Convolutional Neural Networks
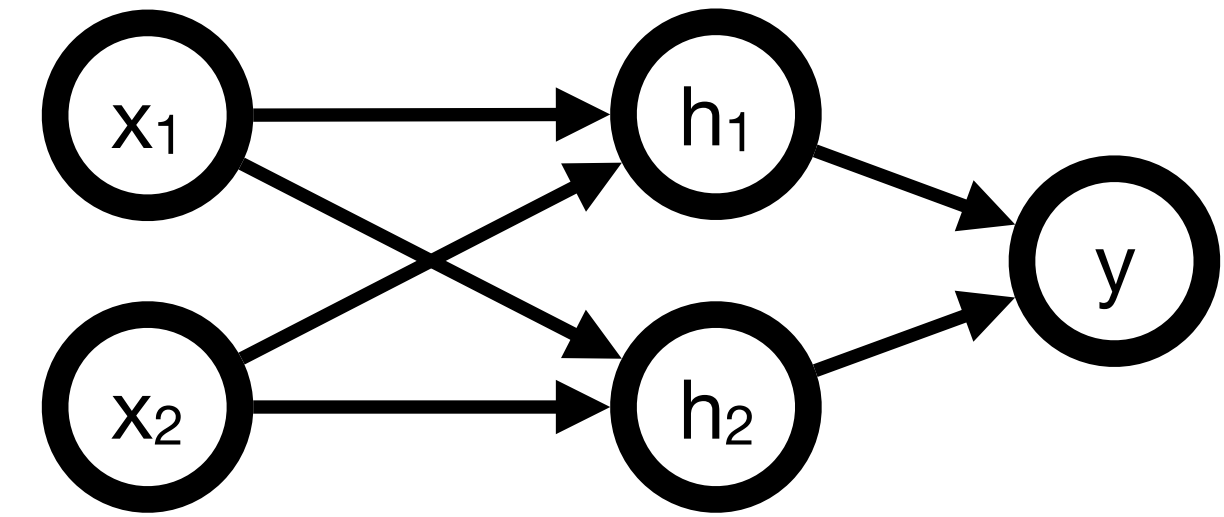
CMPUT 366: Intelligent Systems

GBC §9.0-9.4

Lecture Outline

1. Recap

2. Training Neural Networks

3. Image Recognition

4. Convolutional Neural Networks

Recap: Neural Networks

- Neural networks are **compositions** of **units**

  - Each unit's inputs are outputs from **previous layer's** units

- **Single unit** h: Inputs $\mathbf{x}$, weights $\mathbf{w}$, bias b, activation g
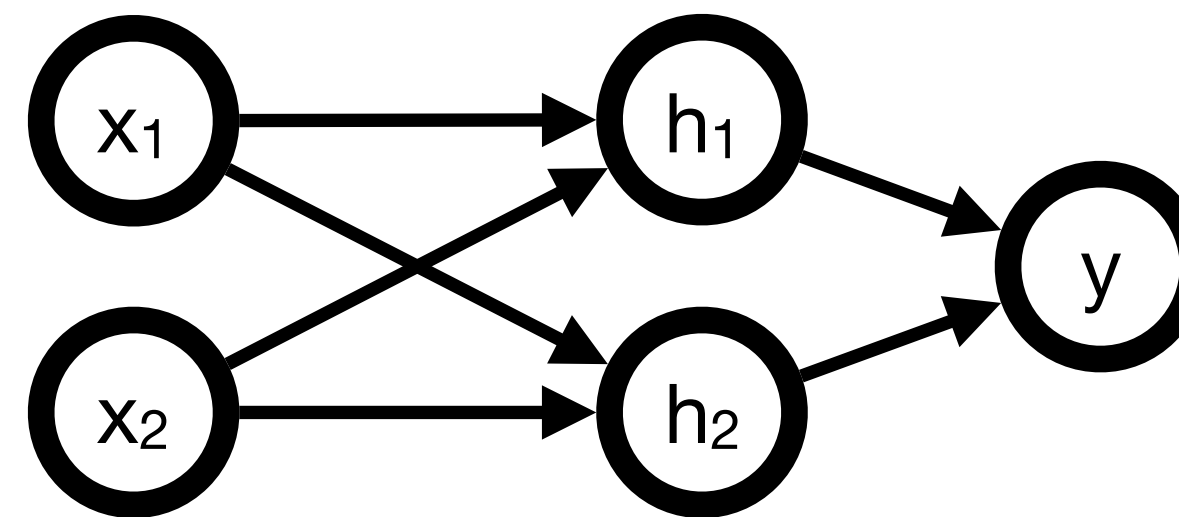
$$h(\mathbf{x}; \mathbf{w}, b) = g(\mathbf{w}^\top\mathbf{x} + b)$$

  - Weights and biases are **parameters** of the model, set during **training**

- Neural networks are **universal approximators**

Neural Networks Parameters

A neural network is just a **supervised model**

$$y = f(\mathbf{x}; \theta)$$

- It is a function that takes inputs **x**, and computes an output y based on parameters θ

- **Question:** What is θ in a feedforward neural network?

  A: the **weights** and **biases**

Training Neural Networks

- Specify a **loss** L and a set of **training examples**
  $E=\{(\mathbf{x}^{(1)}, y^{(1)}), ..., (\mathbf{x}^{(n)}, y^{(n)}),\}$

- Training by **gradient descent**:

  1. Compute **loss** on training data: $L(\mathbf{W}, \mathbf{b}) = \sum_i \ell\left(f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b}), y^{(i)}\right)$

  2. Compute **gradient** of loss: $\nabla L(\mathbf{W}, \mathbf{b})$

  3. **Update parameters** to make loss smaller:

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

Image Classification

Problem: Recognize the handwritten digit from an image

- What are the **inputs**?   A: pixels!

- What are the **outputs**?   A: one-hot vector

- What is the **loss**?   A: many possibilities
  log-loss, cross-entropy,
  0/1 (i.e., accuracy)

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**?

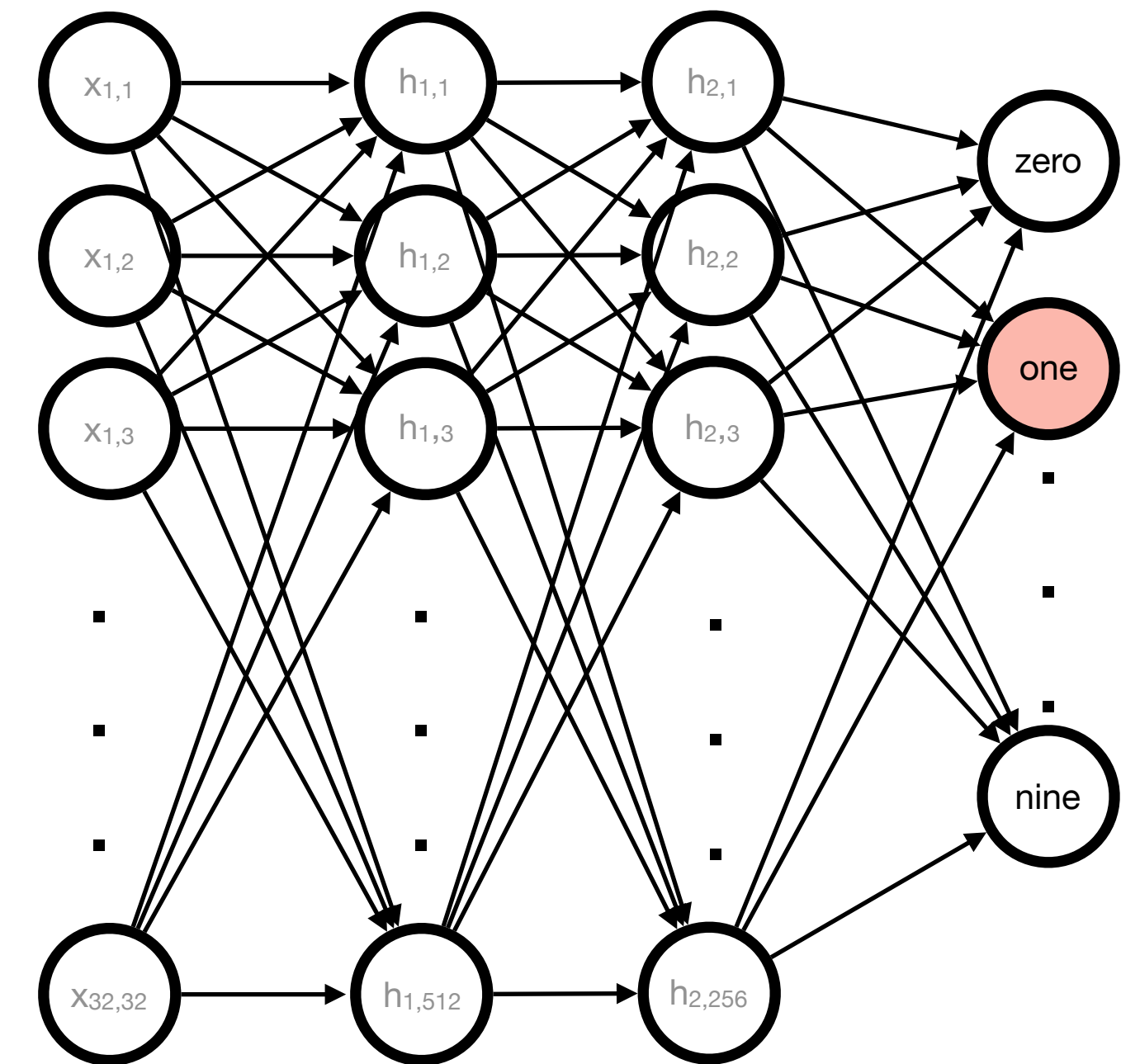- What are the **parameters**?

- What is the **loss**?
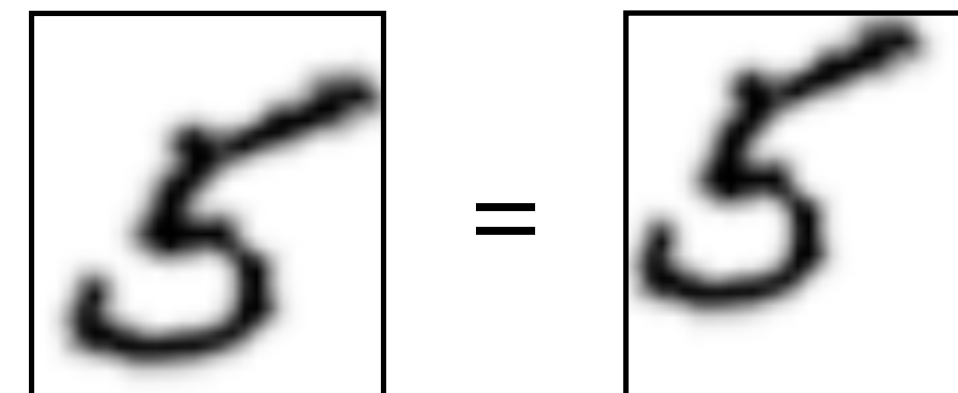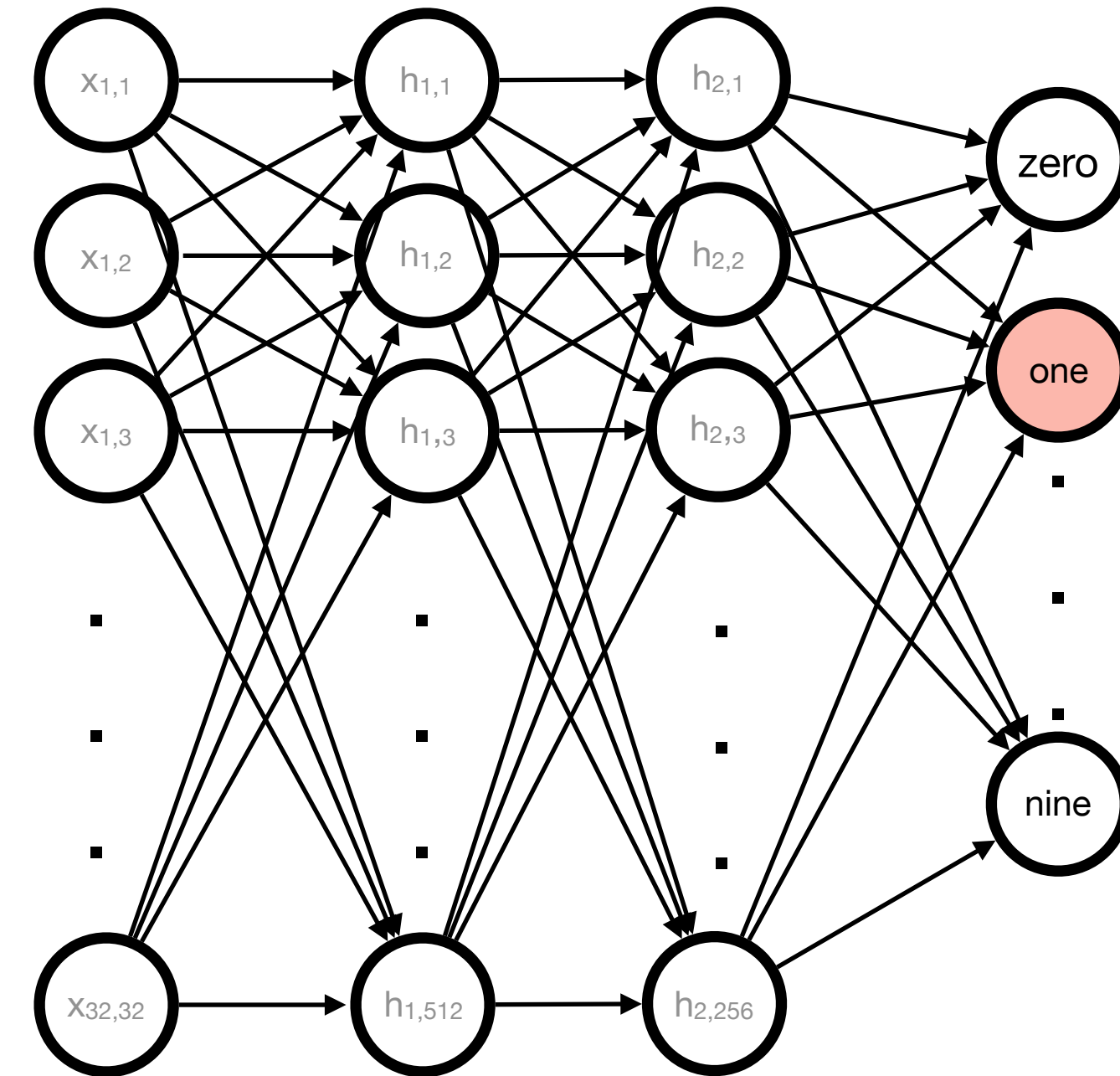
Image Recognition Issues

- 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 biases)

  - Needs **lots of data** to train

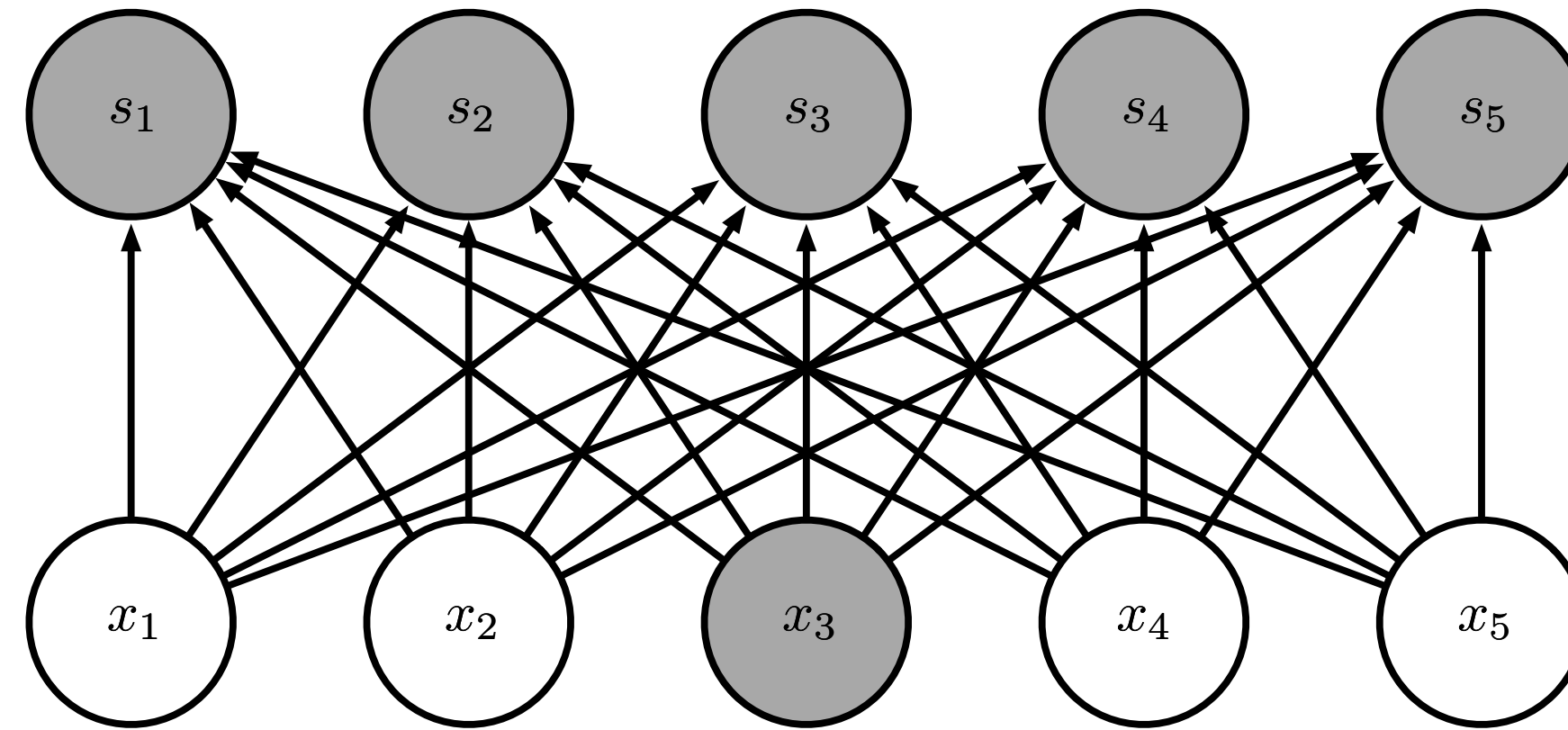- Want to **generalize** over **transformations** of the input
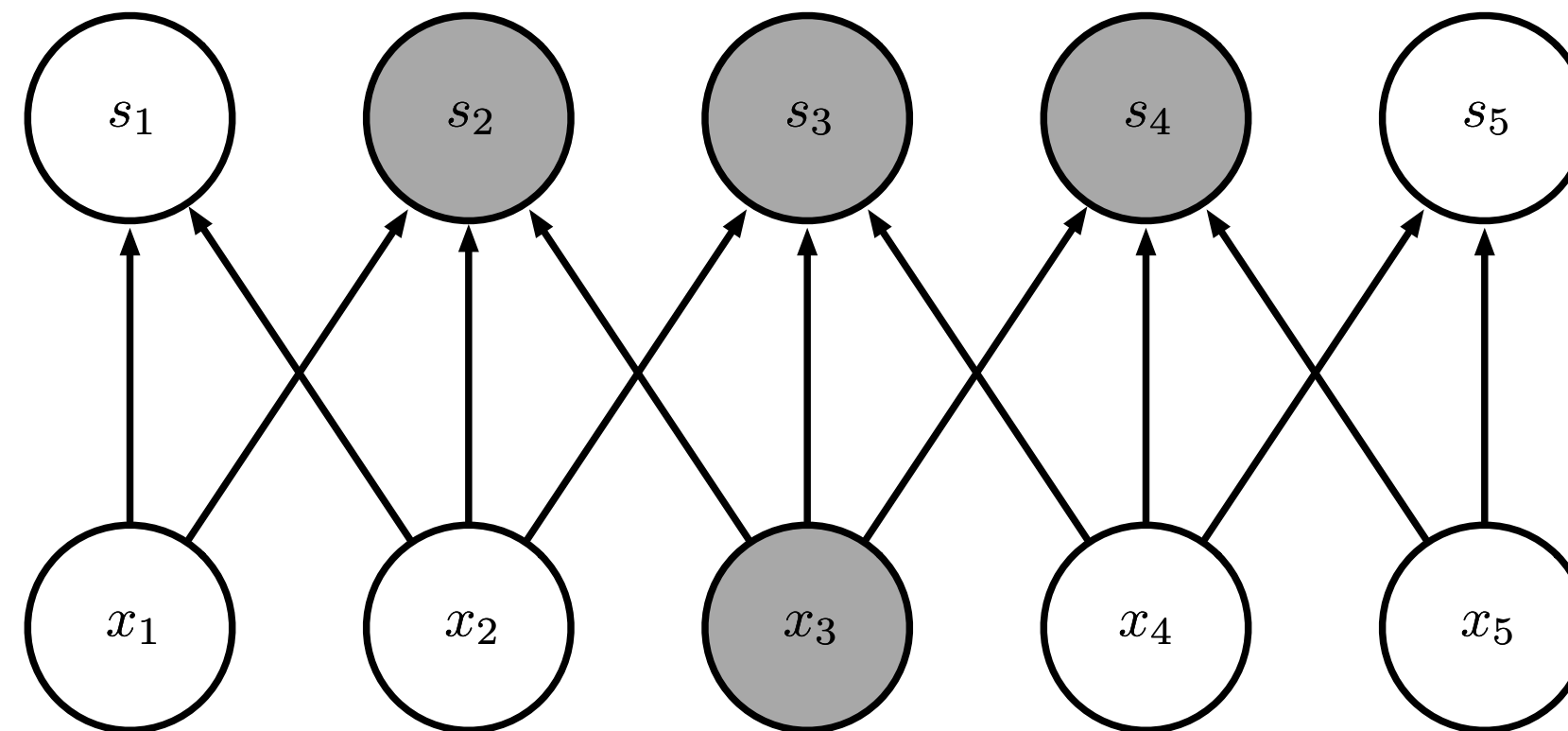
Convolutional Neural Networks

- Convolutional neural networks: a **specialized architecture** for **image recognition**

- Introduce two **new operations**:

  1. Convolutions

  2. Pooling

- Efficient **learning** via:

  1. Sparse interactions

  2. Parameter sharing

  3. Equivariant representations
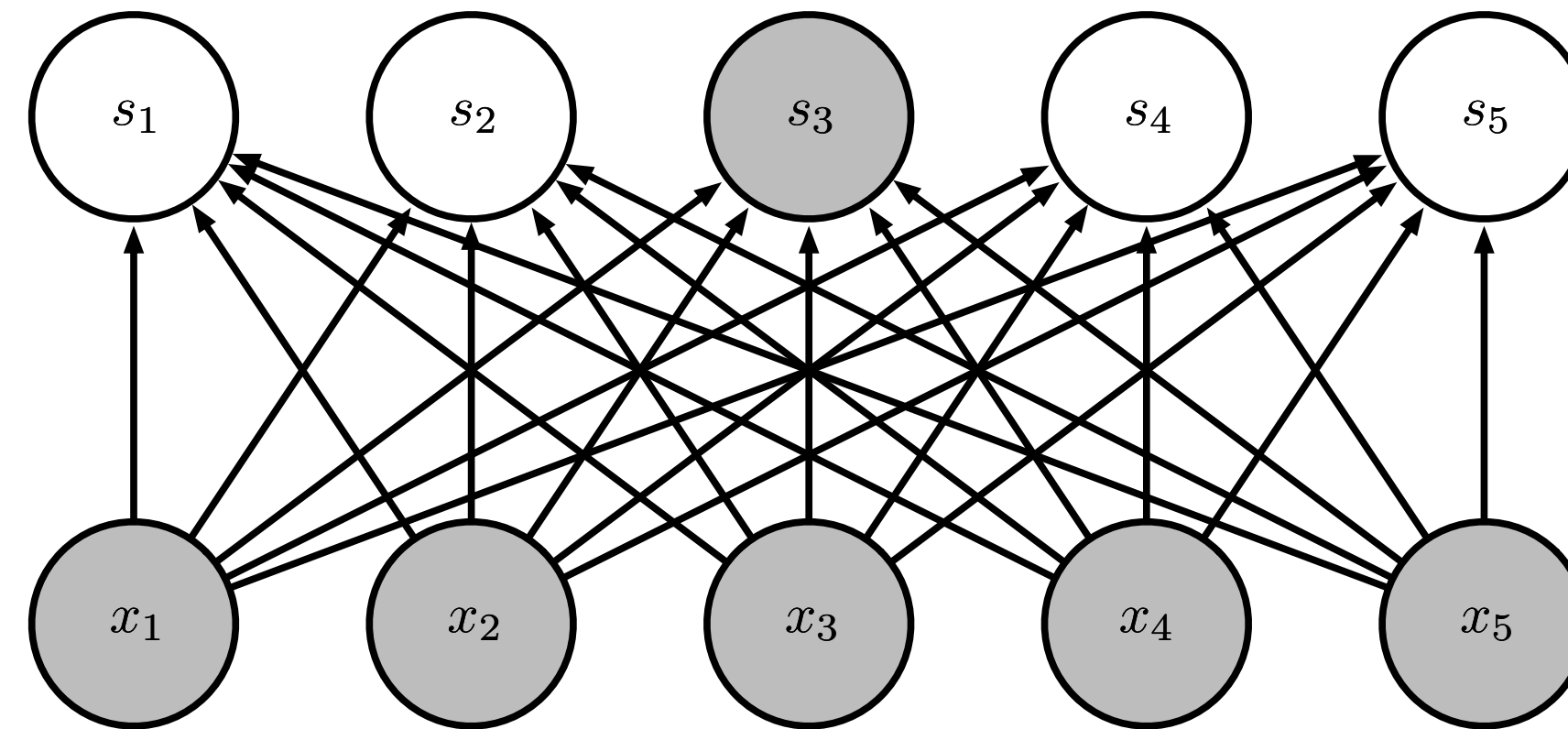
Sparse Interactions
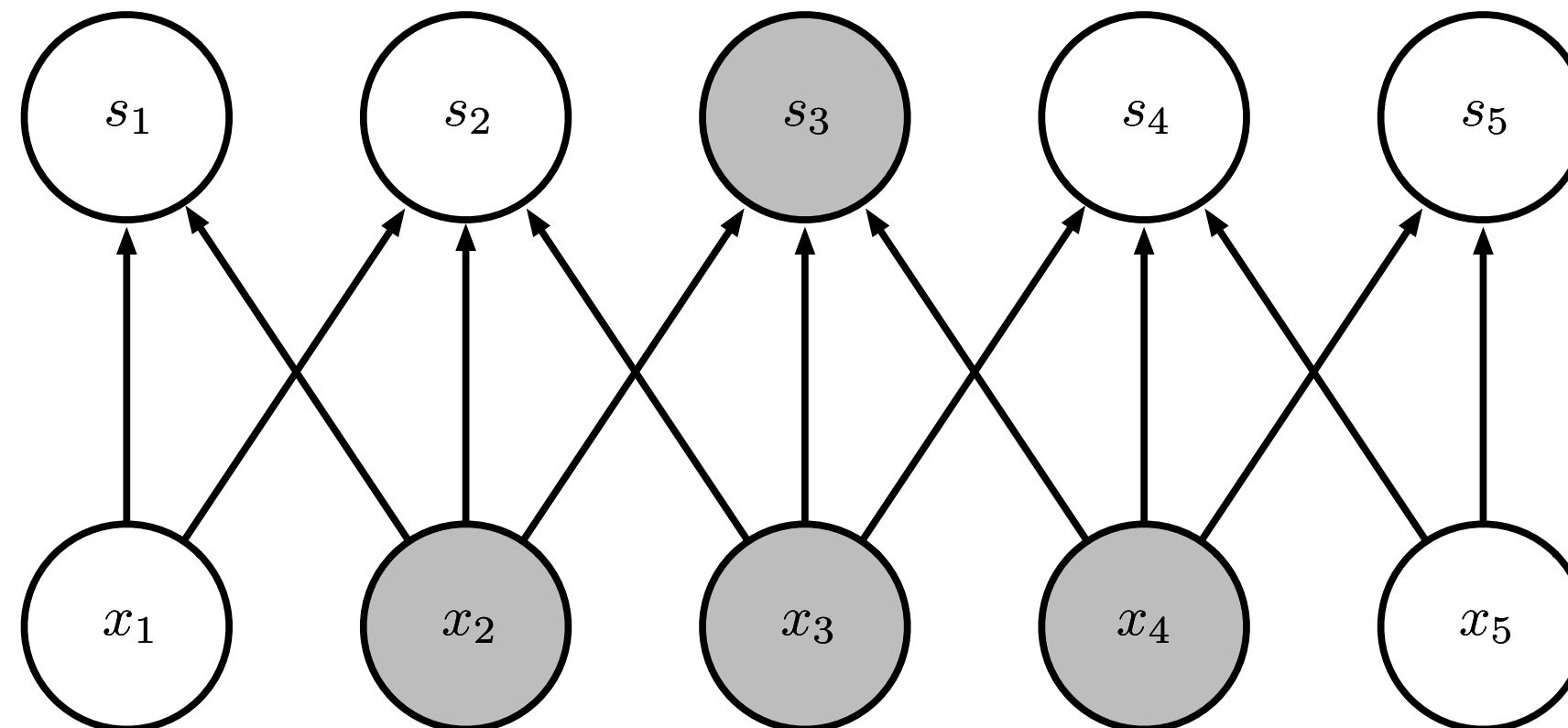

Dense connections

Sparse connections

Sparse Interactions
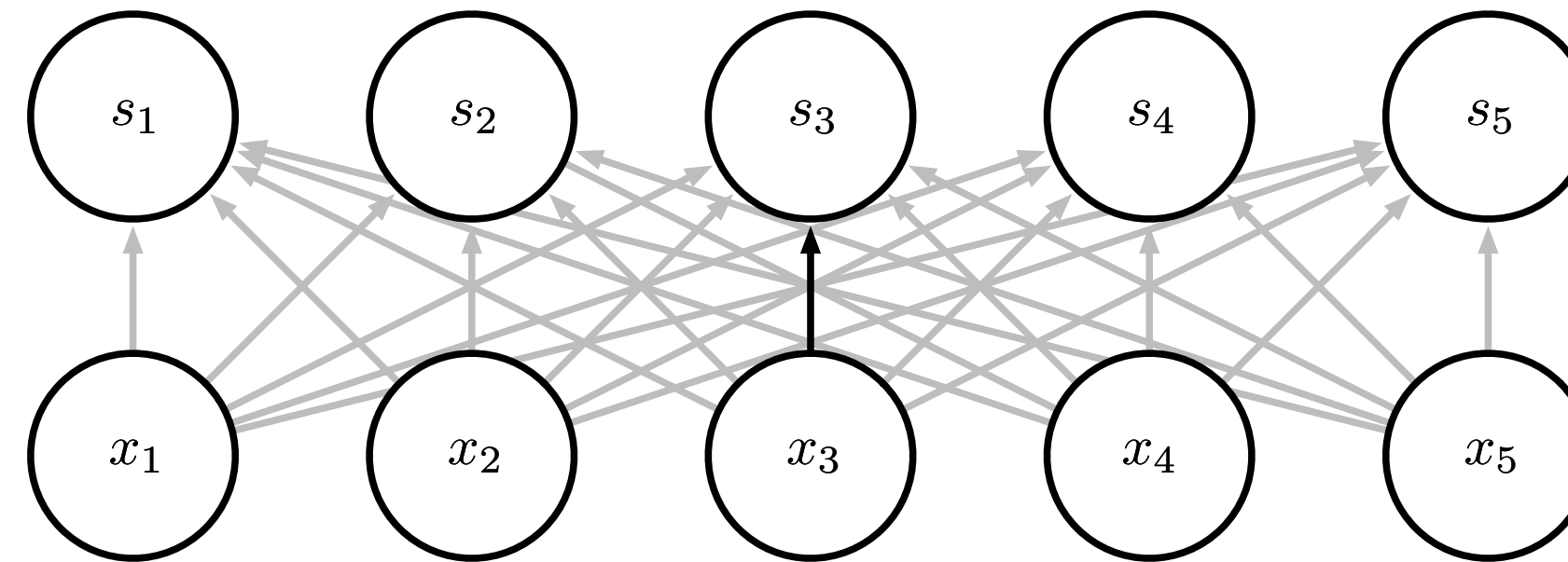
Dense
connections

Sparse
connections

Parameter Sharing

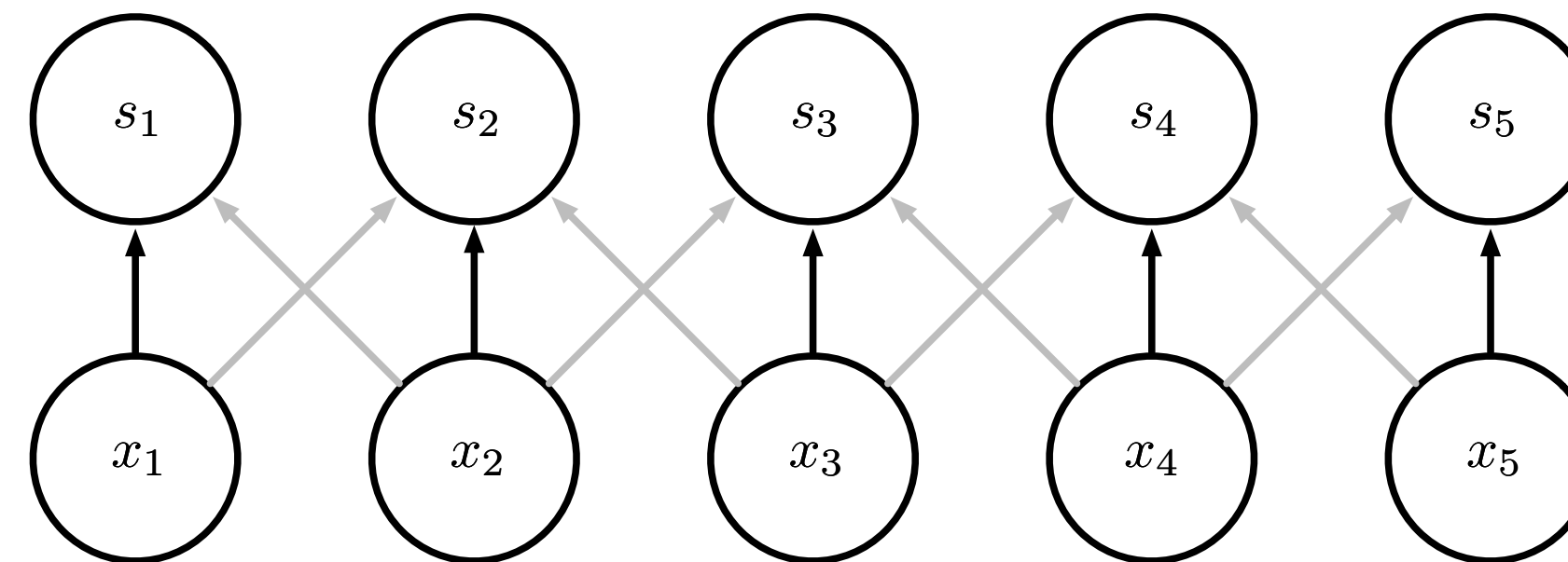Traditional neural nets learn a **unique value** for **each connection**

Convolutional neural nets **constrains** multiple parameters to be **equal**

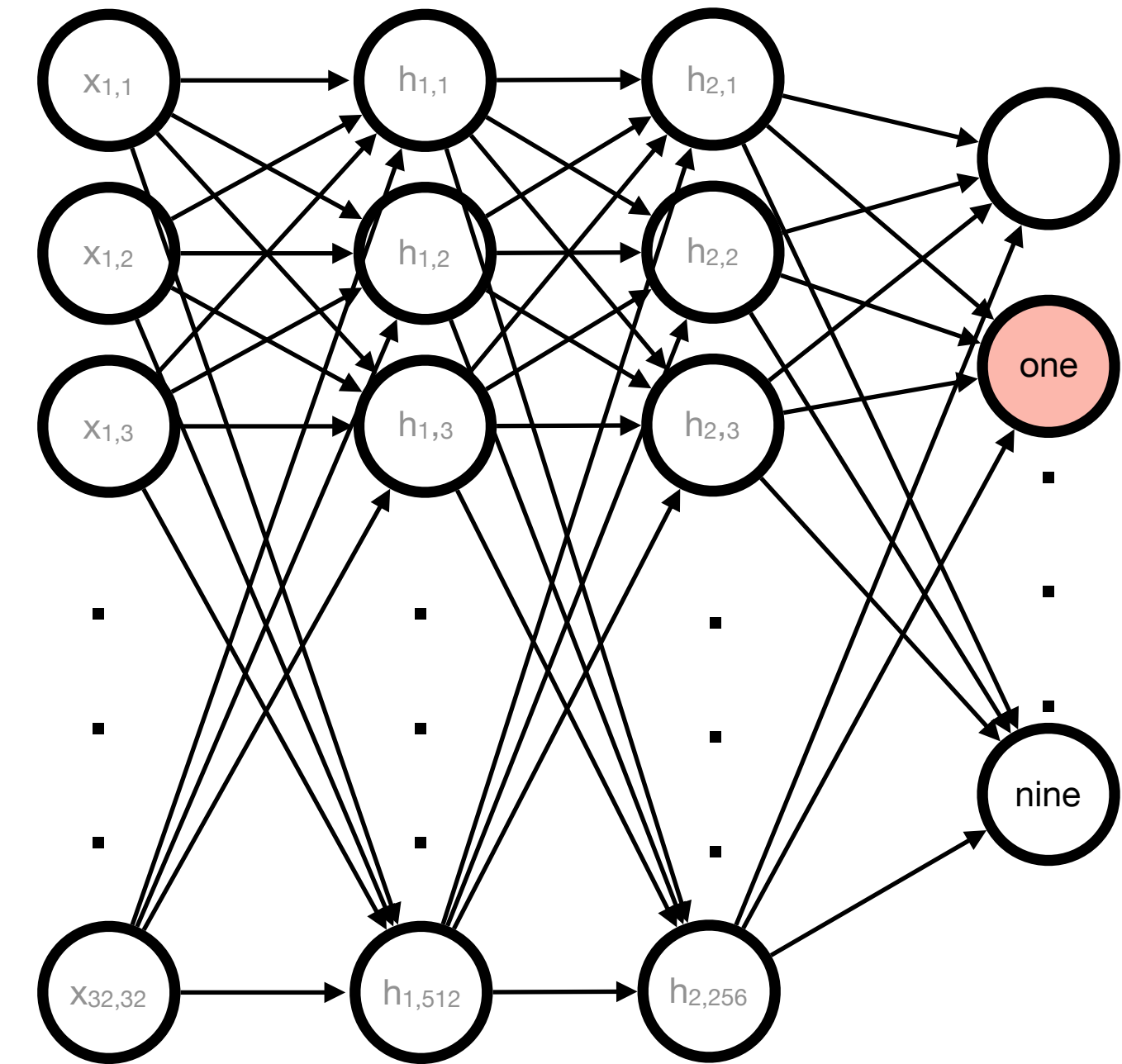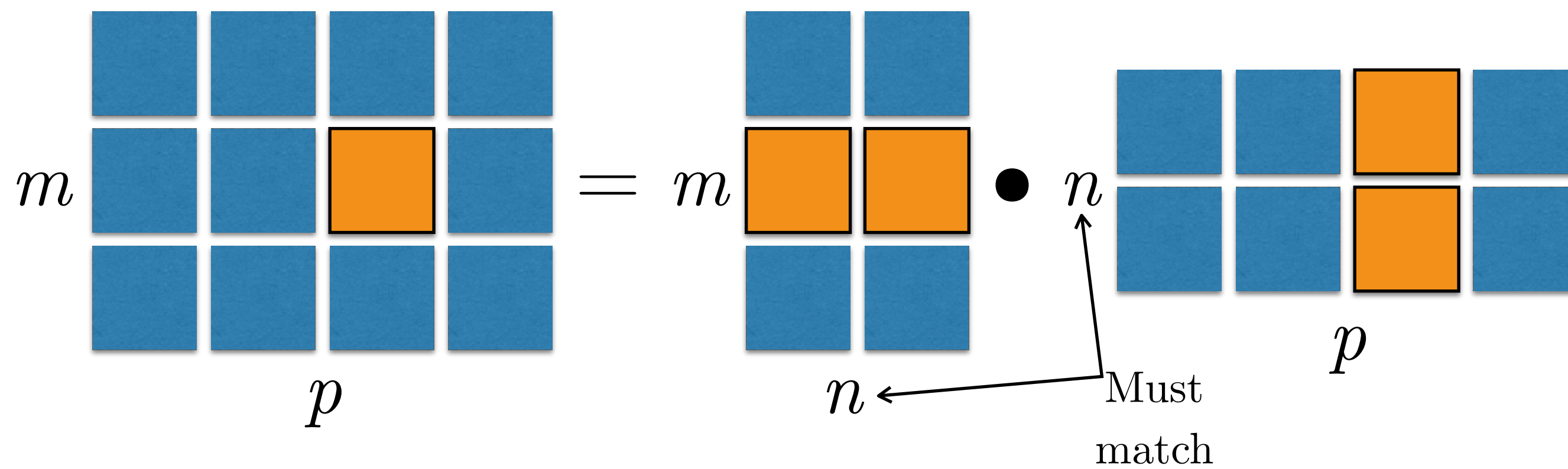

(Images: Goodfellow 2016)

Equivariant Representations

- We want to be able to recognize transformed versions of inputs we have seen before:

  - Translation (moved)

  - Rotation

- Without having been **trained** on all transformed versions

Multiplying Matrices and Vectors

the most important operations involving matrices is multiplication of two
The *matrix product* of matrices A and B is a third matrix C. In order
product to be defined, A must have the same number of columns as B has.
If A is of shape $m \times n$ and B is of shape $n \times p$, then C is of shape $m \times p$.
write the matrix product just by placing two or more matrices together,

$$C = AB. \tag{2.4}$$

product operation is defined by

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}. \tag{2.5}$$

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
t-wise product or *Hadamard product*, and is denoted as $A \odot B$.
duct between two vectors x and y of the same dimensionality is the
product $x^\top y$. We can think of the matrix product $C = AB$ as computing
the dot product between row i of A and column j of B.

Recall that we can represent the **activations** in a neural network by a **matrix product**

$$n \leftarrow \text{Must match}$$

$$\mathbf{h_1} = g_h\left(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{h_2} = g_h\left(W^{(2)}\mathbf{h_1} + \mathbf{b}^{(2)}\right)$$

$$\mathbf{y} = g_y\left(W^{(3)}\mathbf{h_2} + \mathbf{b}^{(3)}\right)$$

(Image: Goodfellow 2016)

34

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)
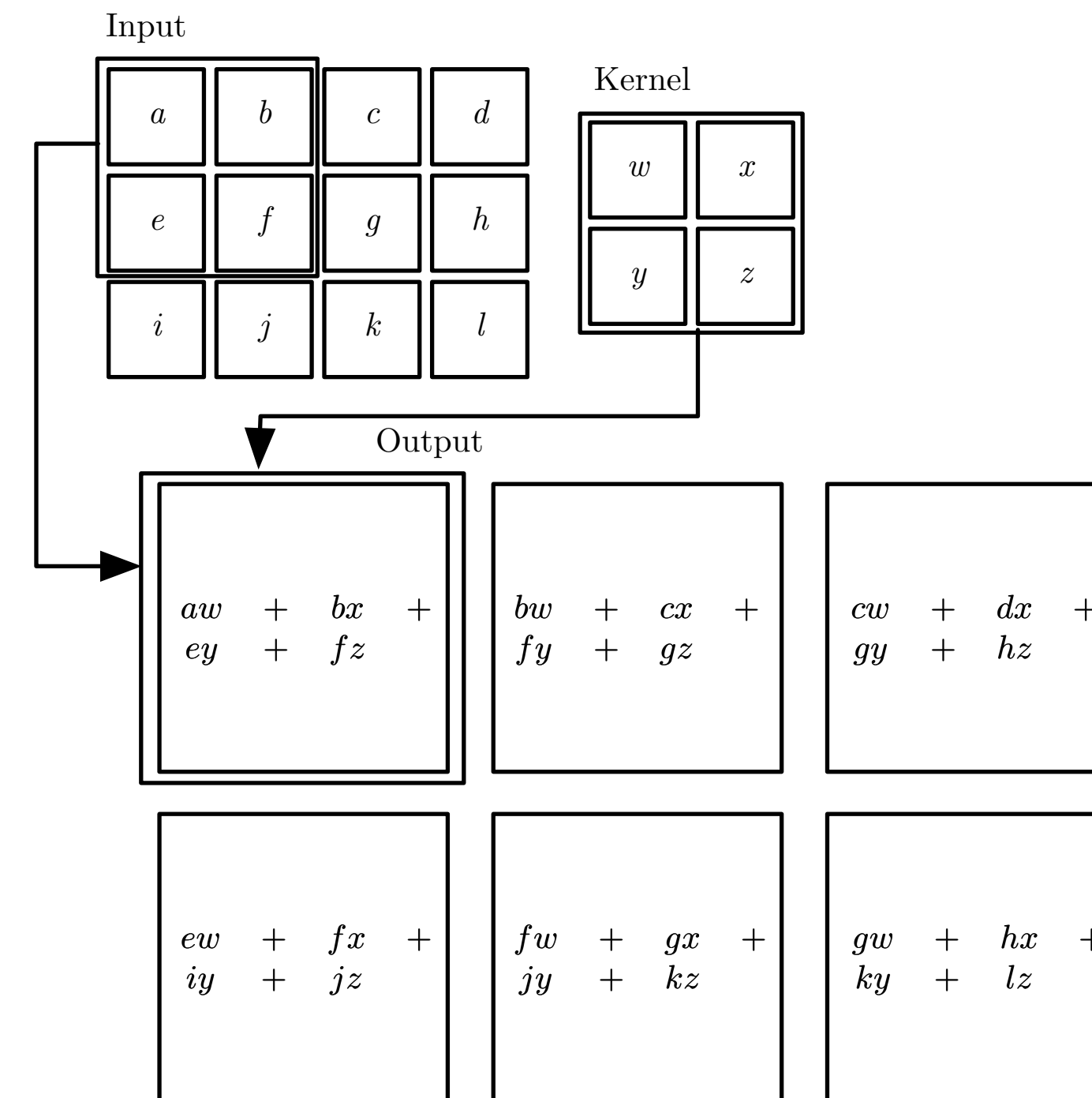
Replace Matrix Multiplication by Convolution

Main idea: Replace matrix multiplications with convolutions

- **Sparsity:** Inputs only combined with neighbours

- **Parameter sharing:** Same kernel used for entire input

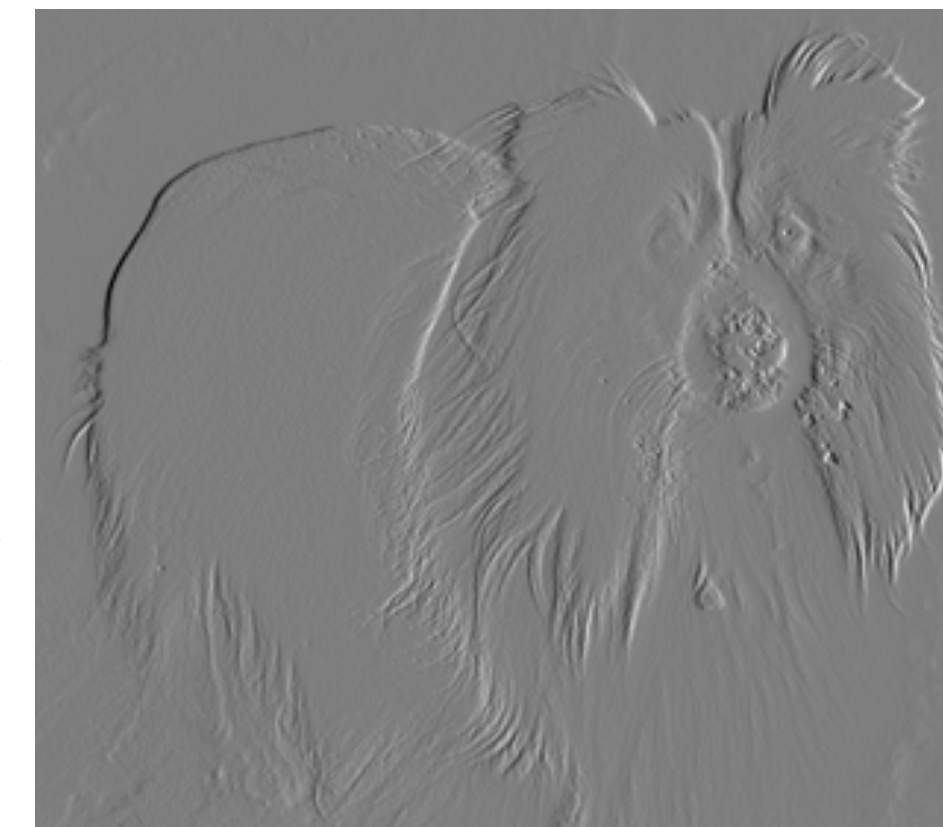# Example: Edge Detection

Input

Output

| 1 | -1 |

Kernel

Efficiency of Convolution

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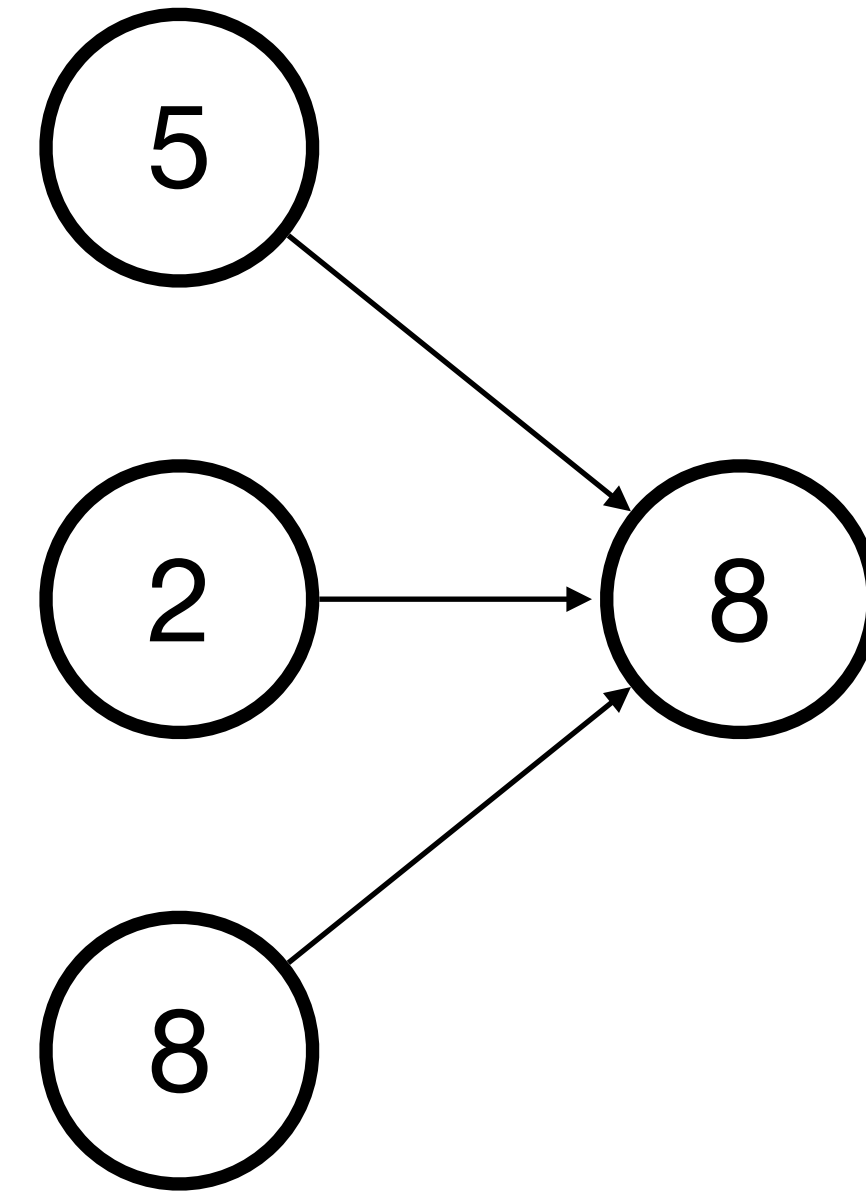Kernel size: 2 by 1
Output size: 319 by 280

	Dense matrix	Sparse matrix	Convolution
Stored floats	319*280*320*280 > 8e9	2*319*280 = 178,640	2
Float muls or adds	> 16e9	Same as convolution (267,960)	319*280*3 = 267,960

(Goodfellow 2016)
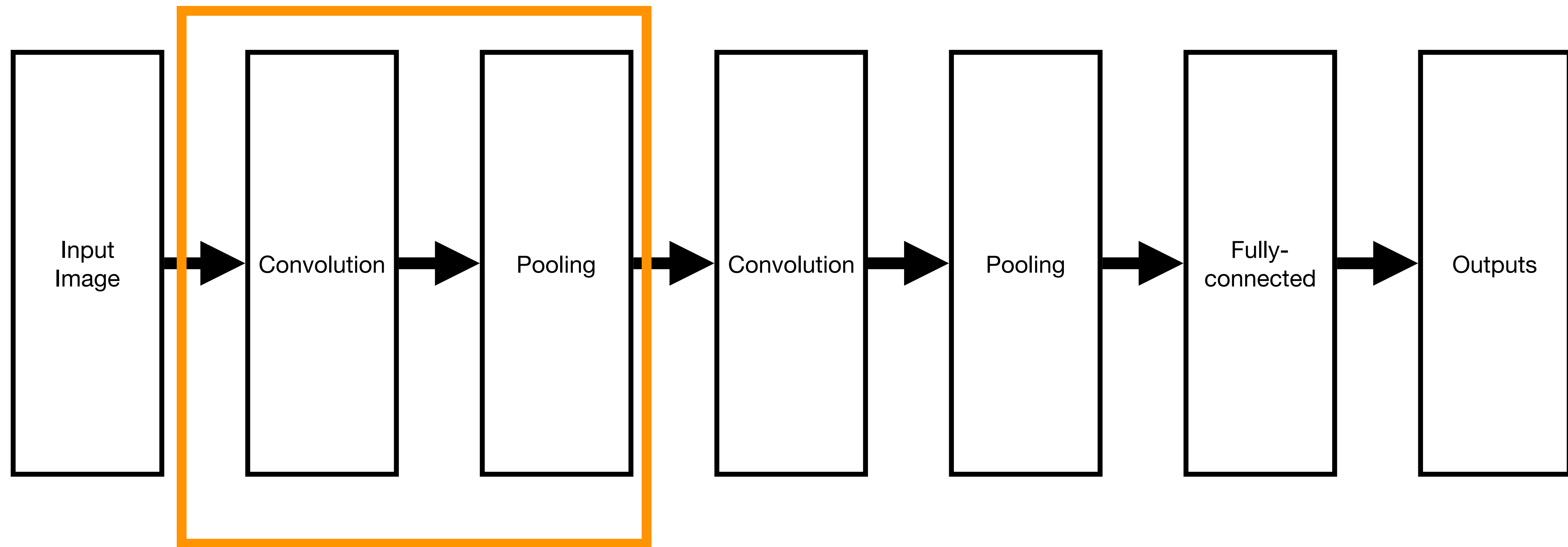
Operation: Pooling

- 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

Typical Architecture

Input Image → Convolution → Pooling → Convolution → Pooling → Fully-connected → Outputs
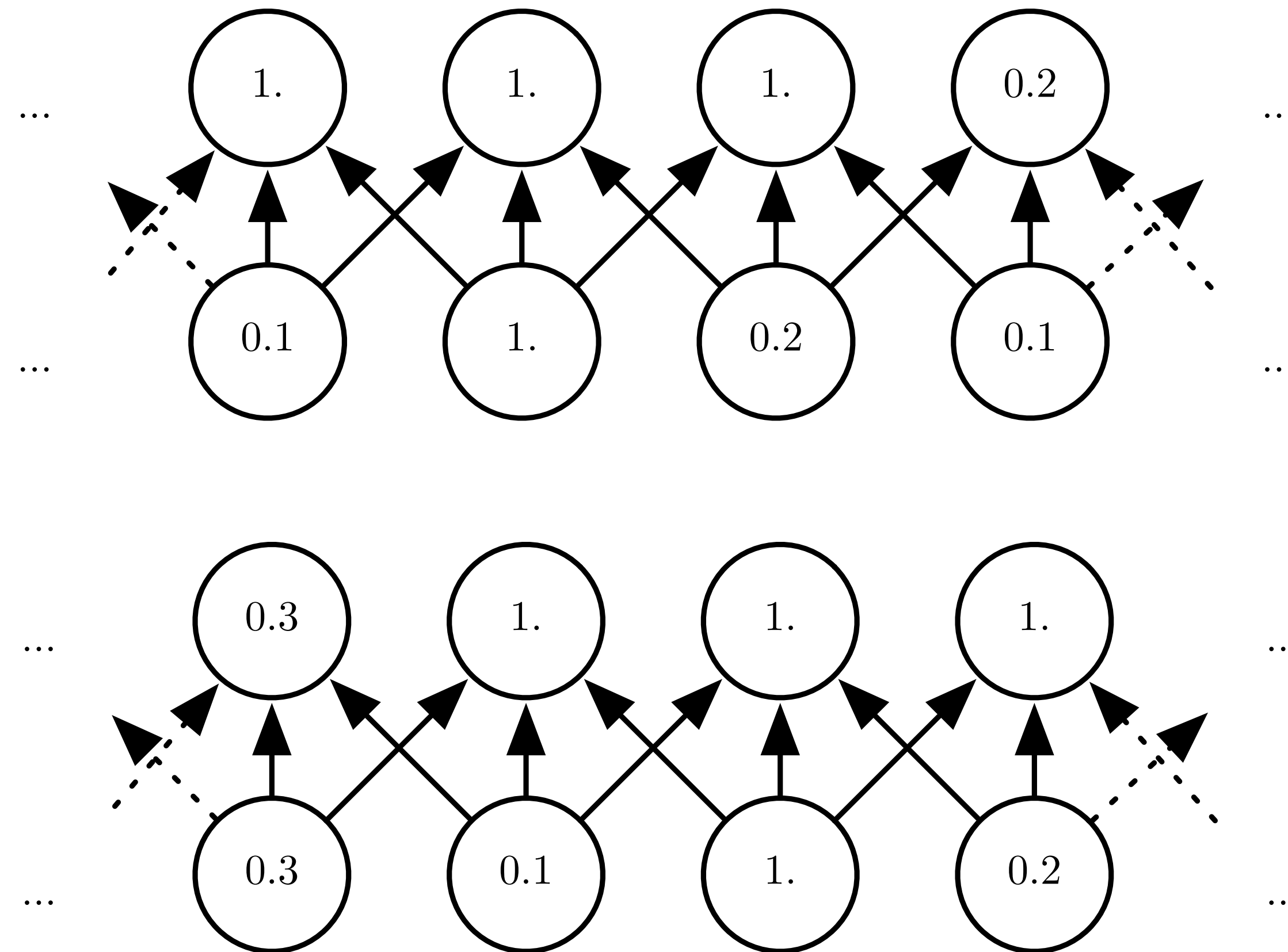
Often convolution-then-pooling is collectively referred to as a **"convolution layer"**

Summary

- Classifying images with a standard feedforward network requires vast quantities of **parameters** (and hence **data**)

- Convolutional networks add **pooling** and **convolution**

  - Sparse connectivity

  - Parameter sharing

  - Translation equivariance

- Fewer parameters means far more **efficient to train**