Introduction to Computer Vision

Lecture 9 - Temporal Analysis I

Prof. He Wang

Logistics

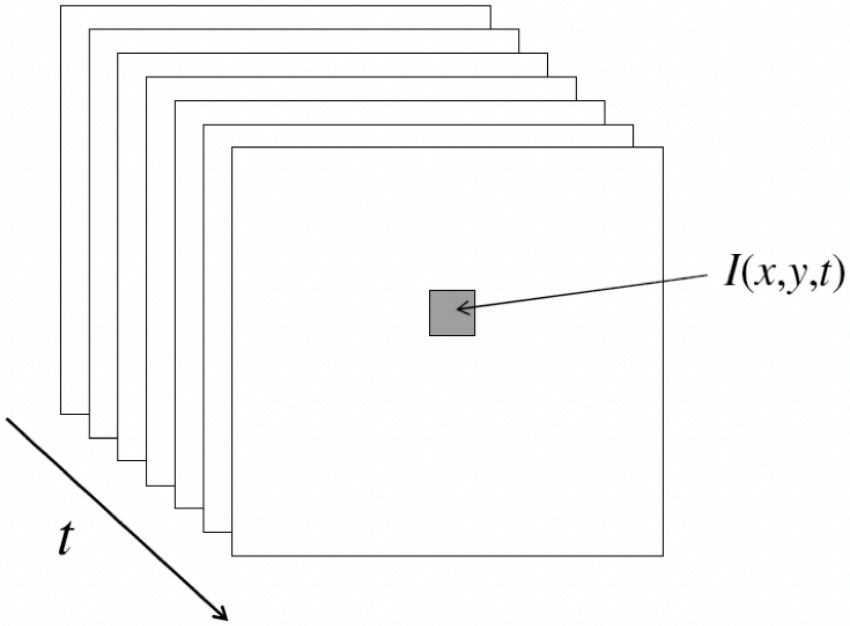
- Assignment 3 will release later this week.

Motion and Optical Flow

Some slides are borrowed from Stanford CS131.

From Single Image to Video

- A video is a sequence of frames captured over time
- Now our image data is a function of space (x, y) and time (t)



Today let's focus on motions between two consecutive frames!

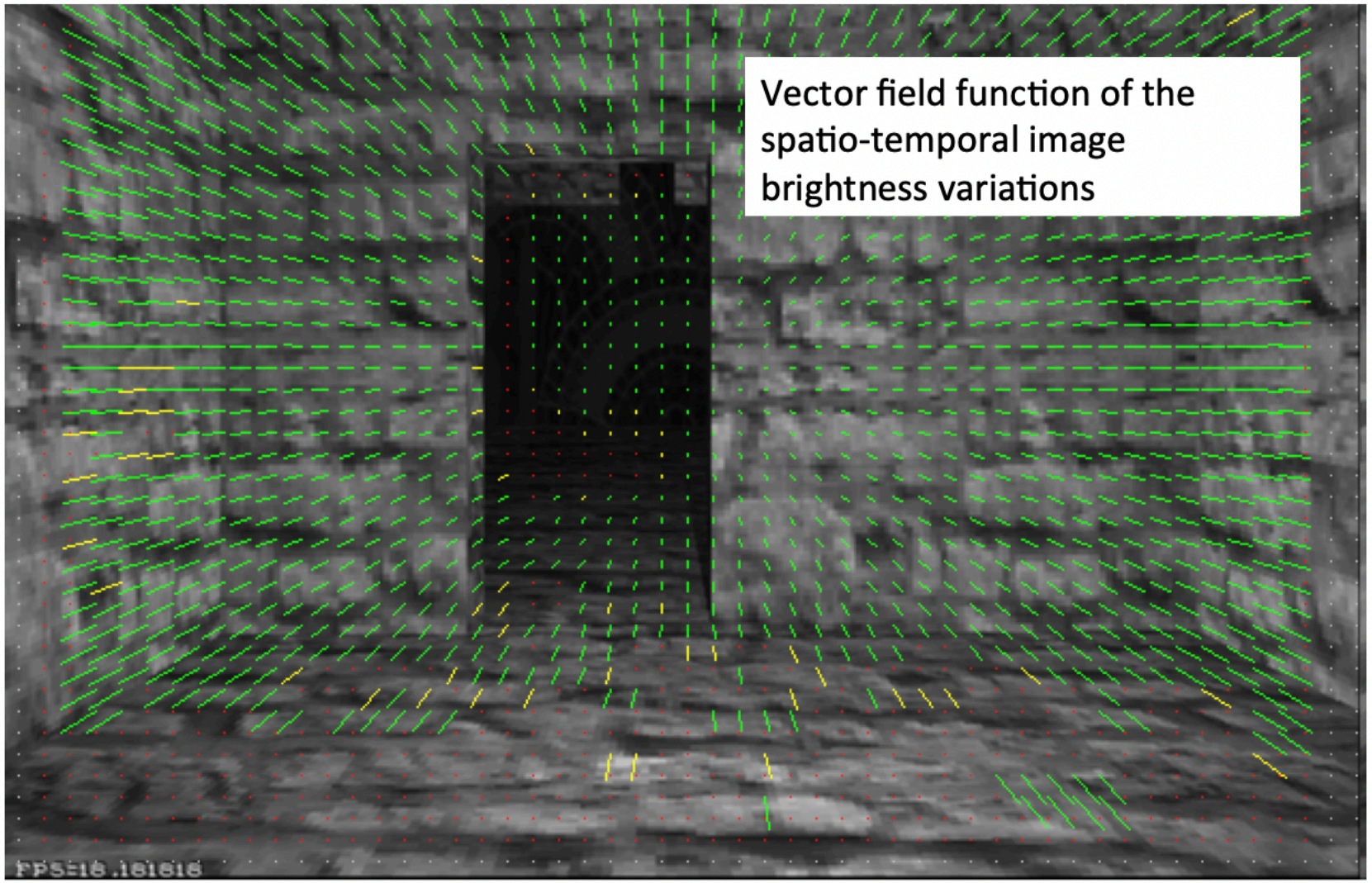
Optical Flow

- Definition: optical flow is the *apparent* motion of brightness patterns in the image

- Note: apparent motion can be caused by lighting changes without any actual motion
  - Think of a uniform rotating sphere under fixed lighting vs. a stationary sphere under moving illumination

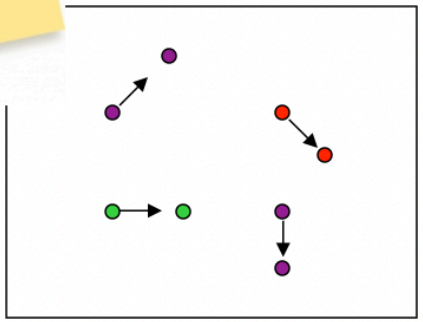
GOAL: Recover image motion at each pixel from optical flow

Optical Flow

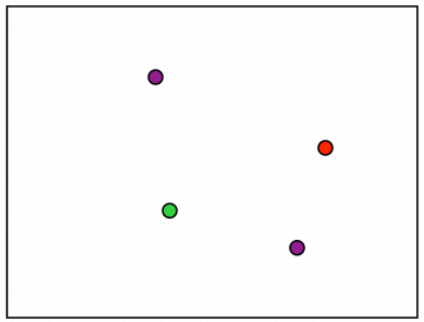


Picture courtesy of Selim Temizer - Learning and Intelligent Systems (LIS) Group, MIT

Estimating Optical Flow



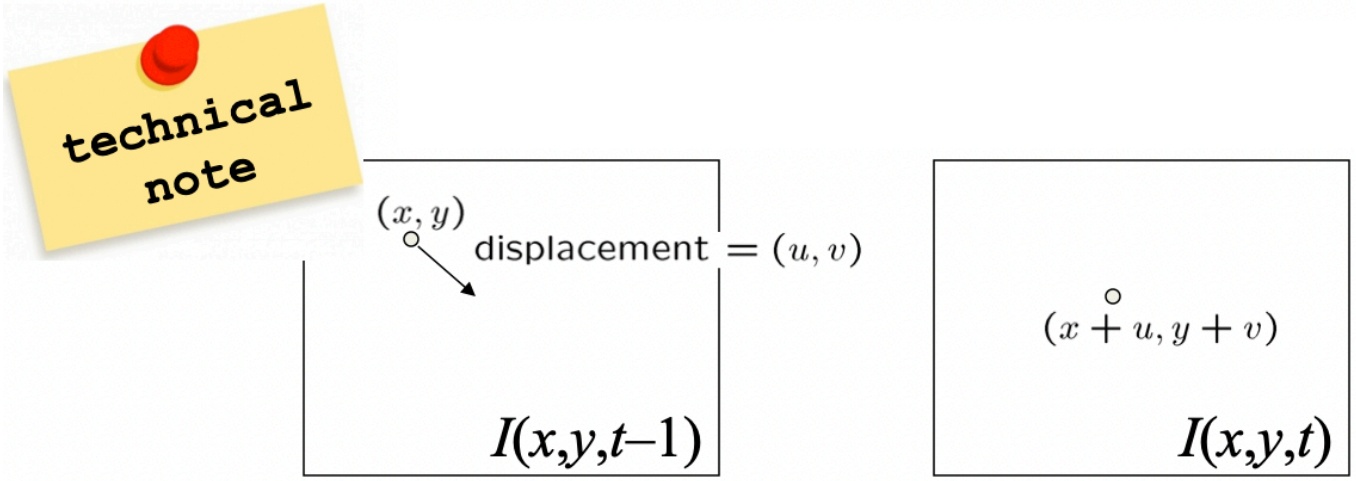
$I(x,y,t-1)$



$I(x,y,t)$

- Given two subsequent frames, estimate the apparent motion field u(x,y), v(x,y) between them
- Key assumptions
  - **Brightness constancy:** projection of the same point looks the same in every frame
  - **Small motion:** points do not move very far
  - **Spatial coherence:** points move like their neighbors

The Brightness Constancy Constraint



- **Brightness Constancy Equation:**

$$I(x,y,t-1) = I(x+u(x,y), y+v(x,y), t)$$

Linearizing the right side using Taylor expansion:

Image derivative along x

$$I(x+u,y+v,t) \approx I(x,y,t-1) + I_x \cdot u(x,y) + I_y \cdot v(x,y) + I_t$$

$$I(x+u,y+v,t) - I(x,y,t-1) = I_x \cdot u(x,y) + I_y \cdot v(x,y) + I_t$$

Hence, $I_x \cdot u + I_y \cdot v + I_t \approx 0 \quad \rightarrow \nabla I \cdot \left[u \; v\right]^T + I_t = 0$

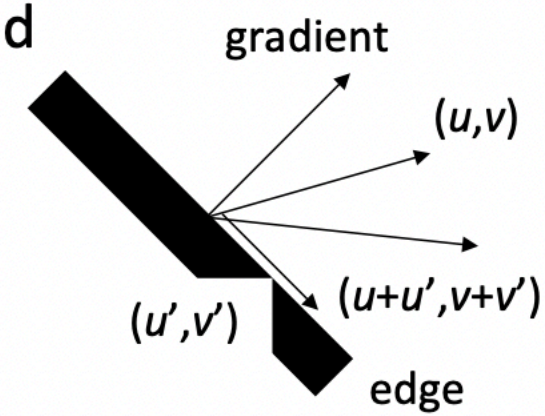
The Brightness Constancy Constraint

Can we use this equation to recover image motion (u,v) at each pixel?

$$\nabla I \cdot \begin{bmatrix} u & v \end{bmatrix}^T + I_t = 0$$

- How many equations and unknowns per pixel?
  - One equation (this is a scalar equation!), two unknowns (u,v)

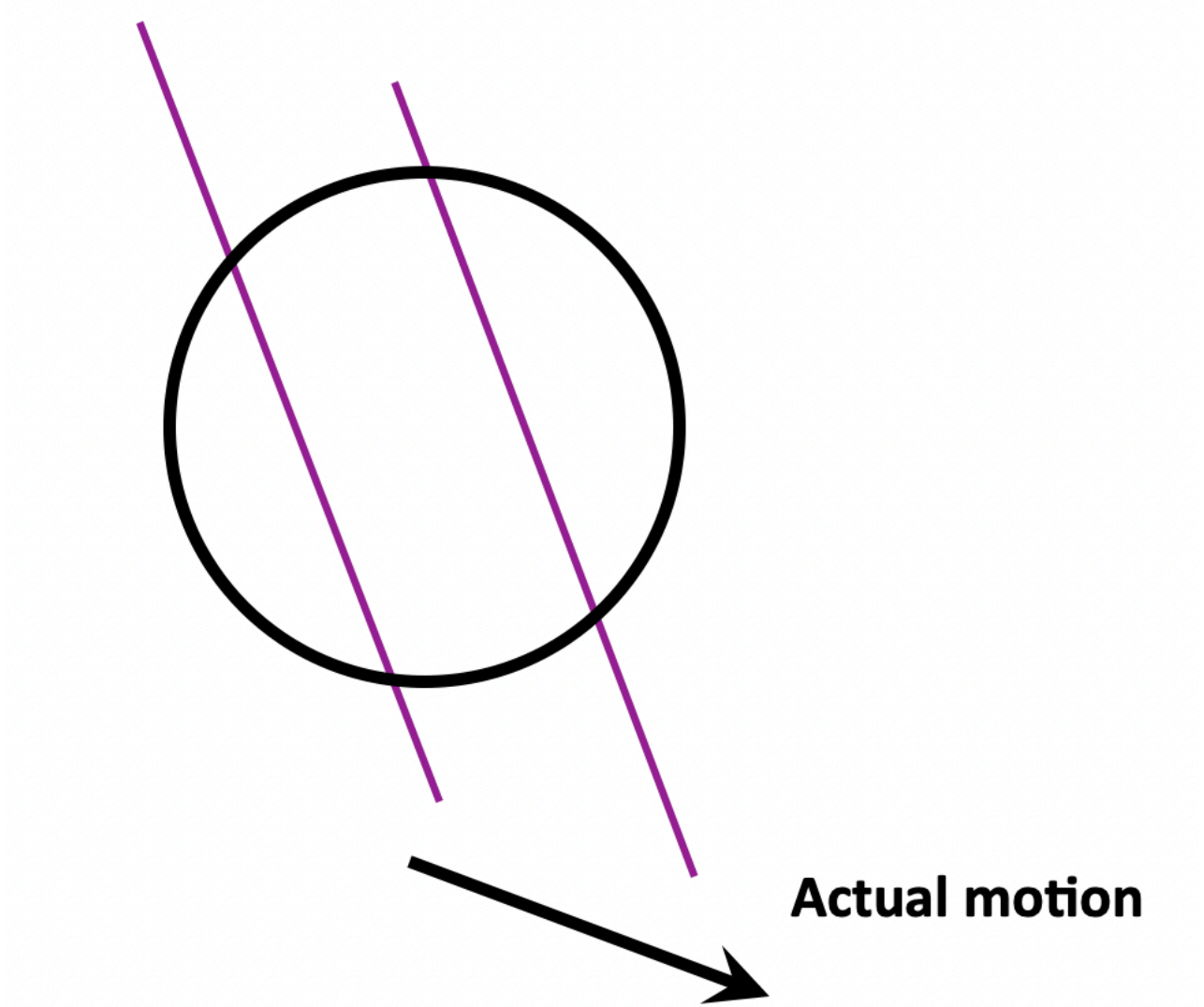
The component of the flow perpendicular to the gradient (i.e., parallel to the edge) cannot be measured



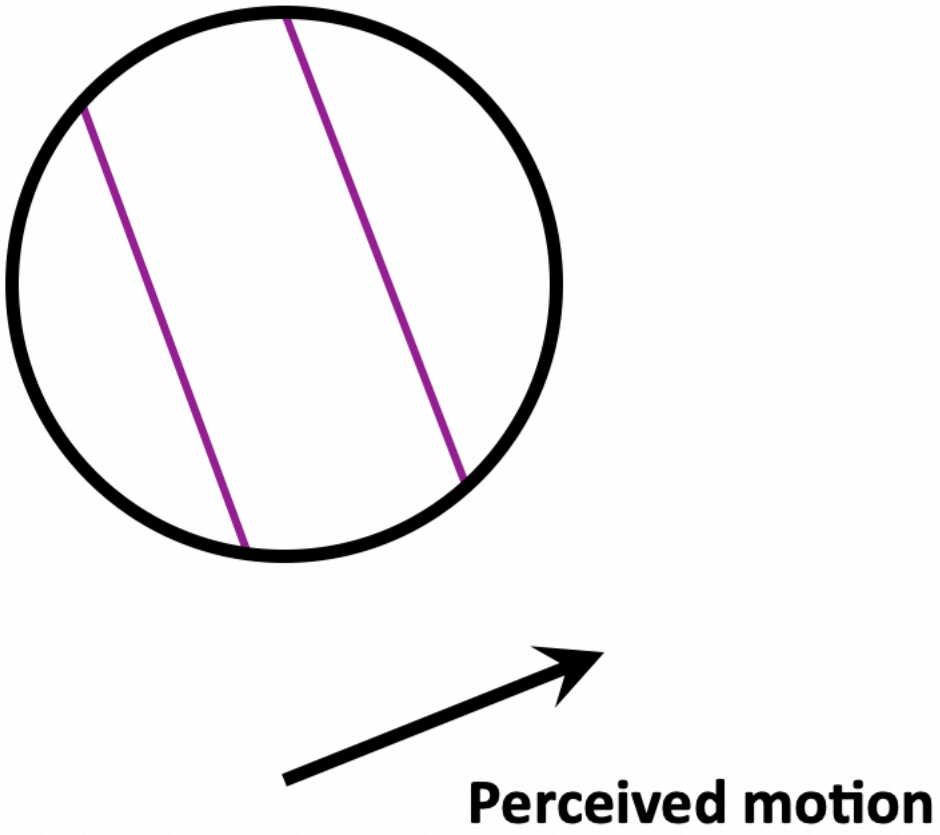
If (u, v) satisfies the equation, so does ($u+u'$, $v+v'$) if

$$\nabla I \cdot \begin{bmatrix} u' & v' \end{bmatrix}^T = 0$$

The Aperture Problem



The Aperture Problem



Barberpole Illusion

- https://en.wikipedia.org/wiki/Barberpole_illusion

Solving the Ambiguity

- How to get more equations for a pixel?
- **Spatial coherence constraint:**
- Assume the pixel's neighbors have the same (u,v)
  - If we use a 5x5 window, that gives us 25 equations per pixel

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u\ v]$$

$$\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}$$

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In Proceedings of the Interna6onal Joint Conference on Artificial Intelligence, pp. 674- 679, 1981.

Lucas-Kanade Flow

- **Overconstrained linear system:**

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

$$\underset{25\text{x}2}{A} \quad \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b}$$

Lucas-Kanade Flow

- Overconstrained linear system:

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix} \qquad \underset{25\text{x}2}{A} \quad \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b}$$

Least squares solution for *d* given by $(A^T A)\ d = A^T b$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $\qquad\qquad$ $A^T b$

The summations are over all pixels in the K x K window

Condition for Solvability

– Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $\qquad$ $A^T b$

When is This Solvable?

- $\mathbf{A^T A}$ should be invertible
- $\mathbf{A^T A}$ should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of $\mathbf{A^T A}$ should not be too small
- $\mathbf{A^T A}$ should be well-conditioned
  - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

Does this remind anything to you?

Condition for Solvability

M = A^TA is the *second moment matrix* !

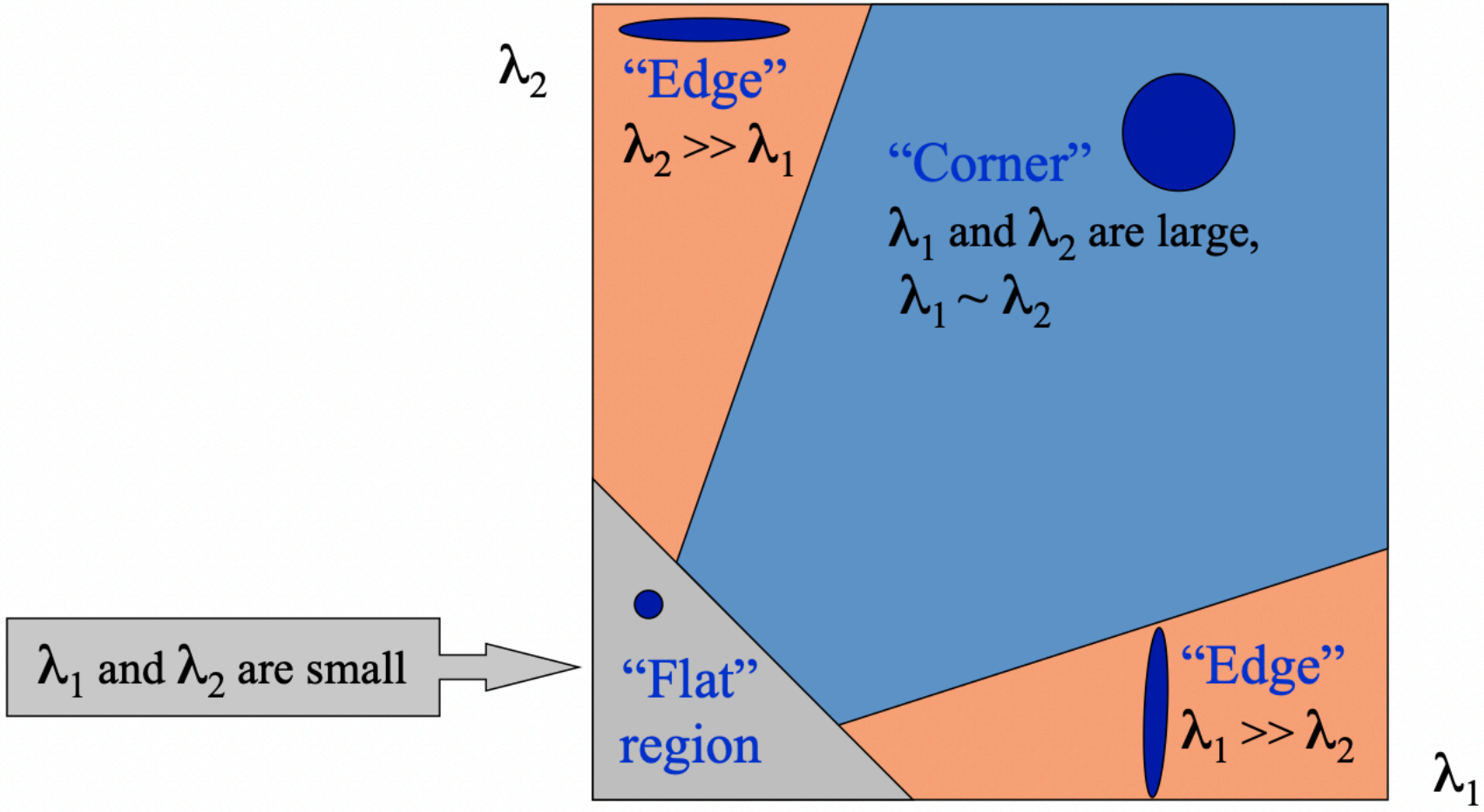
(Harris corner detector…)

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] = \sum \nabla I (\nabla I)^T$$

- Eigenvectors and eigenvalues of A^TA relate to edge direction and magnitude
  - The eigenvector associated with the larger eigenvalue points in the direction of fastest intensity change
  - The other eigenvector is orthogonal to it

Interpreting the Eigenvalues

Classification of image points using eigenvalues of the second moment matrix:



Edge

$$\sum \nabla I (\nabla I)^T$$

– gradients very large or very small

– large λ_1, small λ_2

Low-Texture Region

$$\sum \nabla I (\nabla I)^T$$

- gradients have small magnitude
- small λ_1, small λ_2

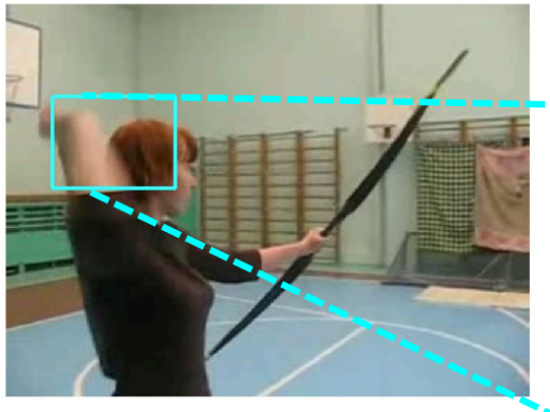
High Texture Region

$$\sum \nabla I (\nabla I)^T$$

- gradients are different, large magnitudes
- large λ_1, large λ_2

Measuring Motion

Image at frame t



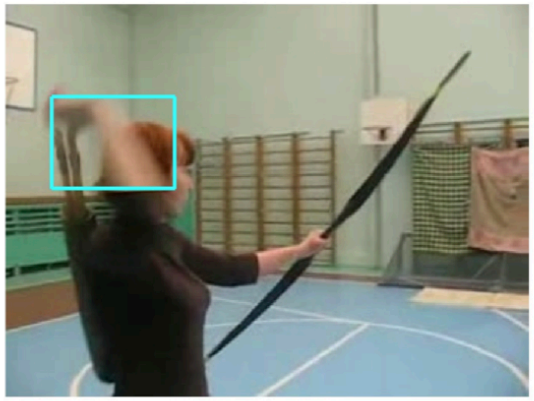
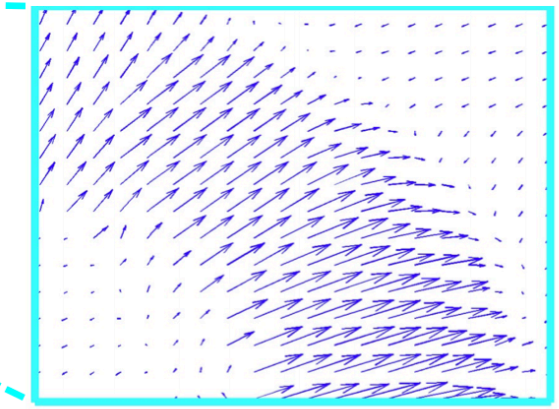


Image at frame t+1

Optical flow gives a displacement field F between images I_t and I_{t+1}



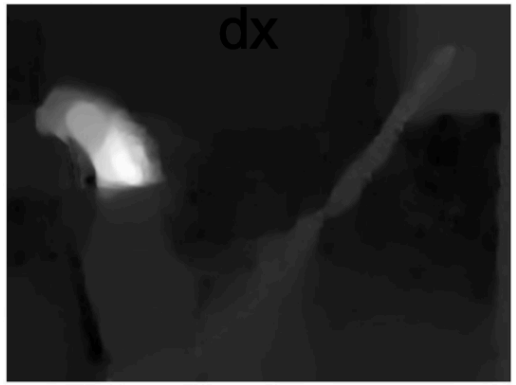
Tells where each pixel will move in the next frame:

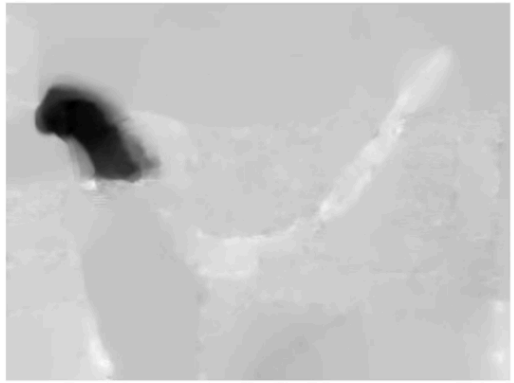
$F(x, y) = (dx, dy)$

$I_{t+1}(x+dx, y+dy) = I_t(x, y)$

Optical Flow highlights **local motion**

Horizontal flow





Vertical Flow dy

Slide credit: Justin Johnson

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

FlowNet: Learning Optical Flow with Convolutional Networks

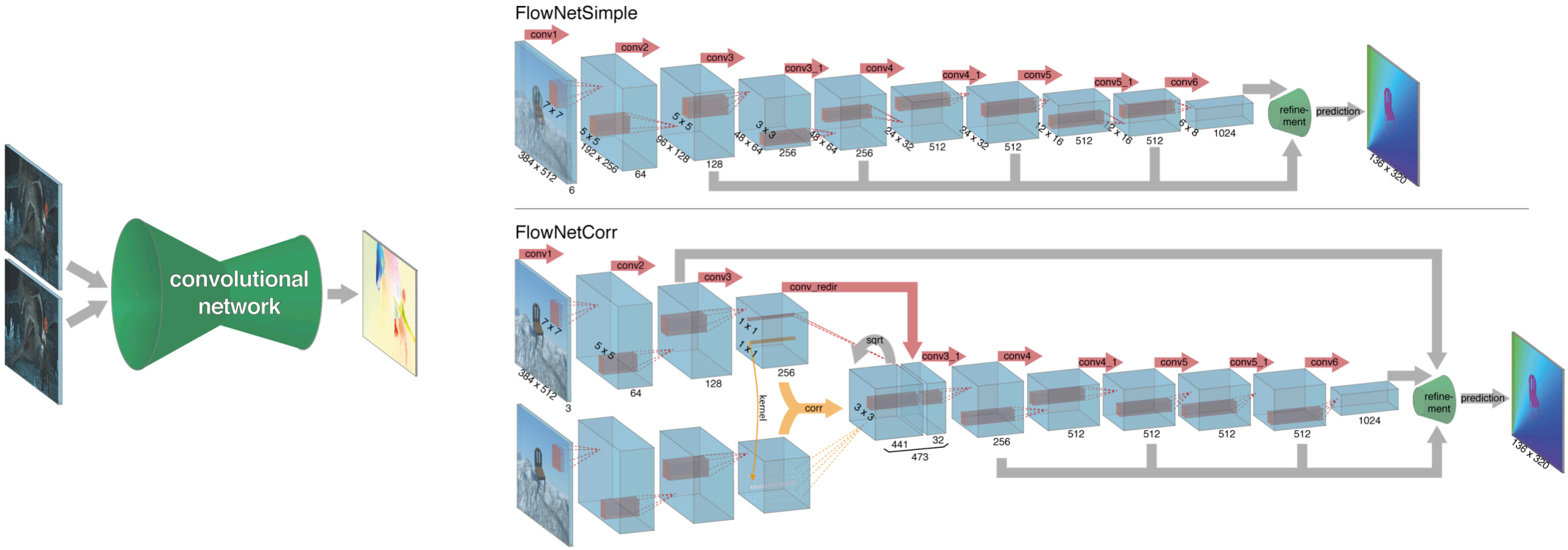


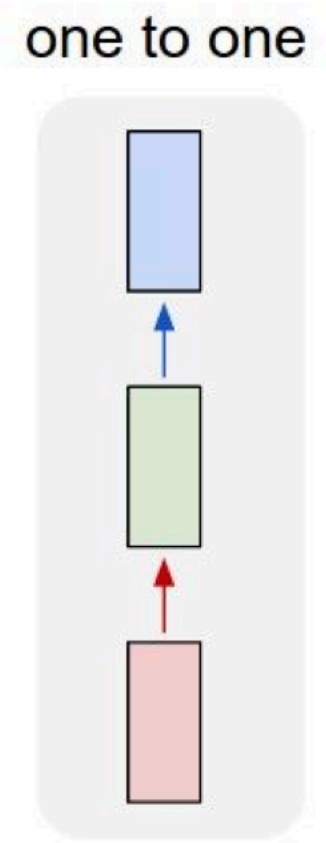
Figure 2. The two network architectures: FlowNetSimple (top) and FlowNetCorr (bottom).

Dosovitskiy, Alexey, et al. "Flownet: Learning optical flow with convolutional networks." *CVPR* 2015.

RNN

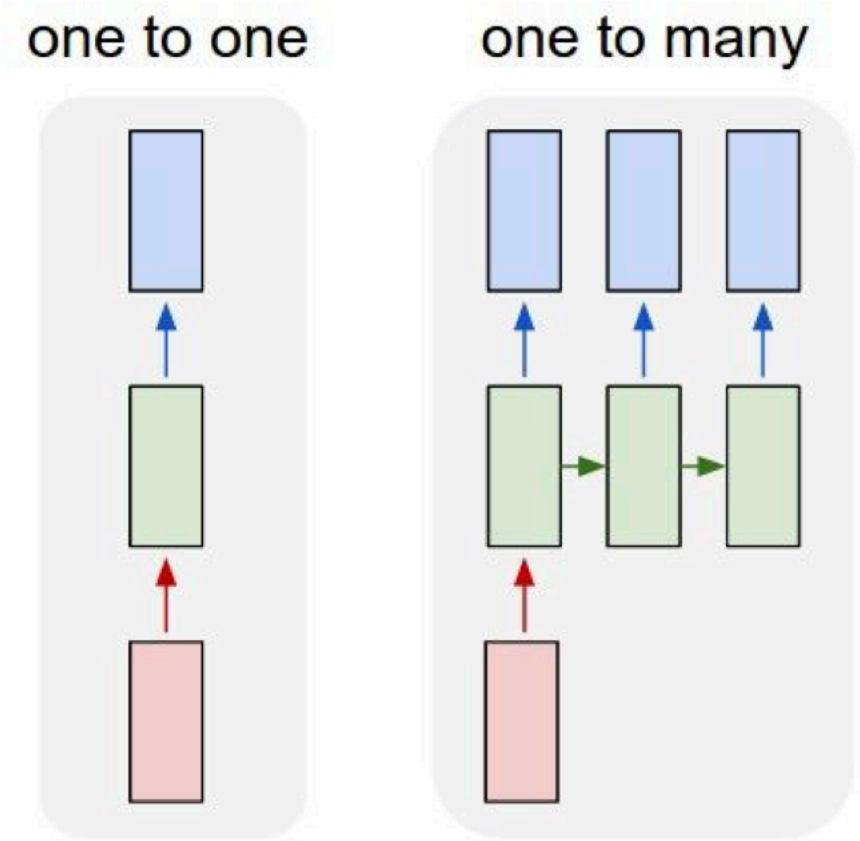
Some slides are borrowed from Stanford CS231N

Single-Frame Neural Network



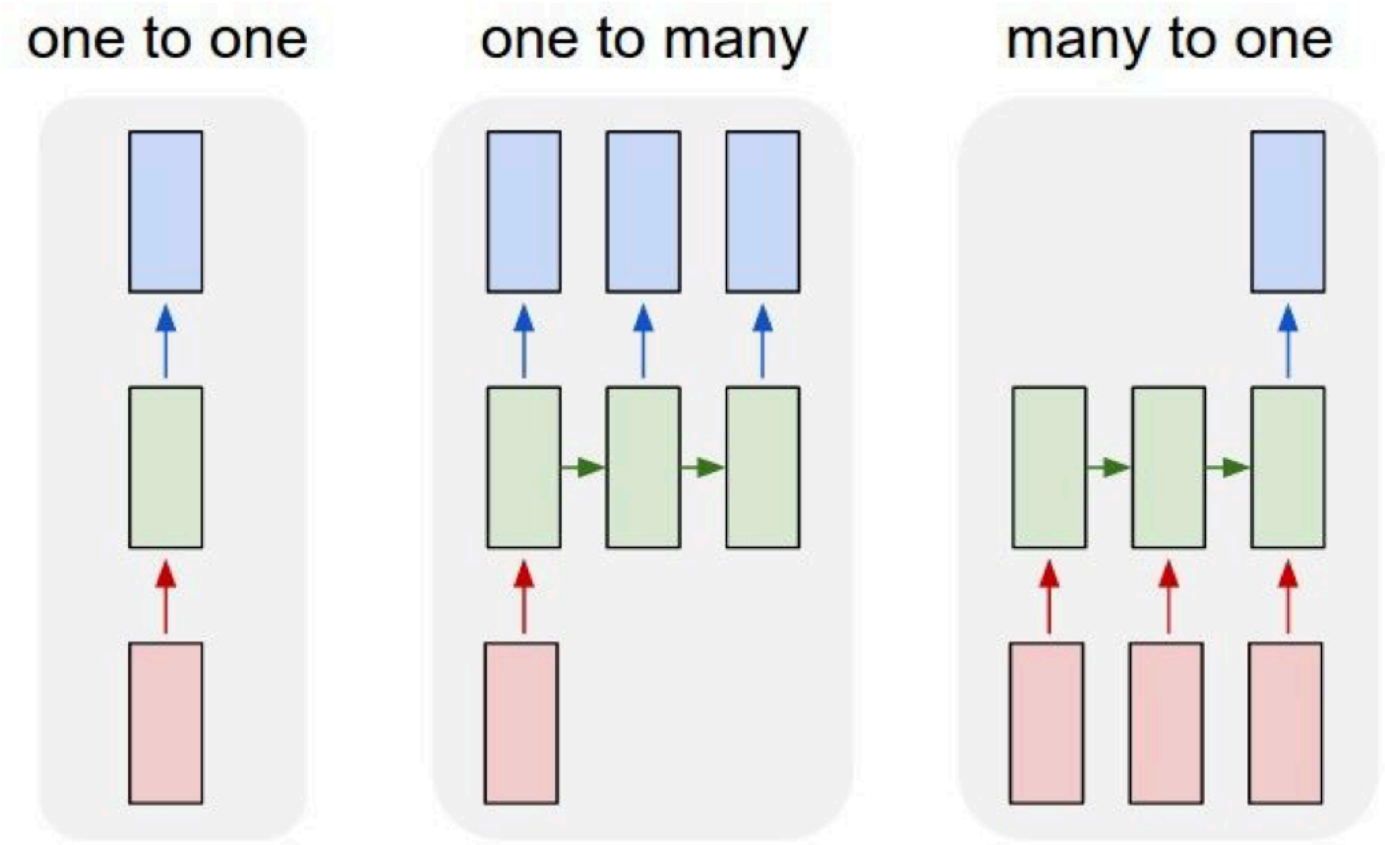
Vanilla Neural Networks

Recurrent Neural Network: Process Sequential Data



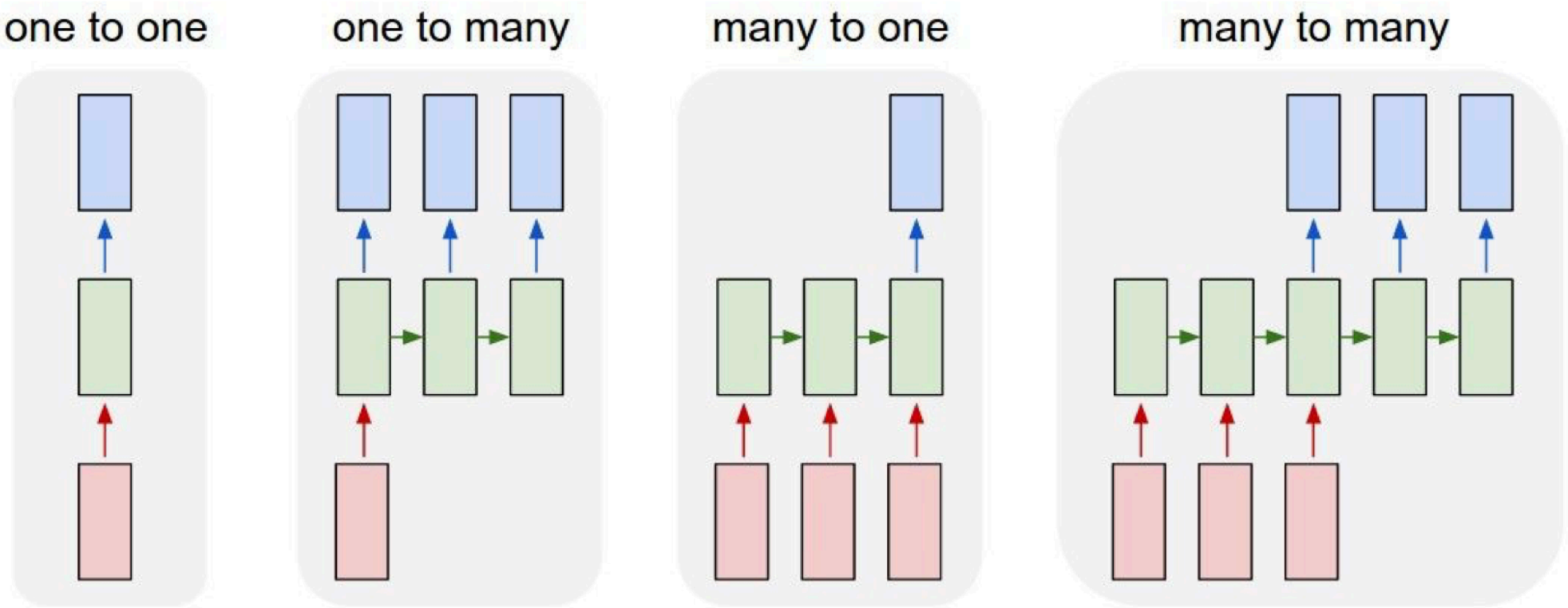
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Network: Process Sequential Data



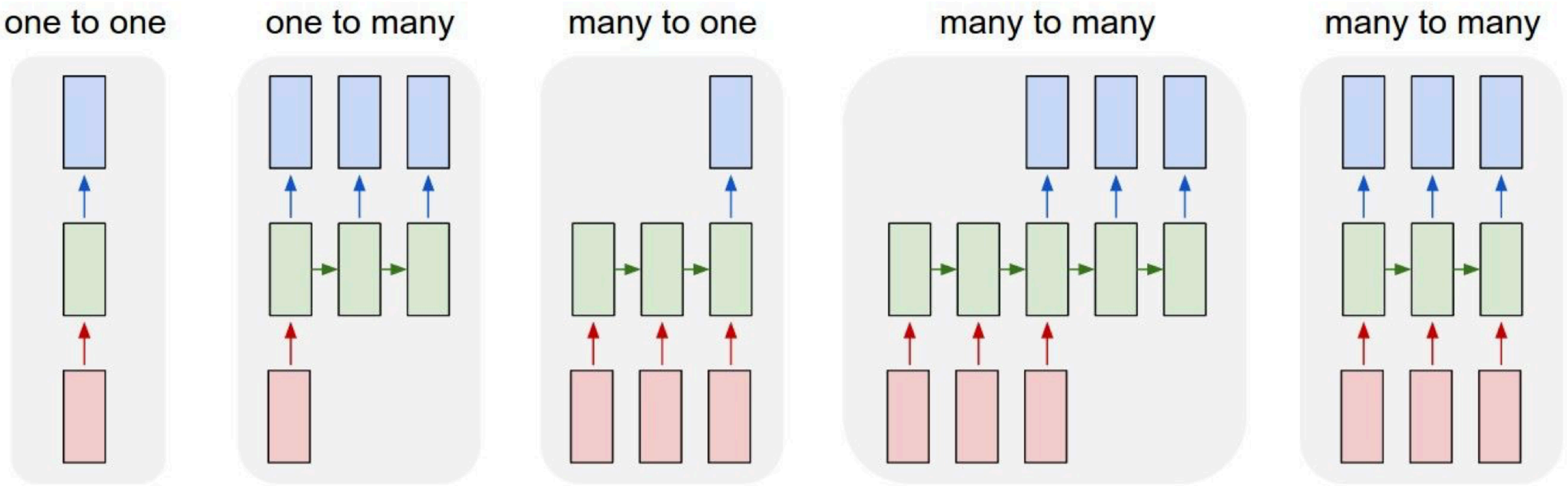
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Network: Process Sequential Data



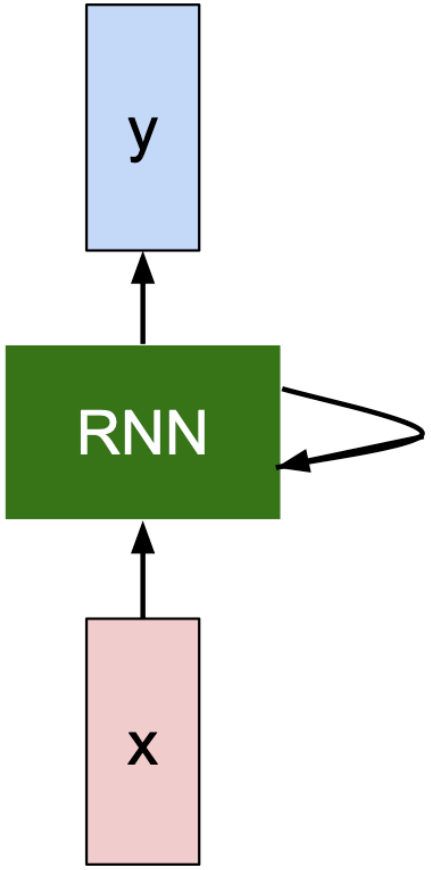
E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Network: Process Sequential Data

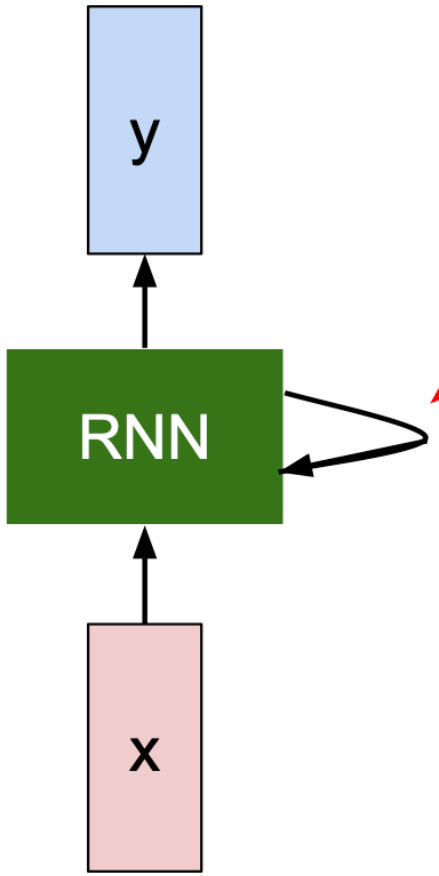


e.g. **Video classification on frame level**

Recurrent Neural Network

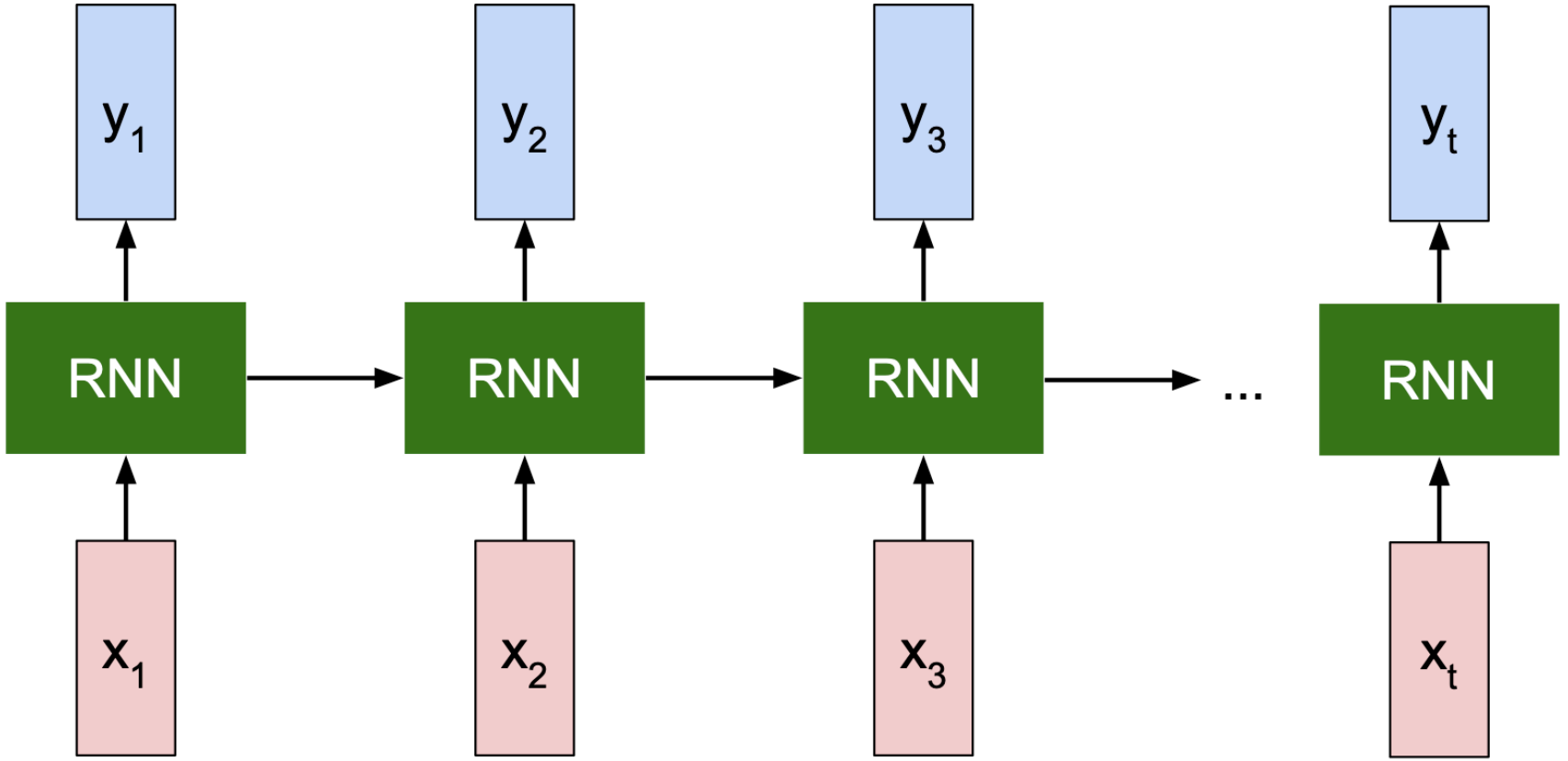


Recurrent Neural Network



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Unrolled RNN



RNN Hidden Sate Update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

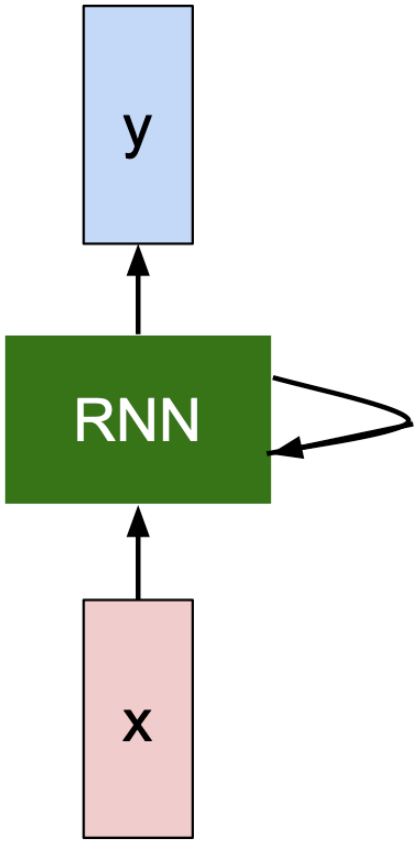
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



RNN Output

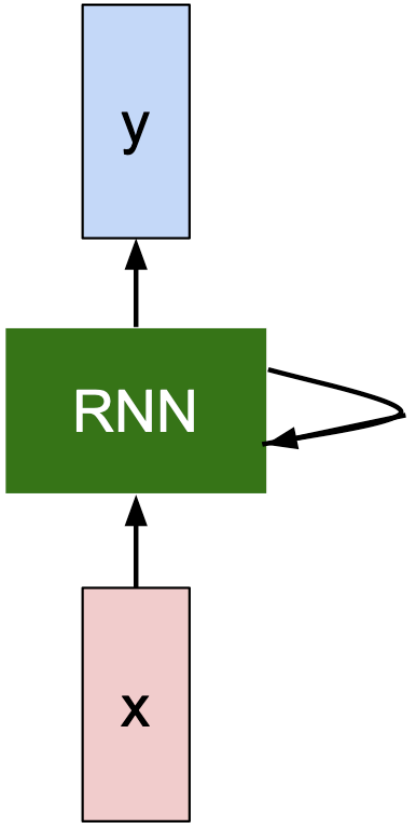
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

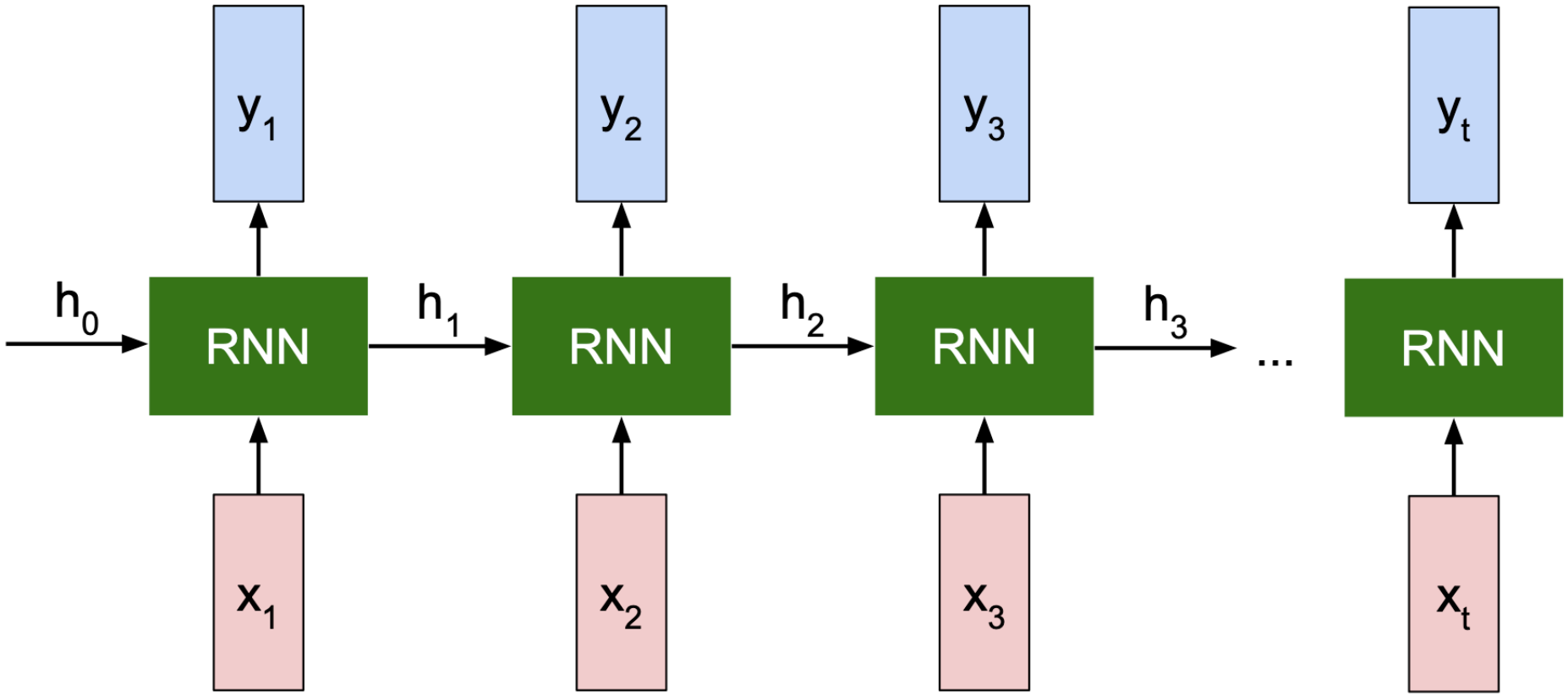
output

new state

another function with parameters W_o



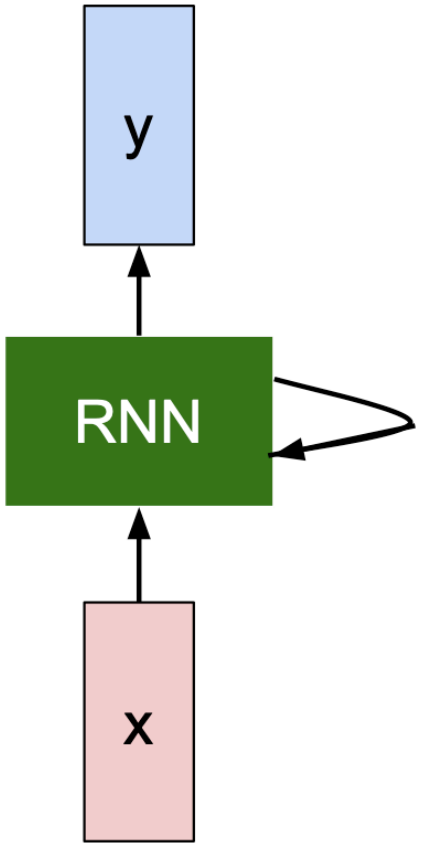
Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

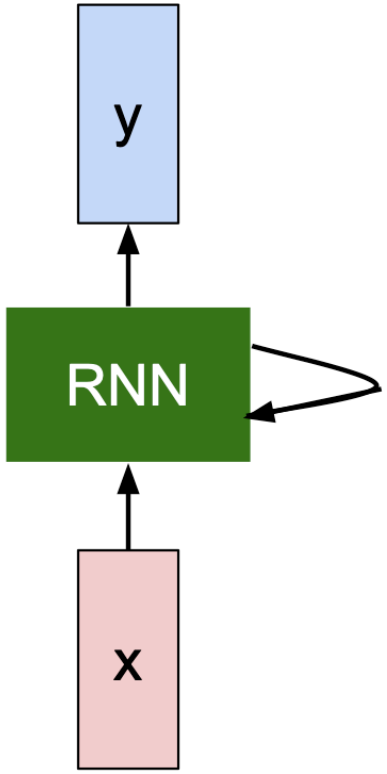
$$h_t = f_W(h_{t-1}, x_t)$$



Notice: the same function and the same set of parameters are used at every time step.

Vanilla RNN

The state consists of a single *"hidden"* vector **h**:



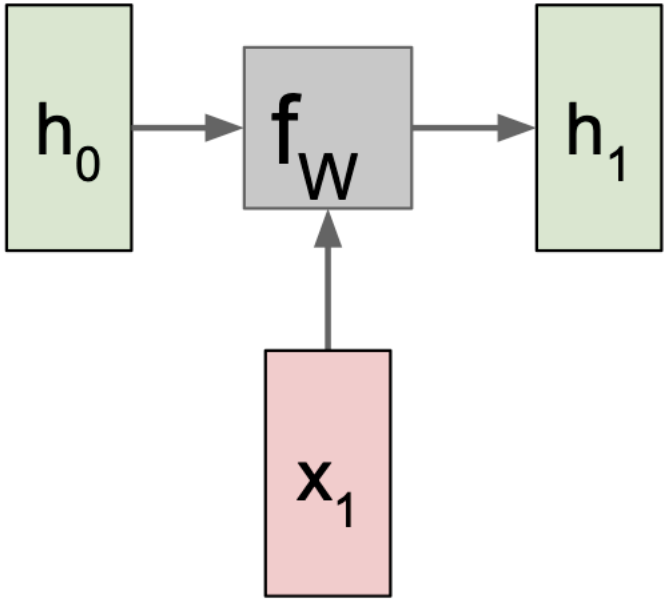
$$h_t = f_W(h_{t-1}, x_t)$$

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

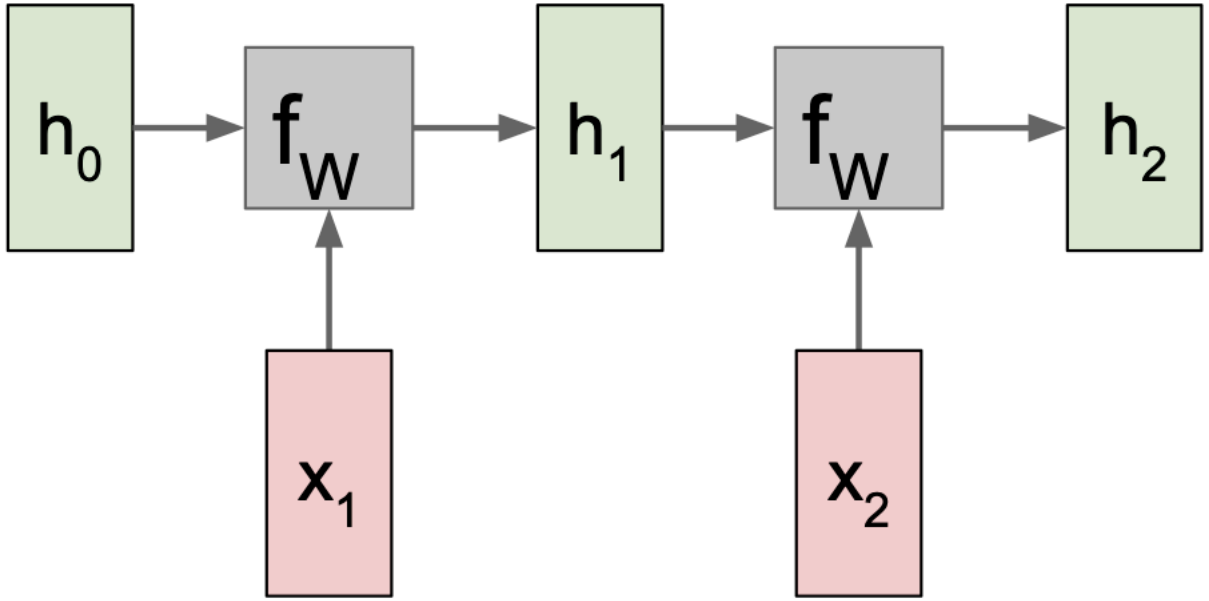
$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

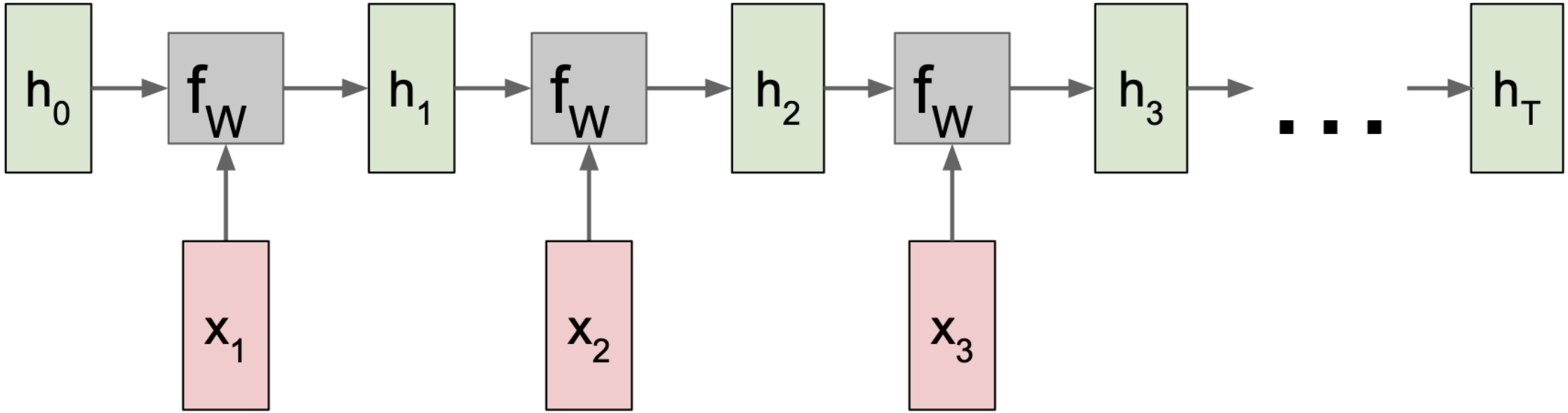
RNN: Computational Graph



RNN: Computational Graph

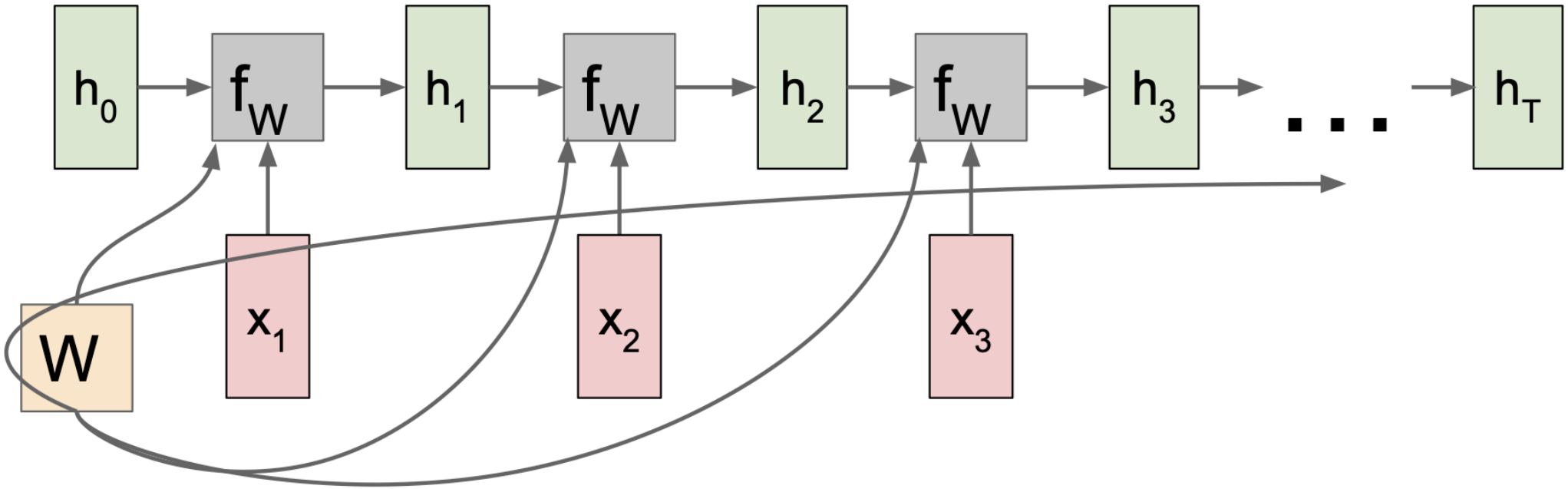


RNN: Computational Graph



RNN: Computational Graph

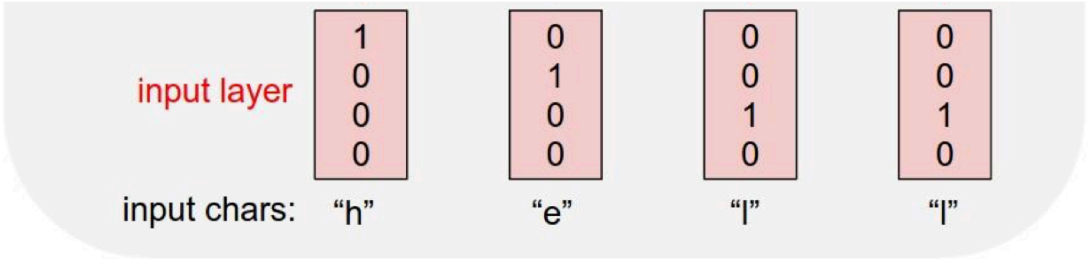
Re-use the same weight matrix at every time-step



Character-Level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

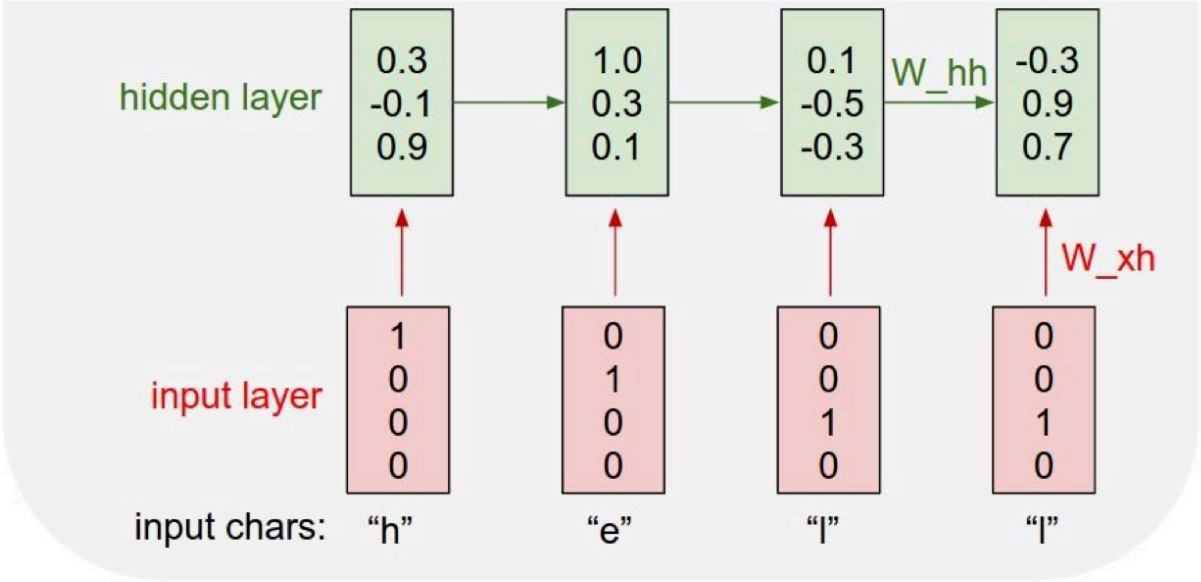


Character-Level Language Model

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

Vocabulary:
[h,e,l,o]

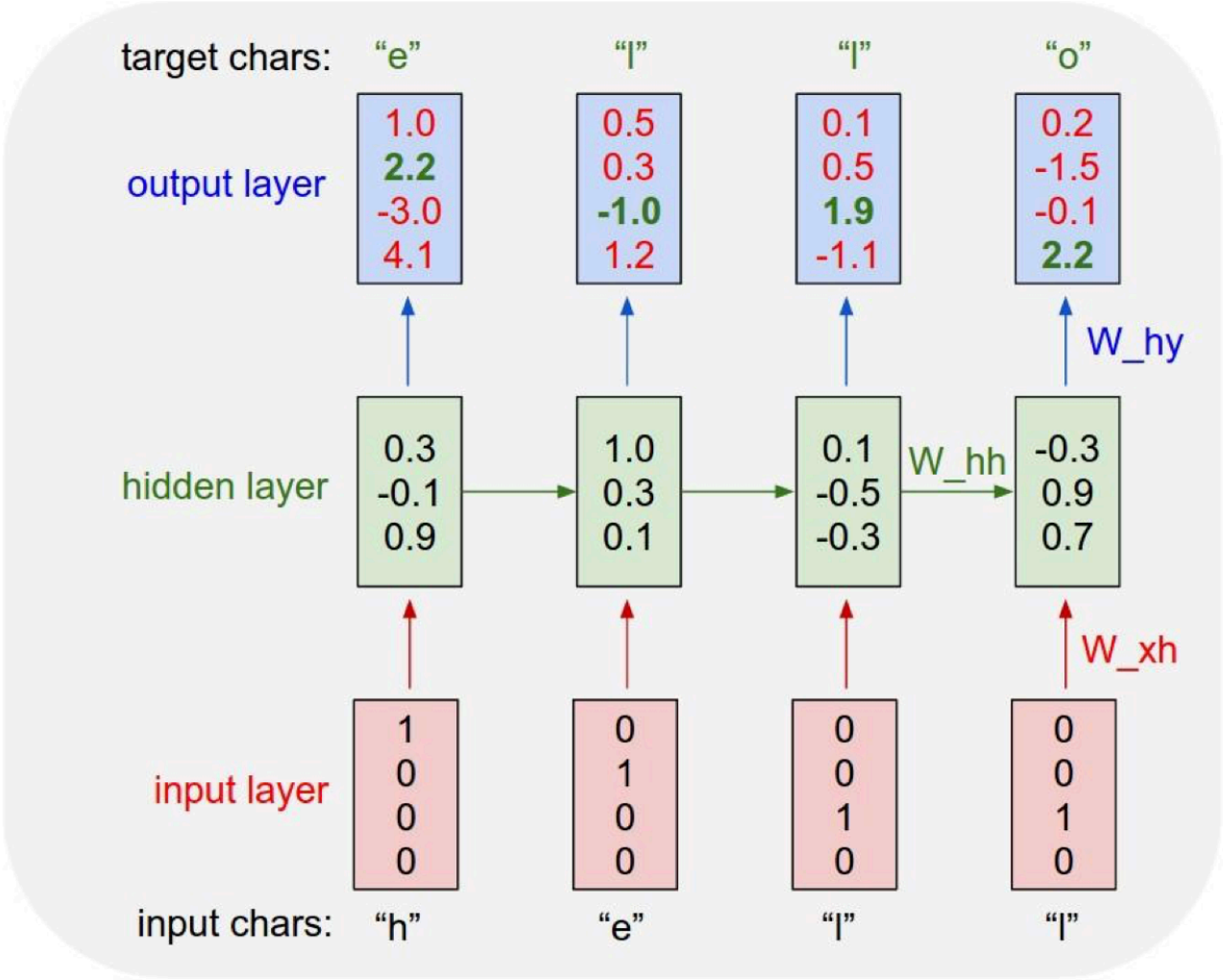
Example training sequence:
“hello”



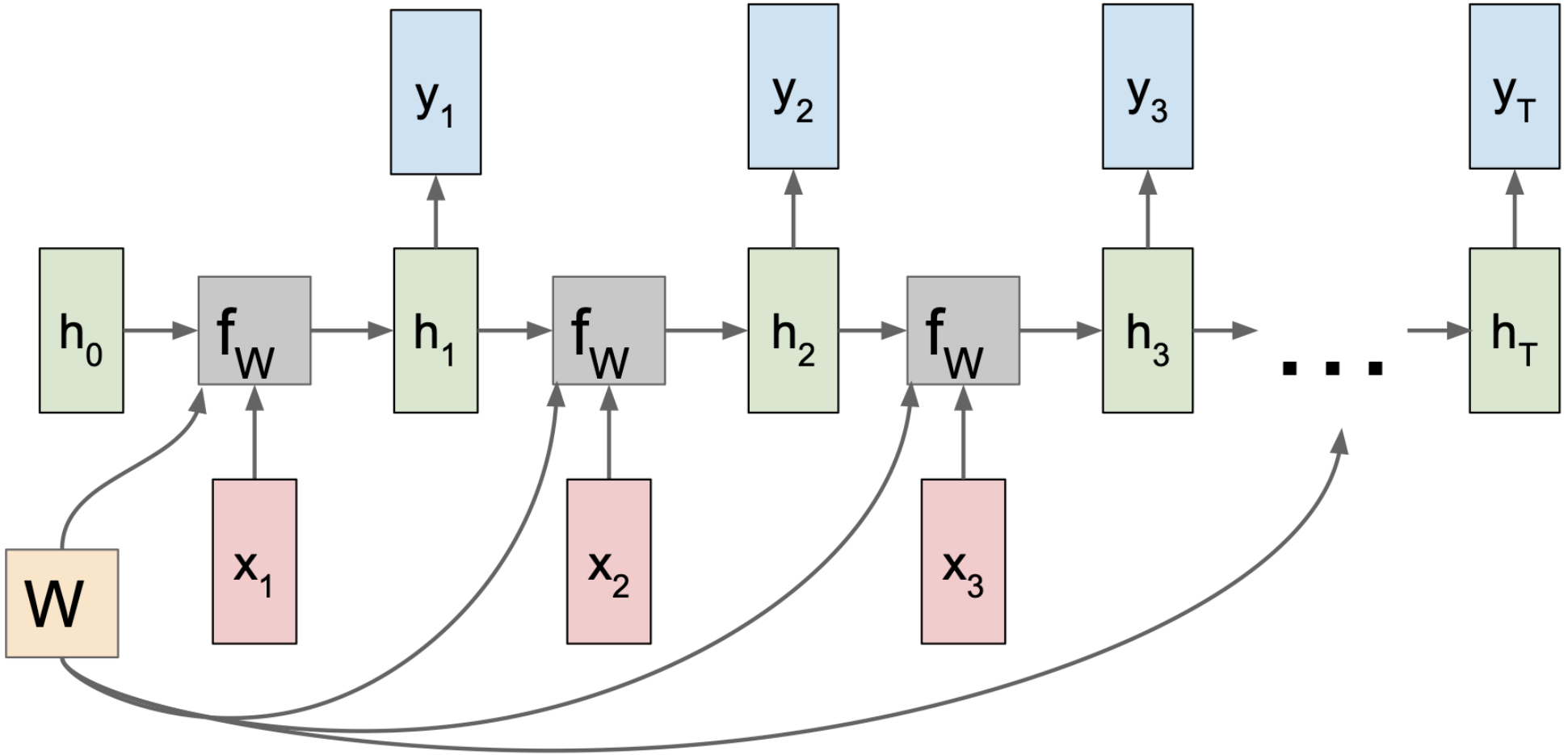
Character-Level Language Model

Vocabulary:
[h,e,l,o]

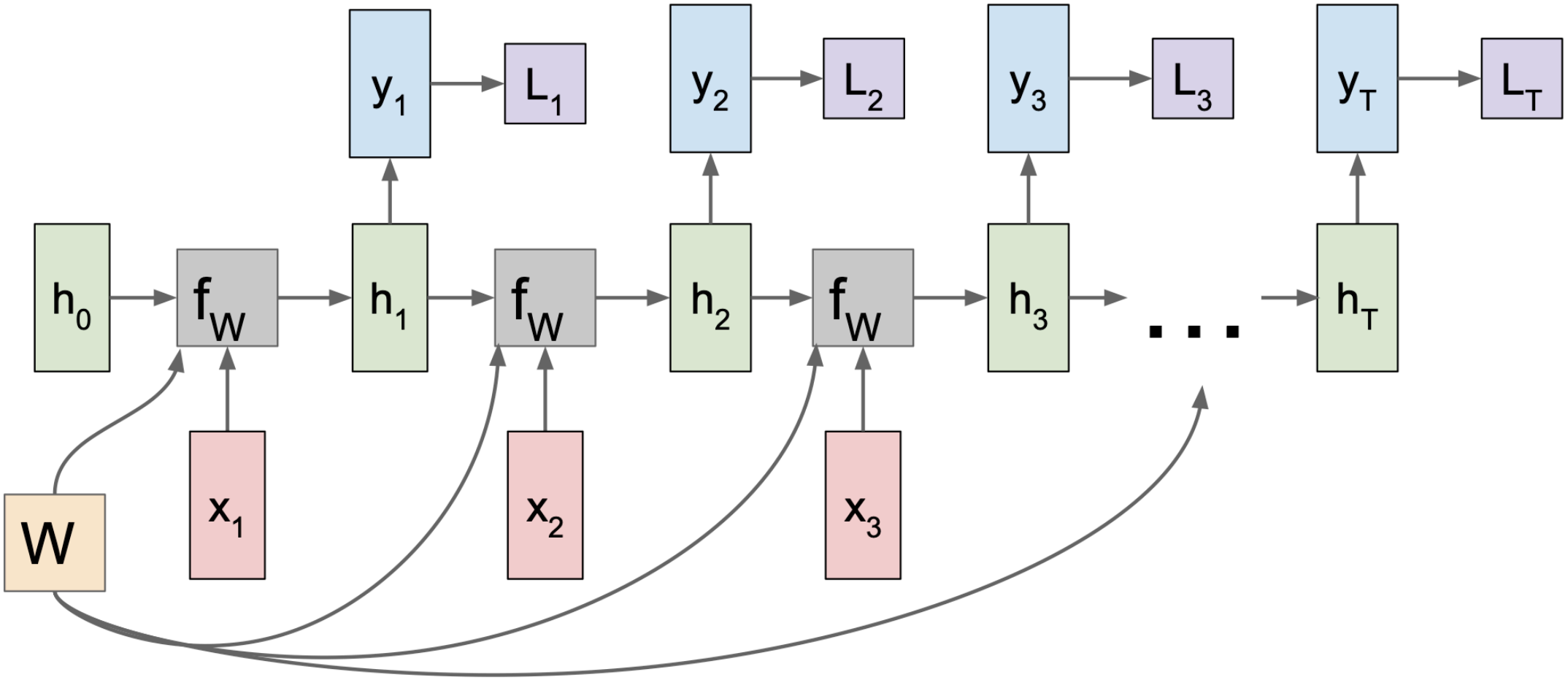
Example training sequence:
“hello”



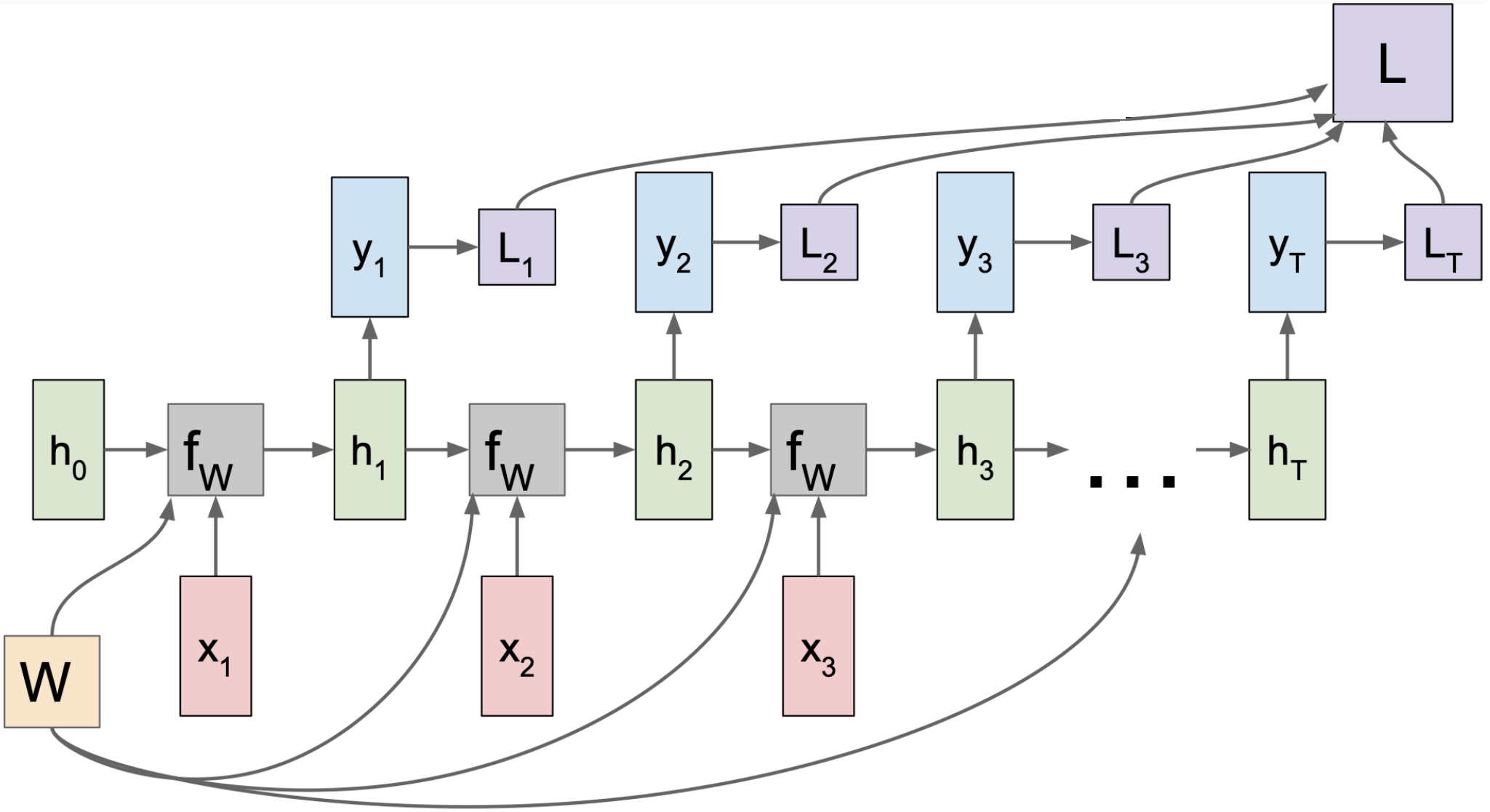
Many-to-Many RNN: Computational Graph



Many-to-Many RNN: Computational Graph

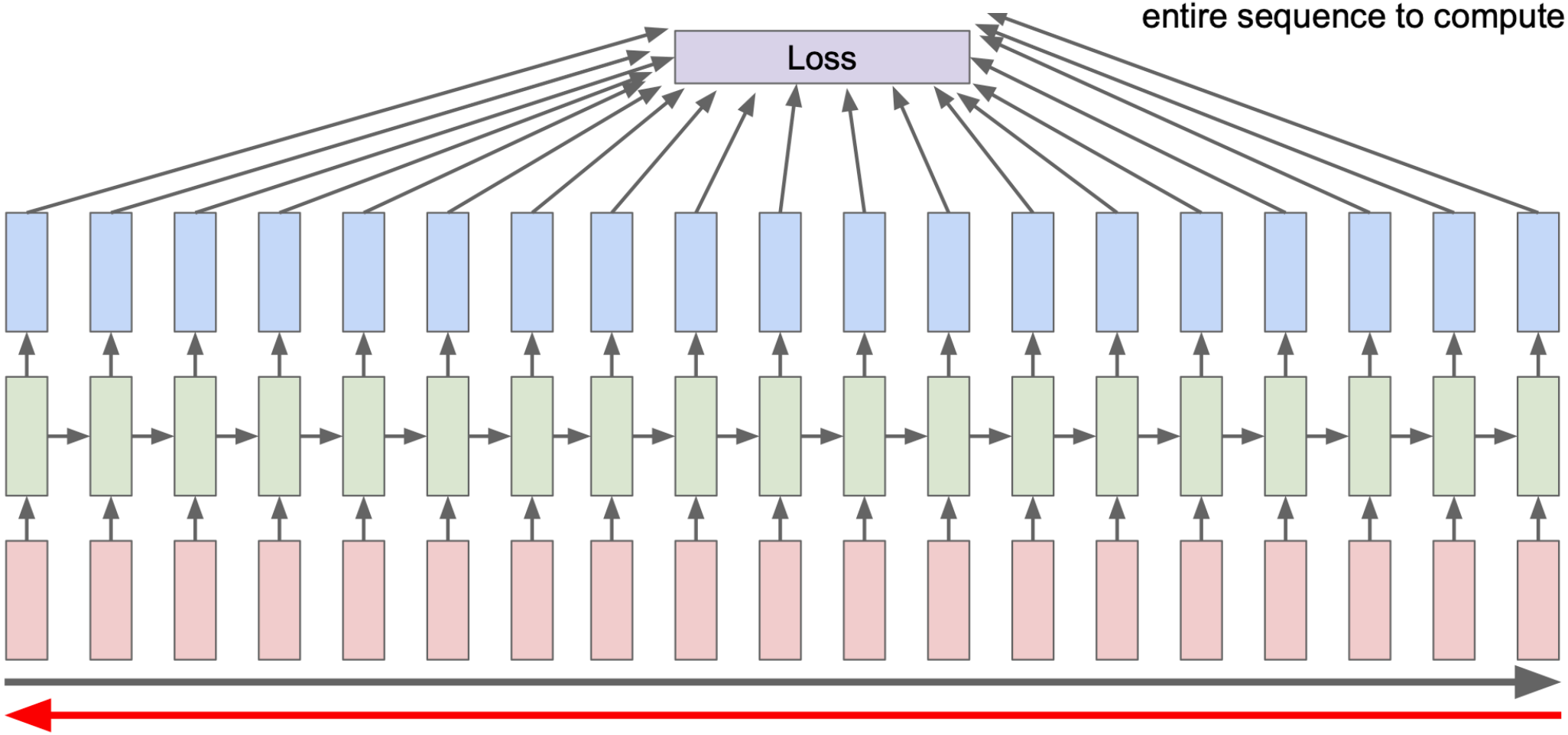


Many-to-Many RNN: Computational Graph

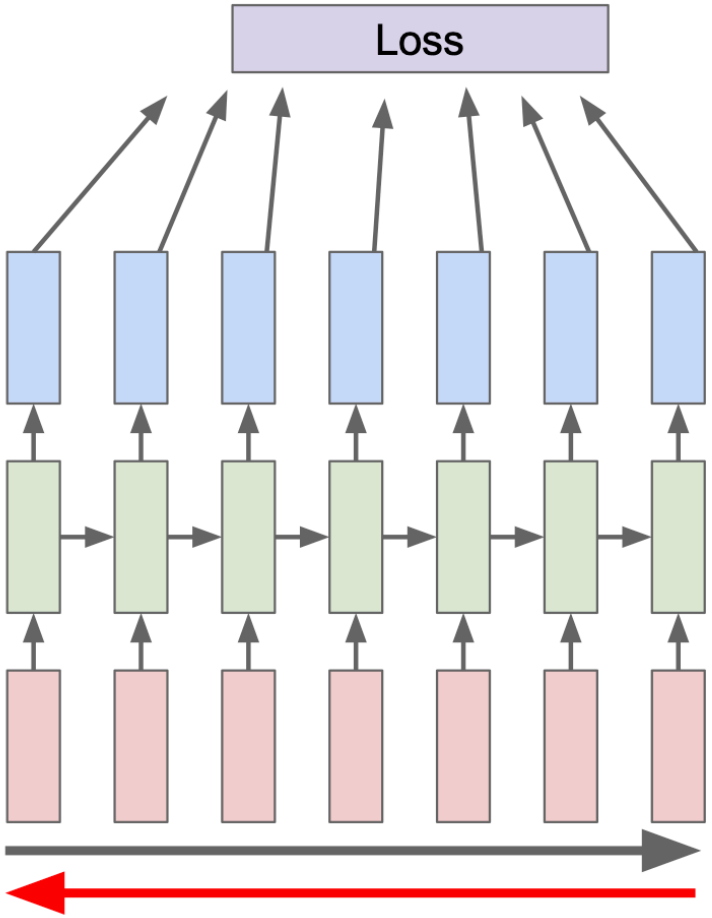


Backpropagation through Time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

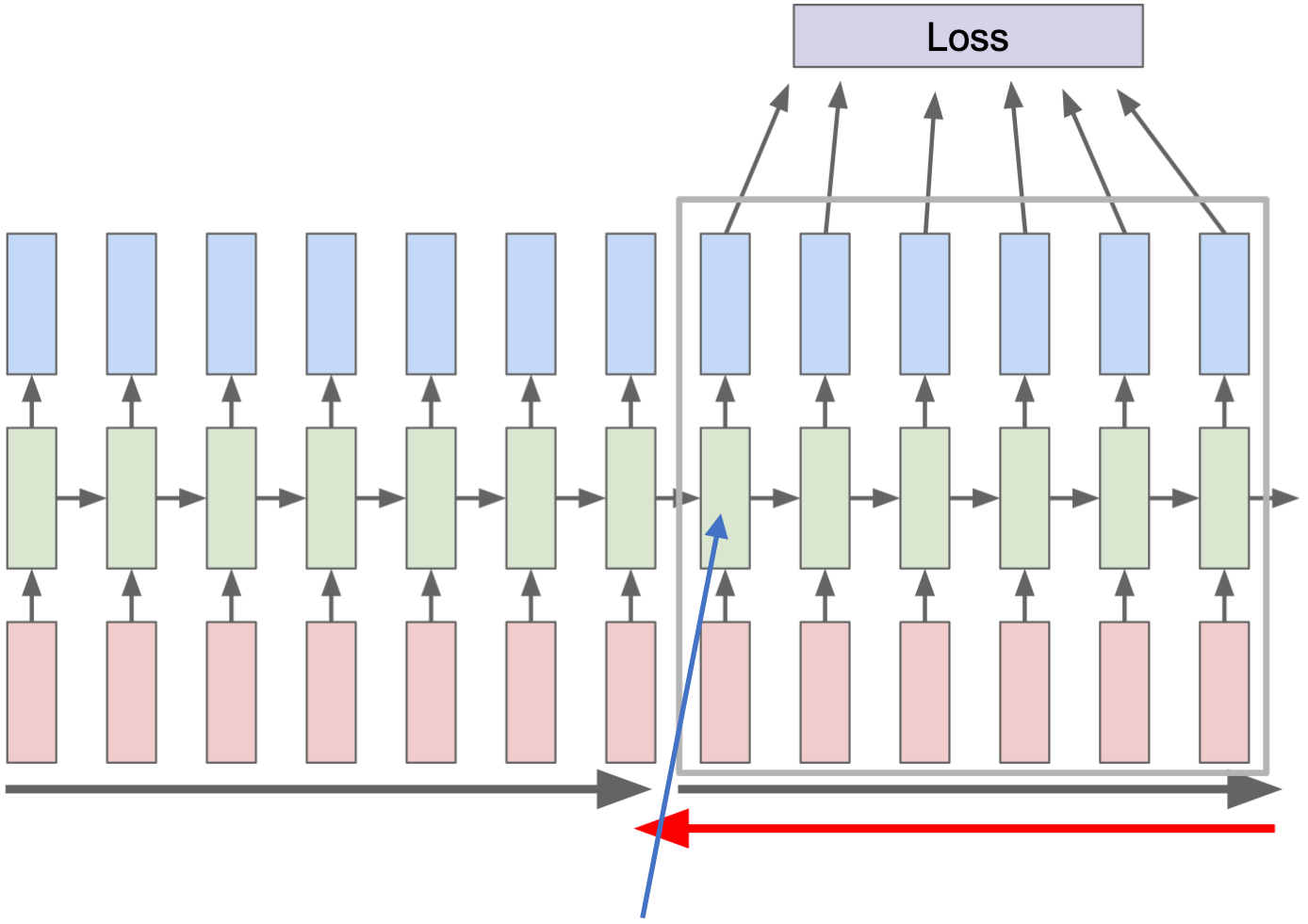


Truncated Backpropagation through Time



Run forward and backward through chunks of the sequence instead of whole sequence

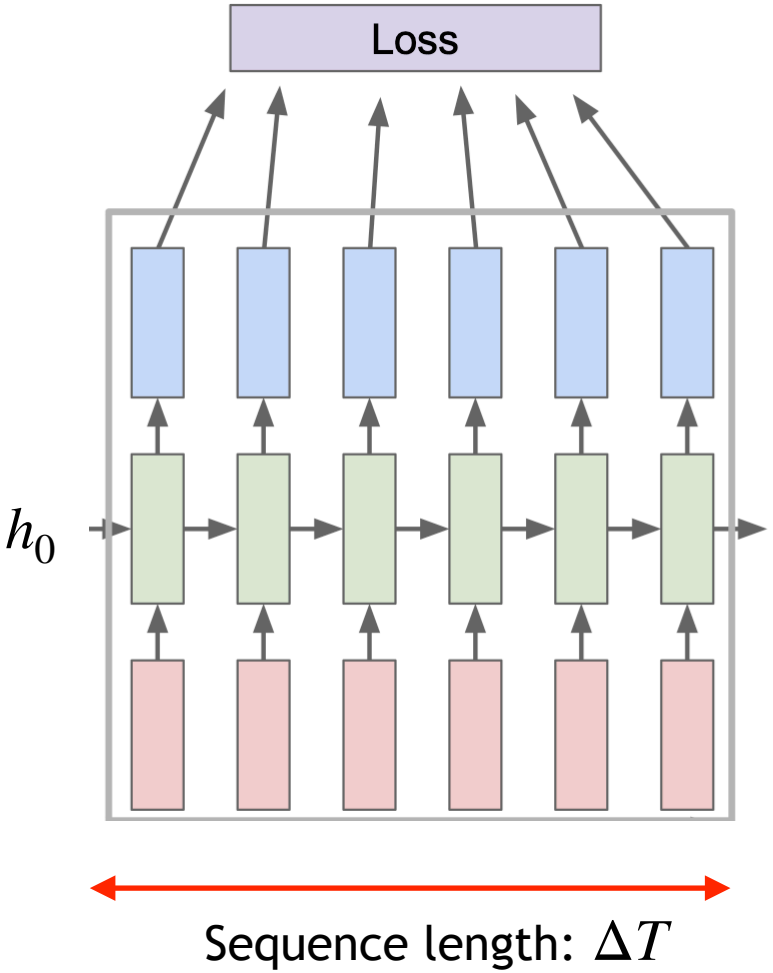
Truncated Backpropagation through Time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

h_t or in practice can use h_0 if t is so very large such that we even don't want to compute h_t during training

Truncated Backpropagation through Time



If use h_0, then training only enforces the temporal relationship inside the window, whose length is sequence length.

Long-term relationship longer than ΔT will probably not be learned.

Introduction to Computer Vision

Next Week: Labor Day Break
Next lecture: Lecture 10,
Temporal Analysis II