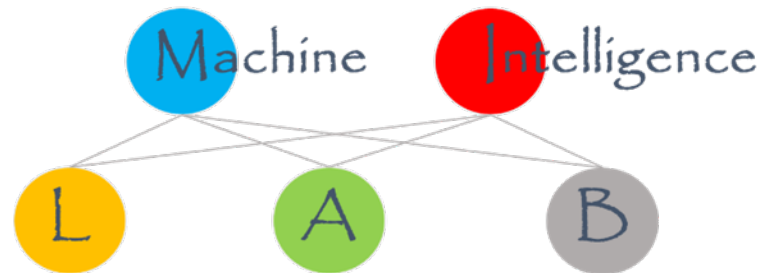




# Recurrent Networks and Transformers

Mu Yadong

Wangxuan Institute of Computer Technology  
Peking University



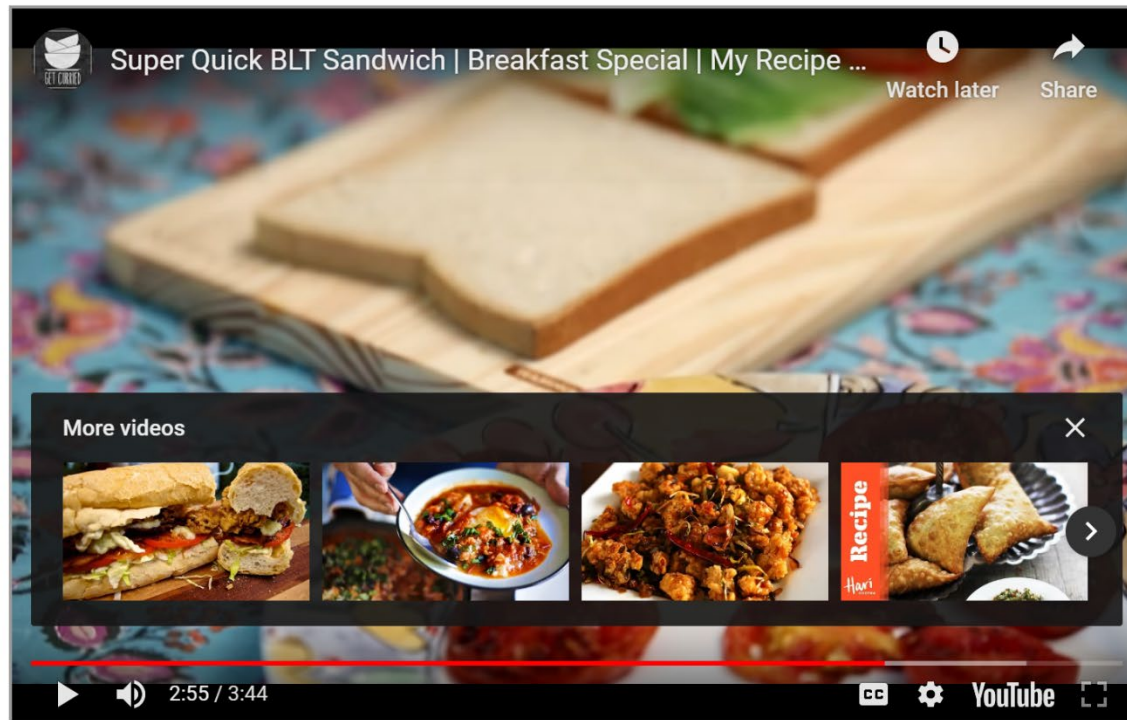
**Several slides are adapted from related courses or tutorials.  
Internal use only. Please do not distribute the slides.**

# Outline

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

# From Static Data to Sequential Data

- Videos as a sequence of frames / shots



# From Static Data to Sequential Data

- Stock price

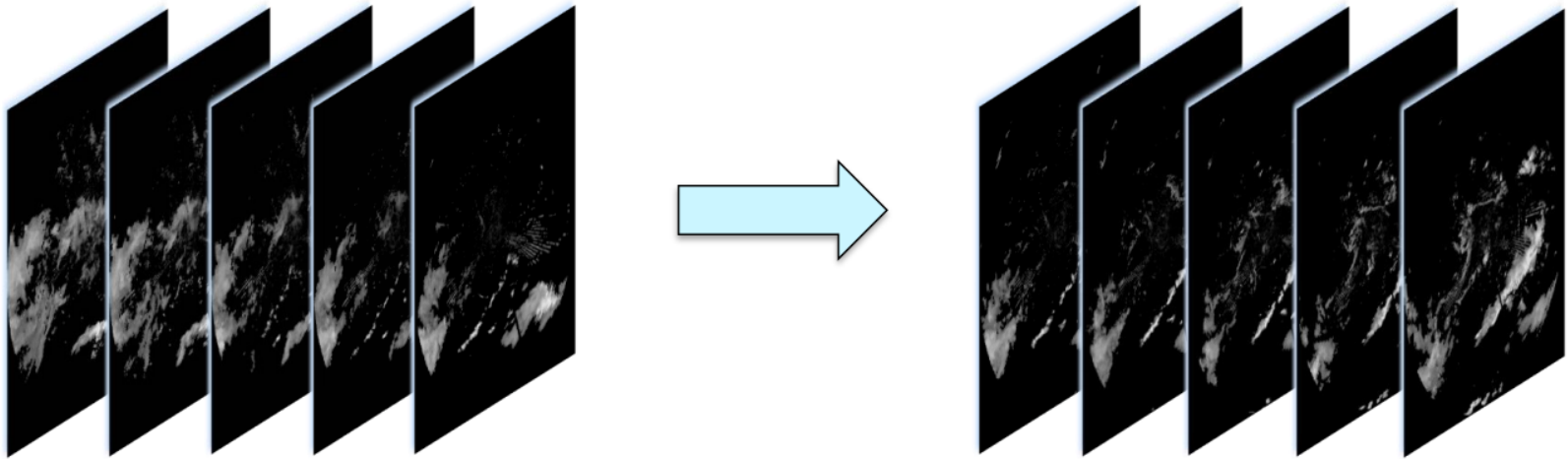


**Stock price of DL/BD related companies**

Volume delayed by 15 mins.  
Prices are not from all markets.  
Sources include SIX.

# From Static Data to Sequential Data

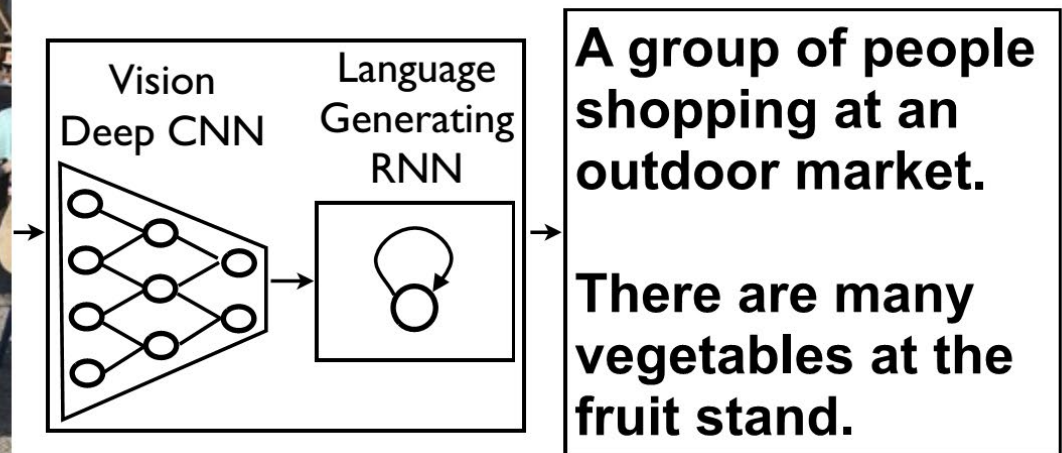
- Predict rainfall
  - **Input sequence:** observed radar maps up to current time step
  - **Output sequence:** predicted radar maps for future time steps



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

# From Static Data to Sequential Data

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



# 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

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

**recurrent** (definition of  $\mathbf{s}$  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

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

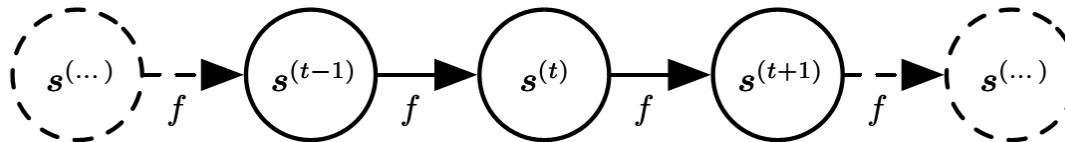


Figure 10.1

**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

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

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

External signal

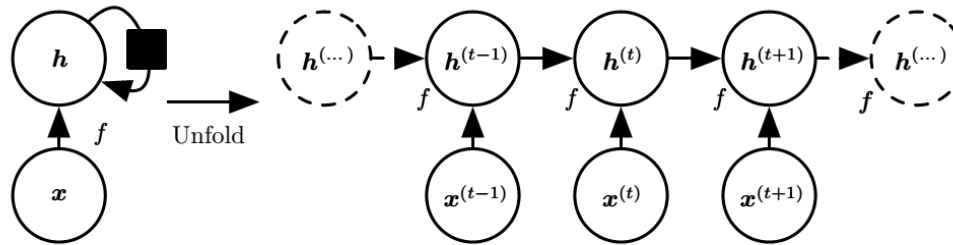


Figure 10.2

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

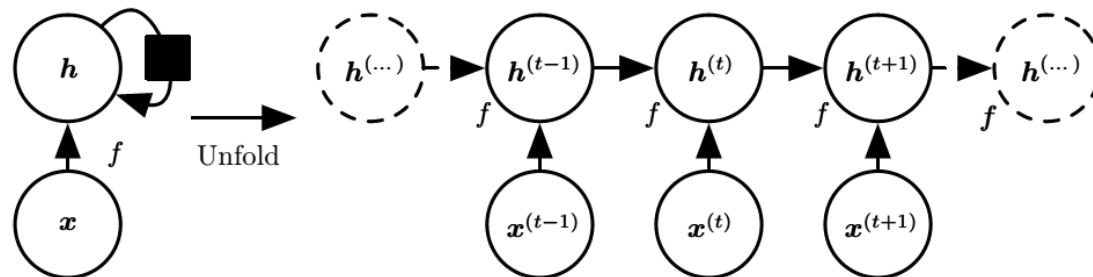


Figure 10.2

# Remarks on Unfolding Computational Graphs

- **Sharing parameters for any sequence length allows more better generalization properties.** 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.

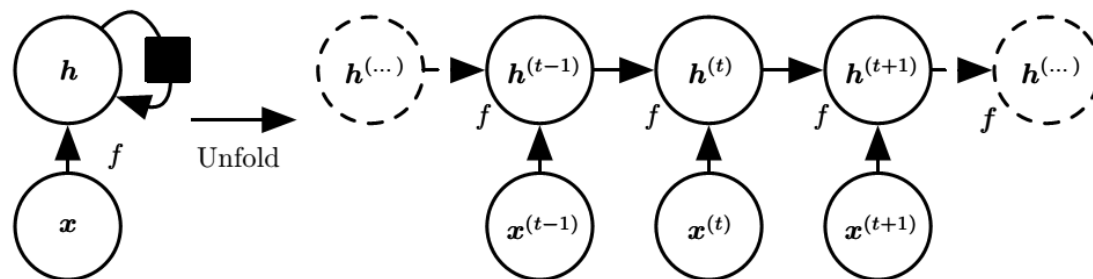


Figure 10.2

# 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  $O(T)$

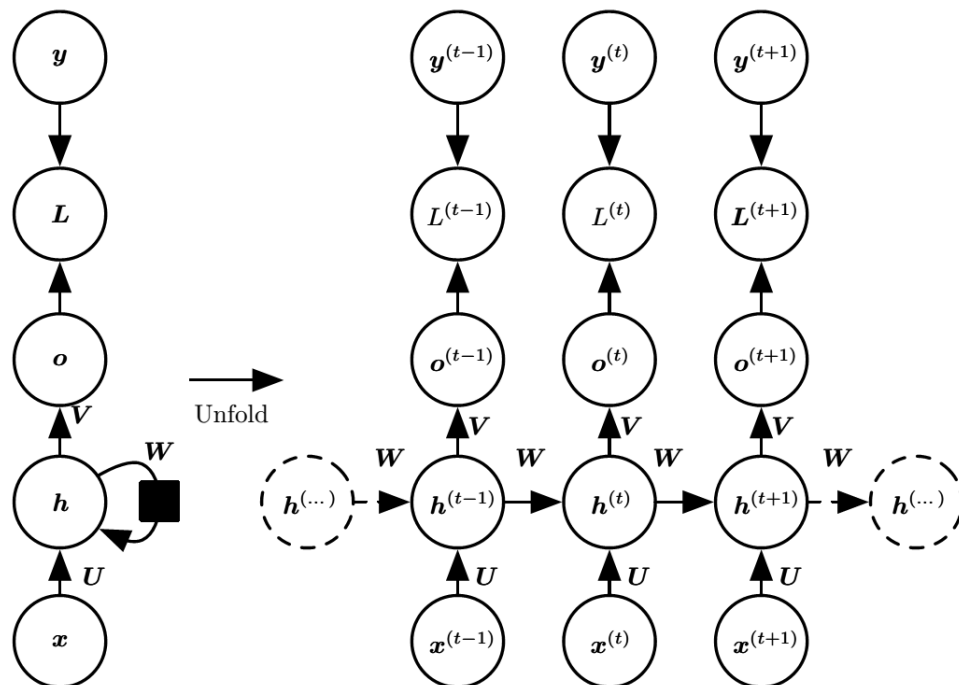
$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

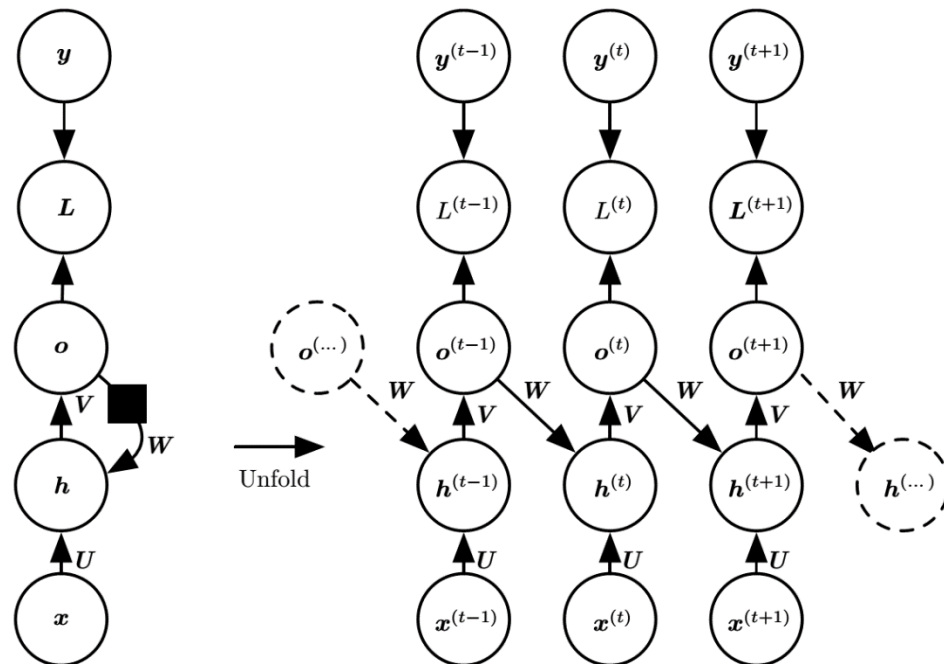
$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

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



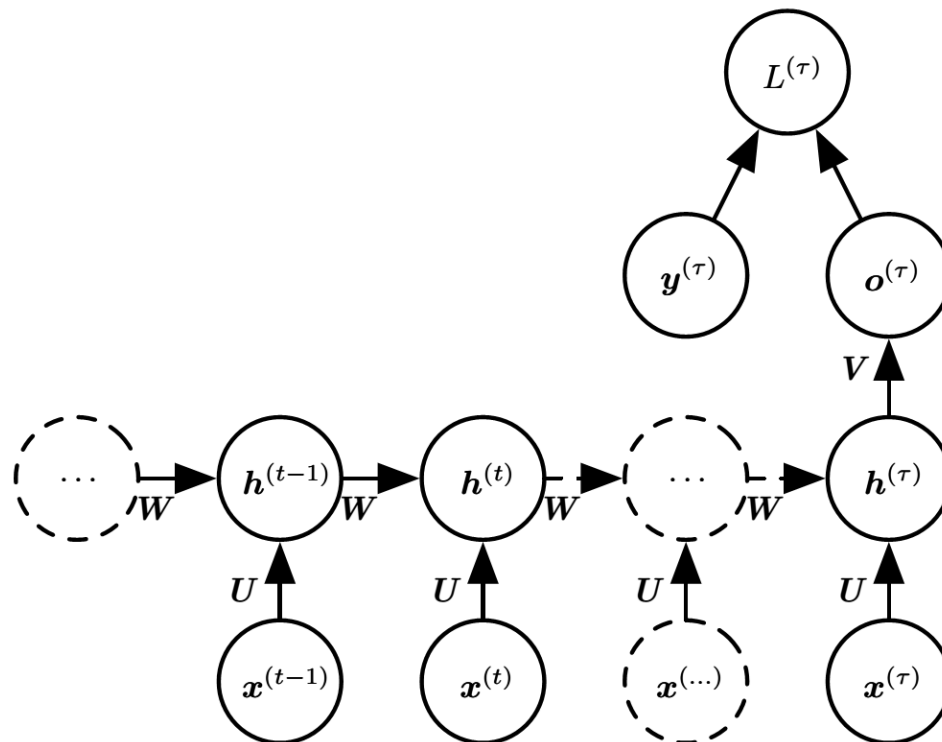
# Recurrent Through Only the Output

- 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.



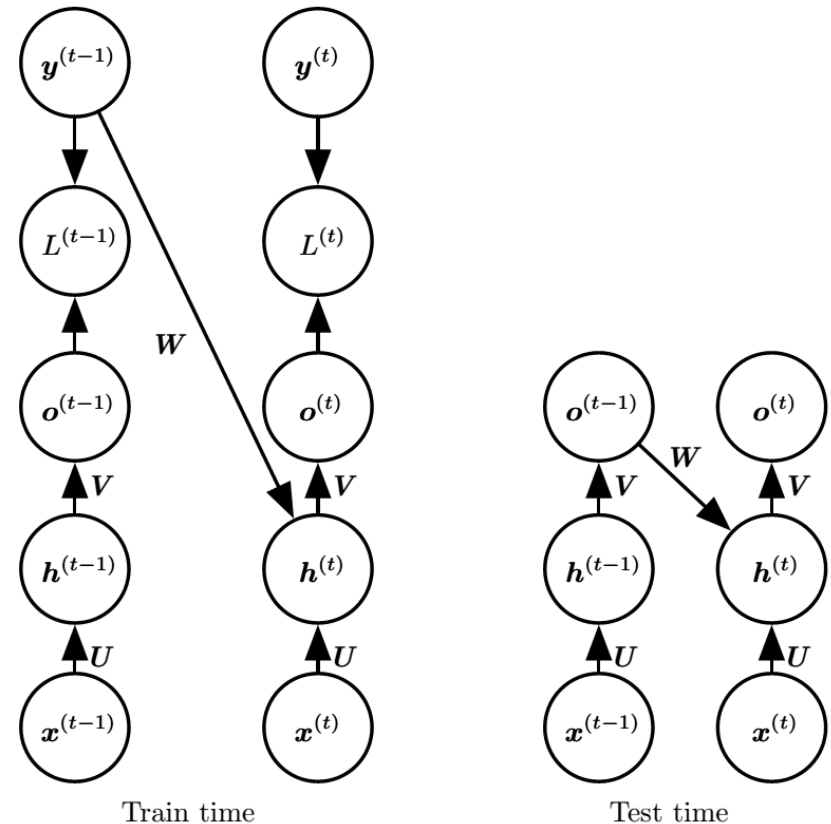
# Sequence Input, Single Output

- 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.”

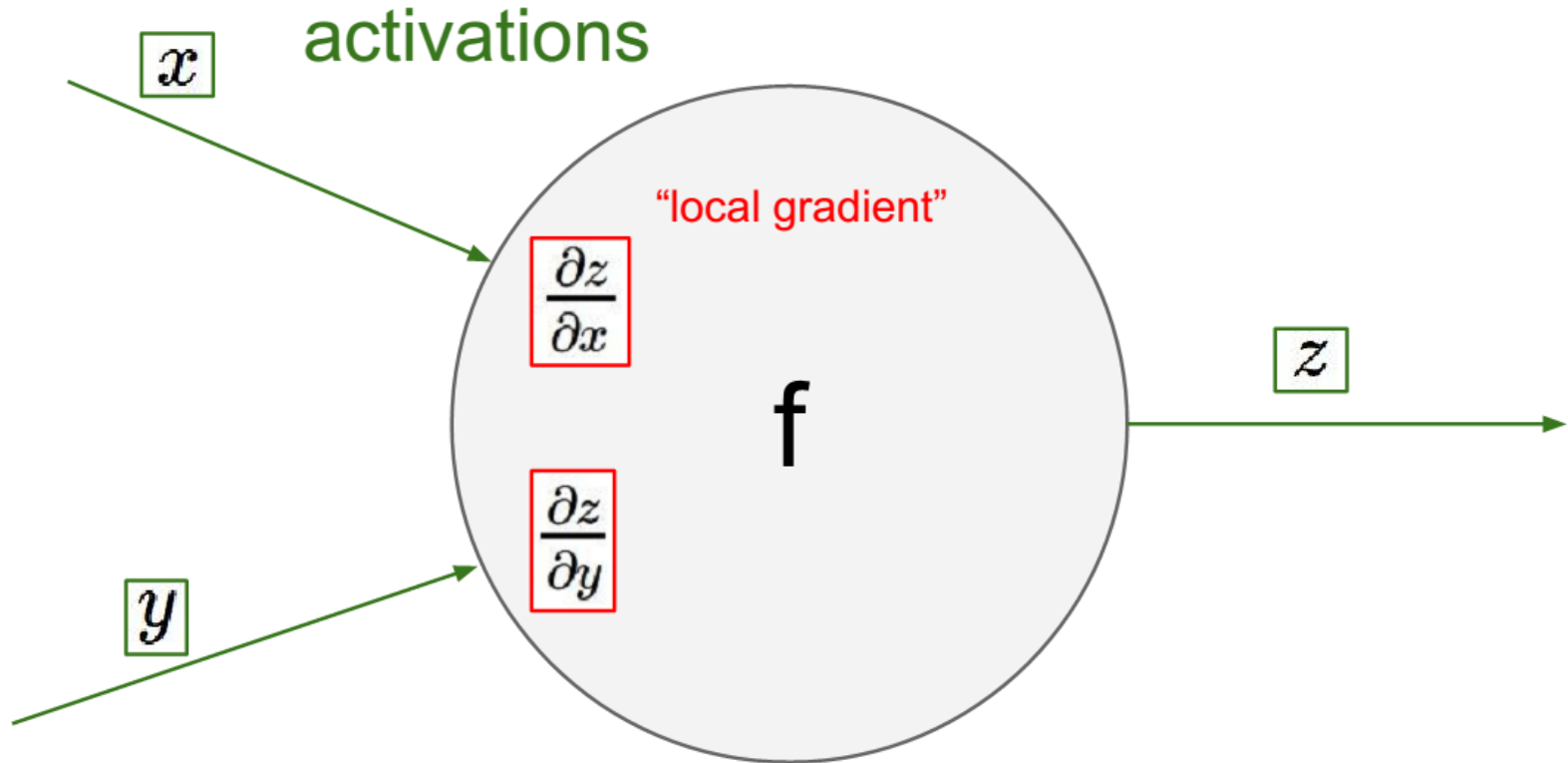


Optimization by a maximum likelihood criterion

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

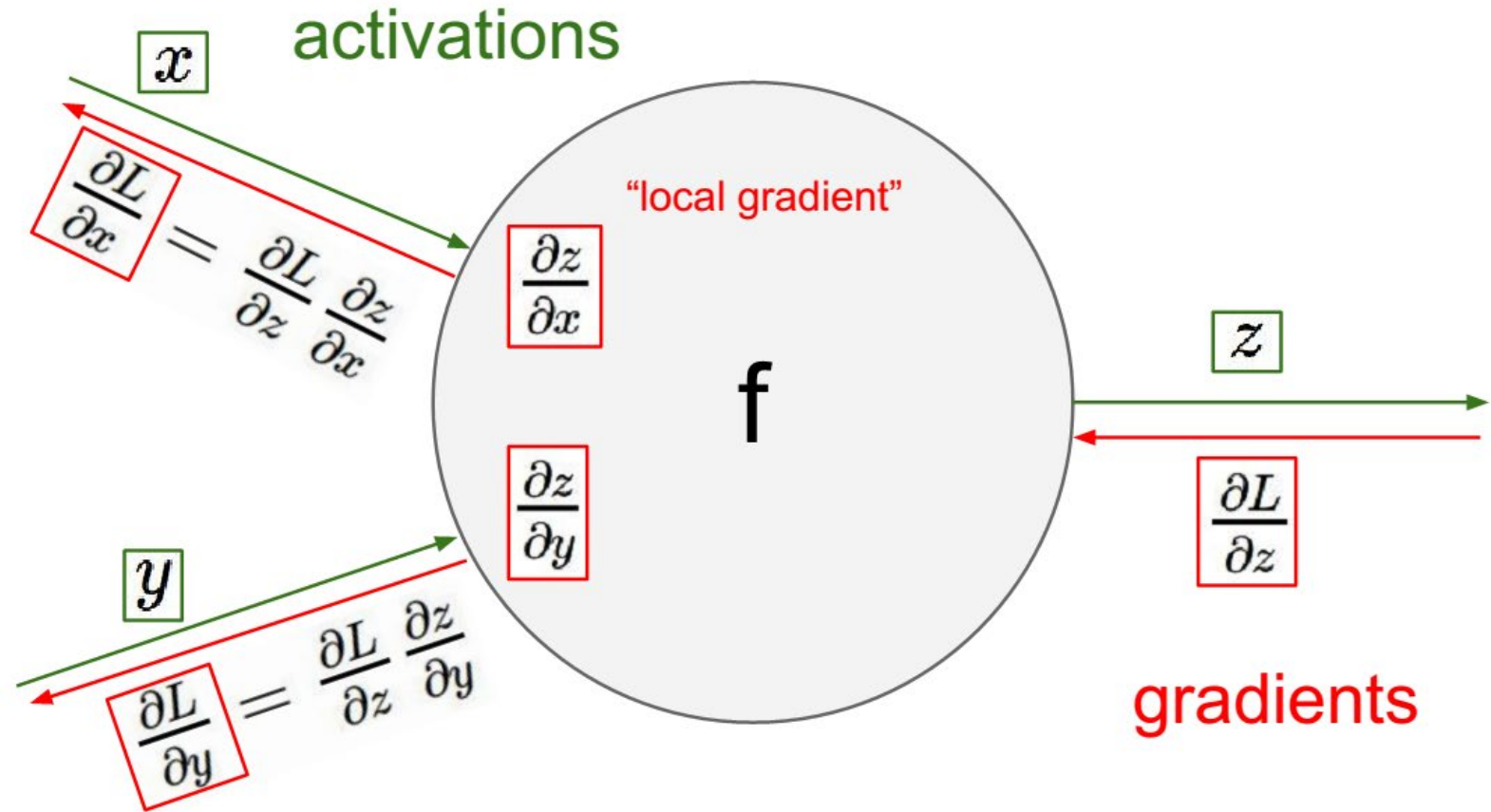
# Review Back-Propagation on Deep Networks

- Anatomy of a neuron



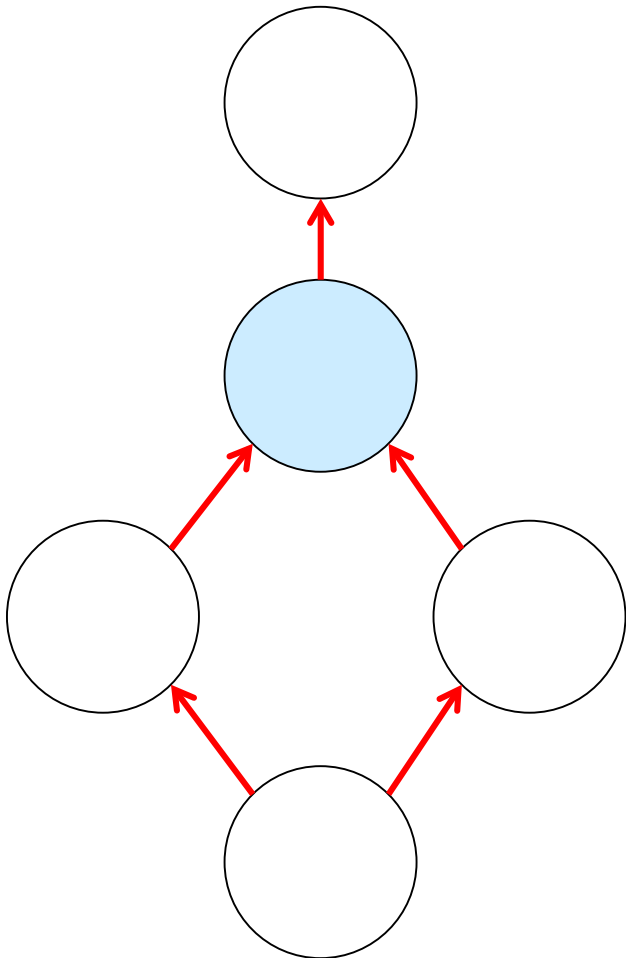
# Review Back-Propagation on Deep Networks

- Anatomy of a neuron

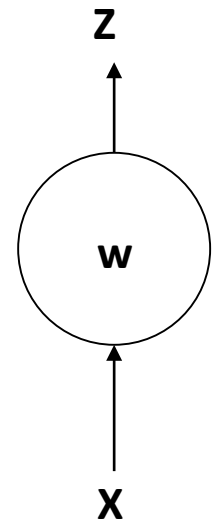
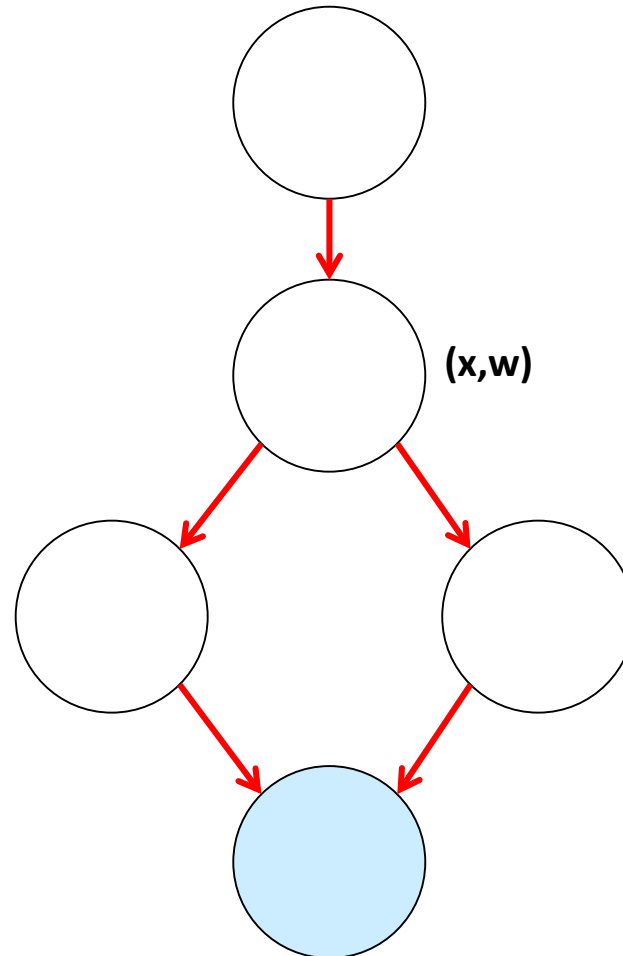


# Review Back-Propagation on Deep Networks

**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)

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

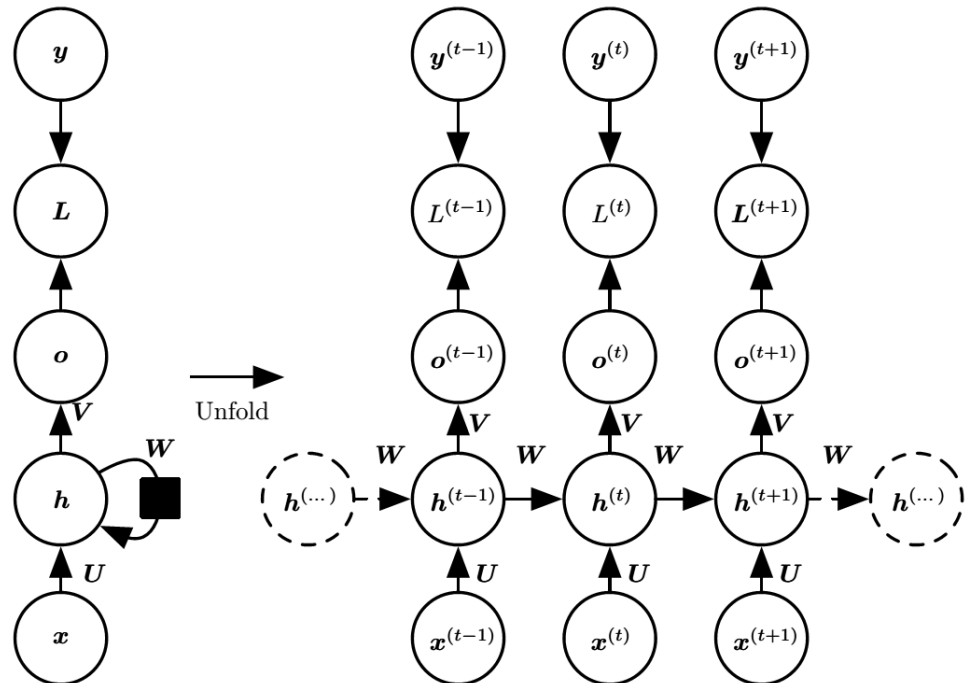
$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

$$L(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\})$$

$$= \sum_t L^{(t)}$$

$$= - \sum_t \log p_{\text{model}}(\mathbf{y}^{(t)} | \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\})$$



# Back-Propagation (BP) on RNN

$$\frac{\partial L}{\partial L^{(t)}} = 1. \quad (\nabla_{\mathbf{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \quad \nabla_{\mathbf{V}} L = \sum_t \sum_i \left( \frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\mathbf{V}} o_i^{(t)}$$

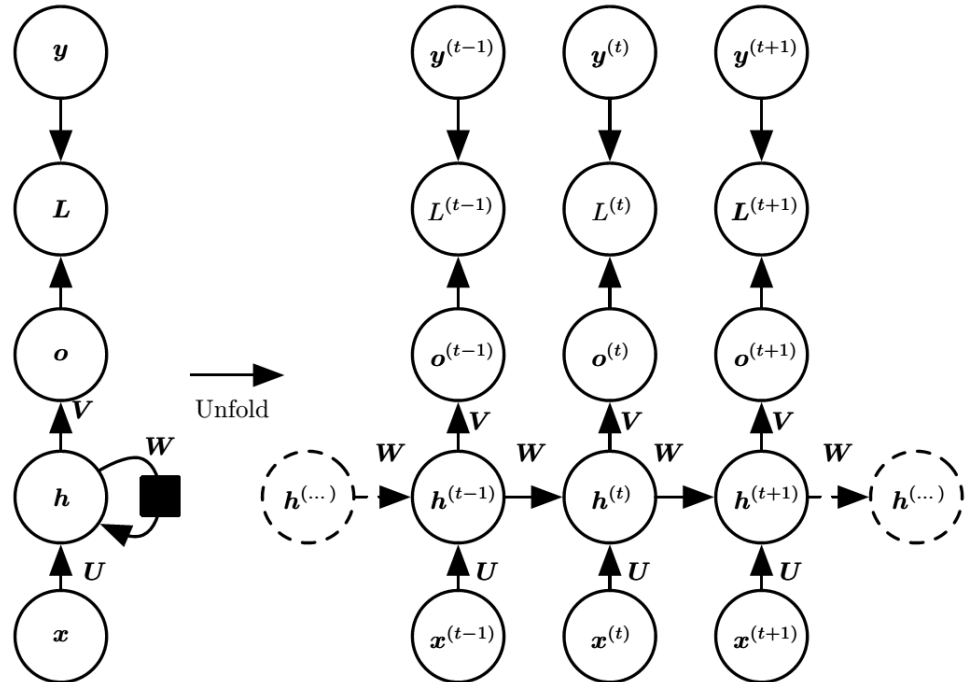
$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

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



# Back-Propagation (BP) on RNN

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

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

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

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

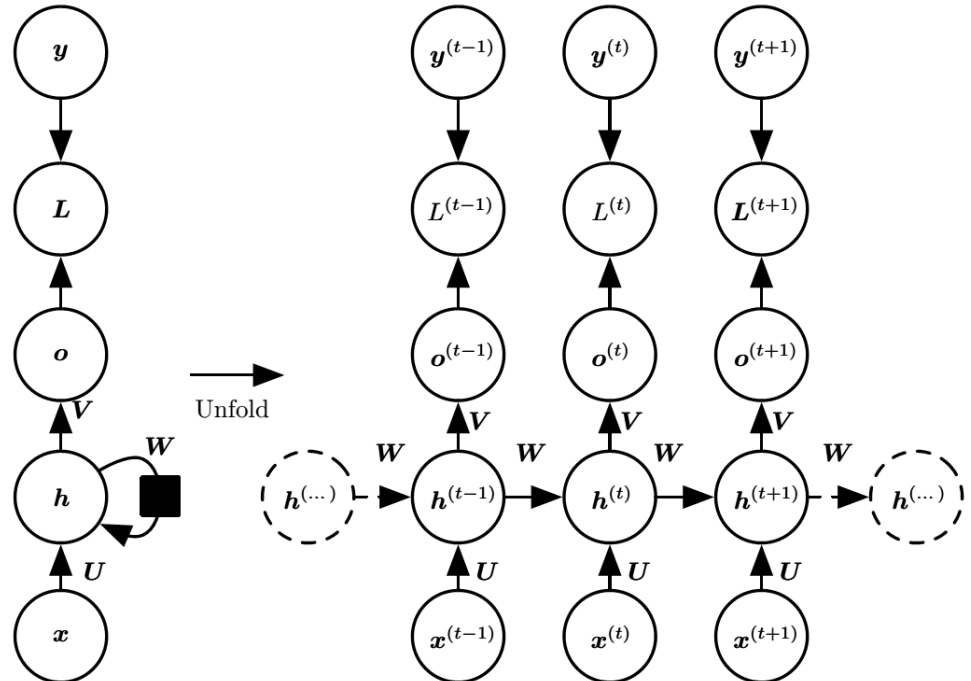
$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

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



# Back-Propagation (BP) on RNN

$$\frac{\partial L}{\partial L^{(t)}} = 1. \quad (\nabla_{\mathbf{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \quad \nabla_{\mathbf{V}} L = \sum_t \sum_i \left( \frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\mathbf{V}} o_i^{(t)}$$

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

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

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

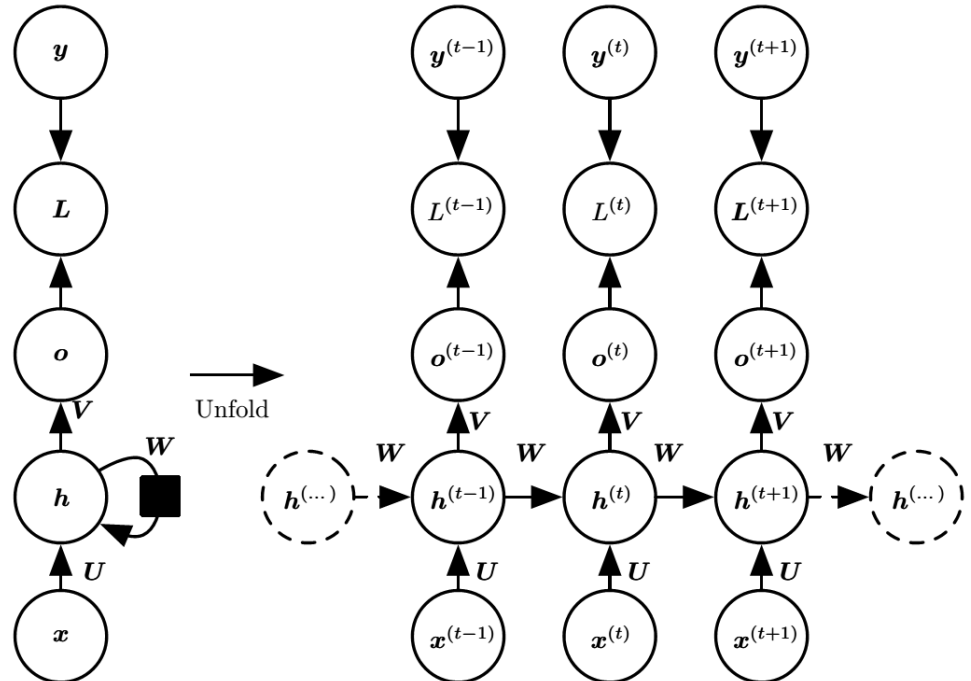
$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

$$L(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\})$$

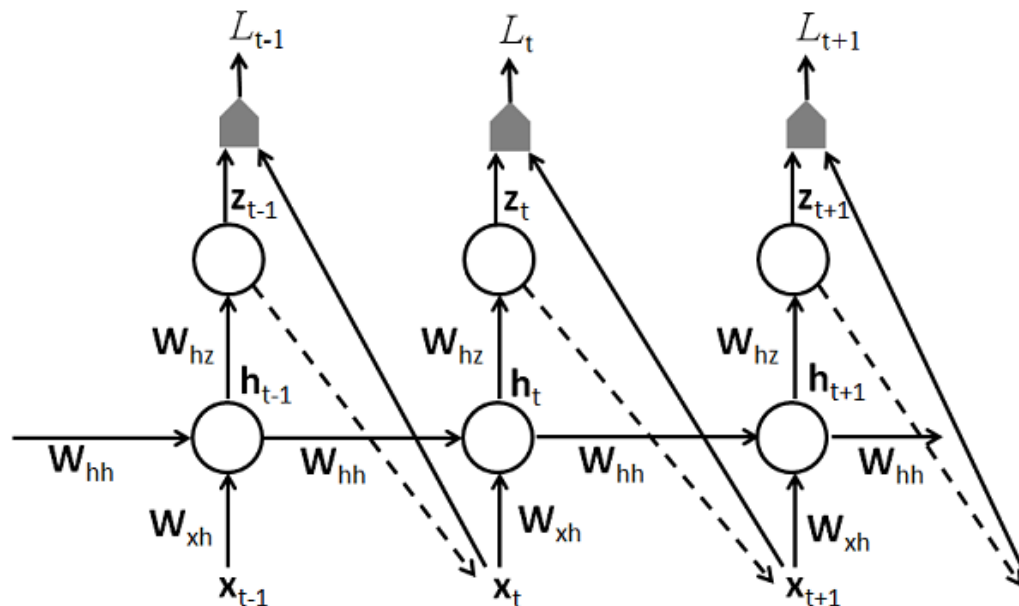
$$= \sum_t L^{(t)}$$

$$= - \sum_t \log p_{\text{model}}(\mathbf{y}^{(t)} | \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\})$$



# 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  $\mathbf{x}_1, \dots, \mathbf{x}_t$



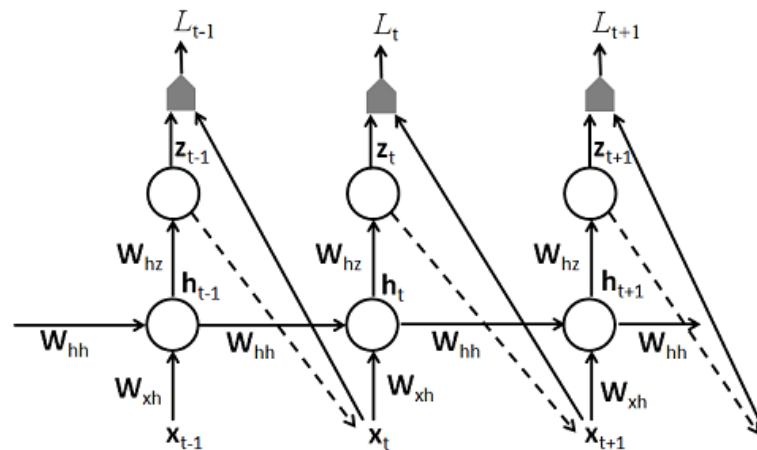
# Generative RNN modeling $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$

- 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

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

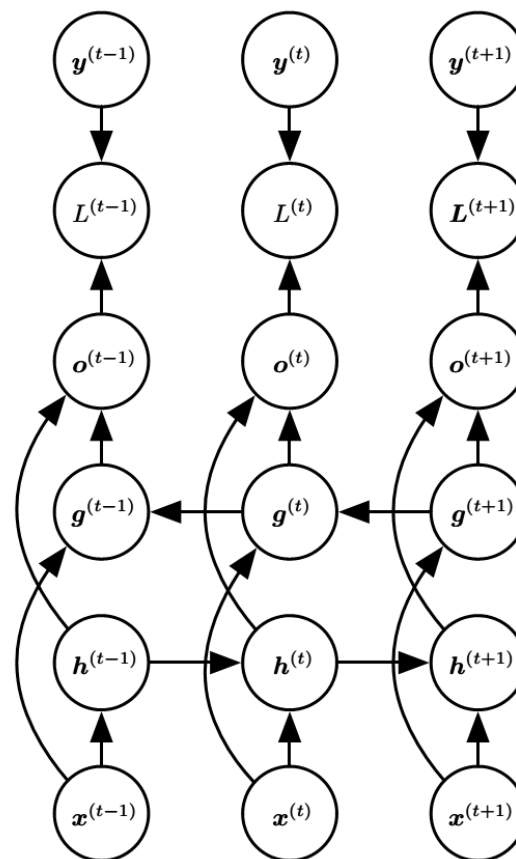
# Bidirectional RNN

- 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:

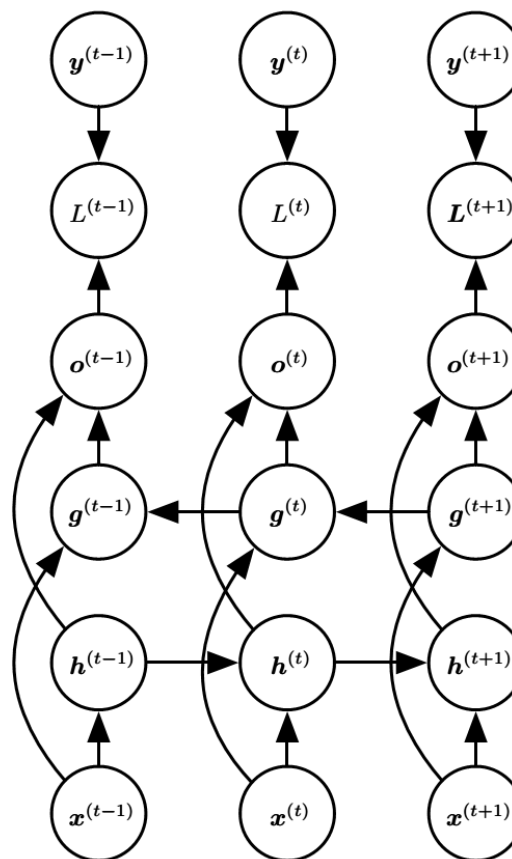
if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them.

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



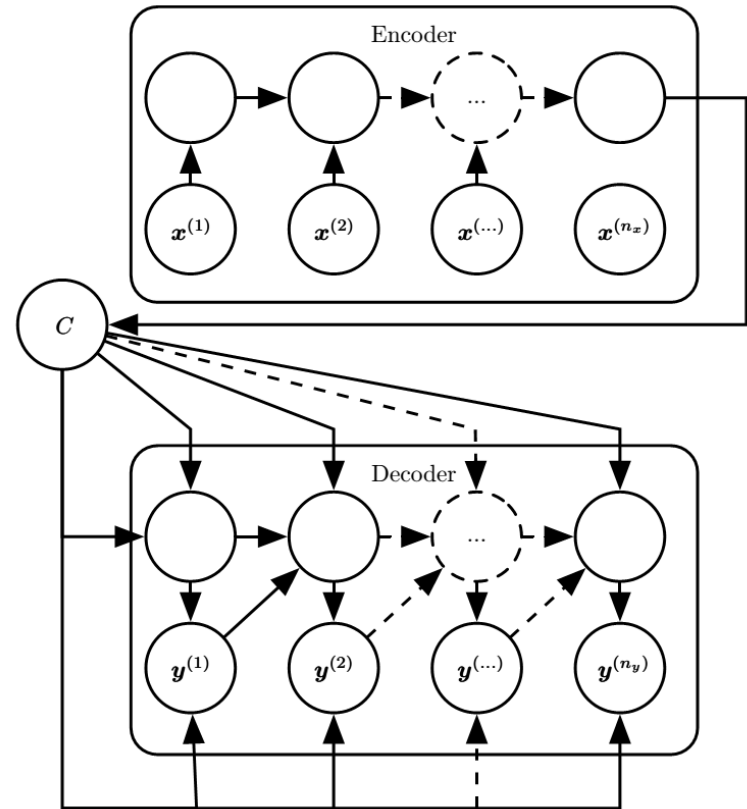
The idea can be extended to 2D input with four RNN going in four directions

# 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

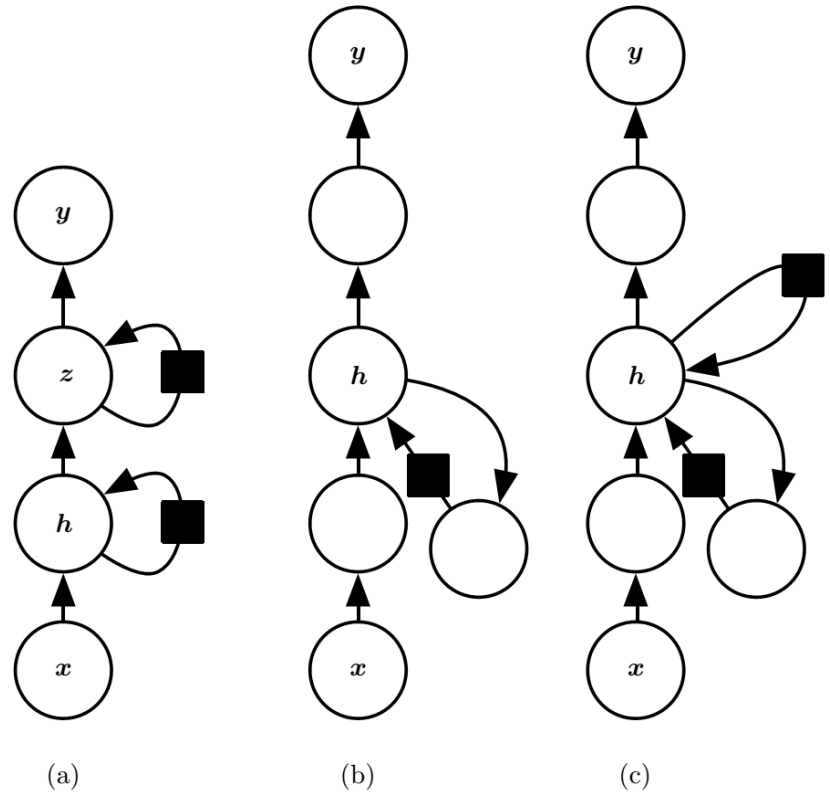
An **encoder** or reader or input RNN processes the input sequence. The encoder emits the context  $C$ , usually as a simple function of its final hidden state.

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.



# 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

$$\mathbf{h}^{(t)} = \mathbf{W}^\top \mathbf{h}^{(t-1)} \quad \text{No input, no nonlinear activation}$$

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

$$\mathbf{W} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top \quad \text{Eigen-decomposition}$$

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

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

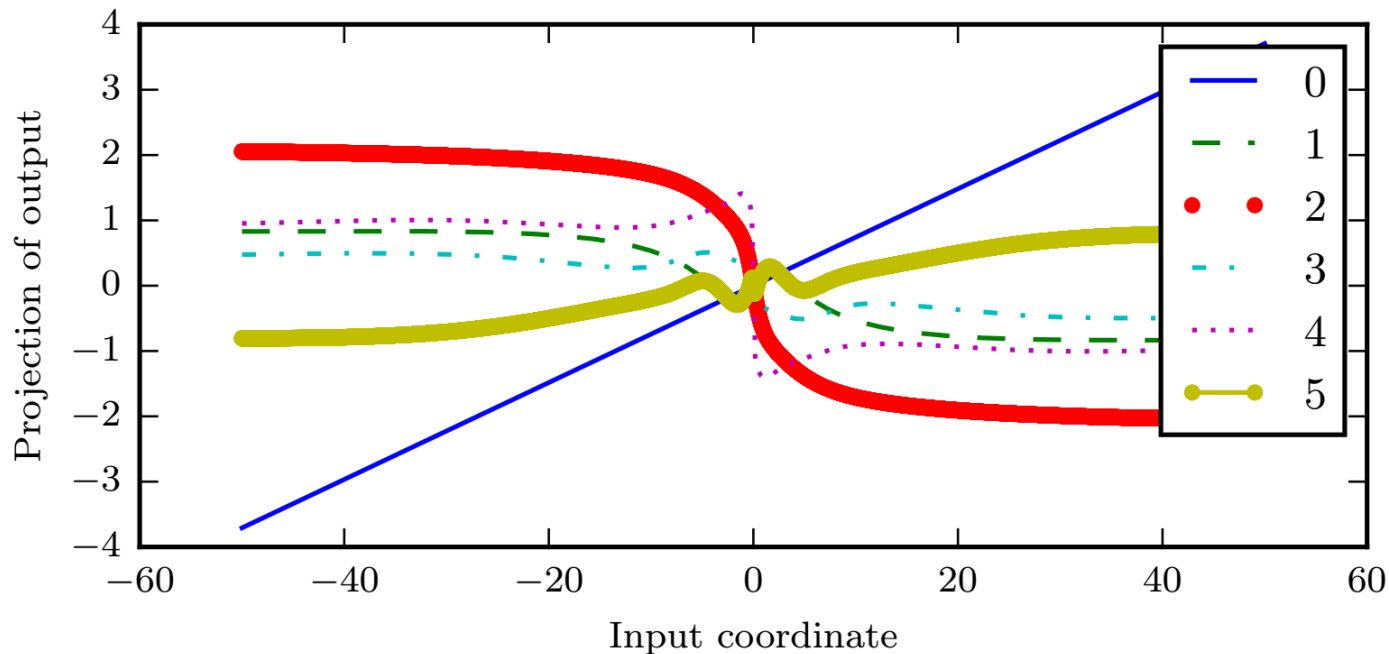


Figure 10.15: Repeated function composition. When composing many nonlinear functions (like the linear-tanh layer shown here), the result is highly nonlinear, typically with most of the values associated with a tiny derivative, some values with a large derivative, and many alternations between increasing 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.

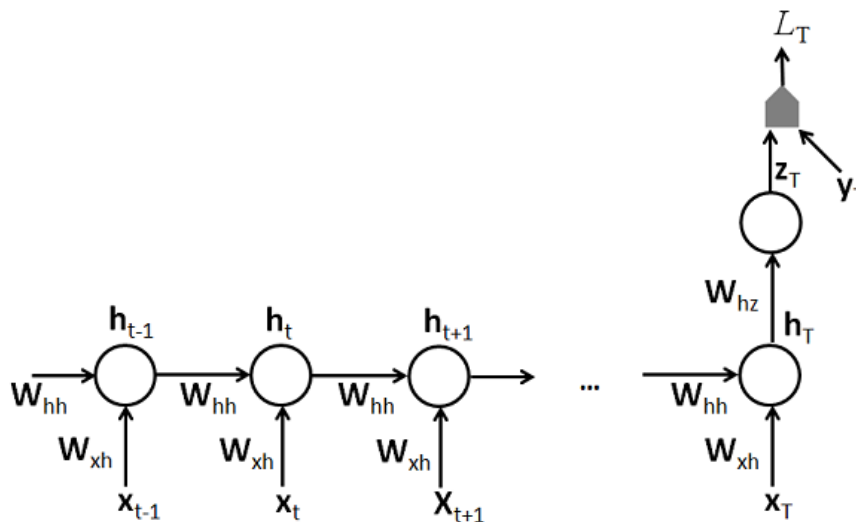
# The Challenge of Long Term Dependencies

- Consider the gradient of a loss  $L_T$  at time  $T$  with respect to the parameter  $\theta$  of the recurrent function  $F_\theta$

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

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

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



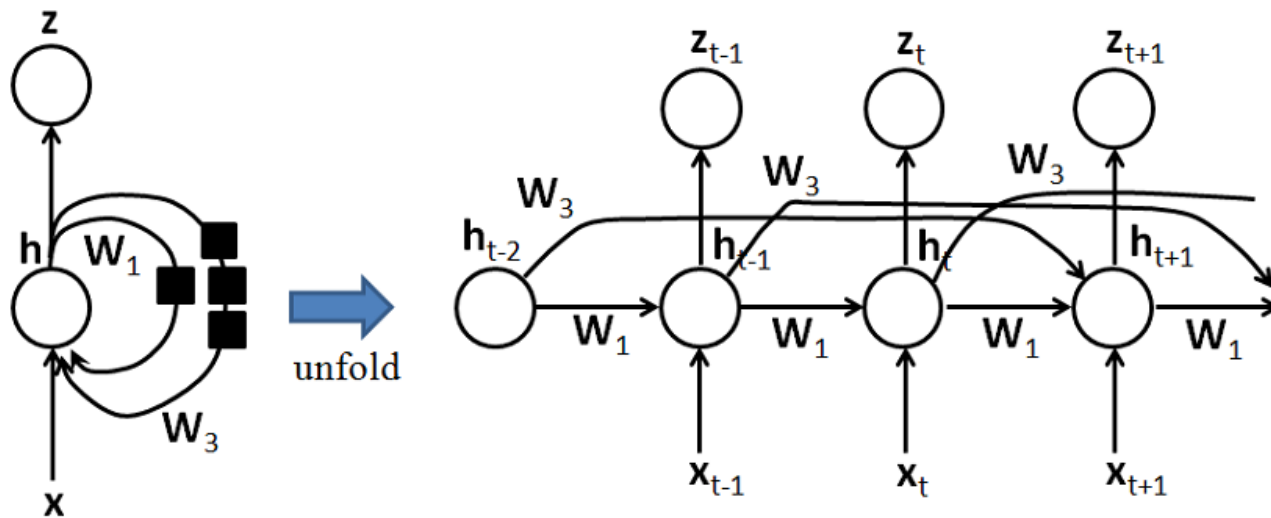
# The Challenge of Long Term Dependencies

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

- 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

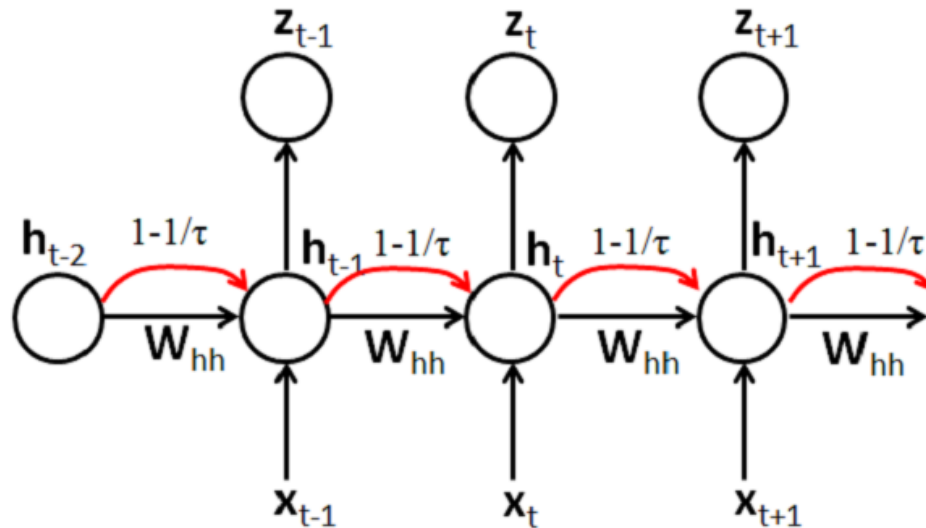
- Longer-delay connections allow to connect the past states to future states through short paths
- Gradients will vanish exponentially with respect to the number of time steps
- If we have recurrent connections with a time-delay of  $D$ , the instead of the vanishing or explosion going as  $O(\lambda^T)$  over  $T$  steps (where  $\lambda$  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



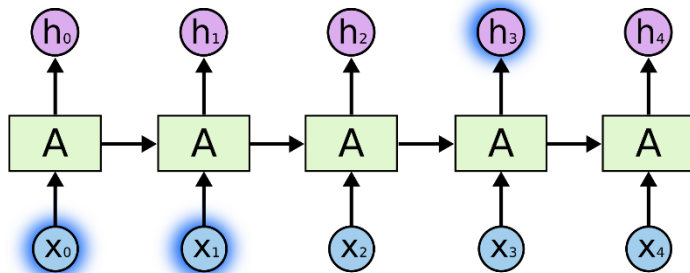
# Leaky units with self-connections

- 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  $\tau$  is large, the state changes very slowly, integrating the past values associated with the input sequence
- $\tau$  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  $\tau$  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

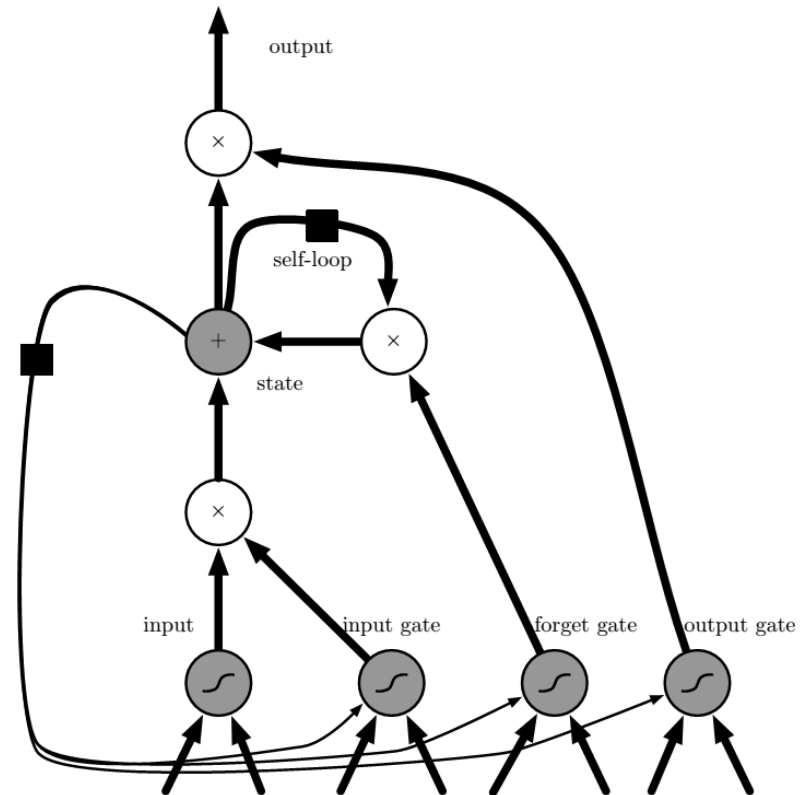
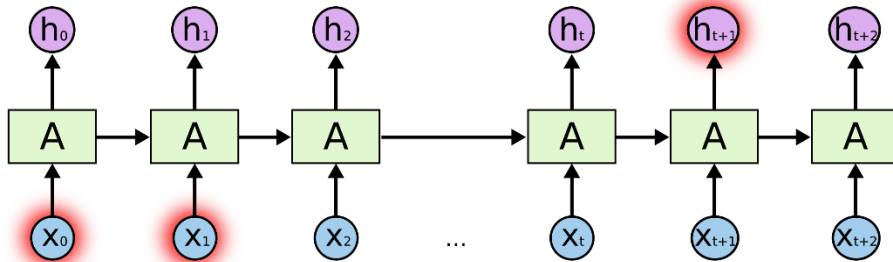
# Long Short-Term Memory (LSTM)

- 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**.”

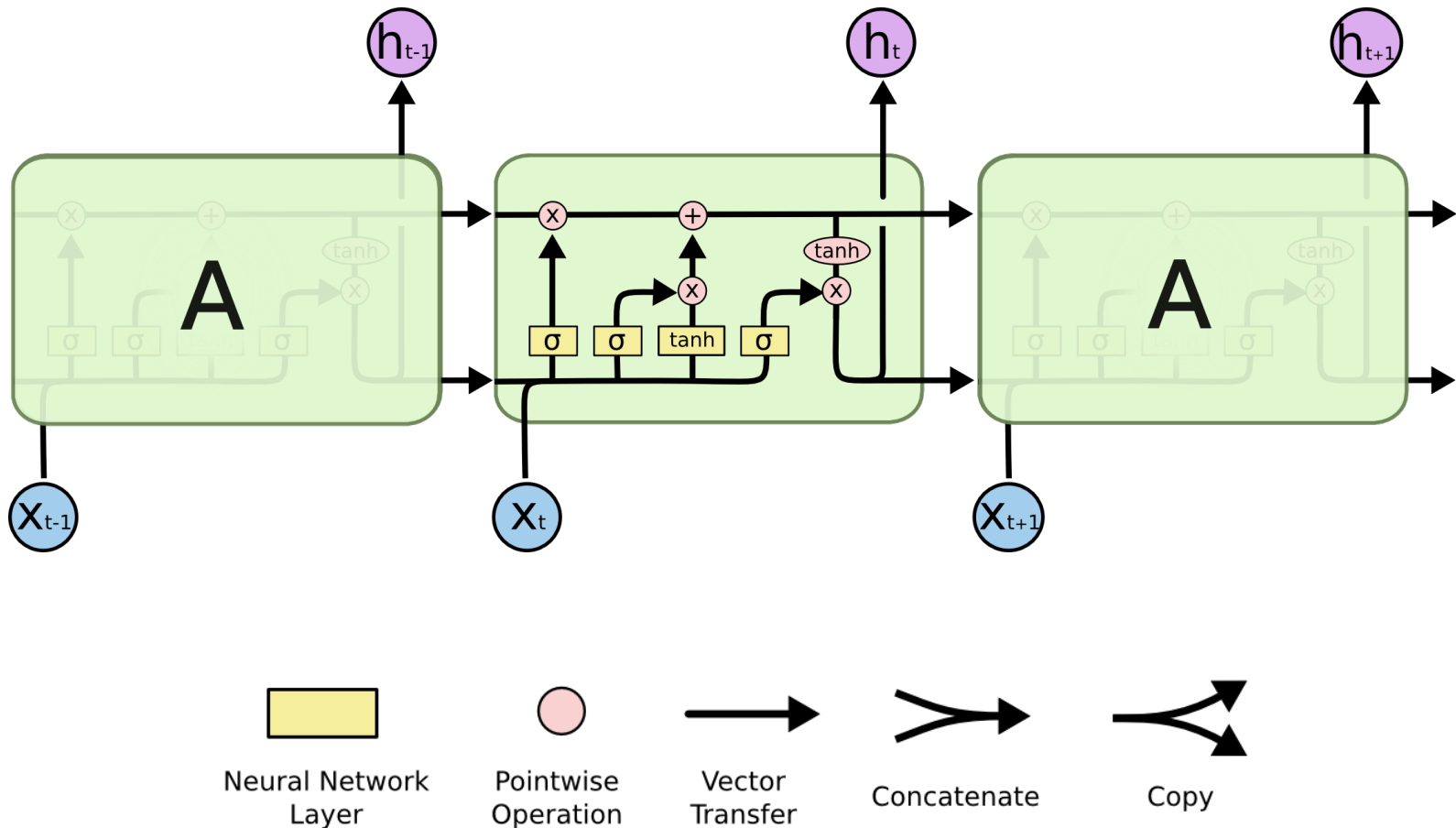


# Long Short-Term Memory (LSTM)

- 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

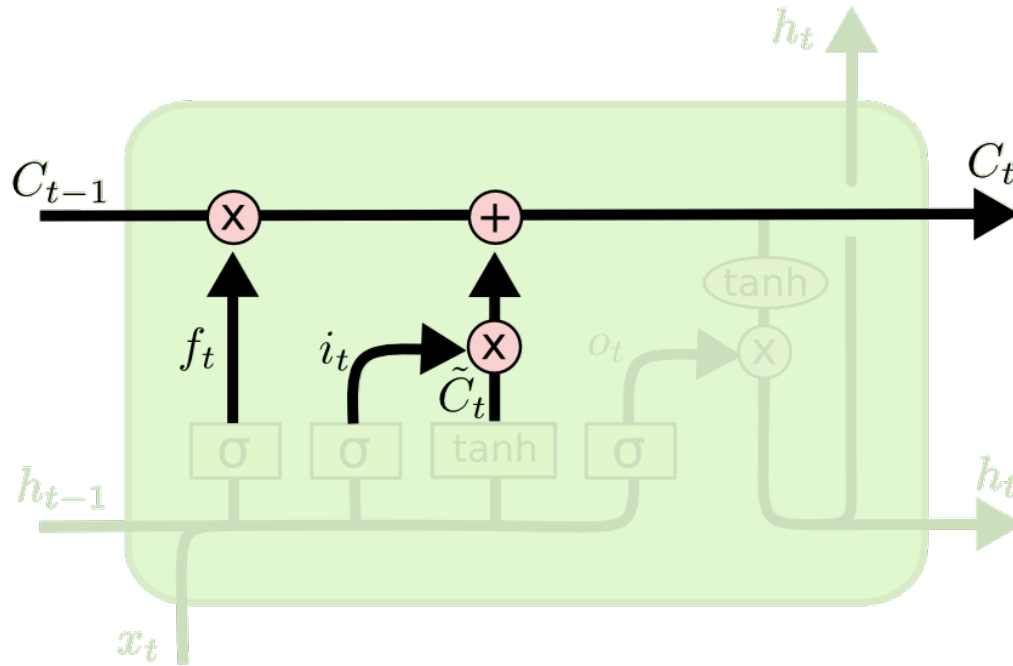
# Long Short-Term Memory (LSTM)

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



# Long Short-Term Memory (LSTM)

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



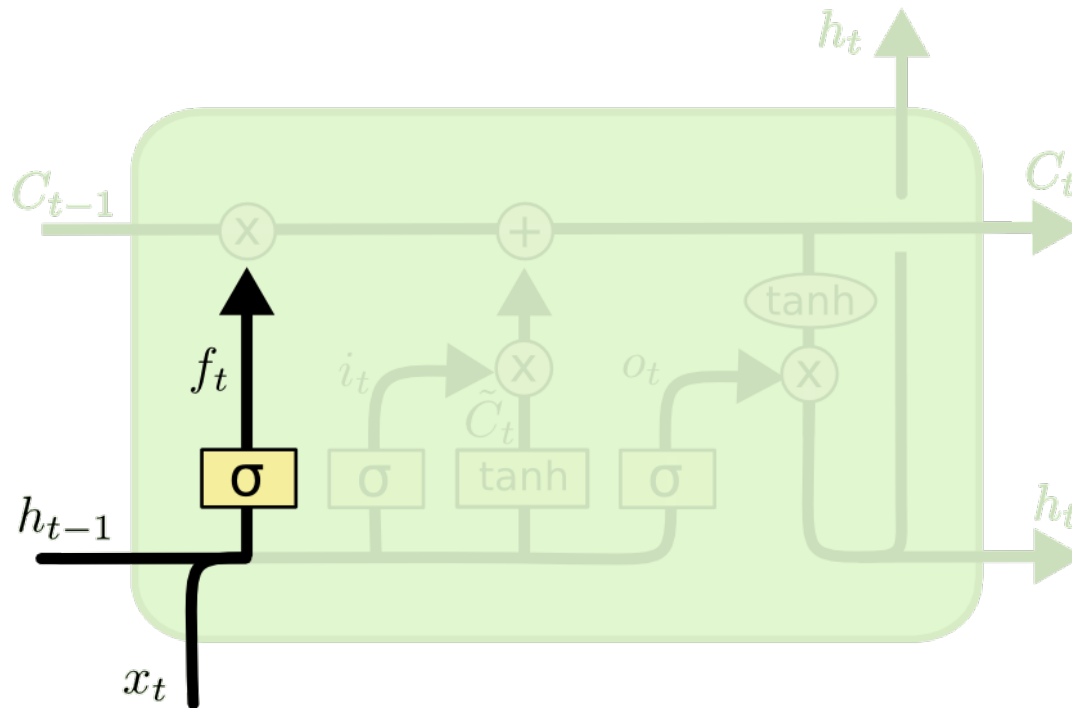
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Updated cell state      Previous cell state      "Proposed" cell state

Forget gate      Input gate

# Long Short-Term Memory (LSTM)

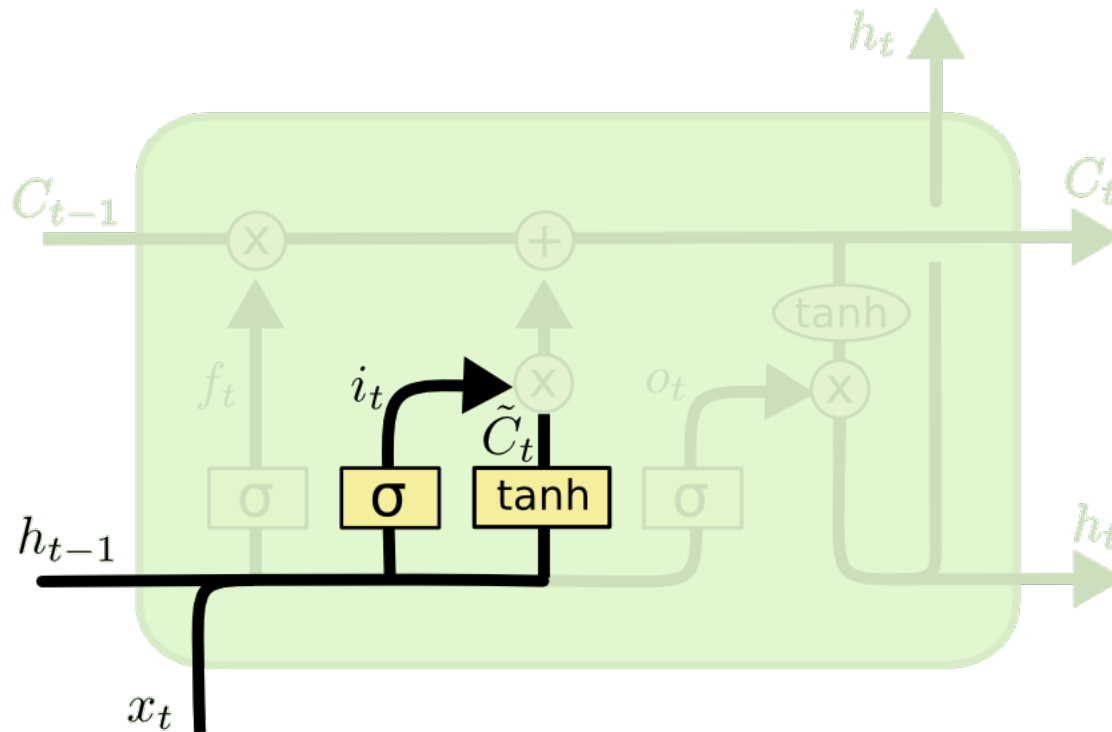
- Forget gate



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

# Long Short-Term Memory (LSTM)

- “proposed” cell state

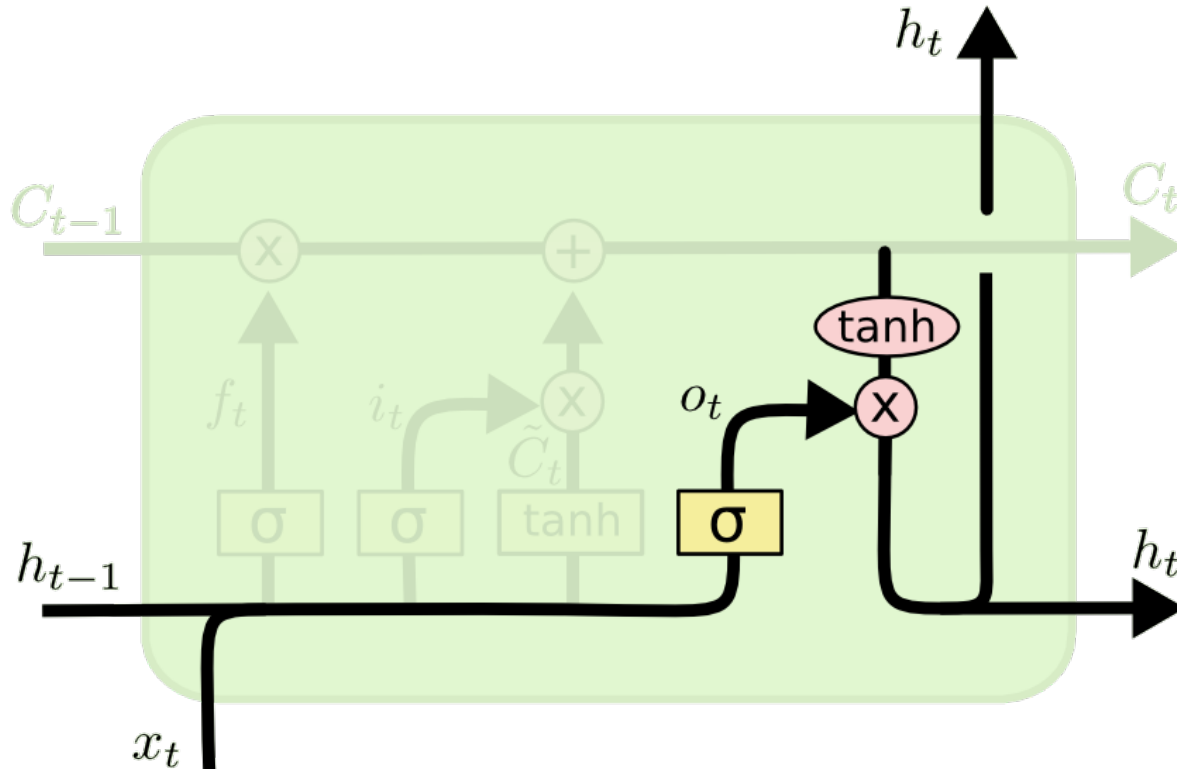


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

# Long Short-Term Memory (LSTM)

- Output gate

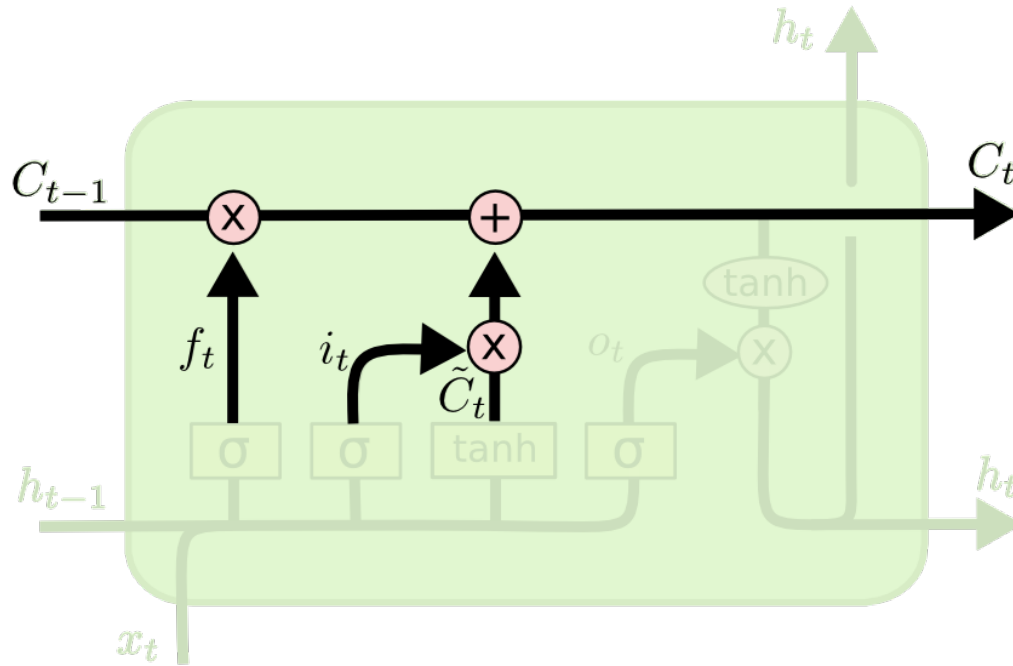


$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

# Long Short-Term Memory (LSTM)

- Review the updating formula



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

Updated cell state      Previous cell state      "Proposed" cell state

Forget gate      Input gate

# Show and Tell: A Neural Image Caption Generator

Oriol Vinyals  
Google

vinyals@google.com

Alexander Toshev  
Google

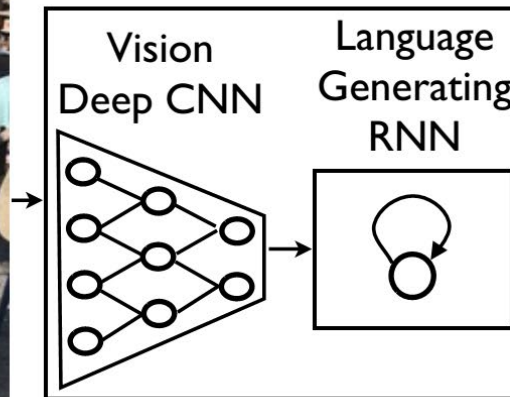
toshev@google.com

Samy Bengio  
Google

bengio@google.com

Dumitru Erhan  
Google

dumitru@google.com



**A group of people  
shopping at an  
outdoor market.**

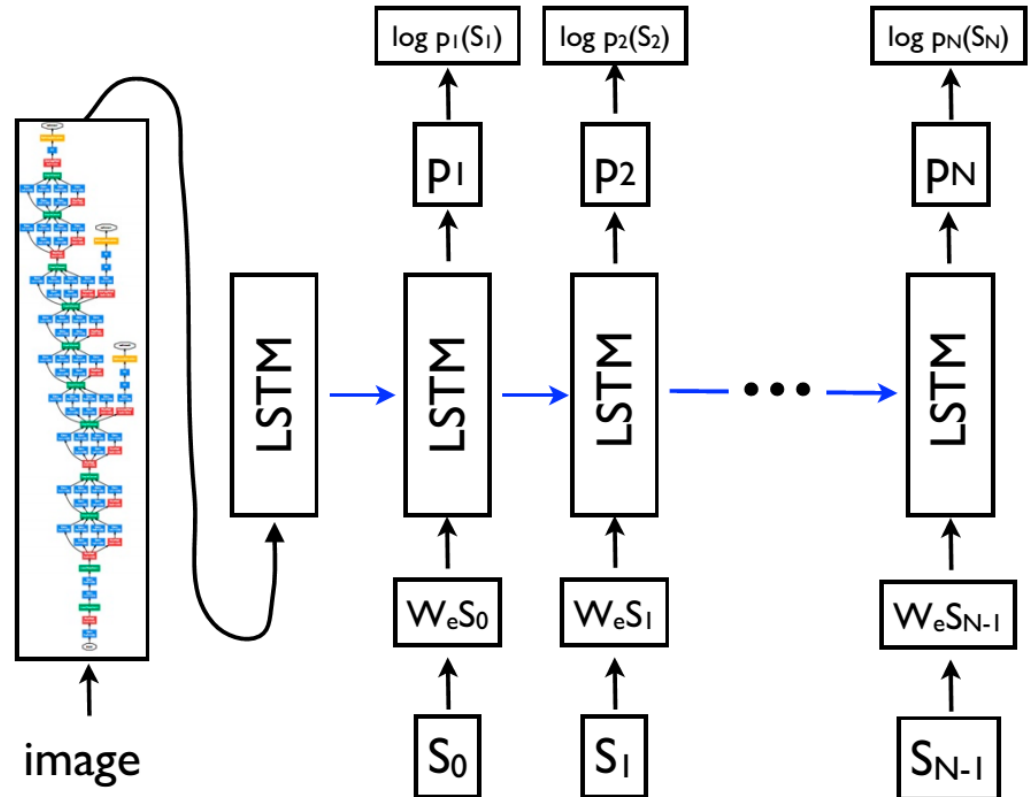
**There are many  
vegetables at the  
fruit stand.**

# 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

A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



A skateboarder does a trick on a ramp.



A dog is jumping to catch a frisbee.



A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



A little girl in a pink hat is blowing bubbles.



A refrigerator filled with lots of food and drinks.



A herd of elephants walking across a dry grass field.



A close up of a cat laying on a couch.



A red motorcycle parked on the side of the road.



A yellow school bus parked in a parking lot.



Describes without errors

Describes with minor errors

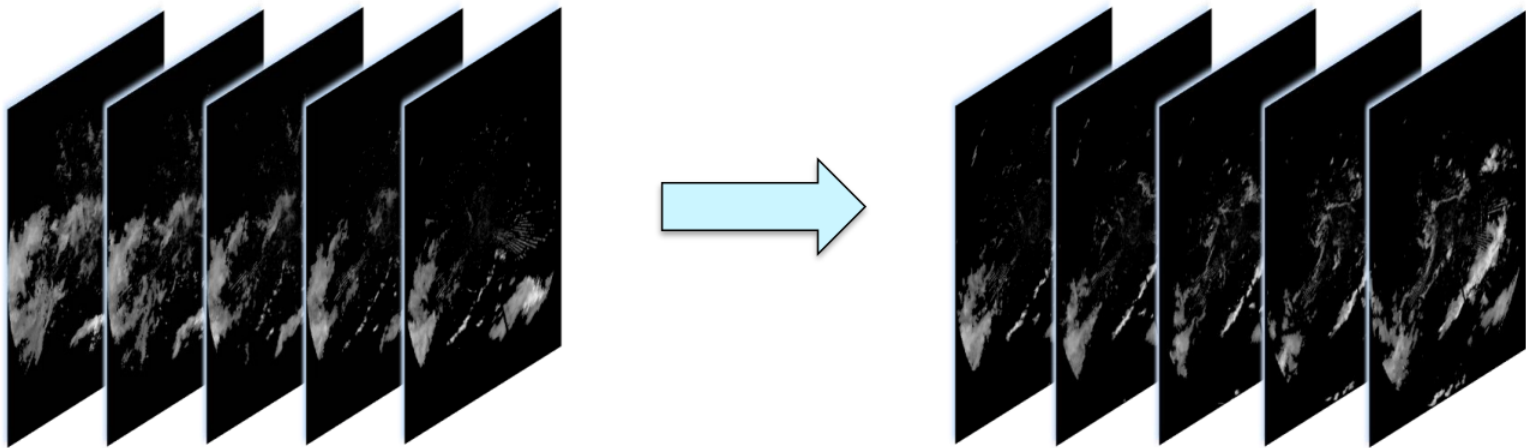
Somewhat related to the image

Unrelated to the image

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

# Convolutional LSTM for precipitation nowcasting

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

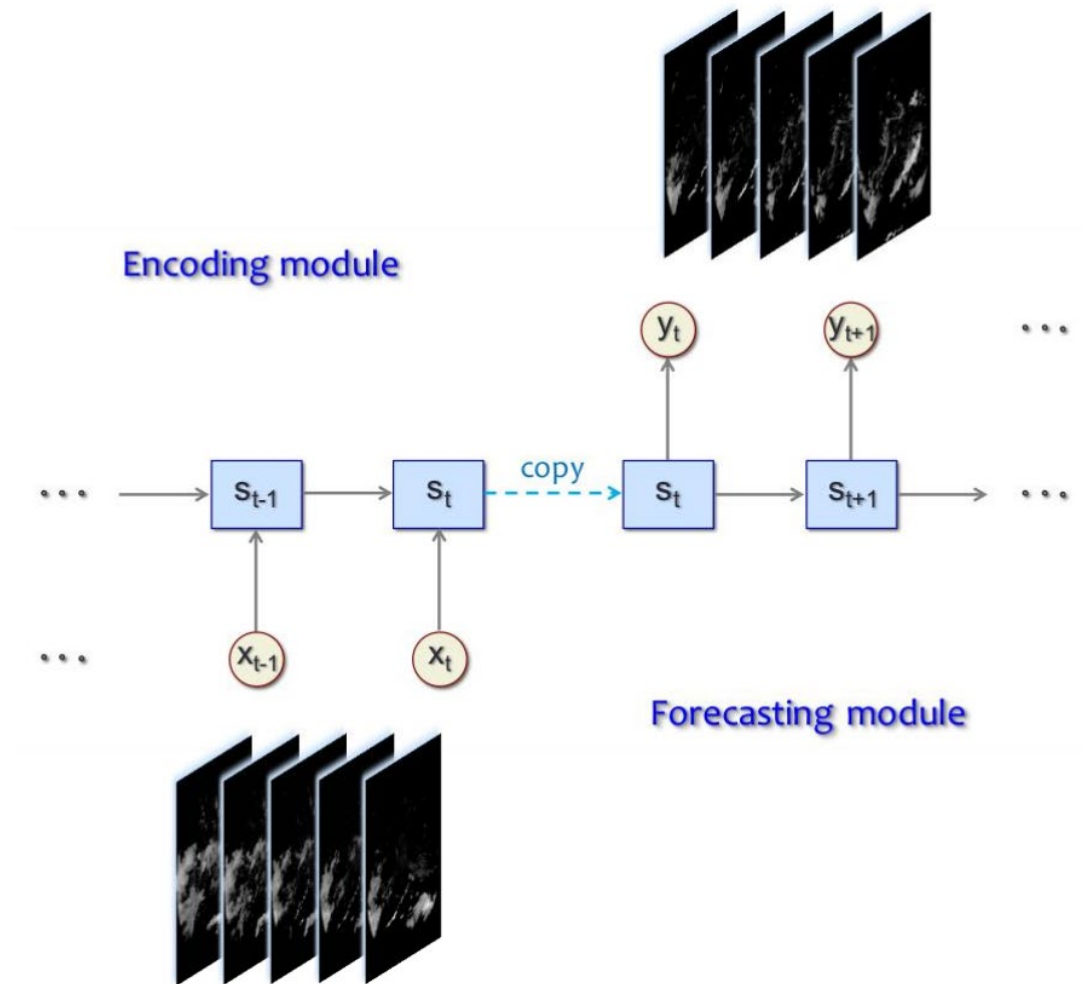


$$\tilde{\mathcal{X}}_{t+1}, \dots, \tilde{\mathcal{X}}_{t+K} = \arg \max_{\mathcal{X}_{t+1}, \dots, \mathcal{X}_{t+K}} p(\mathcal{X}_{t+1}, \dots, \mathcal{X}_{t+K} \mid \hat{\mathcal{X}}_{t-J+1}, \hat{\mathcal{X}}_{t-J+2}, \dots, \hat{\mathcal{X}}_t)$$

X. Shi, Z. Chen, H. Wang, D.Y. Yeung, W.K. Wong, and W.C. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. NIPS 2015.

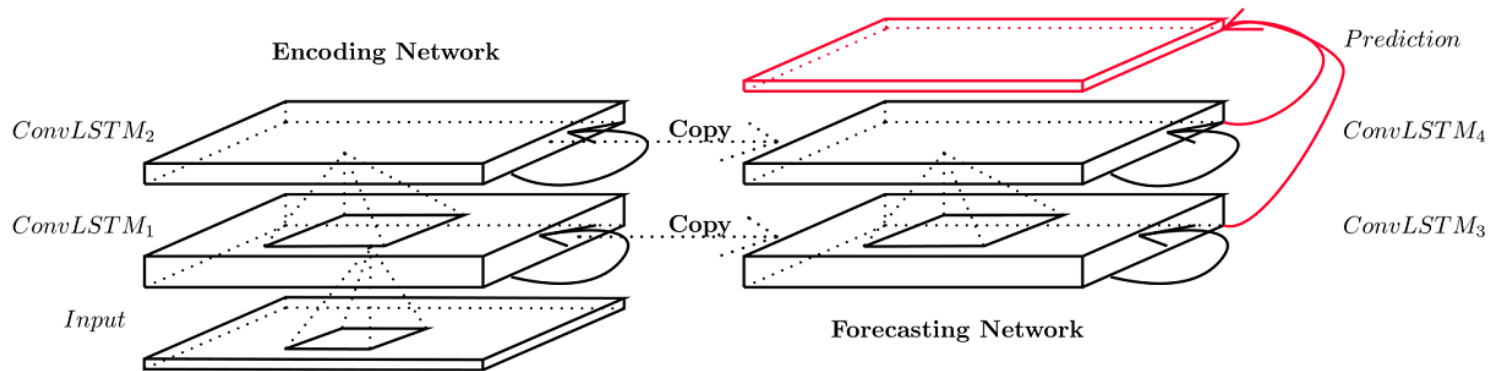
# Convolutional LSTM for precipitation nowcasting

- Encoding-forecasting (encoder-decoder) model



# Convolutional LSTM for precipitation nowcasting

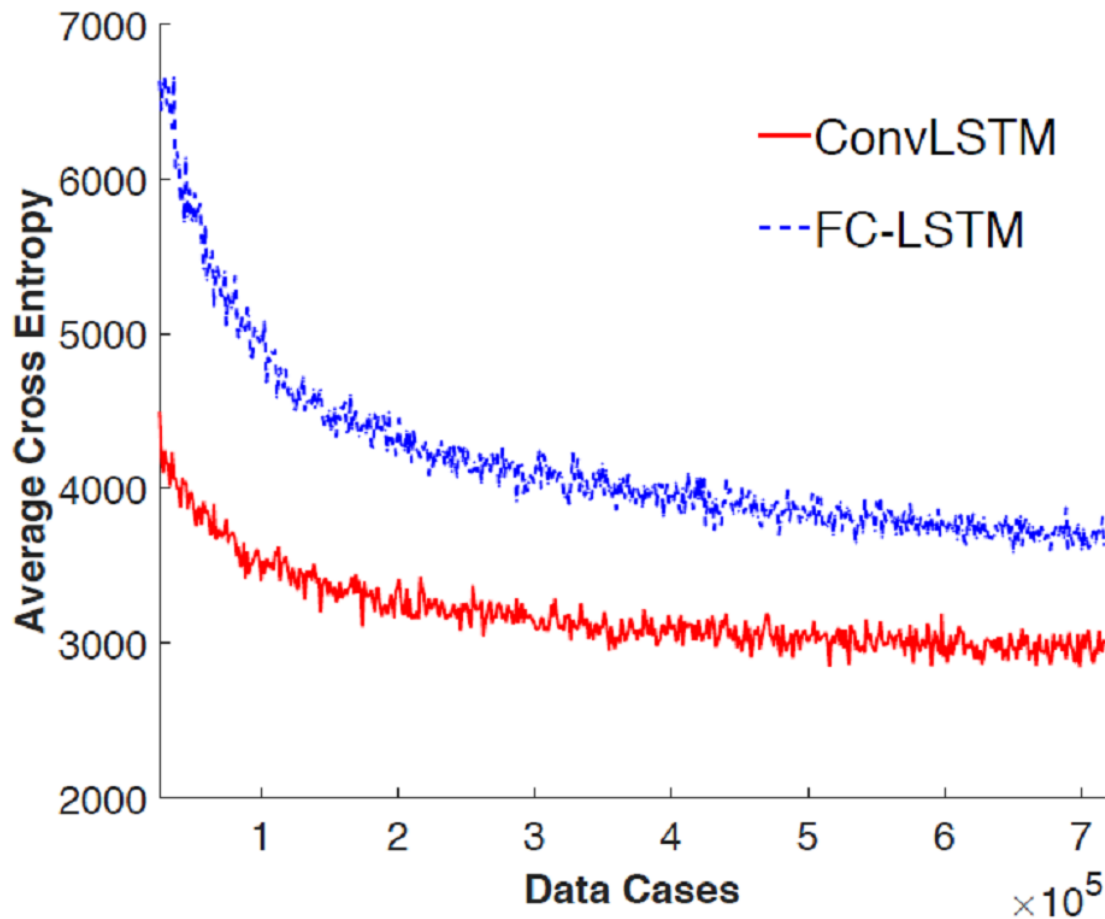
- Last states and cell outputs of encoding network become initial states and cell outputs of forecasting network
- **Encoding network** compresses the input sequence into a hidden state tensor
- **Forecasting network** unfolds the hidden state tensor to make prediction



# Convolutional LSTM for precipitation nowcasting

- ConvLSTM network:
  - 2 ConvLSTM layers, each with 64 units and 3 x 3 kernels
- Fully connected LSTM (FC-LSTM) network:
  - 2 FC-LSTM layers, each with 2000 units
- ROVER:
  - Optical flow estimation
  - 3 variants (ROVER1, ROVER2, ROVER3) based on different initialization schemes

# Convolutional LSTM for precipitation nowcasting

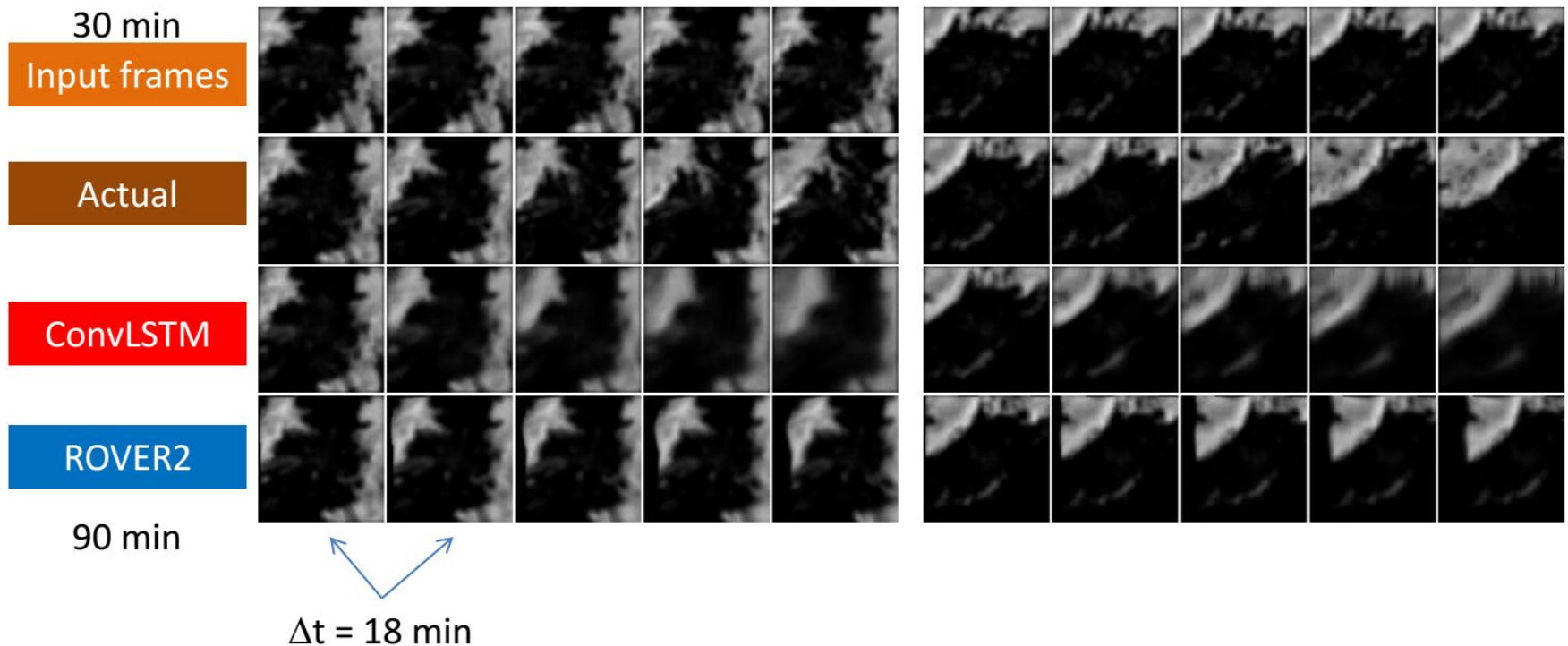


the loss of entropy for ConvLSTM decreases faster than FC-LSTM across all the data cases

→ a better matching with training datasets

# Convolutional LSTM for precipitation nowcasting

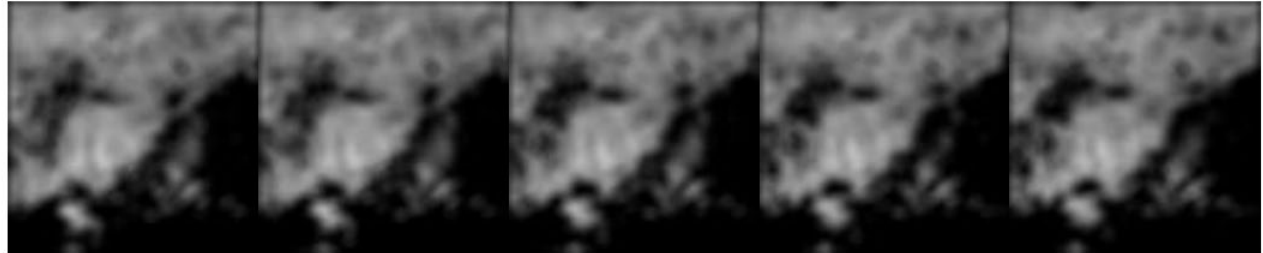
- Radar location (HK) at center ( $\sim 250$  km in x- and y- directions)
- 5 input frames are used and a total of 15 frames (i.e. T+90 min) in forecasts



# Convolutional LSTM for precipitation nowcasting

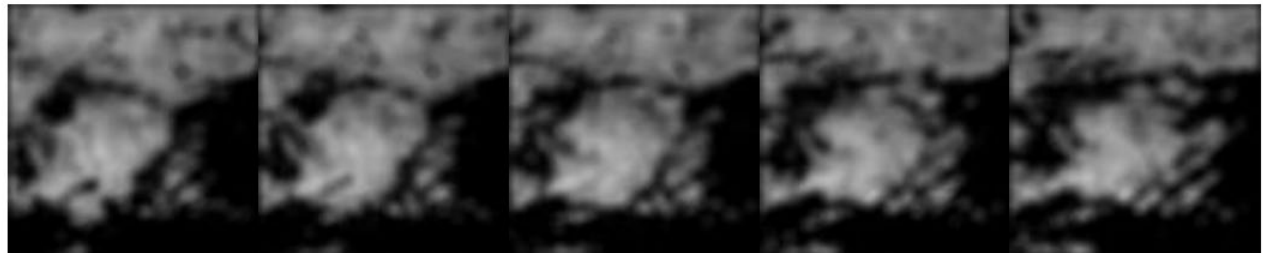
30 min

Input frames

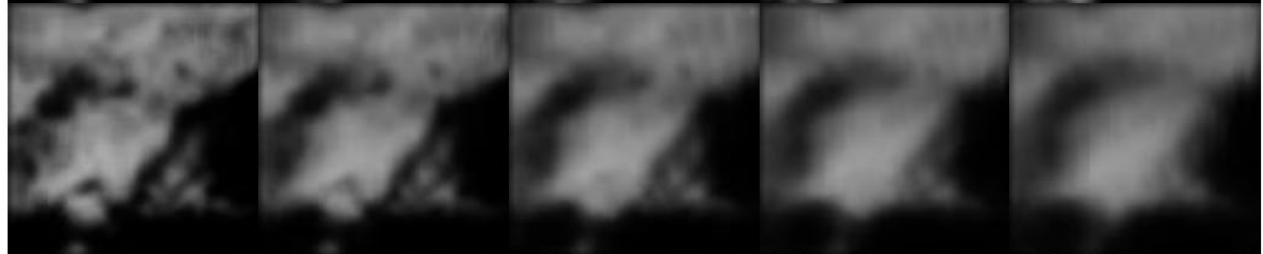


90 min

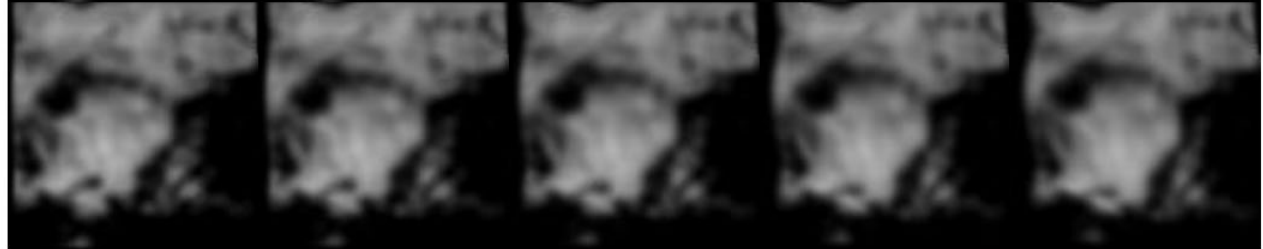
Actual



ConvLSTM



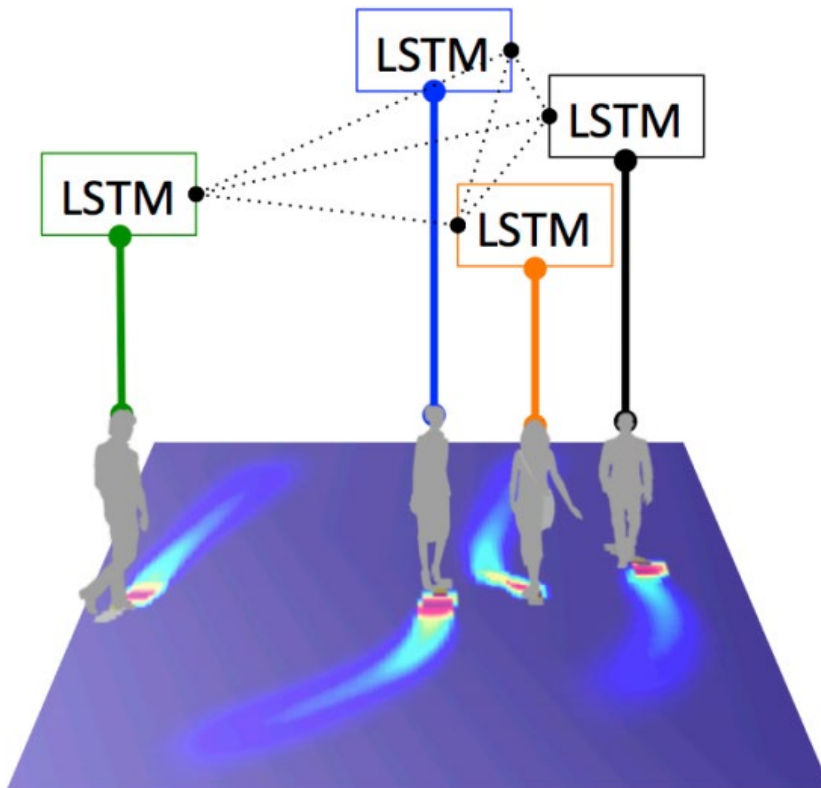
ROVER2



# Social LSTM: Human Trajectory Prediction in Crowded Spaces

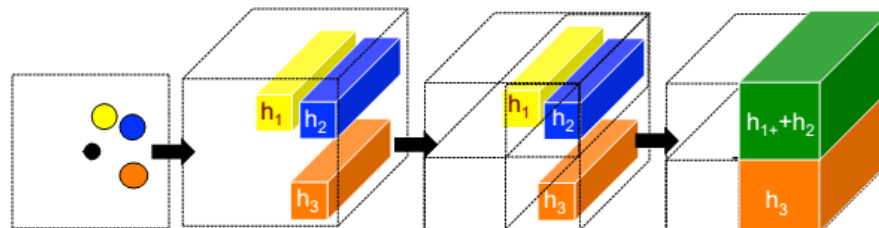
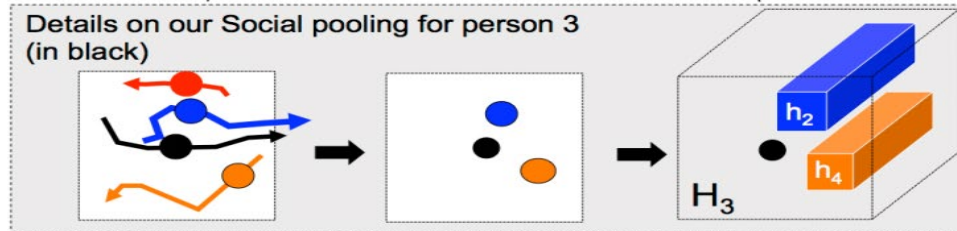
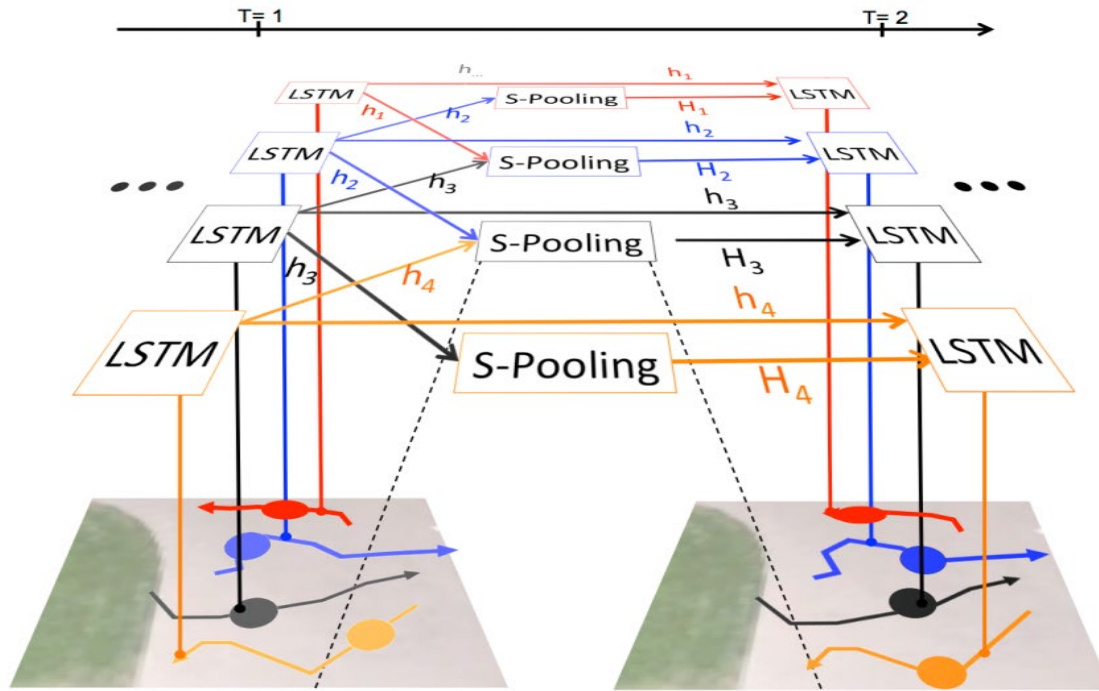
Alexandre Alahi\*, Kratarth Goel\*, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, Silvio Savarese  
Stanford University

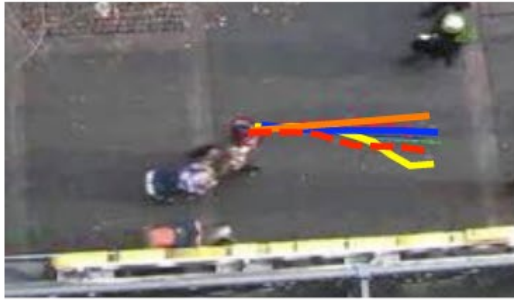
{alahi,kratarth,vigneshr,arobicqu,feifeili,ssilvio}@cs.stanford.edu



The goal of this paper is to predict the motion dynamics in crowded scenes - This is, however, a challenging task as the motion of each person is typically affected by their neighbors.

We propose a new model which we call "Social" LSTM (SocialLSTM) which can jointly predict the paths of all the people in a scene by taking into account the common sense rules and social conventions that humans typically utilize as they navigate in shared environments. The predicted distribution of their future trajectories is shown in the heat-map.

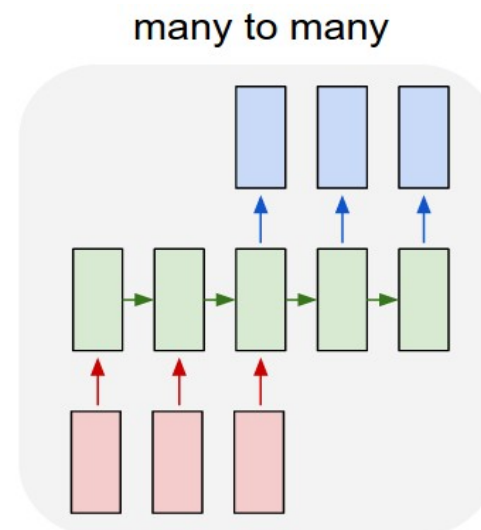




# Soft Attention for Translation

- Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

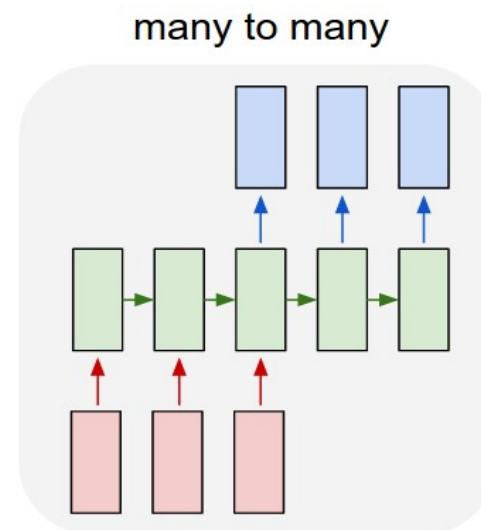
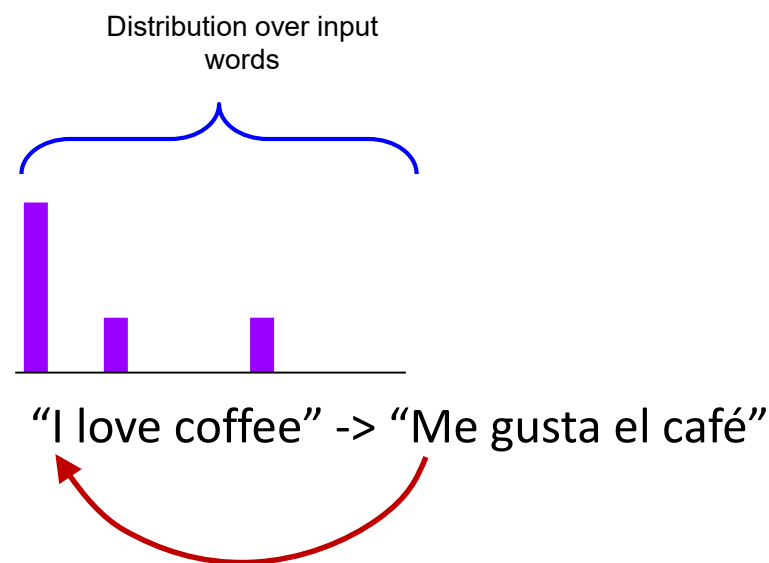
“I love coffee” -> “Me gusta el café”



Base on cs231n by Fei-Fei Li et al.

# Soft Attention for Translation

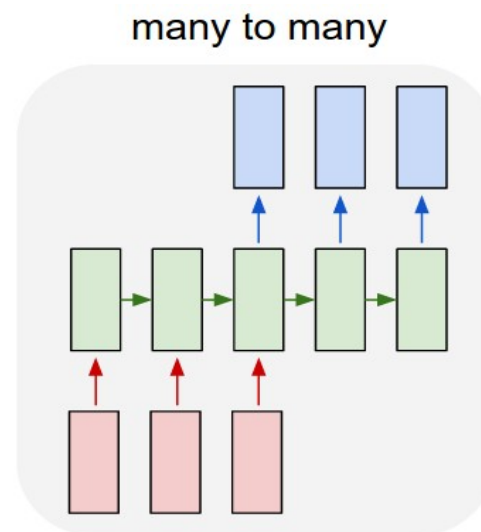
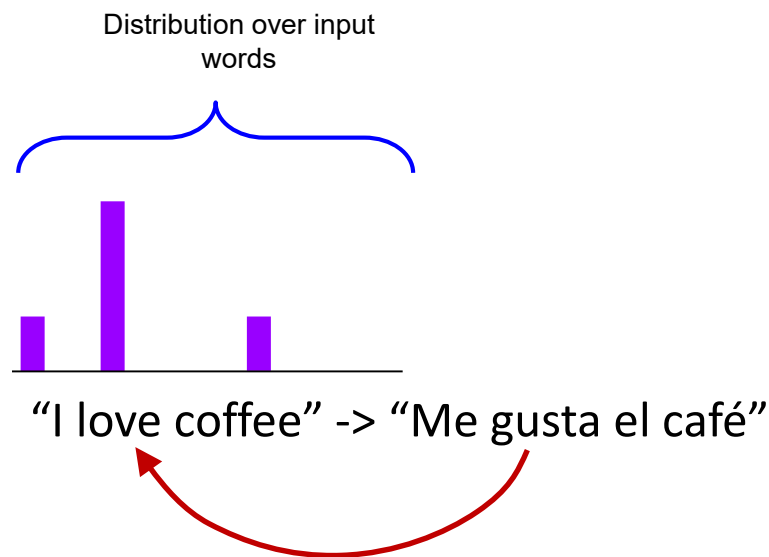
- Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015



Base on cs231n by Fei-Fei Li et al.

# Soft Attention for Translation

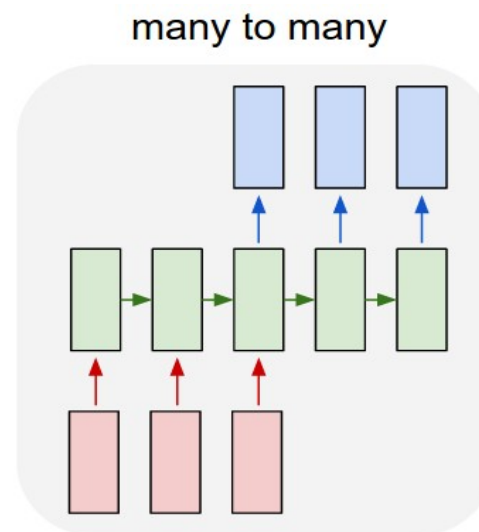
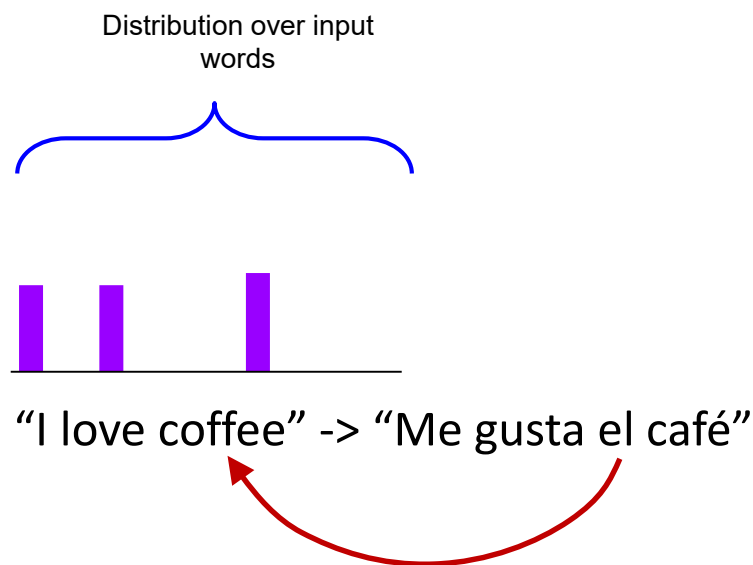
- Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015



Base on cs231n by Fei-Fei Li et al.

# Soft Attention for Translation

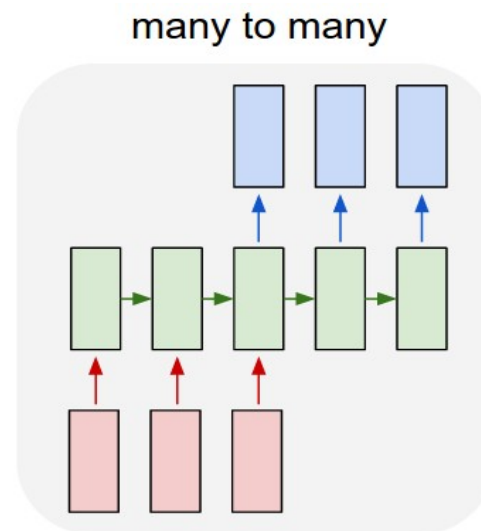
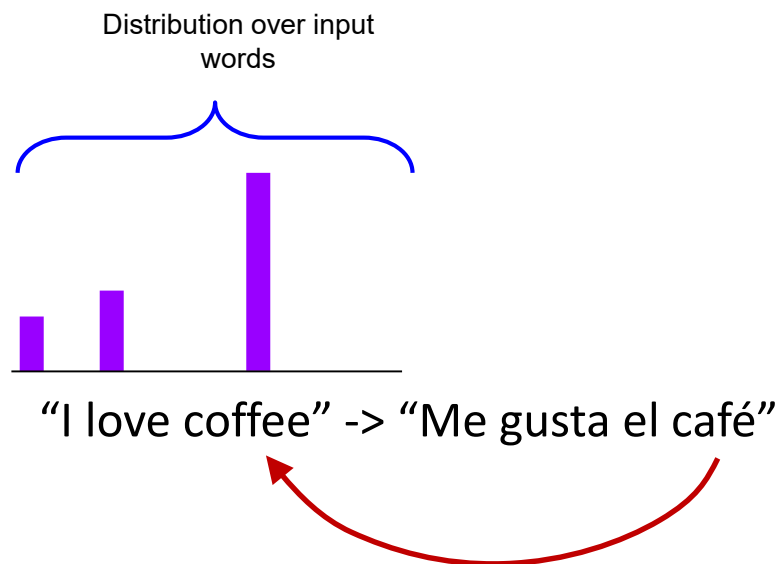
- Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015



Base on cs231n by Fei-Fei Li et al.

# Soft Attention for Translation

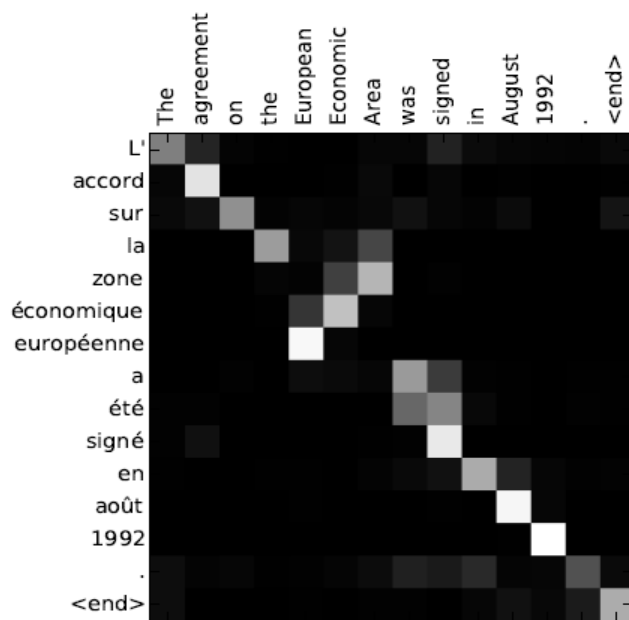
- Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015



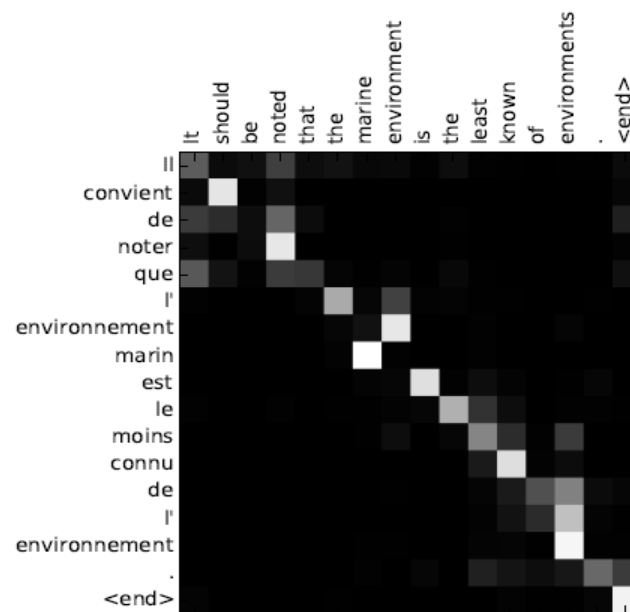
Base on cs231n by Fei-Fei Li et al.

# Soft Attention for Translation

- Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

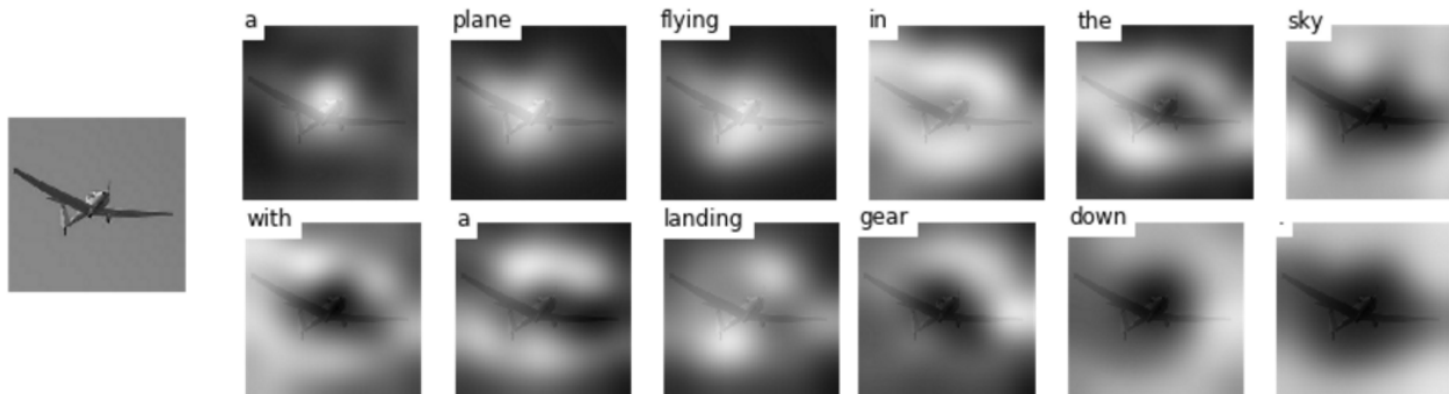
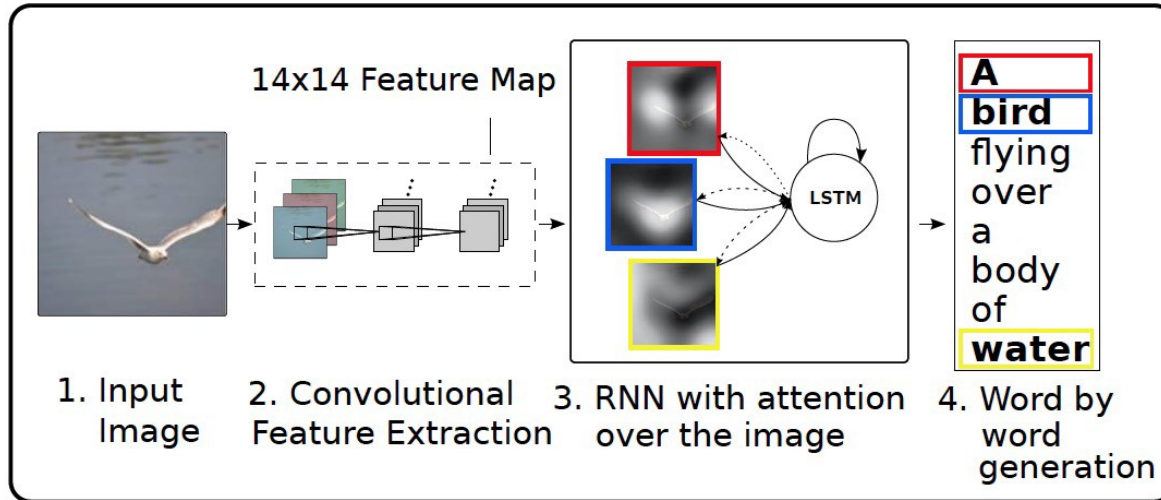


(a)

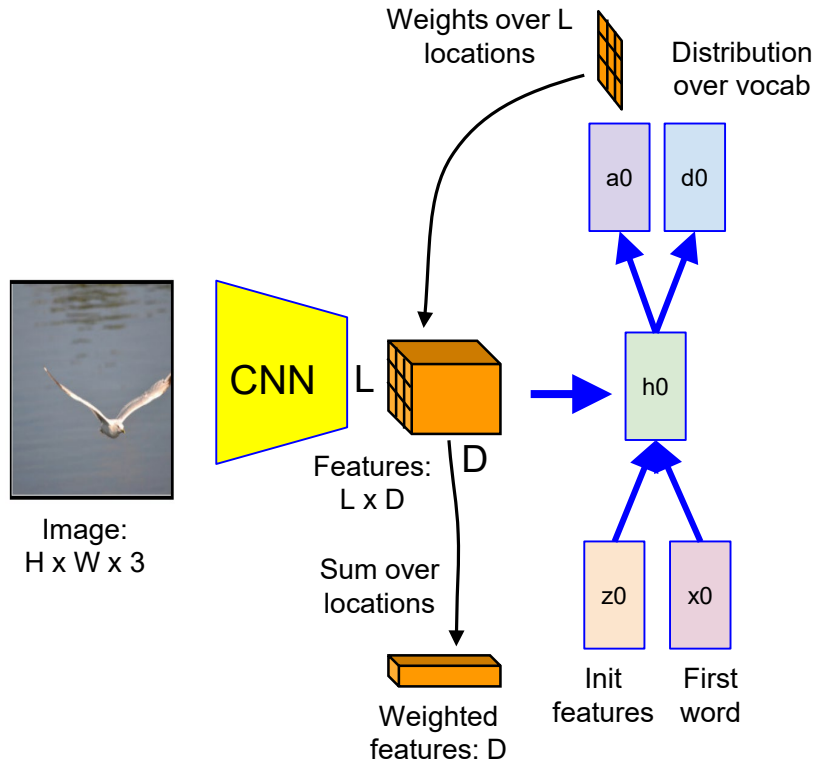


(b)

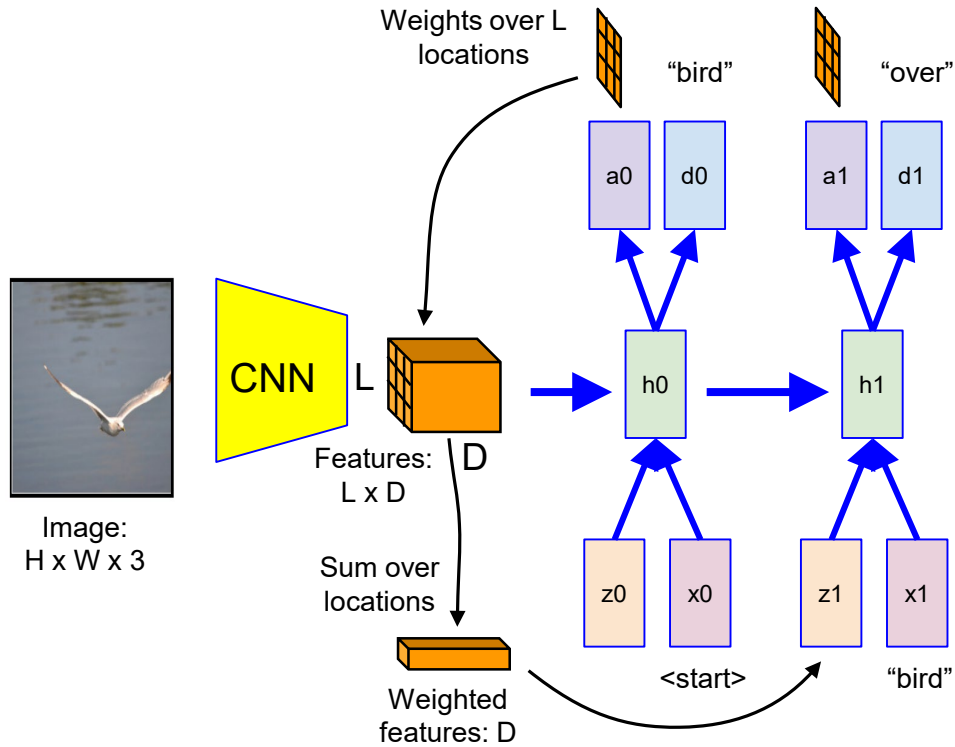
# Show, Attend and Tell



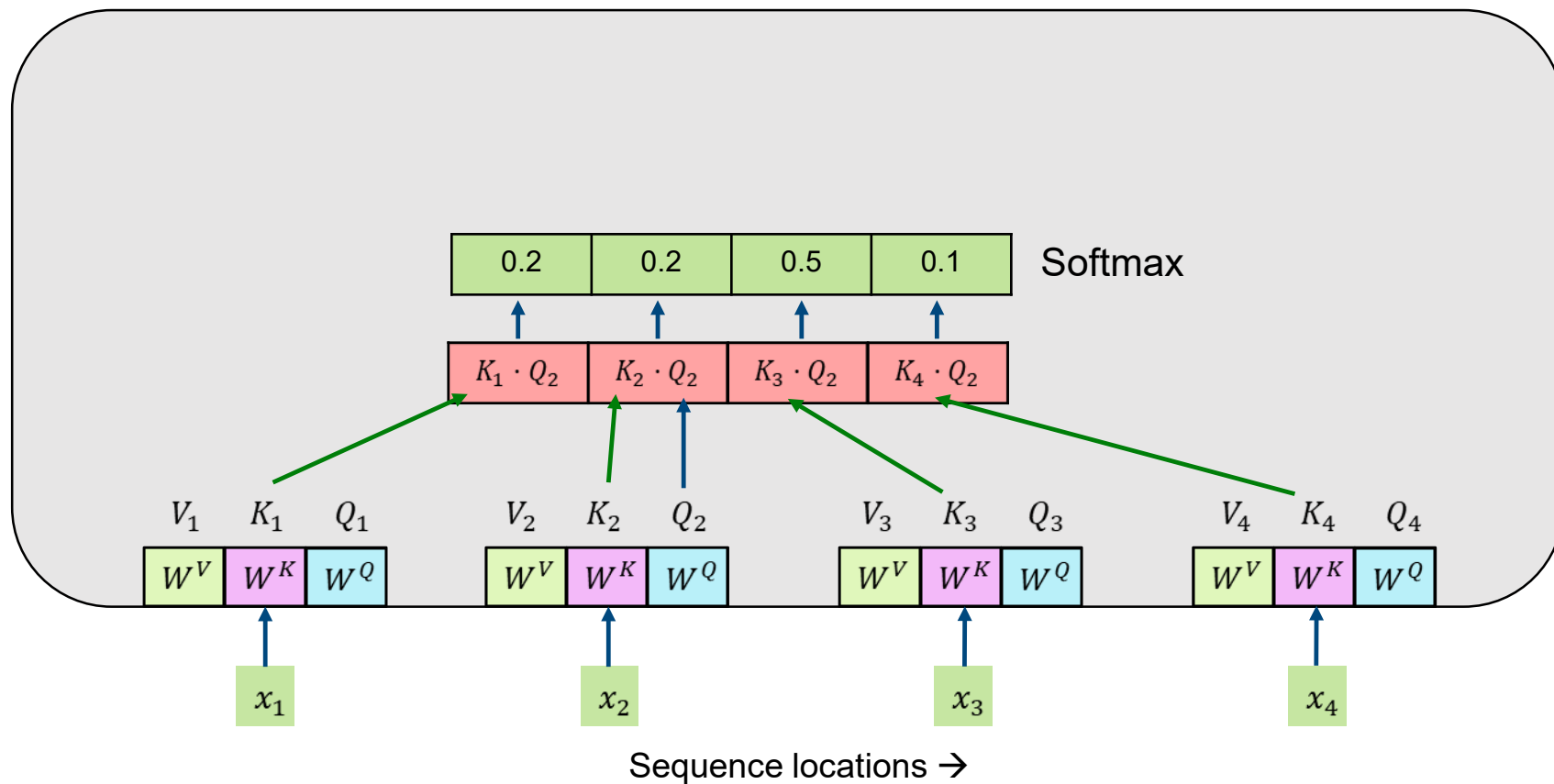
# RNN for Captioning



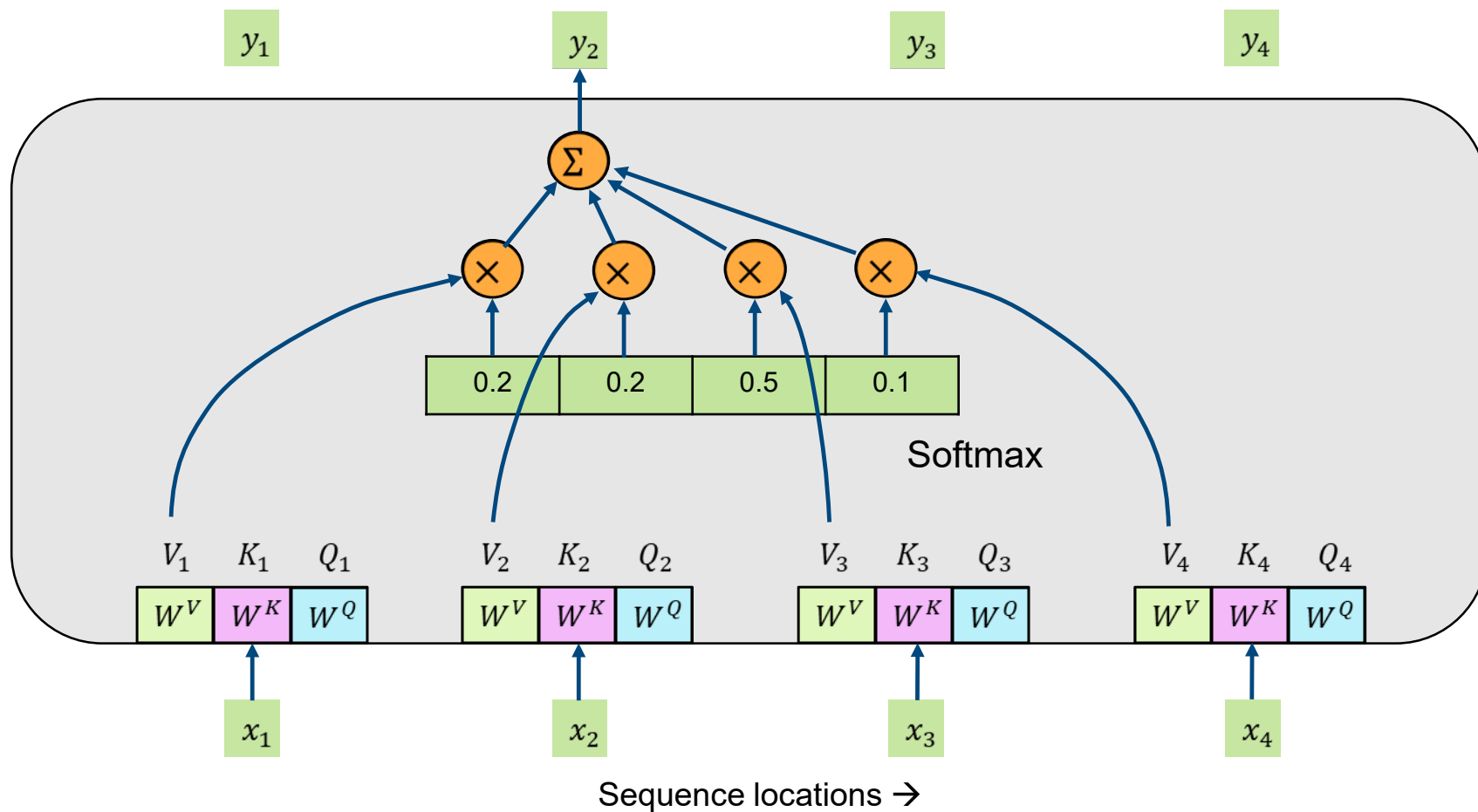
# RNN for Captioning



# The Transformer – Self-Attention Layer



# The Transformer – Self-Attention Layer



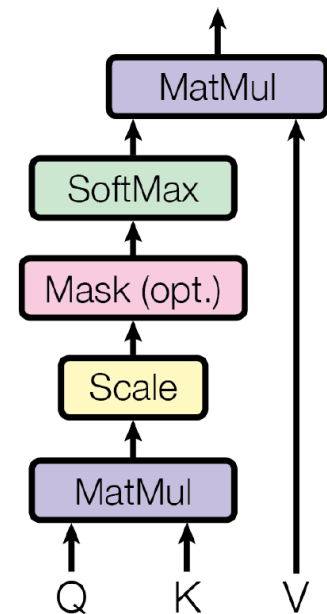
# Attention Implementation with matrices

- Transformer networks have extreme parallelism by using *matrices* to hold all the vectors in the network:  
Q = matrix of all query vectors (as rows)  
K = matrix of all keys (as rows)  
V = matrix of all values (as rows)
- The row index is the position in the sequence.
- The entire attention operation can be computed as a *single matrix formula* as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

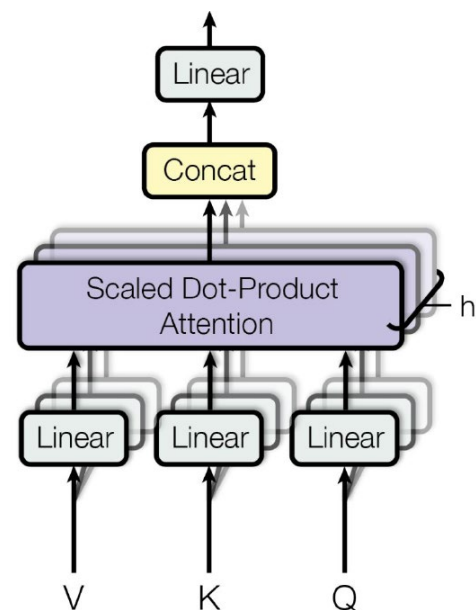
where the softmax is applied across rows (not columns).

Scaled Dot-Product Attention



# Multi-Headed Attention

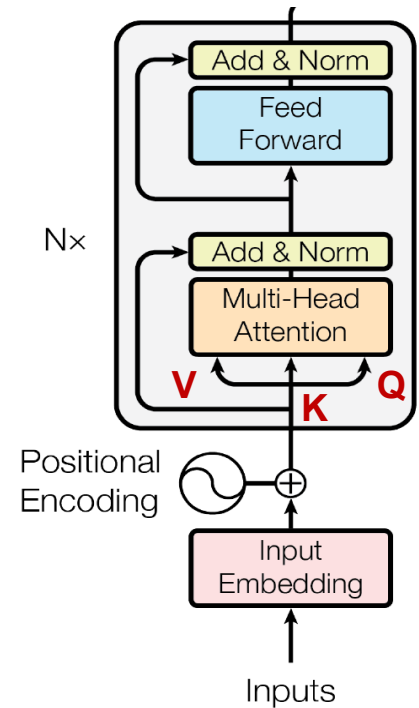
- Standard attention allows each location to attend with a single weight/value embedding to another location.
- We can extend this with “multi-headed” attention by breaking inputs and outputs into ranges, and applying different embeddings for each range.
- The figure to the right shows  $h$  heads.





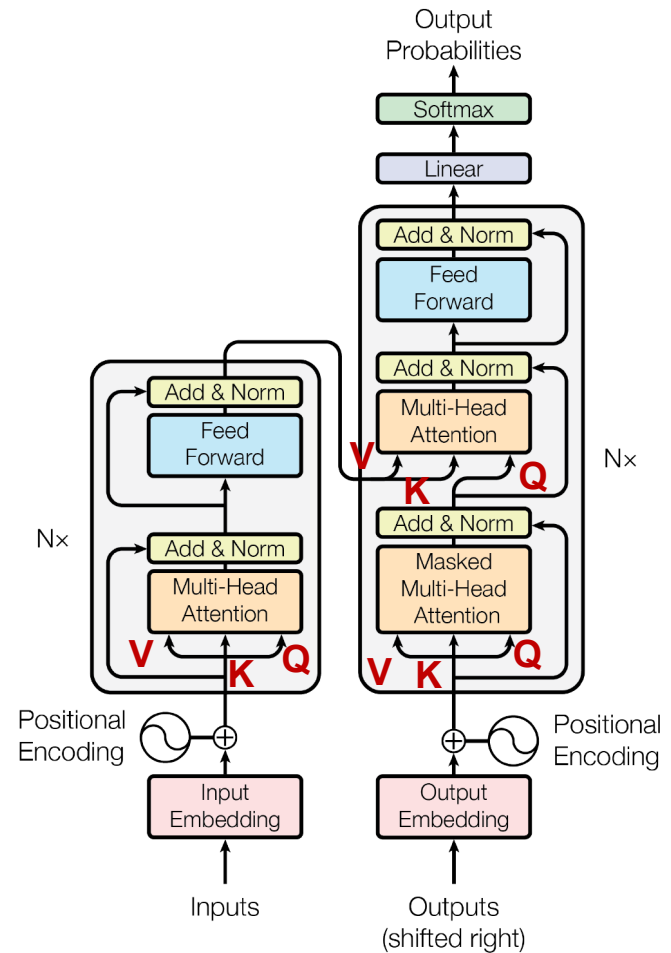
# Transformer Encoder

- Basic unit shown at right.
- The input is a sequence of symbols at the bottom.
- Because different positions are encoded as matrices, its common not to show the sequence positions.
- Multiple layers can be stacked.

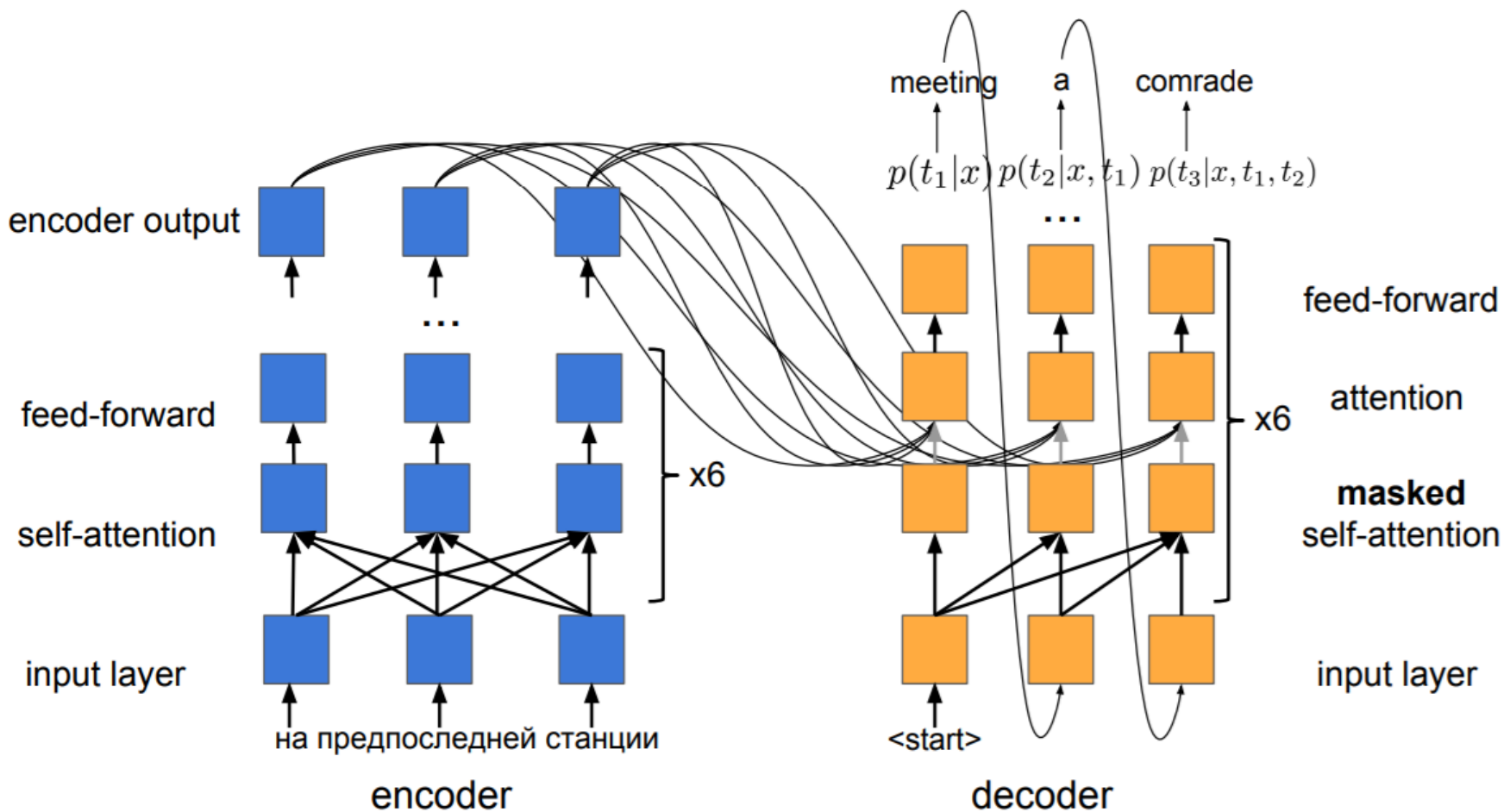


# The Transformer Encoder/Decoder

- Basic unit shown at right.
- Now there is both an encoder with self-attention, and a decoder with both masked self-attention and cross-attention.
- In experiments, stacked with  $N=6$ .
- Inputs and outputs are embedded in vector spaces of fixed dimension.
- Positional encoding: when words are combined through attention, their location is lost. Positional encoding adds it back.



# Transformer: "Attention is All You Need"



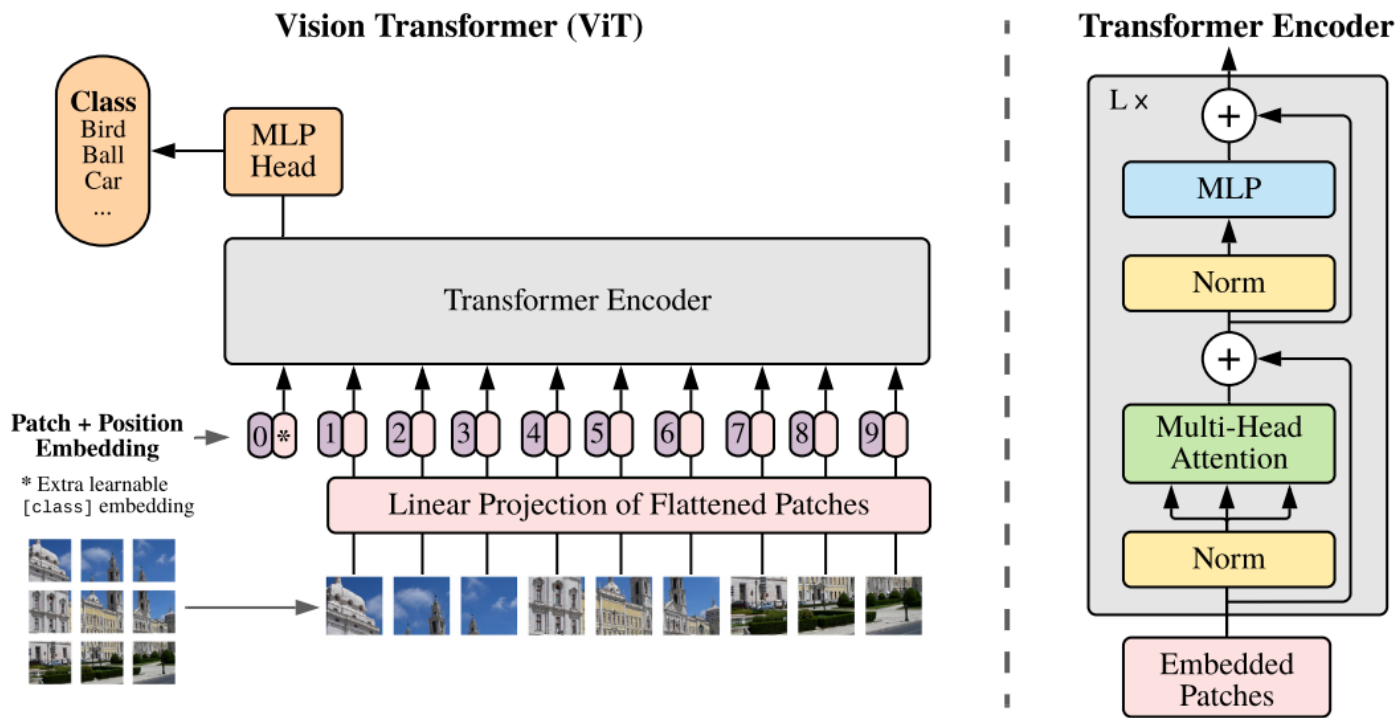


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

# DETR (Detection Transformer)

- Global reasoning via attention
- anchors or hand crafted algorithms like NMS are not needed

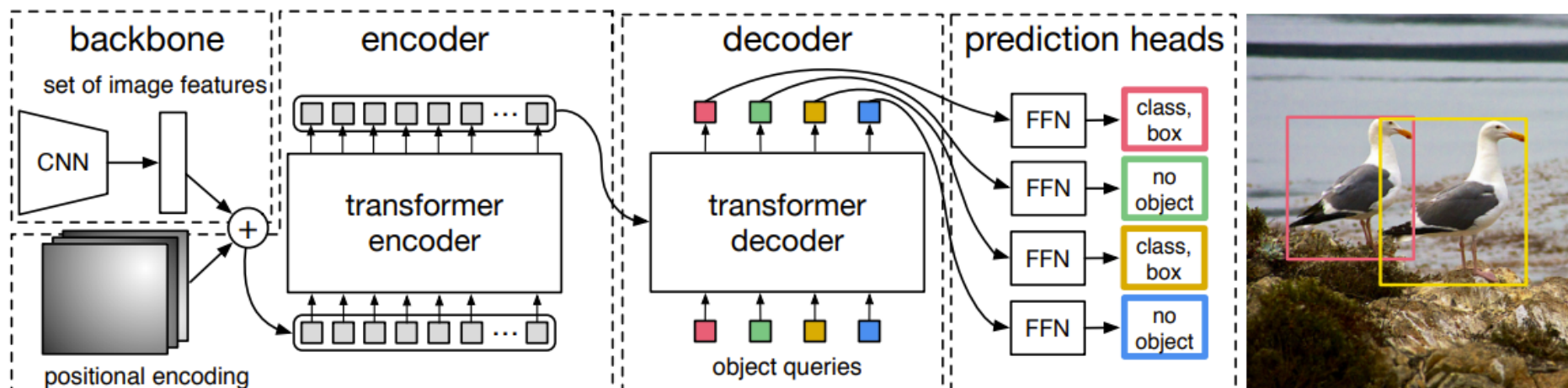
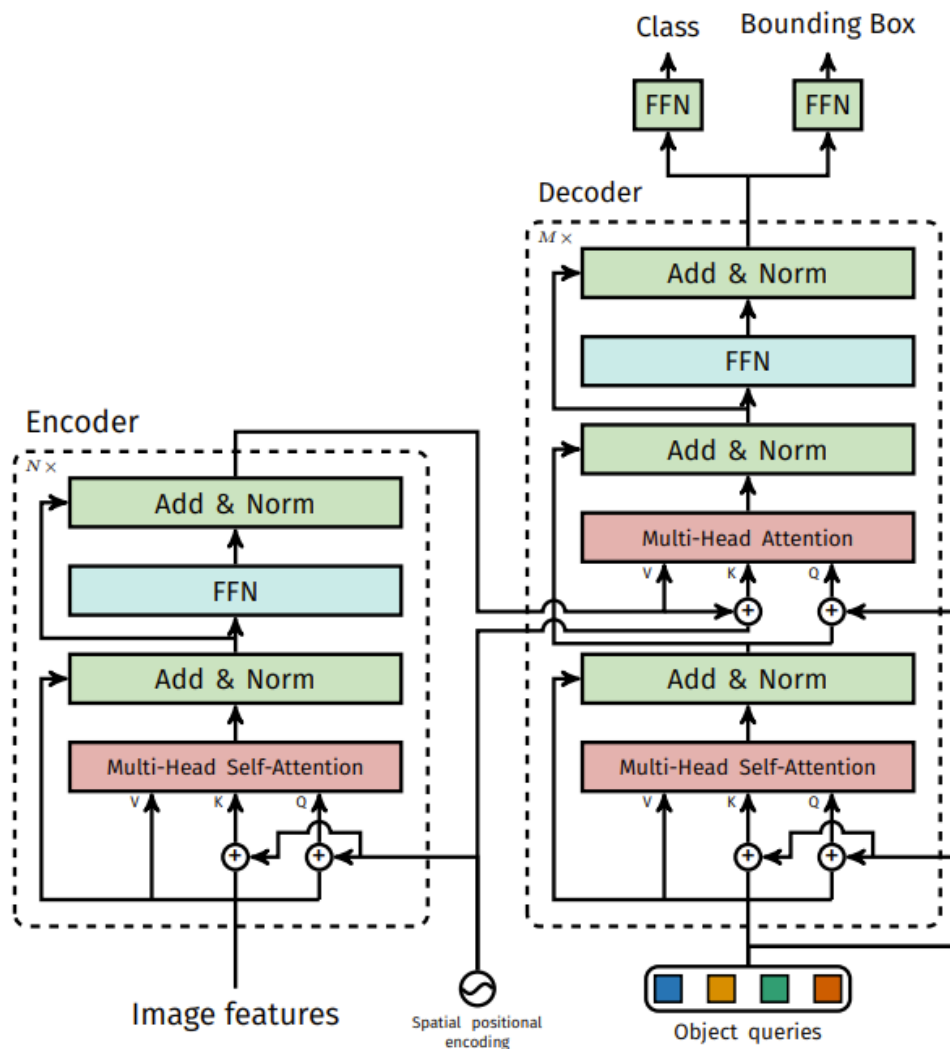


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a “no object” class.

# DETR (Detection Transformer)

- Encoder-decoder architecture
- decode the  $N$  objects in parallel at each decoder layer



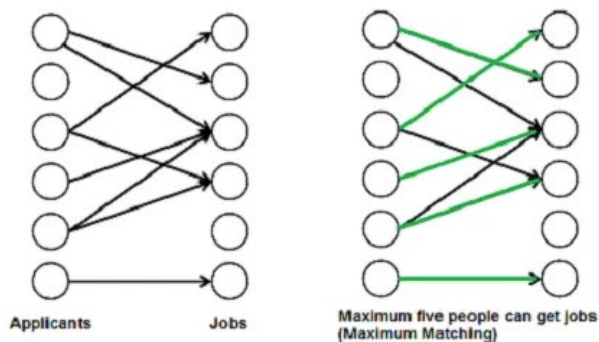
# DETR (Detection Transformer)

- set prediction loss

Let us denote by  $y$  the ground truth set of objects, and  $\hat{y} = \{\hat{y}_i\}_{i=1}^N$  the set of  $N$  predictions. Assuming  $N$  is larger than the number of objects in the image, we consider  $y$  also as a set of size  $N$  padded with  $\emptyset$  (no object). To find a bipartite matching between these two sets we search for a permutation of  $N$  elements  $\sigma \in \mathfrak{S}_N$  with the lowest cost:

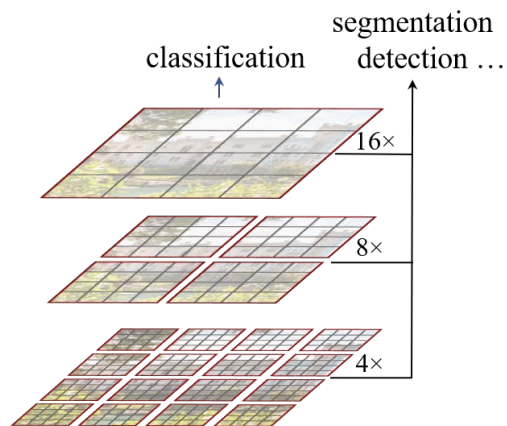
$$\hat{\sigma} = \arg \min_{\sigma \in \mathfrak{S}_N} \sum_i^N \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}), \quad (1)$$

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^N \left[ -\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbf{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}(i)}) \right]$$

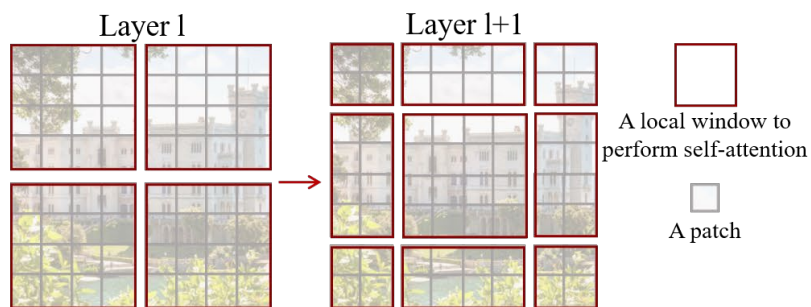


# SwinTransformer (Shifted Window Transformer)

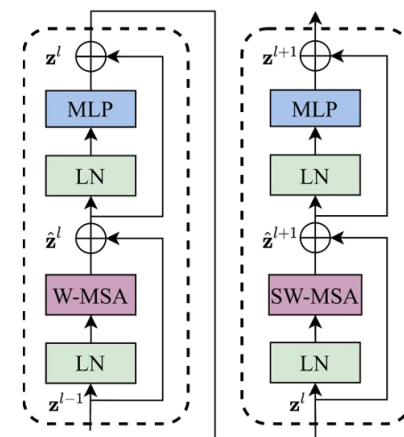
## hierarchical feature maps and shifted window MSA



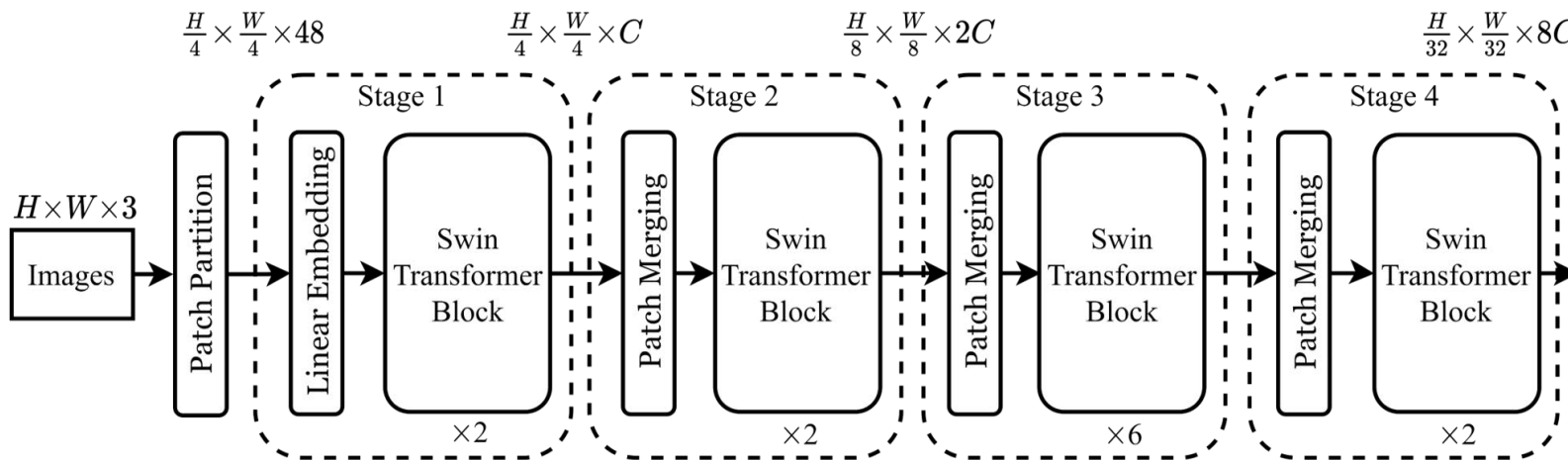
(a) Swin Transformer



(b) Shifted Window



(c) Two Successive Swin Transformer Blocks



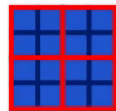
(d) Architecture



# SwinTransformer (Shifted Window Transformer)

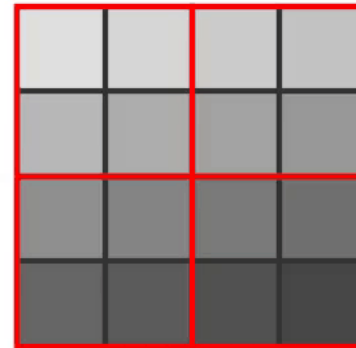
## Window-based MSA

Attention for each patch is only computed within its own window (drawn in red).  
Window size is 2x2 in this example.



## Shifted Window MSA

Step 1: Shift window by a factor of  $M/2$ , where  $M$  = window size  
Step 2: For efficient batch computation, move patches into empty slots to create a complete window.  
This is known as 'cyclic shift' in the paper.



# Retrospect of Transformer and RNN

