Sequence Models

Recurrent Neural Network

INNOPOLIS UNIVERSITY

Contents

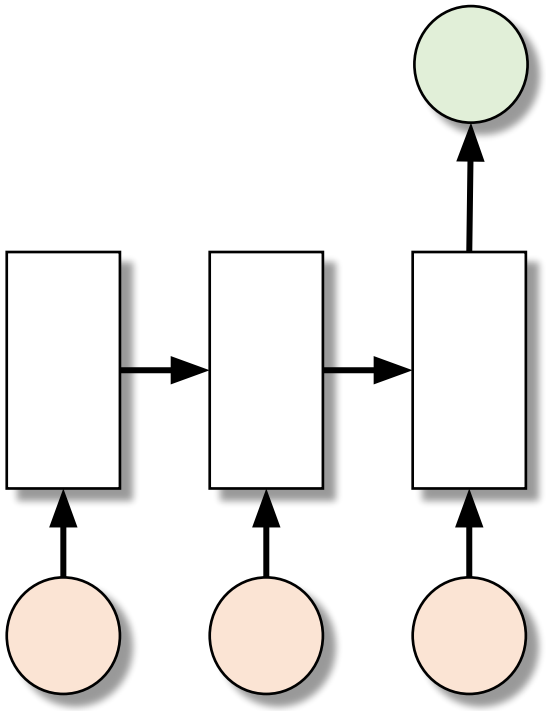
Sequence Modeling

- What is a sequence?
  - Time series
  - Machine translation
  - Music generation
  - Speech-to-text and text-to-speech
  - Many more…

- Technically, an Recurrent Neural Network can models sequences

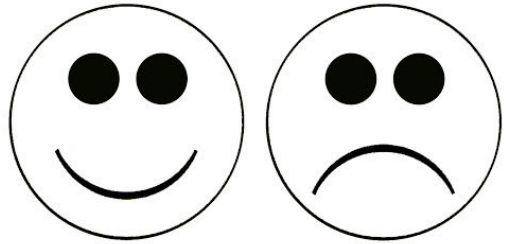
- RNN successfully generate
  - TED Talks https://www.youtube.com/watch?v=-OodHtJ1saY&feature=youtu.be&t=31s
  - Eminem rapper https://soundcloud.com/mrchrisjohnson/recurrent-neural-shady
  - Music https://web.stanford.edu/class/cs224n/reports/2762076.pdf

Sequence Modeling Applications



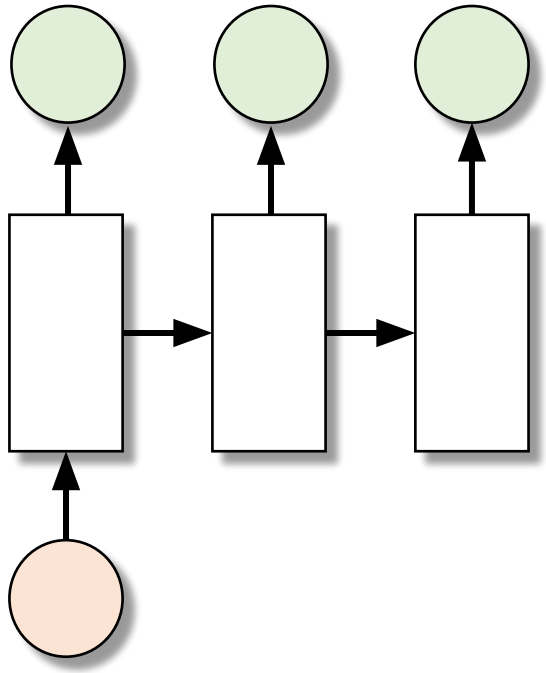
Many to One

- **Input:** Sequence
- **Output:** Single output
- **For Example:** Sentiment analysis



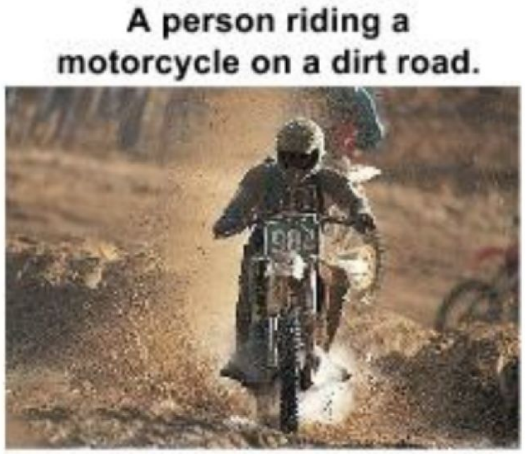
- Where a given tweet is classified as expressing positive or negative sentiment.

Sequence Modeling Applications



One to Many

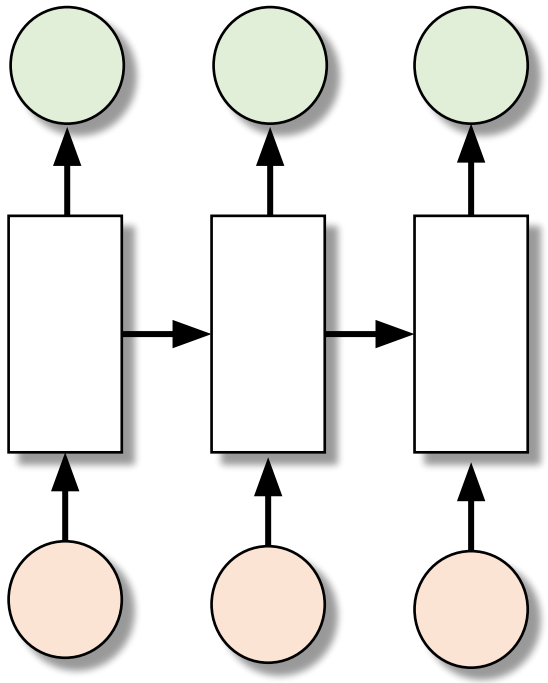
- **Input:** Single
- **Output:** Sequence
- **For Example:** Image captioning



Show and Tell: A Neural Image Caption Generator (CVPR 15)

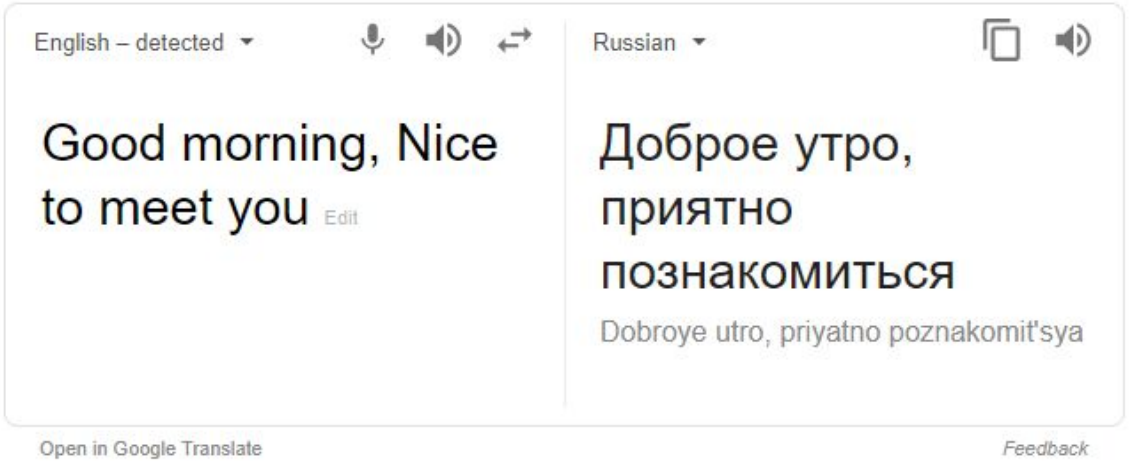
Image captioning takes an image and outputs a sentence of words.

Sequence Modeling Applications



Many to Many

- **Input:** Sequence
- **Output:** Sequence
- **For Example:** Machine Translation



Sequence Modeling

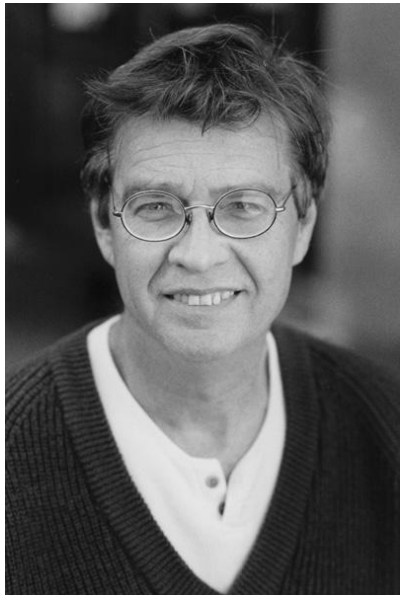
- To model sequences, we need:
  1. To deal with variable-length sequences
  2. To maintain sequence order
  3. To keep track of long-term dependencies
  4. To share parameters across the sequence

Recurrent Neural Network (RNN)

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”

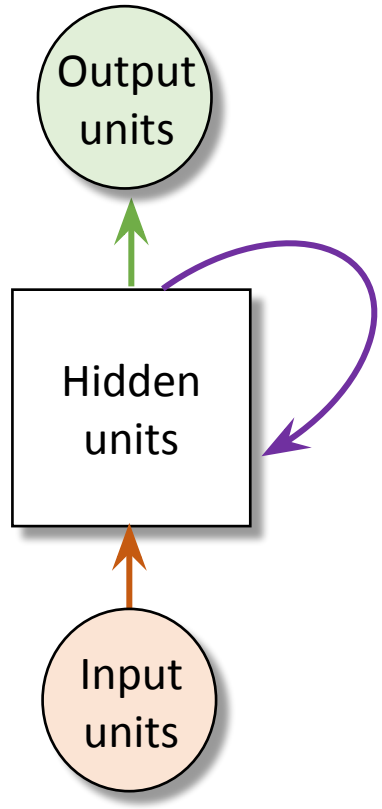


David Rumelhart

Recurrent Neural Network (RNN)

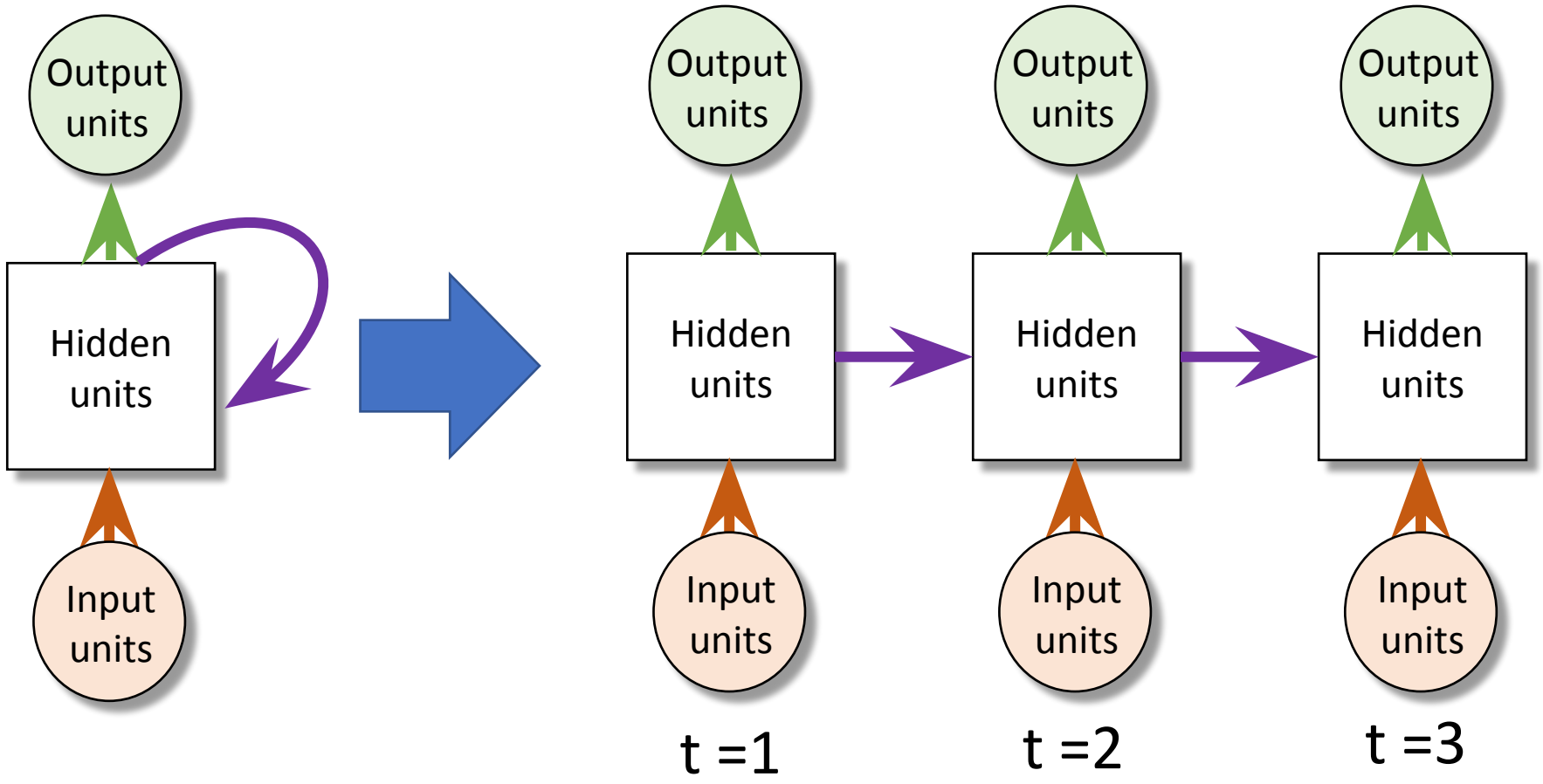
- We can think of an RNN as a dynamical system with one set of hidden units which feed into themselves.

- The network's graph would then have self-loops.



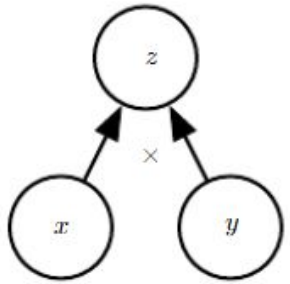
Recurrent Neural Network (RNN)

- Unfold the RNN (e.g., t=3)

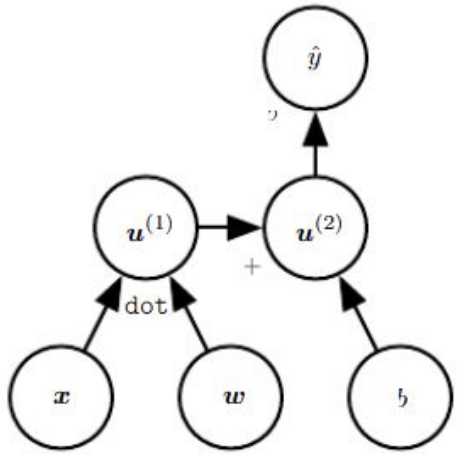


Computational Graph

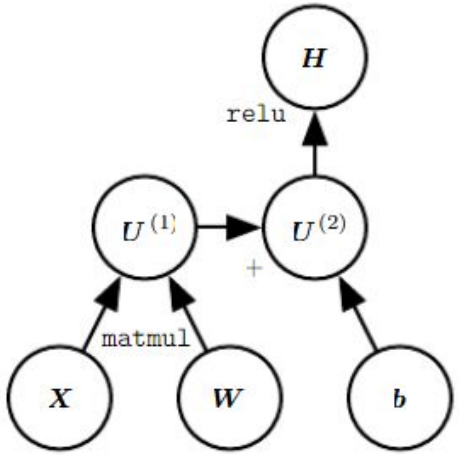
- It is a way to formalize the structure of a set of computations – those involved in mapping inputs and parameters to outputs and loss.



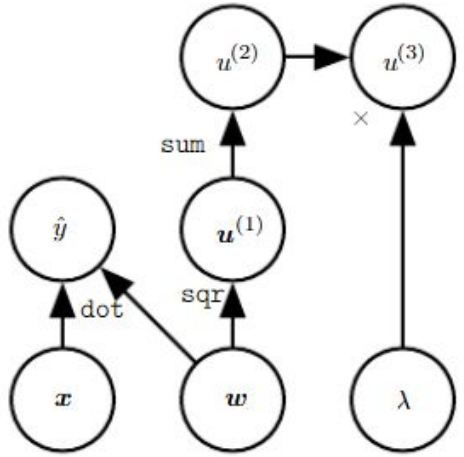
(a)



(b)



(c)



(d)

$$z = xy$$

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

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

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

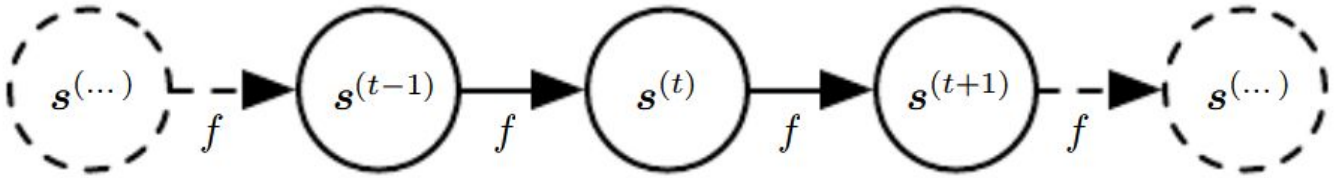
Familiarization with Recurrent Process

- Let's consider a dynamical system

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

- For Example, T = 3-time steps

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



Example II

- Let's consider 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

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

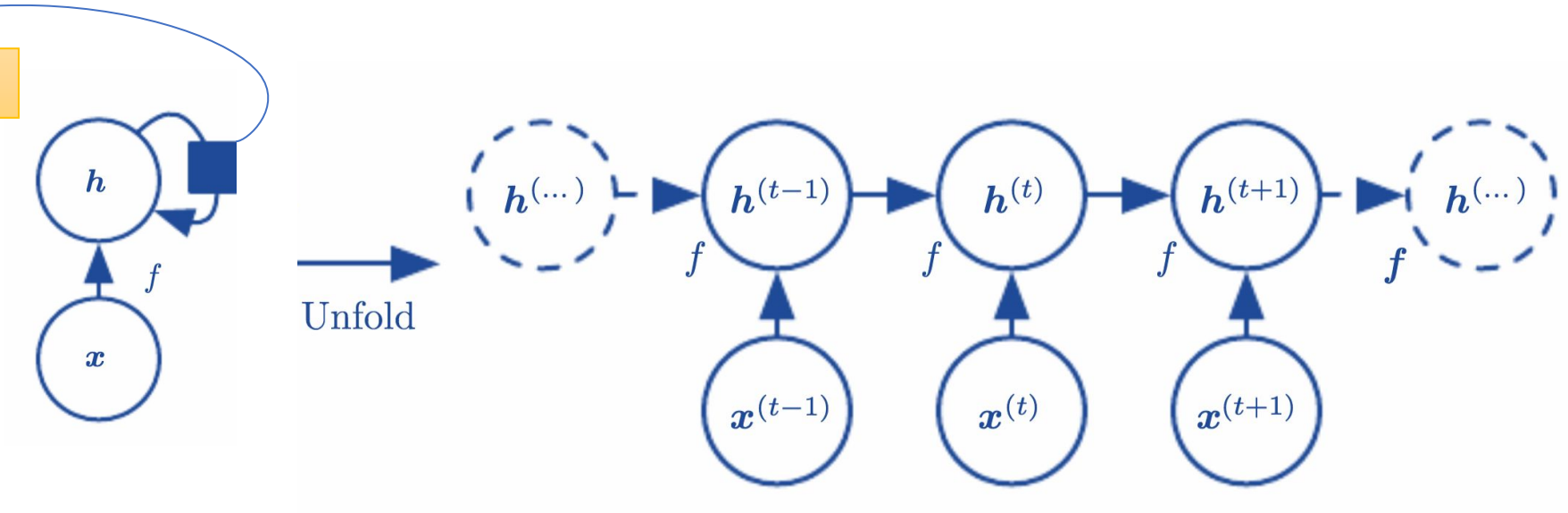
  - **Where**
    - $h^{(t)}$ is the hidden state at time *t* (a vector)
    - $x^{(t)}$ is input vector at time *t*
    - $\boldsymbol{\theta}$ is the parameters of *f*
  - **$\boldsymbol{\theta}$ remains constant as *t* changes. (why?)**

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

RNN with No Outputs

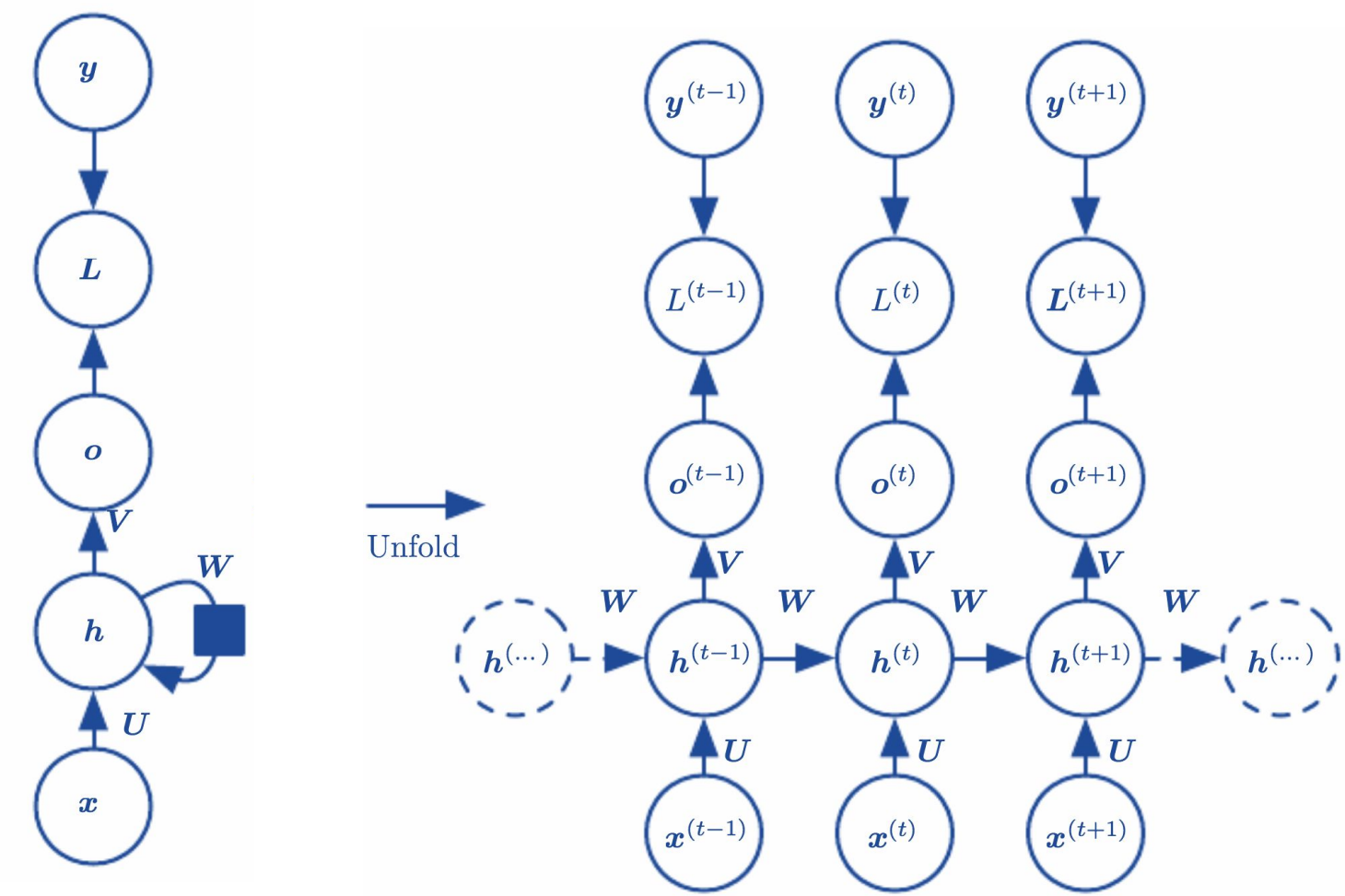
$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

Delay of time step 1.



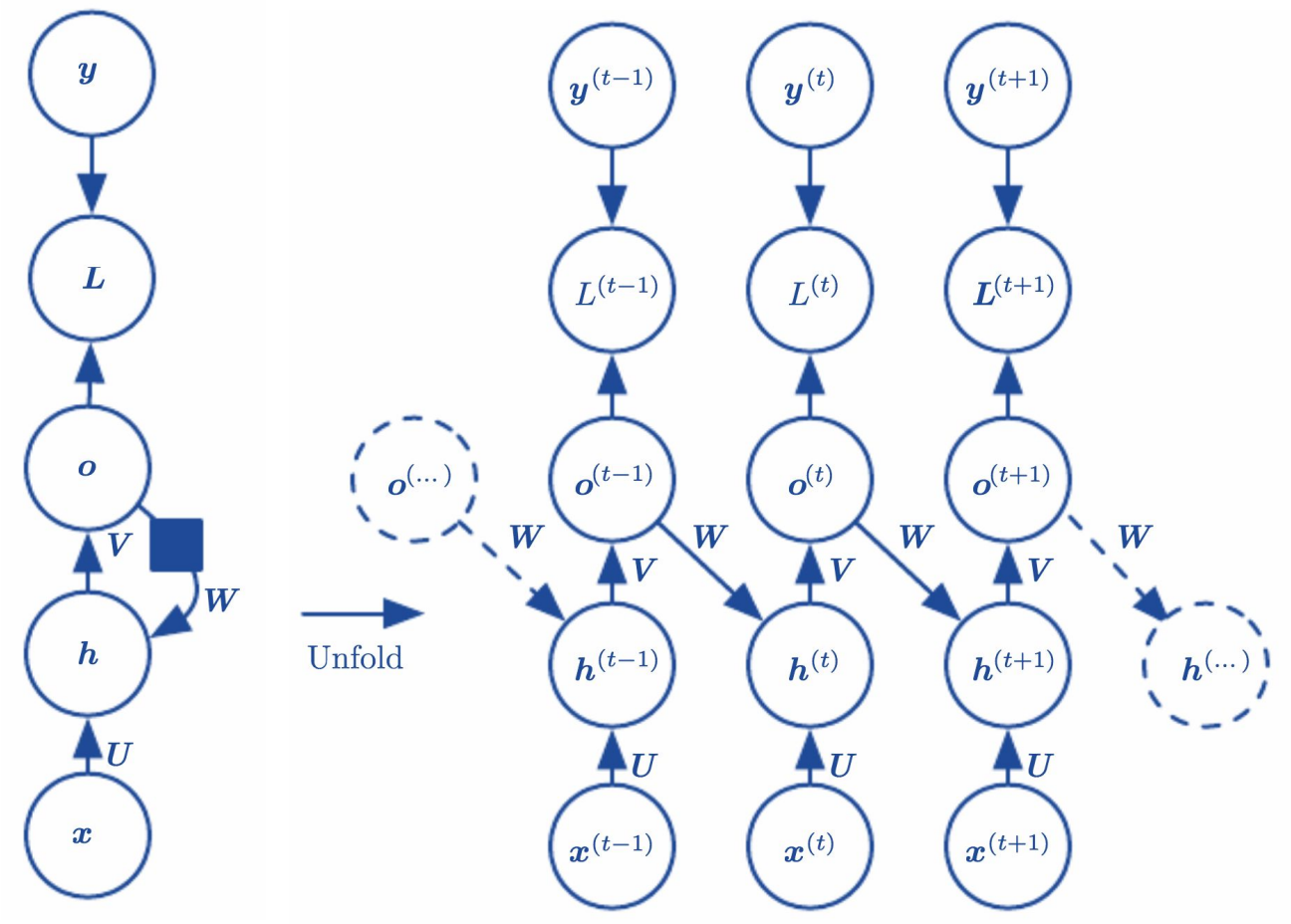
Design Patterns for RNN – 1/3

y: target
L: Loss
o: output
h: hidden
x: input



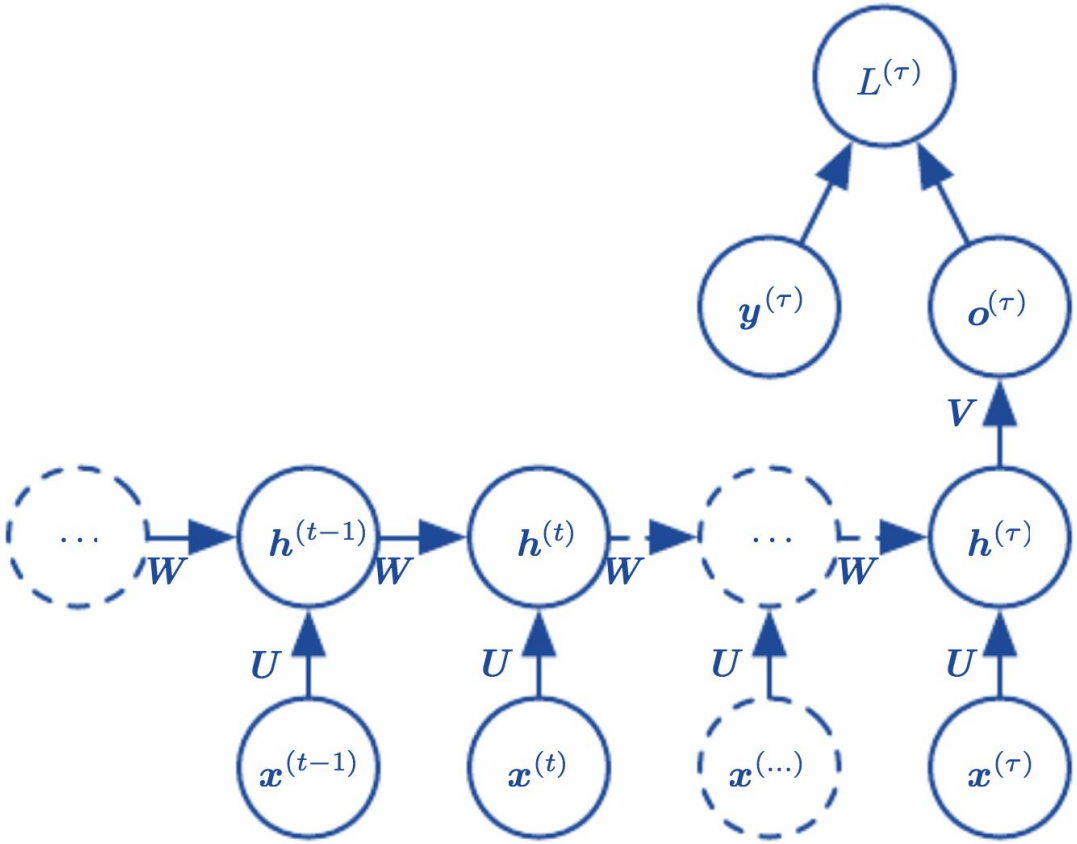
Recurrent networks that produce an output at each time step and have recurrent connections between hidden units

Design Patterns for RNN – 2/3



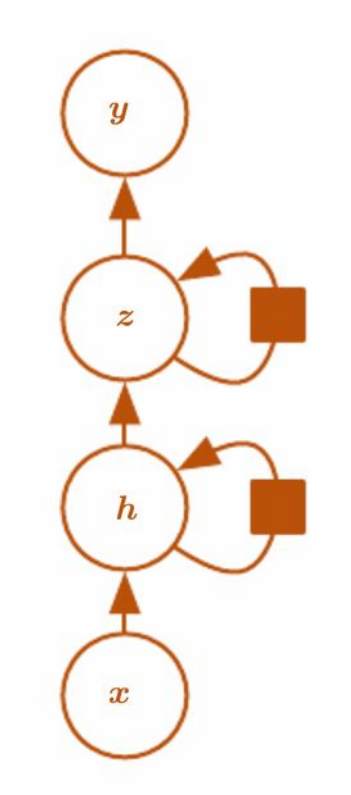
Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step

Design Patterns for RNN – 3/3



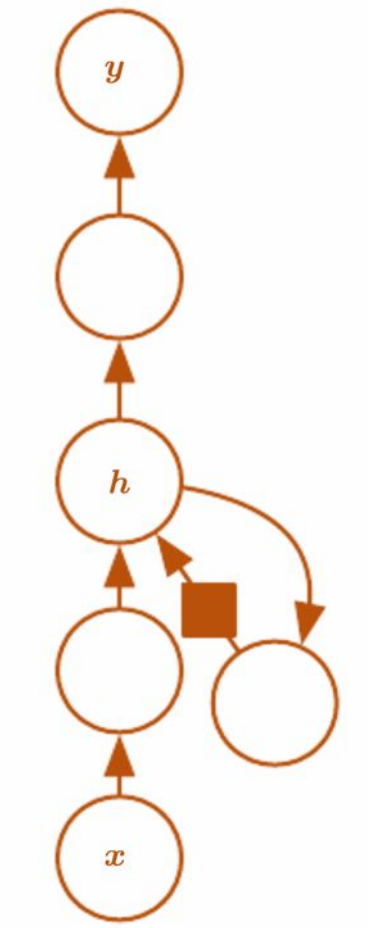
Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output

Deep Recurrent Neural Network – 1/3



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

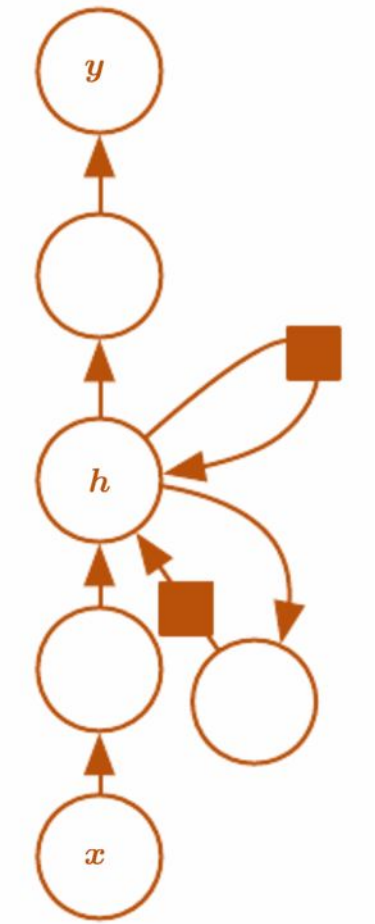
Deep Recurrent Neural Network – 2/3



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

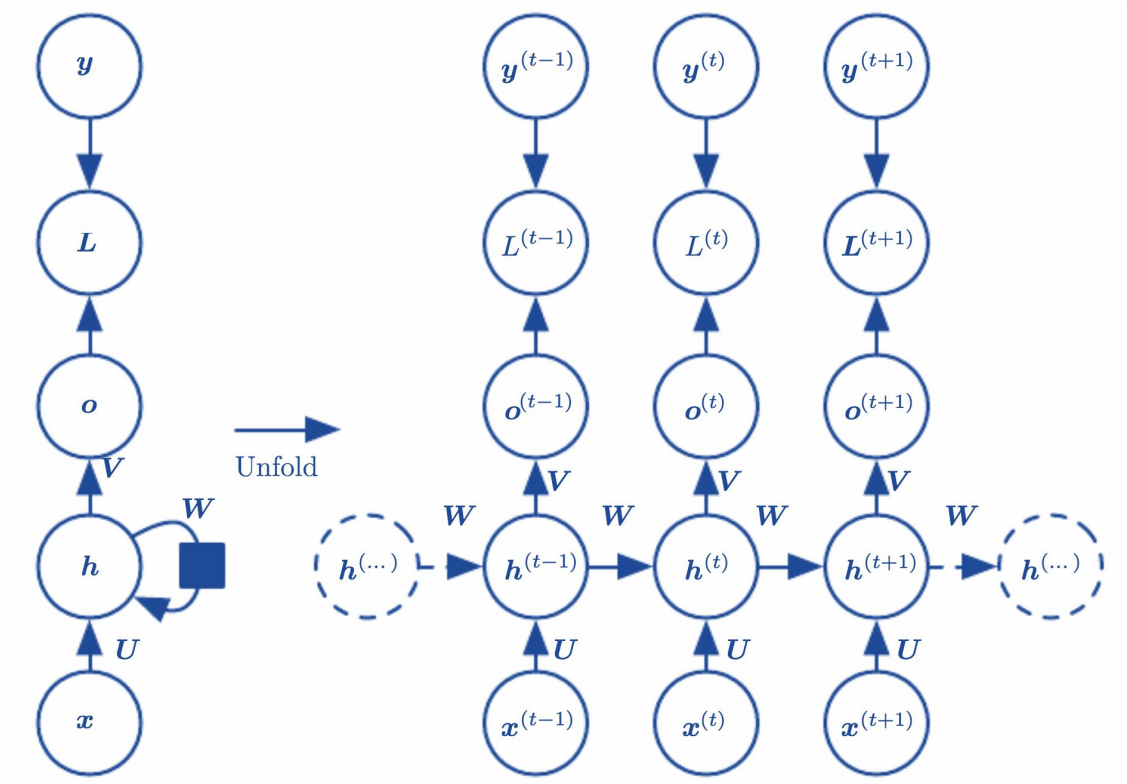
This may lengthen the shortest path linking different time steps.

Deep Recurrent Neural Network – 3/3



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

RNN – Equations!



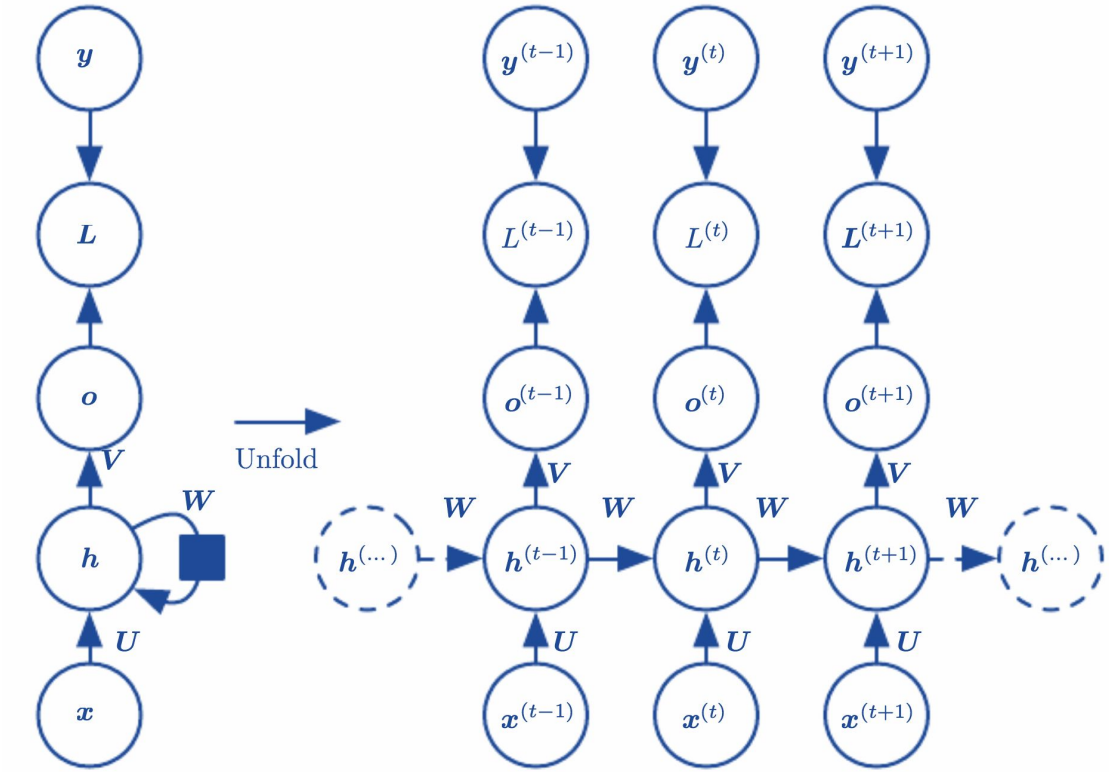
$$\boldsymbol{a}^{(t)} =$$

$$\boldsymbol{h}^{(t)} =$$

$$\boldsymbol{o}^{(t)} =$$

$$\hat{\boldsymbol{y}}^{(t)} =$$

RNN – Equations!

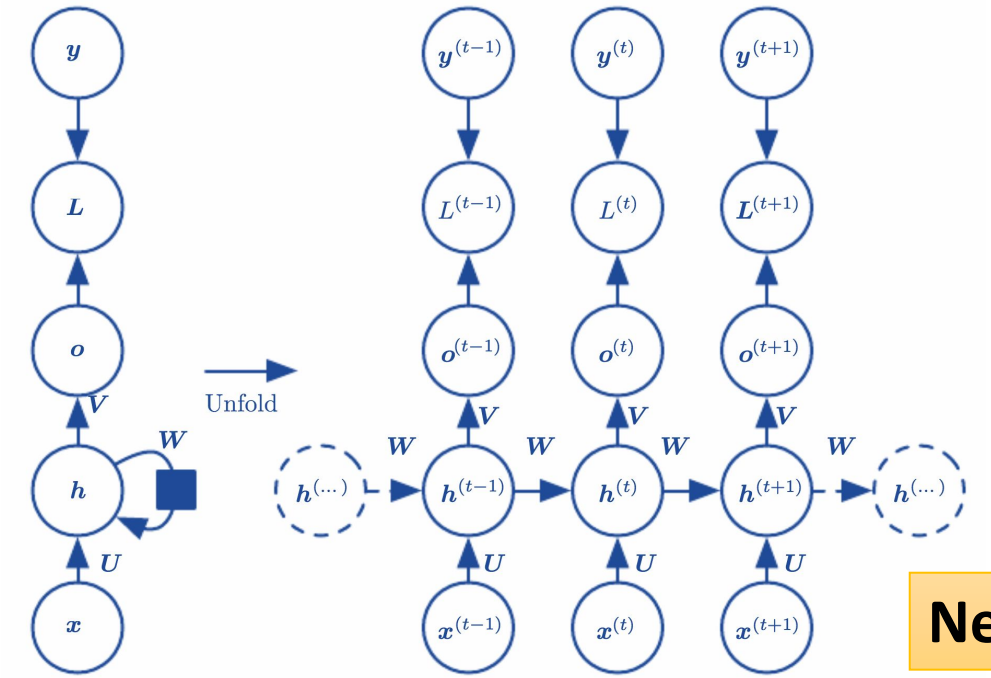


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)}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

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

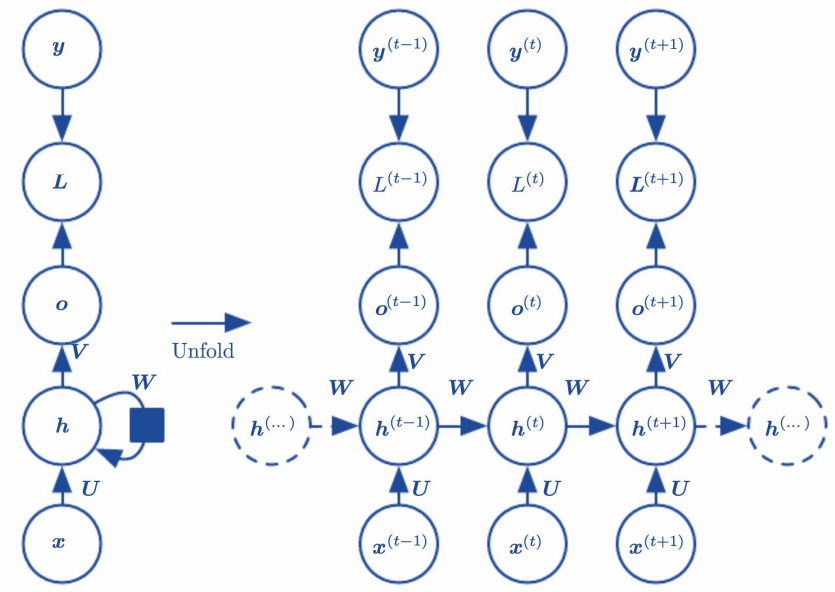


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)}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

Network's Output and Hidden vectors

- $\boldsymbol{a}^{(t)}$ hidden state at time t of size **m x 1** before non-linearity
- $\boldsymbol{h}^{(t)}$ hidden state at time t of size **m x 1**
- $\boldsymbol{o}^{(t)}$ output vector at time t of size **C x 1**
- $\hat{\boldsymbol{y}}^{(t)}$ output probability vector at time t of size **C X 1**

Parameters of the Network



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

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 $\mathbf{o}^{t}$

Properties of RNN

- Most recurrent networks can also process sequences of variable length

- The key idea behind RNNs is parameter sharing

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.

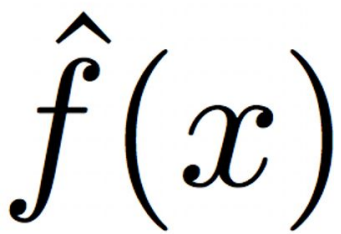
Importance of Parameter Sharing!

- **For Example:**

"I went to Russia in 2019"

"In 2019, I went to Russia."

Which year narrator went to Russia?



Machine Learning Model

2019

- 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?

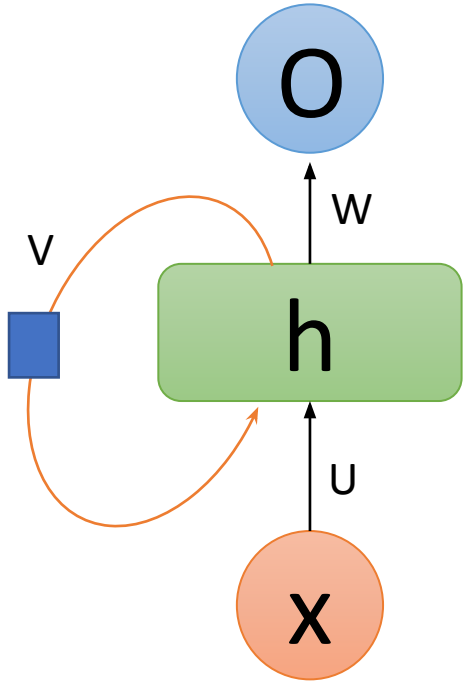
Backpropagation Through Time

Recap!!! Backpropagation

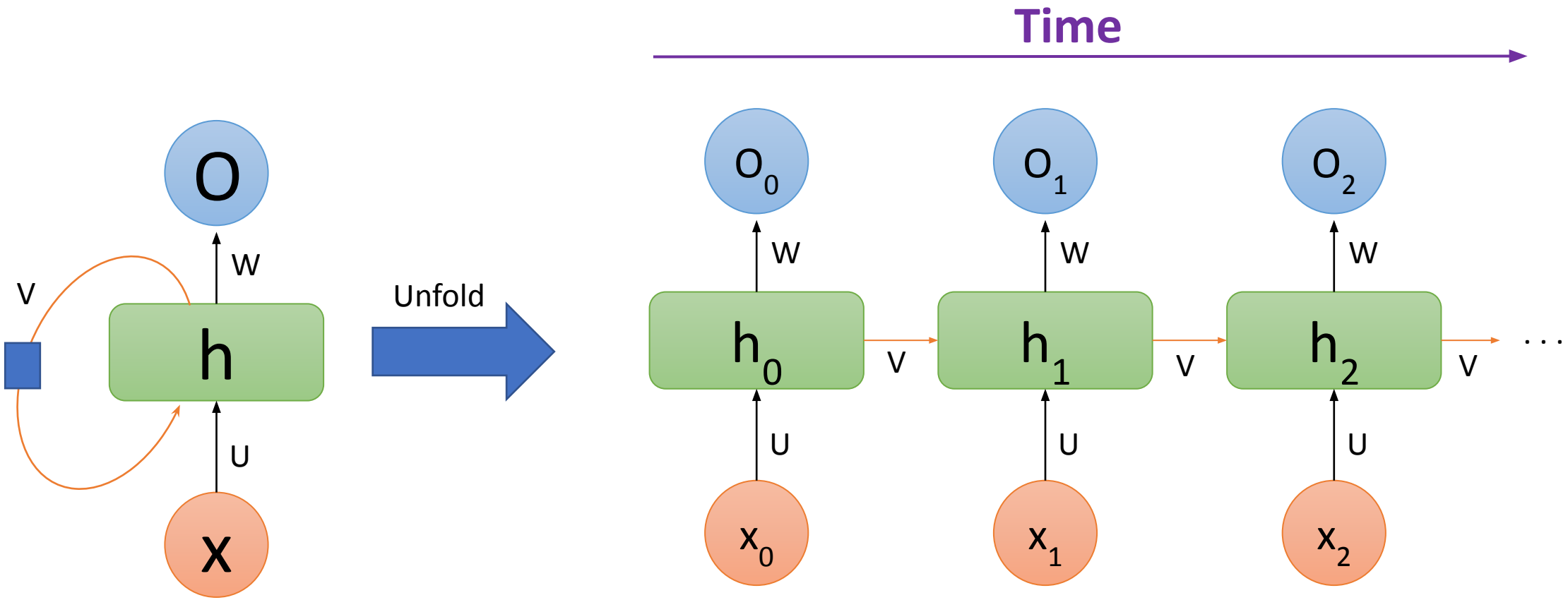
1. Take the derivative (gradient) of the loss with respect to each parameter

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

Backpropagation Through Time



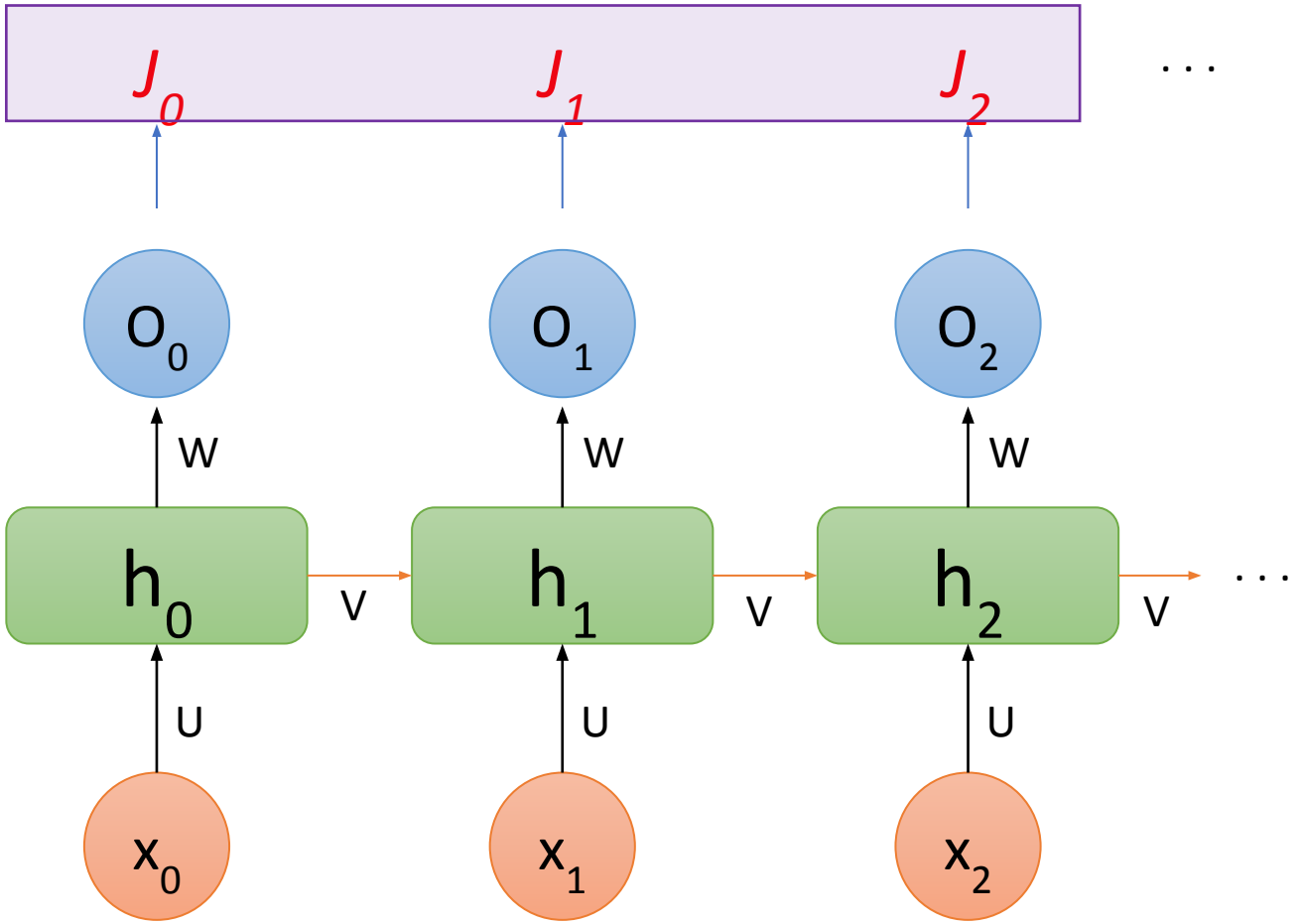
Backpropagation Through Time



Unfolding this graph results in the sharing of parameters

Backpropagation

Considering loss (J) at each timestamp

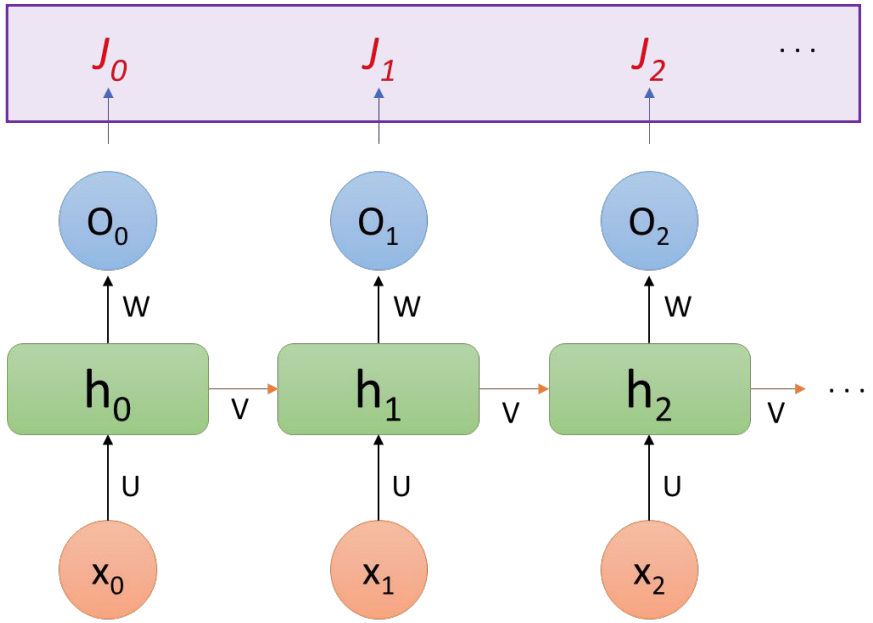


Backpropagation

- We sum the losses across time

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

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



We sum gradients across each time for each parameter P:

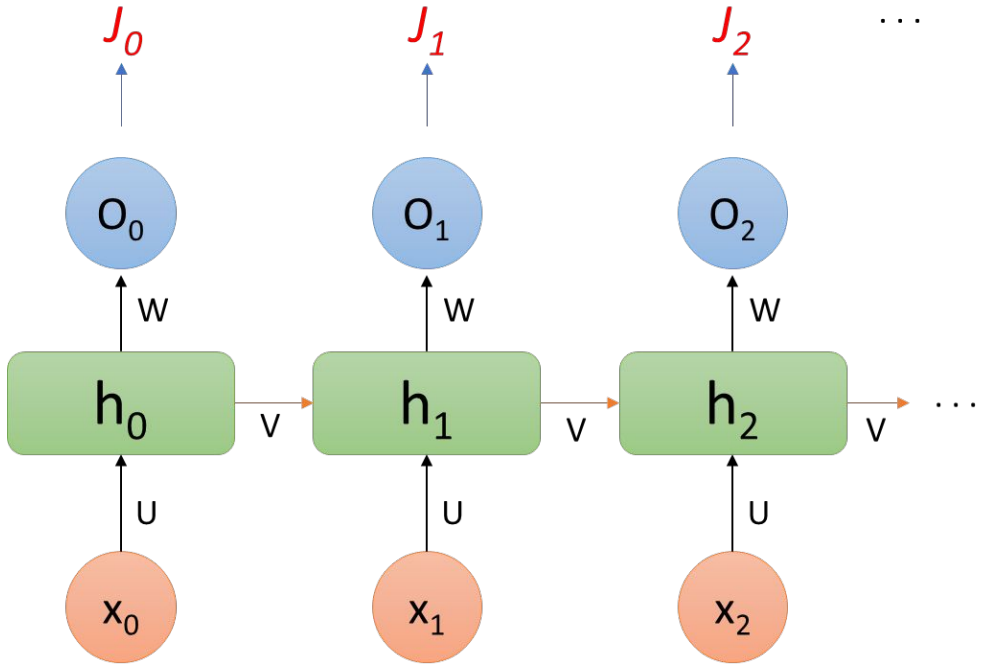
$$\frac{\delta J}{\delta P} = \sum_t \frac{\delta J_t}{\delta P}$$

Backpropagation

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

$$\frac{\delta J}{\delta U} = \sum_t \frac{\delta J_t}{\delta U}$$

$$\frac{\delta J_2}{\delta U} = \frac{\delta J_2}{\delta O_2} \frac{\delta O_2}{\delta h_2} \boxed{\frac{\delta h_2}{\delta U}}$$

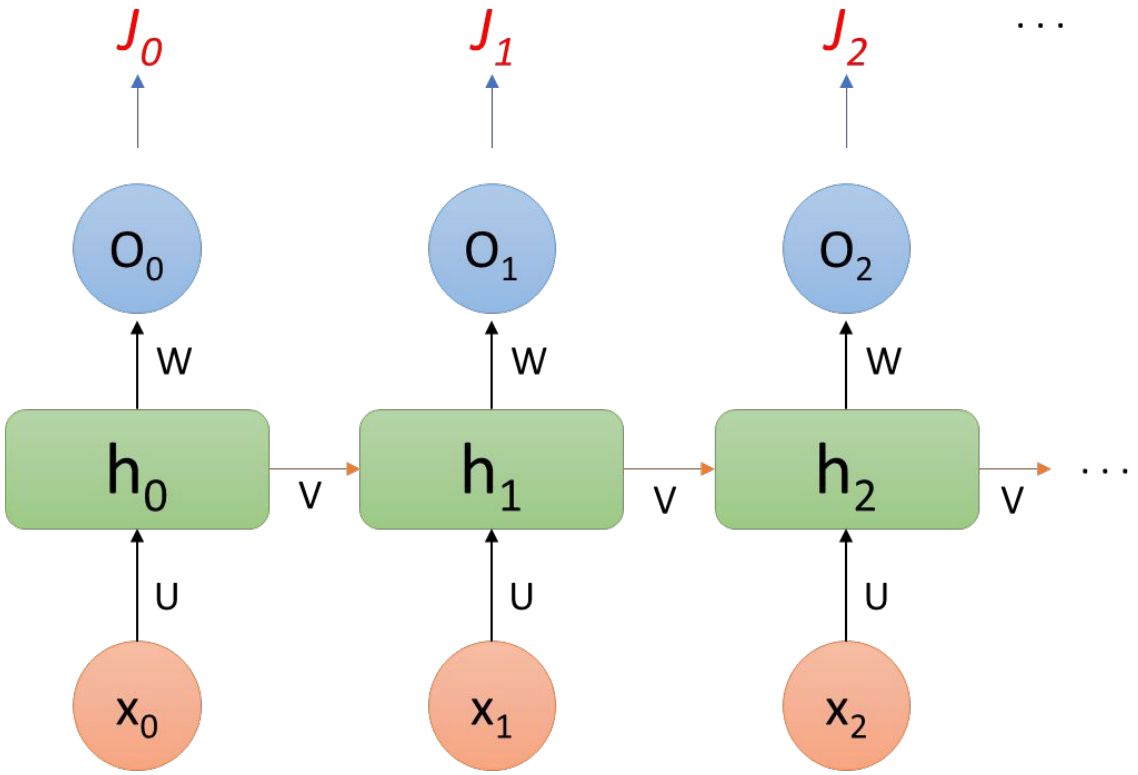


Notice:

h2 also depends on h1 and h1 depends on h0

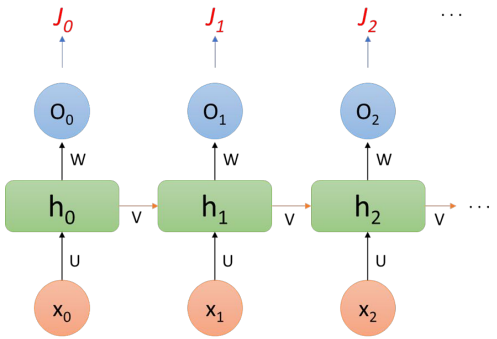
Backpropagation

- How does h_2 depend on U?



$$\frac{\delta h_2}{\delta U} + \frac{\delta h_2}{\delta h_1}\frac{\delta h_1}{\delta U} + \frac{\delta h_2}{\delta h_0}\frac{\delta h_0}{\delta U}$$

Backpropagation Through Time



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

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

$$+ \frac{\delta h_2}{\delta h_0}\frac{\delta h_0}{\delta U}$$

$$\frac{\delta J_2}{\delta U} = \frac{\delta J_2}{\delta O_2}\frac{\delta O_2}{\delta h_2}\boxed{\frac{\delta h_2}{\delta U}}$$

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

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

Backpropagation Through Time

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

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

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

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

Why Expensive?

Steps

1. Initialize weight matrices *U, V, W* from random distribution and bias b, c with zeros
2. Forward propagation to compute predictions
3. Compute the loss
4. Backpropagation to compute gradients
5. Update weights based on gradients
6. Repeat steps 2–5

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.
  - 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

Skip Connections

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.

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

- Recurrent Neural Network (RNN)
- RNN – Architecture
- RNN – Training
- 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
  - https://www.bioinf.jku.at/publications/older/2604.pdf

Thank You