



Recurrent Networks and Applications

Zhanxing Zhu

School of Mathematical Sciences
Peking University

Email: zhanxing.zhu@pku.edu.cn

Website: <https://sites.google.com/view/zhanxingzhu/>

Several slides are adapted from related courses or tutorials
(particularly from Dr. Yadong Mu).
Internal use only. Please do not distribute the slides.

Outline

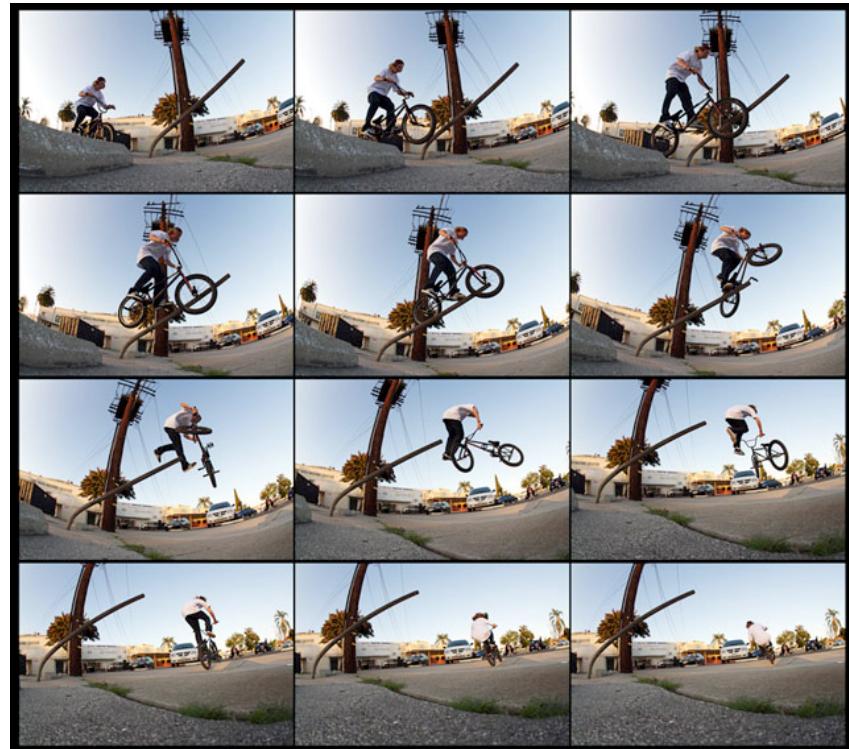
- **Recurrent Networks: Motivation and Introduction**
- **Models**
 - RNN
 - LSTM
- **Applications**
 - Image Captioning
 - Convolutional LSTM
 - Social LSTM

From Static Data to Sequential Data

- Videos as a sequence of frames / shots



Single Image



Frame Sequence

From Static Data to Sequential Data

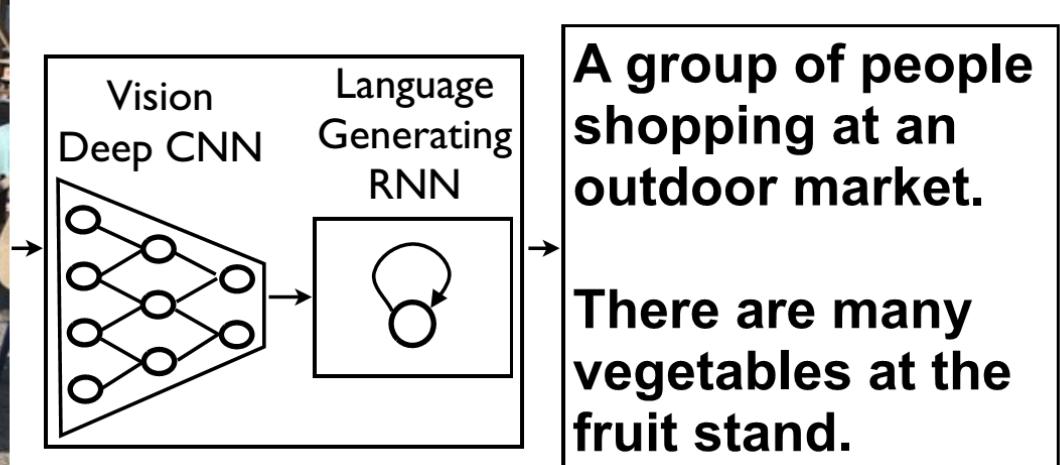
■ Stock price



Stock price of DL/BD related companies

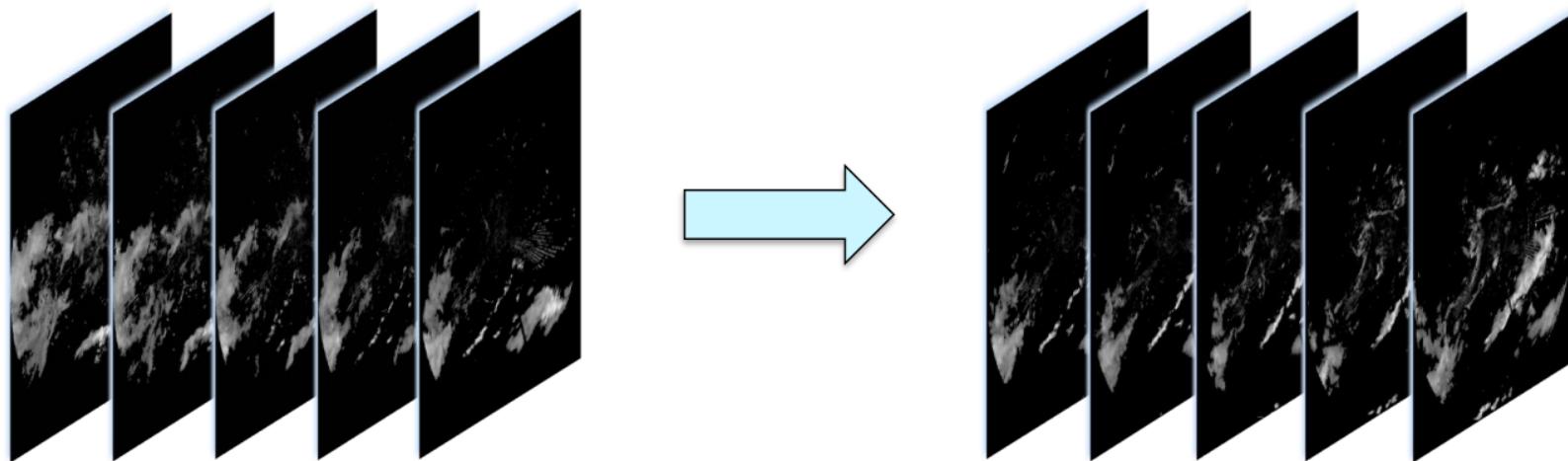
From Static Data to Sequential Data

- Image captioning: generate a sentence to describe the image semantics



From Static Data to Sequential Data

- Predict rainfall
 - Input sequence: observed radar maps up to current time step
 - Output sequence: predicted radar maps for future time steps



$$\tilde{\mathcal{X}}_{t+1}, \dots, \tilde{\mathcal{X}}_{t+K} = \arg \max_{\mathcal{X}_{t+1}, \dots, \mathcal{X}_{t+K}} p(\mathcal{X}_{t+1}, \dots, \mathcal{X}_{t+K} \mid \hat{\mathcal{X}}_{t-J+1}, \hat{\mathcal{X}}_{t-J+2}, \dots, \hat{\mathcal{X}}_t)$$

From Static Data to Sequential Data

- Image captioning
- Sequence of words in an English sentence
- Acoustic features at successive time frames in speech recognition
- Successive frames in video classification
- Rainfall measurements on successive days in Hong Kong
- Daily values of current exchange rate
- Nucleotide base pairs in a strand of DNA
- Instead of making independent predictions on samples, assume the dependency among samples and make a sequence of decisions for sequential samples

Unfolding Computational Graphs

The classic form of a dynamic system

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

recurrent (definition of
s at time t refers back to
the same definition at
time t - 1)

For a finite number of time steps, unfold the graph via applying the definition multiple times

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

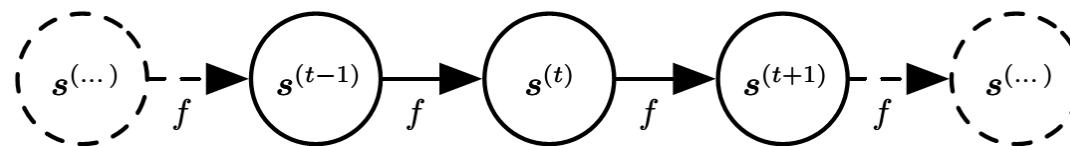


Figure 10.1

Unfolded computational graph. Each node represents the state at some time t , and the function f maps the state at t to the state at $t + 1$.

Unfolding Computational Graphs

Adding an external signal

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

Lossy summary of the task-relevant aspects of the past sequence of inputs

External signal

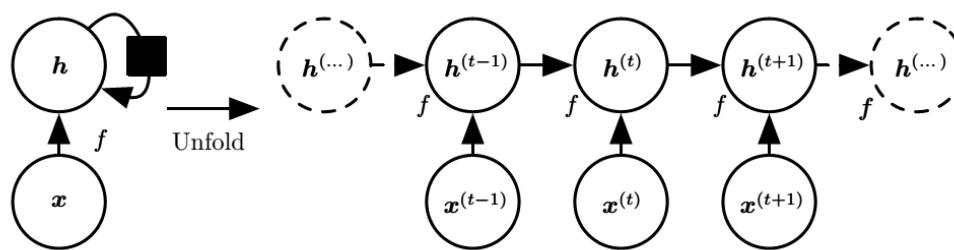


Figure 10.2

A recurrent network with no outputs. (left) circuit diagram. The black square indicates a delay of a single time step. (right) The same network seen as an unfolded computational graph, where each node is now associated with one particular time instance.

Remarks on Unfolding Computational Graphs

- Always same input / output sizes. Apply to input / output sequences of variable lengths
- The summary is lossy. It maps an arbitrary length sequence to a fixed length vector.
- Share a similar idea with CNN: replacing a fully connected network with local connections with parameter sharing

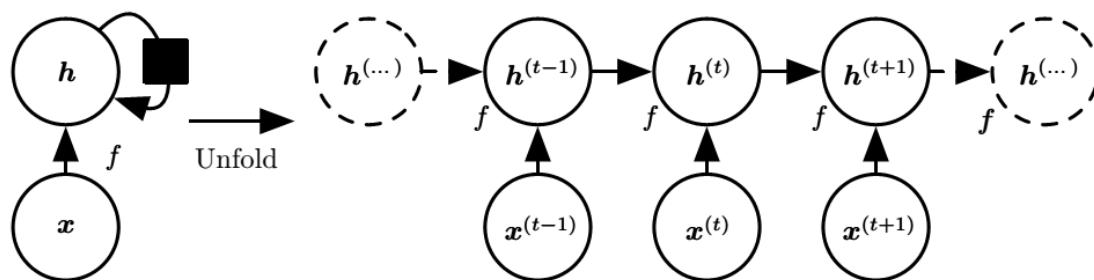


Figure 10.2

Remarks on Unfolding Computational Graphs

- **Sharing parameters for any sequence length allows more better generalization proper ties.** If we have to define a different function G_t for each possible sequence length, each with its own parameters, we would not get any generalization to sequences of a size not seen in the training set. One would need to see a lot more training examples, because a separate model would have to be trained for each sequence length.

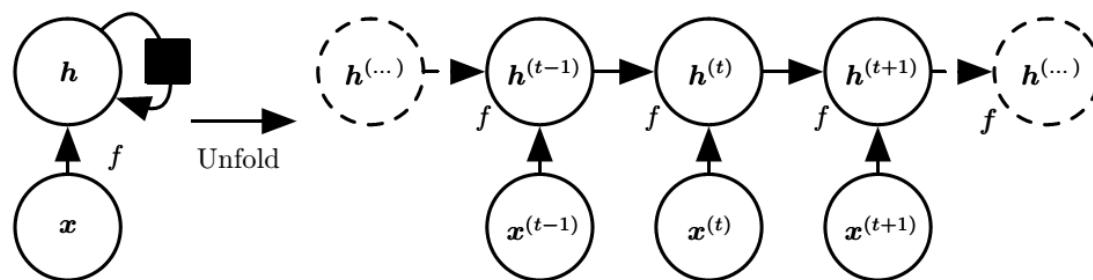


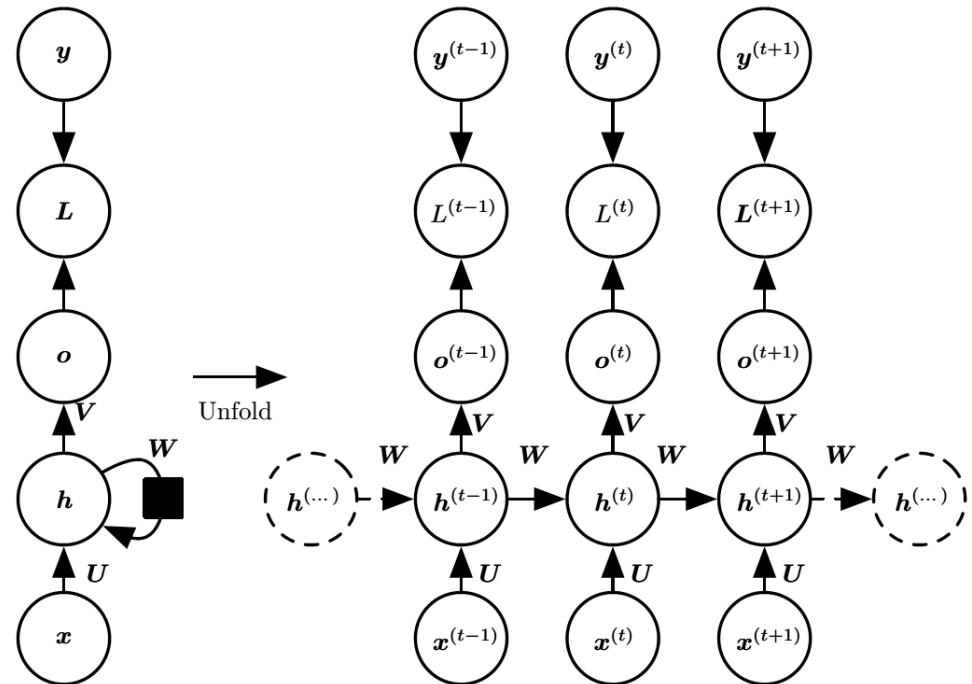
Figure 10.2

RNN with Hidden Units and Supervised Cost Function

- Produce an output at each time step and have recurrent connections between hidden units
- Assume hyperbolic tangent activation and unnormalized log probabilities
- Time and memory complexities are $O(T)$

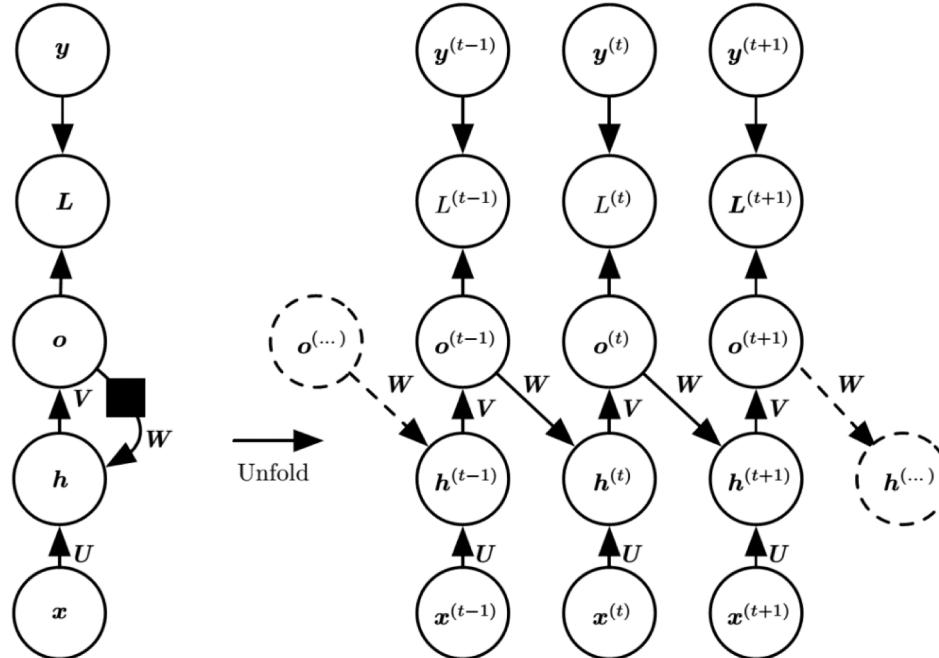
$$\begin{aligned} \mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}, \\ \mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}), \\ \mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}, \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)}) \end{aligned}$$

$$\begin{aligned} L\left(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\}\right) \\ = \sum_t L^{(t)} \\ = - \sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\}\right) \end{aligned}$$



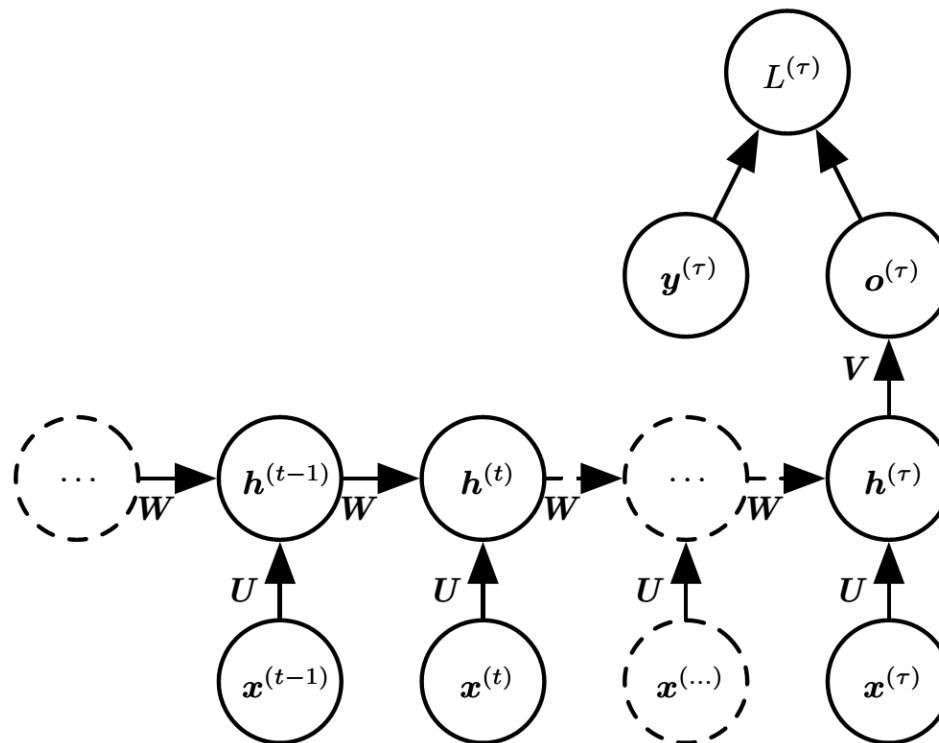
Recurrent Through Only the Output

- An alternative **without hidden-to-hidden recurrence**
- **Drawback:** the output units are close to the training set targets, unlikely to capture the necessary information about the past history of the input
- **Advantage:** Training can be parallelized. The gradient for each step t is computed in isolation.



Sequence Input, Single Output

- Time-unfolded recurrent neural network with a single output at the end of the sequence. Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing

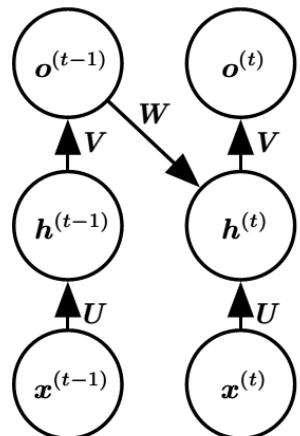
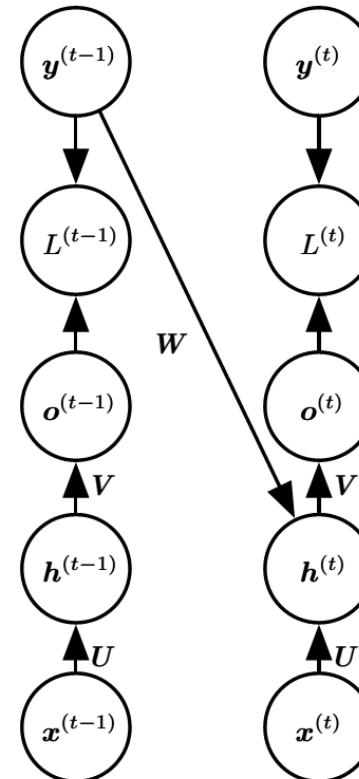


Teacher Forcing

- “...rather than feeding the model’s **own output** back into itself, these connections should be fed with **the target values** specifying what the correct output should be.”

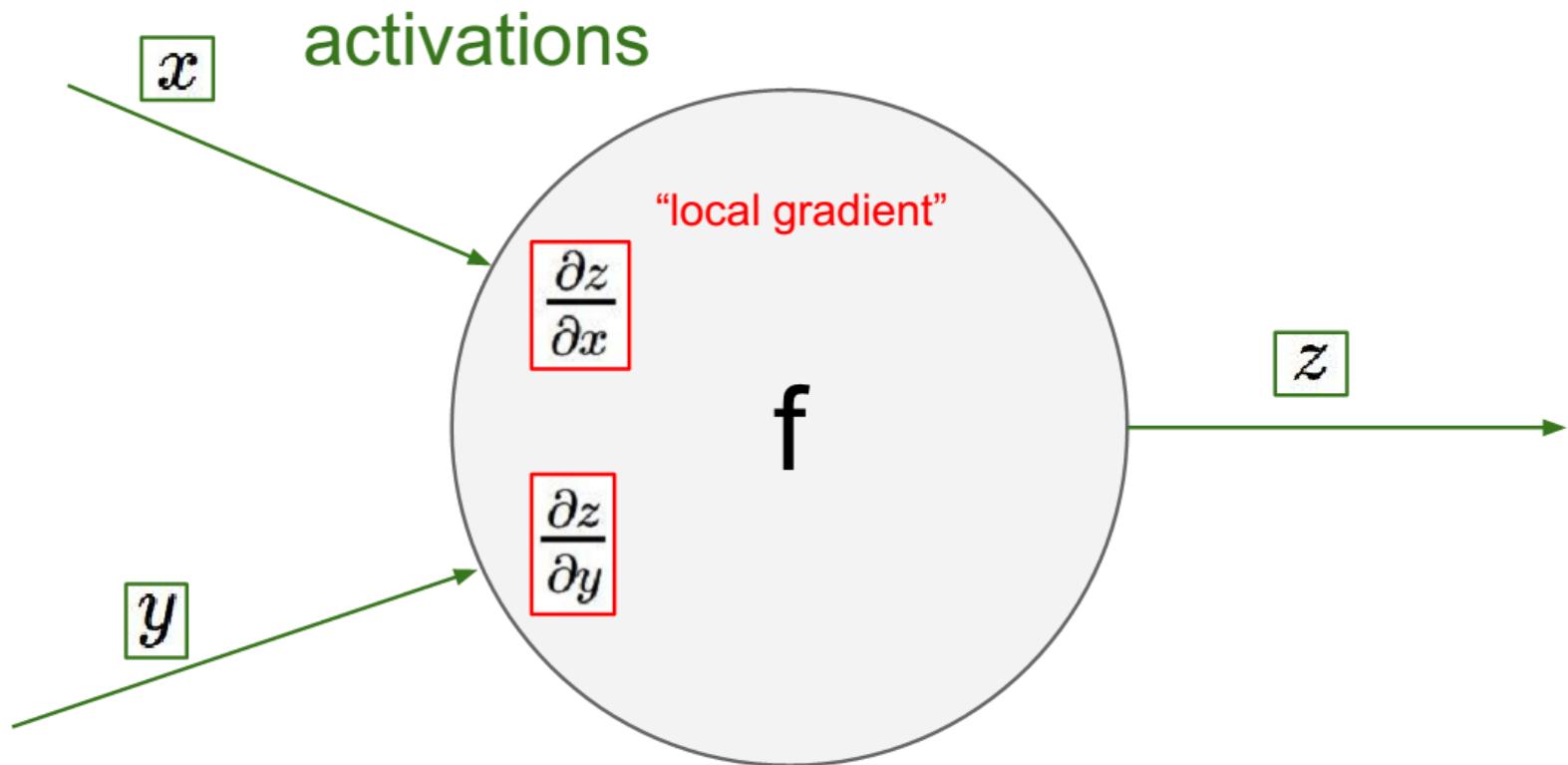
Optimization by a maximum likelihood criterion

$$\begin{aligned} & \log p(\mathbf{y}^{(1)}, \mathbf{y}^{(2)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \\ &= \log p(\mathbf{y}^{(2)} | \mathbf{y}^{(1)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) + \log p(\mathbf{y}^{(1)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}). \end{aligned}$$



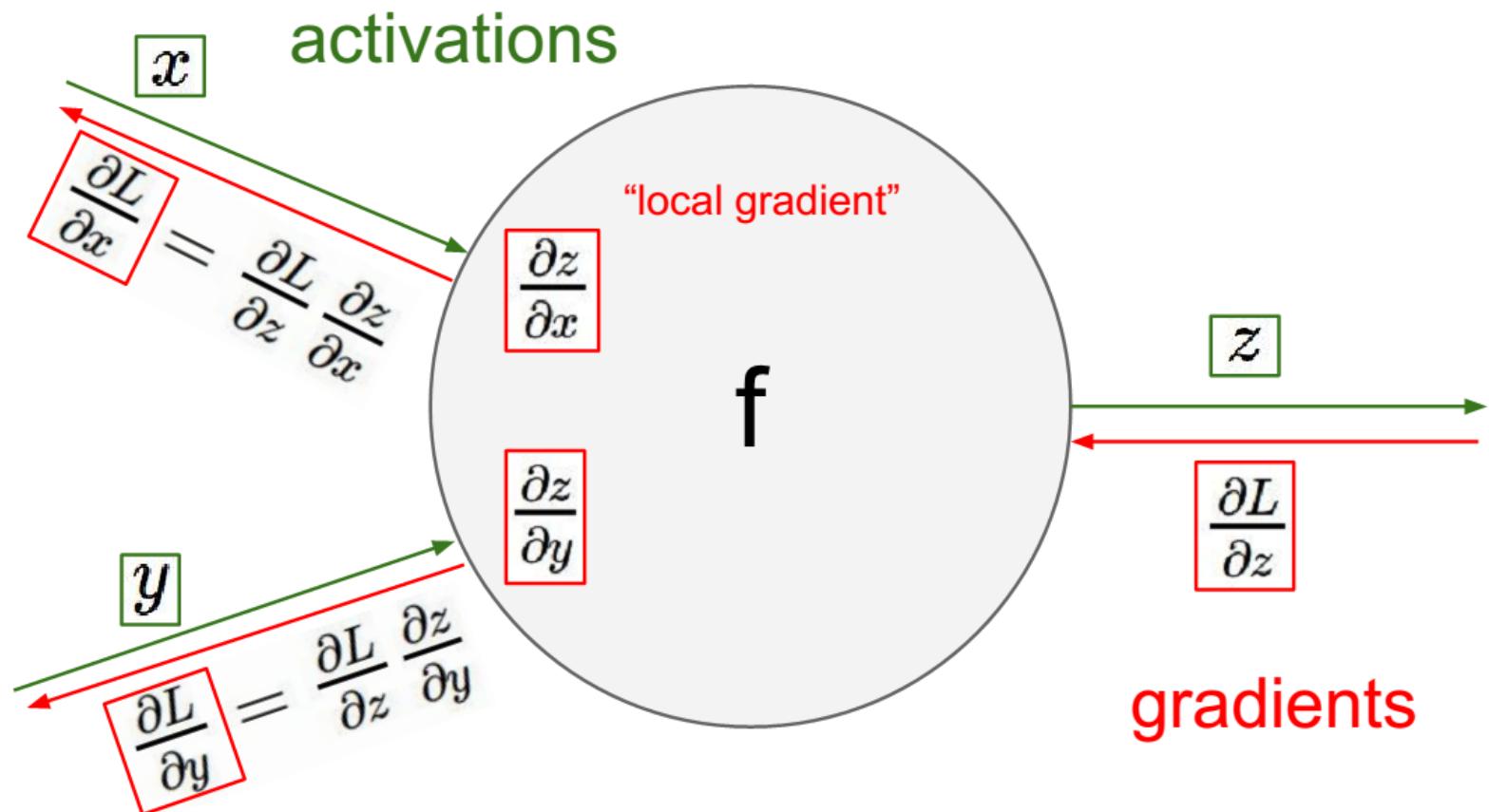
Review Back-Propagation on Deep Networks

- Anatomy of a neuron



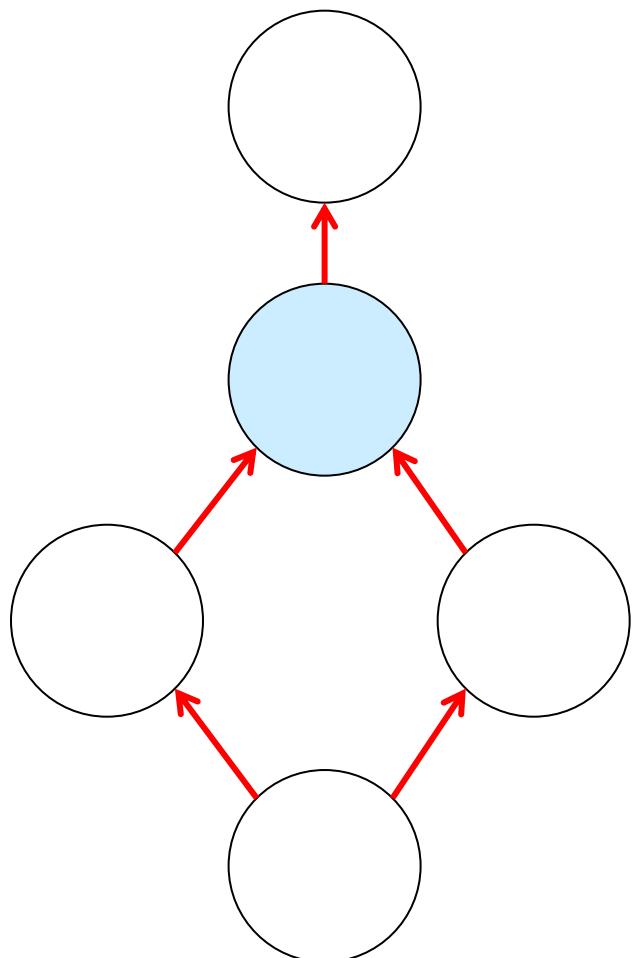
Review Back-Propagation on Deep Networks

- Anatomy of a neuron

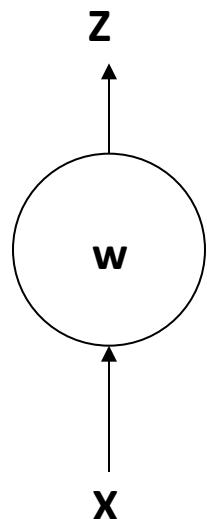
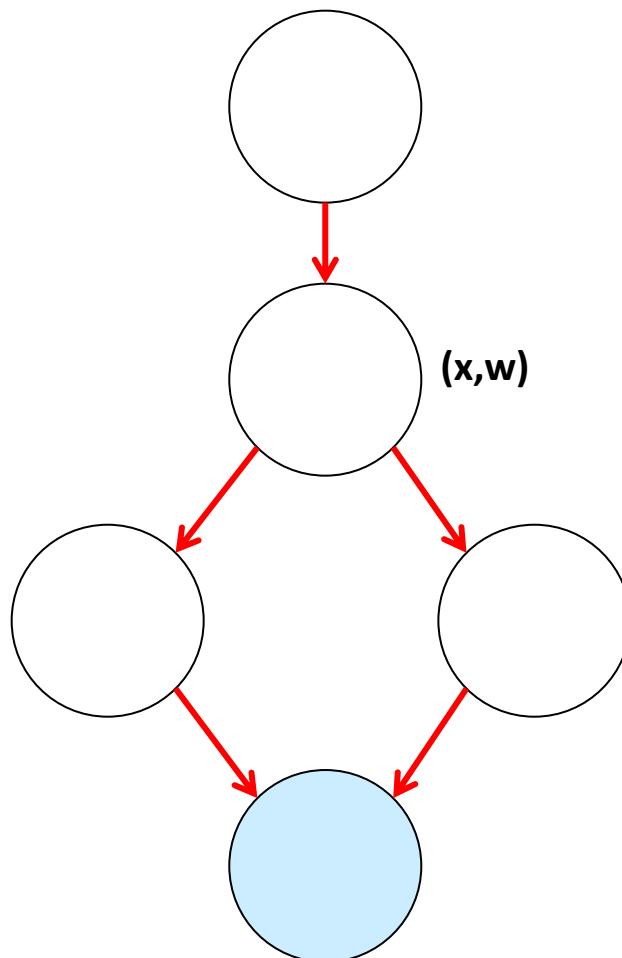


Review Back-Propagation on Deep Networks

Forward Pass
(bottom to top, calculate objective values on top layer)



Backward
(top to bottom, calculate gradient and update parameters)

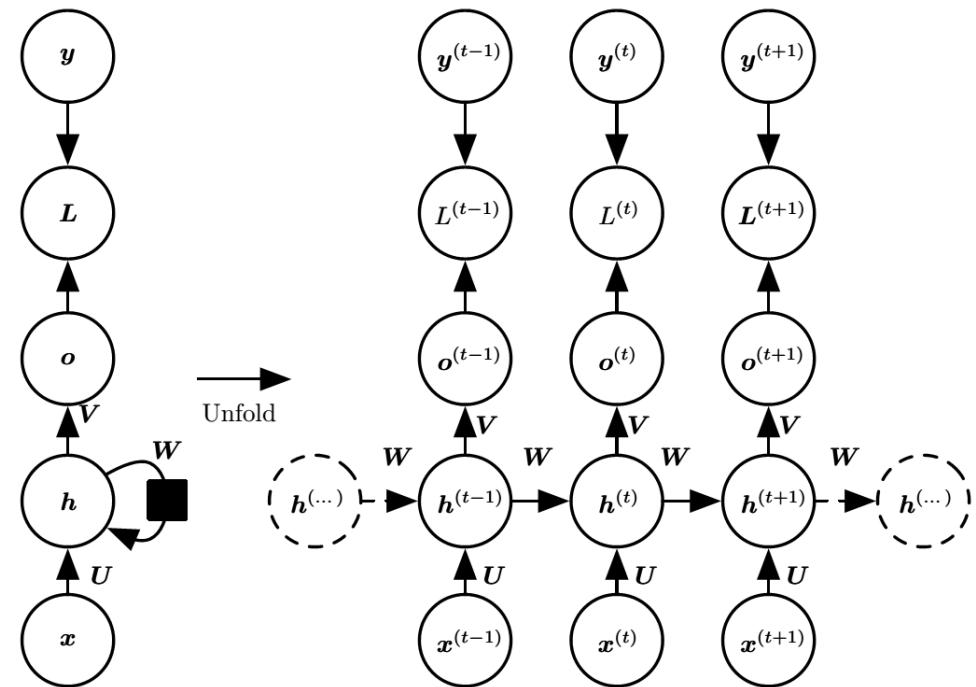


Back-Propagation (BP) on RNN

- No specialized algorithms are necessary. Directly apply BP to unrolled computational graph.
- Back-propagation through time (BPTT)

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

$$\begin{aligned}
 L\left(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\}\right) \\
 = \sum_t L^{(t)} \\
 = - \sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\}\right)
 \end{aligned}$$



Back-Propagation (BP) on RNN

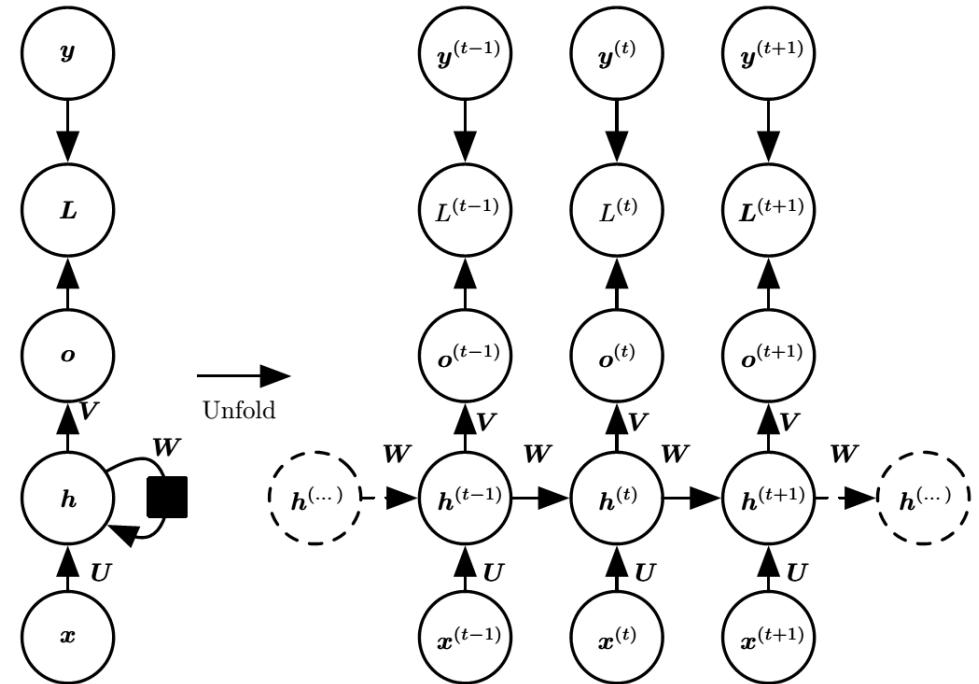
$$\frac{\partial L}{\partial L^{(t)}} = 1.$$

$$(\nabla_{\mathbf{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}}$$

$$\nabla_{\mathbf{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\mathbf{V}} o_i^{(t)}$$

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

$$\begin{aligned}L &\left(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\} \right) \\ &= \sum_t L^{(t)} \\ &= - \sum_t \log p_{\text{model}} \left(y^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\} \right)\end{aligned}$$



Back-Propagation (BP) on RNN

$$\frac{\partial L}{\partial L^{(t)}} = 1.$$

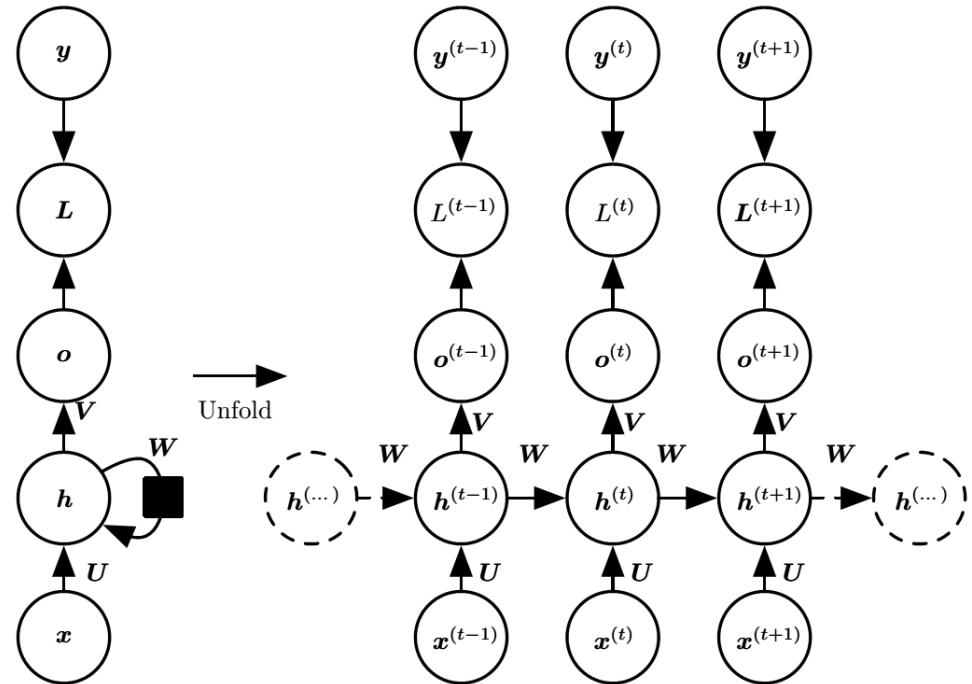
$$(\nabla_{\mathbf{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}}$$

$$\nabla_{\mathbf{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\mathbf{V}} o_i^{(t)}$$

$$\nabla_{\mathbf{h}^{(t)}} L = \left(\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \right)^\top (\nabla_{\mathbf{h}^{(t+1)}} L) + \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \right)^\top (\nabla_{\mathbf{o}^{(t)}} L)$$

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

$$\begin{aligned}L &\left(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\} \right) \\ &= \sum_t L^{(t)} \\ &= - \sum_t \log p_{\text{model}} \left(y^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\} \right)\end{aligned}$$



Back-Propagation (BP) on RNN

$$\frac{\partial L}{\partial L^{(t)}} = 1.$$

$$(\nabla_{\mathbf{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}}$$

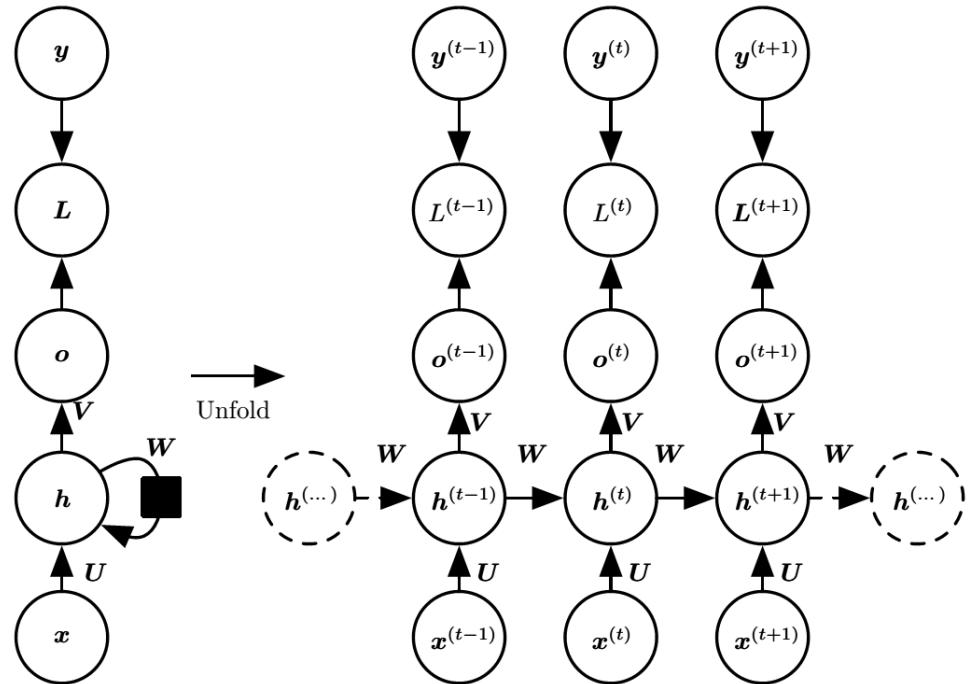
$$\nabla_{\mathbf{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\mathbf{V}} o_i^{(t)}$$

$$\nabla_{\mathbf{h}^{(t)}} L = \left(\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \right)^\top (\nabla_{\mathbf{h}^{(t+1)}} L) + \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \right)^\top (\nabla_{\mathbf{o}^{(t)}} L)$$

$$\nabla_{\mathbf{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\mathbf{W}^{(t)}} h_i^{(t)} \quad \nabla_{\mathbf{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\mathbf{U}^{(t)}} h_i^{(t)}$$

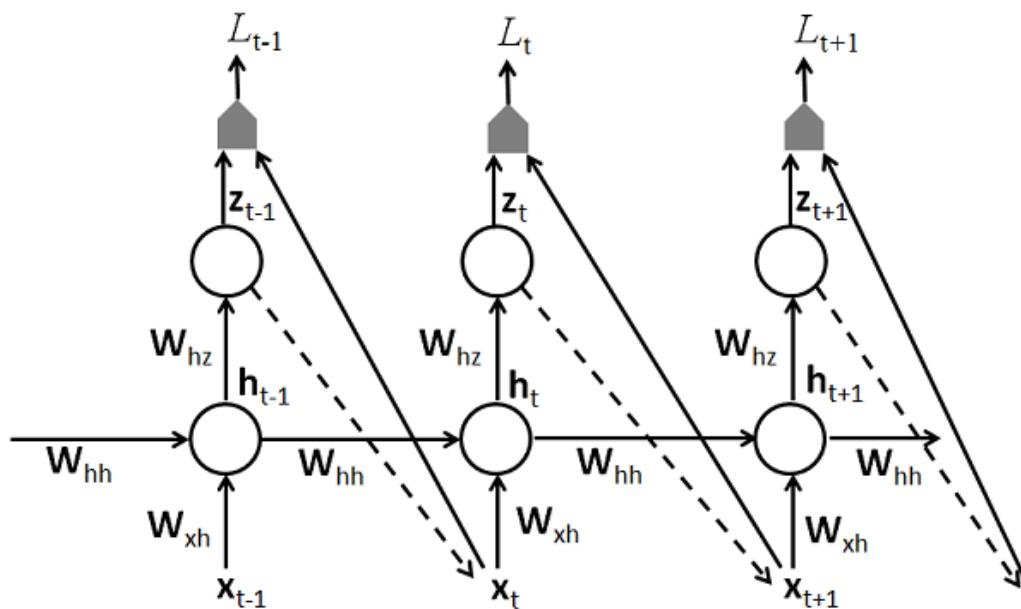
$$\begin{aligned} \mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}, \\ \mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}), \\ \mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}, \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)}) \end{aligned}$$

$$\begin{aligned} L &\left(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\} \right) \\ &= \sum_t L^{(t)} \\ &= - \sum_t \log p_{\text{model}} \left(y^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\} \right) \end{aligned}$$



Generative RNN modeling $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$

- It can generate sequences from this distribution
- At the training stage, each \mathbf{x}_t of the observed sequence serves both as input (for the current time step) and as target (for the previous time step)
- The output \mathbf{z}_t encodes the parameters of a conditional distribution $P(\mathbf{x}_{t+1}|\mathbf{x}_1, \dots, \mathbf{x}_t) = P(\mathbf{x}_{t+1}|\mathbf{z}_t)$ for \mathbf{x}_{t+1} given the past sequence $\mathbf{x}_1, \dots, \mathbf{x}_t$



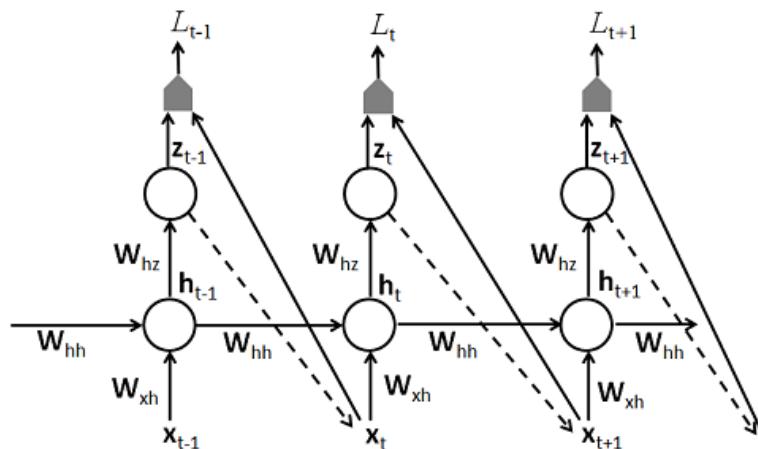
Generative RNN modeling $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$

- Cost function: negative log-likelihood of \mathbf{x} , $L = \sum_t L_t$

$$P(\mathbf{x}) = P(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T P(\mathbf{x}_t | \mathbf{x}_{t-1}, \dots, \mathbf{x}_1)$$

$$L_t = -\log P(\mathbf{x}_t | \mathbf{x}_{t-1}, \dots, \mathbf{x}_1)$$

- In generative mode, \mathbf{x}_{t+1} is sampled from the conditional distribution $P(\mathbf{x}_{t+1} | \mathbf{x}_1, \dots, \mathbf{x}_t) = P(\mathbf{x}_{t+1} | \mathbf{z}_t)$ (dashed arrows) and then that generated sample \mathbf{x}_{t+1} is fed back as input for computing the next state \mathbf{h}_{t+1}



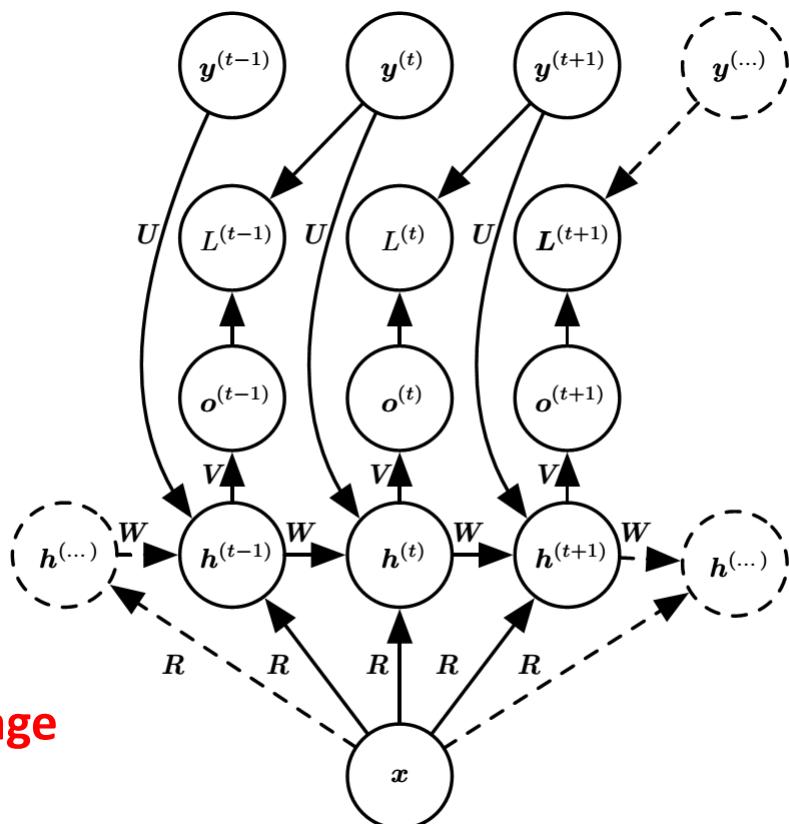
Generative RNN modeling $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$

- If RNN is used to generate sequences, one must also incorporate in the output information allowing to stochastically decide when to stop generating new output elements
- In the case when the output is a symbol taken from a vocabulary, one can add a special symbol corresponding to the end of a sequence
- One could also directly model the length T of the sequence through some parametric distribution. $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$ is decomposed into

$$P(\mathbf{x}_1, \dots, \mathbf{x}_T) = P(\mathbf{x}_1, \dots, \mathbf{x}_T | T)P(T)$$

Vector to Sequence: Modeling Sequences Conditioned on Context with RNNs

- When x is a fixed-size vector, we can simply make it an extra input of the RNN that generates the y sequence. Some common ways of providing an extra input to an RNN are
 - as an extra input at each time step, or
 - as the initial state, or
 - both



Example: generate caption for an image

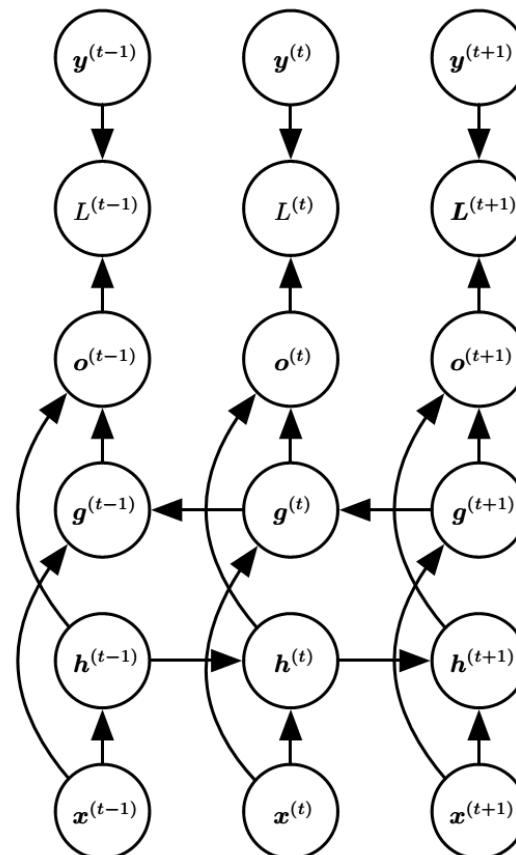
Bidirectional RNN

- In many applications, however, we want to output a prediction that may depend on the whole input sequence.

For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and may even depend on the next few words because of the linguistic dependencies between nearby words:

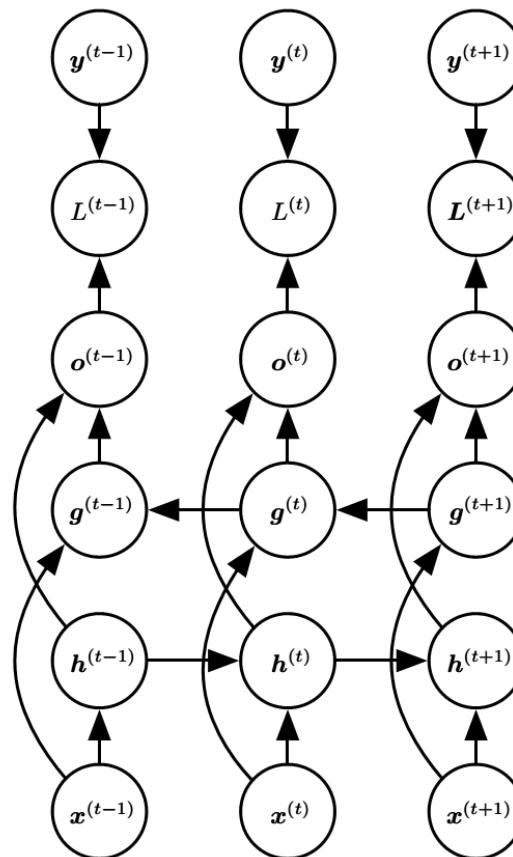
if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them.

This is also true of handwriting recognition and many other sequence-to-sequence learning tasks



Bidirectional RNN

- Bidirectional recurrent neural network was proposed to address such need. It combines a forward-going RNN and a backward-going RNN



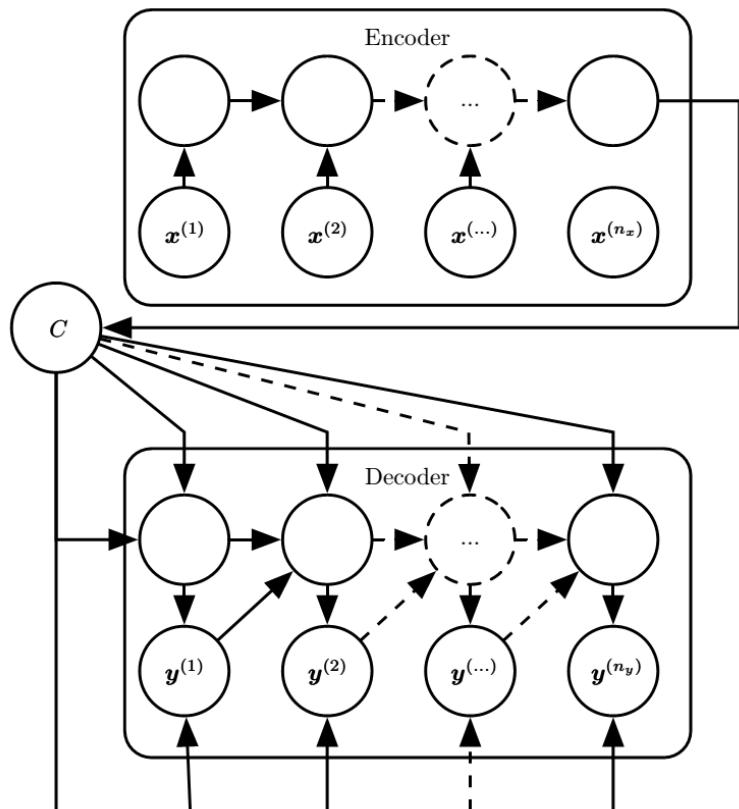
**The idea can be extended to 2D
input with four RNN going in four
directions**

Sequence-to-Sequence Architecture

- map an input sequence to an output sequence which is not necessarily of the same length
- speech recognition, machine translation, question answering

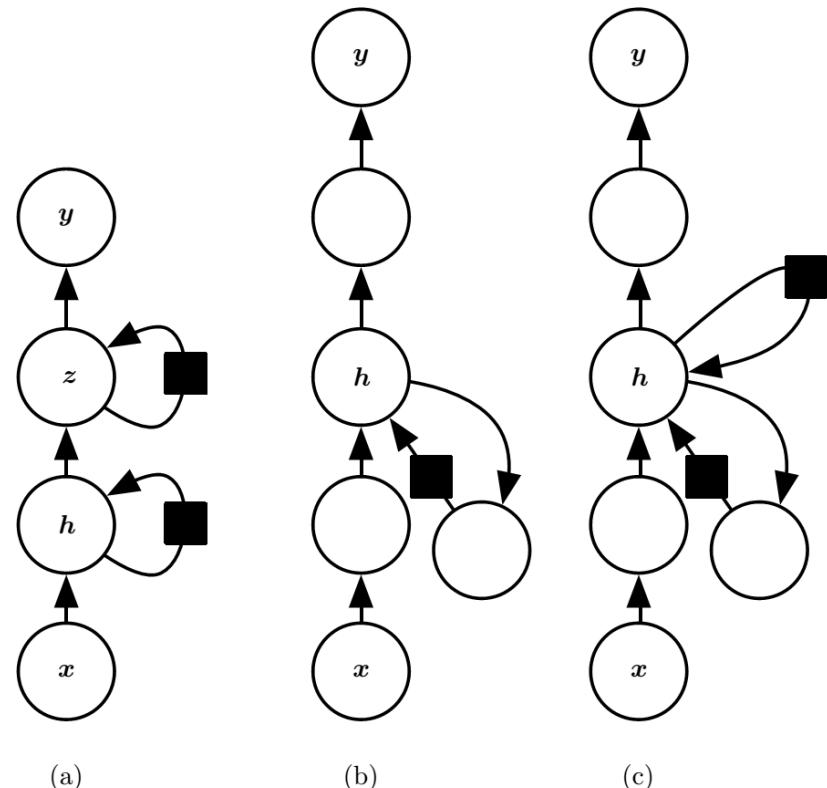
An **encoder** or reader or input RNN processes the input sequence. The encoder emits the context C , usually as a simple function of its final hidden state.

A **decoder** or writer or output RNN is conditioned on that fixed-length vector to generate the output sequence



Deep RNNs

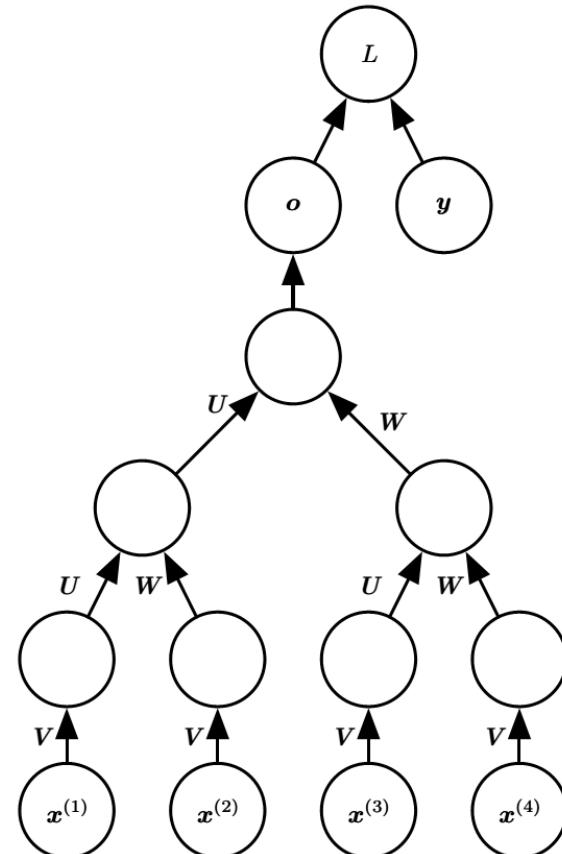
- We can think of the lower layers in the hierarchy as playing a role in transforming the raw input into a representation that is more appropriate, at the higher levels of the hidden state.



Recursive Network

- structured as a deep tree, rather than the chain-like structure of RNNs
- Reduced computation
- Tree structure can be learned or by experience

For example, the tree structure for the recursive network can be fixed to the structure of the parse tree of the sentence provided by a natural language parser



The Challenge of Long Term Dependencies

- Gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization)
- Let us consider a simple case

$$\mathbf{h}^{(t)} = \mathbf{W}^\top \mathbf{h}^{(t-1)}$$

No input, no nonlinear activation

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^\top \mathbf{h}^{(0)}$$

$$\mathbf{W} = \mathbf{Q} \boldsymbol{\Lambda} \mathbf{Q}^\top$$

Eigen-decomposition

$$\mathbf{h}^{(t)} = \mathbf{Q}^\top \boldsymbol{\Lambda}^t \mathbf{Q} \mathbf{h}^{(0)}$$

eigenvalues with magnitude less than one to decay to zero and
eigenvalues with magnitude greater than one to explode

The Challenge of Long Term Dependencies

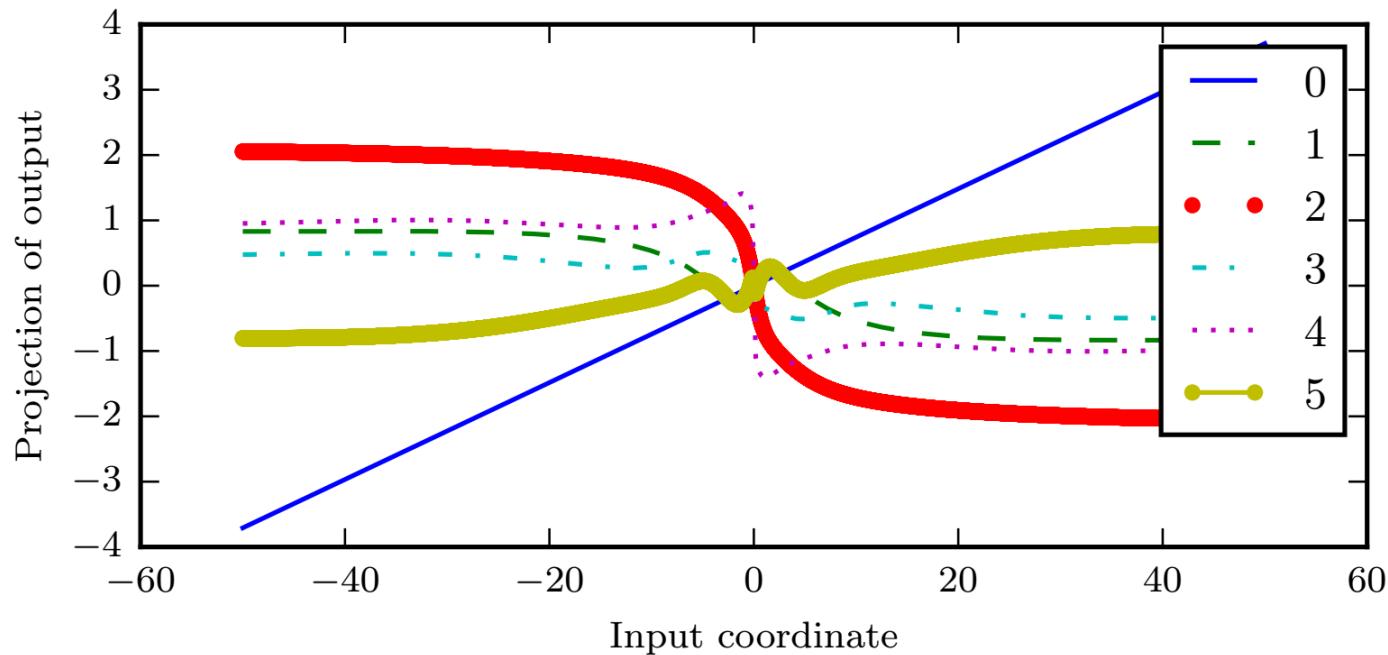


Figure 10.15: Repeated function composition. When composing many nonlinear functions (like the linear-tanh layer shown here), the result is highly nonlinear, typically with most of the values associated with a tiny derivative, some values with a large derivative, and many alternations between increasing and decreasing. Here, we plot a linear projection of a 100-dimensional hidden state down to a single dimension, plotted on the y -axis. The x -axis is the coordinate of the initial state along a random direction in the 100-dimensional space. We can thus view this plot as a linear cross-section of a high-dimensional function. The plots show the function after each time step, or equivalently, after each number of times the transition function has been composed.

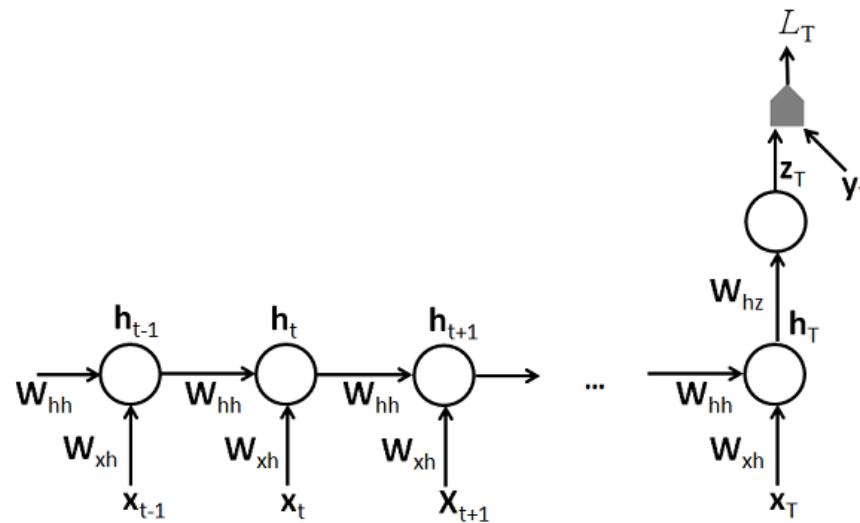
The Challenge of Long Term Dependencies

- Consider the gradient of a loss L_T at time T with respect to the parameter θ of the recurrent function F_θ

$$\mathbf{h}_t = F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\frac{\partial L_T}{\partial \theta} = \sum_{t \leq T} \frac{\partial L_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)}{\partial \theta}$$

$\frac{\partial L_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)}{\partial \theta}$ encodes long-term dependency when $T - t$ is large



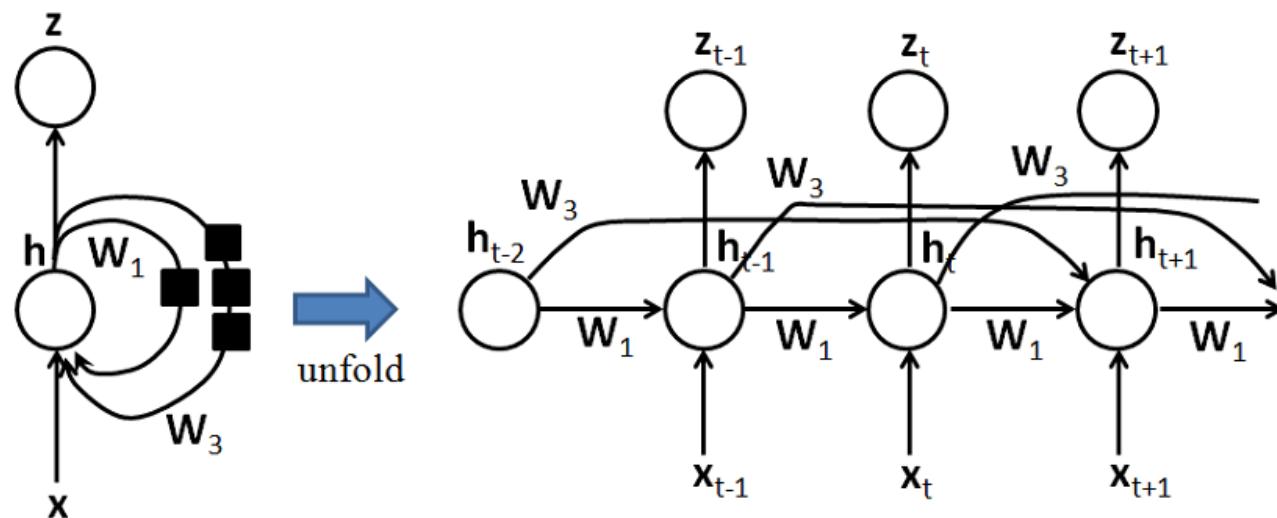
The Challenge of Long Term Dependencies

$$\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} = \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{h}_{T-2}} \dots \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

- Each layer-wise Jacobian $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$ is the product of two matrices: (a) the recurrent matrix \mathbf{W} and (b) the diagonal matrix whose entries are the derivatives of the non-linearities associated with the hidden units, which vary depending on the time step. This makes it likely that successive Jacobians have similar eigenvectors, making the product of these Jacobians explode or vanish even faster
- $\frac{\partial L_T}{\partial \theta}$ is a weighted sum of terms over spans $T - t$, with weights that are exponentially smaller (or larger) for long-term dependencies relating the state at t to the state at T
- The signal about long term dependencies will tend to be hidden by the smallest fluctuations arising from short-term dependencies

Combine Short and Long Paths in Unfolded Flow Graph

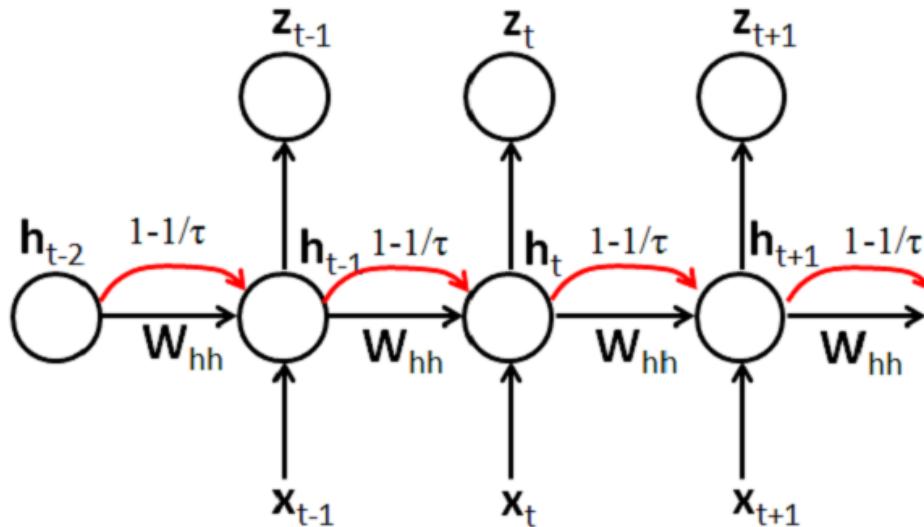
- Longer-delay connections allow to connect the past states to future states through short paths
- Gradients will vanish exponentially with respect to the number of time steps
- If we have recurrent connections with a time-delay of D , the instead of the vanishing or explosion going as $O(\lambda^T)$ over T steps (where λ is largest eigenvalue of the Jacobians $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$), the unfolded recurrent network now has paths through which gradients grow as $O(\lambda^{T/D})$ because the number of effective steps is T/D



Leaky units with self-connections

$$\mathbf{h}_{t+1} = \left(1 - \frac{1}{\tau_i}\right)\mathbf{h}_t + \frac{1}{\tau_i}\tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_t + \mathbf{b}_h)$$

- The new value of the state \mathbf{h}_{t+1} is a combination of linear and non-linear parts of \mathbf{h}_t
- The errors are easier to be back propagated through the paths of red lines, which are linear



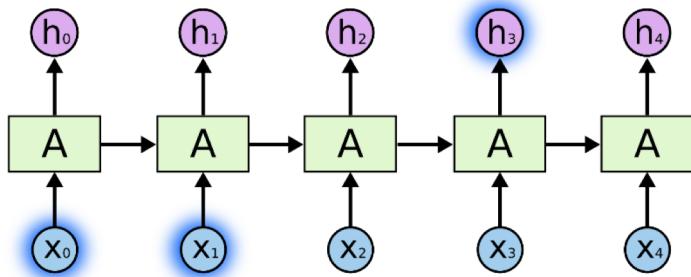
Leaky units with self-connections

- When $\tau = 1$, there is no linear self-recurrence, only the nonlinear update which we can find in ordinary recurrent networks
- When $\tau > 1$, this linear recurrence allows gradients to propagate more easily. When τ is large, the state changes very slowly, integrating the past values associated with the input sequence
- τ controls the rate of forgetting old states. It can be viewed as a smooth variant of the idea of the previous model
- By associating different time scales τ with different units, one obtains different paths corresponding to different forgetting rates
- Those time constants can be fixed manually or can be learned as free parameters

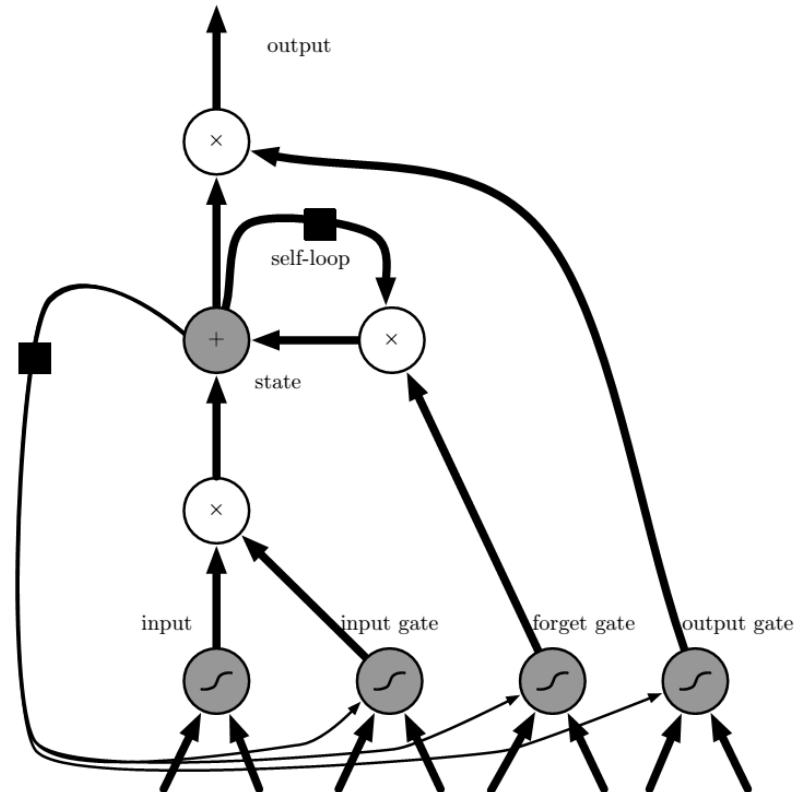
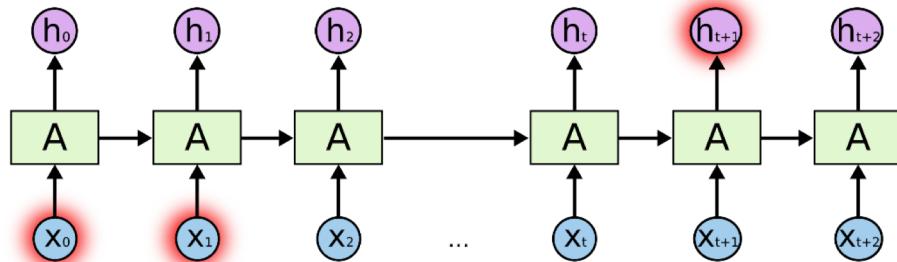
Long Short-Term Memory (LSTM)

- Sometimes, recent information is not sufficient. We need long-term dependencies

“the clouds are in the **sky**,”



“I grew up in France... I speak fluent **French**.”

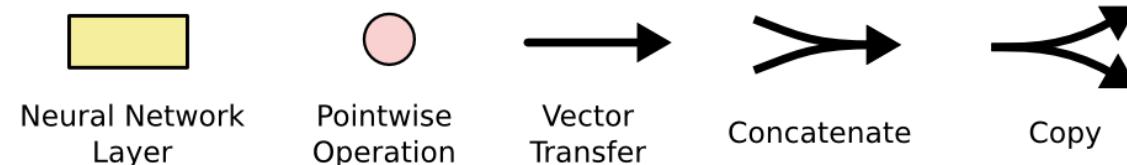
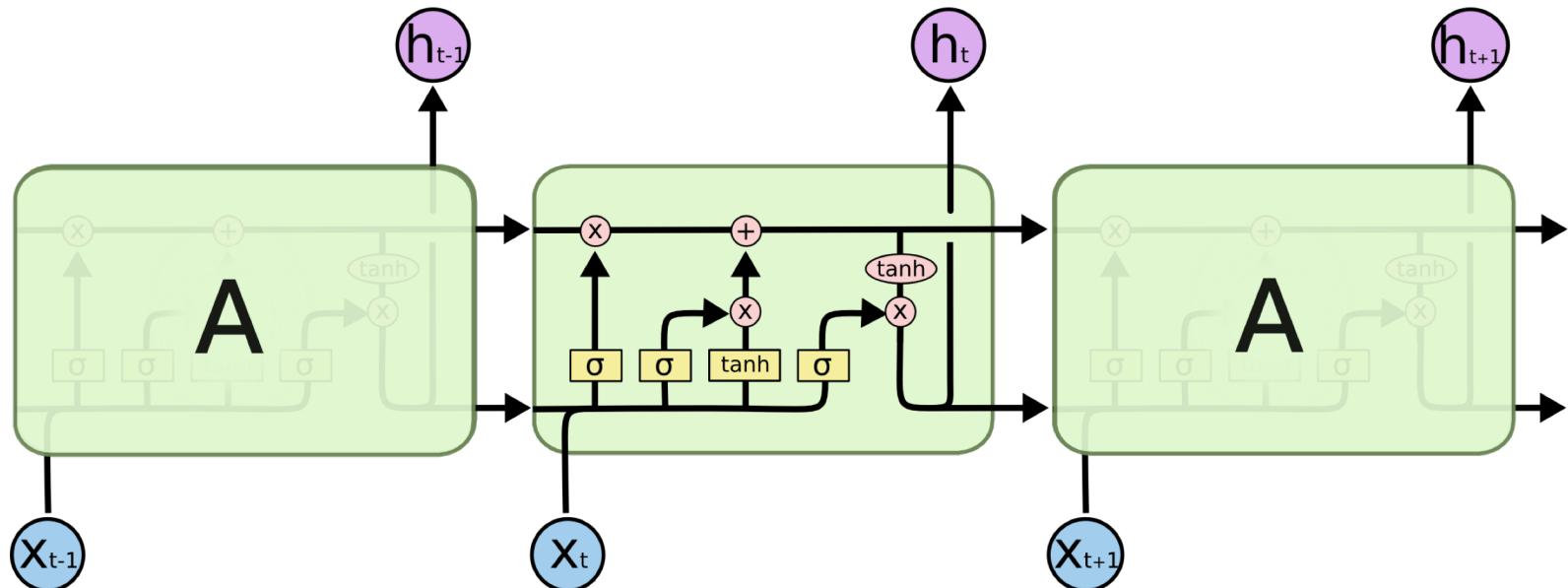


Long Short-Term Memory (LSTM)

- In the leaky units with self-connections, the forgetting rate is constant during the whole sequence.
- The role of leaky units is to accumulate information over a long duration. However, once that information gets used, it might be useful for the neural network to forget the old state.
 - ▶ For example, if a video sequence is composed as subsequences corresponding to different actions, we want a leaky unit to accumulate evidence inside each subsequence, and we need a mechanism to forget the old state by setting it to zero and starting to count from fresh when starting to process the next subsequence
- The forgetting rates are expected to be different at different time steps, depending on their previous hidden states and current input (conditioning the forgetting on the context)
- Parameters controlling the forgetting rates are learned from train data

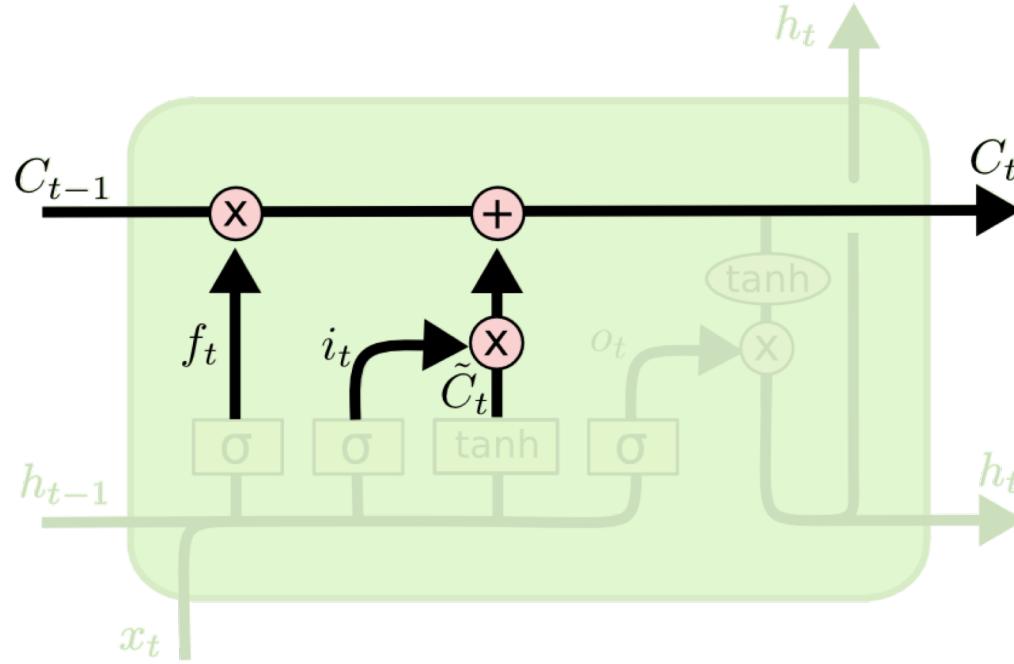
Long Short-Term Memory (LSTM)

- Cell state C, hidden state h, input x, output o



Long Short-Term Memory (LSTM)

- The principle to update the cell state at a new moment

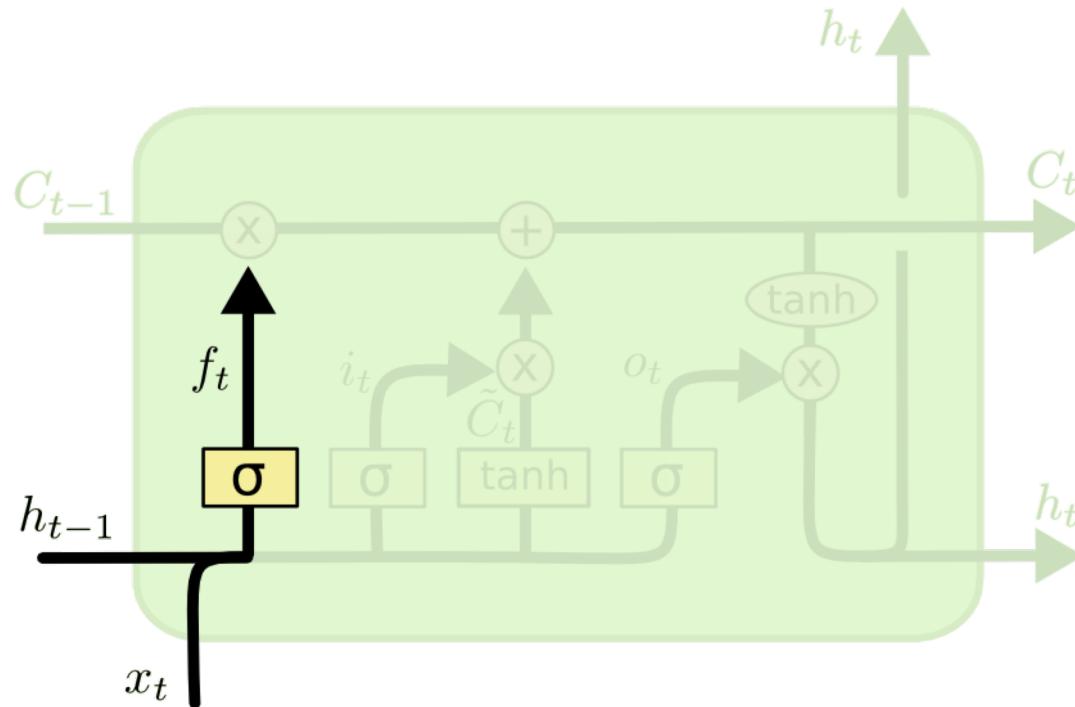


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Updated cell state Forget gate Input gate
Previous cell state "Proposed" cell state

Long Short-Term Memory (LSTM)

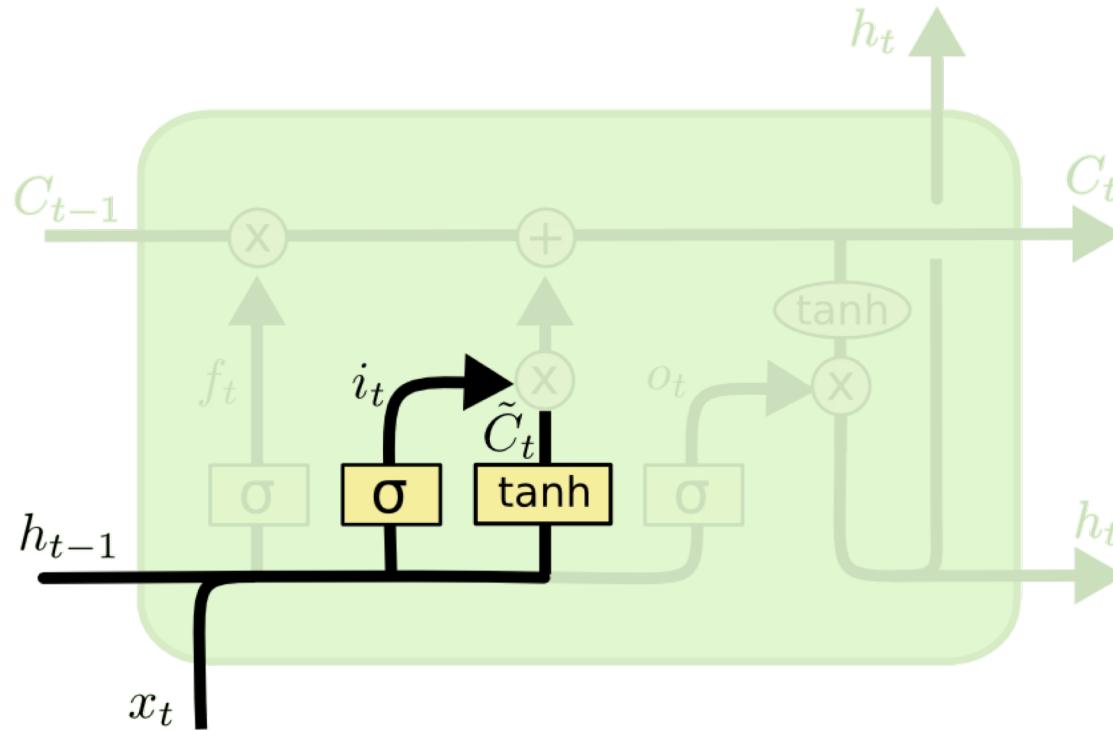
- Forget gate



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

Long Short-Term Memory (LSTM)

- “proposed” cell state

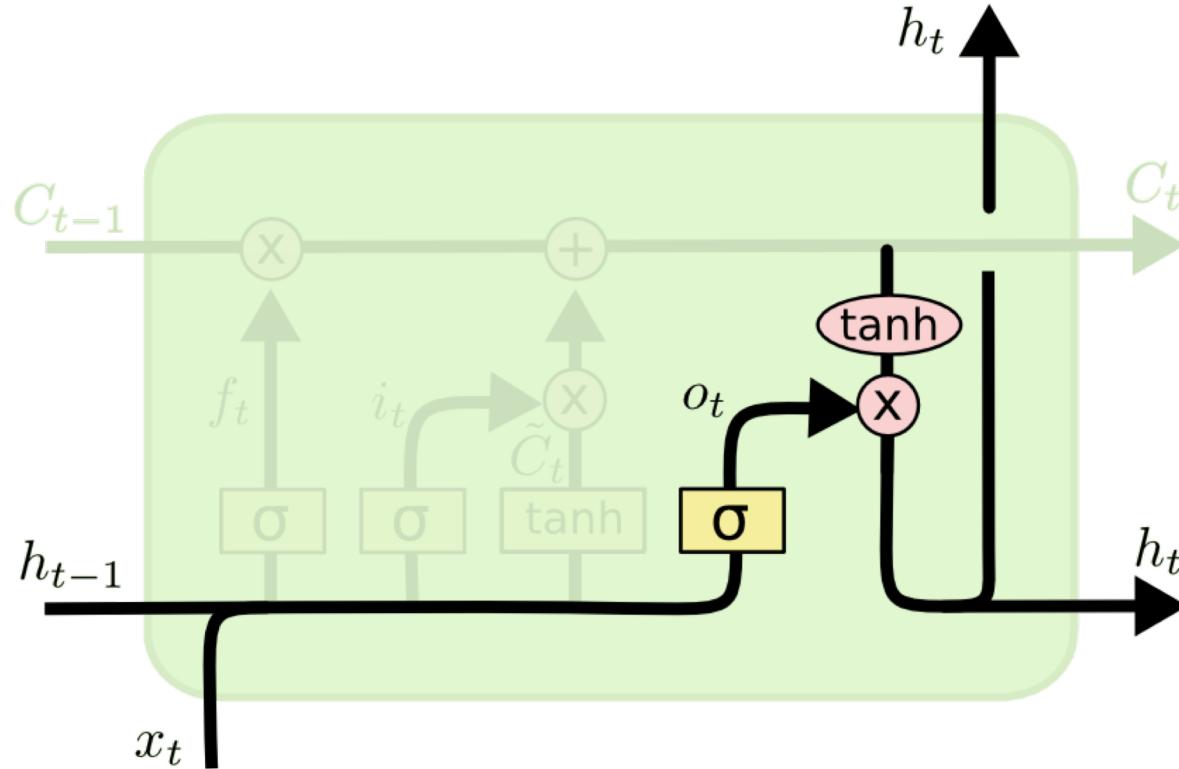


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

Long Short-Term Memory (LSTM)

- Output gate

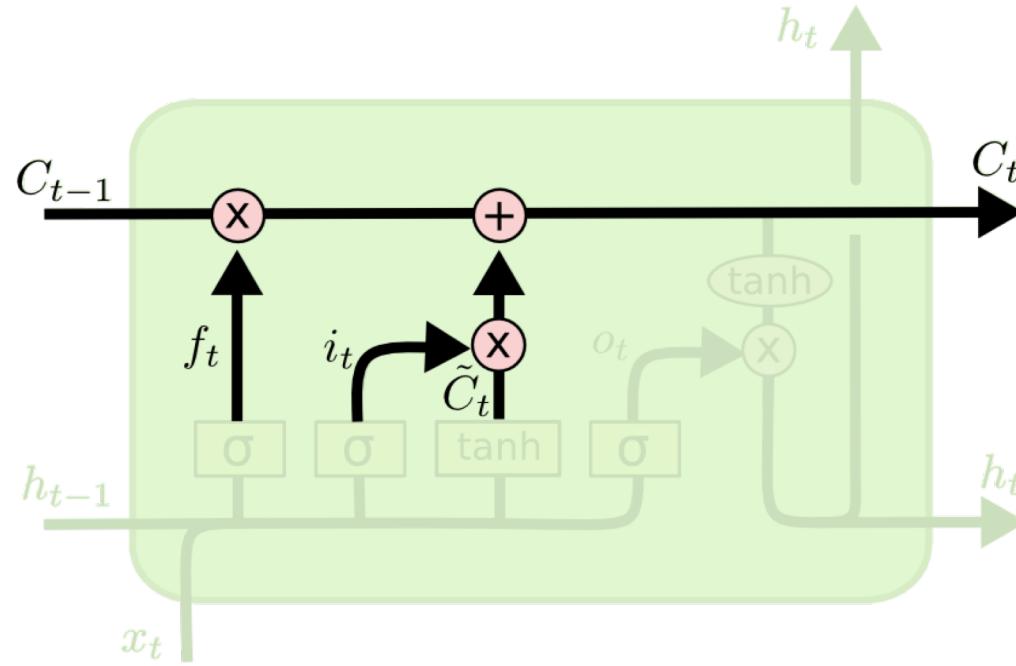


$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

Long Short-Term Memory (LSTM)

- Review the updating formula



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

Labels for the equation:

- Forget gate:** Points to the term f_t .
- Input gate:** Points to the term i_t .
- Updated cell state:** Points to the term C_t .
- Previous cell state:** Points to the term C_{t-1} .
- “Proposed” cell state:** Points to the term \tilde{C}_t .

Long Short-Term Memory (LSTM)

- The core of LSTM is a memory cell \mathbf{c}_t which encodes, at every time step, the knowledge of the inputs that have been observed up to that step.
- The memory cell \mathbf{c}_t has the same inputs (\mathbf{h}_{t-1} and \mathbf{x}_t) and outputs (\mathbf{h}_t) as a normal recurrent network, but has more parameters and a system of gating units that controls the flow of information
- \mathbf{c}_t has a linear self-connection similar to the leaky units, but the self-connection weight is controlled by a forget gate unit \mathbf{f}_t , that sets this weight to a value between 0 and 1 via a sigmoid unit $\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$
- The input gate unit \mathbf{i}_t is computed similarly to the forget gate, but with its own parameters
- The output \mathbf{h}_t of the LSTM cell can also be shut off , via the output gate \mathbf{o}_t ($\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$), which is also a sigmoid unit for gating
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$$

Show and Tell: A Neural Image Caption Generator

Oriol Vinyals
Google

vinyals@google.com

Alexander Toshev
Google

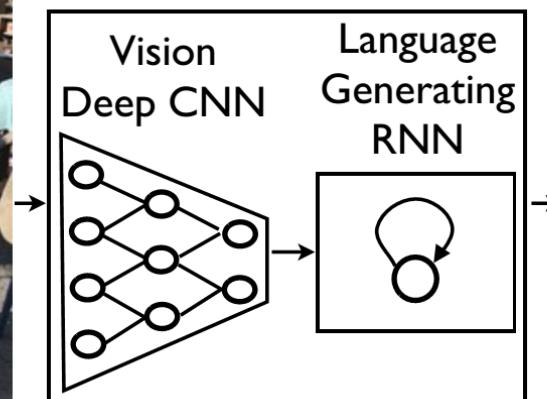
toshev@google.com

Samy Bengio
Google

bengio@google.com

Dumitru Erhan
Google

dumitru@google.com



A group of people shopping at an outdoor market.

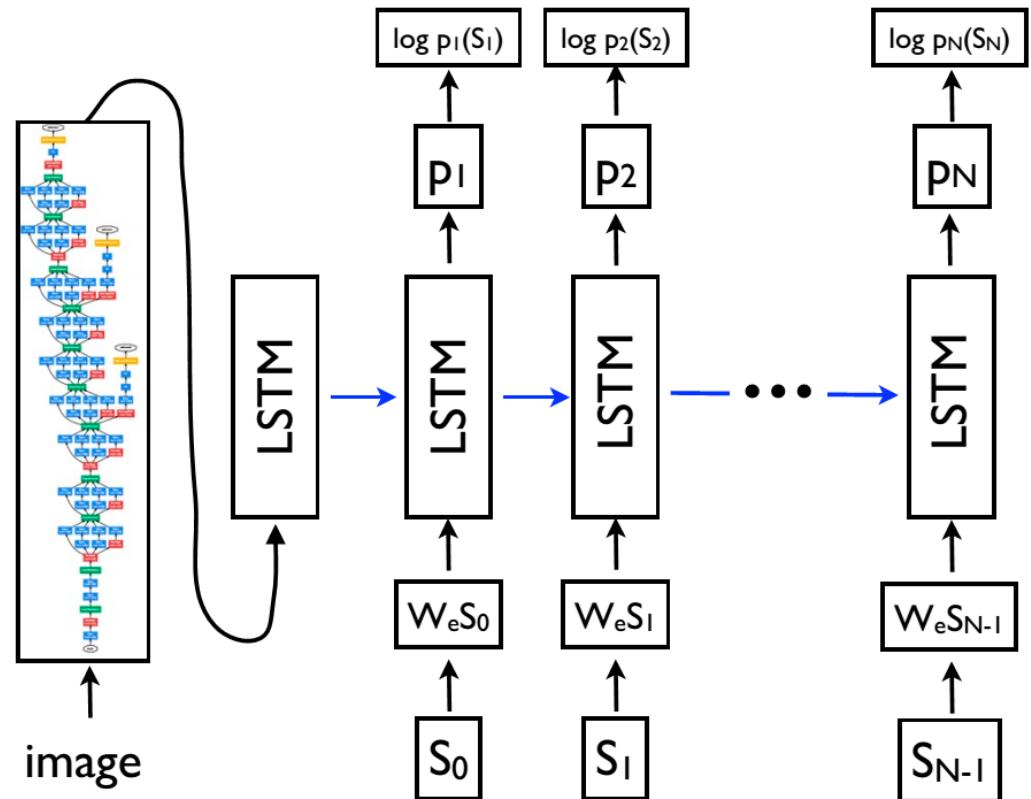
There are many vegetables at the fruit stand.

Image Captioning

$$\theta^* = \arg \max_{\theta} \sum_{(I, S)} \log p(S|I; \theta)$$

$$\log p(S|I) = \sum_{t=0}^N \log p(S_t|I, S_0, \dots, S_{t-1})$$

$$L(I, S) = - \sum_{t=1}^N \log p_t(S_t) .$$



Results

<p>A person riding a motorcycle on a dirt road.</p> 	<p>Two dogs play in the grass.</p> 	<p>A skateboarder does a trick on a ramp.</p> 	<p>A dog is jumping to catch a frisbee.</p> 
<p>A group of young people playing a game of frisbee.</p> 	<p>Two hockey players are fighting over the puck.</p> 	<p>A little girl in a pink hat is blowing bubbles.</p> 	<p>A refrigerator filled with lots of food and drinks.</p> 
<p>A herd of elephants walking across a dry grass field.</p> 	<p>A close up of a cat laying on a couch.</p> 	<p>A red motorcycle parked on the side of the road.</p> 	<p>A yellow school bus parked in a parking lot.</p> 

Describes without errors

Describes with minor errors

Somewhat related to the image

Unrelated to the image

Figure 5. A selection of evaluation results, grouped by human rating.

Poem Generation

tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

↓ train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, ammerenith ol sivh I lalterthend Bleipile shuwyl fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

↓ train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.

↓ train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.

Generating Latex Source: Algebraic Geometry

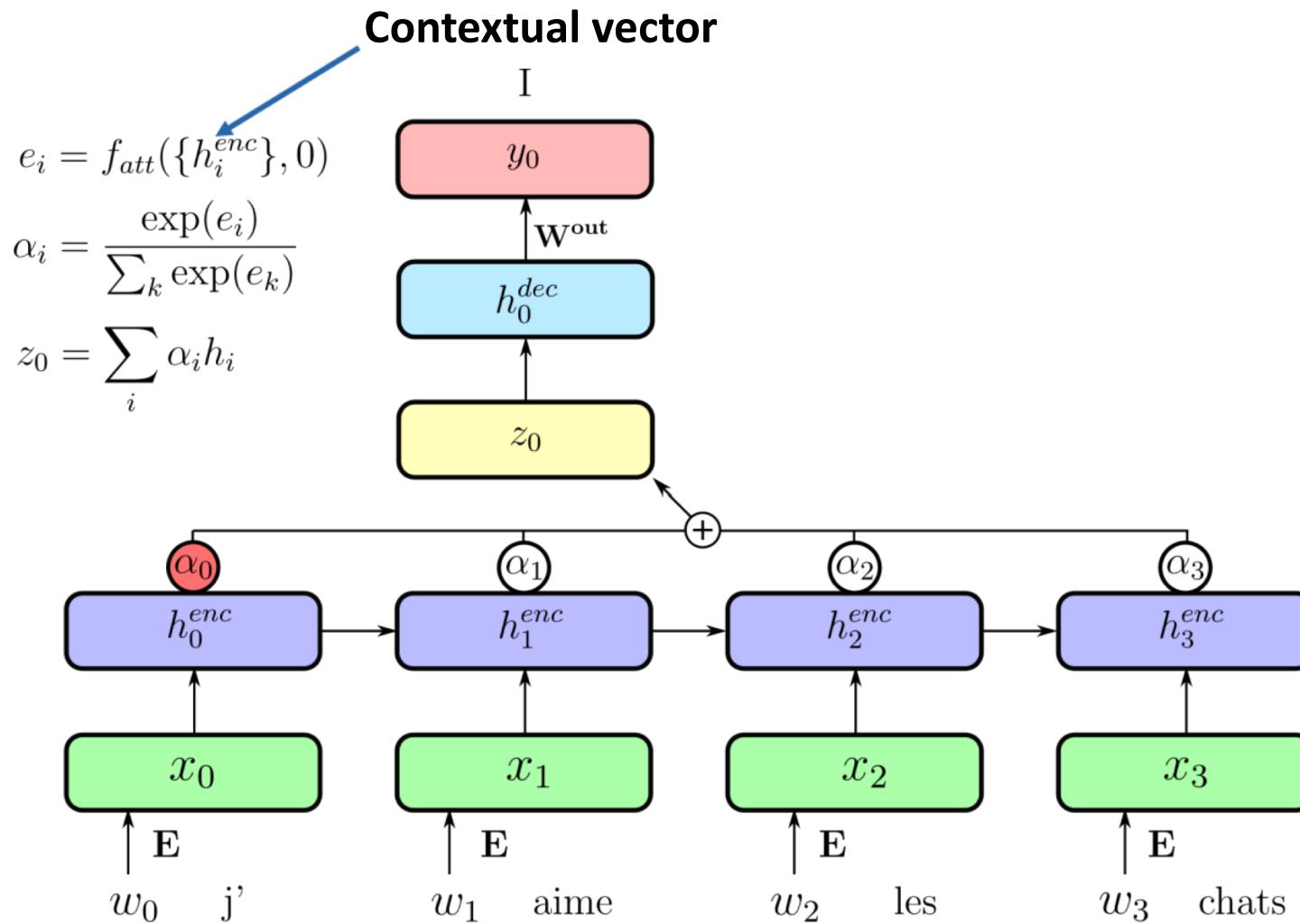
- 16MB Latex source files are used for training LSTM
- Could generate most of compliable Latex source file(Only some of them needs minor correction)

```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on $\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \rightarrow \mathcal{I}$
is an injective and let $\mathfrak{q}$ be an abelian sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$ be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on $\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module. Then
$\mathcal{F}$ is an abelian catenary over $\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{enumerate}
\end{enumerate}
\end{proof}
```

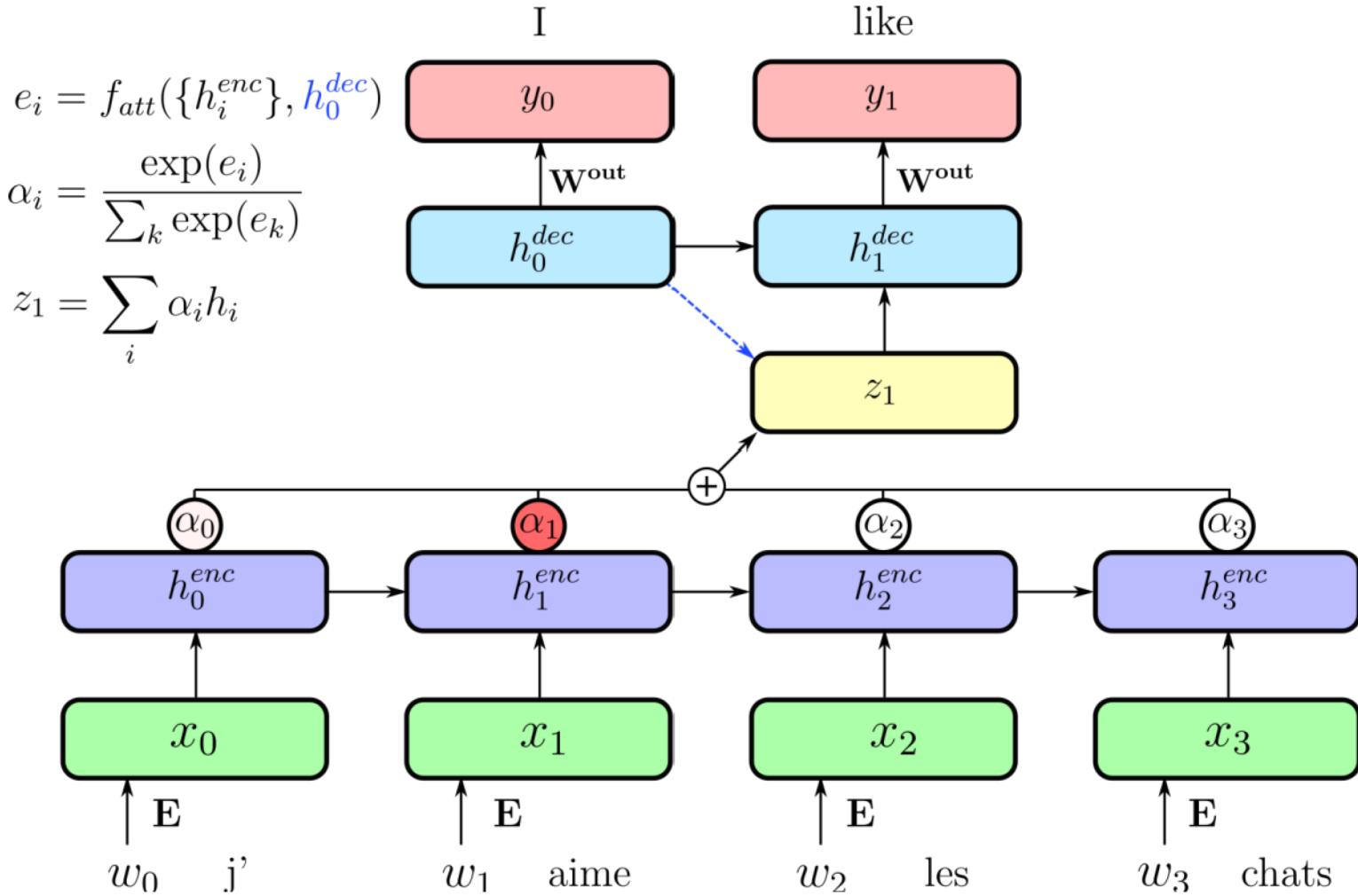
Attention Mechanism

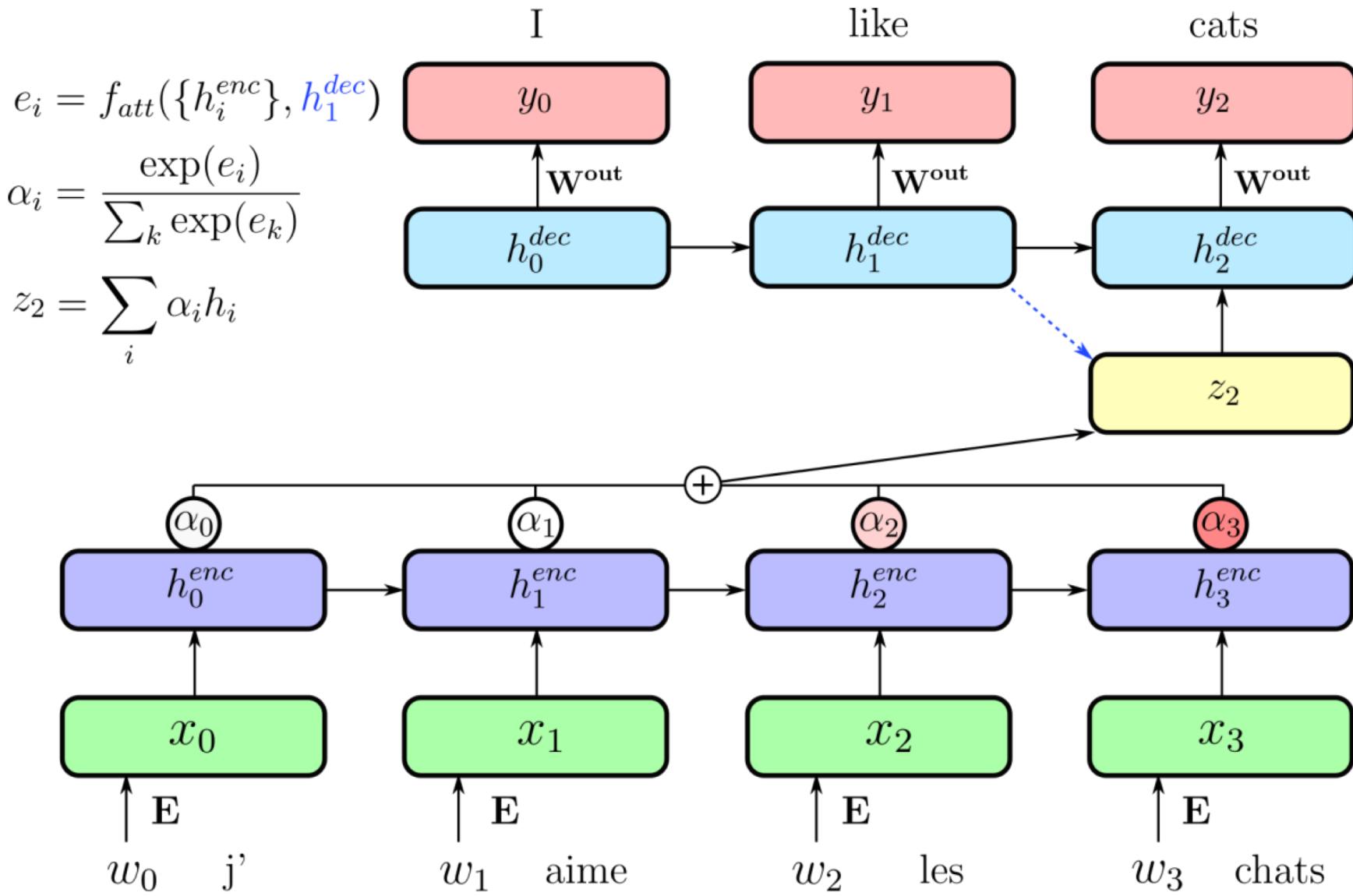
- Seq2Seq model
 - One sequence is encoded as a single vector, which might lose some info.
“I like cats but I don’t like dogs”
- Solution:
 - Use all the latent encoding h_i
 - Attention Mechanism: decoding each word while focusing some related latent encodings

Seq2Seq model with attention mechanism



*Neural machine translation by jointly learning to align and translate,
D Bahdanau, K Cho, Y Bengio 2014*





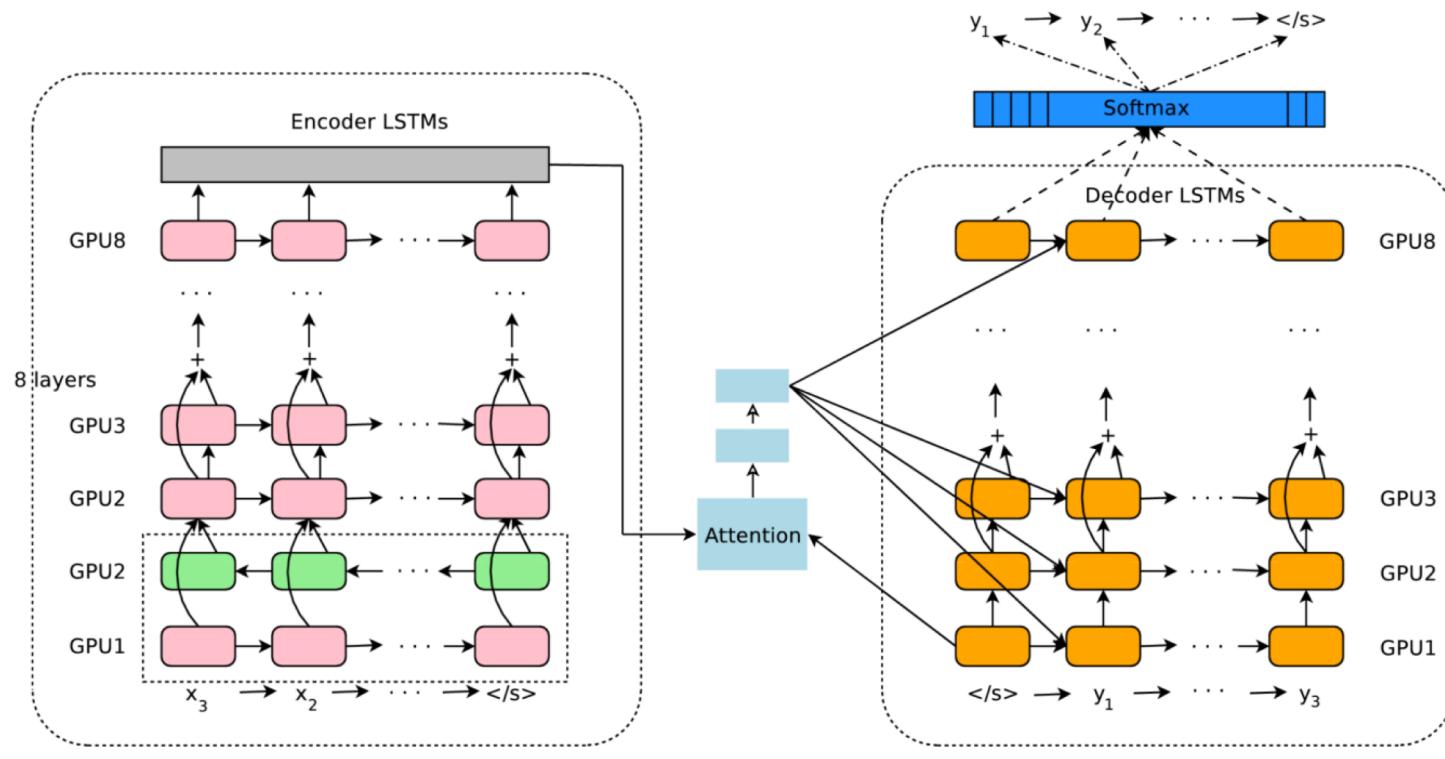
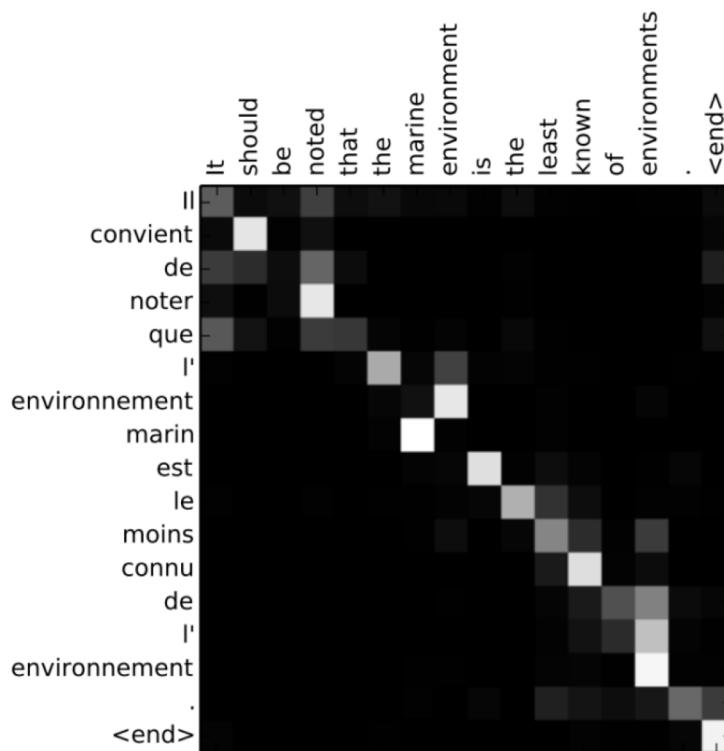
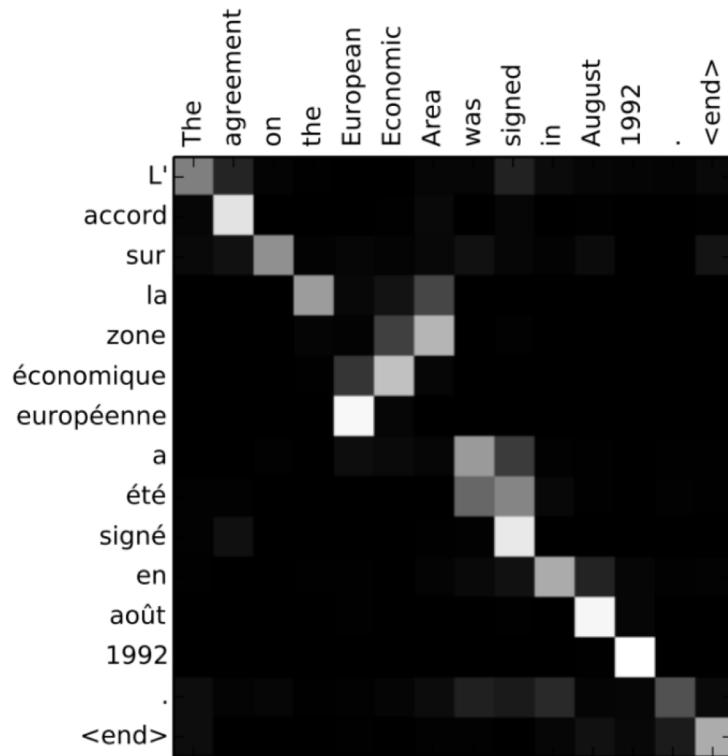


Figure 1: The model architecture of GNMT, Google’s Neural Machine Translation system. On the left is the encoder network, on the right is the decoder network, in the middle is the attention module. The

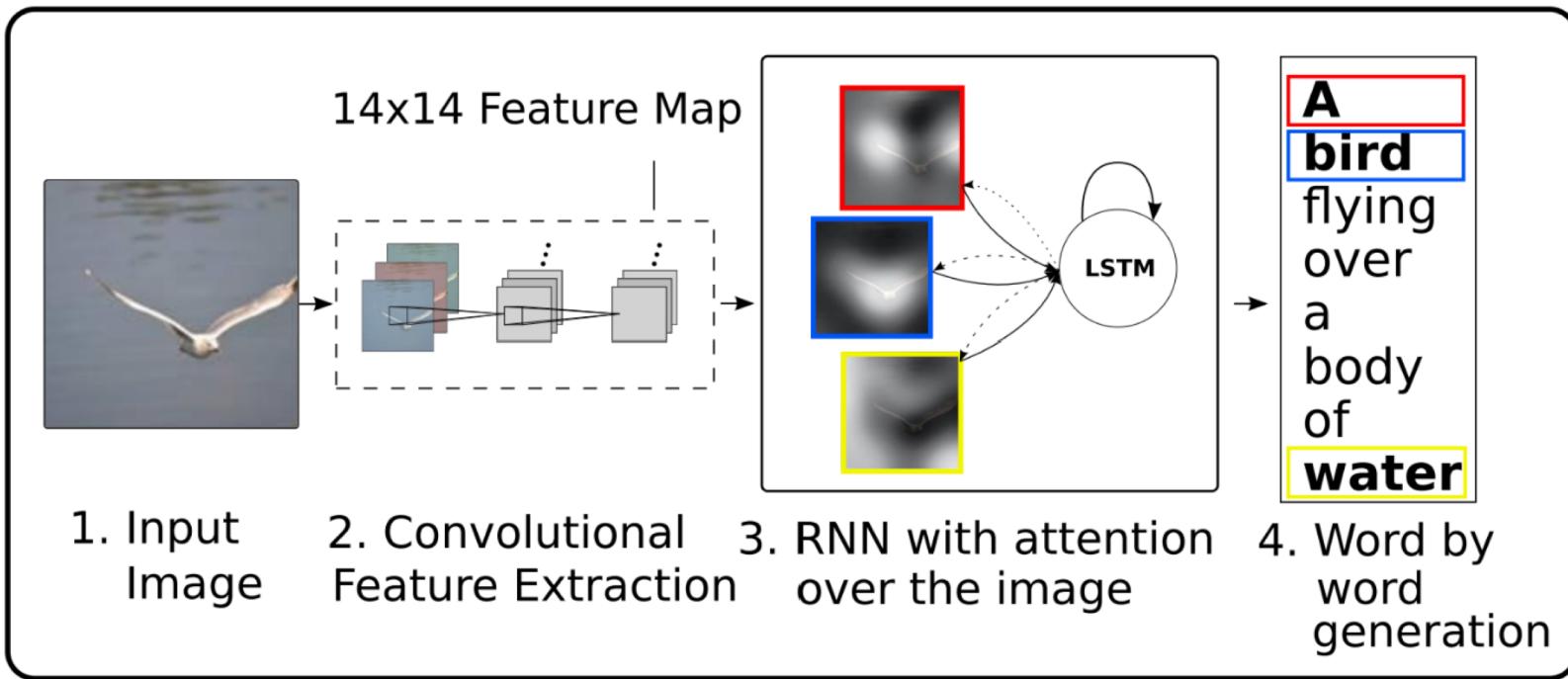
Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation"

Visualization of attention mech.

- Word alignment



Attention Mech. for Image Captioning



Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention" , ICML 2015