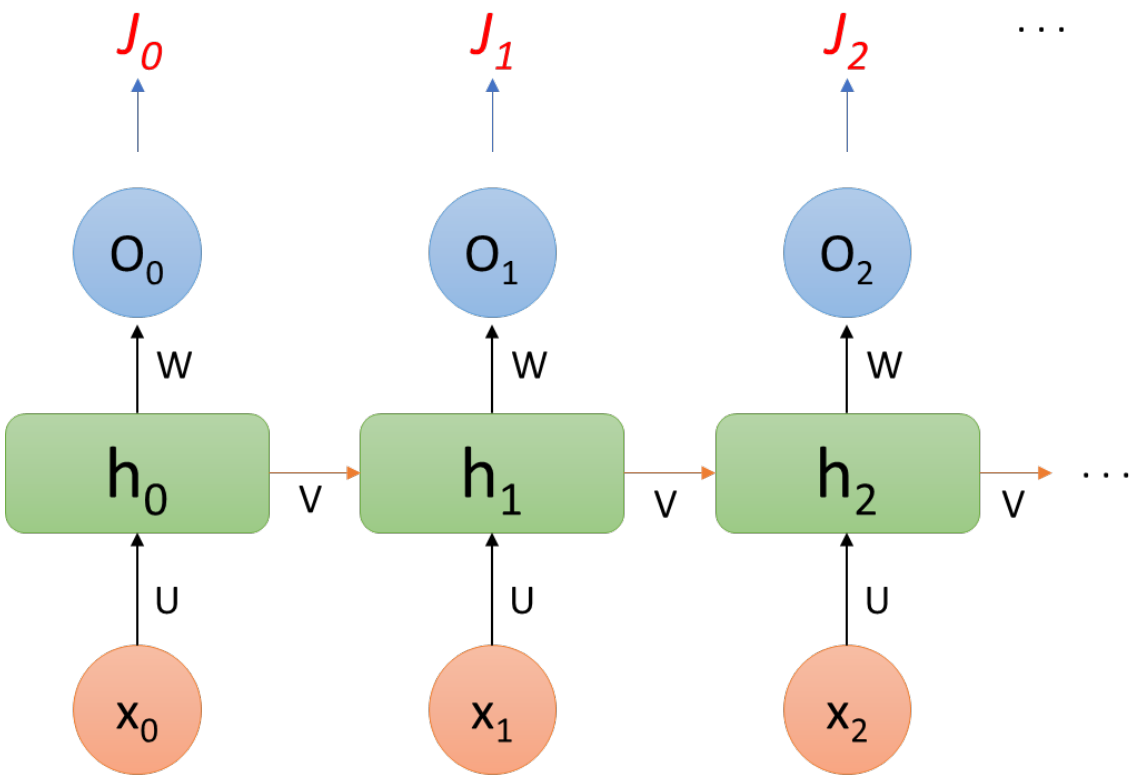
but wait…

$$h_2 = activiation\ function\ (Vh_1 + Ux_2)$$

- h1 also depends on U so we can't just treat $\frac{\delta h_2}{\delta U}$ as a constant!

Backpropagation

- How does h_2 depend on U?



$$\frac{\delta h_2}{\delta U}$$

$$+ \frac{\delta h_2}{\delta h_1} \frac{\delta h_1}{\delta U}$$

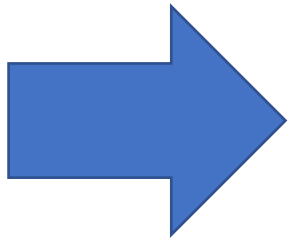
$$+ \frac{\delta h_1}{\delta h_0} \frac{\delta h_0}{\delta U}$$

Backpropagation Through Time

$$\frac{\delta h_2}{\delta U}$$

$$+ \frac{\delta h_2}{\delta h_1} \frac{\delta h_1}{\delta U}$$

$$+ \frac{\delta h_1}{\delta h_0} \frac{\delta h_0}{\delta U}$$



$$\frac{\delta J_2}{\delta U} = \sum_{k=0}^{2} \frac{\delta J_2}{\delta O_2} \frac{\delta O_2}{\delta h_2} \frac{\delta h_2}{\delta h_k} \frac{\delta h_k}{\delta U}$$

- Contributions of U in previous timesteps to the error at timestep t

Backpropagation Through Time

- Contributions of U in previous timesteps to the error at timestep t

$$\frac{\delta J_t}{\delta U} = \sum_{k=0}^{t} \frac{\delta J_t}{\delta O_t} \frac{\delta O_t}{\delta h_t} \frac{\delta h_t}{\delta h_k} \frac{\delta h_k}{\delta U}$$

The network with recurrence between hidden units is thus very powerful but also expensive to train.

Why Expensive?

Training Challenges!!

- **Two common issues are**
  - Exploding gradients
  - Vanishing gradients

Training Challenges!!

- **Exploding gradients** can occur when the gradient becomes too large *and it can make learning unstable..*

- **Solution**
  - Clipping Gradient: clipping gradients if their norm exceeds a given threshold.
  - Use Weight Regularization: Using an L1 or L2 penalty on the recurrent weights can help with exploding gradients
  - Many others…

Training Challenges!!

- **Vanishing gradients** can happen when optimization gets stuck at a certain point because the gradient is too small to progress.

Solution #1

Use ReLU instead of tanh as the non-linear activation function.

Solution # 2

Initialize W as the identity matrix as opposed a random initialization

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

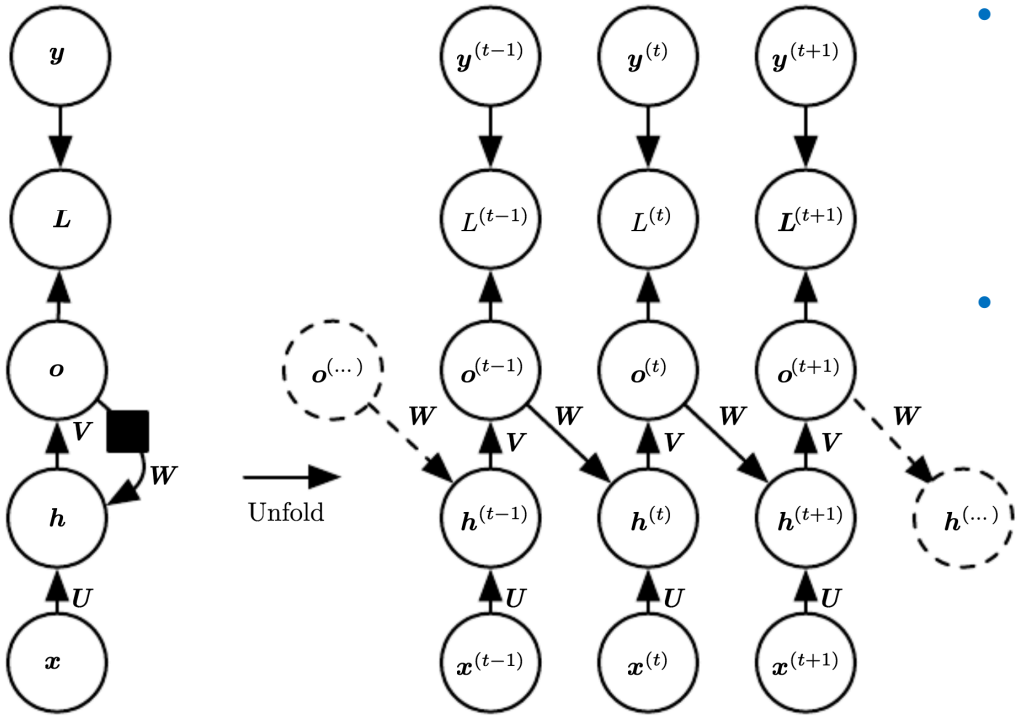
Prevents W from shrinking the gradients

Solution #3: Gated cells

- Rather each node being just a simple RNN cell, make each node a more complex unit with gates controlling what information is passed through.

- **Long Short Term Memory (LSTM)** cells are able to keep track of information throughout many timesteps.

Teacher Forcing and Networks with Output Recurrence



- **Less powerful than with hidden-to hidden recurrent connections**
  - It cannot simulate a universal Turing Machine
  - It requires that the output capture all information of past to predict future
- **Advantage**
  - In comparing loss function to output all time steps are decoupled
    - Each step can be trained in isolation
  - Training can be parallelized
    - Gradient for each step t computed in isolation
    - No need to compute output for the previous step first, because training set provides ideal value of output
- Can be trained with **Teacher Forcing**

Teacher Forcing Procedure

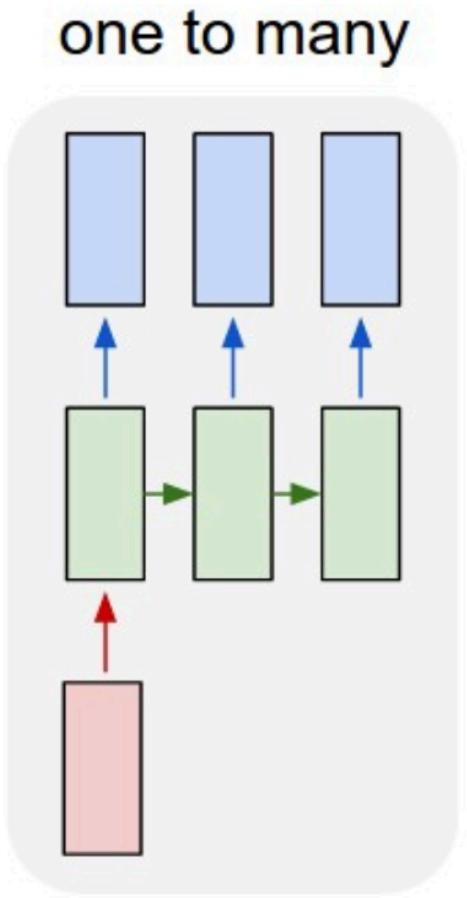
- Teacher forcing is a procedure for training RNNs with output-to-hidden recurrence
- It emerges from the maximum likelihood criterion
- During training time model receives ground truth output $o^{(t)}$ as input at time t+1

Teacher forcing can still be applied to models that have hidden to-hidden connections

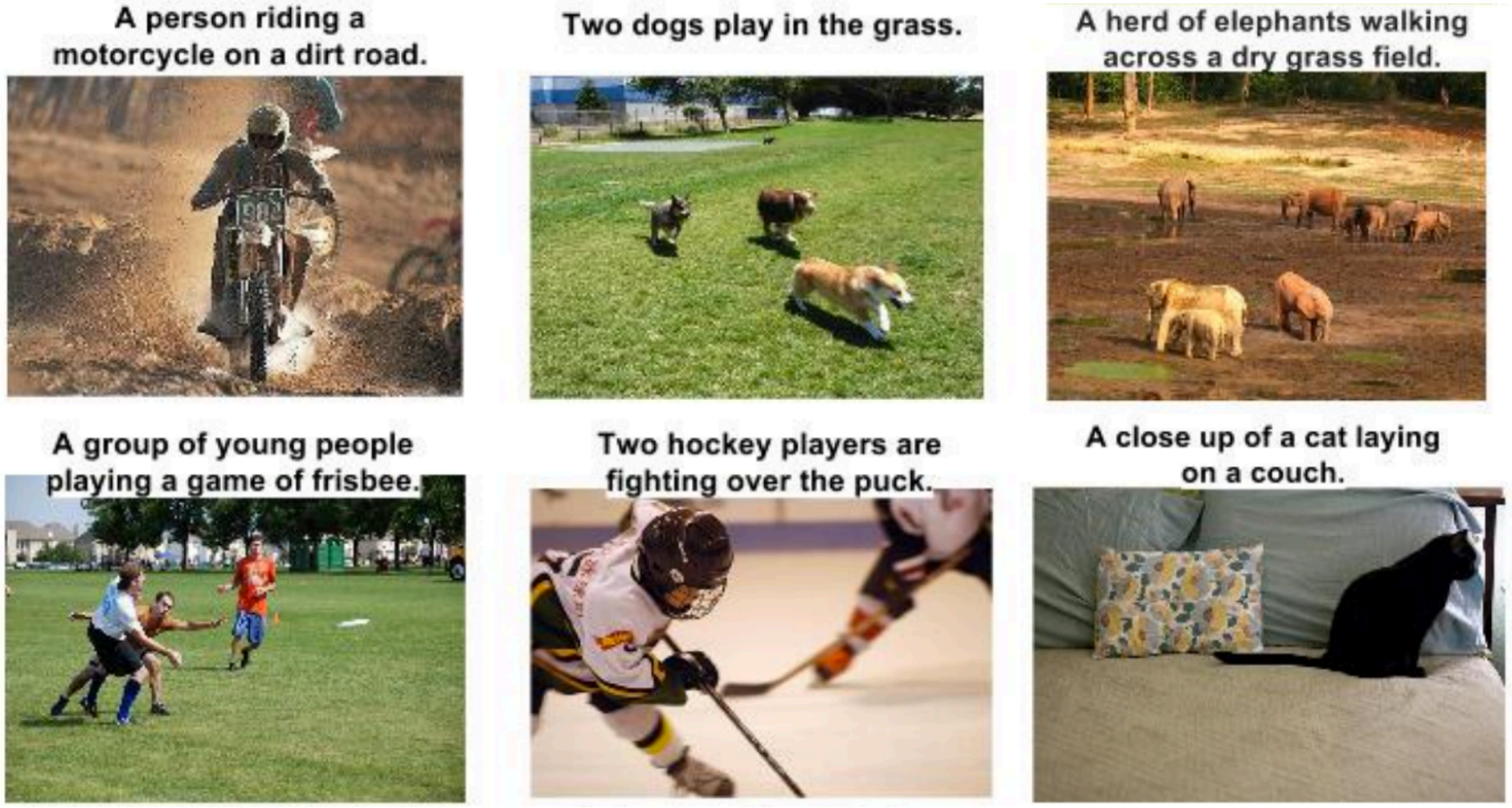
- **Professor Forcing algorithm (NIPS 2016)**
  - Uses adversarial domain adaptation to encourage the dynamics of the RNN to be the same when training the network and when sampling from the network over multiple time steps.

Use cases of RNN

- **Input:** Single
- **Output:** Sequence
- **For Example:** Image captioning takes an image and outputs a sentence of words.



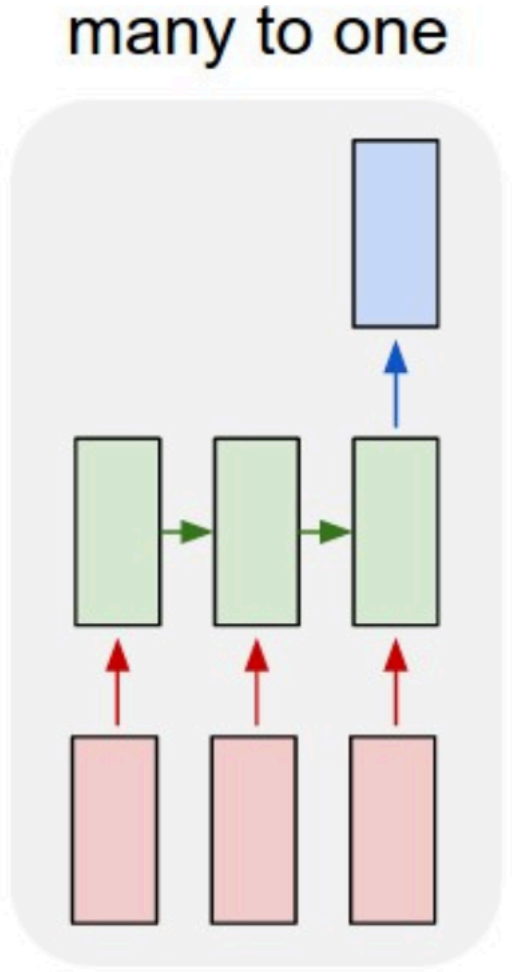
Use cases of RNN – Image Captions



Show and Tell: A Neural Image Caption Generator, CVPR 15

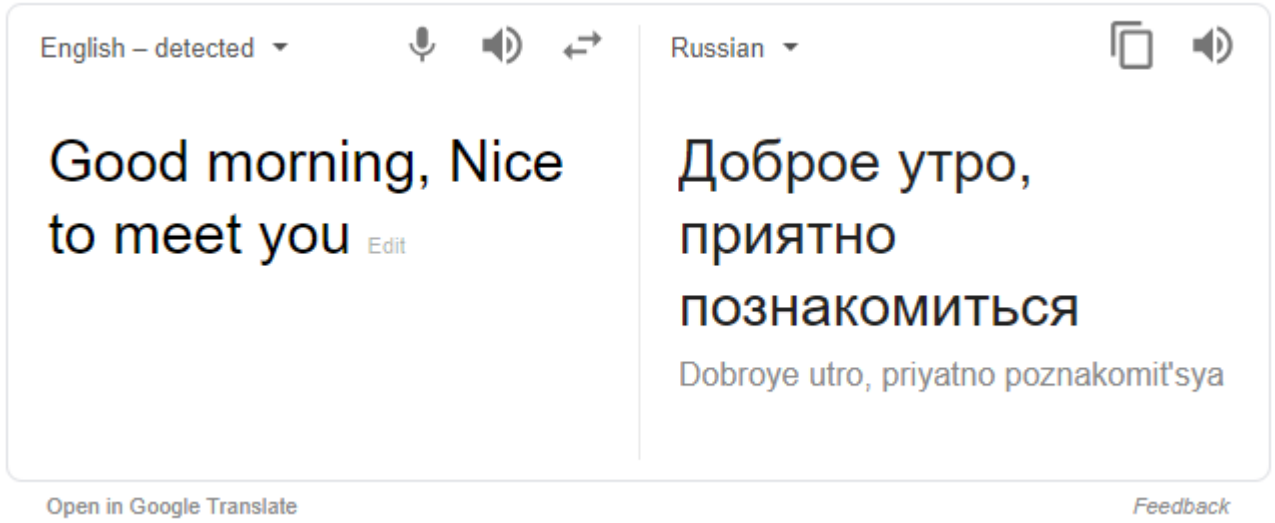
Use cases of RNN

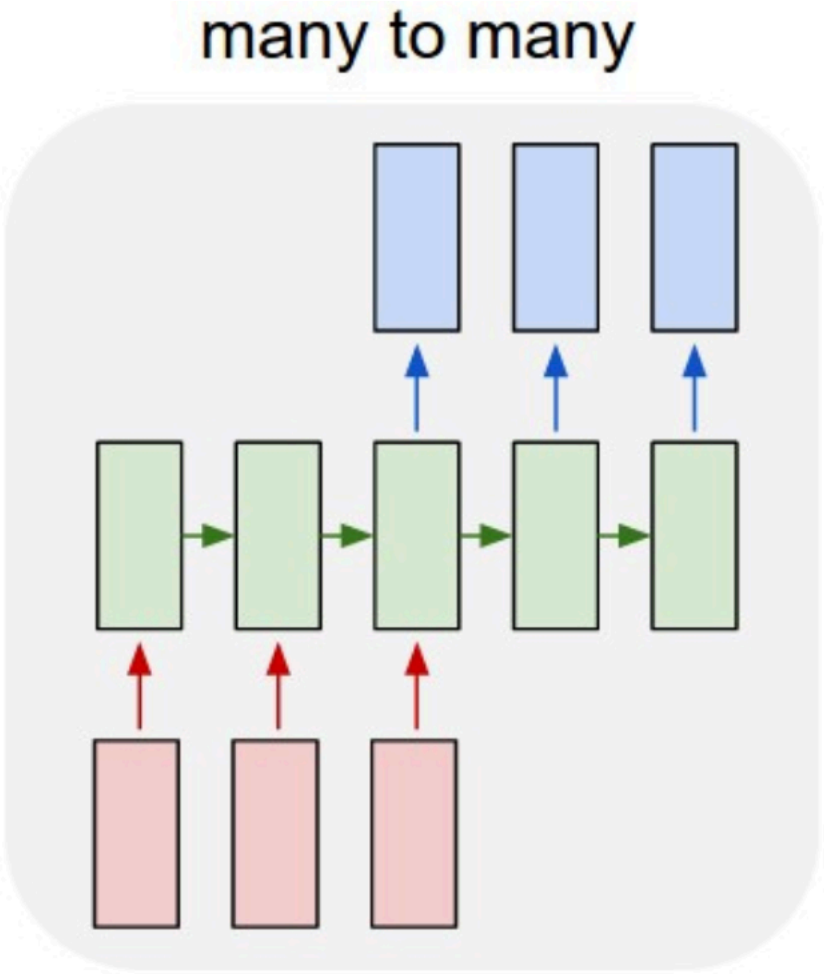
- **Input:** Sequence
- **Output:** Single output
- **For Example:** Sentiment analysis
  - Where a given sentence is classified as expressing positive or negative sentiment.



Use cases of RNN

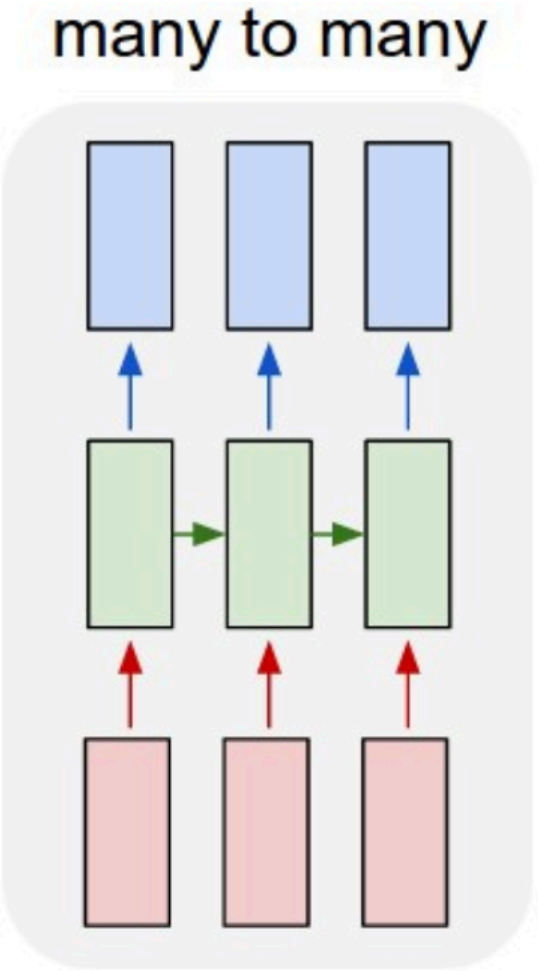
- **Input:** Sequence
- **Output:** Sequence
- **Example:** Machine translation



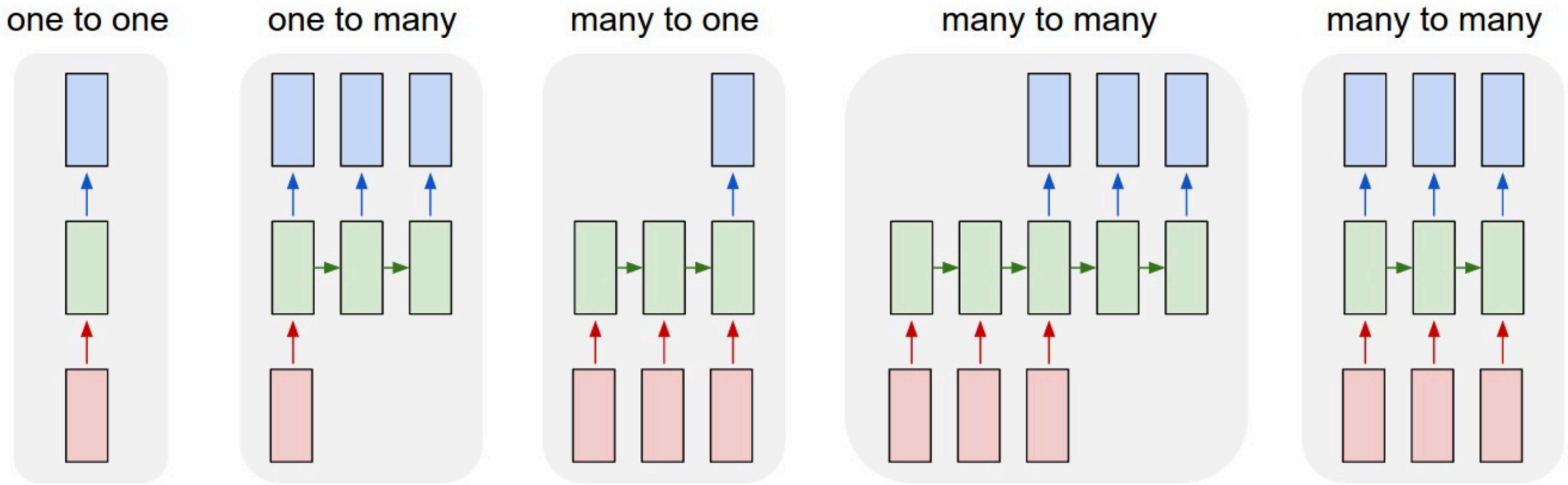


Use cases of RNN

- **Input:** Sequence
- **Output:** Sequence
- **Example:** Video classification where we wish to label each frame of the video



Use cases of RNN

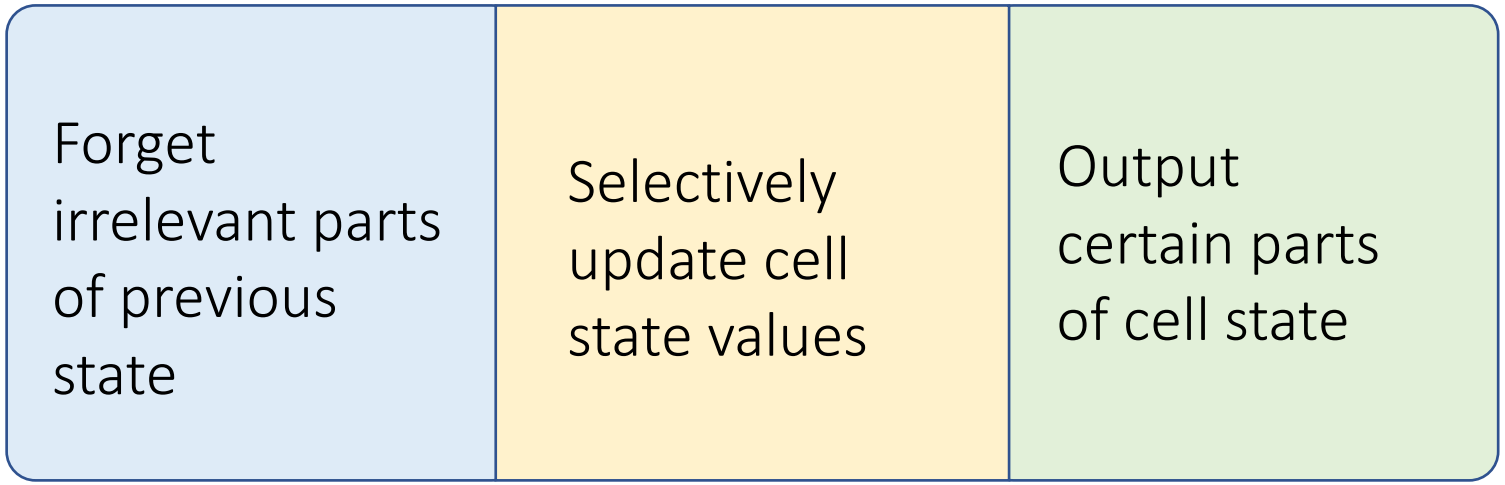


- Notice that in every case there is no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Long-Short-Term-Memories

(Hochreiter & Schmidhuer, 1997)

LSTM – Introduce a Memory Cell



Long-Short-Term-Memories (LSTM)

- It's applied to capture the long-range dependencies.
- LSTMs similar to RNN but they introduce a memory cell state $\boldsymbol{c_t}$.
- LSTMs have the ability to remove or add information to $\boldsymbol{c_t}$ regulated by structures called gates based on context.
- Update of $\boldsymbol{c_t}$ designed so gradients flows these nodes backward in time easily.
- $\boldsymbol{c_t}$ then controls what information from $\boldsymbol{h_{t-1}}$ and $\boldsymbol{x_t}$ and $\boldsymbol{c_{t-1}}$ should be used to generate $\mathbf{h_t}$.

Former description of LSTM Unit

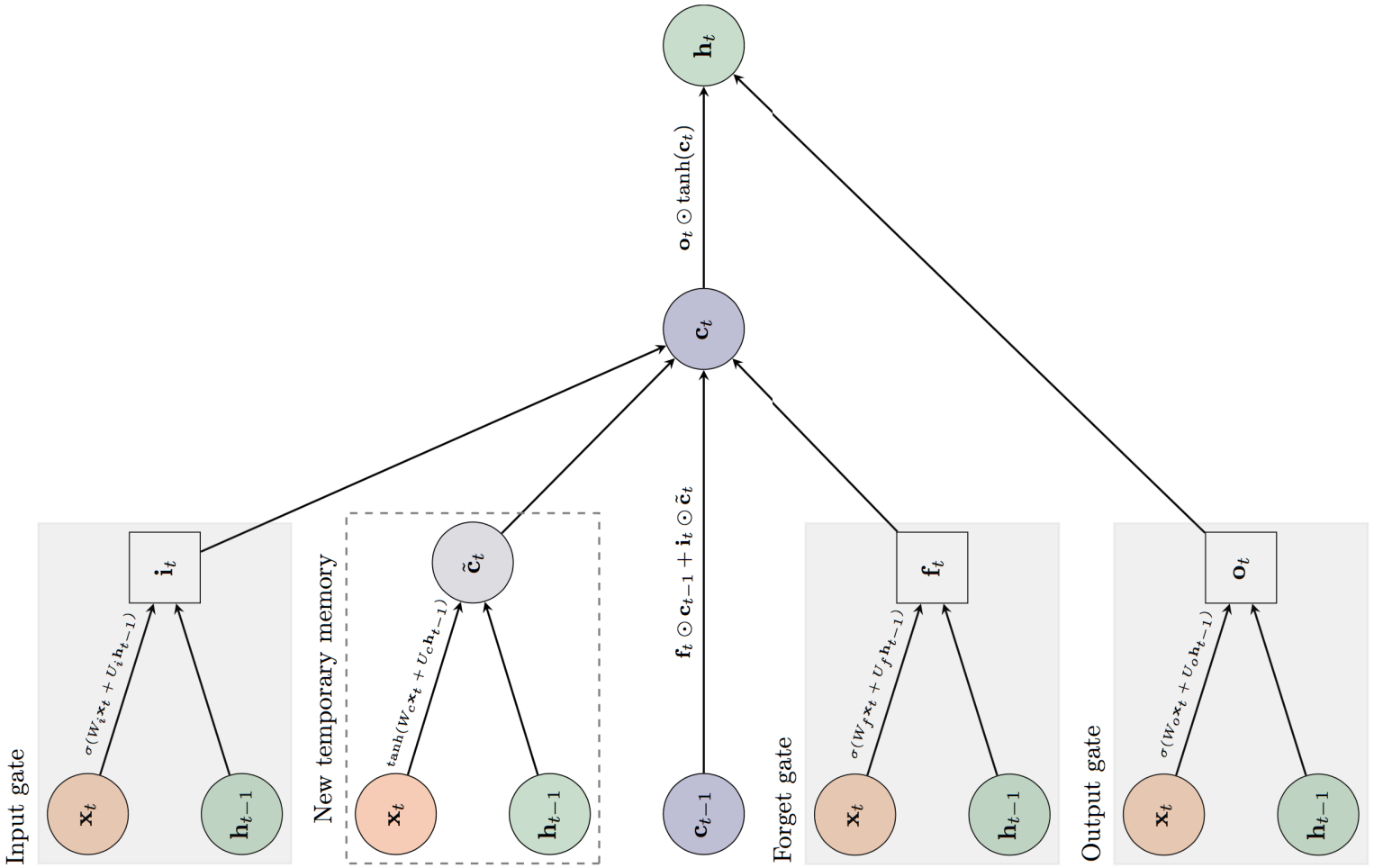
- LSTM introduces gates to calculate $\boldsymbol{h_t}$, $\boldsymbol{c_t}$ from $\boldsymbol{c_{t-1}}$, $\boldsymbol{h_{t-1}}$ and $\boldsymbol{x_t}$.
- Formal description of a LSTM unit:

$$\mathbf{i}_t = \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1}) \quad \text{Input gate}$$
$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1}) \quad \text{Forget gate}$$
$$\mathbf{o}_t = \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1}) \quad \text{Output/Exposure gate}$$
$$\tilde{\mathbf{c}}_t = \tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}) \quad \text{New memory cell}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad \text{Final memory cell}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where

- $\sigma(\cdot)$ is the sigmoid function and
- $\odot$ denotes element by element multiplication.

Former description of LSTM Unit

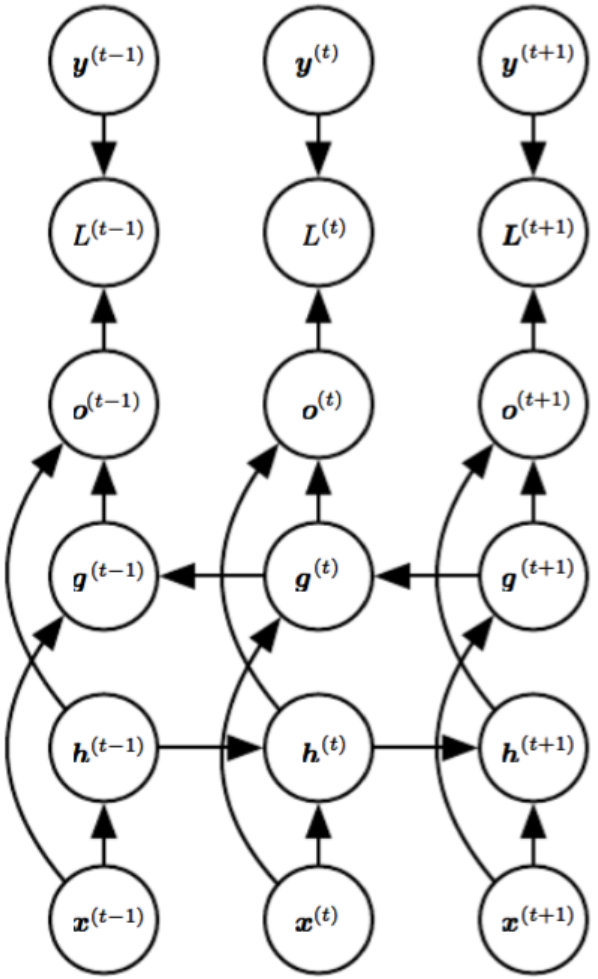


- **New temporary memory:**
  - Use x_t and h_{t-1} to generate new memory that includes aspects of x_t.
- **Input gate:**
  - Use x_t and h_{t-1} to determine whether the temporary memory c˜t is worth preserving.
- **Forget gate:**
  - Assess whether the past memory cell ct−1 should be included in ct.
- **Updated memory state:**
  - Use the forget and input gates to combine the new temporary memory and the current memory cell state to get ct.
- **Output gate:**
  - Decides which part of ct should be exposed to ht.

Source Image: From the course slide of Deep Learning in Data Science at KTH

Bidirectional RNNs

- It combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence.



Recursive Neural Network

- A **recursive network** has a computational graph that generalizes that of the recurrent network from a chain to a **tree**.

- **Pro**: Compared with a RNN, for a sequence of the same length τ, the depth (measured as the number of compositions of nonlinear operations) can be drastically reduced from τ to $O(\log\tau)$.

- **Con**: **how to best structure the tree?** Balanced binary tree is an optional but not optimal for many data. For natural sentences, one can use a **parser** to yield the tree structure, but this is both expensive and inaccurate.

- Thus recursive NN is NOT popular.

Summary

- Natural Language Processing (NLP)
- Recurrent Neural Network (RNN)
- RNN – Architecture
- RNN – Training
- Use cases of RNN
- Variants of RNN
- Recursive Neural Network

Reference

- This lecture is based on the following resources + other web resources.
  - Chapter 10: http://www.deeplearningbook.org/contents/rnn.html
  - https://arxiv.org/pdf/1703.03091.pdf
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
  - http://introtodeeplearning.com/materials/2018_6S191_Lecture2.pdf
  - RNN and LSTM slides: http://bit.ly/2sO00ZC
  - https://medium.com/@jianqiangma/all-about-recurrent-neural-networks-9e5ae2936f6e

Thank You ☺

Appendix

in RNN's Input is considered as time steps.

ex : input(X) = ["this", "movie", "is", "not", "good"]

Time stamp for "this" is x(0), "movie" is x(1), "is" is x(2) ,"not" is x(3) and "good" is x(4).

Matrix calculation (Refresher)

https://medium.com/from-the-scratch/deep-learning-deep-guide-for-all-your-matrix-dimensions-and-calculations-415012de1568