Convolutional Neural Networks

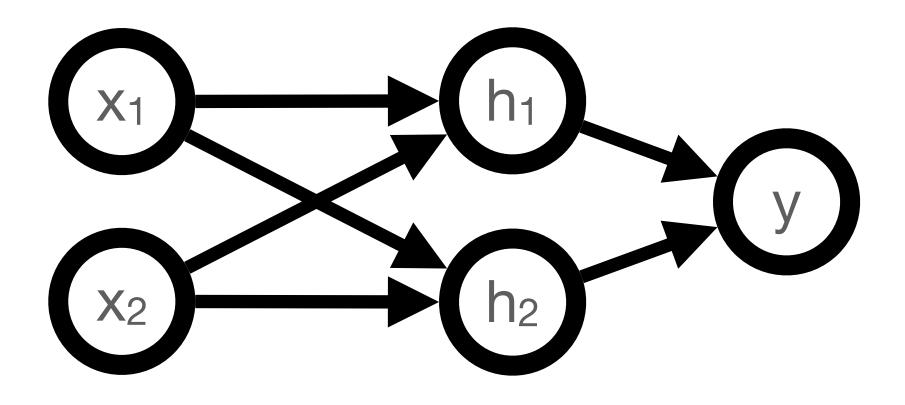
CMPUT 366: Intelligent Systems

GBC §9.0-9.4

Lecture Outline

- 1. Recap
- 2. Neural Networks for Image Recognition
- 3. Convolutional Neural Networks

Recap: Feedforward Neural Network



- A neural network is many units composed together
- Feedforward neural network: Units arranged into layers
 - Each layer takes outputs of previous layer as its inputs

$$h_1(\mathbf{x}; \mathbf{w}^{(1)}, b^{(1)}) = g\left(b^{(1)} + \sum_{i=1}^n w_i^{(1)} x_i\right)$$

$$y(\mathbf{x}; \mathbf{w}, \mathbf{b}) = g\left(b^{(y)} + \sum_{i=1}^{n} w_i^{(y)} h_i(\mathbf{x}_i; \mathbf{w}^{(i)}, b^{(i)})\right)$$
$$= g\left(b^{(y)} + \sum_{i=1}^{n} w_i^{(y)} g\left(b^{(i)} + \sum_{j=1}^{n} w_j^{(i)} x_j\right)\right)$$

Recap: Training Neural Networks

- Specify a loss L and a set of training examples: $E = (\mathbf{x}^{(1)}, \mathbf{y}^{(1)})$
- Training by gradient descent:
 - Compute loss on training da
 - 2. Compute **gradient** of loss:
 - 3. Update parameters to make loss smaller:

$$(1)), \ldots, (\mathbf{x}^{(n)}, y^{(n)})$$

ata:
$$L(\mathbf{W}, \mathbf{b}) = \sum_{i} \mathscr{C}\left(f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b}), y^{(i)}\right)$$

i Prediction Target

$\nabla L(\mathbf{W}, \mathbf{b})$

$$\begin{bmatrix} \mathbf{W}^{new} \\ \mathbf{b}^{new} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^{old} \\ \mathbf{b}^{old} \end{bmatrix} - \eta \, \nabla L(\mathbf{W}^{old}, \mathbf{b}^{old})$$

Recap: Automatic Differentiation

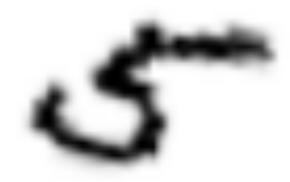
- Forward mode sweeps through the gr
 - The numerator varies, and the denominator is fixed
 - At the end, we have computed S'_n
- **Backward mode** does the opposite: ullet
 - For each s_i , computes the local gr
 - The numerator is fixed, and the denominator varies
 - At the end, we have computed $\overline{x_i}$ =
- Key point: The intermediate results are computed numerically at each step

raph, computing
$$s'_i = \frac{\partial s_i}{\partial s_1}$$
 for each s_i

$$= \frac{\partial s_n}{\partial x_i}$$
 for a **single** input x_i

radient
$$\overline{s_i} = \frac{\partial s_n}{\partial s_i}$$

$$= \frac{\partial s_n}{\partial x_i}$$
 for each input x_i



Problem: Recognize the handwritten digit from an image

- What are the **inputs**?
- What are the **outputs**? •
- What is the **loss**? \bullet

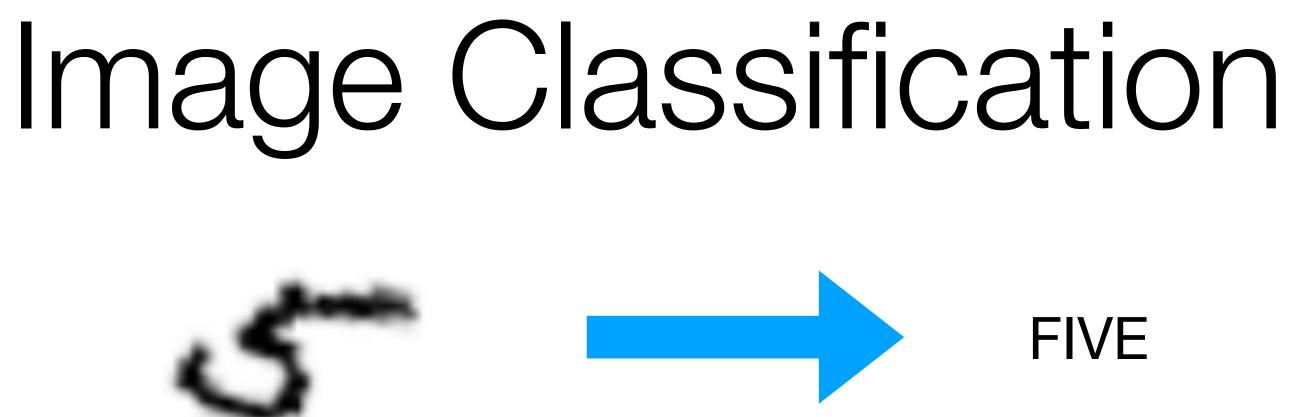


Image Classification with Neural Networks

How can we use a **neural network** to solve this problem?

- How to represent the inputs?
- How to represent the **outputs**?
- What are the **parameters**?
- What is the loss?

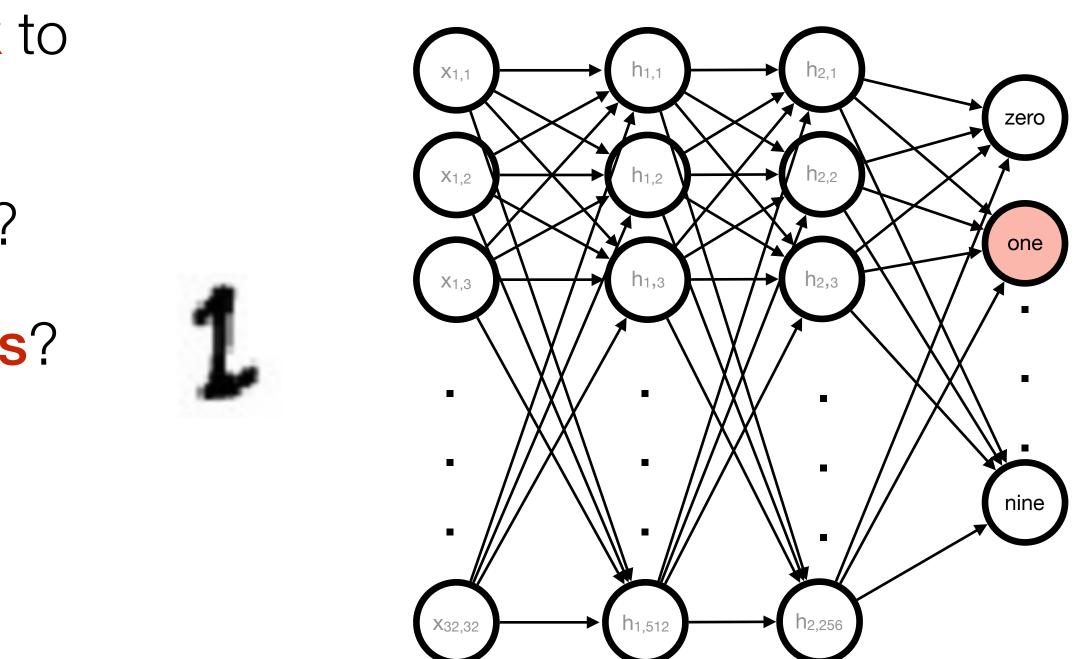


Image Recognition Issues

- For a large image, the number of parameters will be very large
 - For 32x32 greyscale image, hidden layer of 512 units hidden layer of 256 units, $1024 \times 512 + 512 \times 256 + 256 \times 10$ = 657,920 weights (and 1802 offsets)
 - Needs lots of data to train
- Want to generalize over transformations of the input

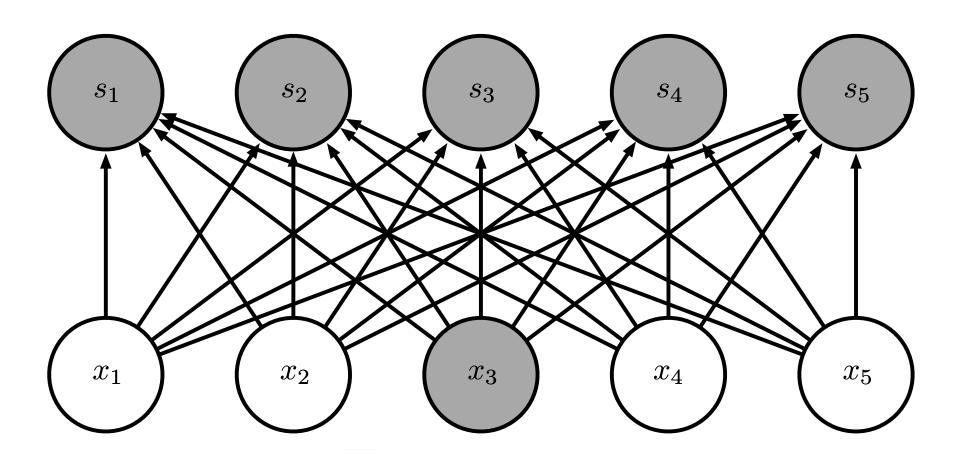
zero one nine

- Introduce two new operations:
 - 1. Convolutions
 - 2. Pooling
- Efficient **learning** via:
 - 1. Sparse interactions
 - 2. Parameter sharing
 - 3. Equivariant representations

Convolutional Neural Networks

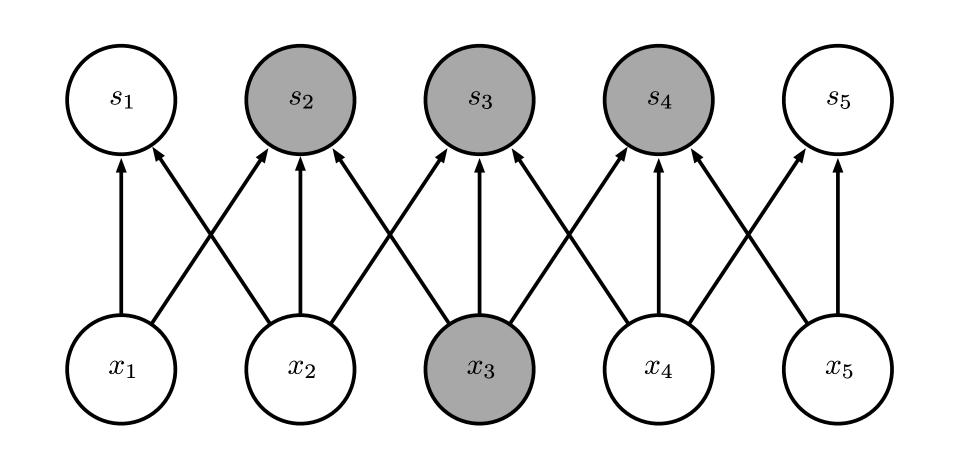
Convolutional neural networks: a specialized architecture for image recognition





Dense connections

Sparse connections



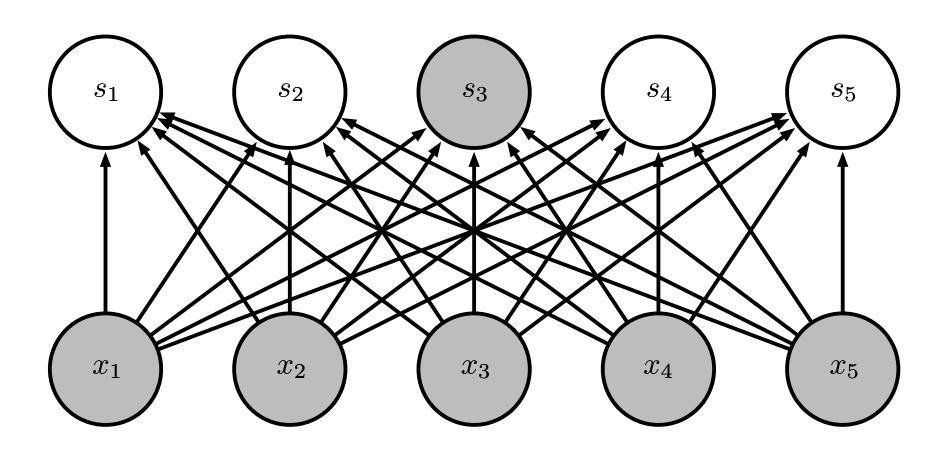
 S_1

Sparse Interactions

(Images: Goodfellow 2016)

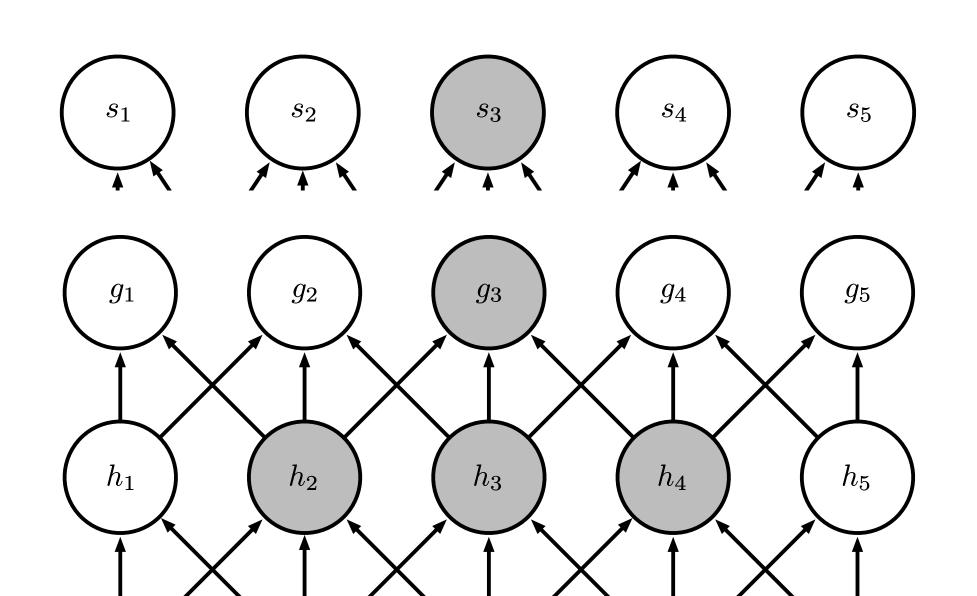
S5





Dense connections

Sparse connections

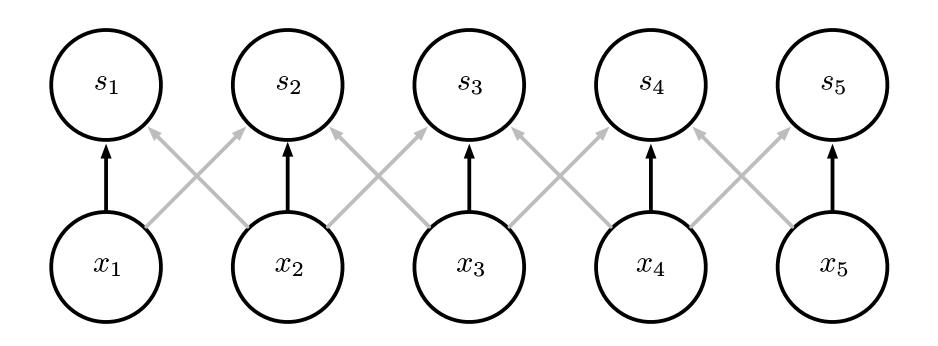


Sparse Interactions

nages: Goodfellow 2016)

Traditional neural nets learn a unique value for each connection

Convolutional neural nets constrain multiple parameters to be equal



 s_3

 s_1

 s_2

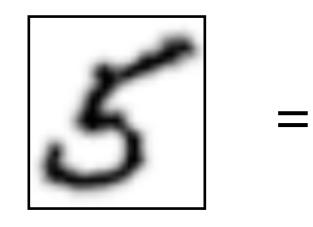
 s_4

(Images: Goodfellow 2016)

Equivariant Representations

- We want to be able to recognize transformed versions of inputs we have seen before:
 - Translation (moved)
 - Rotation \bullet
- Without having been trained on all transformed versions





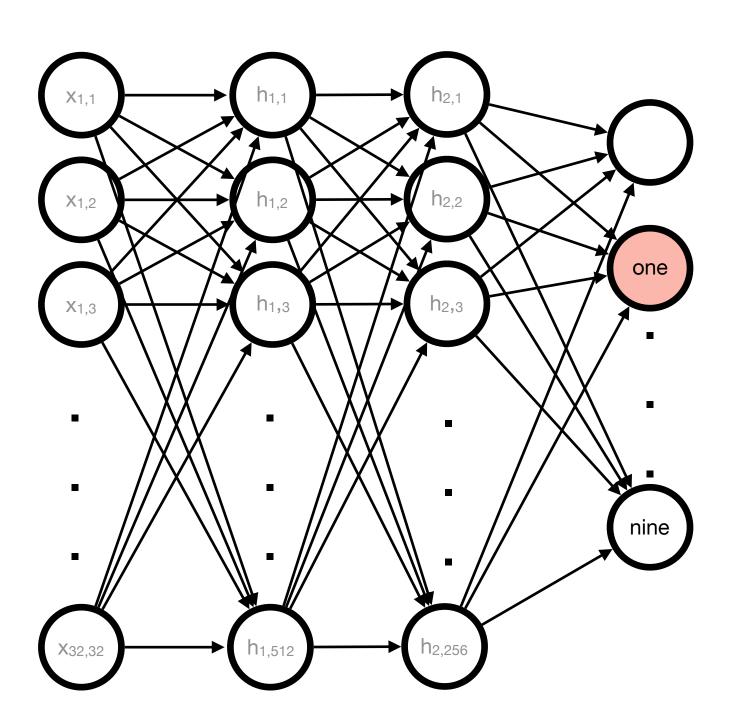


Multiplying Matrices and Vectors tiplying Matrices and Vectors

the most important operations involving matrices is multiplication of two sstTiheportant poperations involving matrices in this triat control of the TIX Product of the Pro CALIS DE STATTER, Annard Bais Charappe number of Company B Kas of shape matrix and Bet just hy placing two per more matrices together. the matrix product just by placing two or more matrices together, RecalPthat we can represent the4)

product operation is defined by a product operation is defined by a act operation is define $d_{i,j} = \sum_{k,k} A_{i,k} B_{k,j}$. (2.5)

 $C_{i,j} = \sum_{k \in I} A_{i,k} B_{k,j}.$ (2.5) e that the standard product of two matrices is *not* just a matrix containing duct of the individual elements. Such an operation exists and is called the this standard opr Hadamard product, and is donoted as A fr R. c f ddtep individue a two vectors & and peof. Re same dimensishality is the product or $\mathcal{Y}_{Ia} \otimes \mathcal{W}_{e}$ can think of the matrix enotiate $C \doteq AB$ as computing the dot product between row i of A and column j of B. Foduct between two vectors \tilde{x} and \tilde{y} of the same dimensionality is the pct $x^{\top}y$. We can think of the matrix product $C^{n} \leftarrow AB$ as computing match t product between row i of A and column j of B.

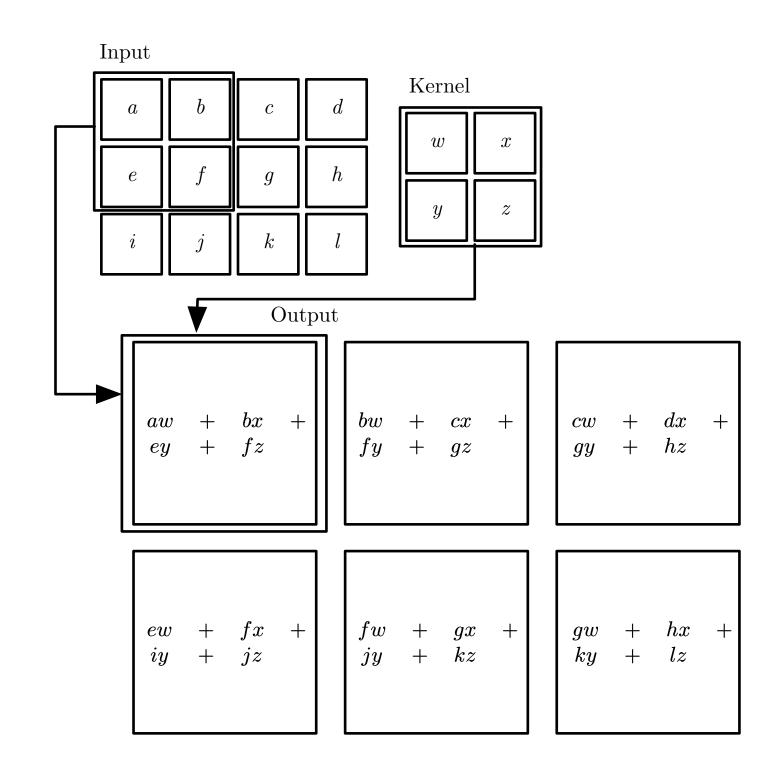


 $\mathbf{h_1} = g_h \left(W^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right)$ $\mathbf{h_2} = g_h \left(W^{(2)} \mathbf{h_1} + \mathbf{b}^{(2)} \right)$ $\mathbf{y} = g_{v} \left(W^{(3)} \mathbf{h}_{2} + \mathbf{b}^{(3)} \right)$

Operation: 2D Convolution

Convolution scans a small block of weights (called the **kernel**) over the elements of the inputs, taking weighted averages

- Note that input and output dimensions need not match
- Same weights used for very many combinations



(Image: Goodfellow 2016)

Replace Matrix Multiplication by Convolution

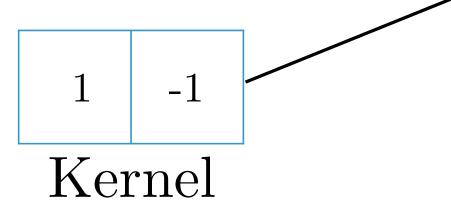
Main idea: Replace matrix multiplications with convolutions

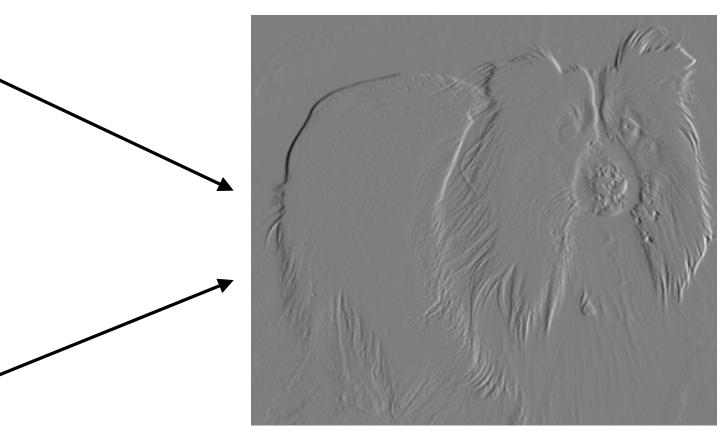
- **Sparsity:** Inputs only combined with **neighbours**
- Parameter sharing: Same kernel used for entire input

Example: Edge Detection



Input





Output

(Image: Goodfellow 2016)

Efficiency of Convolution

Input size: 320 by 280 Kernel size: 2 by 1 Output size: 319 by 280

Dense matrix

Stored floats 31

319*280*320*2 0 > 8e9

Float muls or adds

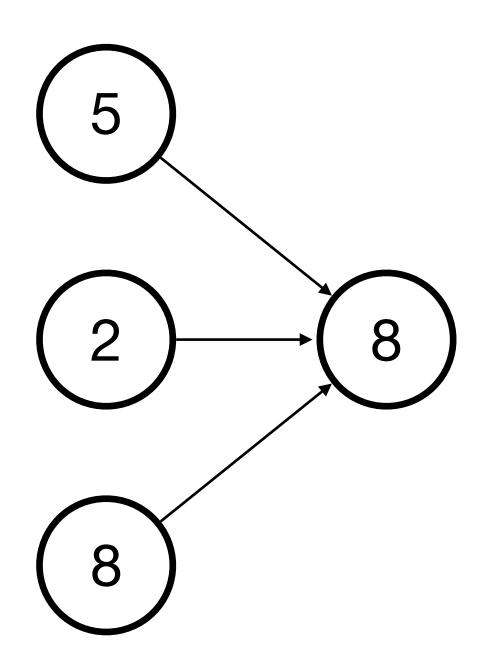
> 16e9

X	Sparse matrix	Convolution
28	2*319*280 = 178,640	2
	Same as convolution (267,960)	319*280*3 = 267,960

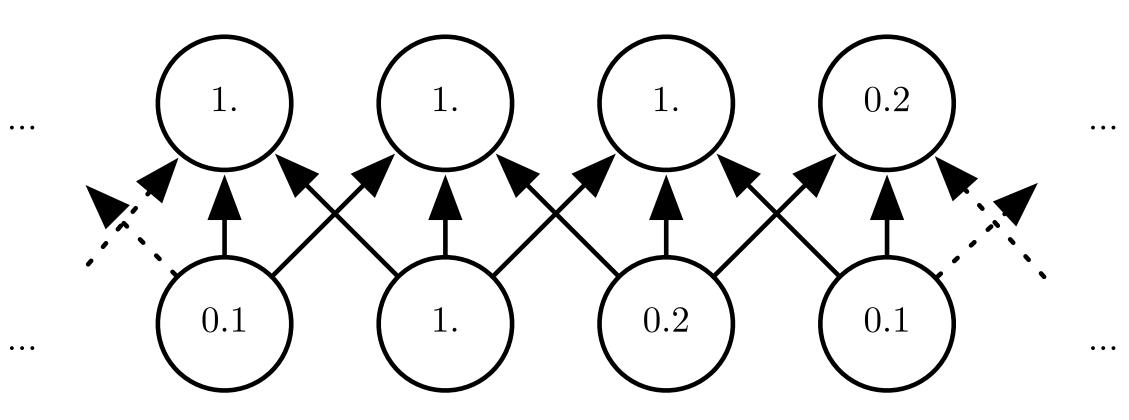
Operation: Pooling

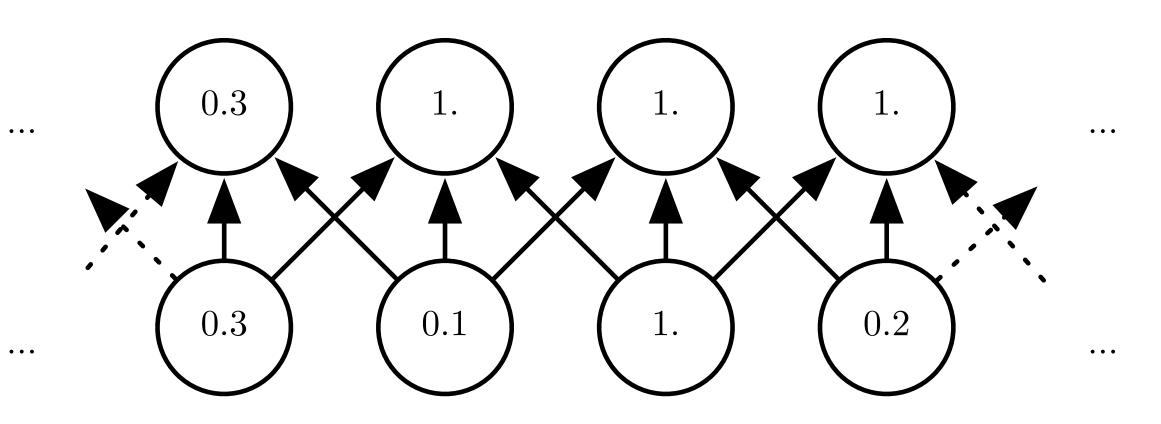
- Pooling **summarizes** its inputs into a single value, e.g.,
 - max
 - average
- Max-pooling is **parameter-free** (no bias or edge weights)





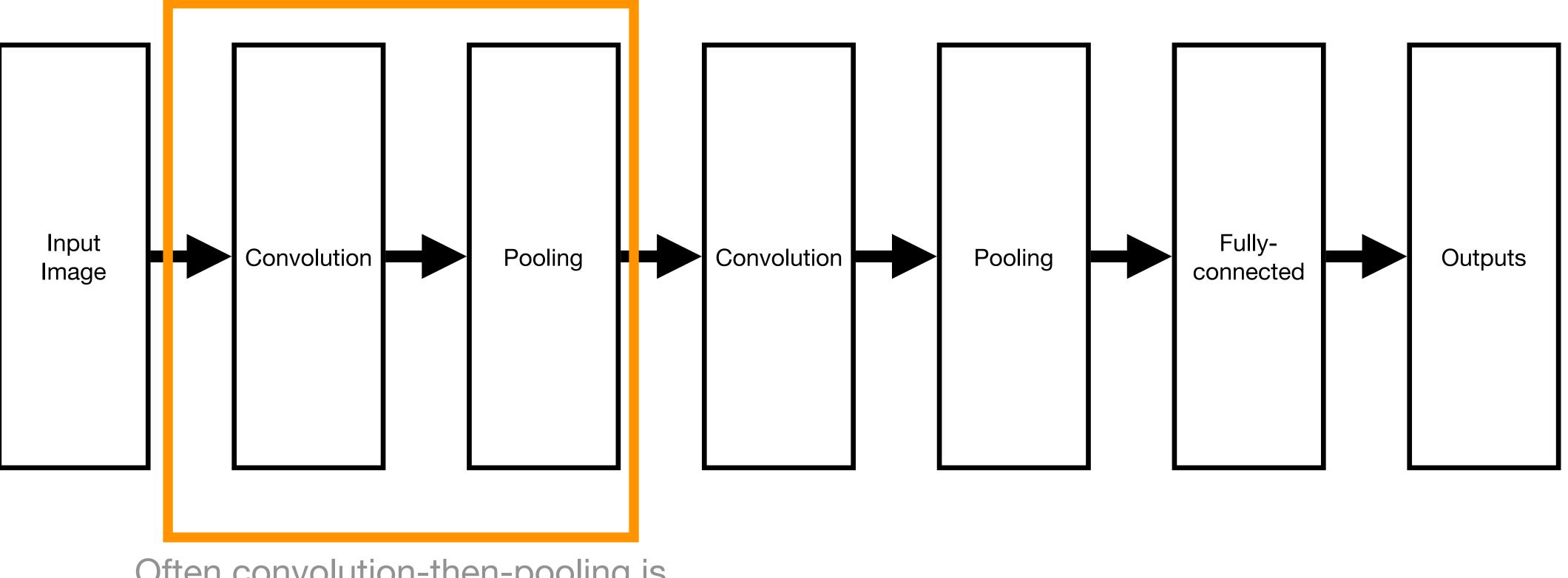
Example: Translation Invariance





(Goodfellow 2016)

Typical Architecture



Often convolution-then-pooling is collectively referred to as a "convolution layer"

Summary

- \bullet quantities of **parameters** (and hence **data**)
- Convolutional networks add **pooling** and **convolution** \bullet
 - Sparse connectivity
 - Parameter sharing
 - Translation equivariance
- Fewer parameters means far more efficient to train

Classifying images with a standard feedforward network requires vast