

Introduction to Computer Vision Convolutional Networks (Part 1)

Laurens van der Maaten

Course Schedule

- Four lectures: Mon, Tue, Wed, Thu 10am-noon
- Three labs: Mon, Tue, Wed 2:30-4:30pm
- One Q&A: Thu 2:30-4:30pm
- One quiz: Fri 10am

Course Setup

- I will post relevant course information on CampusWire
- Labs: https://github.com/lvdmaaten/convnet_tutorials
- Project: Kaggle competition (3 weeks) on cassava disease recognition
- Grading: Labs + Exam + Project report (1 page)

Introduction



Introduction

Computer vision tries to get computers to extract information from images.

Human vision

- Can you name some things that influence what we see?

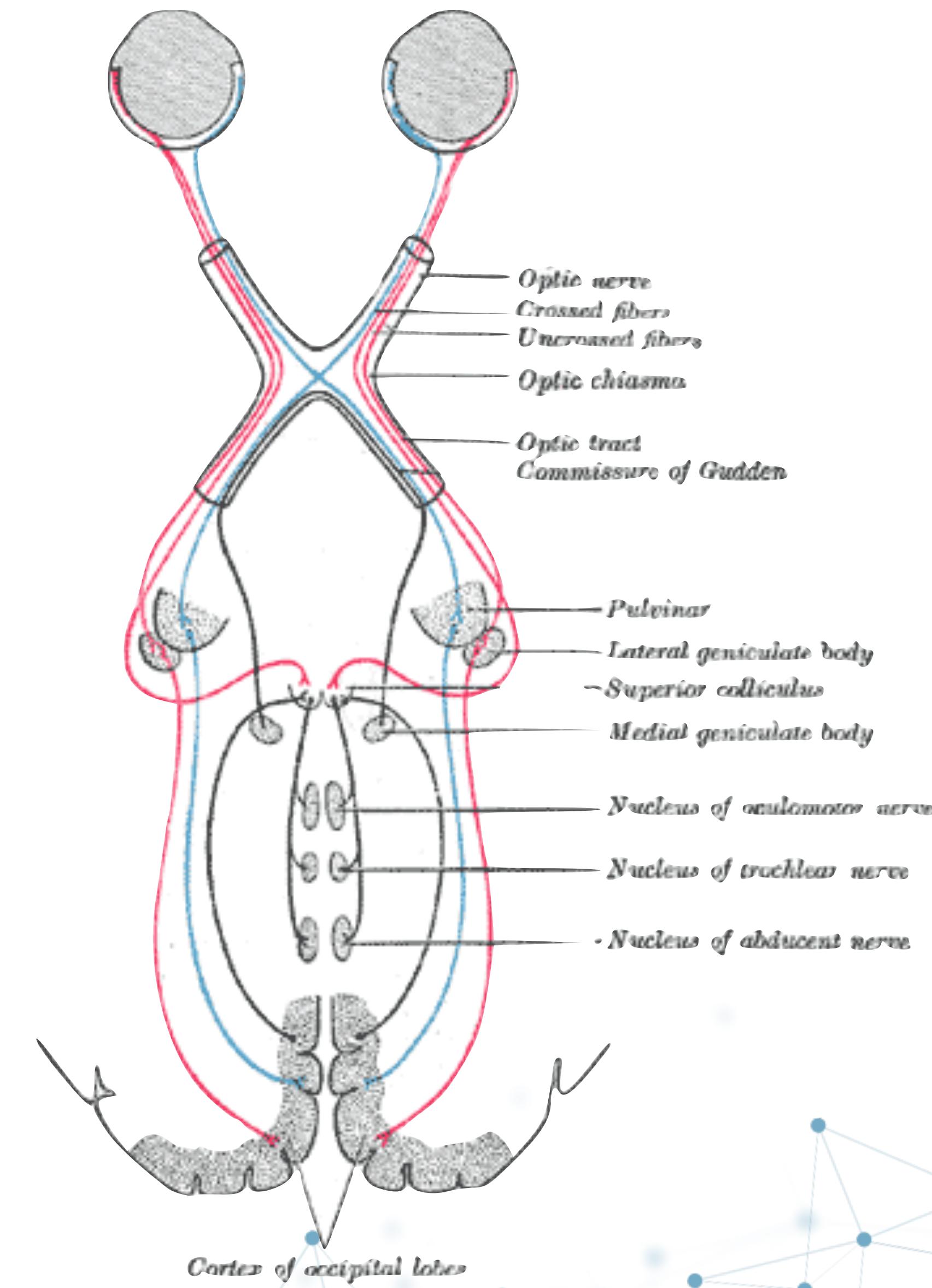
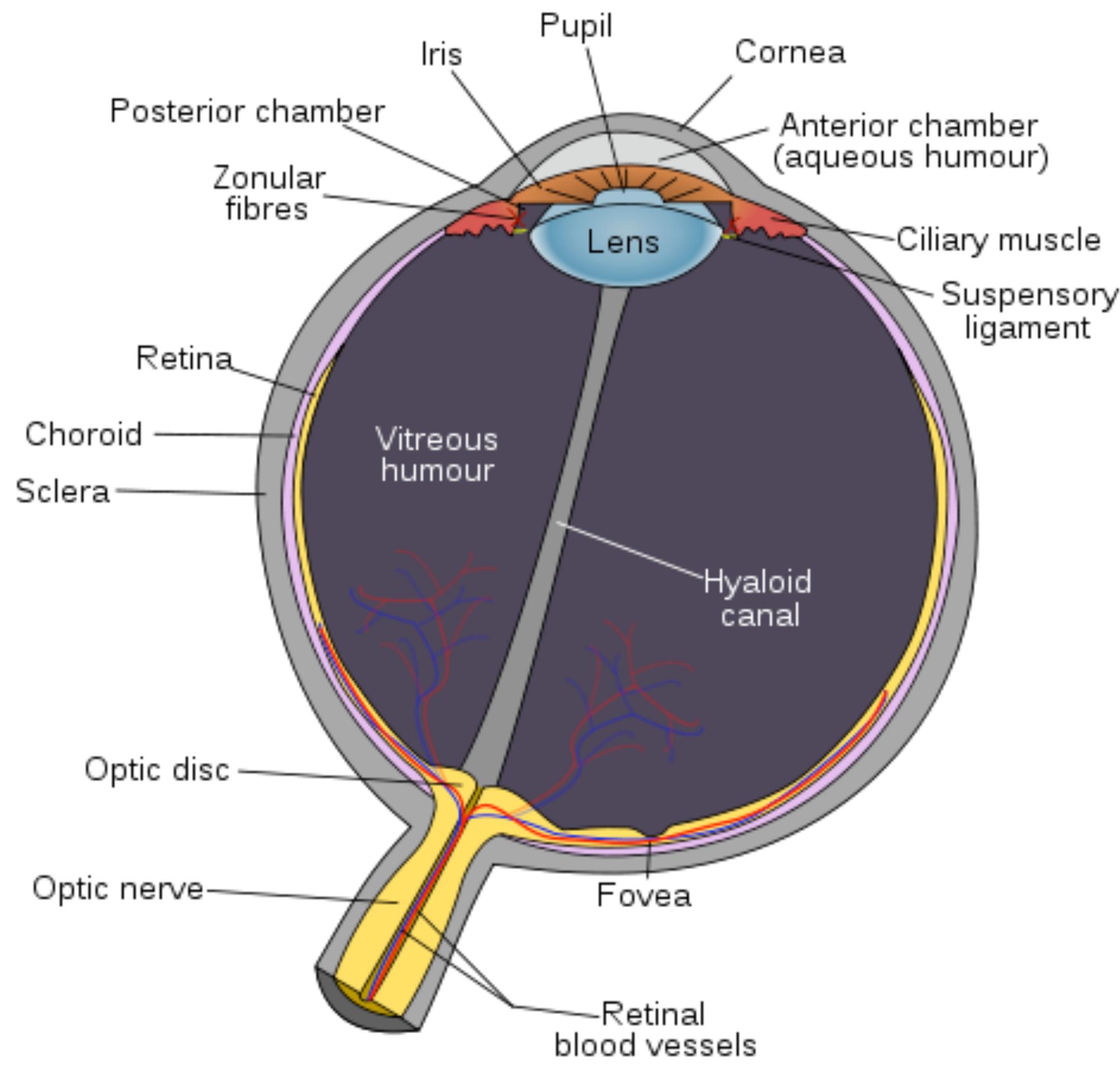
Human vision

- Can you name some things that influence what we see?
 - What objects are where in the world (and how they are deformed)
 - Lighting conditions of the environment
 - Position and orientation of the eyes (viewpoint)

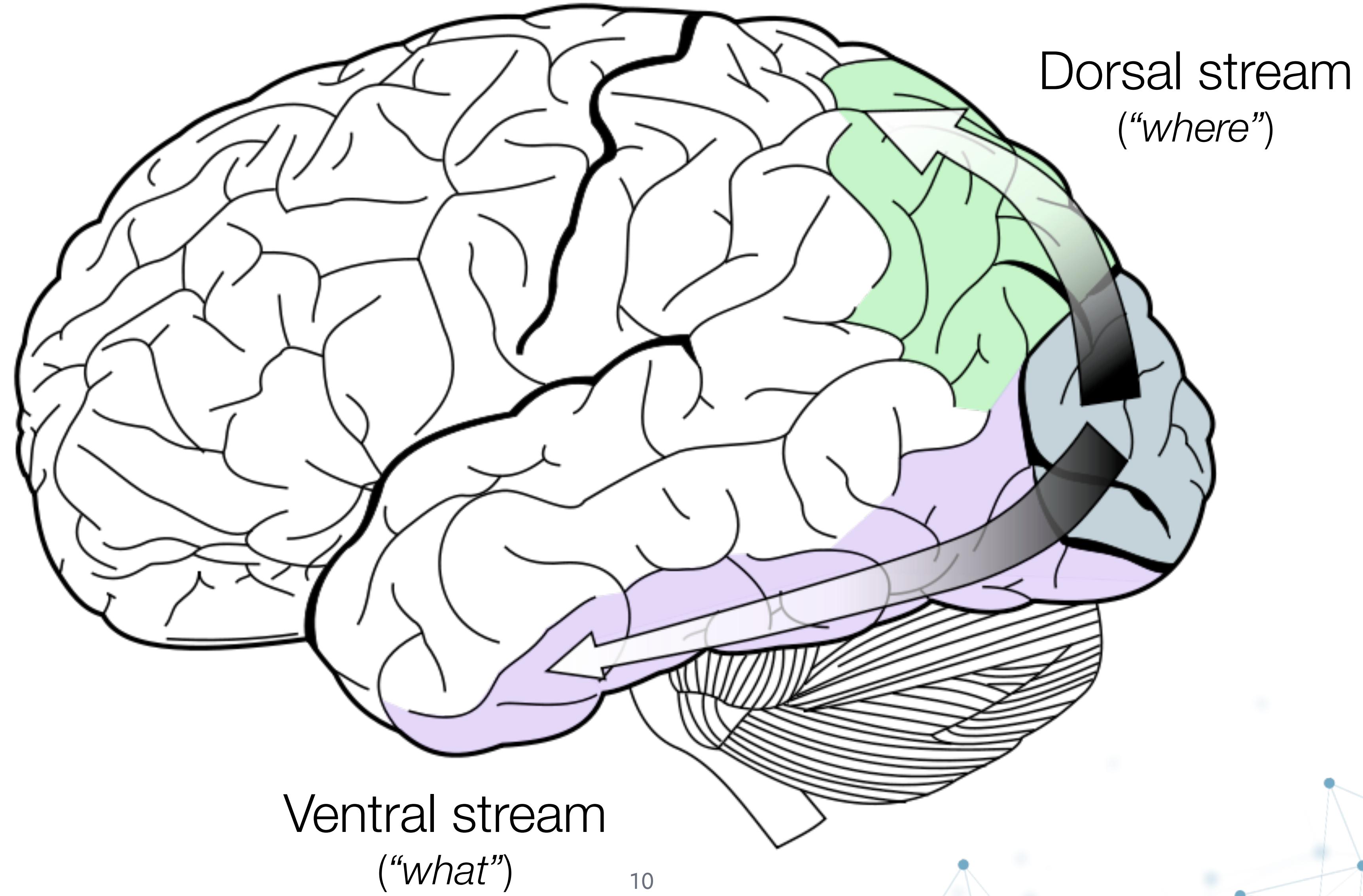
Human vision

- Can you name some things that influence what we see?
 - What objects are where in the world (and how they are deformed)
 - Lighting conditions of the environment
 - Position and orientation of the eyes (viewpoint)
 - Your brain!!!

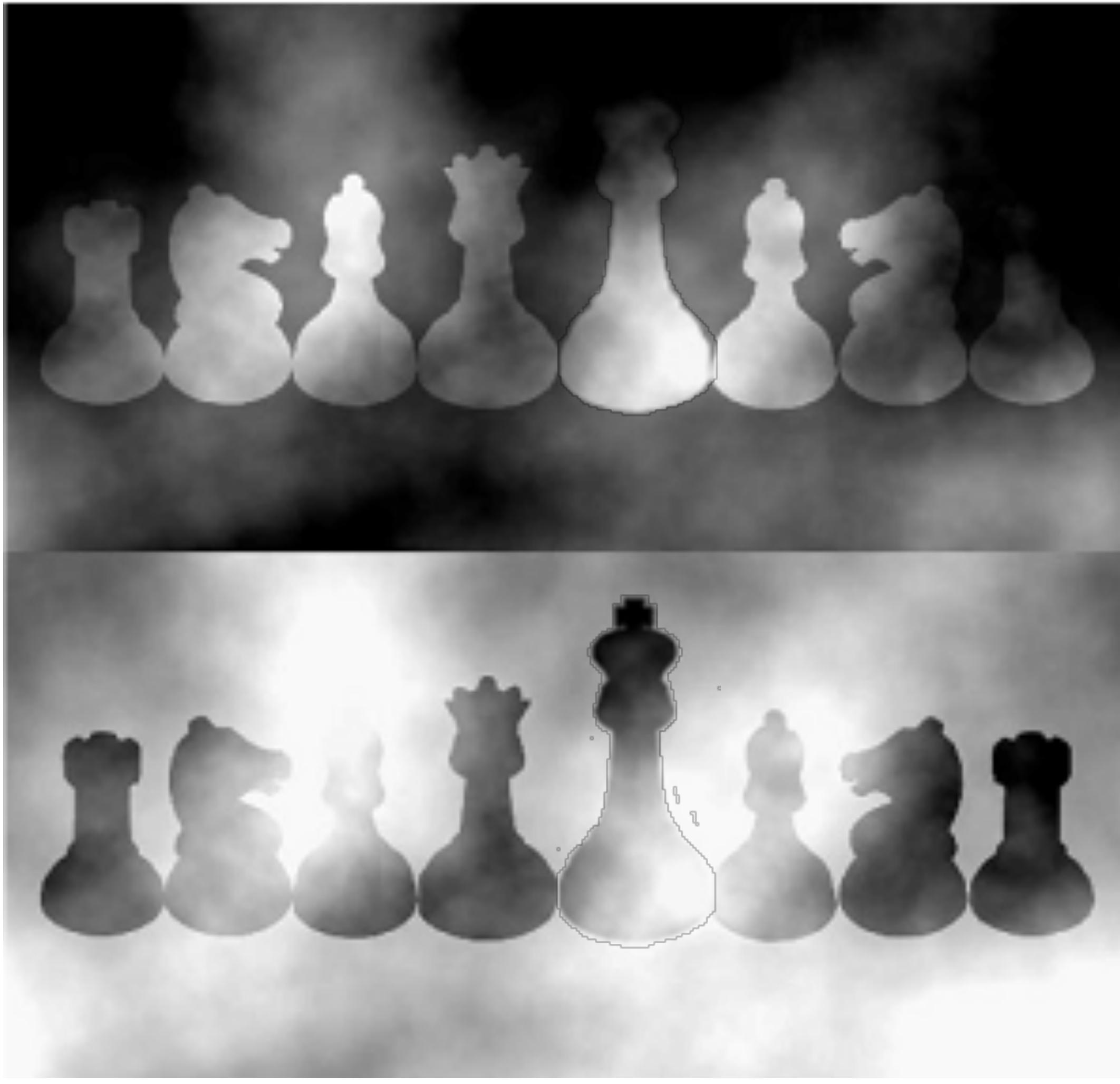
Human vision



Human vision



Human vision



Human vision

X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X

Human vision

O	X	O	X	O	X	X
X	O	X	X	X	O	X
O	X	X	O	X	X	O
X	X	O	X	O	O	X
O	X	X	O	X	X	X
X	O	X	X	X	O	X
O	X	X	O	X	X	O
X	O	X	X	X	O	X
X	X	X	O	O	X	X
X	O	X	X	X	O	X

Human vision



Computer vision

- Vision turns out to be very difficult:
 - Your brain needs 25% of the cortex to solve vision
- Main problem: Vision needs to deal with enormous variations in signal
 - Some of these variations are relevant and others should be ignored

Computer vision

- How do we describe variations within the class “chair”?



- Invariance to some variations can be handcrafted
- We will *learn* invariance to the remaining variations from examples

Computer vision



Image classification



Image classification



tench



alligator lizard



bullet train



agaric



artichoke



bride and groom

Image classification

- Train a **neural network** on **millions** of **annotated images**



Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)}$$

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)}$$



input data

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), \boxed{y})]_{p(\mathbf{x}, y)}$$


label

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)}$$



model parameters

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)}$$



model output

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)}$$



loss function

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)}$$

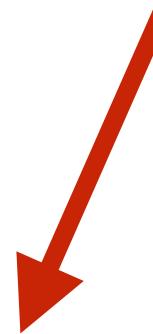


expected loss

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)}$$



data distribution

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)} \approx \frac{1}{N} \sum_{n=1}^N \ell(f(\mathbf{x}_n; \mathbf{w}), y_n)$$


input sample

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)} \approx \frac{1}{N} \sum_{n=1}^N \ell(f(\mathbf{x}_n; \mathbf{w}), \boxed{y_n})$$


sample label

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)} \approx \boxed{\frac{1}{N} \sum_{n=1}^N \ell(f(\mathbf{x}_n; \mathbf{w}), y_n)}$$



empirical loss

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

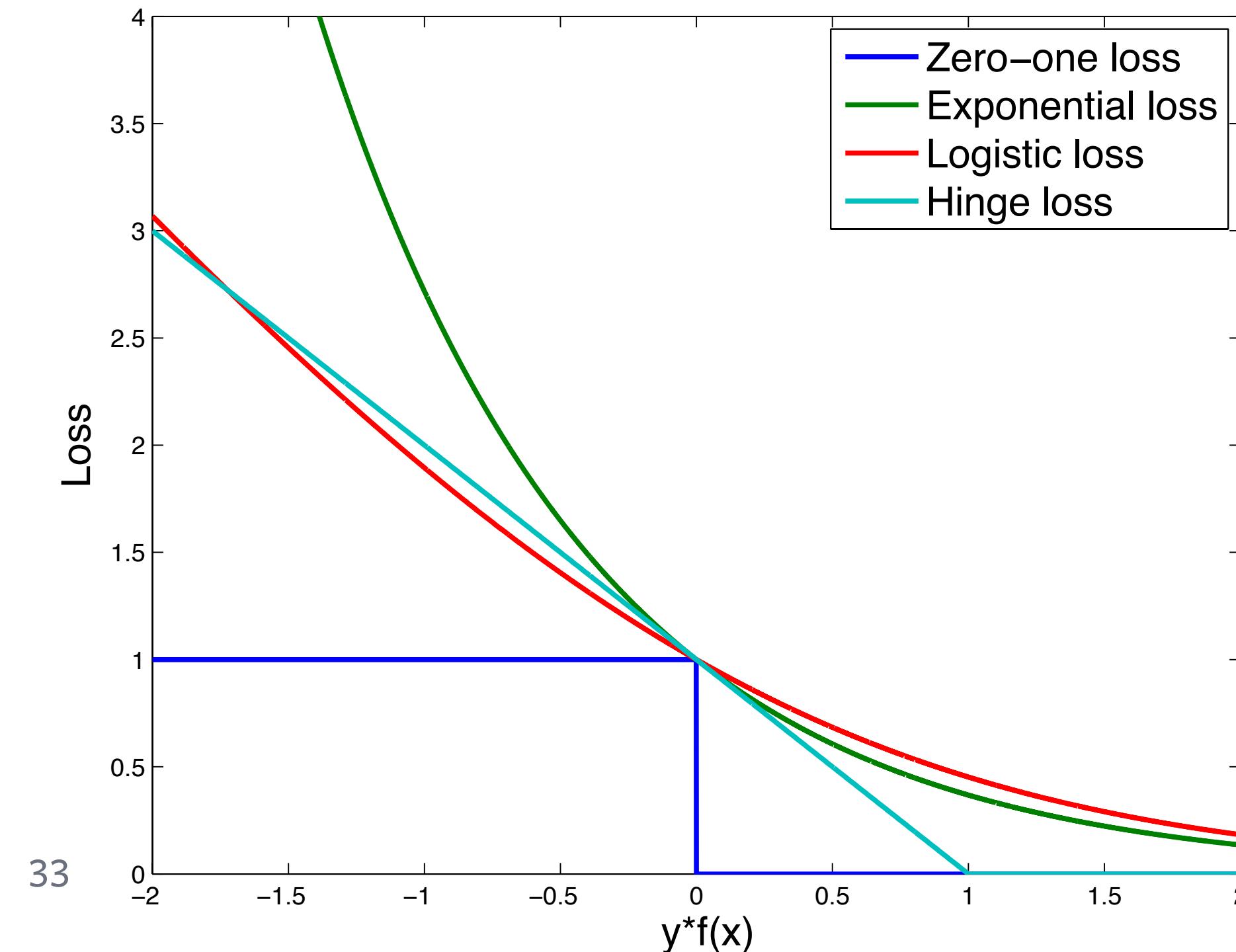
$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)} \approx \frac{1}{N} \sum_{n=1}^N \ell(f(\mathbf{x}_n; \mathbf{w}), y_n) = \mathcal{L}(\mathcal{D}; \mathbf{w})$$

Empirical Risk Minimization

- Learn model based on training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ by minimizing:

$$\mathbb{E}[\ell(f(\mathbf{x}; \mathbf{w}), y)]_{p(\mathbf{x}, y)} \approx \frac{1}{N} \sum_{n=1}^N \ell(f(\mathbf{x}_n; \mathbf{w}), y_n) = \mathcal{L}(\mathcal{D}; \mathbf{w})$$

- Examples of **loss functions**:



* We assume the label y is -1 or +1.

Empirical Risk Minimization

- A **logistic regressor** is an ERM model with two properties:
 - The model is linear: $y' = f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$
 - The loss function is the logistic loss*: $\ell(y', y) = \log(1 + \exp(-y'y))$

* We assume the label y is -1 or +1.

Empirical Risk Minimization

- A **logistic regressor** is an ERM model with two properties:
 - The model is linear: $y' = f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$
 - The loss function is the logistic loss*: $\ell(y', y) = \log(1 + \exp(-y'y))$
- In the **multi-class** setting, logistic regression can be written as**:

$$\begin{aligned}\ell(f(\mathbf{x}; \mathbf{w}), y) &= -\log \frac{\exp(\mathbf{y}^\top f(\mathbf{x}; \mathbf{w}))}{\sum_{\mathbf{y}'} \exp(\mathbf{y}'^\top f(\mathbf{x}; \mathbf{w}))} \\ &= -\mathbf{y}^\top f(\mathbf{x}; \mathbf{w}) + \log \sum_{\mathbf{y}'} \exp(\mathbf{y}'^\top f(\mathbf{x}; \mathbf{w}))\end{aligned}$$

* We assume the label y is -1 or +1.

** We assume the label y is a one-hot vector.

Empirical Risk Minimization

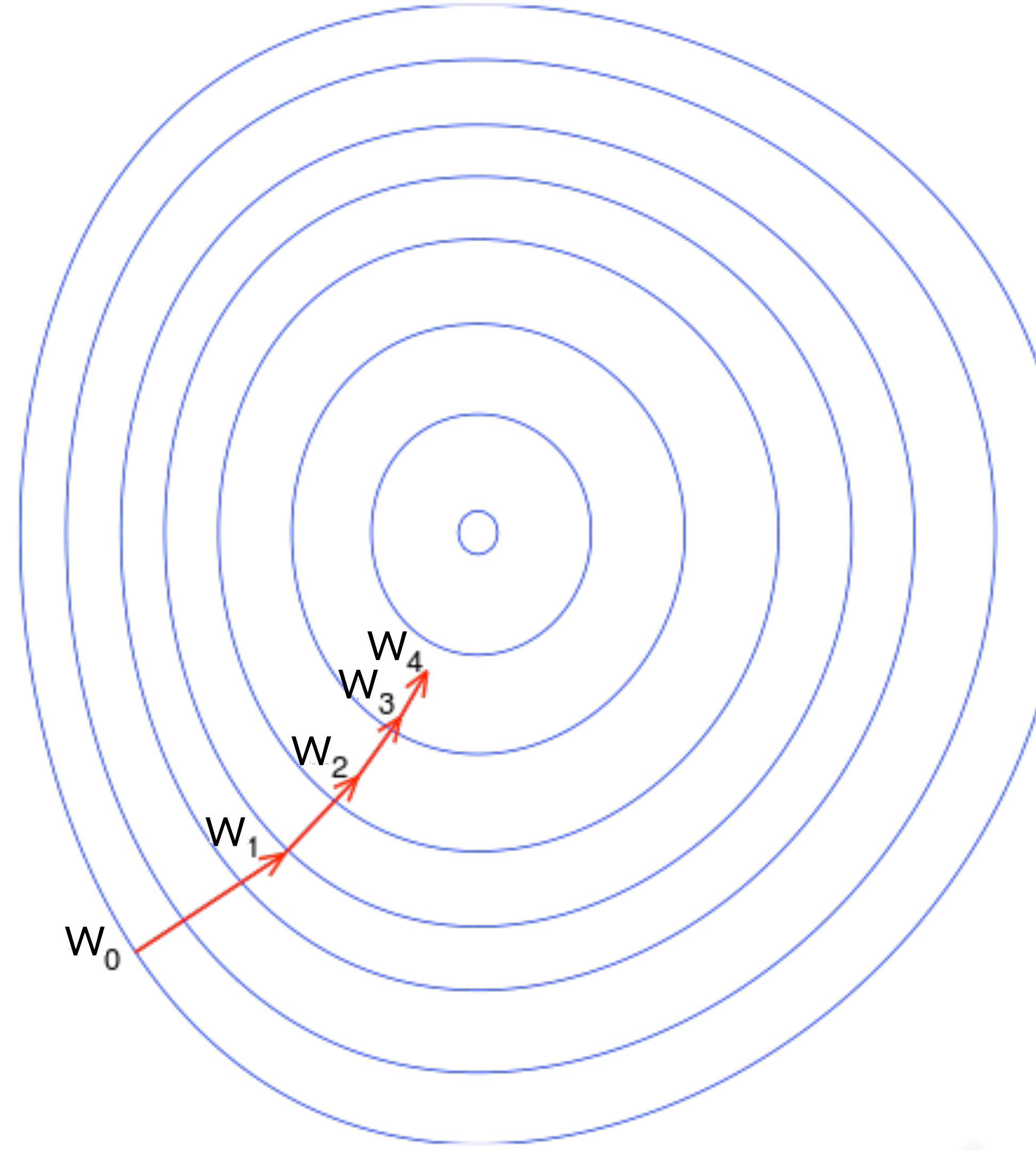
- How do we **minimize** the empirical loss?

Empirical Risk Minimization

- How do we **minimize** the empirical loss?
- **Gradient descent!** At each iteration, do a small step in the direction of the loss gradient:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{N} \sum_{n=1}^N \frac{\partial \ell(f(\mathbf{x}_n), y_n)}{\partial \mathbf{w}}$$

Gradient descent



Note how the gradient is orthogonal to the isolines.³⁸

Gradient descent

- Recall the **logistic loss** function for linear models:

$$\mathcal{L}(\mathcal{D}; \mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \log(1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n))$$

Gradient descent

- Recall the **logistic loss** function for linear models:

$$\mathcal{L}(\mathcal{D}; \mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \log(1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n))$$

- Derivation of the **gradient**:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= -\frac{1}{N} \sum_{n=1}^N \frac{1}{1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)} \exp(-y_n \mathbf{w}^\top \mathbf{x}_n) \mathbf{x}_n y_n \\ &= -\frac{1}{N} \sum_{n=1}^N \frac{\exp(-y_n \mathbf{w}^\top \mathbf{x}_n)}{1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)} \mathbf{x}_n y_n\end{aligned}$$

Gradient descent

- Recall the **logistic loss** function for linear models:

$$\mathcal{L}(\mathcal{D}; \mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \log(1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n))$$

- Derivation of the **gradient**:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= -\frac{1}{N} \sum_{n=1}^N \frac{1}{1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)} \exp(-y_n \mathbf{w}^\top \mathbf{x}_n) \mathbf{x}_n y_n \\ &= -\frac{1}{N} \sum_{n=1}^N \frac{1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n) - 1}{1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)} \mathbf{x}_n y_n \\ &= -\frac{1}{N} \sum_{n=1}^N (1 - p(y_n = 1 | \mathbf{x}_n)) \mathbf{x}_n y_n\end{aligned}$$

relation to
maximum likelihood
interpretation

The need for speed



Is gradient descent efficient?

- How does the gradient computation **scale** in the amount of data?

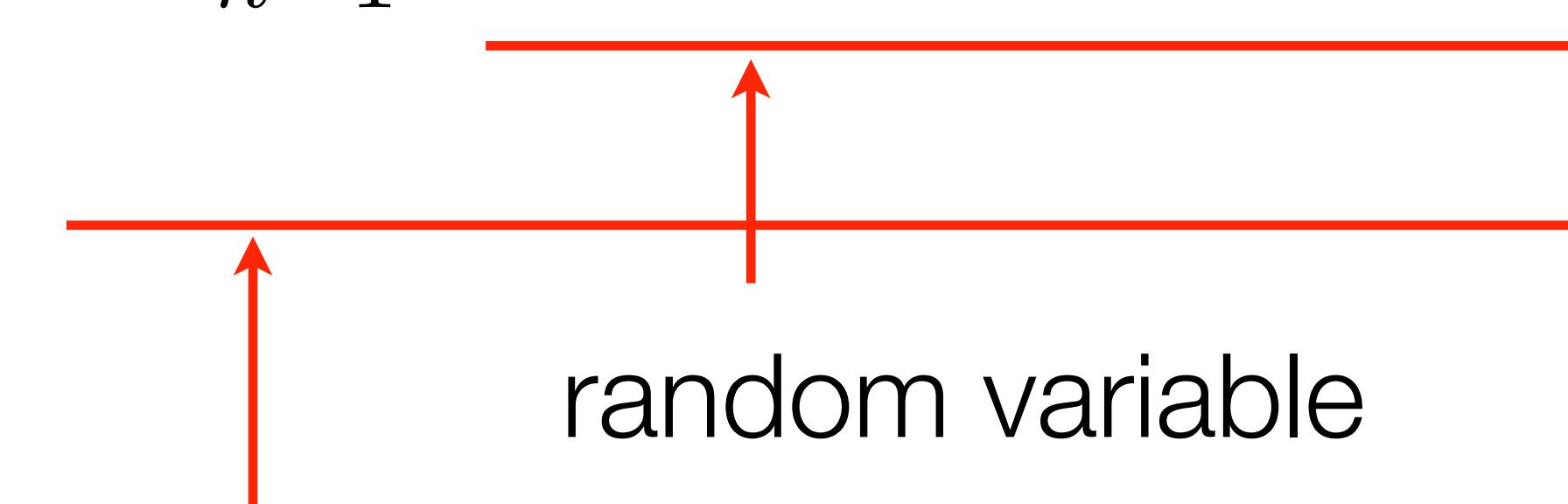
Is gradient descent efficient?

- How does the gradient computation **scale** in the amount of data?
- Do we really need to consider **all data** to get a good search direction?

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = -\frac{1}{N} \sum_{n=1}^N (1 - p(y_n = 1 | \mathbf{x}_n)) \mathbf{x}_n y_n$$

Is gradient descent efficient?

- How does the gradient computation **scale** in the amount of data?
- Do we really need to consider **all data** to get a good search direction?

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = -\frac{1}{N} \sum_{n=1}^N (1 - p(y_n = 1 | \mathbf{x}_n)) \mathbf{x}_n y_n$$


Stochastic gradient descent

- Selects a **random** data point n , and updates parameters using:

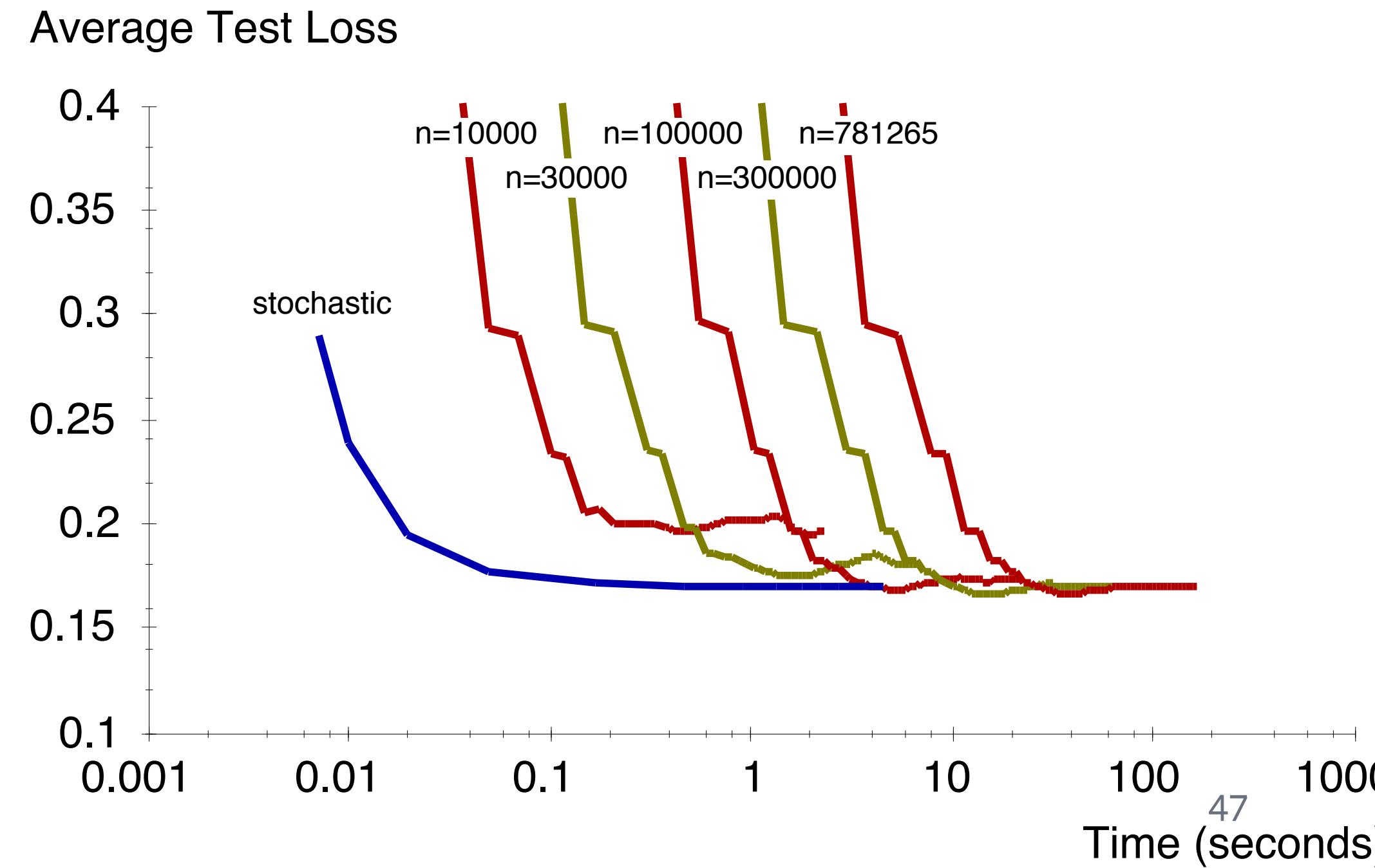
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \frac{\partial L(\mathbf{x}_n, y_n; \mathbf{w})}{\partial \mathbf{w}}$$

Stochastic gradient descent

- Selects a **random** data point n , and updates parameters using:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \frac{\partial L(\mathbf{x}_n, y_n; \mathbf{w})}{\partial \mathbf{w}}$$

- This minimization technique converges **much faster** in practice:



Stochastic gradient descent

- In practice, we often use **mini-batch SGD**. Why?

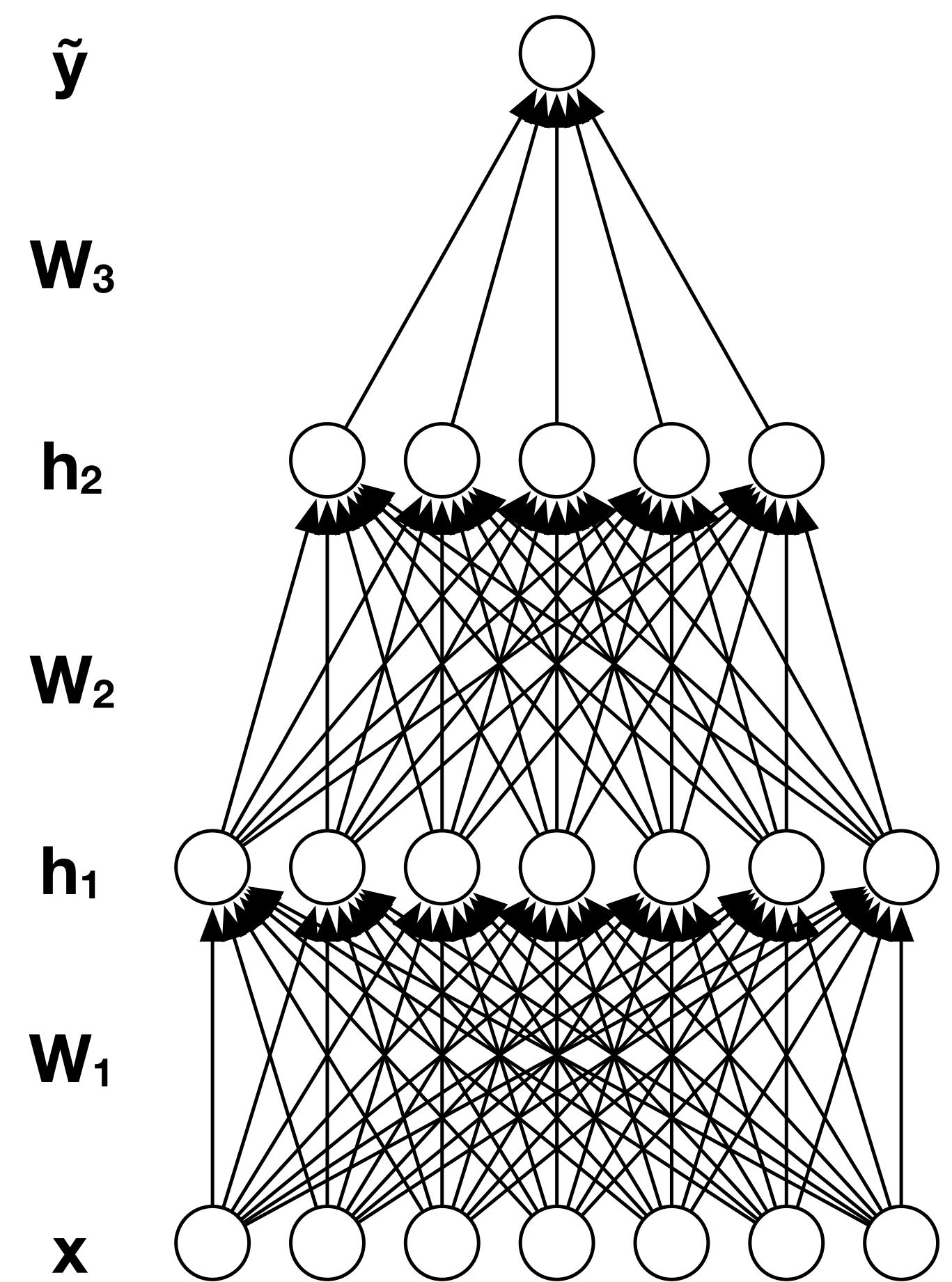
Stochastic gradient descent

- In practice, we often use **mini-batch SGD**. Why?



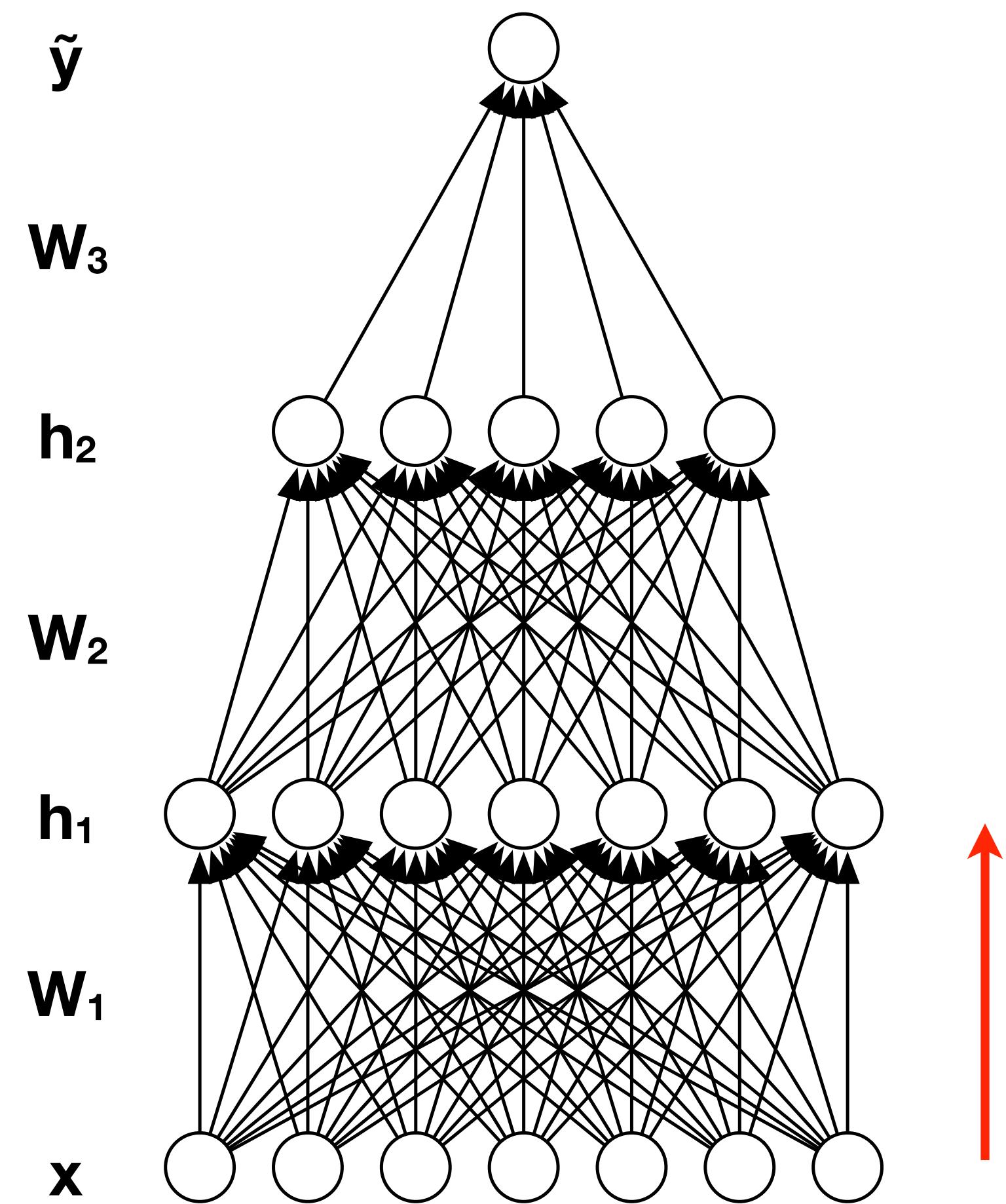
Multilayer networks

- Multilayer networks repeatedly apply linear and nonlinear transforms:



Multilayer networks

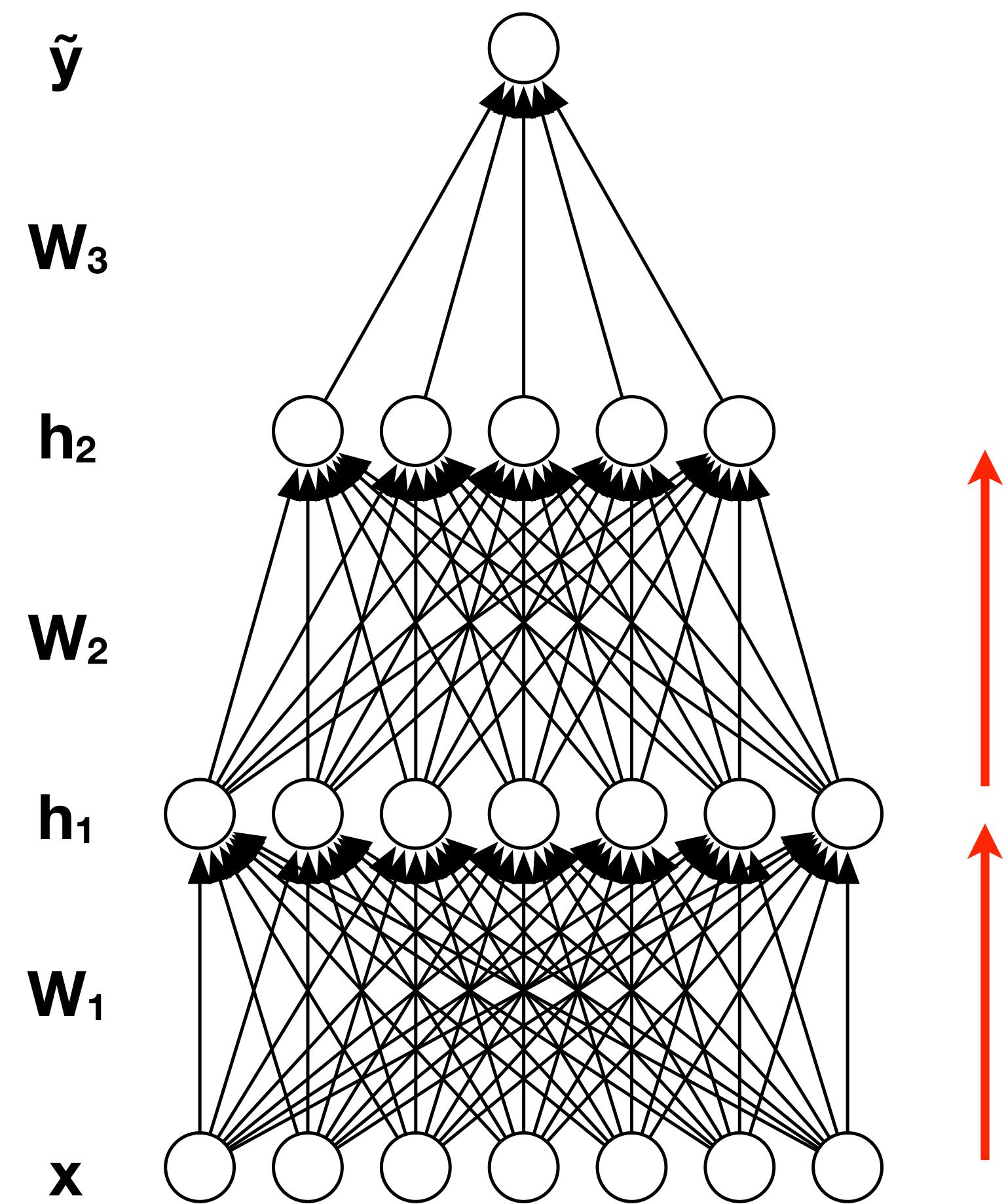
- Multilayer networks repeatedly apply linear and nonlinear transforms:



$$\mathbf{h}_1 = f_1(\mathbf{W}_1^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{W}_1^T \mathbf{x})}$$

Multilayer networks

- Multilayer networks repeatedly apply linear and nonlinear transforms:

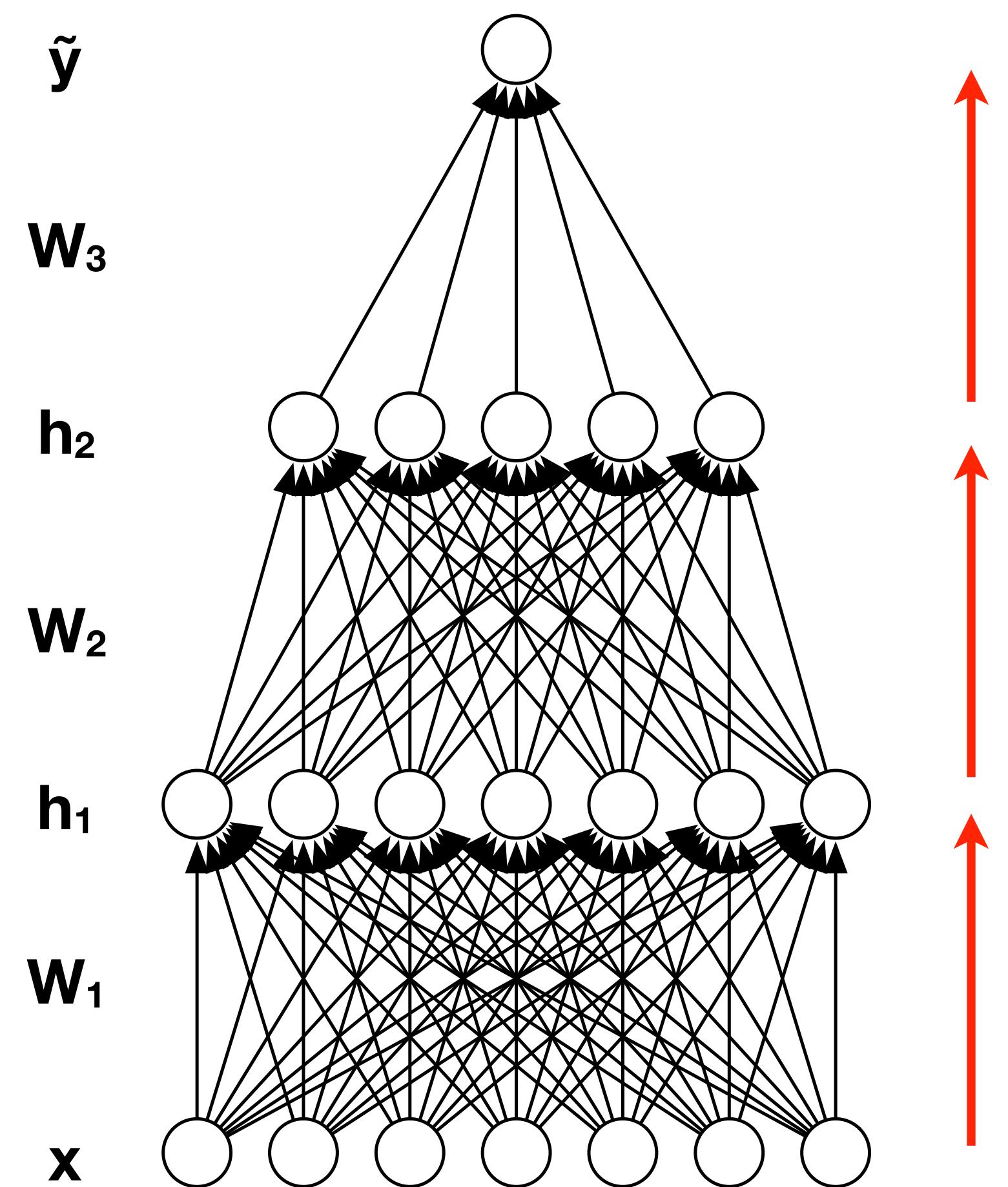


$$\mathbf{h}_2 = f_2(\mathbf{W}_2^T \mathbf{h}_1) = \frac{1}{1 + \exp(-\mathbf{W}_2^T \mathbf{h}_1)}$$

$$\mathbf{h}_1 = f_1(\mathbf{W}_1^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{W}_1^T \mathbf{x})}$$

Multilayer networks

- Multilayer networks repeatedly apply linear and nonlinear transforms:



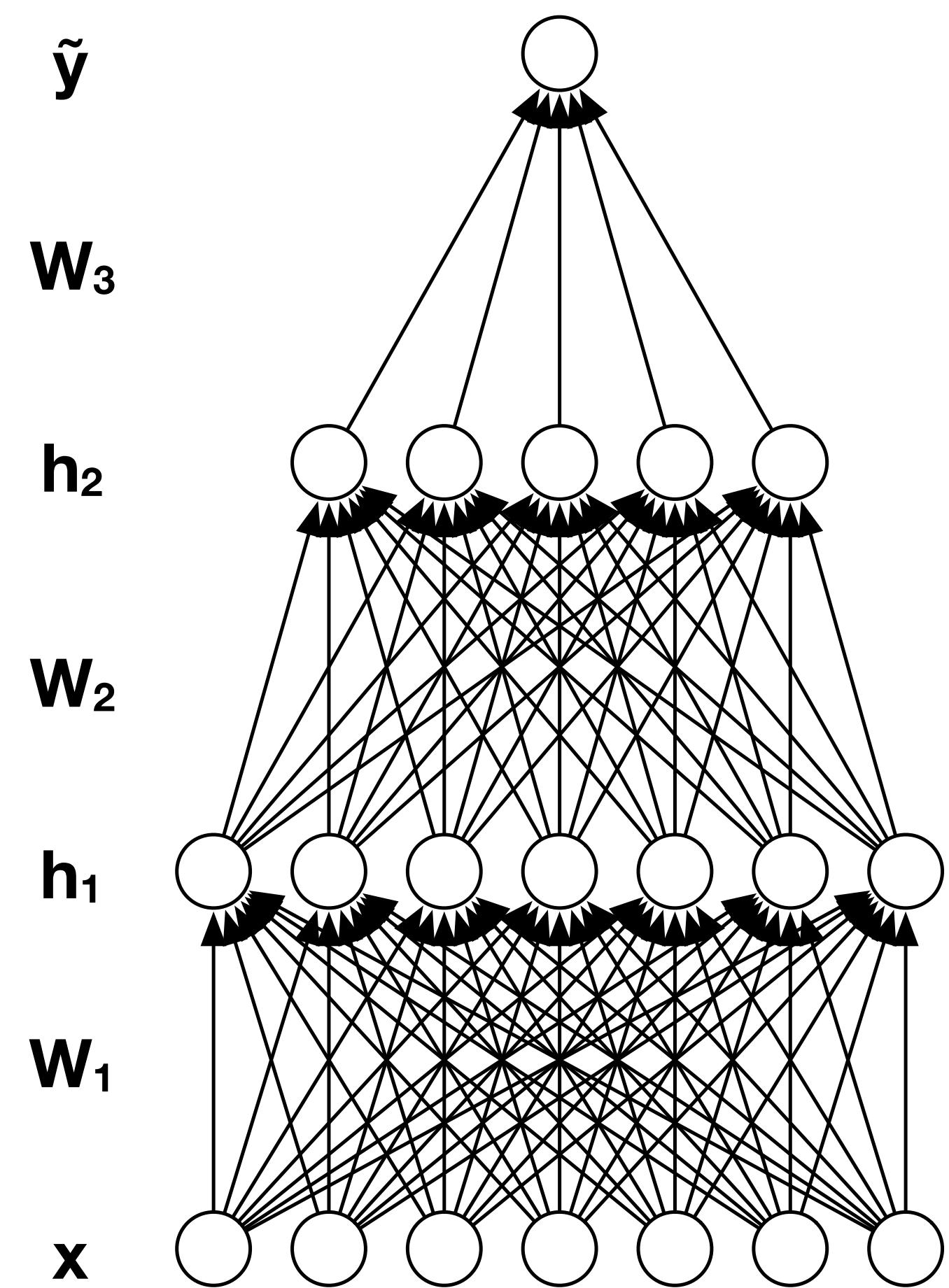
$$\tilde{y} = f_3(\mathbf{W}_3^T \mathbf{h}_2) = \frac{1}{1 + \exp(-\mathbf{W}_3^T \mathbf{h}_2)}$$

$$\mathbf{h}_2 = f_2(\mathbf{W}_2^T \mathbf{h}_1) = \frac{1}{1 + \exp(-\mathbf{W}_2^T \mathbf{h}_1)}$$

$$\mathbf{h}_1 = f_1(\mathbf{W}_1^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{W}_1^T \mathbf{x})}$$

Backpropagation

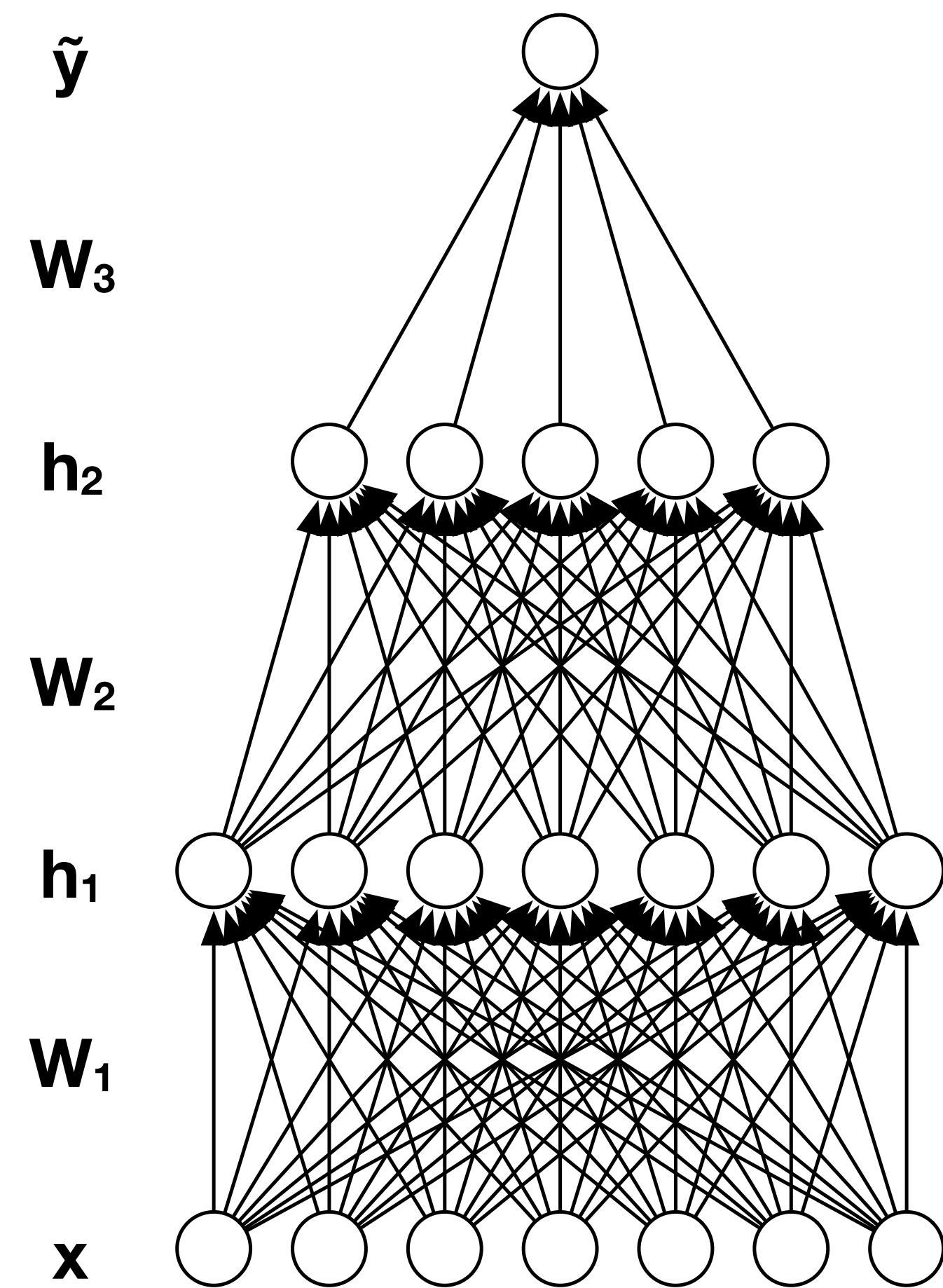
- Backpropagation uses **chain rule** to compute **loss gradient**:



Assume quadratic loss: $C = \frac{1}{2}(y - \tilde{y})^2$

Backpropagation

- Backpropagation uses **chain rule** to compute **loss gradient**:



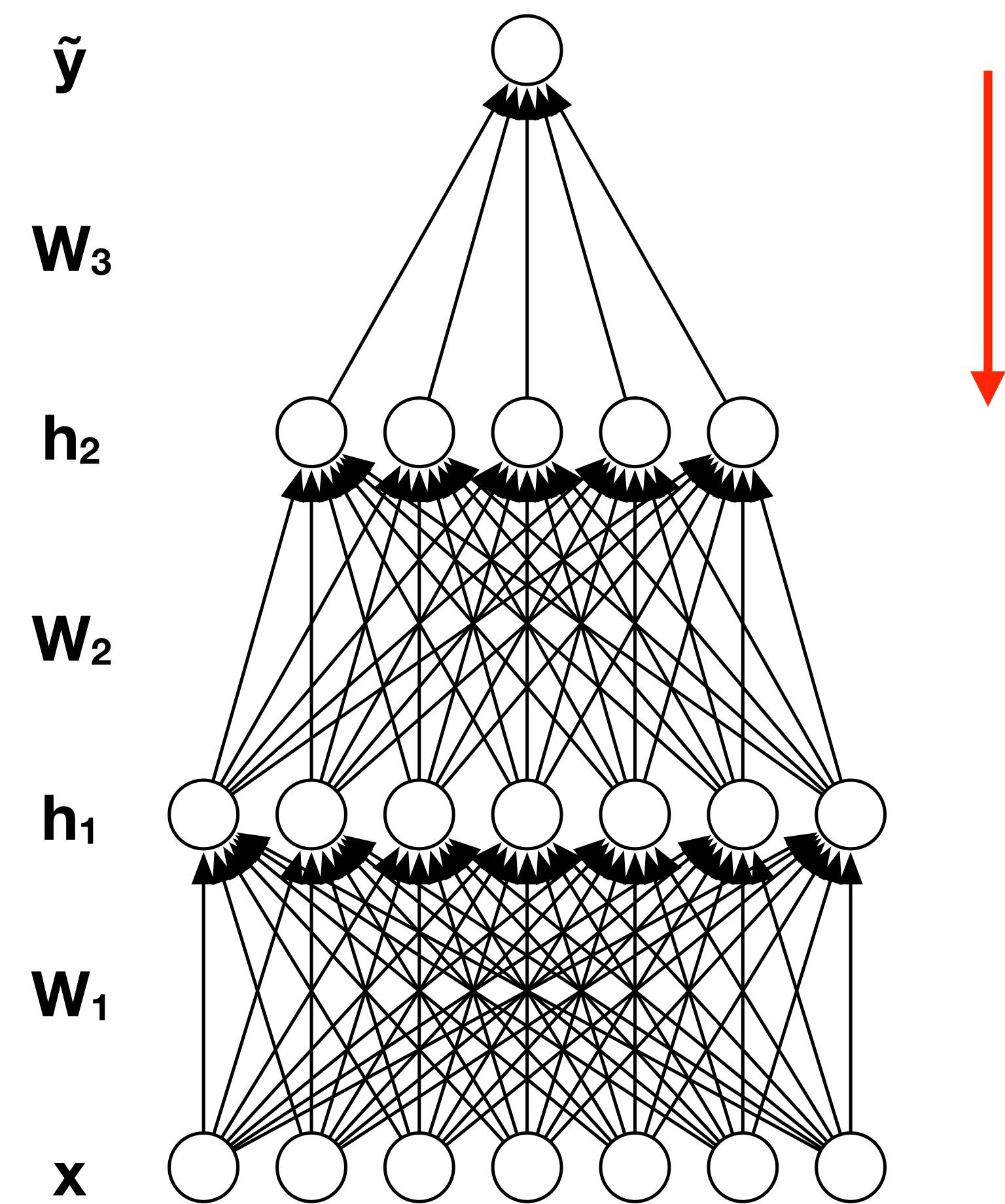
Assume quadratic loss: $C = \frac{1}{2}(y - \tilde{y})^2$

Backpropagation proceeds in two steps:

1. **Propagate** the error signal
2. Compute the associated **weight gradient**

Backpropagation

- Backpropagation uses **chain rule** to compute **loss gradient**:

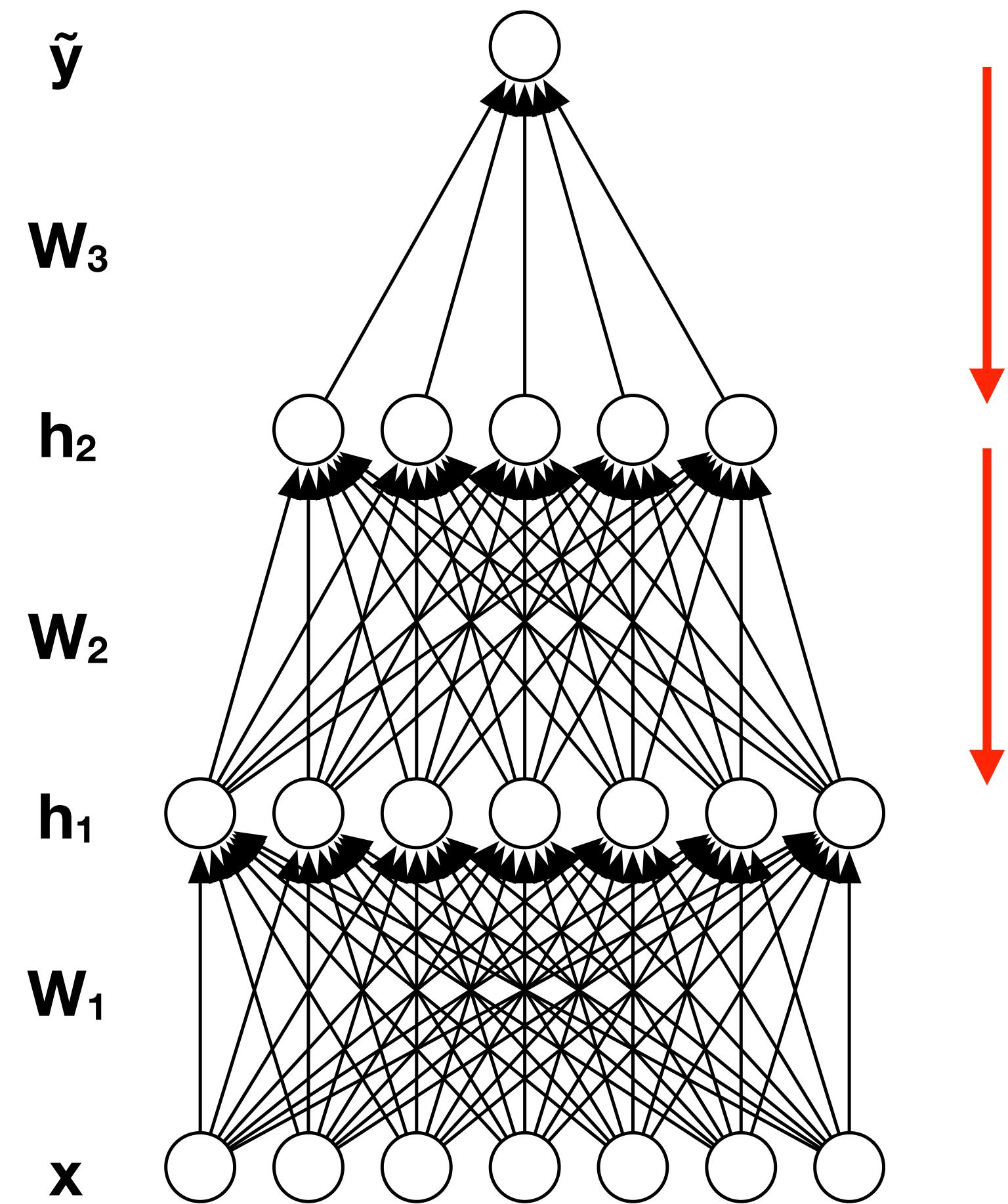


$$e_3 = (y - \tilde{y}) \frac{\partial f_3}{\partial \mathbf{W}_3^T \mathbf{h}_2} = (y - \tilde{y}) \tilde{y}(1 - \tilde{y})$$

$$\frac{\partial C}{\partial \mathbf{W}_3} = e_3 \mathbf{h}_2$$

Backpropagation

- Backpropagation uses **chain rule** to compute **loss gradient**:



$$e_3 = (y - \tilde{y}) \frac{\partial f_3}{\partial \mathbf{W}_3^T \mathbf{h}_2} = (y - \tilde{y}) \tilde{y}(1 - \tilde{y})$$

$$\frac{\partial C}{\partial \mathbf{W}_3} = e_3 \mathbf{h}_2$$

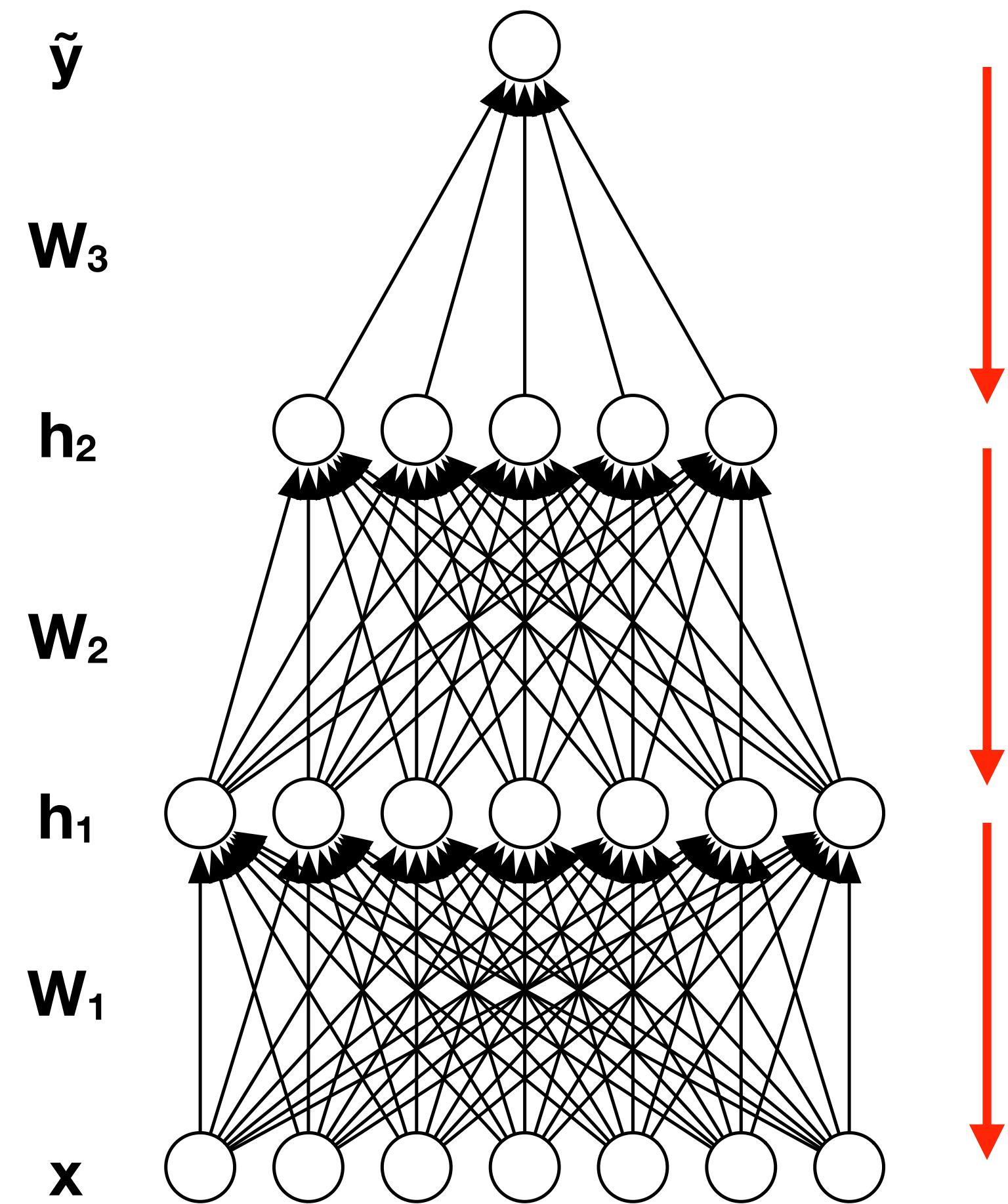
$$\mathbf{e}_2 = \mathbf{h}_2 \circ (1 - \mathbf{h}_2) \circ (\mathbf{W}_3 e_3)$$

$$\frac{\partial C}{\partial \mathbf{W}_2} = \mathbf{h}_1 \mathbf{e}_2^\top$$

$$\mathbf{e}_1 = \mathbf{h}_1 \circ (1 - \mathbf{h}_1) \circ (\mathbf{W}_2 \mathbf{e}_2)$$

Backpropagation

- Backpropagation uses **chain rule** to compute **loss gradient**:



$$e_3 = (y - \hat{y}) \frac{\partial f_3}{\partial \mathbf{W}_3^T \mathbf{h}_2} = (y - \hat{y}) \hat{y}(1 - \hat{y})$$

$$\frac{\partial C}{\partial \mathbf{W}_3} = e_3 \mathbf{h}_2$$

$$\mathbf{e}_2 = \mathbf{h}_2 \circ (1 - \mathbf{h}_2) \circ (\mathbf{W}_3 e_3)$$

$$\frac{\partial C}{\partial \mathbf{W}_2} = \mathbf{h}_1 \mathbf{e}_2^\top$$

$$\mathbf{e}_1 = \mathbf{h}_1 \circ (1 - \mathbf{h}_1) \circ (\mathbf{W}_2 \mathbf{e}_2)$$

$$\frac{\partial C}{\partial \mathbf{W}_1} = \mathbf{x} \mathbf{e}_1^\top$$

Multilayer networks

- Suppose we wanted to train a multilayer network to recognize images
- What would go wrong?

Multilayer networks

- Suppose we wanted to train a multilayer network to recognize images
- What would go wrong?

**Suppose the image has 256x256 RGB pixels,
and we have 1,000 classes.**

How many parameters would a single layer have?

Multilayer networks

- Suppose we wanted to train a multilayer network to recognize images
- What would go wrong?

**Suppose the image has 256x256 RGB pixels,
and we have 1,000 classes.**

How many parameters would a single layer have?

$$3 \times 256 \times 256 \times 1,000 = 196,608,000$$

Convolutional networks

- You can think of each column in a linear layer as a “filter” for the image
- Do different parts of an image have to be filtered differently?

Convolutional networks

- You can think of each column in a linear layer as a “filter” for the image
- Do different parts of an image have to be filtered differently? **No!**

Convolution

- Discrete convolution*: $(f \star g)[i] = \sum_{k=1}^K f[i+k]g[k]$

* Technically, this operation is called **cross-correlation**.
Most machine learning work uses the two interchangeably.

Convolution

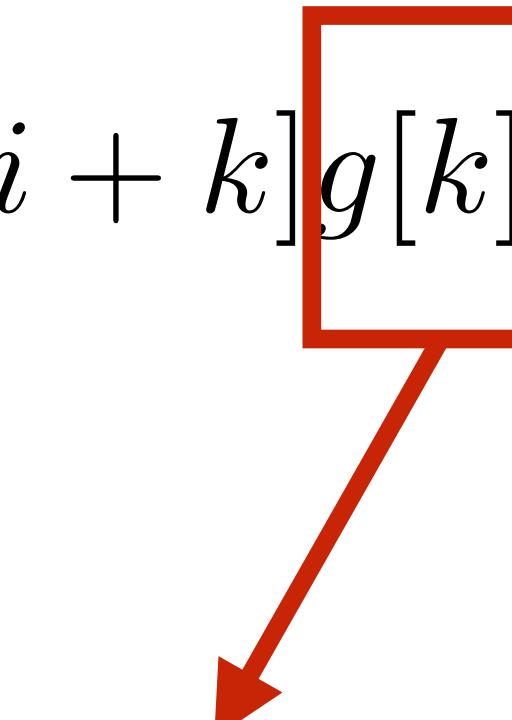
- Discrete convolution*: $(f \star g)[i] = \sum_{k=1}^K f[i+k]g[k]$

$$\sum_{k=1}^K f[i+k]g[k]$$

value in
input signal

* Technically, this operation is called **cross-correlation**.
Most machine learning work uses the two interchangeably.

Convolution

- Discrete convolution*: $(f \star g)[i] = \sum_{k=1}^K f[i + k]g[k]$
- 
- value in
filter ("kernel")**

* Technically, this operation is called **cross-correlation**.
Most machine learning work uses the two interchangeably.

Convolution

- Discrete convolution*: $(f \star g)[i] = \sum_{k=1}^K f[i + k]g[k]$

$$(f \star g)$$

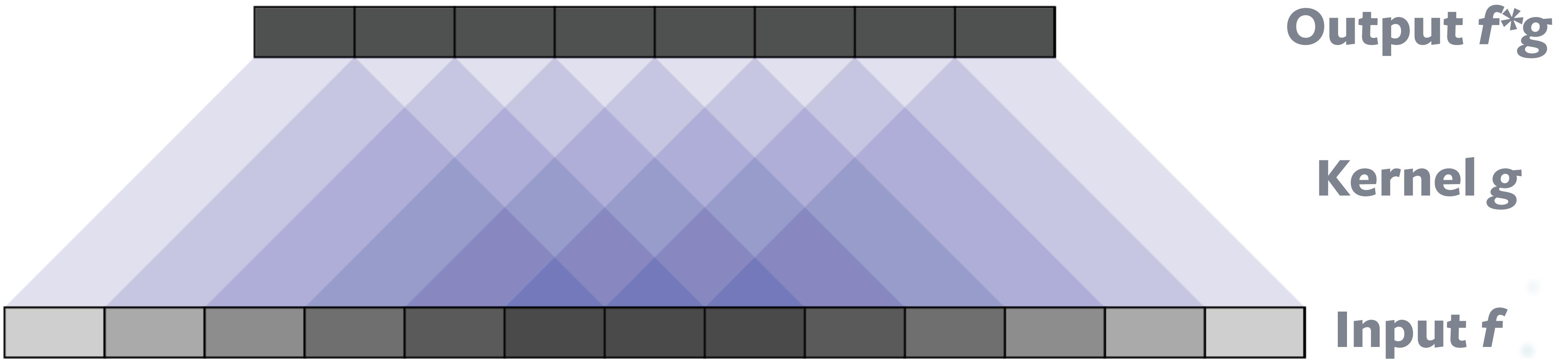


**value in
output**

* Technically, this operation is called **cross-correlation**.
Most machine learning work uses the two interchangeably.

Convolution

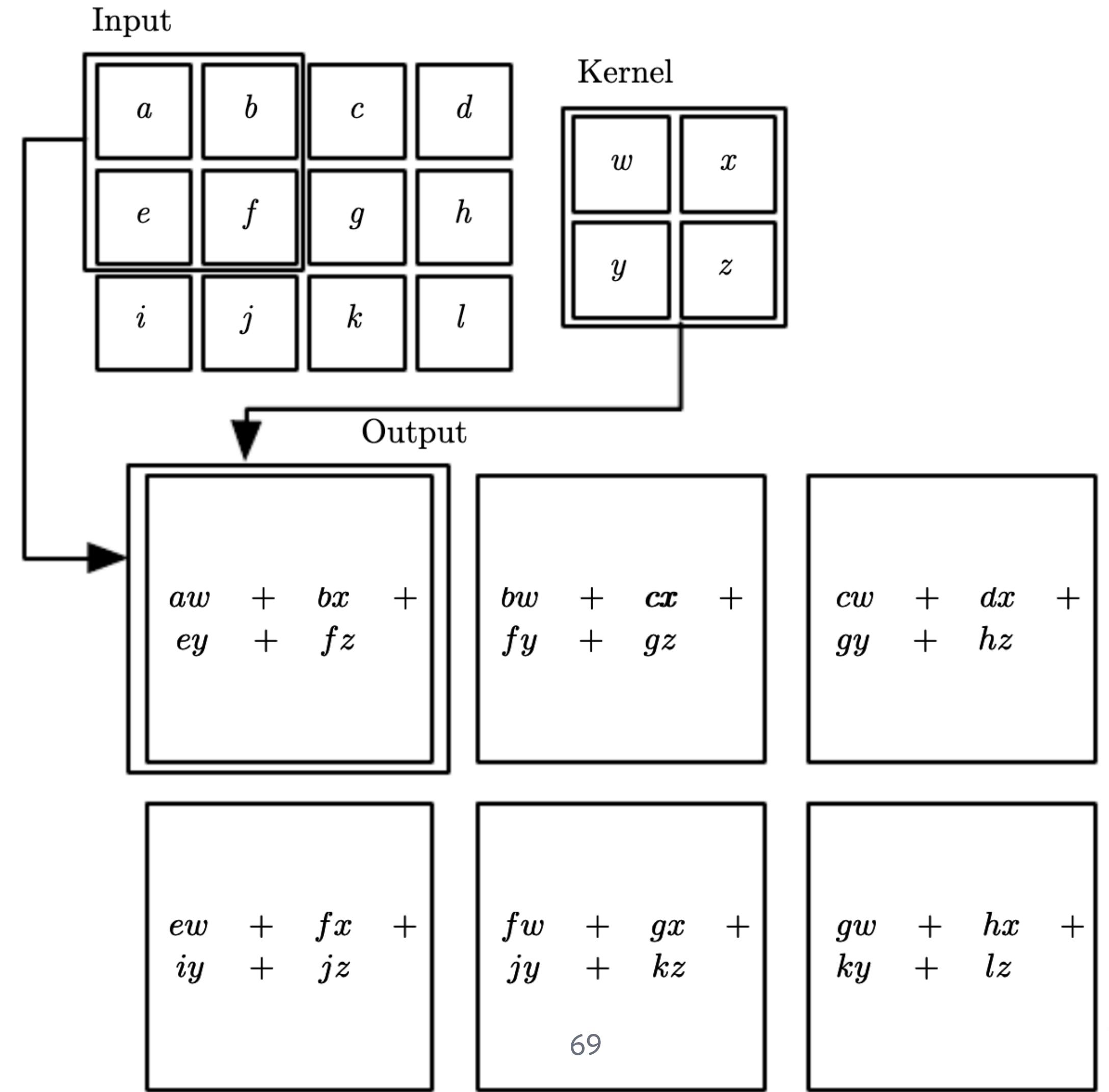
- Discrete convolution: $(f \star g)[i] = \sum_{k=1}^K f[i+k]g[k]$



* Credits: Chris Olah

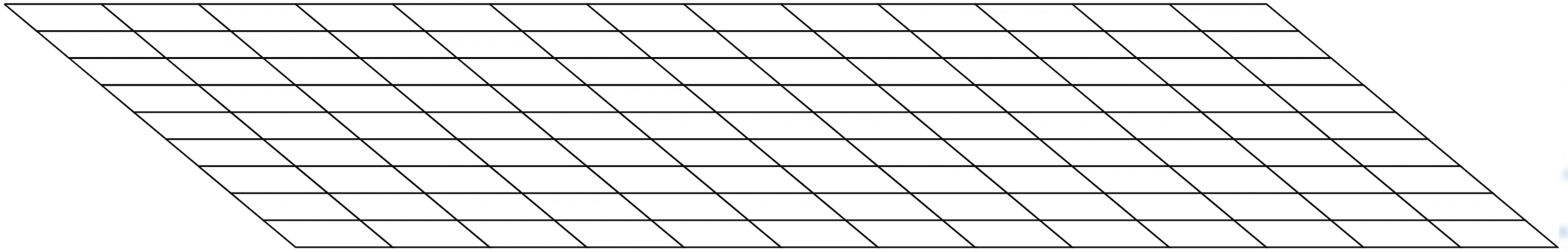
Convolution

- Input and kernel can have two (or more) dimensions:



Convolution

- Illustration of 2D convolution:

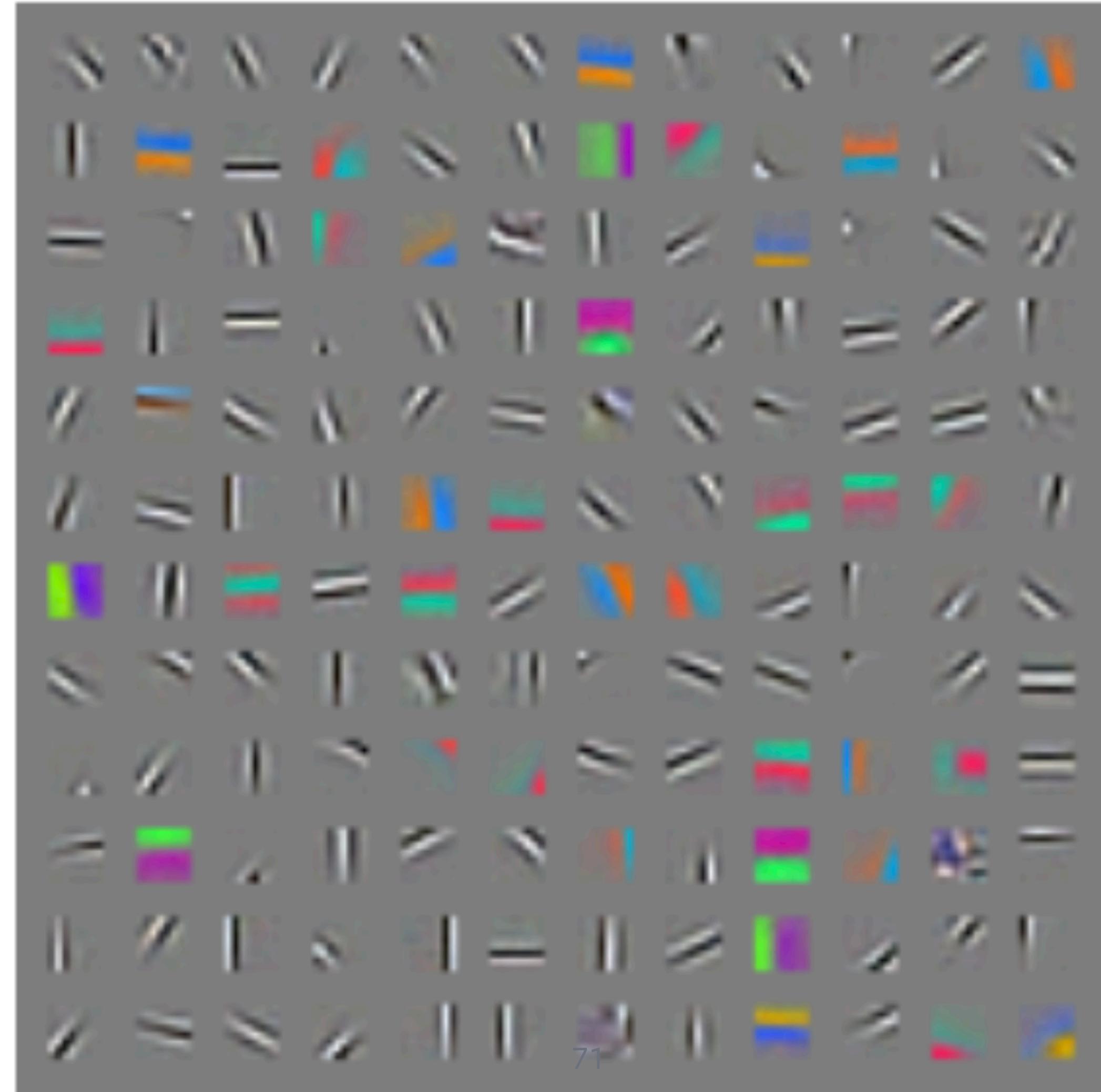


* Credits: Chris Olah

Convolution

- **Visualization** of filters learned in first layer of convolutional network:

$$(f \star g)[i] = \sum_{k=1}^K f[i + k]g[k]$$

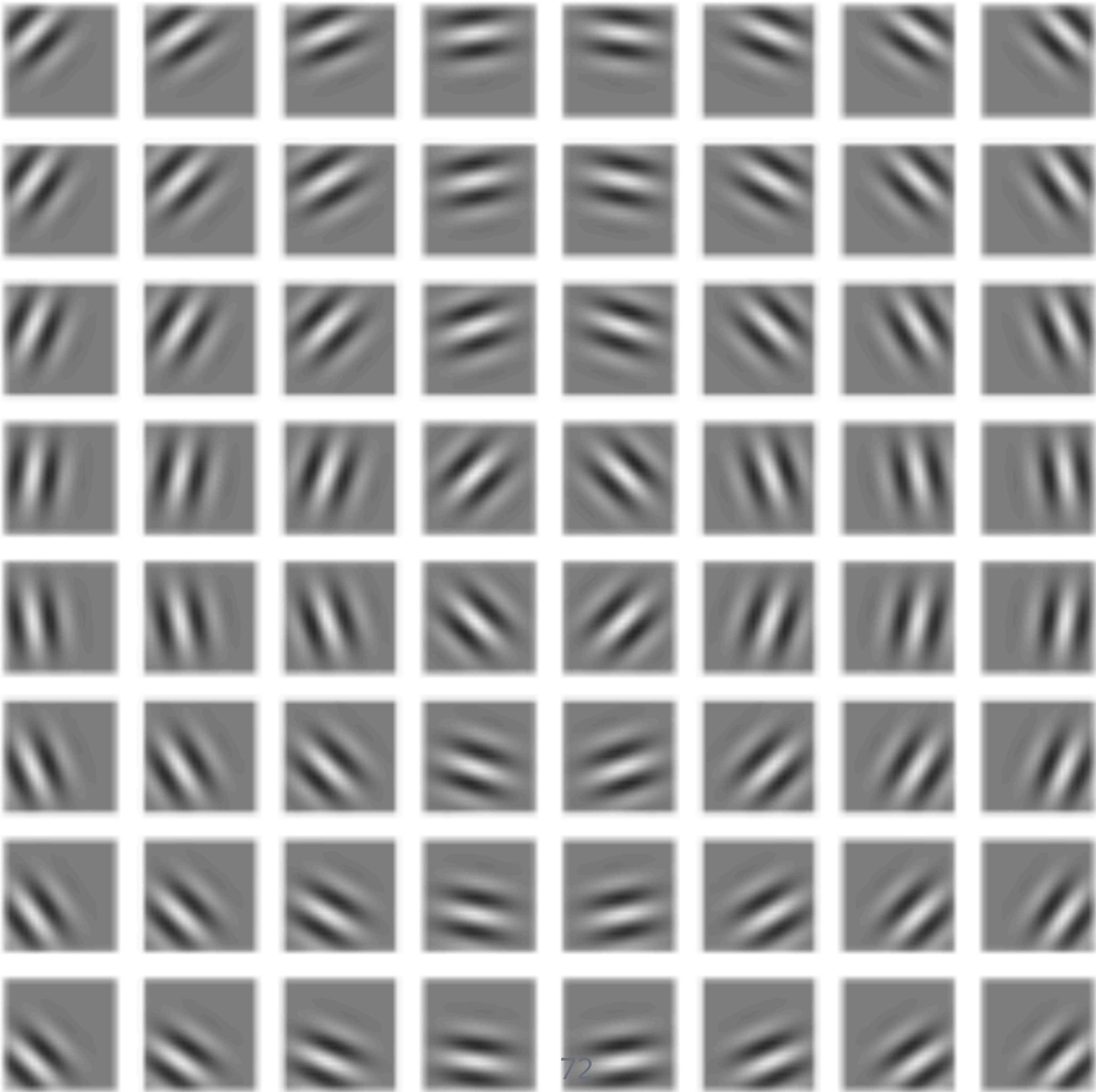


71

* Credits: Ian Goodfellow

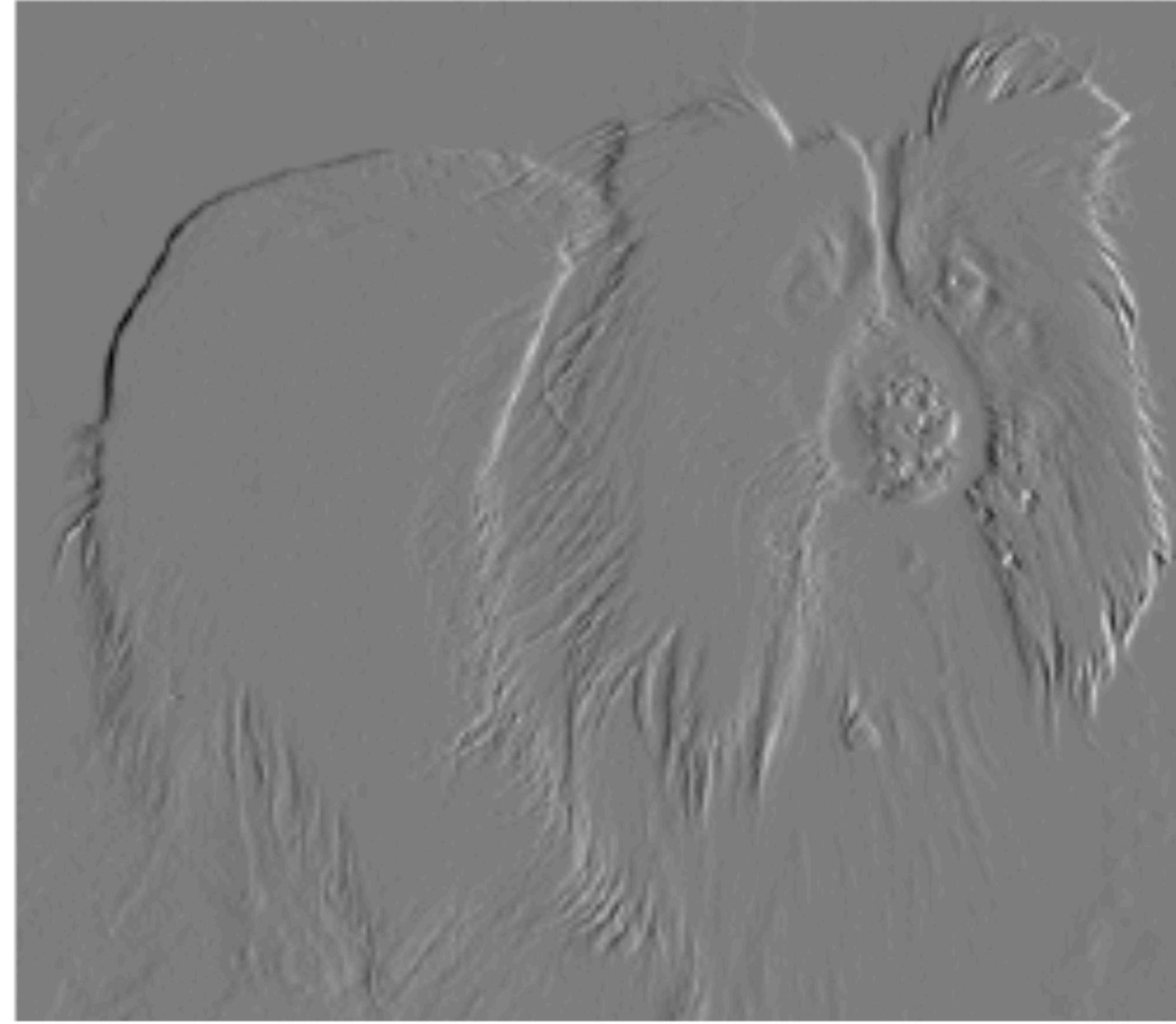
Convolution

- These look a lot like **Gabor filters** commonly used for edge detection:



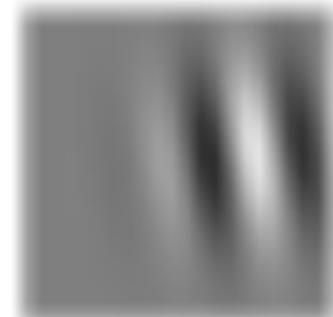
Convolution

- These look a lot like **Gabor filters** commonly used for edge detection:



$$(f \star g)[i] = \sum_{k=1}^K f[i+k]g[k]$$

73



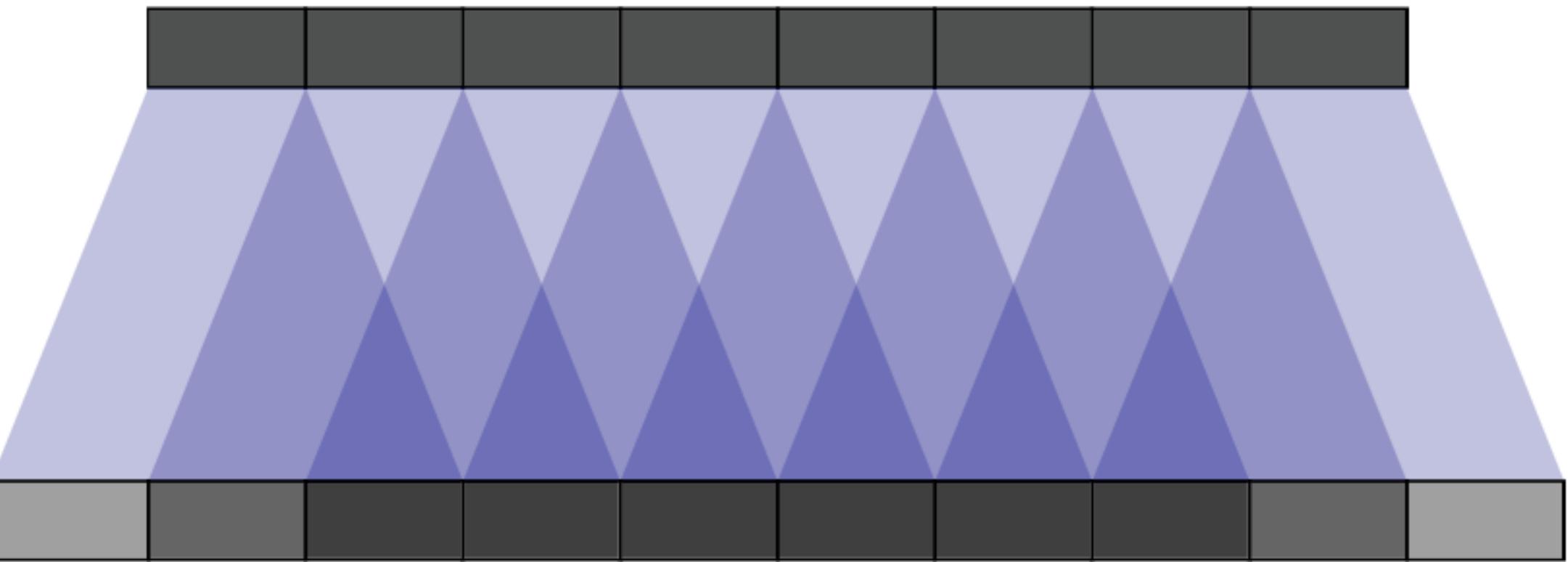
$$(f \star g)[i] = \sum_{k=1}^K f[i+k]g[k]$$

* Credits: Ian Goodfellow

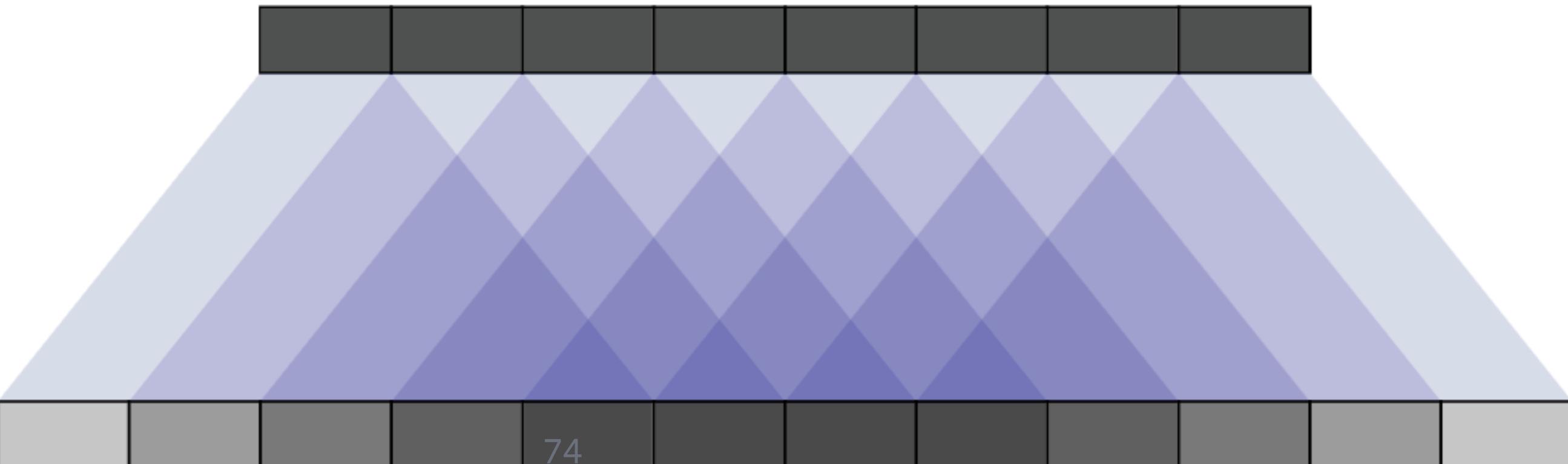
Convolution

- Convolutional kernels have a **size**:

kernel size = 3:



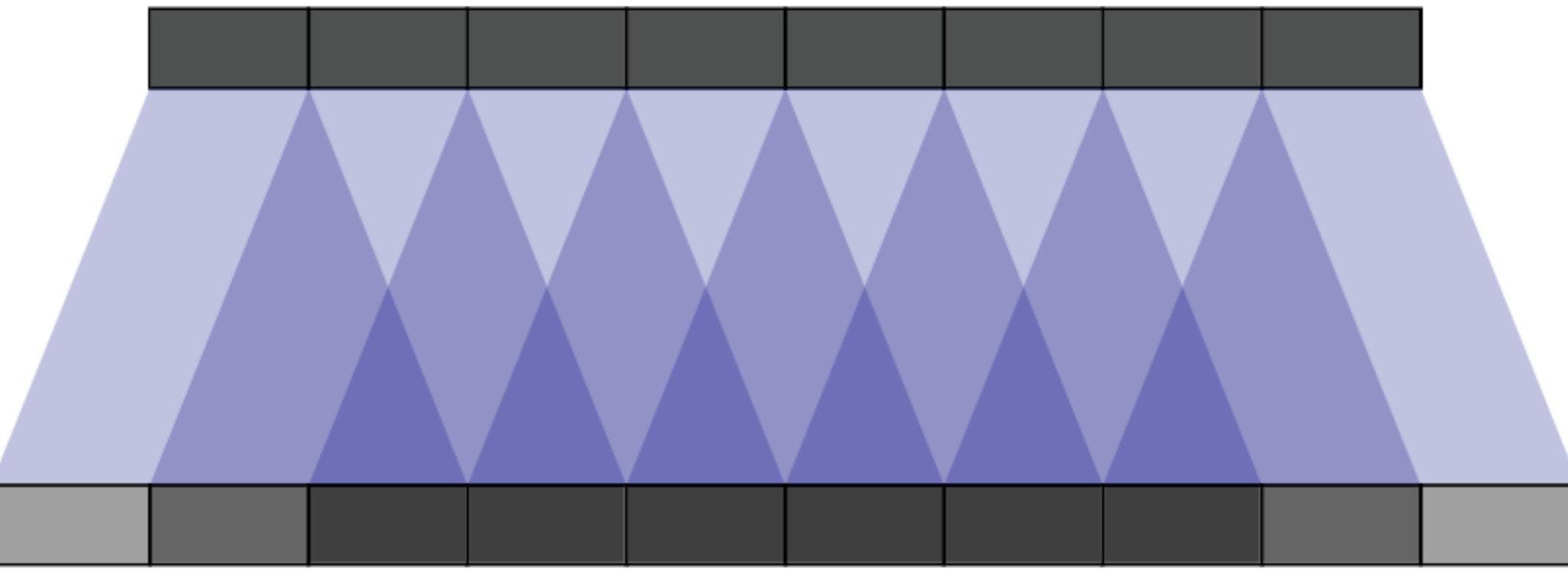
kernel size = 5:



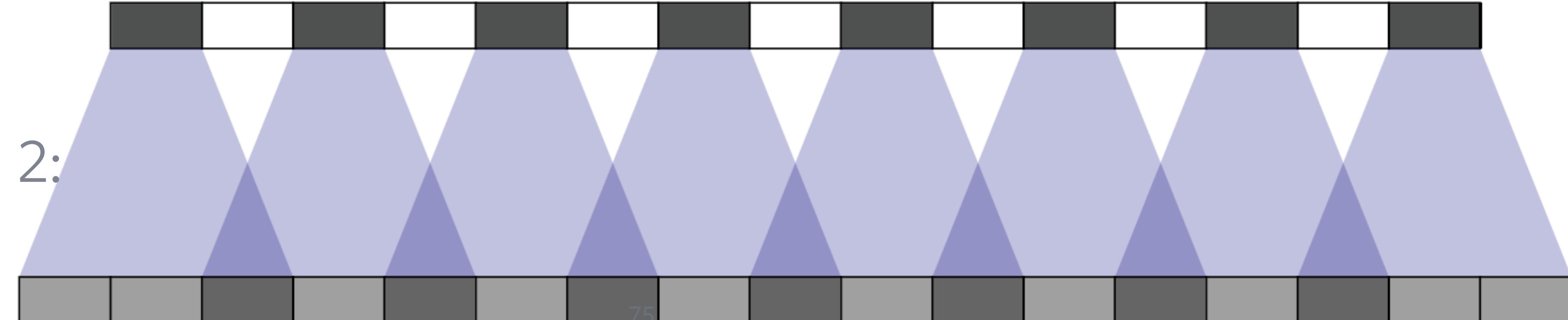
Convolution

- Convolutional kernels are applied with a **stride**:

kernel stride = 1:



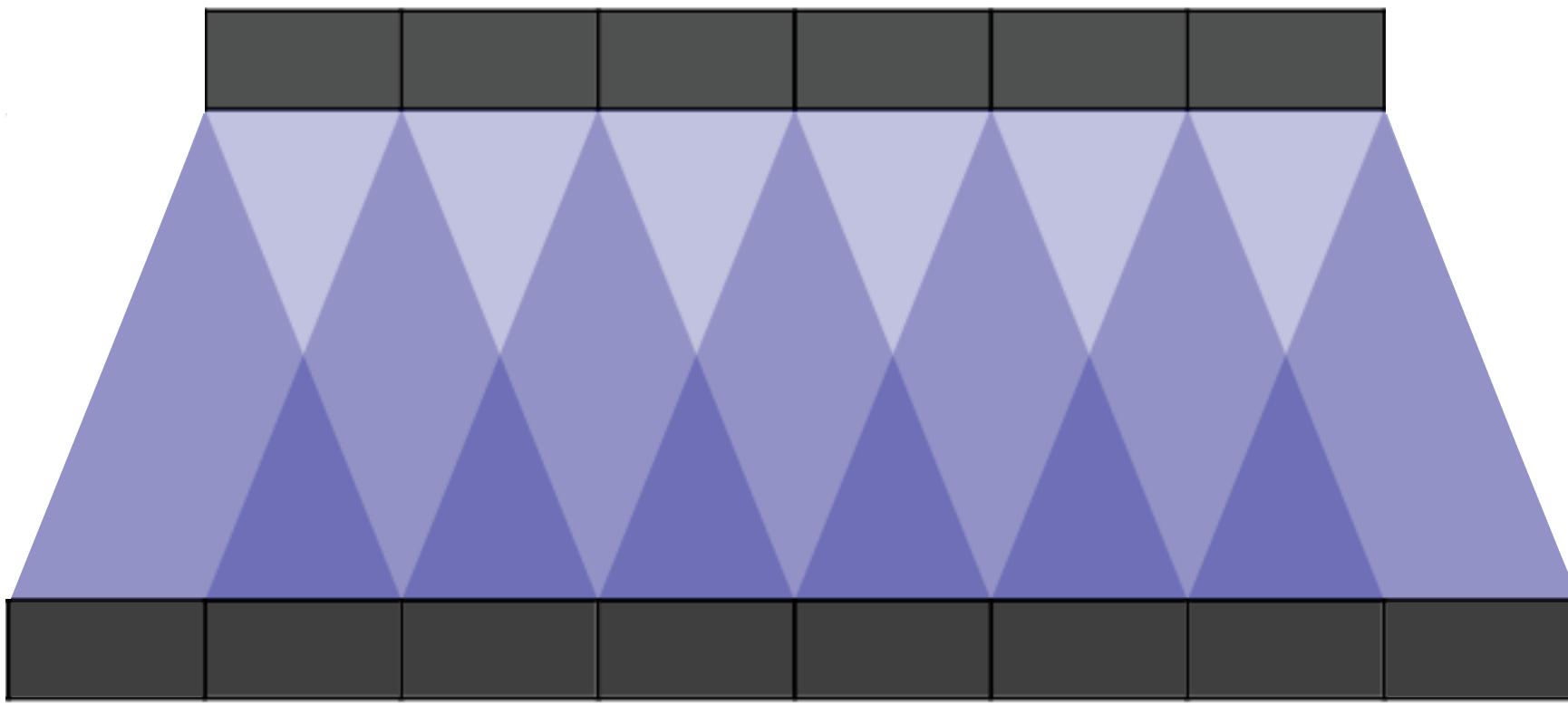
kernel stride = 2:



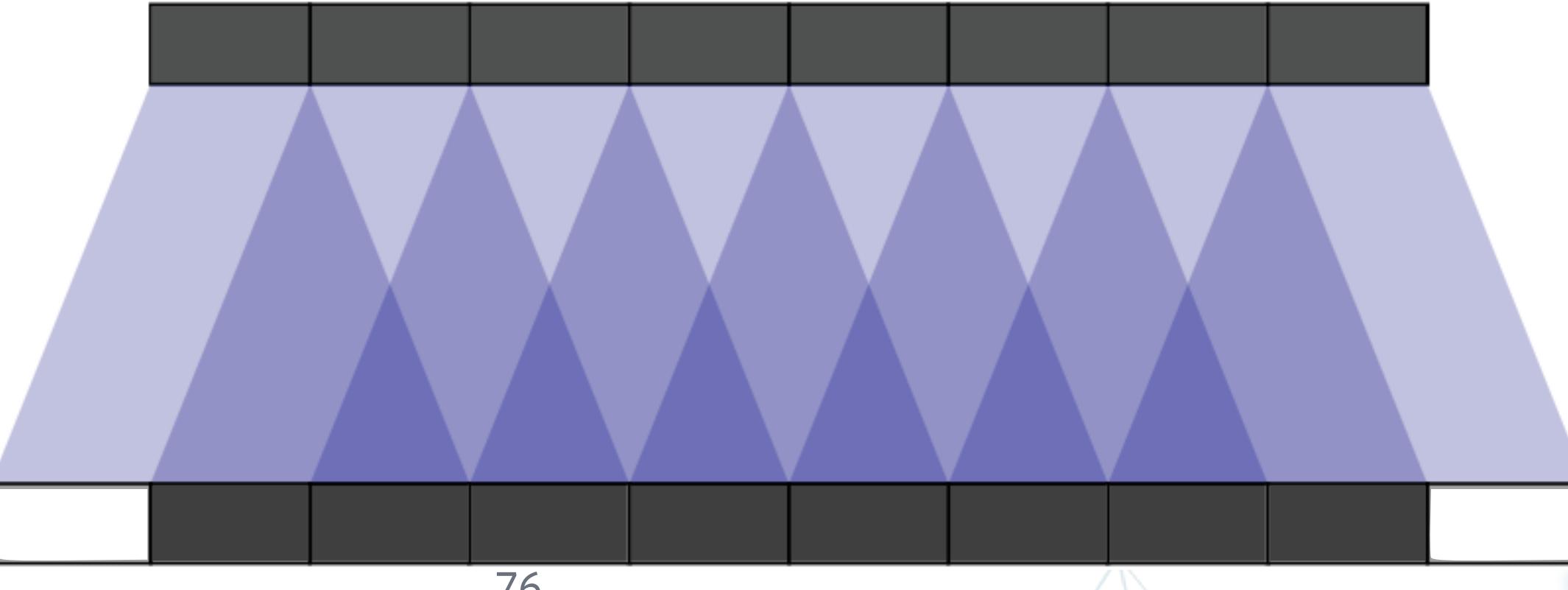
Convolution

- To keep output the same size as input, we can use **padding**:

padding = 0:



padding = 1:



Summary

- Loss measures discrepancy between a prediction and the true label
- ERM minimizes the loss averaged over the training data
- Logistic regression is a linear ERM model that uses logistic loss
- Multi-layer networks iteratively apply learned linear functions and fixed non-linear functions: gradients computed by chain rule (backprop)
- Convolutions filter signals and are the basis of convolutional networks

Reading material

- V. Vapnik. **The Nature of Statistical Learning Theory**, 1999:
 - Chapter 1.
- I. Goodfellow, Y. Bengio, and A. Courville. **Deep Learning**, 2016.
 - Chapter 6, 8, and 9.
- <https://en.wikipedia.org/wiki/Convolution>

Questions?