Convolutional Neural Networks

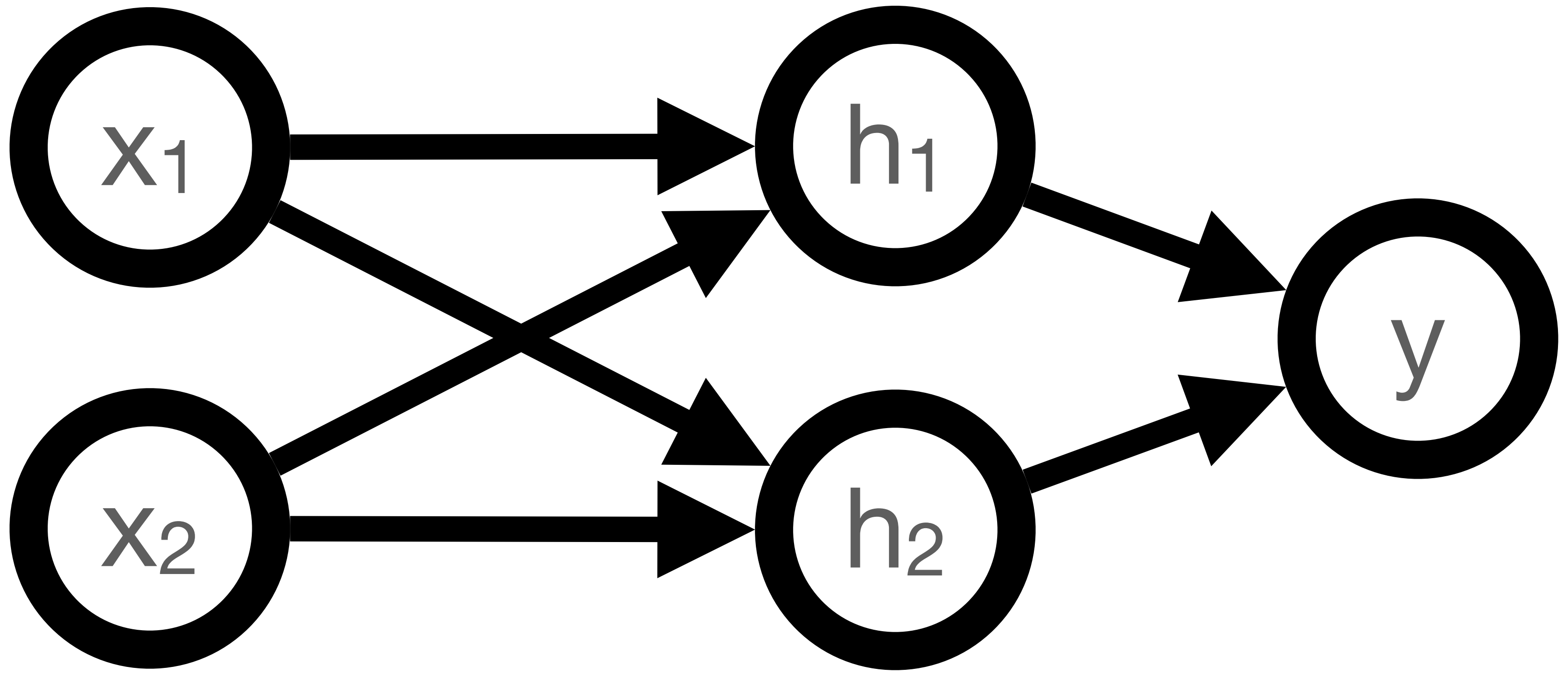
CMPUT 366: Intelligent Systems

GBC §9.0-9.4

Logistics

- Midterm is next **Friday, March 11**
  - Exam delivered via eclass
  - 24 hour window to take the exam
  - 1 hour from start
- There will be a review class next Wednesday
  - More details about format, likely questions, etc.

Recap: Feedforward Neural Network



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

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

$$= g\left(b^{(y)} + \sum_{i=1}^{n} w_i^{(y)} g\left(b^{(i)} + \sum_{j=1}^{n} w_j^{(i)} x_j\right)\right)$$

- 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**

Recap: Training Neural Networks

- Specify a **loss** L and a set of **training examples:**

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

- Training by **gradient descent**:

Loss function
(e.g., squared error)

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

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

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

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

Recap: Automatic Differentiation

- Forward mode sweeps through the graph, computing $s_i' = \dfrac{\partial s_i}{\partial s_1}$ for each s_i
  - The numerator varies, and the denominator is fixed
  - At the end, we have computed $s_n' = \dfrac{\partial s_n}{\partial x_i}$ for a single input x_i
- Backward mode does the opposite:
  - For each s_i, computes the local gradient $\overline{s_i} = \dfrac{\partial s_n}{\partial s_i}$
  - The numerator is fixed, and the denominator varies
  - At the end, we have computed $\overline{x_i} = \dfrac{\partial s_n}{\partial x_i}$ for each input x_i
- **Key point:** The intermediate results are computed numerically at each step

Lecture Outline

1. Recap & Logistics
2. Neural Networks for Image Recognition
3. Convolutional Neural Networks

Image Classification

Problem: Recognize the handwritten digit from an image

- What are the **inputs**?
- What are the **outputs**?
- What is the **loss**?

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**?

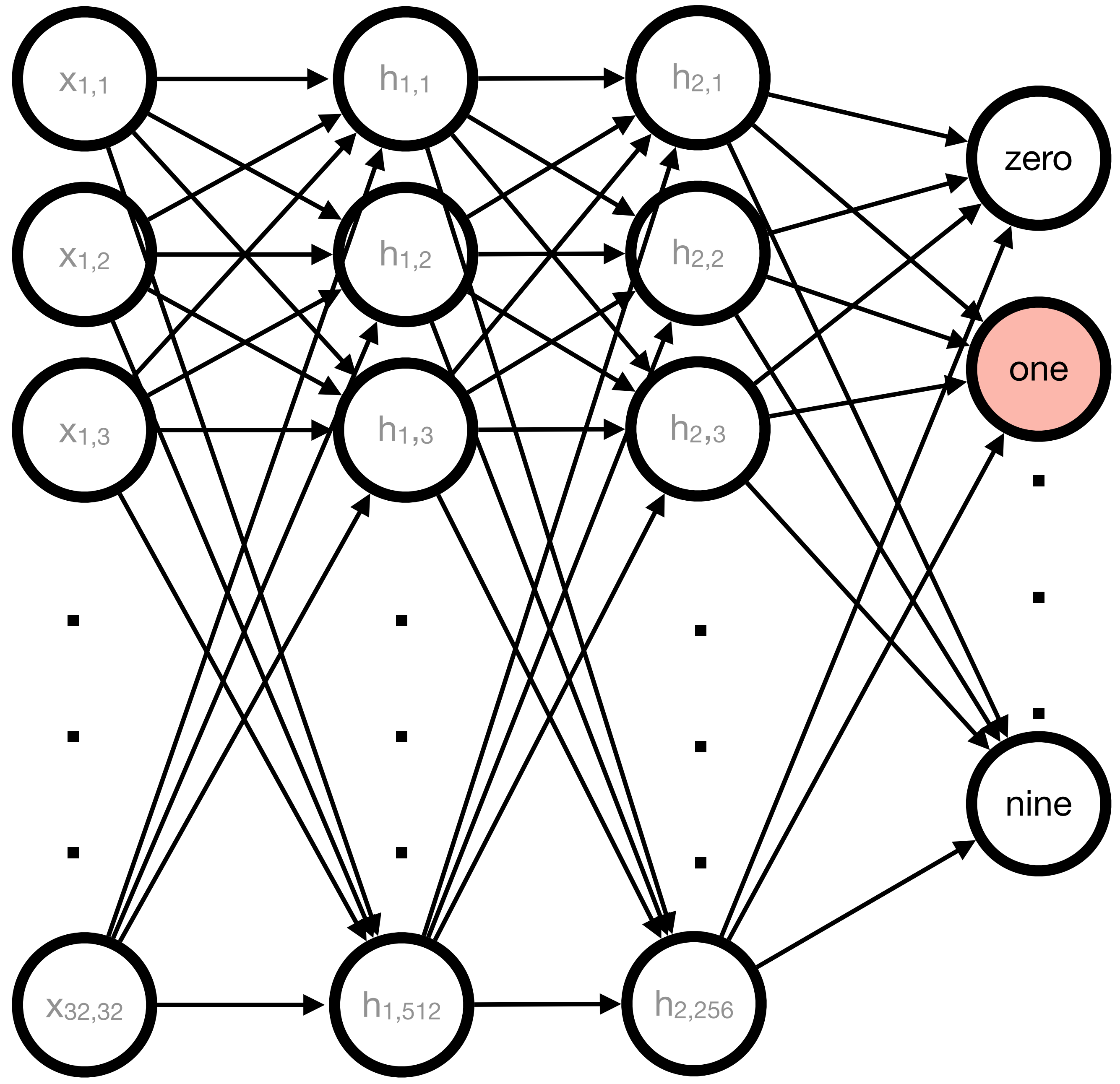
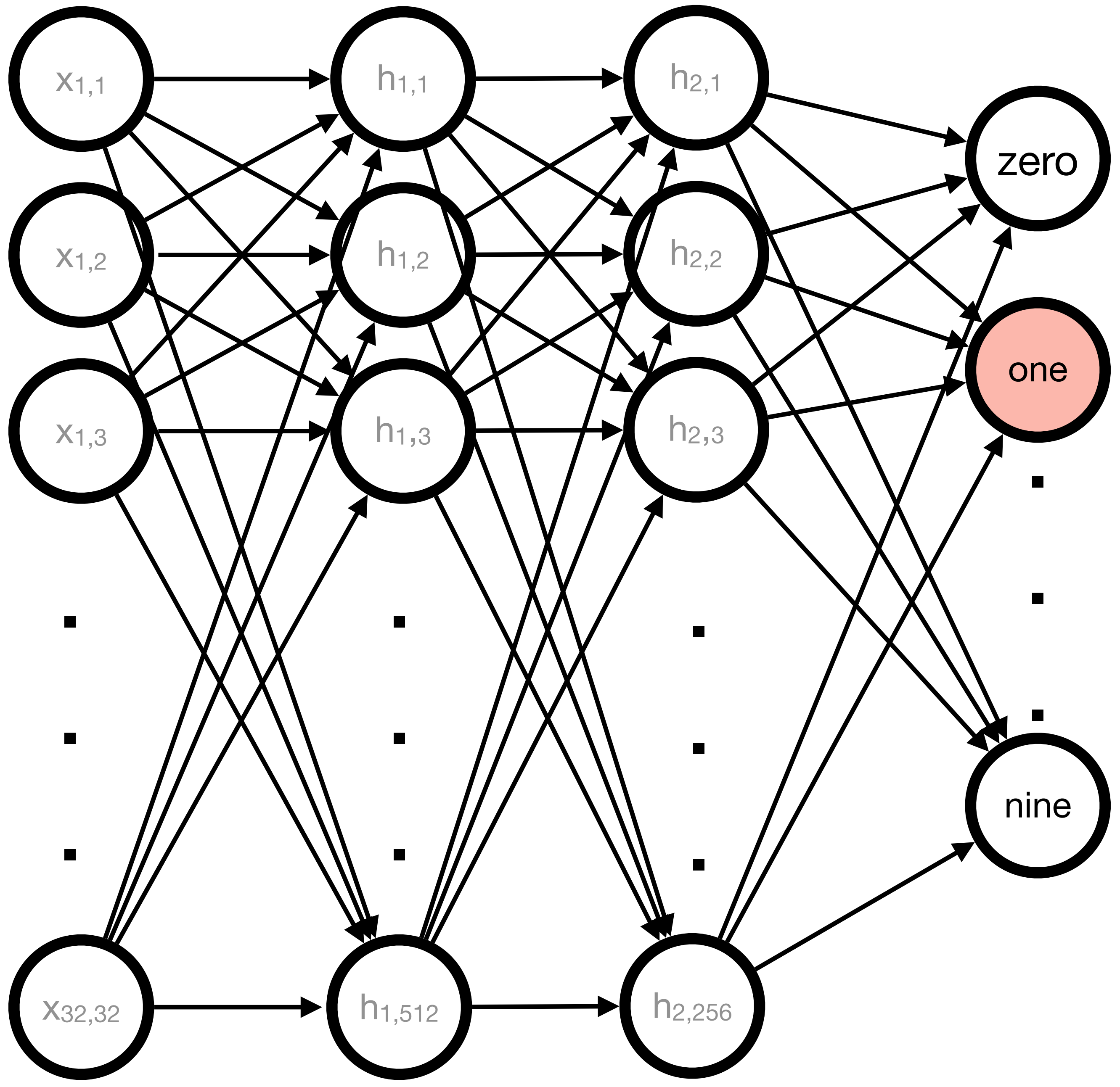


Image Recognition Issues

- 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

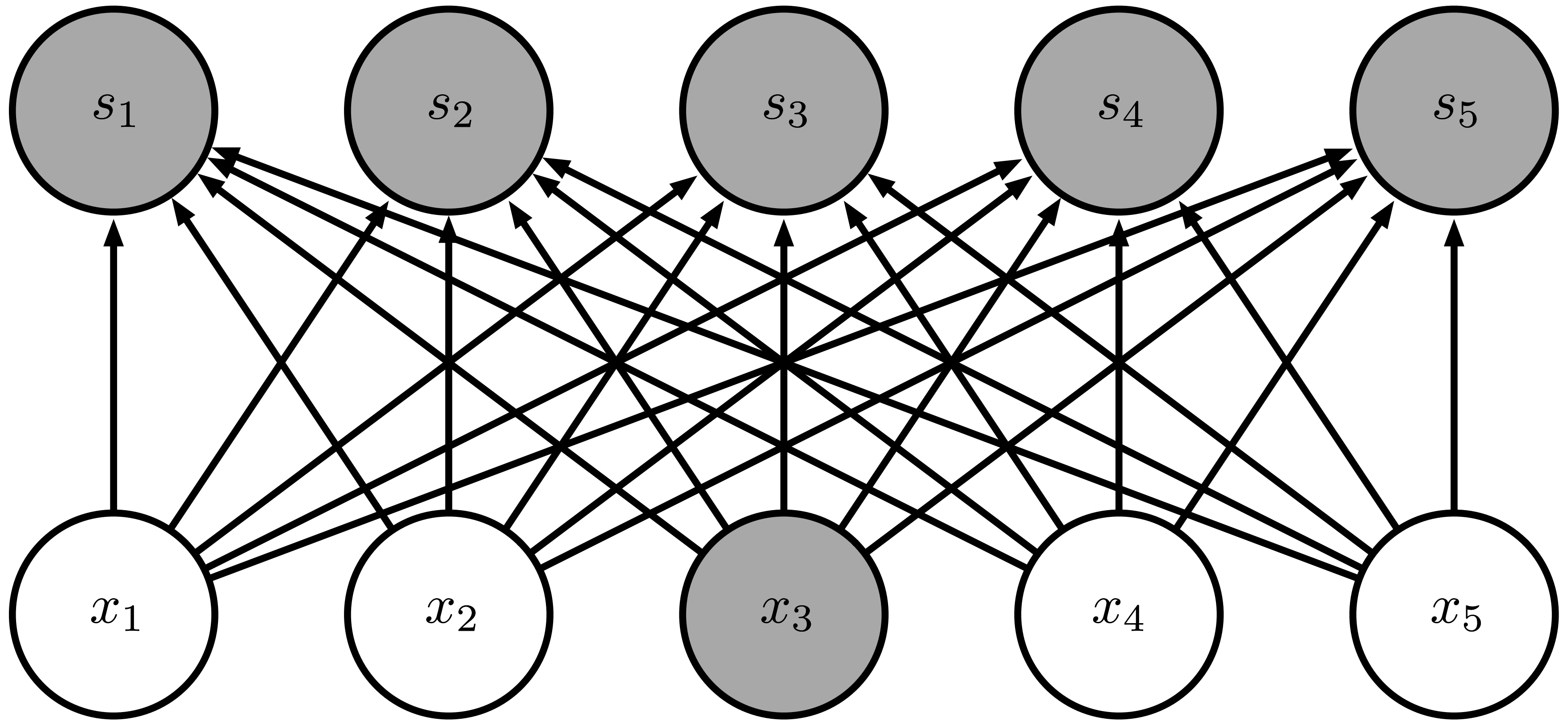


Convolutional Neural Networks

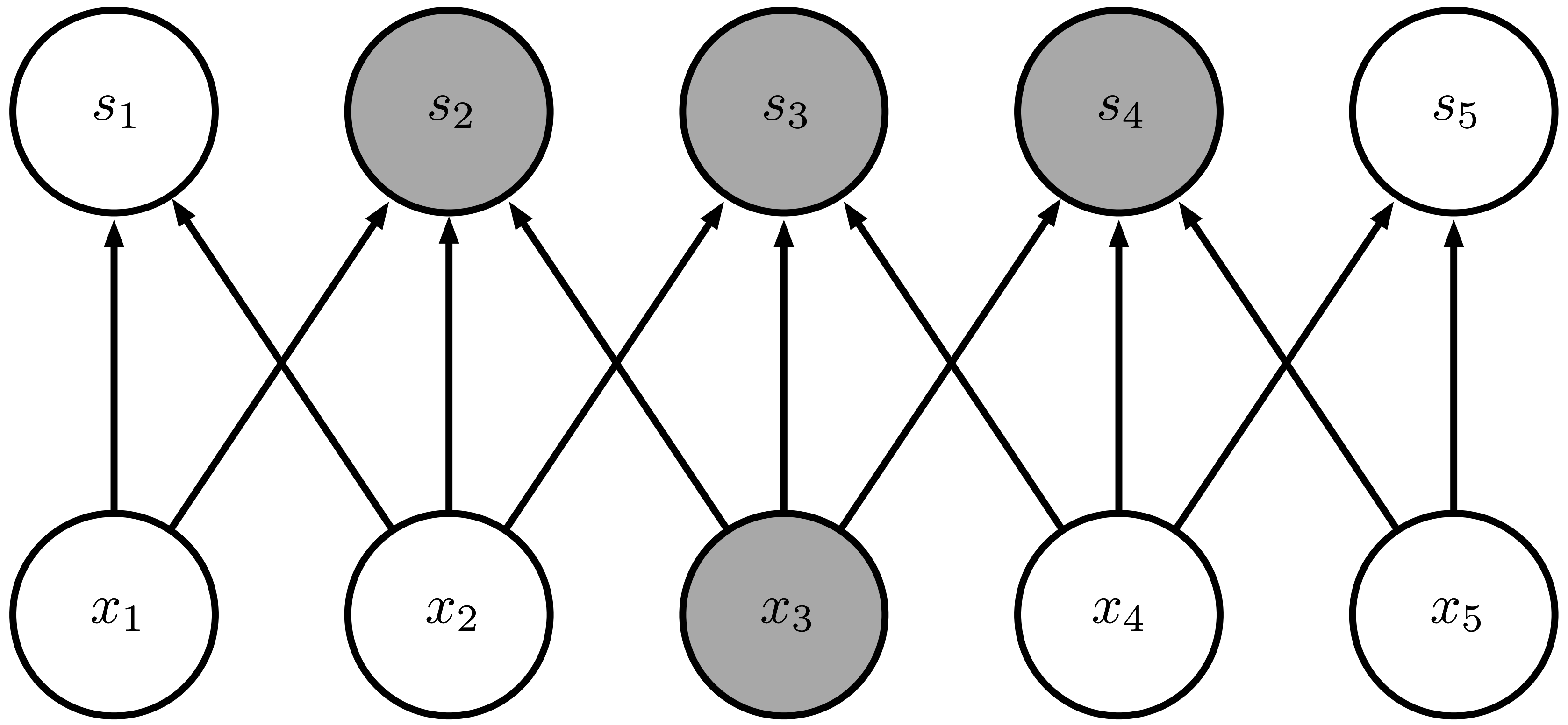
- **Convolutional neural networks:** a **specialized architecture** for image recognition
- Introduce two **new operations**:
    1. Convolutions
    2. Pooling
- Efficient **learning** via:
    1. Sparse interactions
    2. Parameter sharing
    3. Equivariant representations

1. Sparse Interactions

Dense connections



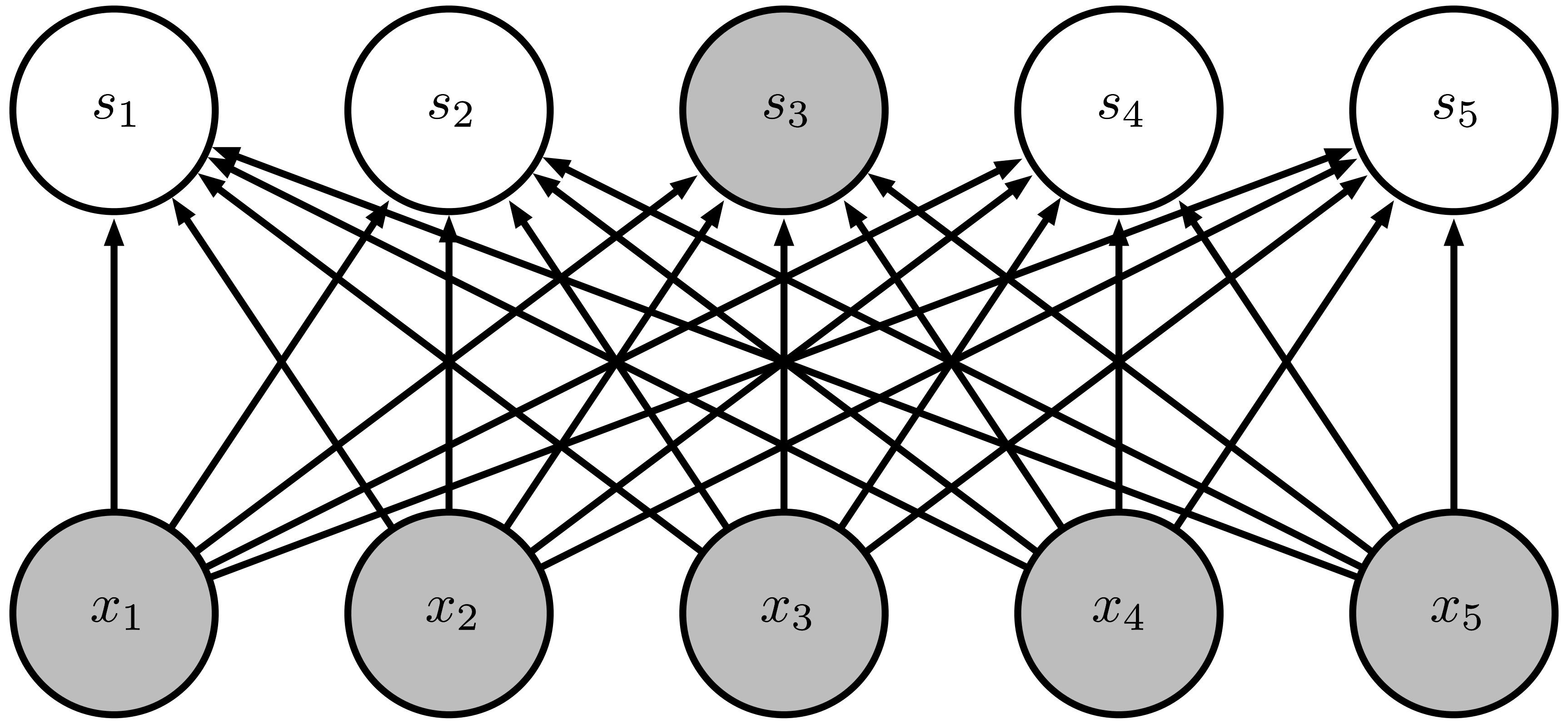
Sparse connections



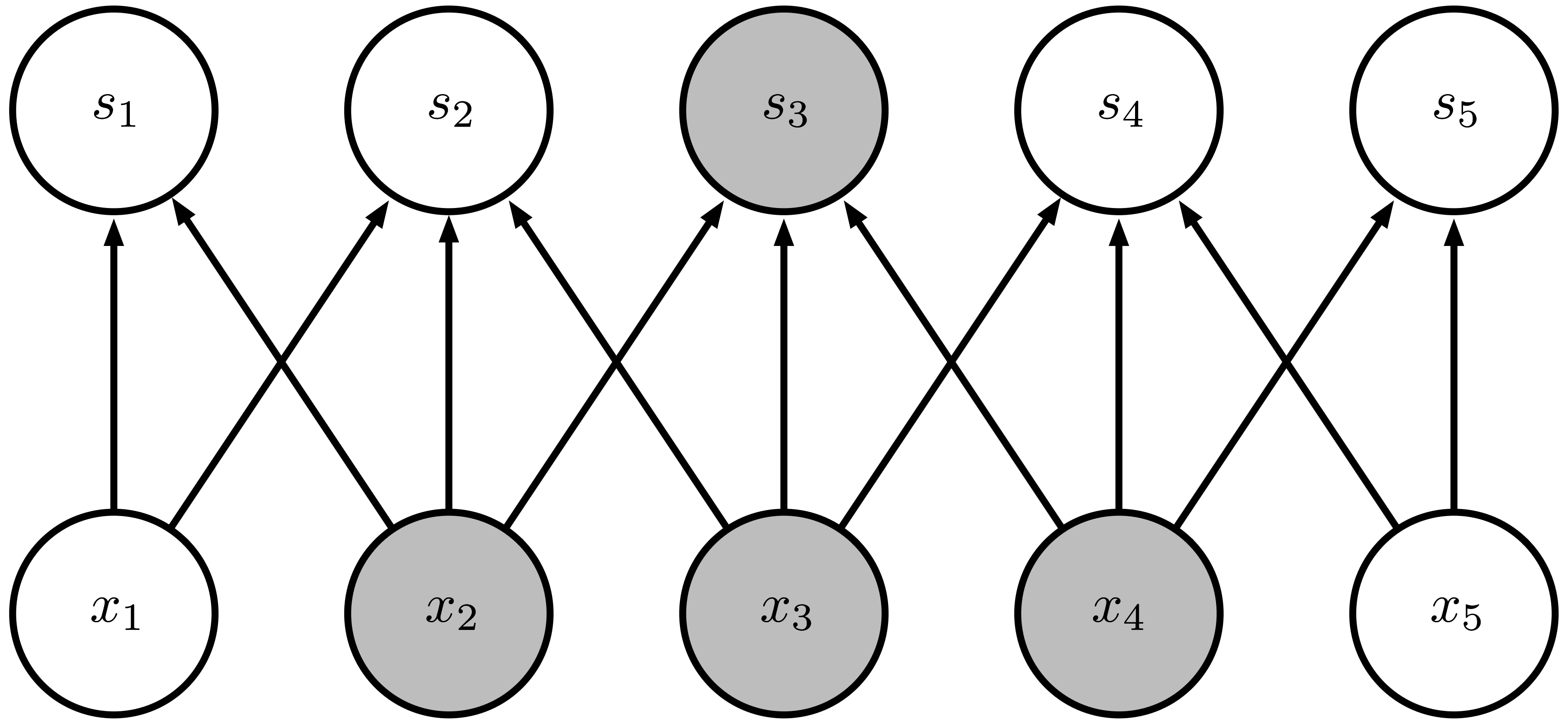
(Images: Goodfellow 2016)

1. Sparse Interactions

Dense connections



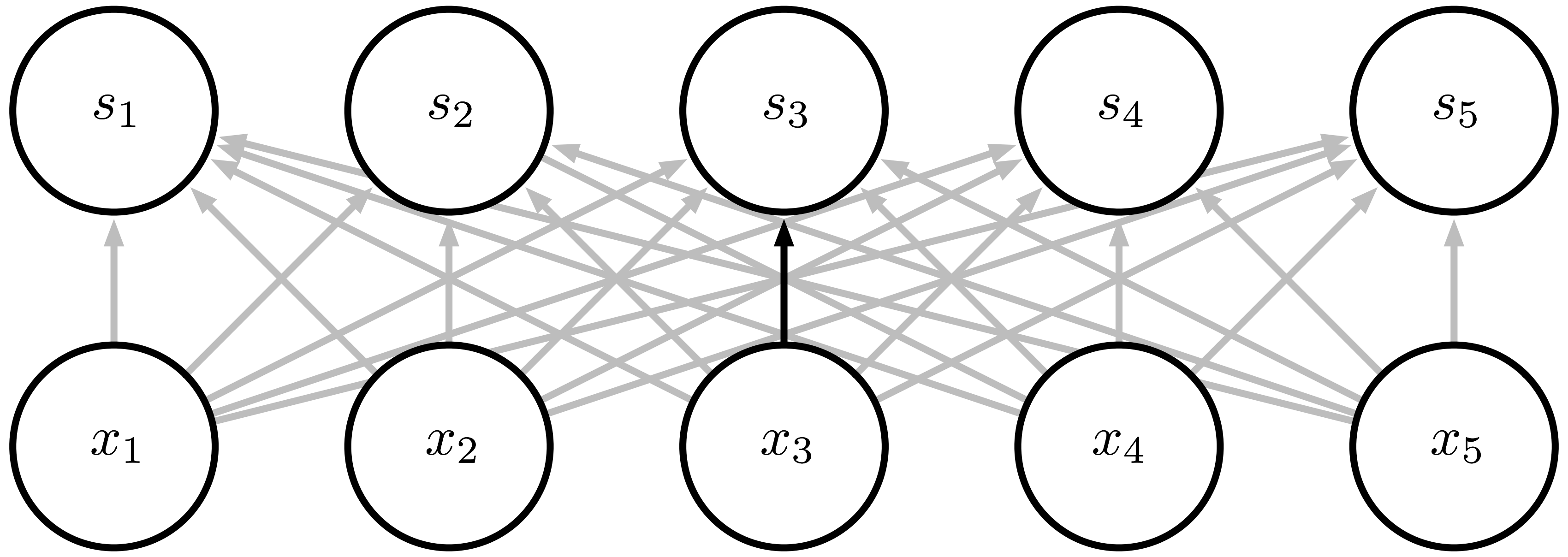
Sparse connections



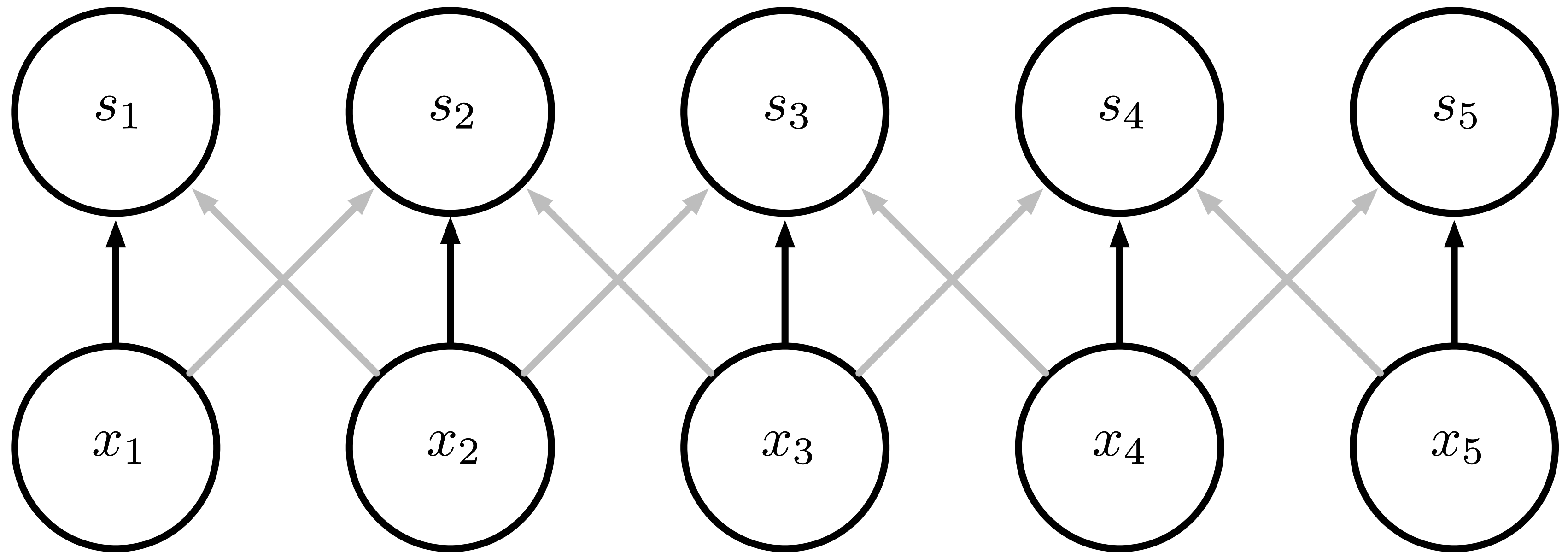
(Images: Goodfellow 2016)

2. Parameter Sharing

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



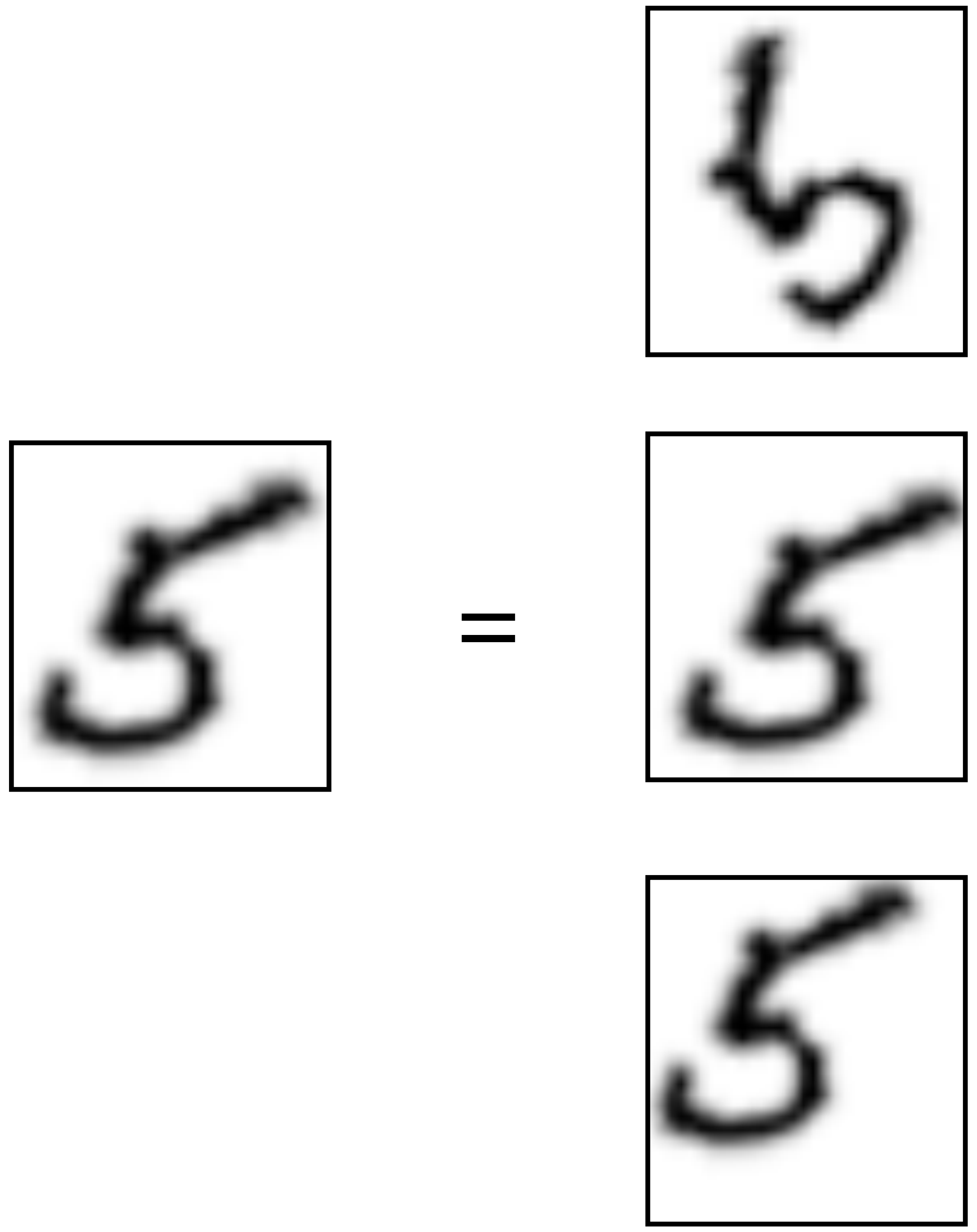
Convolutional neural nets **constrain** multiple parameters to be **equal**



(Images: Goodfellow 2016)

3. Equivariant Representations

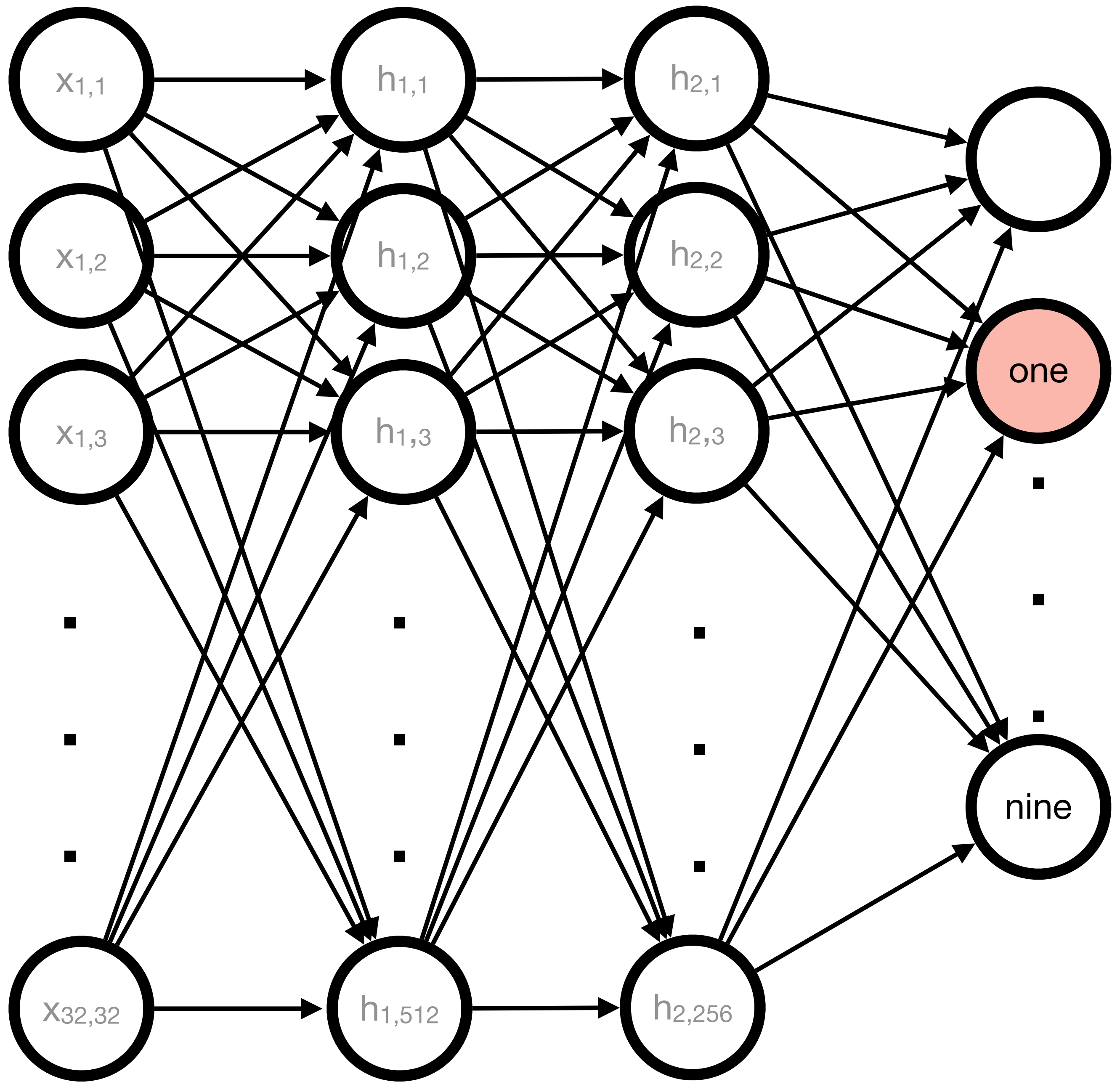
- We want to be able to recognize transformed versions of inputs we have seen before:
  - Translation (moved)
  - Rotation
- Without having been **trained** on all transformed versions



Operation: Matrix Product

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

$$W^{(1)}\mathbf{x} = \begin{bmatrix} w^{(1)}_{x_1\to h_1} & w^{(1)}_{x_2\to h_1} & \cdots w^{(1)}_{x_n\to h_1} \\ w^{(1)}_{x_1\to h_2} & \ddots & \\ \vdots & & \\ w^{(1)}_{x_1\to h_m} & & w^{(1)}_{x_n\to h_m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



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

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

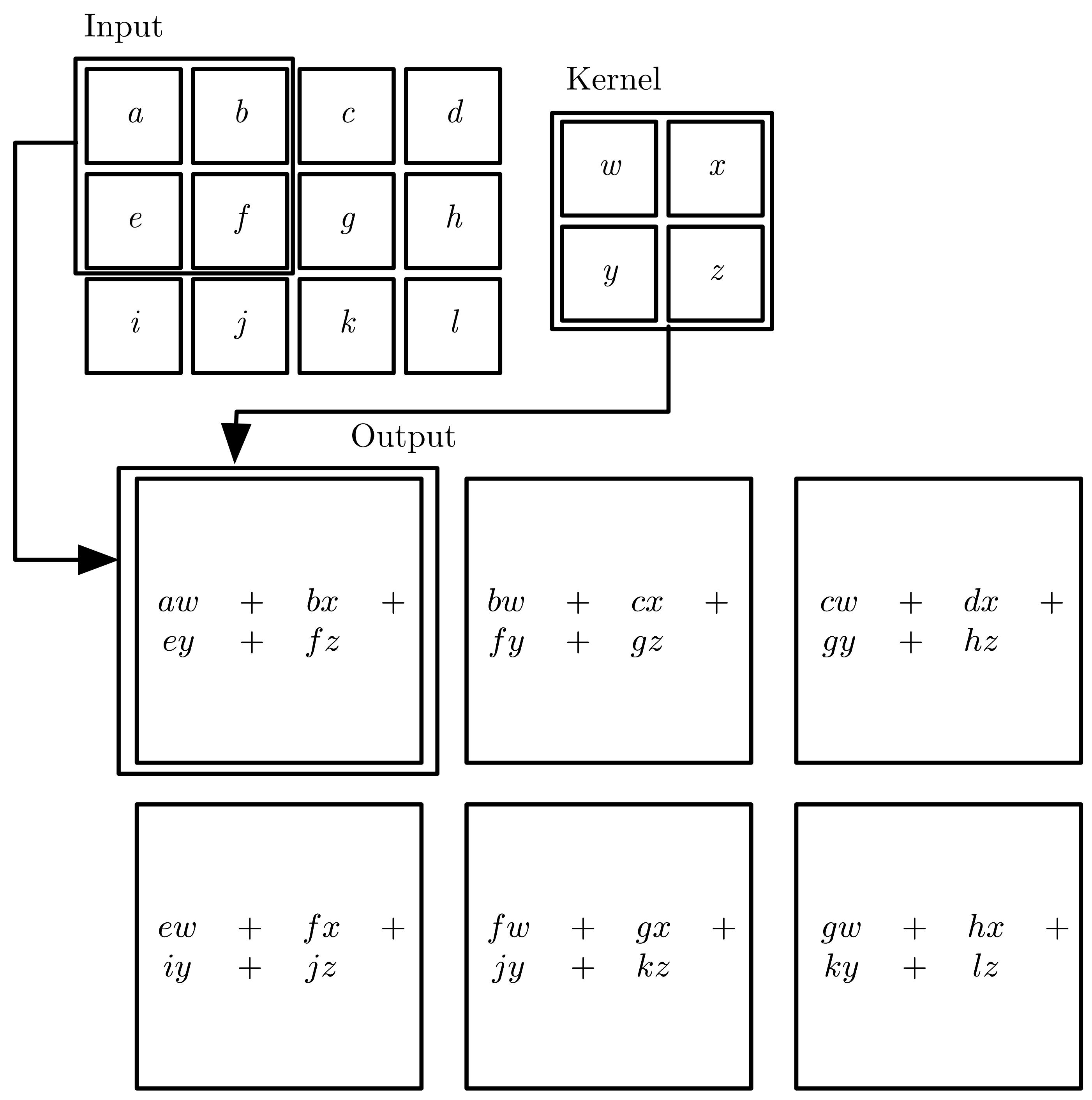
$$\mathbf{y} = g_y\left(W^{(3)}\mathbf{h_2} + \mathbf{b}^{(3)}\right)$$

(Image: Goodfellow 2016)

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



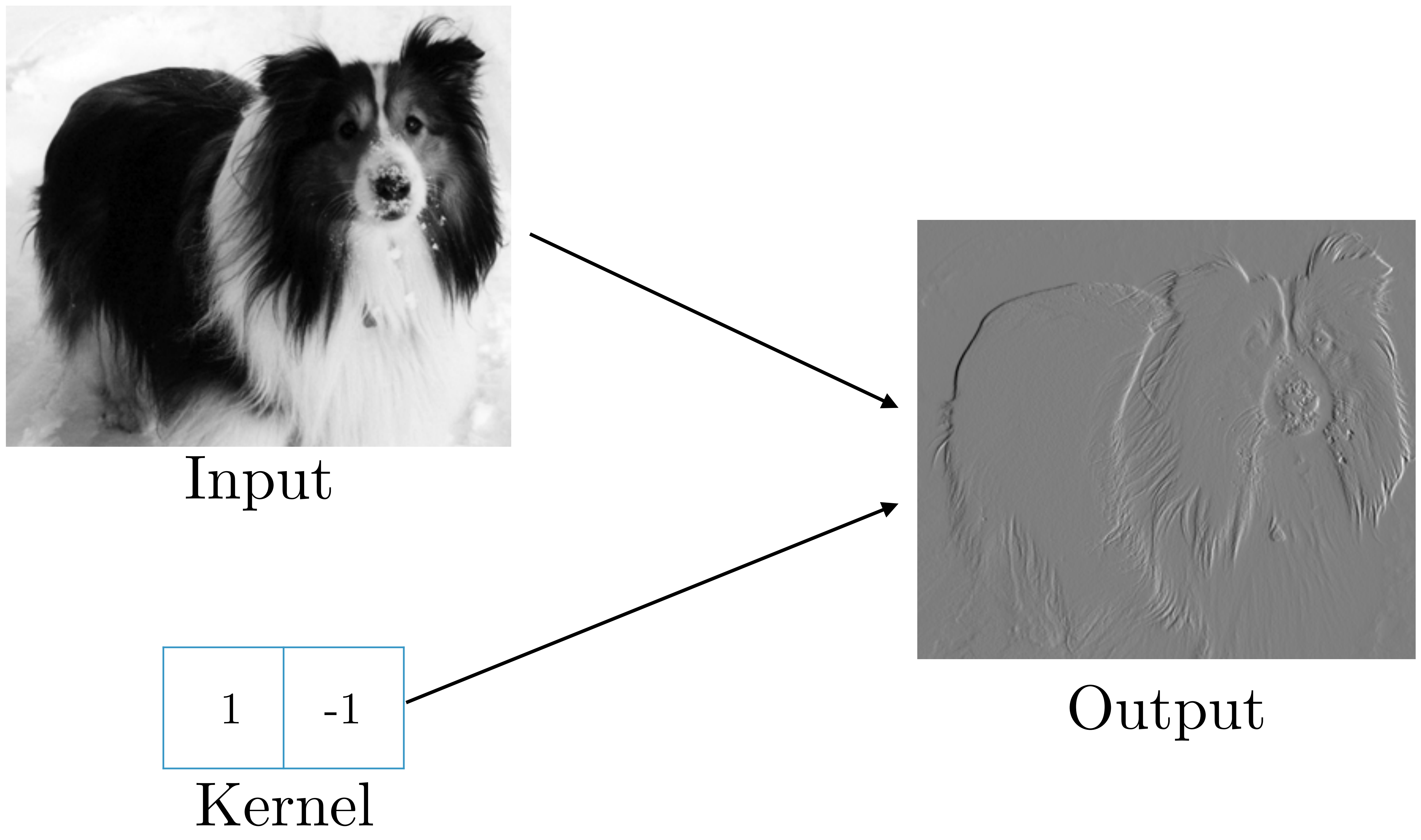
(Image: Goodfellow 2016)

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



(Image: Goodfellow 2016)

Efficiency of Convolution

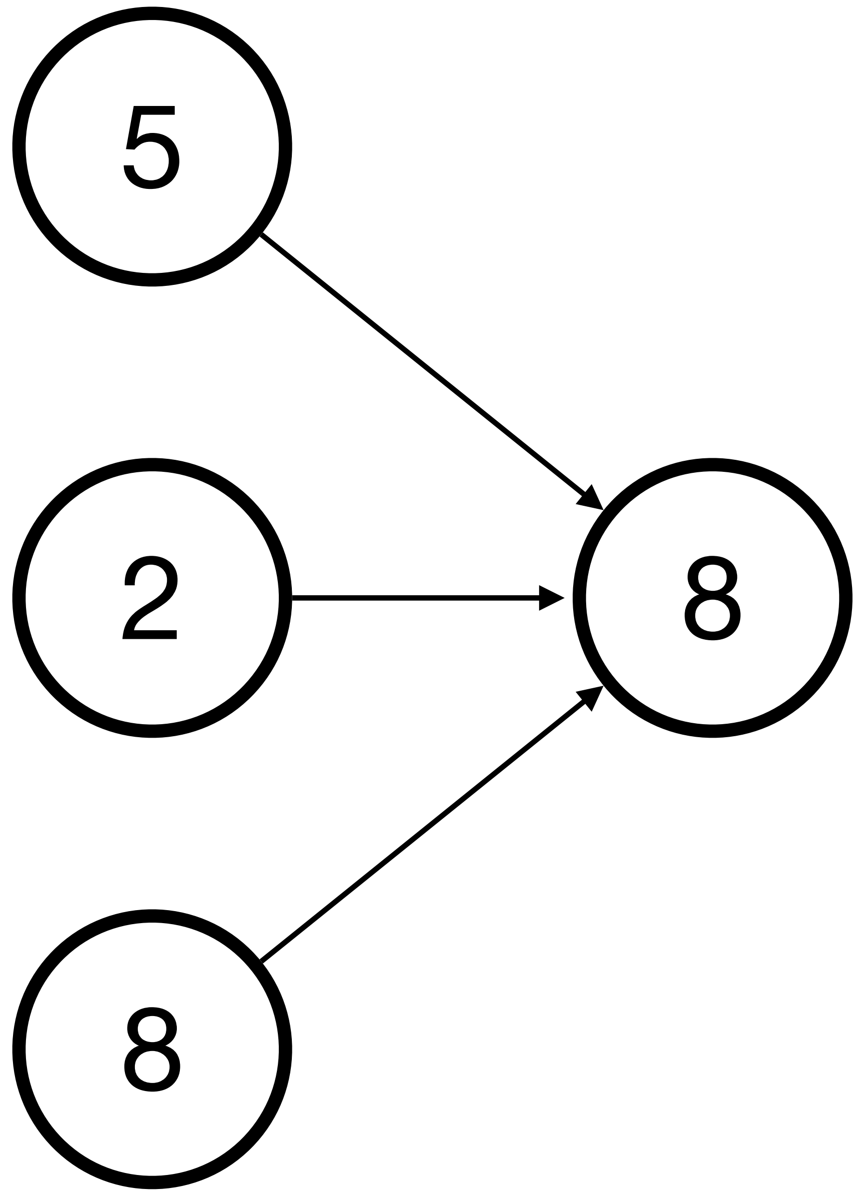
Input size: 320 by 280
Kernel size: 2 by 1
Output size: 319 by 280

	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

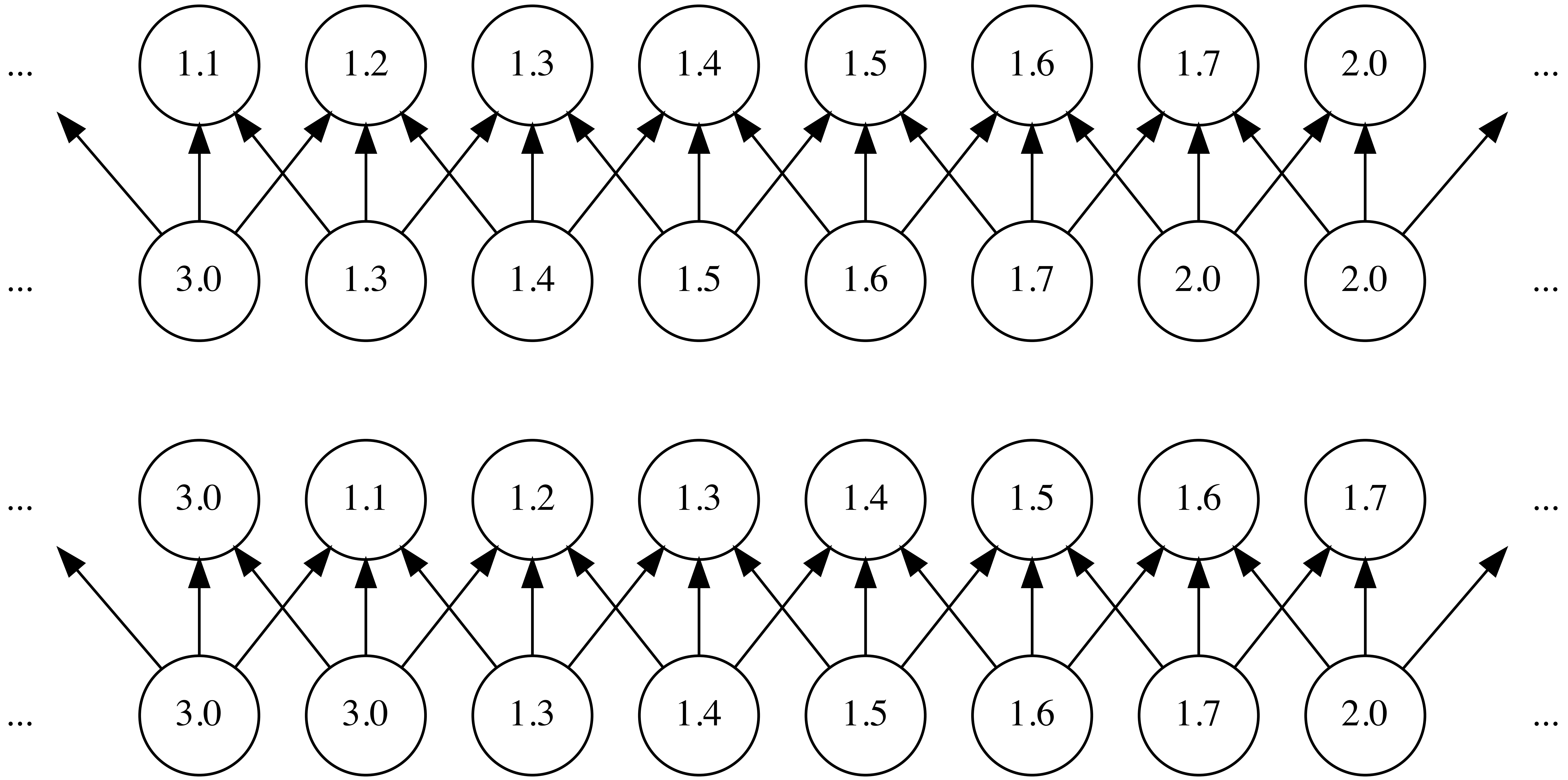
(Goodfellow 2016)

Operation: Pooling

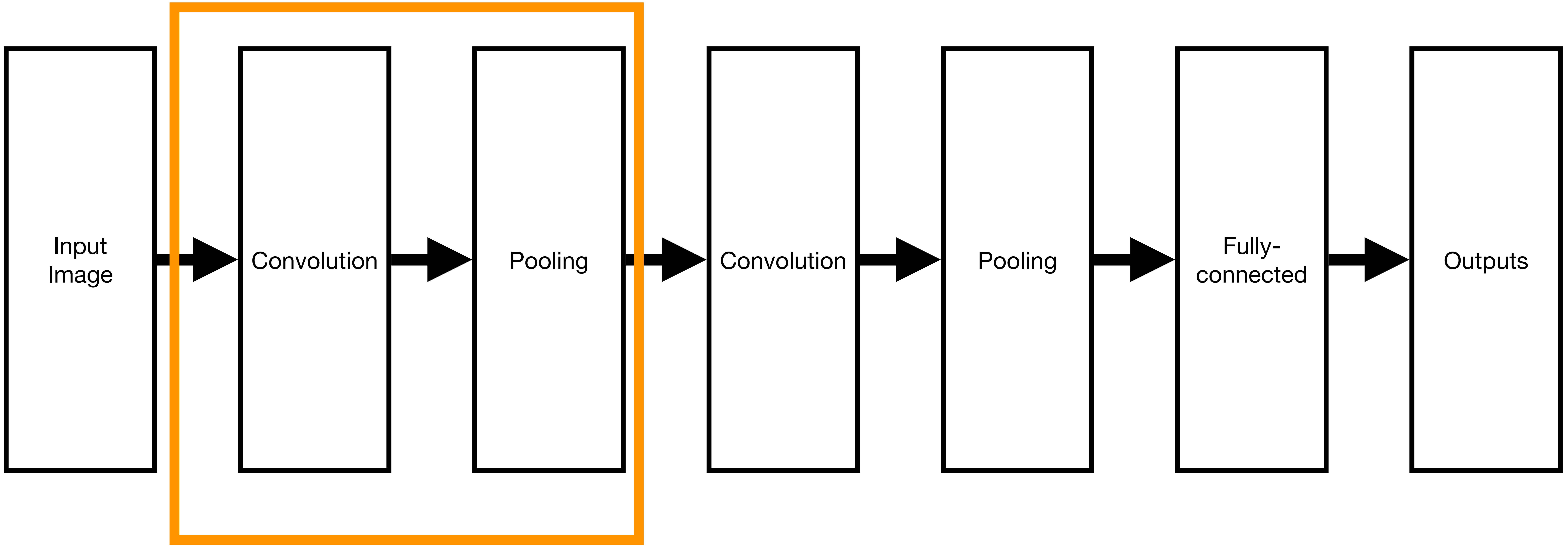
- Pooling **summarizes** its inputs into a single value, e.g.,
  - max
  - average
- Max-pooling is **parameter-free** (no bias or edge weights)



Example:
Translation Equivariance



Typical Architecture



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**