CRIPAC
智能感知与计算研究中心
Center for Research on Intelligent Perception and Computing

中国科学院自动化研究所
Institute of Automation
Chinese Academy of Sciences

2023

"Deep Learning Lecture"

Lecture 5: Recurrent Neural Network

Wang Liang

Center for Research on Intelligent Perception and Computing (CRIPAC)
State Key Laboratory of Multimodal Artificial Intelligence Systems(MAIS)
Institute of Automation, Chinese Academy of Science (CASIA)

Outline

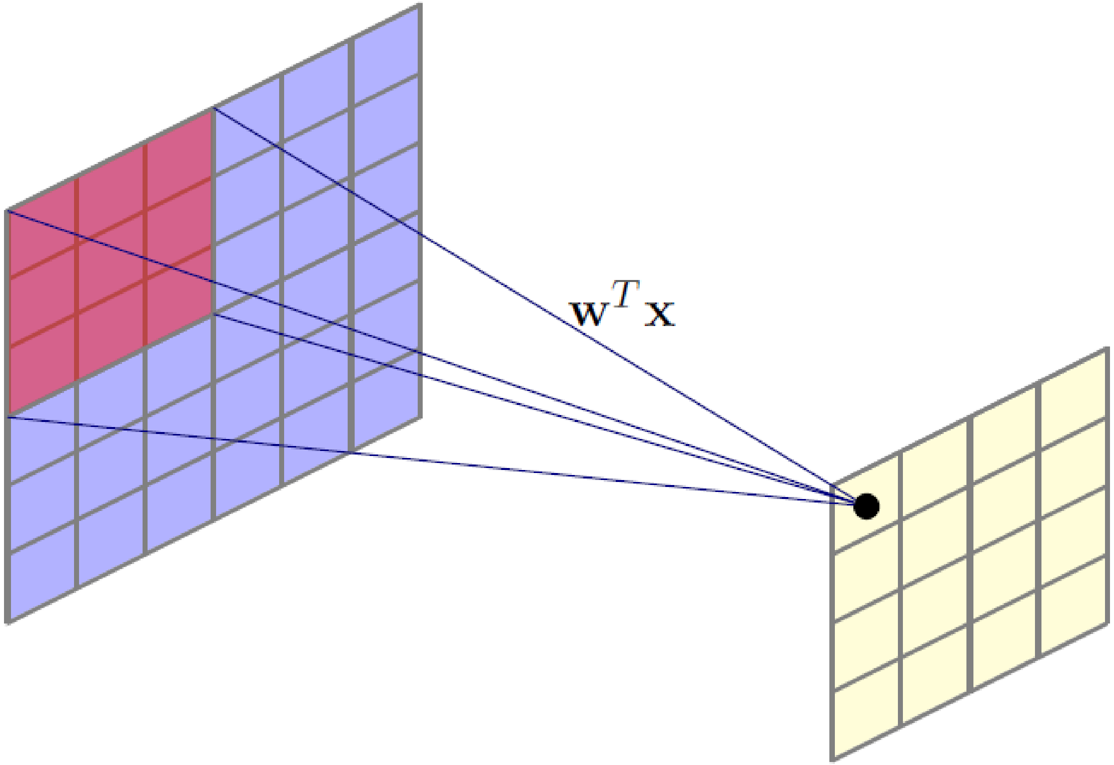
1 Course Review

2 Basic Recurrent Neural Network (RNN)

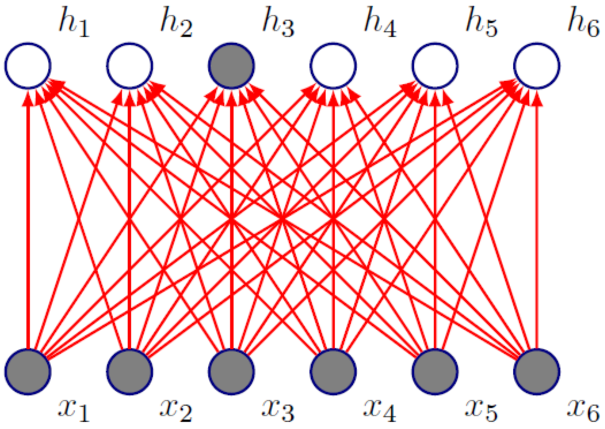
3 Variants of RNN

4 Applications

Review: Convolution

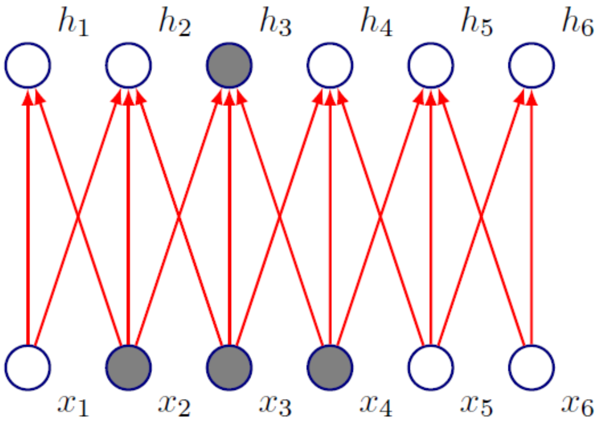


Review: Convolution-Sparse Connectivity

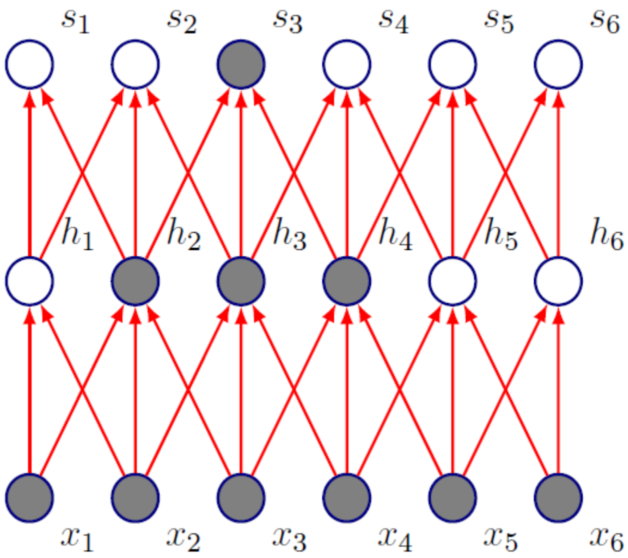


Fully connected network

Vs

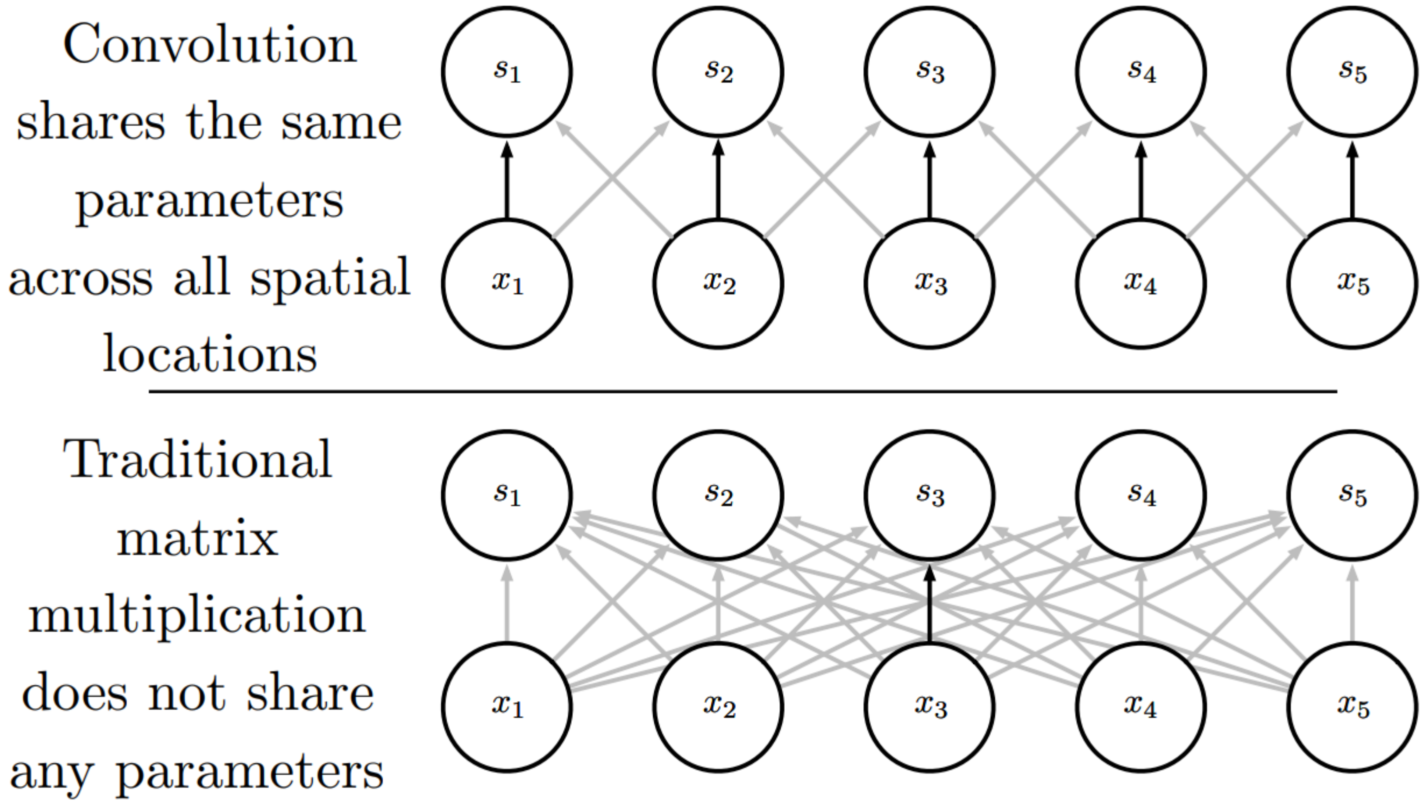


Sparse connectivity



→ Connections in CNNs are sparse, but units in deeper layers are connected to all of the input (larger receptive field sizes)

Review: Convolution-Parameter Sharing

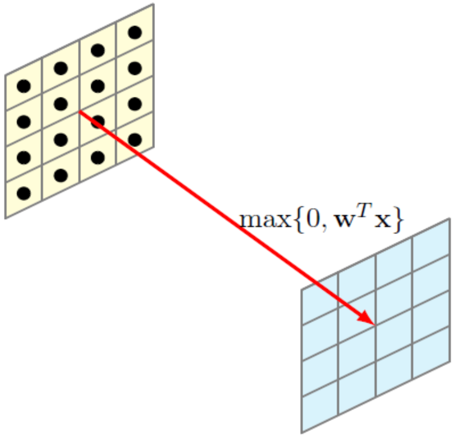

Convolution shares the same parameters across all spatial locations
s_1
s_2
s_3
s_4
s_5
x_1
x_2
x_3
x_4
x_5
Traditional matrix multiplication does not share any parameters
s_1
s_2
s_3
s_4
s_5
x_1
x_2
x_3
x_4
x_5

Review: Other Basic Operations

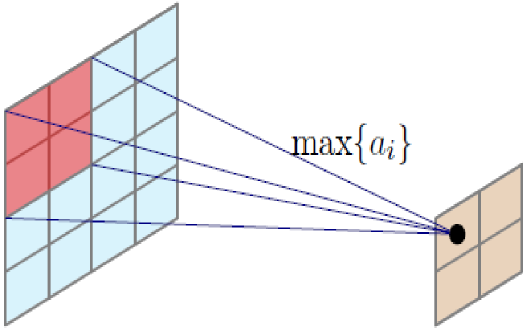
Zero padding

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

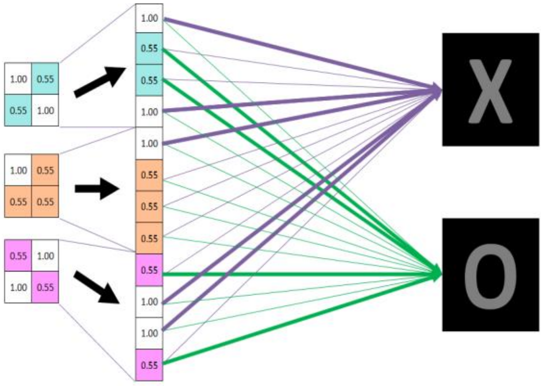
Non-Linearity



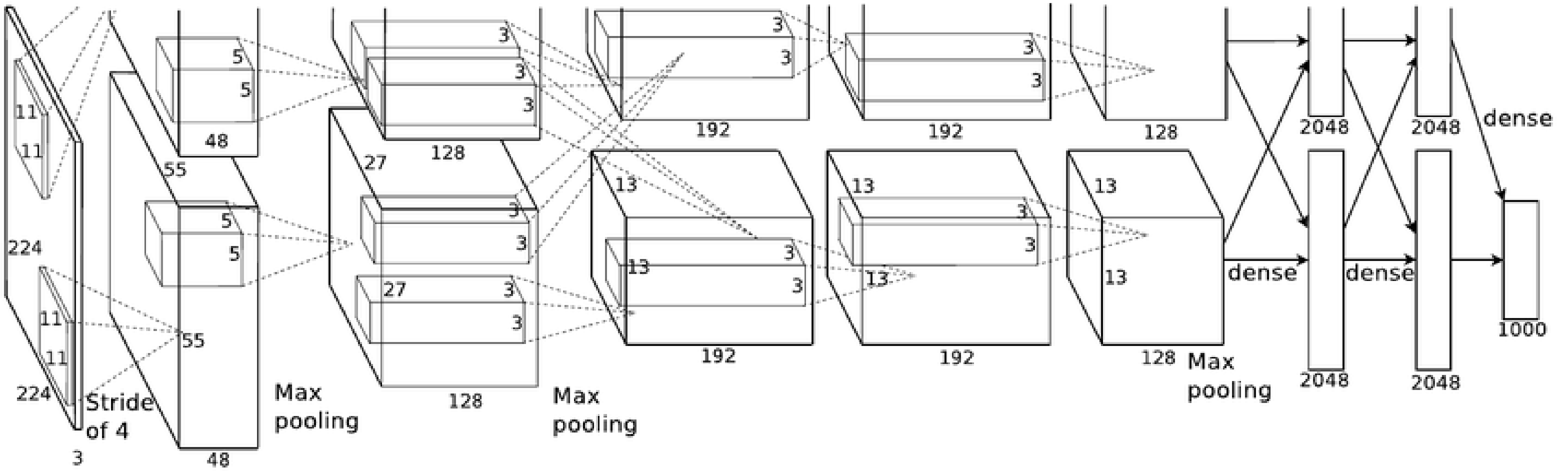
Pooling



Fully connected Layers



Review: Major Architecture-AlexNet

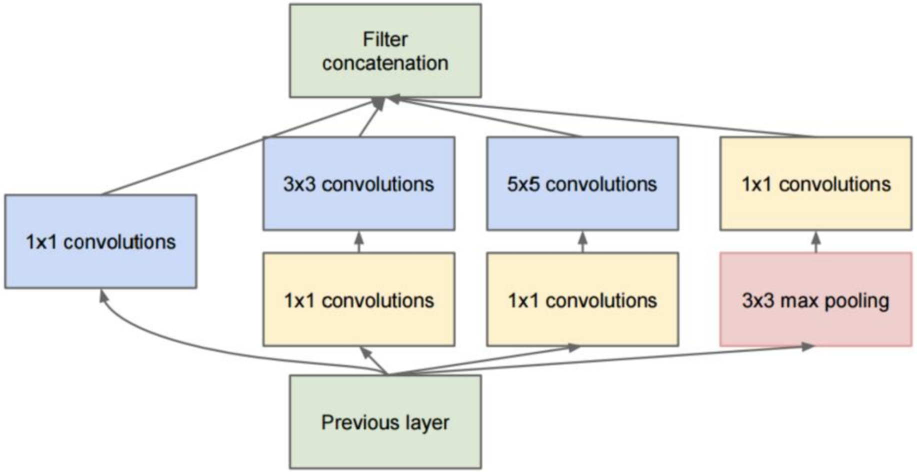


- Max pooling, ReLU nonlinearity
- More data and bigger model (7 hidden layers, 650K units, 60M params)
- GPU implementation (50x speedup over CPU), trained on two GPUs for a week
- Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

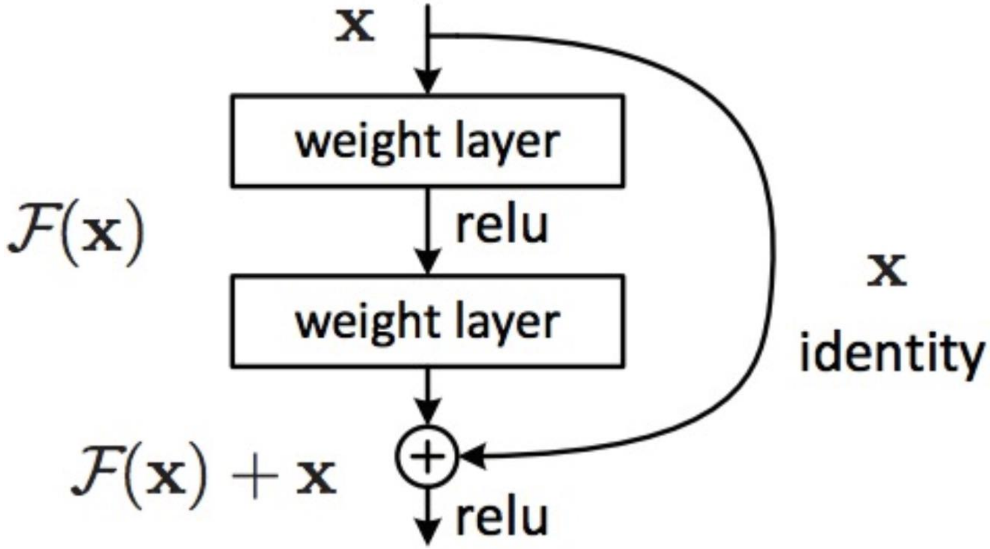
Review: Major Architecture-GoogLeNet

- The Inception Module
  - Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps
  - Use 1x1 convolutions for dimensionality reduction before expensive convolutions



C. Szegedy et al., Going deeper with convolutions, CVPR 2015

Review: Major Architecture-ResNet


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

- The residual module
  - Introduce skip or shortcut connections (existing before in various forms in literature)
  - Make it easy for network layers to represent the identity mapping
  - Need to skip at least two layers

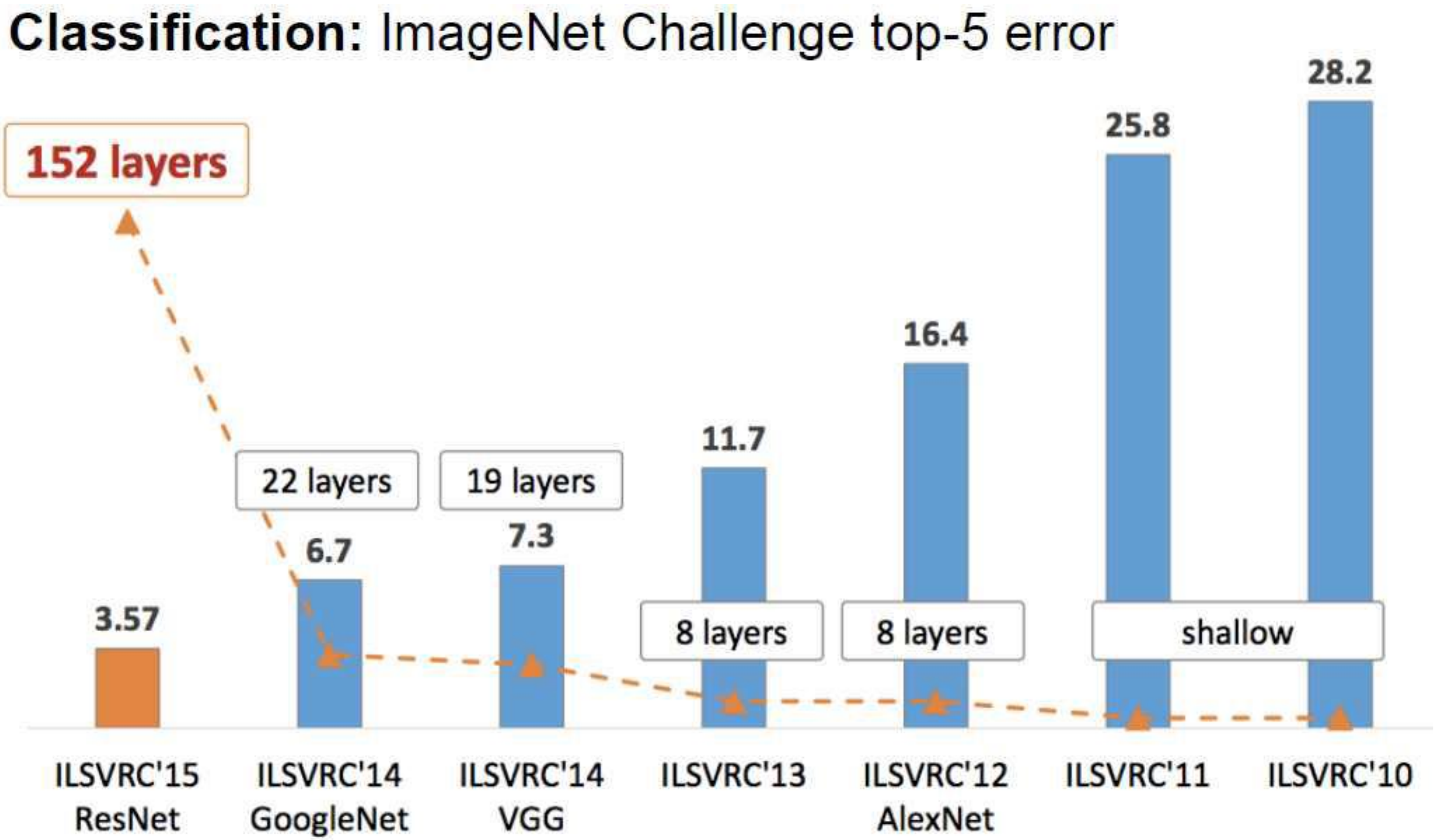
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

Review: Image Classification

- **ImageNet**
  - 1,000 classes
  - 1.28 million training images
  - 50k validation images
  - 100k test images

1000 object classes that we recognize
poster created by Fengjun Lv using VIPBase
images courtesy of ImageNet (http://www.image-net.org/challenges/LSVRC/2010/index)

Review: Image Classification



Going Deeper !

Outline

1 Course Review

2 Basic Recurrent Neural Network (RNN)

3 Variants of RNN

4 Applications

Motivation

- Not all problems can be converted into one with fixed-length inputs and outputs

- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information
  - Simple case: Output YES if the number of 1s is even, else NO
    1000010101 – YES, 100011 – NO, …

- Hard/Impossible to choose a fixed context window
  - There can always be a new sample longer than anything seen

Motivation

- Recurrent Neural Networks take the previous output or hidden states as inputs

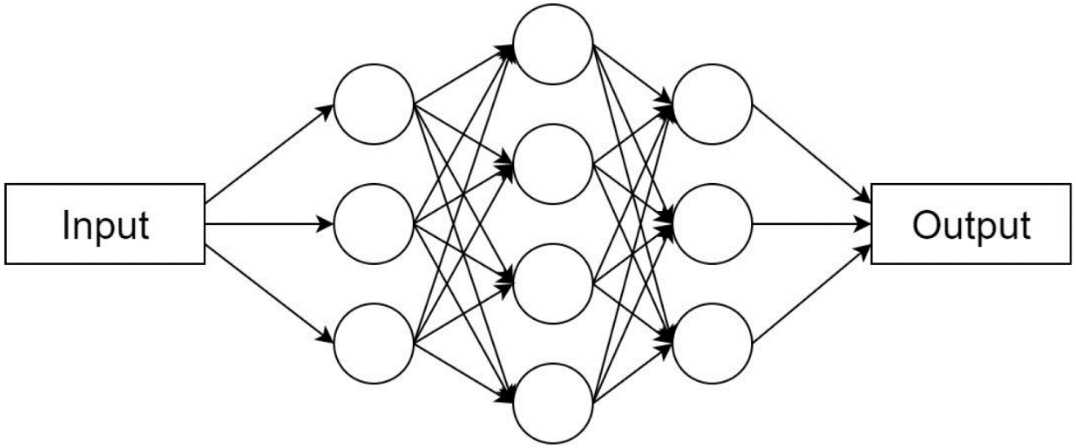
- The composite input at time T has some historical information about the happenings at time t < T

- RNNs are useful as their intermediate values (states) can store information about past inputs for a time that is not fixed

What is an RNN

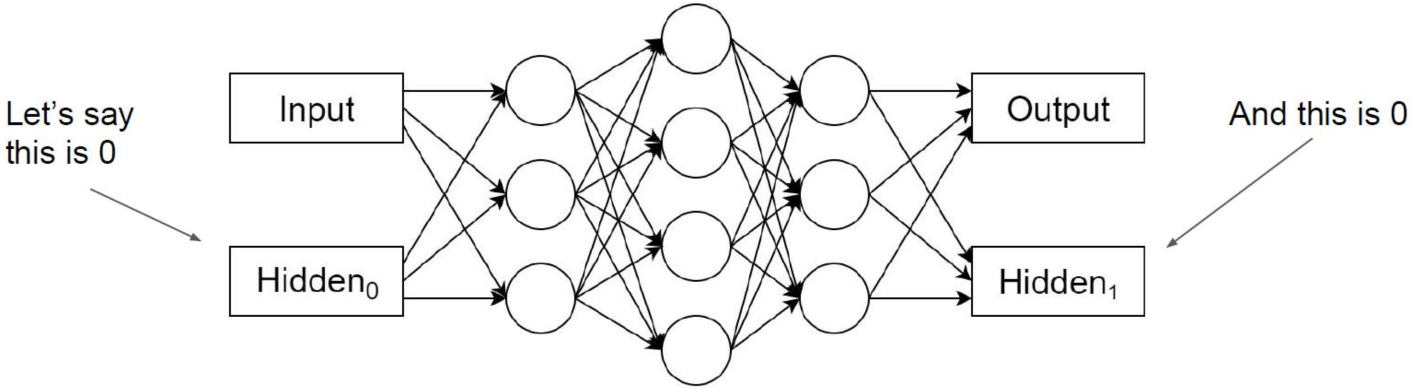
An RNN is a type of artificial neural network where the weights form a directed cycle

Let's take a step back to a typical feedforward NN to explain what this means…



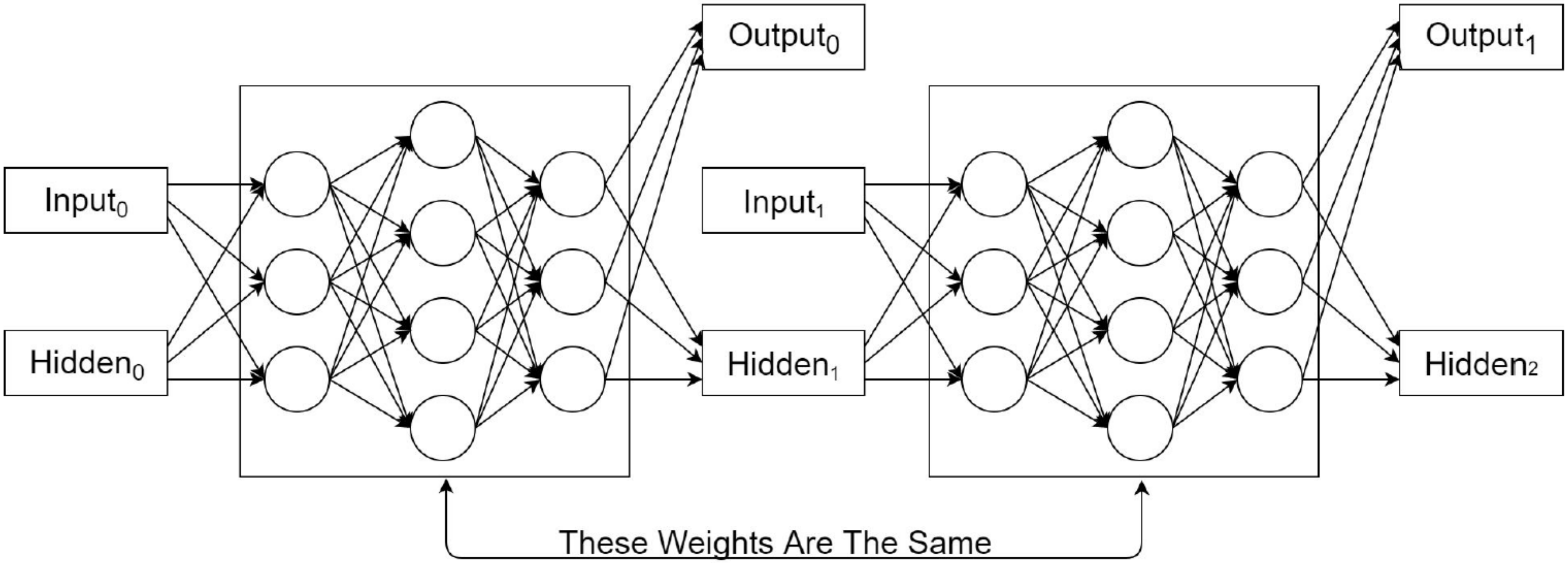
What is an RNN

An RNN is a type of artificial neural network where the weights form a directed cycle



What is an RNN

An RNN is a type of artificial neural network where the weights form a directed cycle



What is an RNN

- Recurrent Neural Networks (Rumelhart, 1986) are a family of neural networks for handling sequential data
- Sequential data: Each example consists of a pair of sequences. Each example can have different lengths
- Need to take advantage of an old idea in Machine Learning: Share parameters across different parts of a model
- Make it possible to extend the model to apply it to sequences of different lengths not seen during training
- Without parameter sharing it would not be possible to share statistical strength and generalize to lengths of sequences not seen during training
- Recurrent networks share parameters: Each output is a function of the previous outputs, with the same update rule applied

What is an RNN

- Consider the classical form of a dynamical system:

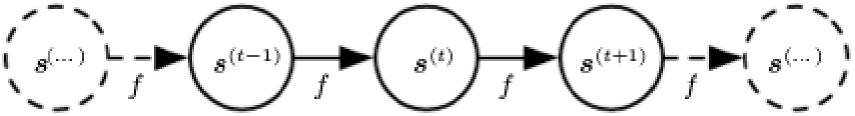
$$s^{(t)} = f(s^{(t-1)}; \theta)$$

- This is recurrent because the definition of s at time t refers back to the same definition at time $t - 1$
- For some finite number of time steps τ, the graph represented by this recurrence can be unfolded by using the definition τ-1 times. For example when τ = 3

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta)$$

- This expression does not involve any recurrence and can be represented by a traditional directed acyclic computational graph

What is an RNN



- Consider another dynamical system, that is driven by an external signal x$^{(t)}$

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

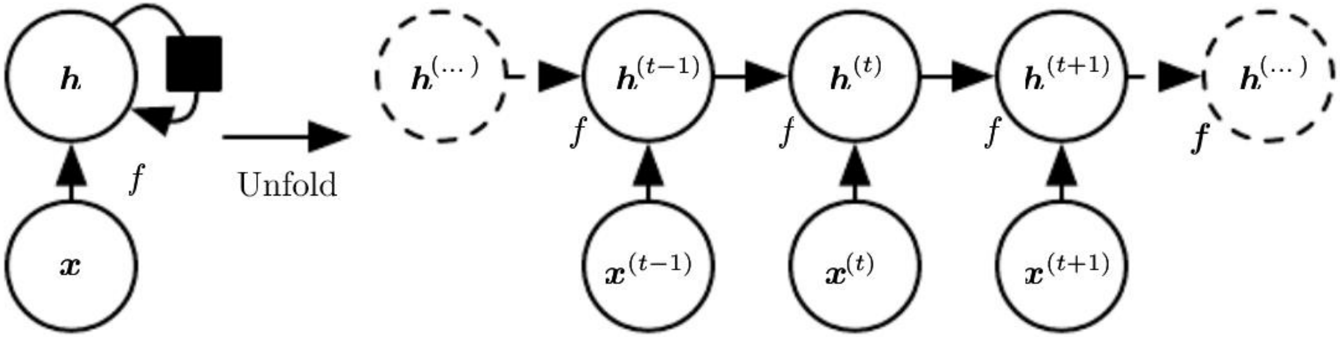
- The state now contains information about the whole past sequence
- RNNs can be built in various ways: Just as any function can be considered a feedforward network, any function involving a recurrence can be considered a recurrent neural network

What is an RNN

- We can consider the states to be the hidden units of the network, so we replace $s^{(t)}$ by $h^{(t)}$

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

- This system can be drawn in two ways:

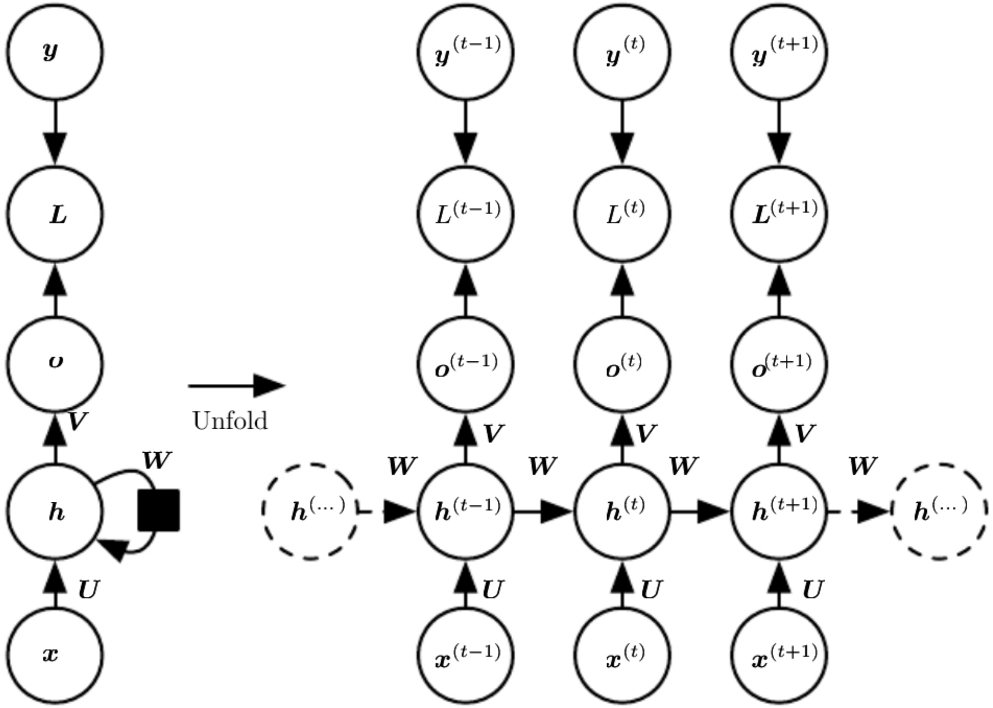


- We can have additional architectural features: Such as output layers that read information from h to make predictions

What is an RNN

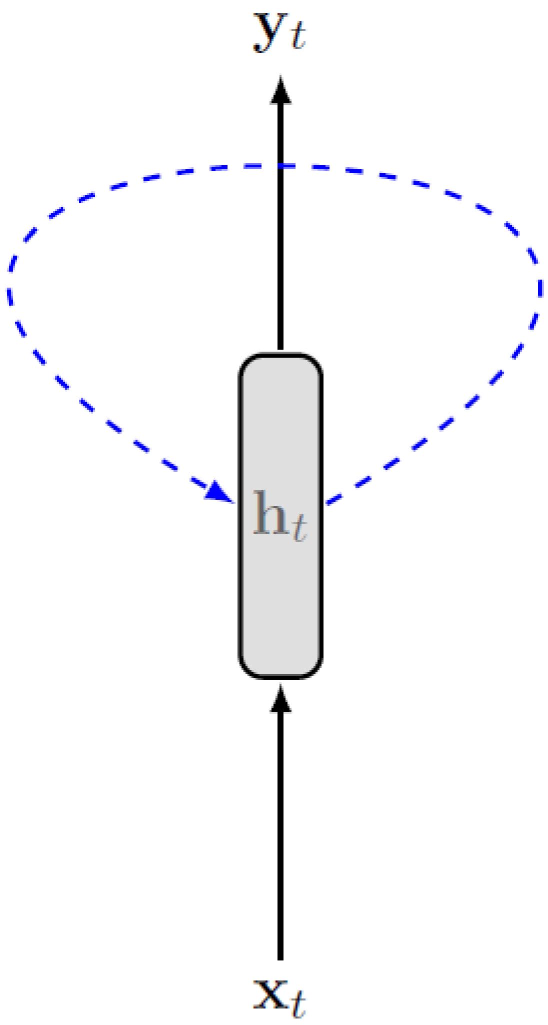
- When the task is to predict the future from the past, the network learns to use $h^{(t)}$ as a summary of task relevant aspects of the past sequence up to time t
- This summary is lossy because it maps an arbitrary length sequence ($x^{(t)}; x^{(t-1)}; \ldots ; x^{(2)}; x^{(1)}$) to a fixed vector $h^{(t)}$
- Depending on the training criterion, the summary might selectively keep some aspects of the past sequence with more precision (e.g. statistical language modeling)
- Most demanding situation for $h^{(t)}$: Approximately recover the input sequence

Plain Vanilla RNN

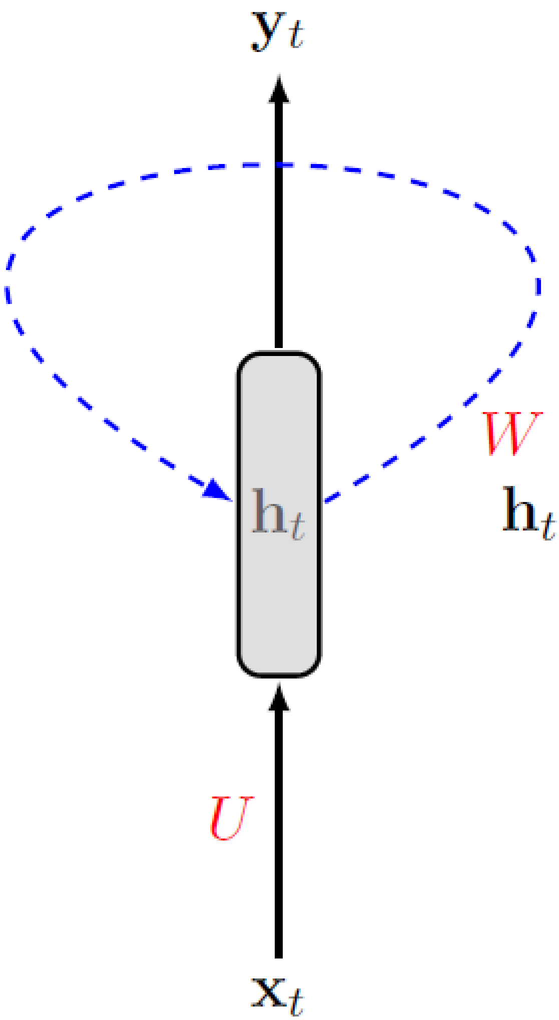


- Produce an output at each time step and have recurrent connections between hidden units
- Is in fact Turing Complete (Siegelmann, 1991, 1995, 1995)

Plain Vanilla RNN

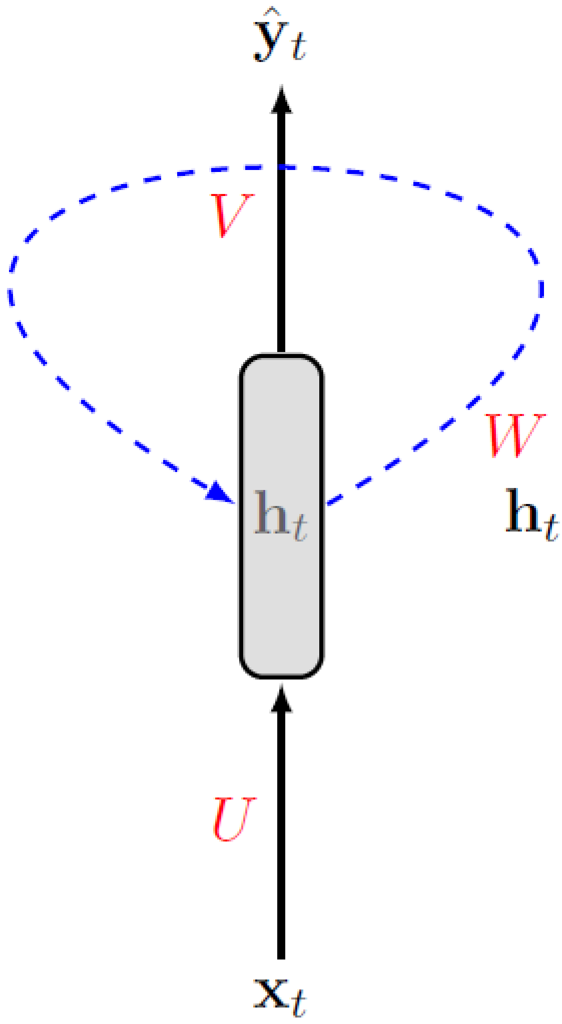


Recurrent Connections



$$\mathbf{h}_t = \psi(U\mathbf{x}_t + W\mathbf{h}_{t-1})$$

Recurrent Connections



$$\hat{\mathbf{y}}_t = \phi(V\mathbf{h}_t)$$

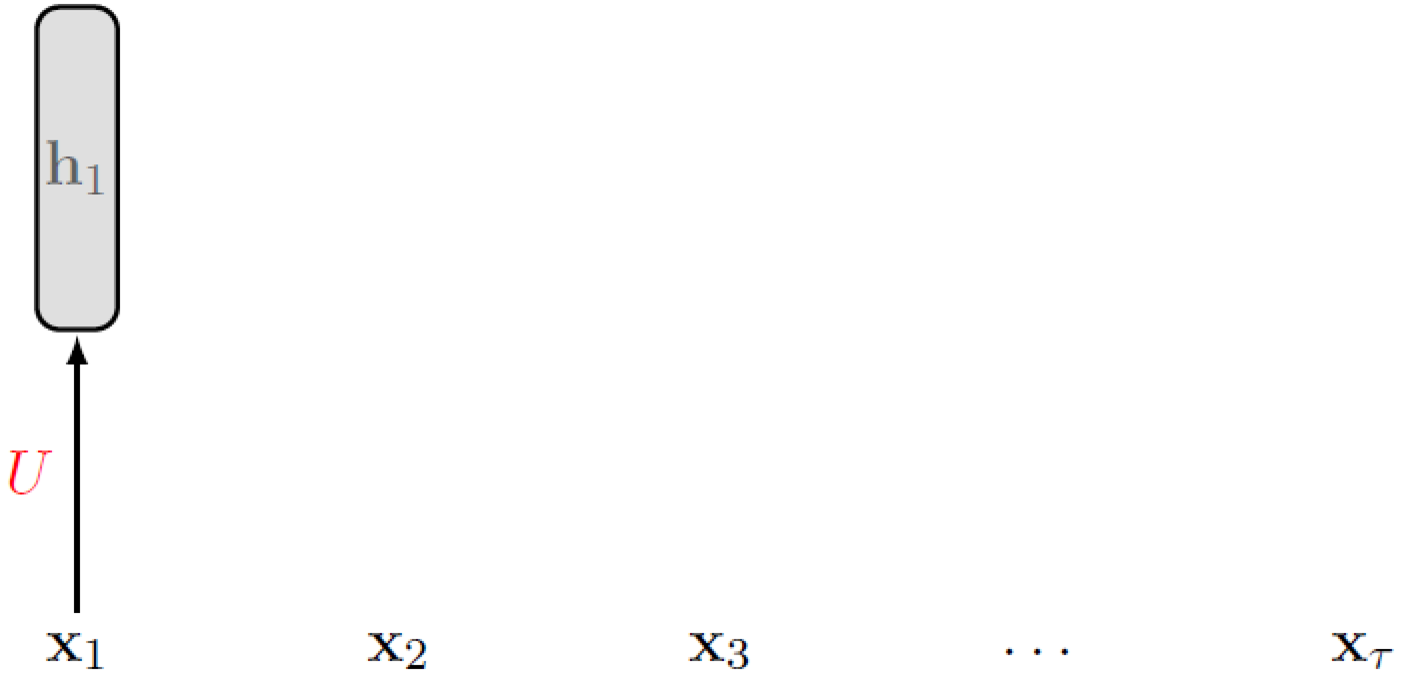
$$\mathbf{h}_t = \psi(U\mathbf{x}_t + W\mathbf{h}_{t-1})$$

ψ can be tanh and ϕ can be softmax

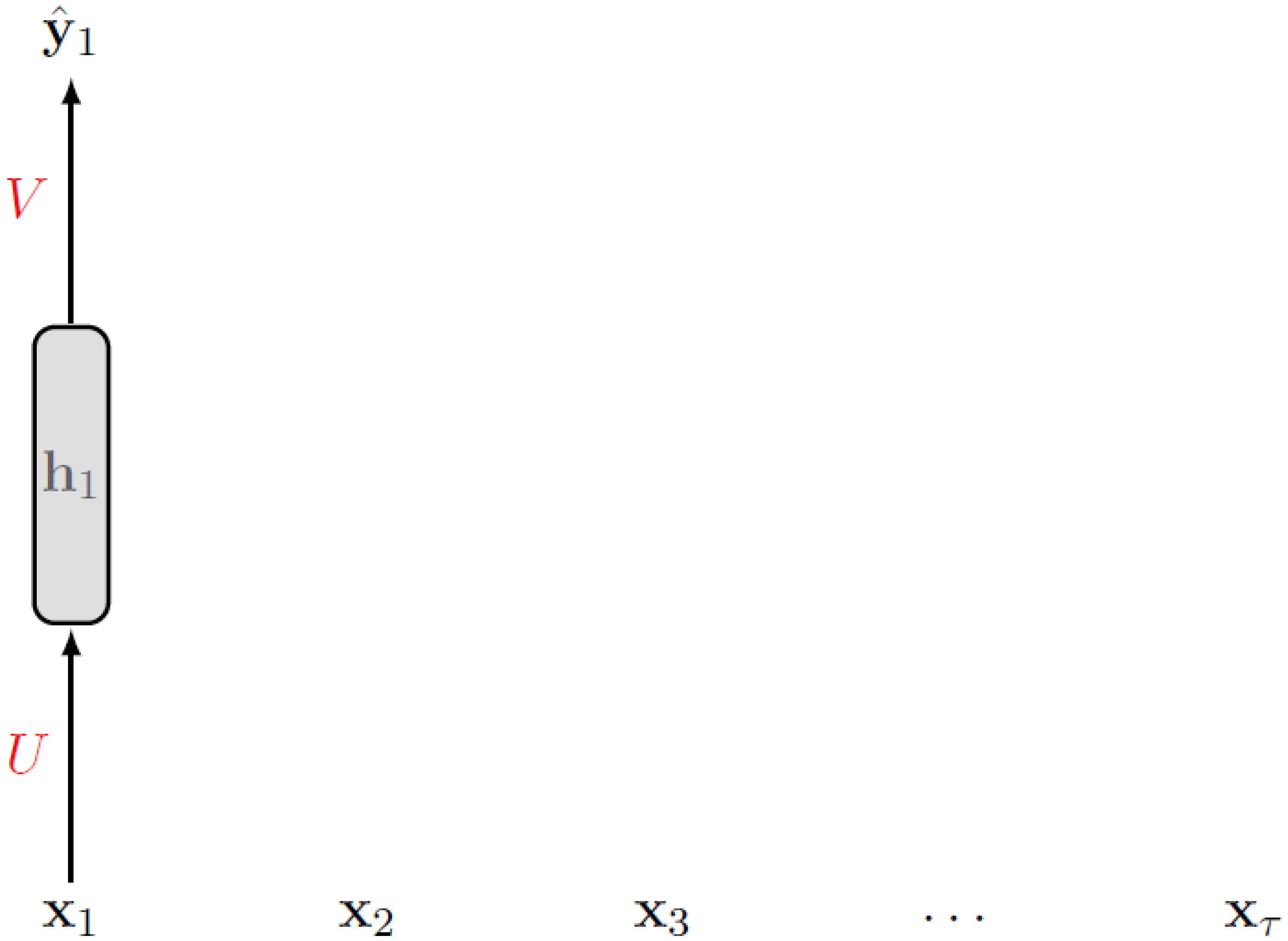
Unrolling the Recurrence

$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\cdots$ $\mathbf{x}_T$

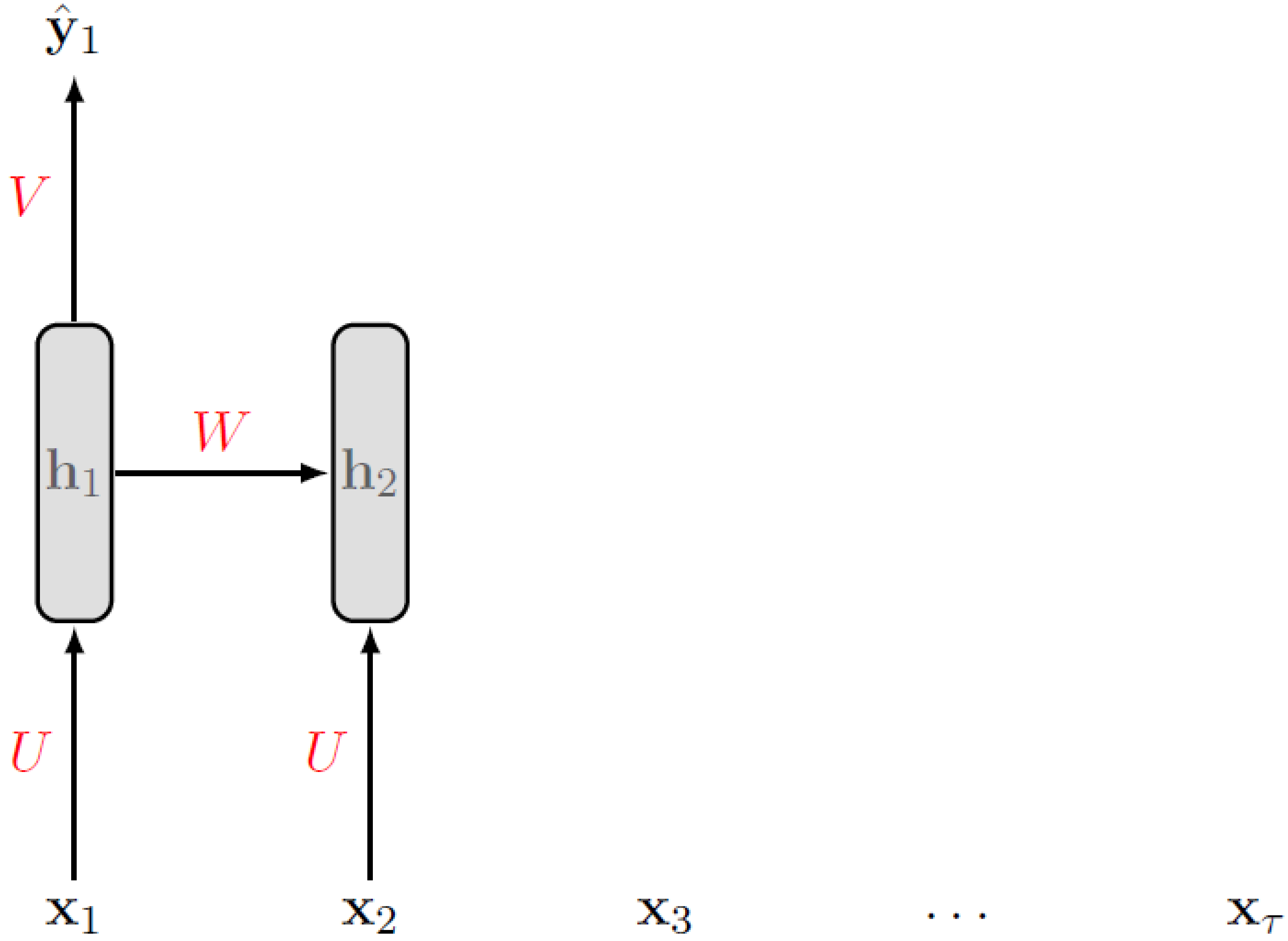
Unrolling the Recurrence



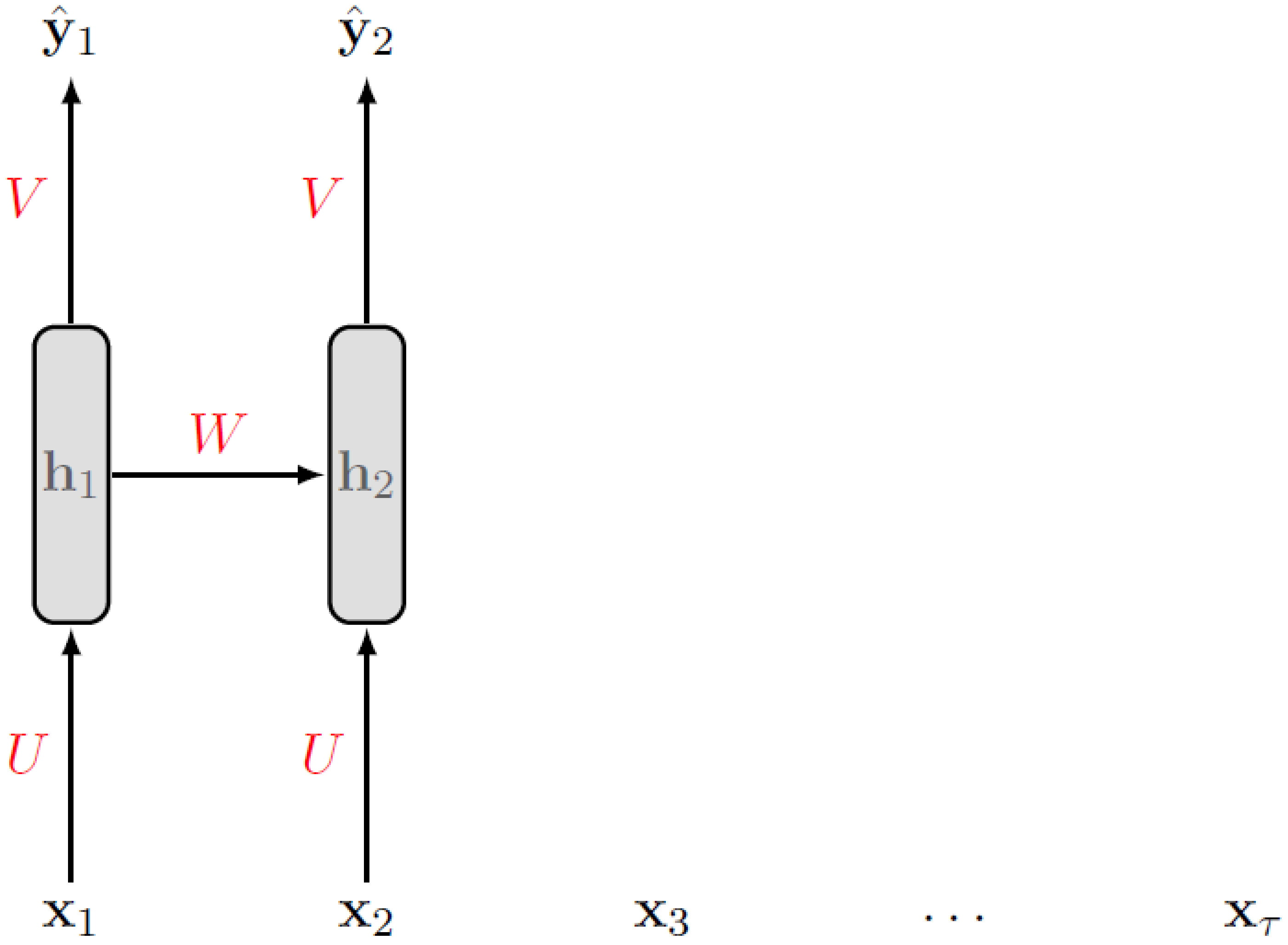
Unrolling the Recurrence



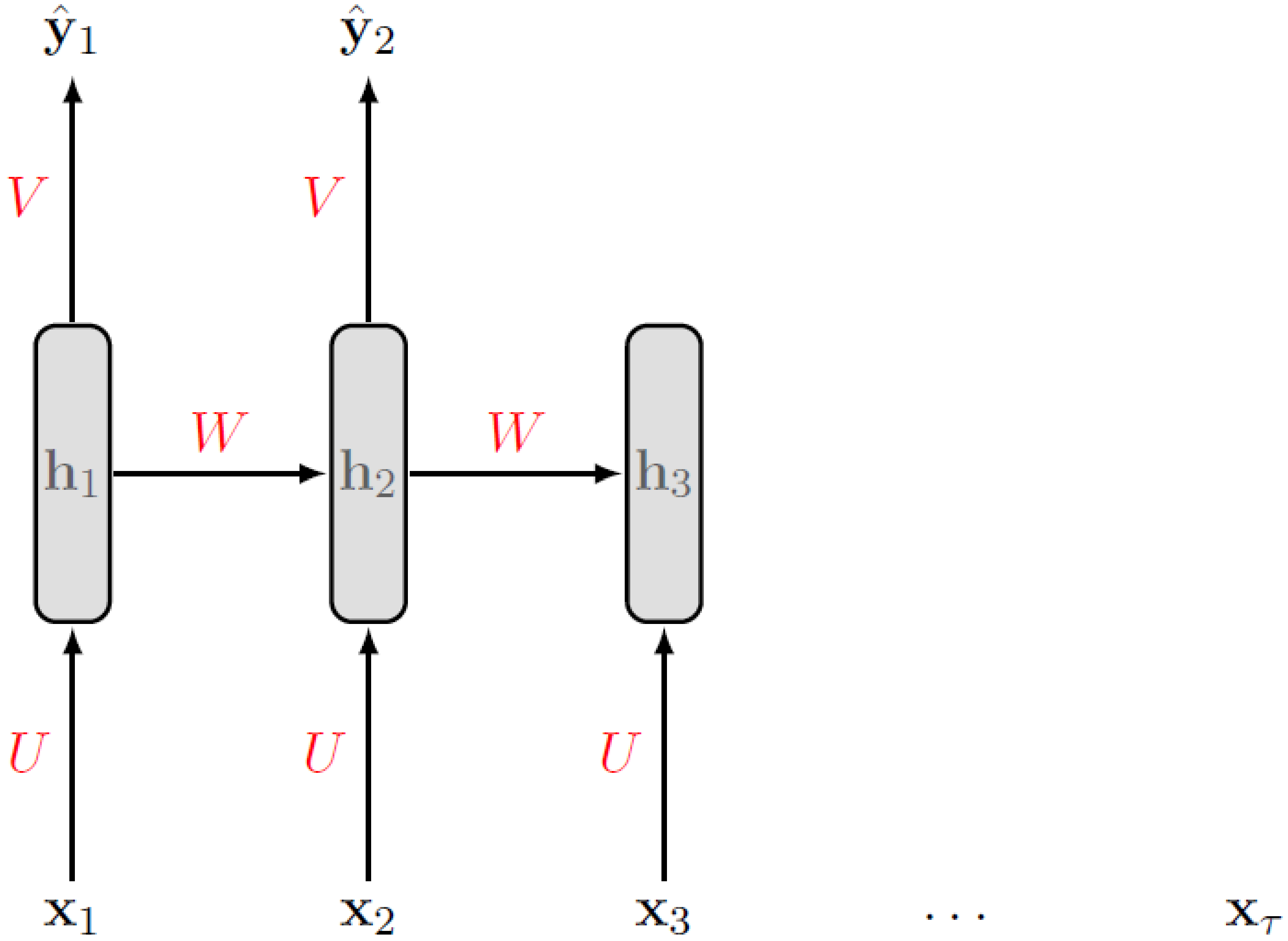
Unrolling the Recurrence



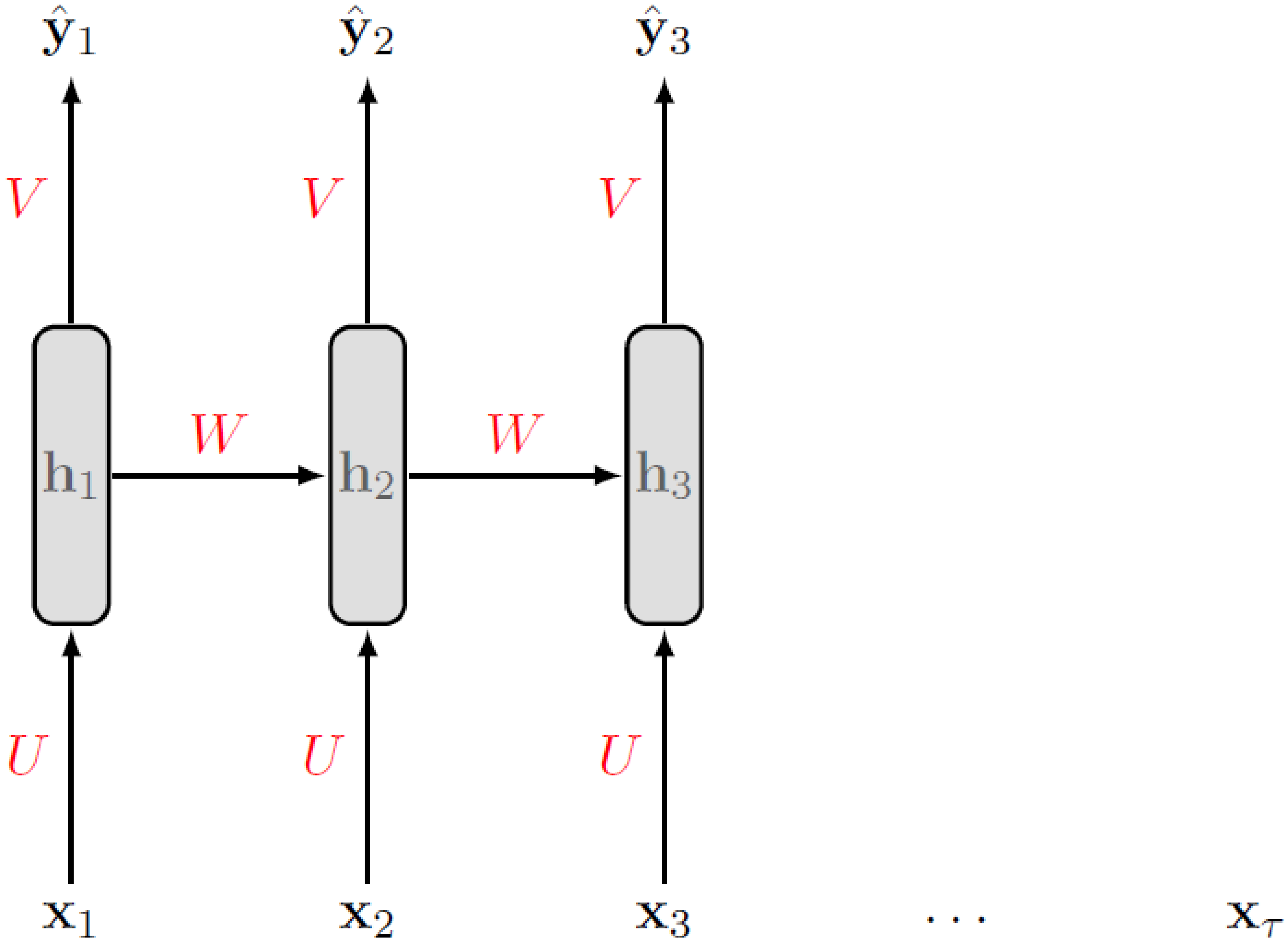
Unrolling the Recurrence



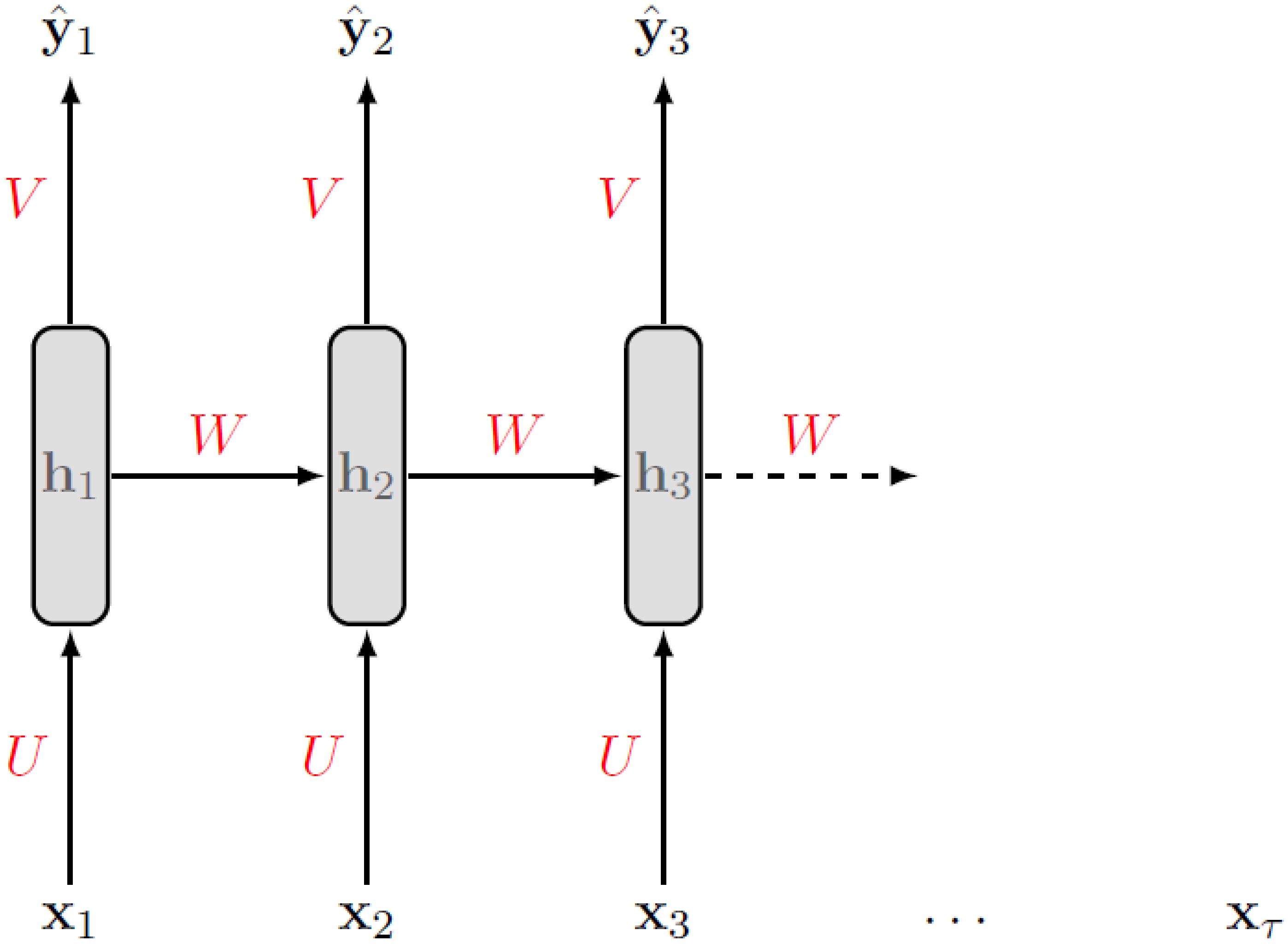
Unrolling the Recurrence



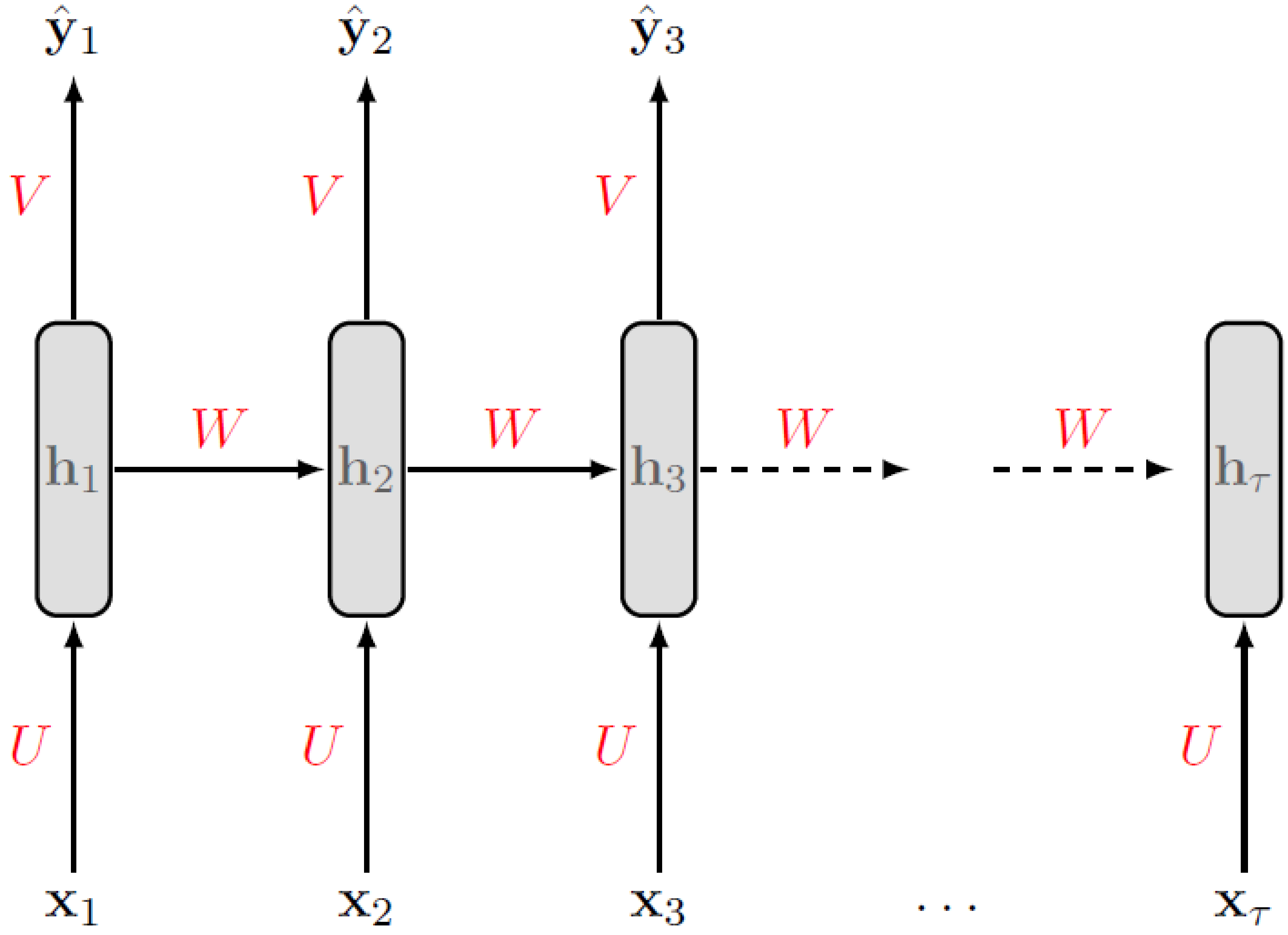
Unrolling the Recurrence



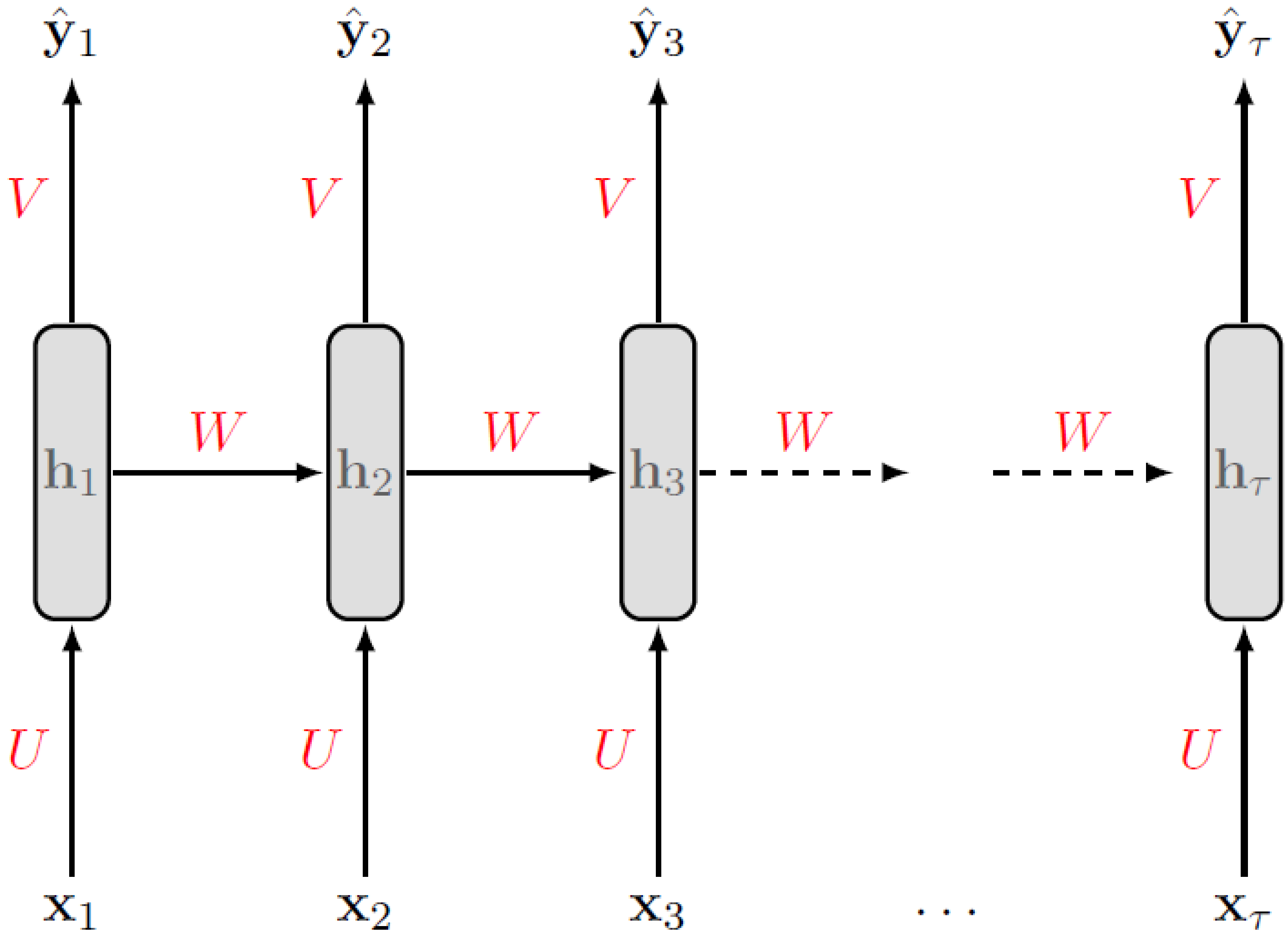
Unrolling the Recurrence



Unrolling the Recurrence



Unrolling the Recurrence



Feedforward Propagation

- This is a RNN where the input and output sequences are of the same length
- Feedforward operation proceeds from left to right
- Update Equations:

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

Feedforward Propagation

- Loss would just be the sum of losses over time steps
- If L_t is the negative log-likelihood of y_t given $x_1, \ldots, x_t$, then:

$$L\Big(\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}, \{\mathbf{y}_1, \ldots, \mathbf{y}_t\}\Big) = \sum_t L_t$$

- With:

$$\sum_t L_t = -\sum_t \log p_{\text{model}}\left(\mathbf{y}_t | \{\mathbf{x}_1, \ldots, \mathbf{x}_t\}\right)$$

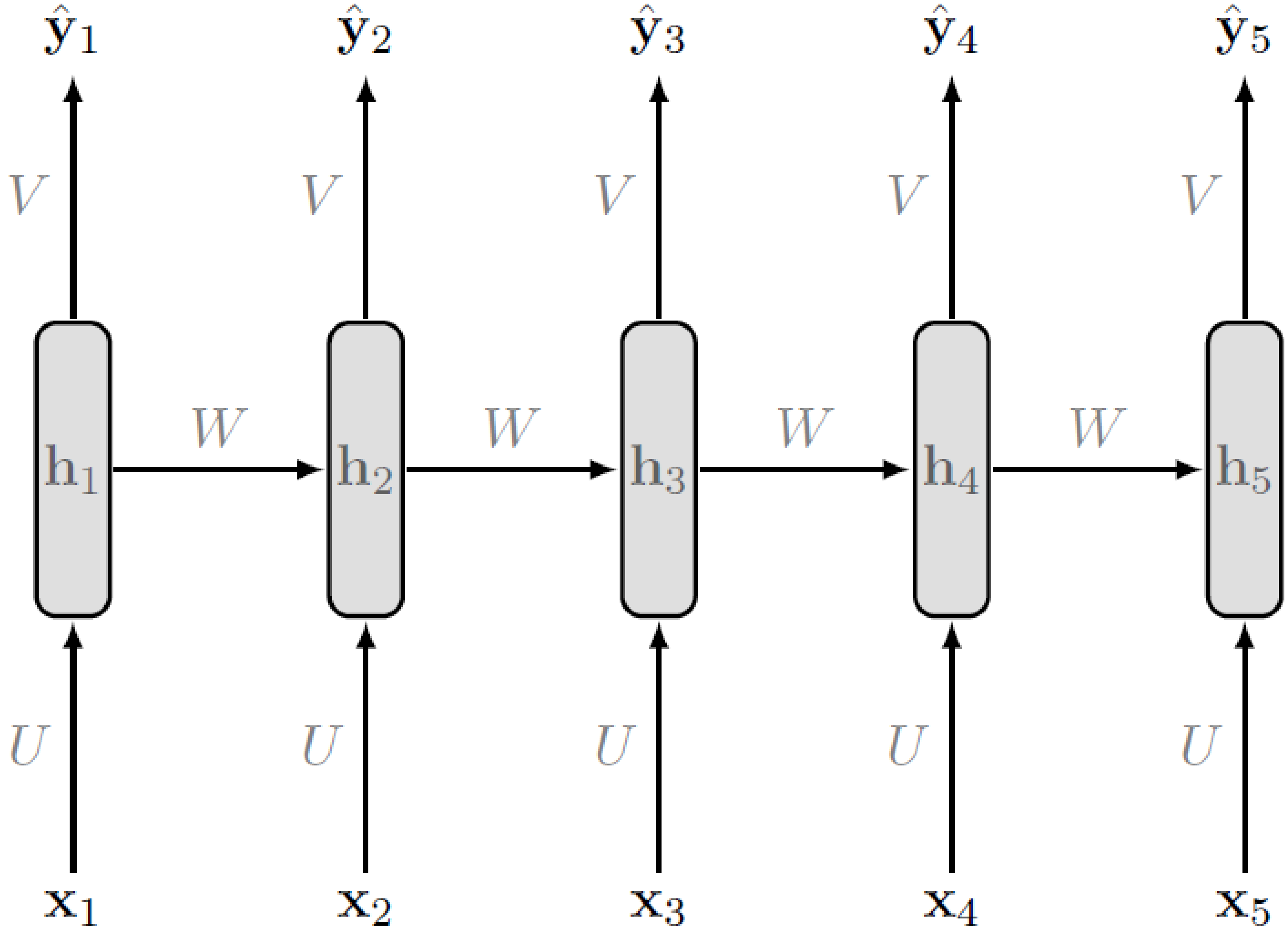
- Observation: Forward propagation takes time O(t)
- Can't be parallelized

Backward Propagation

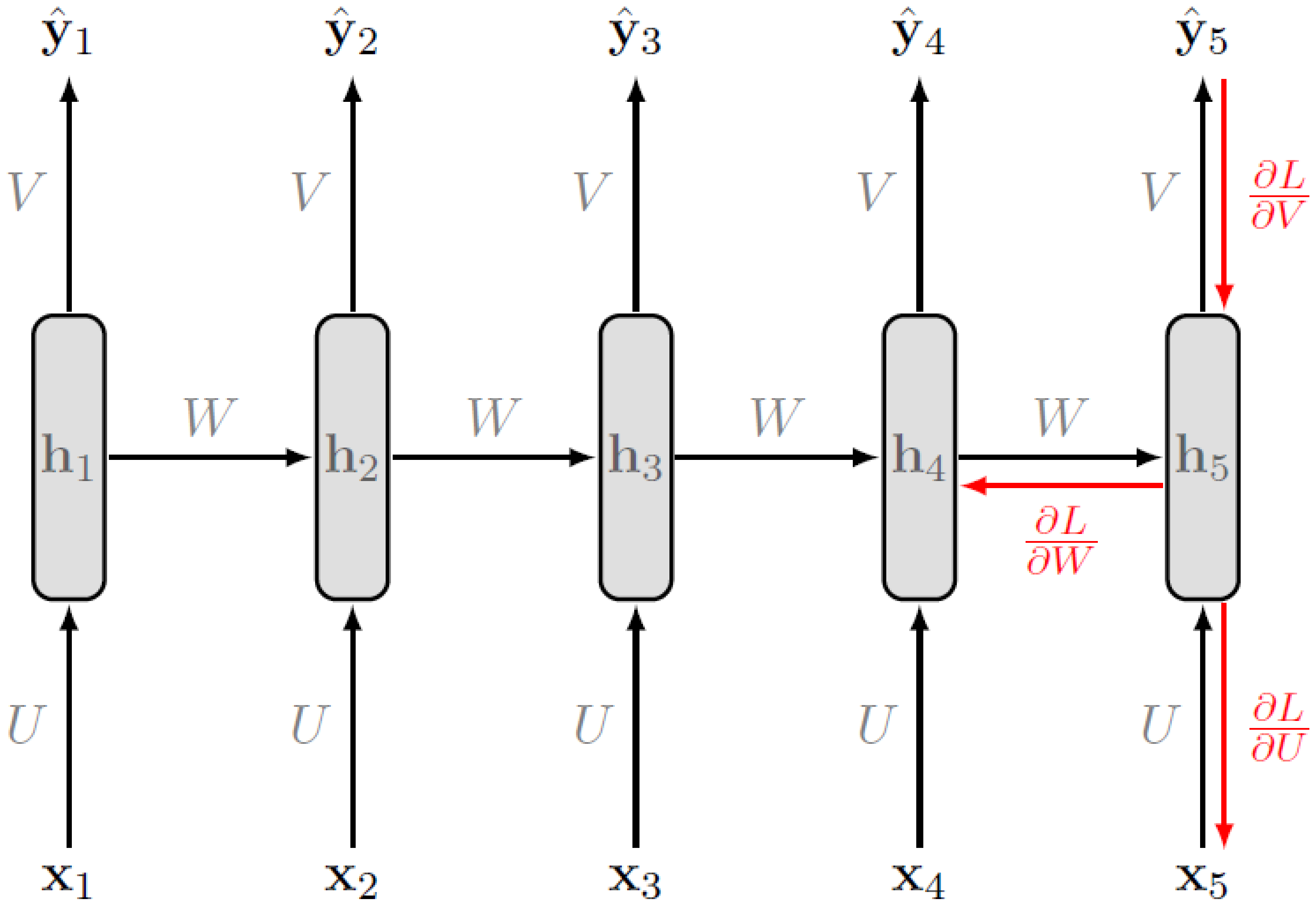
$$
\begin{aligned}
\mathbf{a}_t &= b + W\mathbf{h}_{t-1} + U\mathbf{x}_t \\
\mathbf{h}_t &= \tanh \mathbf{a}_t \\
\mathbf{o}_t &= c + V\mathbf{h}_t \\
\hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{o}_t)
\end{aligned}
$$

- Need to find: $\nabla_V L, \nabla_W L, \nabla_U L$
- And the gradients w.r.t biases: $\nabla_c L$ and $\nabla_b L$
- Treat the recurrent network as a usual multilayer network and apply back propagation on the unrolled network
- We move from the right to left: This is called Back Propagation Through Time (BPTT)
- Also takes time O(t)

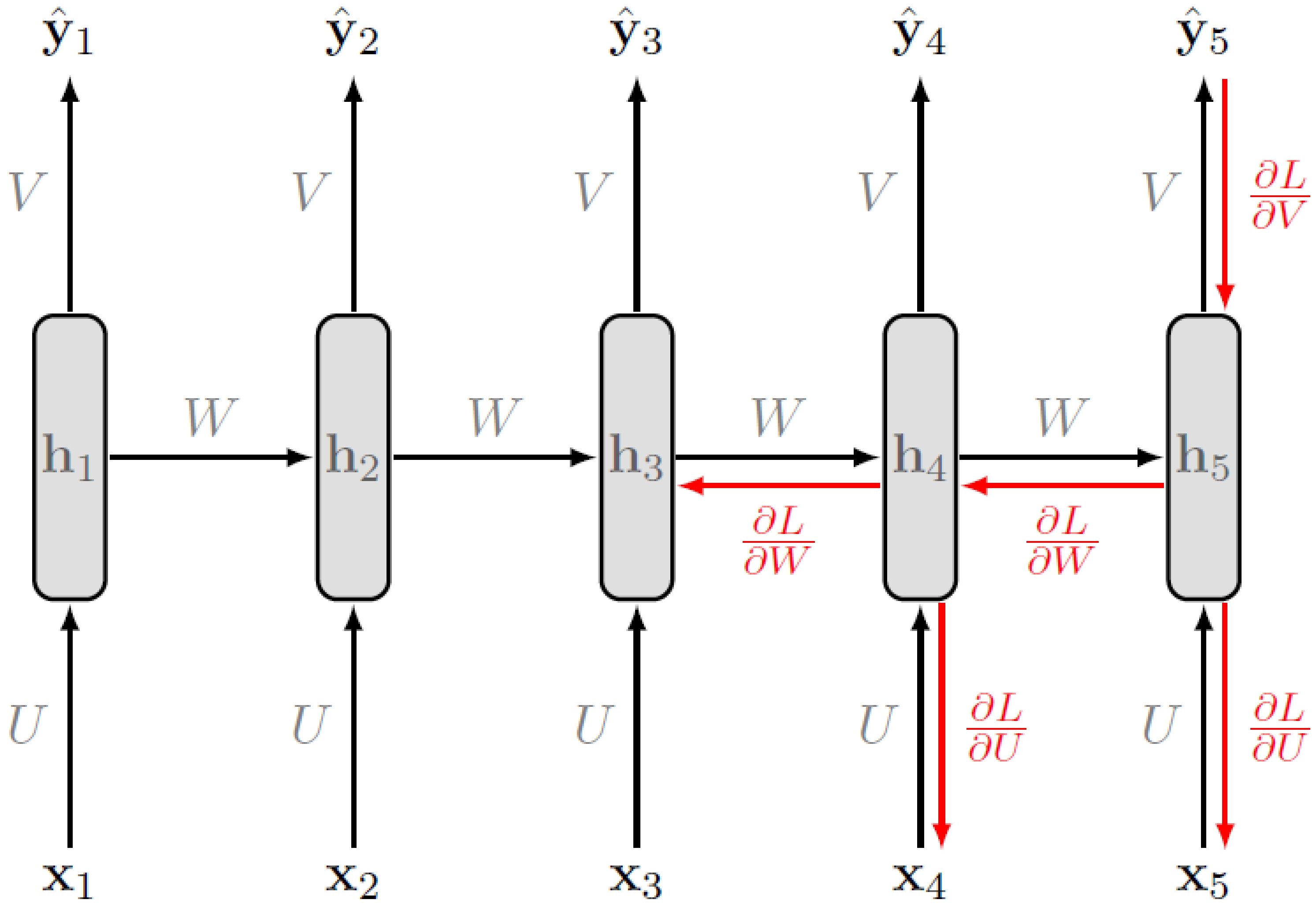
BPTT


$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\hat{\mathbf{y}}_5$
V
V
V
V
V
W
W
W
W
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_5$
U
U
U
U
U
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

BPTT

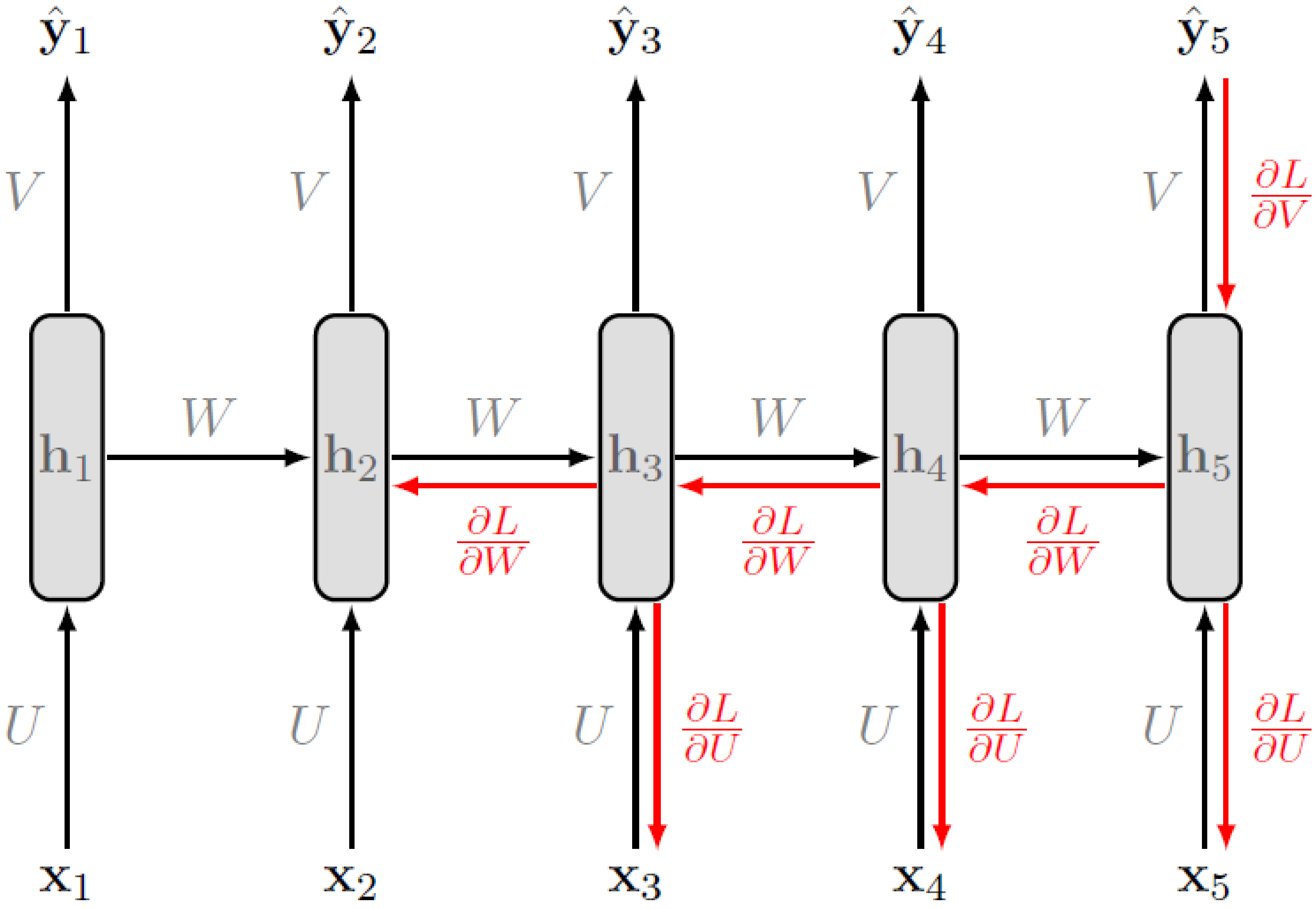

$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\hat{\mathbf{y}}_5$
V
V
V
V
V
$\frac{\partial L}{\partial V}$
$\mathbf{h}_1$
W
$\mathbf{h}_2$
W
$\mathbf{h}_3$
W
$\mathbf{h}_4$
W
$\mathbf{h}_5$
$\frac{\partial L}{\partial W}$
U
U
U
U
U
$\frac{\partial L}{\partial U}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

BPTT

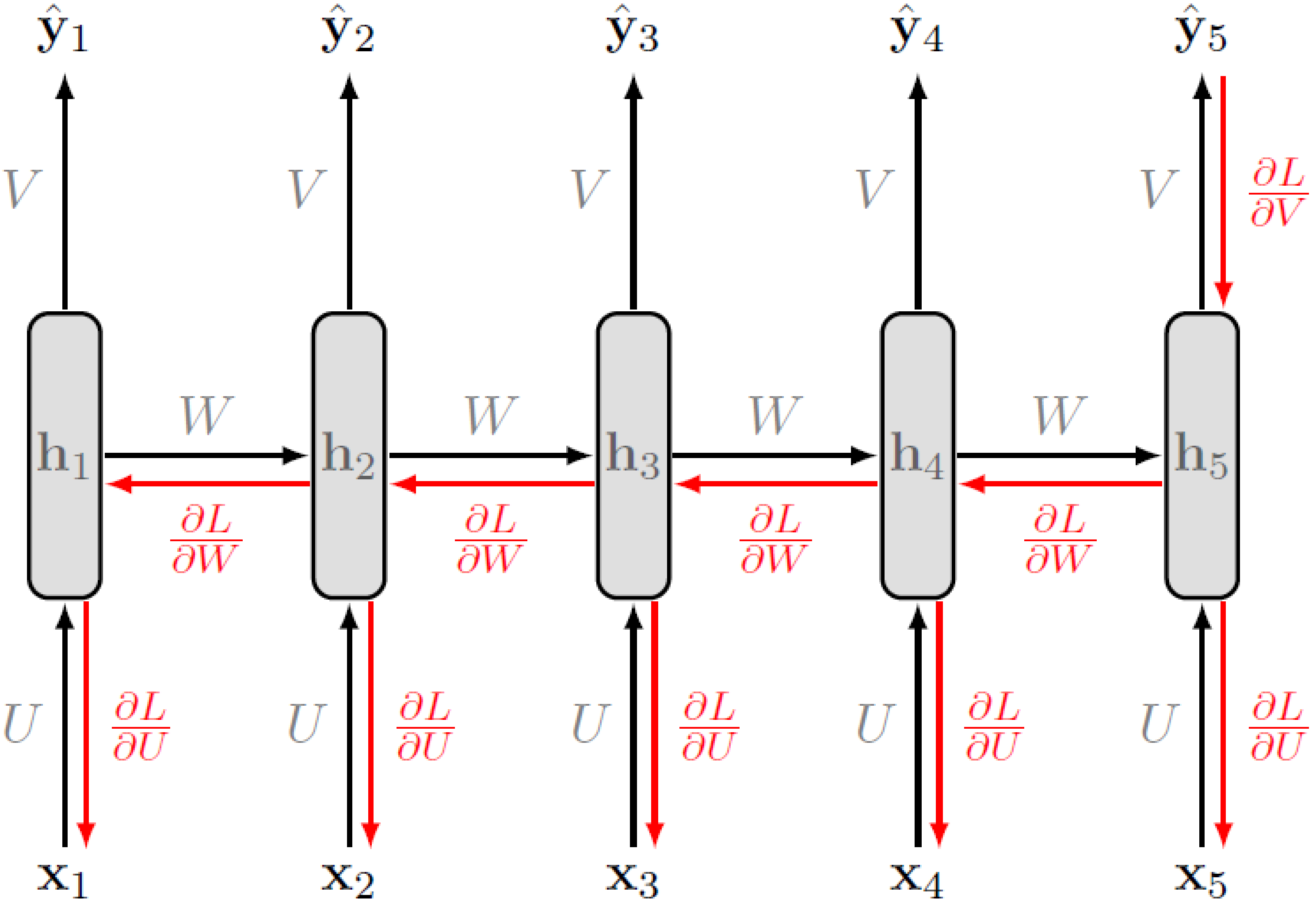


$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\hat{\mathbf{y}}_5$
V
V
V
V
V
$\frac{\partial L}{\partial V}$
$\mathbf{h}_1$
W
$\mathbf{h}_2$
W
$\mathbf{h}_3$
W
$\mathbf{h}_4$
W
$\mathbf{h}_5$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
U
U
U
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

BPTT


$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\hat{\mathbf{y}}_5$
V
V
V
V
V
$\frac{\partial L}{\partial V}$
$\mathbf{h}_1$
W
$\mathbf{h}_2$
W
$\mathbf{h}_3$
W
$\mathbf{h}_4$
W
$\mathbf{h}_5$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
U
U
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

BPTT



$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\hat{y}_5$
V
V
V
V
V
$\frac{\partial L}{\partial V}$
h_1
h_2
h_3
h_4
h_5
W
W
W
W
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
x_1
x_2
x_3
x_4
x_5

Gradient Computation

Refer to:

$$\begin{aligned}
\mathbf{a}_t &= b + W\mathbf{h}_{t-1} + U\mathbf{x}_t \\
\mathbf{h}_t &= \tanh \mathbf{a}_t \\
\mathbf{o}_t &= c + V\mathbf{h}_t \\
\hat{\mathbf{y}}_t &= \mathsf{softmax}(\mathbf{o}_t)
\end{aligned}$$

$$\nabla_V L = \sum (\nabla_{\mathbf{o}_t} L)\mathbf{h}_t^T$$

Where:

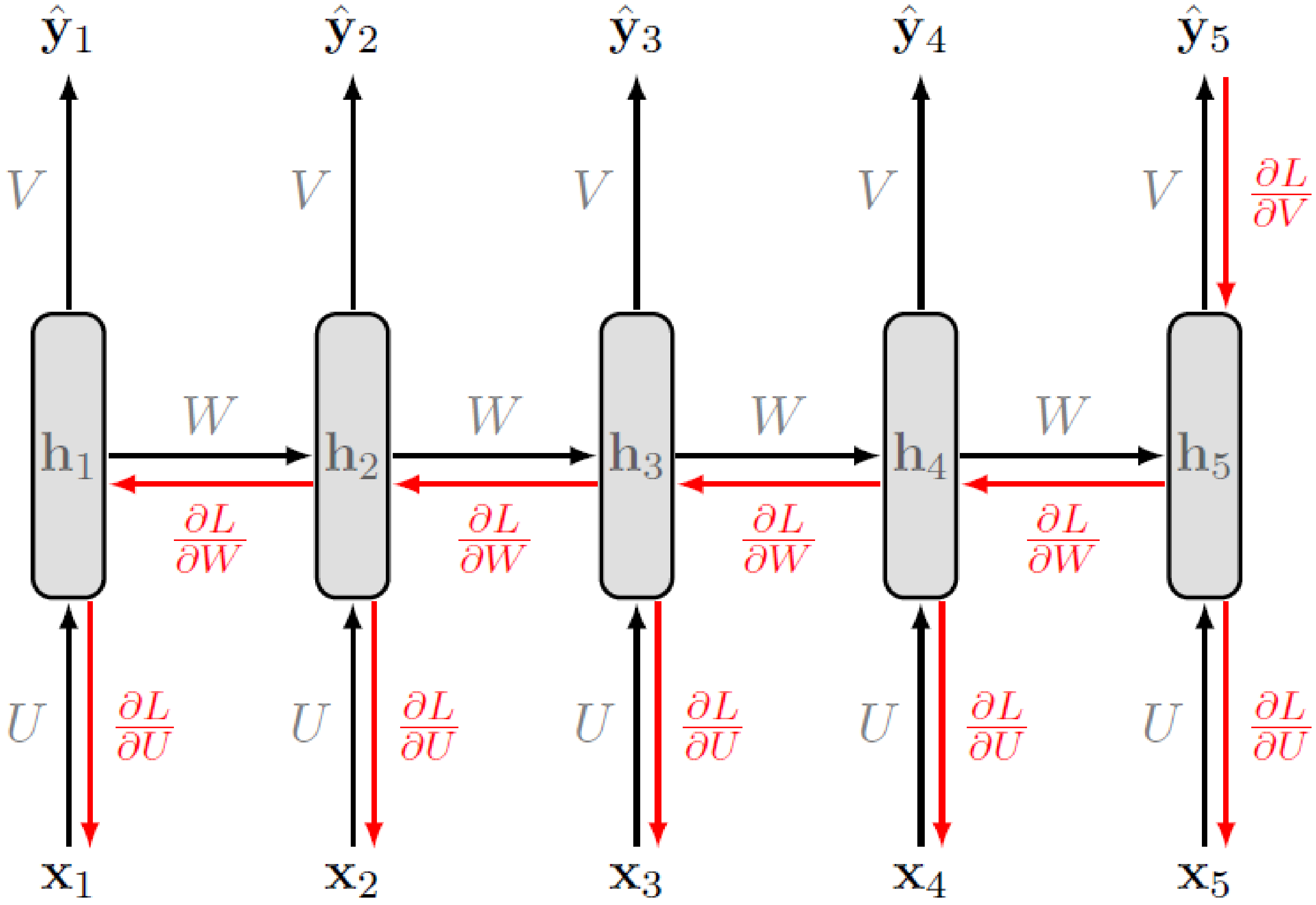
$$(\nabla_{\mathbf{o}_t} L)_i = \frac{\partial L}{\partial \mathbf{o}_t^{(i)}} = \frac{\partial L}{\partial L_t} \frac{\partial L_t}{\partial \mathbf{o}_t^{(i)}}$$

$$= \frac{\partial L_t}{\partial o_t^{(i)}} = \frac{\partial L_t}{\partial \hat{y}_t^{(i)}} \cdot \frac{\partial \hat{y}_t^{(i)}}{\partial o_t^{(i)}}$$

$$= \hat{\mathbf{y}}_t^{(i)} - \mathbf{1}_{i,\mathbf{y}_t}$$

More details：https://www.zerahhah.com/article/13

BPTT


$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\hat{\mathbf{y}}_5$
V
$\frac{\partial L}{\partial V}$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_5$
W
$\frac{\partial L}{\partial W}$
U
$\frac{\partial L}{\partial U}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

Gradient Computation

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

$$\tanh'(s) = 1 - \tanh^2(s)$$

$$\nabla_W L = \sum_t \overbrace{diag\left(1-(\mathbf{h}_t)^2\right)}^{\frac{\partial h_t}{\partial a_t}} \overbrace{(\nabla_{\mathbf{h}_t} L)}^{\frac{\partial L}{\partial h_t}} \overbrace{\mathbf{h}_{t-1}^T}^{\frac{\partial a_t}{\partial W}}$$

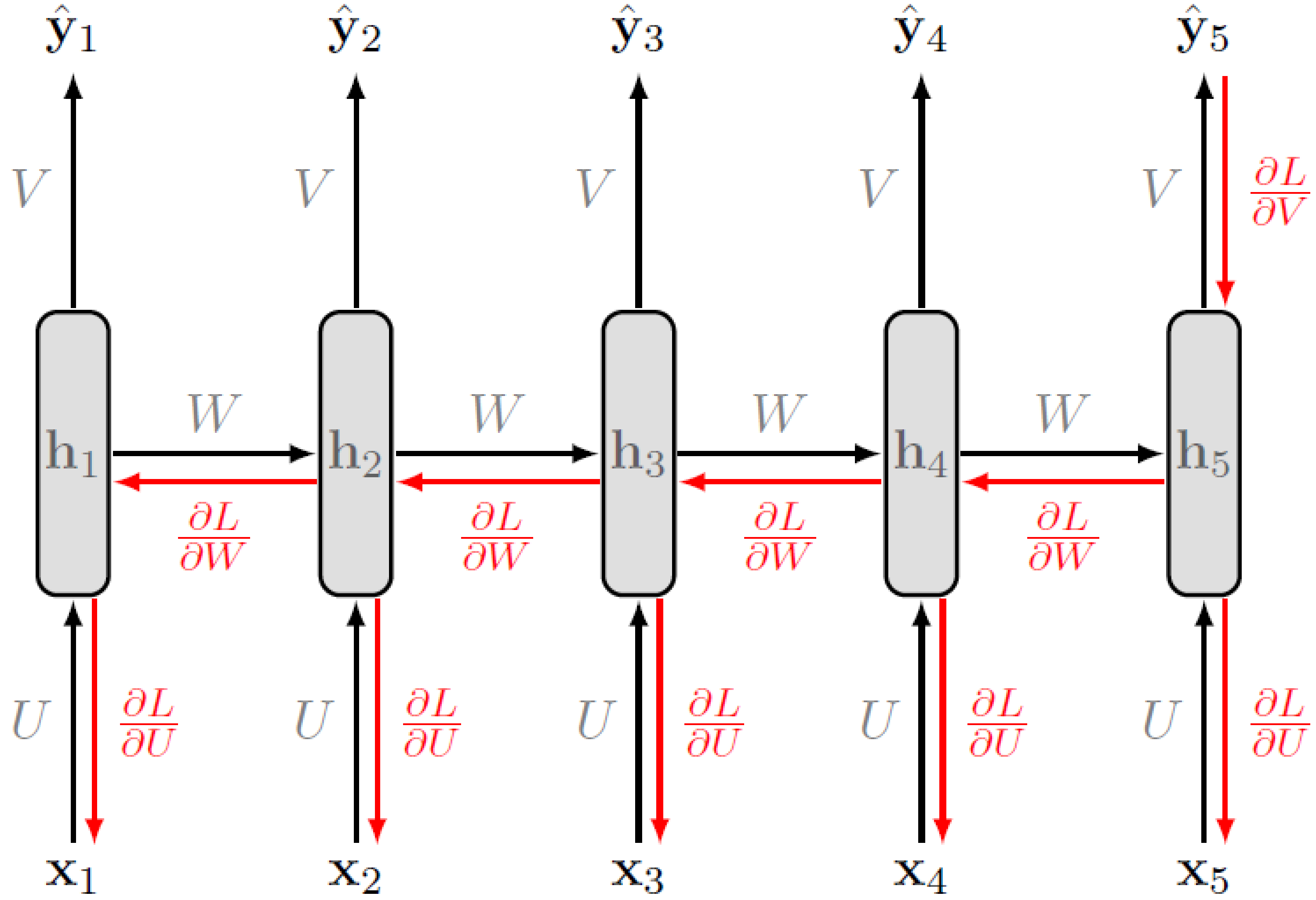
Where, for $t = \tau$ (one descendant):

$$(\nabla_{\mathbf{h}_\tau} L) = V^T(\nabla_{\mathbf{o}_\tau} L)$$

For some $t < \tau$ (two descendants)

$$\begin{aligned}(\nabla_{\mathbf{h}_t}L) &= \left(\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right)^T(\nabla_{\mathbf{h}_{t+1}}L) + \left(\frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right)^T(\nabla_{\mathbf{o}_t}L)\\ &= W^T(\nabla_{h_{t+1}}L)diag(1-\mathbf{h_{t+1}}^2) + V(\nabla_{\mathbf{o}_t}L)\end{aligned}$$

BPTT


$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\hat{\mathbf{y}}_5$
V
V
V
V
V
$\frac{\partial L}{\partial V}$
W
W
W
W
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_5$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
$\frac{\partial L}{\partial W}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
U
$\frac{\partial L}{\partial U}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

Gradient Computation

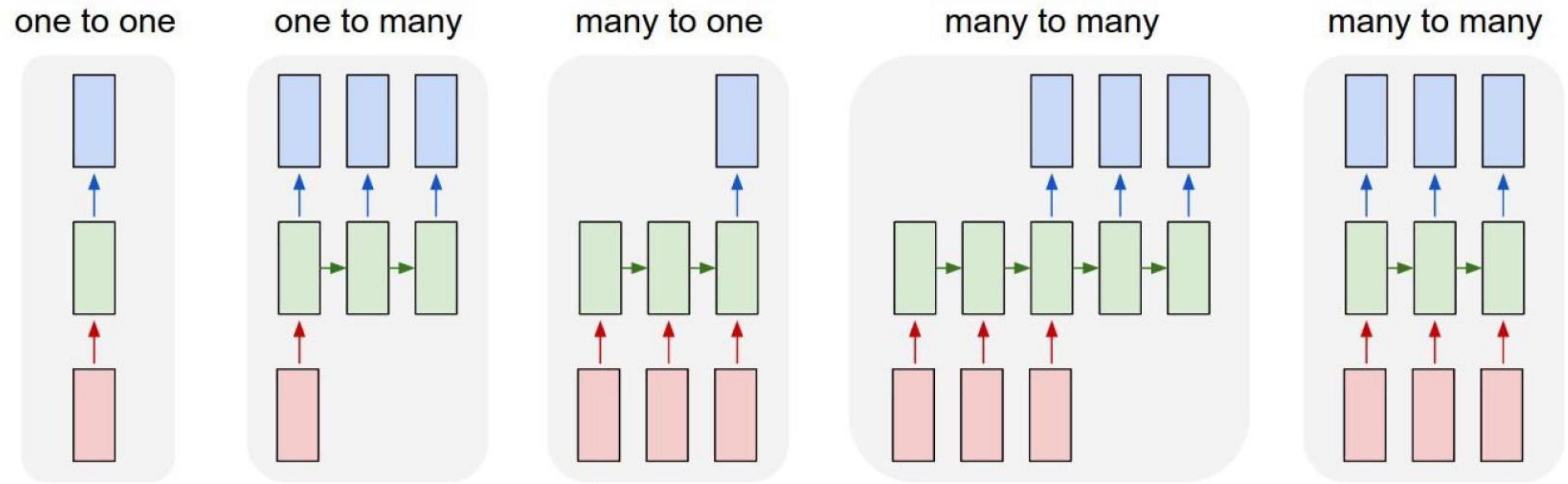
Refer to:

$$
\begin{aligned}
\mathbf{a}_t &= b + W\mathbf{h}_{t-1} + U\mathbf{x}_t \\
\mathbf{h}_t &= \tanh \mathbf{a}_t \\
\mathbf{o}_t &= c + V\mathbf{h}_t \\
\hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{o}_t)
\end{aligned}
$$

$$\tanh'(s) = 1 - \tanh^2(s)$$

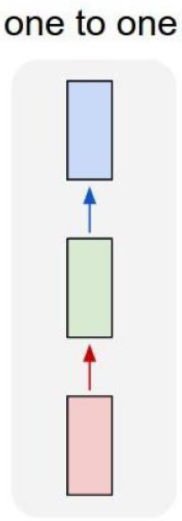
$$\nabla_U L = \sum_t \overbrace{diag\left(1 - (\mathbf{h}_t)^2\right)}^{\frac{\partial h_t}{\partial a_t}} \overbrace{(\nabla_{\mathbf{h}_t} L)}^{\frac{\partial L}{\partial h_t}} \overbrace{\mathbf{x}_t^T}^{\frac{\partial a_t}{\partial U}}$$

RNN Paradigms



Different problems are more suited for different RNN paradigms

RNN Paradigms



Given a single input predict a single output

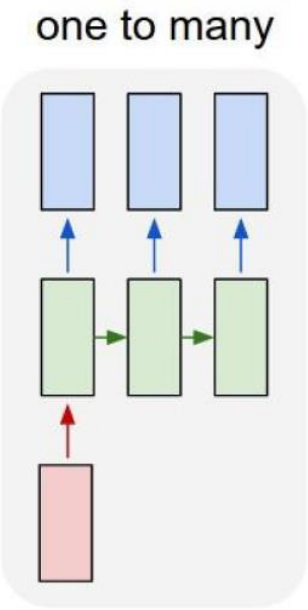
This is just a simple feedforward neural network

Ex. Image Classification

Given an image, judge the image category

Different problems are more suited for different RNN paradigms

RNN Paradigms



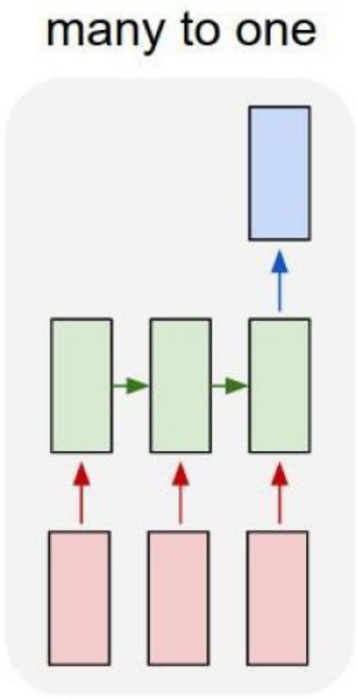
Given a single input predict a sequence of outputs

Ex. Image Captioning

Given an image and describe the image textually

Different problems are more suited for different RNN paradigms

RNN Paradigms



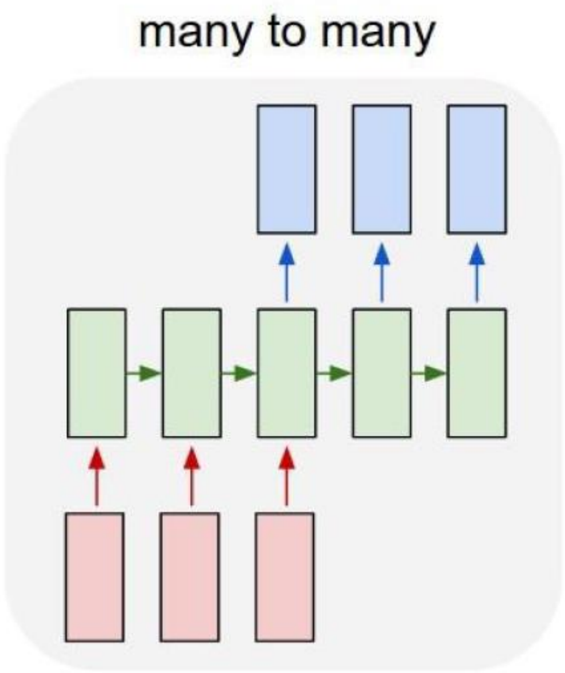
Given a sequence of inputs predict a single output

Ex. Sentiment Analysis

Given text predict positive or negative sentiment

Different problems are more suited for different RNN paradigms

RNN Paradigms



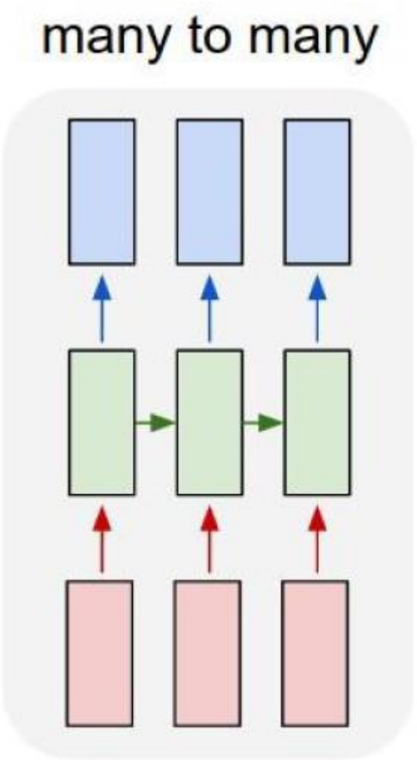
Given a sequence of inputs predict a sequence of outputs (of potentially different length)

Ex. Machine Translation

Given text in language A, translate it to language B

Different problems are more suited for different RNN paradigms

RNN Paradigms



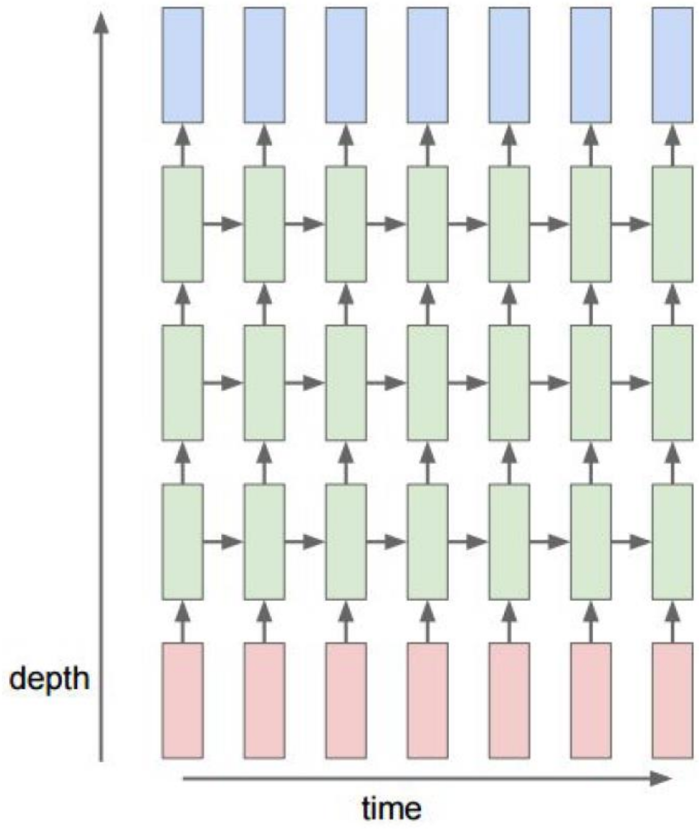
Given a sequence of inputs predict a sequence of outputs (of the same length)

Ex. Part of Speech Tagging

Given a sequence of words, label each word with its part of speech (Noun, Verb, etc.)

Different problems are more suited for different RNN paradigms

How Layering RNNs Work

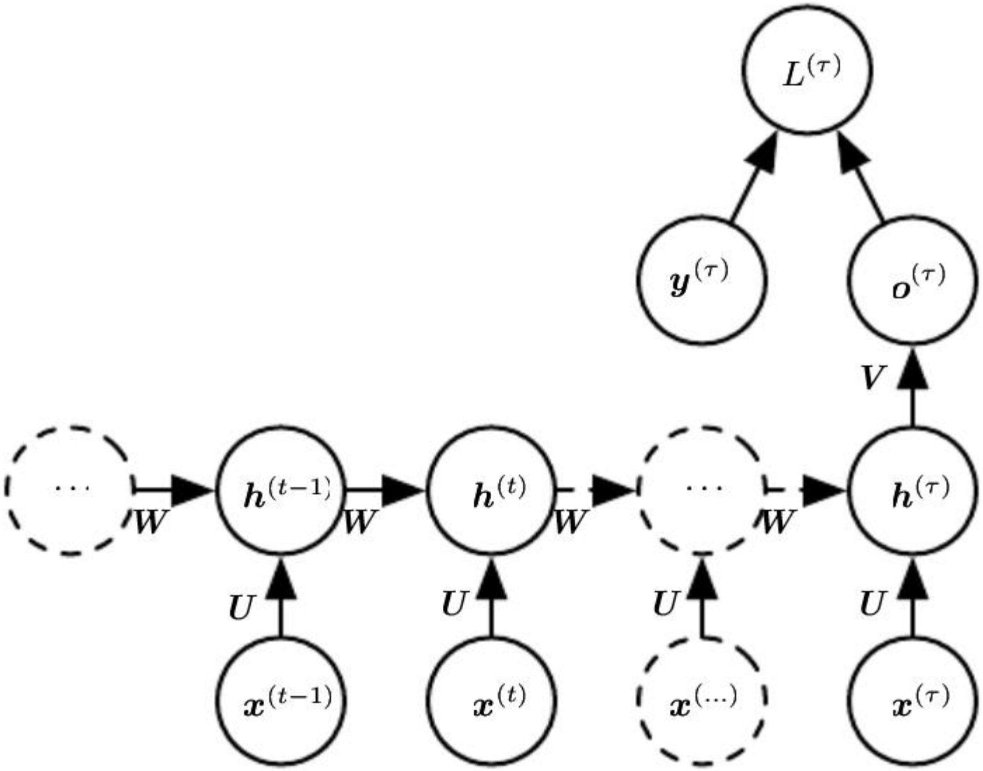


You just stack them like this… Nothing special

You can change the order or the direction

You can change the granularity as well (Hierarchical Networks)

Design Patterns of Recurrent Networks



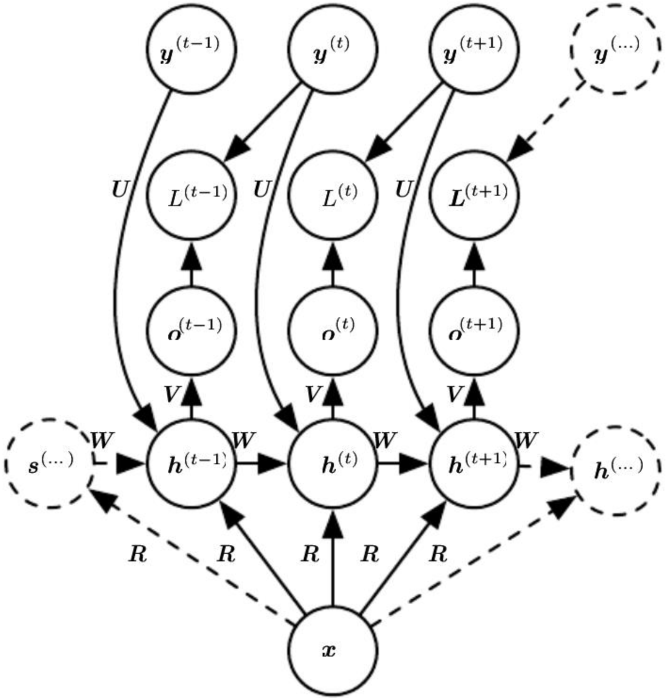
- Summarization: Produce a single output and have recurrent connections from output between hidden units
- Useful for summarizing a sequence (e.g. sentiment analysis)

Design Patterns: Fixed vector as input

- We have considered RNNs in the context of a sequence of vectors $x^{(t)}$ with $t = 1, . . . , \tau$ as input
- Sometimes we are interested in only taking a single, fixed sized vector x as input, that generates the sequence y
- Some common ways to provide an extra input to a RNN are:
  - As an extra input at each time step
  - As the initial state $h^{(0)}$
  - both

Design Patterns: Fixed vector as input

- The first option (extra input at each time step) is the most common:

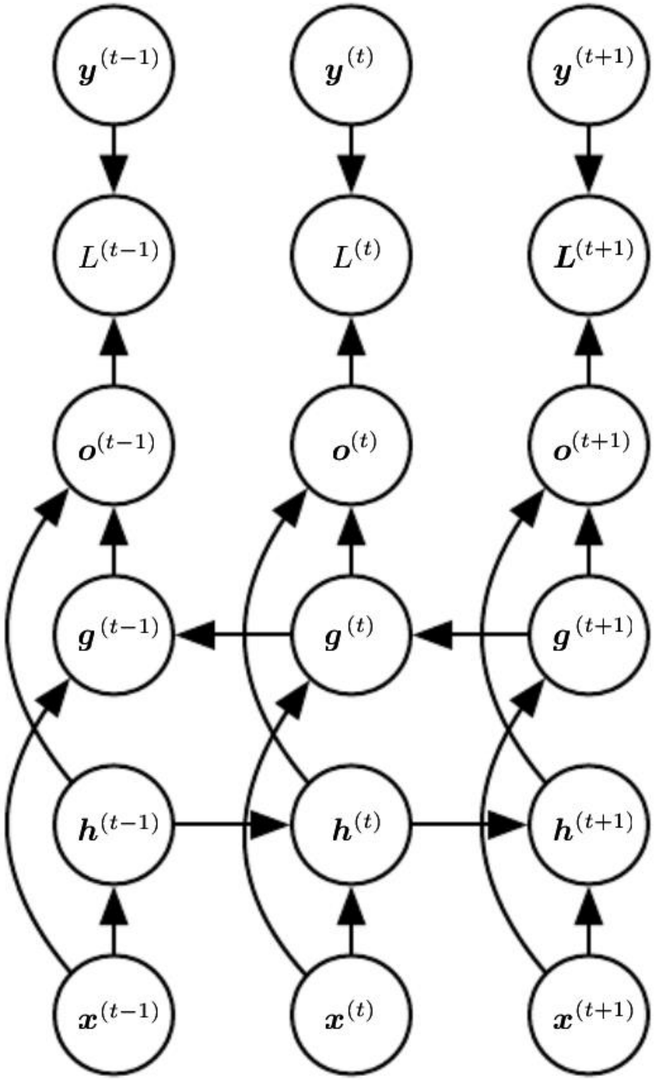


- Maps a fixed vector x into a distribution over sequences Y (x^TR effectively is a new bias parameter for each hidden unit)

Design Patterns: Bidirectional RNNs

- RNNs considered till now, all have a causal structure: State at time t only captures information from the past $x^{(1)}, . . . , x^{(t-1)}$
- Sometimes we are interested in an output $y^{(t)}$ which may depend on the whole input sequence
- Example: Interpretation of a current sound as a phoneme may depend on the next few due to co-articulation
- Basically, in many cases we are interested in looking into the future as well as the past to disambiguate interpretations
- Bidirectional RNNs were introduced to address this need (*Schuster and Paliwal, 1997*), and have been used in handwriting recognition (*Graves 2012, Graves and Schmidhuber 2009*), speech recognition (*Graves and Schmidhuber 2005*) and bioinformatics (*Baldi 1999*)

Design Patterns: Bidirectional RNNs

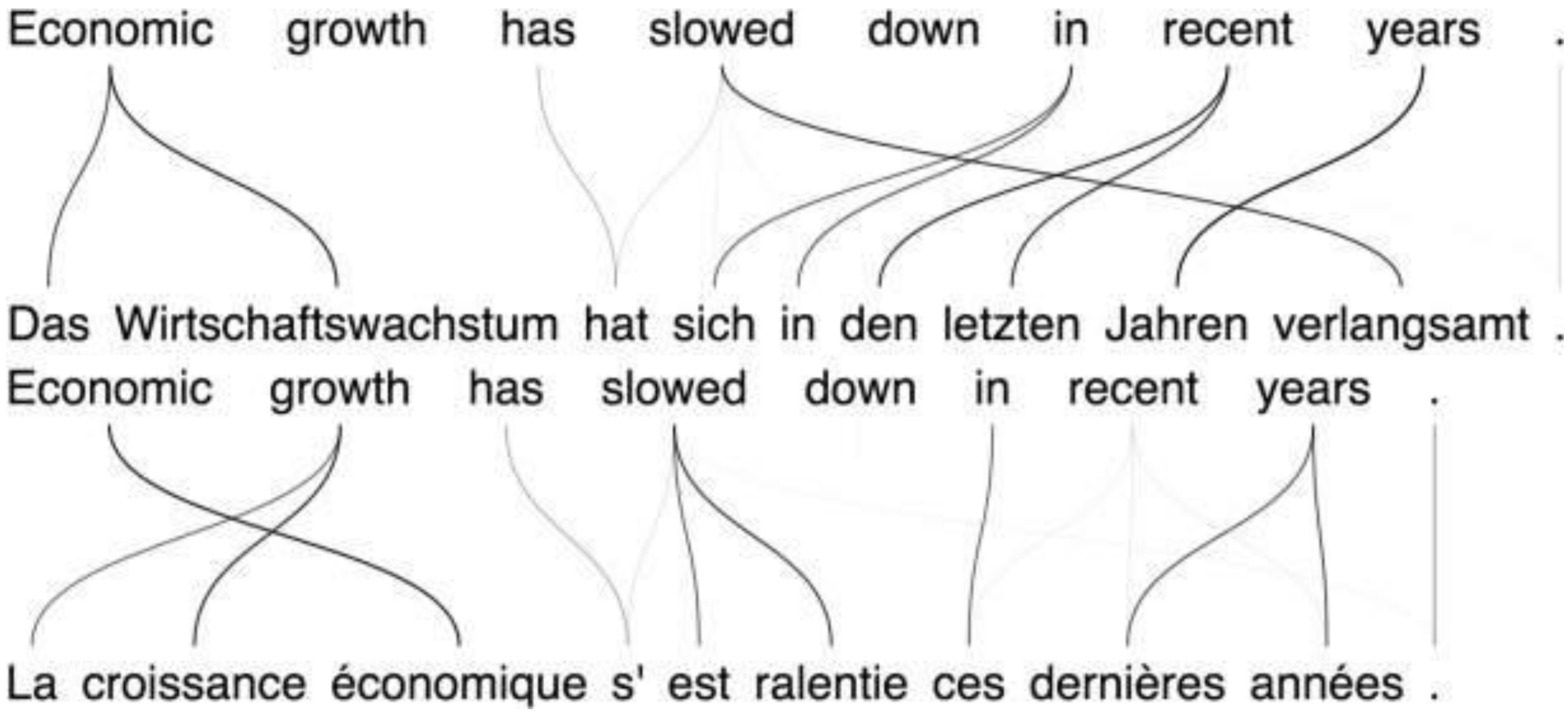


Design Patterns: Encoder-Decoder

- How do we map input sequences to output sequences that are not necessarily of the same length?
- Example:
  - Input – ‘*Kerem jojjenek maskor es kulonosen mashoz’*.
  - Output - *'Please come rather at another time and to another person.'*
- Other example applications: Speech Recognition, Question Answering, etc.
- The input to this RNN is called the context, we want to find a representation of the context C
- C could be a vector or a sequence that summarizes $X = \{x^{(1)}, \ldots, x^{(n_x)}\}$

Design Patterns: Encoder-Decoder

English-to-French translation



Economic growth has slowed down in recent years .

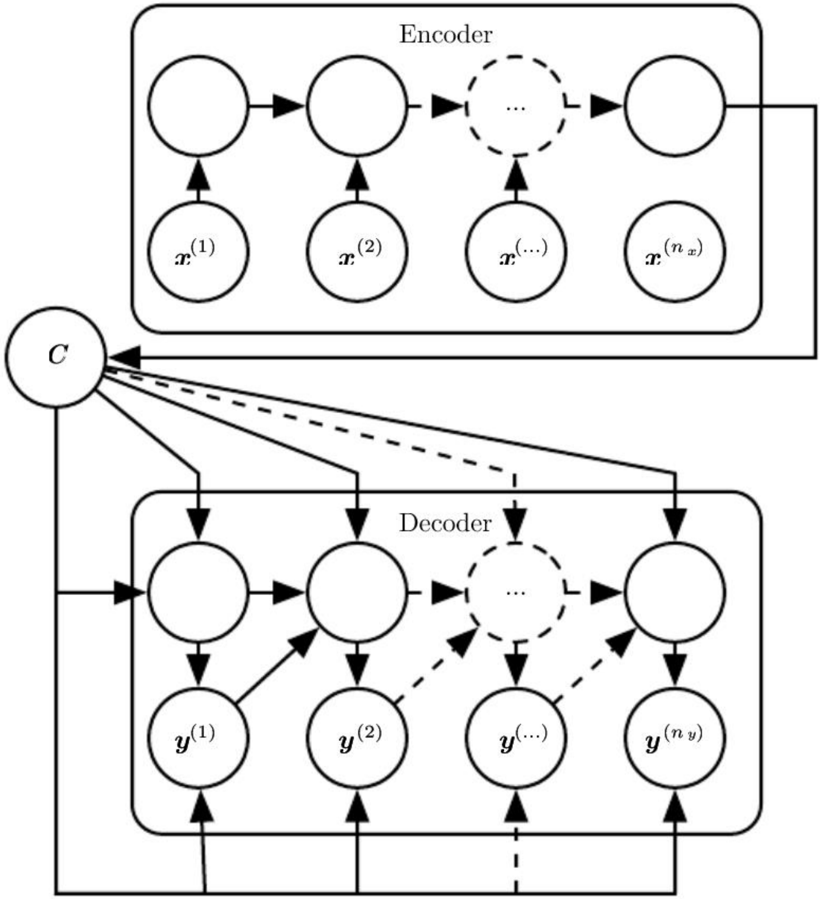
Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt .

Economic growth has slowed down in recent years .

La croissance économique s' est ralentie ces dernières années .

- Far more complicated mappings

Design Patterns: Encoder-Decoder



- In the context of Machine Translation *C* is called a thought vector

Deep Recurrent Networks

- The computations in RNNs can be decomposed into three blocks of parameters/associated transformations:
  - Input to hidden state
  - Previous hidden state to the next
  - Hidden state to the output
- Each of these transforms till now were learned affine transformations followed by a fixed nonlinearity
- Introducing depth in each of these operations is advantageous (Graves et al. 2013, Pascanu et al. 2014)
- The intuition on why depth should be more useful is quite similar to that in deep feed-forward networks
- Optimization can be made much harder, but can be mitigated by tricks such as introducing skip connections

Outline

1⁄ Course Review

2⁄ Basic Recurrent Neural Network (RNN)

3⁄ Variants of RNN

4⁄ Applications

Challenge of Long-Term Dependencies

- Basic problem: Gradients propagated over many stages tend to vanish (most of the time) or explode (relatively rarely)
- Difficulty with long-term interactions (involving multiplication of many Jacobians) arises due to exponentially smaller weights, compared to short-term interactions
- The problem was first analyzed by Hochreiter and Schmidhuber in 1991 and Bengio et al in 1993

Challenge of Long-Term Dependencies

- Recurrent Networks involve the composition of the same function multiple times, once per time step
- The function composition in RNNs somewhat resembles matrix multiplication
- Consider the recurrence relationship:

$$h^{(t)} = W^T h^{(t-1)}$$

- This could be thought of as a very simple recurrent neural network without a nonlinear activation and lacking *x*
- This recurrence essentially describes the power method and can be written as:

$$h^{(t)} = (W^t)^T h^{(0)}$$

Challenge of Long-Term Dependencies

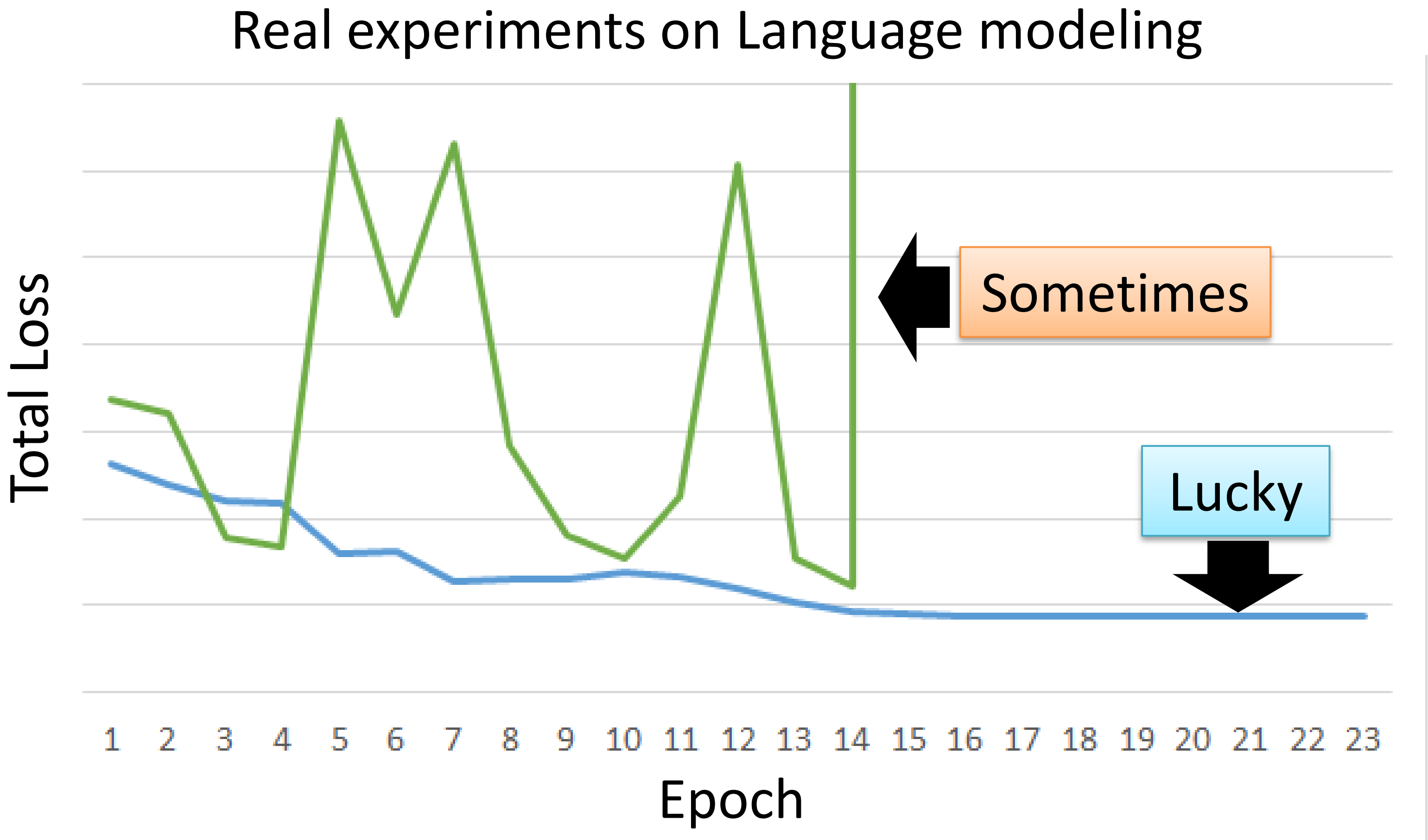
- If W admits a decomposition W = QΛQ^T with orthogonal *Q*
- The recurrence becomes:

$$h^{(t)} = (W^t)^T h^{(0)} = Q^T \Lambda^t Q h^{(0)}$$

- Eigenvalues are raised to *t*: Quickly decay to zero or explode
- Problem particular to RNNs

Challenge of Long-Term Dependencies

- RNN-based network is not always easy to learn

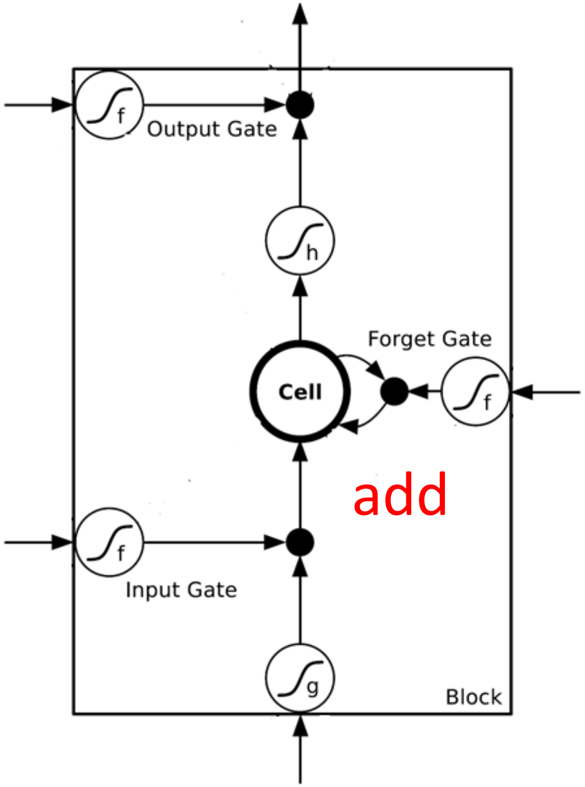


Challenge of Long-Term Dependencies

- Long Short-term Memory (LSTM)
  - Can deal with gradient vanishing (not gradient explode)

➢ Memory and input are ***added***

➢ The influence never disappears unless forget gate is closed



And:

Gated Recurrent Unit (GRU): simpler than LSTM

[Cho, EMNLP'14]

Helpful Techniques!

Long Short-Term Memory (LSTM)

- The LSTM uses this idea of “Constant Error Flow” for RNNs to create a “Constant Error Carousel” (CEC) which ensures that gradients do not decay
- The key component is a memory cell that acts like an accumulator (contains the identity relationship) over time
- Instead of computing new state as a matrix product with the old state, it rather computes the difference between them. Expressivity is the same, but gradients are better behaved

——Long Short-Term Memory, Hochreiter et al., 1997

Details and another way to explain：https://www.cnblogs.com/bonelee/p/10475453.html

Long Short Term Memory

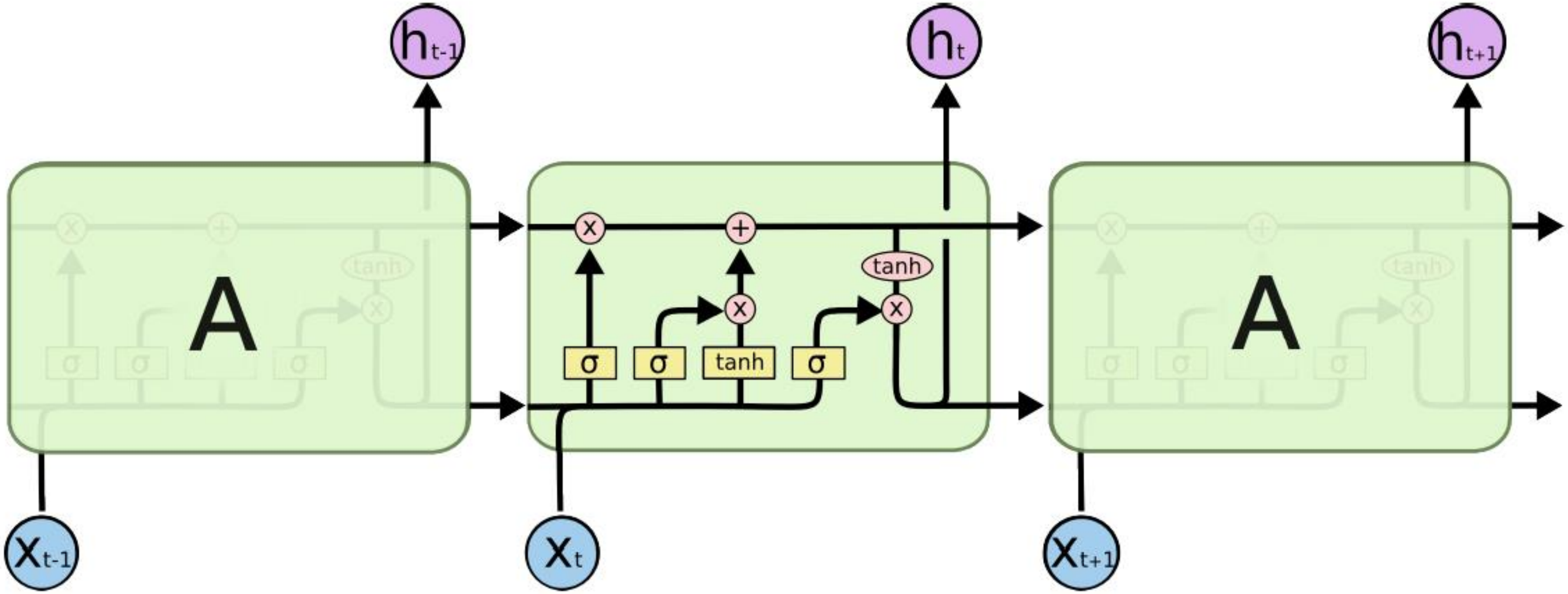
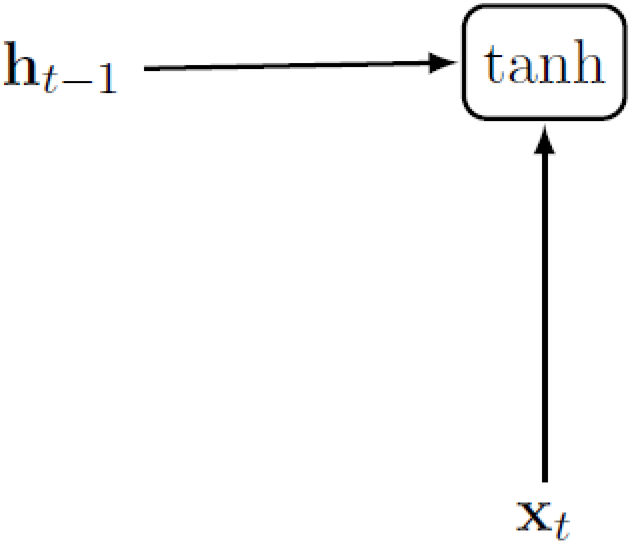


Figure: Chris Olah

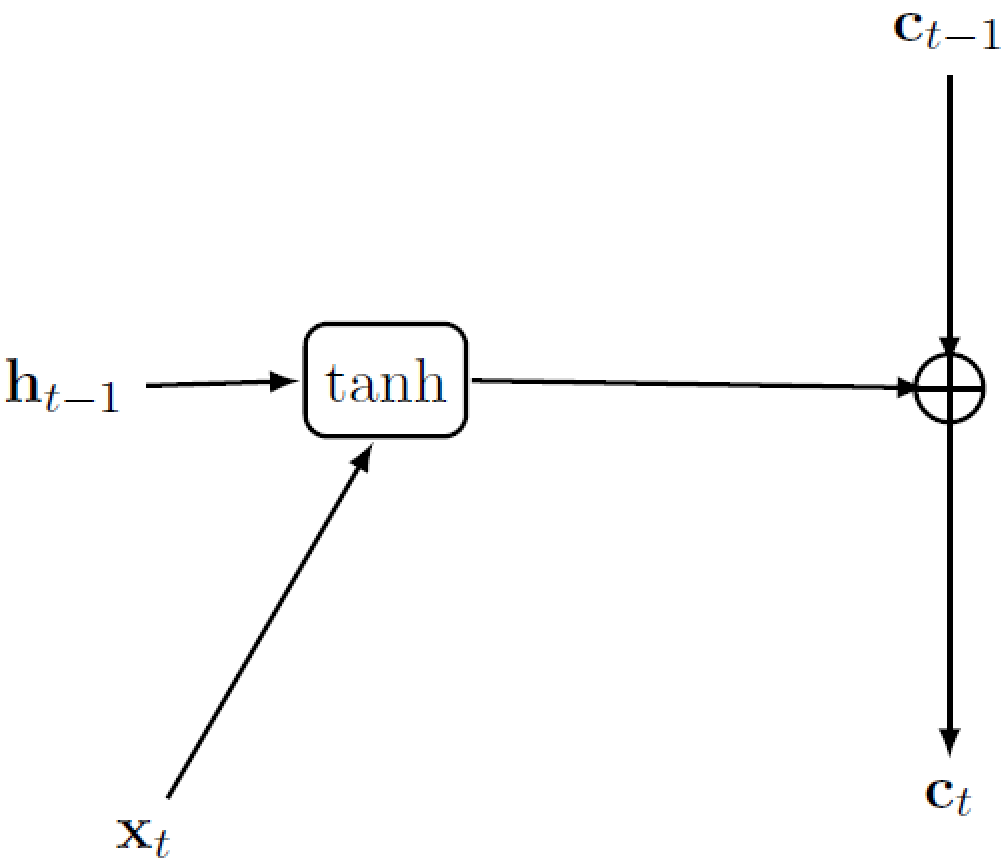
- Proposed by Hochreiter and Schmidhuber (1997)
- Now let's try to understand each memory cell!

Long Short Term Memory



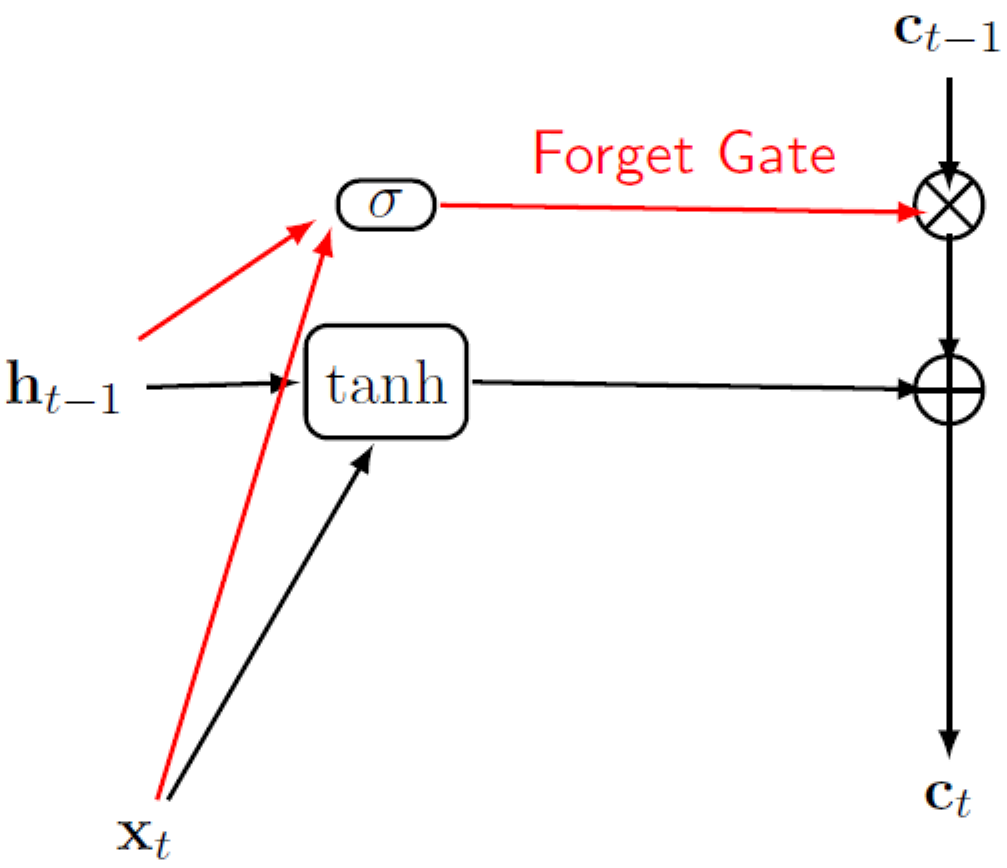
$$\mathbf{h}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

Long Short Term Memory



$$\tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$
$$\mathbf{c}_t = \mathbf{c}_{t-1} + \tilde{\mathbf{c}}_t$$

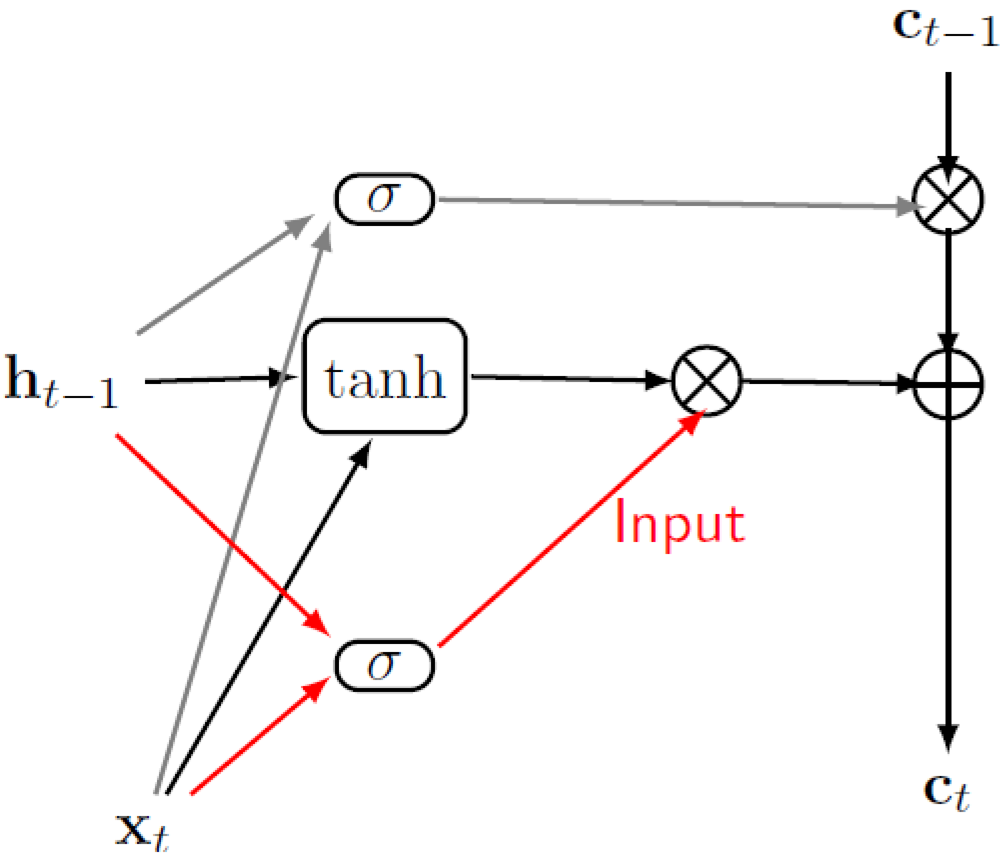
Long Short Term Memory



$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + \tilde{\mathbf{c}}_t$$

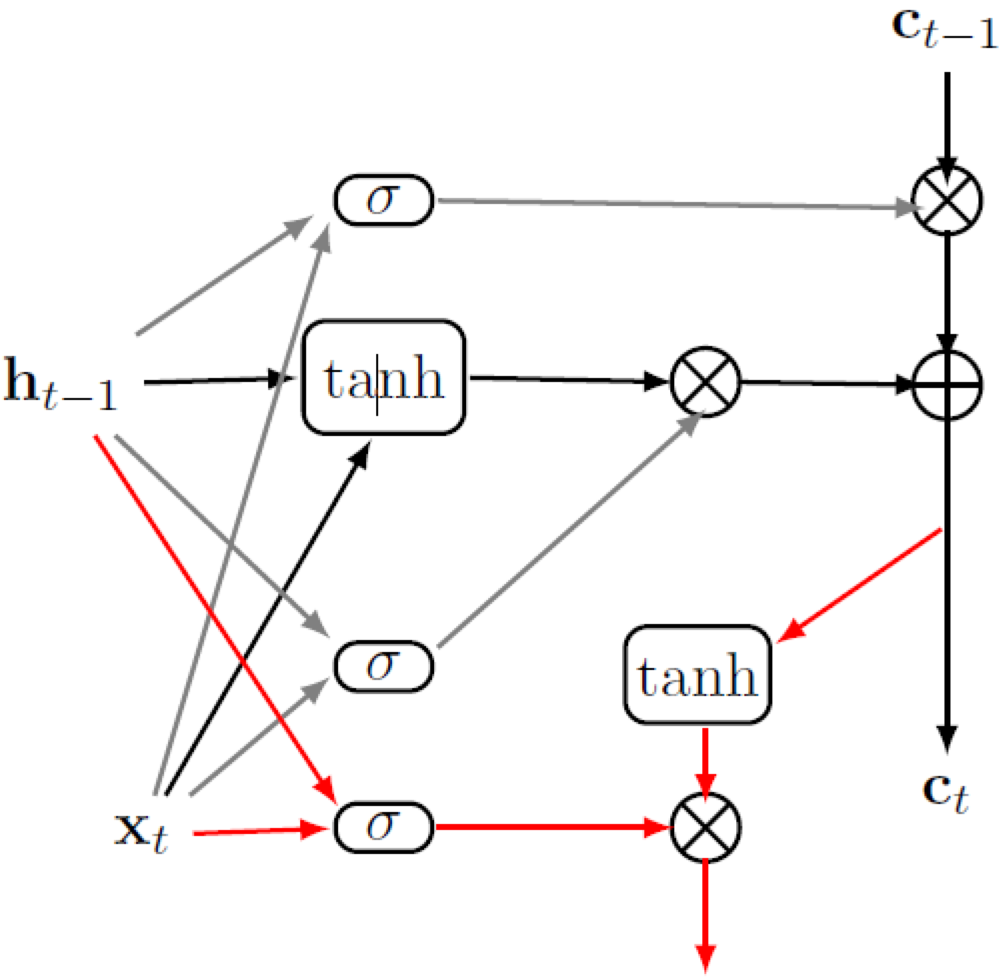
Long Short Term Memory



$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$
$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W \mathbf{h}_{t-1} + U \mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t$$

Long Short Term Memory



$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$
$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$
$$o_t = \sigma(W_o \mathbf{h}_{t-1} + U_o \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W \mathbf{h}_{t-1} + U \mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LSTM: Further Intuition

- The Cell State

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t \text{ with } \tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

- Useful to think of the cell as a *conveyor belt* (Olah), which runs across time
- Only interrupted with linear interactions
- The memory cell can add or delete information from the cell state by gates
- Gates are constructed by using a sigmoid and an element-wise multiplication

LSTM: Further Intuition

- The Forget Gate

$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$

- Help to decide what information to throw away from the cell state
- Once we have thrown away what we want from the cell state, we want to update it

LSTM: Further Intuition

- First we decide how much of the input we want to store in the updated cell state via the Input Gate

$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$

- We then update the cell state:

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t$$

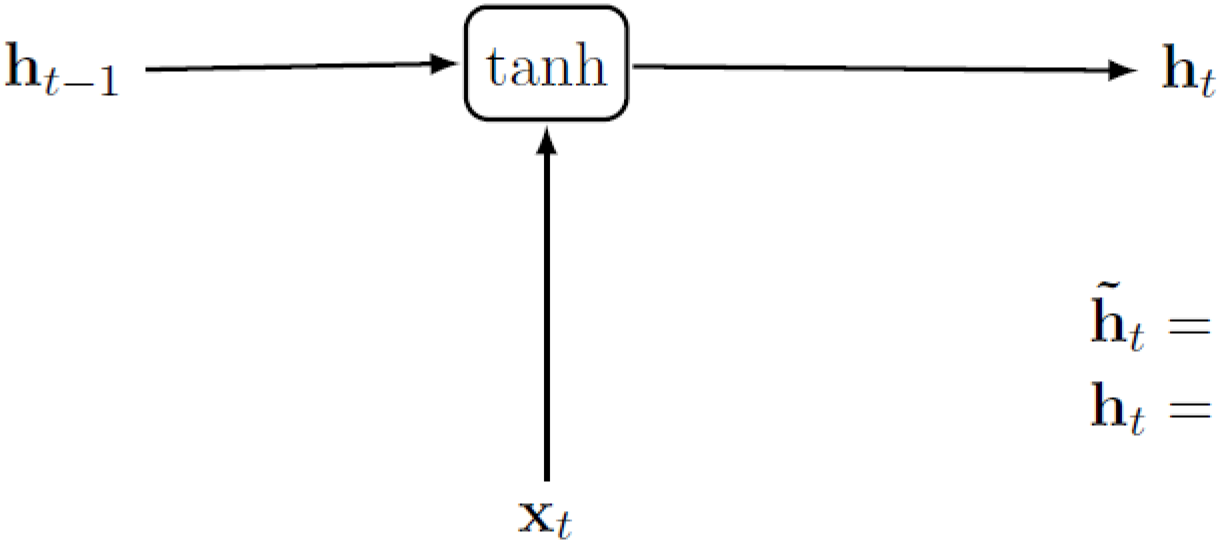
- We then need to output, and use the output gate $o_t = \sigma(W_o \mathbf{h}_{t-1} + U_o \mathbf{x}_t)$ to pass on the filtered version

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Gated Recurrent Unit

- Let $\tilde{\mathbf{h}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$ and $\mathbf{h}_t = \tilde{\mathbf{h}}_t$
- Reset gate: $r_t = \sigma(W_r\mathbf{h}_{t-1} + U_r\mathbf{x}_t)$
- New $\tilde{\mathbf{h}}_t = \tanh(W(r_t \odot \mathbf{h}_{t-1}) + U\mathbf{x}_t)$
- Find: $z_t = \sigma(W_z\mathbf{h}_{t-1} + U_z\mathbf{x}_t)$
- Update $\mathbf{h}_t = z_t \odot \tilde{\mathbf{h}}_t$
- Finally: $\mathbf{h}_t = (1 - z_t) \odot \mathbf{h}_{t-1} + z_t \odot \tilde{\mathbf{h}}_t$
- Comes from attempting to factor LSTM and reduce gates
- Example: One gate controls forgetting as well as decides if the state needs to be updated

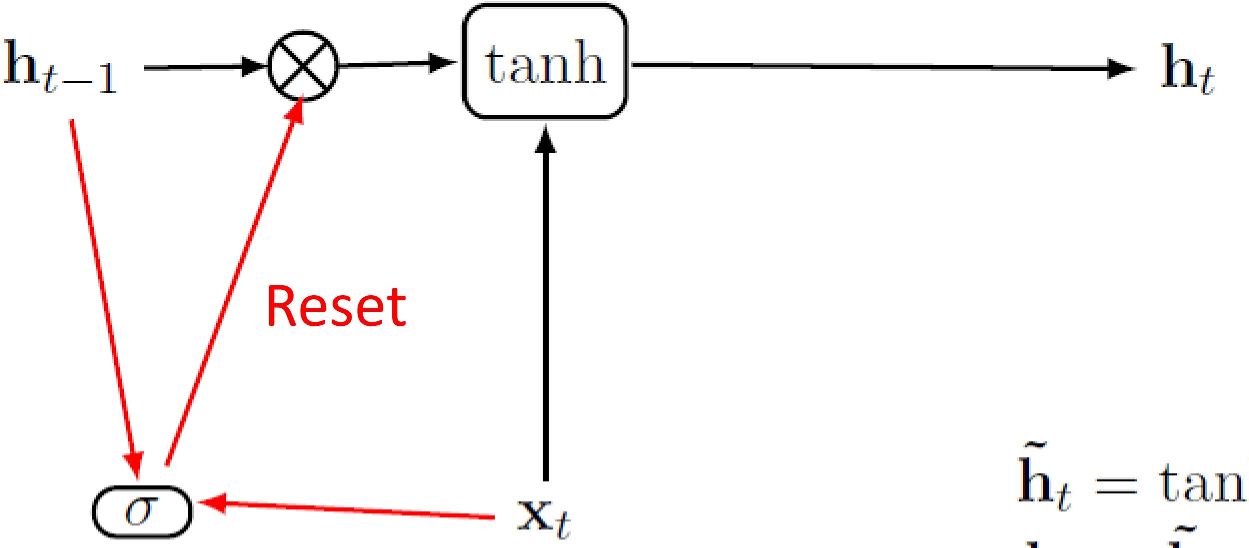
Gated Recurrent Unit



$$\tilde{\mathbf{h}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$
$$\mathbf{h}_t = \tilde{\mathbf{h}}_t$$

Gated Recurrent Unit

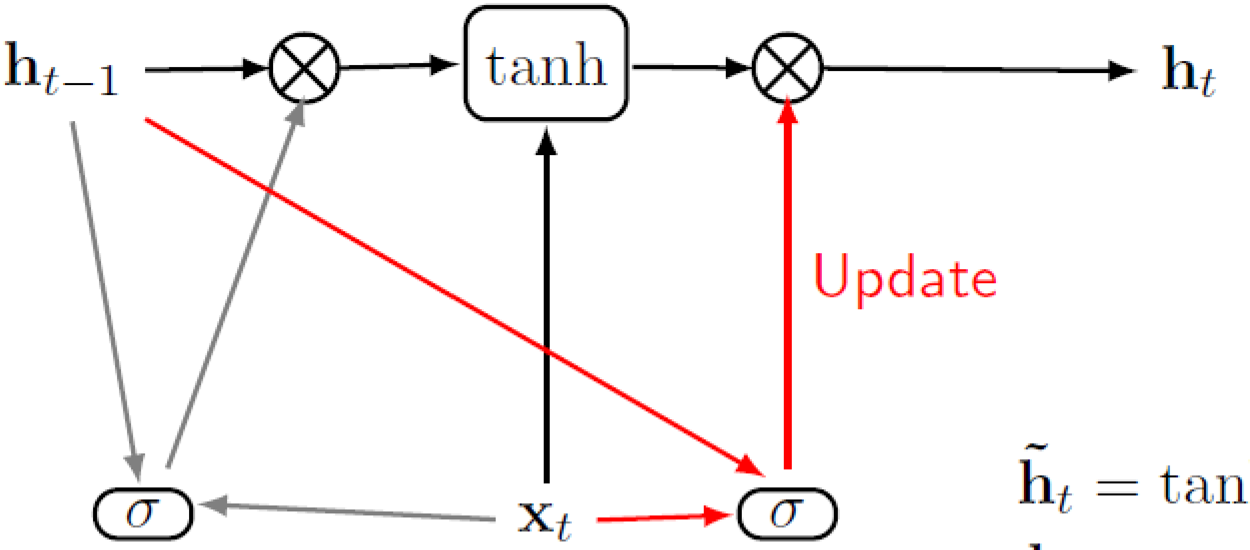
$$r_t = \sigma(W_r \mathbf{h}_{t-1} + U_r \mathbf{x}_t)$$



$$\tilde{\mathbf{h}}_t = \tanh(W(r_t \odot \mathbf{h}_{t-1}) + U\mathbf{x}_t)$$
$$\mathbf{h}_t = \tilde{\mathbf{h}}_t$$

Gated Recurrent Unit

$$r_t = \sigma(W_r \mathbf{h}_{t-1} + U_r \mathbf{x}_t)$$
$$z_t = \sigma(W_z \mathbf{h}_{t-1} + U_z \mathbf{x}_t)$$

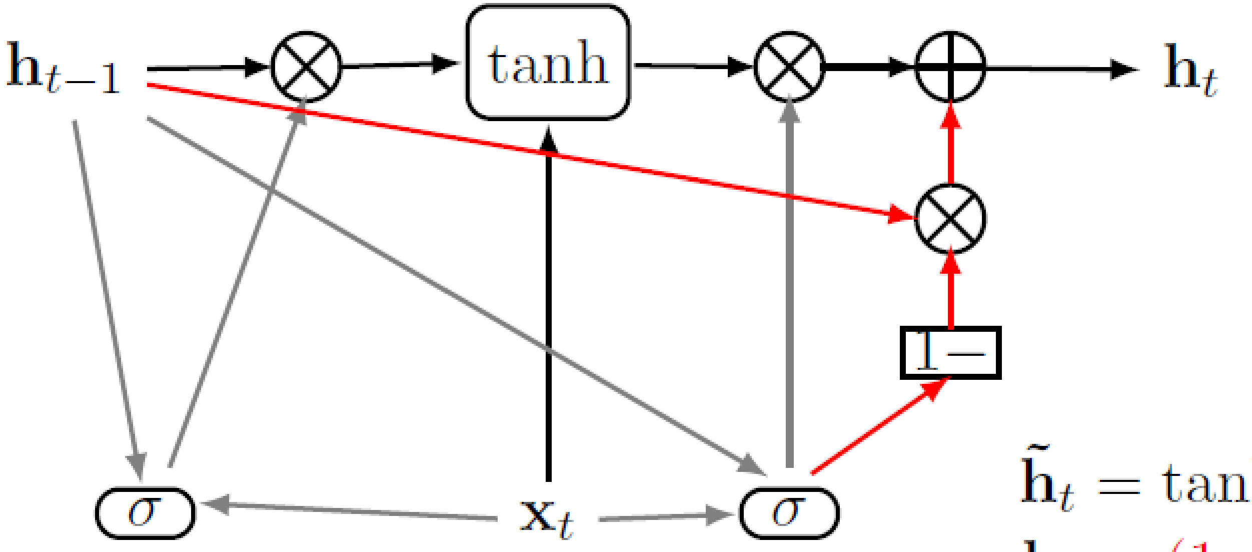


$$\tilde{\mathbf{h}}_t = \tanh(W(r_t \odot \mathbf{h}_{t-1}) + U\mathbf{x}_t)$$
$$\mathbf{h}_t = z_t \odot \tilde{\mathbf{h}}_t$$

Gated Recurrent Unit

$$r_t = \sigma(W_r \mathbf{h}_{t-1} + U_r \mathbf{x}_t)$$

$$z_t = \sigma(W_z \mathbf{h}_{t-1} + U_z \mathbf{x}_t)$$



$$\tilde{\mathbf{h}}_t = \tanh(W(r_t \odot \mathbf{h}_{t-1}) + U\mathbf{x}_t)$$

$$\mathbf{h}_t = (1 - z_t) \odot \mathbf{h}_{t-1} + z_t \odot \tilde{\mathbf{h}}_t$$

Common Pitfalls of RNNs

- These models can overfit incredibly easily
  - Start with an incredibly simple model, with small gates and few layers, then expand
- Vanishing/Exploding Gradients
  - Depending on your data, BPTT can cause your gradients to become incredibly small (vanish) or become incredibly large (explode)
    - Gated cells can mitigate this to an extent, but not entirely
    - Be sure to regularize and keep an eye on your gradients to see how they are doing

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but do not work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed

Outline

1/ Course Review

2/ Basic Recurrent Neural Network (RNN)

3/ Variants of RNN

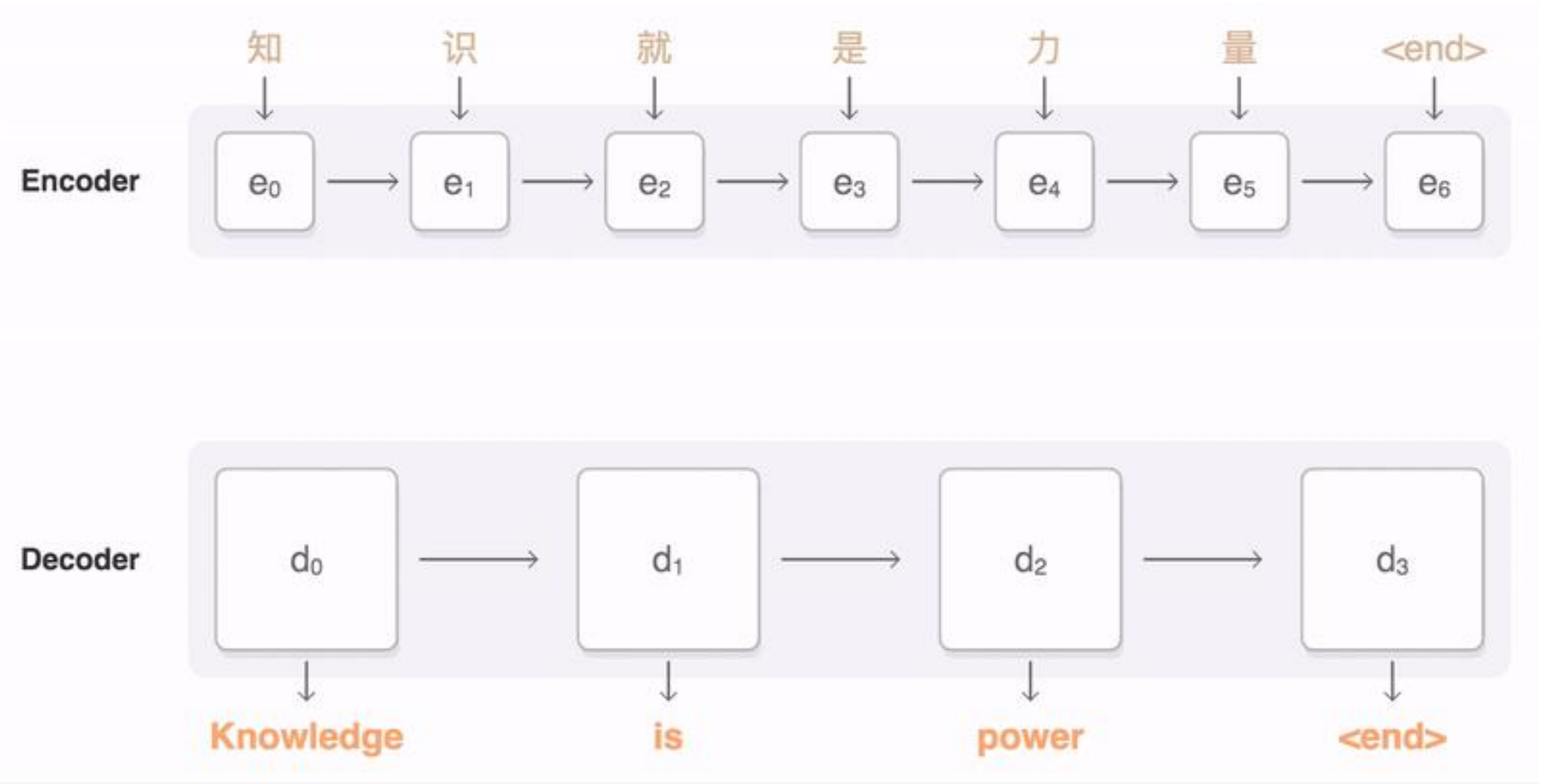
4/ Applications

Applications

We will look at two classic papers on the topic:

- **Machine Translation**: Neural Machine Translation by Jointly Learning to Align and Translate by Bahdanau *et al*.
- **Image Caption Generation**: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015 by Xu *et al*.

Neural Machine Translation



A general-purpose encoder-decoder framework for Machine Translation

Neural Machine Translation

- Recall our encoder-decoder model for machine translation

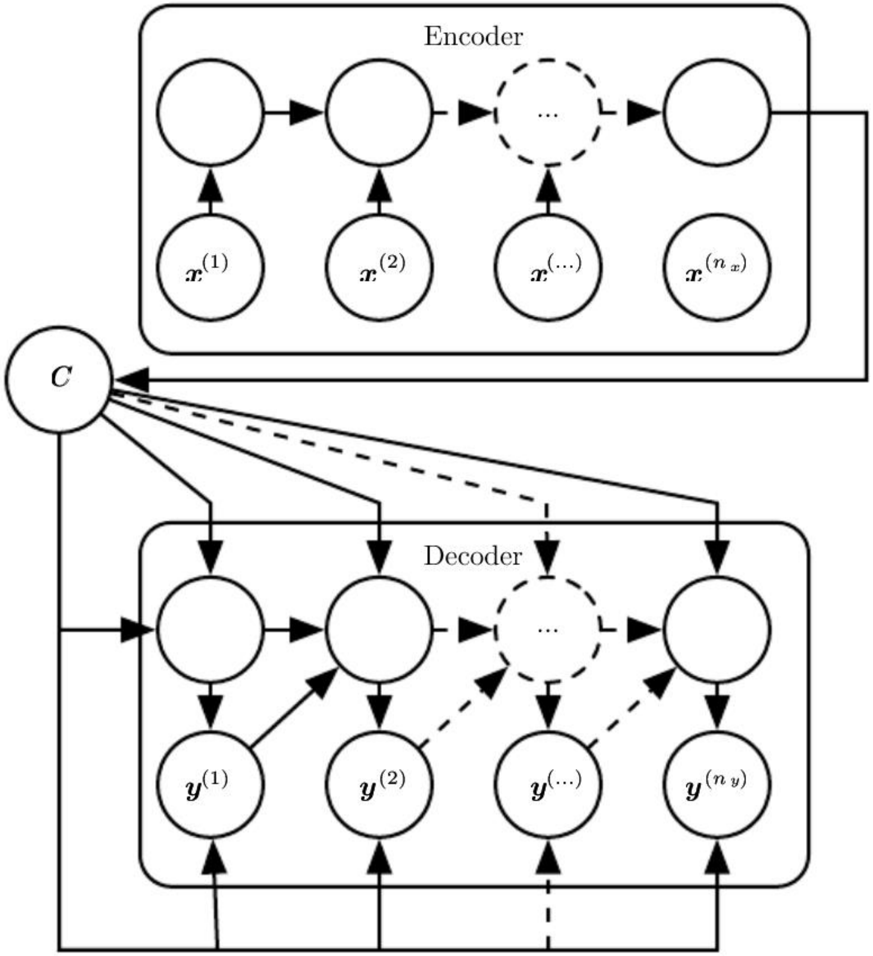


Figure: Goodfellow *et al*.

Neural Machine Translation

Let's look at the steps for translation again:

- The input sentence $x_1 , . . . , x_n$ via hidden unit activations $h_1 , . . . , h_n$ is encoded into the thought vector C
- Using C, the decoder then generates the output sentence $y_1, ... , y_p$
- We stop when we sample a terminating token i.e. <END>

A Problem? For long sentences, it might not be useful to only give the decoder access to the vector C

Neural Machine Translation

- When we ourselves are translating a sentence from one language to another, we do not consider the whole sentence at all times
- Intuition: Every word in the output only depends on a word or a group of words in the input sentence
- We can help the decoding process by allowing the decoder to refer to the input sentence
- We would like the decoder, while it is about to generate the next word, to attend to a group of words in the input sentence most relevant to predicting the right next word
- Maybe it would be more efficient to also be able to attend to these words while encoding

Neural Machine Translation

- First Observation: For each word we have a hidden unit, and this encodes a representation for each word
- Let's first try to incorporate both forward and backward contexts for each word using a bi-directional RNN and concatenate the result representations
- We have already seen why using a bidirectional RNN is useful

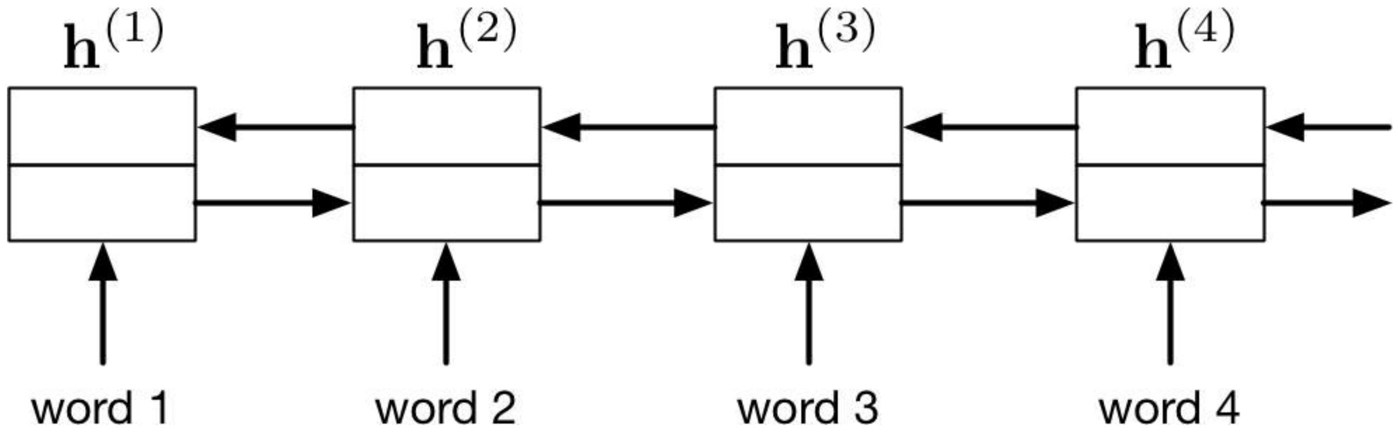


Figure: Roger Grosse

Neural Machine Translation

- The decoder generates the sentence one word at a time conditioned on C
- We can instead have a context vector for every time step
- These vectors $C^{(t)}$ learn to attend to specific words of the input sentence

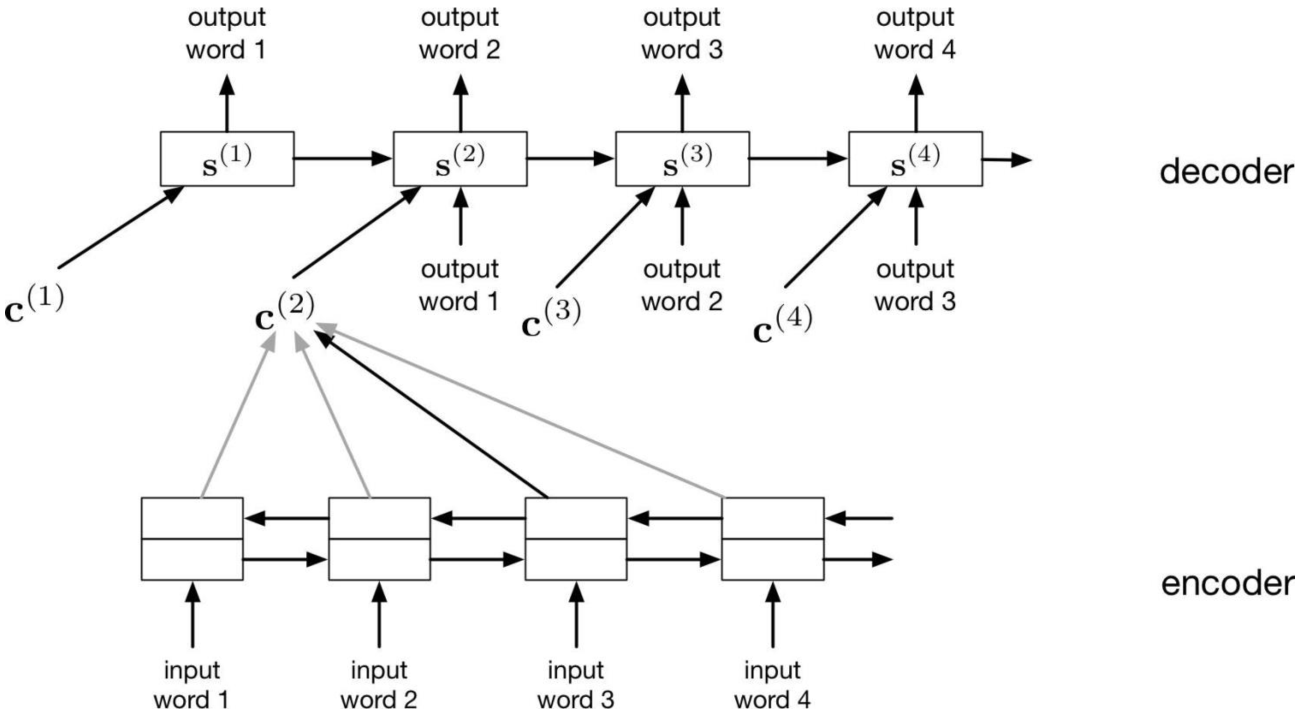


Figure: Roger Grosse

Neural Machine Translation

How do we learn $C^{(t)}$ so that they can attend to relevant words?

- First: Let the representations of the bidirectional RNN for each word be h_i
- Define $C^{(t)}$ to be the weighted average of encoder's representations:

$$C^{(t)} = \sum_i \alpha_{ti} \mathbf{h}_i$$

- α_t defines a probability distribution over the input words

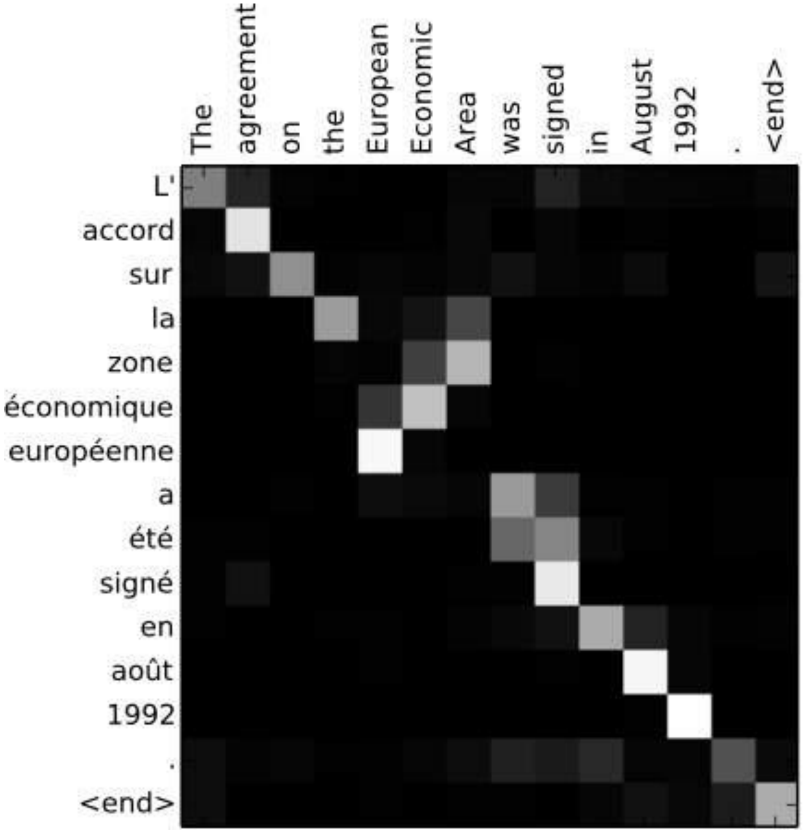
Neural Machine Translation

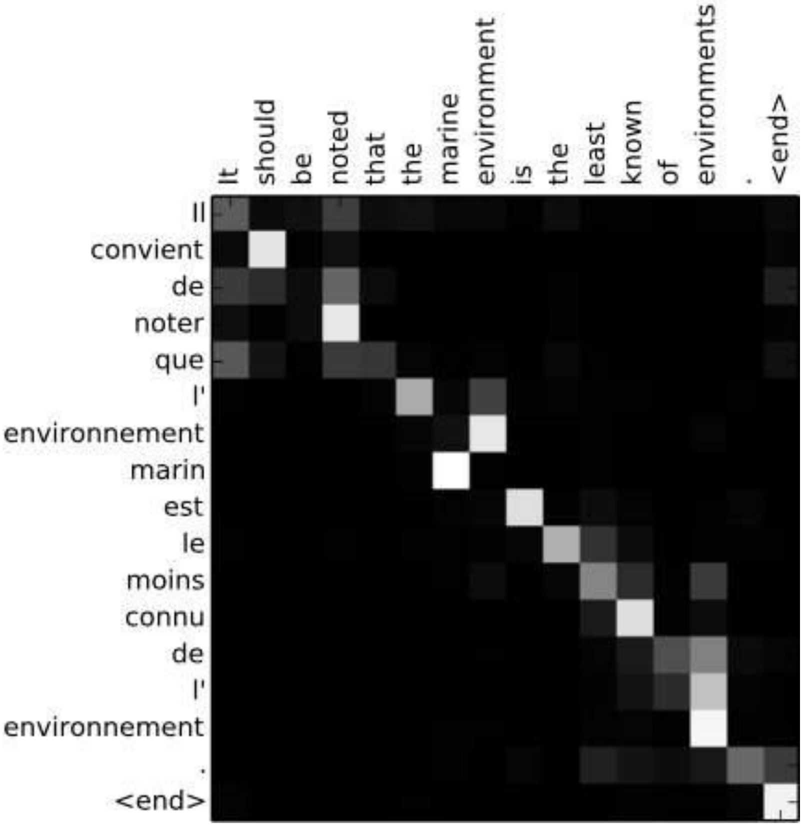
- α_{ti} is a function of the representations of the words and the previous states

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_k \exp(e_{tk})}$$

- With $e_{ti} = a(\mathbf{s}^{(t-1)}, \mathbf{h}^{(i)})$

Neural Machine Translation



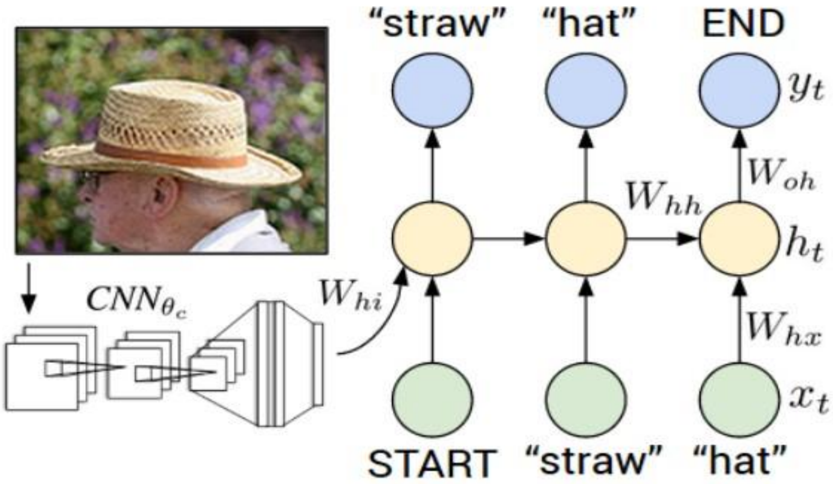


- For each word in the translation, the matrix gives the degree of focus on all the input words
- A linear order is not forced, but it figures out that the translation is approximately linear

Image Captioning

- We will look at one illustrative example: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015
- Humans do not process a visual scene all at once. The Fovea gives high resolution vision in only a tiny region of our field of view
- A series of glimpses are then integrated

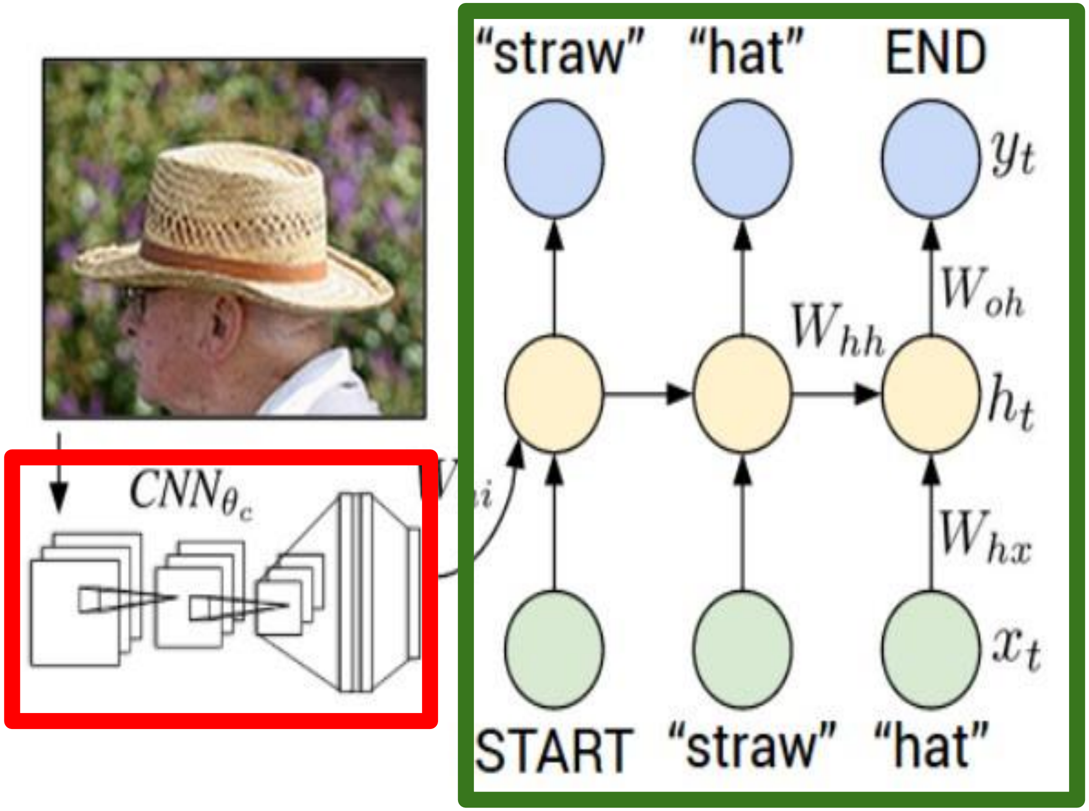
Image Captioning



- Explain images with Multimodal Recurrent Neural Networks, *Mao et al.*
- Deep Visual-Semantic Alignments for Generating Image Descriptions, *Karpathy and Fei-Fei*
- Show and Tell: A Neural Image Caption Generator, Vinyals et al.
- Long-term Recurrent Convolutional Networks for Visual Recognition and Description, *Donahue et al.*
- Learning a Recurrent Visual Representation for Image Caption Generation, *Chen and Zitnick*

Image Captioning

Recurrent Neural Network

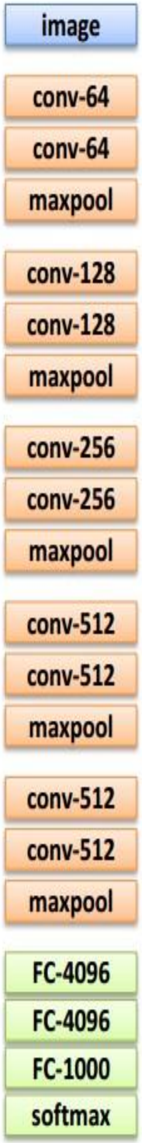


Convolutional Neural Network

Image Captioning

test image

Image Captioning



image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

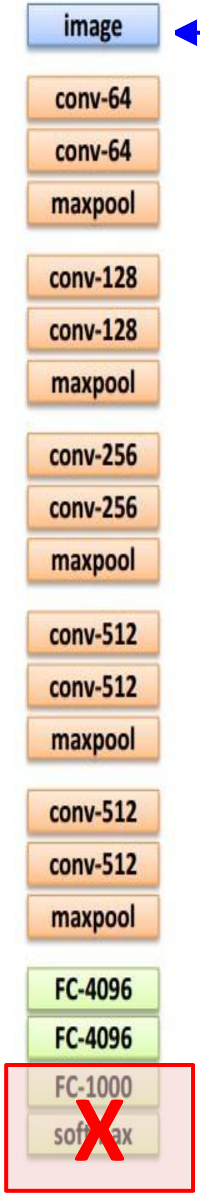
conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096
FC-1000
softmax

test image

Image Captioning



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

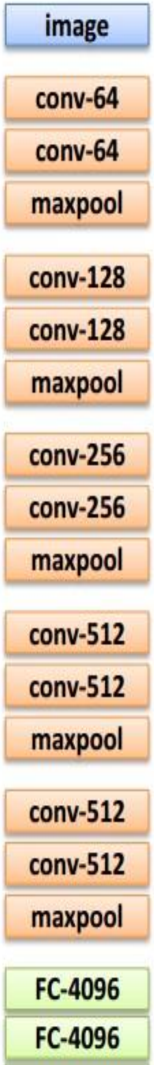
FC-1000

softmax

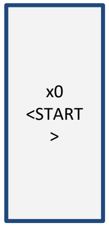
X

test image

Image Captioning

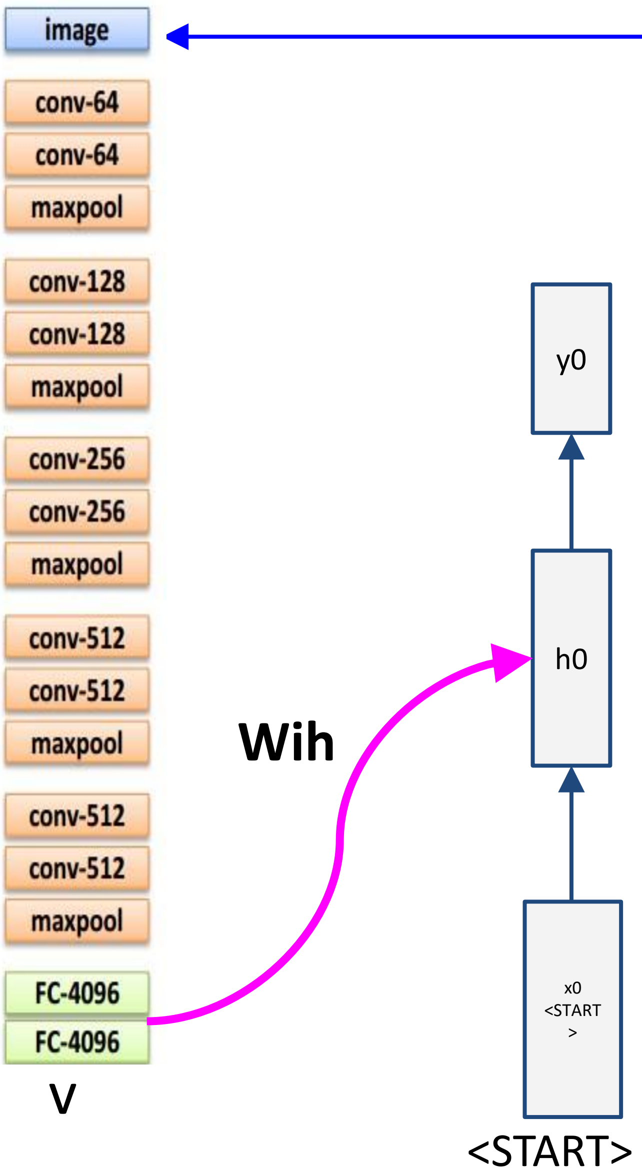


test image



<START>

Image Captioning



test image

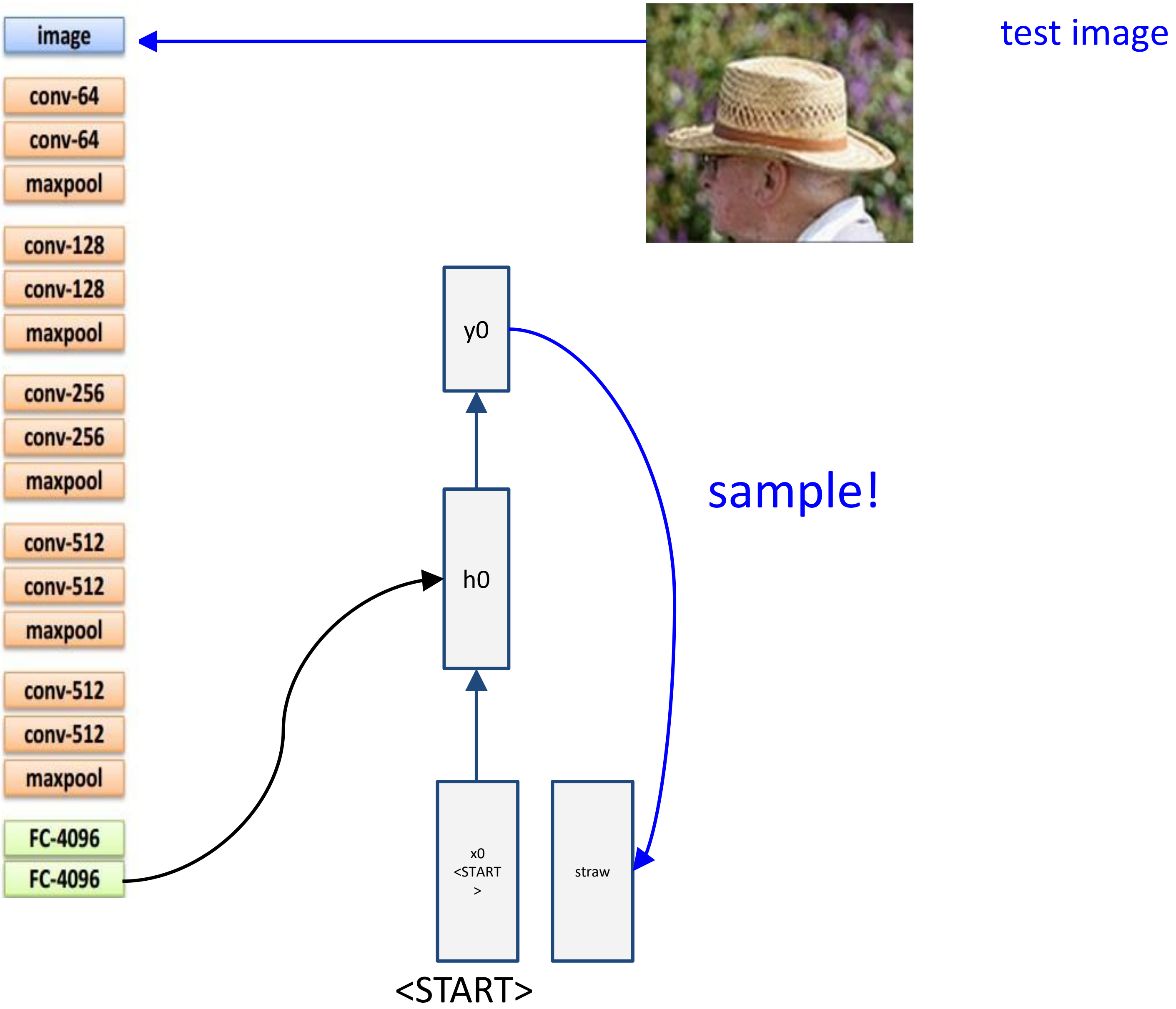
before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih * v**)

Image Captioning



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

test image

y0

sample!

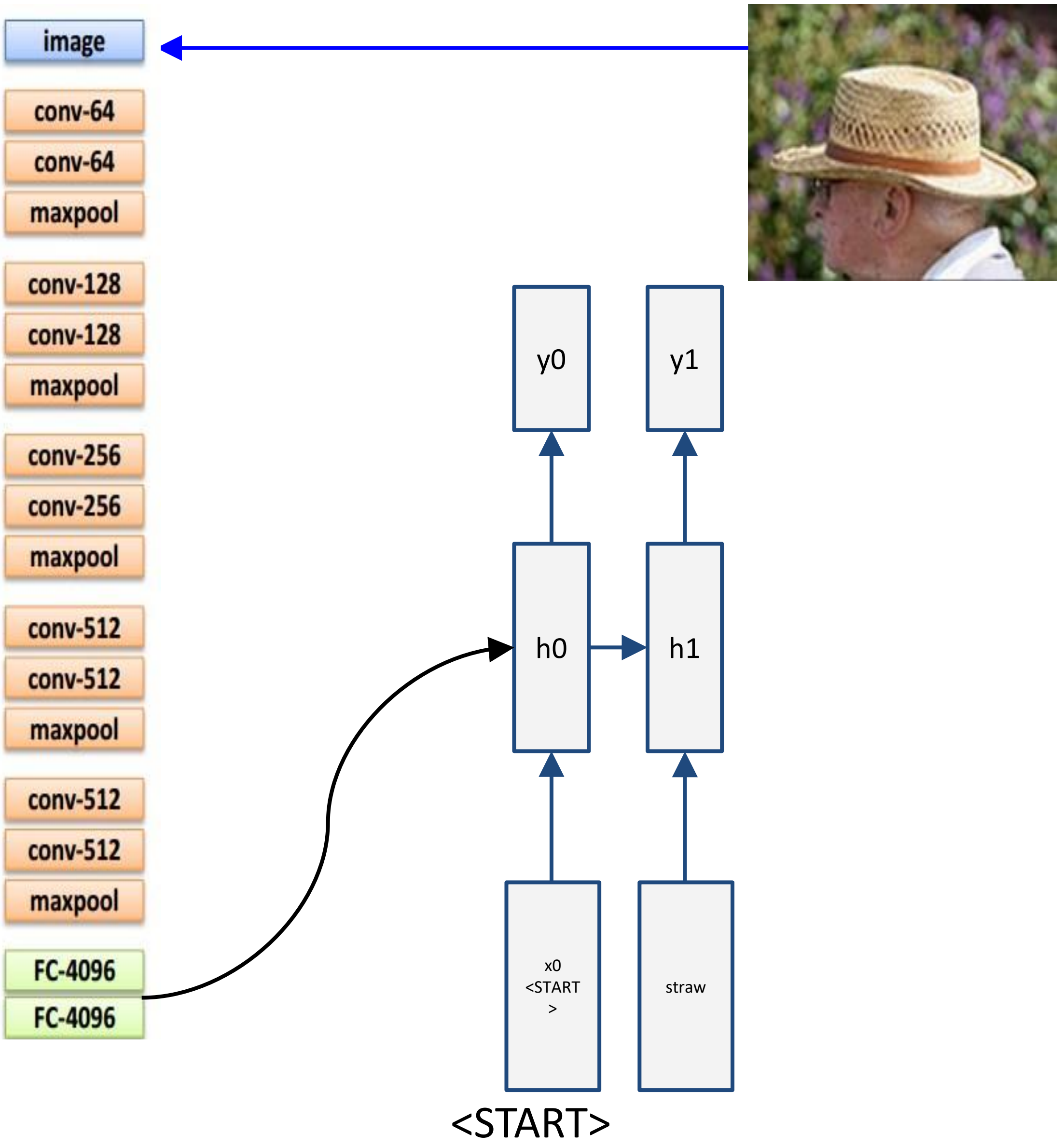
h0

x0 <START >

straw

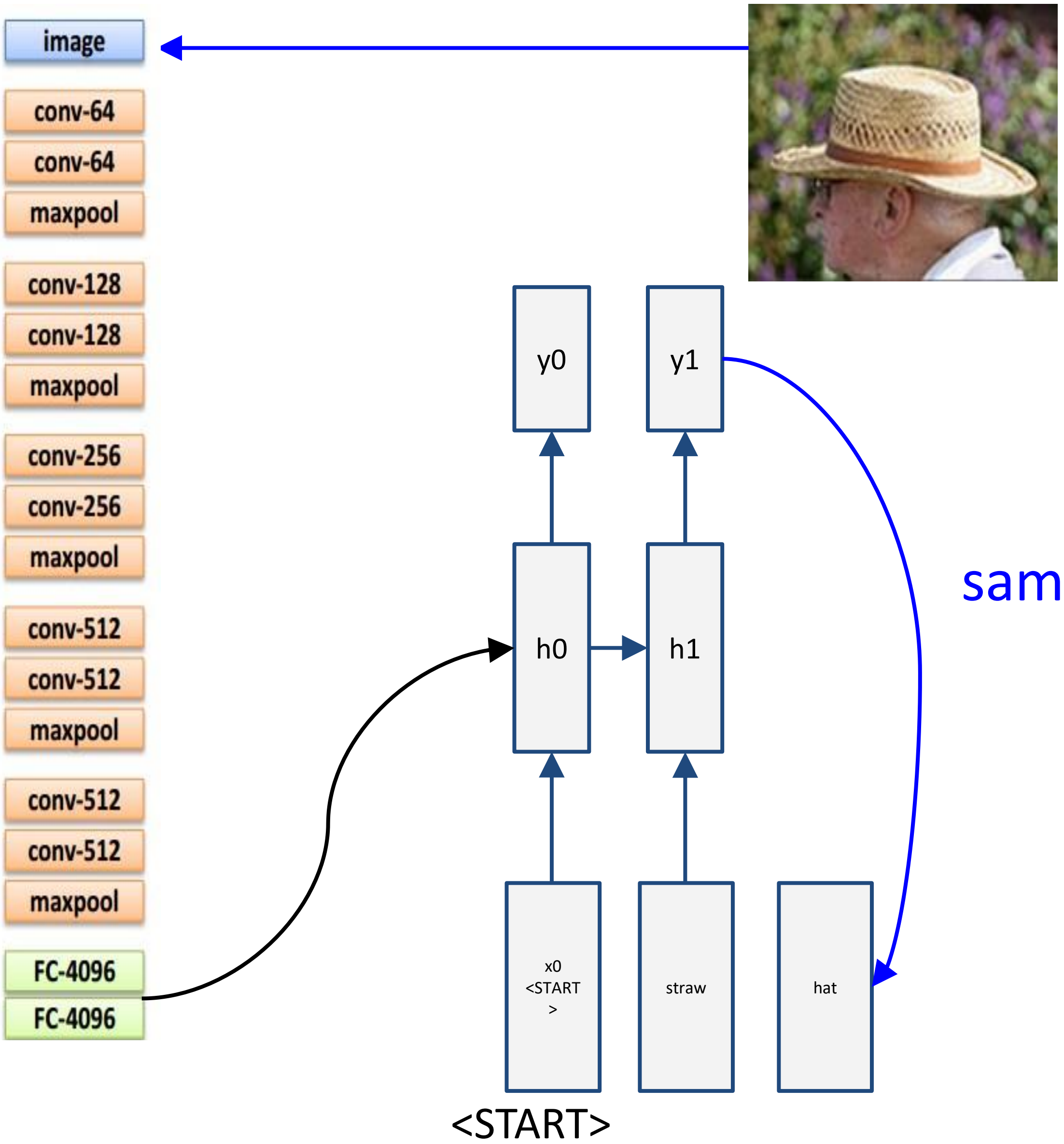
<START>

Image Captioning



test image

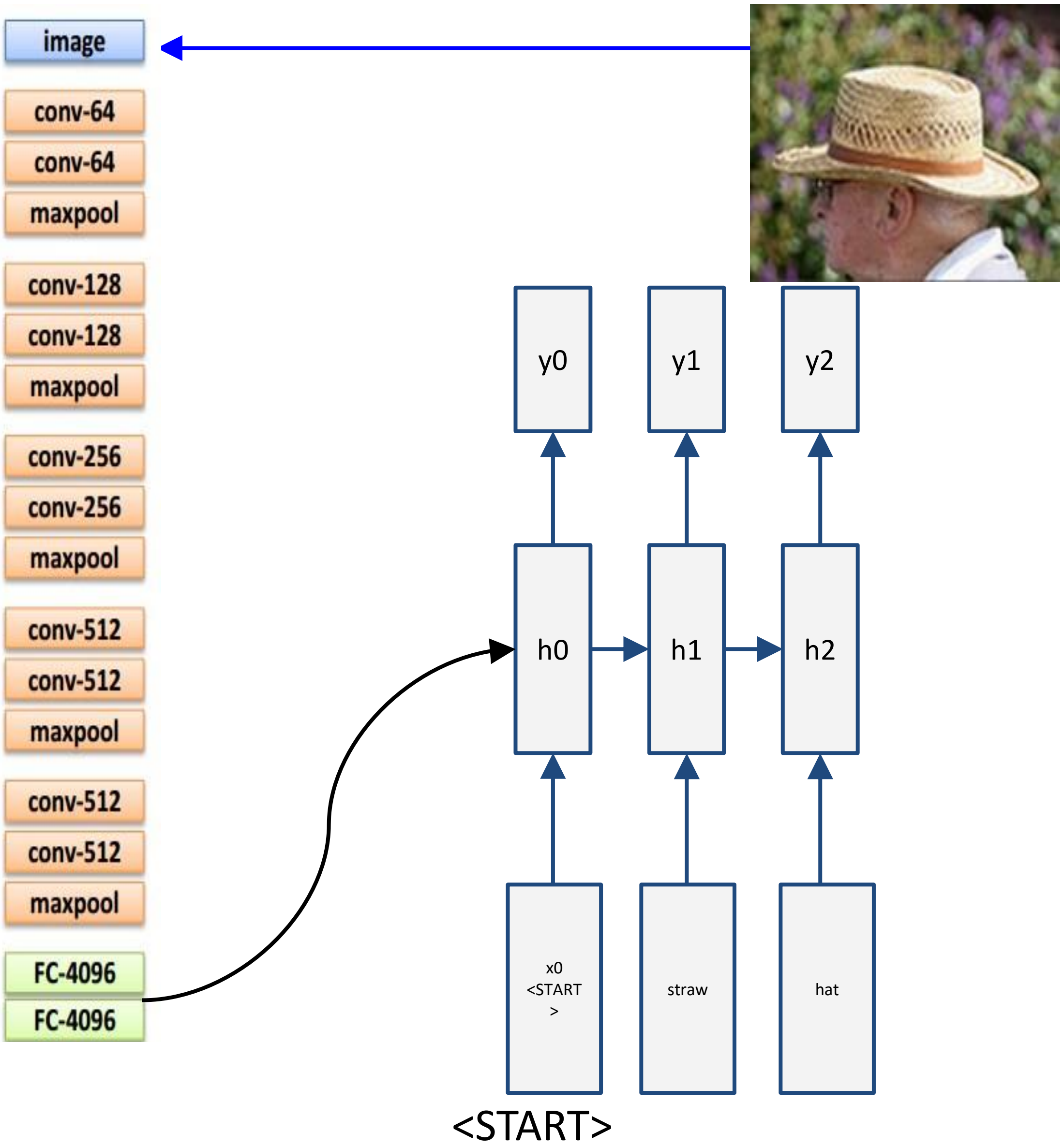
Image Captioning



test image

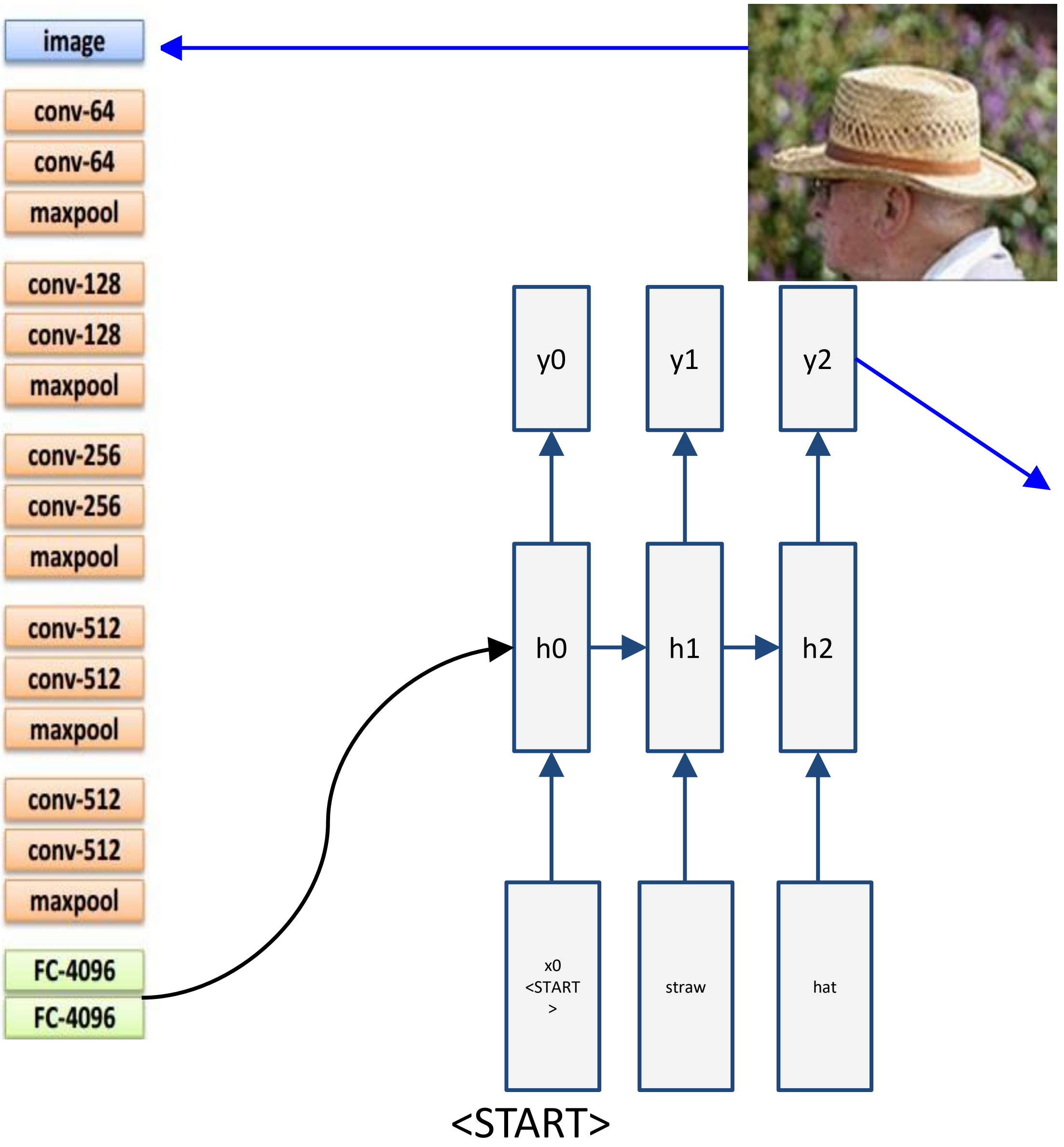
sample!

Image Captioning



test image

Image Captioning



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

y0 y1 y2

h0 h1 h2

x0 <START >

straw

hat

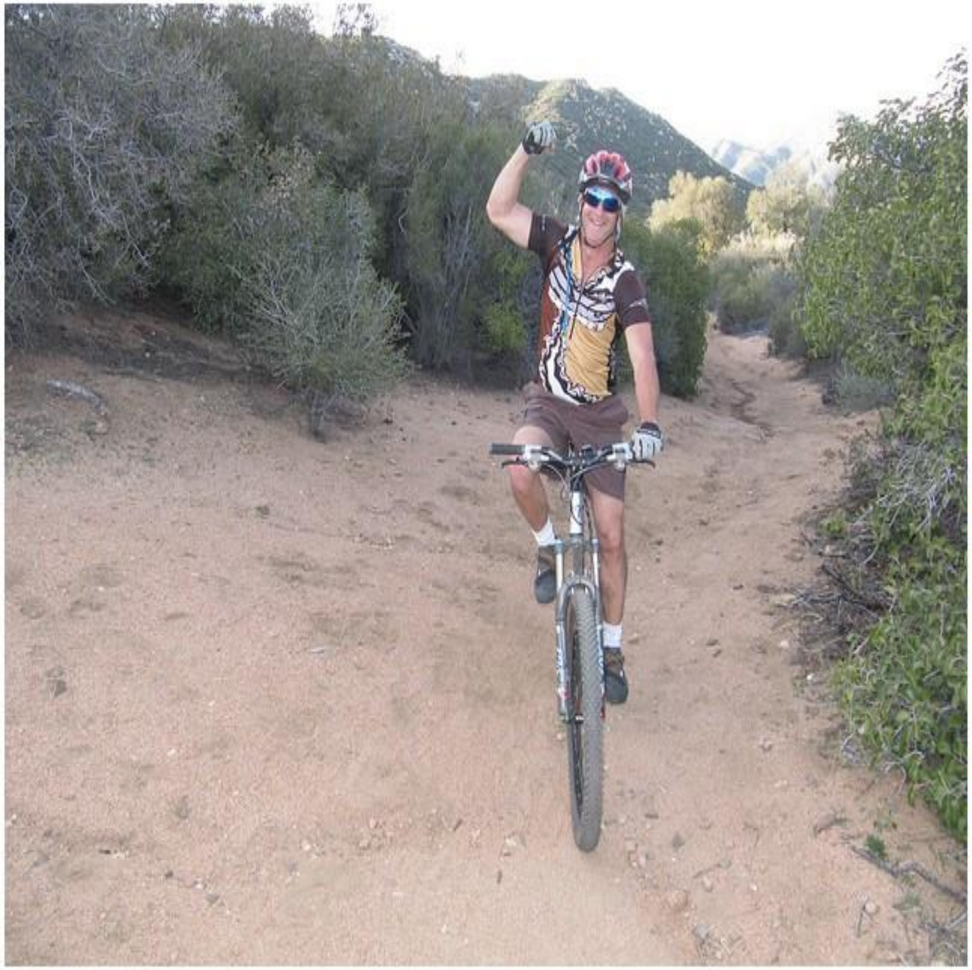
<START>

test image

sample
<END> token
=> finish.

Image Captioning Dataset

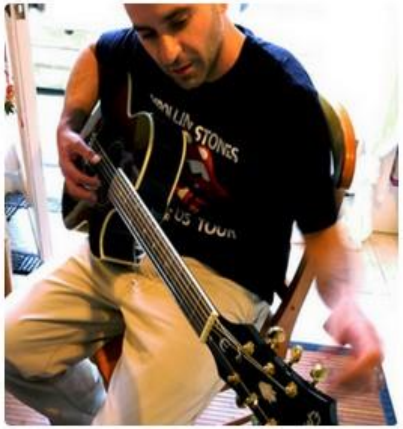
a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.



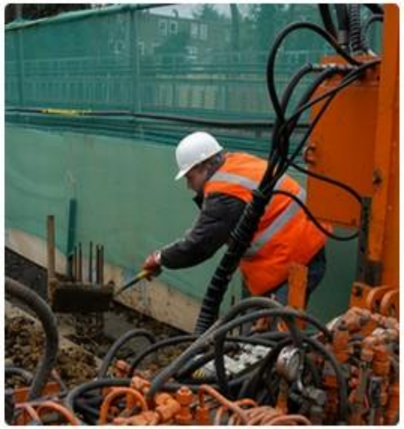
Microsoft COCO
[Tsung-Yi Lin et al. 2014]
mscoco.org

currently:
~120K images
~5 sentences each

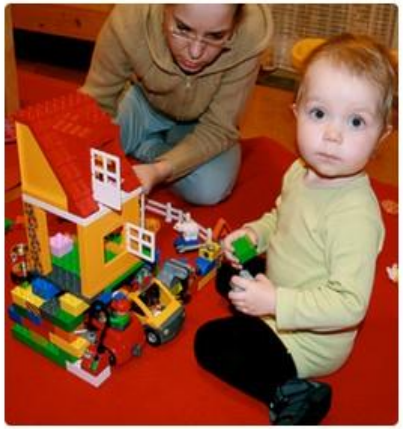
Image Captioning Results



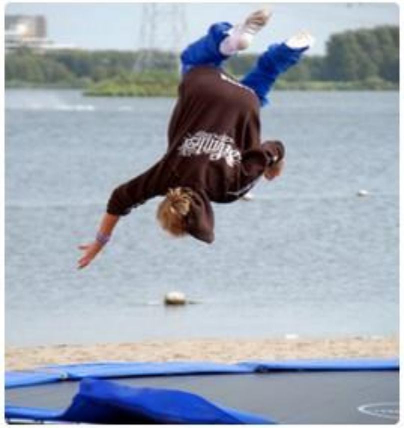
"man in black shirt is playing guitar."



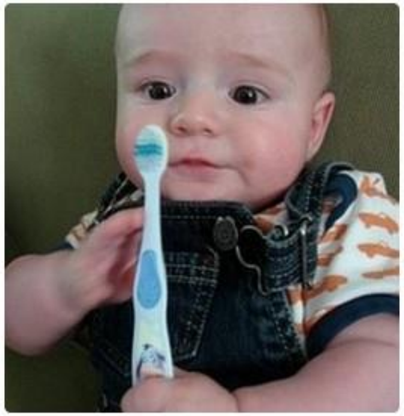
"construction worker in orange safety vest is working on road."



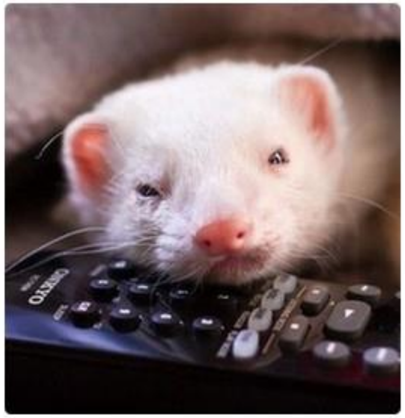
"two young girls are playing with lego toy."



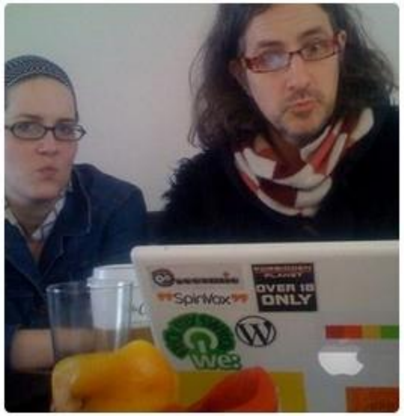
"boy is doing backflip on wakeboard."



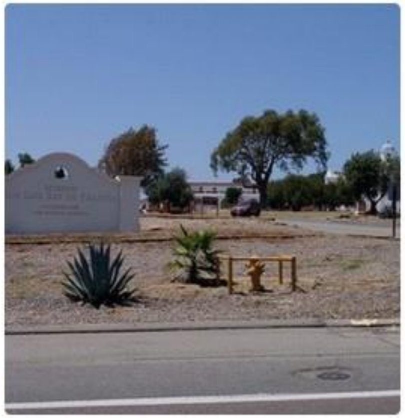
"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."



"a woman holding a teddy bear in front of a mirror."



"a horse is standing in the middle of a road."

Next Time:

- Regularization and Optimization

Acknowledgement

Some of the materials in these slides are drawn inspiration from:

- Shubhendu Trivedi and Risi Kondor, University of Chicago, Deep Learning Course
- Hung-yi Lee, National Taiwan University, Machine Learning and having it Deep and Structured course
- Xiaogang Wang, The Chinese University of Hong Kong, Deep Learning Course
- Fei-Fei Li, Standord University, CS231n Convolutional Neural Networks for Visual Recognition course

Questions?

Thank You !