CS 4803 / 7643: Deep Learning

Topics:

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

Dhruv Batra

Georgia Tech

Administrativia

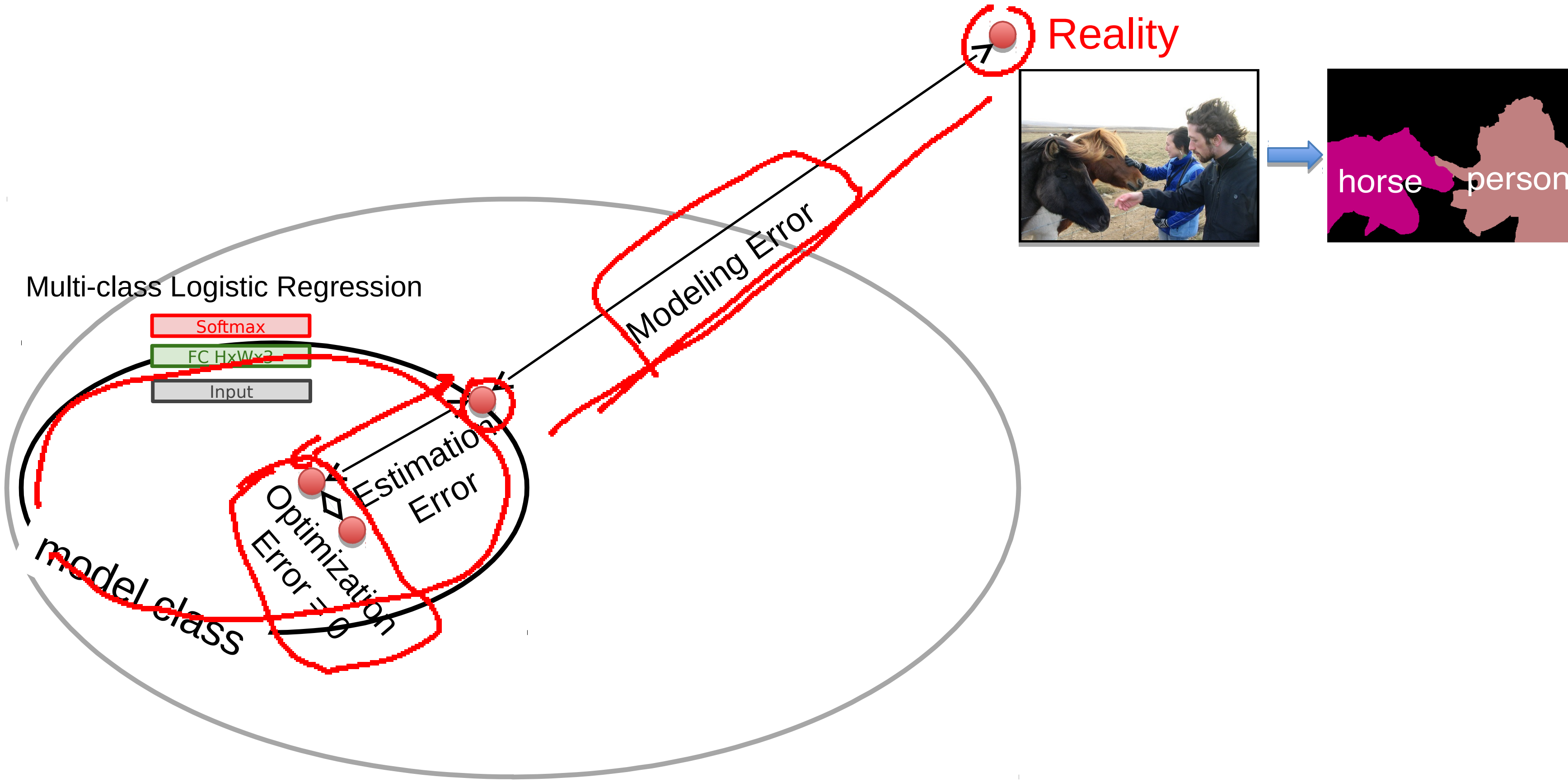
- 5 min talk by Vadini Agrawal (of CS + Social Good)
  - Talk on Ethical considerations within deep learning

Administrativia

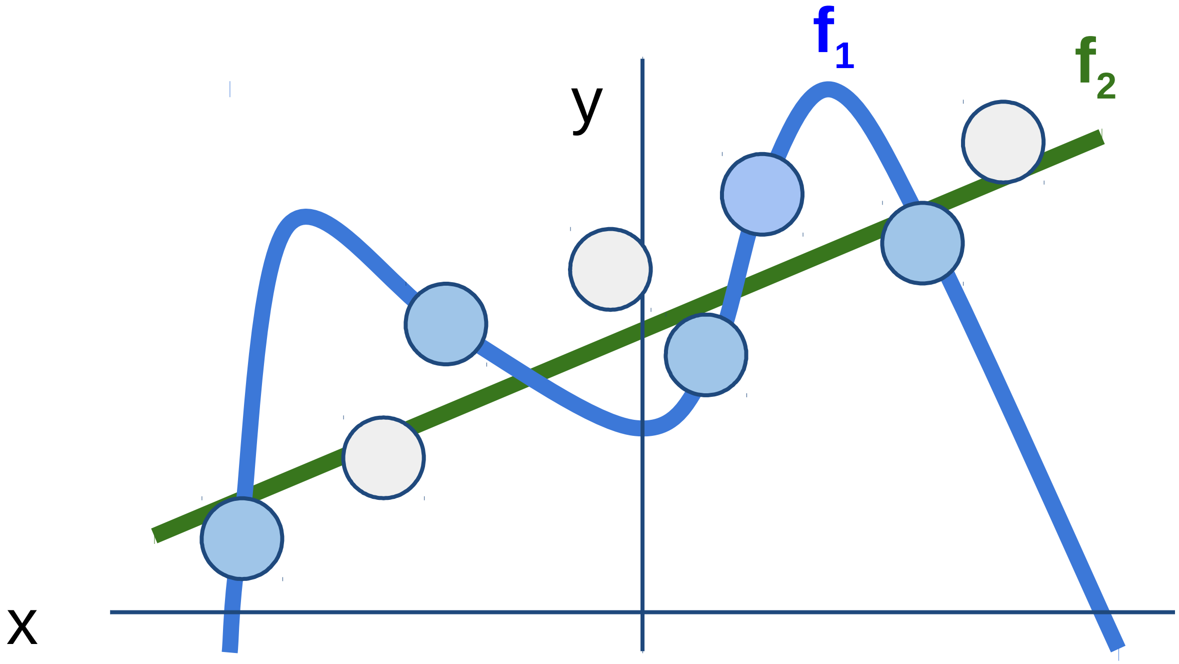
- HW3 Reminder
  - Due: 10/07 11:59pm
  - Theory: Convolutions, Representation Capacity, Double Descent
  - Implementation: Saliency methods (e.g. Grad-CAM) in Python and PyTorch/Captum

Thoughts on Zhang et al. ICLR17

Error Decomposition



Regularization: Prefer Simpler Models



Regularization pushes against fitting the data *too* well so we don't fit noise in the data

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

Thoughts on Zhang et al. ICLR17

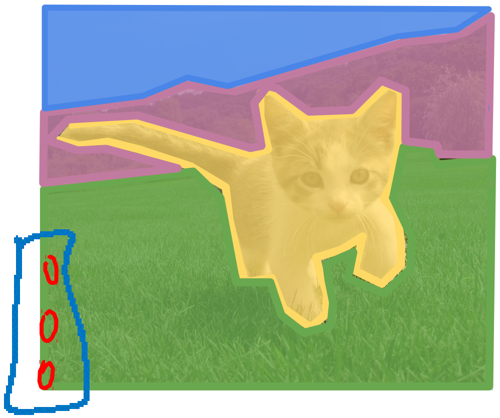
- Randomization testing is a powerful tool
- What does 0 training error on random labels mean?
  - No optimization error
  - No approximation/modeling error
- Explicit regularization is helpful, but not essential
- Inductive bias
  - Conv is a specific inductive bias, but even when data doesn't satisfy that, the model class is expressive enough
- Implicit regularization of SGD
  - See HW3 Q6
- These results are not specific to deep learning / NN
  - Also known to happen for decision trees

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

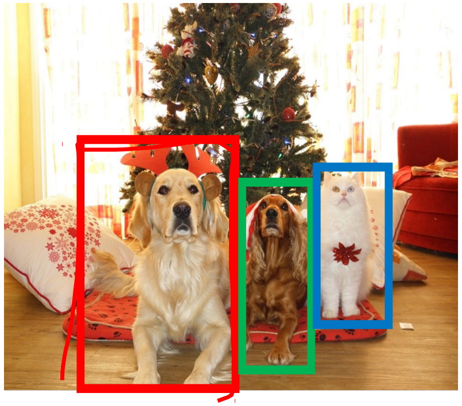
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

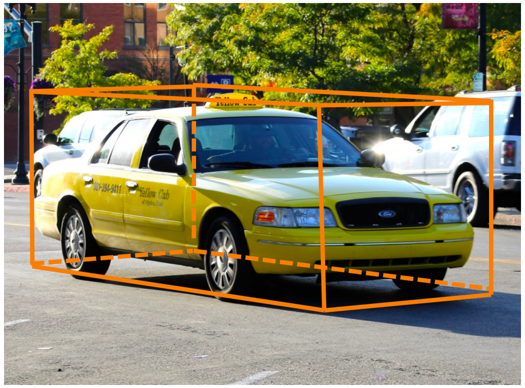
2D Object Detection



DOG, DOG, CAT

Object categories + 2D bounding boxes

3D Object Detection



Car

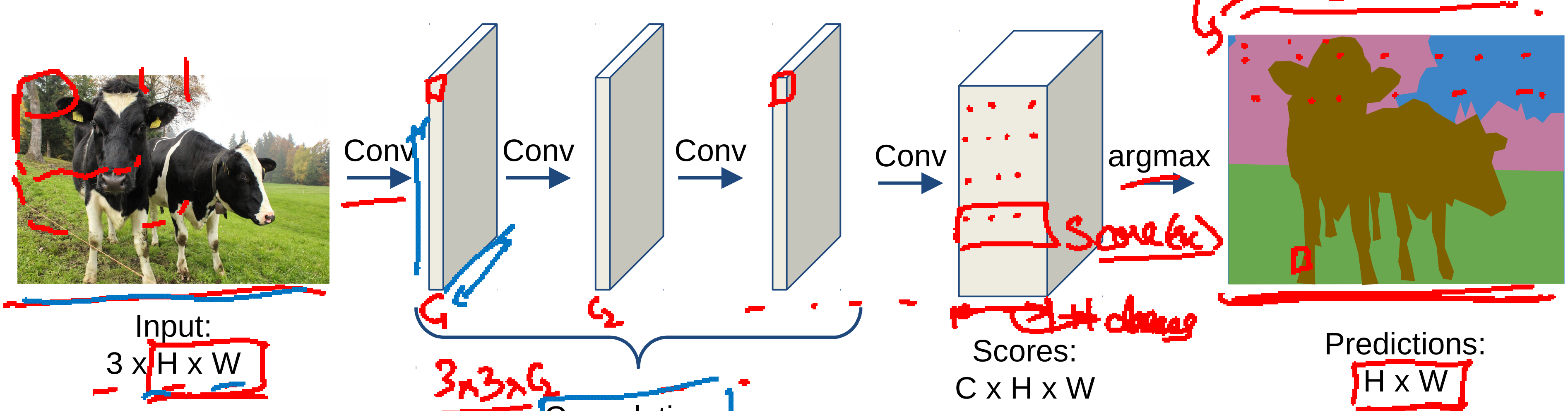
Object categories + 3D bounding boxes

This image is CC0 public domain

Semantic Segmentation Idea: Fully Convolutional

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

y^{gt} $\hat{y}$

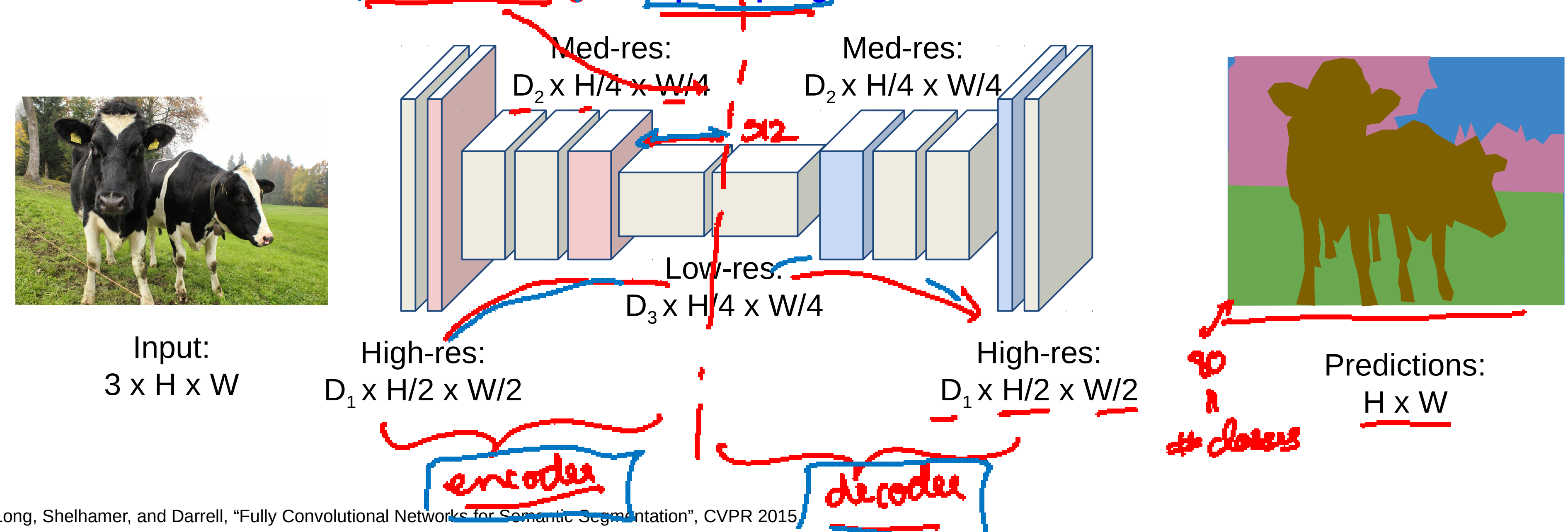


Problem: convolutions at original image resolution will be very expensive ...

$$loss = -\sum_{x,c} \log \hat{p}(y^{gt}_{x,c})$$

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

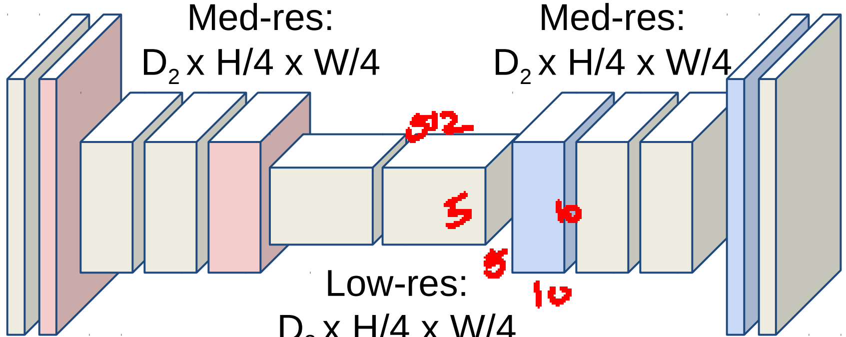
Semantic Segmentation Idea: Fully Convolutional

Downsampling: Pooling, strided convolution

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

Upsampling: ???

Input:
3 x H x W



Med-res:
D_2 x H/4 x W/4

Med-res:
D_2 x H/4 x W/4

Low-res:
D_3 x H/4 x W/4

High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x 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”

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

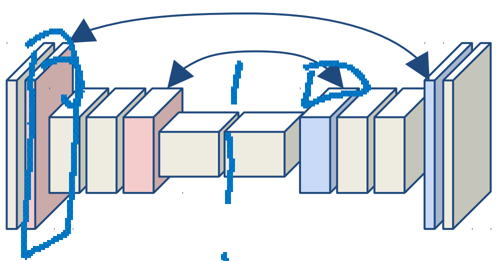
1	2
3	4

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

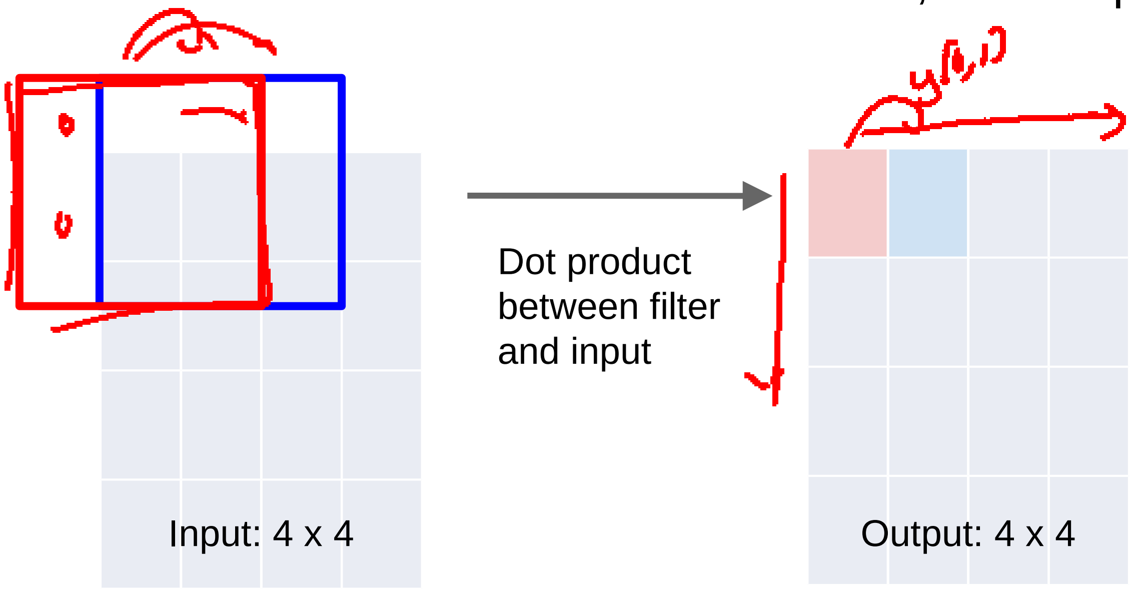


Transposed Convolutions

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

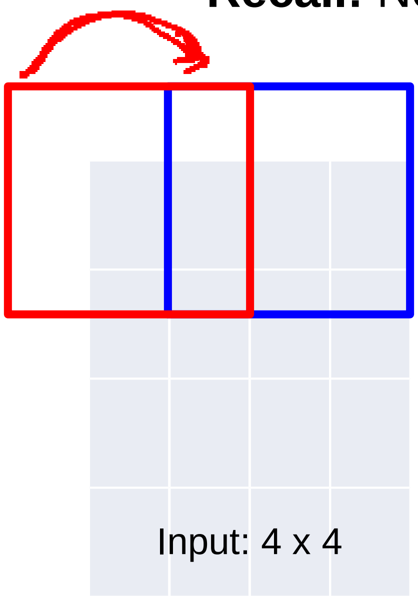
Learnable Upsampling: Transpose Convolution

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

Dot product between filter and input

Output: 2 x 2

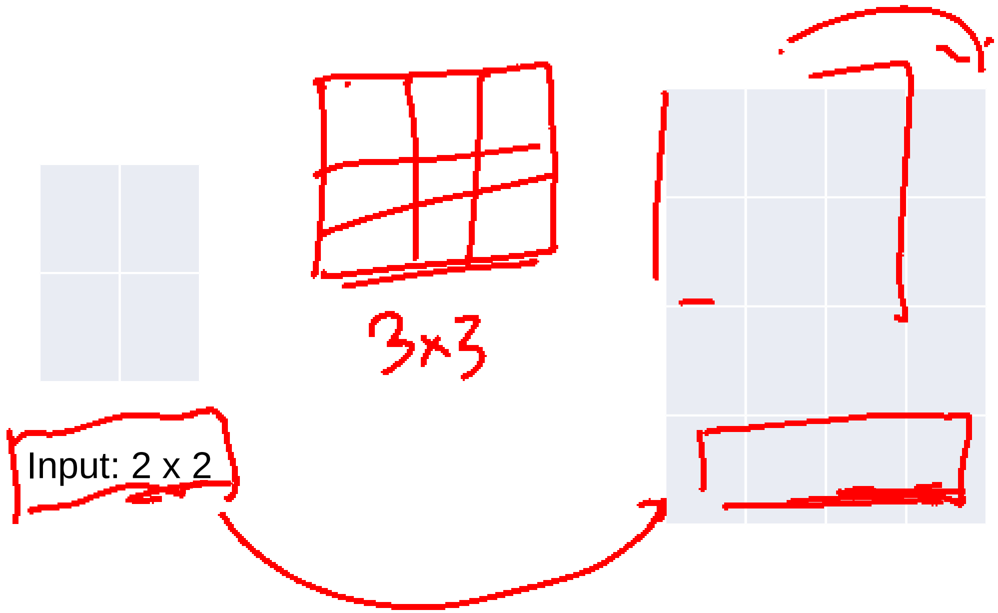
Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

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

Learnable Upsampling: Transpose Convolution

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



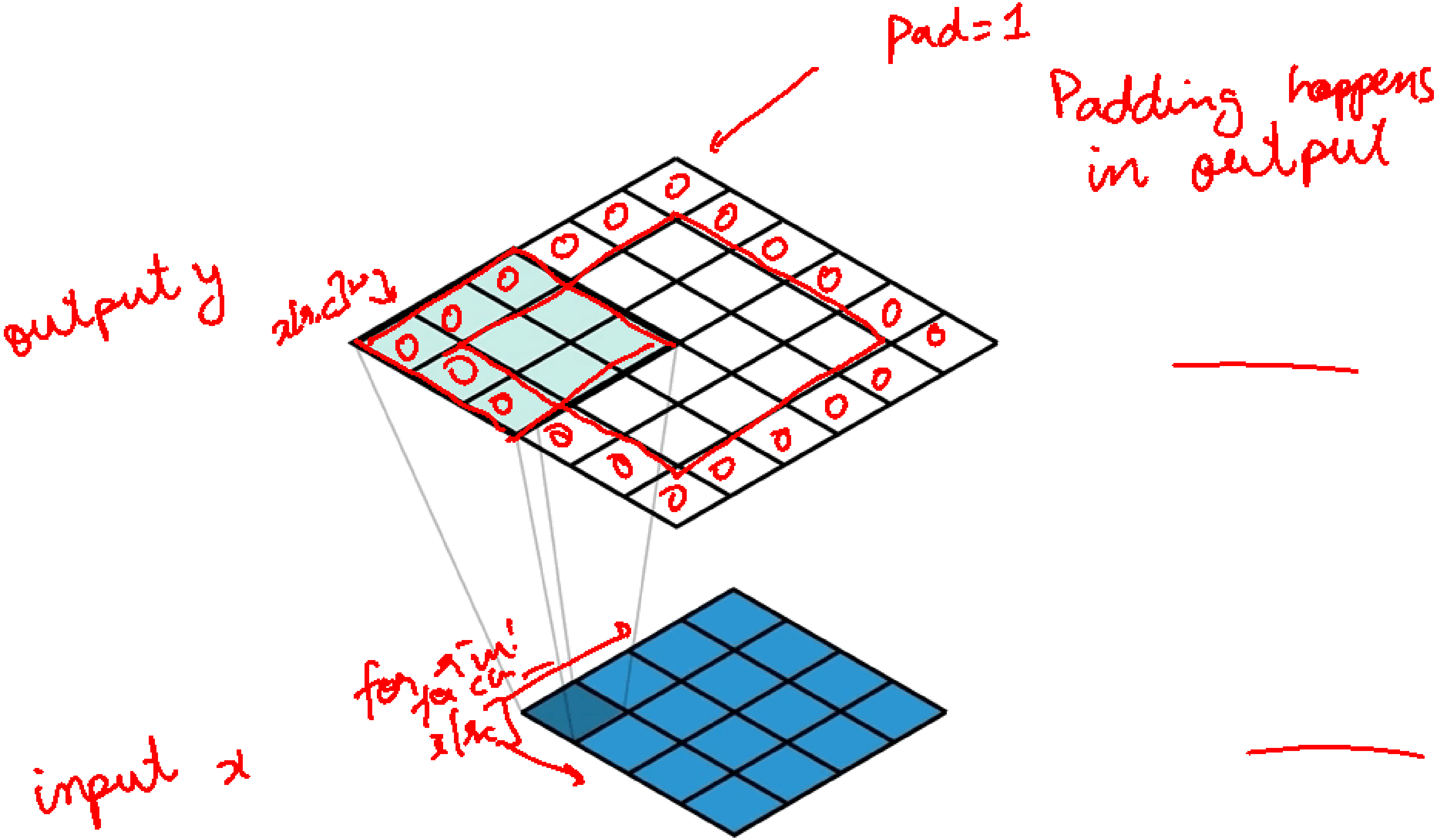
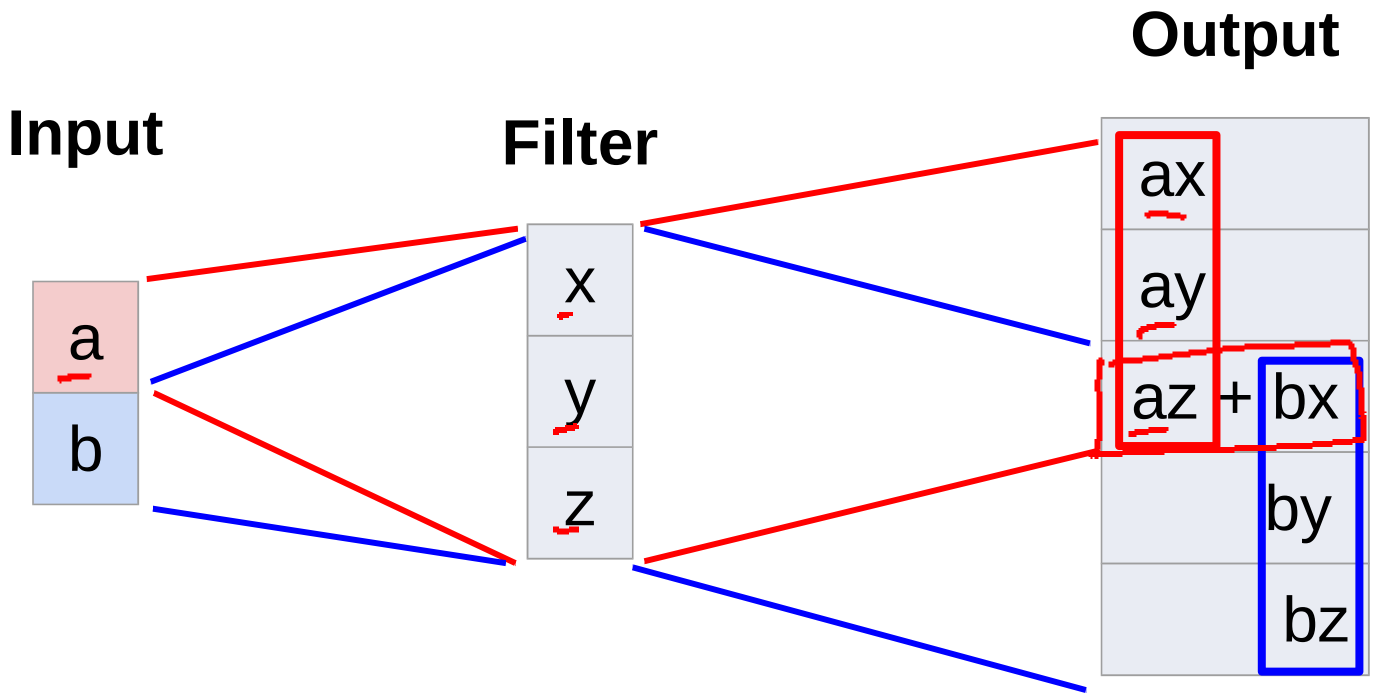


Figure Credit: https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

Transpose Convolution: 1D Example

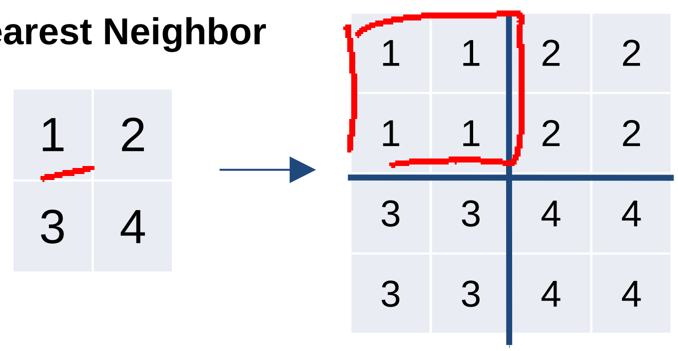


Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

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

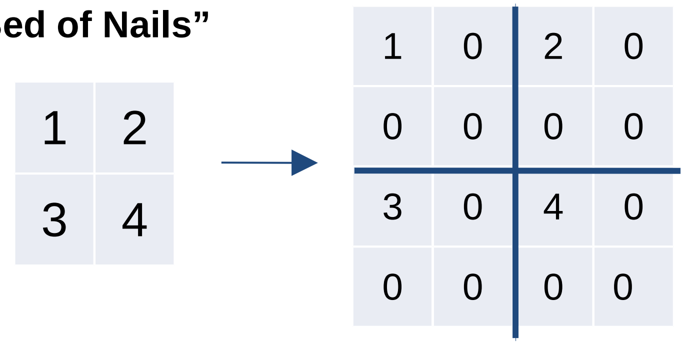
In-Network upsampling: “Unpooling”

Nearest Neighbor



Input: 2 x 2 Output: 4 x 4

“Bed of Nails”



Input: 2 x 2 Output: 4 x 4

Why this operation?

Why is it called “transposed convolution”?

Toeplitz Matrix

- Diagonals are constants

$$\begin{bmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{bmatrix}.$$

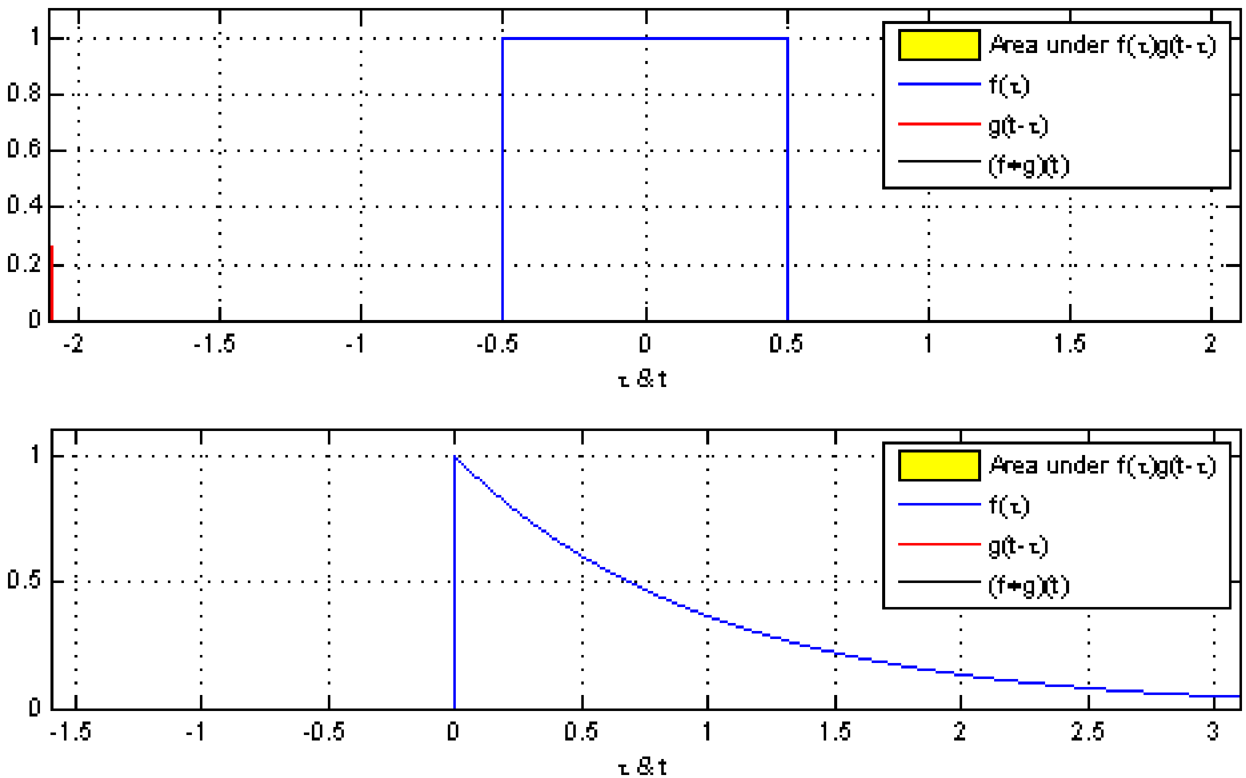
- $\boxed{A_{ij}} = \boxed{a_{i-j}}$

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{bmatrix}$$

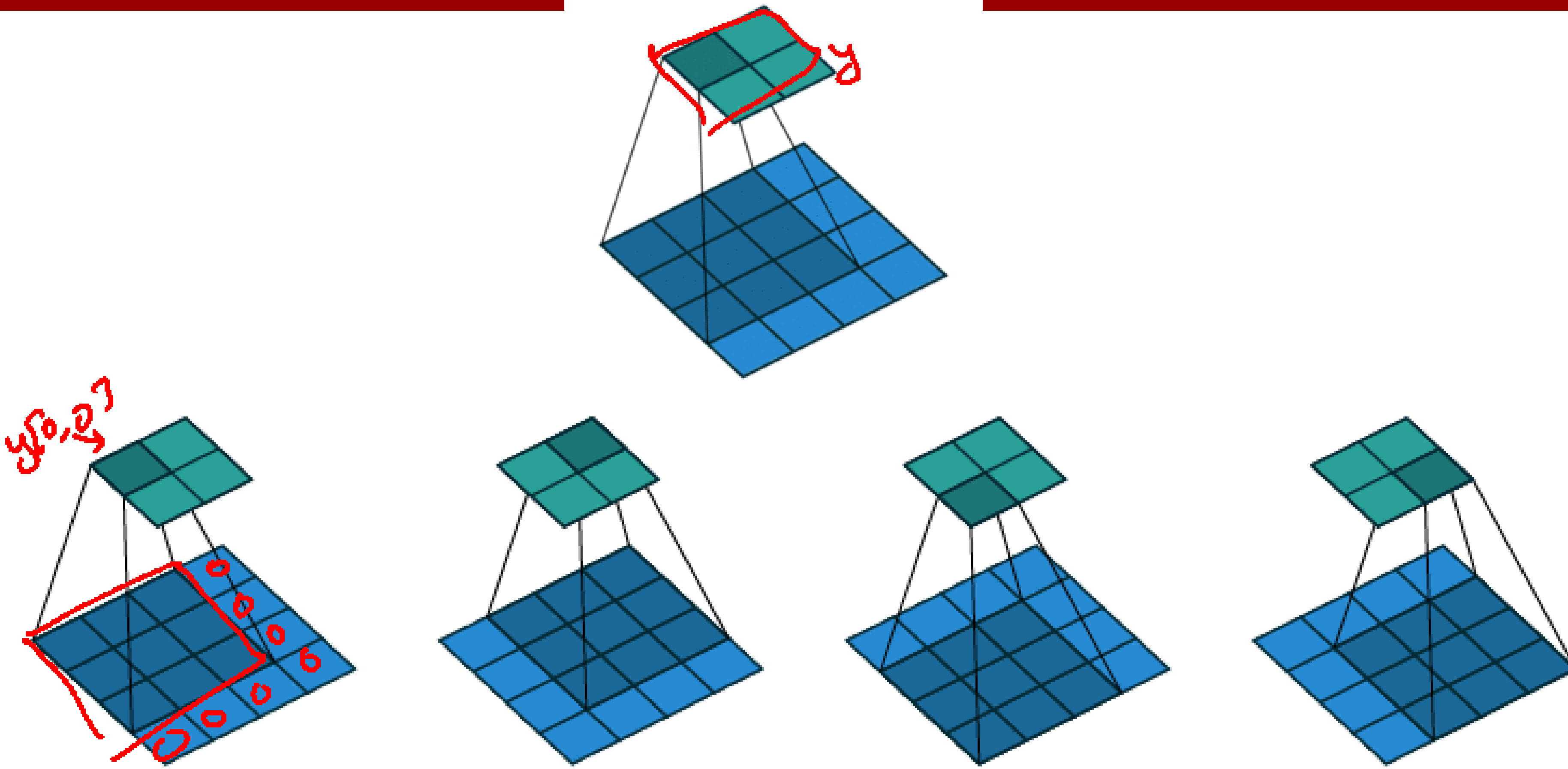
Why do we care?

- (Discrete) Convolution = Matrix Multiplication
  - with Toeplitz Matrices

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



"Convolution of box signal with itself2" by Convolution_of_box_signal_with_itself.gif: Brian Ambergderivative work: Tinos (talk) - Convolution_of_box_signal_with_itself.gif. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Convolution_of_box_signal_with_itself2.gif#/media/File:Convolution_of_box_signal_with_itself2.gif



y-vec(:)

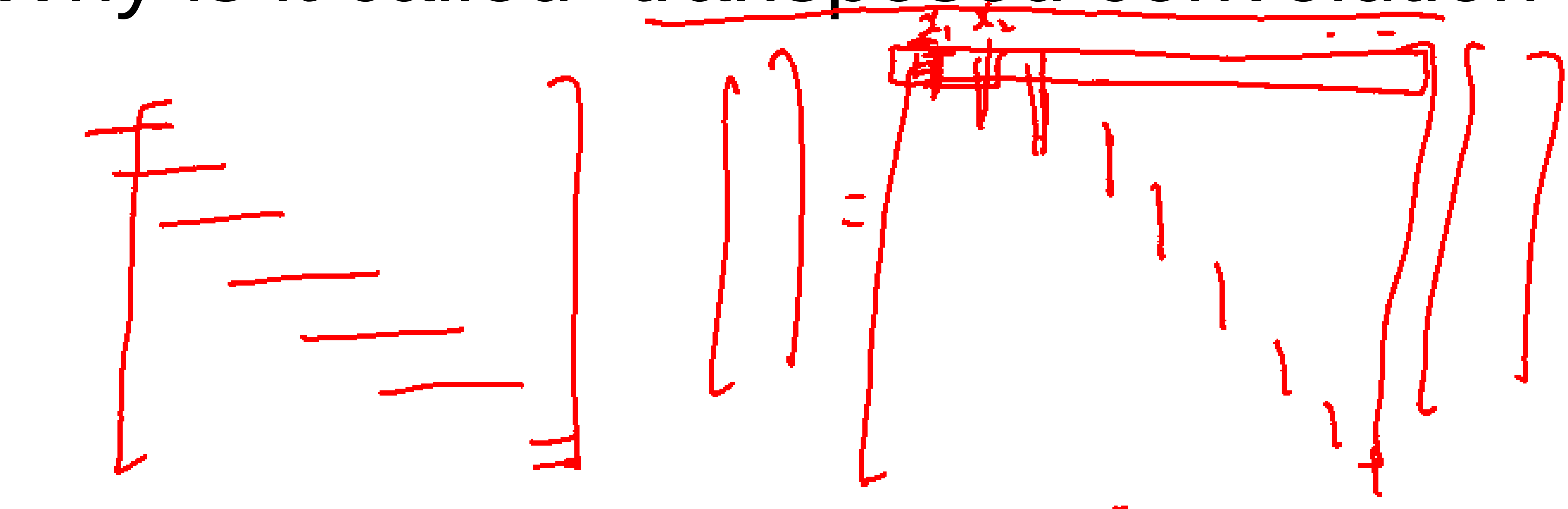
$$
\begin{bmatrix} y[0,0] \\ \vdots \end{bmatrix} =
\begin{pmatrix}
w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\
0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\
0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2}
\end{pmatrix}
\begin{bmatrix} x[0,0] \\ x[0,1] \\ \vdots \end{bmatrix}
$$

x-vec(:)

y-vec = W x-vec

Figure Credit: Dumoulin and Visin, https://arxiv.org/pdf/1603.07285.pdf

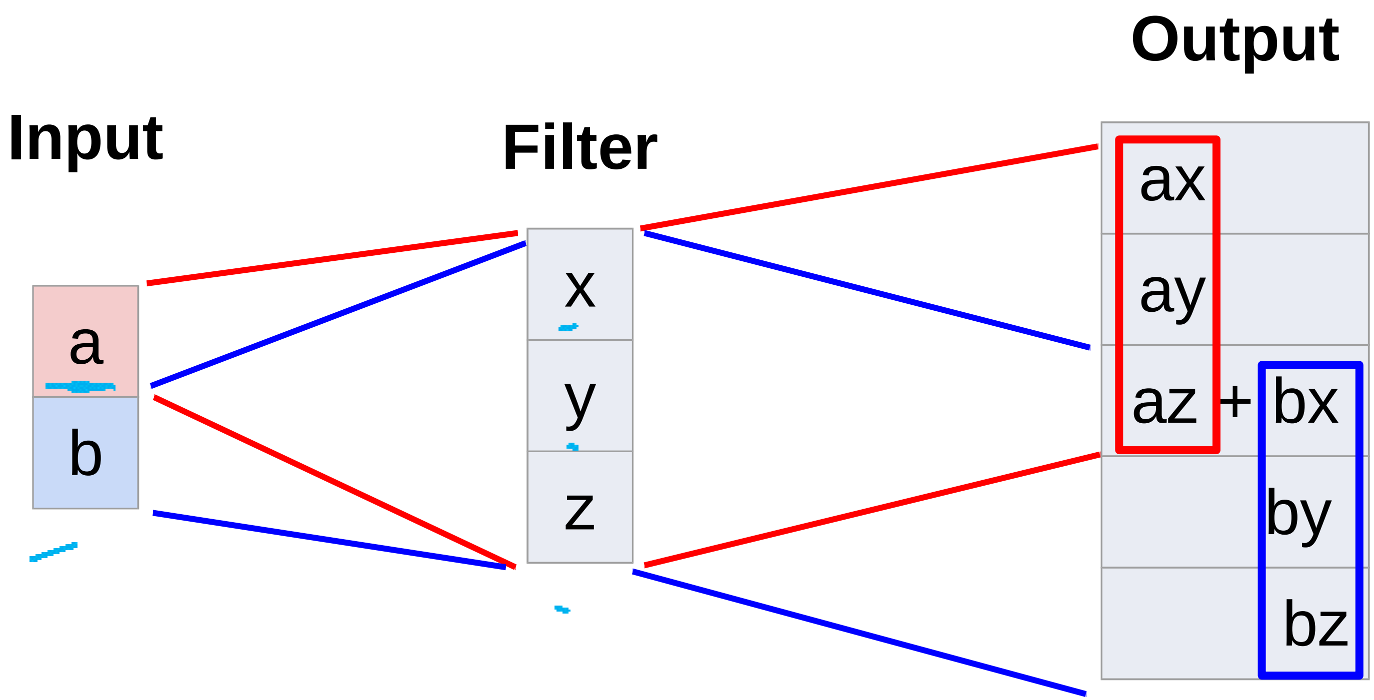
Why is it called “transposed convolution”



$y = W x$ $\qquad$ $W^T x$

$\underbrace{\qquad\qquad\qquad}_{\text{Transposed Conv}}$

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

What is deconvolution?

- (Non-blind) Deconvolution

Conv $y = x * w$

Deconv

→ Blind: Given y, produce estimate w, x

→ Non-blind: Given y, w, find/estimate x

What is deconvolution?

Assume: W is orthonormal/orthogonal

$y = Wx \Rightarrow x = W^T y$

- (Non-blind) Deconvolution

$\tilde{w} = [-1 \; 0 \; +1]\frac{1}{\sqrt{2}}$

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

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

“transposed convolution” $\boxed{\text{is a}}$ convolution!

with filter w

with a different filter

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:

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

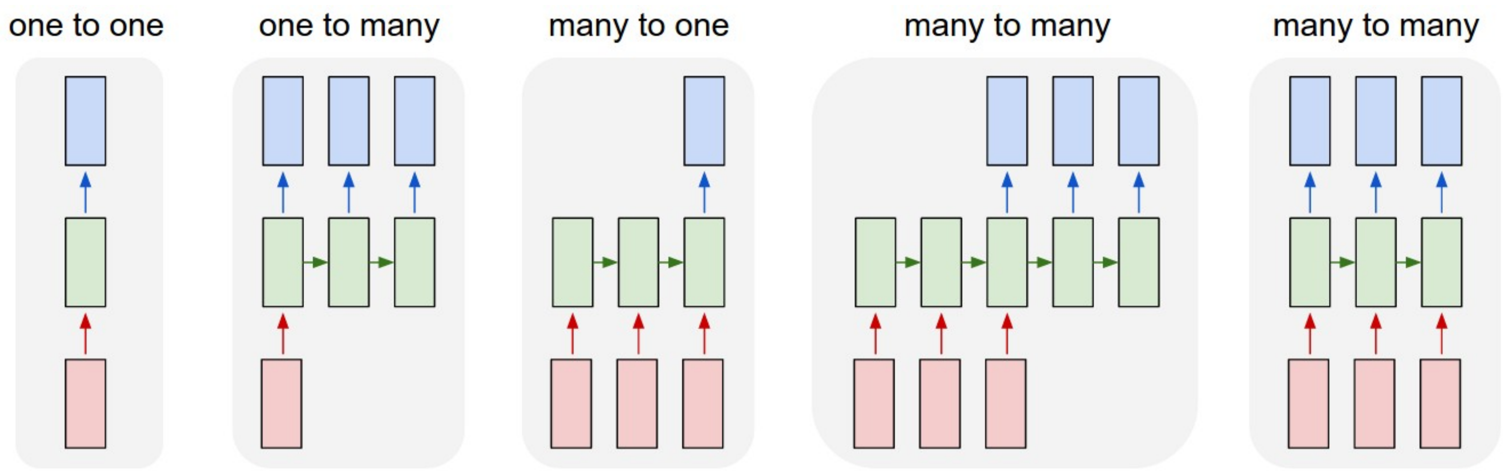
$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

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

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



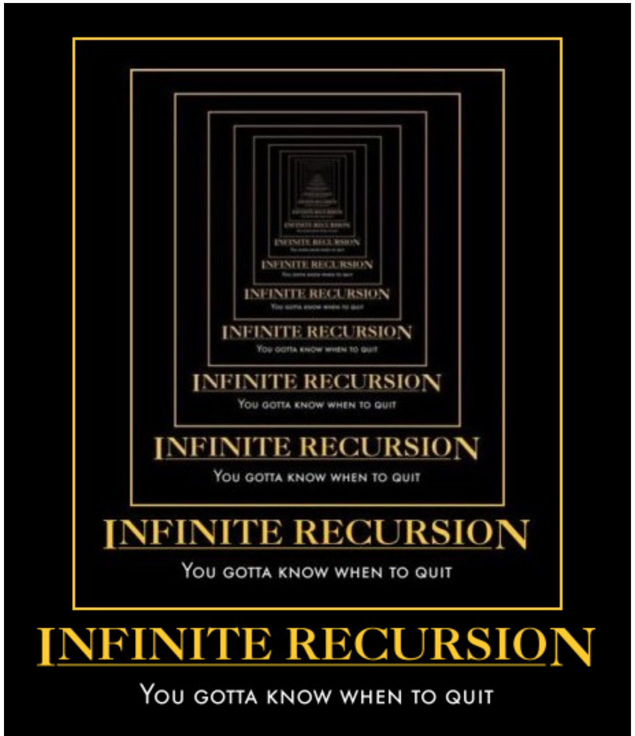
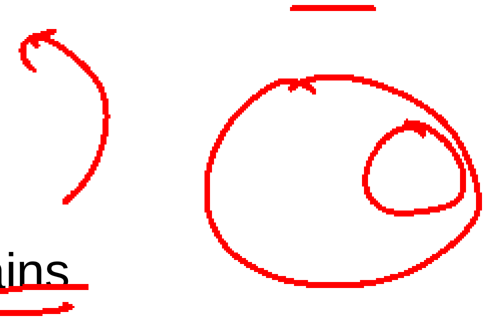


Image Credit: Andrej Karpathy

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

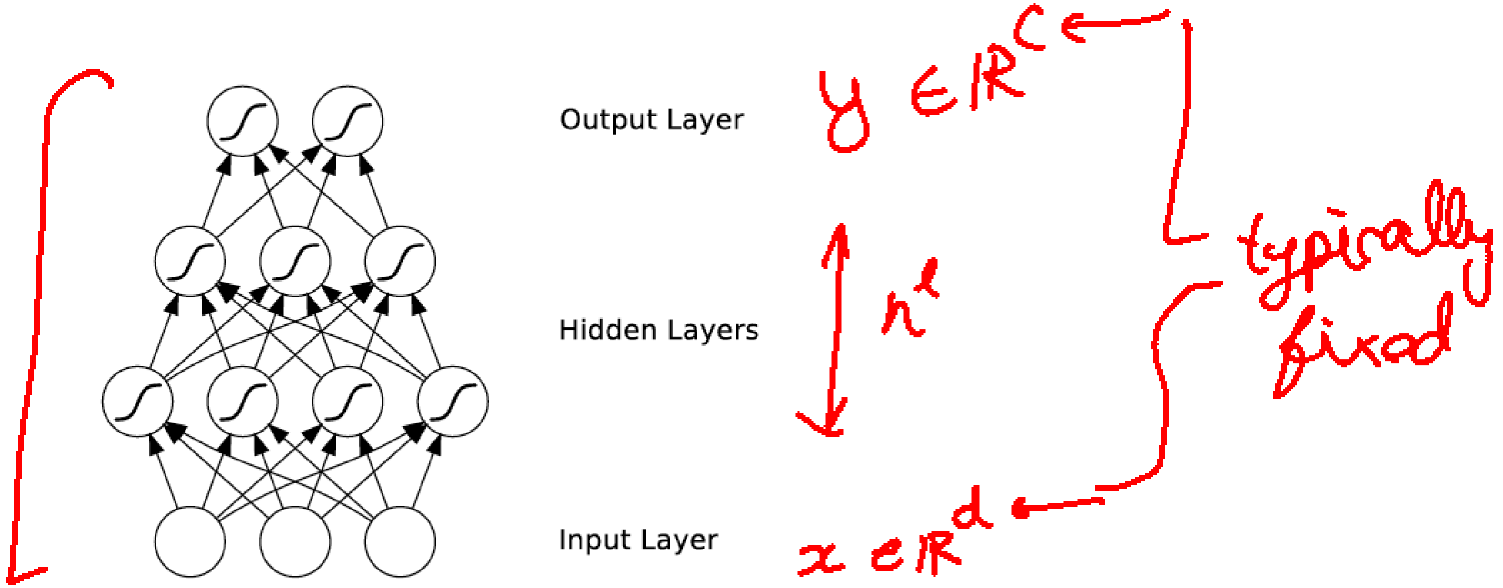


Image Credit: Alex Graves, book

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

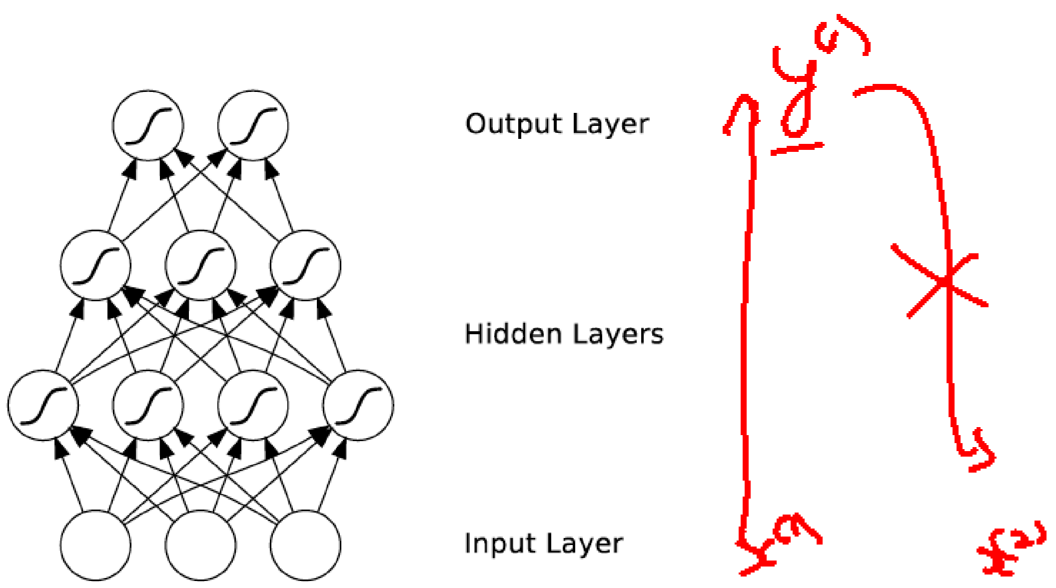


Image Credit: Alex Graves, book

Why model sequences?

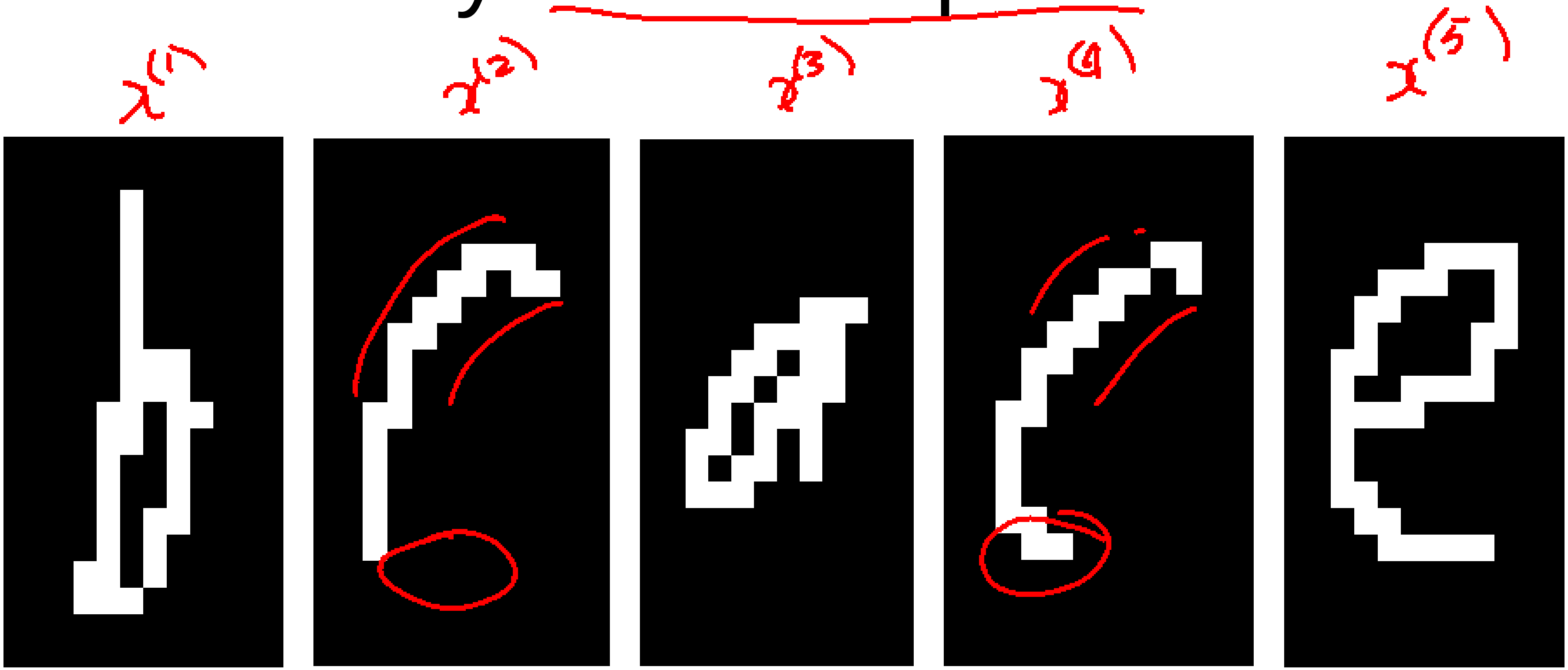


Figure Credit: Carlos Guestrin

Why model sequences?

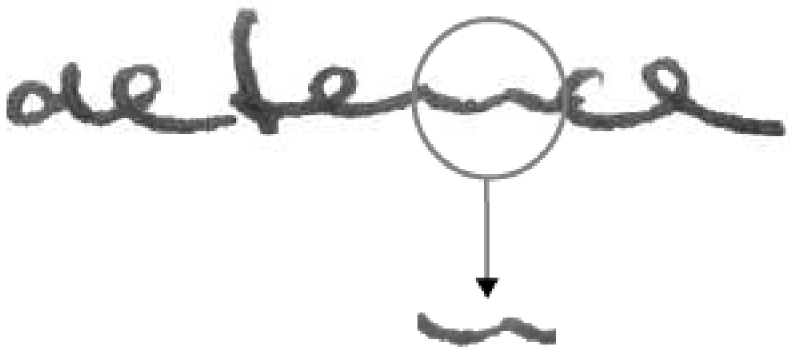
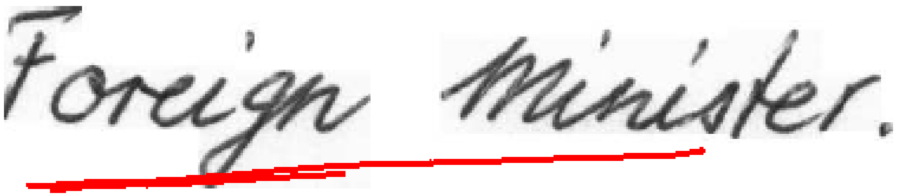
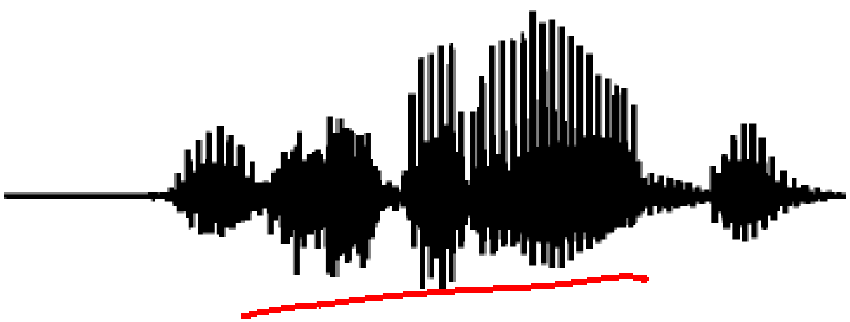


Image Credit: Alex Graves

Sequences are everywhere…



FOREIGN MINISTER.



THE SOUND OF

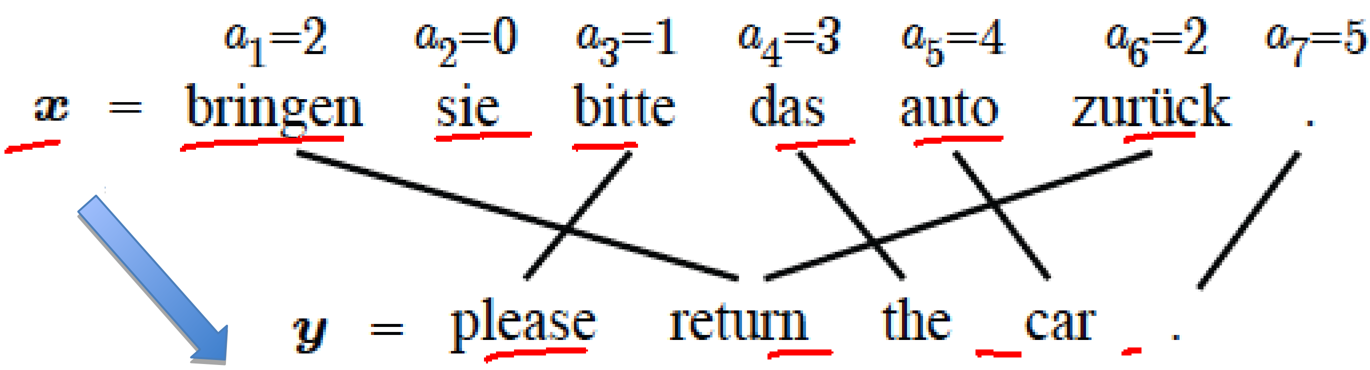


Image Credit: Alex Graves and Kevin Gimpel

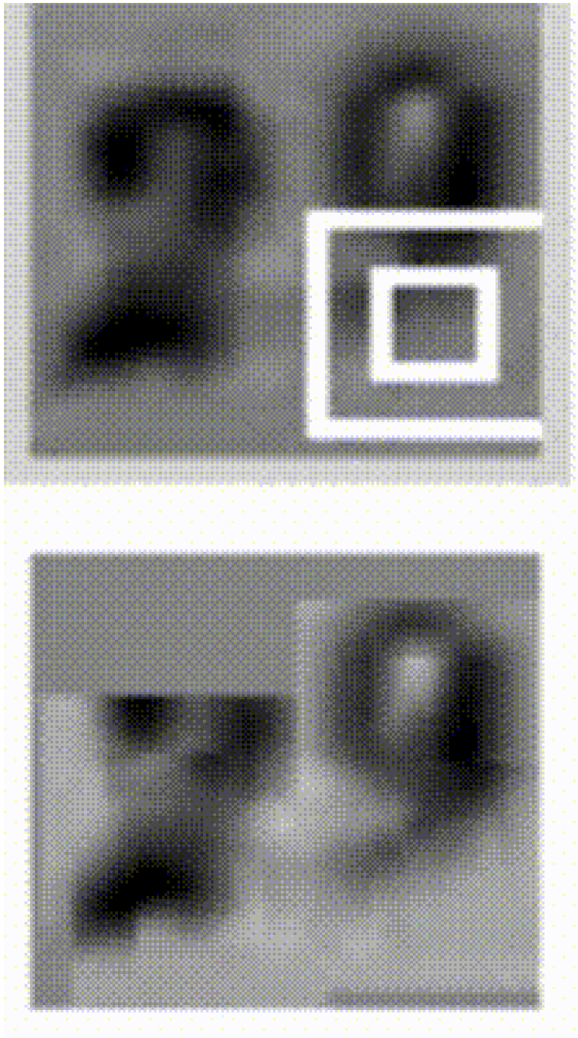
Even where you might not expect a sequence…

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
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

Even where you might not expect a sequence…

- Output ordering = sequence



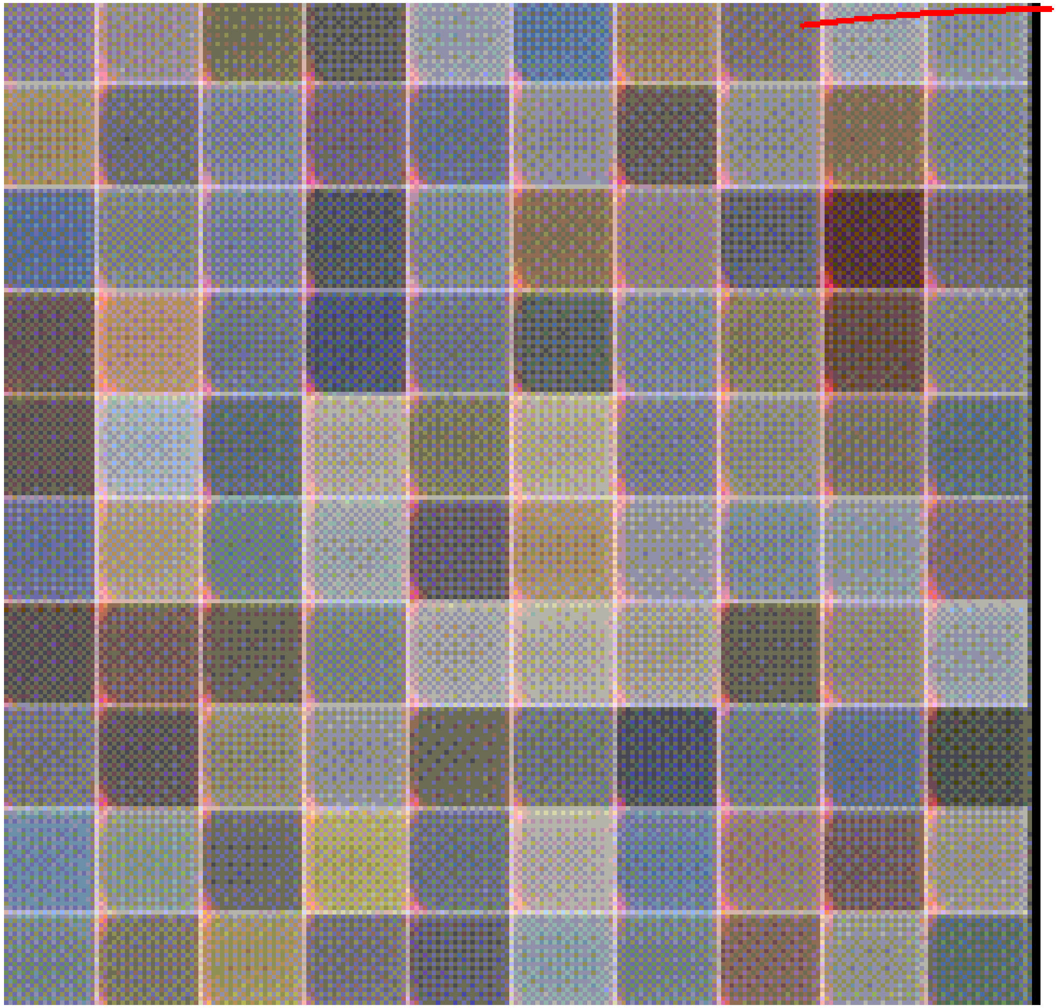
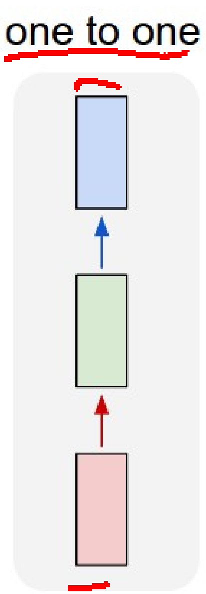


Image Credit: Ba et al.; Gregor et al

Sequences in Input or Output?

- It's a spectrum…



Input: No sequence

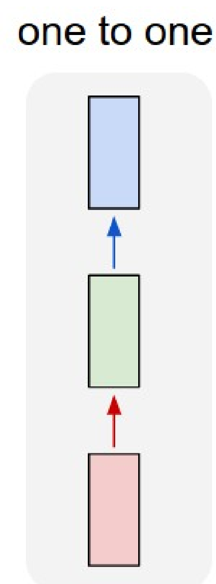
Output: No sequence

Example: "standard" classification / regression problems

Image Credit: Andrej Karpathy

Sequences in Input or Output?

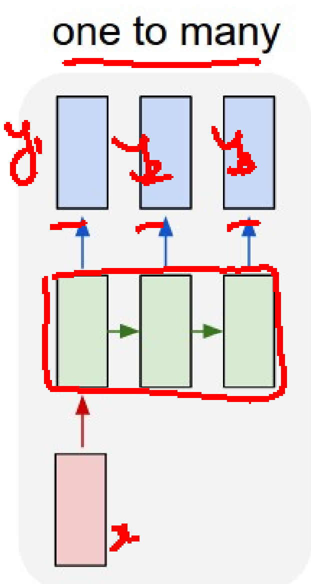
- It's a spectrum…



Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems



Input: No sequence

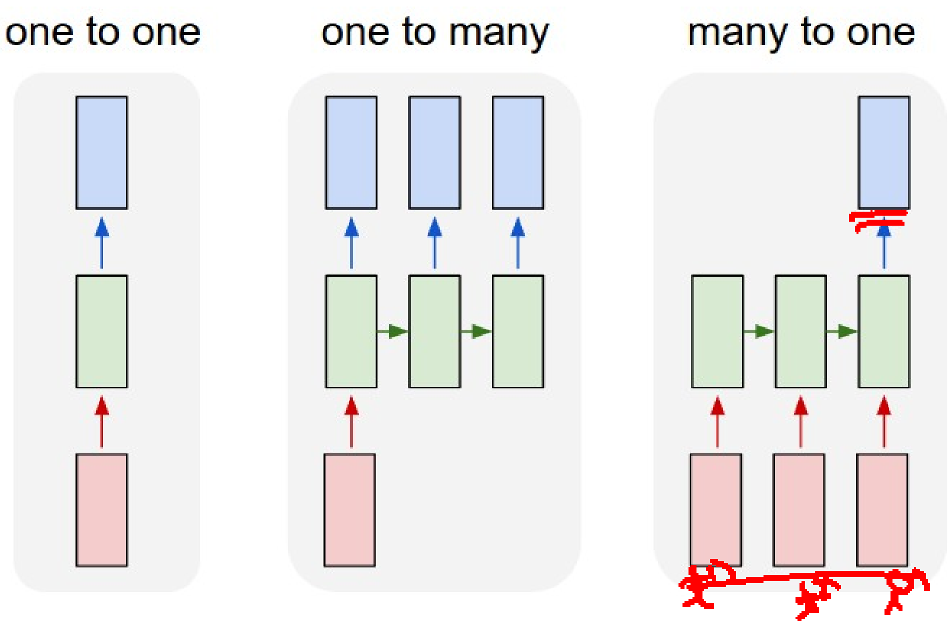
Output: Sequence

Example: Im2Caption

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

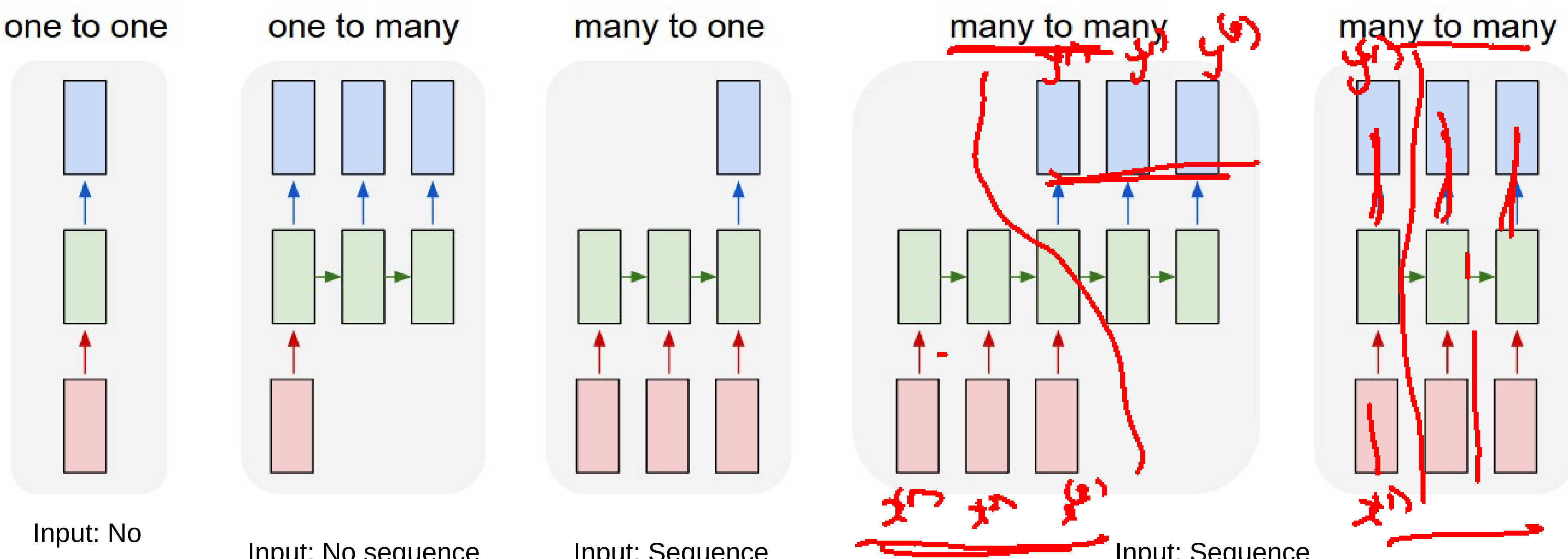
Input: Sequence

Output: No sequence

Example: sentence classification, multiple-choice question answering

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

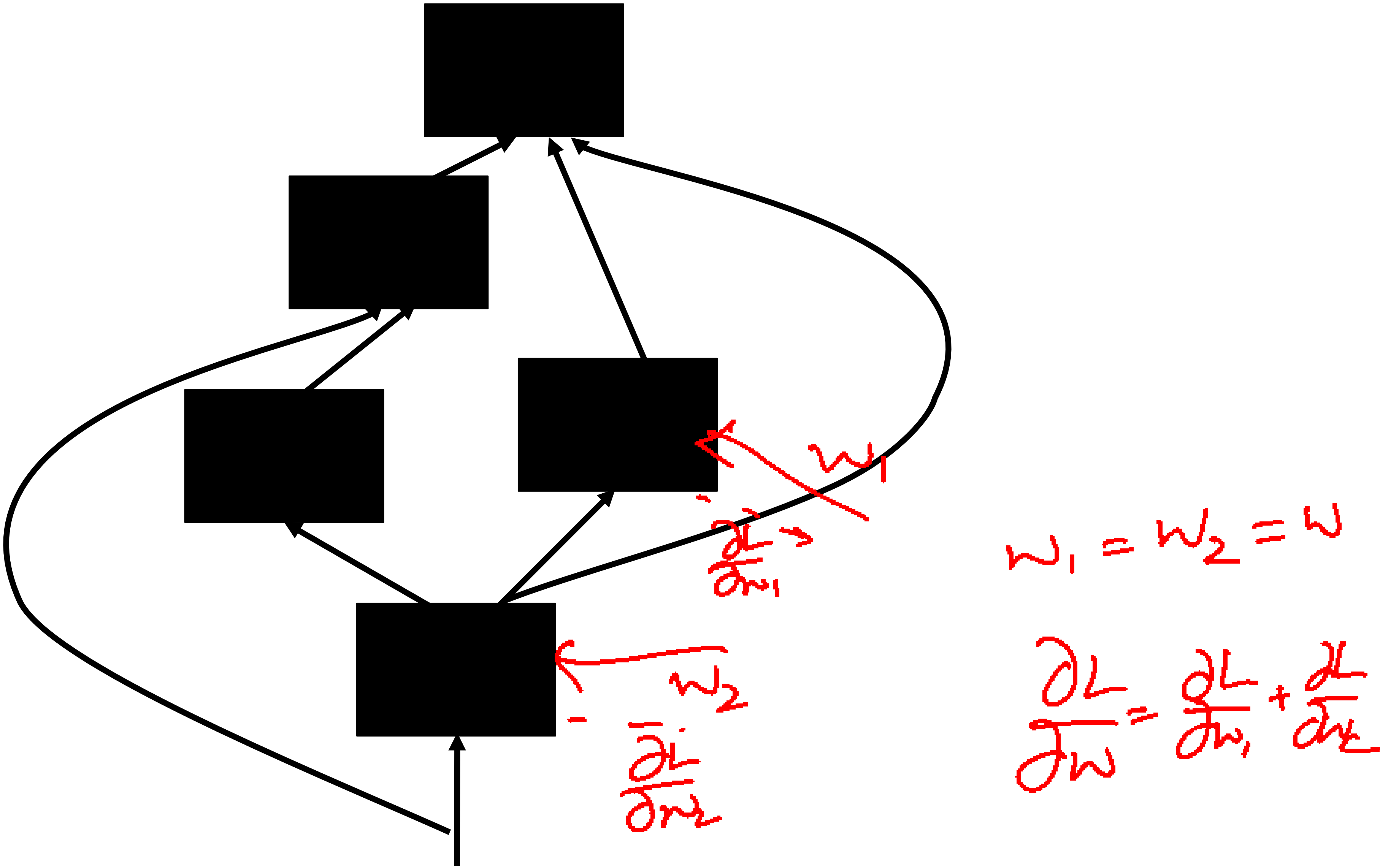
Output: Sequence

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

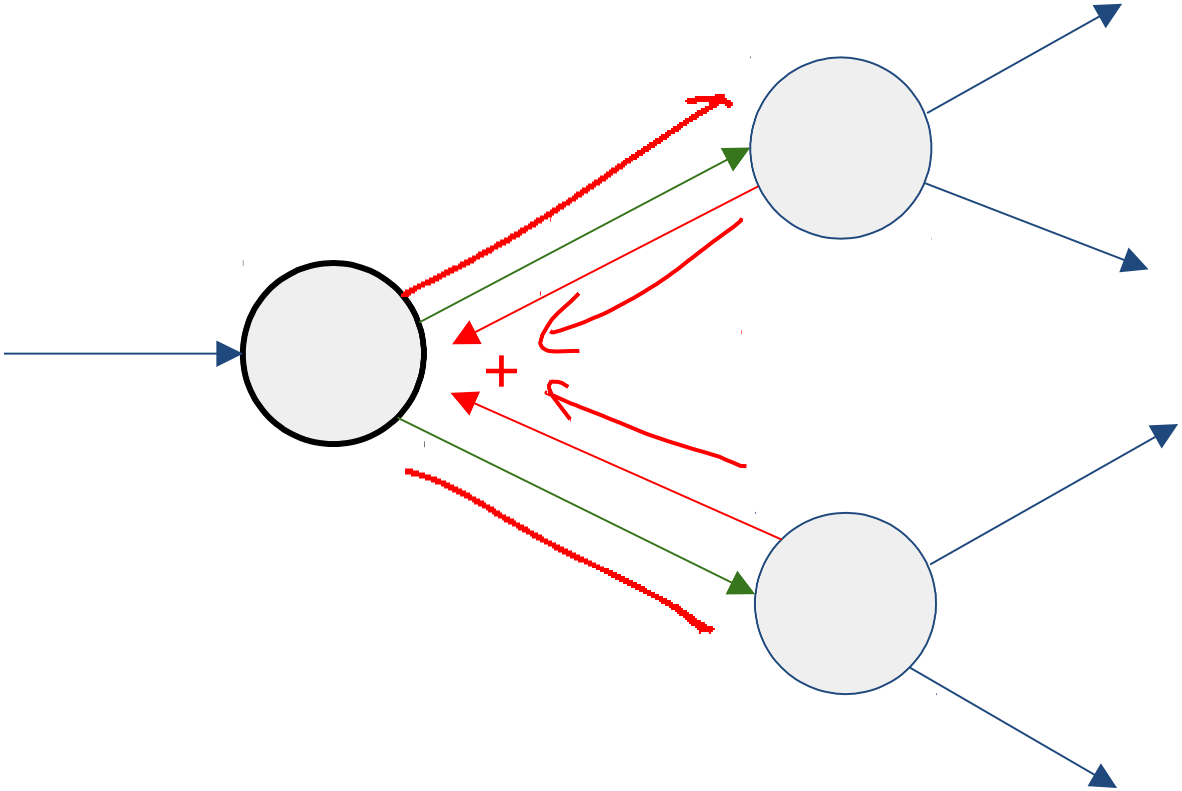
2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients

Computational Graph



Gradients add at branches



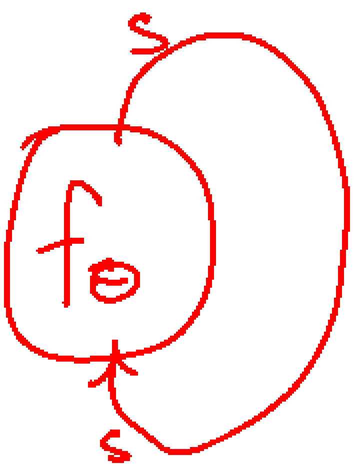
2 Key Ideas

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

How do we model sequences?

- No input

$$\mathbf{s}_{\boxed{t}} = f_\theta(\mathbf{s}_{t-1})$$



"unrolling"

$s_0 \xrightarrow{s_0} f_\theta \xrightarrow{s_1} f_\theta \xrightarrow{s_2} f_\theta$ - - - - - -

No loops

Image Credit: Bengio, Goodfellow, Courville

How do we model sequences?

- With inputs

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

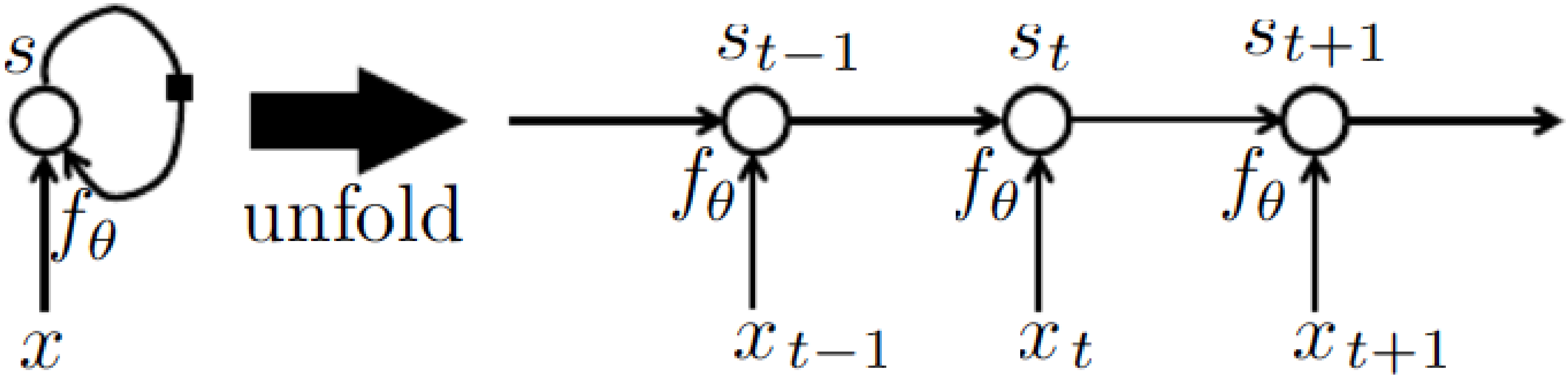


Image Credit: Bengio, Goodfellow, Courville

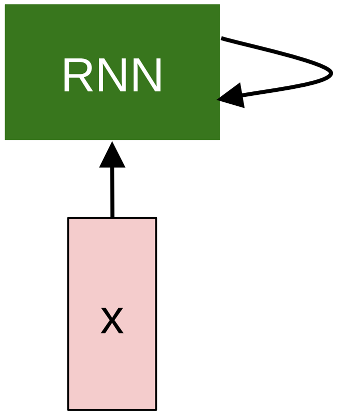
2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients

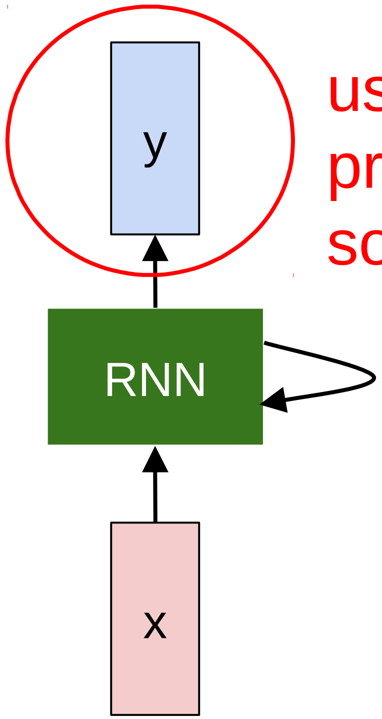
- “Unrolling”
  - in computation graphs with parameter sharing

- Parameter sharing + Unrolling
  - Allows modeling arbitrary sequence lengths!
  - Keeps numbers of parameters in check

Recurrent Neural Network



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:

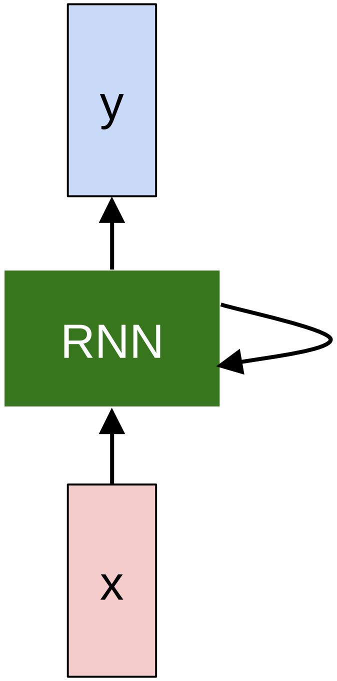
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step

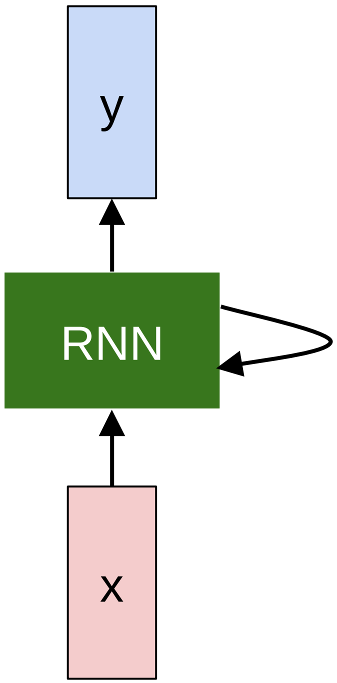


Recurrent Neural Network

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

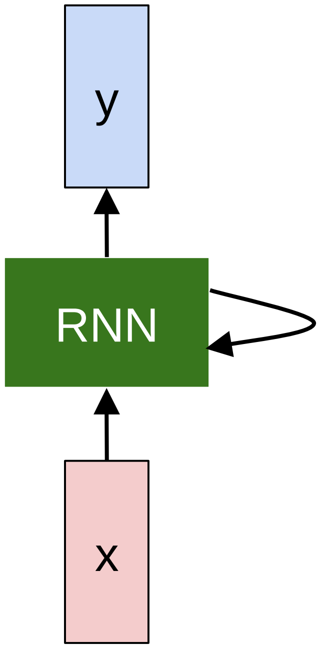
$$h_t = f_W(h_{t-1}, x_t)$$

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



(Vanilla) Recurrent Neural Network

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



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

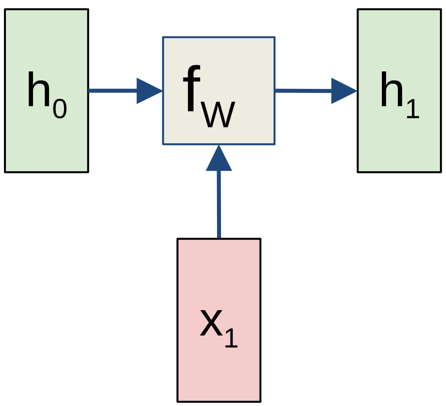
$$h_t = f_W(h_{t-1}, x_t)$$

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

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

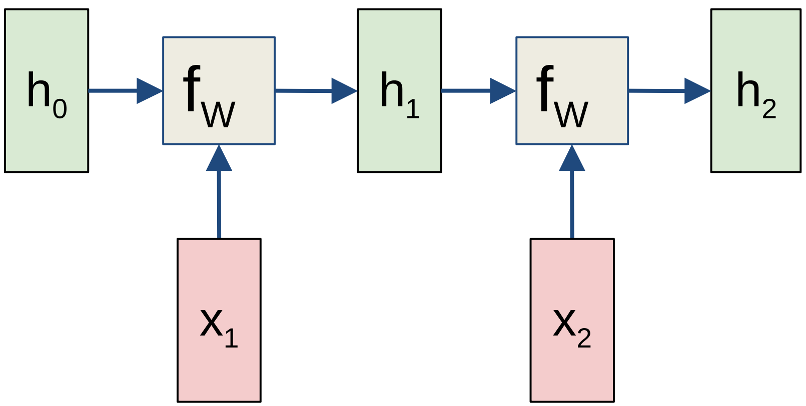
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph



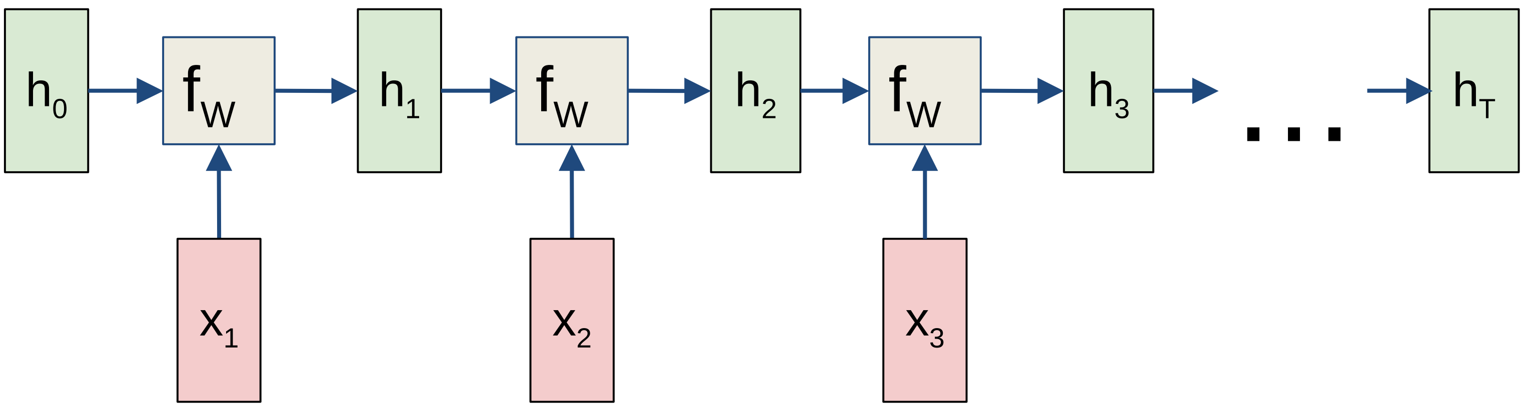
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph



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

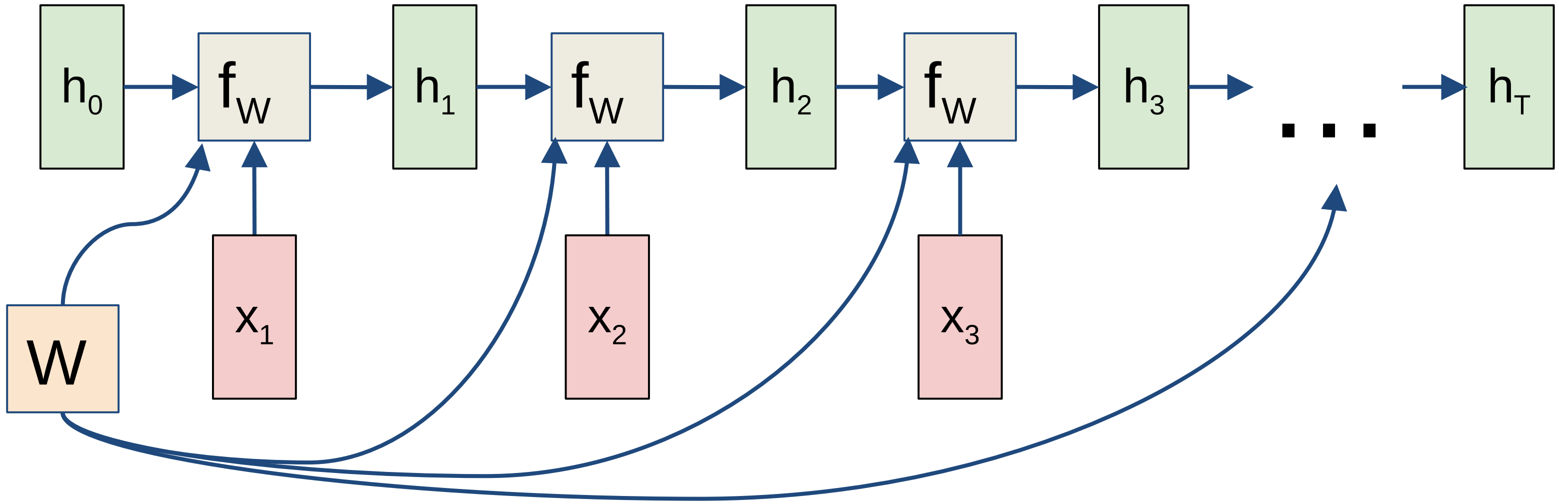
RNN: Computational Graph



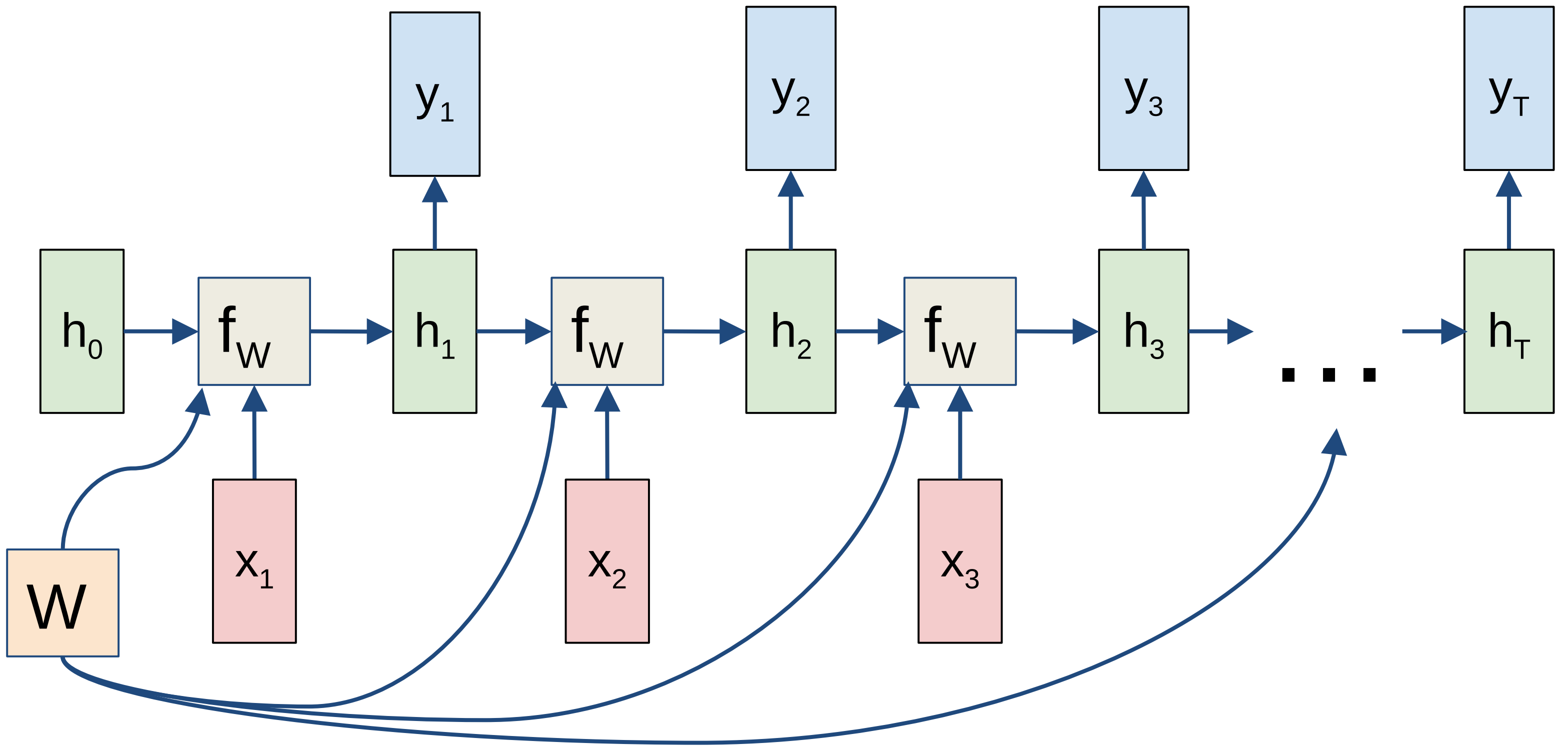
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph

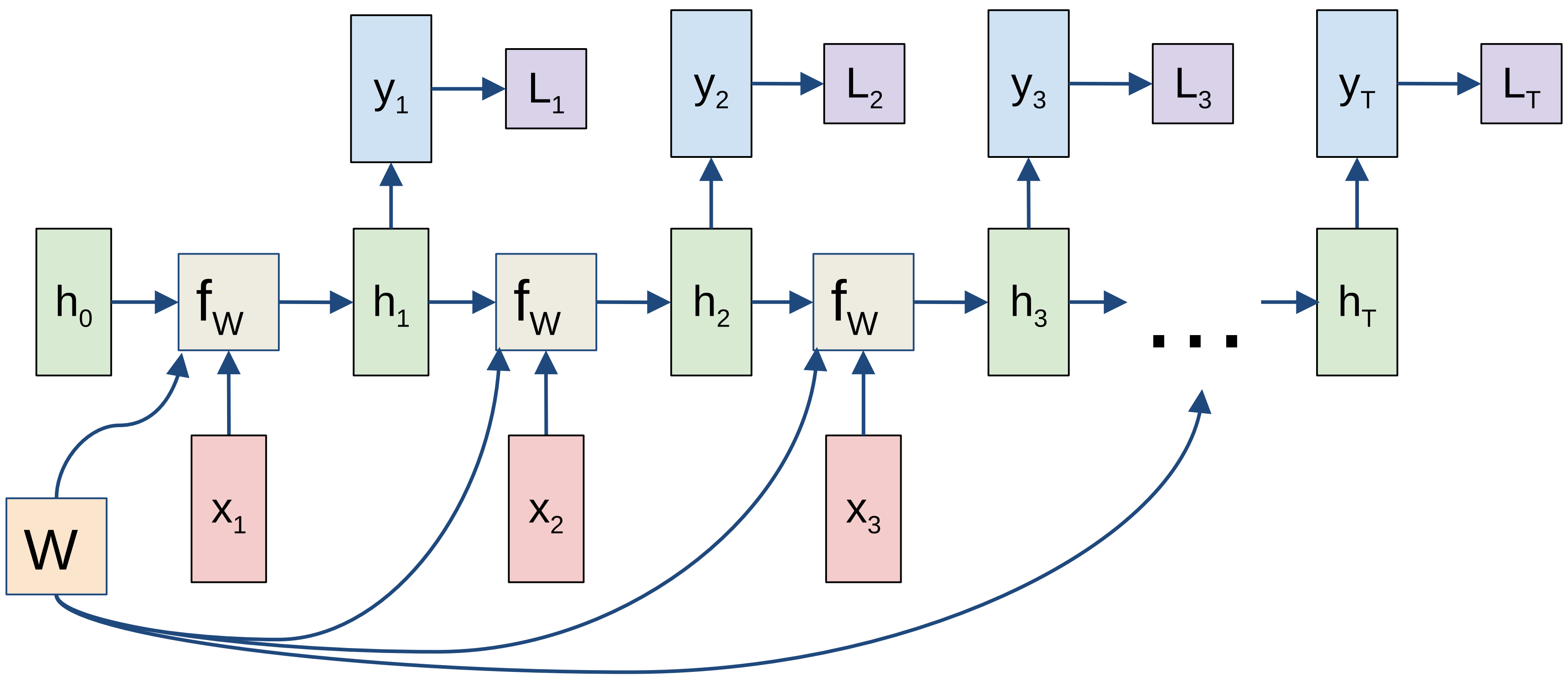
Re-use the same weight matrix at every time-step



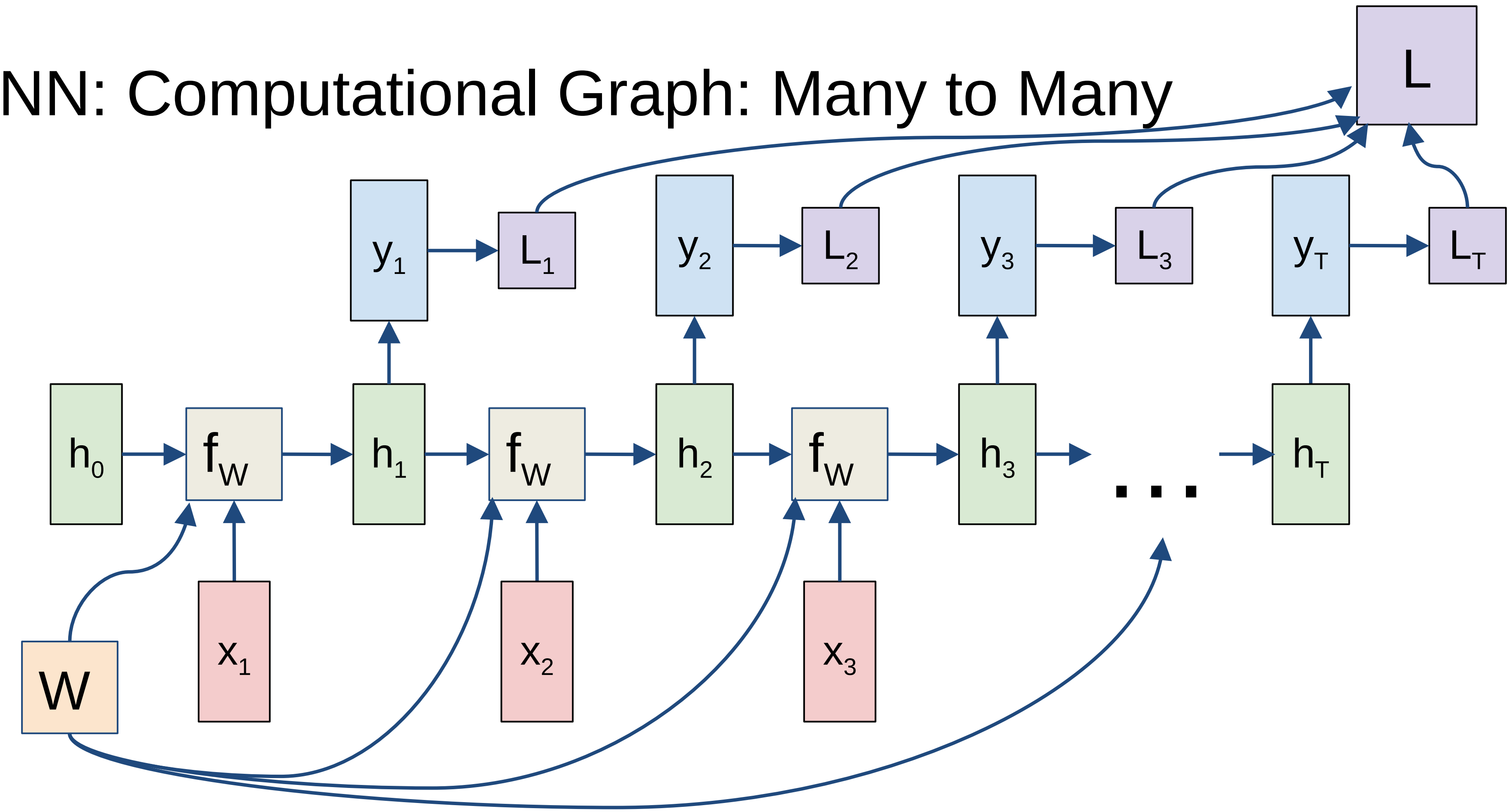
RNN: Computational Graph: Many to Many



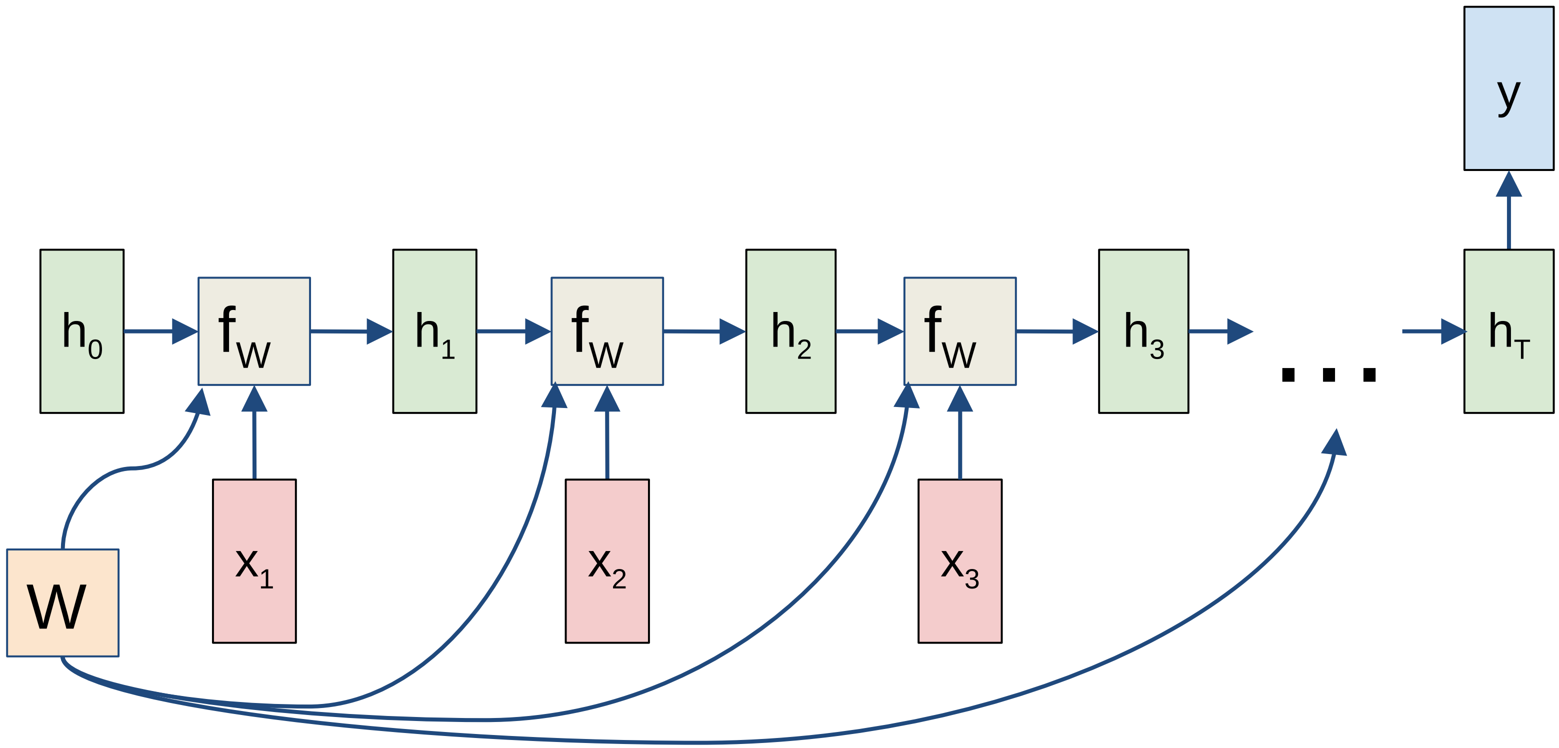
RNN: Computational Graph: Many to Many



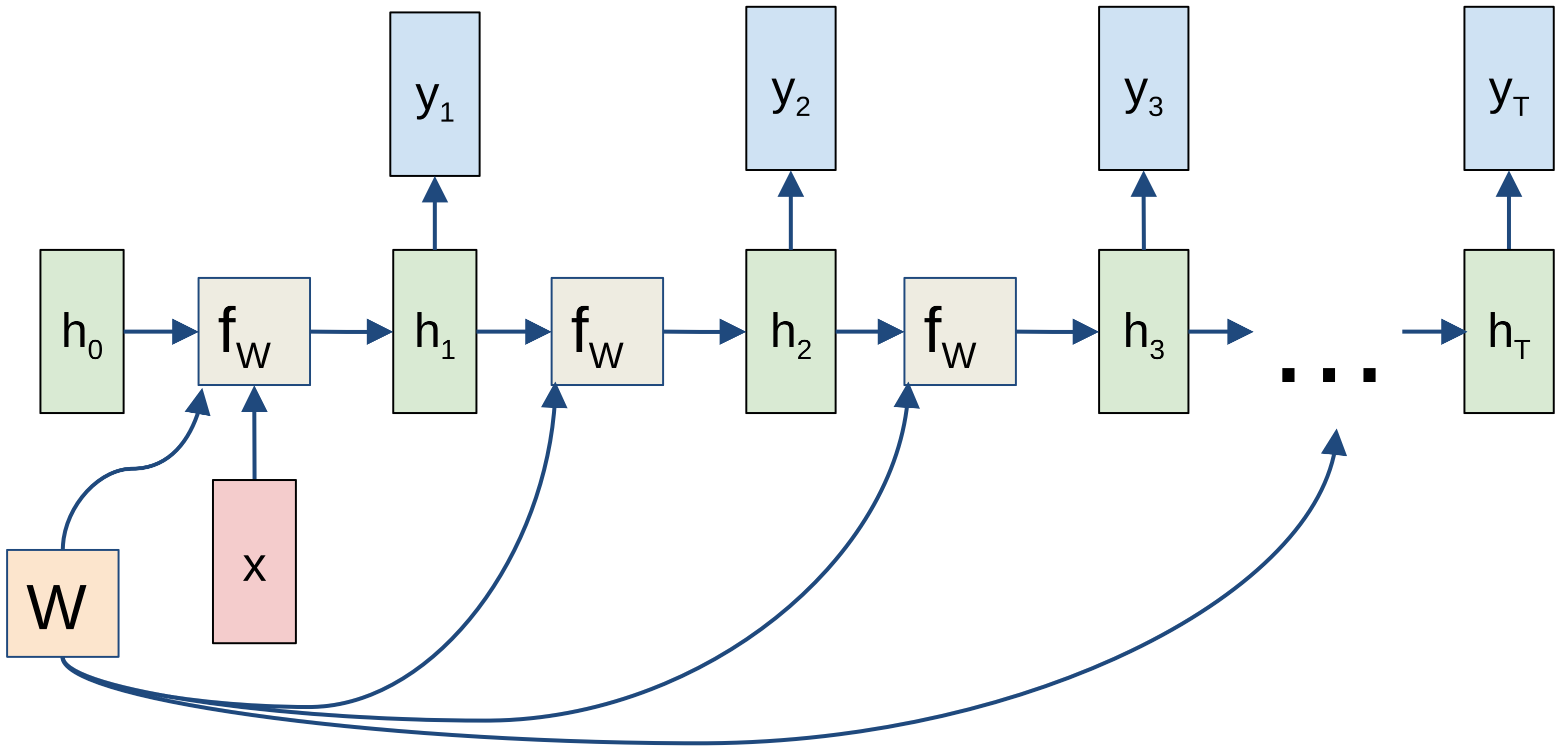
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

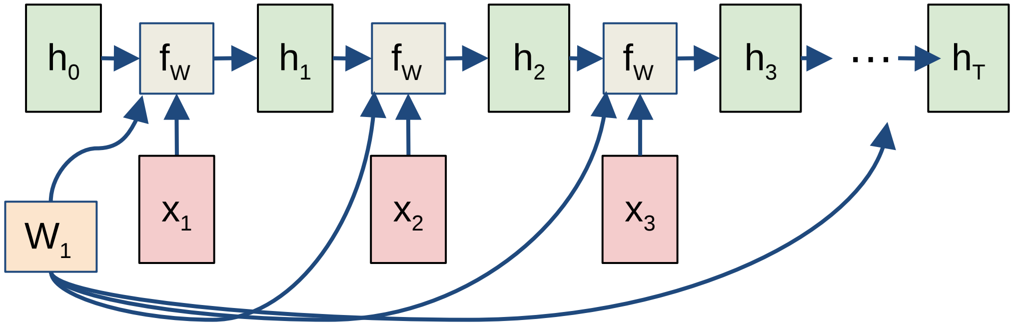


RNN: Computational Graph: One to Many



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

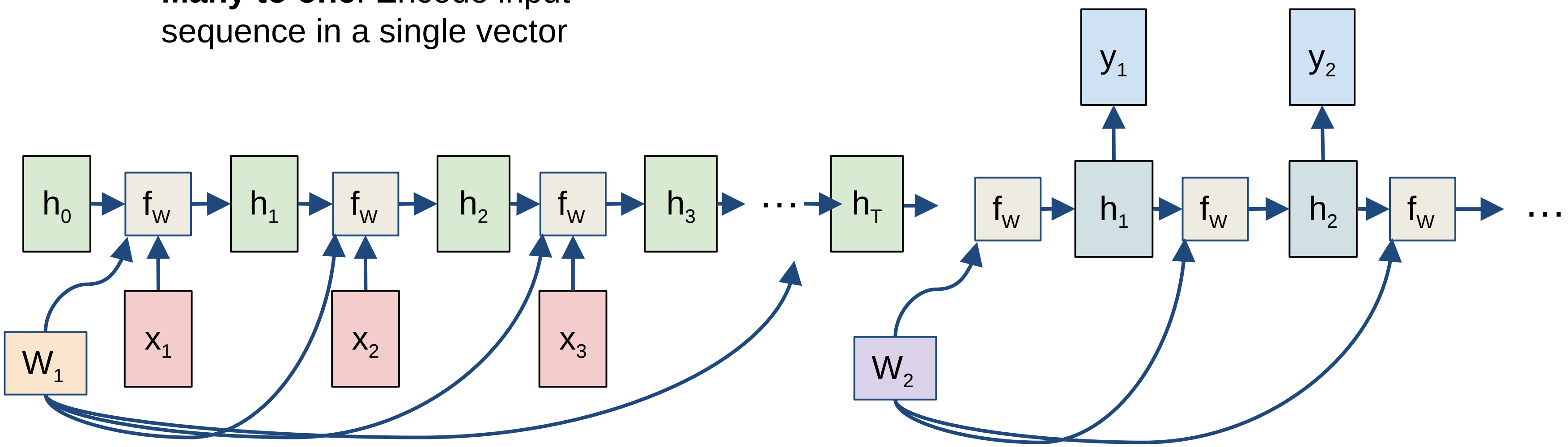
Many to one: Encode input sequence in a single vector



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

One to many: Produce output sequence from single input vector

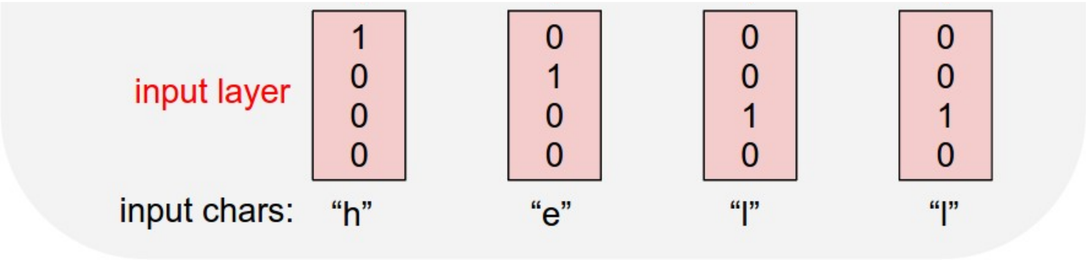
Many to one: Encode input sequence in a single vector



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

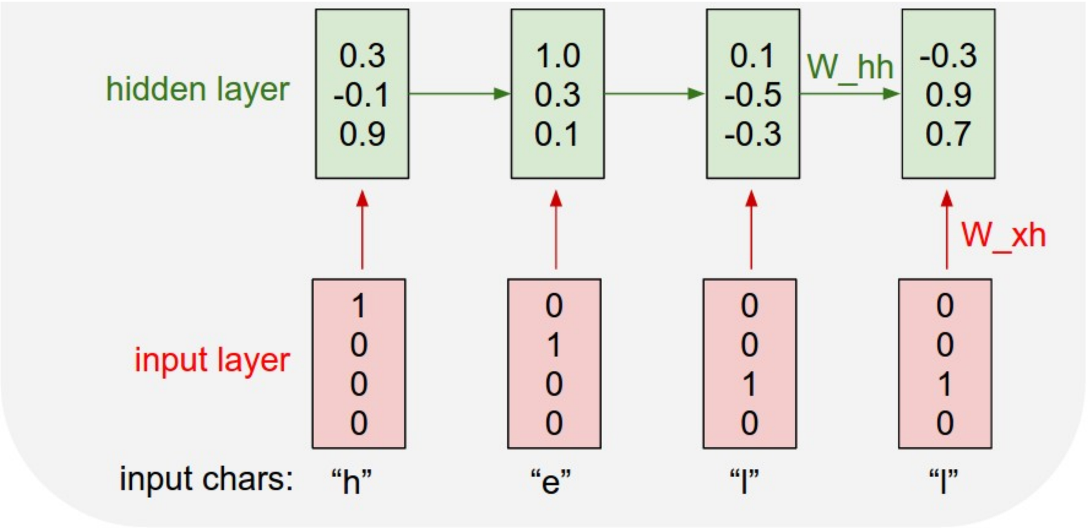


Example: Character-level Language Model

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

Vocabulary: [h,e,l,o]

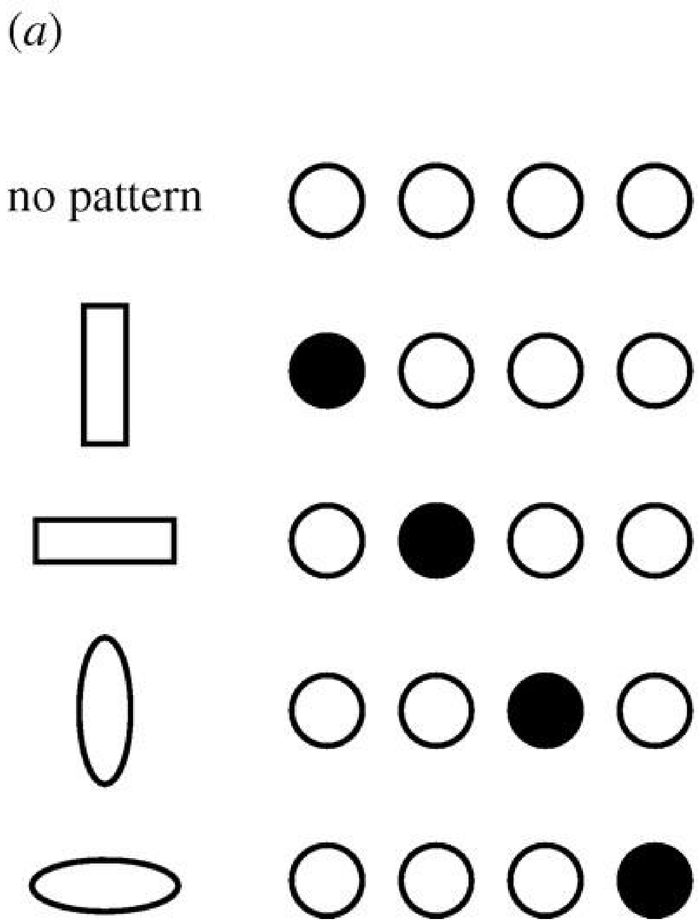
Example training sequence: **“hello”**



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

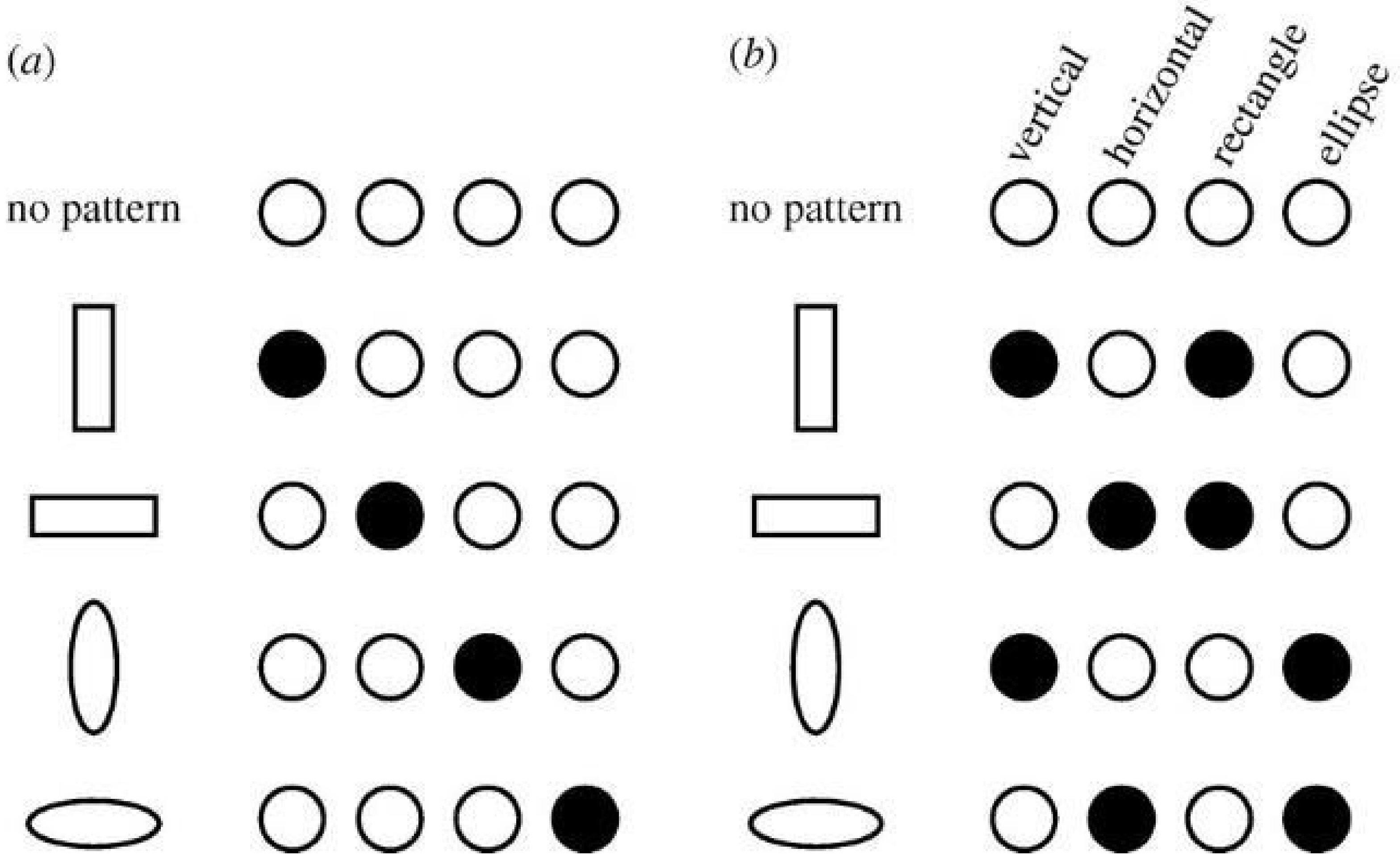
Distributed Representations Toy Example

- Local vs Distributed



Distributed Representations Toy Example

- Can we interpret each dimension?



Power of distributed representations!

Local

Distributed