# Recurrent Neural Network

Deep Learning Course
Sharif University of technology
Fall 2020
E. Fatemizadeh

# Recurrent Neural Network

- Till now: Static mapping from input to output
  - Feedforward calculation
- RNN: Temporal Dynamic
  - Feedback Connection
- Good for sequential data analysis and prediction

# Recurrent Neural Network

- Application:
  - Time series prediction/forecasting
  - Natural Language Processing
  - Speech/Signal Processing
  - Question/Answering Machine
  - Image Captioning
  - ….

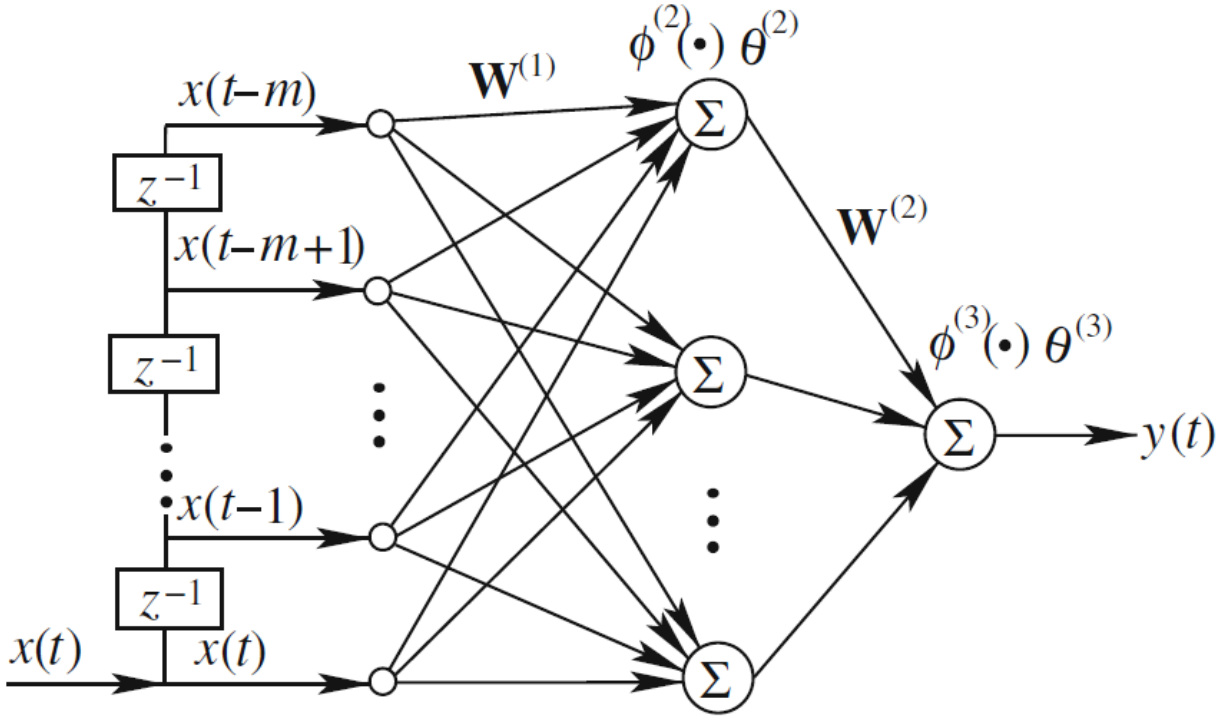
# Recurrent Neural Network

- Again an old fashion idea!
  - Start with a nonlinear dynamical system
- Nonlinear AutoRegressive eXogenous model (NARX):
  - $y$: output
  - $u$: input
  - $q$: shift (Lag) operator

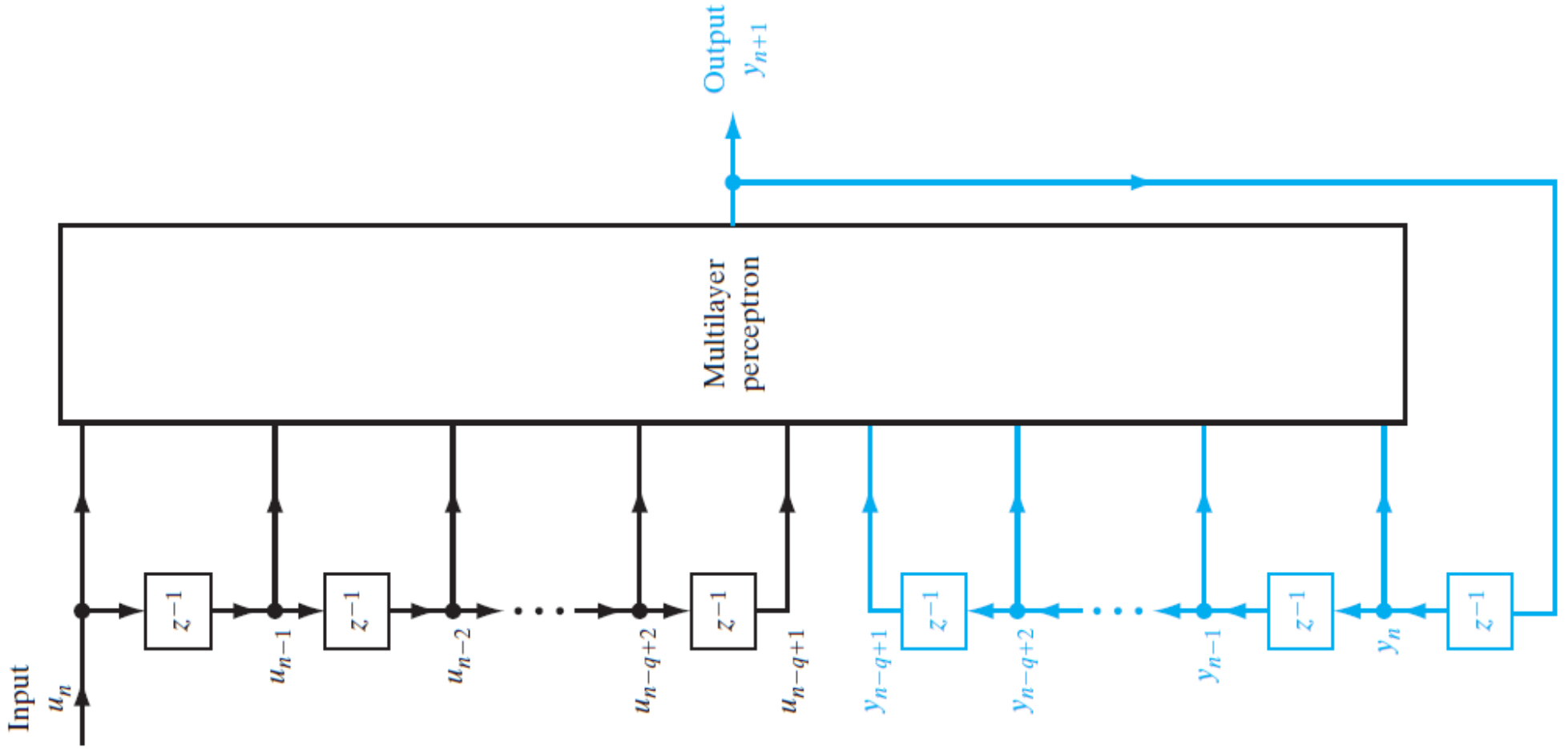
$$y_{n+1} = F(y_n, \ldots, y_{n-q+1}; u_n, \ldots, u_{n-q+1})$$

# **NoT** Recurrent Neural Network

› Time-Delayed Neural Network, TDNN (for dynamic system):



# NARX


Output
$y_{n+1}$
Multilayer perceptron
Input
$u_n$
$z^{-1}$
$u_{n-1}$
$u_{n-2}$
$u_{n-q+2}$
$u_{n-q+1}$
$y_{n-q+1}$
$y_{n-q+2}$
$y_{n-1}$
$y_n$

# Recurrent Neural Network

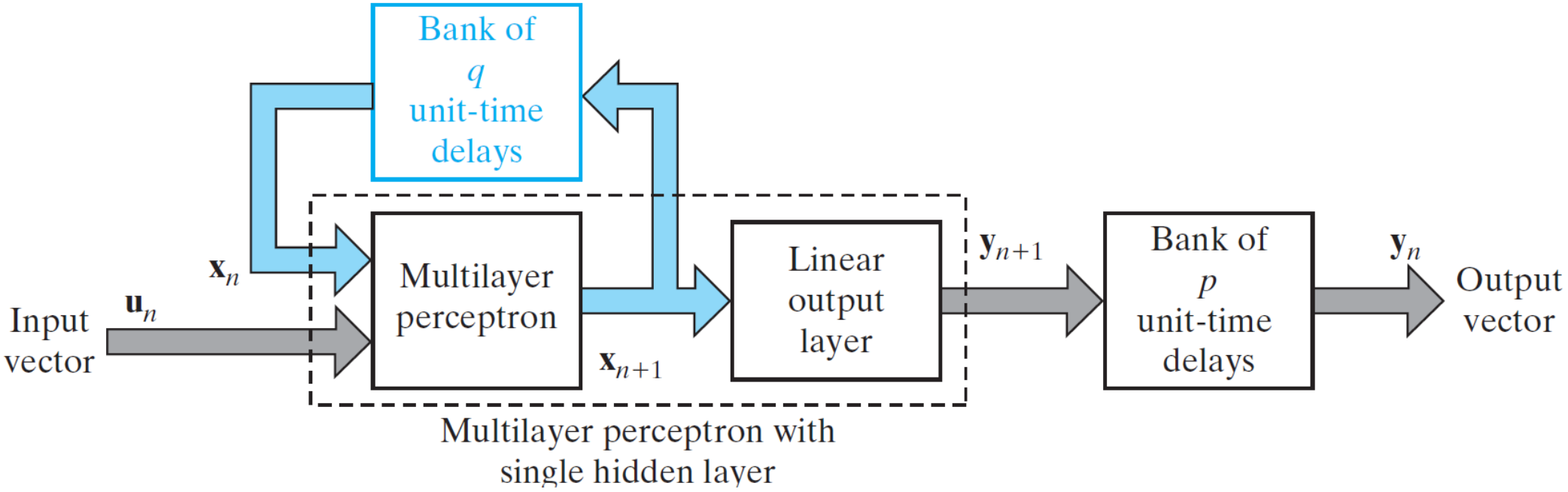
- State-Space Model:
  - $y$: output
  - $u$: input
  - $x$: state (output of hidden neuron)

$$\mathbf{x}_{n+1} = \mathbf{a}(\mathbf{x}_n, \mathbf{u}_n)$$

$$\mathbf{y}_n = \mathbf{B}\mathbf{x}_n$$

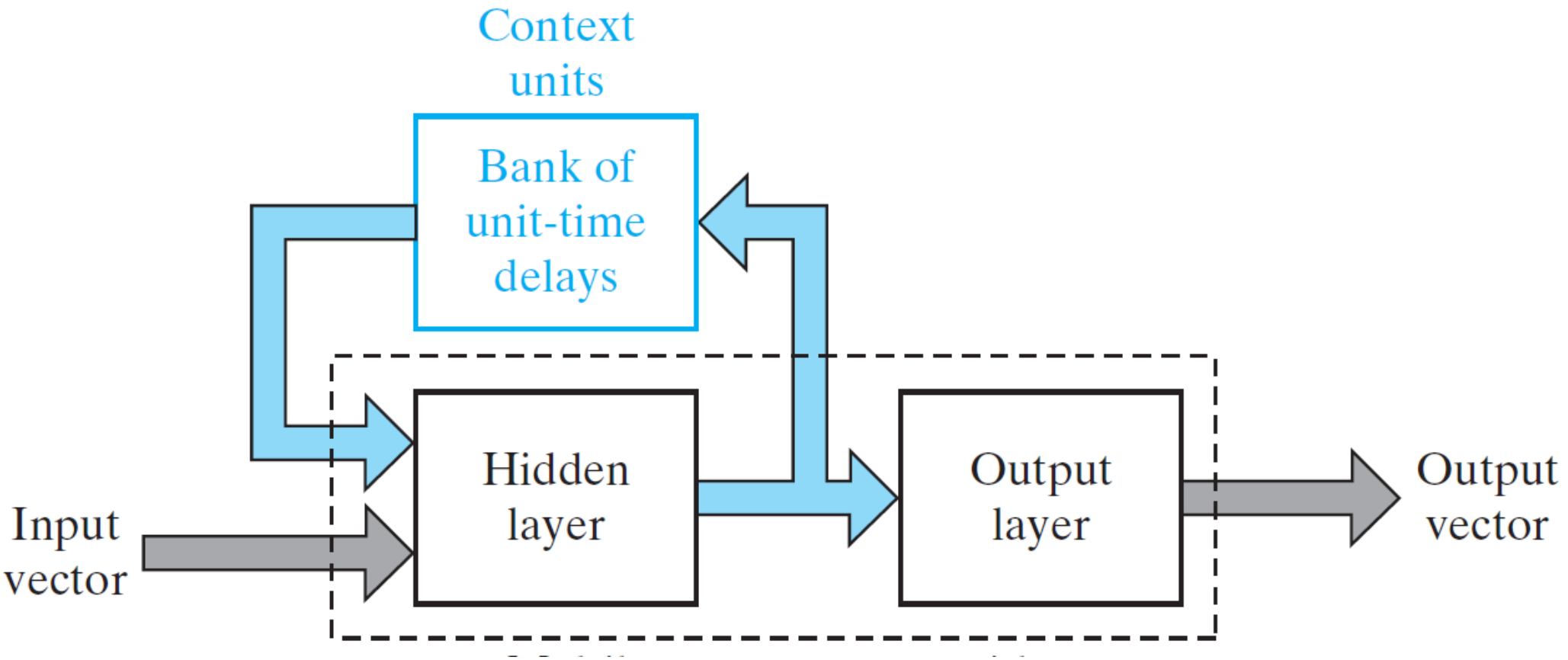
# Recurrent Neural Network

› State-Space Model:



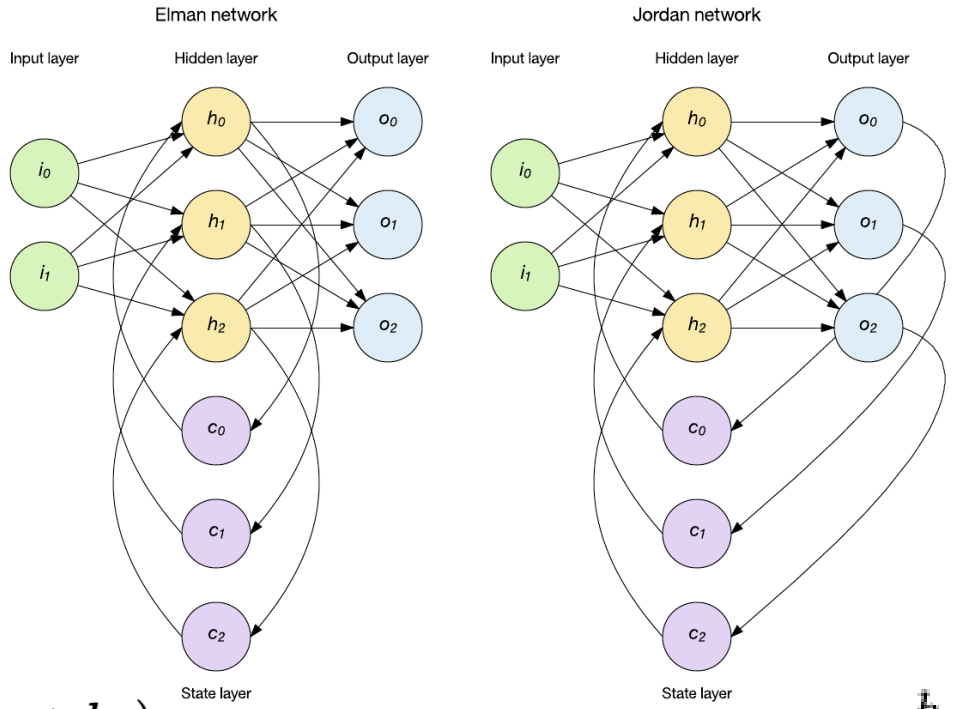
# Recurrent Neural Network

› Simple Recurrent Network (SRN):



# Recurrent Neural Network

› Elman (1990) and Jordan (1997) Network:

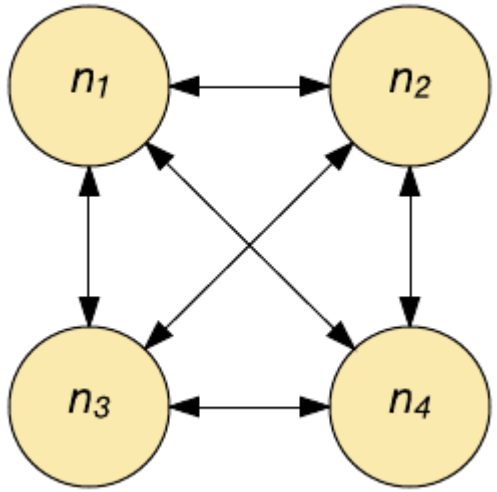



$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

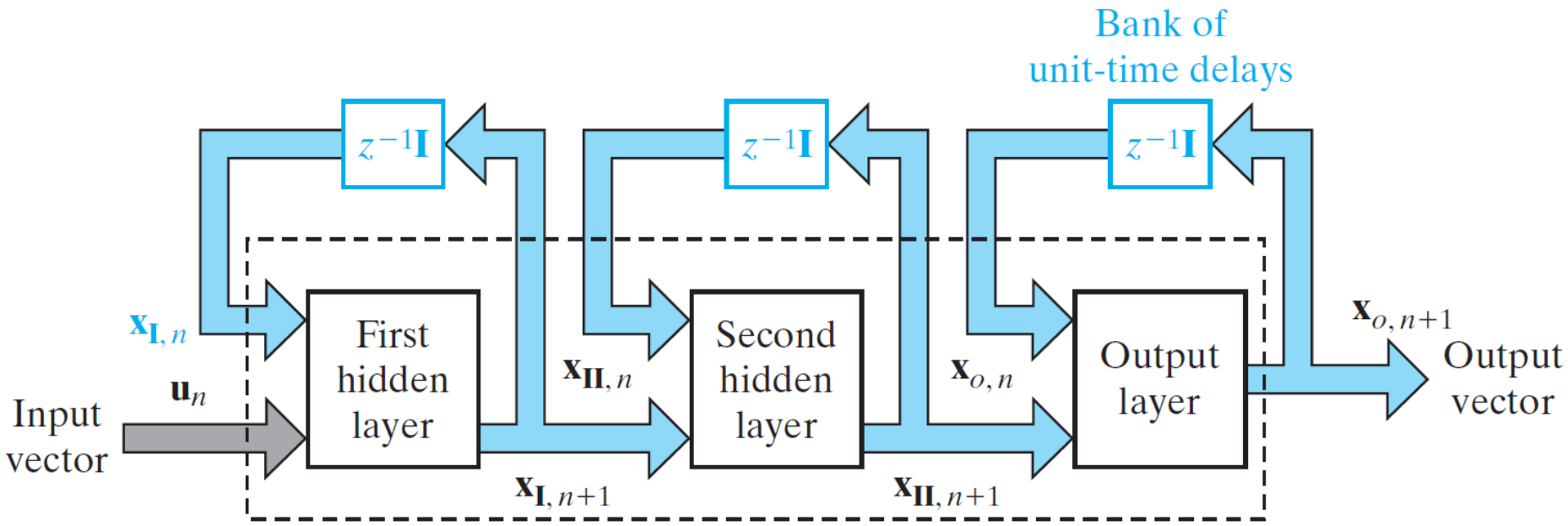
# Recurrent Neural Network

- Hopfield Network (1982):
  - Symmetric Weights
  - Working with initial condition

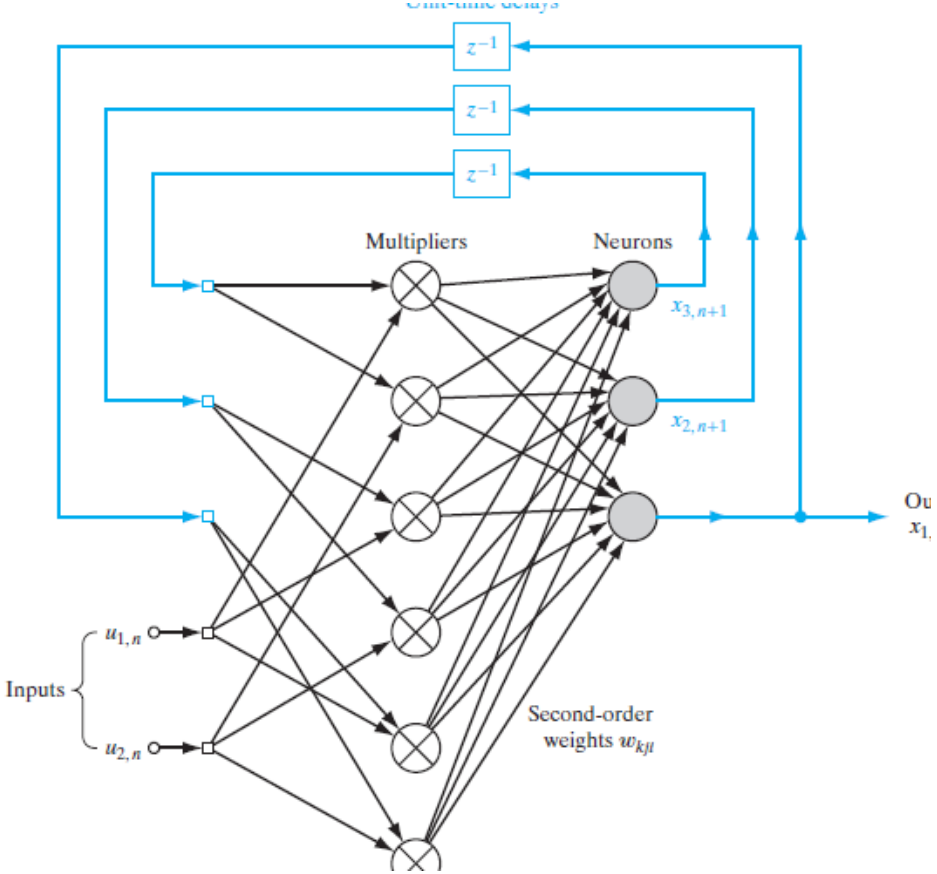


# Recurrent Neural Network

› Recurrent Multilayer Perceptron (RMLP):



# 2nd Order Recurrent NN



# Recurrent Neural Network

› Universal Approximation Theorem (Dynamic Systems):

$$\mathbf{x}_{n+1} = \boldsymbol{\phi}(\mathbf{W}_a \mathbf{x}_n + \mathbf{W}_b \mathbf{u}_n)$$

$$\mathbf{y}_n = \mathbf{W}_c \mathbf{x}_n$$

*Any nonlinear dynamic system may be approximated by a recurrent neural network to any desired degree of accuracy and with no restrictions imposed on the compactness of the state space, provided that the network is equipped with an adequate number of hidden neurons.*

› Dynamical system identification by recurrent multilayer perceptrons, 1993

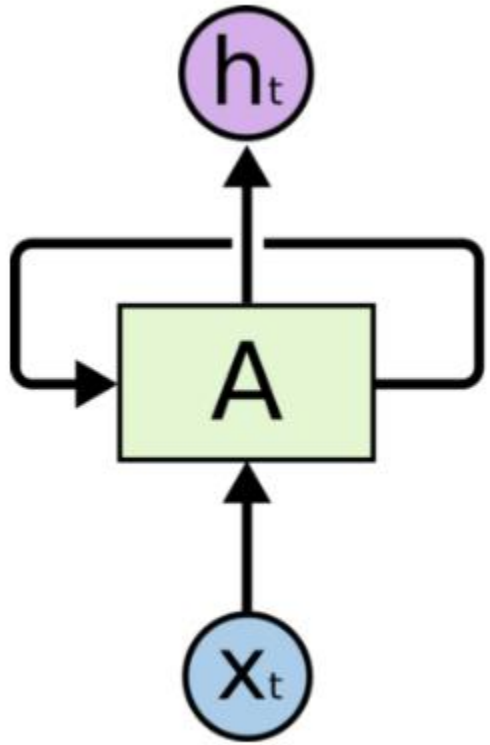
# Recurrent Neural Network

- Type of RNNs (Sequence or Single input/output):
  - One-to-One
  - One-to-Many:
    - Image Captioning (image → sequence of words)
  - Many-to-One:
    - Sentiment/Emotion Classification (sequence of words → Emotion)
  - Many-to-Many:
    - Machine Translation (sequence of words → sequence of words )

# Recurrent Neural Network

- RNN Goal:
  - Predict a vector at some time steps
    - $X_t$: input (vector) at some time step
    - Same function, $f_W$, in each time step
    - Same Parameter, $W$, in each time step

$$h_t = f_W(h_{t-1}, x_t)$$



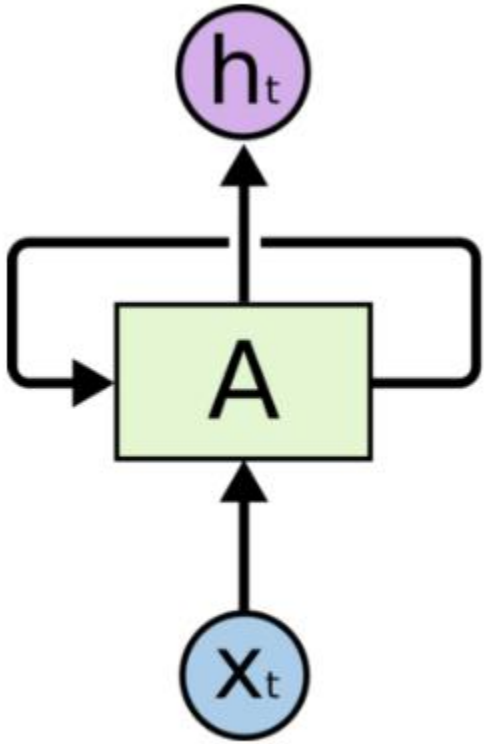
# Recurrent Neural Network

› (Simple/Vanilla/Elman) RNN:
  – State: A single hidden vector (layer)

$$h_t = f_W(h_{t-1}, x_t)$$

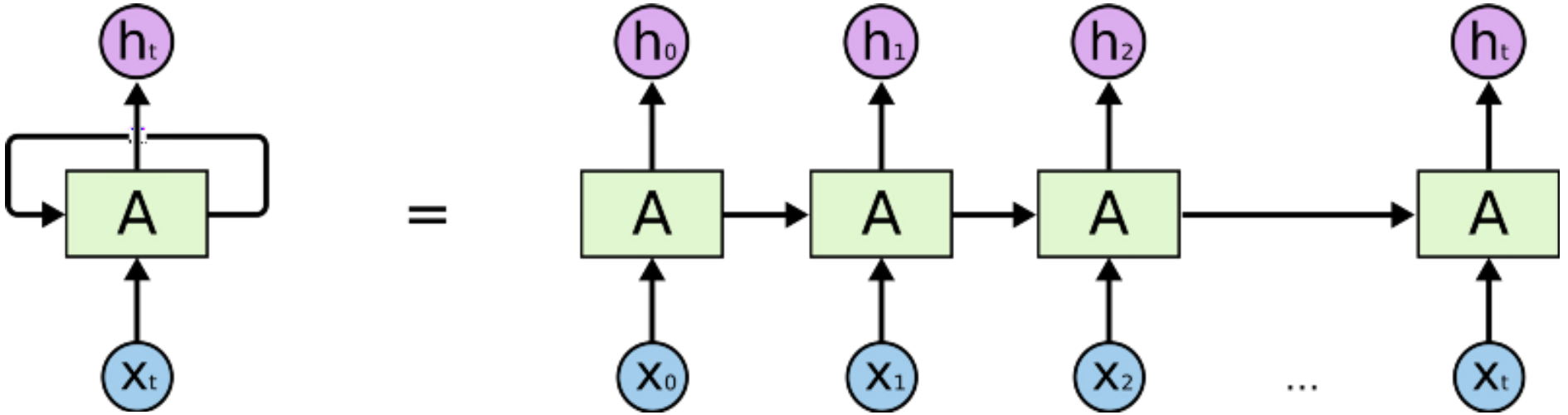
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$



# Recurrent Neural Network

› Unfolding/Unrolling Computational Graph (Simple RNN):




$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

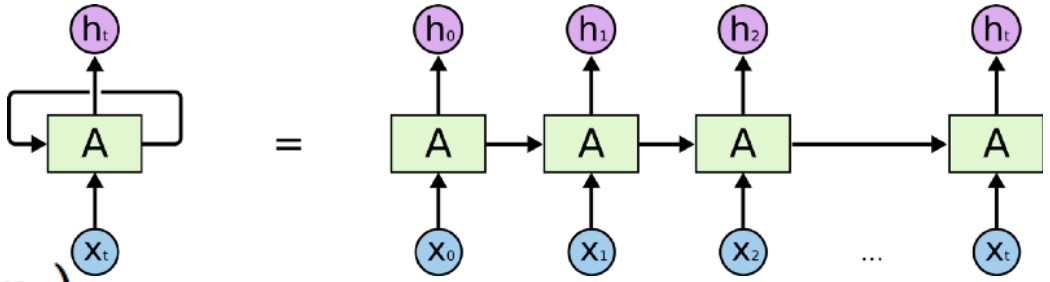
# Recurrent Neural Network

› Computational:

Let $a_t$ represent the ouput from the previous node

$$a_t = f(h_{t-1}, x_t)$$

$$g(x) = \tanh x$$



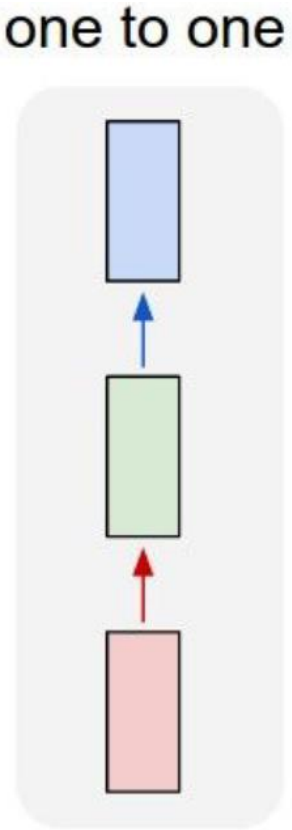

$$a_t = g(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

$$a_t = \tanh W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t$$

$$h_t = W_{hy} \cdot a_t$$

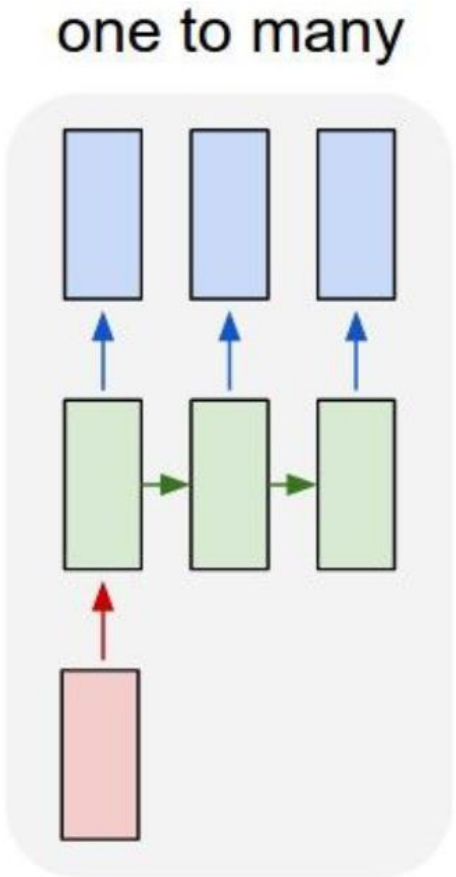
# Recurrent Neural Network

› One-to-One



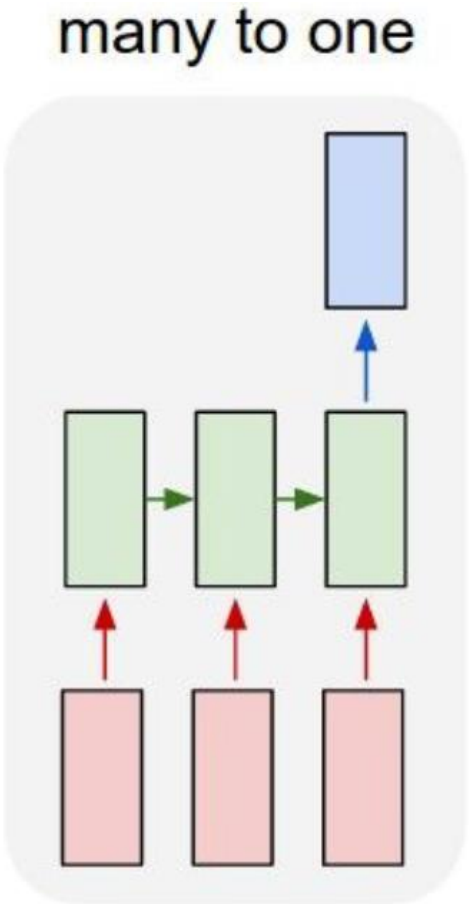
# Recurrent Neural Network

- One-to-Many
- Image Captioning
- Input: Image
- Output: Sequence of words



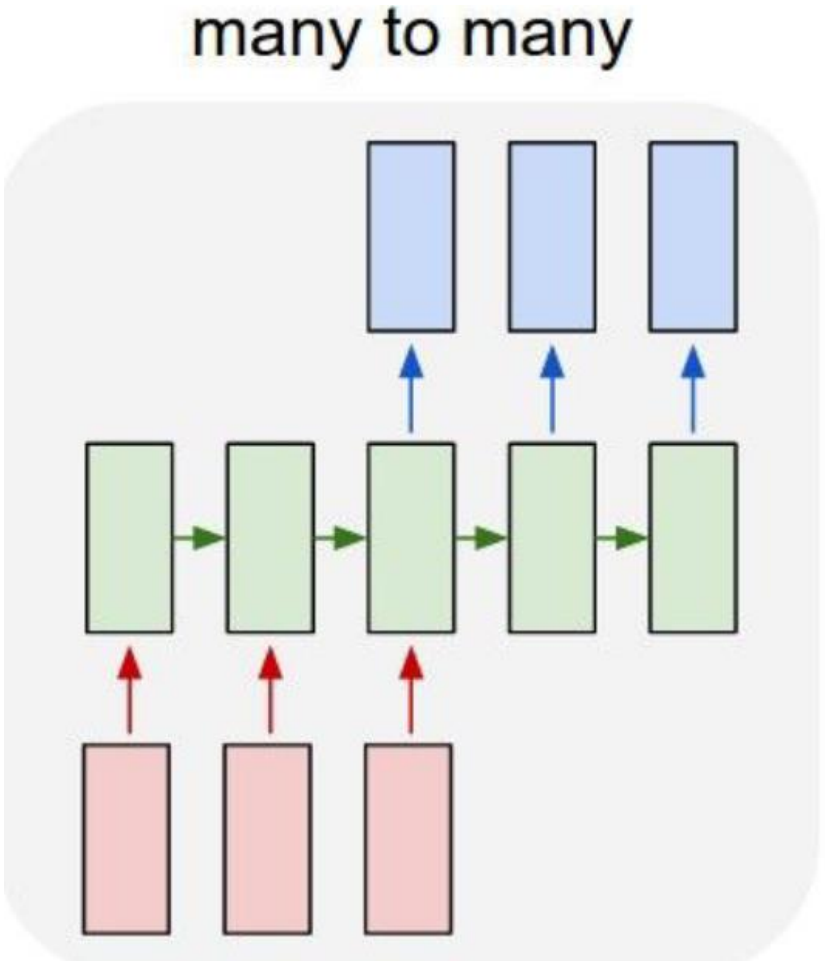
# Recurrent Neural Network

- Many-to-One
- Emotion/Sentiment Classification
- Input: Sequence of words
- Output: Emotion/Sentiment



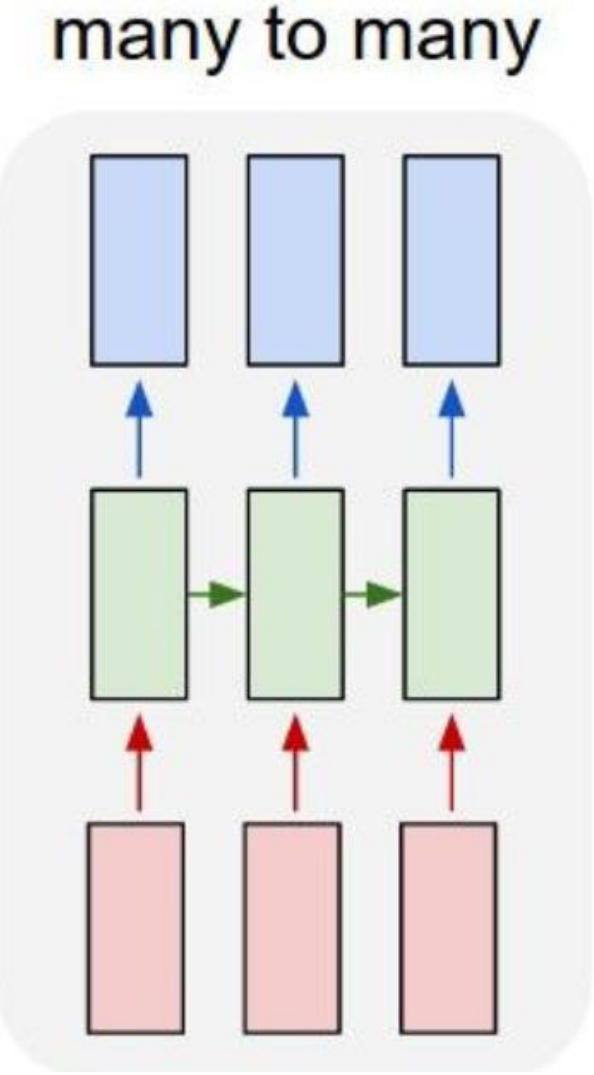
# Recurrent Neural Network

- Many-to-Many
- Machine Translator
- Input: Sequence of words
- Output: Sequence of words



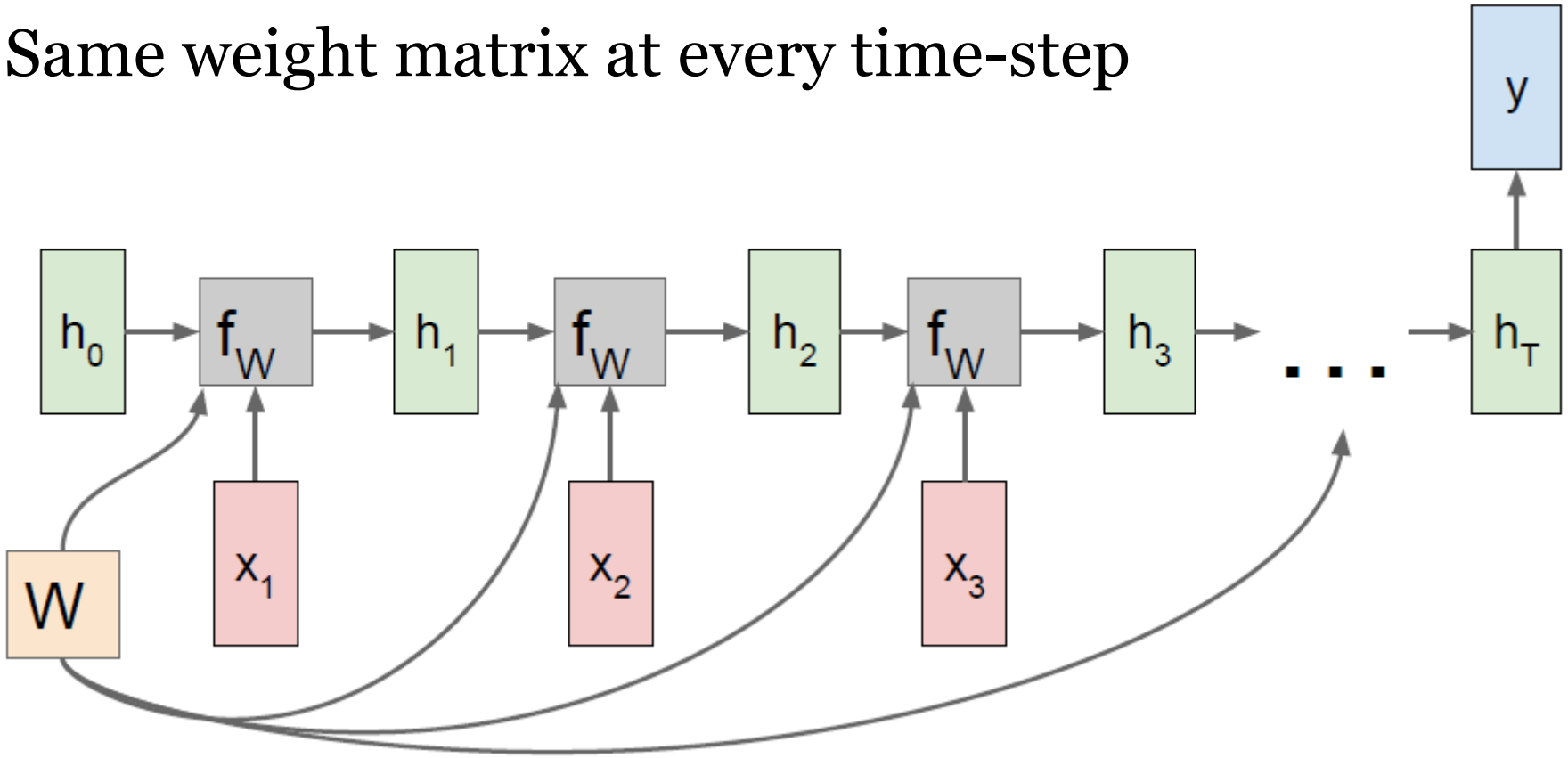
# Recurrent Neural Network

- Many-to-Many (1)
- Video Classification (frame by frame)
- Input: Sequence of frames
- Output: Sequence of label

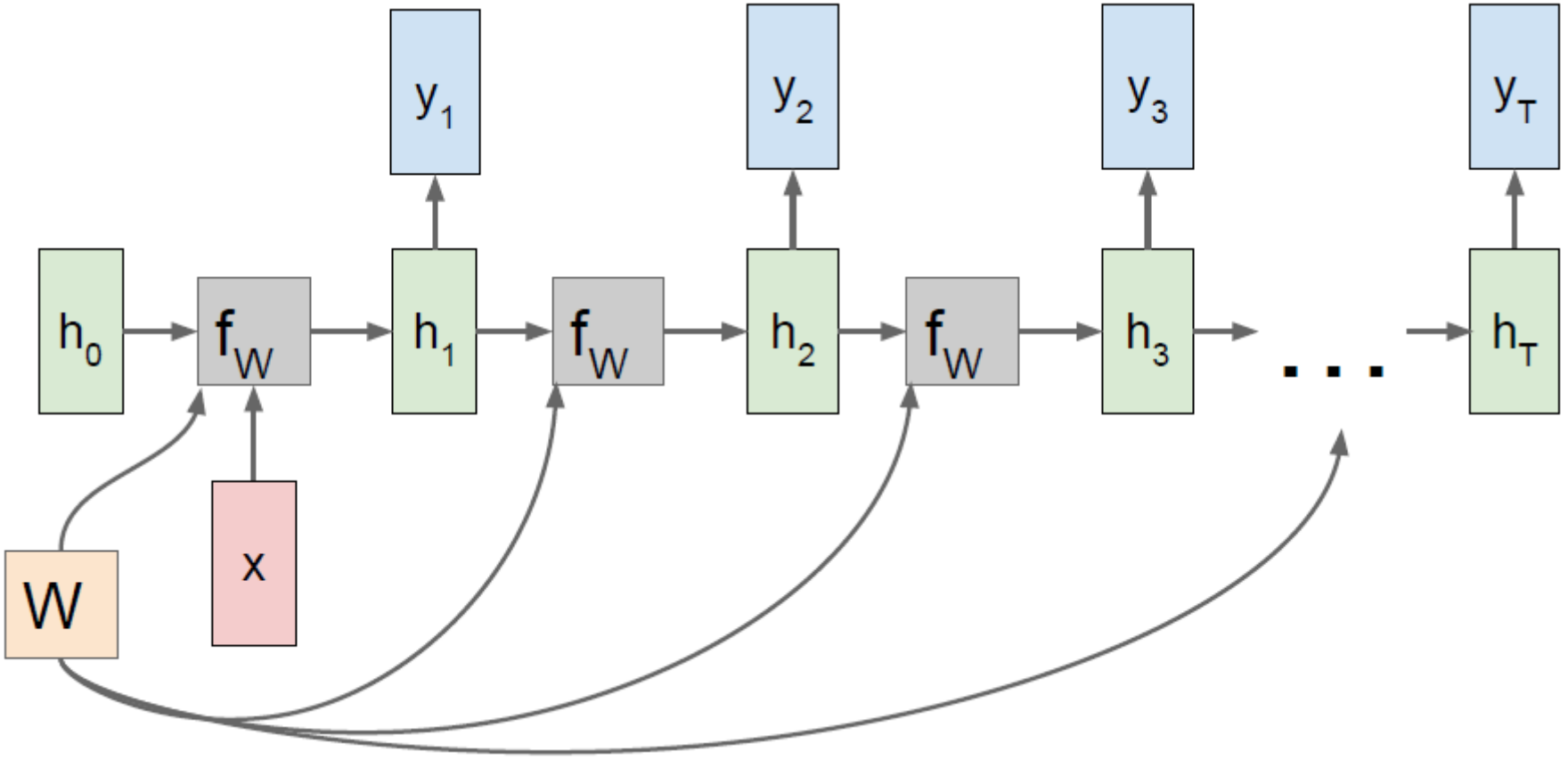


# Computational Graph - Many-to-One

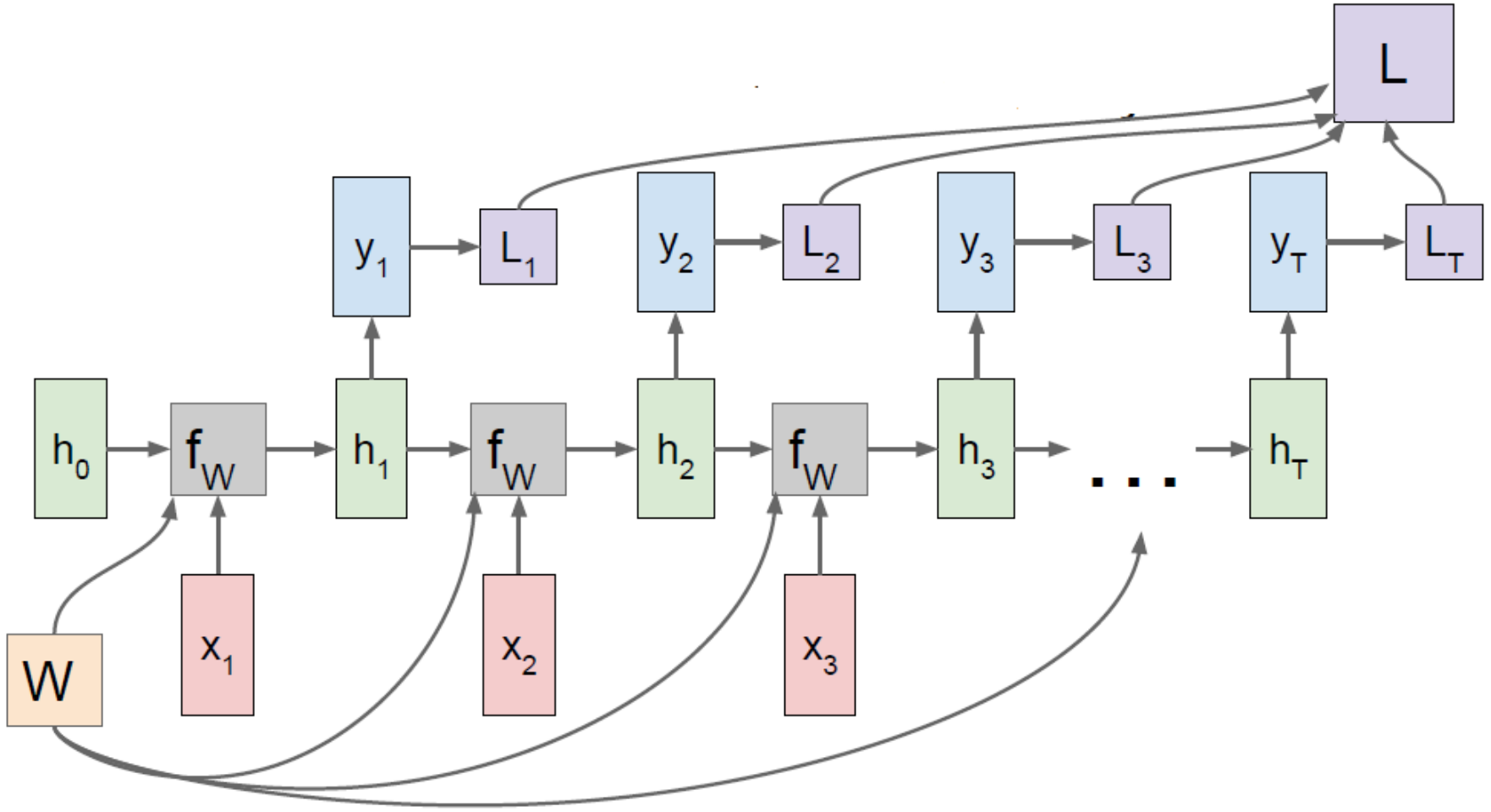
› Same weight matrix at every time-step



# Computational Graph: **One-to-Many**

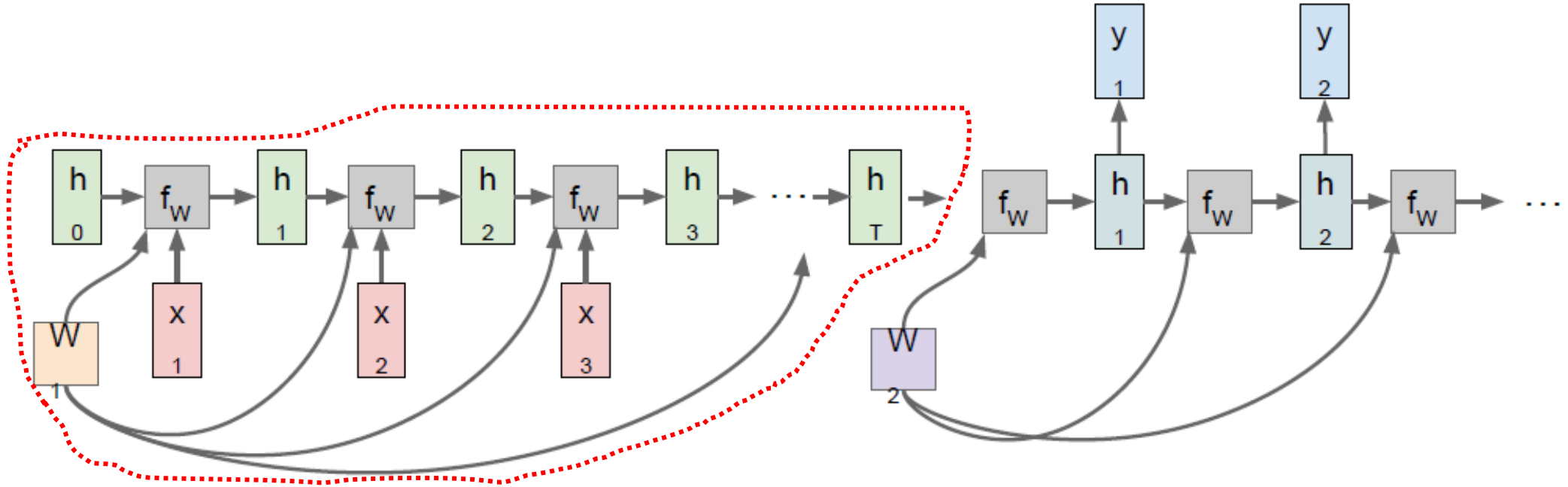


# Recurrent Neural Network: **Many-to-Many**



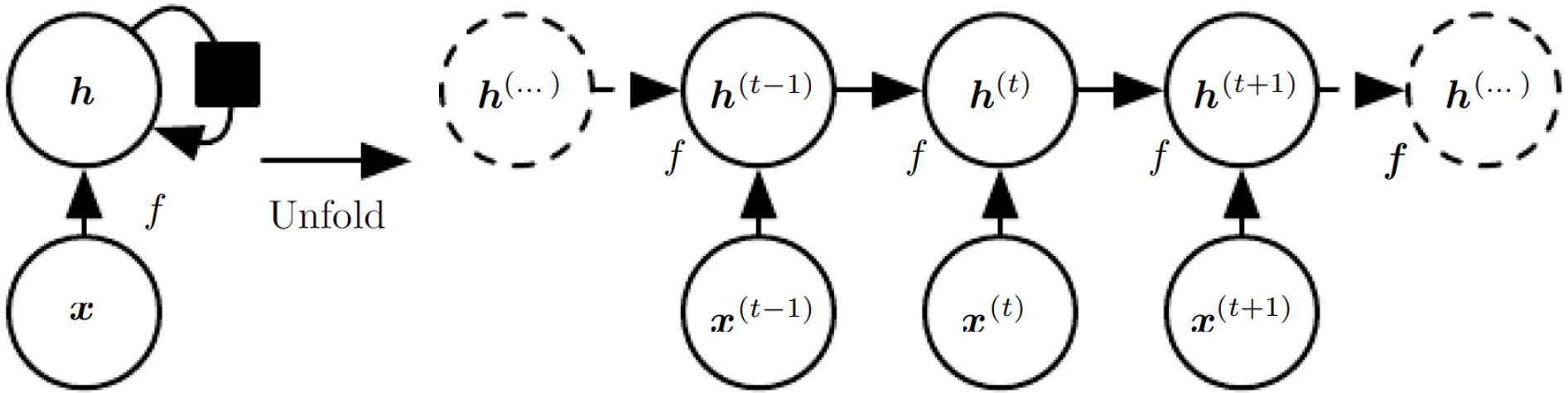
# Recurrent Neural Network

› Sequence-to-Sequence (Many-to-One + One-to-Many):
  – Many-to-One: Encode input to SINGLE vector
  – One-to-Many: Decode SINGLE vector to output



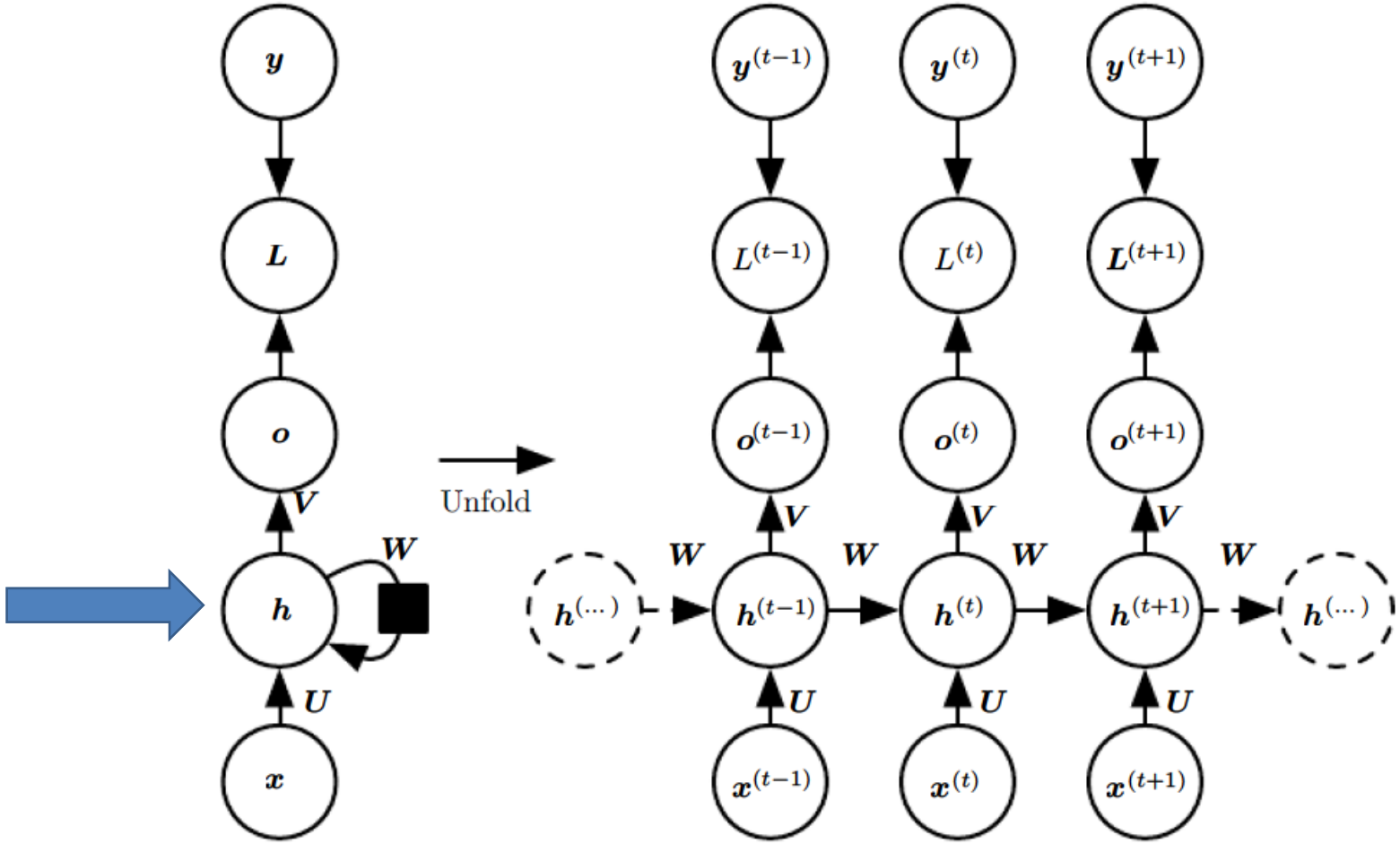
# Recurrent Neural Network

› Goodfellow Illustration of Computational Graph:



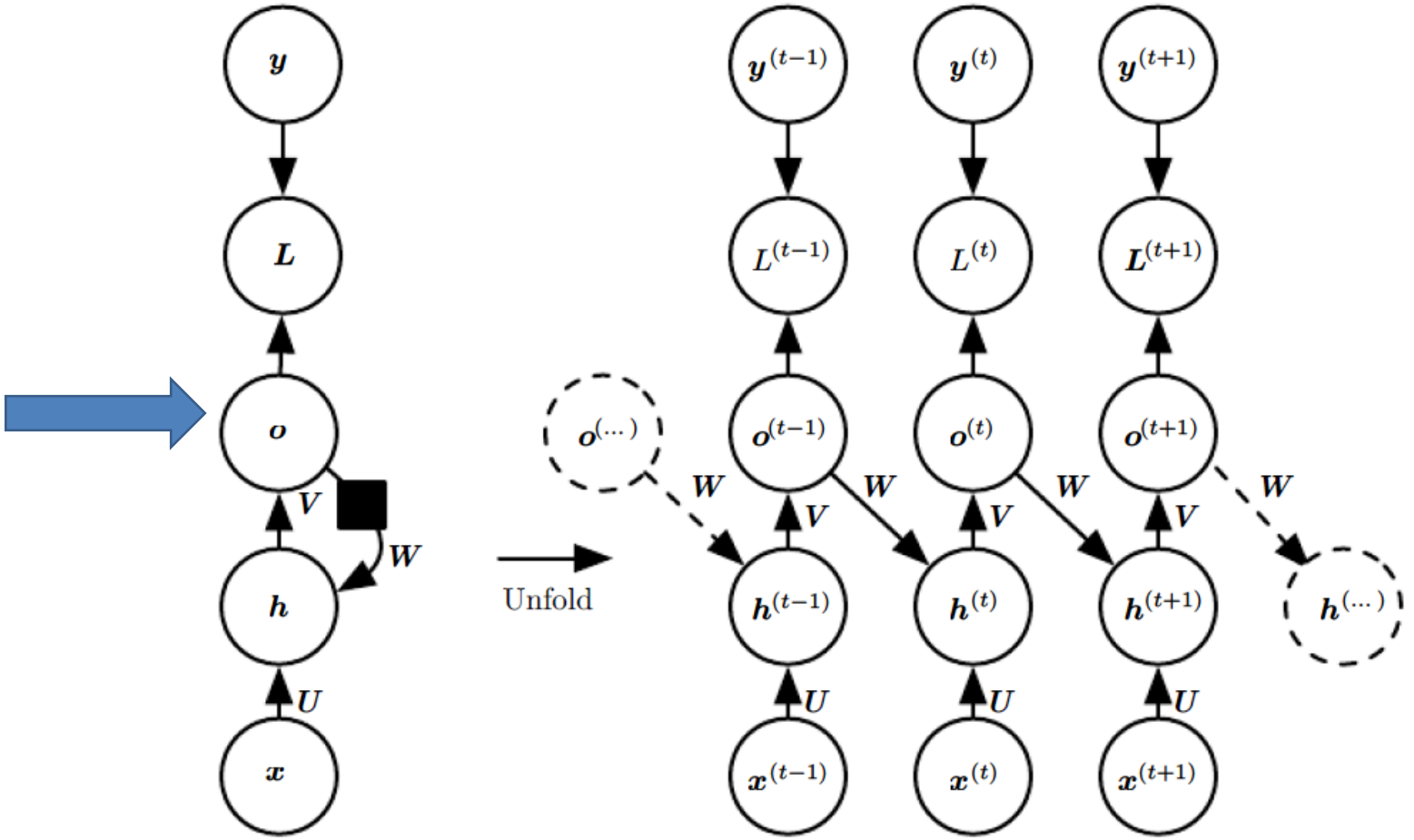
# Recurrent Neural Network

› Computational Graph with Loss and Desired output (#1):



# Recurrent Neural Network

› Teacher Forcing (Less Complex/Easy to Train):



# Recurrent Neural Network

› Computational Graph with Loss and Desired output (#2):

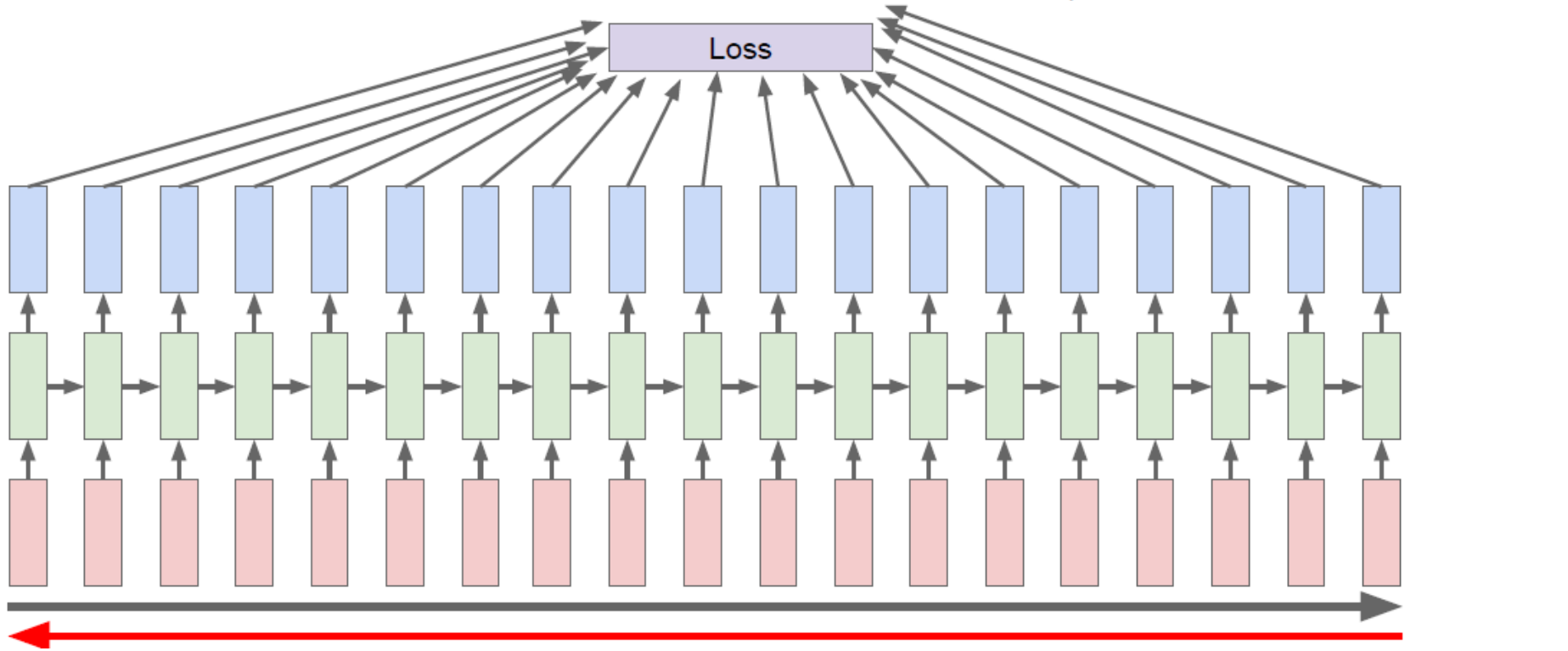
$t = 1$ to $t = \tau$, we apply the following update equations:

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

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

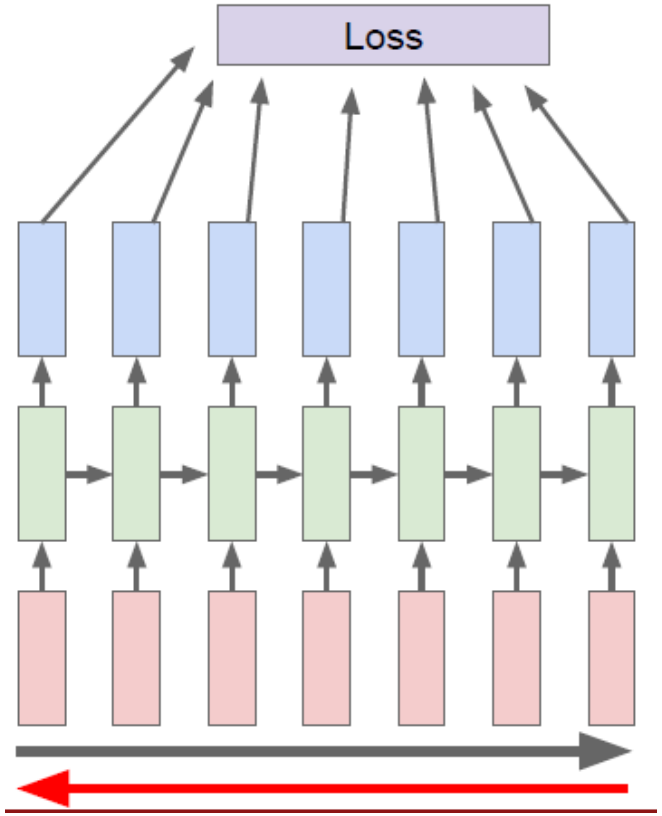
# RNN – Training Strategy

› BPTT (BackPropagation Through Time)
  – Backward/Forward through the whole seq



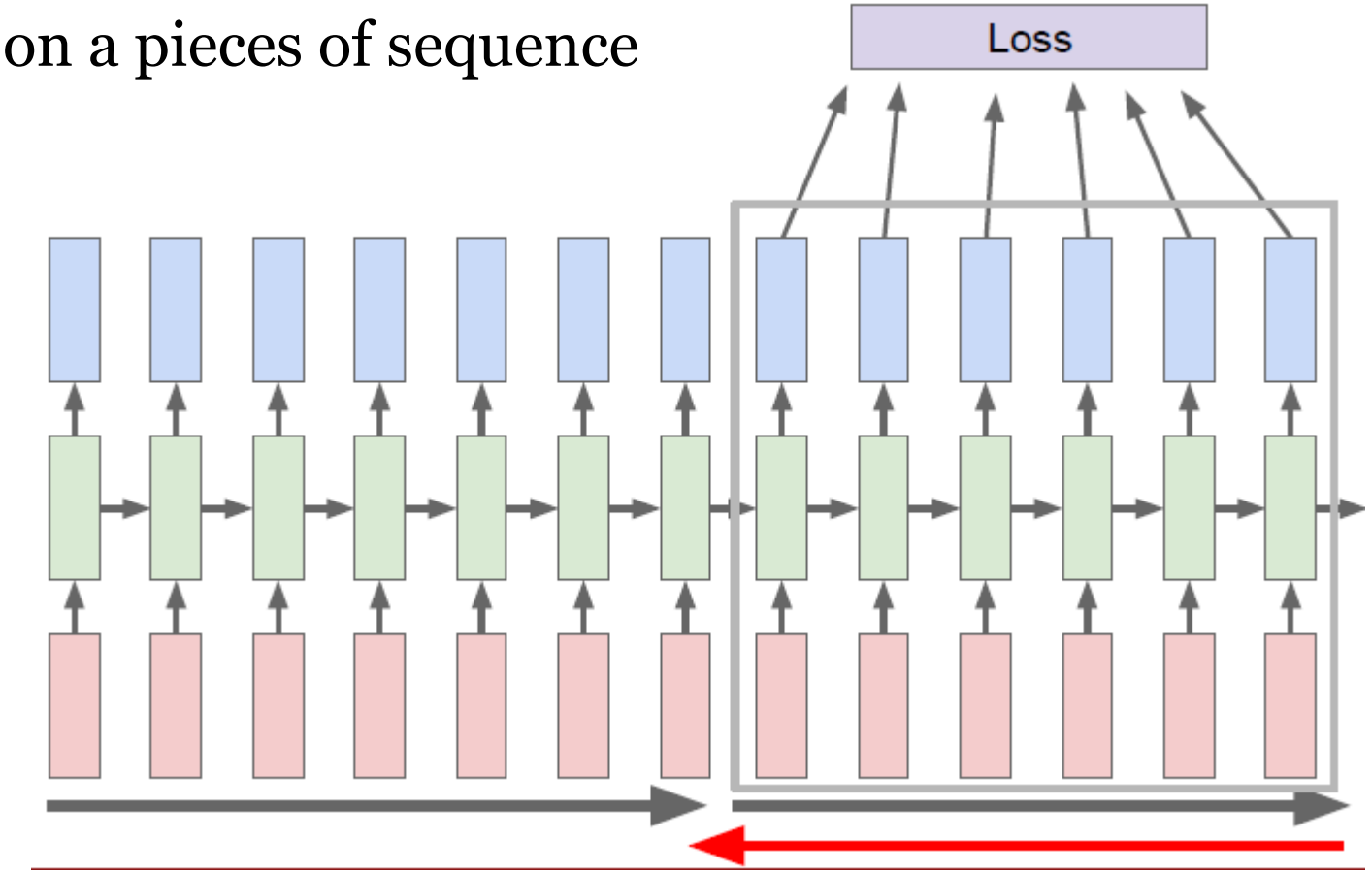
# RNN – Training Strategy

› Truncated BPTT:
  – Backward/Forward  through a pieces of sequence



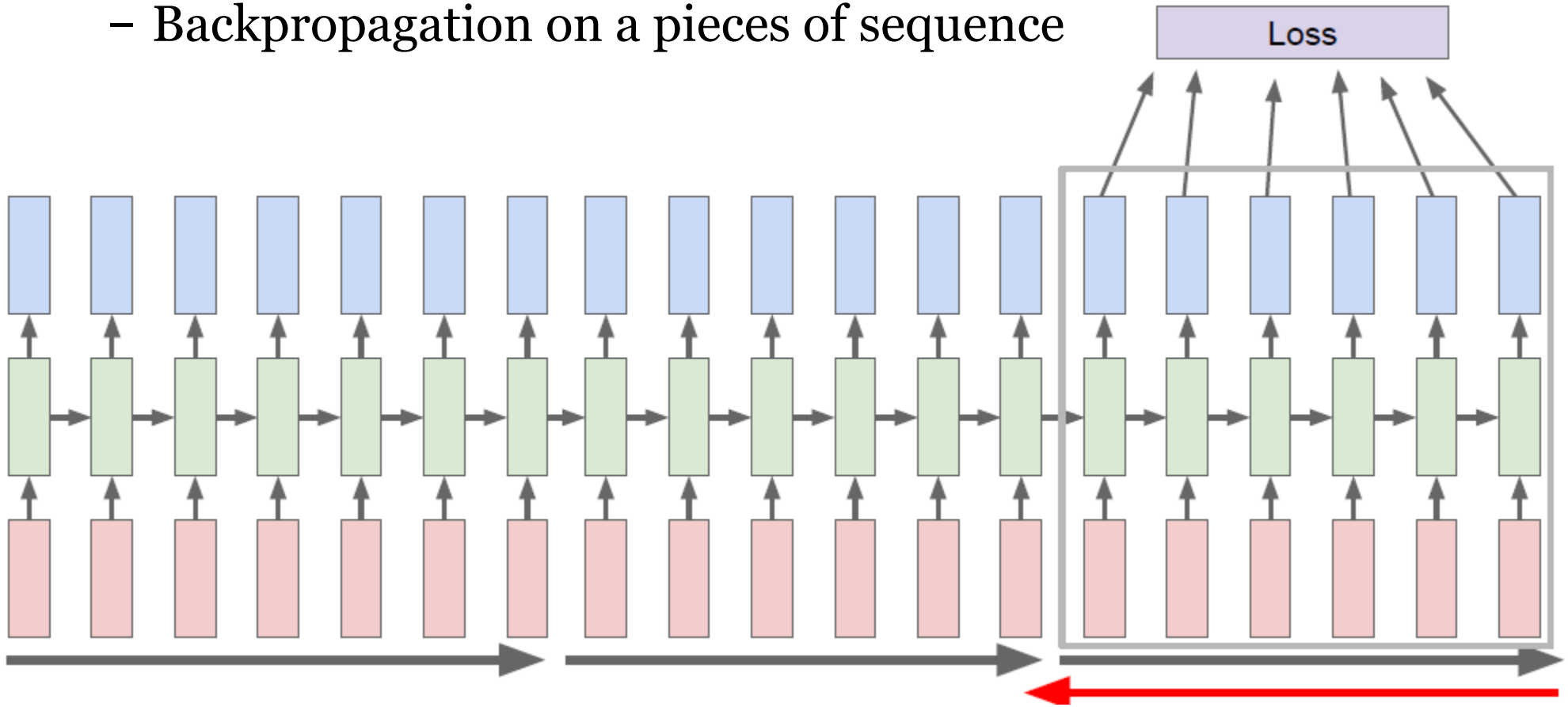
# RNN – Training Strategy

› Truncated BPTT:
  – Backpropagation on a pieces of sequence



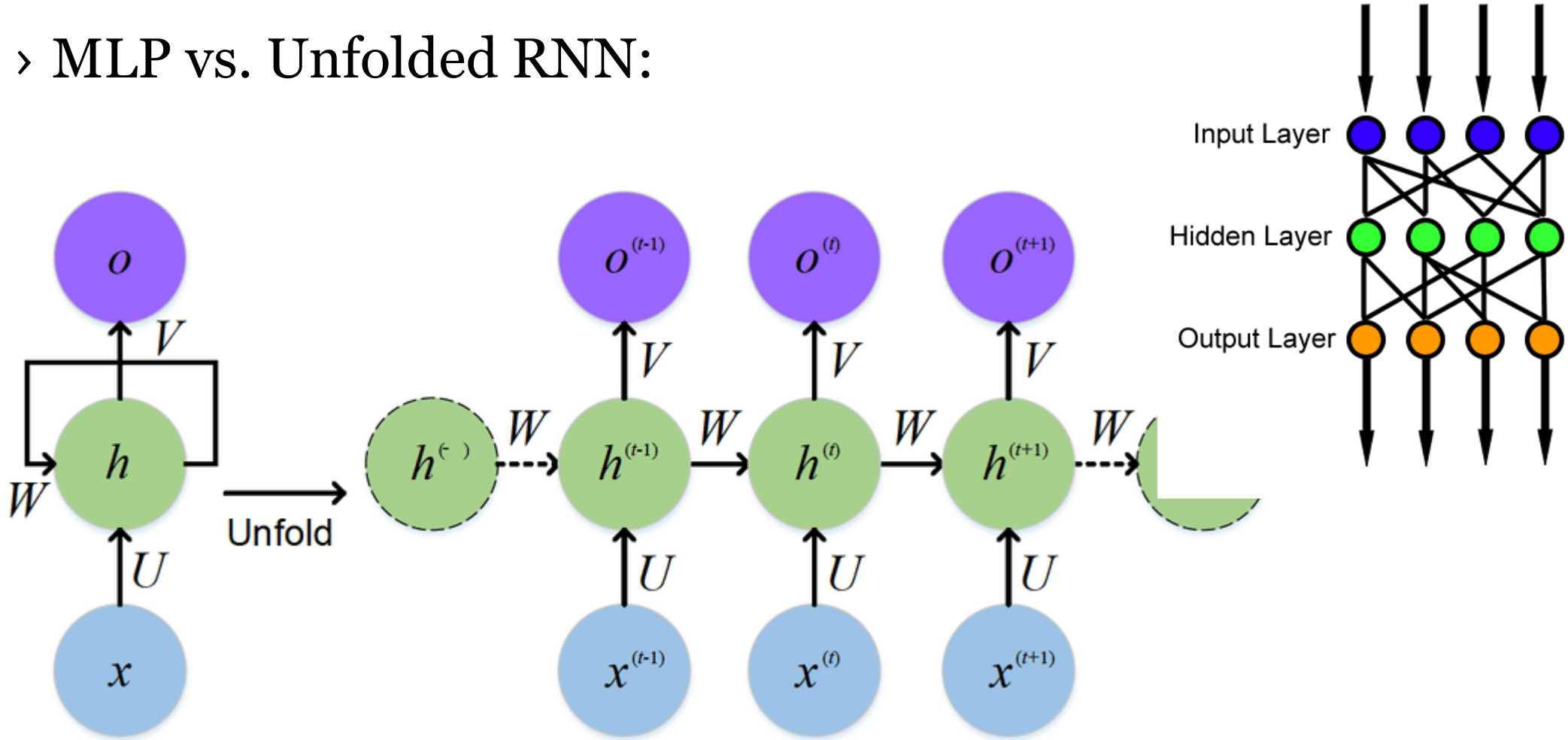
# RNN – Training Strategy

› Truncated BPTT:
  – Backpropagation on a pieces of sequence



# RNN - BPTT (Backpropagation Through Time)

› MLP vs. Unfolded RNN:



# RNN - BPTT (Backpropagation Through Time)

- MLP vs. Unfolded RNN:
  - Same Activity and Shared Weight!
- EBP in MLP (with previous notation):

$$\frac{\partial J_n}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r}\frac{\partial z_{nj}^r}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r}\boldsymbol{y}_n^{r-1}. \quad \delta_{nj}^r := \frac{\partial J_n}{\partial z_{nj}^r}.$$

- Last Layer ($L$):

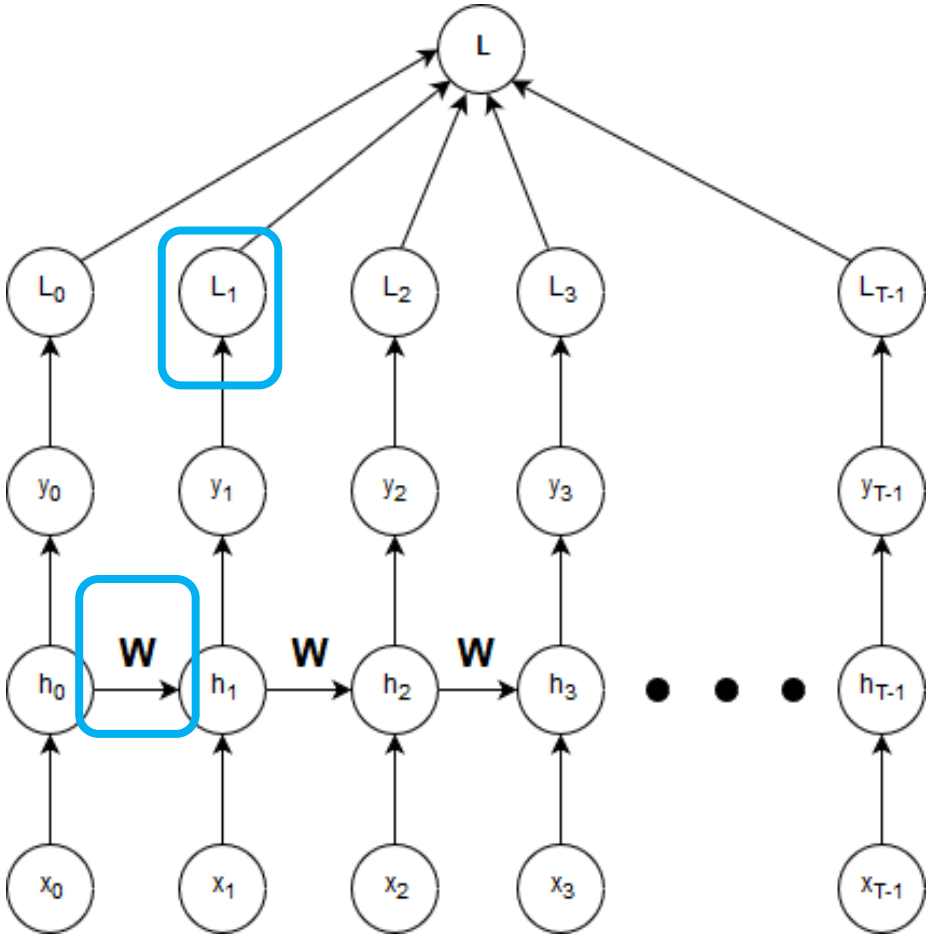
$$\delta_{nj}^L = (\hat{y}_{nj} - y_{nj})f'(z_{nj}^L),$$

- Hidden Layer ($r$, $r$-1 from $r$):!!!
  - Just one backward path!

$$\delta_{nj}^{r-1} = \left(\sum_{k=1}^{k_r} \delta_{nk}^r \theta_{kj}^r\right) f'(z_{nj}^{r-1}),$$

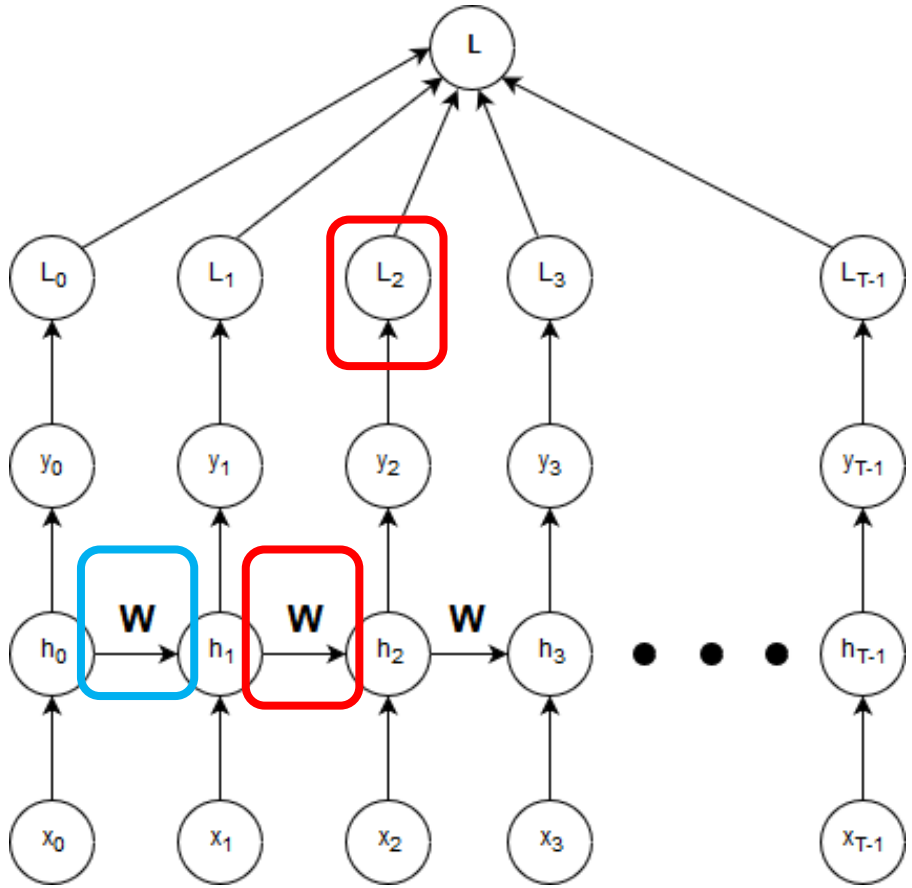
# RNN - BPTT (Backpropagation Through Time)

- How Many path from weight(s) to temporal Loss?
  - Path #1 originate at $h_0$



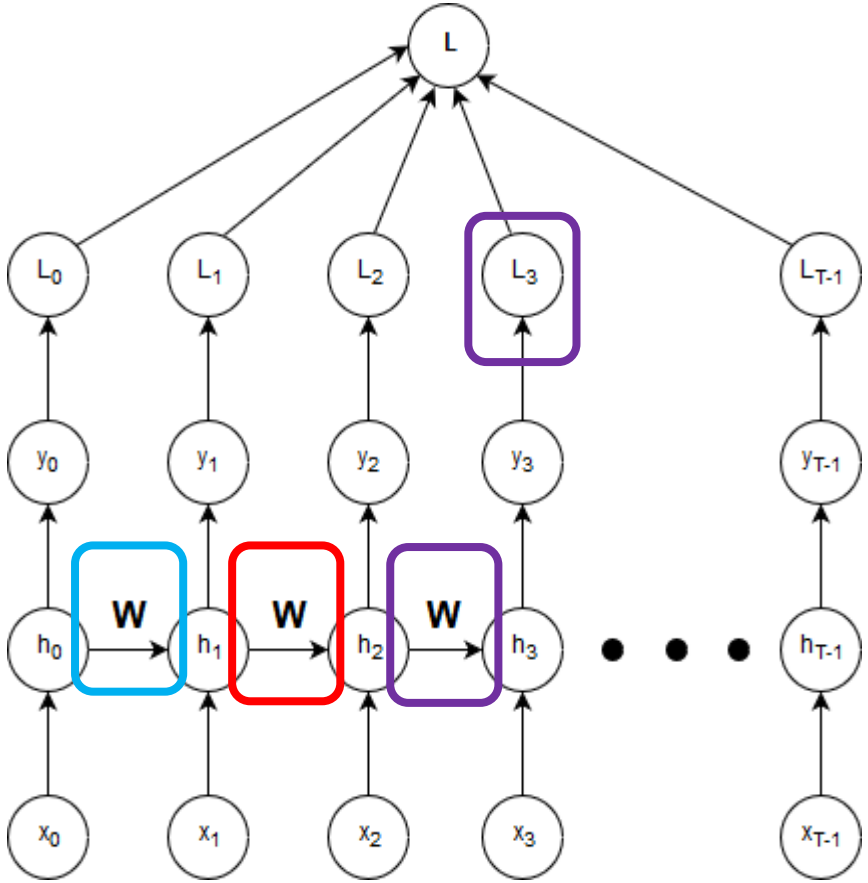
# RNN - BPTT (Backpropagation Through Time)

- How Many path from weight(s) to temporal Loss?
  - Path #2 originate at $h_0$ and $h_1$



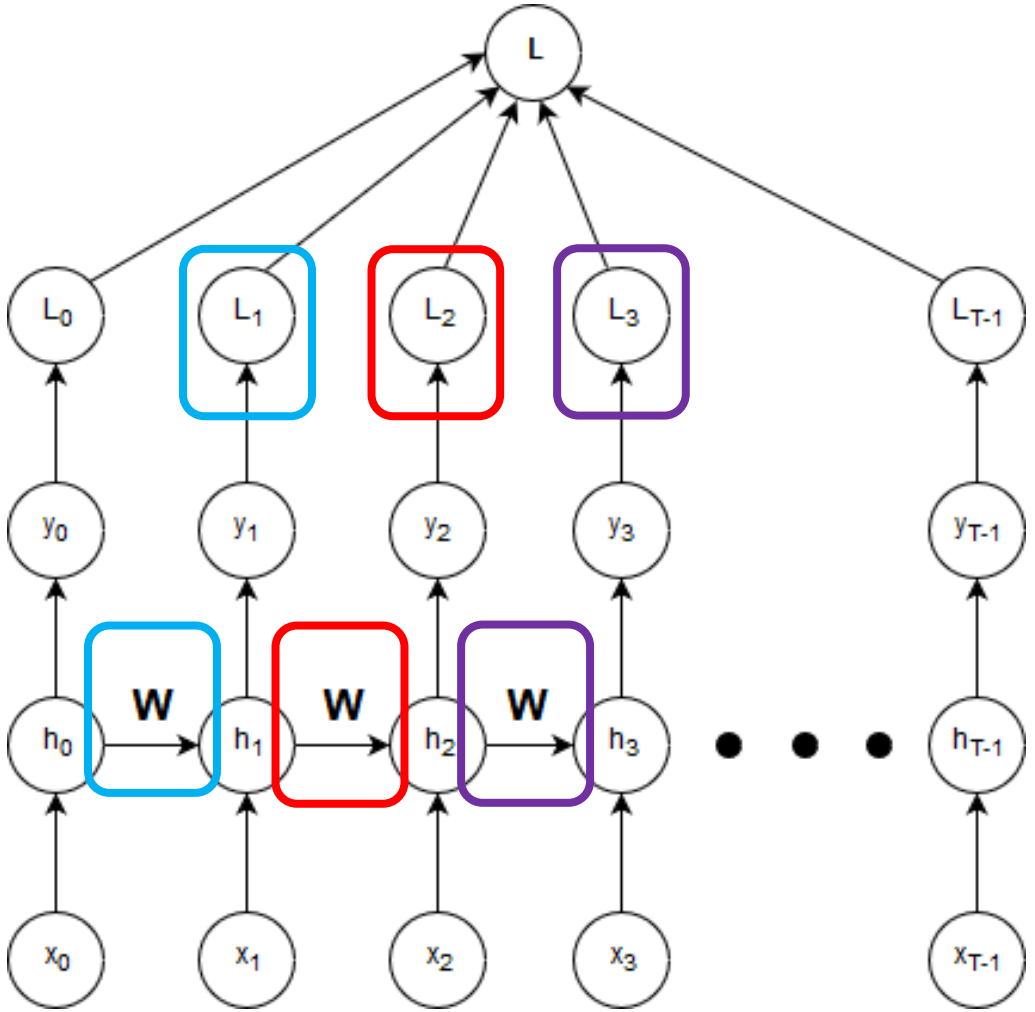
# RNN - BPTT (Backpropagation Through Time)

- How Many path from weight(s) to temporal Loss?
  - Path #3 originate at $h_0$ , $h_1$ and $h_2$



# RNN - BPTT (Backpropagation Through Time)

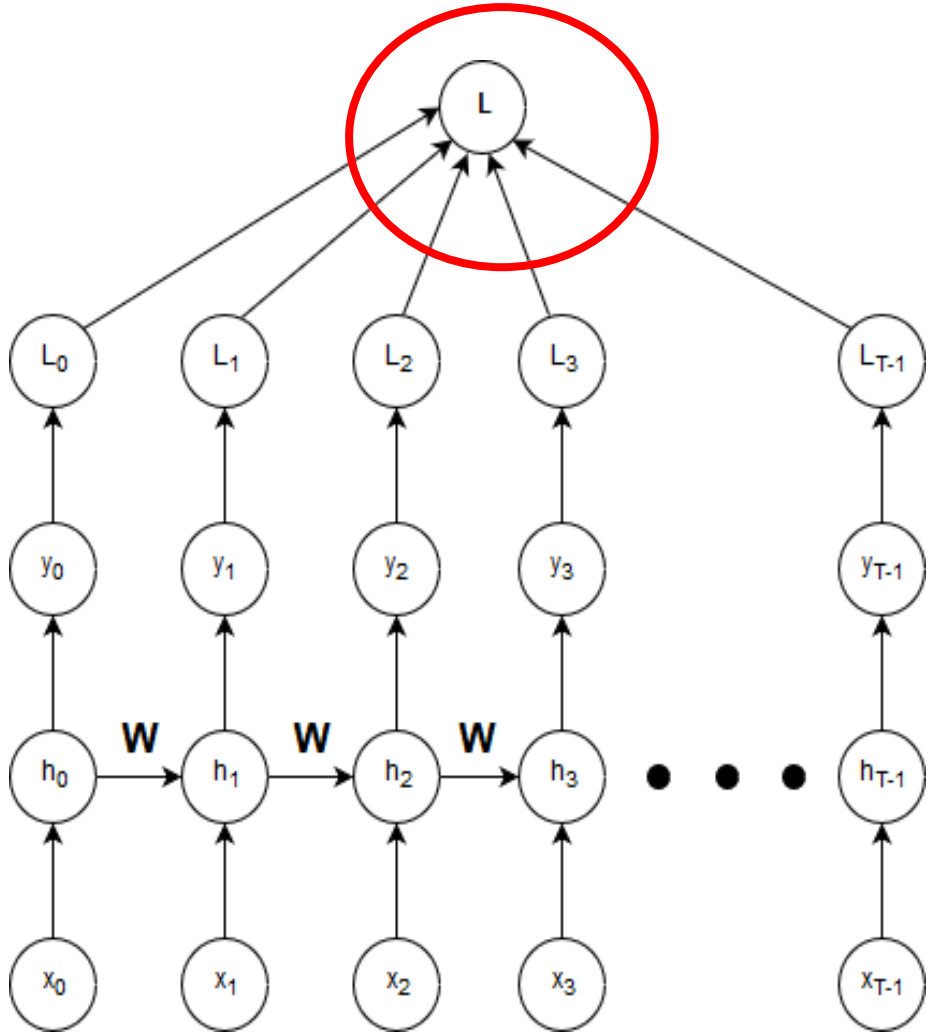
- For EBP, we need: $\partial Loss / \partial W$
- Two Summation:
  - Sum of temporal Loss
  - Sum of gradient path(s)



# RNN - BPTT (Backpropagation Through Time)

- Temporal Loss(es)→ Total Loss:
- *J*: Temporal Index

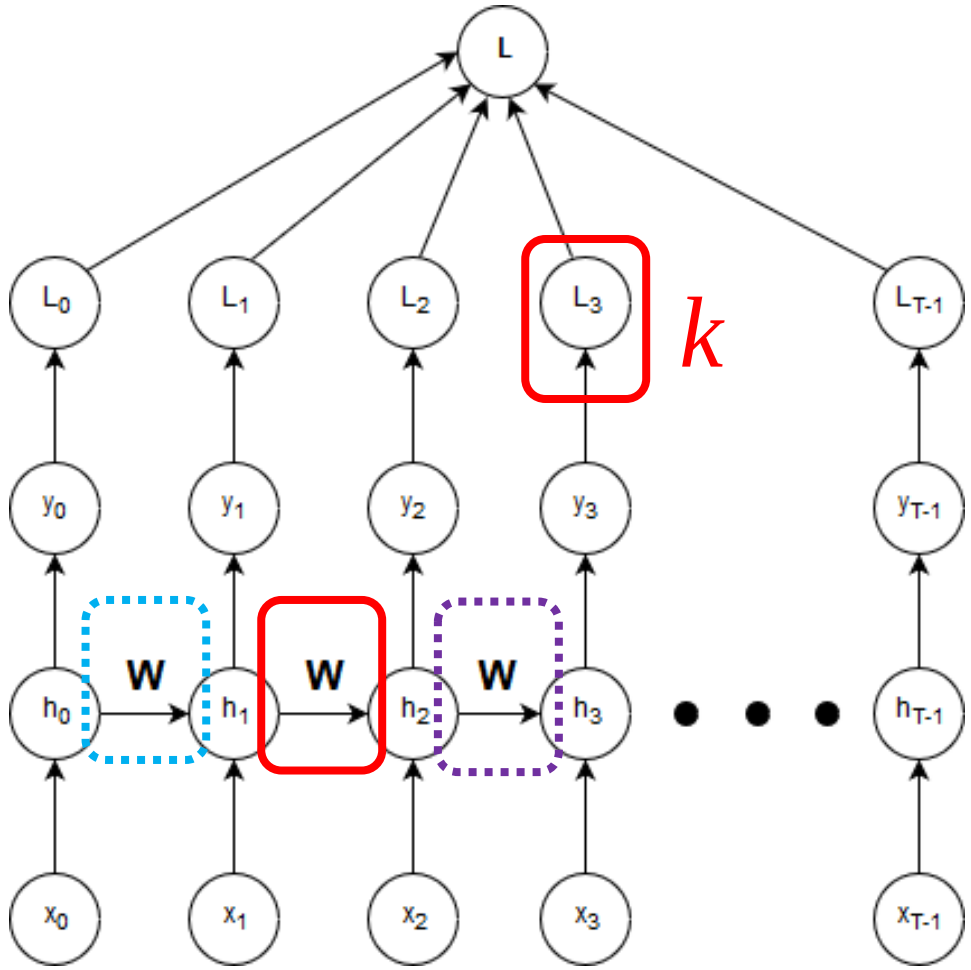
$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}}$$



# RNN - BPTT (Backpropagation Through Time)

- Gradient Summation:
  - Several dependency!
  - Indirect dependency

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$



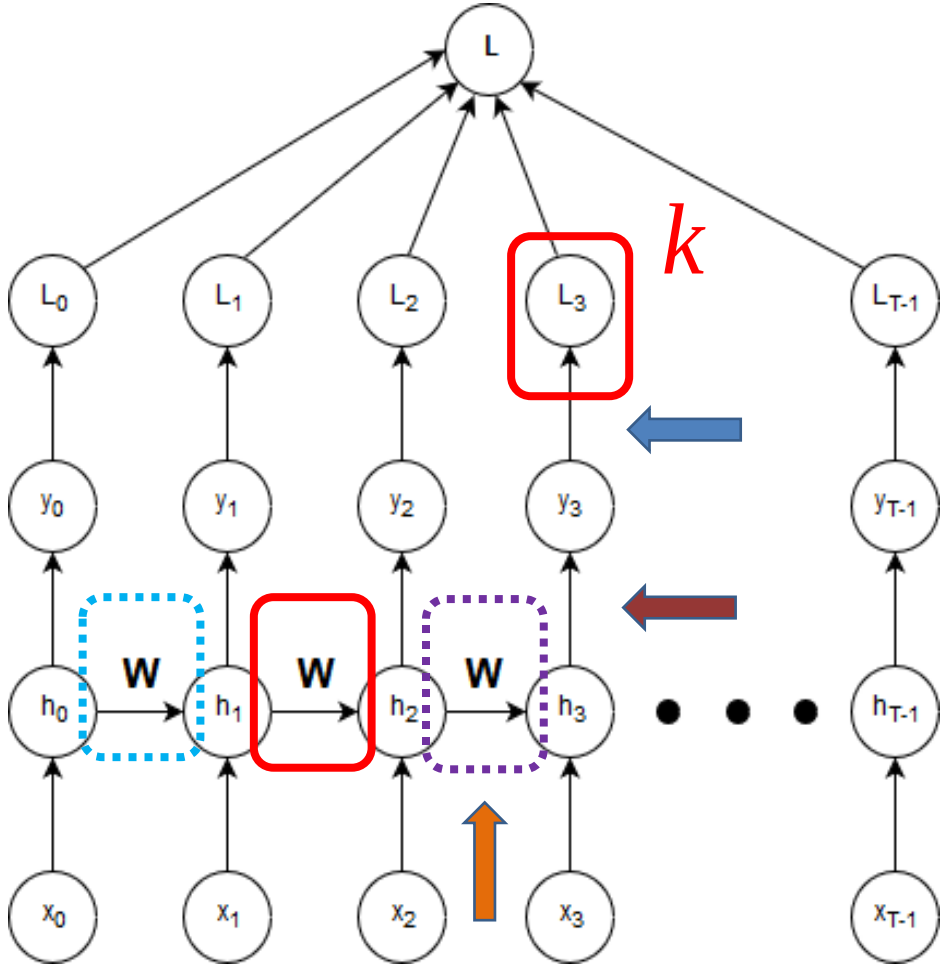
# RNN - BPTT (Backpropagation Through Time)

- Chain Rule:
  - Convert to direct dependency!

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

- Jacobian:

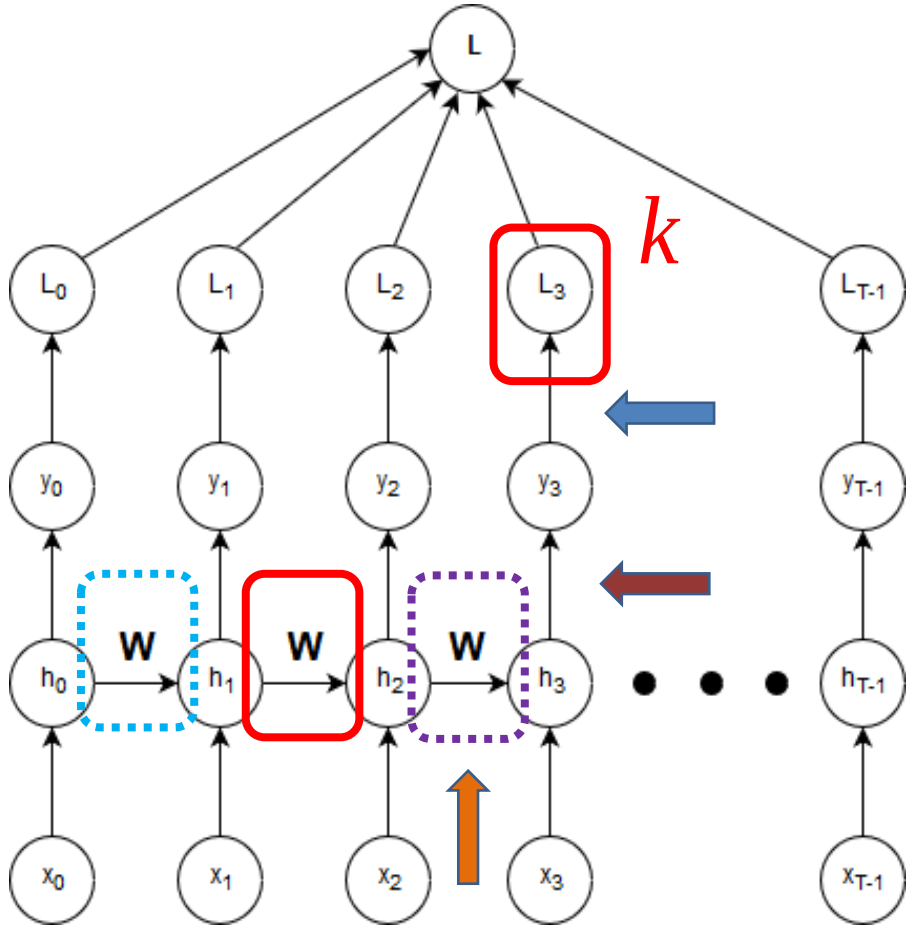
$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$



# RNN - BPTT (Backpropagation Through Time)

› Final EBP:

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$



# RNN - BPTT (Backpropagation Through Time)

- Problem with consider all paths:
  - Memory
  - Gradient Flow (why?)

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

- Solution:
  - Truncate the network!

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=j-p}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

# RNN - BPTT (Backpropagation Through Time)

› Unwrap the state dependency:

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

# RNN - BPTT (Backpropagation Through Time)

› Dependency depth problem:

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

› Weight Matrix and derivative of activity function
  – Long Term Dependency → Repeated Matrix Multiplication

Gradient Vanishing and Exploding ☹

# RNN - BPTT (Backpropagation Through Time)

› For *ReLU* activation function:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

$$\frac{\partial h_j}{\partial h_k} = (\mathbf{W}_h^T)^n = Q^{-1} * \Lambda^n * Q$$

› Weight Matrix and derivative of activity function
  – Long Term Dependency → Repeated Matrix Multiplication

Gradient Vanishing and Exploding ☹

# RNN – Gradient Vanishing/Exploding

Bengio et al,. "On the difficulty of training recurrent neural networks." (2012)

› Gradient Vanishing/Exploding (just to remember):

– SVD (Singular Value Decomposition) for **W**

$$W_h = Q^{-1} * \Lambda * Q$$

$$W_h^n = Q^{-1} * \Lambda^n * Q$$

$$\begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{10} = \begin{bmatrix} 0.00098 & 0 \\ 0 & 57.665039 \end{bmatrix}$$

# RNN – Initialization

Improving Performance of Recurrent Neural Network With ReLu Nonlinearity, ICLR2016

› Gradient Vanishing/Exploding (just to remember):

– Initialization: Initialize with suitable eigenvalues:

| RNN Type | Description |
|---|---|
| sRNN | RNN with an input layer, single hidden layer and an output layer |
| IRNN | sRNN with recurrent weight matrix initialized to Identity matrix |
| iRNN | sRNN with recurrent weight matrix initialized to 0.01 times Identity matrix |
| nRNN | sRNN with recurrent weight matrix initialized to a normalized matrix with all except the highest eigenvalue less than 1 |
| np-RNN | sRNN with recurrent weight matrix initialized to a normalized-positive definite matrix with all except the highest eigenvalue less than 1 |
| np-tanhRNN | np-RNN with tanh hidden node nonlinearity |
| oRNN | sRNN with recurrent weight matrix initialized to orthogonal random matrix |
| gRNN | sRNN with recurrent weight matrix initialized to a random Gaussian matrix |

# RNN - Exploding

› Avoid by clipping:

**Algorithm 1** Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
**if** $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
**end if**

› Avoid by unit feedback gain (ResNet idea):

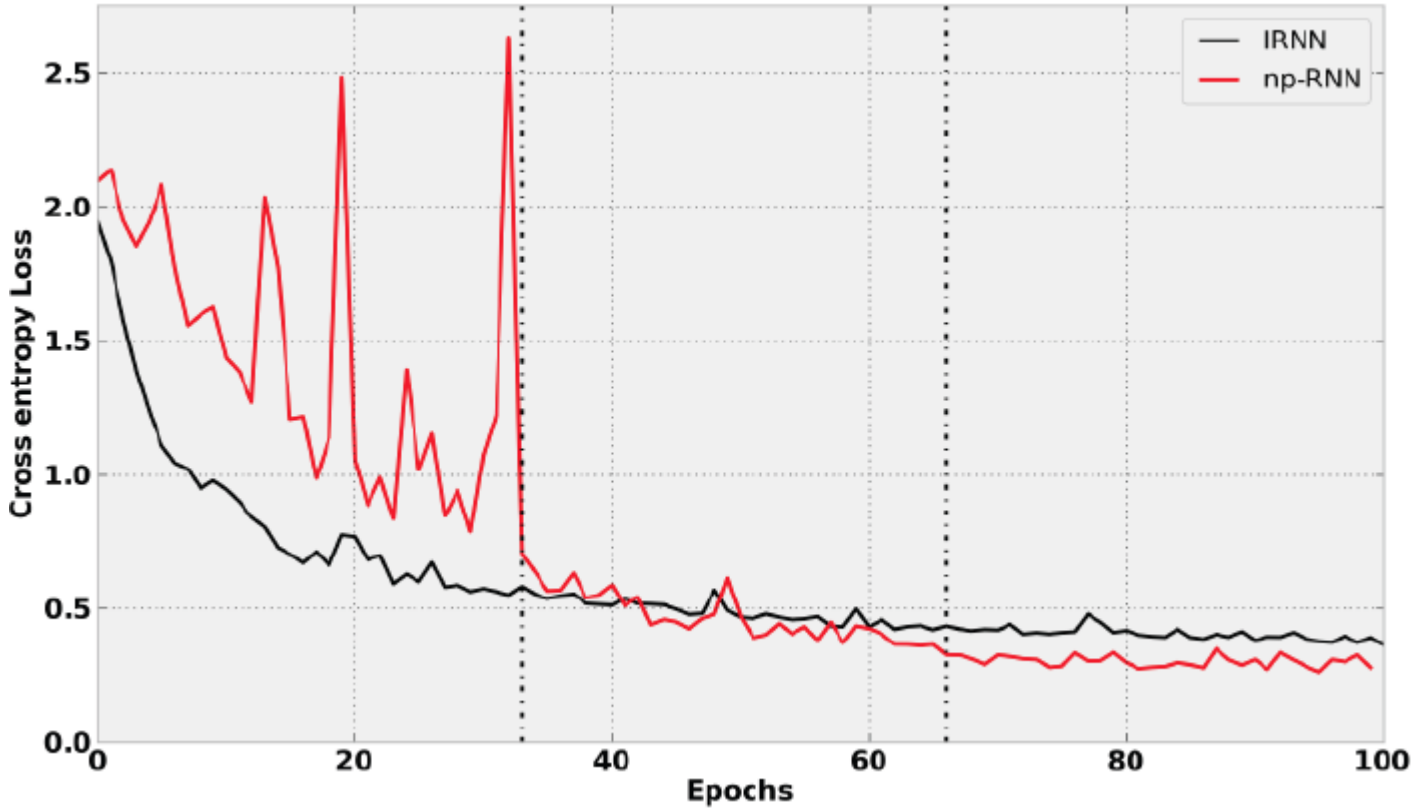
$$h_t = h_{t-1} + F(x_t)$$

$$\Rightarrow \left(\frac{\partial h_t}{\partial h_{t-1}}\right) = 1$$

# RNN – Initialization

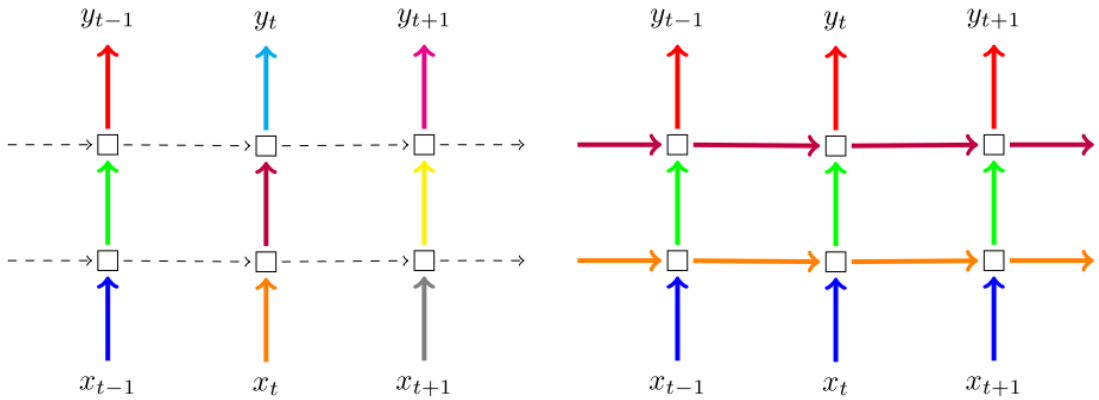
Improving Performance Of Recurrent Neural Network With ReLu Nonlinearity, ICLR2016

› MNIST experiment:



# RNN Dropout - A Theoretically Grounded Application of Dropout in Recurrent Neural Networks – NIPS 2016

- May apply on input/output dependency path not/or recurrent layer:
  - Only Apply on non-recurrent connections (Solid/Color Connection)
  - All Connection but each timestep applies the same dropout mask, different colors corresponding to different dropout masks




(a) Naive dropout RNN (b) Variational RNN

# RNN – How to Deepen!

- Input to the Hidden state, (Convert weight matrix to a NET), $W_{xh}$
  - Higher level representation
- Recurrent Layer, $W_{hh}$
  - Add high nonlinearity (+temporal dependency)
- Hidden state to Output layer, $W_{hy}$
  - Facilitate the output prediction,

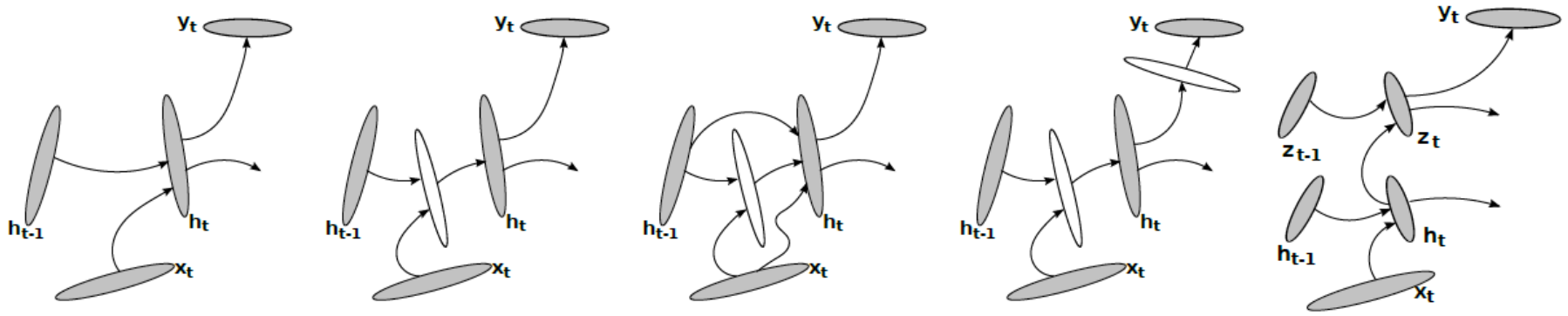
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

# RNN Deepening – How to Construct Deep Recurrent Neural Networks, ICLR2014

› Deep Transition RNN (DT-RNN):

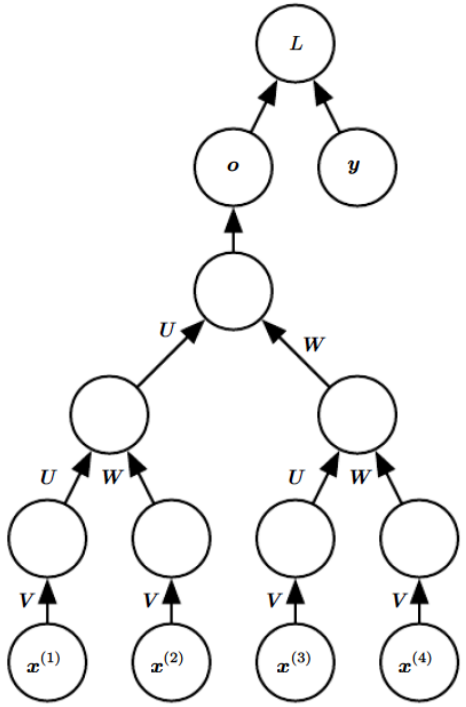



(a) RNN (b) DT-RNN (b*) DT(S)-RNN (c) DOT-RNN (d) Stacked RNN

Figure 2: Illustrations of four different recurrent neural networks (RNN). (a) A conventional RNN. (b) Deep Transition (DT) RNN. (b*) DT-RNN with shortcut connections (c) Deep Transition, Deep Output (DOT) RNN. (d) Stacked RNN

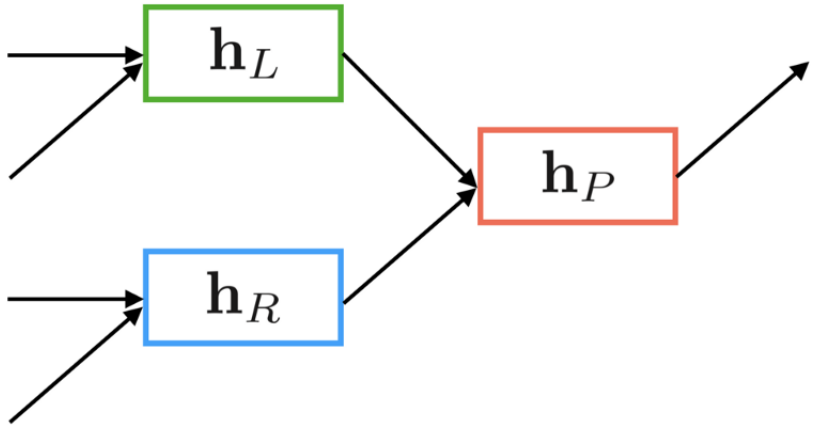
# RNN Variations

- Recursive Neural Network:
  - Tree (not chain as RNN) like computational graph



# RNN Variations

- Recursive Neural Network:
  - A tree structure with fixed number of branches

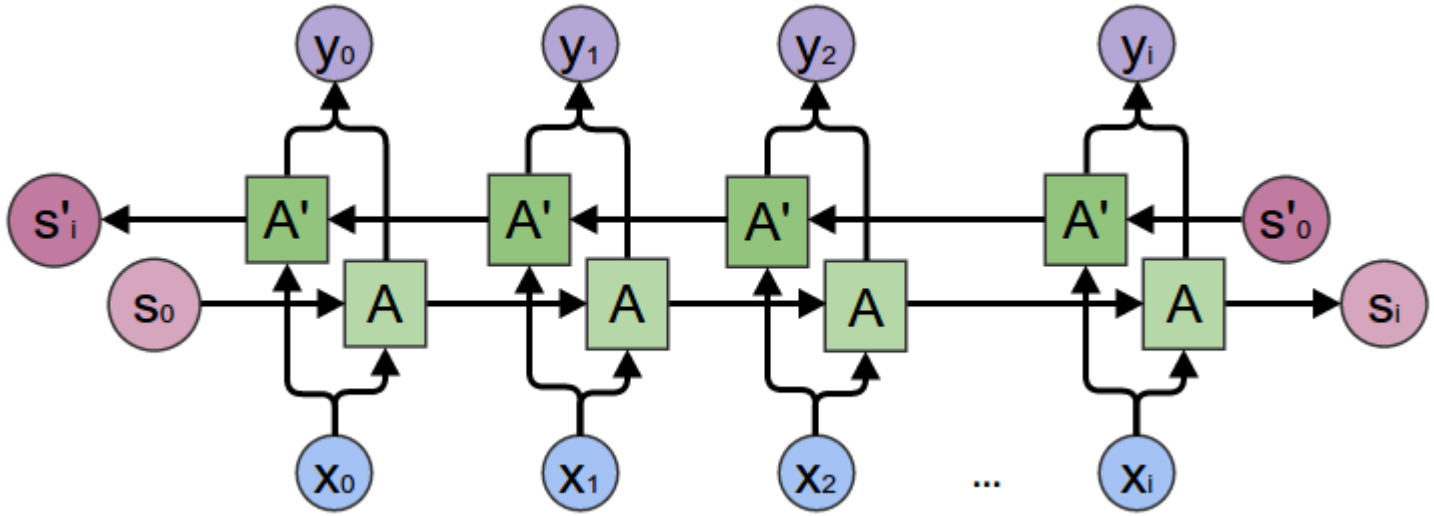



$$\mathbf{h}_P = a\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_L \\ \mathbf{h}_R\end{bmatrix} + \mathbf{b}\right)$$

  - *Above operation is sequentially performed from the **leaf** nodes toward the **root** node*

# RNN Variations

- Bidirectional Recurrent Neural Network:
  - Non-Causal inference (Back to future ☺)
    - Merge to RNN (one forward through time 1→t and one backward through time t→1)



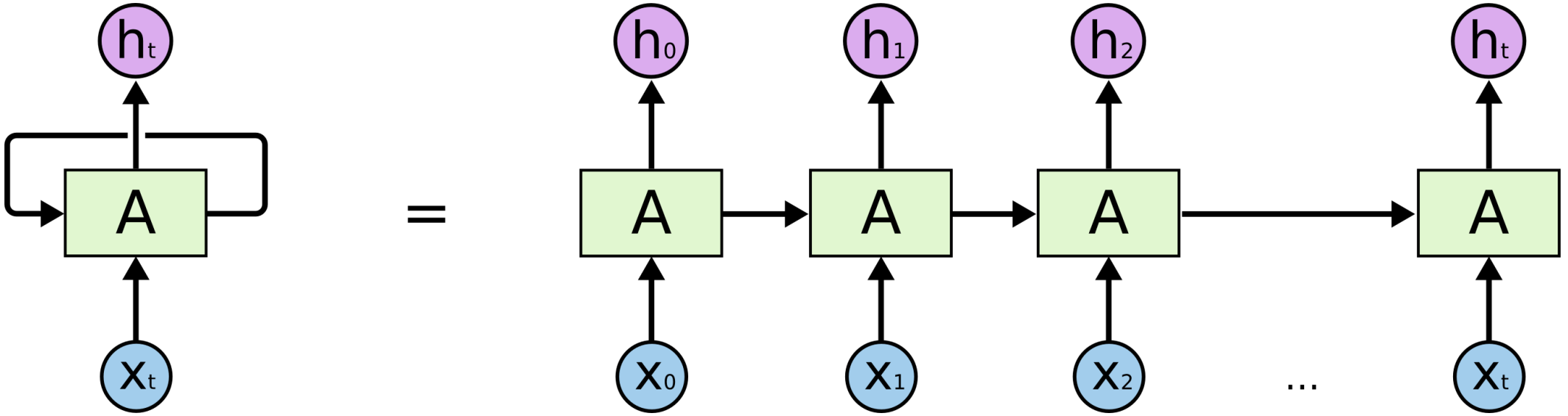
# Long Short-Term Memory

- First Generation: Long Short-Term Memory, *Neural Computation*, 1997.
  - Constant Error Carousel (CEC)
- To be continued! Gated Recurrent Unit (GRU), 2014.
- Aims/Idea:

  A memory cell that is not subject to matrix multiplication, thereby avoiding gradient decay

# RNN – Remember

› Conventional RNN



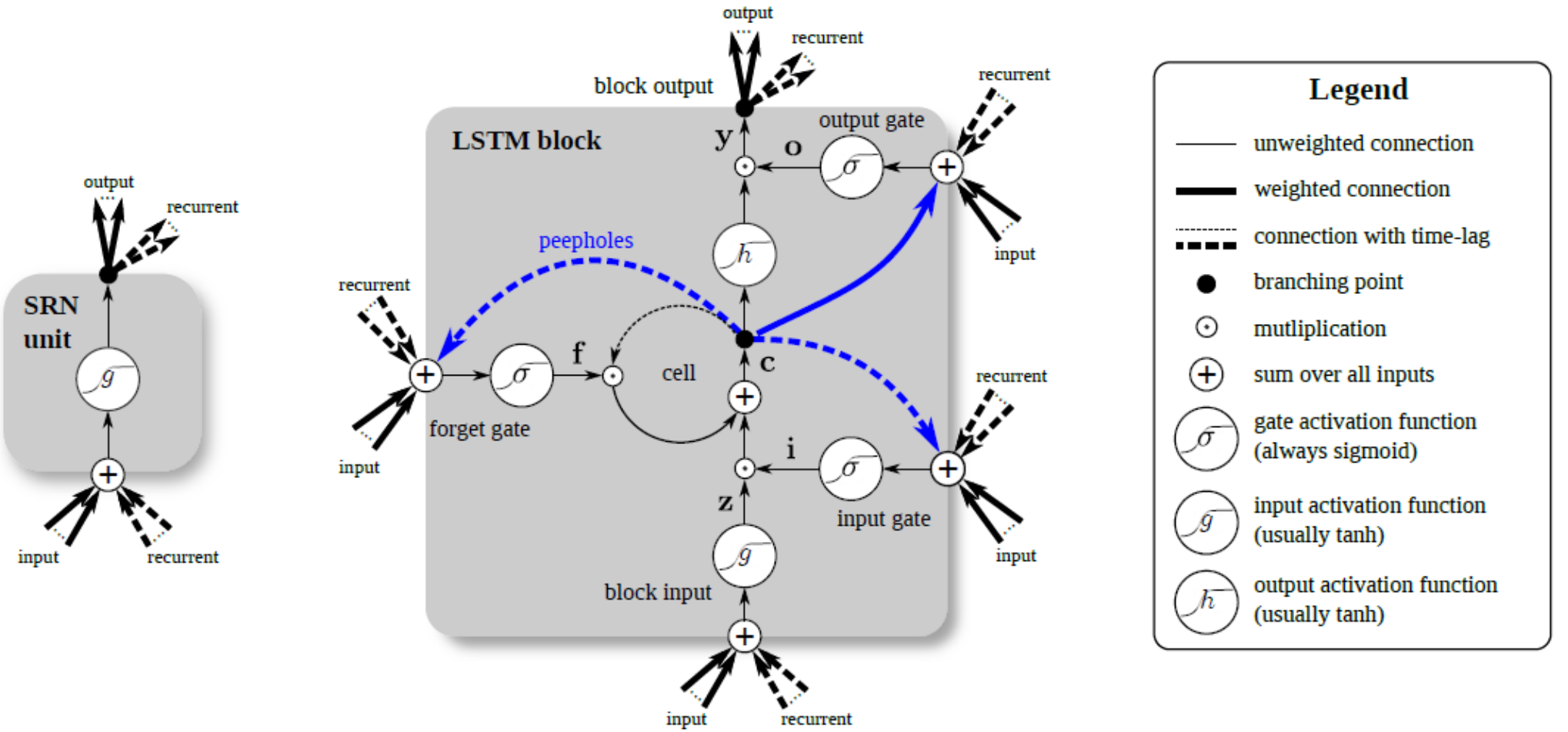

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

# LSTM

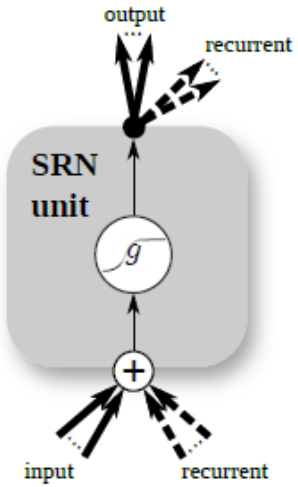
- Motivation: Memory Block Operation!
  - Read, How much?
  - Write, How much?
  - Reset , How much?
- "How much?" implementation
  - Read: input gate (if closed, overwrite by new input is not possible)
  - Write: output gate
  - Reset: clear saved memory

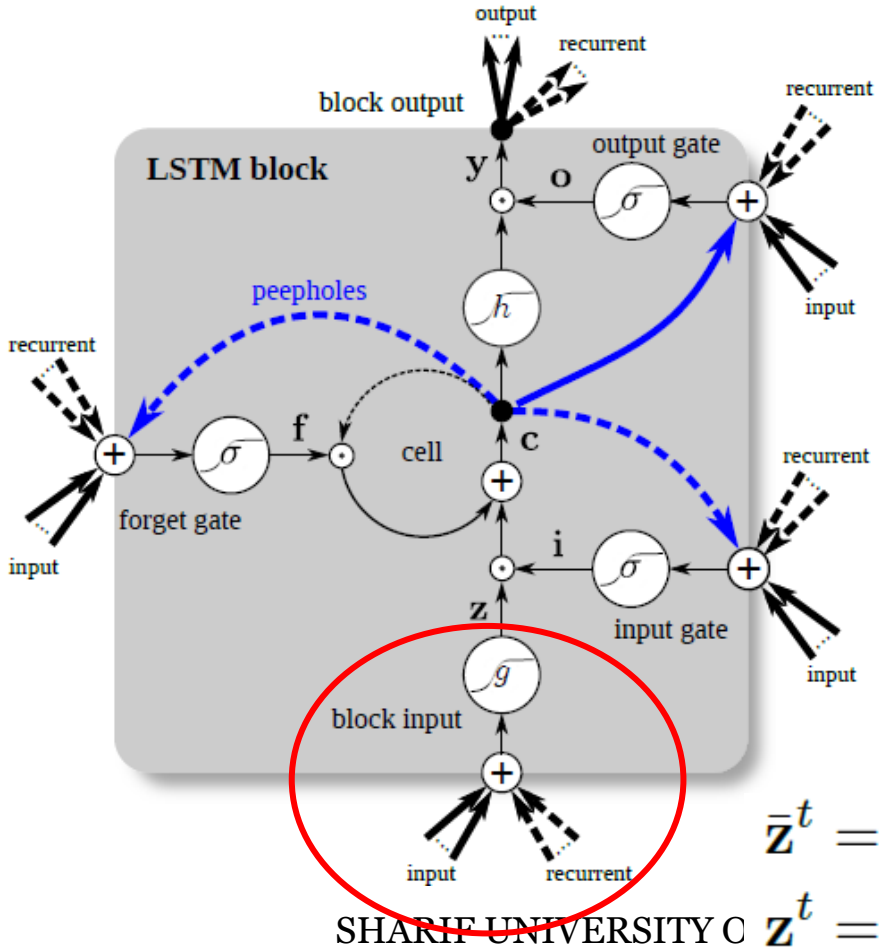
# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

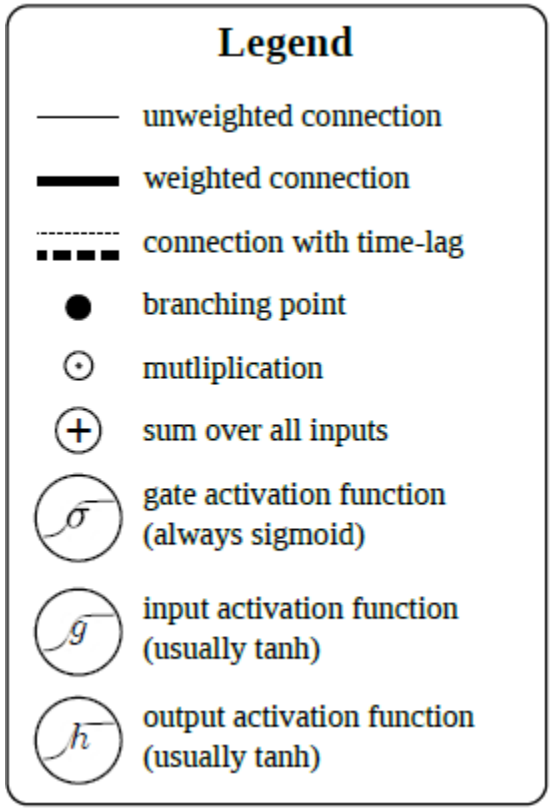


# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

› 1 - Input Block



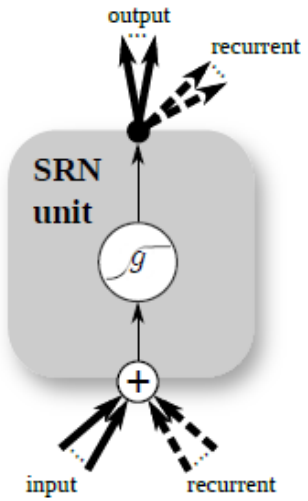



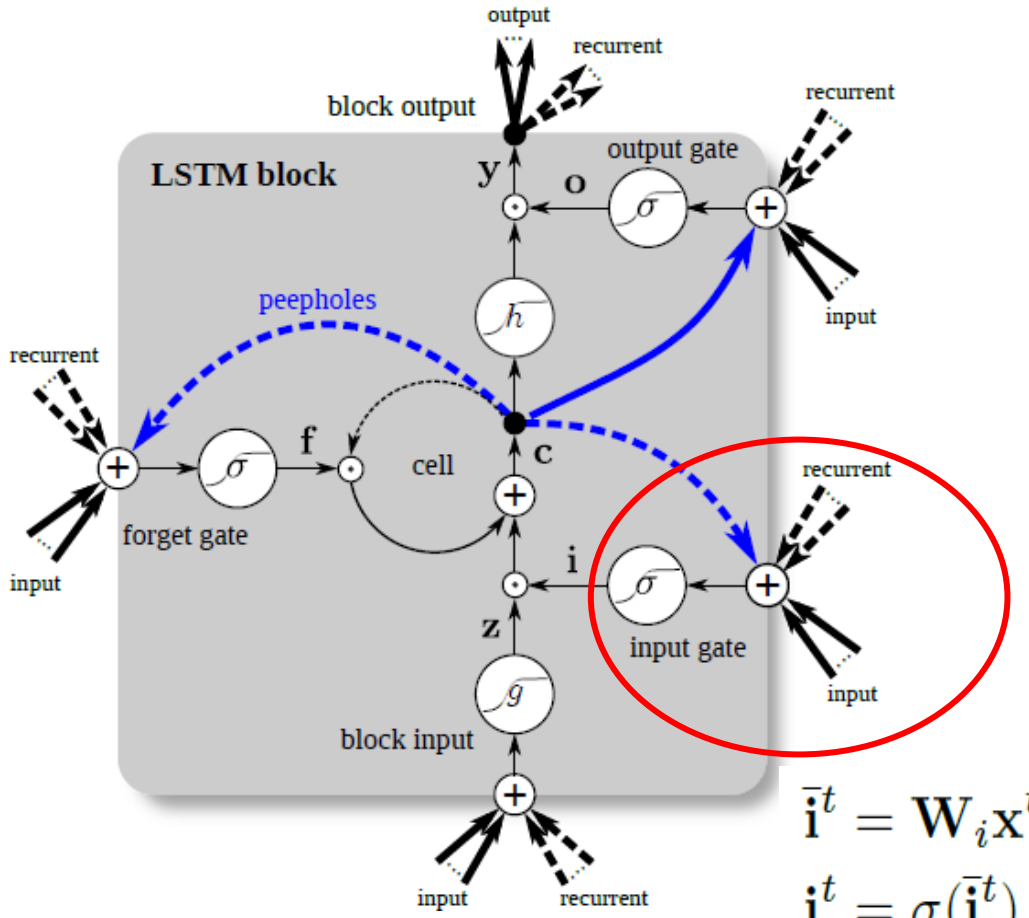

$$\bar{\mathbf{z}}^t = \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z$$

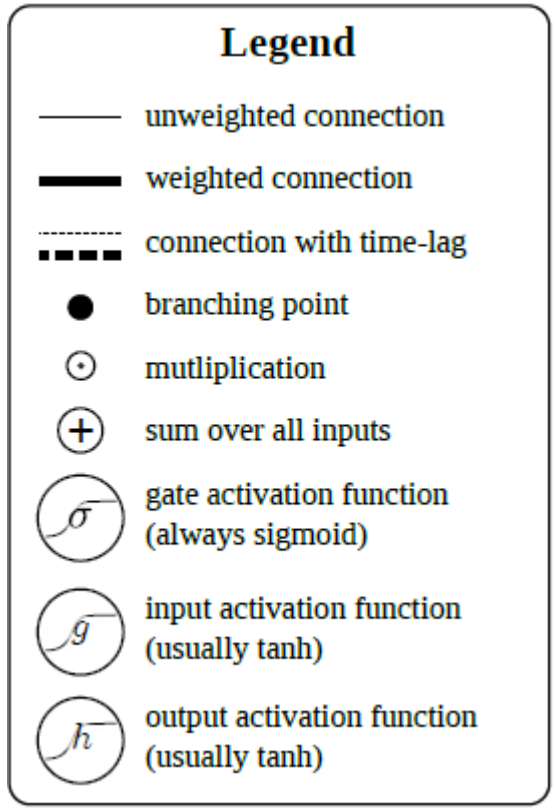
$$\mathbf{z}^t = g(\bar{\mathbf{z}}^t)$$

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

› 2 - Input Gait



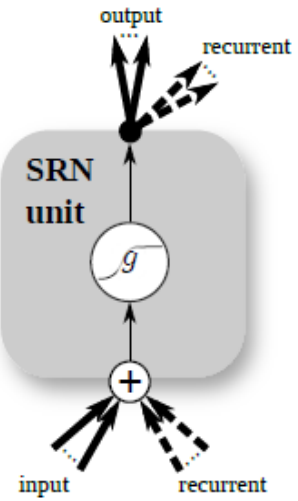



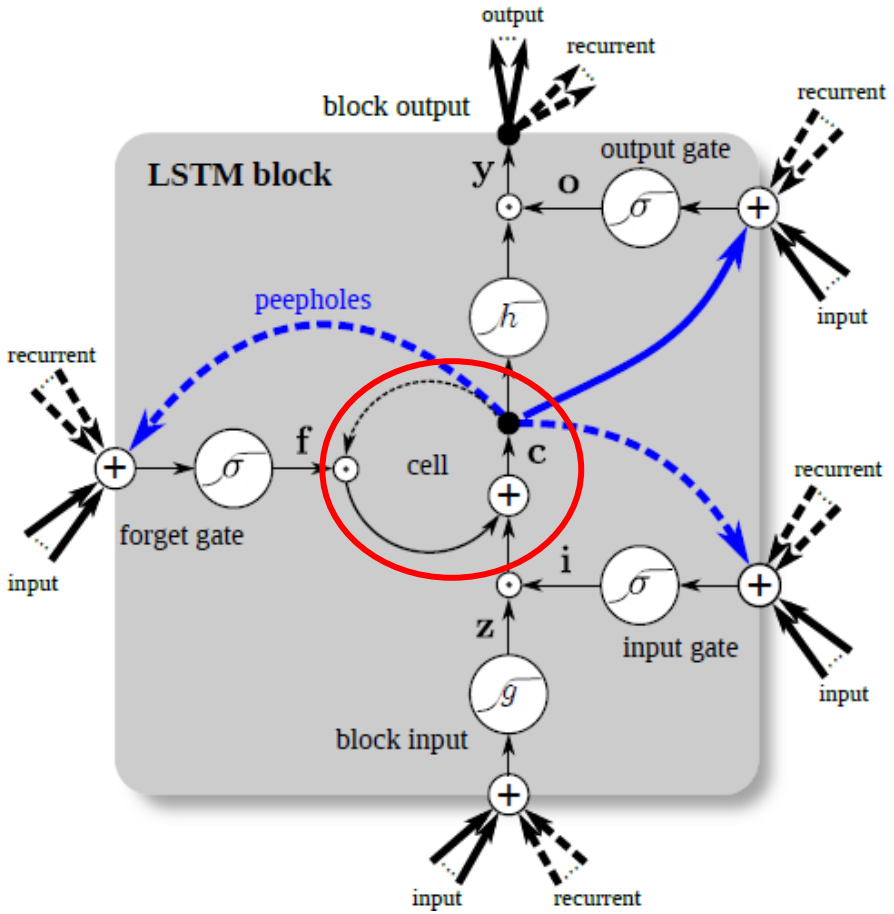

$$\bar{\mathbf{i}}^t = \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i$$

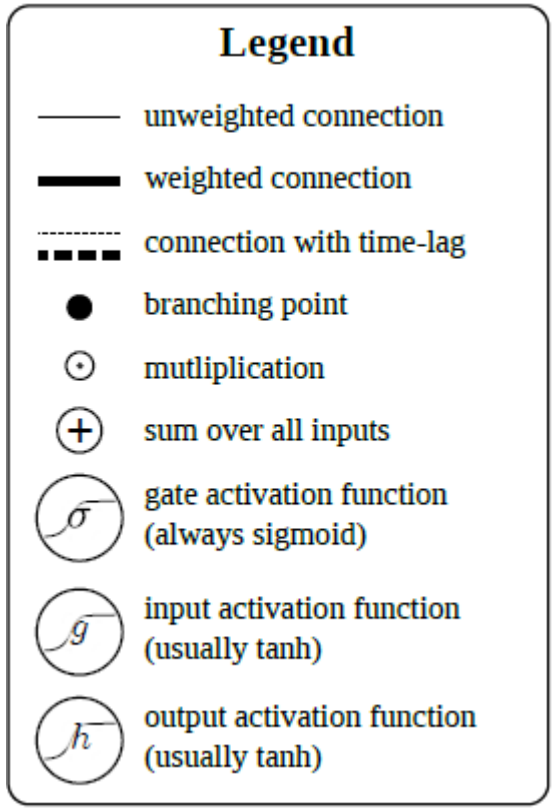
$$\mathbf{i}^t = \sigma(\bar{\mathbf{i}}^t)$$

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

› 3 – Memory **Cell** (No Weight)

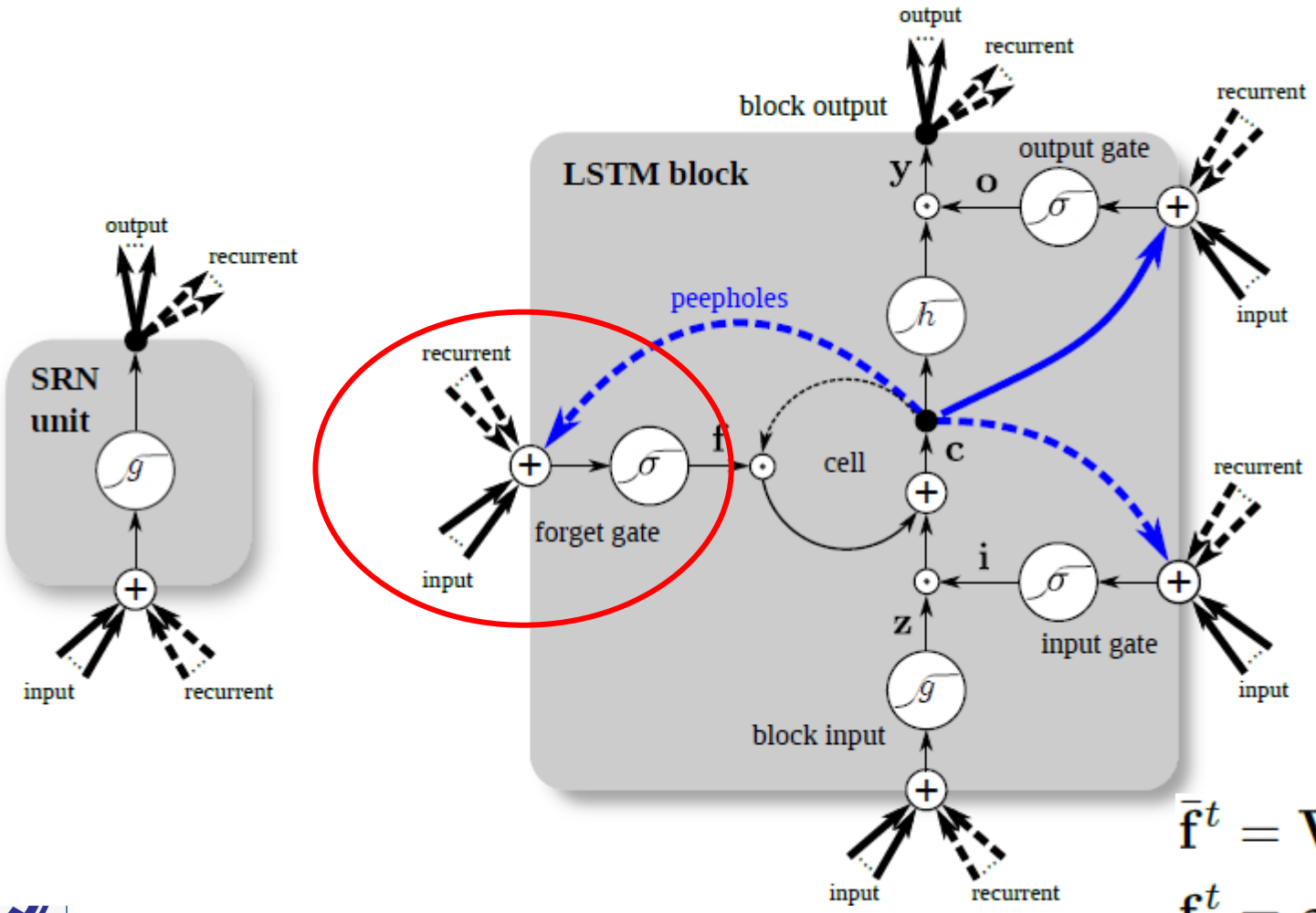


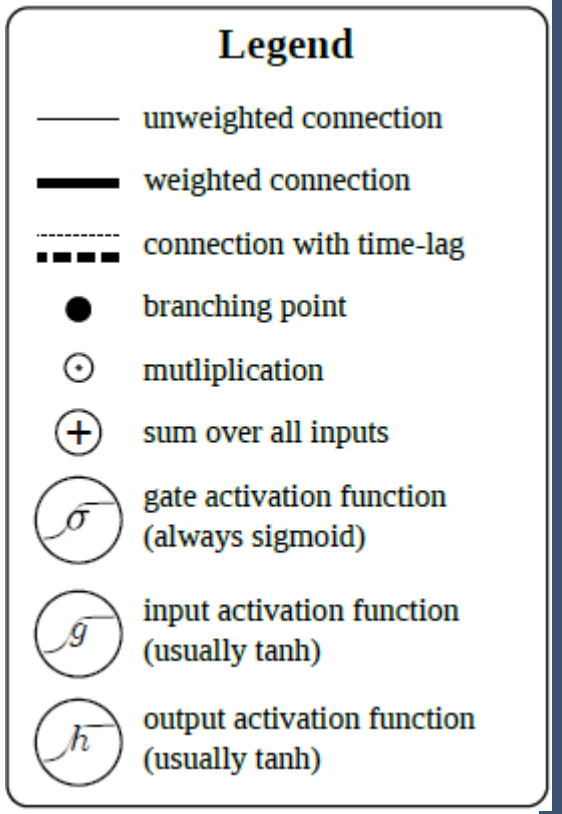




$$\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t$$

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

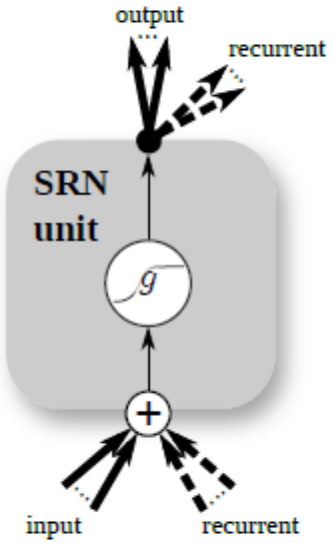


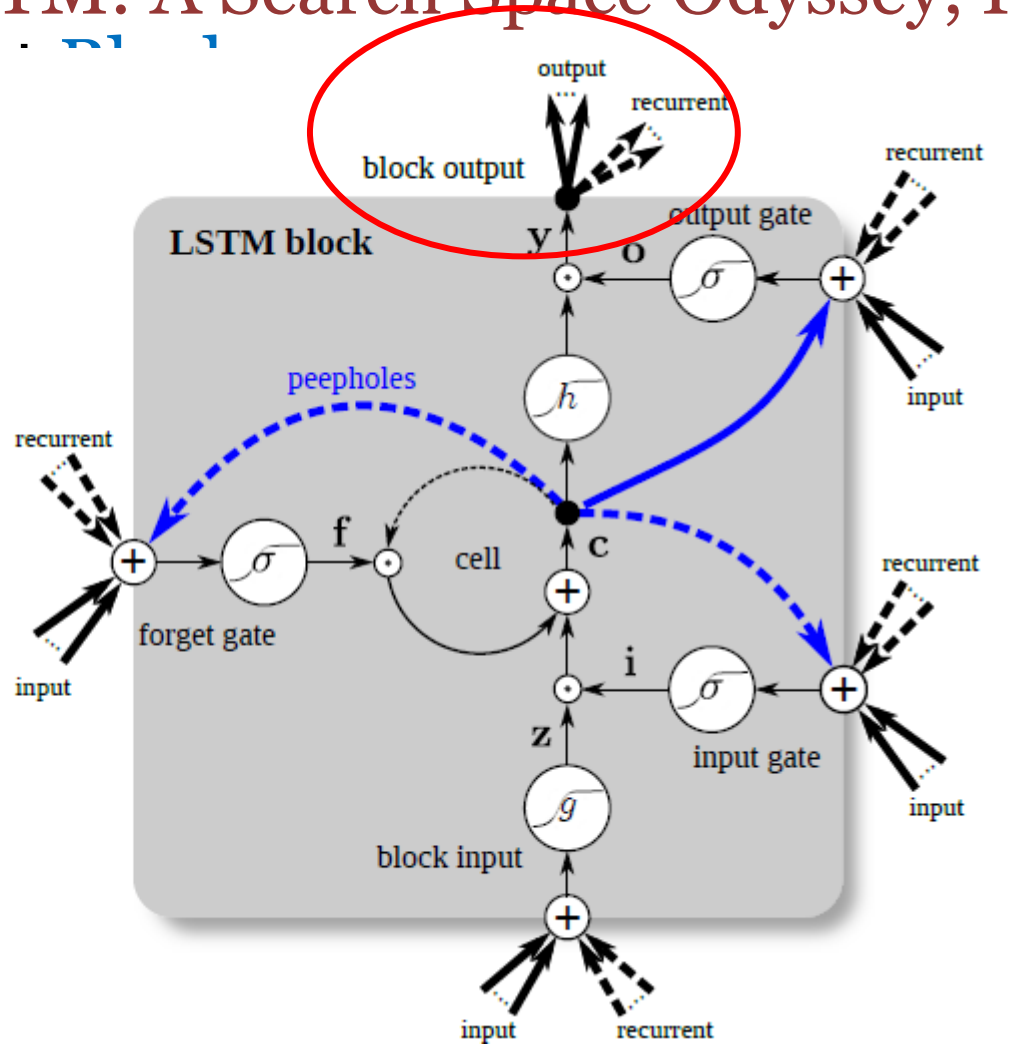



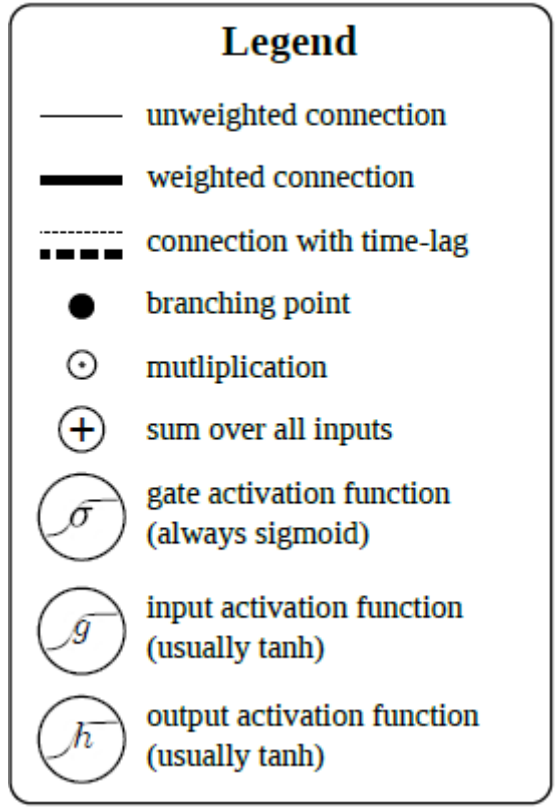

$$\bar{\mathbf{f}}^t = \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f$$

$$\mathbf{f}^t = \sigma(\bar{\mathbf{f}}^t)$$

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

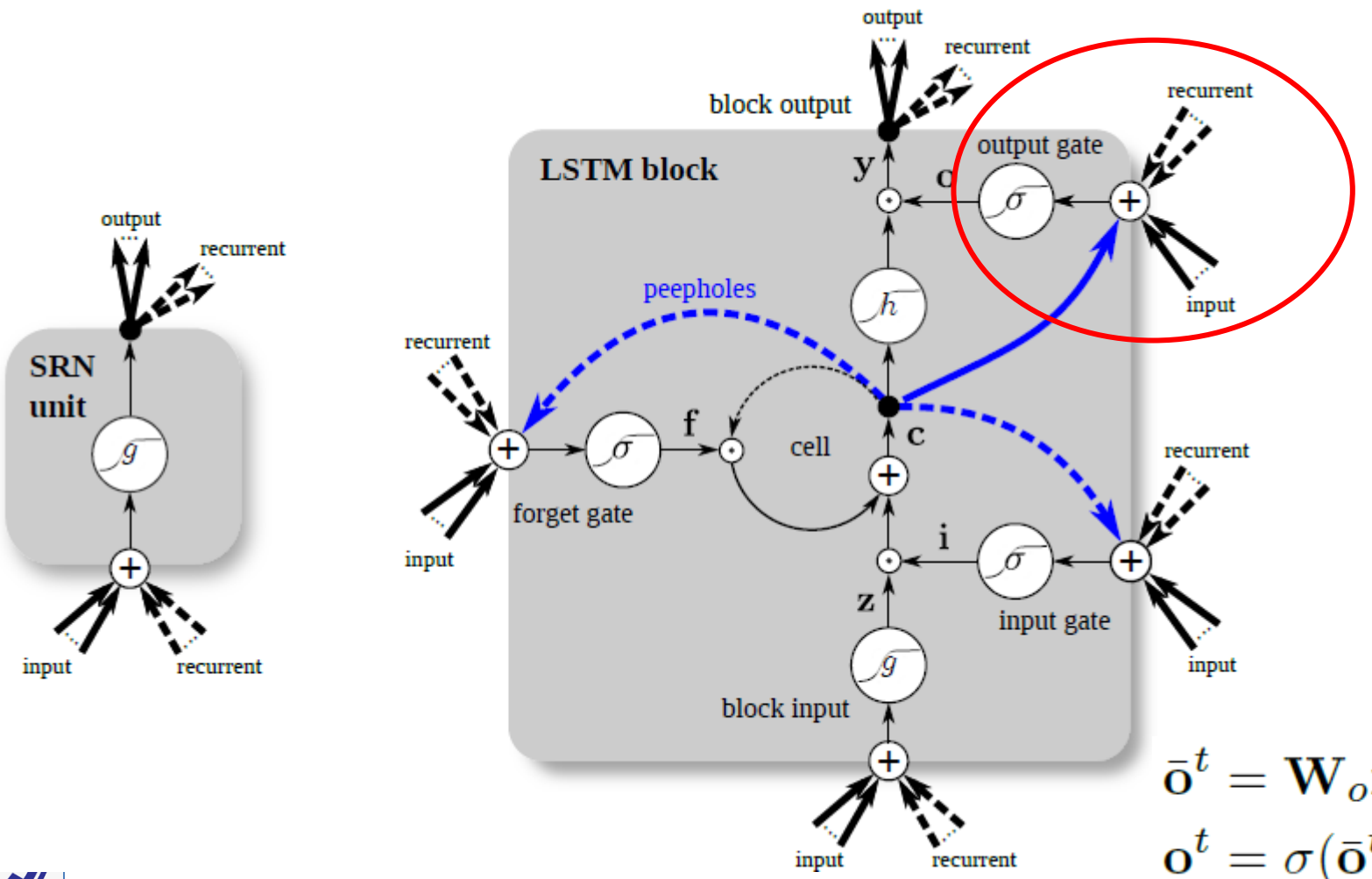


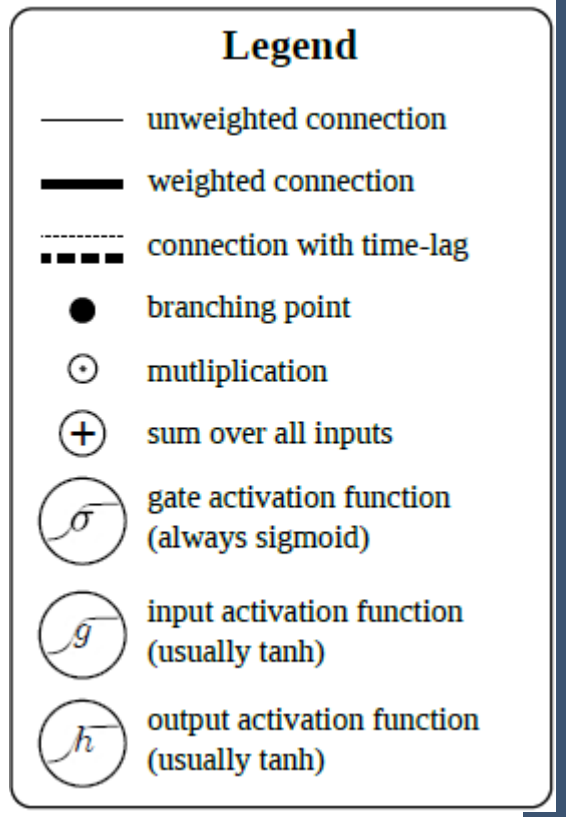




$$\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t$$

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017




$$\bar{\mathbf{o}}^t = \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o$$

$$\mathbf{o}^t = \sigma(\bar{\mathbf{o}}^t)$$

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

› Details:

– $x^t$: input vector in time t,

– $N$: # of LSTM blocks

– $M$: # of inputs

- Input weights: $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $\mathbf{R}_z, \mathbf{R}_i, \mathbf{R}_f, \mathbf{R}_o \in \mathbb{R}^{N \times N}$
- Peephole weights: $\mathbf{p}_i, \mathbf{p}_f, \mathbf{p}_o \in \mathbb{R}^N$
- Bias weights: $\mathbf{b}_z, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^N$

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

› Forward Calculation:

$$\bar{\mathbf{z}}^t = \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z$$

$$\mathbf{z}^t = g(\bar{\mathbf{z}}^t) \qquad \textit{block input}$$

$$\bar{\mathbf{i}}^t = \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i$$

$$\mathbf{i}^t = \sigma(\bar{\mathbf{i}}^t) \qquad \textit{input gate}$$

$$\bar{\mathbf{f}}^t = \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f$$

$$\mathbf{f}^t = \sigma(\bar{\mathbf{f}}^t) \qquad \textit{forget gate}$$

$$\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t \qquad \textit{cell}$$

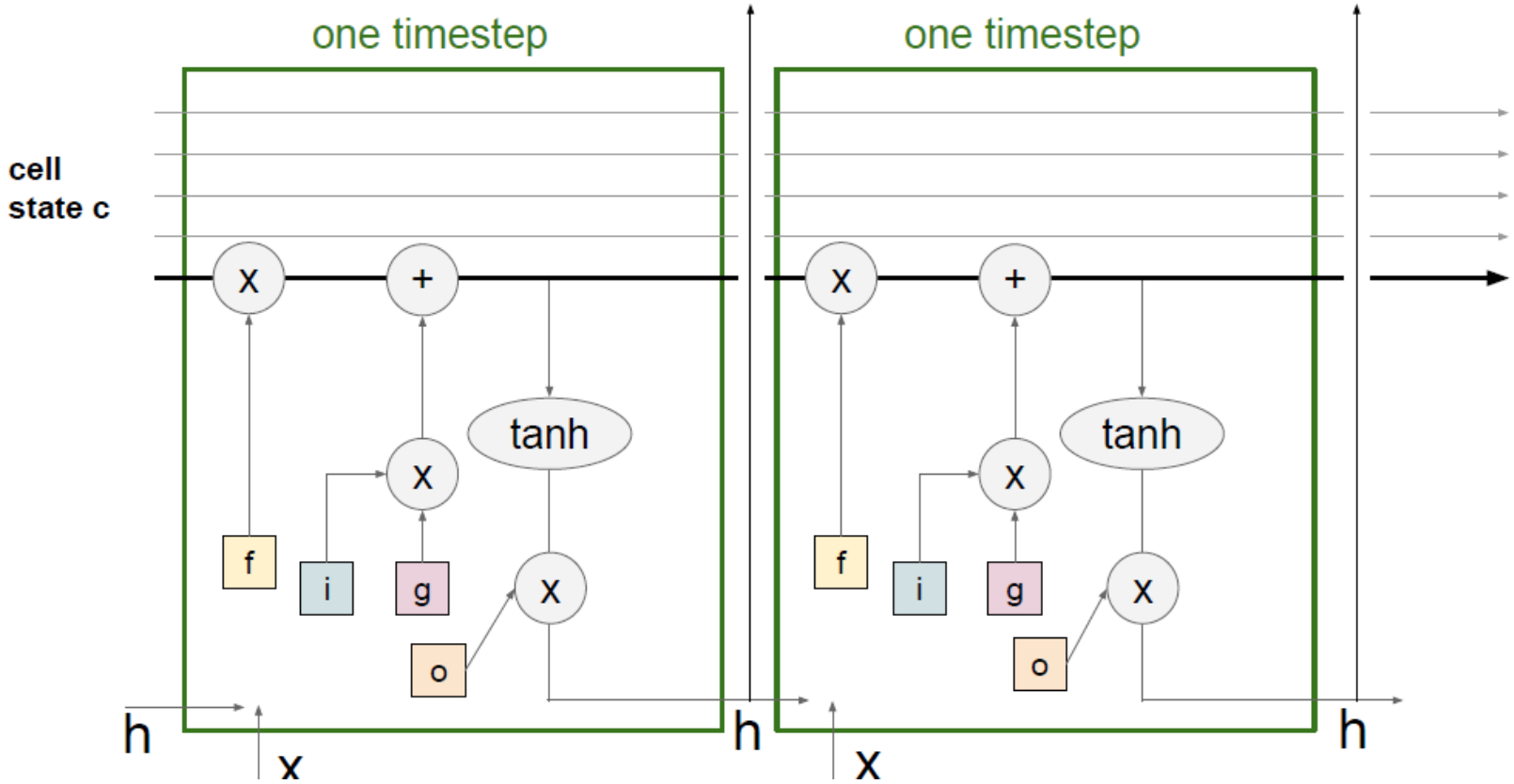
$$\bar{\mathbf{o}}^t = \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o$$

$$\mathbf{o}^t = \sigma(\bar{\mathbf{o}}^t) \qquad \textit{output gate}$$

$$\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t \qquad \textit{block output}$$

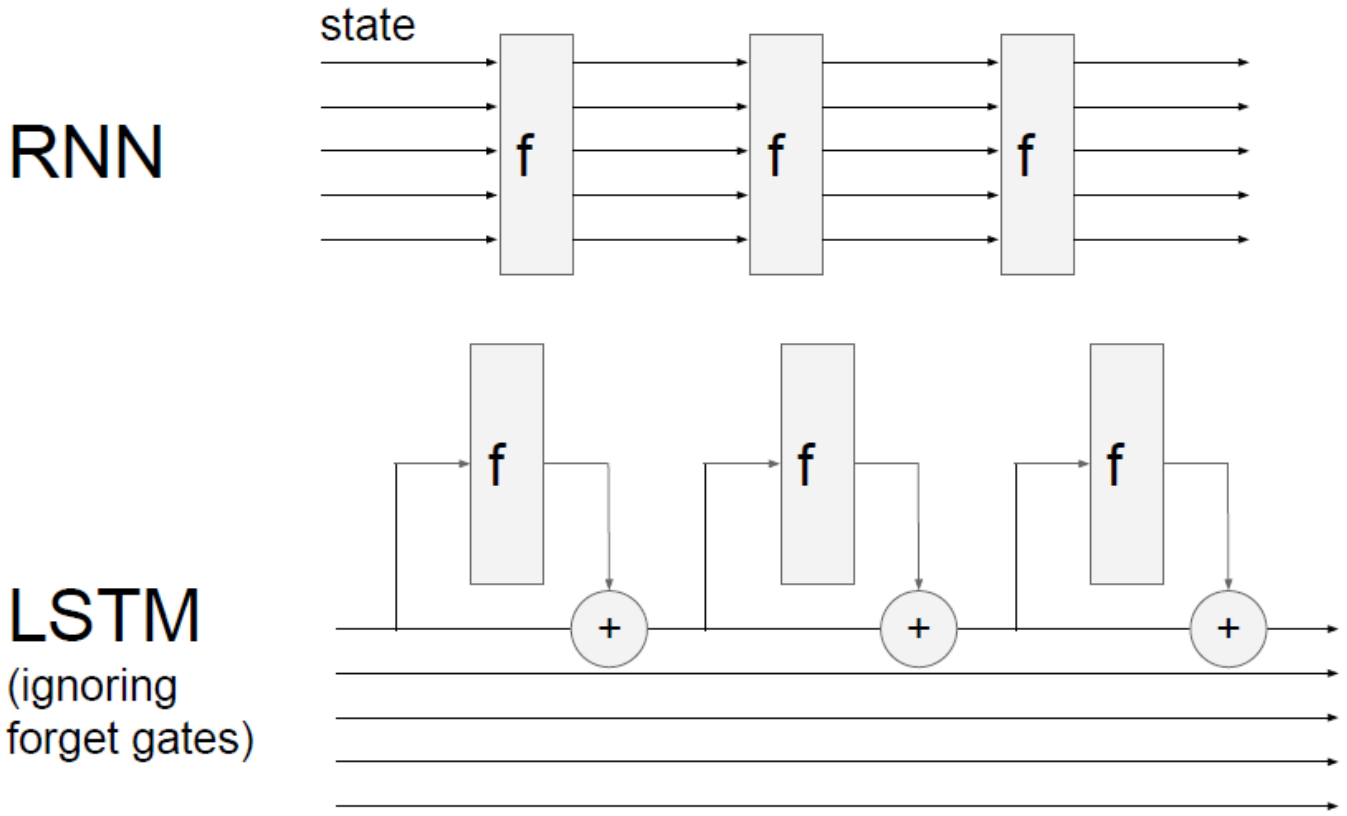
# LSTM- Fei-Fei, CS231,2016

› Some illustration (unfolding)



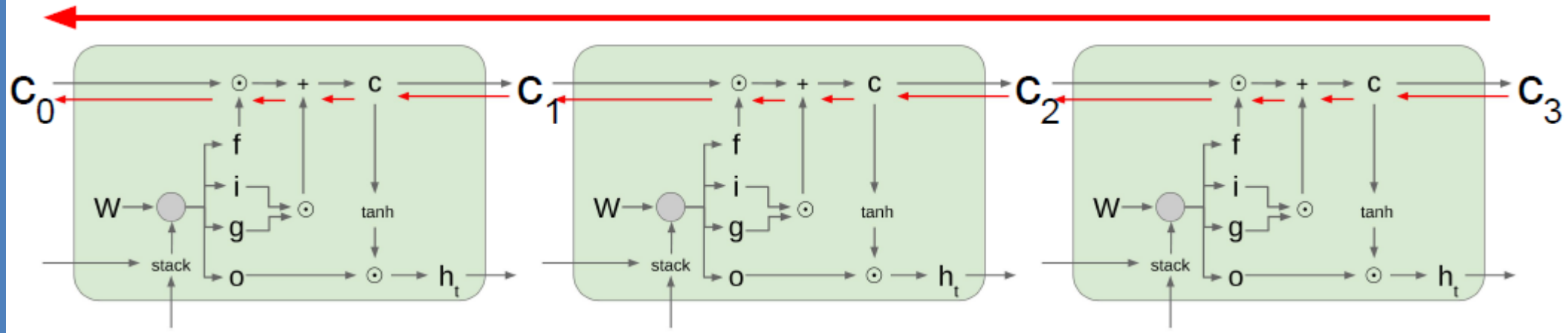
# LSTM- Fei-Fei, CS231,2016

› Remember ResNet idea (Skip Connection)



# LSTM – Why works!

› Backpropagation from $C_t$ to $C_{t-1}$: Elementwise multiplication by $f$, not matrix multiply by $W$



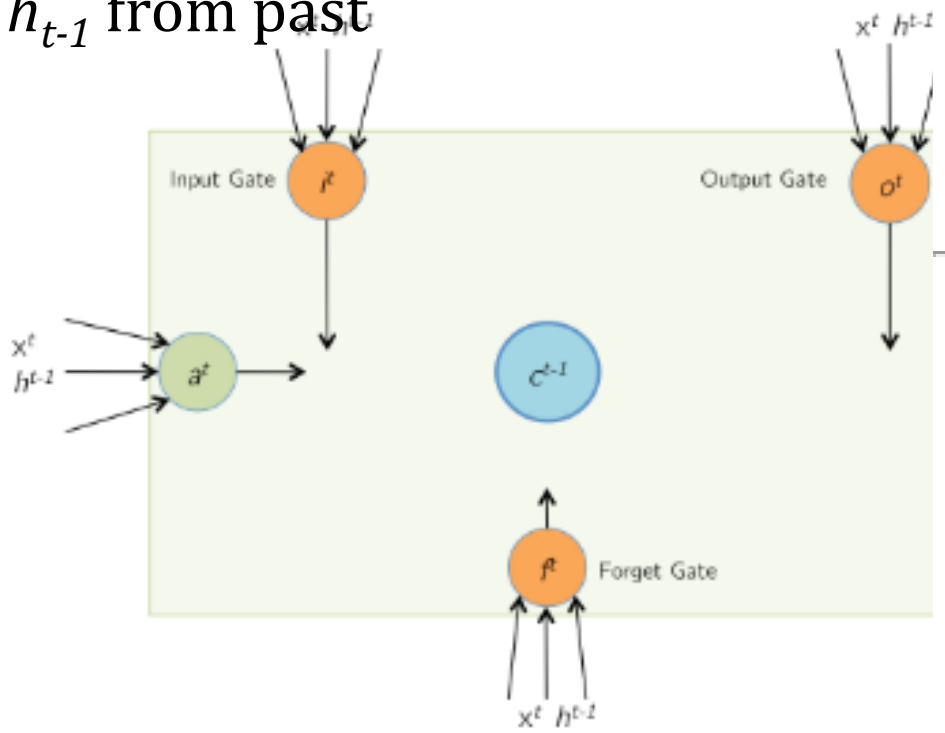
# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

› At time t:

– $x_t$ from input (now)

– $h_{t-1}$ from past




$$a^t = \tanh(W_c x^t + U_c h^{t-1}) = \tanh(\hat{a}^t)$$
$$i^t = \sigma(W_i x^t + U_i h^{t-1}) = \sigma(\hat{i}^t)$$
$$f^t = \sigma(W_f x^t + U_f h^{t-1}) = \sigma(\hat{f}^t)$$
$$o^t = \sigma(W_o x^t + U_o h^{t-1}) = \sigma(\hat{o}^t)$$

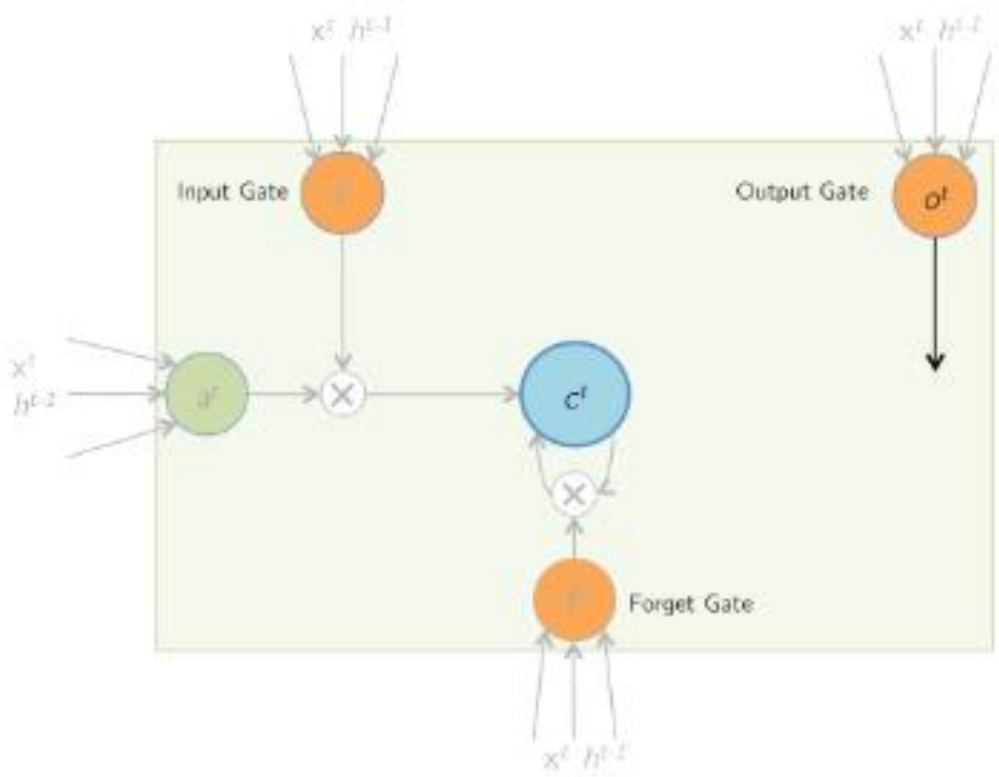
Ignoring the non-linearities,

$$z^t = \begin{bmatrix} \hat{a}^t \\ \hat{i}^t \\ \hat{f}^t \\ \hat{o}^t \end{bmatrix} = \begin{bmatrix} W^c & U^c \\ W^i & U^i \\ W^f & U^f \\ W^o & U^o \end{bmatrix} \times \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix}$$
$$= W \times I^t$$

# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

› Memory Cell Update!



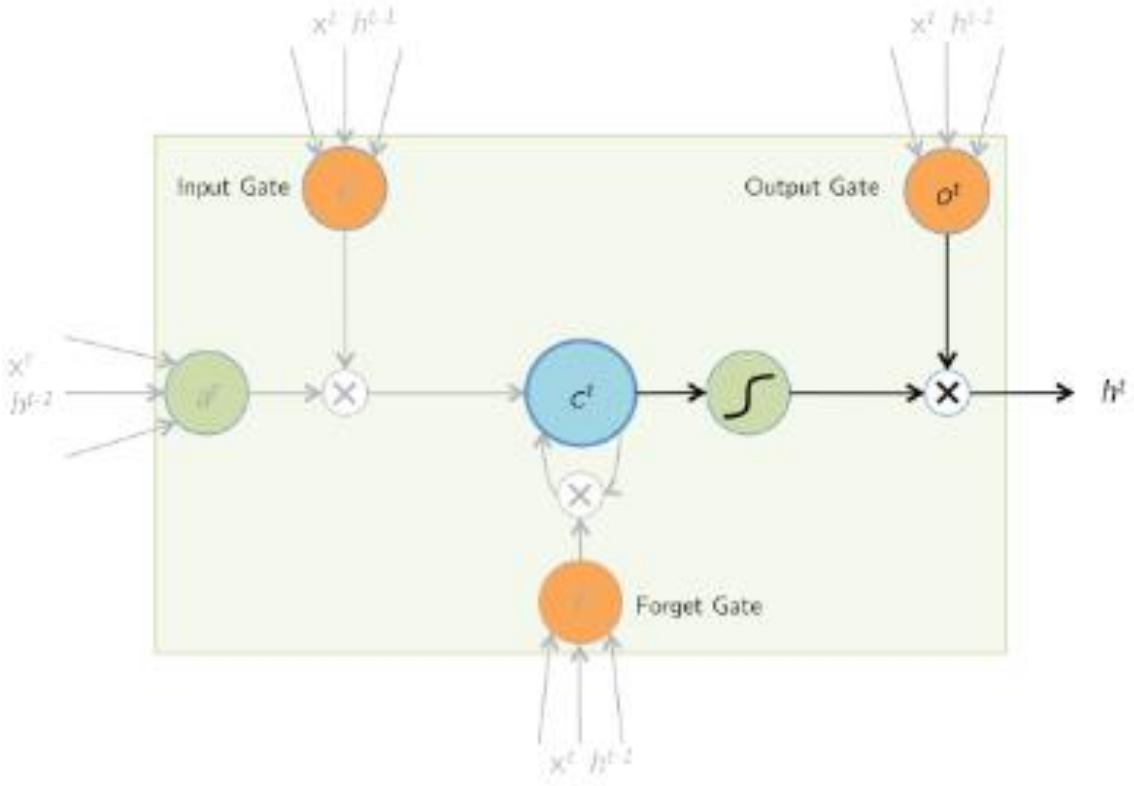

$$c^t = i^t \odot a^t + f^t \odot c^{t-1}$$

$$c^{t-1} \rightarrow c^t$$

# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

› Output Update!

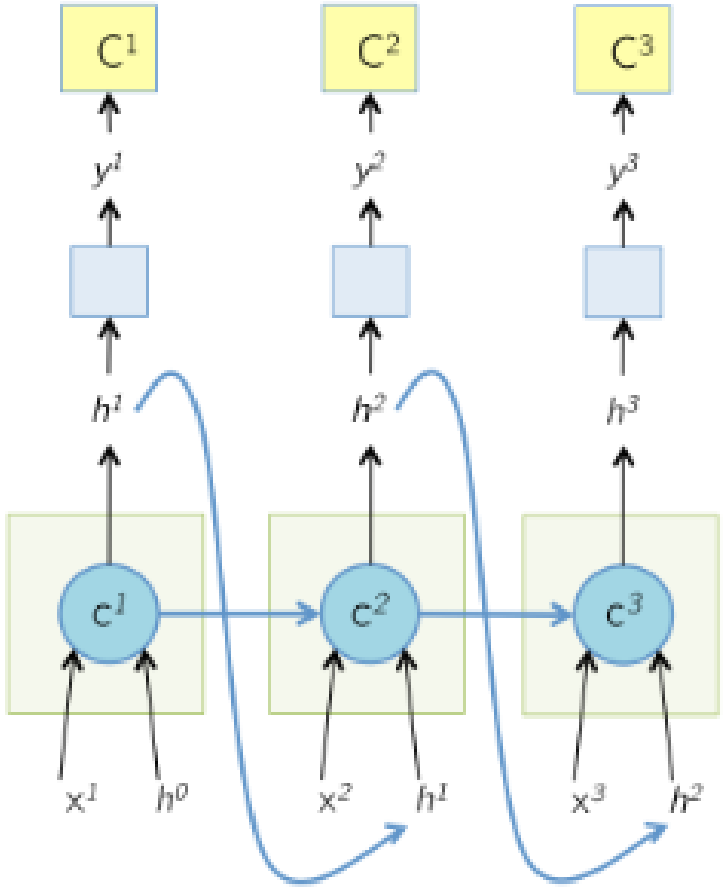



$$h^t = o^t \odot \tanh(c^t)$$

# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

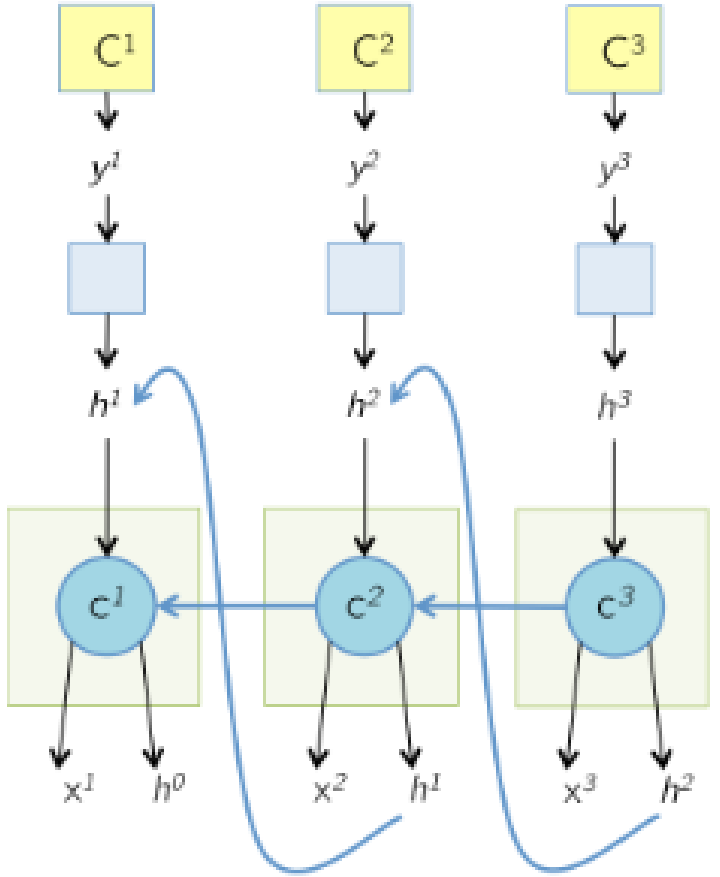
› Unfolding Forward Pass:



# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

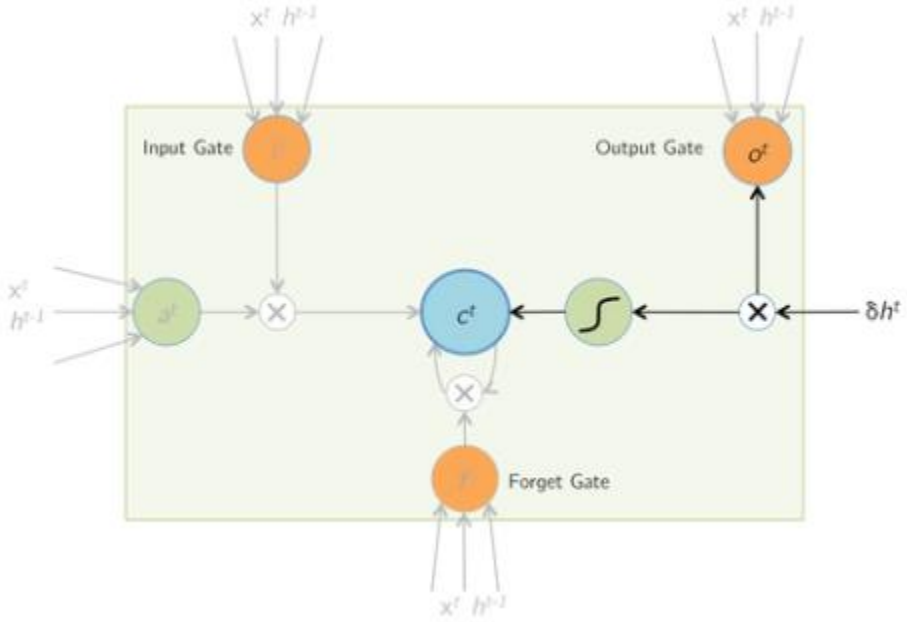
› Unfolding Backward Pass:



# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

› Unfolding Backward Pass:




$$\text{Forward Pass: } h^t = o^t \odot \tanh(c^t)$$

$$\text{Given } \delta h^t = \frac{\partial E}{\partial h^t}, \text{find } \delta o^t, \delta c^t$$

---

$$\begin{aligned}\frac{\partial E}{\partial o_i^t} &= \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial o_i^t} \\ &= \delta h_i^t \cdot \tanh(c_i^t) \\ \therefore \delta o^t &= \delta h^t \odot \tanh(c^t)\end{aligned}$$

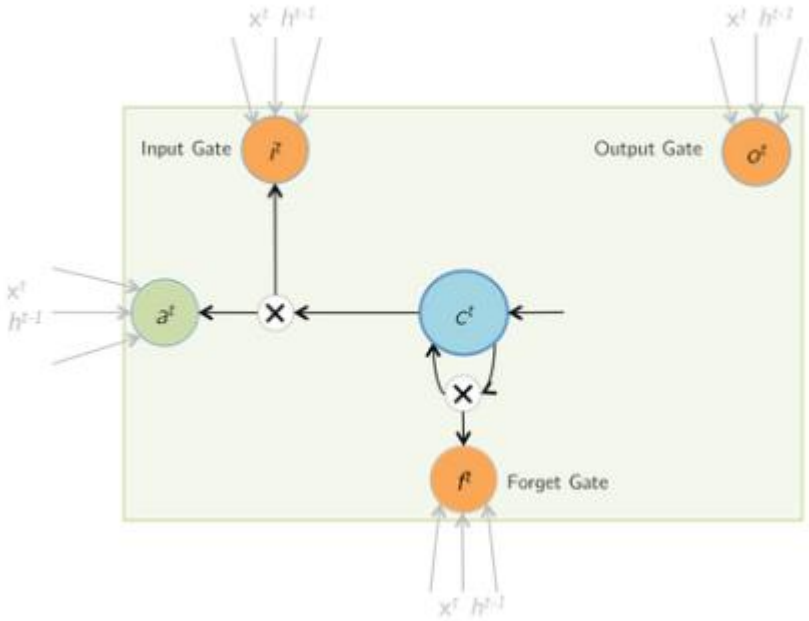
$$\begin{aligned}\frac{\partial E}{\partial c_i^t} &= \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial c_i^t} \\ &= \delta h_i^t \cdot o_i^t \cdot (1 - \tanh^2(c_i^t)) \\ \therefore \delta c^t &+= \delta h^t \odot o^t \odot (1 - \tanh^2(c^t))\end{aligned}$$

Note that the $+=$ above is so that this gradient is added to gradient from time step $(t+1)$ (calculated on next slide, refer to the gradient accumulation mentioned in the previous slide)

# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

› Unfolding Backward Pass:




Forward Pass: $c^t = i^t \odot a^t + f^t \odot c^{t-1}$

Given $\delta c^t = \frac{\partial E}{\partial c^t}$, find $\delta i^t, \delta a^t, \delta f^t, \delta c^{t-1}$

$$\frac{\partial E}{\partial i_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial i_i^t} = \delta c_i^t \cdot a_i^t \quad \therefore \delta i^t = \delta c^t \odot a^t$$

$$\frac{\partial E}{\partial f_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial f_i^t} = \delta c_i^t \cdot c_i^{t-1} \quad \therefore \delta f^t = \delta c^t \odot c^{t-1}$$

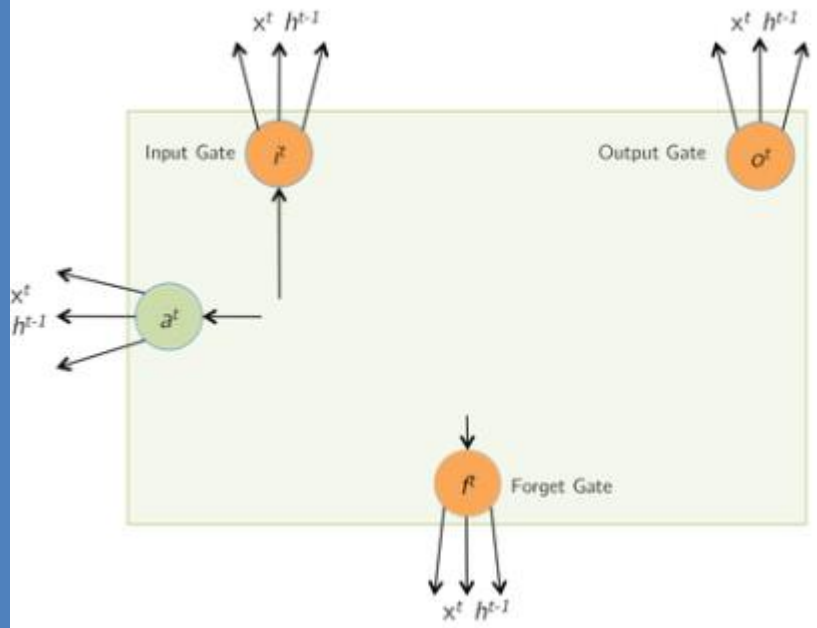
$$\frac{\partial E}{\partial a_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial a_i^t} = \delta c_i^t \cdot i_i^t \quad \therefore \delta a^t = \delta c^t \odot i^t$$

$$\frac{\partial E}{\partial c_i^{t-1}} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial c_i^{t-1}} = \delta c_i^t \cdot f_i^t \quad \therefore \delta c^{t-1} = \delta c^t \odot f^t$$

# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

› Unfolding Backward Pass:




$$\text{Forward Pass: } z^t = \begin{bmatrix} \hat{a}^t \\ \hat{i}^t \\ \hat{f}^t \\ \hat{o}^t \end{bmatrix} = W \times I^t$$

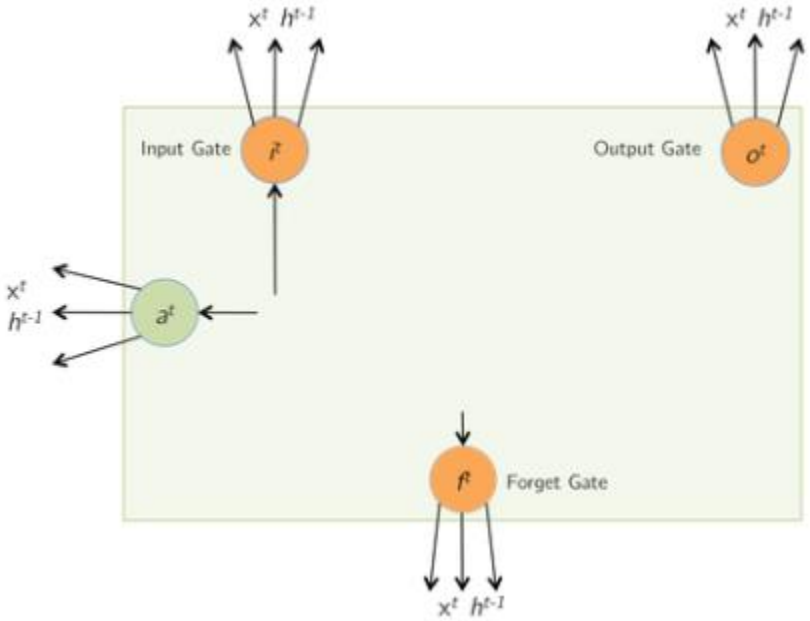
Given $\delta a^t, \delta i^t, \delta f^t, \delta o^t,$ find $\delta z^t$

$$\delta \hat{a}^t = \delta a^t \odot (1 - \tanh^2(\hat{a}^t))$$
$$\delta \hat{i}^t = \delta i^t \odot i^t \odot (1 - i^t)$$
$$\delta \hat{f}^t = \delta f^t \odot f^t \odot (1 - f^t)$$
$$\delta \hat{o}^t = \delta o^t \odot o^t \odot (1 - o^t)$$
$$\delta z^t = \left[\delta \hat{a}^t, \delta \hat{i}^t, \delta \hat{f}^t, \delta \hat{o}^t\right]^T$$

# Forward/Backward Pass -

http://arunmallya.github.io/writeups/nn/lstm/

› Unfolding Backward Pass:




Forward Pass: $z^t = W \times I^t$

Given $\delta z^t$, find $\delta W^t, \delta h^{t-1}$

---

$$\delta I^t = W^T \times \delta z^t$$

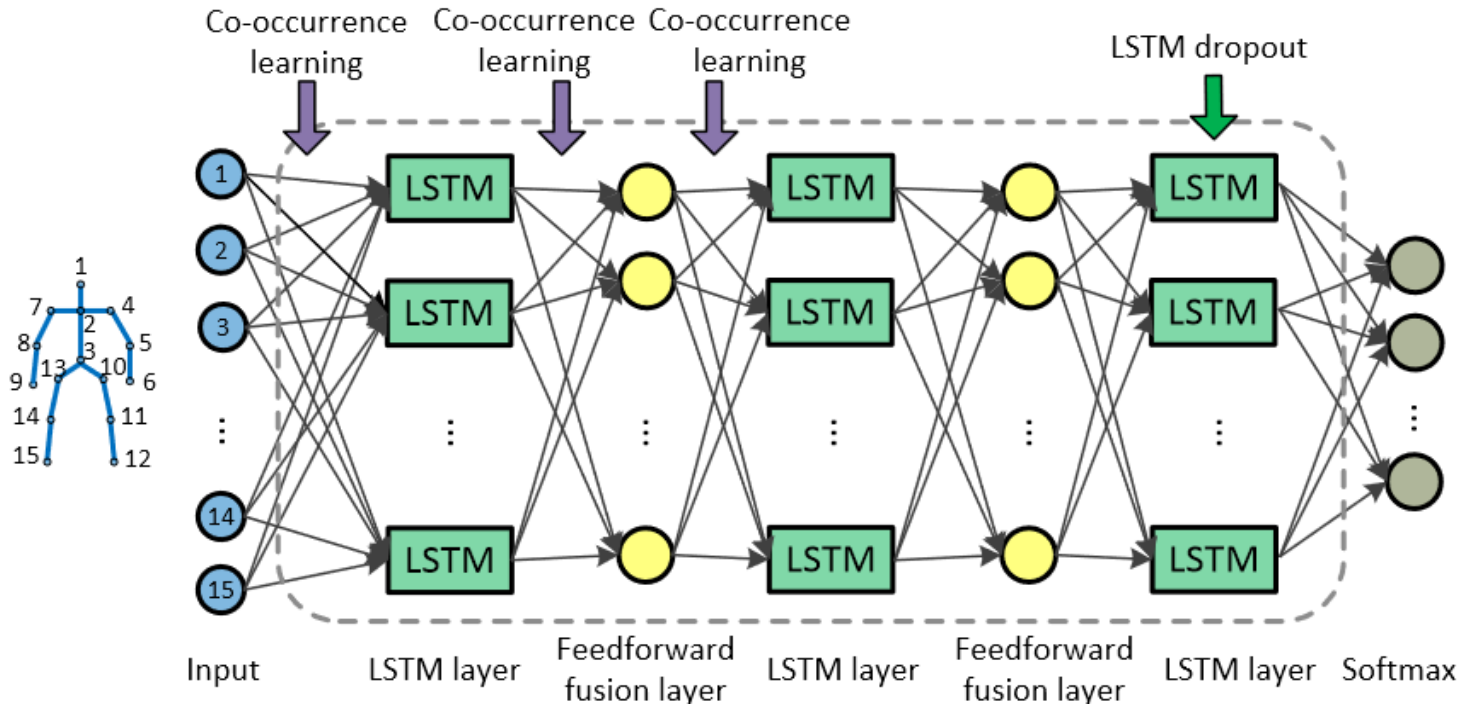
$$\text{As } I^t = \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix},$$

$\delta h^{t-1}$ can be retrieved from $\delta I^t$

$$\delta W^t = \delta z^t \times (I^t)^T$$

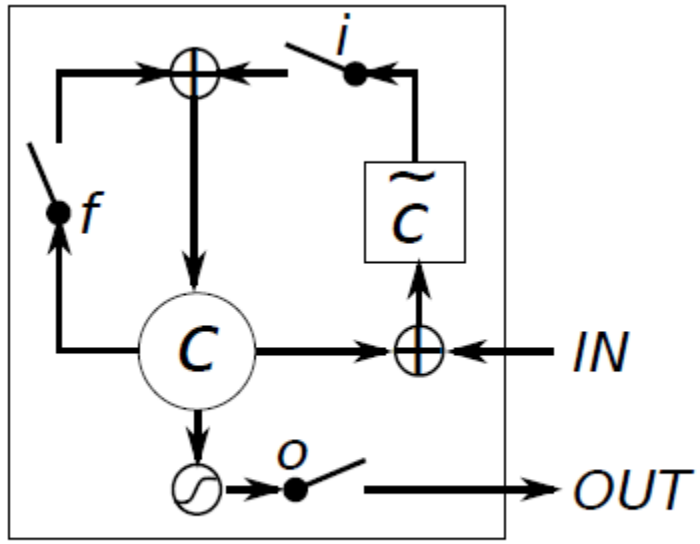
# Deep LSTM

› A Complex Net

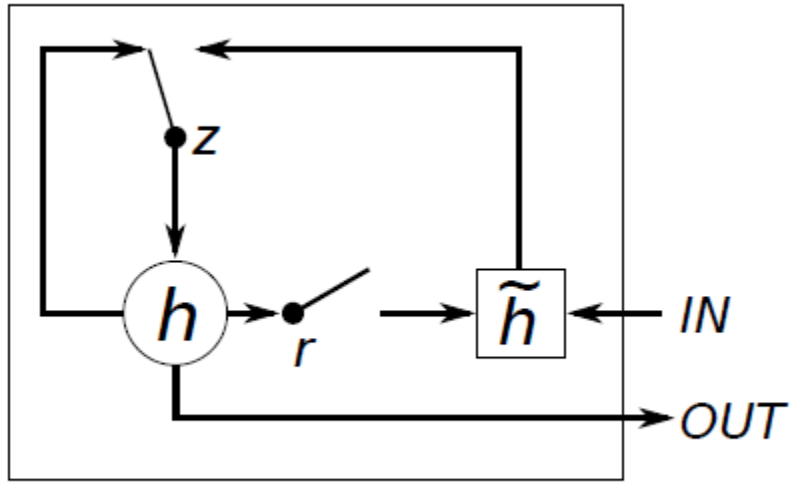


# GRU - Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

› LSTM vs GRU

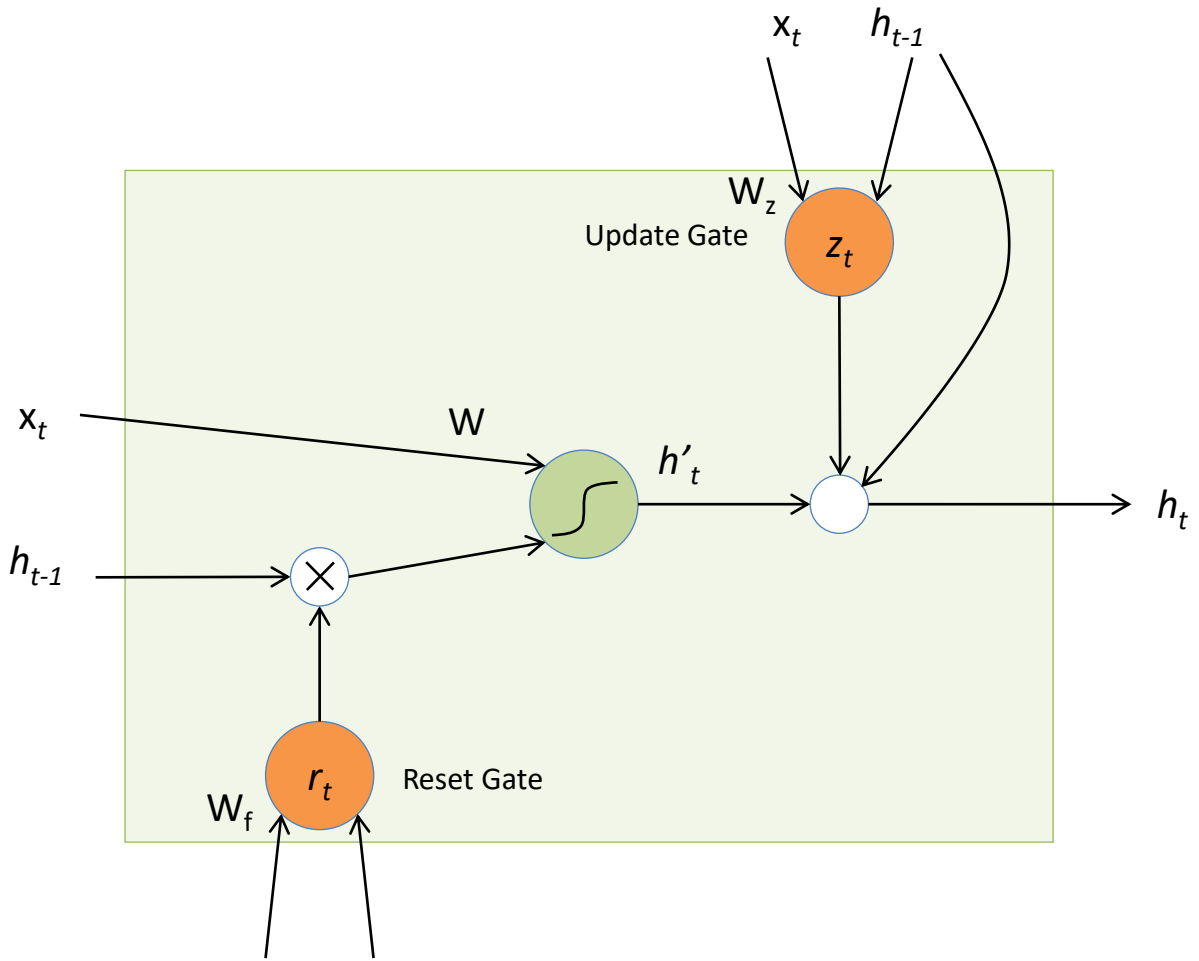



(a) Long Short-Term Memory




(b) Gated Recurrent Unit

# GRU



# GRU - Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014

› Activation of GRU at t:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j,$$

› Update Gate:

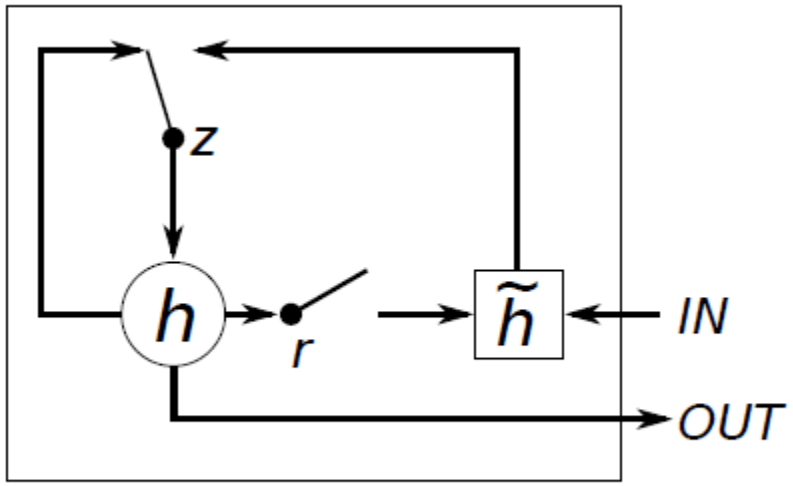
$$z_t^j = \sigma\left(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1}\right)^j$$

› Candidate Activation:

$$\tilde{h}_t^j = \tanh\left(W \mathbf{x}_t + U\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)\right)^j$$

› Reset Gate:

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




(b) Gated Recurrent Unit

# GRU - Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014

- GRU vs LSTM:
  - GRU: “update” and “reset” gates
  - LSTM: “input”, “output”, and “forget” gates
  - No internal memory state **c** in GRU
  - No nonlinearity (sigmoid) before the output gate in GRU

# LSTM - LSTM: A Search Space Odyssey, IEEE T-NNLS, 2017

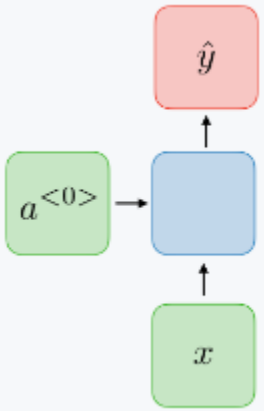
- LSTM Evolution:
  - Initial version (1997):
    - Not forget cell (unity gain), Constant Error Carousel (CEC)
    - Real-Time Recurrent Learning (RTRL)
    - Back-Propagation Through Time (BPTT), Full for cell, truncated for others
  - Forget Gate (1999)
  - Peephole Connection (2000)
    - Memory Cell to gates connection
  - Full Gradient (2000)
    - Full BPTT
  - ---

# RNN Family – Applications

## One-to-One (Language Modelling)

One-to-one

$T_x = T_y = 1$



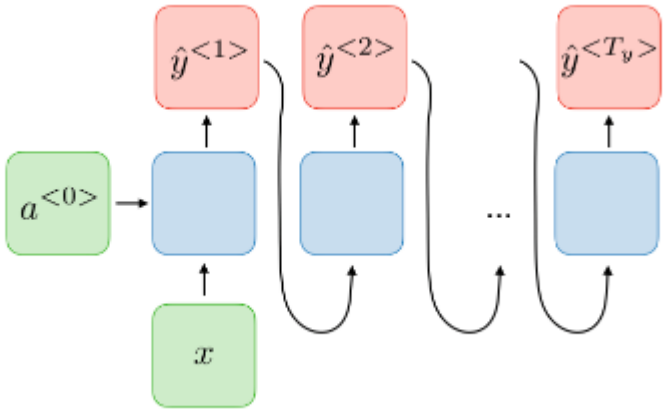

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

# RNN Family – Applications

## One-to-Many (Music Generation/Image Captioning)

One-to-many

$T_x = 1, T_y > 1$



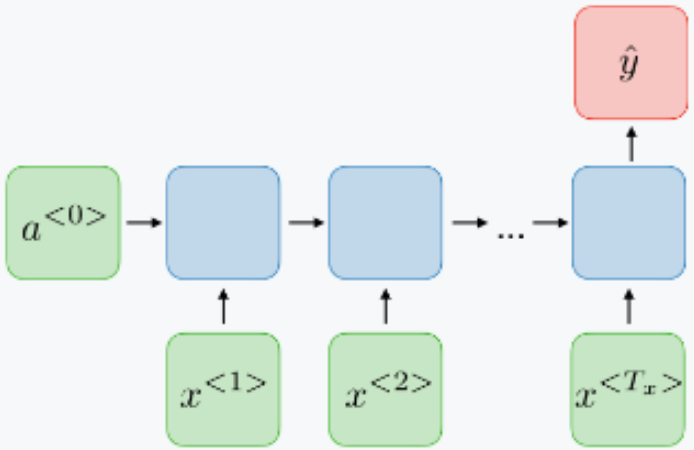

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

# RNN Family – Applications

## Many-to-one (Sentiment classification)

Many-to-one

$T_x > 1, T_y = 1$



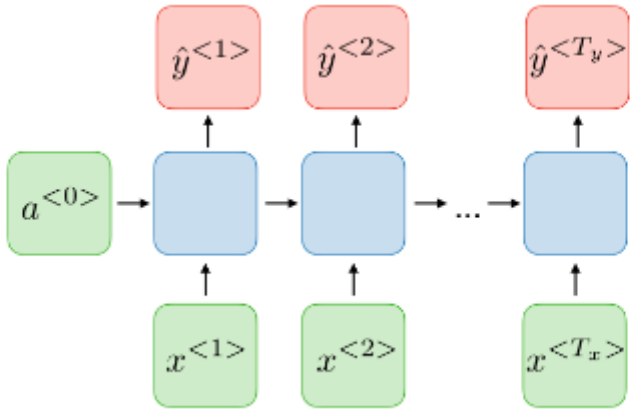

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

# RNN Family – Applications

## Many-to-Many (video classification per Frame)

Many-to-many

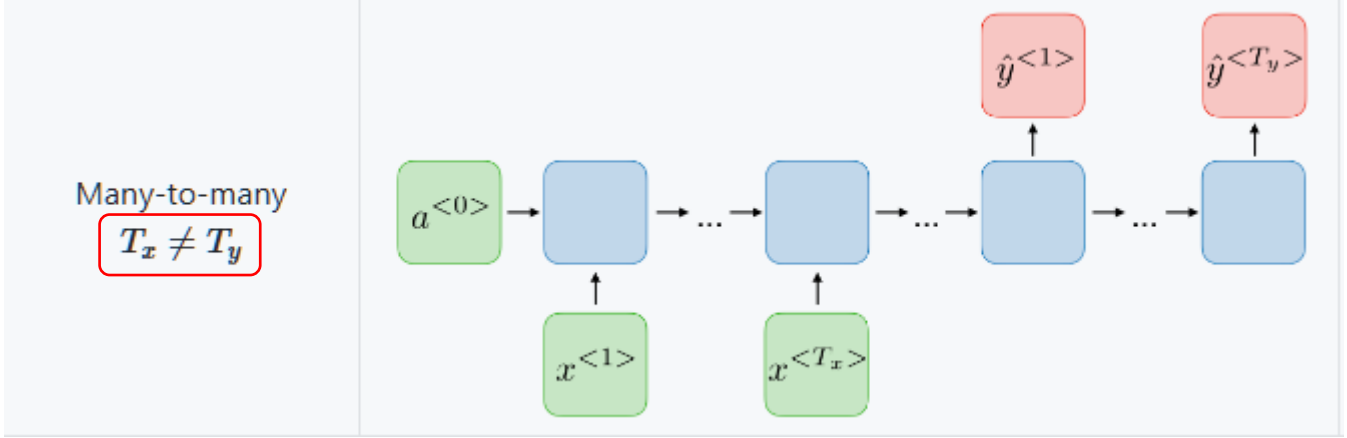
$T_x = T_y$




$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

# RNN Family – Applications

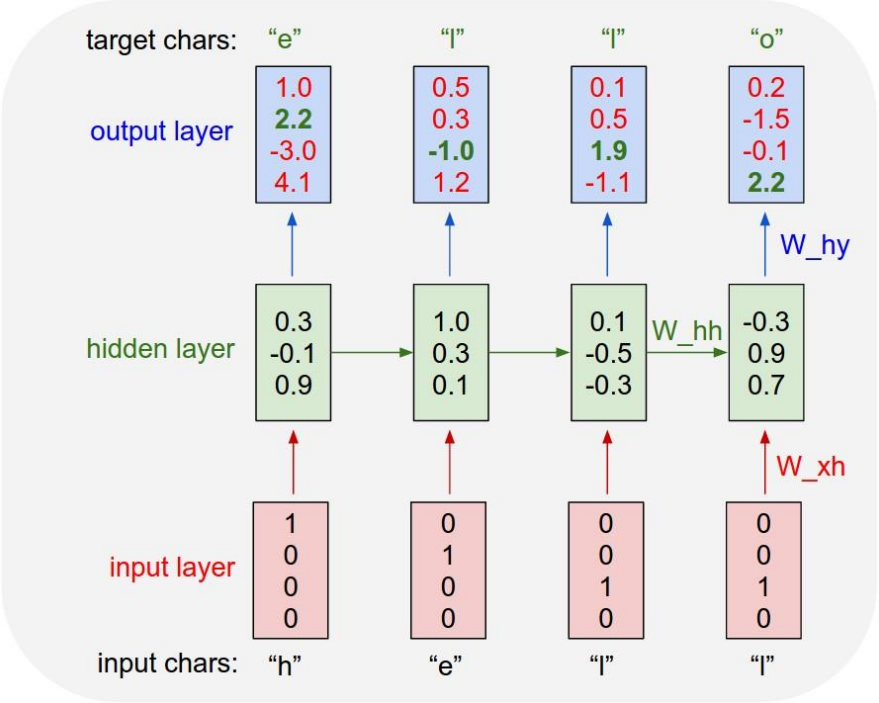
## Many-to-Many (Machine Translation)




$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

# RNN Family – Applications

- Character-Level Language Models, P(Next|Previous) >>
  - Goal: Predict the next character given the current character
  - Suppose four member codebook h/e/l/o
- Test time: Feed a character and sample (randomly) output and feed it to input
- Generating Baby Names:
  - Input and output: baby name
- Generate fake book/paper



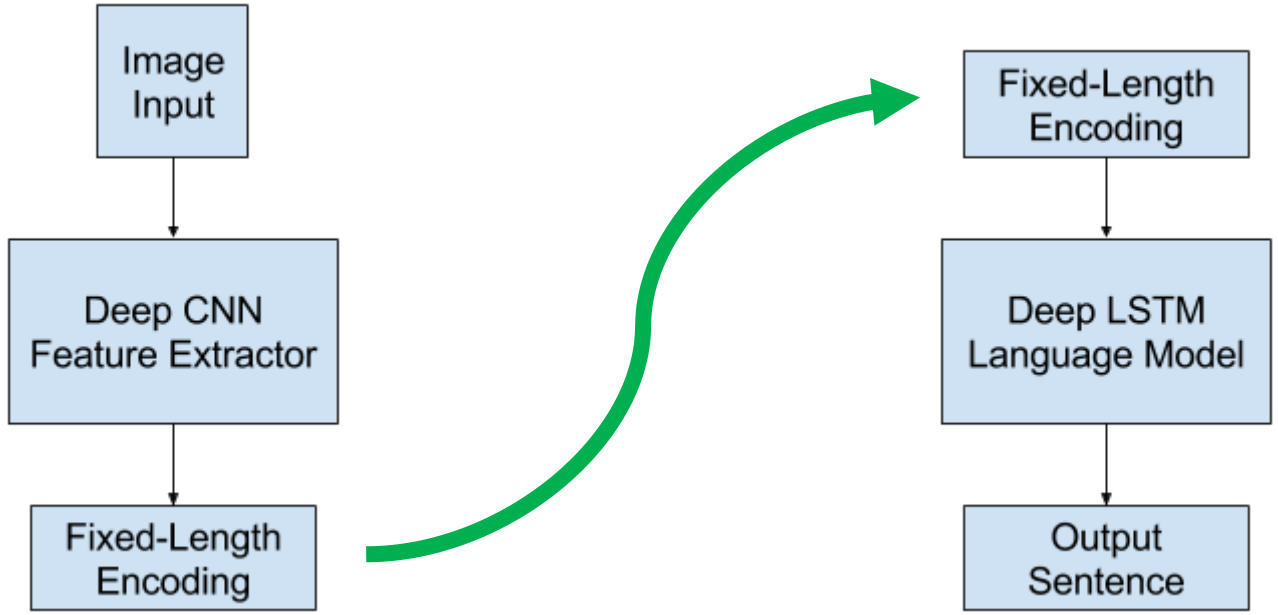

- See this interesting blog!
- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# RNN Family – Applications

- Image Captioning Problem:
- A firefighter extinguishes a fire under the hood of a car.
- A fireman spraying water into the hood of small white car on a jack
- A fireman sprays inside the open hood of small white car, on a jack.
- A fireman using a firehose on a car engine that is up on a carjack.
- Firefighter uses water to extinguish a car that was

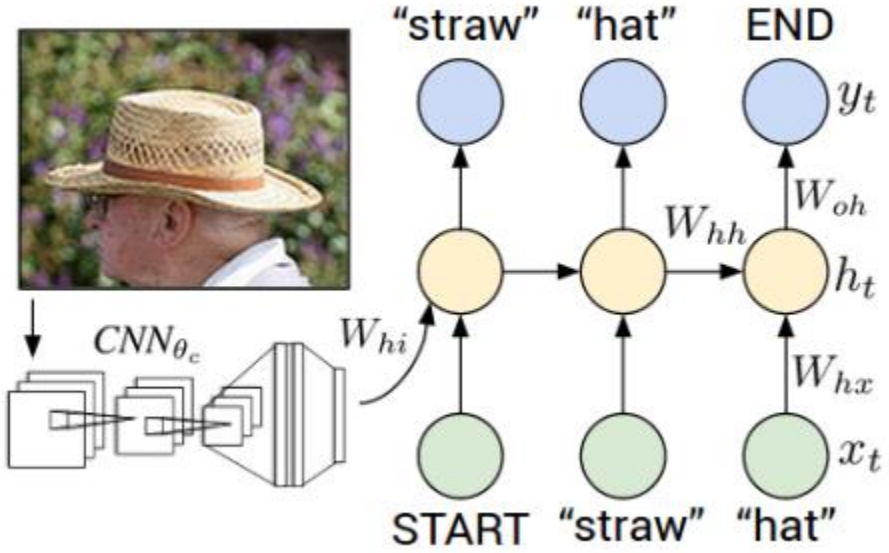
# RNN Family – Applications

- Image Captioning Problem (Deep Solution):
  - Encoder: Encode image to a vector (Latest hidden stage of CNN), **FC4096**
  - Decoder: An RNN which in the first time step receives the encoded output from the encoder and also the **<START>** vector.



# RNN Family – Applications

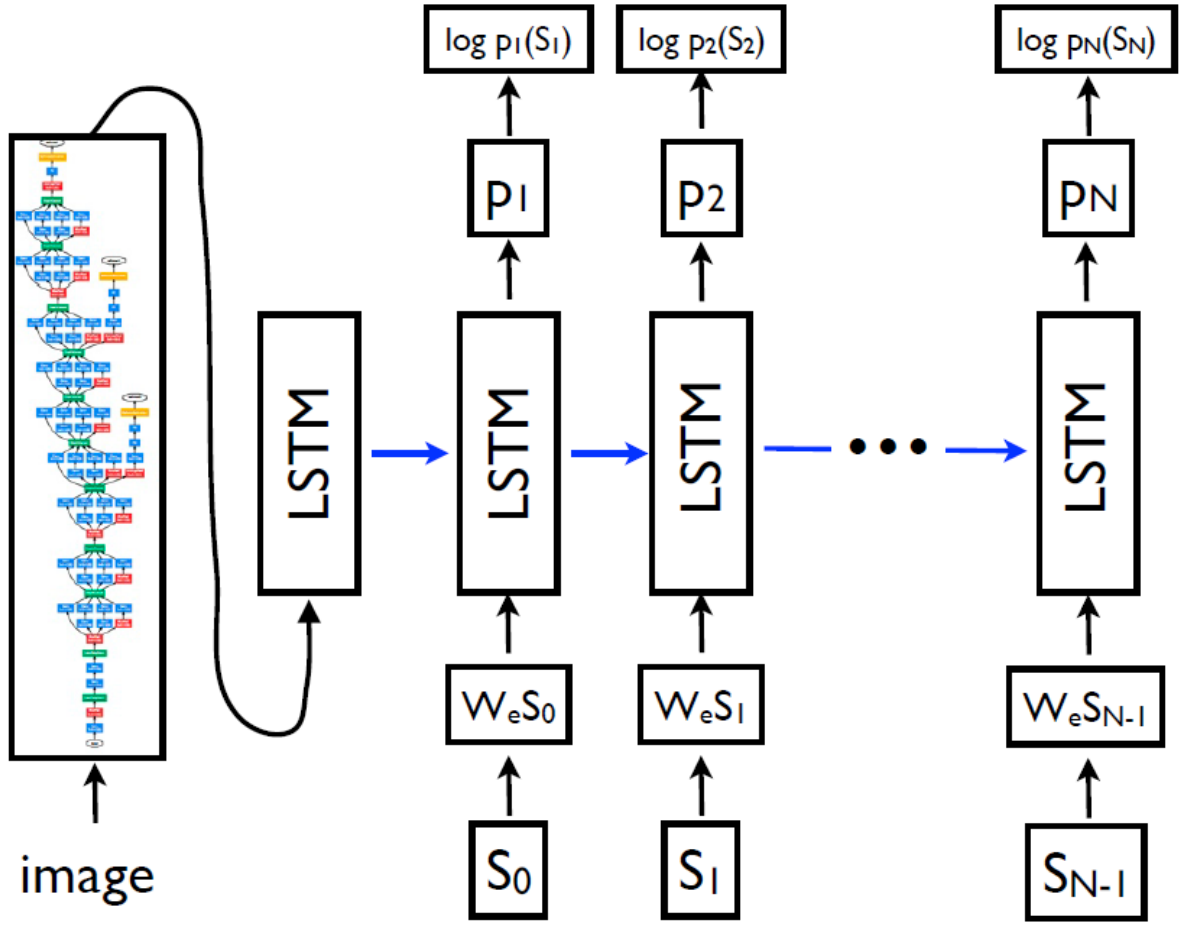
- Image Captioning Problem (Deep Solution):
- Training: Label/Target pair:
  - $\boldsymbol{x}^{(t)}$ : <START>/"straw"/"hat"
  - $\boldsymbol{y}^{(t)}$ : "straw"/"hat"/<END>
- Test:
  - $\boldsymbol{x}^{(0)}$ : <START>
  - Wait for <END> @ output



# Google Image Captioning! - Show and Tell: A Neural Image Caption Generator, 2015

› Unfoldded Model:

$$x_{-1} = CNN(\text{Image})$$
$$x_t = W_e S_t, \quad t \in \{0 \ldots N-1\}$$
$$p_{t+1} = LSTM(x_t), \quad t \in \{0 \ldots N-1\}$$
$$S_0 = <START>$$
$$S_N = <STOP>$$



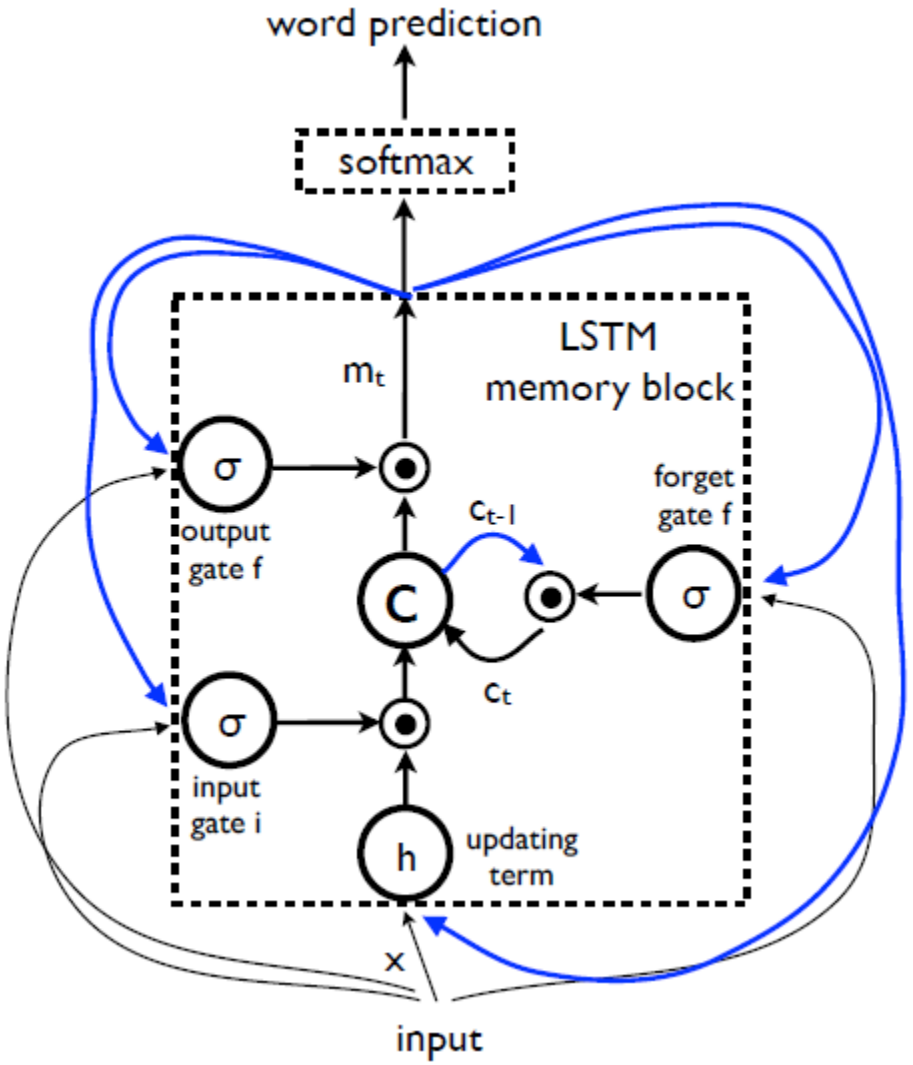
# Google Image Captioning! - Show and Tell: A Neural Image Caption Generator, 2015

› Details – LSTM Model:

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1}) \\
f_t &= \sigma(W_{fx}x_t + W_{fm}m_{t-1}) \\
o_t &= \sigma(W_{ox}x_t + W_{om}m_{t-1}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot h(W_{cx}x_t + W_{cm}m_{t-1}) \\
m_t &= o_t \odot c_t \\
p_{t+1} &= \text{Softmax}(m_t)
\end{aligned}
$$

› $p_{t+1}$: Probability dist. over all wor

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



# Google Image Captioning! - Show and Tell: A Neural Image Caption Generator, 2015

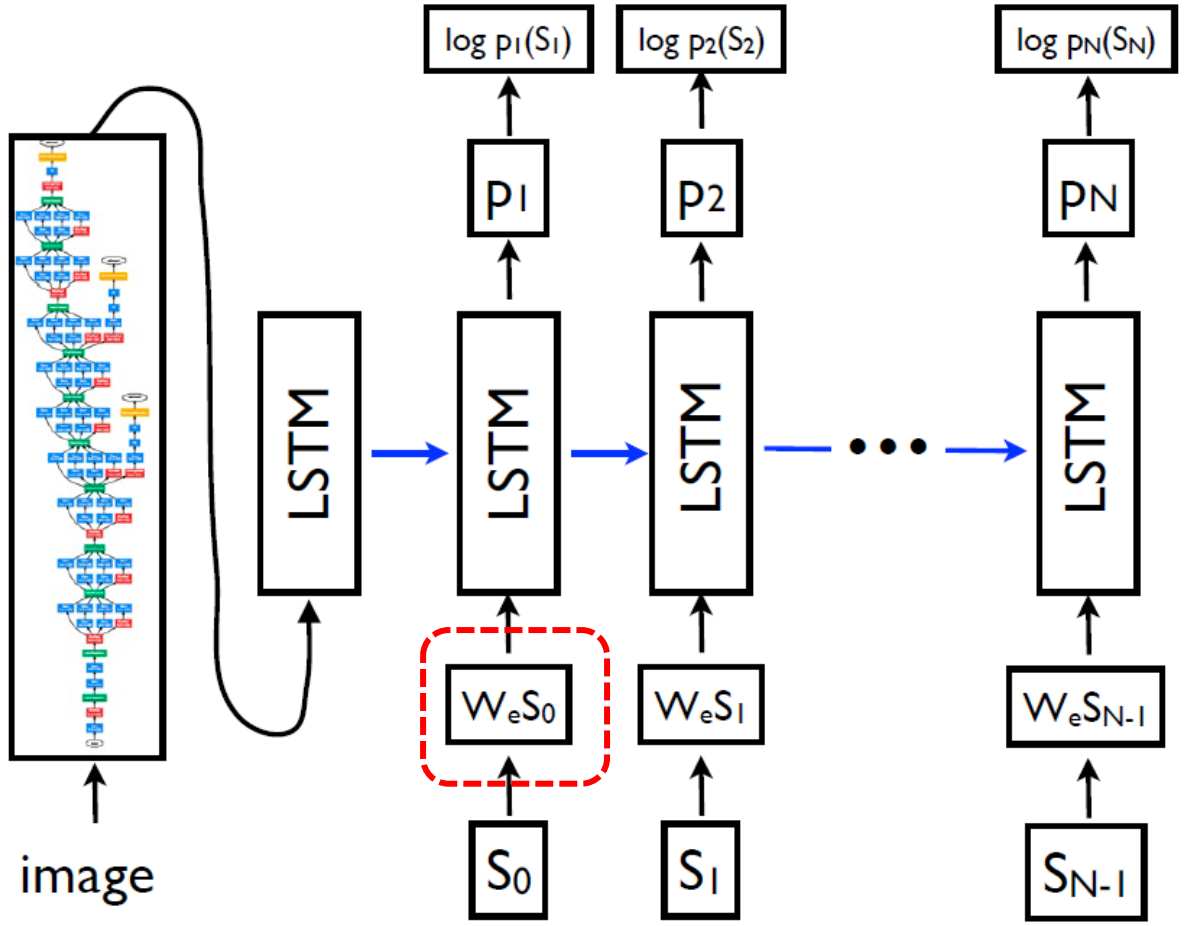
› Word embedding?

– Convert text into numbe

**building a low-dimensional vector representation from corpus of text, which preserves the contextual similarity of words.**

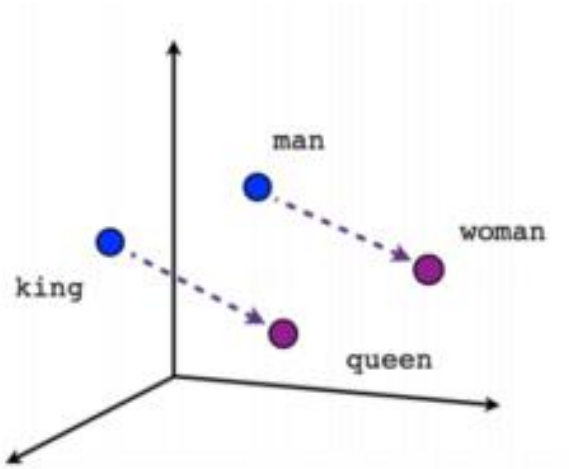
– Same size of FCxxxx

– Initialize and learnable

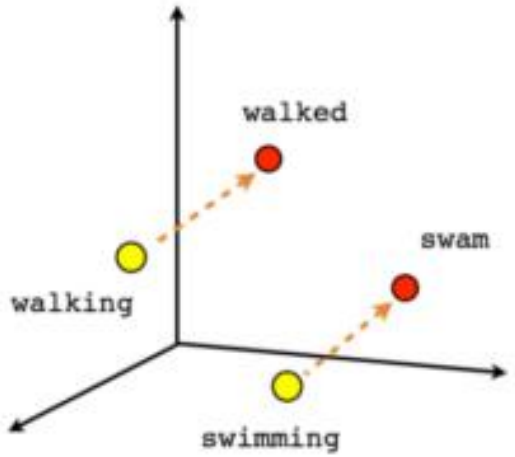


# Google Image Captioning! - Show and Tell: A Neural Image Caption Generator, 2015

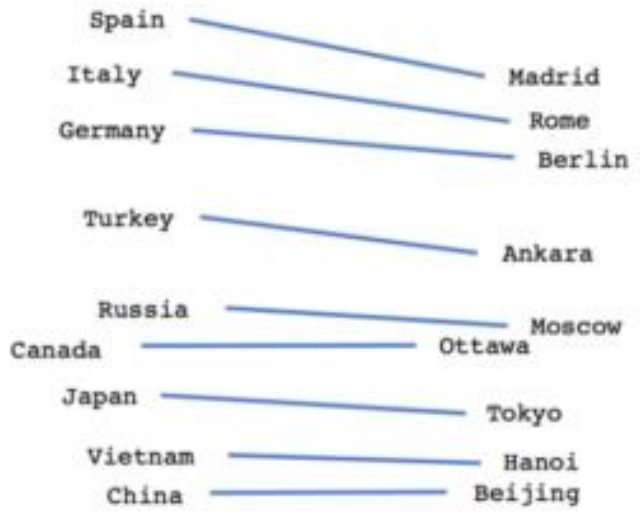
› Word embedding?
  – Convert text into numbers!




Male-Female



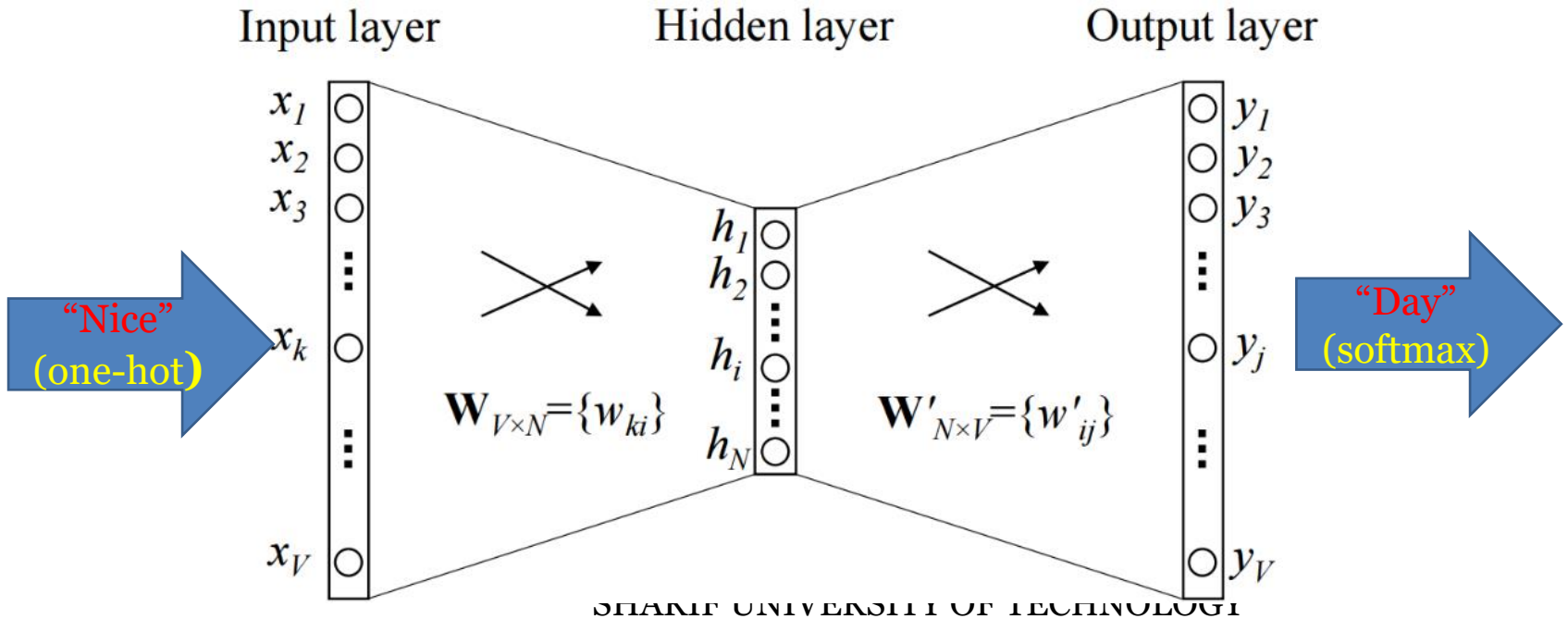

Verb tense




Country-Capital

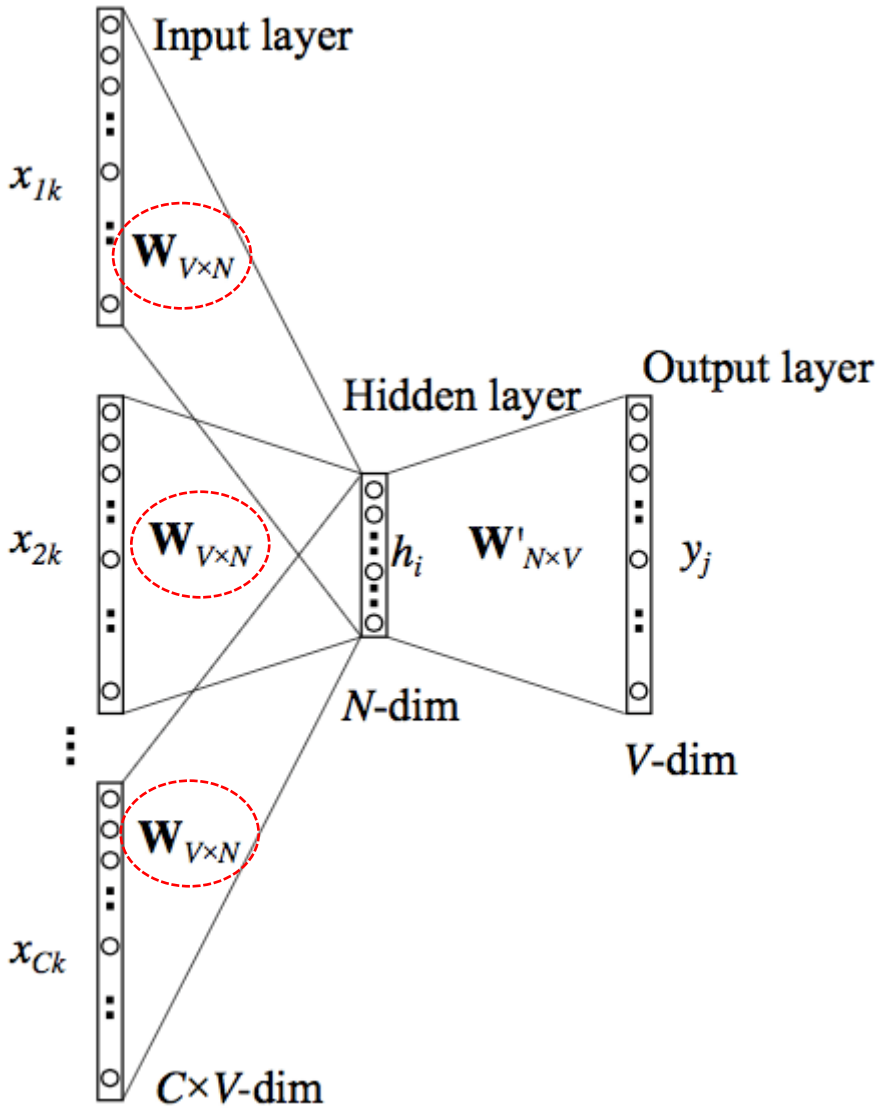
# Word Embedding – word2vec!

- A Simple (Shallow) Network:
  - The input and the output patterns are one-hot encoded vectors
    - In the process of predicting the target word, we learn the vector representation of the target word. "*Have a nice day*", *input: nice, output: day* (The hidden layer is target code)



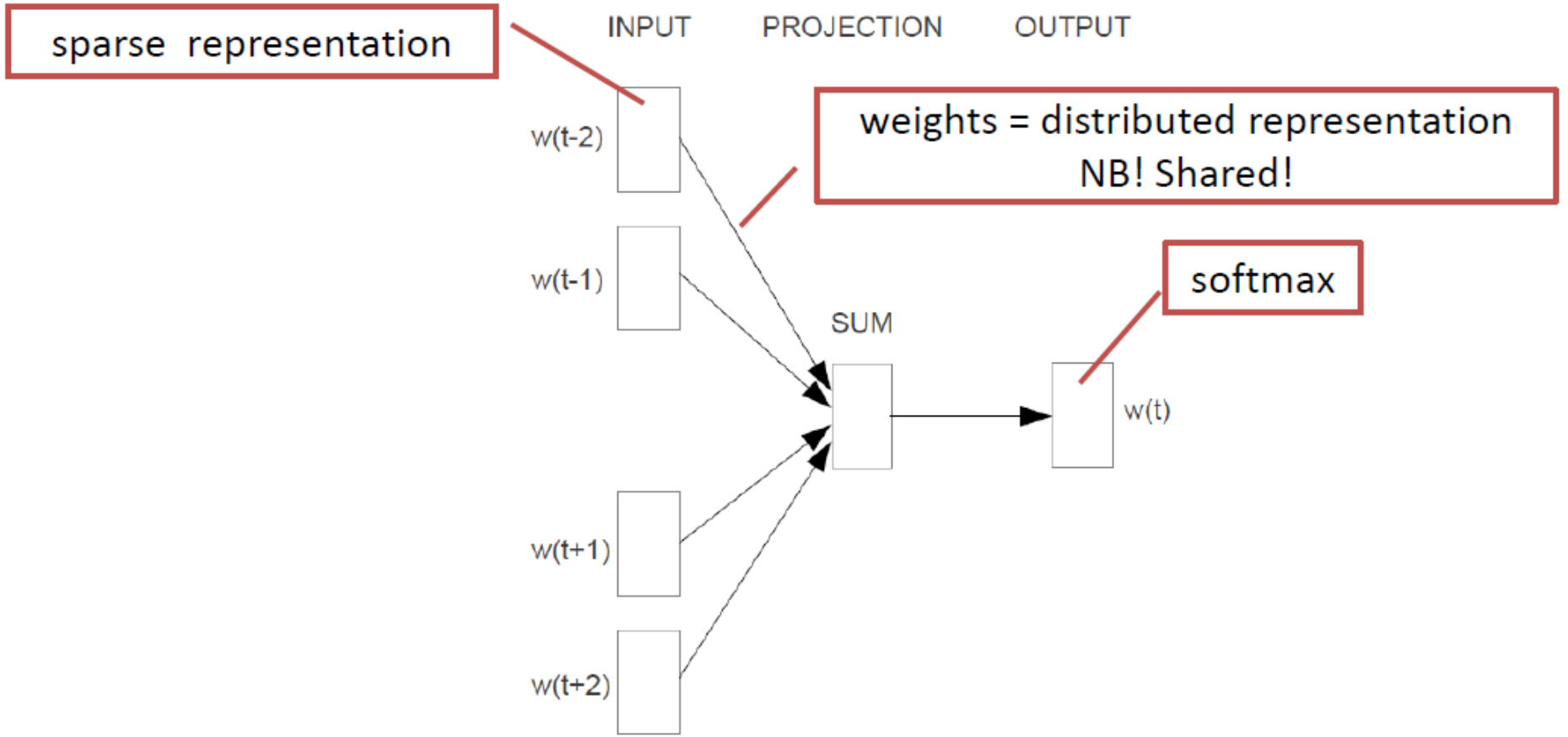
# Word Embedding – word2vec!

- **C**ontinuous **B**ag-**O**f-**W**ords (CBOW)
- Example: *the dog barked at the mailma*
- Input: ['*the*','*barked*','*at*']
- Output: '*dog*'
- See **wevi**: word embedding visual inspe
  https://ronxin.github.io/wevi/



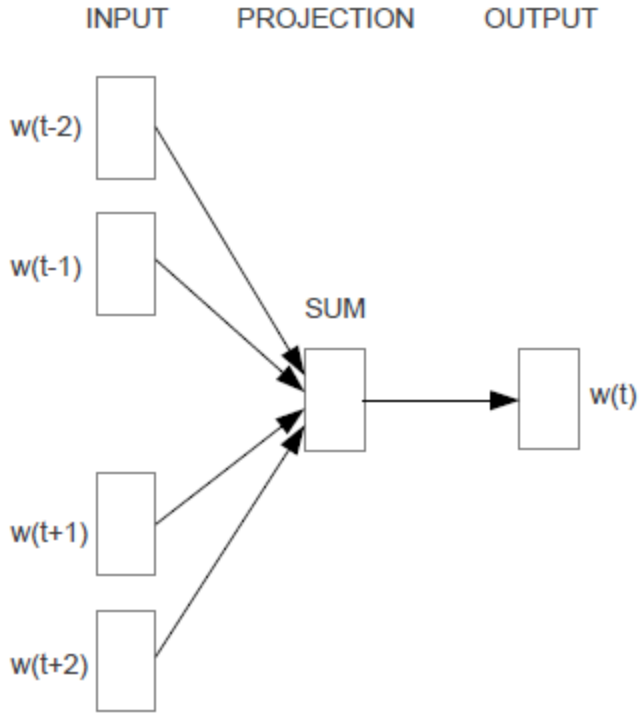
# Google Word2Vec - Efficient Estimation of Word Representations in Vector Space, 2013

› CBOW architecture (NB: No Bias)



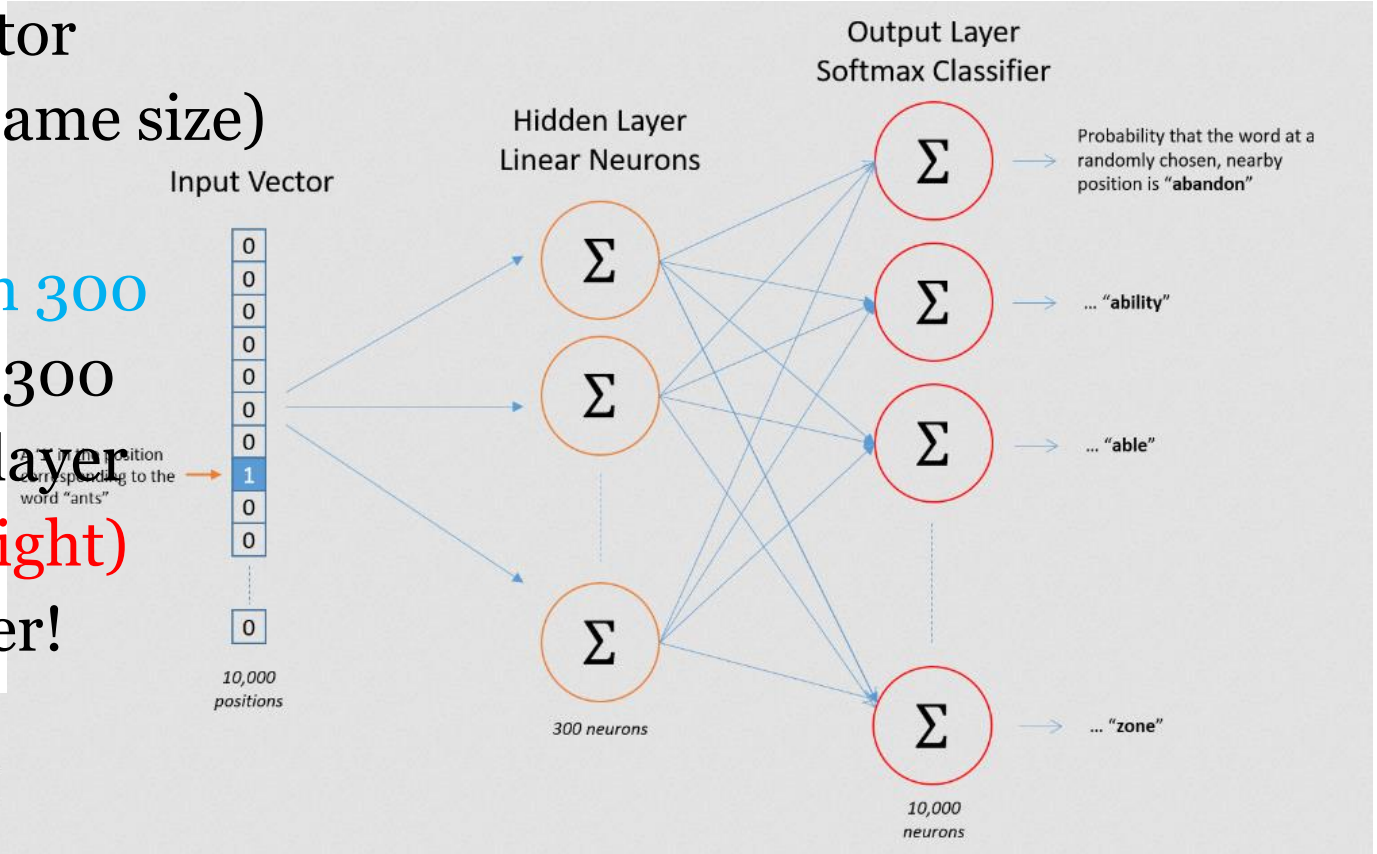
# Google Word2Vec - Efficient Estimation of Word Representations in Vector Space, 2013

- CBOW
- Input: one-hot
- Output: Probability
- "the quick brown fox jumps over the laz
- ["quick", "fox"] → "brown"



# Google Word2Vec - Efficient Estimation of Word Representations in Vector Space, 2013

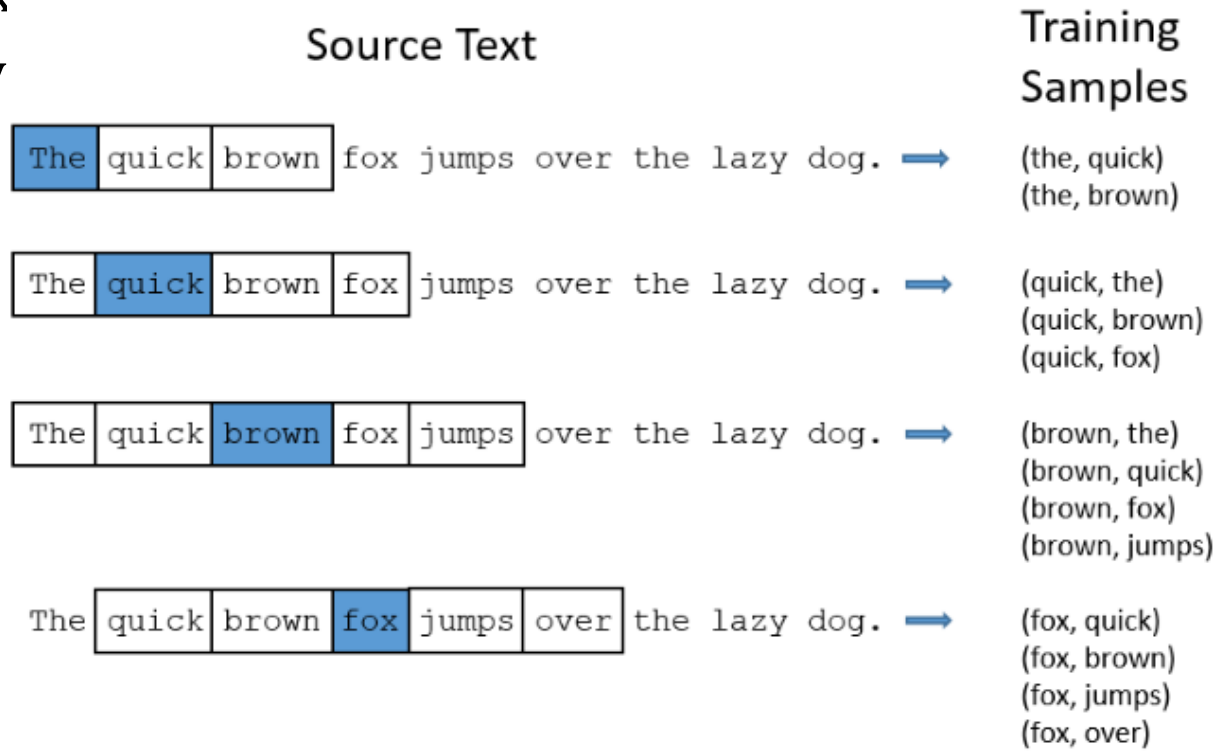
- Skip-Gram architecture:
  - Input: one-hot vector
  - Output: softmax (same size)
  - 10,000 words
  - Codebook of length 300
  - We have a 10000×300 matrix in hidden layer (3M unknown weight)
  - We use hidden layer!




$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

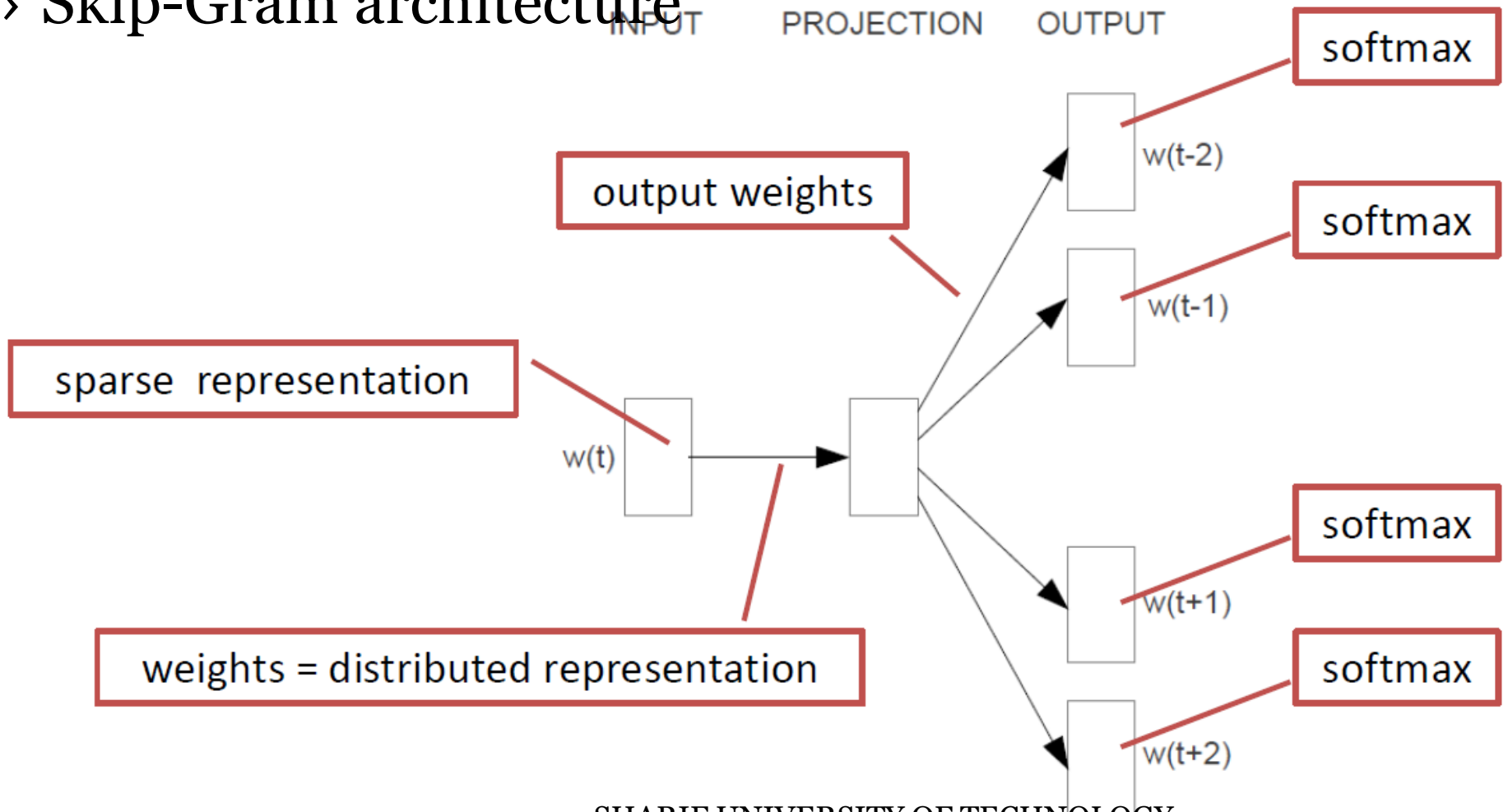
# Word Embedding

- Training:
  - Subsampling Frequent Words (windows size 2):
    - input: “the” target “quick"
    - input: “the” target “brow
    - ….
    - input: “fox” target “over”
    - ….



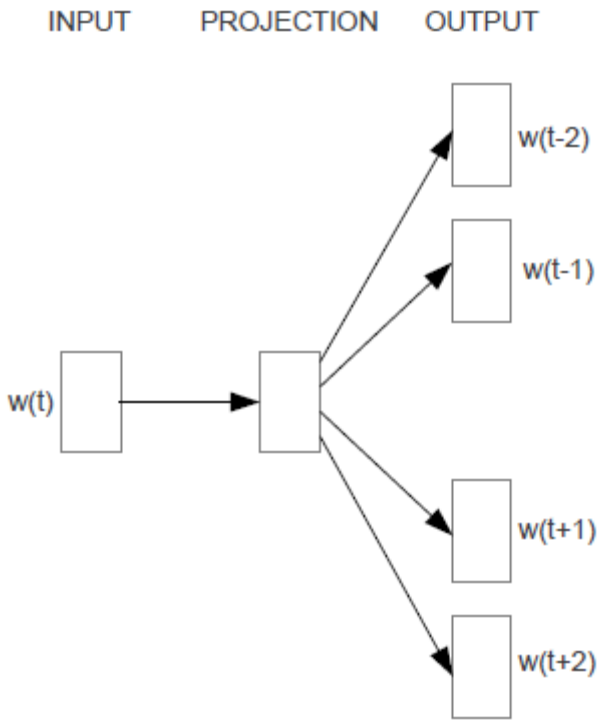
# Google Word2Vec - Efficient Estimation of Word Representations in Vector Space, 2013

› Skip-Gram architecture



# Google Word2Vec - Efficient Estimation of Word Representations in Vector Space, 2013

- Skip-Gram
- Input: one-hot
- Output: Probability
- “the quick brown fox jumps over the la
- “brown” → [“quick”, “fox”]



# WOW! Works!

› RNN Generated Caption!

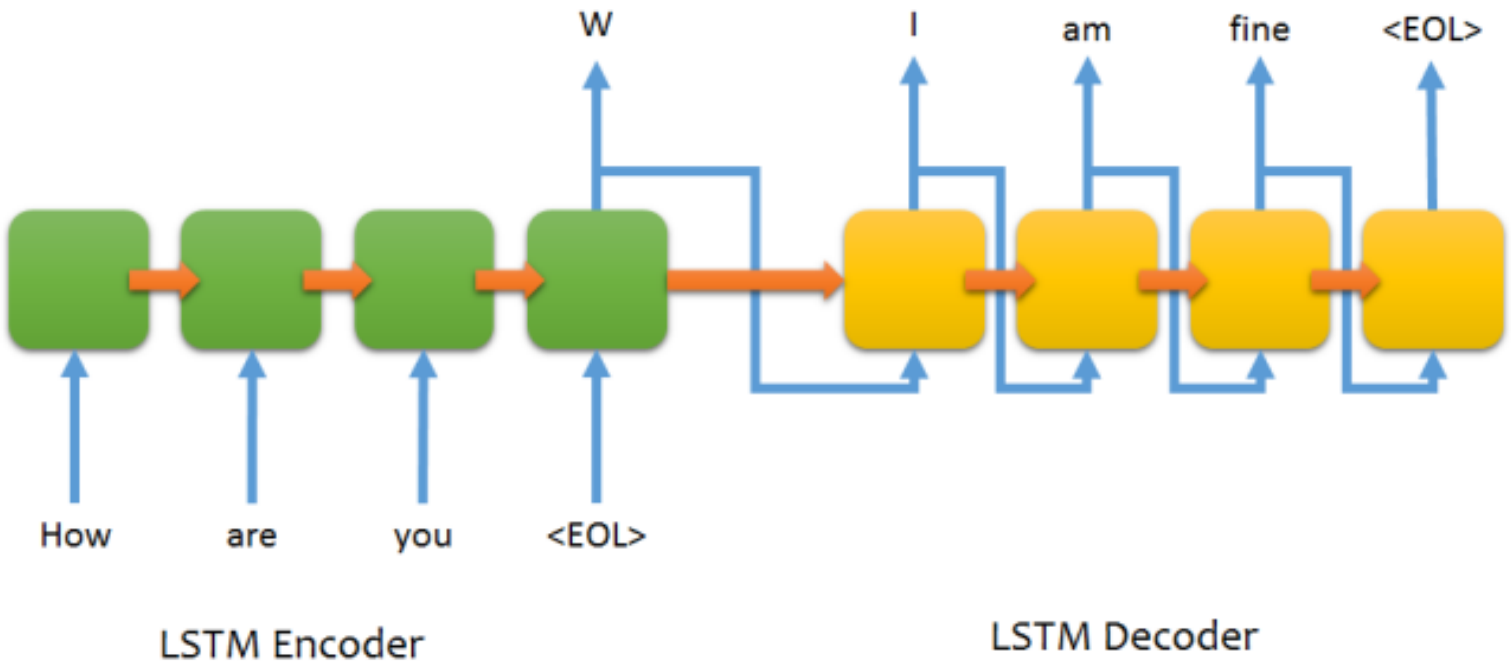
› Browse *neuraltalk*:

› https://github.com/karpathy/neur

a cow is standing in the middle of a street
logprob: -8.84

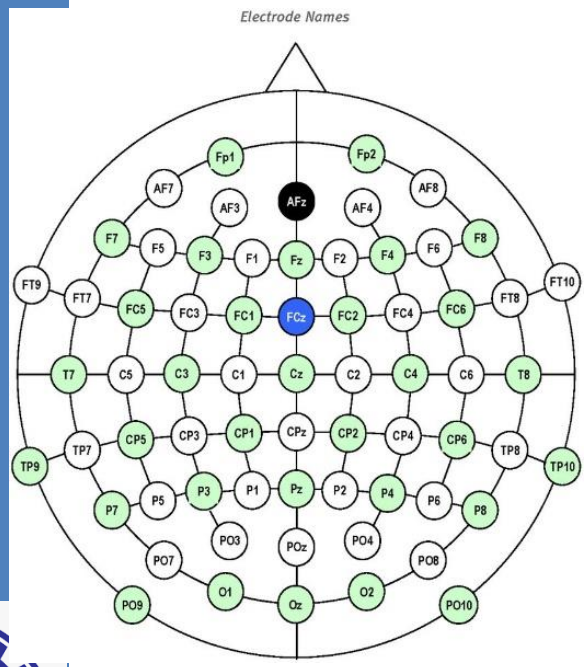
# Sequence to Sequence Modeling (seq2seq)

› Machine Translation



# Medical Application – EEG Classification

› EEG – Brain Electrical Activity acquired by several sensors
  – Sensors → Signal Bank → Topographic Map



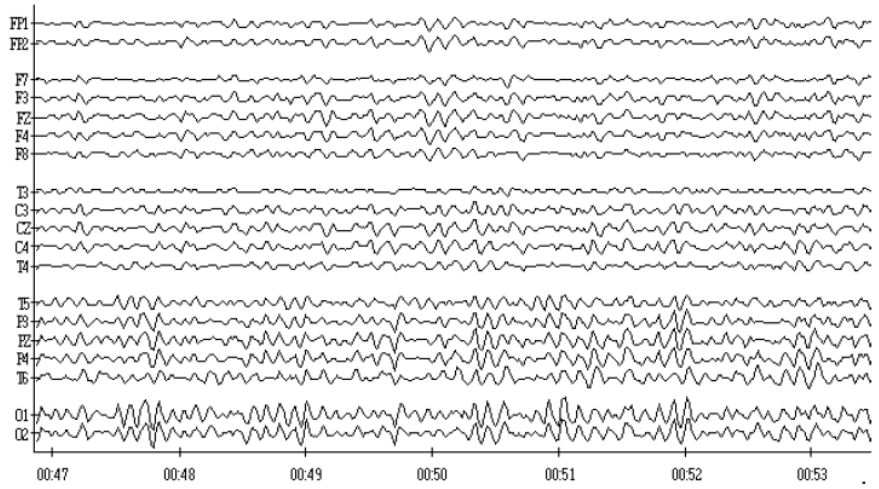
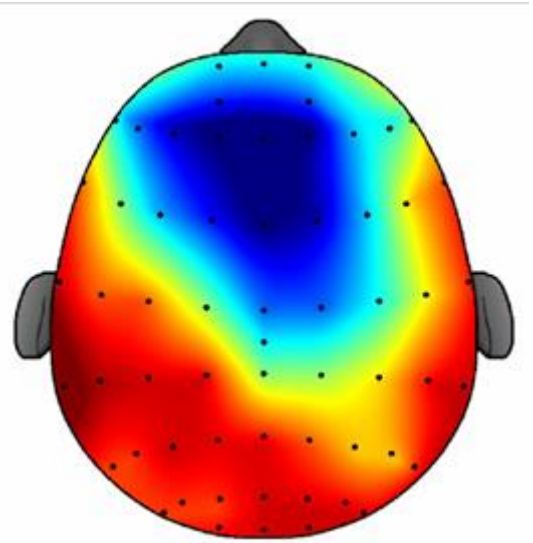
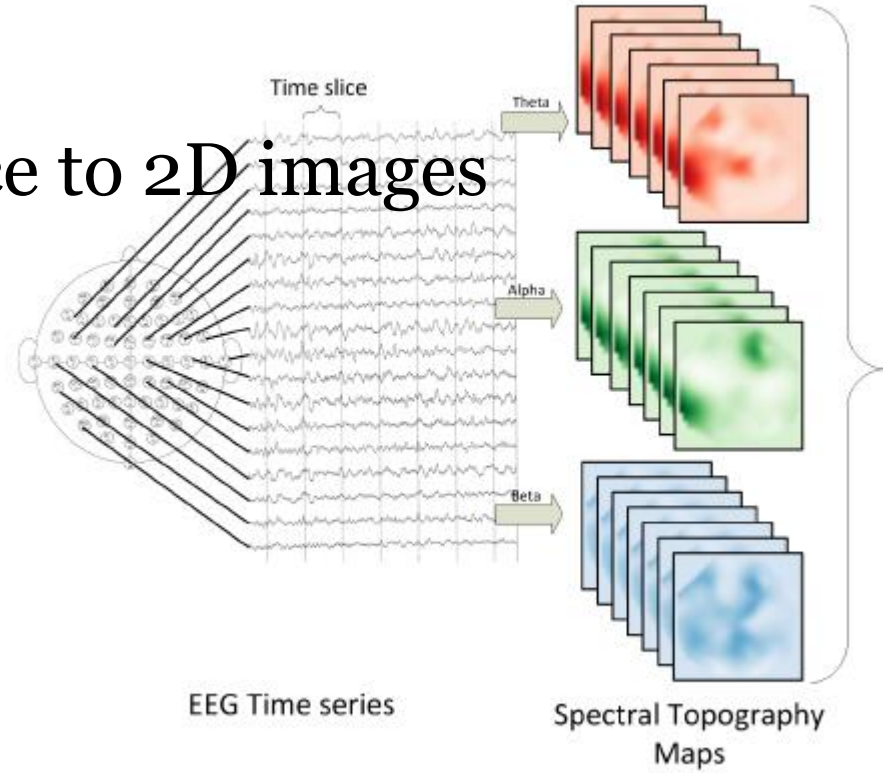




Figure 5. EEG data from 19 channels recorded from an adult.



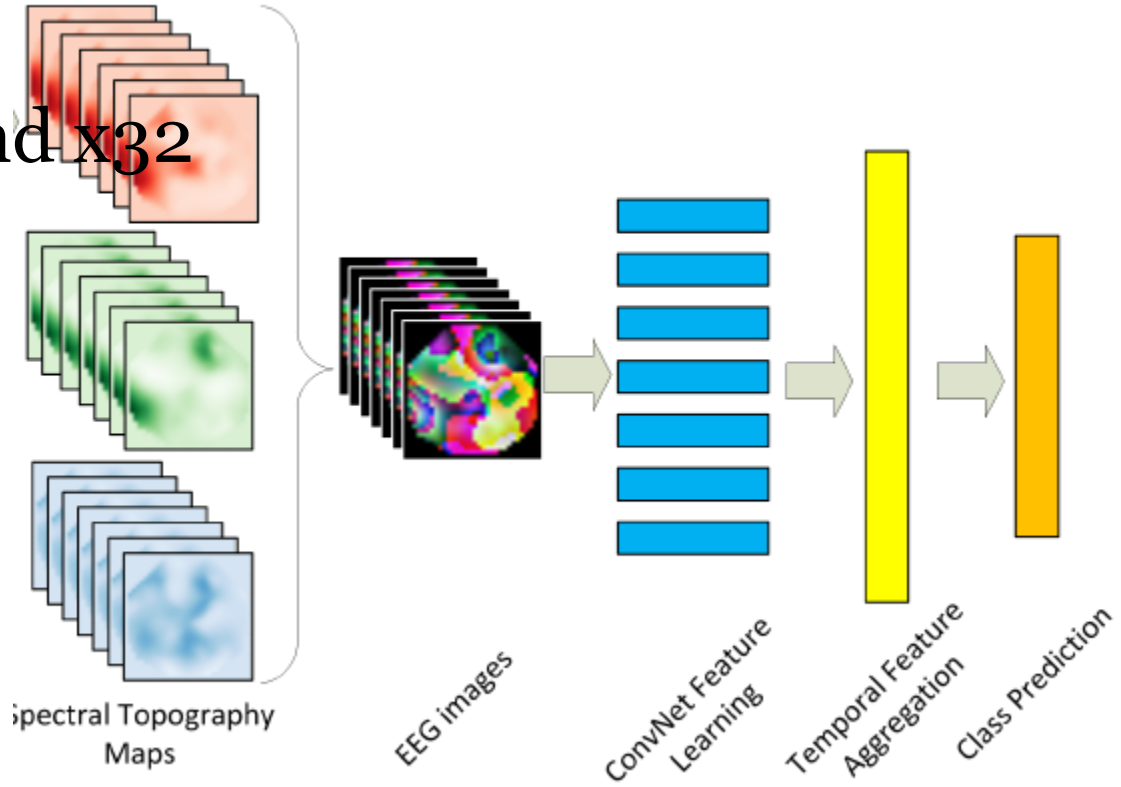
# EEG Classification - Learning Representations From EEG With Deep Recurrent-Convolutional Neural Networks, ICLR2016

- Sliding Windows
- Three Frequency filtering
- Spectral power for each sensor
- Interpolate brain manifold surface to 2D images



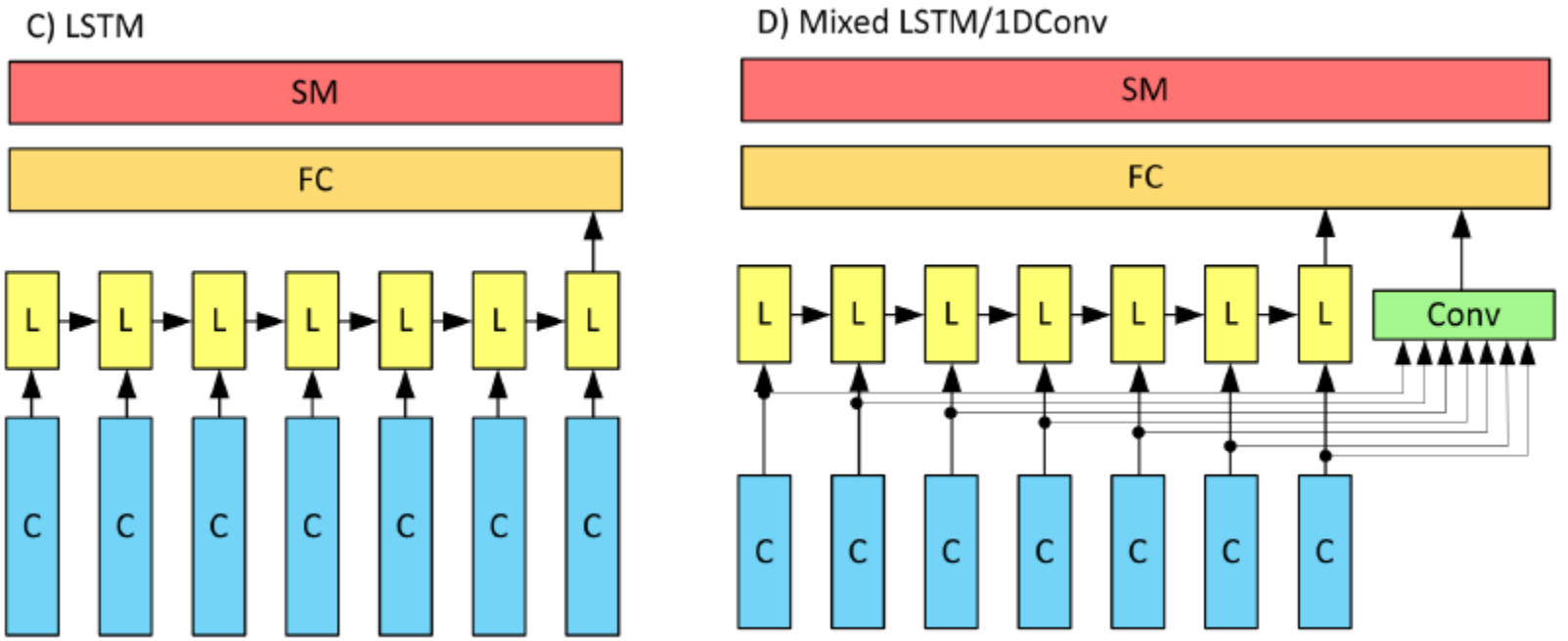
# EEG Classification - Learning Representations From EEG With Deep Recurrent-Convolutional Neural Networks, ICLR2016

- Create a RGB image of brain
- VGG ConvNet
- 1D Temporal Conv. x16 and x32
- LSTM



# EEG Classification - Learning Representations From EEG With Deep Recurrent-Convolutional Neural Networks, ICLR2016

› LSTM:



# RNN-LSM - Advances in Optimizing Recurrent Networks, 2012

› Summary:
  - RNN family handle variable length input/output
  - (Single/Multi) (input/output) scenario are possible.
  - LSTM solve the curse of vanishing gradient
  - Mini-batch/Dropout/Momentum/Regularization/… help
  - Gradient Clipping handle gradient exploding
  - ReLU (Sparse Gradient)
  - Leaky integration:

$$h_{t,i} = \alpha_i h_{t-1,i} + (1 - \alpha_i) F_i(h_{t-1}, x_t)$$