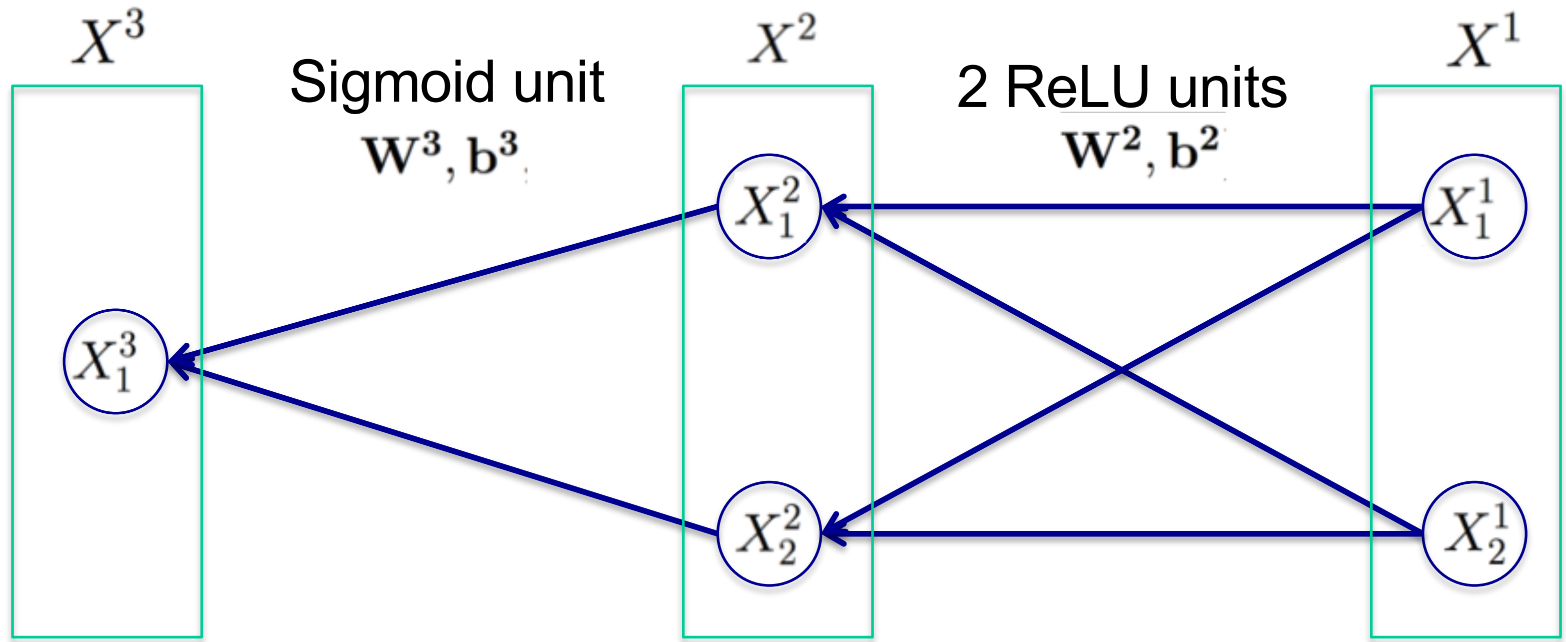
Machine Learning 10-601/301

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

March 10, 2021

This section:
- Representation learning
- Convolutional neural nets
- Recurrent neural nets

Reading:
- Goodfellow: Chapter 6
- optional: Mitchell: Chapter 4



Loss function $J(\theta)$ to be minimized: negative log likelihood of training data D

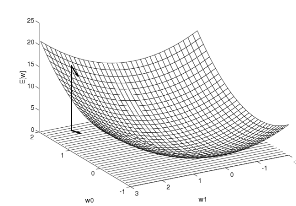
$$J(\theta) = \sum_{\langle x,y \rangle \in D} -\log P_\theta(Y = y | X = x;\ \theta)$$

where we define

$$X_1^3 = P(Y = 1 | X^1; \theta), \quad \theta = \{\mathbf{W^3}, \mathbf{b^3}, \mathbf{W^2}, \mathbf{b^2}\}$$

gradient

$$\nabla_\theta J(\theta) = \langle \frac{\partial J(\theta)}{\partial W_1^3}, \frac{\partial J(\theta)}{\partial W_2^3}, \frac{\partial J(\theta)}{\partial b^3}, \ldots, \frac{\partial J(\theta)}{\partial b_1^2} \rangle$$



note by chain rule

$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{\partial J(\theta)}{\partial X_1^3} \cdot \frac{\partial X_1^3}{\partial \theta_i}$$

Feed Forward

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$

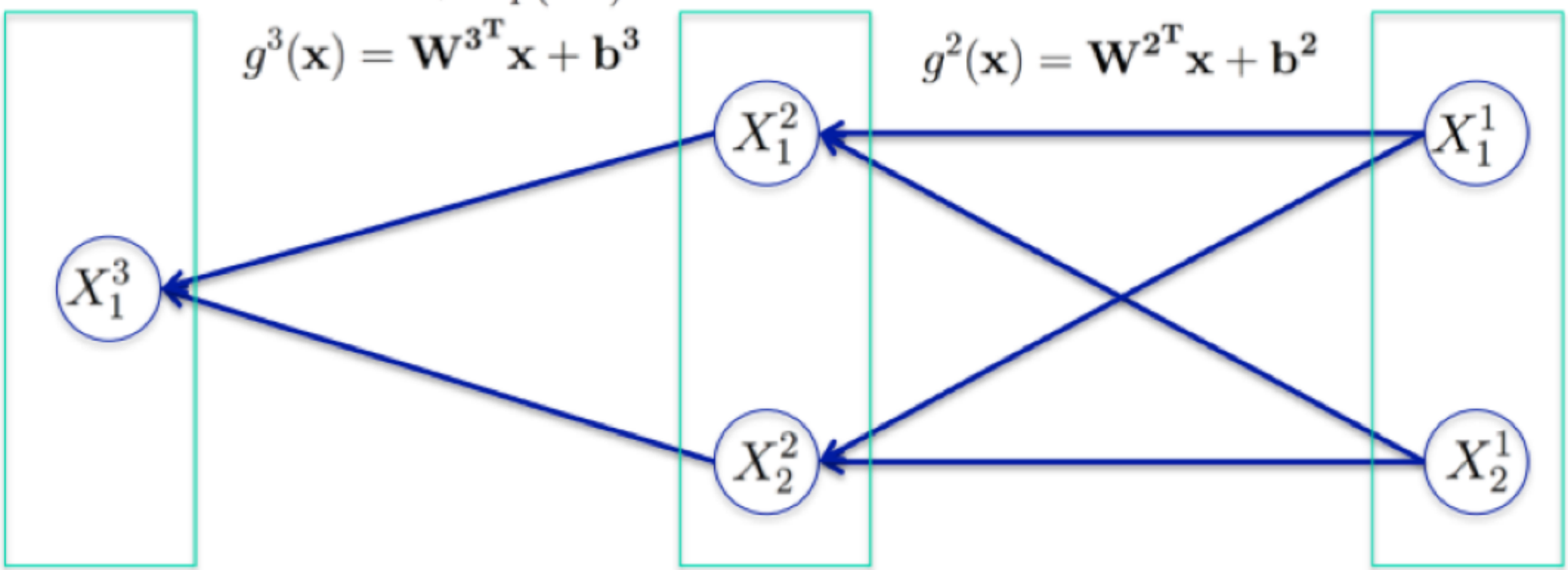
X^3 X^2 X^1

$$f^3(z) = \frac{1}{1+\exp(-z)}$$

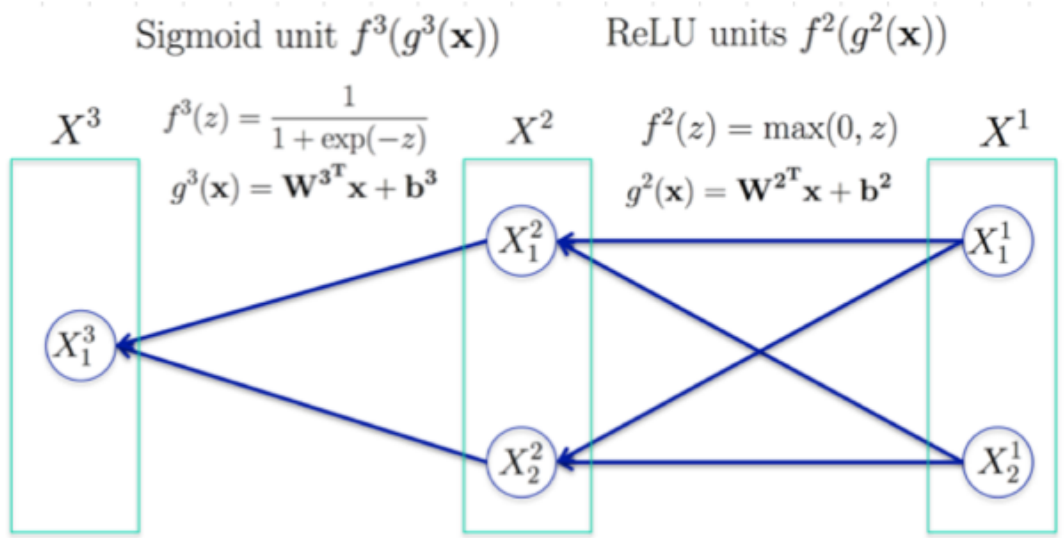
$$g^3(\mathbf{x}) = \mathbf{W^3}^{\mathbf{T}}\mathbf{x} + \mathbf{b}^3$$

$$f^2(z) = \max(0, z)$$

$$g^2(\mathbf{x}) = \mathbf{W^2}^{\mathbf{T}}\mathbf{x} + \mathbf{b}^2$$



X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*



$$\frac{\partial J(\theta)}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3}\frac{\partial X_1^3}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3} X_1^3(1 - X_1^3)$$

$$\frac{\partial J(\theta)}{\partial w_i^3} = \frac{\partial J(\theta)}{\partial g^3}\frac{\partial g^3}{\partial w_i^3} = \frac{\partial J(\theta)}{\partial g^3} X_i^2$$

already know this

$$\frac{\partial J(\theta)}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3}\frac{\partial g^3}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3} w_i^3$$

$$\frac{\partial J(\theta)}{\partial g_i^2} = \frac{\partial J(\theta)}{\partial X_i^2}\frac{\partial X_i^2}{\partial g_i^2} = \frac{\partial J(\theta)}{\partial X_i^2} \times [\text{if } g_i^2 > 0 \text{ then } 1 \text{ else } 0]$$

$$\frac{\partial J(\theta)}{\partial w_{ik}^2} = \frac{\partial J(\theta)}{\partial g_i^2}\frac{\partial g_i^2}{\partial w_{ik}^2} = \frac{\partial J(\theta)}{\partial g_i^2} X_k^1$$

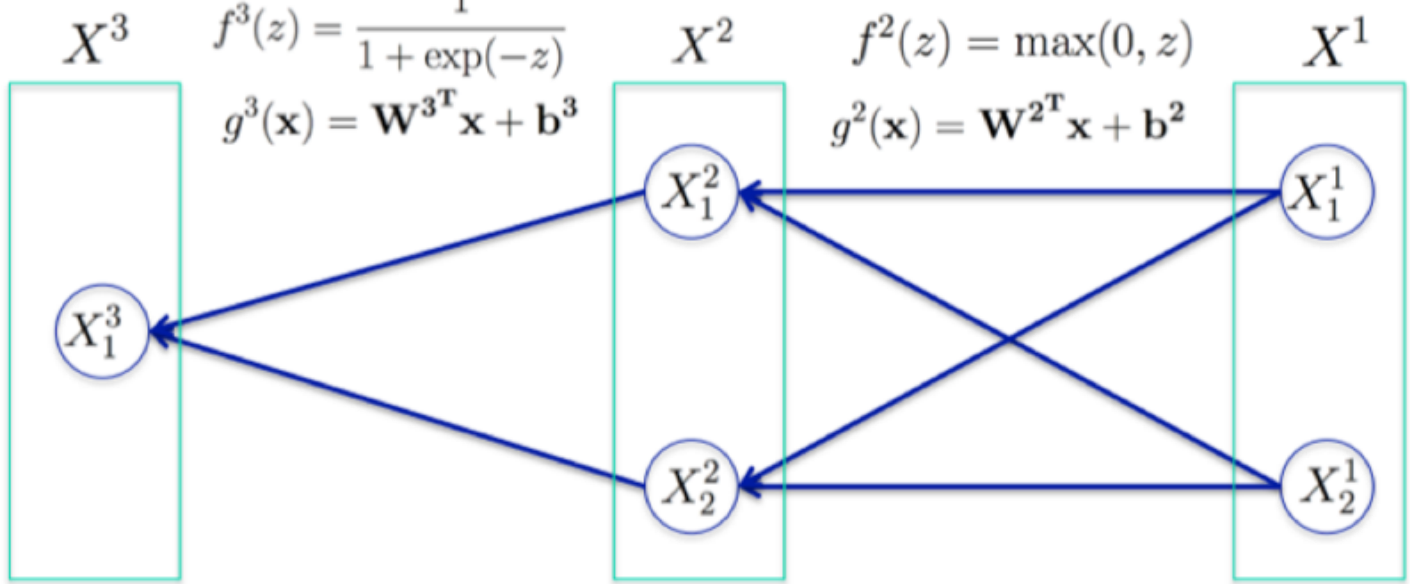
Back propagation

Ytrue	1		
dJ/dX3	-1.89		
dJ/dg3	-0.47		
dJ/dW3T	-0.09	0.00	-0.47
dJ/dX2	-0.05		
	0.04		
dJ/dg2	-0.05		
	0.00		
dJ/dW2T	-0.05	0.00	-0.05
	0.00	0.00	0.00

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$

$$f^3(z) = \frac{1}{1+\exp(-z)} \qquad g^3(\mathbf{x}) = \mathbf{W^3}^{\mathbf{T}}\mathbf{x} + \mathbf{b^3}$$

$$f^2(z) = \max(0, z) \qquad g^2(\mathbf{x}) = \mathbf{W^2}^{\mathbf{T}}\mathbf{x} + \mathbf{b^2}$$



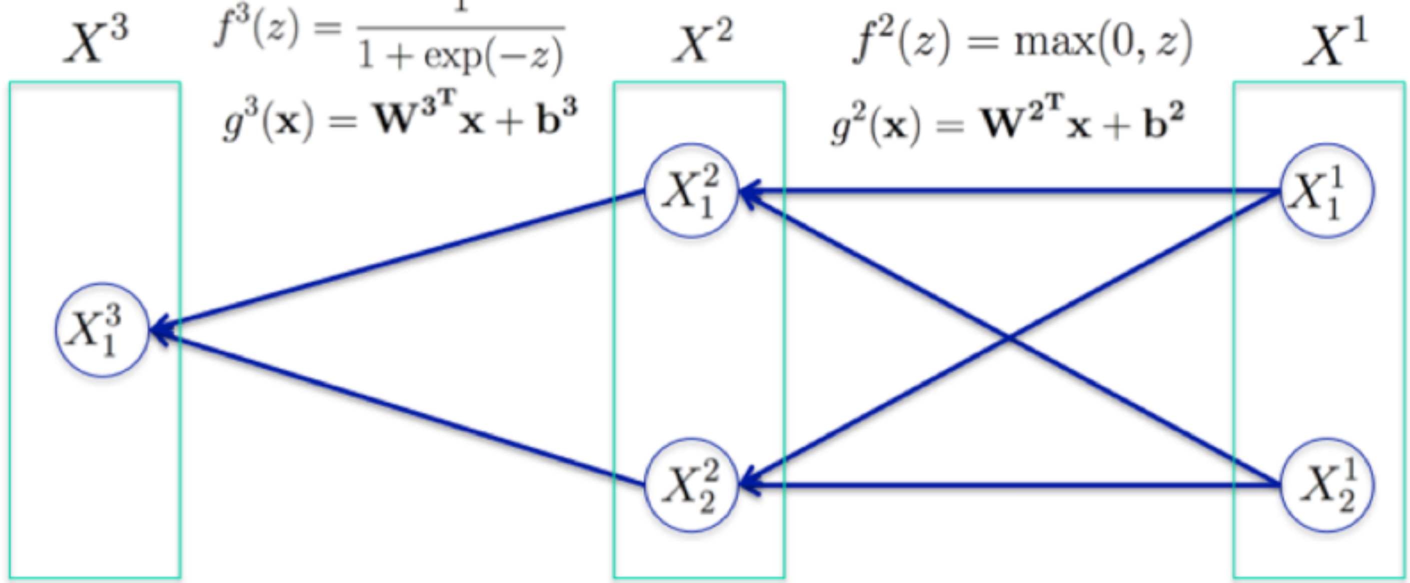
X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*

$$\boxed{\text{update each parameter according to } \theta_i \leftarrow \quad \theta_i - \eta\frac{\partial J(\theta)}{\partial \theta_i}}$$

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$

$$f^3(z) = \frac{1}{1+\exp(-z)} \qquad f^2(z) = \max(0, z)$$

$$g^3(\mathbf{x}) = \mathbf{W^3}^{\mathbf{T}}\mathbf{x} + \mathbf{b^3} \qquad g^2(\mathbf{x}) = \mathbf{W^2}^{\mathbf{T}}\mathbf{x} + \mathbf{b^2}$$



Ytrue	1		
dJ/dX3	-1.89		
dJ/dg3	-0.47		
dJ/dW3T	-0.09	0.00	-0.47
dJ/dX2	-0.05		
	0.04		
dJ/dg2	-0.05		
	0.00		
dJ/dW2T	-0.05	0.00	-0.05
	0.00	0.00	0.00

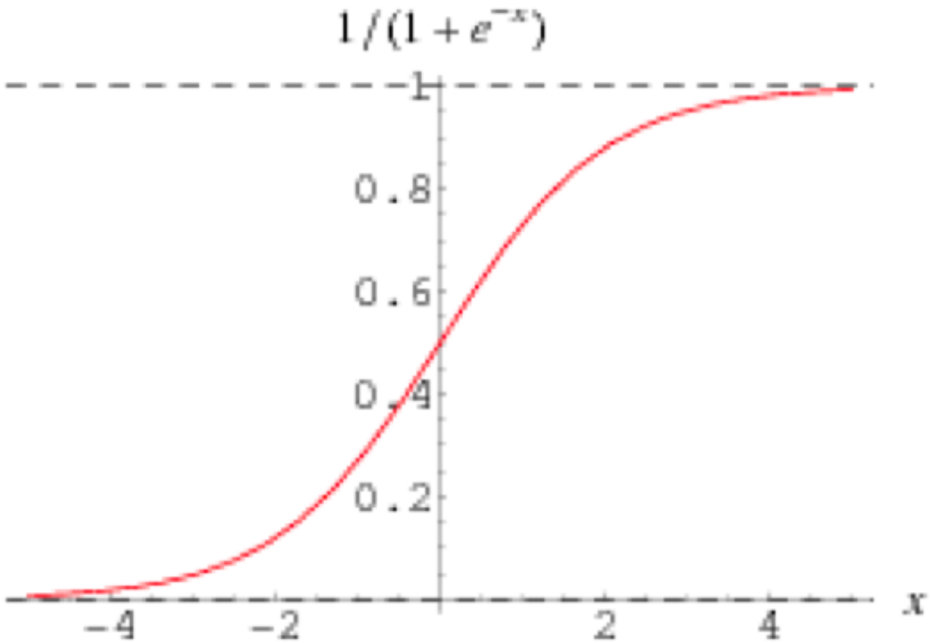
X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*

update each parameter according to $\theta_i \leftarrow \quad \theta_i - \eta \frac{\partial J(\theta)}{\partial \theta_i}$

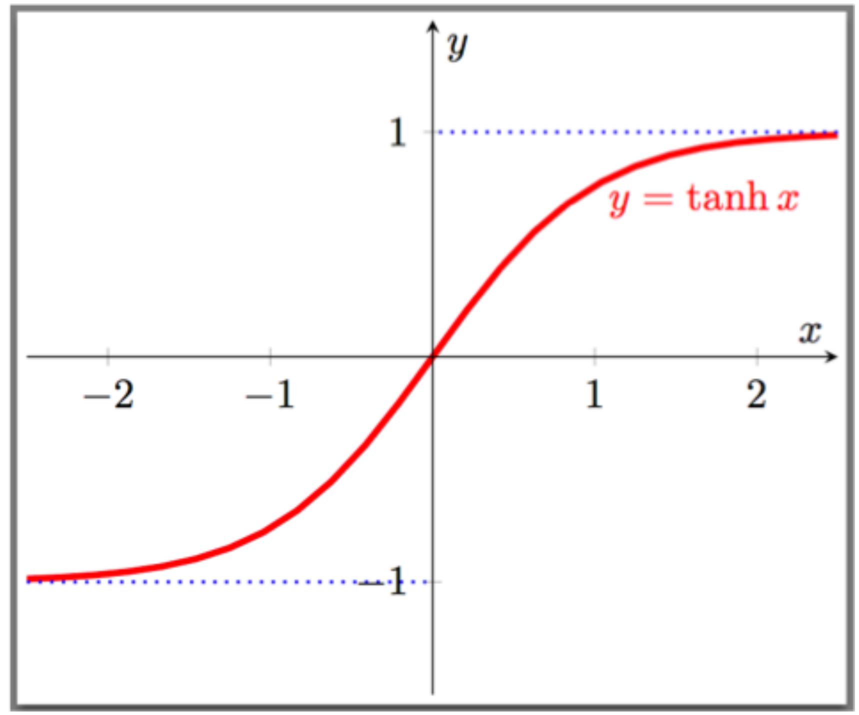
using $\eta = 1$:

X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.65	0.63	0.19	-0.09	0.57	0.30	0.30	0.15	-0.10	0.15	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*

Sigmoid function



tanh function



What you should know: Artificial Neural Networks

- Highly non-linear regression/classification
- ~~Vector~~ Tensor-valued inputs and outputs
- Potentially billions of parameters to estimate
- Hidden layers learn intermediate representations

aka backpropagation

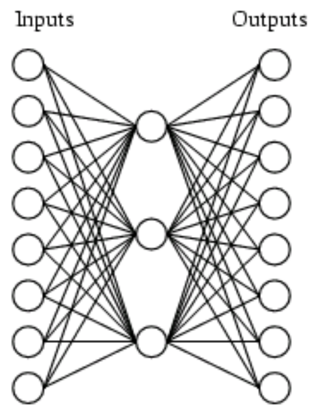
- Directed acyclic graph, trained by gradient descent
- Chain rule over this DAG allows computing all derivatives
- Can use any differentiable loss function
  - we used neg. log likelihood in order to learn outputs P(Y|X)

- Gradient descent, local minima problems
- Overfitting and how to deal with it

Learning hidden representations

Learning Hidden Layer Representations

Network with sigmoid units only:



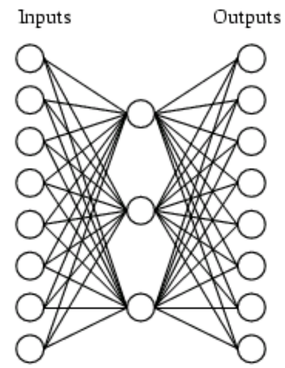
A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned??

Learning Hidden Layer Representations

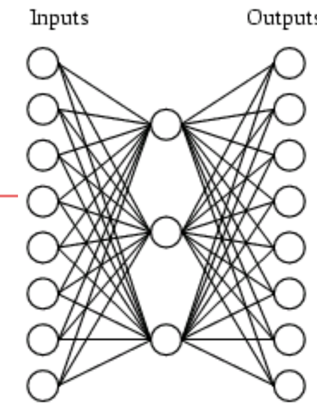
A network:

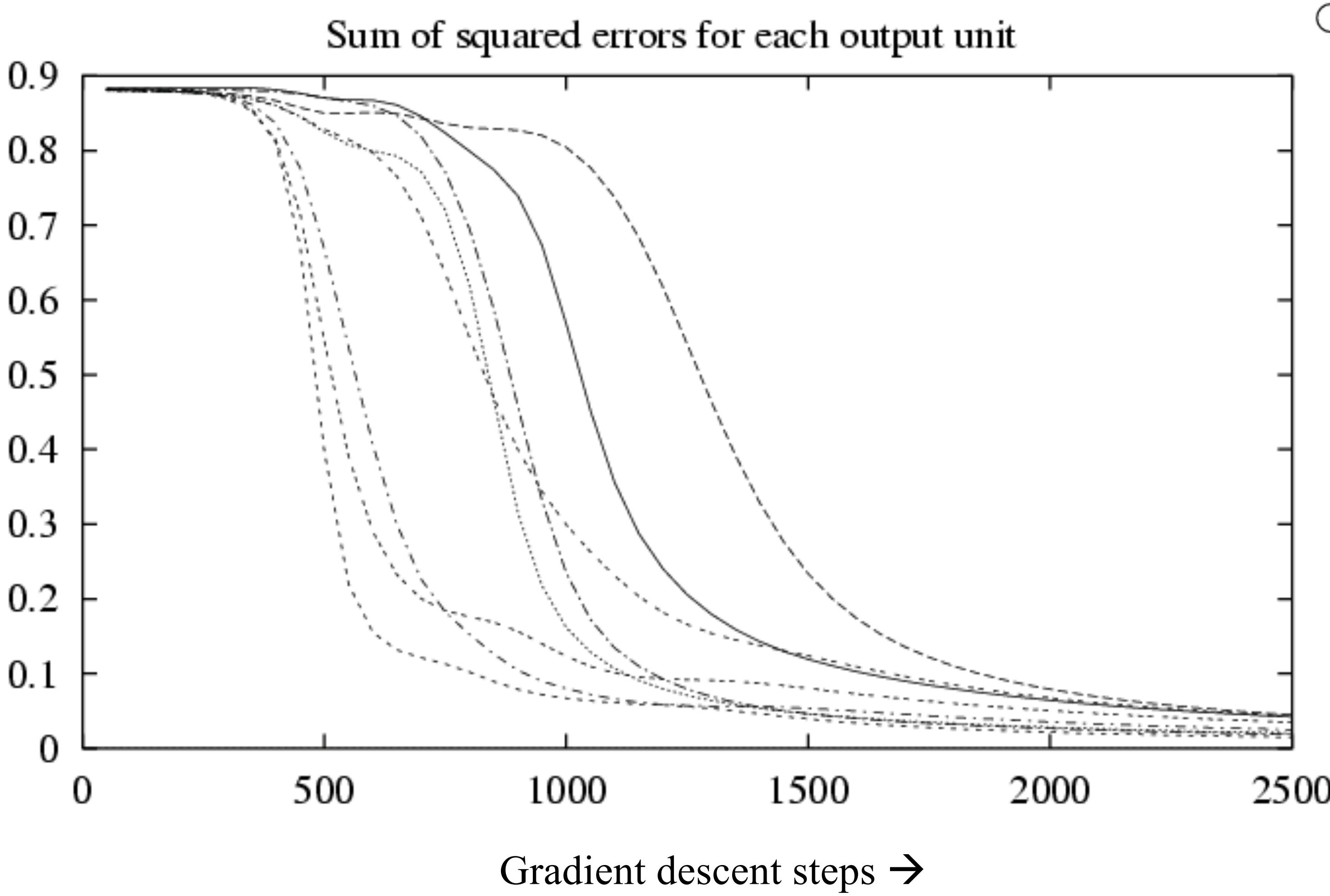


Learned hidden layer representation:

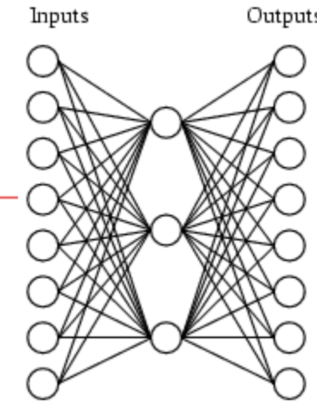
Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$	00001000
00000100	$\rightarrow$	.22	.99	.99	$\rightarrow$	00000100
00000010	$\rightarrow$	.80	.01	.98	$\rightarrow$	00000010
00000001	$\rightarrow$	.60	.94	.01	$\rightarrow$	00000001

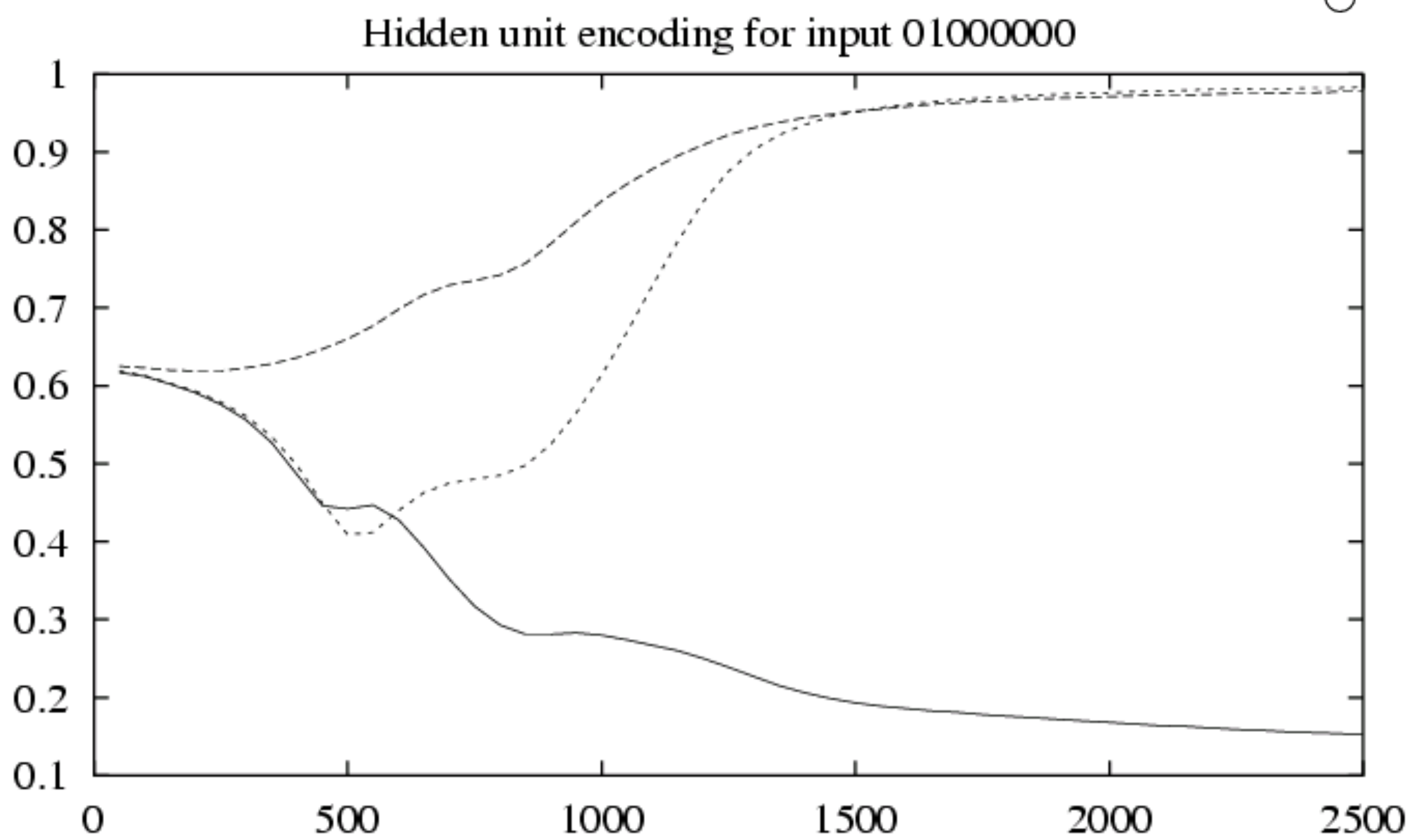
Training


Inputs
Outputs


Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500
Gradient descent steps →

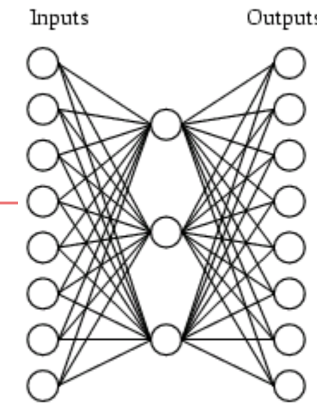
Training

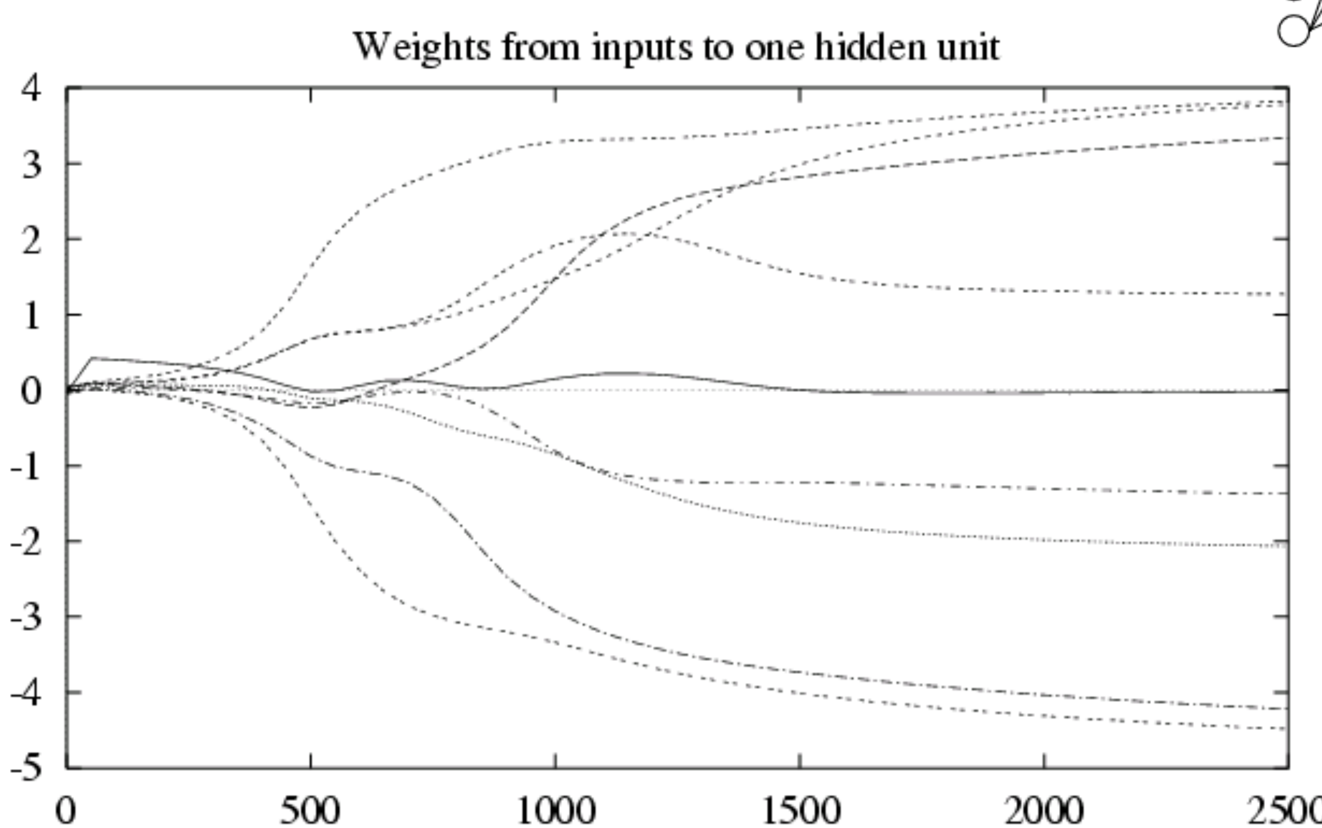




Gradient descent steps →

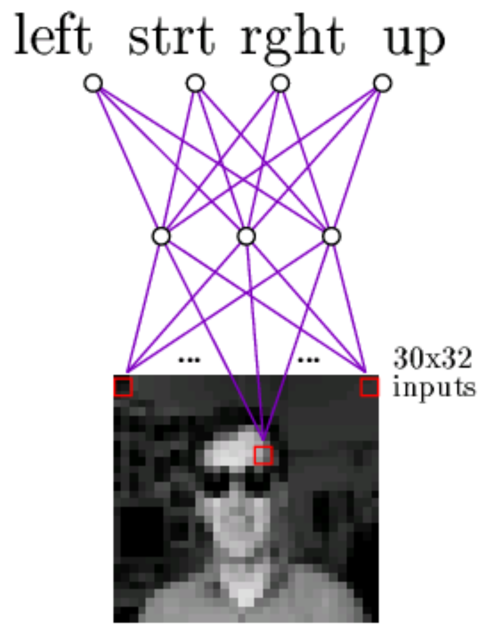
Training





Gradient descent steps →

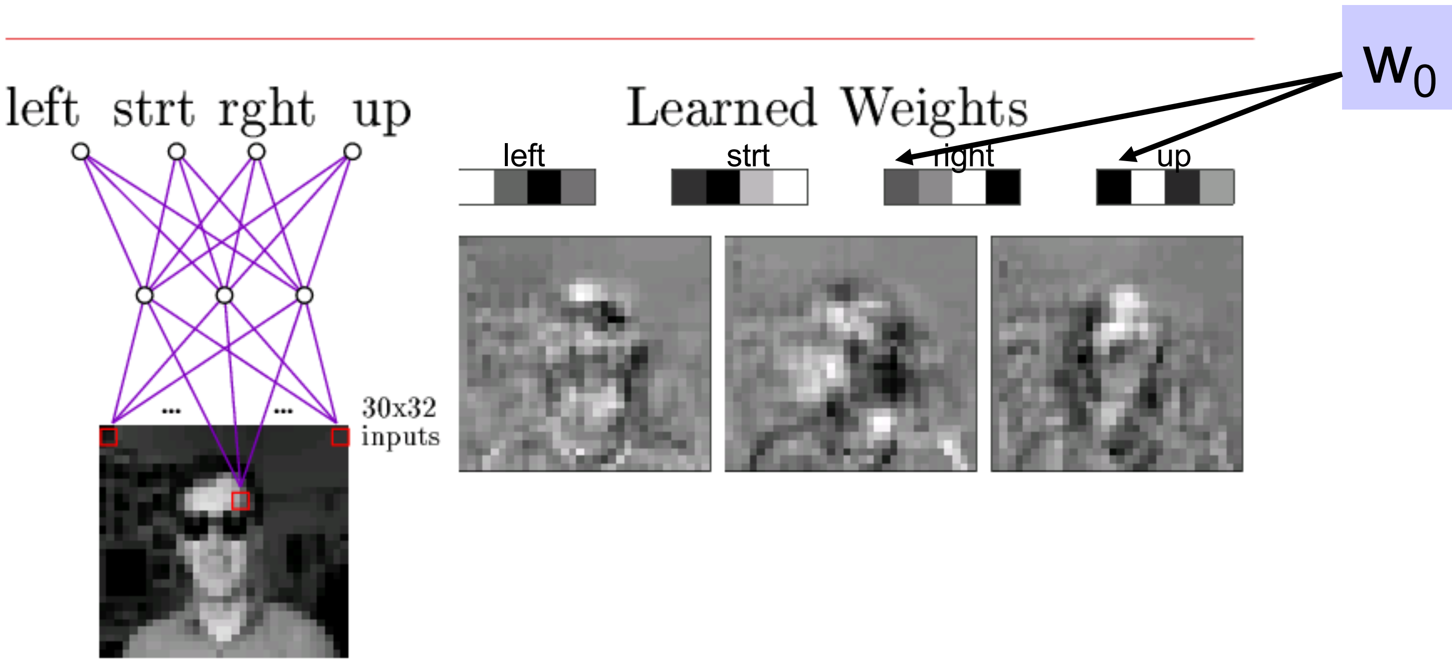
Neural Nets for Face Recognition



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Learned Hidden Unit Weights



Typical input images

http://www.cs.cmu.edu/~tom/faces.html

Word embeddings

Learning Distributed Representations for Words

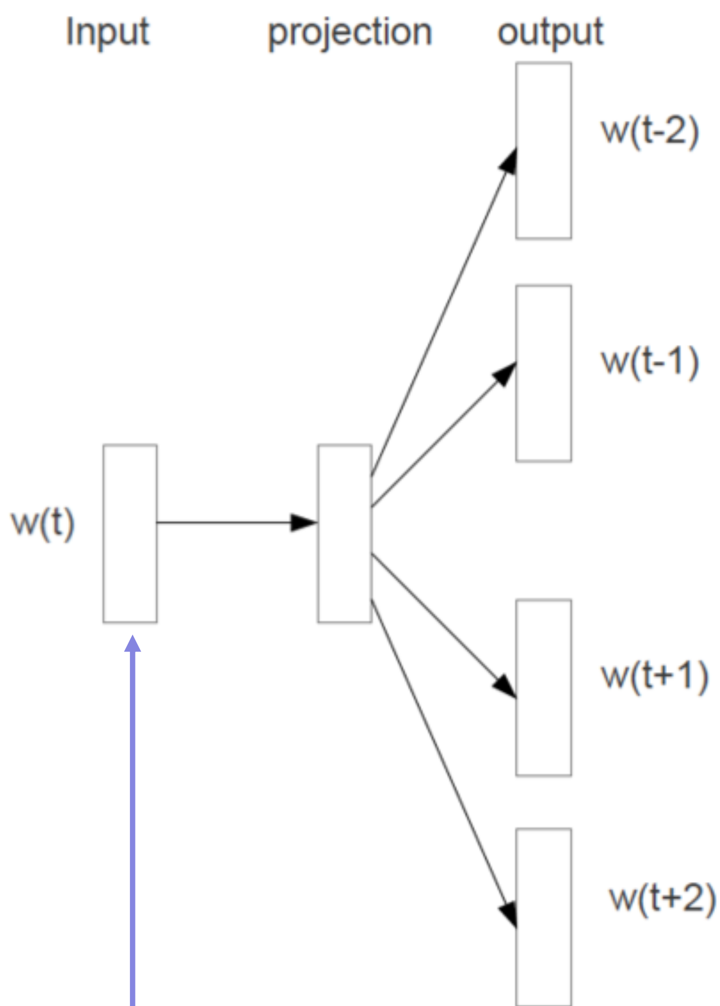
- also called "word embeddings"
- word2vec is one commonly used embedding
- based on "skip gram" model

Key idea: given word sequence w_1 w_2 … w_T
train network to predict surrounding words.

for each word w_t predict w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}

e.g., "the dog jumped over the fence in order to get to.."
"the cat jumped off the widow ledge in order to ..."

Word2Vec Word Embeddings



Learned dense vector v_w for each word w. 300 dimensional

"one hot" word encoding: all zeros except 1 position. ~50k dimensional

basic skip gram model:

train to maximize:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

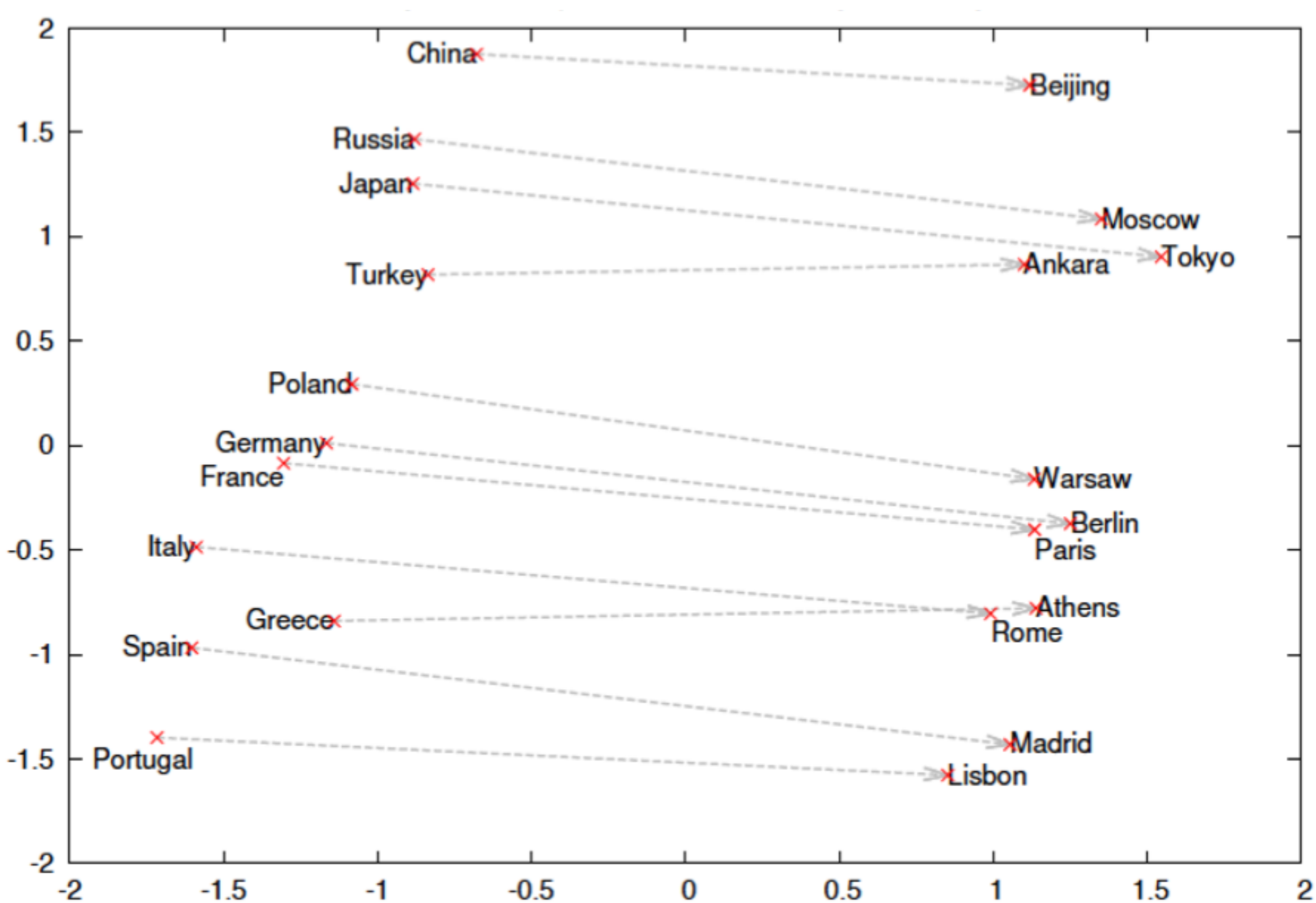
where

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top} v_{w_I}\right)}$$

Modifications to training…
+ hierarchical softmax
+ negative sampling
+ subsample frequent w's

[Mikolav et al., 2013]

100 Dimensional Skip-gram embeddings, projected to two dimensions by PCA



Skip-gram Word Embeddings

analogy: w1 is to w2, as w3 is to ?w
algorithm: ?w = w2-w1+w3

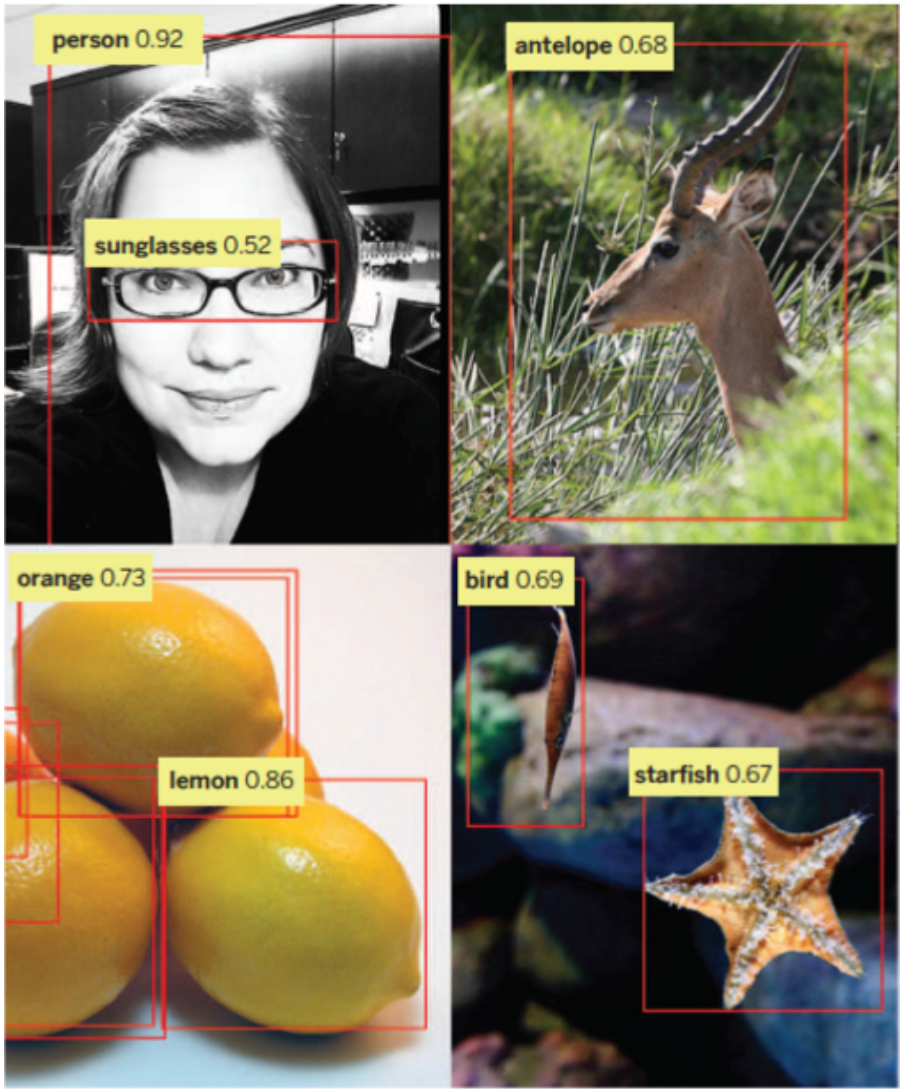
Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

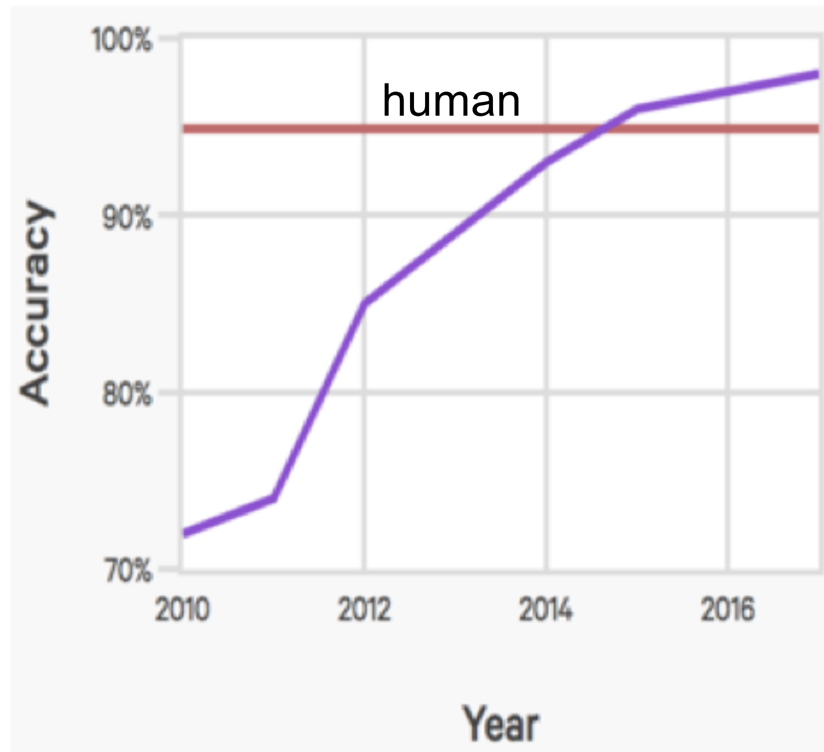
[Mikolav et al., 2013]

Convolutional Neural Nets

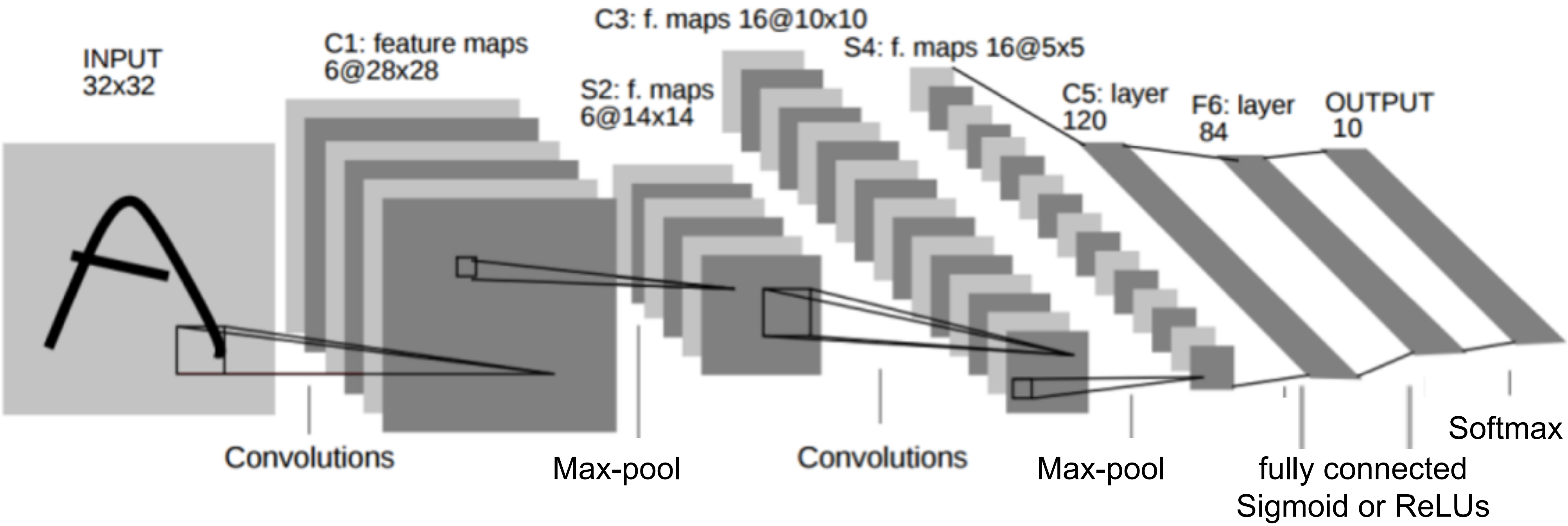
Computer Vision



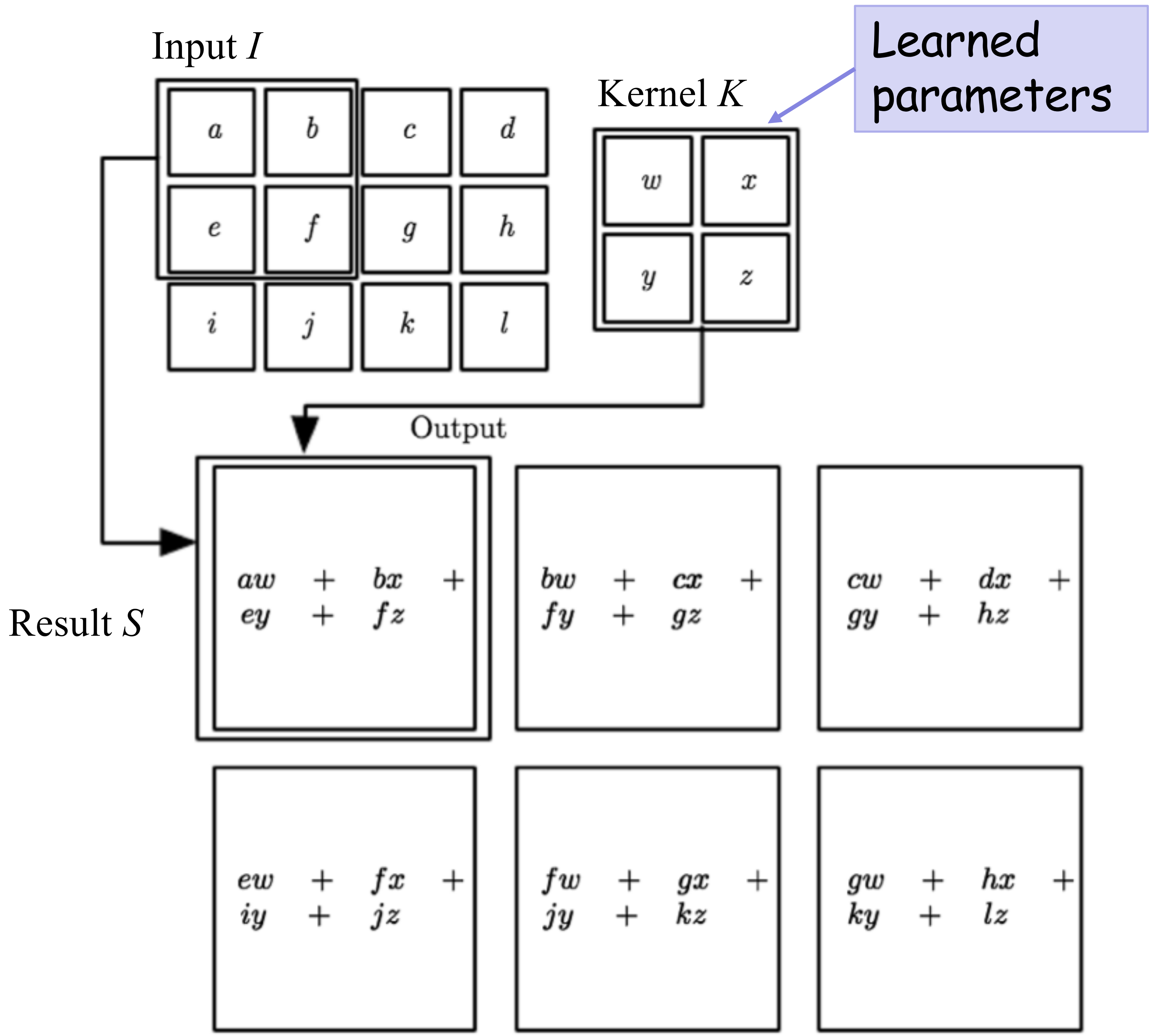
Imagenet Visual Recognition Challenge



A Convolutional Neural Net for Handwritten Digit recognition: LeNet5



Convolution Layer

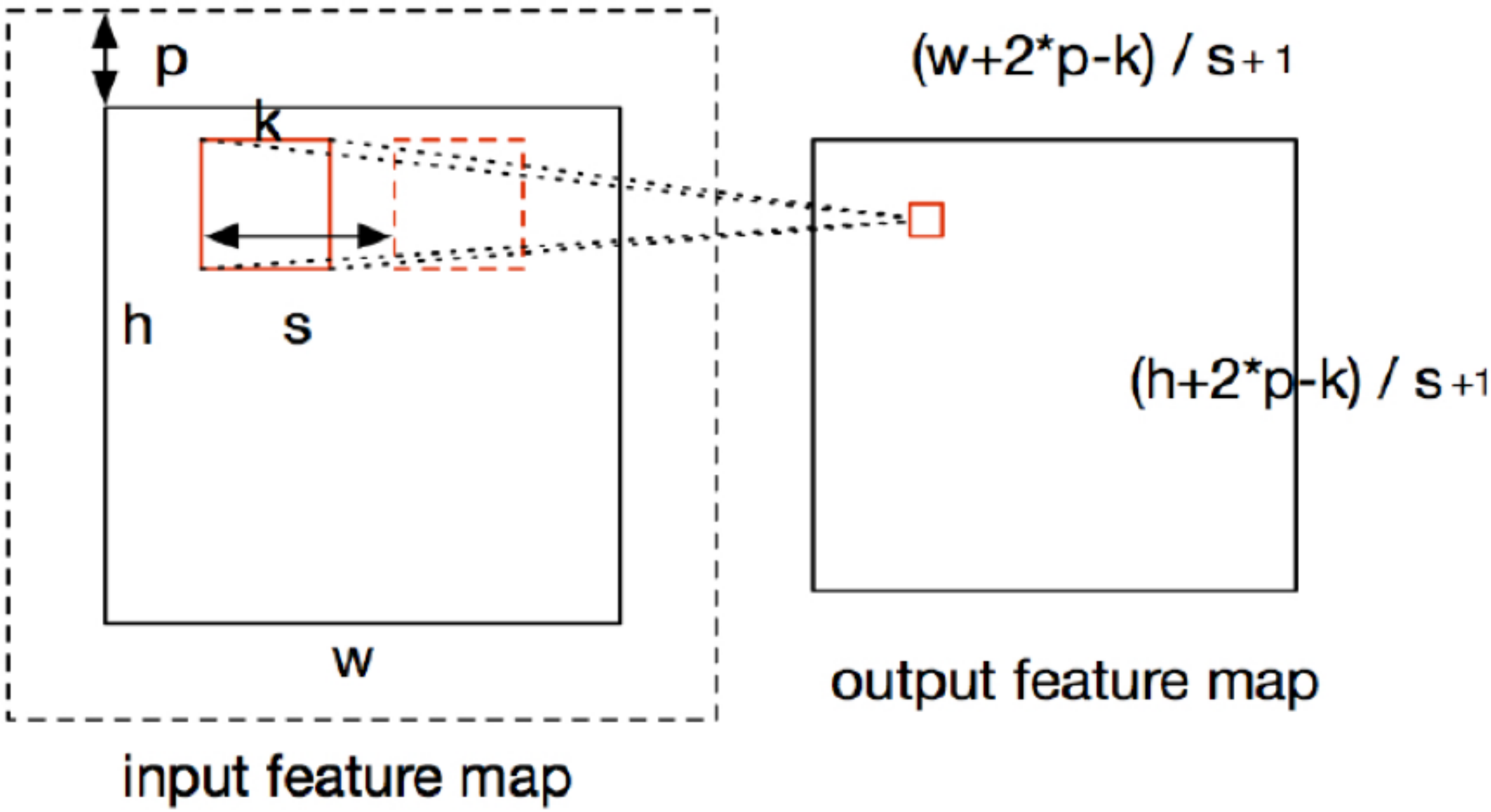


$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

[from Goodfellow et al.]

Convolution layer

p = padding
s = stride



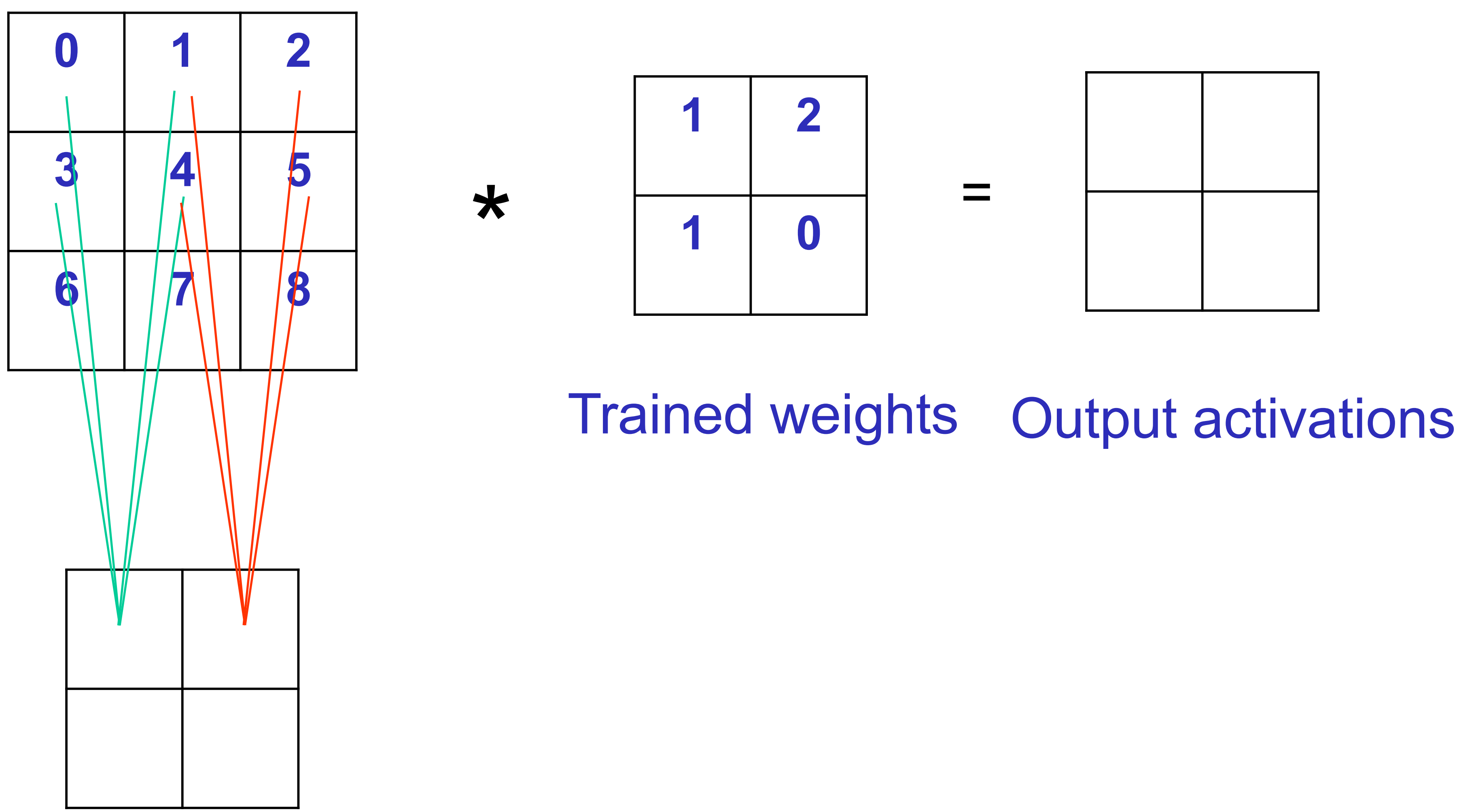
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Convolution example

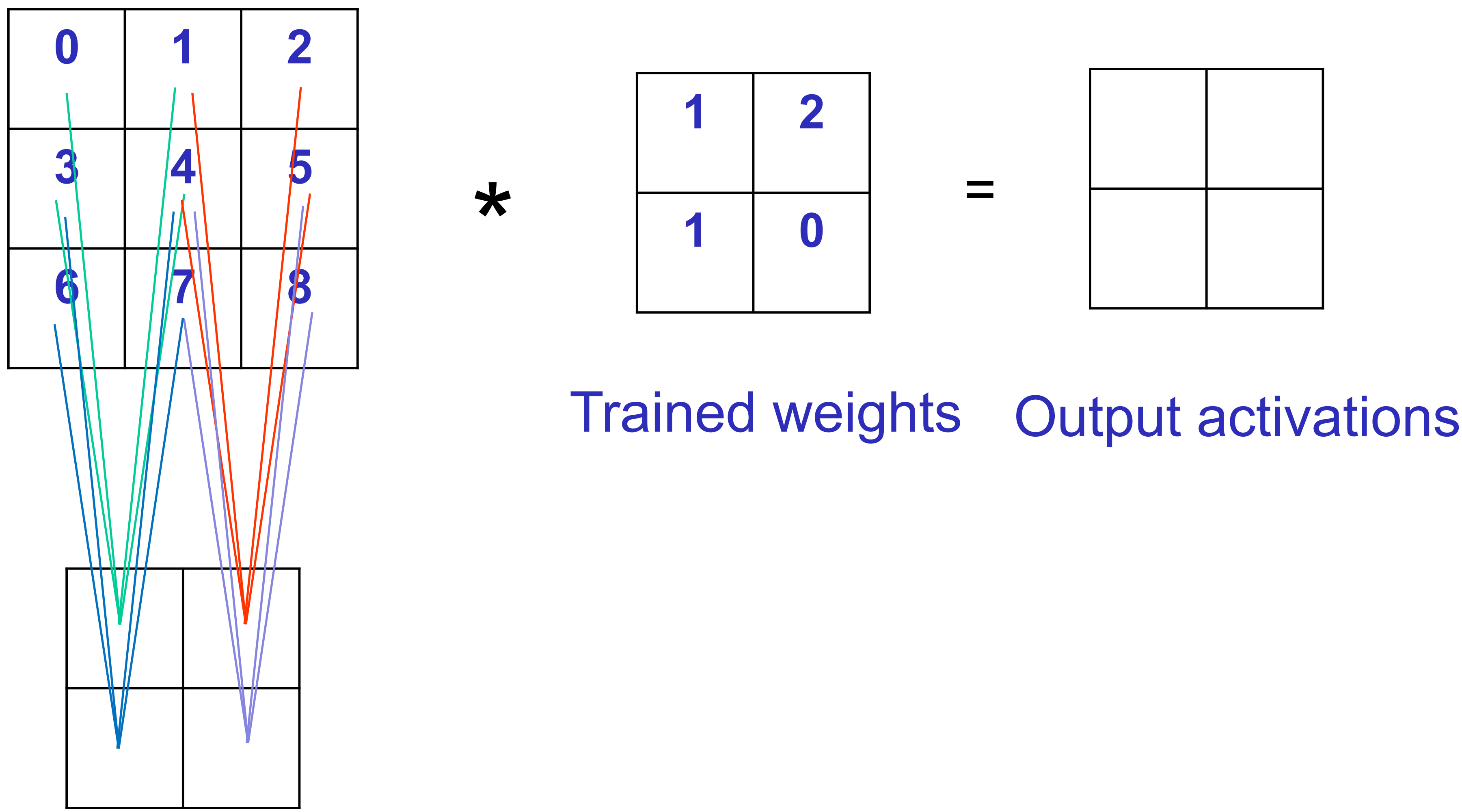
$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$

Input activations — Trained weights — Output activations

Convolution as parameter sharing



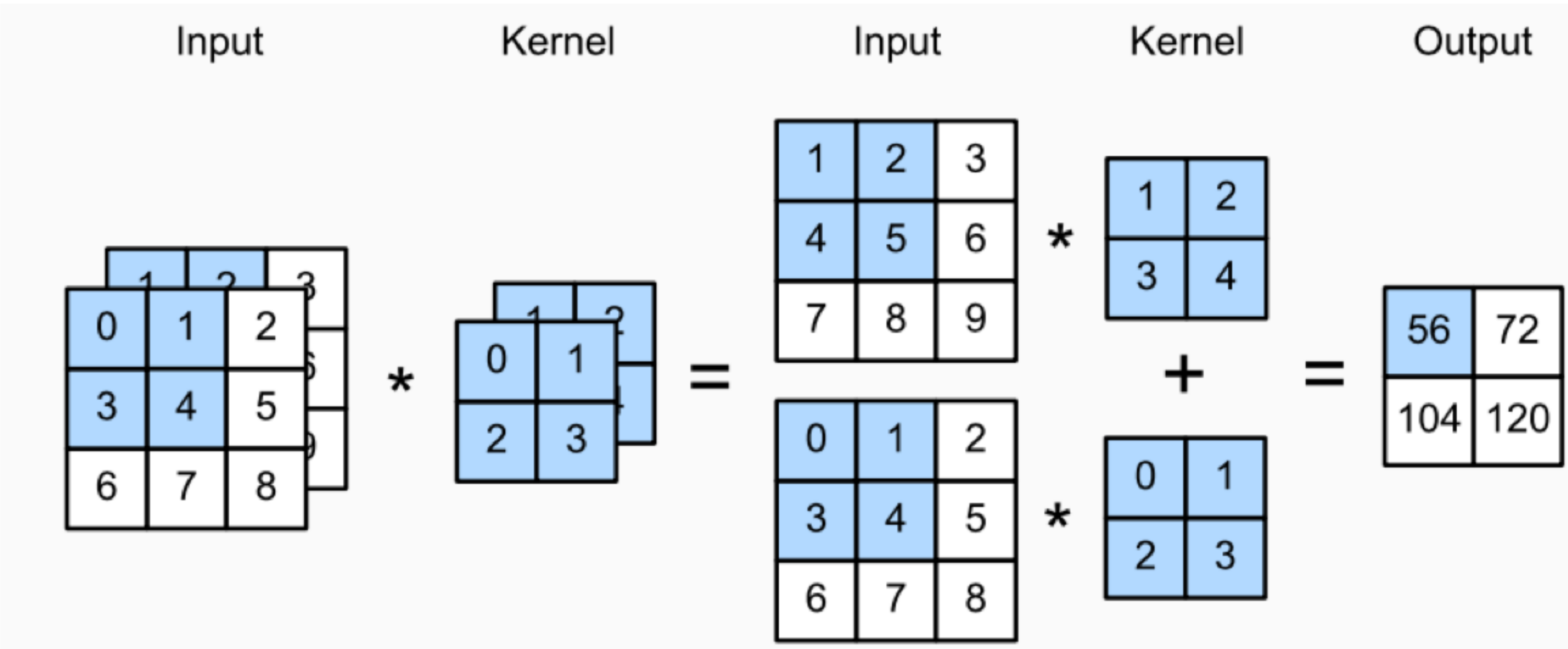
Convolution as parameter sharing



Convolution with padding

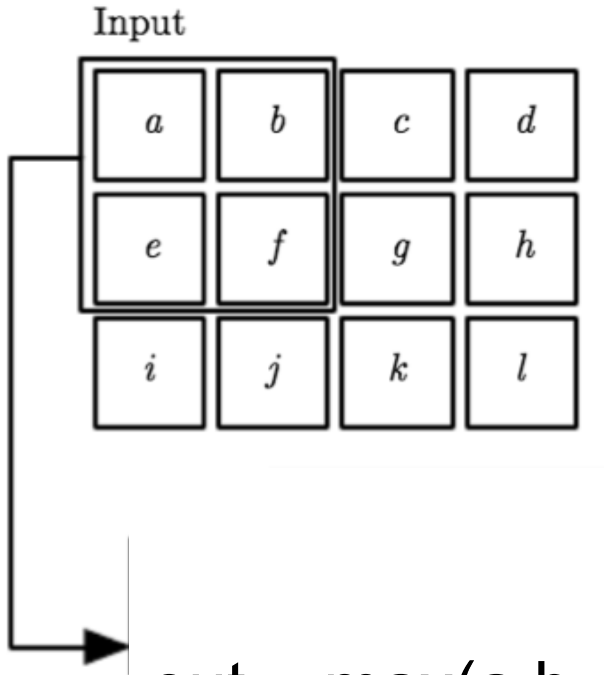
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

Multichannel Convolution

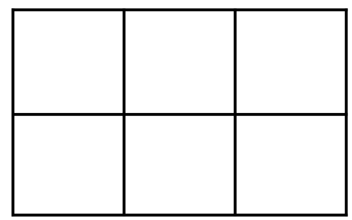


[from Zhang et al., Dive into Deep Learning"]

Maxpool Layer

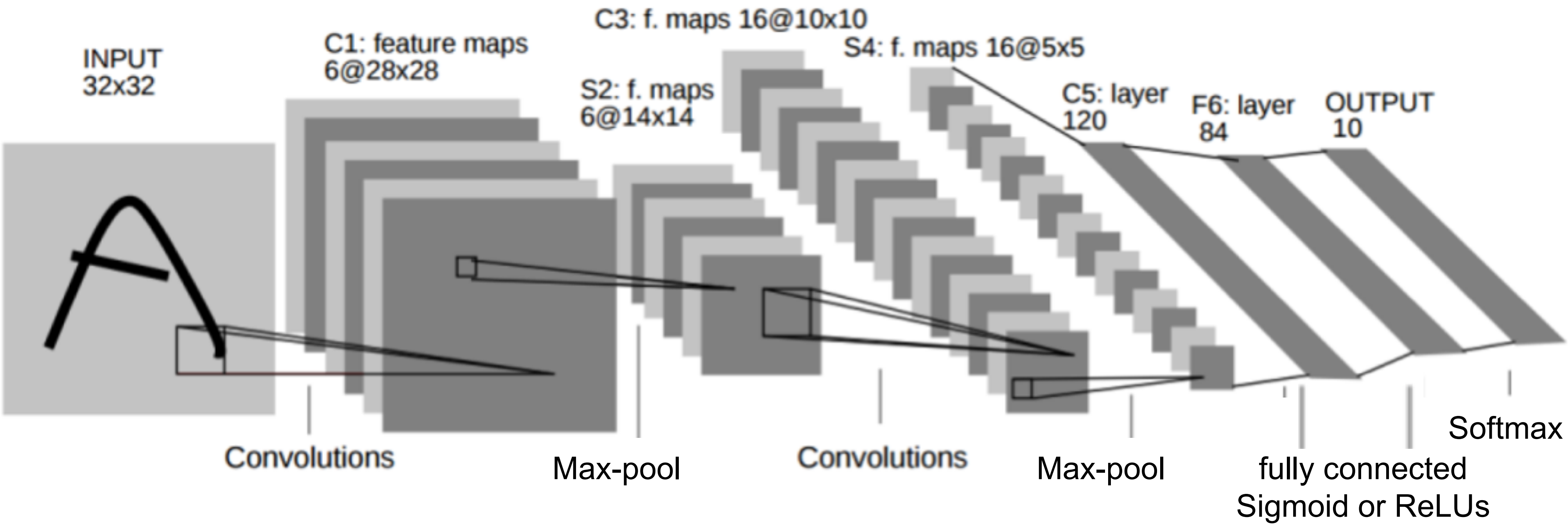


out = max(a,b,e,f)



[from Goodfellow et al.]

A Convolutional Neural Net for Handwritten Digit recognition: LeNet5



Softmax Layer: Predict Probability Distribution over discrete-valued variables

- Logistic Regression: when Y has two possible values

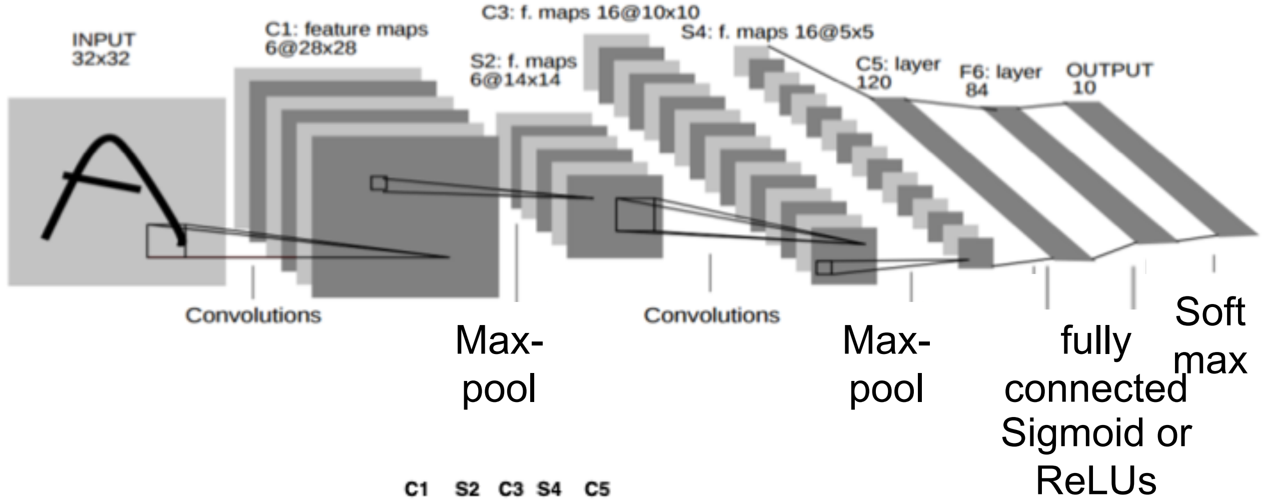
$$P(Y=1|X=\langle X_1,\ldots X_n\rangle)=\frac{1}{1+\exp(w_0+\sum_i w_i X_i)}$$

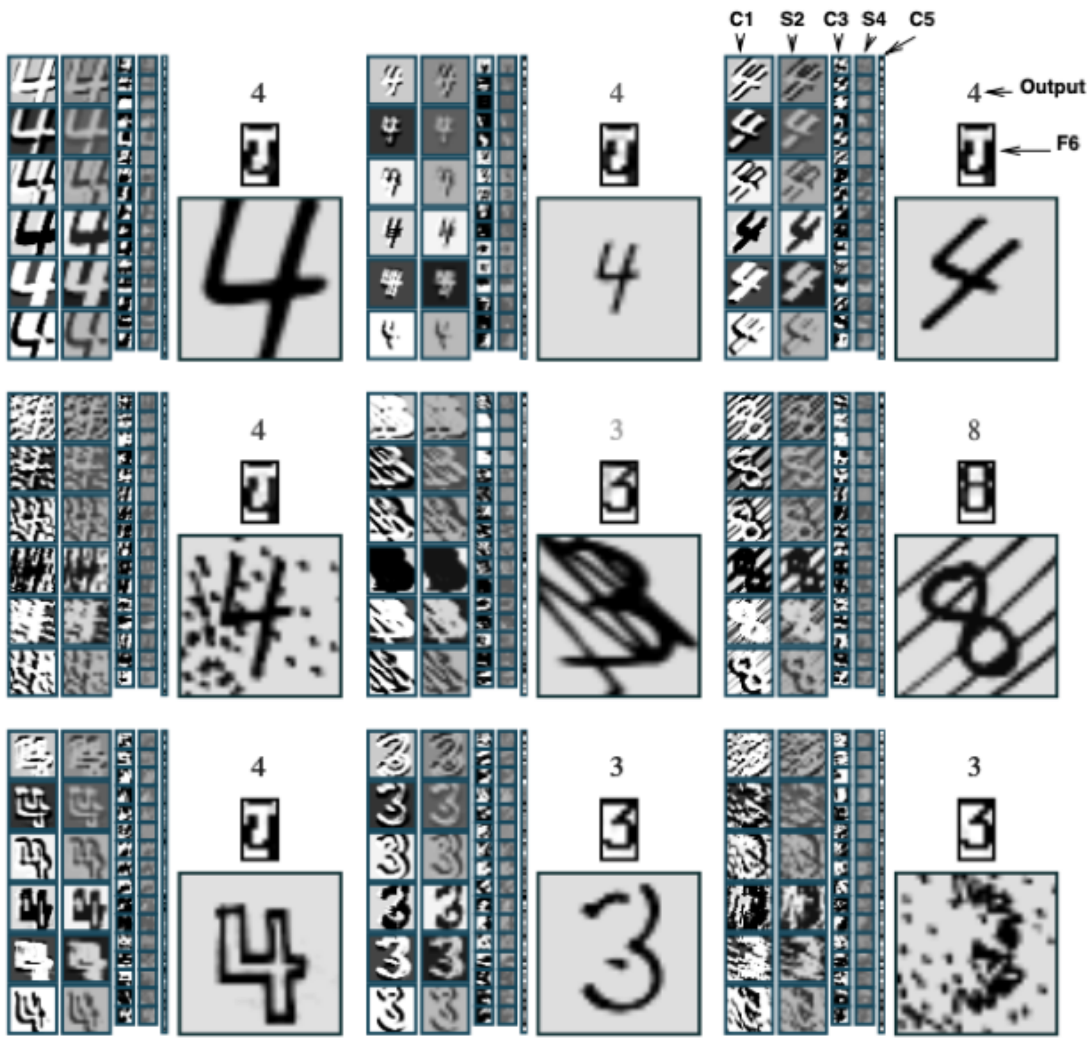
$$P(Y=0|X=\langle X_1,\ldots X_n\rangle)=\frac{\exp(w_0+\sum_i w_i X_i)}{1+\exp(w_0+\sum_i w_i X_i)}$$

- Softmax: when *Y* has *R* values $\{y_1 \ldots y_R\}$, then learn *R* sets of weights to predict *R* output probabilities

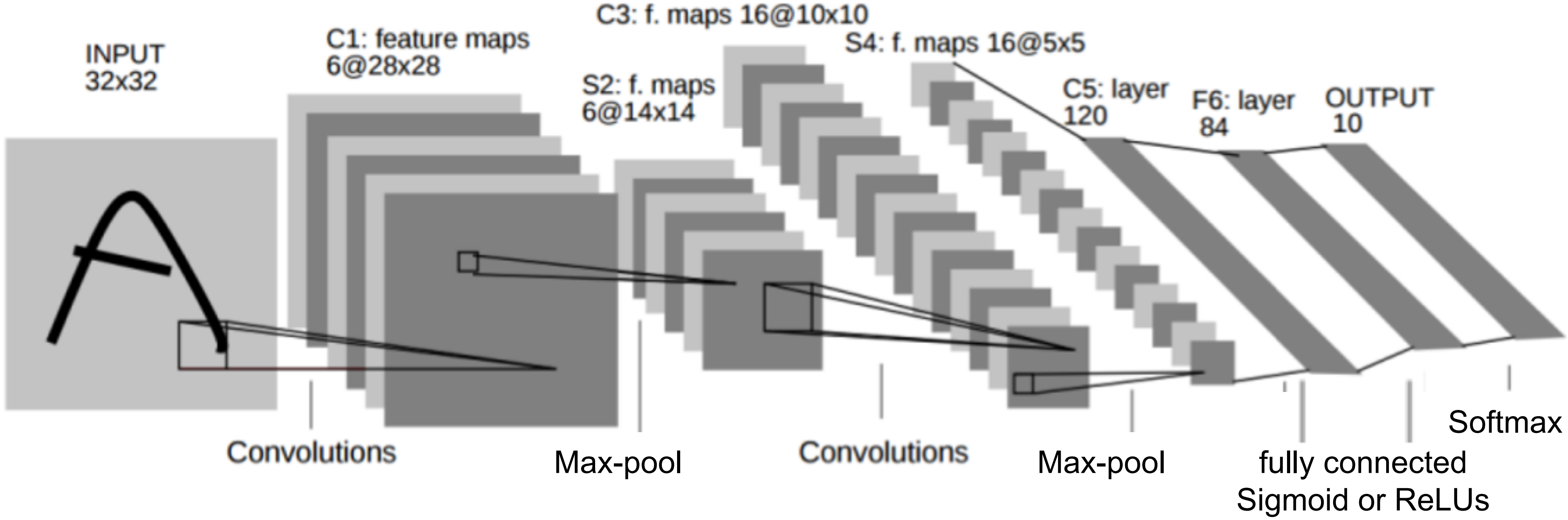
$$P(Y=y_k|X)=\frac{exp(w_{k0}+\sum_i w_{ki}X_i)}{\sum_{j=1}^{R} exp(w_{j0}+\sum_i w_{ji}X_i)}$$

LeNet


INPUT 32x32
C1: feature maps 6@28x28
S2: f. maps 6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer 120
F6: layer 84
OUTPUT 10
Convolutions
Max-pool
Convolutions
Max-pool
fully connected Sigmoid or ReLUs
Soft max


C1
S2
C3
S4
C5
4
4
4
Output
F6
4
3
8
4
3
3

A Convolutional Neural Net for Handwritten Digit recognition: LeNet



- Shrinking size of feature maps
- Multiple channels
- LeNet-5 Demos:
  http://yann.lecun.com/exdb/lenet/index.html
    - Vary scale
    - Vary stroke width
    - Squeeze
    - Noisy-2, Noisy-4

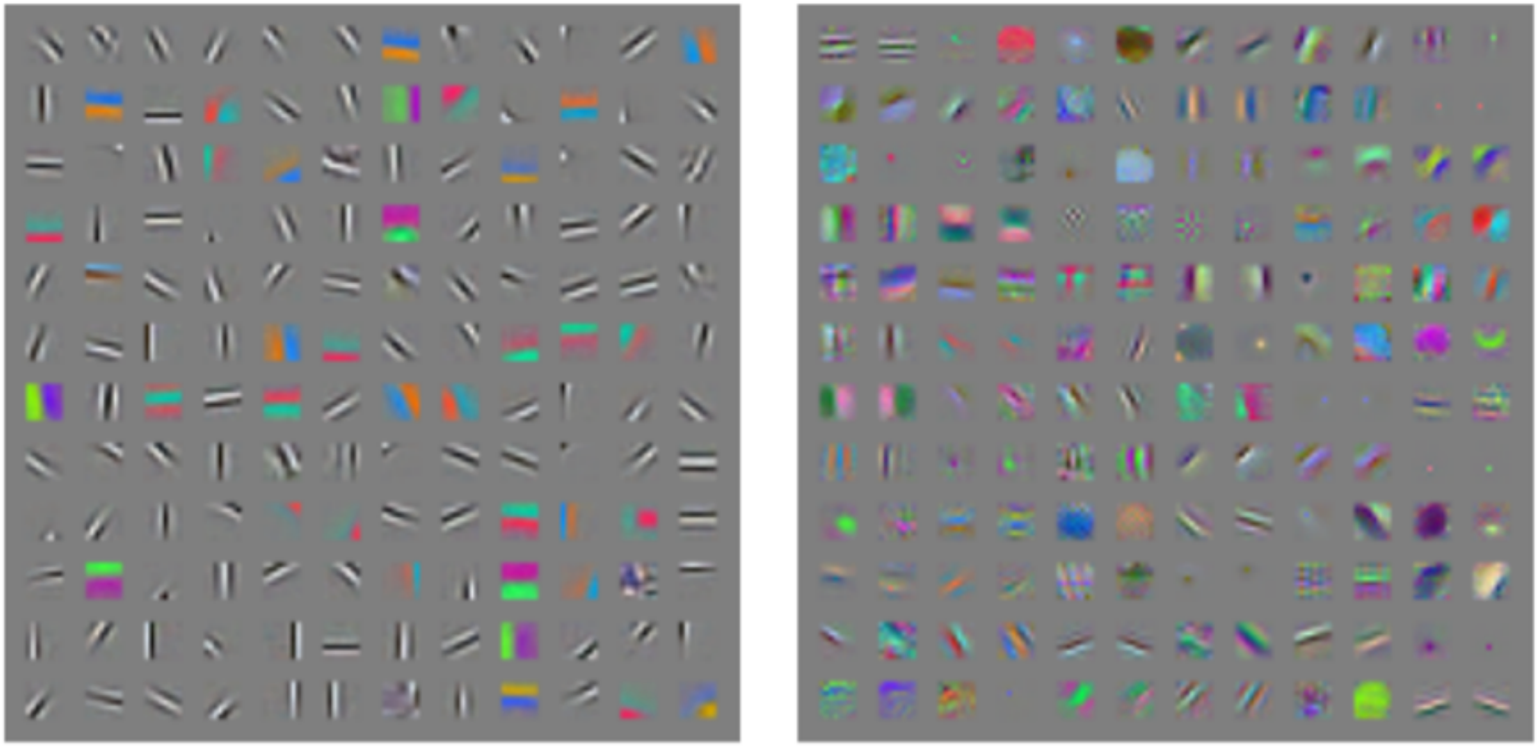


Figure 9.19: Many machine learning algorithms learn features that detect edges or specific colors of edges when applied to natural images. These feature detectors are reminiscent of the Gabor functions known to be present in primary visual cortex. *(Left)*Weights learned by an unsupervised learning algorithm (spike and slab sparse coding) applied to small image patches. *(Right)*Convolution kernels learned by the first layer of a fully supervised convolutional maxout network. Neighboring pairs of filters drive the same maxout unit.

[from Goodfellow et al.]

Recurrent Neural Nets for Sequential Data

Sequences

- Words, Letters

> *50 years ago, the fathers of artificial intelligence convinced everybody that logic was the key to intelligence. Somehow we had to get computers to do logical reasoning. The alternative approach, which they thought was crazy, was to forget logic and try and understand how networks of brain cells learn things. Curiously, two people who rejected the logic based approach to AI were Turing and Von Neumann. If either of them had lived I think things would have turned out differently... now neural networks are everywhere and the crazy approach is winning.*

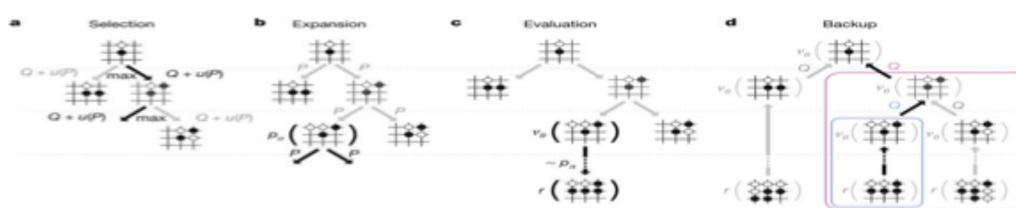
- Speech

- Images, Videos

- Programs

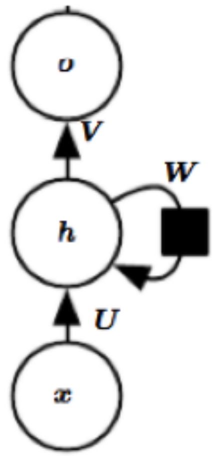
```
while (*d++ = *s++);
```

- Sequential Decision Making (RL)



Recurrent Networks

- Key idea: recurrent network uses (part of) its state at t as input for t+1



$$\mathbf{o_t} = \phi_2(\mathbf{V h_t} + \mathbf{b_o})$$

$$\mathbf{h_t} = \phi_1(\mathbf{U x} + \mathbf{W h_{t-1}} + \mathbf{b_h})$$

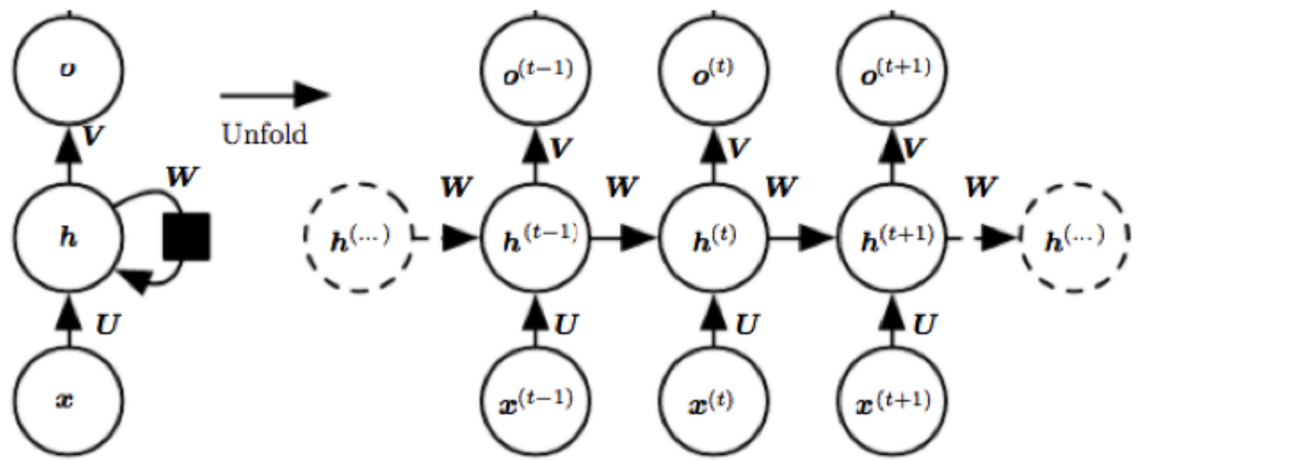
Nonlinearity

Hidden State at previous time step

[Goodfellow et al., 2016]

Recurrent Networks

- Key idea: recurrent network uses (part of) its state at t as input for t+1



$$\mathbf{o_t} = \phi_2(\mathbf{Vh_t} + \mathbf{b_o})$$

$$\mathbf{h_t} = \phi_1(\mathbf{Ux} + \mathbf{Wh_{t-1}} + \mathbf{b_h})$$

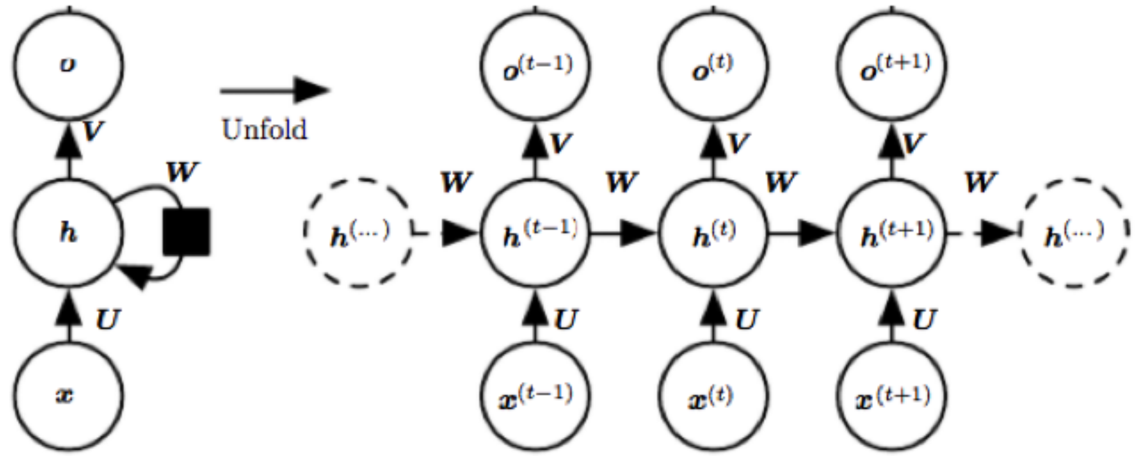
Nonlinearity

Hidden State at previous time step

[Goodfellow et al., 2016]

Recurrent Networks

- Key idea: recurrent network uses (part of) its state at t as input for t+1



$$\mathbf{o_t} = \phi_2(\mathbf{V}\mathbf{h_t} + \mathbf{b_o})$$

$$\mathbf{h_t} = \phi_1(\mathbf{U}\mathbf{x} + \mathbf{W}\mathbf{h_{t-1}} + \mathbf{b_h})$$

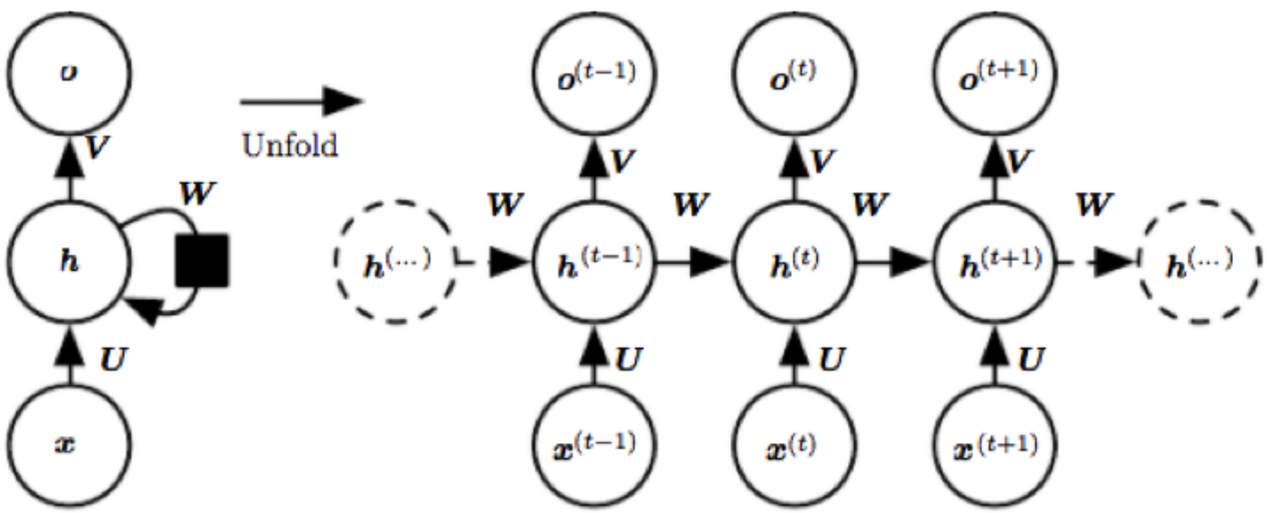
Another example of parameter sharing, like CNNs

[Goodfellow et al., 2016]

Training Recurrent Networks

Key principle for training:

1. Treat as if unfolded in time, resulting in directed acyclic graph
2. Note shared parameters in unfolded net → sum the gradients

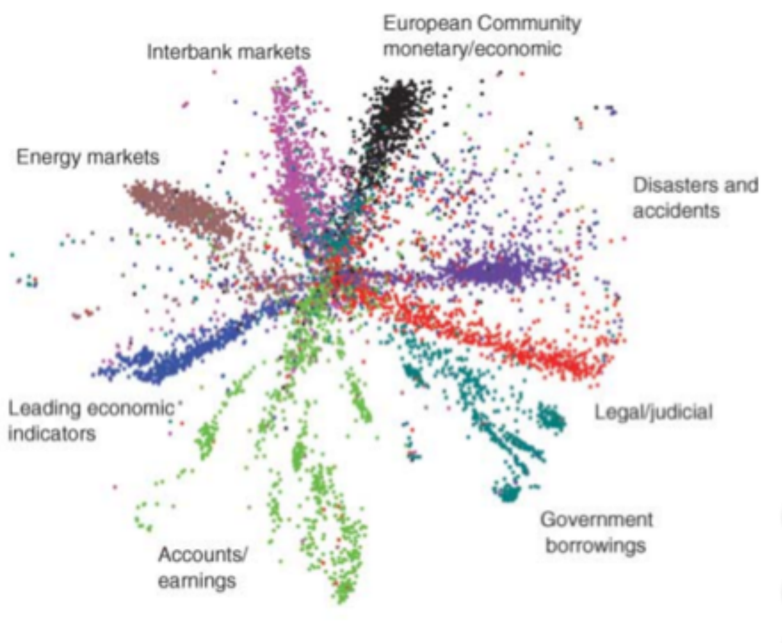


* problem: vanishing and/or exploding gradients

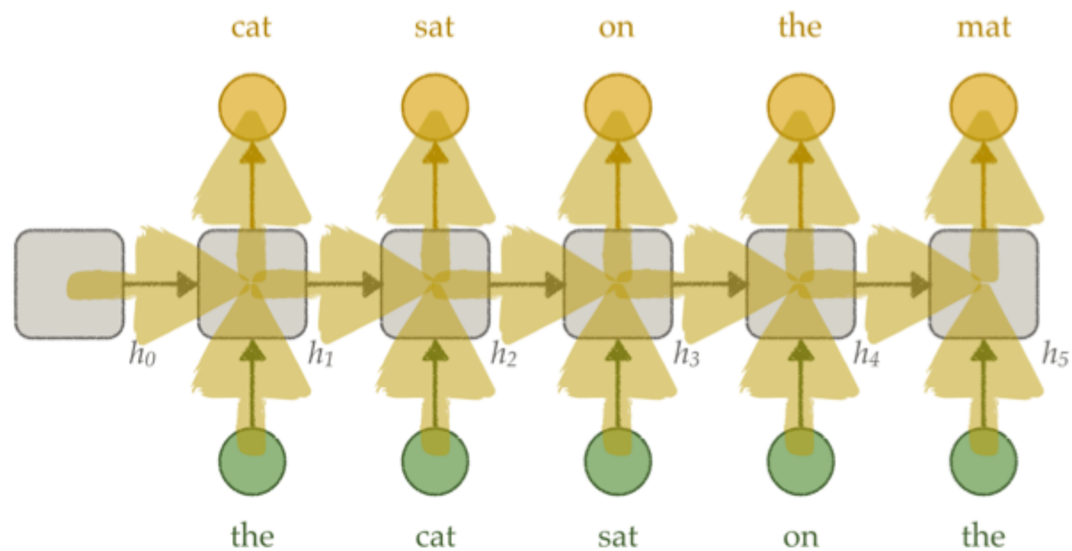
[Goodfellow et al., 2016]

Language model: Two Key Ingredients

Neural Embeddings



Recurrent Language Models



Hinton, G., Salakhutdinov, R. "Reducing the Dimensionality of Data with Neural Networks." *Science (2006)*

Mikolov, T., et al. "Recurrent neural network based language model." *Interspeech (2010)*

Language Models

context					*target*	$P(w_t \mid w_{t-1}, w_{t-2}, \ldots w_{t-5})$
the	cat	sat	on	the	**mat**	0.15
$w_{t\text{-}5}$	$w_{t\text{-}4}$	$w_{t\text{-}3}$	$w_{t\text{-}2}$	$w_{t\text{-}1}$	w_t	
the	cat	sat	on	the	**rug**	0.12
the	cat	sat	on	the	**hat**	0.09
the	cat	sat	on	the	**dog**	0.01
the	cat	sat	on	the	**the**	0
the	cat	sat	on	the	**sat**	0
the	cat	sat	on	the	**robot**	?
the	cat	sat	on	the	**printer**	?

Slide Credit: Piotr Mirowski

What do we Optimize?

$$\theta^* = \arg\max_\theta E_{w\sim data} \log P_\theta(w_1, \ldots, w_T)$$

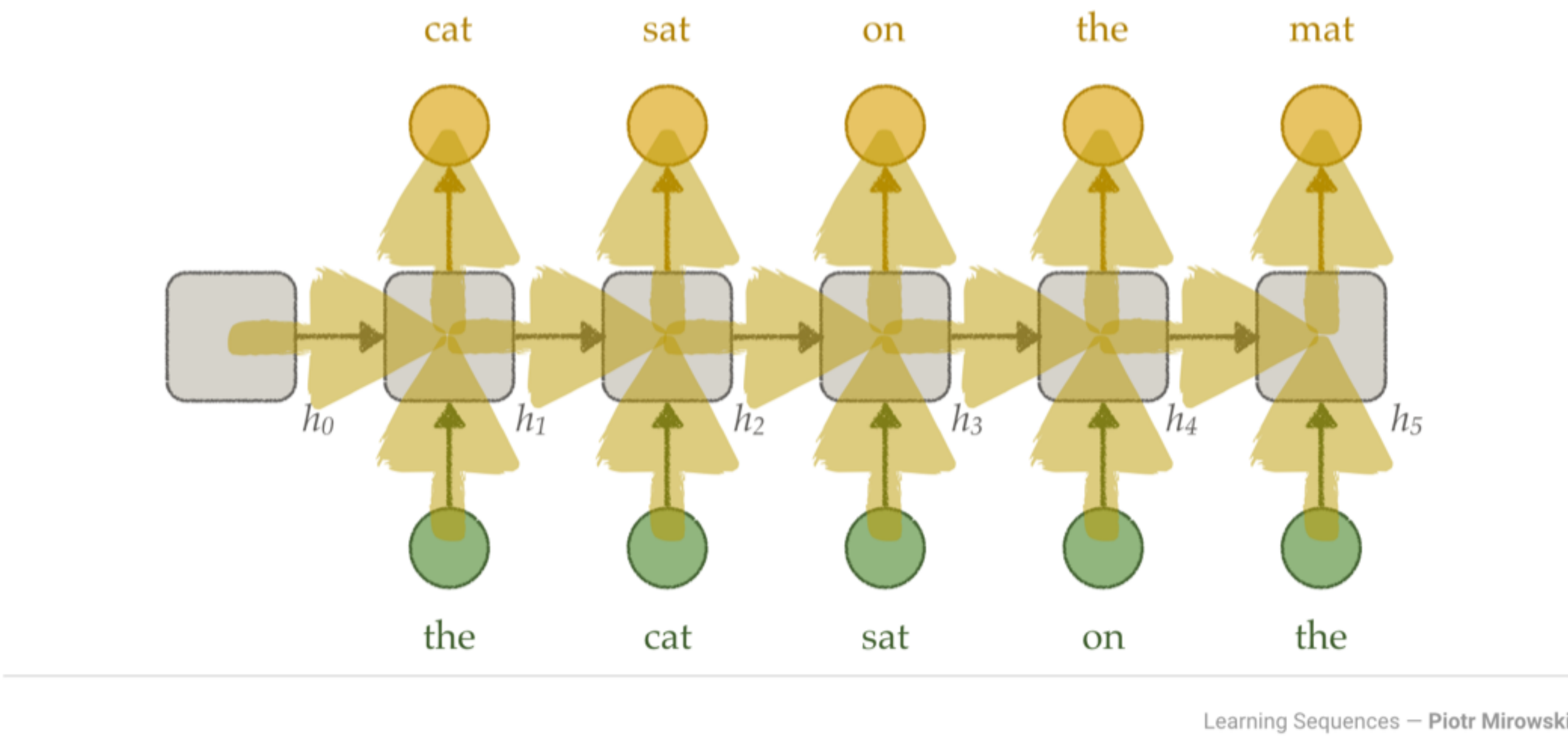
Chain Rule

$$P(w_1, w_2, \ldots, w_{T-1}, w_T) = \prod_{t=1}^{T} P(w_t | w_{t-1}, w_{t-2}, \ldots, w_1)$$

the	cat	sat	on	the	mat	$P(w_1)$
the	**cat**	sat	on	the	mat	$P(w_2 \vert w_1)$
the	cat	**sat**	on	the	mat	$P(w_3 \vert w_2, w_1)$
the	cat	sat	**on**	the	mat	$P(w_4 \vert w_3, w_2, w_1)$
the	cat	sat	on	**the**	mat	$P(w_5 \vert w_4, w_3, w_2, w_1)$
the	cat	sat	on	the	**mat**	$P(w_6 \vert w_5, w_4, w_3, w_2, w_1)$

Slide Credit: Piotr Mirowski

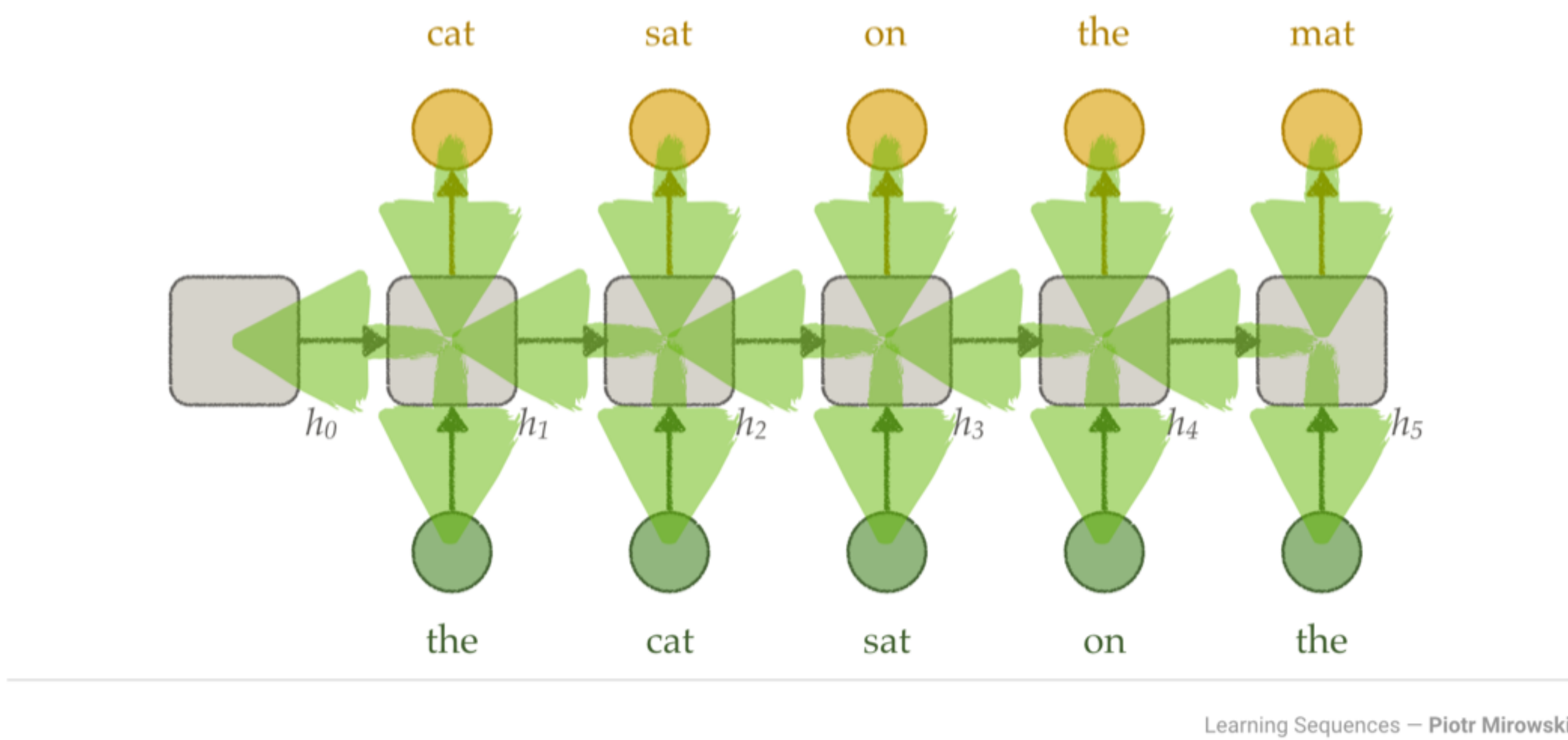
Recurrent Neural Network Language Models



- Forward Pass

Slide Credit: Piotr Mirowski

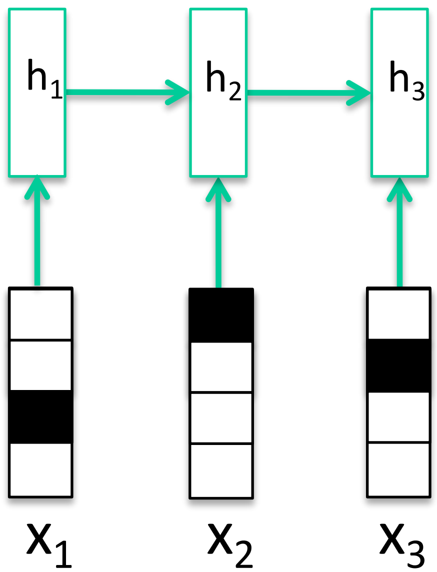
Recurrent Neural Network Language Models



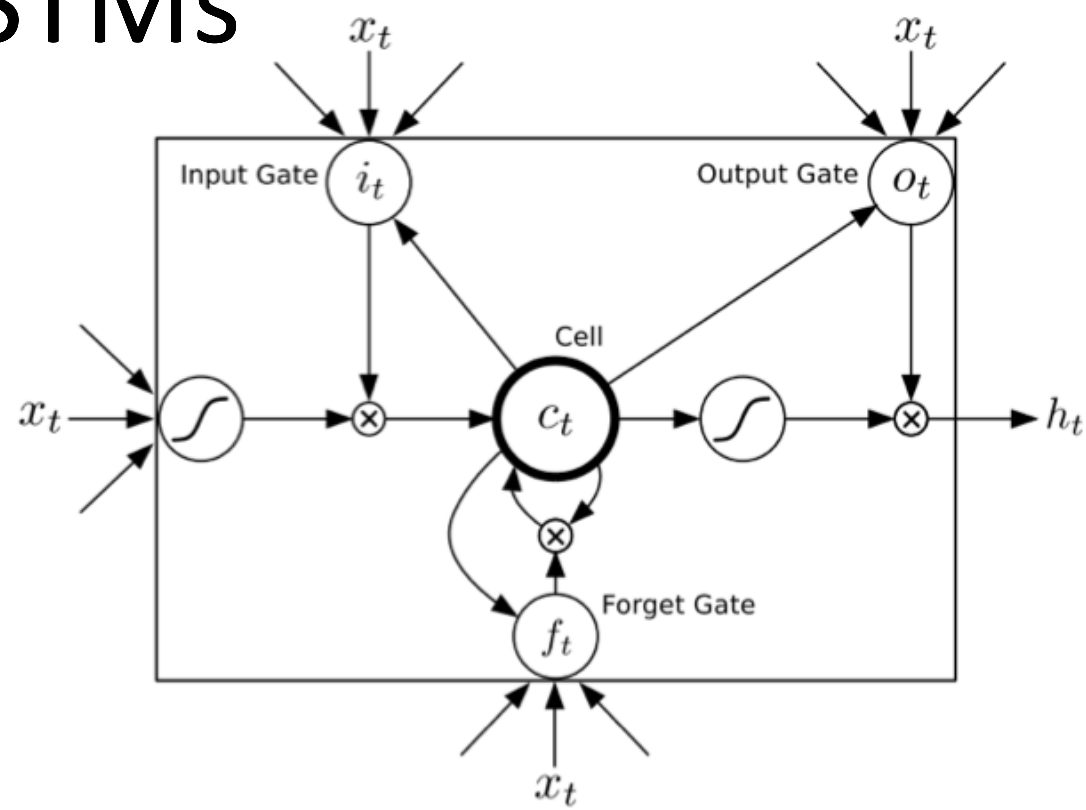
- Backward Pass

Slide Credit: Piotr Mirowski

LSTMs

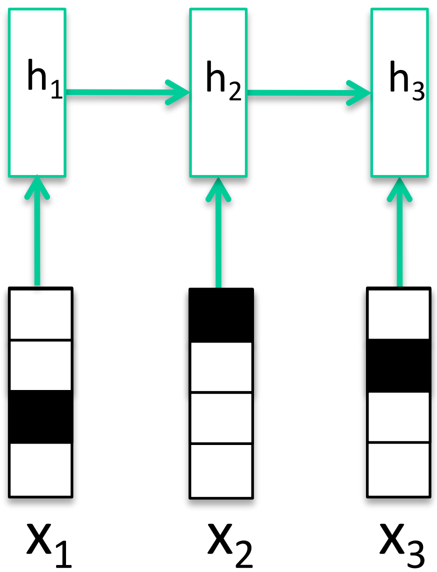


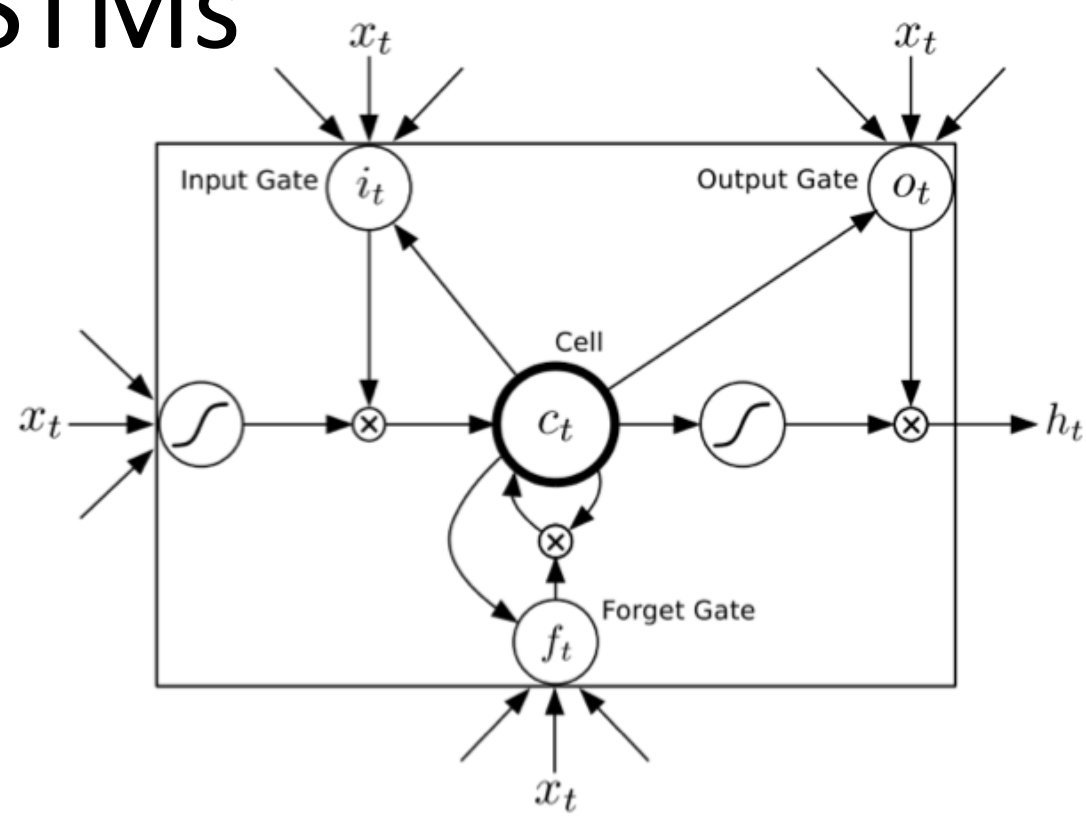
h1
h2
h3
x1
x2
x3



x_t
x_t
Input Gate
i_t
Output Gate
o_t
Cell
x_t
c_t
h_t
Forget Gate
f_t
x_t

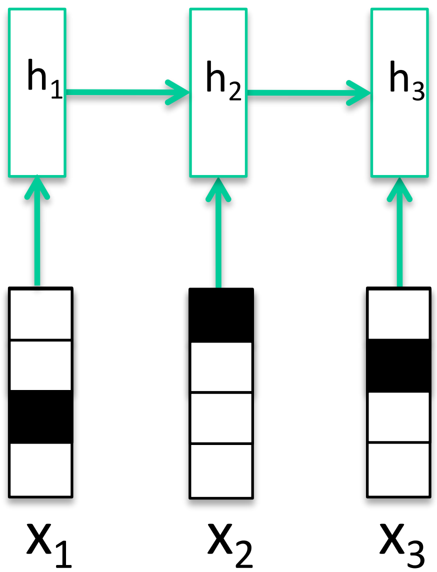
LSTMs

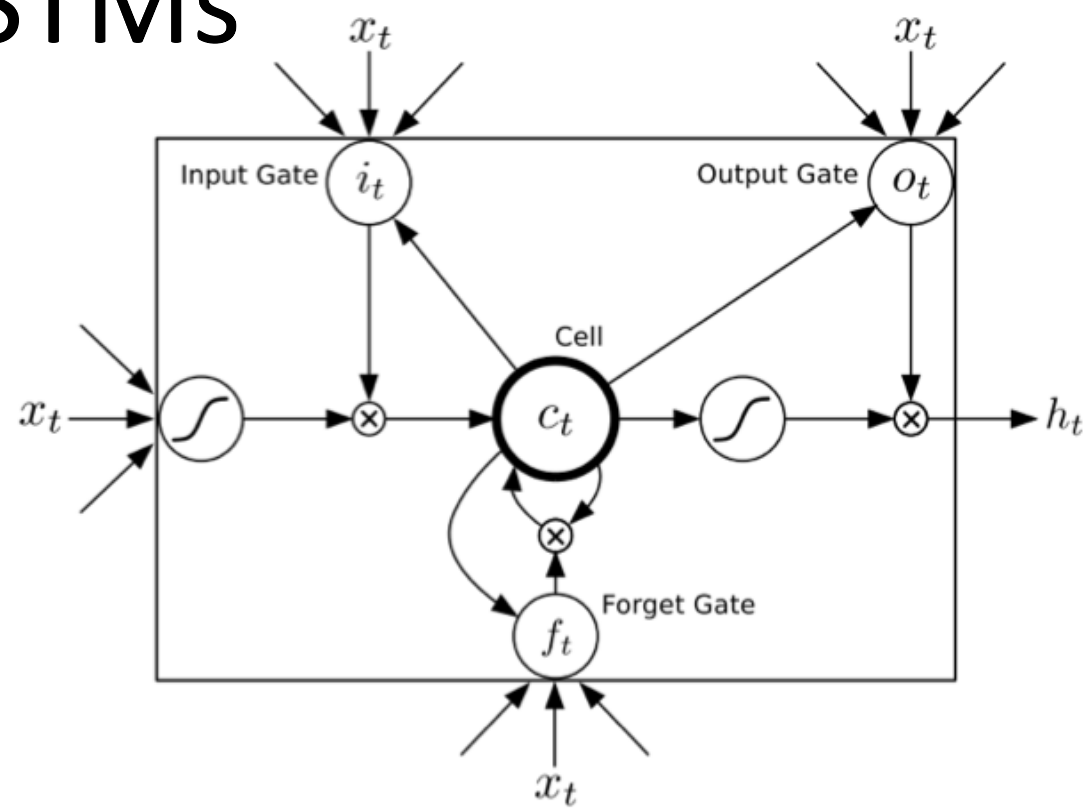




$$\mathbf{i}_t = \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right),$$

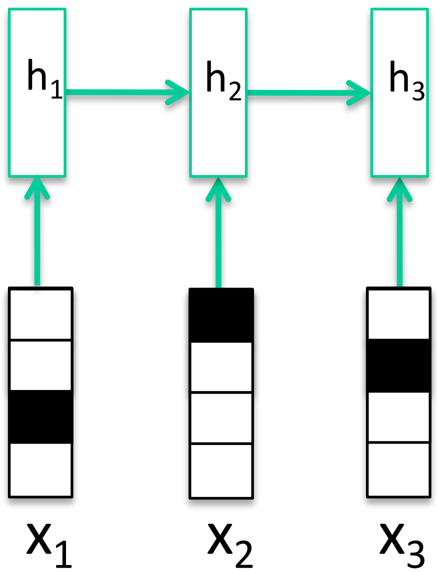
LSTMs

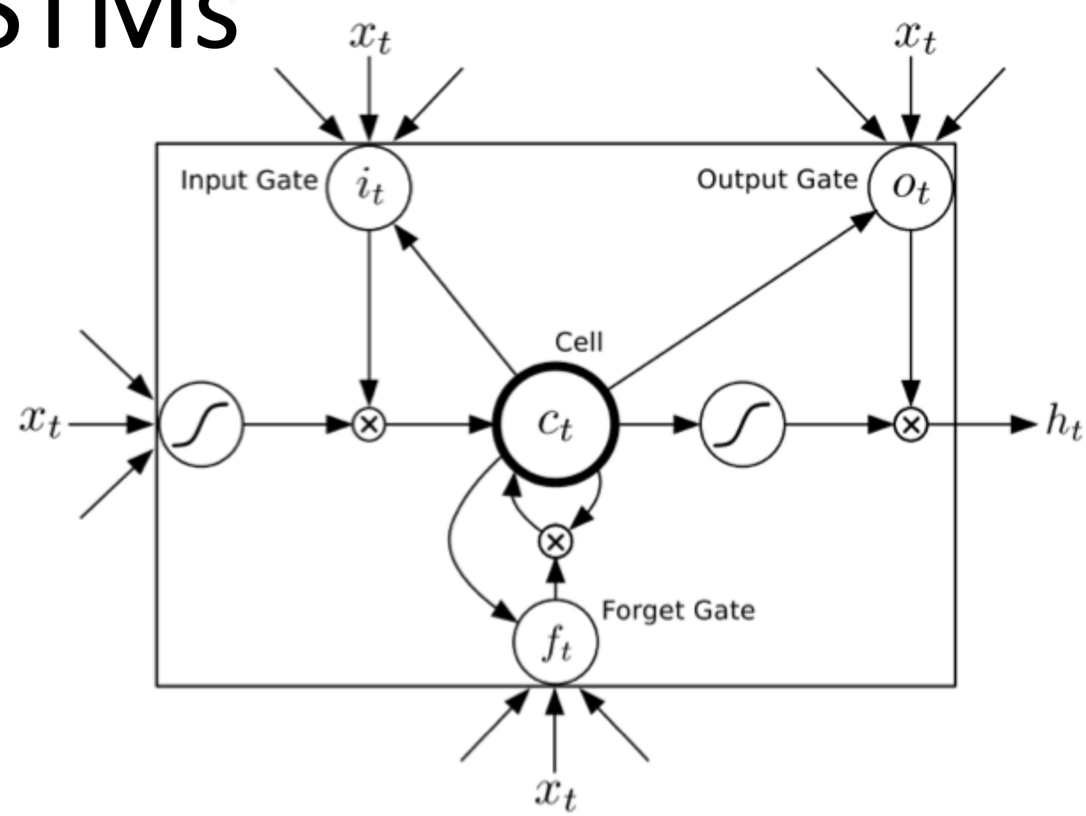




$$\mathbf{i}_t = \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right),$$
$$\mathbf{f}_t = \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right),$$

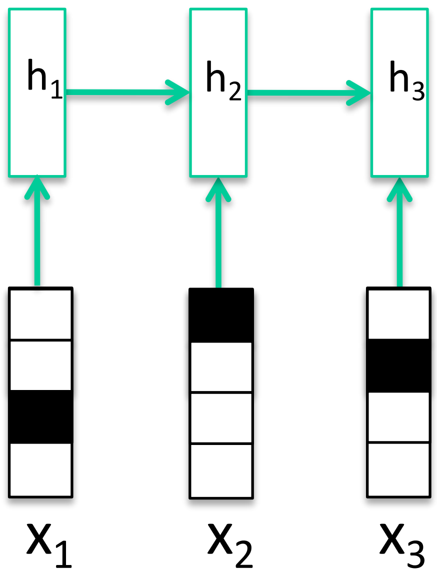
LSTMs

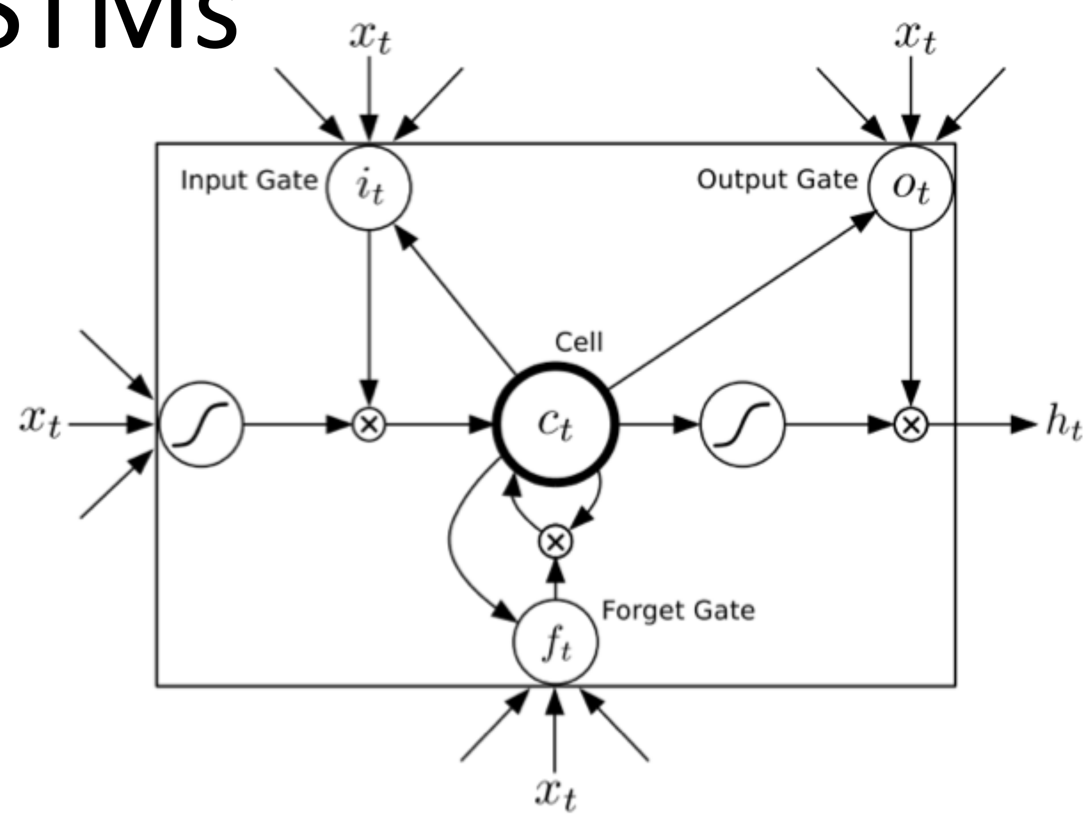




$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right), \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right), \\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\right),
\end{aligned}
$$

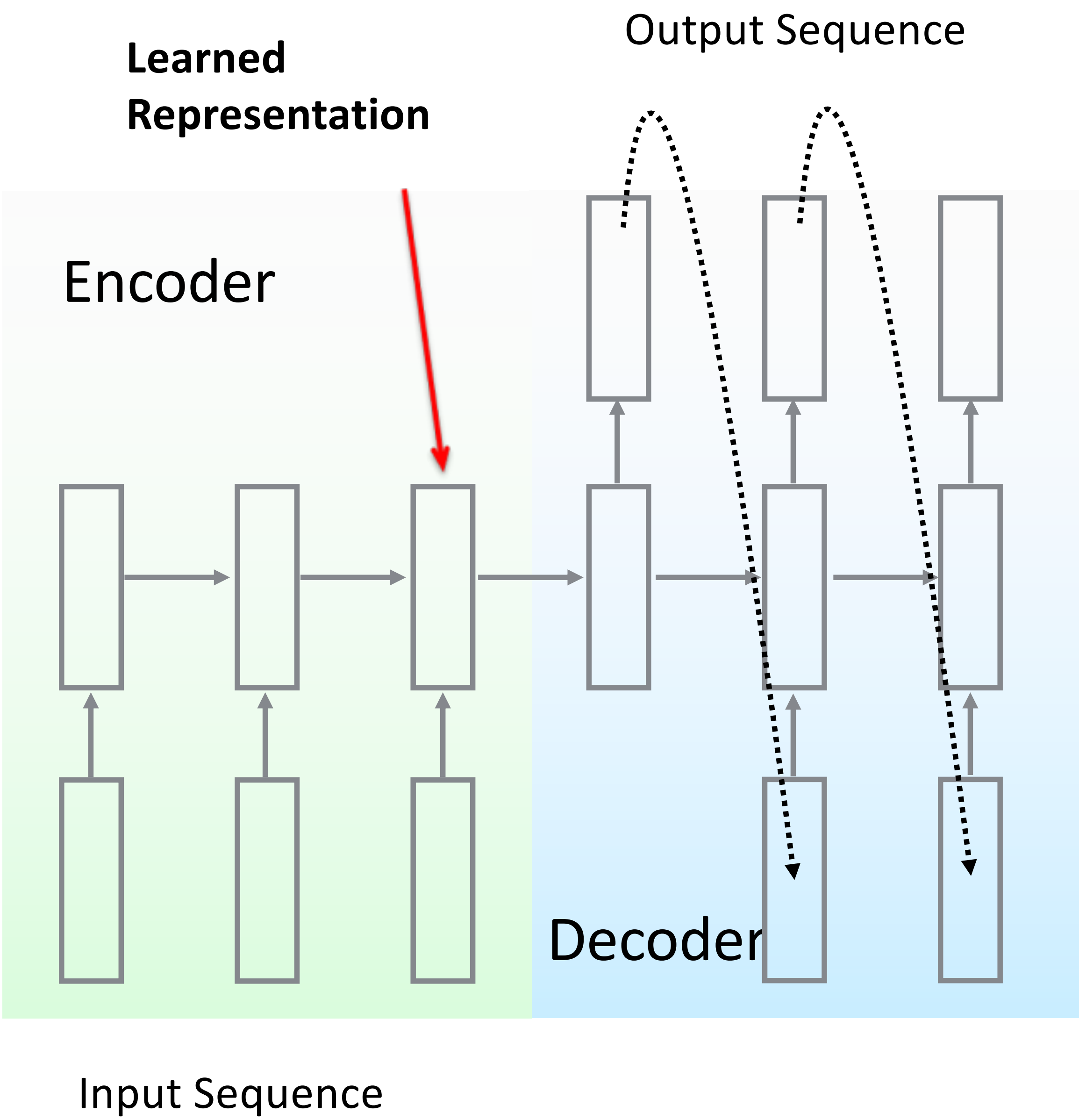
LSTMs





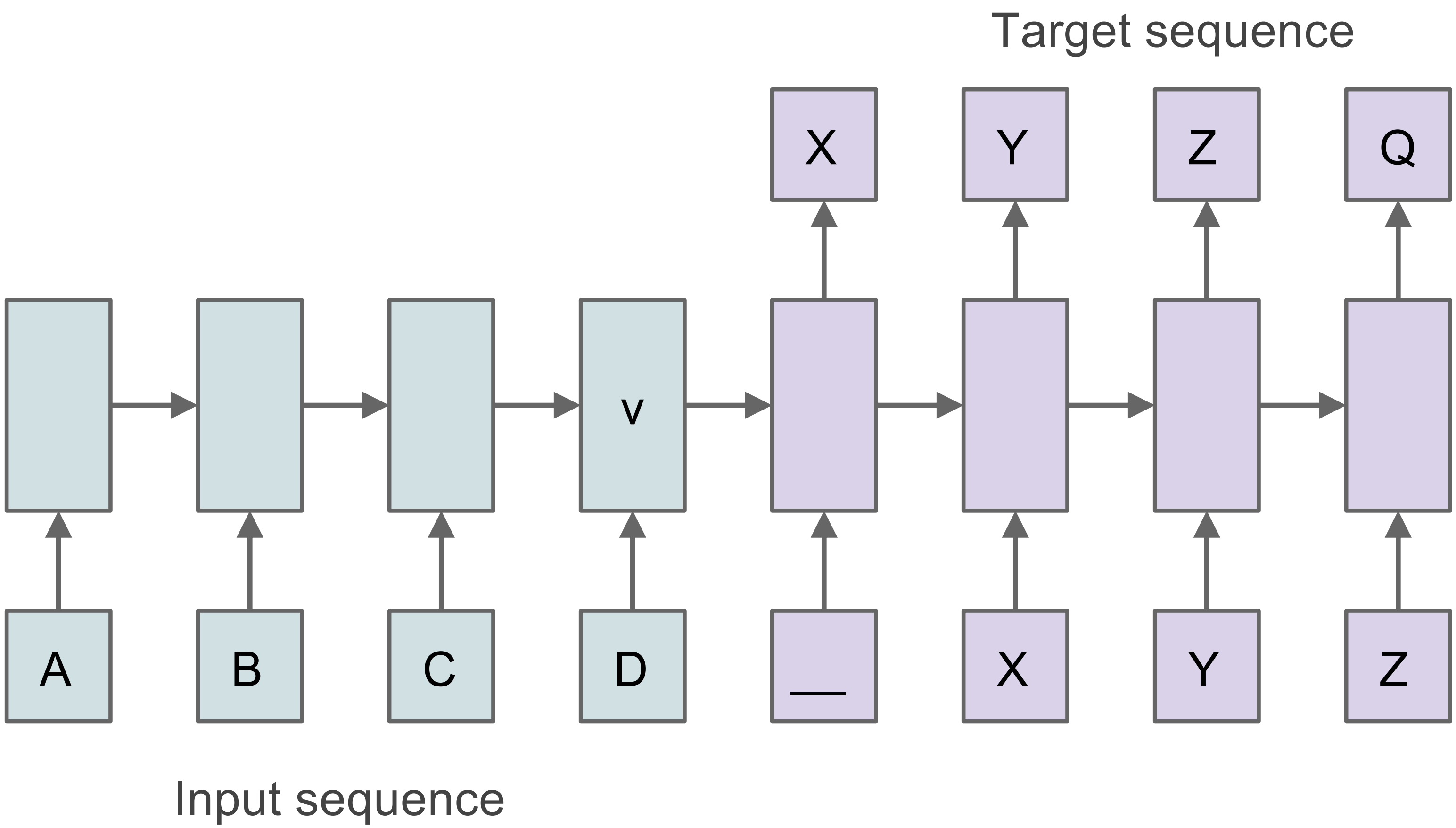
$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right), \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right), \\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\right), \\
\mathbf{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + W_{co}\mathbf{c}_t + \mathbf{b}_o\right), \\
\mathbf{h}_t &= \mathbf{o}_t \tanh(\mathbf{c}_t).
\end{aligned}
$$

Sequence to Sequence Learning



- RNN Encoder-Decoders for Machine Translation (Sutskever et al. 2014; Cho et al. 2014; Kalchbrenner et al. 2013, Srivastava et.al., 2015)

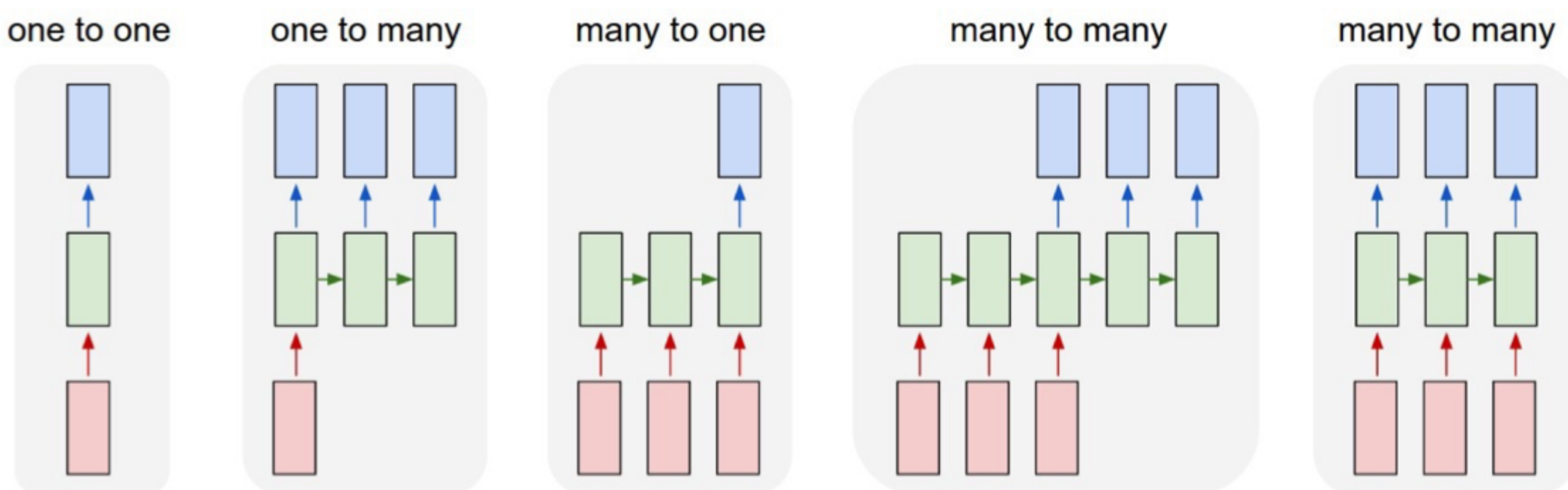
Seq2Seq



$$P(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

Sequence to Sequence Models

- Natural language processing is concerned with tasks involving language data



Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks

What you should know:

- Representation learning
  - Hidden layers re-represent inputs in form allowing out predictions
  - Autoencoders
  - Task-specific encoding (e.g., depend on Y in f: X→Y)
  - Sometimes reused widely (e.g., word2vec)
- Convolutional neural networks
  - Convolution provides translation invariance
  - Network stages with reducing spatial resolution, Mult. channels,
- Recurrent neural networks
  - Learn to represent history in time series
  - Backpropagation as unfolding in time
- Neural architectures
  - Shared parameters across multiple computations
  - Layers with different structures/functions
  - Probabilistic classification → output Softmax layer