# CS 4803 / 7643: Deep Learning

Topics:

- (Finish) Convolutional Neural Networks
  - Transposed convolutions
- Recurrent Neural Networks (RNNs)

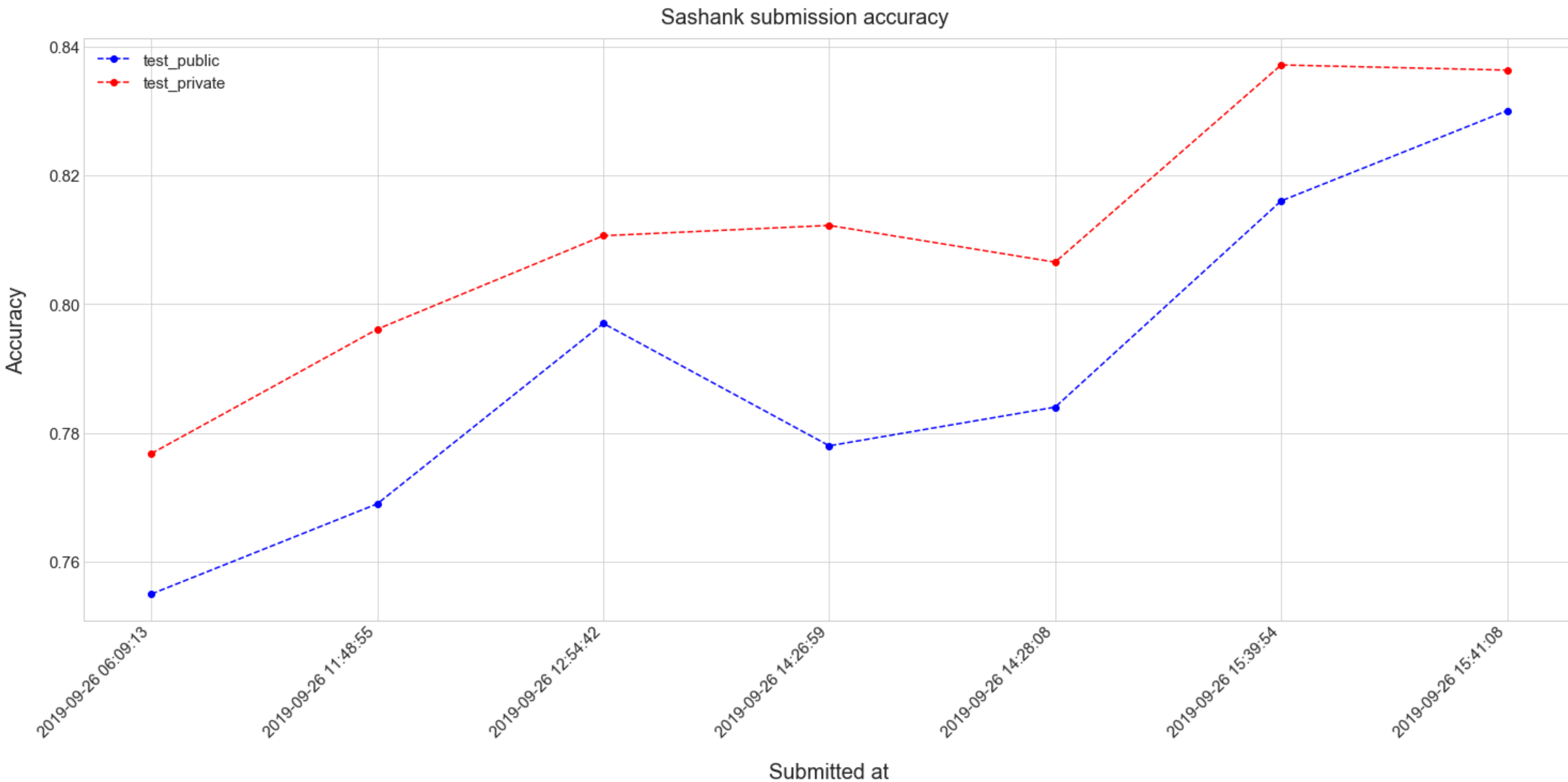

Dhruv Batra

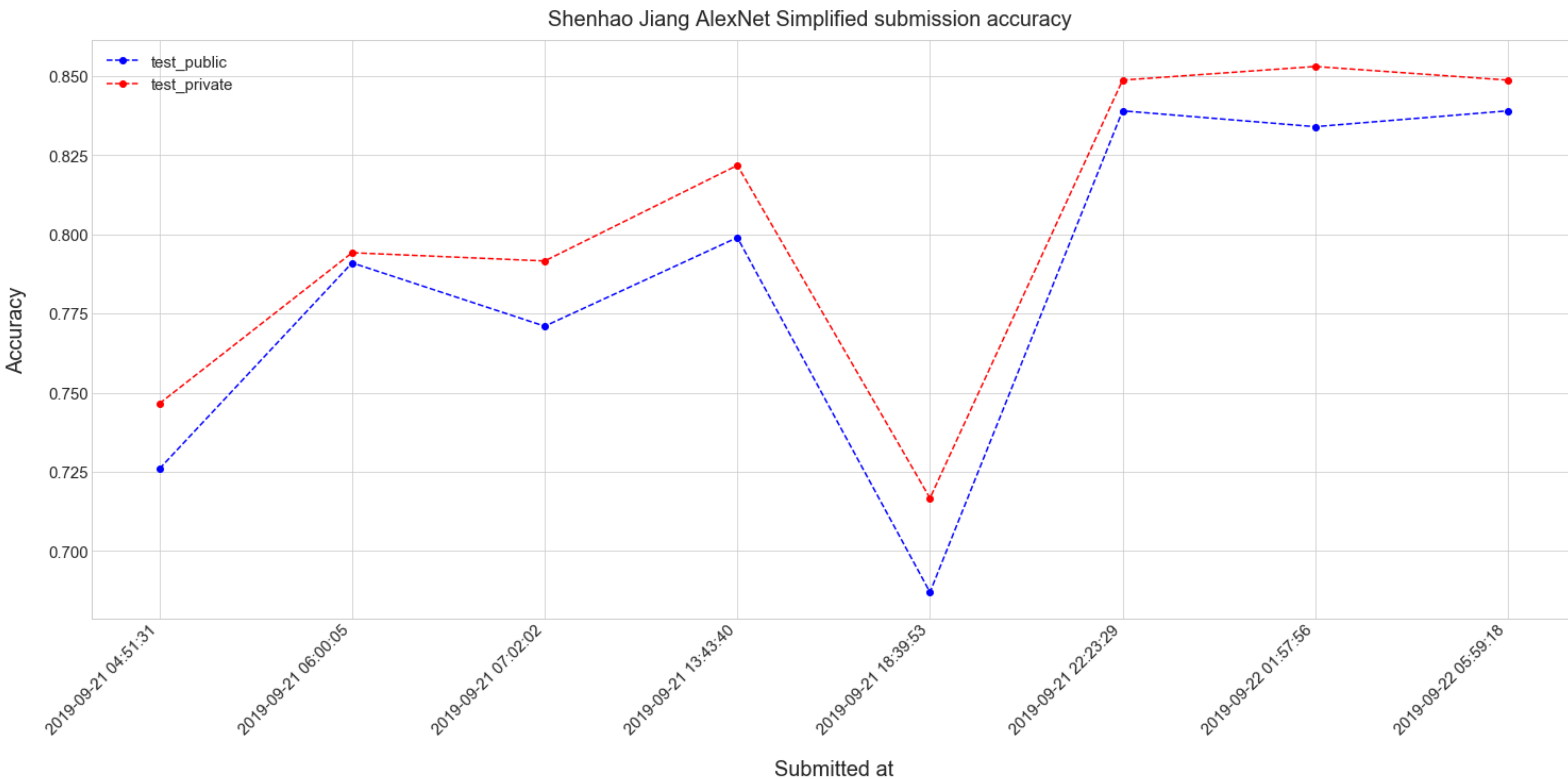
Georgia Tech

# Administrativia

- HW1 Challenge Final Analysis
  - https://evalai.cloudcv.org/web/challenges/challenge-page/431/leaderboard/1200
  - Qualitative Trends

- HW2 Reminder
  - Due: 10/10, 11:55pm
  - https://www.cc.gatech.edu/classes/AY2020/cs7643_fall/assets/hw2.pdf


Sashank submission accuracy
test_public
test_private
Accuracy
0.84
0.82
0.80
0.78
0.76
2019-09-26 06:09:13
2019-09-26 11:48:55
2019-09-26 12:54:42
2019-09-26 14:26:59
2019-09-26 14:28:08
2019-09-26 15:39:54
2019-09-26 15:41:08
Submitted at


Shenhao Jiang AlexNet Simplified submission accuracy
test_public
test_private
0.850
0.825
0.800
0.775
0.750
0.725
0.700
Accuracy
2019-09-21 04:51:31
2019-09-21 06:00:05
2019-09-21 07:02:02
2019-09-21 13:43:40
2019-09-21 18:39:53
2019-09-21 22:23:29
2019-09-22 01:57:56
2019-09-22 05:59:18
Submitted at

# Plan for Today

- (Finish) Convolutional Neural Networks
  - Transposed convolutions
- Recurrent Neural Networks (RNNs)
  - A new model class
  - Learning: BackProp Through Time (BPTT)

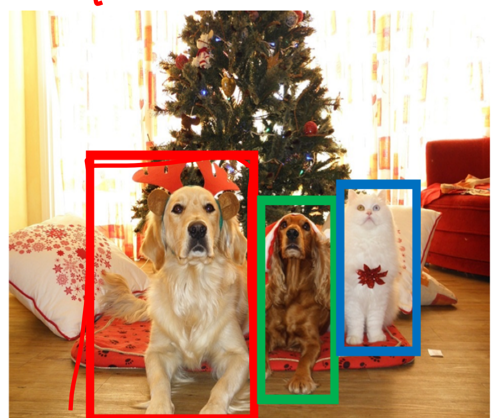
# Other Computer Vision Tasks

**Semantic Segmentation**

GRASS, CAT, TREE, SKY

No objects, just pixels

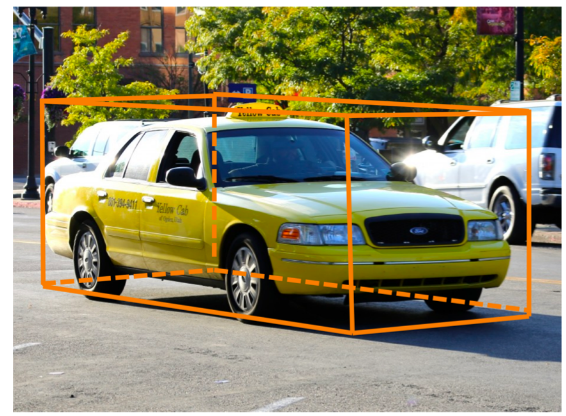
**2D Object Detection**



DOG, DOG, CAT

Object categories + 2D bounding boxes

**3D Object Detection**



Car

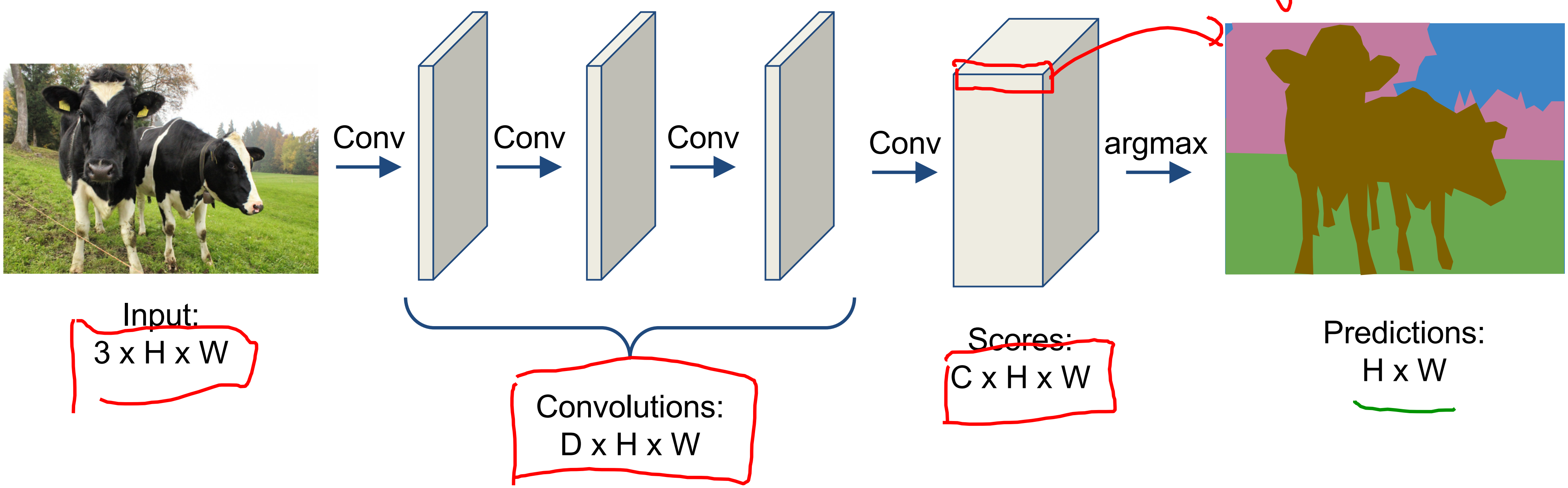
Object categories + 3D bounding boxes



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

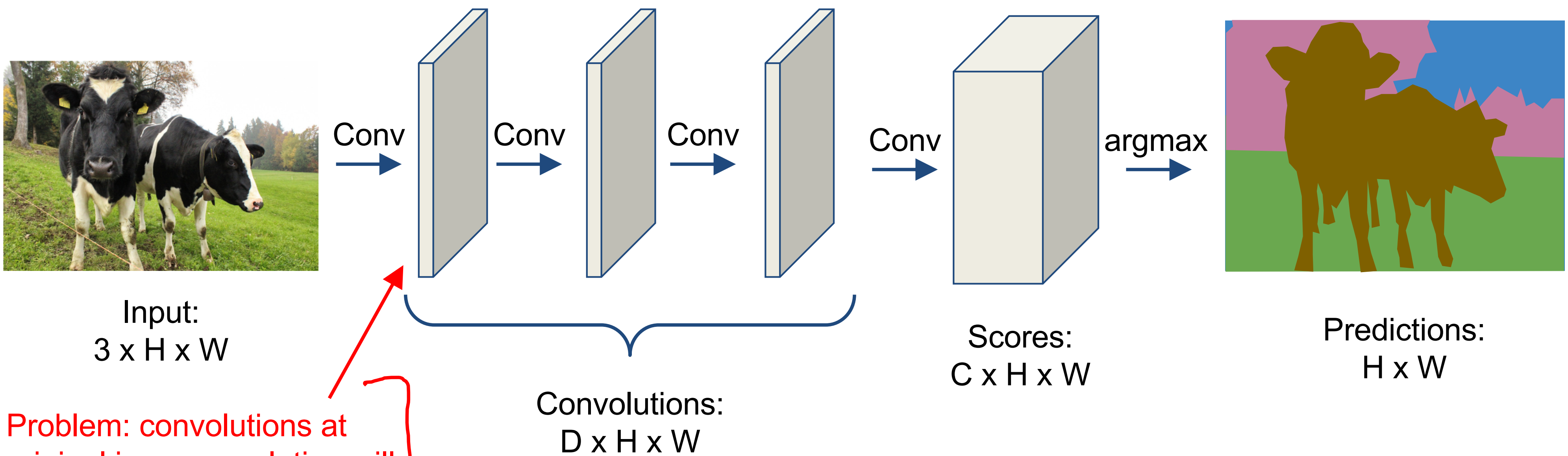
# Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!



# Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers
to  make predictions for pixels all at once!

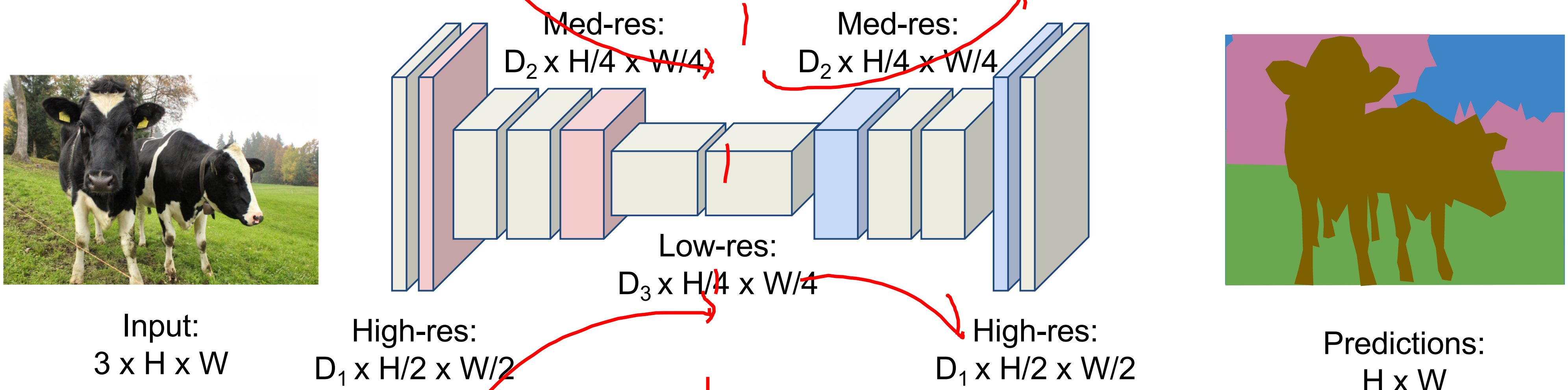



Problem: convolutions at
original image resolution will
be very expensive ...

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!





Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

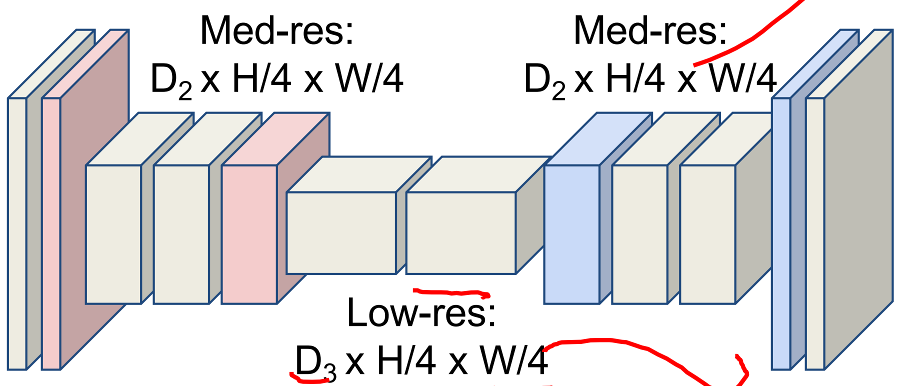
# Semantic Segmentation Idea: Fully Convolutional

**Downsampling**: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

**Upsampling**: ???

Input:
$3 \times H \times W$




High-res:
$D_1 \times H/2 \times W/2$

High-res:
$D_1 \times H/2 \times W/2$

Predictions:
$H \times W$

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

# In-Network upsampling: “Unpooling”

2x2

**Nearest Neighbor**

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Input: 2 x 2

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 4 |

Output: 4 x 4

**“Bed of Nails”**

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Input: 2 x 2

| | | | |
|---|---|---|---|
| 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 |

Output: 4 x 4

# In-Network upsampling: "Max Unpooling"

**Max Pooling**
Remember which element was max!

| | | | |
|---|---|---|---|
| 1 | 2 | 6 | 3 |
| 3 | 5 | 2 | 1 |
| 1 | 2 | 2 | 1 |
| 7 | 3 | 4 | 8 |

Input: 4 x 4

| | |
|---|---|
| 5 | 6 |
| 7 | 8 |

Output: 2 x 2

…

Rest of the network

**Max Unpooling**
Use positions from pooling layer

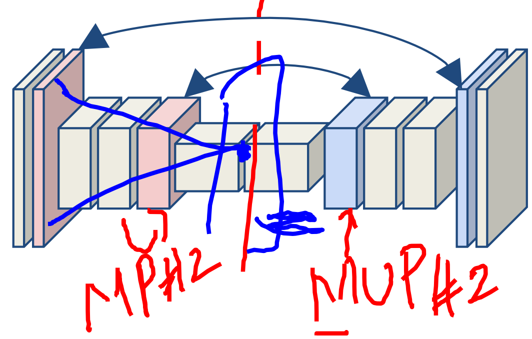
| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Input: 2 x 2

| | | | |
|---|---|---|---|
| 0 | 0 | 2 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 4 |

Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers



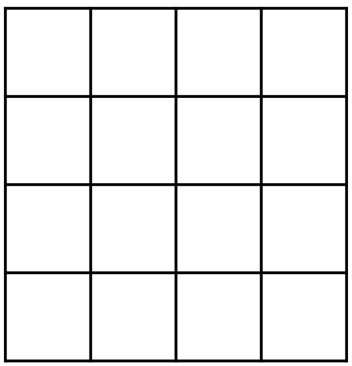

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Transposed Convolutions

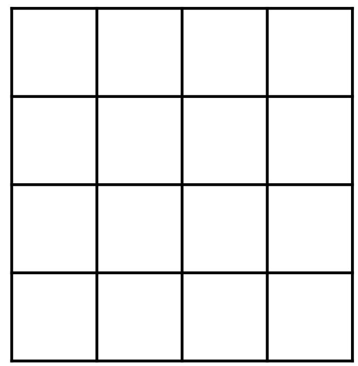
- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

# Learnable Upsampling: Transpose Convolution

**Recall:** Typical 3 x 3 convolution, stride 1 pad 1



Input: 4 x 4

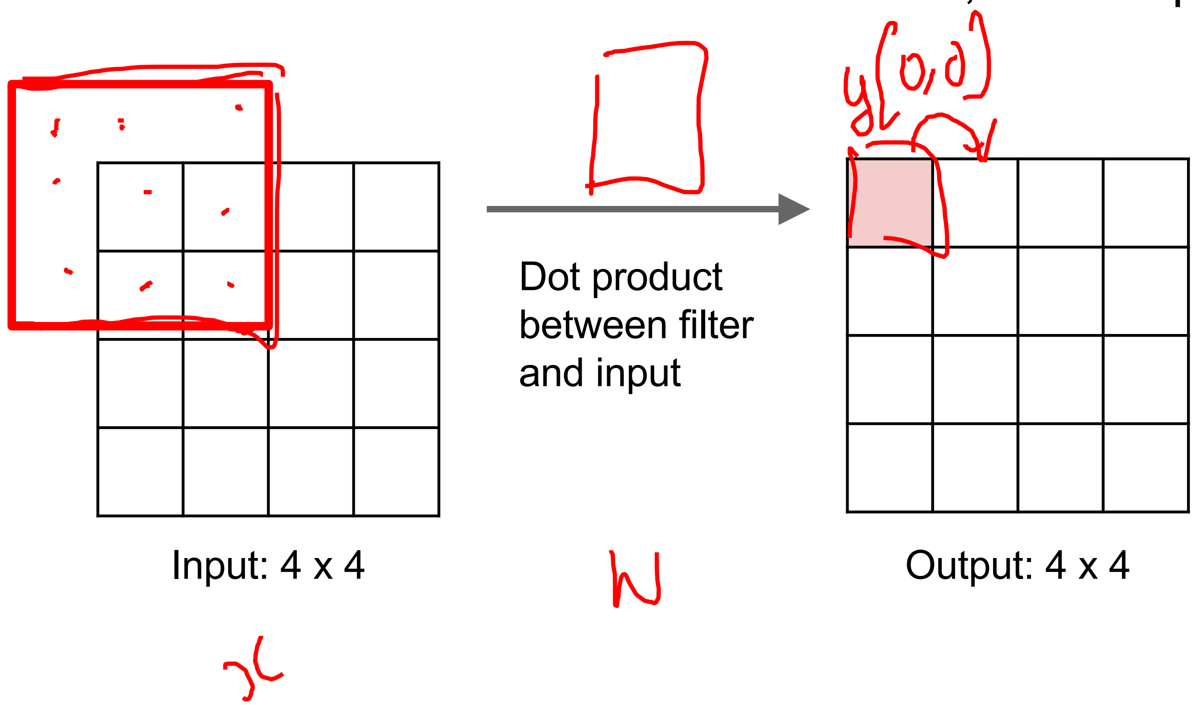


Output: 4 x 4

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

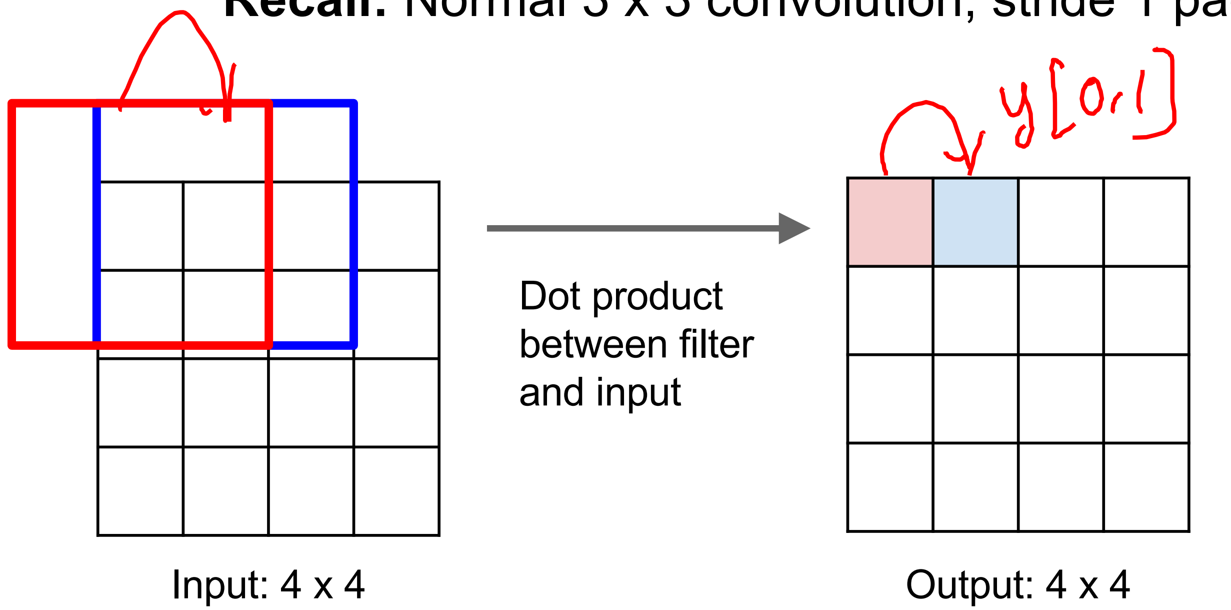
# Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 1 pad 1



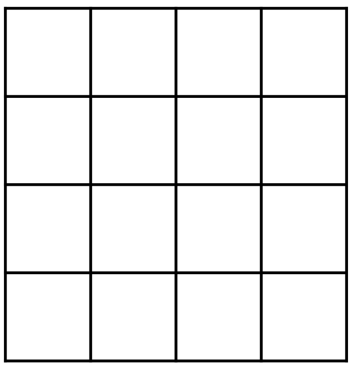
# Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 1 pad 1

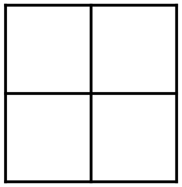


# Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 2 pad 1



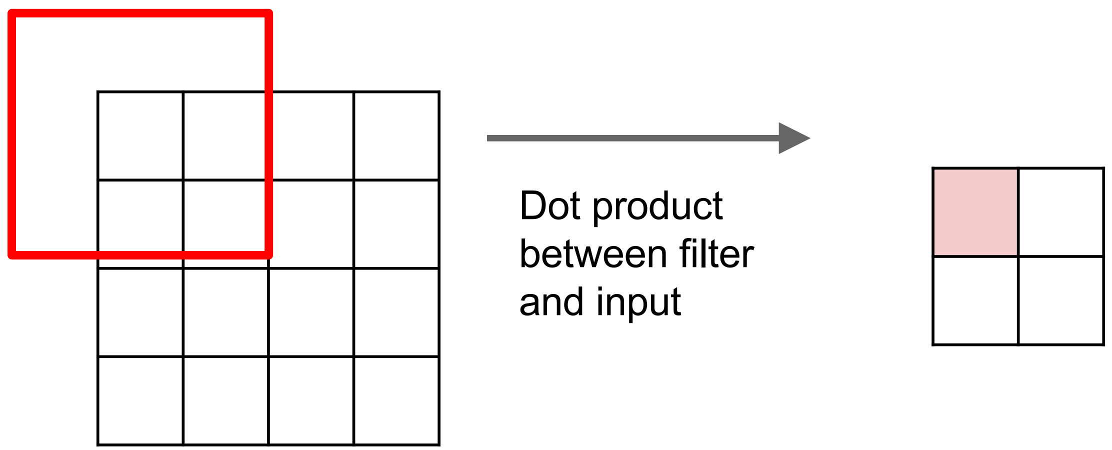
Input: 4 x 4



Output: 2 x 2

# Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 2 pad 1

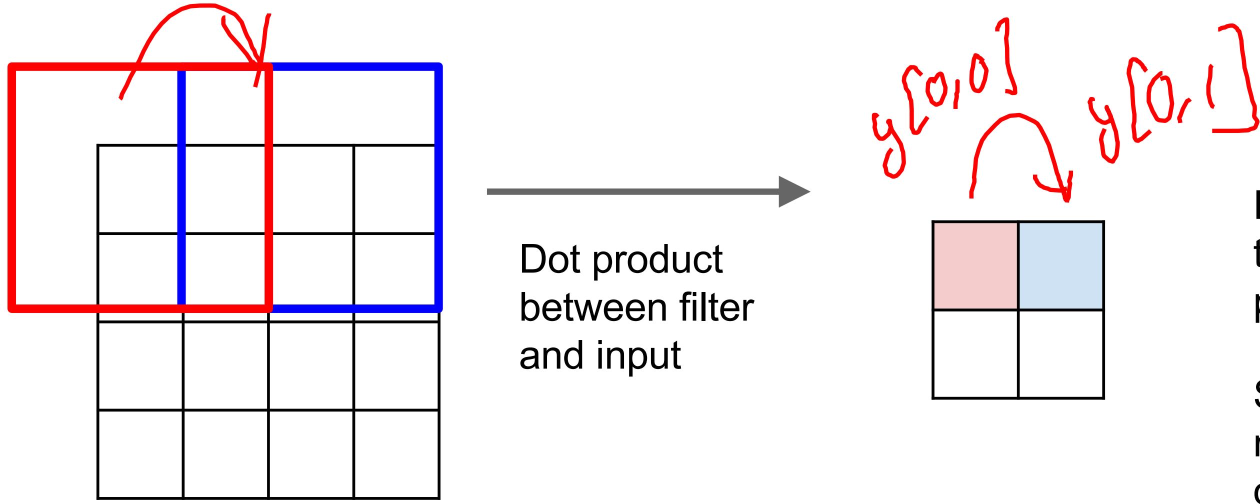



Input: 4 x 4

Output: 2 x 2

# Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 2 pad 1




Input: 4 x 4

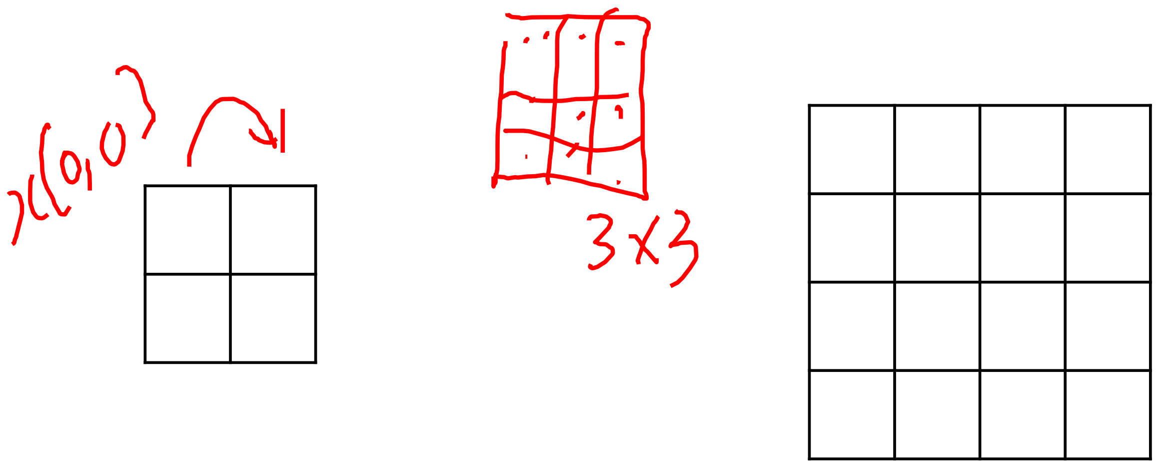
Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

# Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



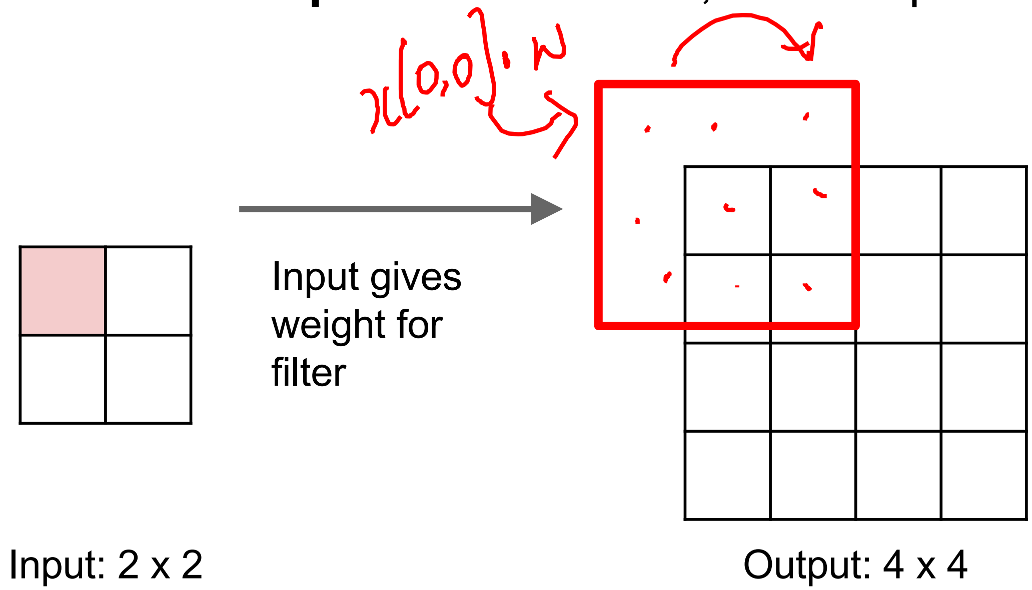

Input: 2 x 2

Output: 4 x 4

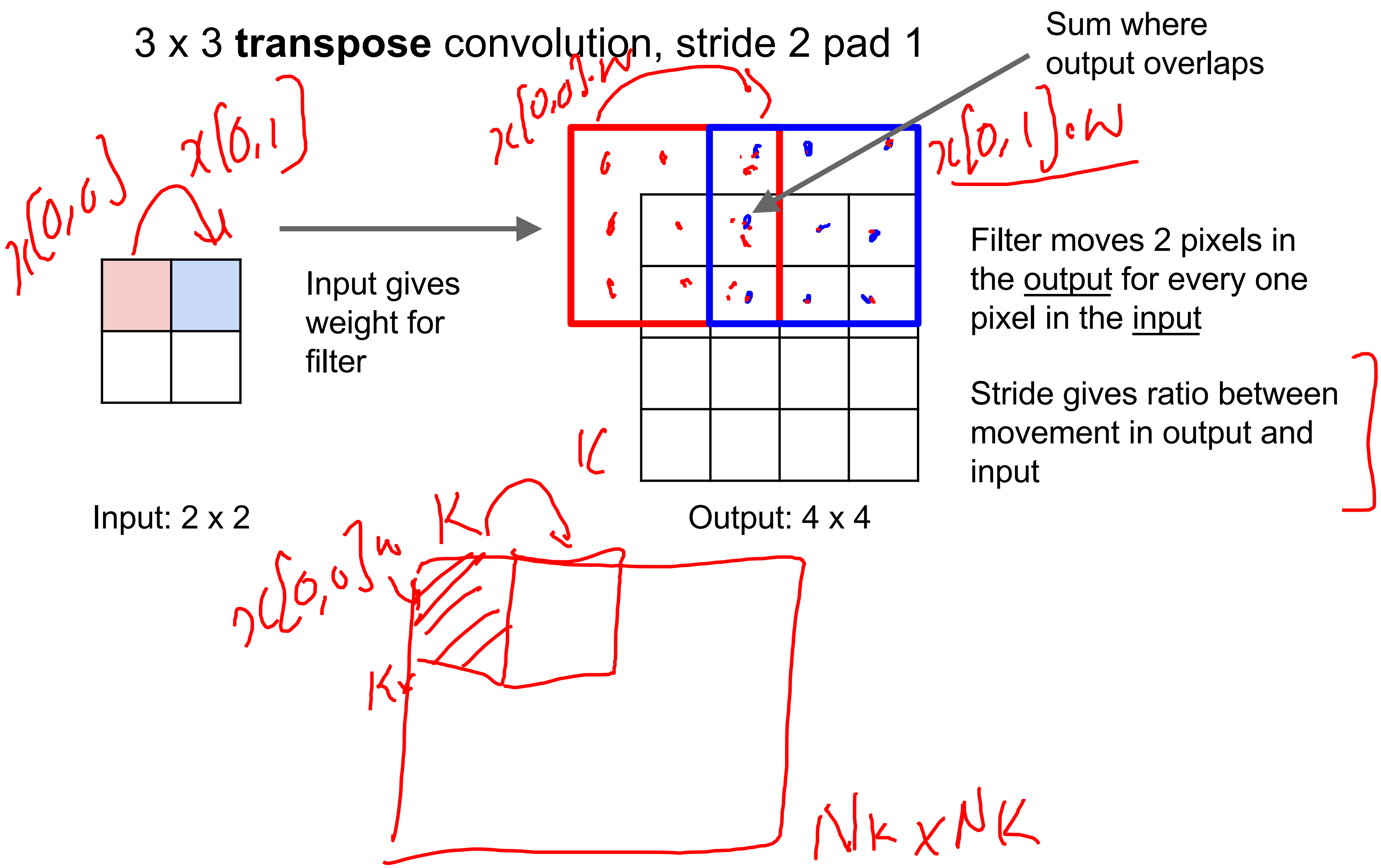
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



# Learnable Upsampling: Transpose Convolution

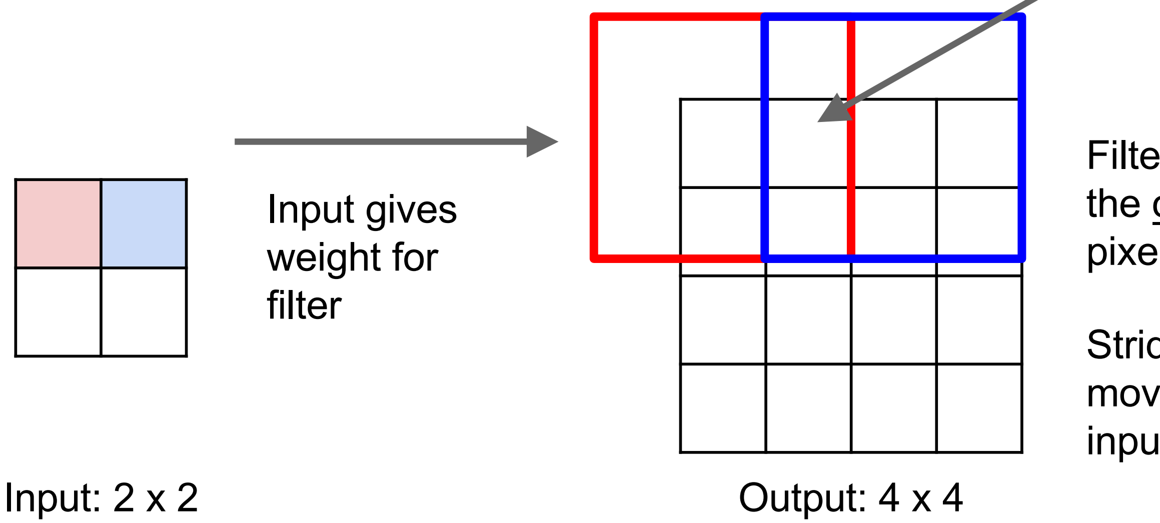



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Learnable Upsampling: Transpose Convolution

**Other names:**
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

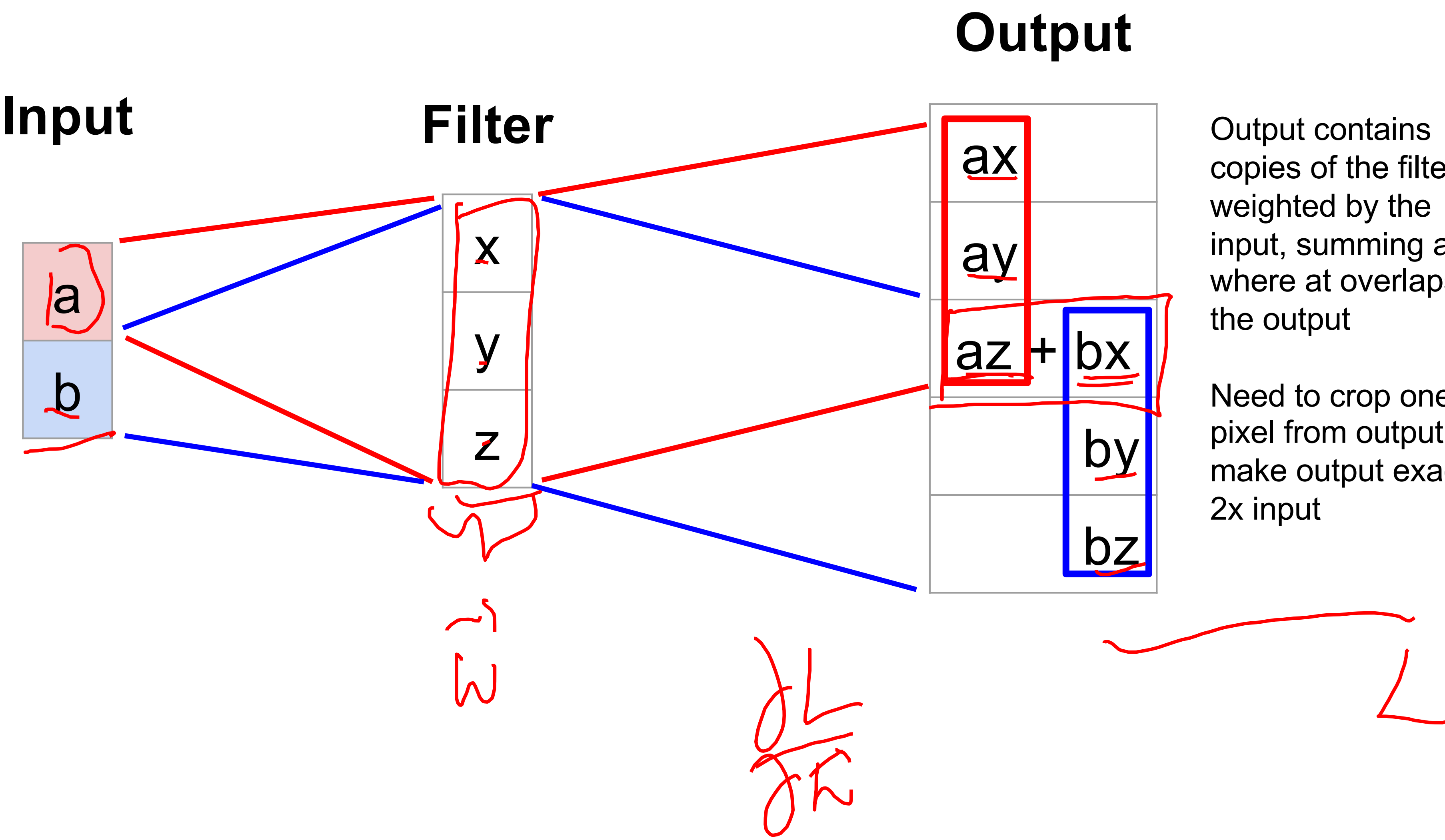
3 x 3 **transpose** convolution, stride 2 pad 1




Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

# Transpose Convolution: 1D Example




Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

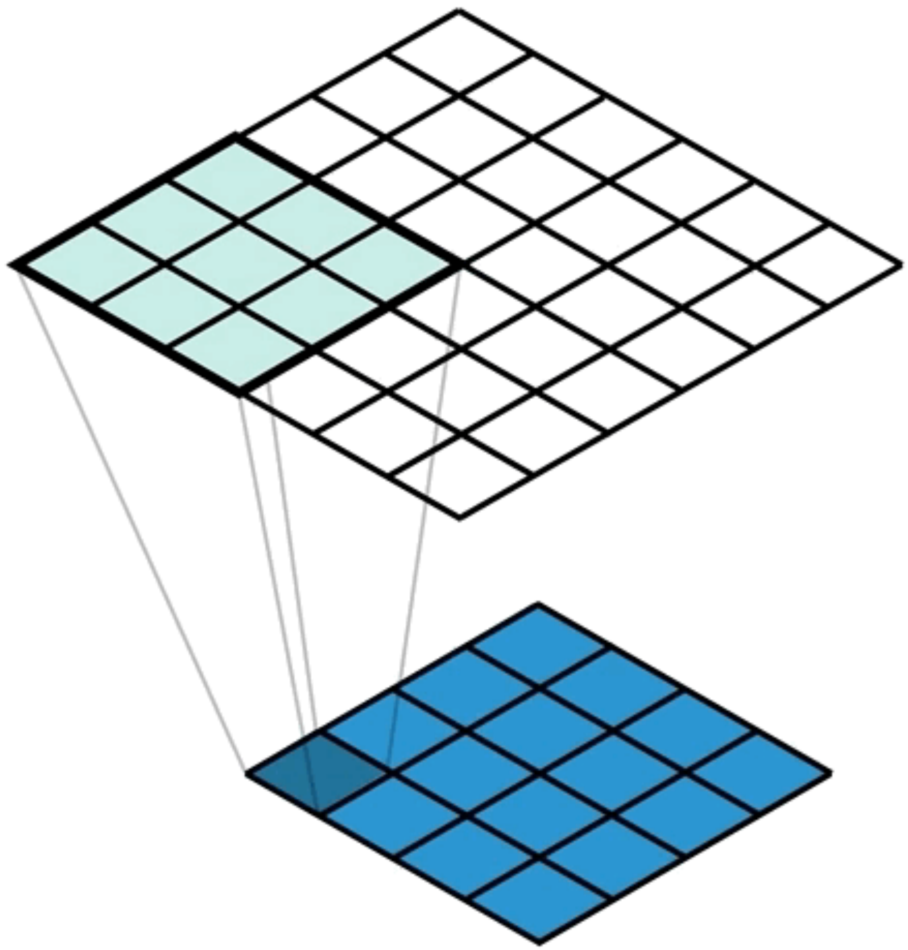


Figure Credit: https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

# Transposed Convolution

- https://distill.pub/2016/deconv-checkerboard/

# In-Network upsampling: “Unpooling”

**Nearest Neighbor**

Input: 2 x 2

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Output: 4 x 4

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 4 |

**“Bed of Nails”**

Input: 2 x 2

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Output: 4 x 4

| | | | |
|---|---|---|---|
| 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 |

# Why this operation?

# What is deconvolution?

- (Non-blind) Deconvolution

(Conv) $y = x * w$

Deconv → Blind : Given y, produce x, w

→ Non-blind: Given y & w, find x

# What is deconvolution?

- (Non-blind) Deconvolution

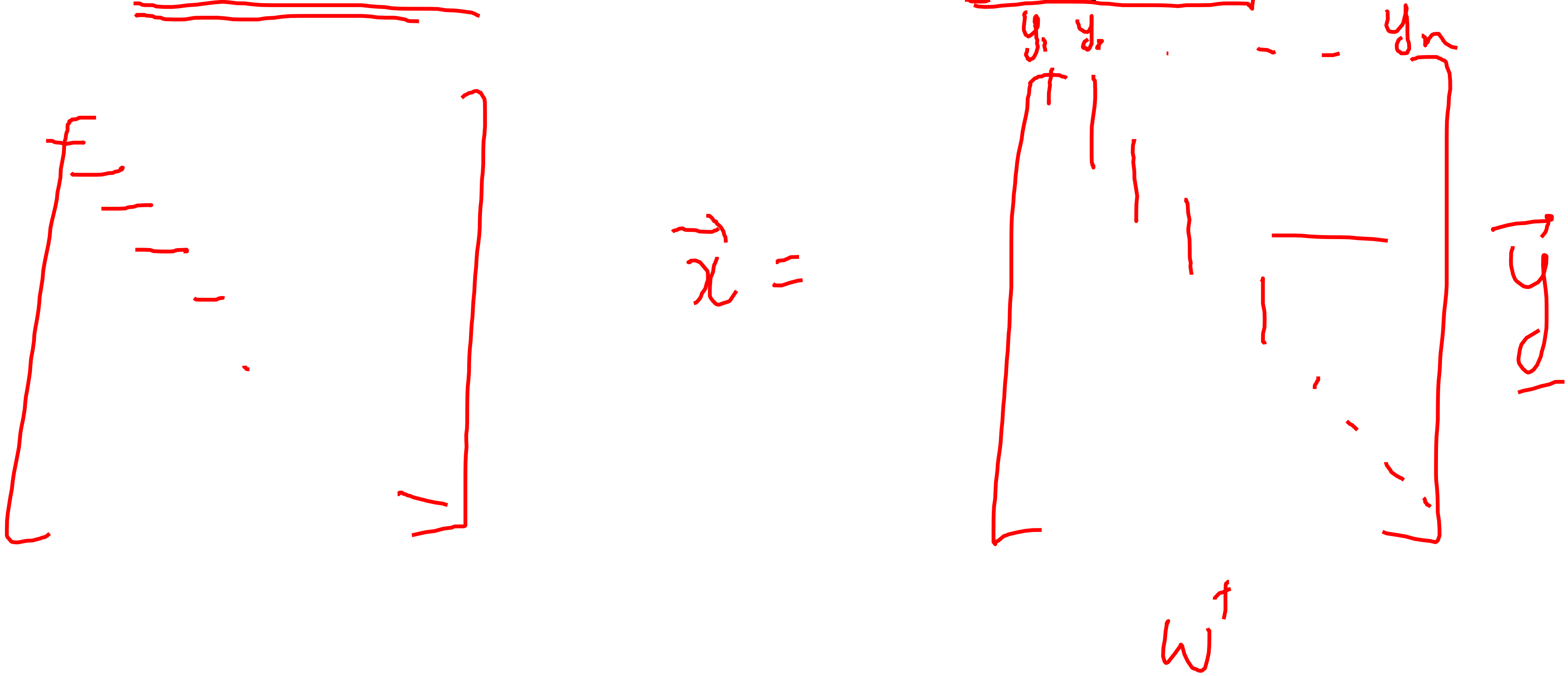
$\vec{y} = W\vec{x} \iff x = W^T y$

If $W$ orthonormal

$\vec{w} = [-1 \; 0 \; +1]$

$$
y = w * x \qquad \underbrace{\begin{bmatrix} w_k & 0 & \dots & 0 & 0 \\ w_{k-1} & w_k & \dots & 0 & 0 \\ w_{k-2} & w_{k-1} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_1 & \dots & \dots & w_k & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & w_1 & \dots & w_{k-1} & w_k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \vdots & w_1 & w_2 \\ 0 & 0 & \vdots & 0 & w_1 \end{bmatrix}}_{W} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}
$$

What does “deconvolution” have to do with “transposed convolution”?



# “transposed convolution” is a convolution!

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

# "transposed convolution" is a convolution!

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

[x y z]

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

[z y x]

# “transposed convolution” is a convolution!

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

# But not always

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

# But not always

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

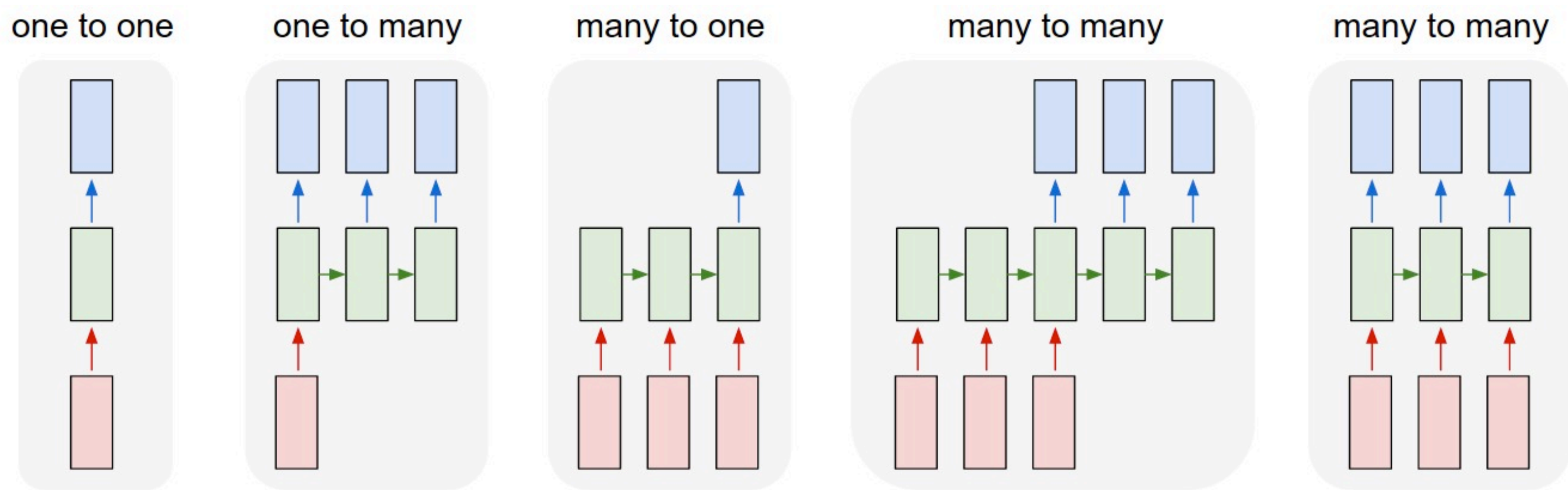
When stride>1, convolution transpose is no longer a normal convolution!

# Plan for Today

- (Finish) Convolutional Neural Networks
  - Transposed convolutions

- Recurrent Neural Networks (RNNs)
  - A new model class
  - Learning: BackProp Through Time (BPTT)

# New Topic: RNNs



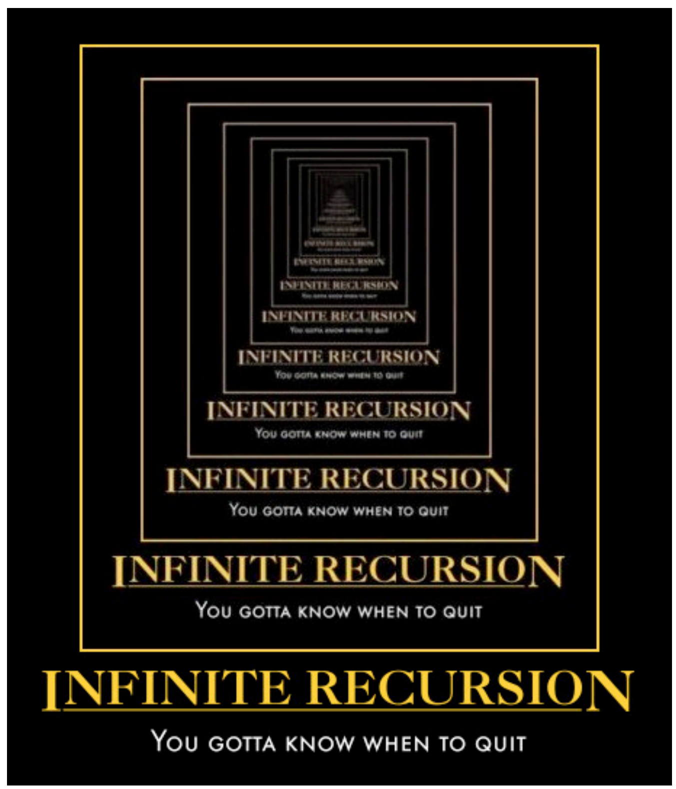





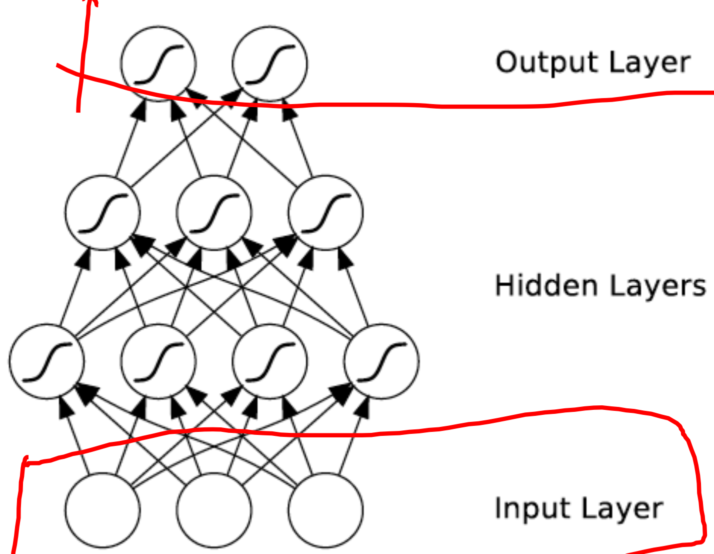

Image Credit: Andrej Karpathy

# New Words

- Recurrent Neural Networks (RNNs)
- Recursive Neural Networks
  - General family; think graphs instead of chains
- Types:
  - "Vanilla" RNNs (Elman Networks)
  - Long Short Term Memory (LSTMs)
  - Gated Recurrent Units (GRUs)
  - …
- Algorithms
  - BackProp Through Time (BPTT)
  - BackProp Through Structure (BPTS)

# What's wrong with MLPs?

- Problem 1: Can't model sequences
  - Fixed-sized Inputs & Outputs
  - No temporal structure




$y \in \mathbb{R}^c$

$\vec{x} \in \mathbb{R}^{d_1}$

# What's wrong with MLPs?

- Problem 1: Can't model sequences
  - Fixed-sized Inputs & Outputs
  - No temporal structure

- Problem 2: Pure feed-forward processing
  - No "memory", no feedback

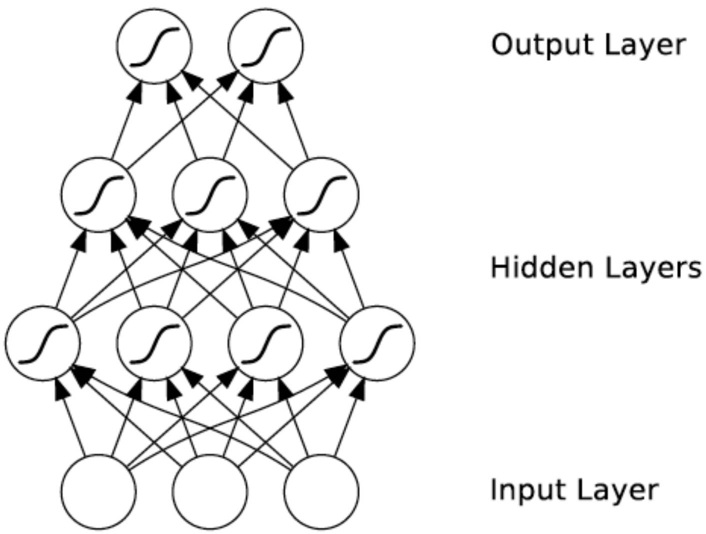




Image Credit: Alex Graves, book

# Why model sequences?

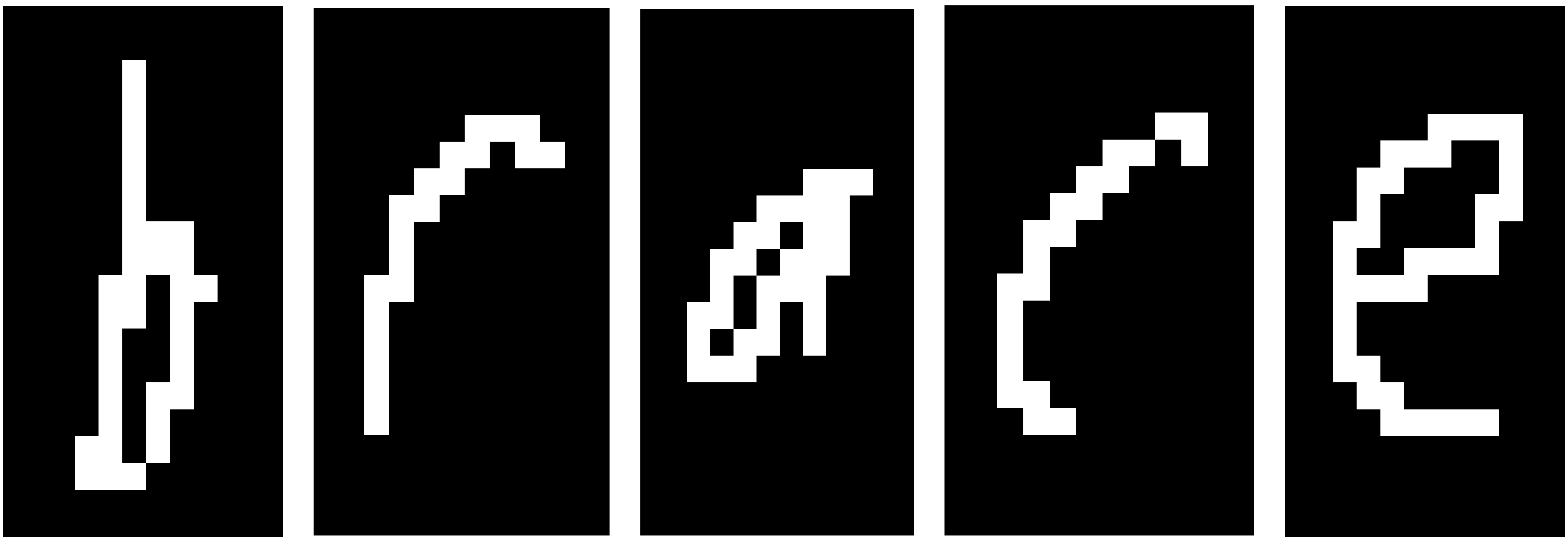


Figure Credit: Carlos Guestrin

# Why model sequences?

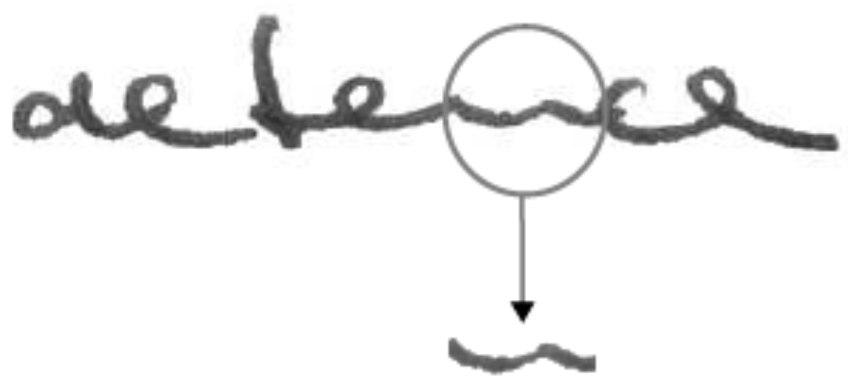
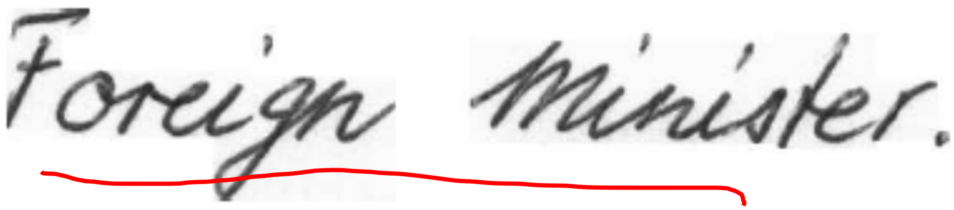


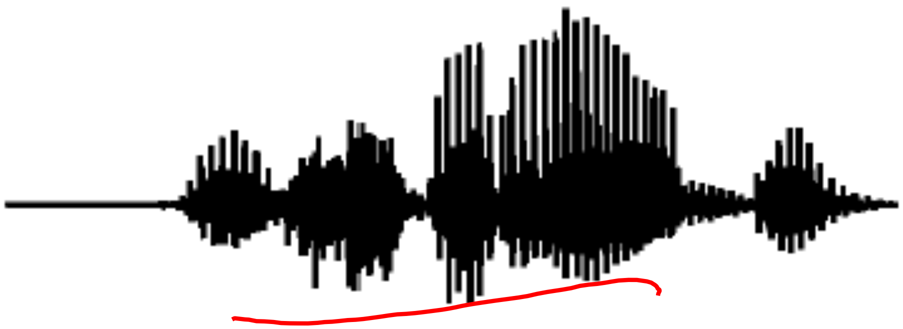


Image Credit: Alex Graves

# Sequences are everywhere…



FOREIGN MINISTER.



THE SOUND OF

$$
\begin{array}{cccccccc}
 & a_1{=}2 & a_2{=}0 & a_3{=}1 & a_4{=}3 & a_5{=}4 & a_6{=}2 & a_7{=}5 \\
\boldsymbol{x} = & \text{bringen} & \text{sie} & \text{bitte} & \text{das} & \text{auto} & \text{zurück} & .
\end{array}
$$

$$\boldsymbol{y} = \text{please} \quad \text{return} \quad \text{the} \quad \text{car} \quad .$$



Image Credit: Alex Graves and Kevin Gimpel

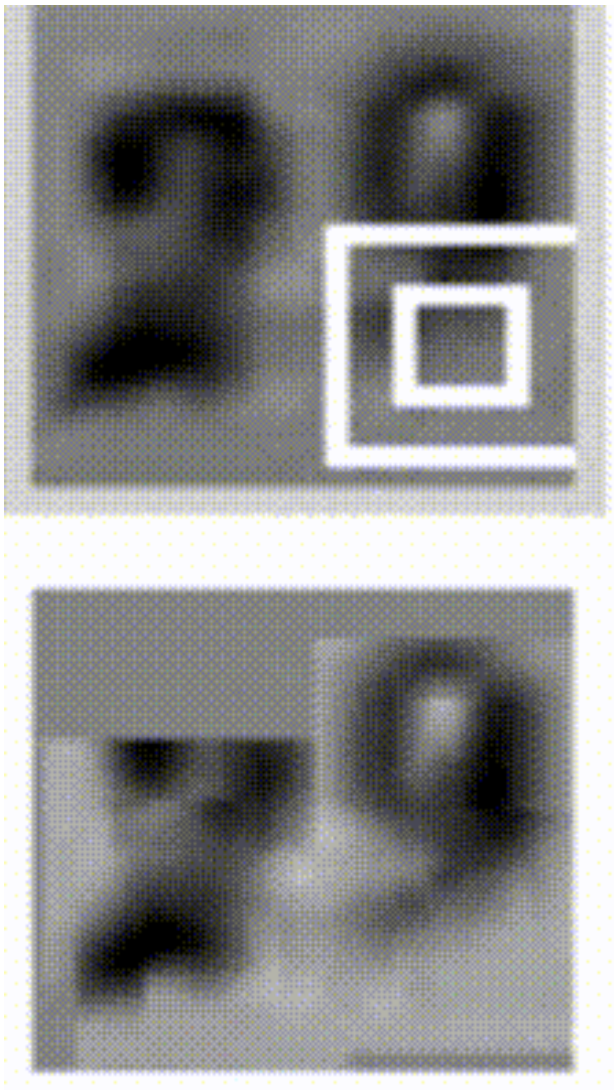
# Even where you might not expect a sequence…

Classify images by taking a series of “glimpses”

Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

# Even where you might not expect a sequence…

- Output ordering = sequence



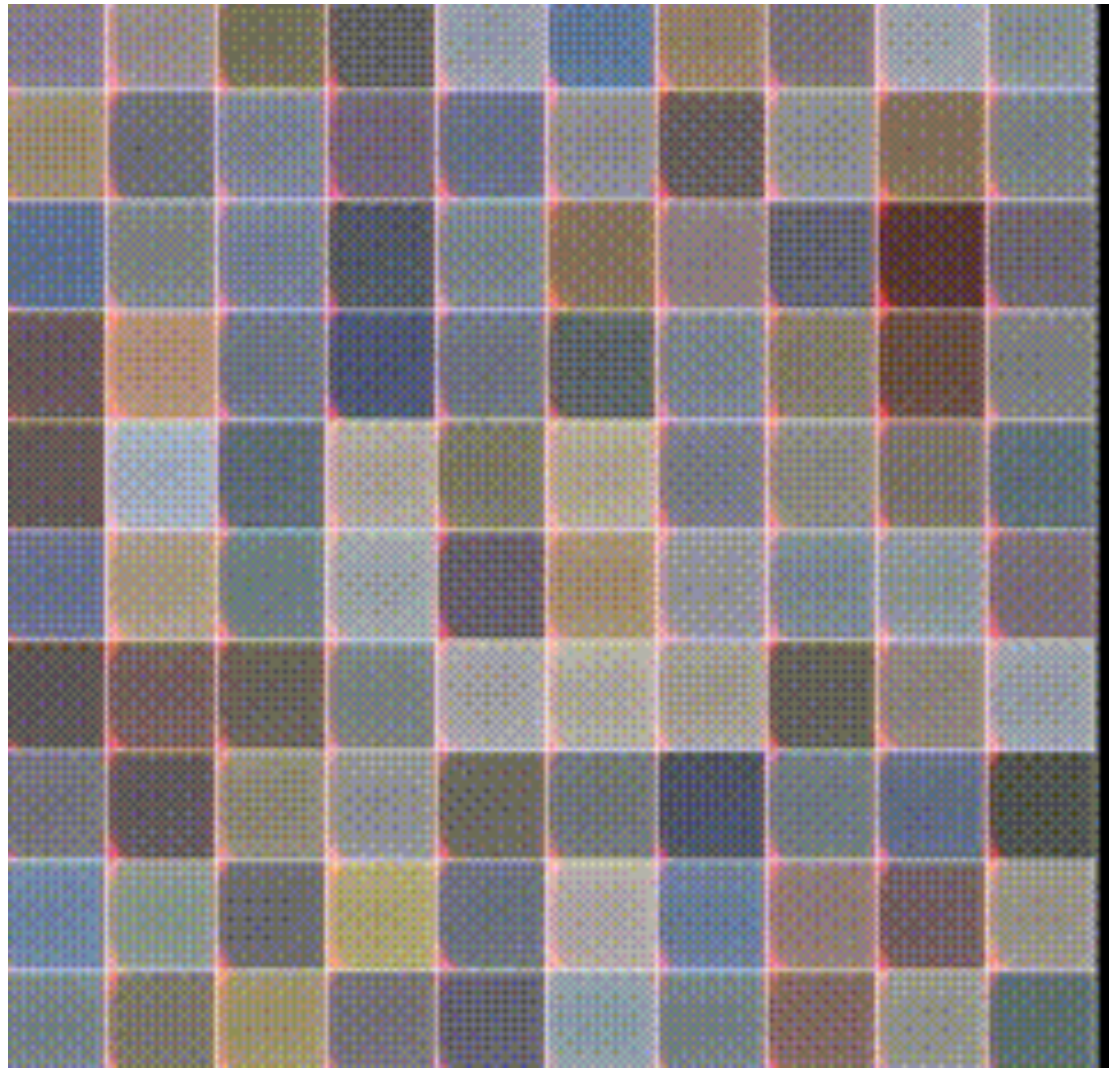
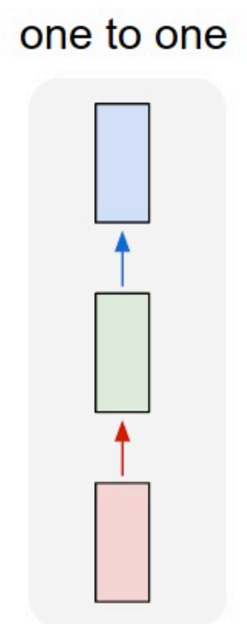




Image Credit: Ba et al.; Gregor et al

# Sequences in Input or Output?

- It's a spectrum…



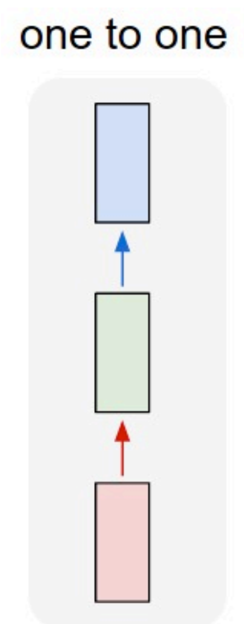

Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems

# Sequences in Input or Output?

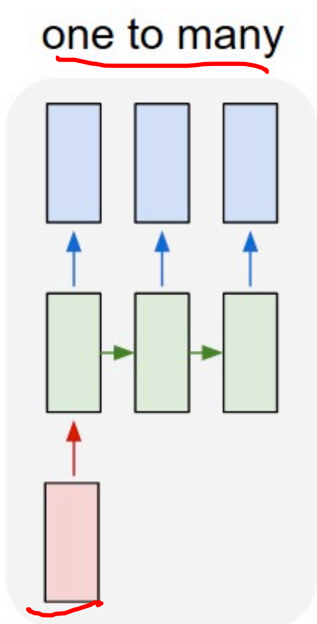
- It's a spectrum…




Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems



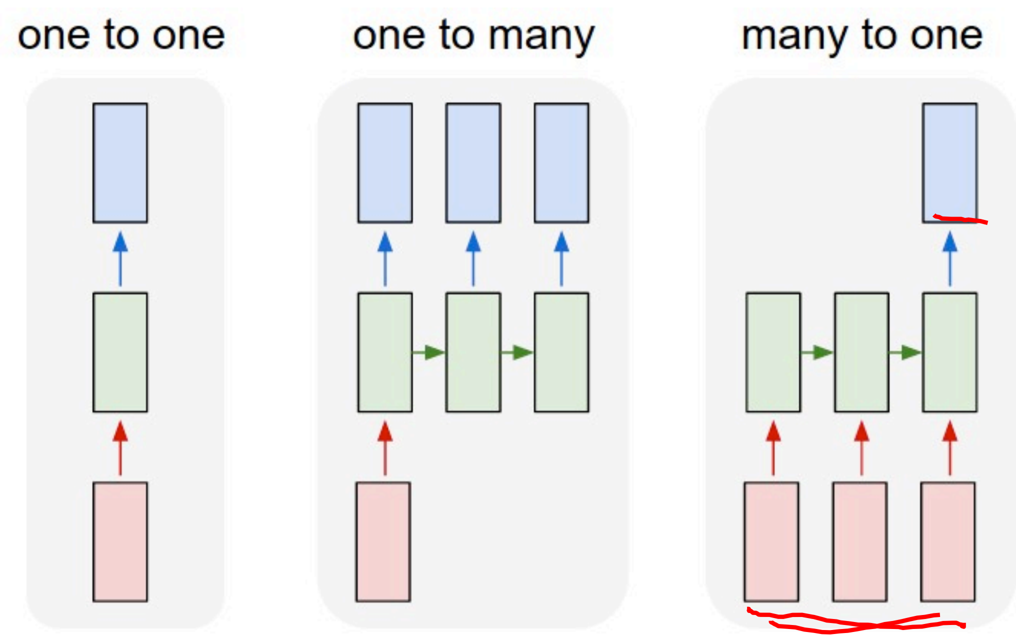

Input: No sequence

Output: Sequence

Example: Im2Caption

# Sequences in Input or Output?

- It's a spectrum…




Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems

Input: No sequence

Output: Sequence

Example: Im2Caption

Input: Sequence

Output: No sequence

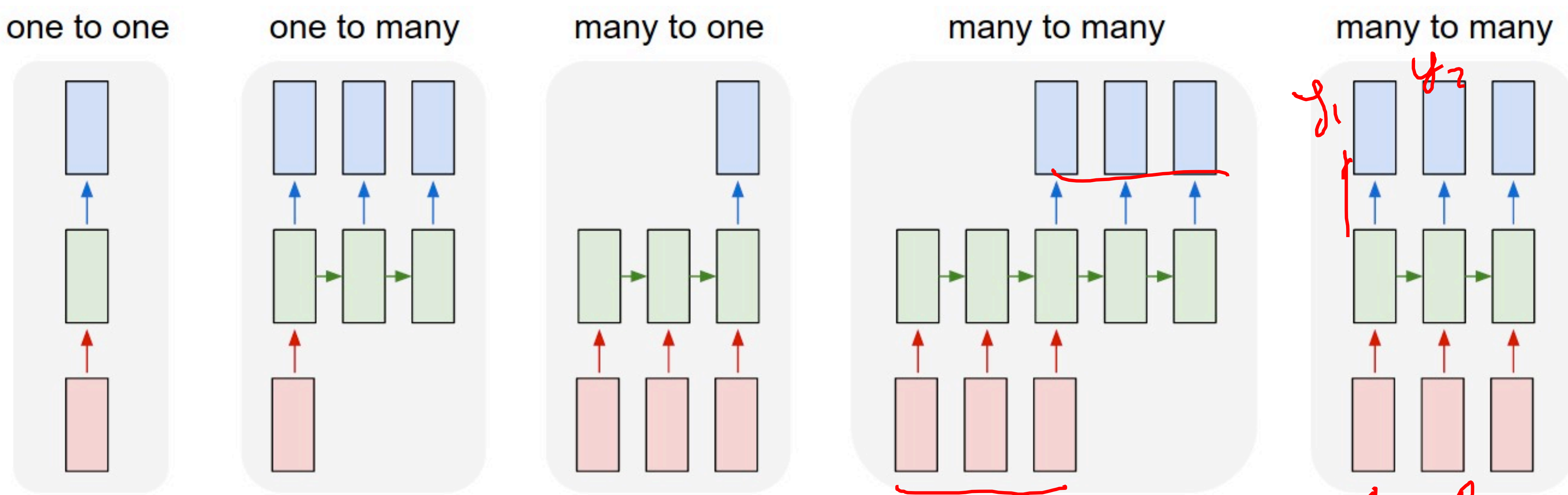
Example: sentence classification, multiple-choice question answering



Image Credit: Andrej Karpathy

# Sequences in Input or Output?

- It's a spectrum…




Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems

Input: No sequence

Output: Sequence

Example: Im2Caption

Input: Sequence

Output: No sequence

Example: sentence classification, multiple-choice question answering

Input: Sequence

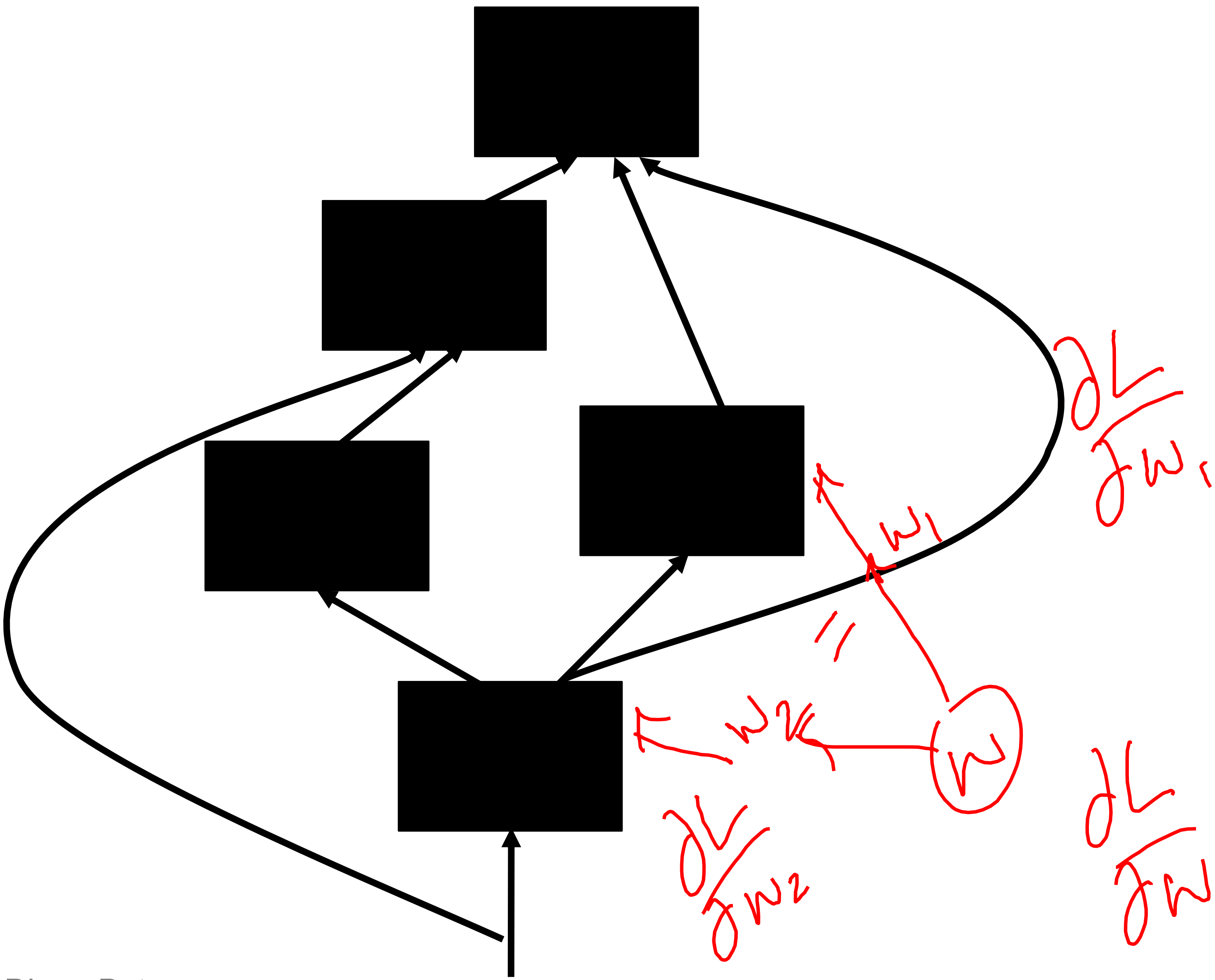
Output: Sequence

Example: machine translation, video classification, video captioning, open-ended question answering

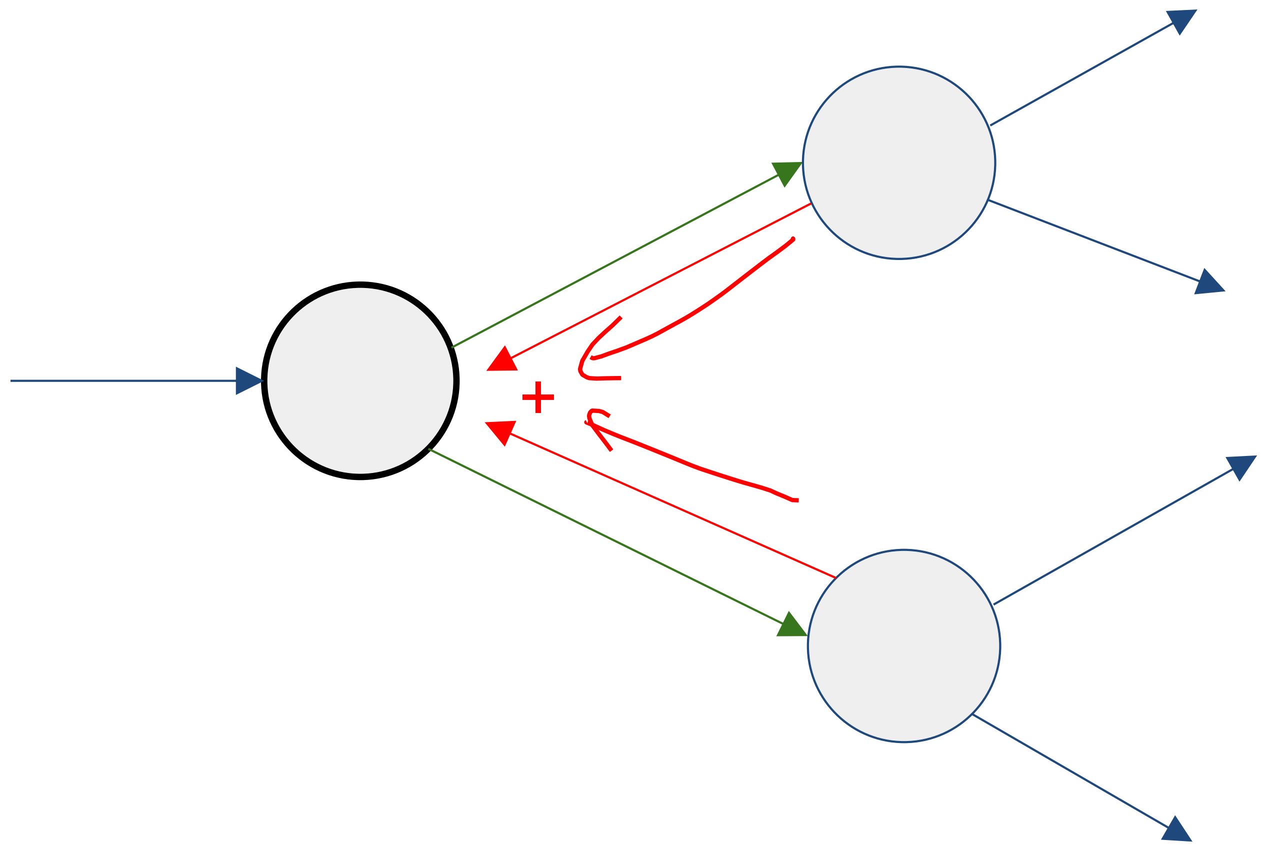
# 2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients

# Computational Graph



# Gradients add at branches




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# 2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients
- “Unrolling”
  - in computation graphs with parameter sharing

# How do we model sequences?

- No input

$$s_t = f_\theta(s_{t-1})$$

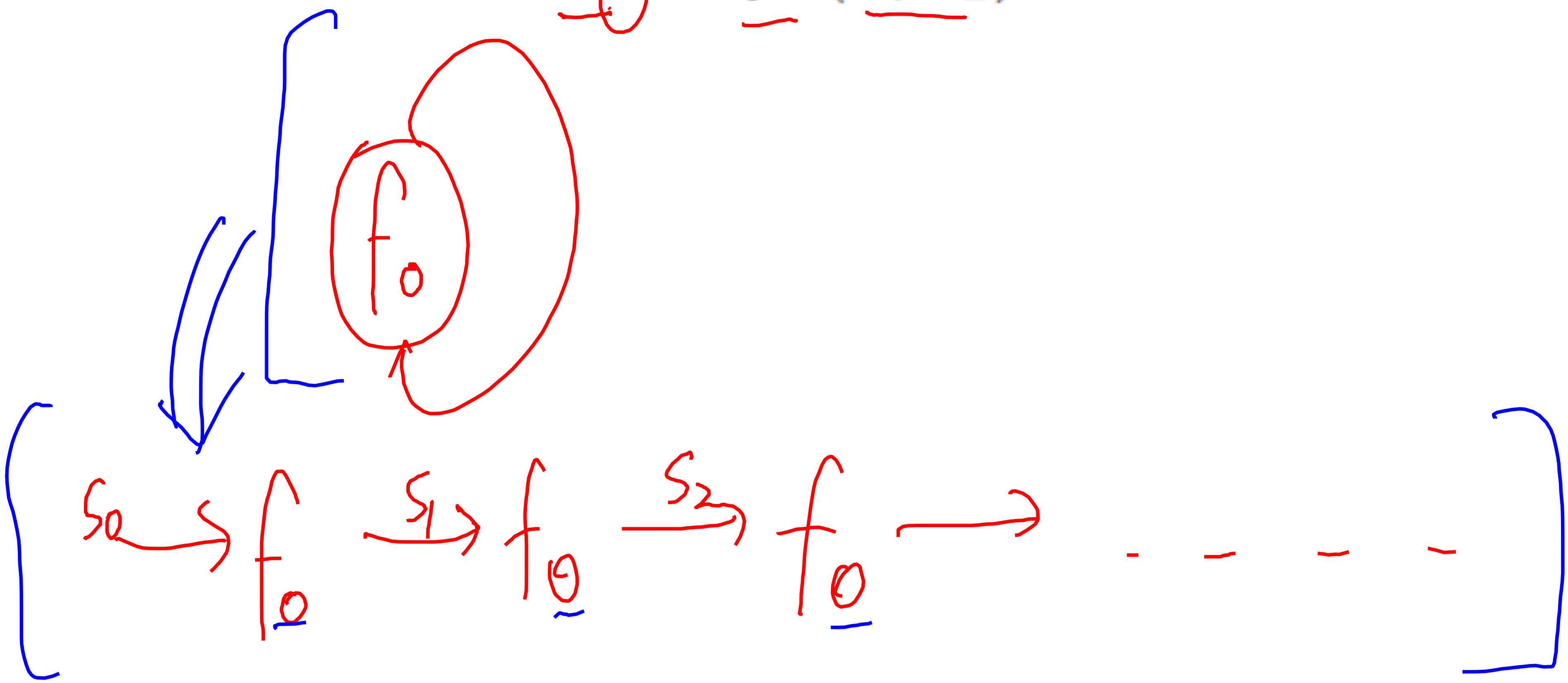


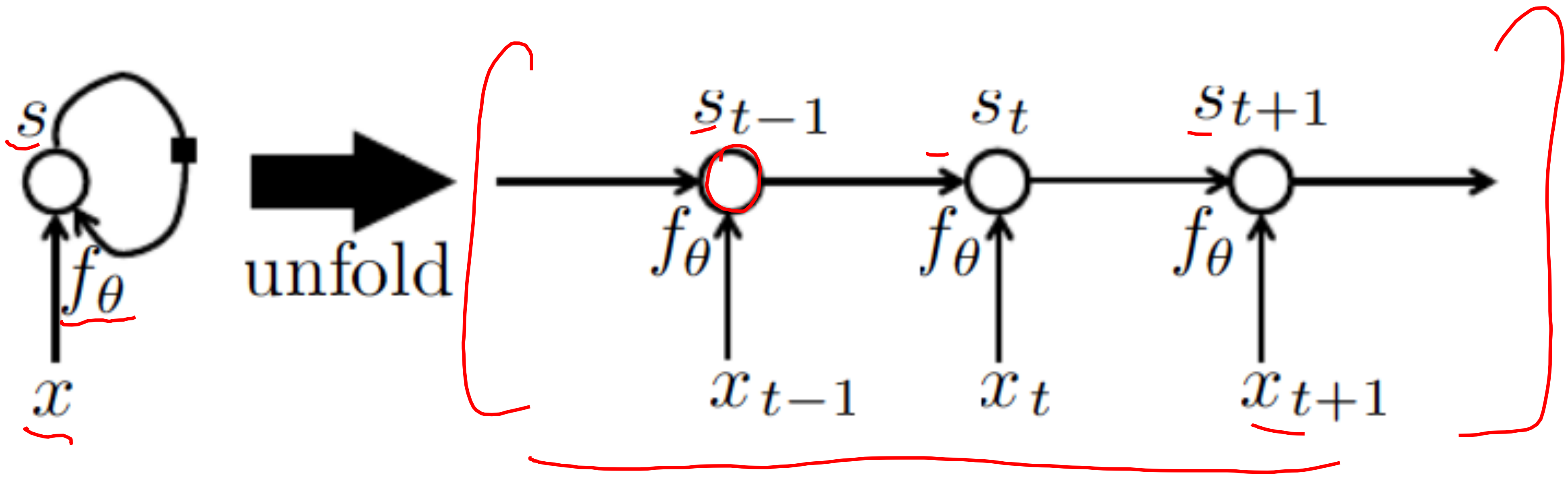



Image Credit: Bengio, Goodfellow, Courville

# How do we model sequences?

- With inputs

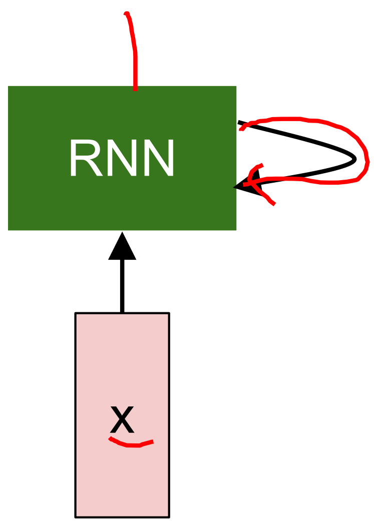
$$\boldsymbol{s}_t = f_\theta(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t)$$



# 2 Key Ideas

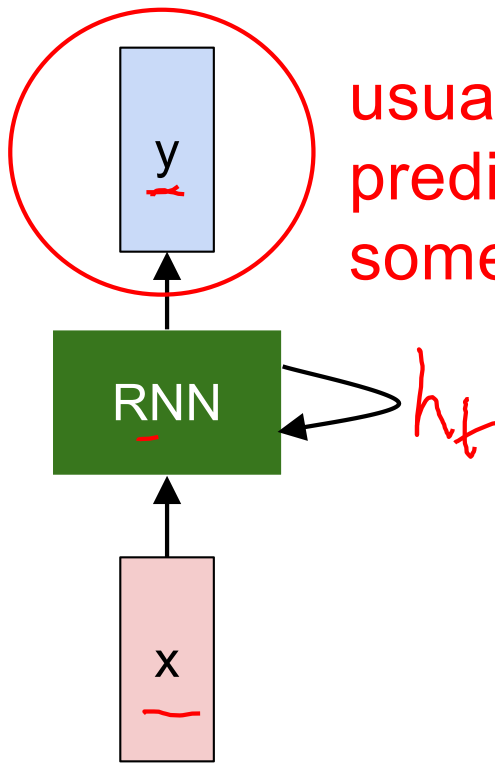
- Parameter Sharing
  - in computation graphs = adding gradients
- "Unrolling"
  - in computation graphs with parameter sharing
- Parameter sharing + Unrolling
  - Allows modeling arbitrary sequence lengths!
  - Keeps numbers of parameters in check

# Recurrent Neural Network




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Recurrent Neural Network




usually want to predict a vector at some time steps

# Recurrent Neural Network

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

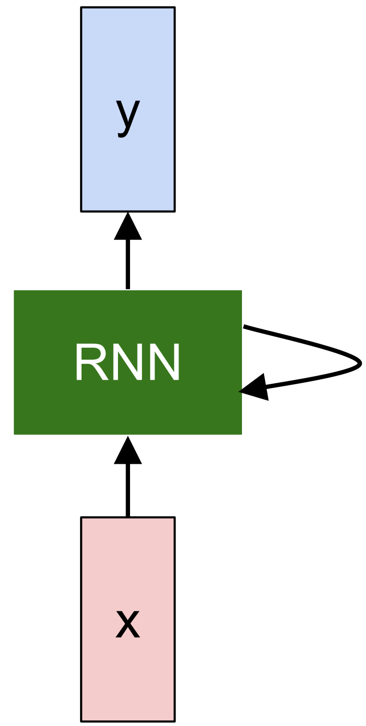
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step



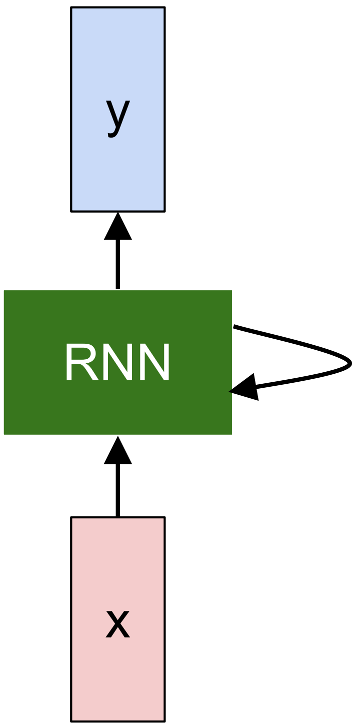

$$\left[\, y_t = f_{W_2}(h_t) \,\right]$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Recurrent Neural Network

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

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

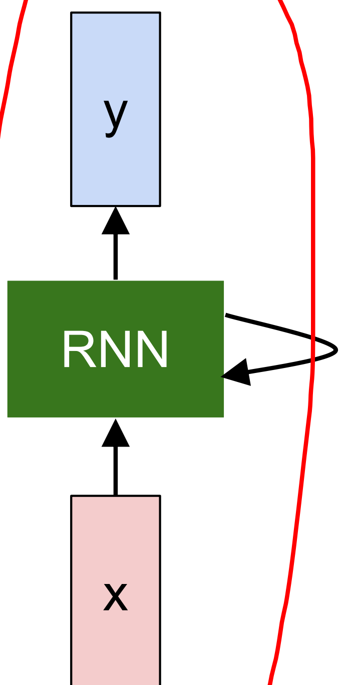



Notice: the same function and the same set of parameters are used at every time step.

$h_t \in \mathbb{R}^{d_2}$

# (Vanilla) Recurrent Neural Network $x_t \in \mathbb{R}^{d_1}$

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




$$y_t = W_{hy} h_t + b_y$$

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

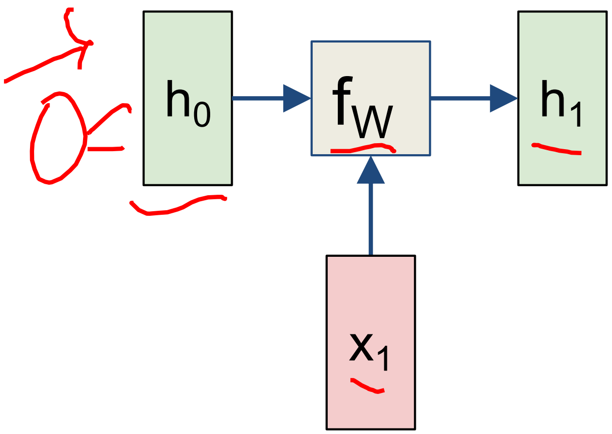
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

$$\begin{bmatrix} W_{hh} & W_{xh} \end{bmatrix} \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

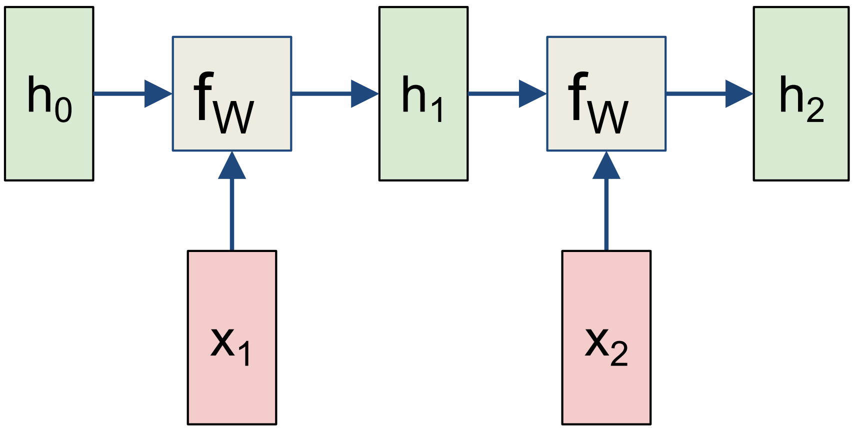
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

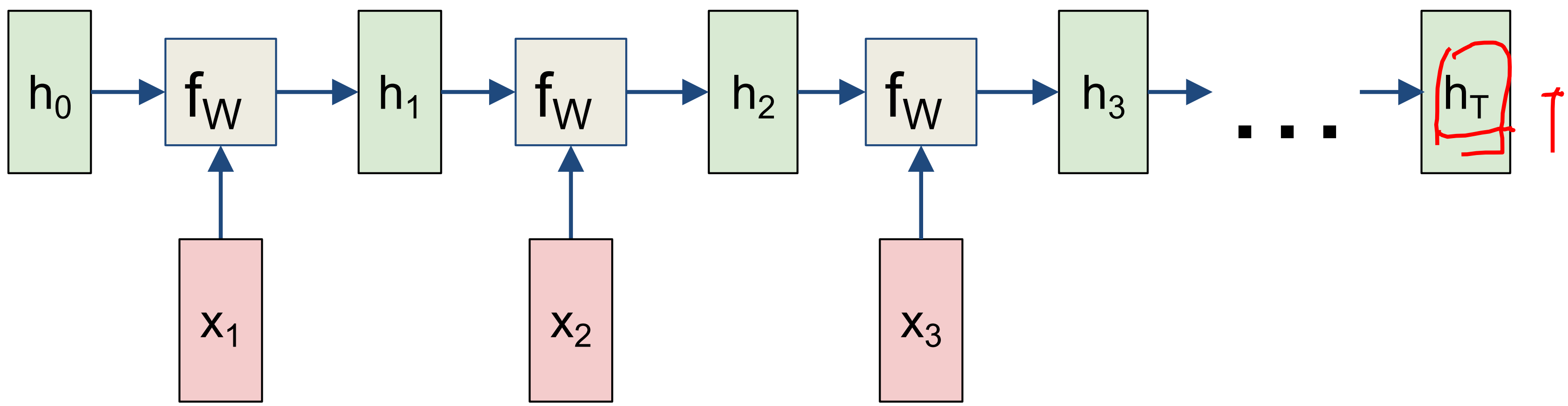
# RNN: Computational Graph



# RNN: Computational Graph

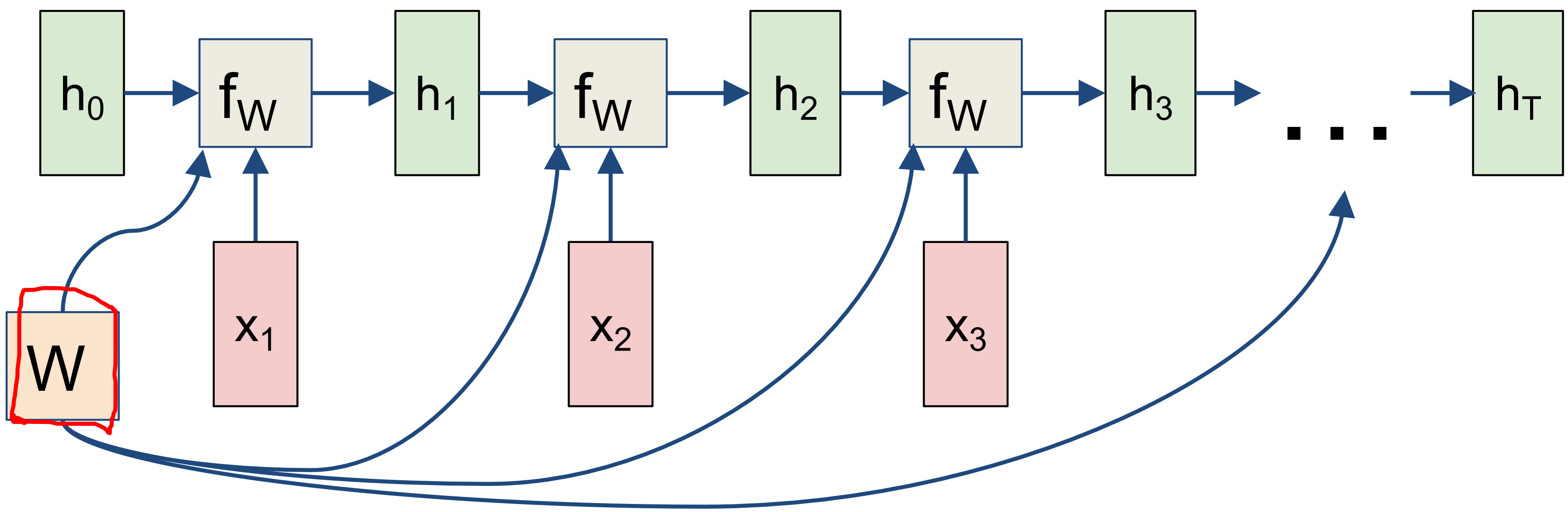


# RNN: Computational Graph



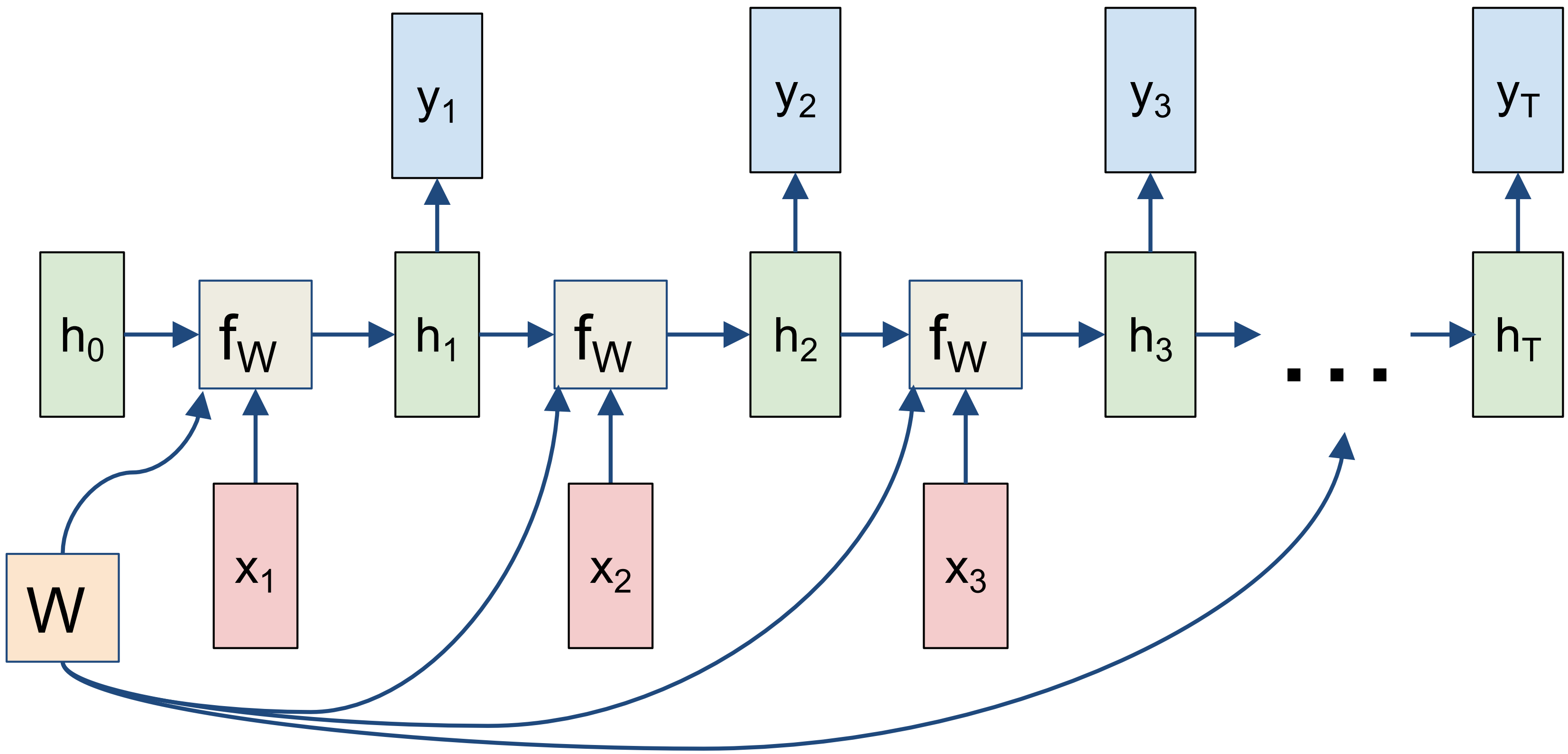
# RNN: Computational Graph

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

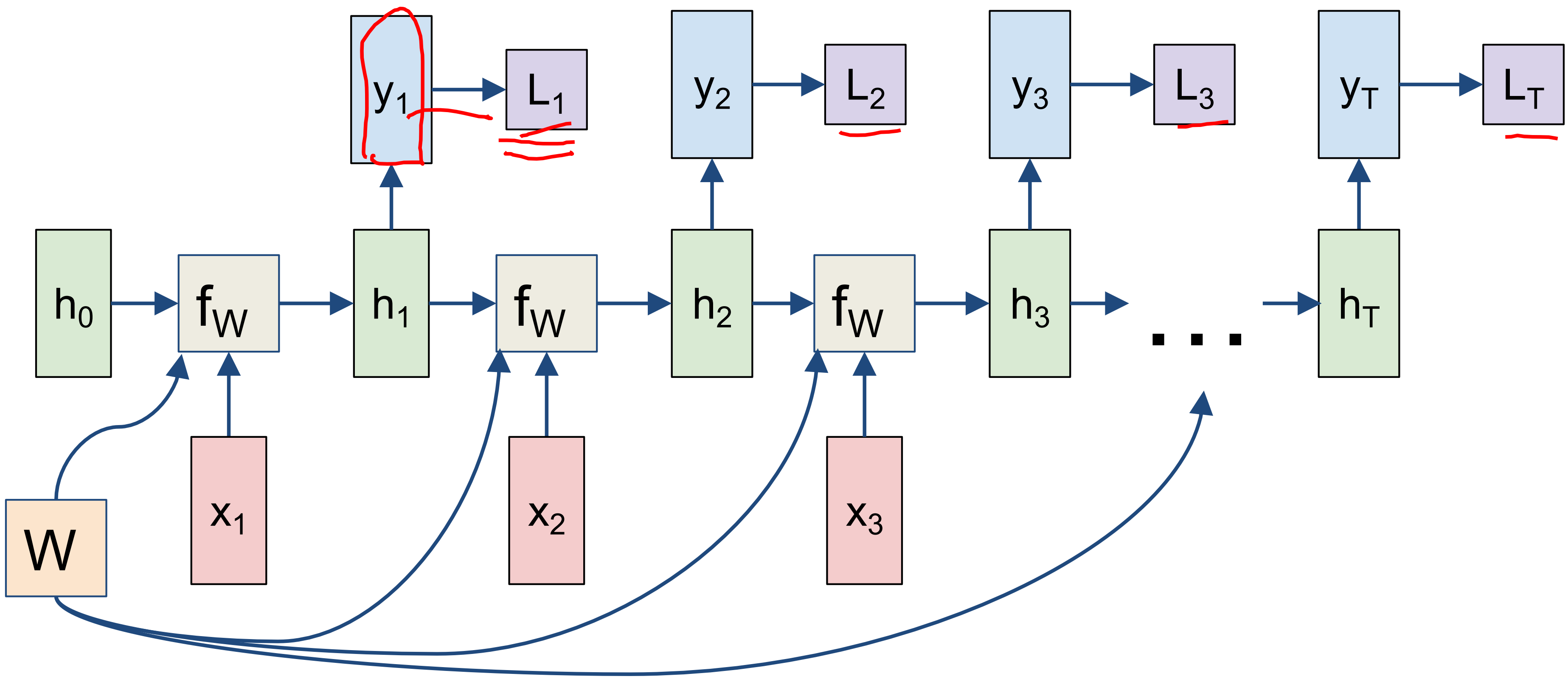



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# RNN: Computational Graph: Many to Many

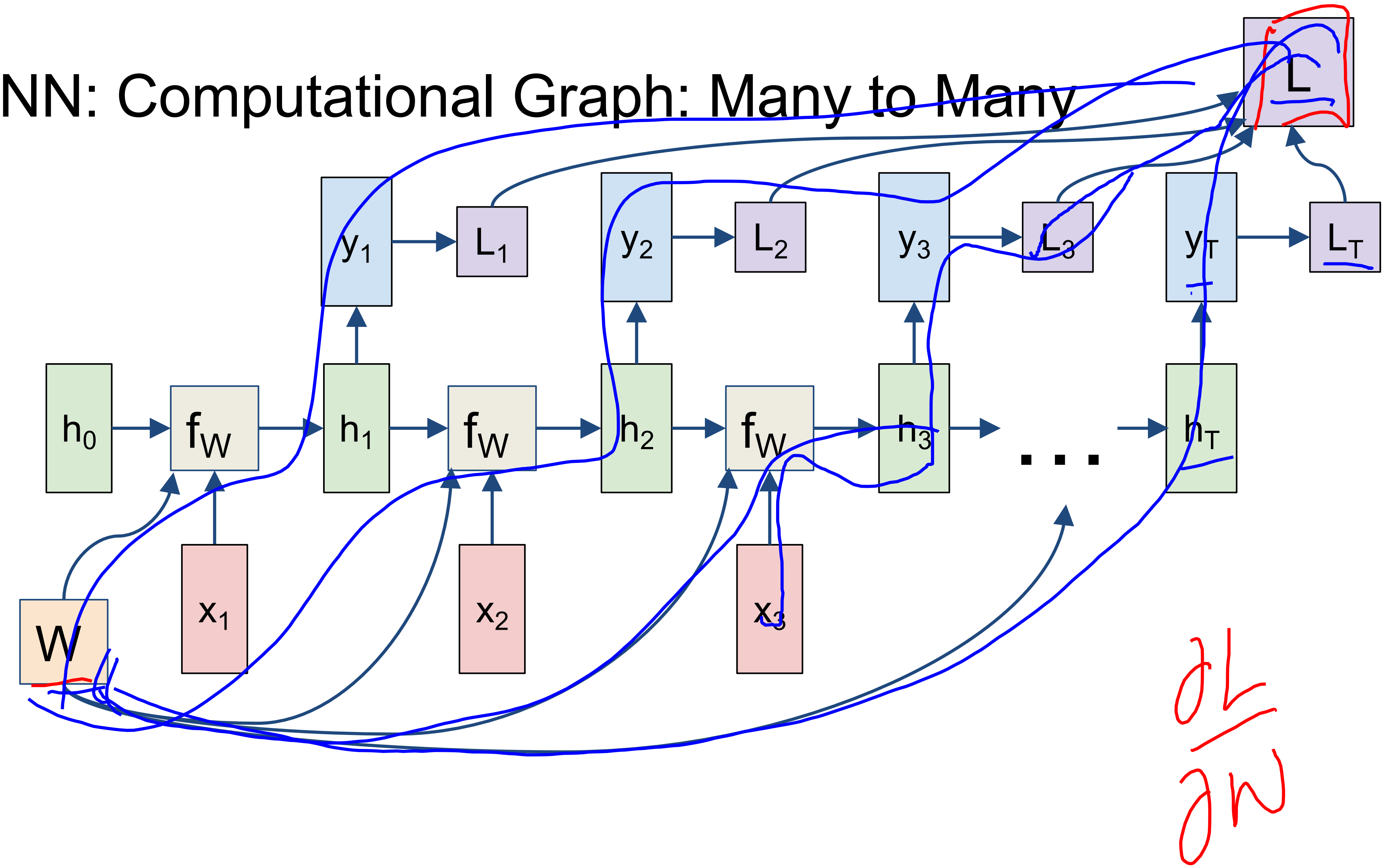


# RNN: Computational Graph: Many to Many



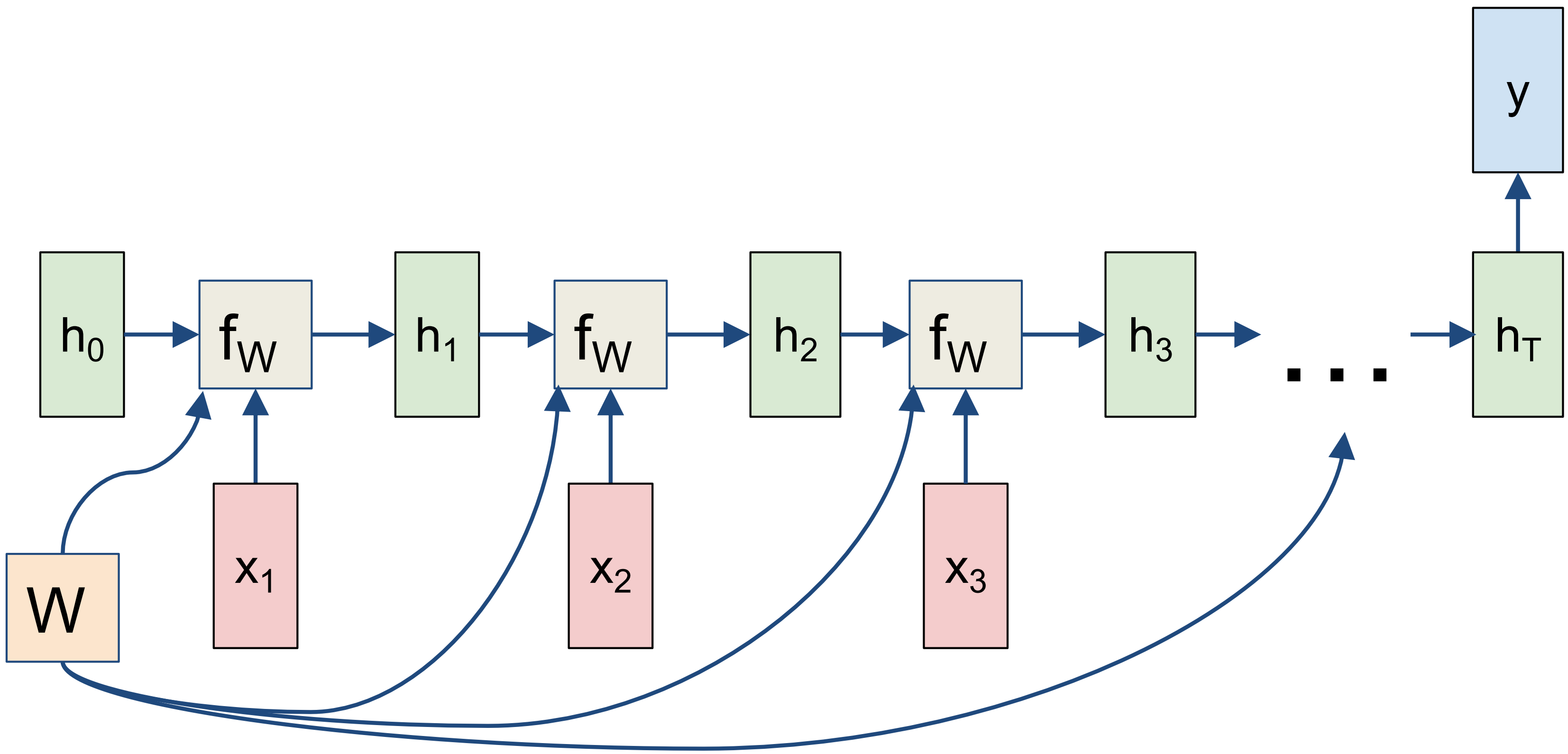

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# RNN: Computational Graph: Many to Many

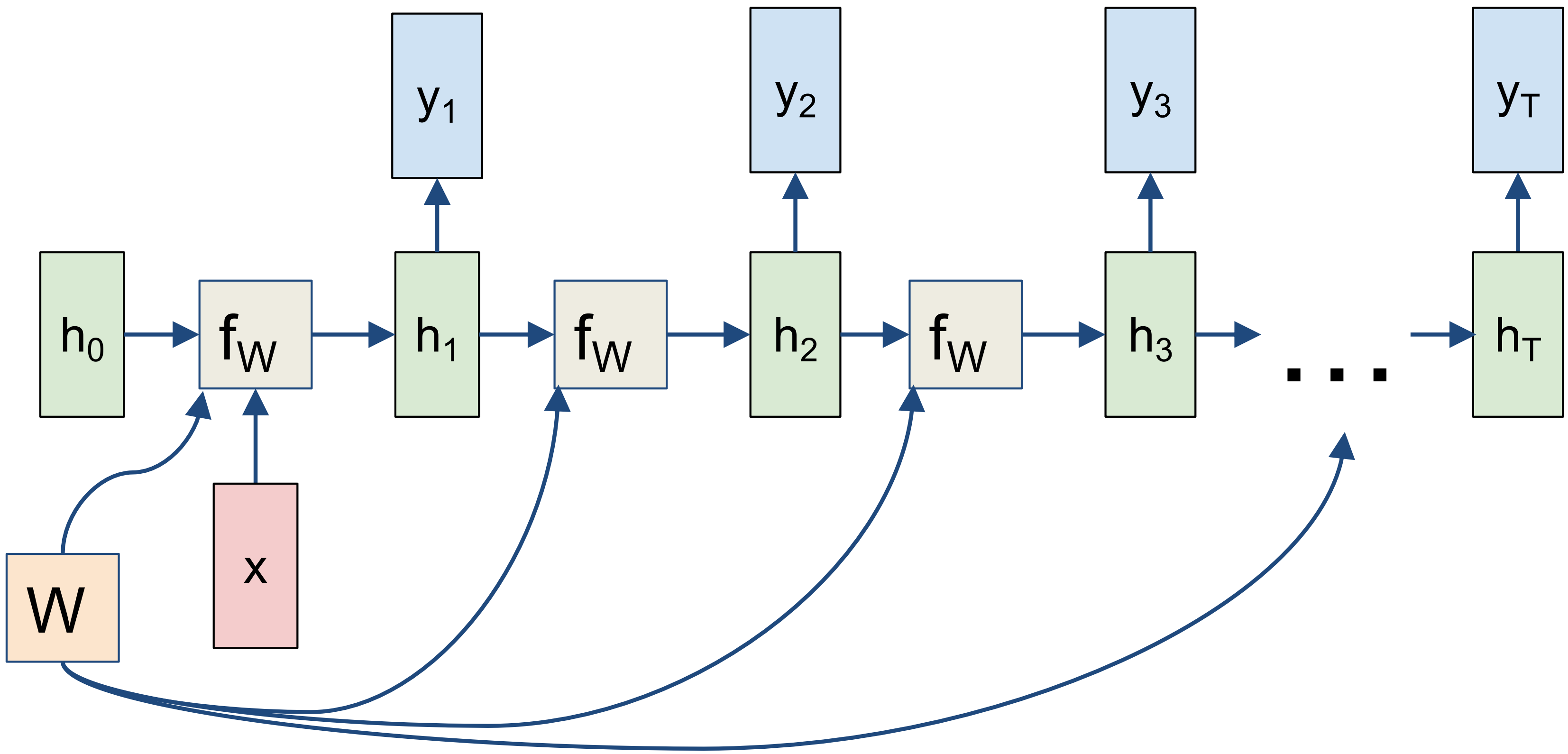



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# RNN: Computational Graph: Many to One

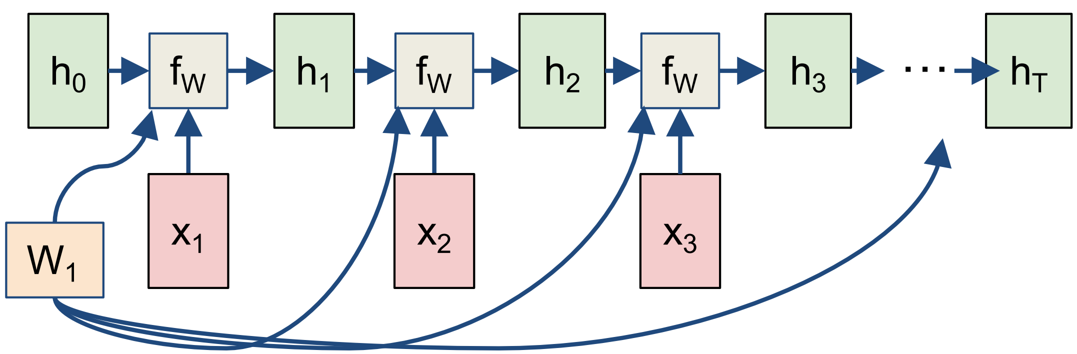


# RNN: Computational Graph: One to Many



# Sequence to Sequence: Many-to-one + one-to-many

**Many to one**: Encode input sequence in a single vector



# Sequence to Sequence: Many-to-one + one-to-many

**Many to one**: Encode input sequence in a single vector

**One to many**: Produce output sequence from single input vector