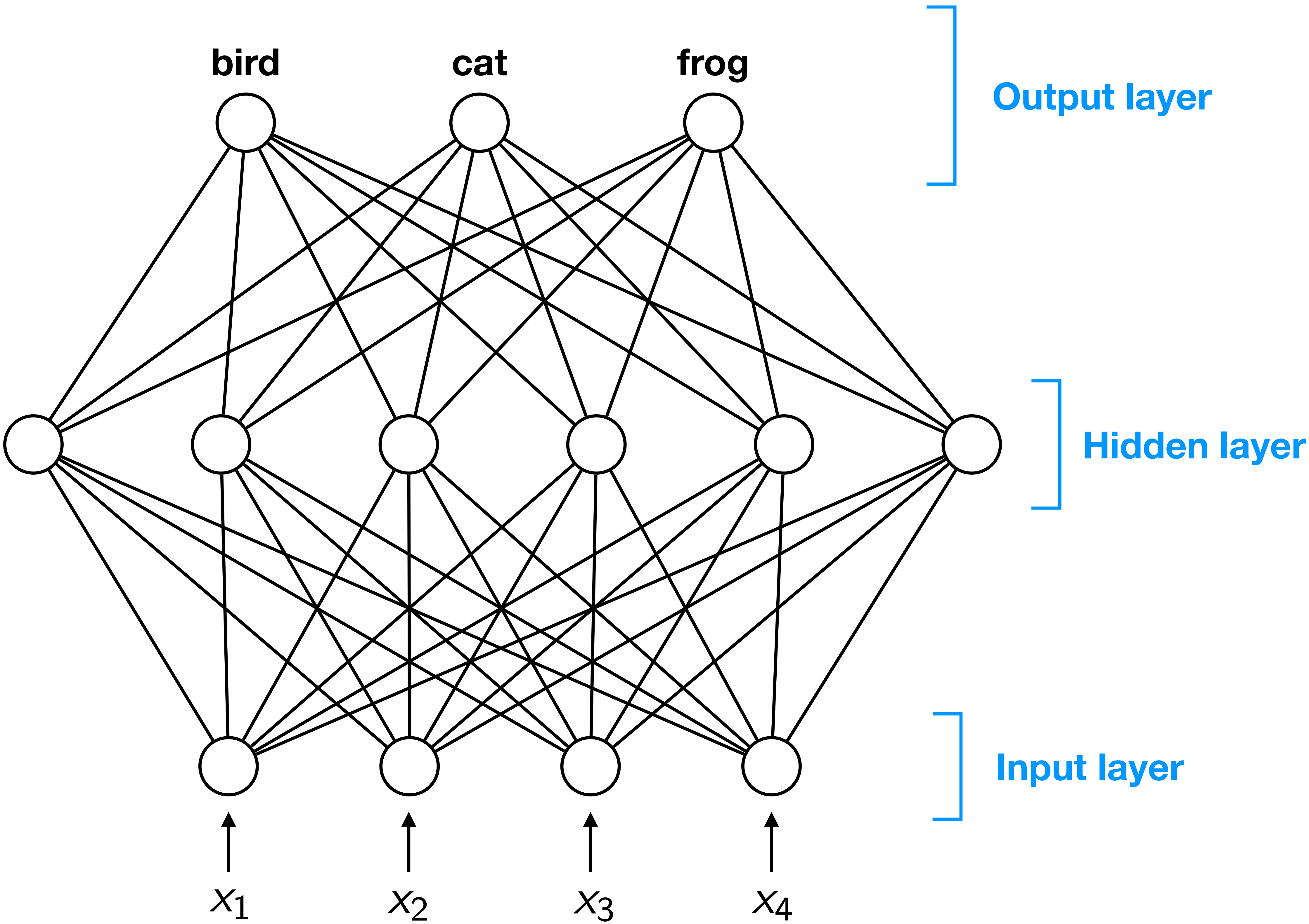
Neural Networks

(note: many parts of this lecture are on the whiteboard)

Nishant Mehta

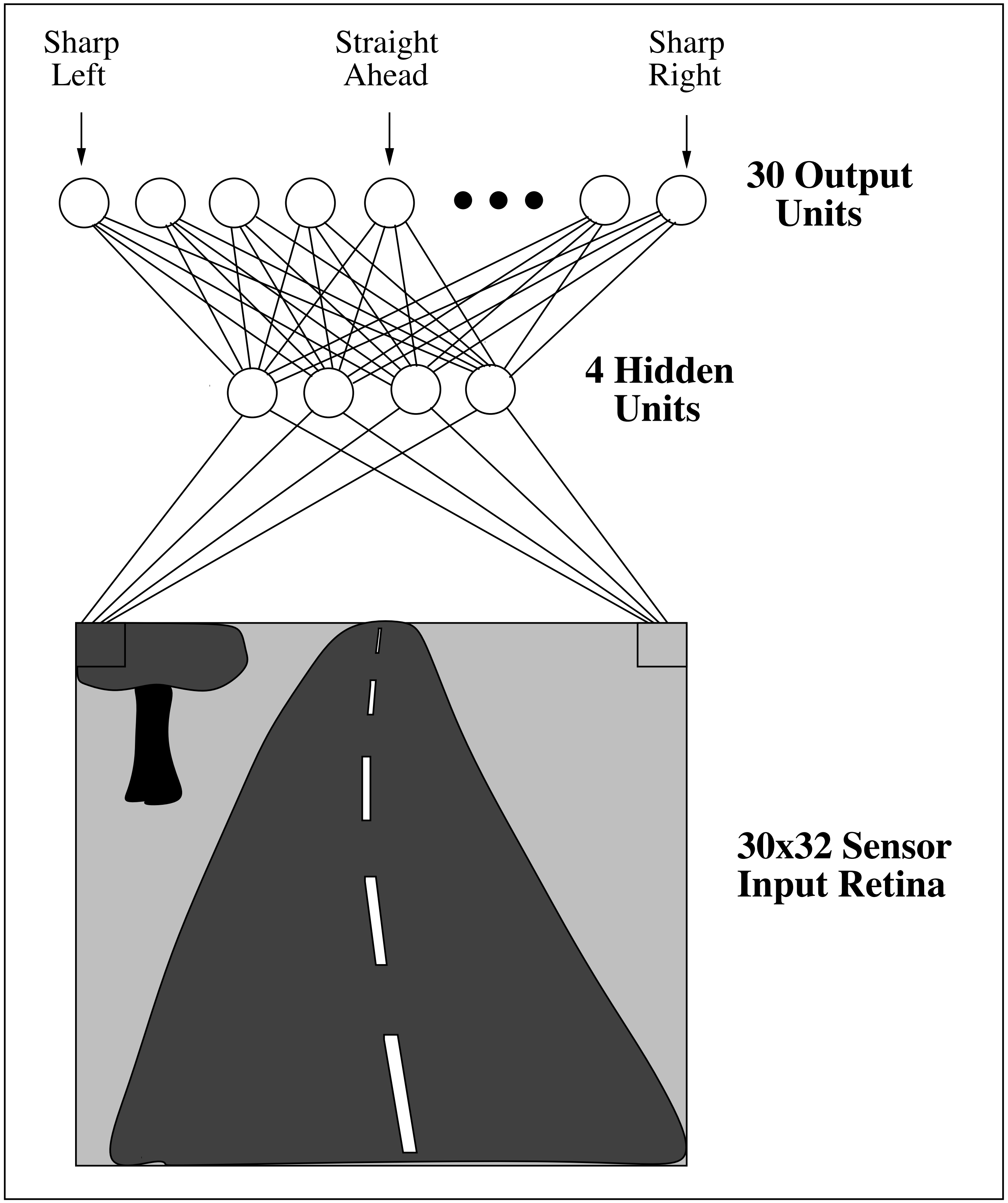
Lectures 7–11

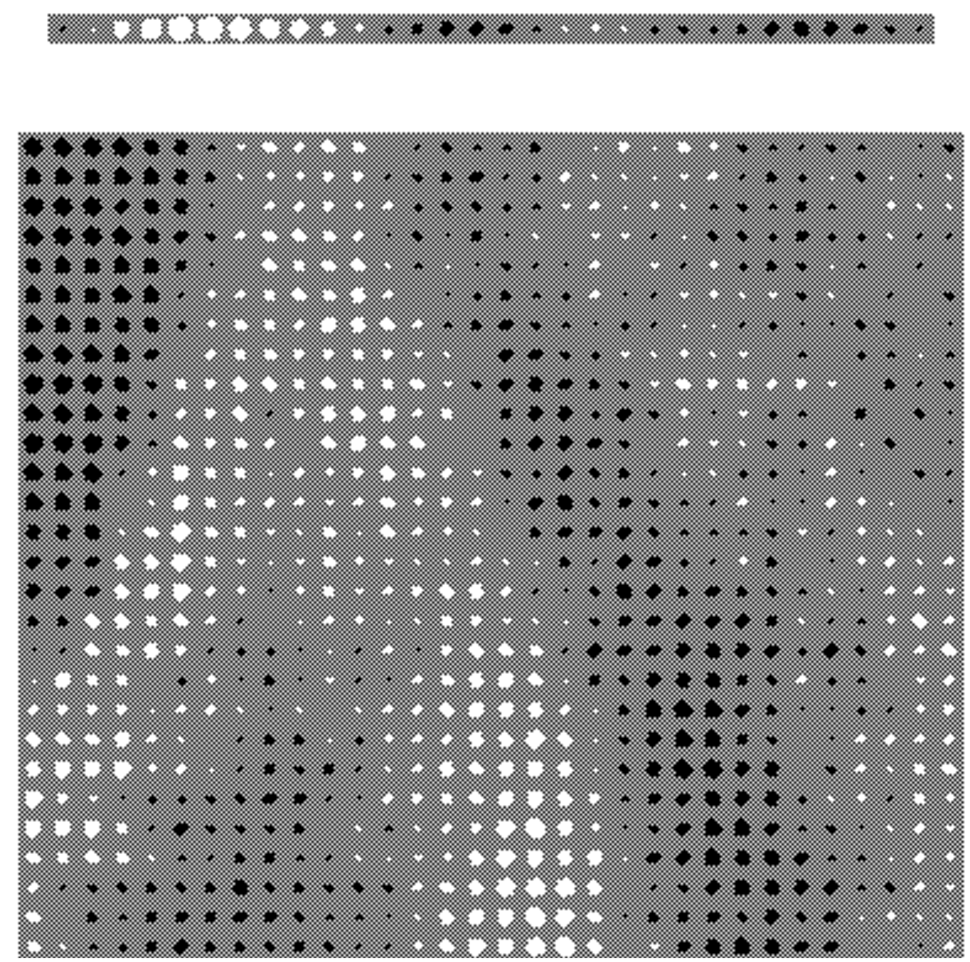
Artificial Neural Network



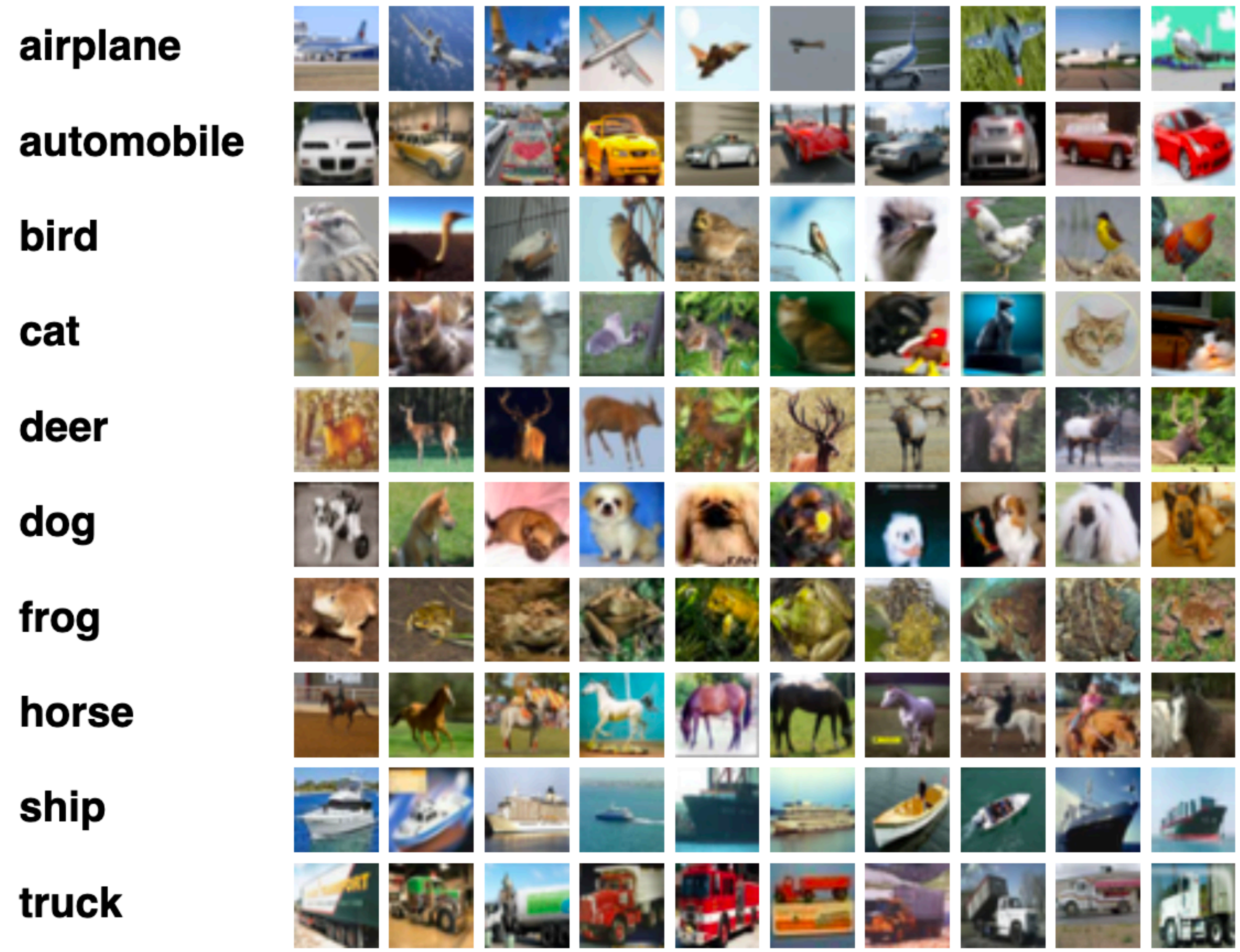
ALVINN

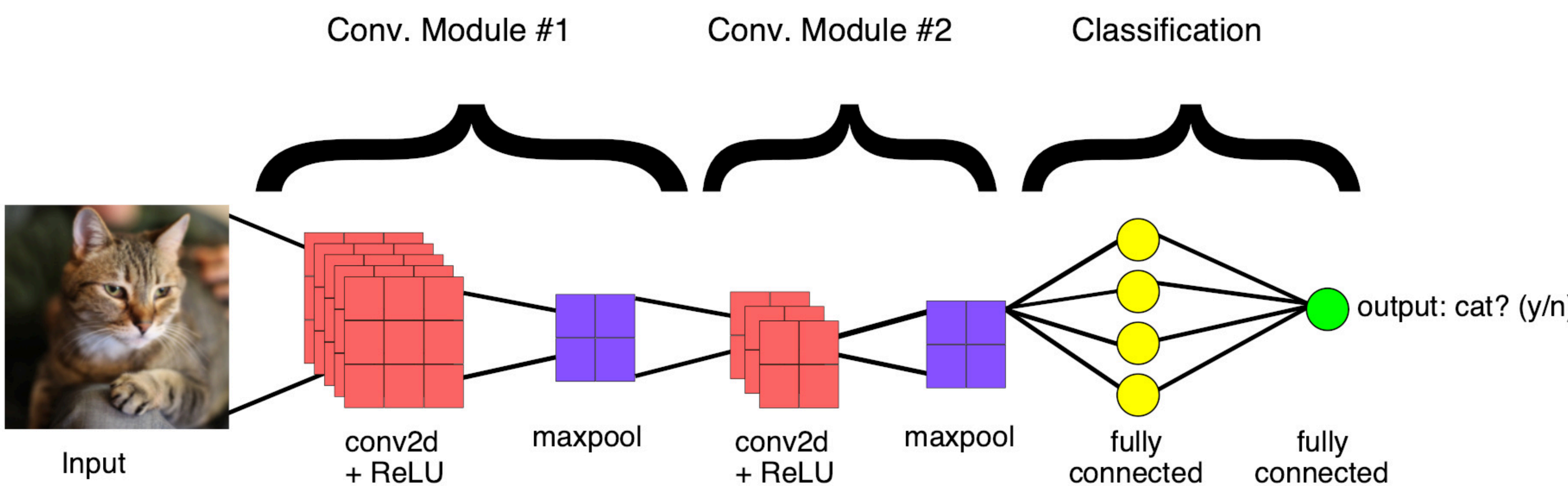
Autonomous Land Vehicle In a Neural Network





[CIFAR-10 dataset]

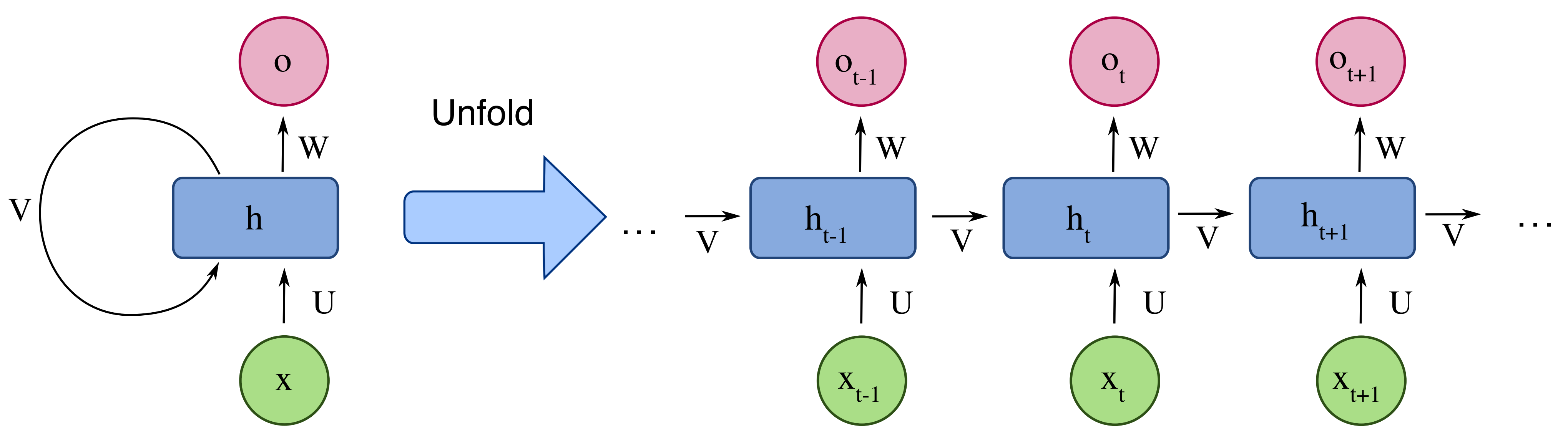




Automated speech recognition (ASR)

A history of neural network approaches for ASR

- 2012: English, model: hybrid models involving deep neural networks and more classical approaches
- 2014: English, model: Long short-term memory networks (LSTMs)
- 2014: English, model: Recurrent neural networks (RNNs)
- 2016: English/Mandarin, model: RNNs with Gated recurrent units (GRUs)



An RNN

The estimated costs of training a model once

In practice, models are usually trained many times during research and development.

	Date of original paper	Energy consumption (kWh)	Carbon footprint (lbs of CO2e)	Cloud compute cost (USD)
Transformer (65M parameters)	Jun, 2017	27	26	$41-$140
Transformer (213M parameters)	Jun, 2017	201	192	$289-$981
ELMo	Feb, 2018	275	262	$433-$1,472
BERT (110M parameters)	Oct, 2018	1,507	1,438	$3,751-$12,571
Transformer (213M parameters) w/ neural architecture search	Jan, 2019	656,347	626,155	$942,973-$3,201,722
GPT-2	Feb, 2019	-	-	$12,902-$43,008

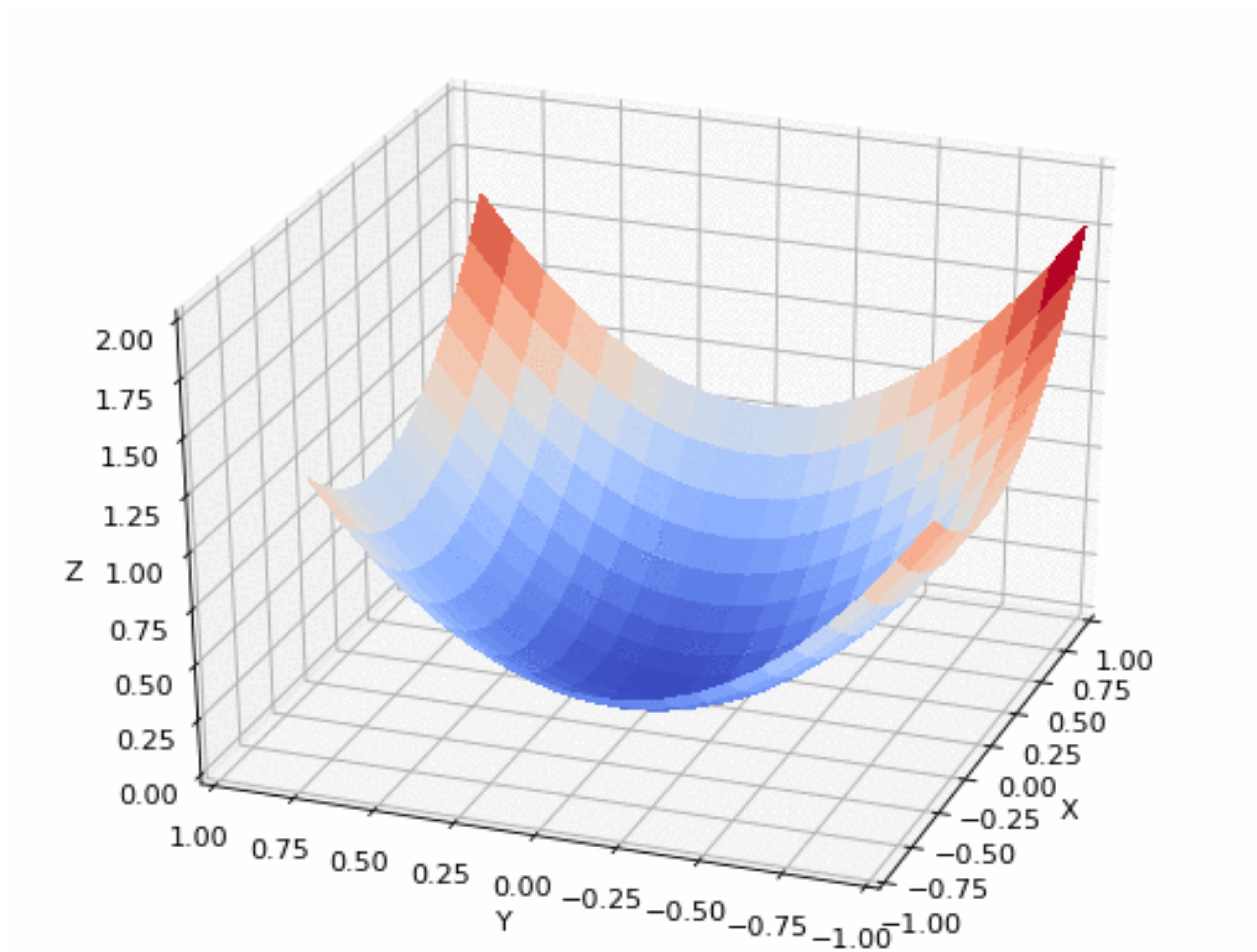
(Strubell et al., 2019)
"Energy and Policy Considerations for Deep Learning in NLP"

Gradient descent: main idea

Given current weights w, update using the rule:

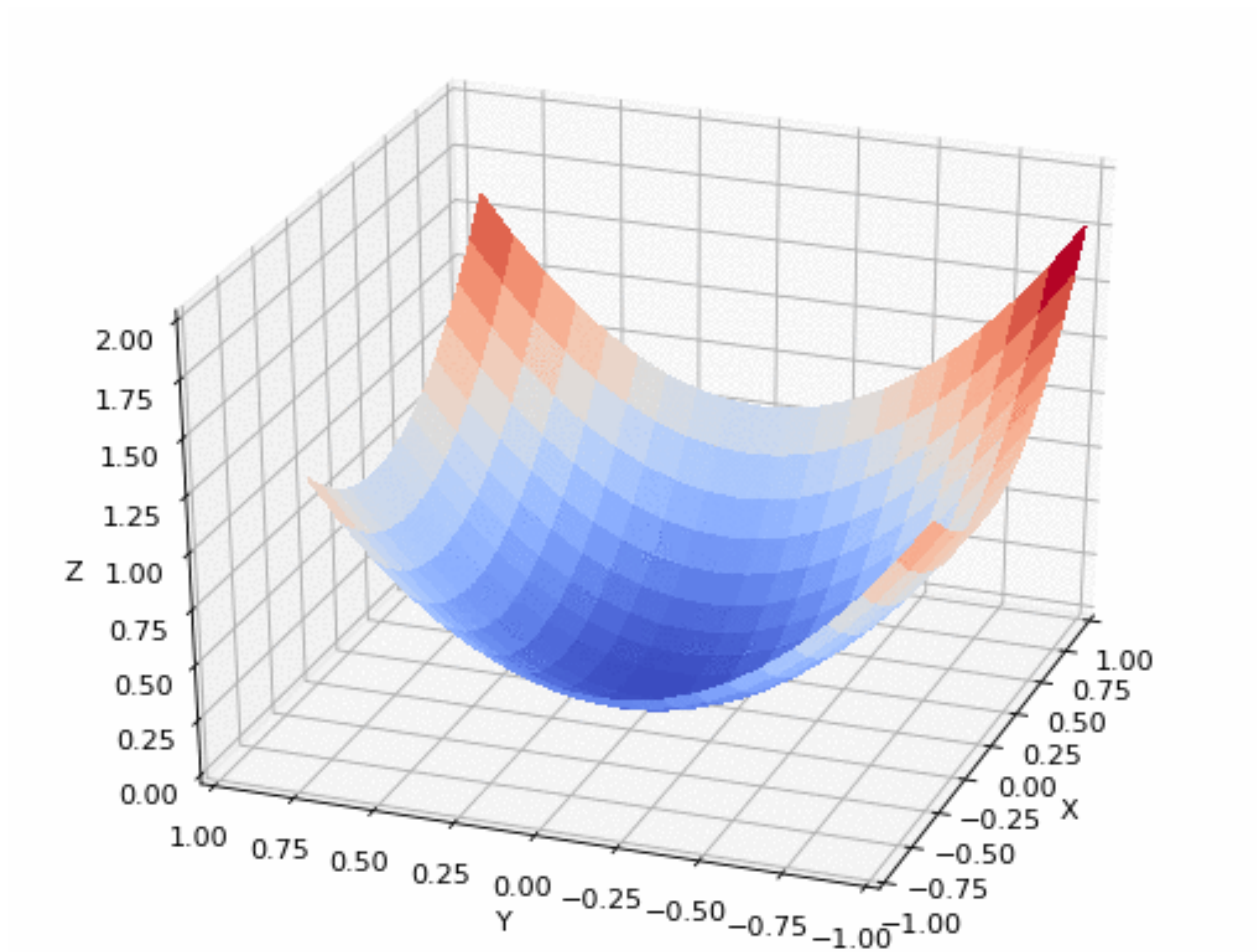
$$w^{(\text{new})} \leftarrow w - \eta \nabla E(w)$$

Gradient descent



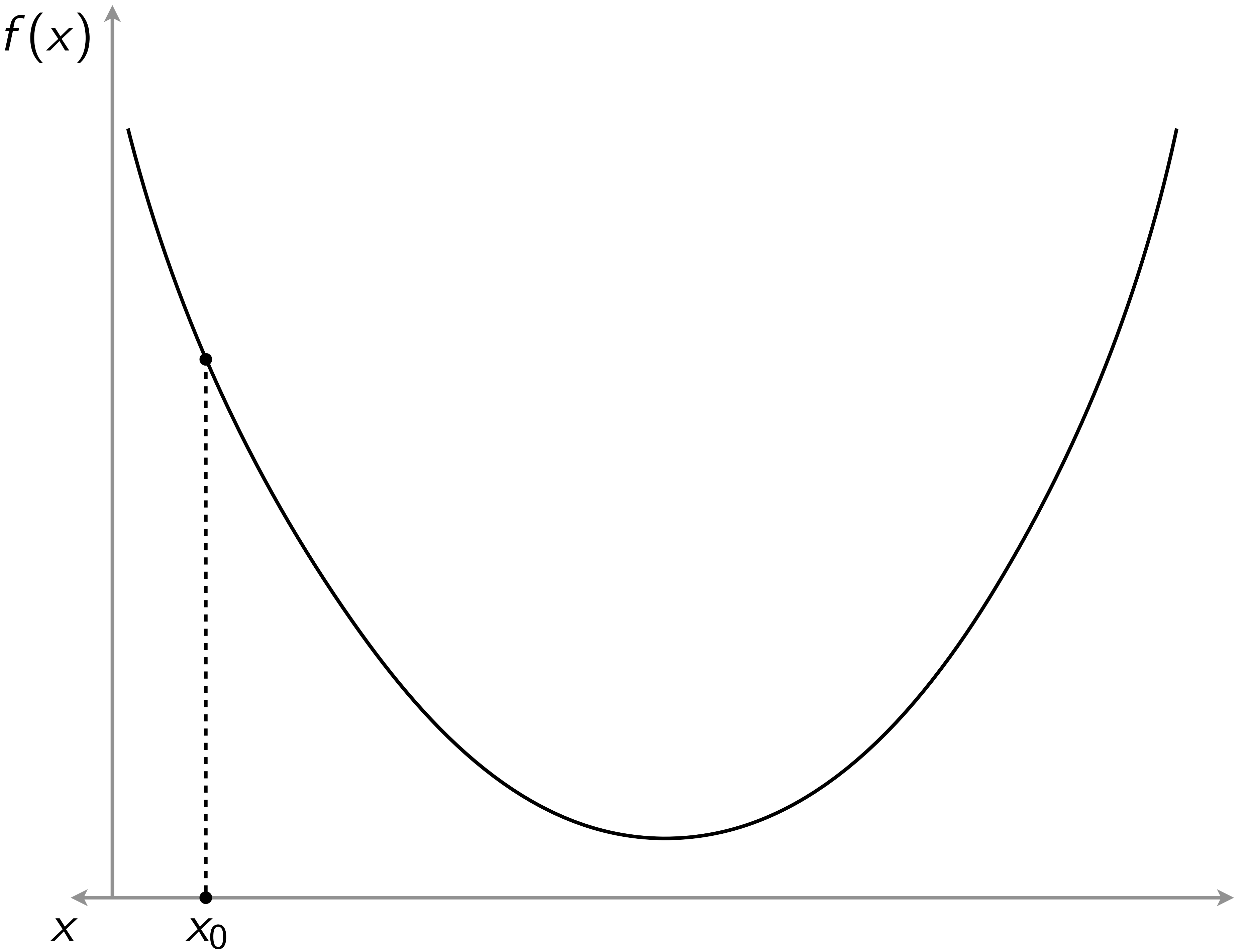
Thanks to Pierre Vigier ([link]())

Gradient descent

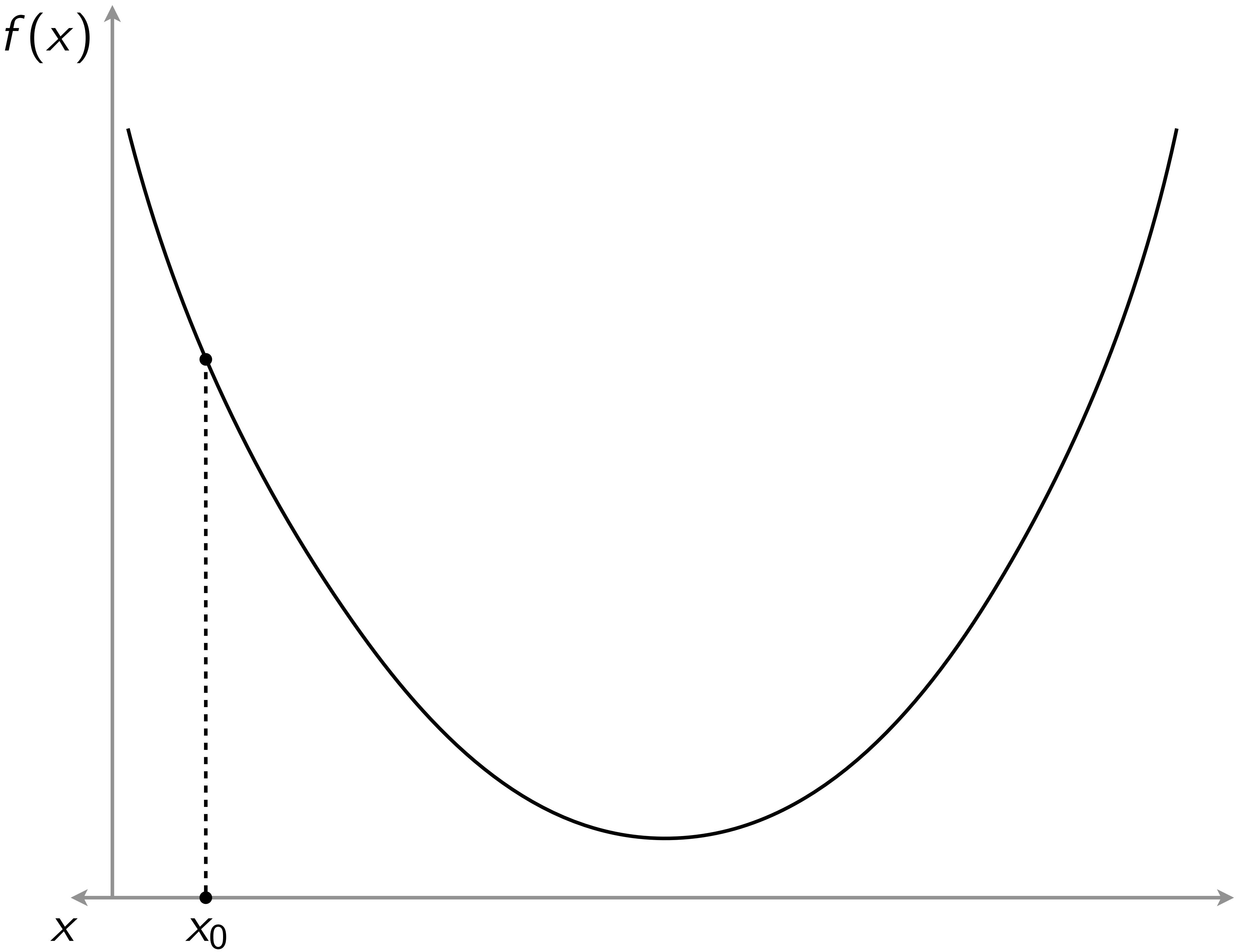


Thanks to Pierre Vigier (link)

Gradient descent step size

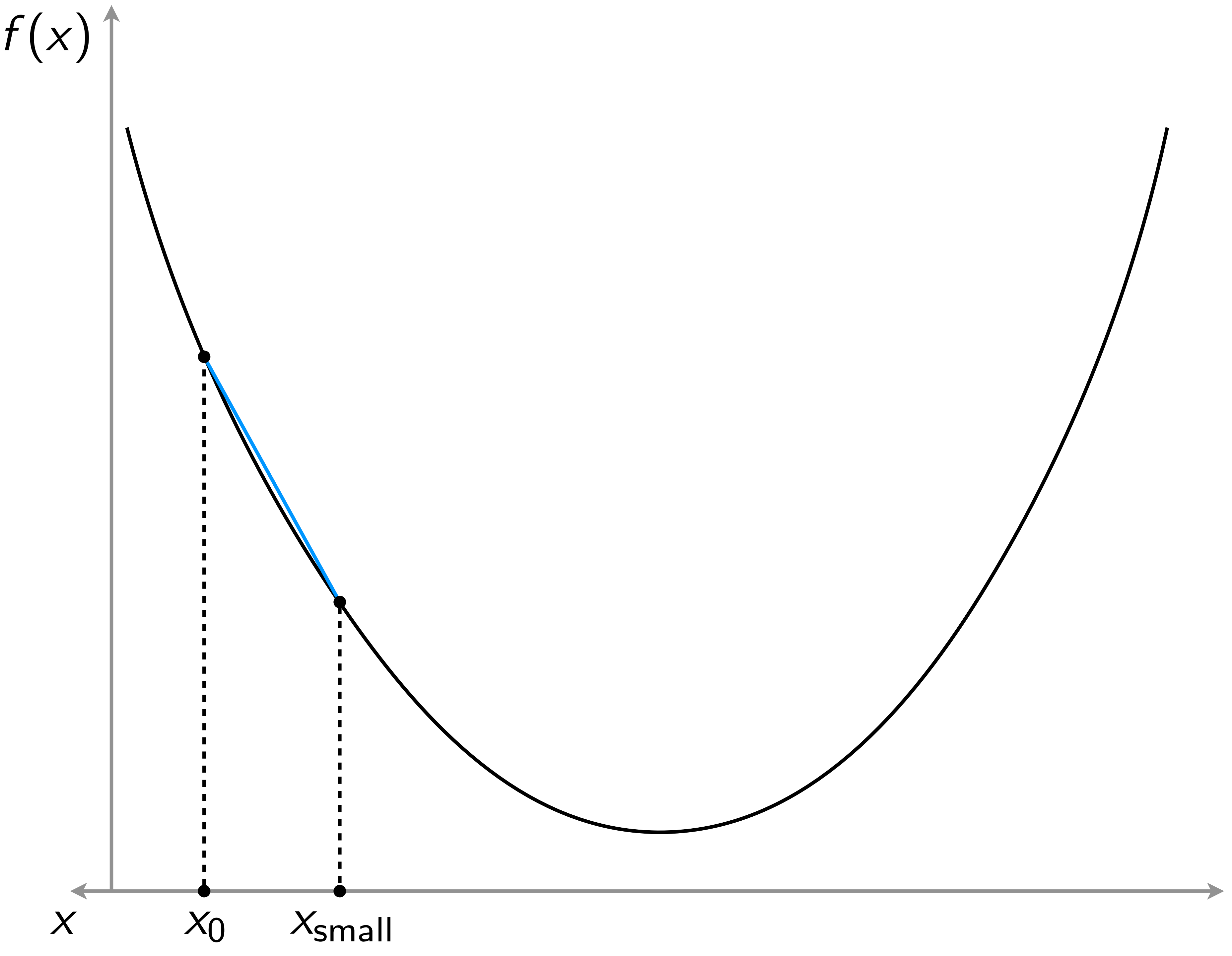


Gradient descent step size



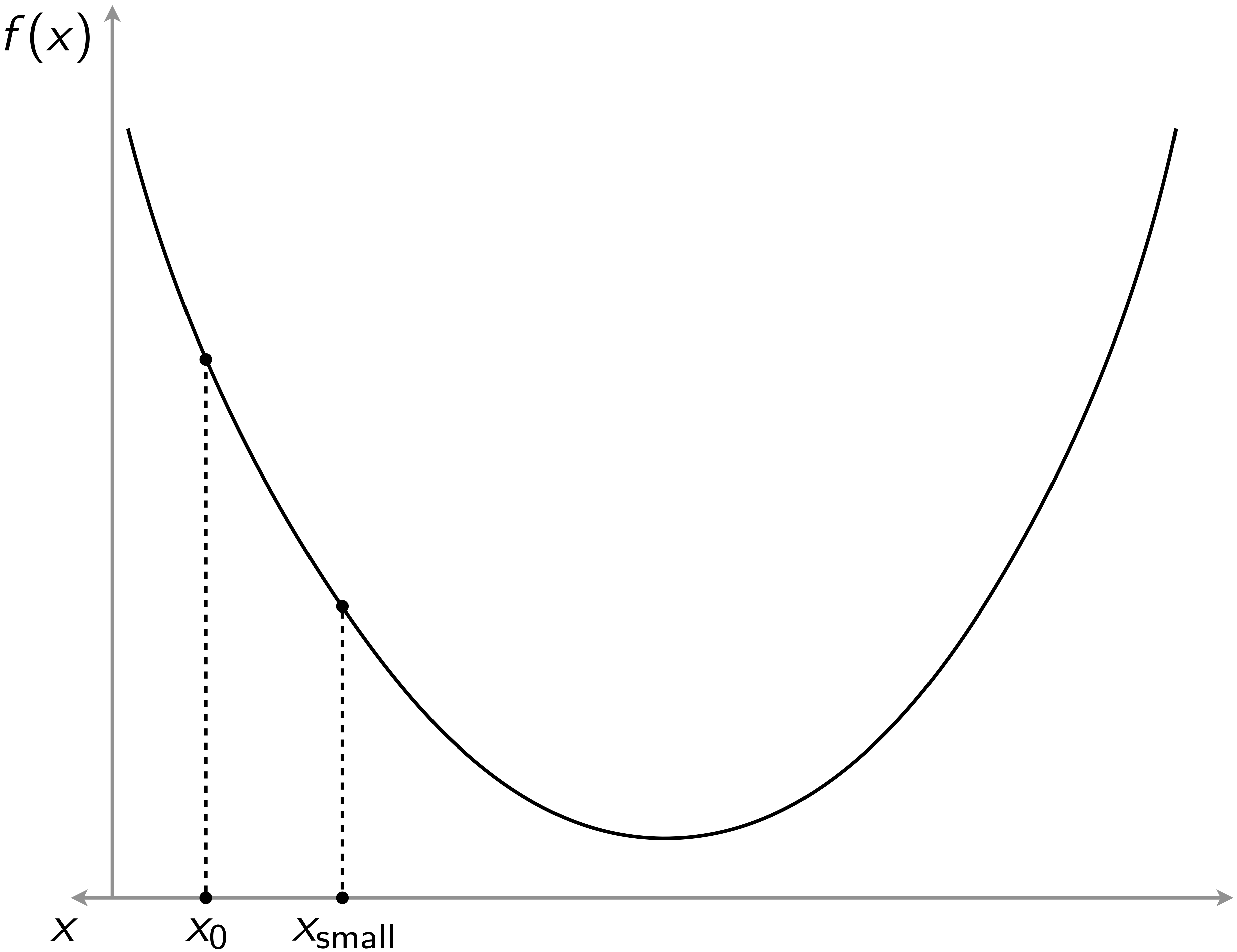
$$x_{\text{small}} = x_0 - 0.1 f'(x)$$

Gradient descent step size



$x_{\text{small}} = x_0 - 0.1 f'(x)$

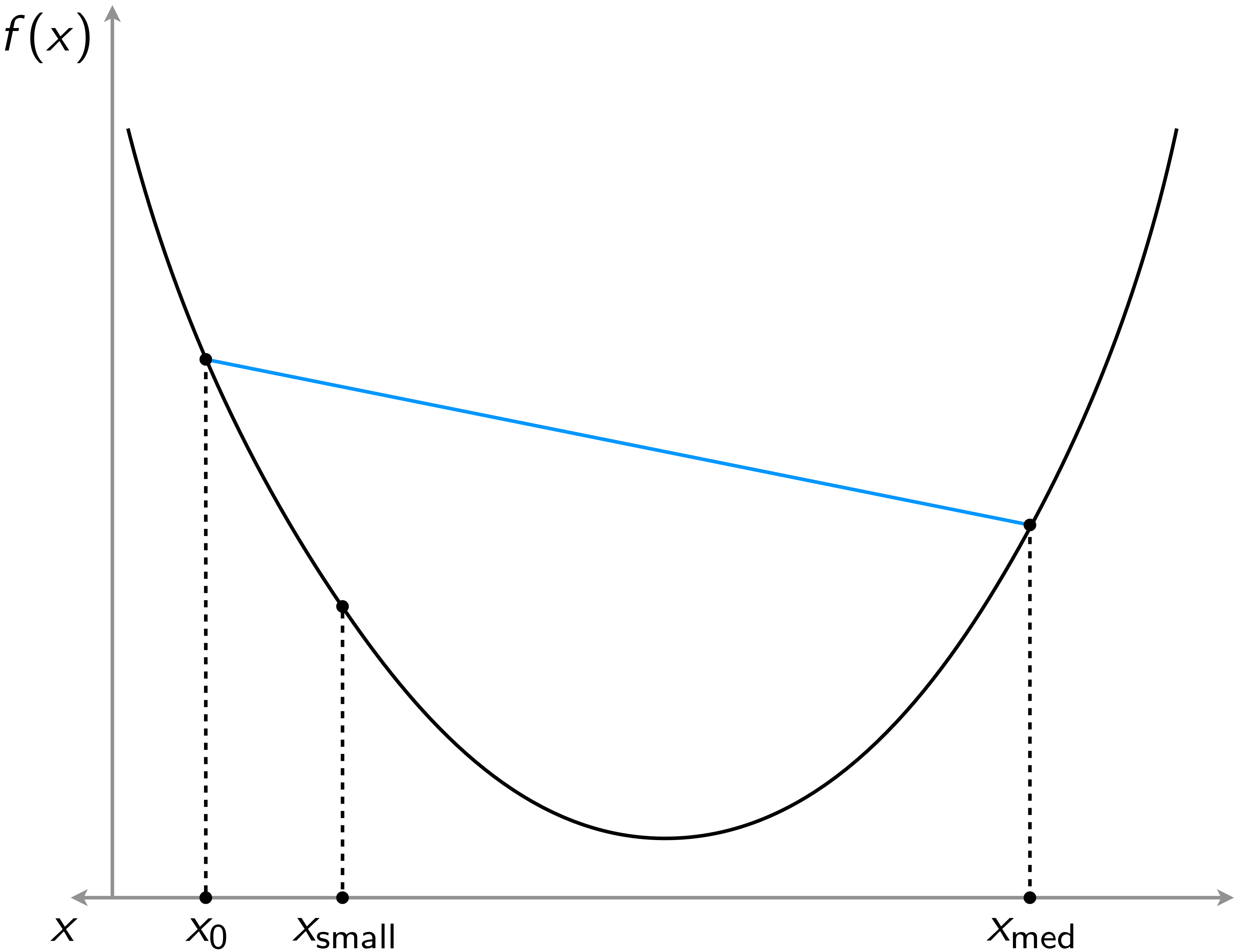
Gradient descent step size



$$x_{\text{small}} = x_0 - 0.1f'(x)$$

$$x_{\text{med}} = x_0 - 0.5f'(x)$$

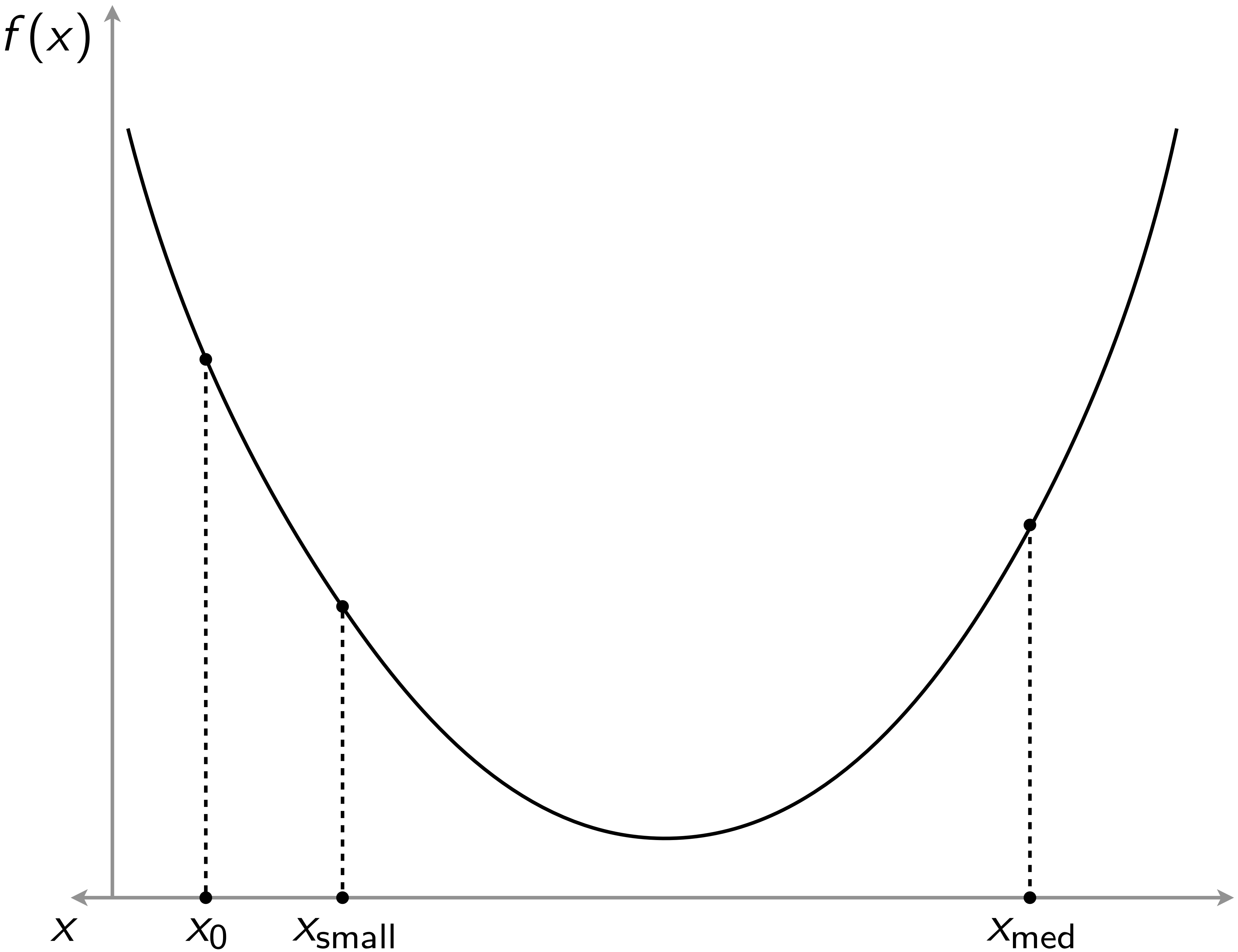
Gradient descent step size



$$x_{\text{small}} = x_0 - 0.1f'(x)$$

$$x_{\text{med}} = x_0 - 0.5f'(x)$$

Gradient descent step size

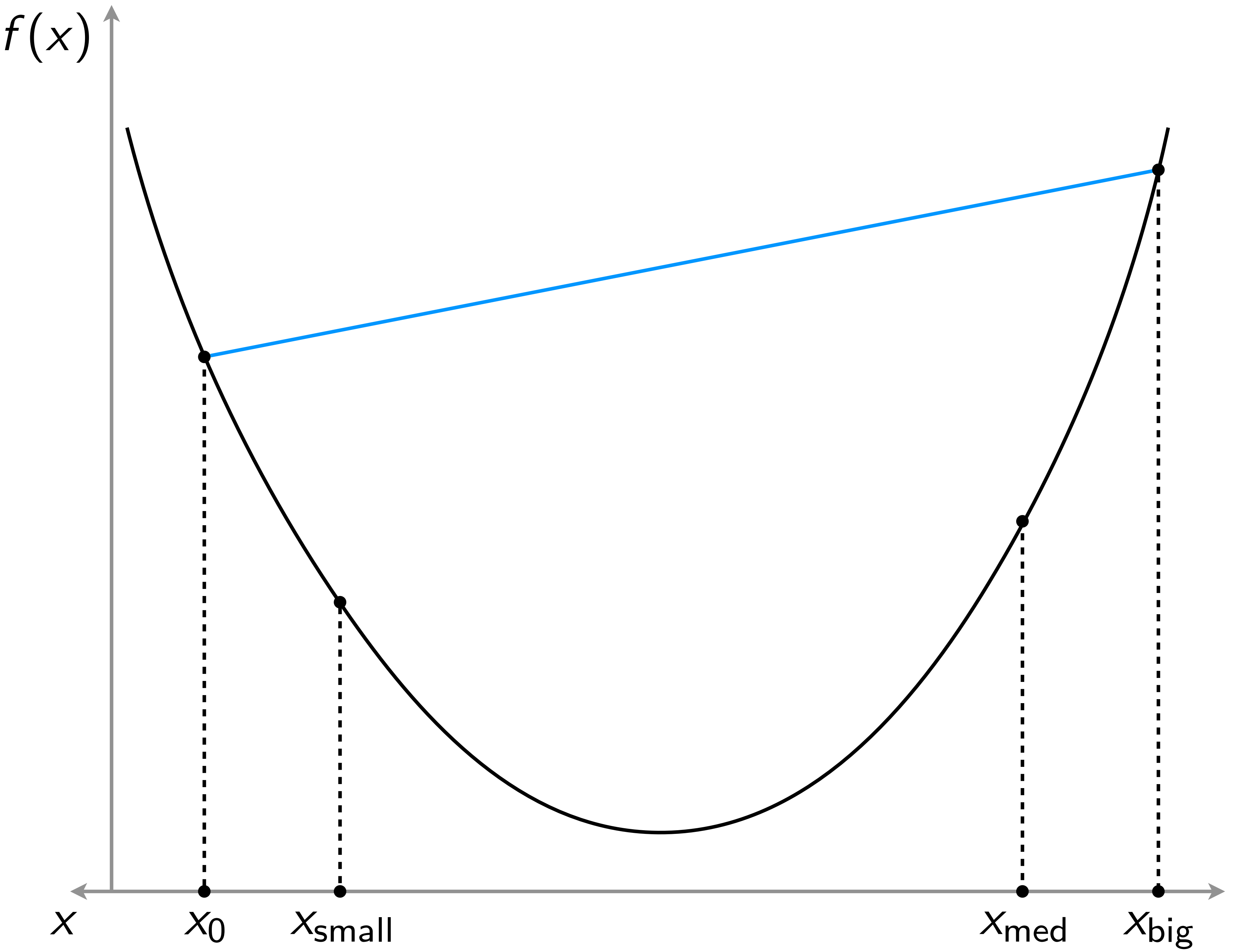


$$x_{\text{small}} = x_0 - 0.1f'(x)$$

$$x_{\text{med}} = x_0 - 0.5f'(x)$$

$$x_{\text{big}} = x_0 - 0.7f'(x)$$

Gradient descent step size



$$x_{\text{small}} = x_0 - 0.1 f'(x)$$

$$x_{\text{med}} = x_0 - 0.5 f'(x)$$

$$x_{\text{big}} = x_0 - 0.7 f'(x)$$

Example: linear regression with squared loss

- In linear regression with the squared loss:
  - each $w \in \mathbb{R}^d$ corresponds to a linear hypothesis

$$h_w(x) = \langle w, x \rangle = \sum_{j=1}^{d} w_j x_j$$

- The training error of w is

$$E(w) = \frac{1}{2} \sum_{i=1}^{n} (y_i - o_i)^2$$

Gradient descent for linear regression

- Gradient descent rule:

$$w^{(\text{new})} = w - \eta \nabla E(w) = w - \eta \begin{pmatrix} \frac{\partial E(w)}{\partial w_1} \\ \vdots \\ \frac{\partial E(w)}{\partial w_d} \end{pmatrix}$$

Gradient descent for linear regression

- Gradient descent rule:

$$w^{(\text{new})} = w - \eta \nabla E(w) = w - \eta \begin{pmatrix} \frac{\partial E(w)}{\partial w_1} \\ \vdots \\ \frac{\partial E(w)}{\partial w_d} \end{pmatrix}$$

- Compute one partial derivative:

$$\frac{\partial E(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_{i=1}^{n} (y_i - o_i)^2 = \sum_{i=1}^{n} \frac{\partial}{\partial w_j} (y_i - o_i)^2$$

Gradient descent for linear regression

- Gradient descent rule:

$$w^{(\text{new})} = w - \eta \nabla E(w) = w - \eta \begin{pmatrix} \frac{\partial E(w)}{\partial w_1} \\ \vdots \\ \frac{\partial E(w)}{\partial w_d} \end{pmatrix}$$

- Compute one partial derivative:

$$\frac{\partial E(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_{i=1}^{n} (y_i - o_i)^2 = \sum_{i=1}^{n} \frac{\partial}{\partial w_j} (y_i - o_i)^2$$

- … of loss on one example:

$$\begin{aligned} \frac{\partial}{\partial w_j} (y_i - o_i)^2 = \frac{\partial (y_i - o_i)^2}{\partial o_i} \frac{\partial o_i}{\partial w_j} &= -(y_i - o_i) \frac{\partial \langle w, x_i \rangle}{\partial w_j} \\ &= -(y_i - o_i) x_{i,j} \end{aligned}$$

Gradient descent for linear regression

- Gradient descent rule:

$$w^{(\text{new})} = w - \eta \nabla E(w) = w - \eta \begin{pmatrix} \frac{\partial E(w)}{\partial w_1} \\ \vdots \\ \frac{\partial E(w)}{\partial w_d} \end{pmatrix}$$

- Compute one partial derivative:

$$\frac{\partial E(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_{i=1}^{n} (y_i - o_i)^2 = \sum_{i=1}^{n} \frac{\partial}{\partial w_j} (y_i - o_i)^2 = -\sum_{i=1}^{n} (y_i - o_i) x_{i,j}$$

- … of loss on one example:

$$\frac{\partial}{\partial w_j}(y_i - o_i)^2 = \frac{\partial (y_i - o_i)^2}{\partial o_i} \frac{\partial o_i}{\partial w_j} = -(y_i - o_i) \frac{\partial \langle w, x_i \rangle}{\partial w_j}$$

$$= -(y_i - o_i) x_{i,j}$$

Gradient descent for linear regression

- Gradient descent rule:

$$w^{(\text{new})} = w - \eta \nabla E(w) = w - \eta \begin{pmatrix} \frac{\partial E(w)}{\partial w_1} \\ \vdots \\ \frac{\partial E(w)}{\partial w_d} \end{pmatrix} = w + \eta \cdot \sum_{i=1}^{n} (y_i - o_i) x_i$$

- Compute one partial derivative:

$$\frac{\partial E(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_{i=1}^{n} (y_i - o_i)^2 = \sum_{i=1}^{n} \frac{\partial}{\partial w_j} (y_i - o_i)^2 = -\sum_{i=1}^{n} (y_i - o_i) x_{i,j}$$

- … of loss on one example:

$$\begin{aligned} \frac{\partial}{\partial w_j} (y_i - o_i)^2 = \frac{\partial (y_i - o_i)^2}{\partial o_i} \frac{\partial o_i}{\partial w_j} &= -2(y_i - o_i) \frac{\partial \langle w, x_i \rangle}{\partial w_j} \\ &= -2(y_i - o_i) x_{i,j} \end{aligned}$$

Gradient descent training for linear regression

For each w_j, initialize it to a small random value

Until termination condition met, do:

(1) Compute output $o_i = \langle w, x_i \rangle$ for each input vector x_i

(2) Update $$w \leftarrow w + \eta \cdot \underbrace{\sum_{i=1}^{n} (y_i - o_i) x_i}_{-\nabla E(w)}$$

Stochastic gradient descent (SGD) for linear regression

For each w_j, initialize it to a small random value

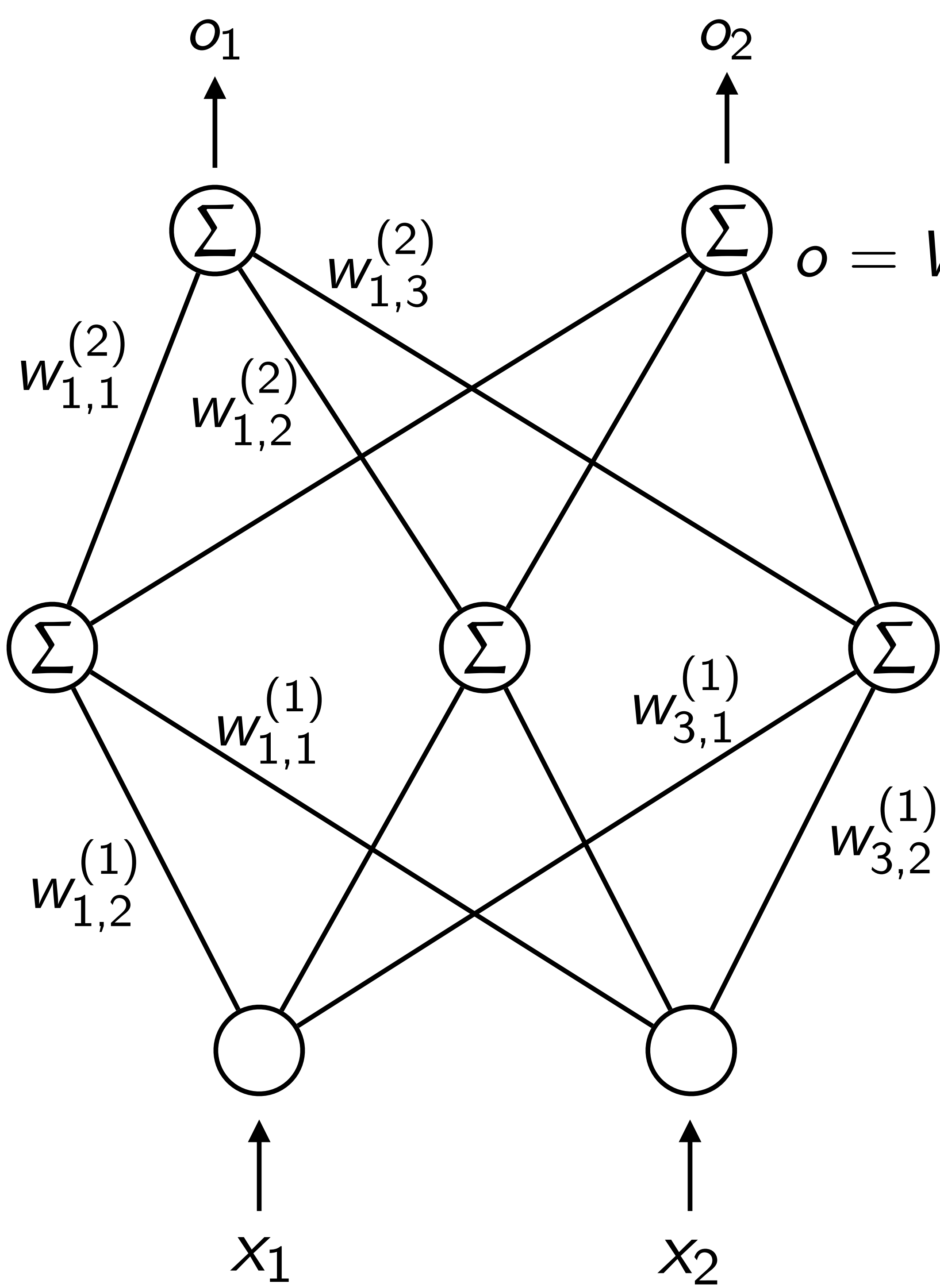
For each example (x, y) in training set, do:

(1) Compute output $o = \langle w, x \rangle$

(2) Update: $w \leftarrow w + \eta \cdot (y - o)x$

In practice, we loop over the training set multiple times until some termination criterion is met.

Multi-layer network without nonlinearity



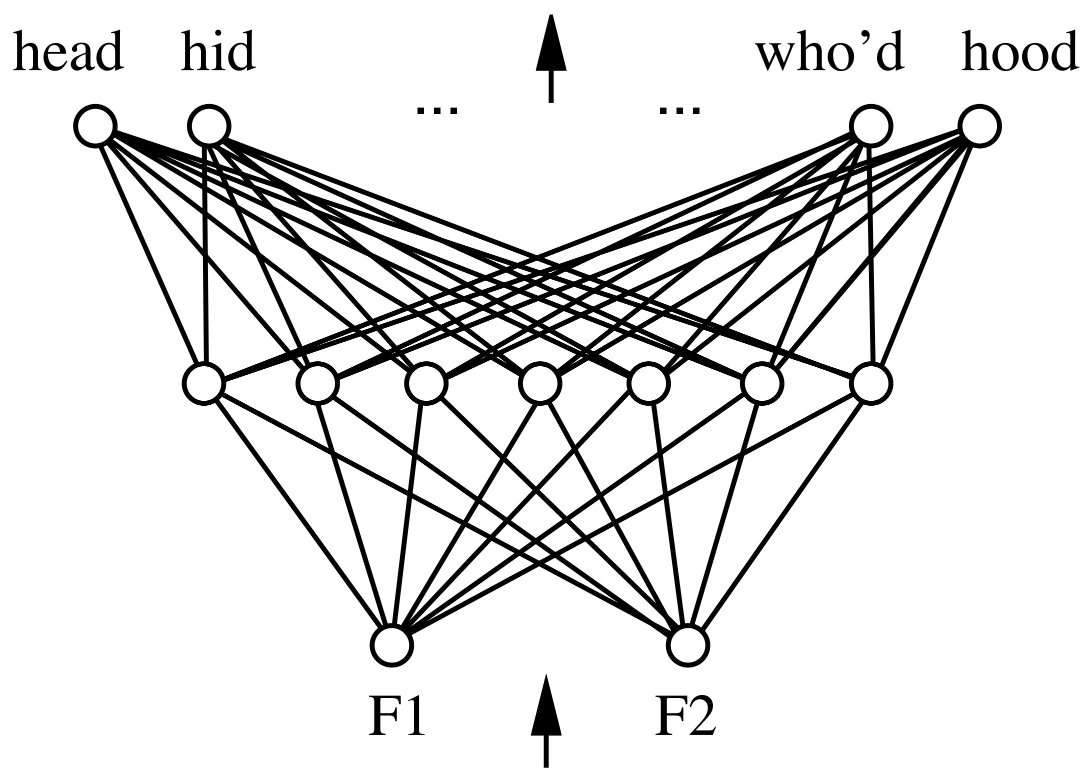
$$o = W^{(2)}W^{(1)}x = Ax$$

$$o = W^{(2)}W^{(1)}x = Ax$$

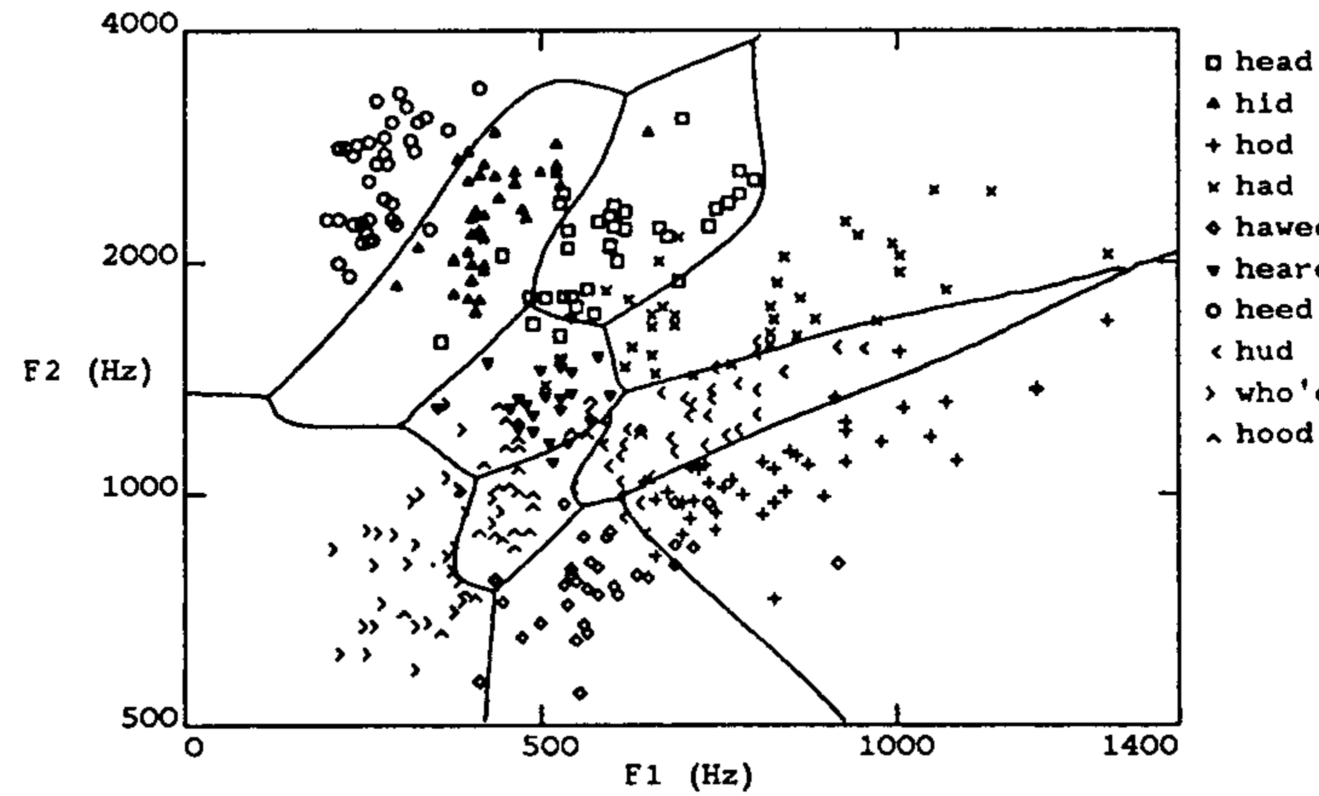
for $A = W^{(2)}W^{(1)}$

Multi-layer network with nonlinearity

Multi-layer network (with nonlinearity)



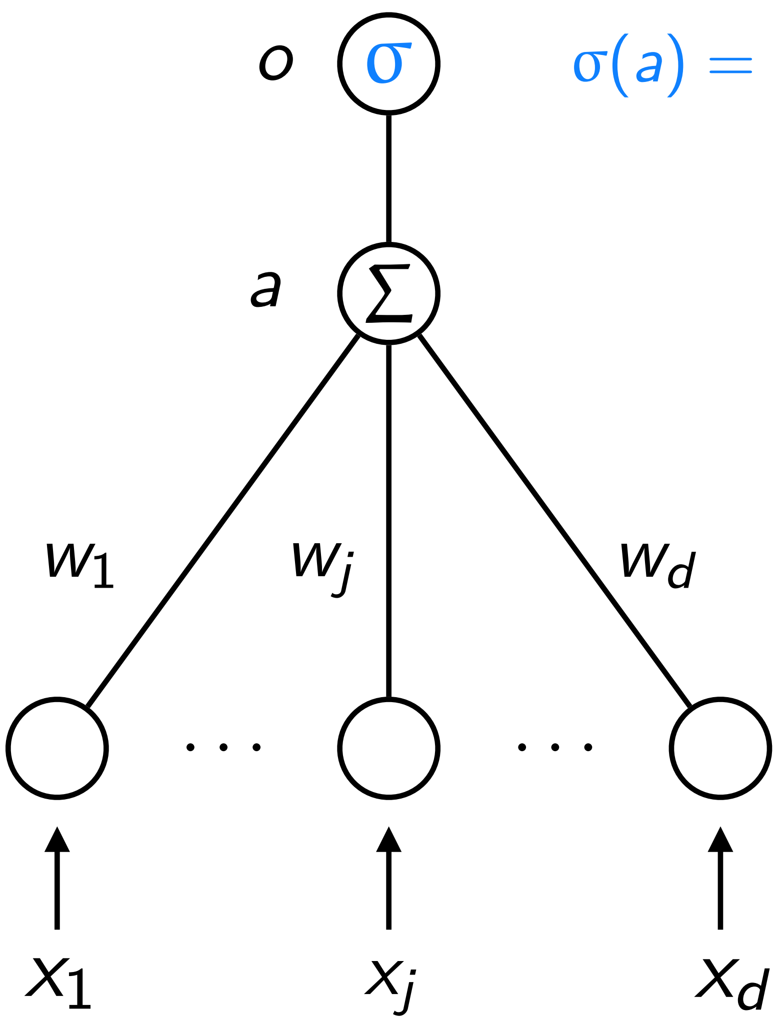
(nonlinear!) decision surface



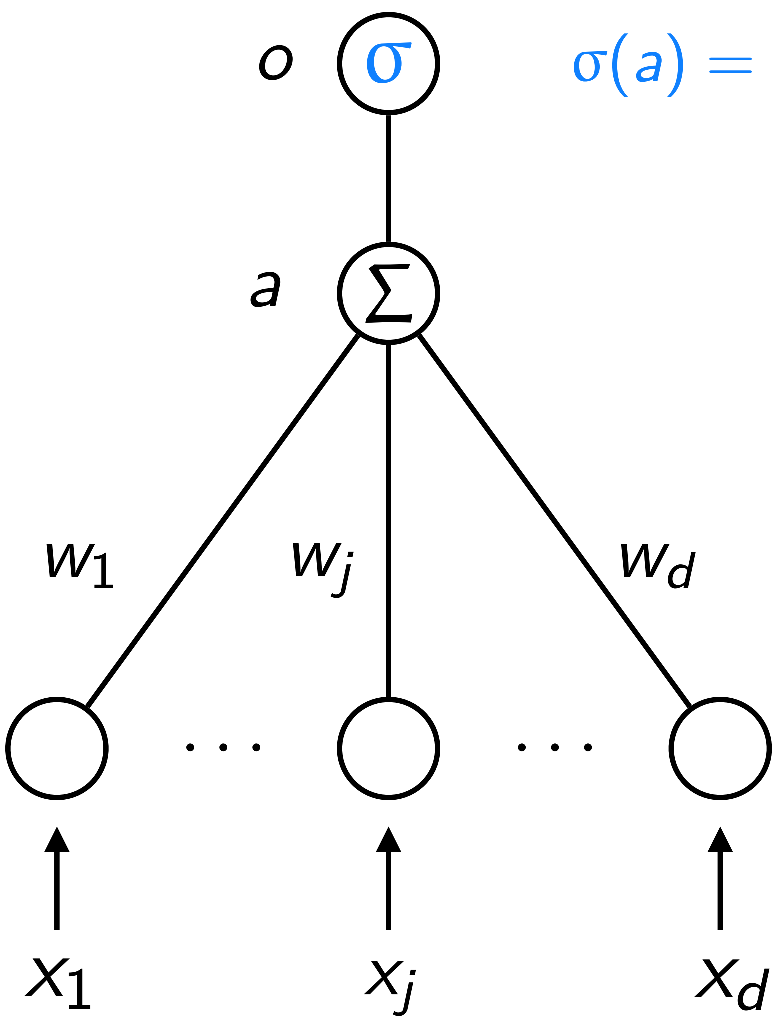
Sigmoid unit

Sigmoid function

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$



Sigmoid unit



Sigmoid function

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Assume we used squared loss:

$$E(w) = \frac{1}{2}(y - o)^2$$

How to compute gradient?

We need to compute $\frac{\partial E(w)}{w_j}$

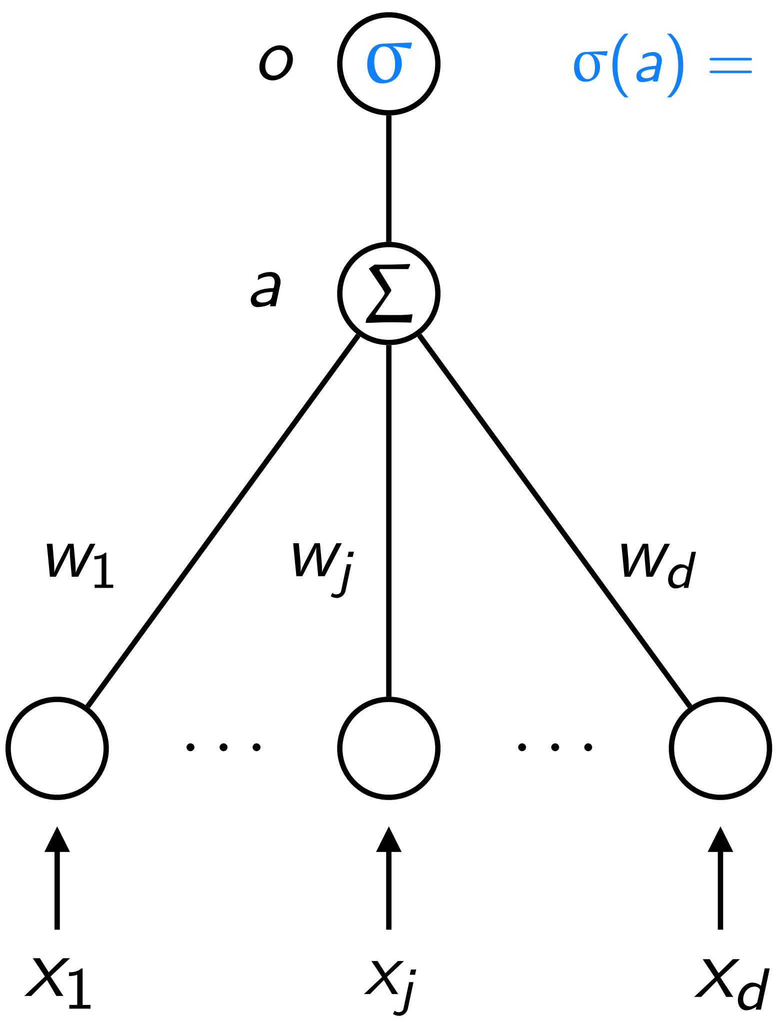
Use chain rule:

$$\frac{\partial E(w)}{w_j} = \frac{\partial E(w)}{o} \quad \frac{\partial o}{\partial a} \quad \frac{\partial a}{\partial w_j}$$

Sigmoid unit

Sigmoid function

$$\sigma(a) = \frac{1}{1+e^{-a}}$$



Assume we used squared loss:

$$E(w) = \frac{1}{2}(y-o)^2$$

How to compute gradient?

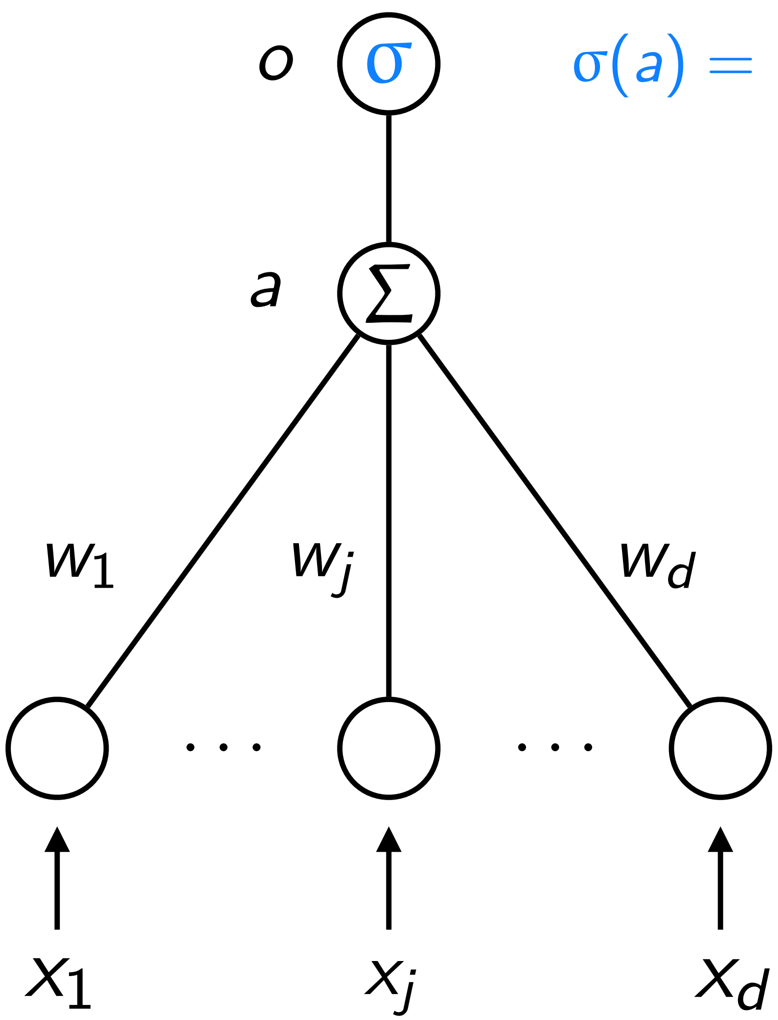
We need to compute $\dfrac{\partial E(w)}{w_j}$

Use chain rule:

$$\frac{\partial E(w)}{w_j} = \frac{\partial E(w)}{o} \quad \frac{\partial o}{\partial a} \quad \frac{\partial a}{\partial w_j}$$

$$= -(y-o) \quad \frac{\partial o}{\partial a} \quad x_j$$

Sigmoid unit



Sigmoid function

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Assume we used squared loss:

$$E(w) = \frac{1}{2}(y - o)^2$$

How to compute gradient?

We need to compute $\dfrac{\partial E(w)}{w_j}$

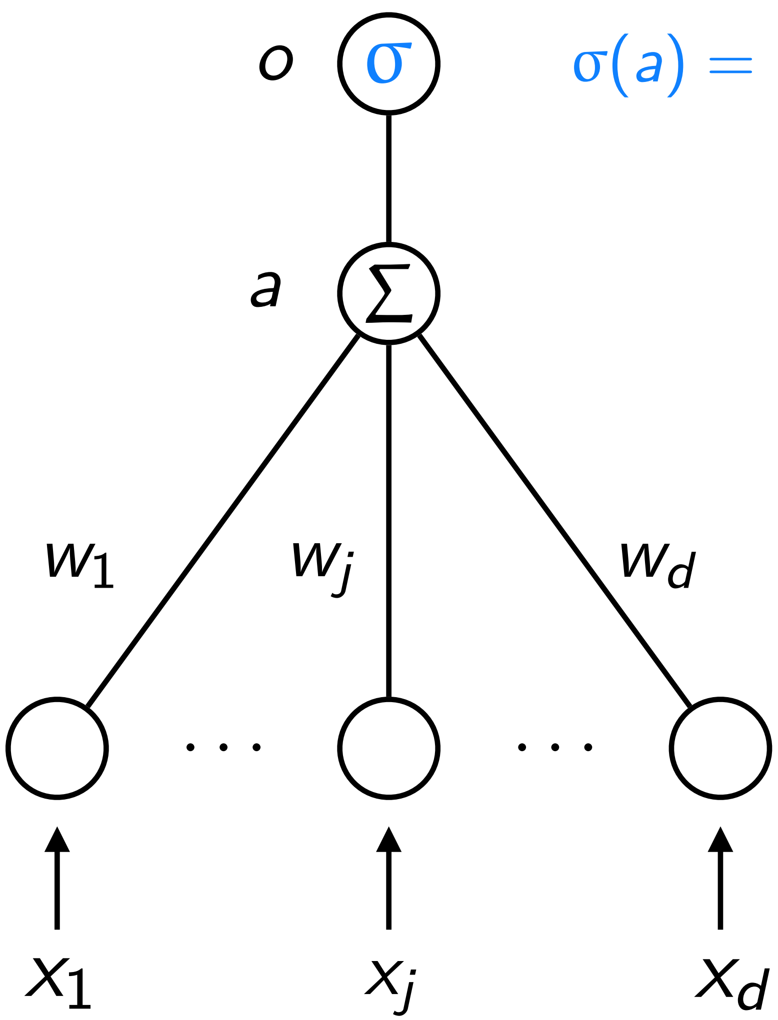
Use chain rule:

$$\frac{\partial E(w)}{w_j} = \frac{\partial E(w)}{o} \quad \frac{\partial o}{\partial a} \quad \frac{\partial a}{\partial w_j}$$

$$= -(y - o) \quad \frac{\partial o}{\partial a} \quad x_j$$

$$\frac{\partial o}{\partial a} = \frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1 - \sigma(a)) = o(1 - o)$$

Sigmoid unit



Sigmoid function

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

Assume we used squared loss:

$$E(w) = \frac{1}{2}(y-o)^2$$

How to compute gradient?

We need to compute $\dfrac{\partial E(w)}{w_j}$

Use chain rule:

$$\frac{\partial E(w)}{w_j} = \frac{\partial E(w)}{o} \quad \frac{\partial o}{\partial a} \quad \frac{\partial a}{\partial w_j}$$

$$= -(y-o) \quad \frac{\partial o}{\partial a} \quad x_j$$

$$\frac{\partial o}{\partial a} = \frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a)) = o(1-o)$$

So $\dfrac{\partial E(w)}{w_j} = -(y-o)o(1-o)x_j$

Stochastic gradient descent (SGD) for “sigmoid” regression

For each w_j , initialize it to a small random value

For each example (x, y) in training set, do:

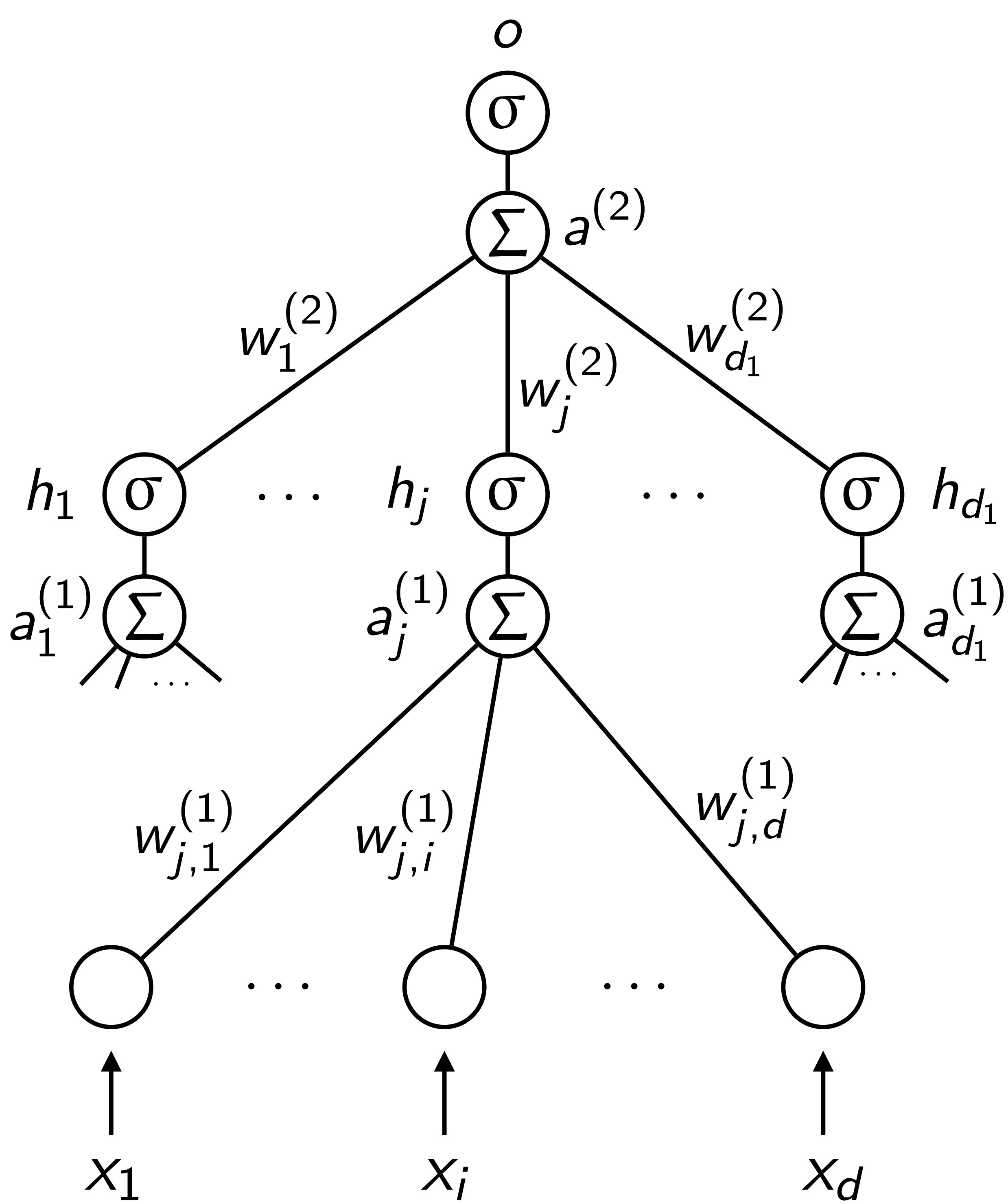
(1) Compute preactivation $a = \langle w, x \rangle$

(2) Compute output $o = \sigma(a)$

(3) Update: $w \leftarrow w + \eta(y - o)o(1 - o)x$

In practice, we loop over the training set multiple times until some termination criterion is met.

Forward propagation



Unit computations (in reverse order)

$$o(x) = \sigma(a^{(2)}(x))$$

$$a^{(2)}(x) = \langle w^{(2)}, h(x) \rangle = \sum_{j=1}^{d_1} w_j^{(2)} h_j(x)$$

$$h_j(x) = \sigma(a_j^{(1)}(x))$$

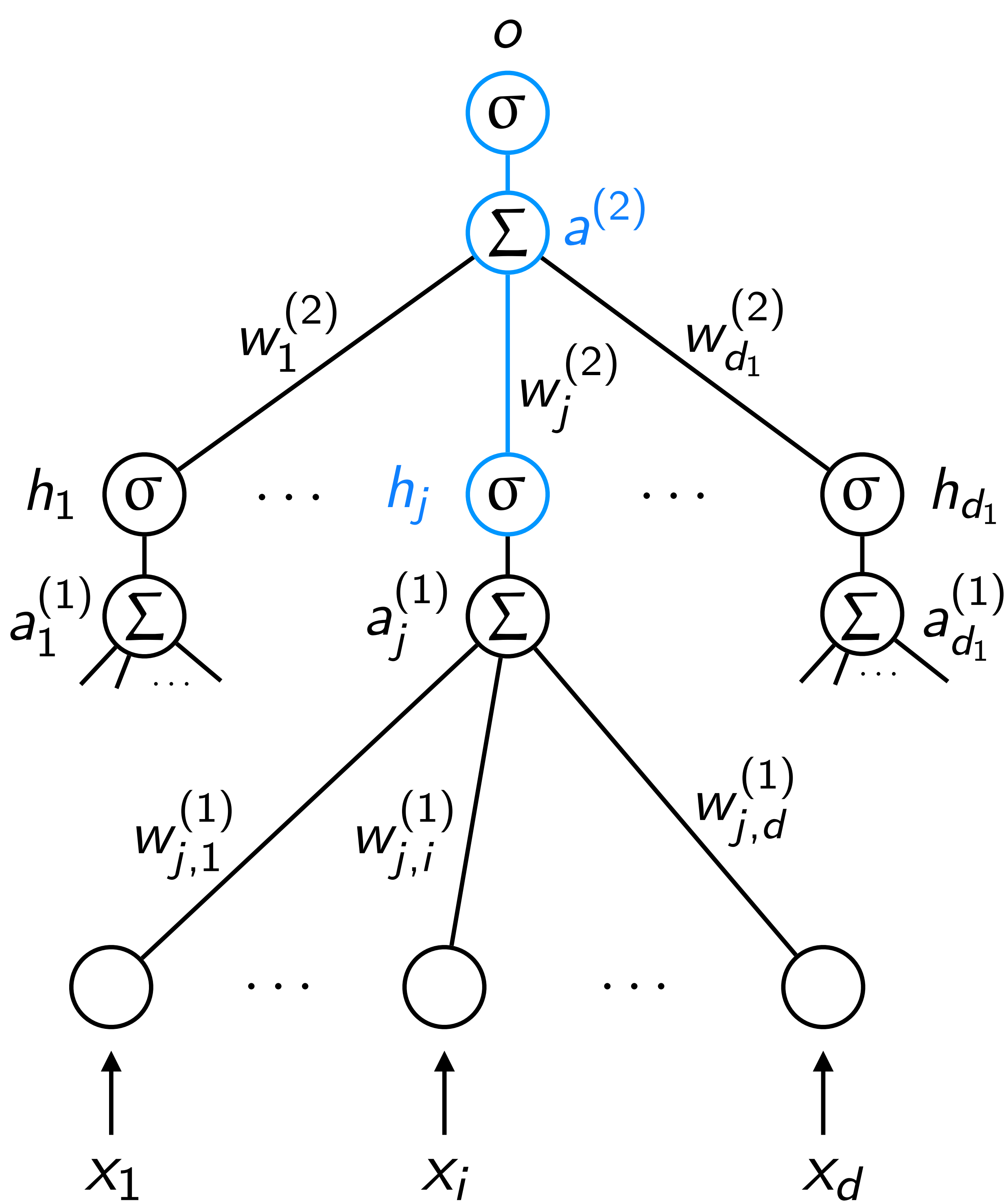
$$a_j^{(1)}(x) = \langle w_j^{(1)}, x \rangle = \sum_{i=1}^{d} w_{j,i}\, x_i$$

(Note the following vector definitions)

$$w_j^{(1)} = (w_{j,1}^{(1)} \;\; w_{j,2}^{(1)} \;\; \ldots \;\; w_{j,d}^{(1)})$$

$$w^{(2)} = (w_1^{(2)} \;\; w_2^{(2)} \;\; \ldots \;\; w_{d_1}^{(2)})$$

Backpropagation - weights to output layer



First, we compute the gradient updates for the weights going to the output layer:

$$\frac{\partial E(w)}{\partial w_j^{(2)}} = \underbrace{\frac{\partial E(w)}{\partial o}\frac{\partial o}{\partial a^{(2)}}}_{\delta}\frac{\partial a^{(2)}}{\partial w_j^{(2)}}$$

$$= \underbrace{-(y-o)o(1-o)}_{\delta}\, h_j$$

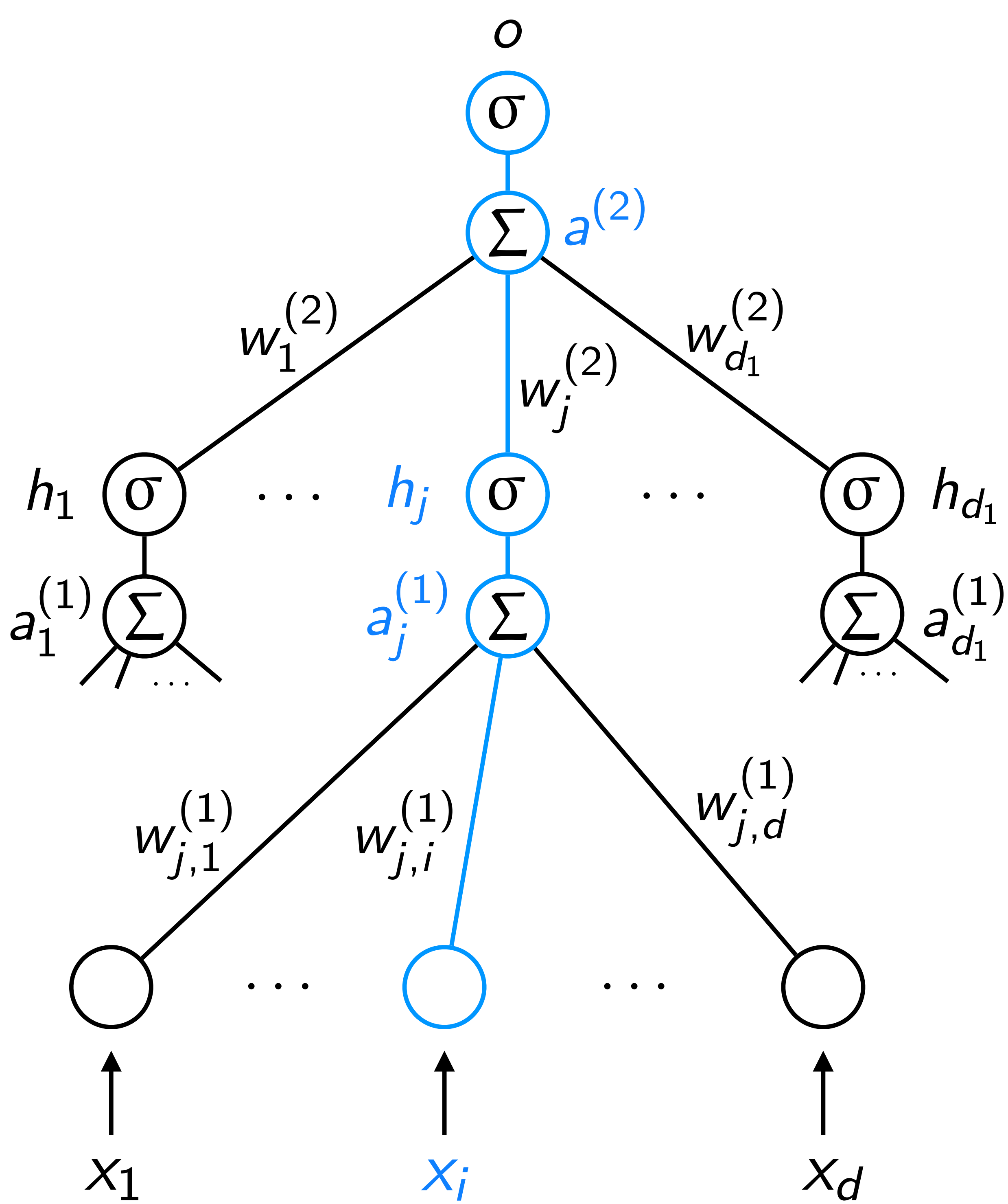
We are using the squared error:

$$E(w) = \frac{1}{2}(y - o(x))^2$$

Recall the sigmoid function:

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

Backpropagation - weights to hidden layer



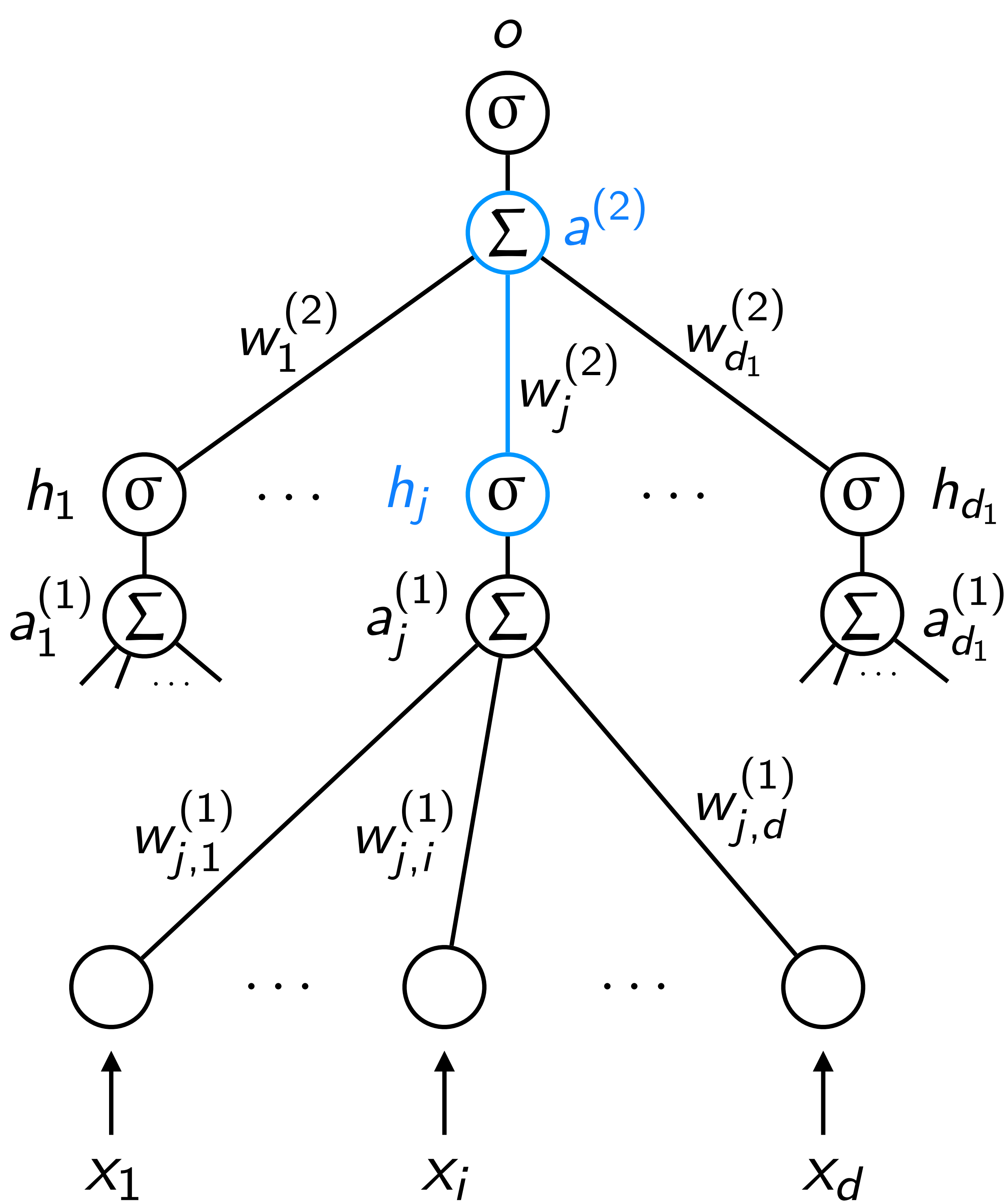
Next, we compute the gradient updates for the weights going to the hidden layer:

$$\frac{\partial E(w)}{\partial w_{j,i}^{(1)}}$$

$$= \frac{\partial E(w)}{\partial o} \frac{\partial o}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$= \delta \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

Backpropagation - weights to hidden layer



Next, we compute the gradient updates for the weights going to the hidden layer:

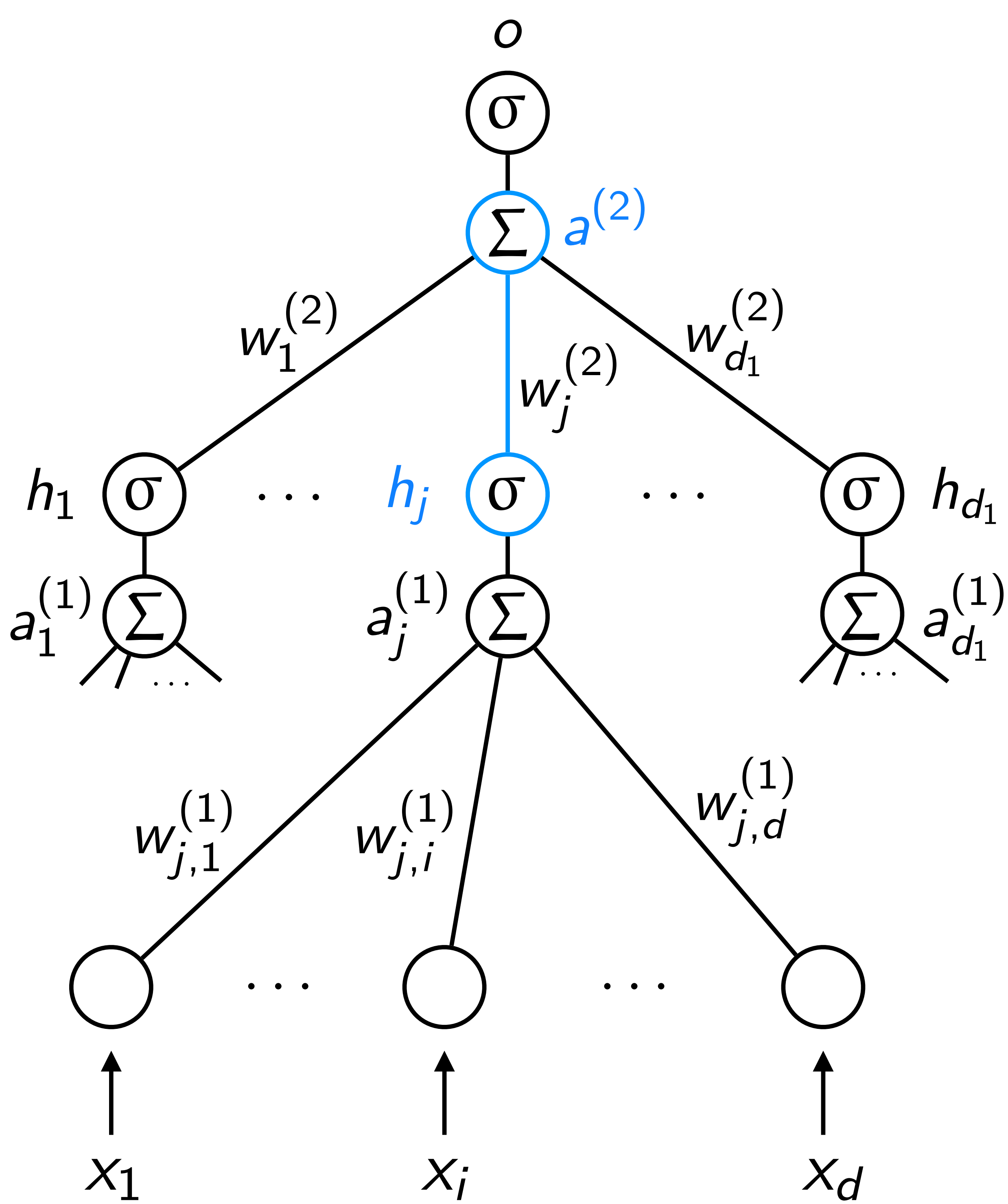
$$\frac{\partial E(w)}{\partial w_{j,i}^{(1)}}$$

$$= \frac{\partial E(w)}{\partial o} \frac{\partial o}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$= \delta \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$\boxed{\frac{\partial a^{(2)}}{\partial h_j} = w_j^{(2)}}$$

Backpropagation - weights to hidden layer



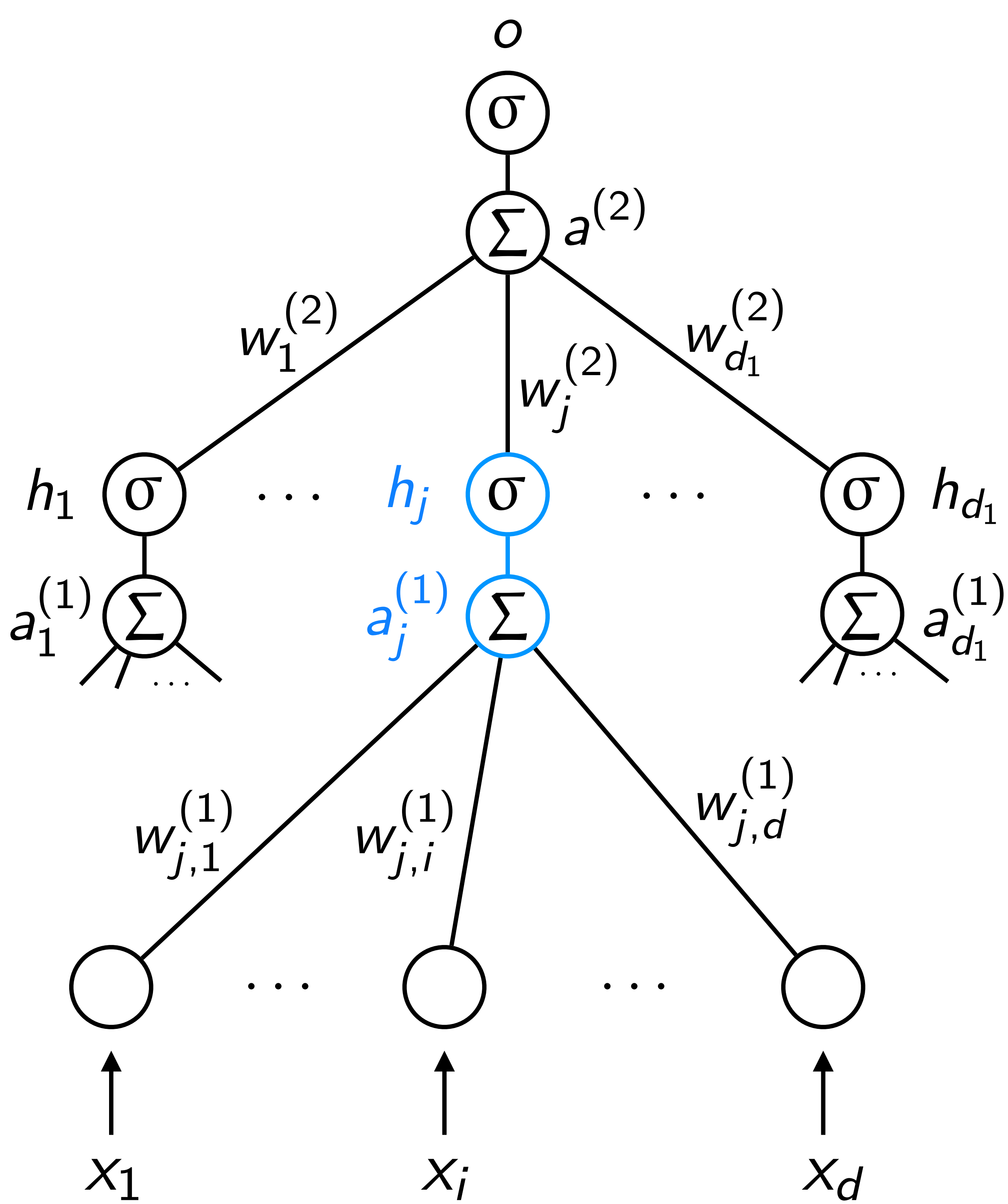
Next, we compute the gradient updates for the weights going to the hidden layer:

$$\frac{\partial E(w)}{\partial w_{j,i}^{(1)}}$$

$$= \frac{\partial E(w)}{\partial o} \frac{\partial o}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$= \delta \, w_j^{(2)} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

Backpropagation - weights to hidden layer



Next, we compute the gradient updates for the weights going to the hidden layer:

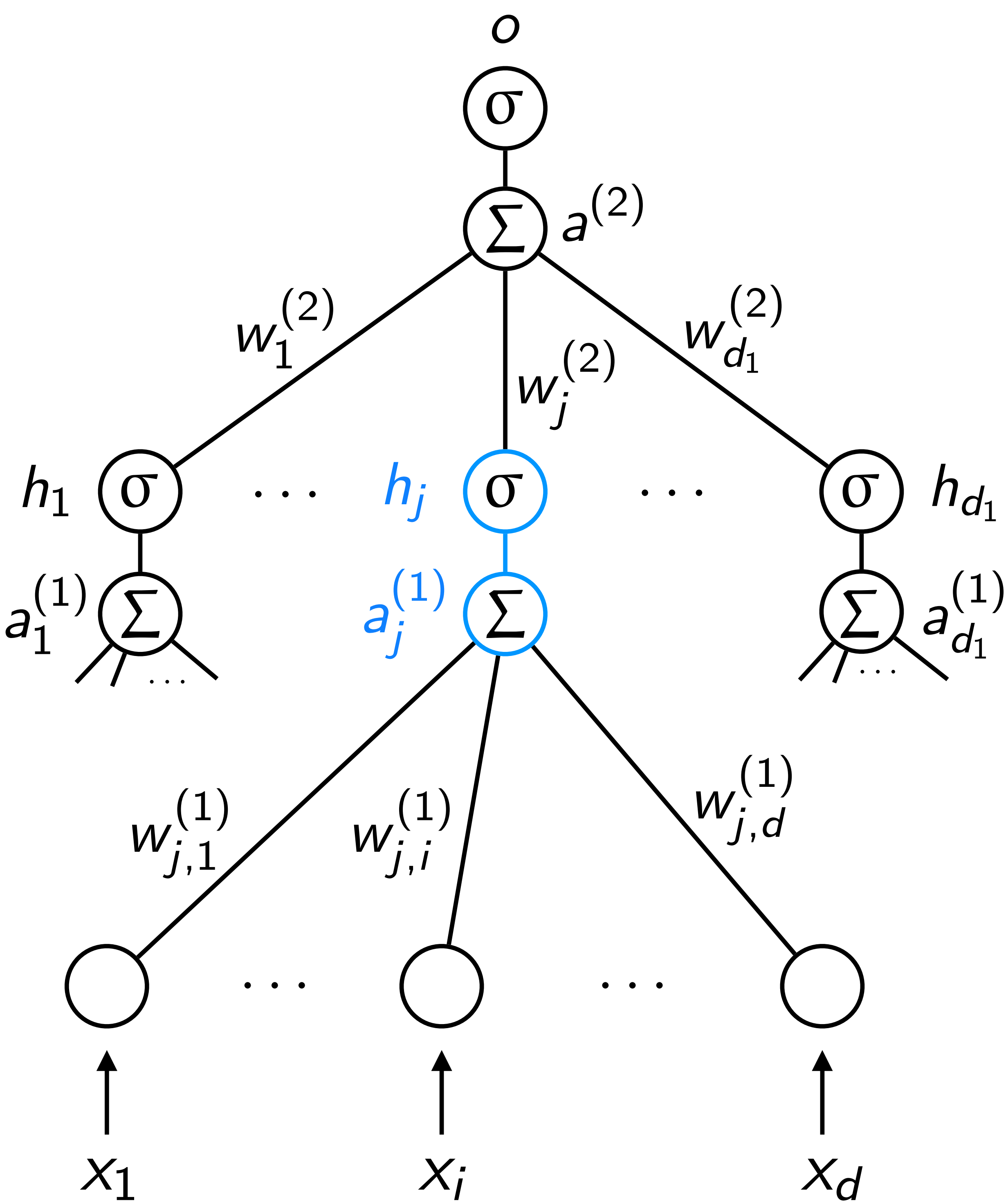
$$\frac{\partial E(w)}{\partial w_{j,i}^{(1)}}$$

$$= \frac{\partial E(w)}{\partial o} \frac{\partial o}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$= \delta \, w_j^{(2)} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$\boxed{\frac{\partial h_j}{\partial a_j^{(1)}} = h_j(1 - h_j)}$$

Backpropagation - weights to hidden layer



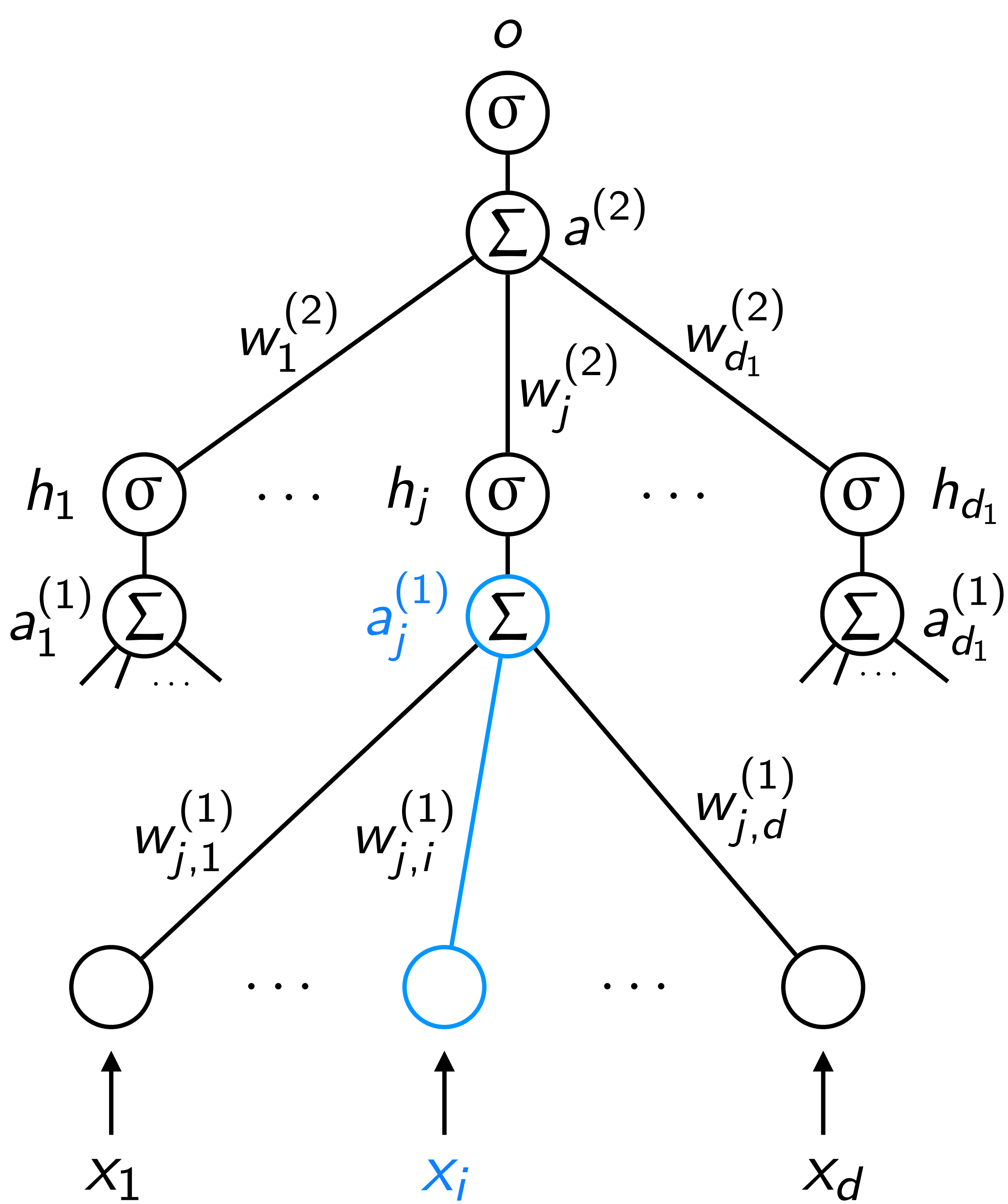
Next, we compute the gradient updates for the weights going to the hidden layer:

$$\frac{\partial E(w)}{\partial w_{j,i}^{(1)}}$$

$$= \frac{\partial E(w)}{\partial o} \frac{\partial o}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$= \delta \, w_j^{(2)} h_j (1 - h_j) \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

Backpropagation - weights to hidden layer



Next, we compute the gradient updates for the weights going to the hidden layer:

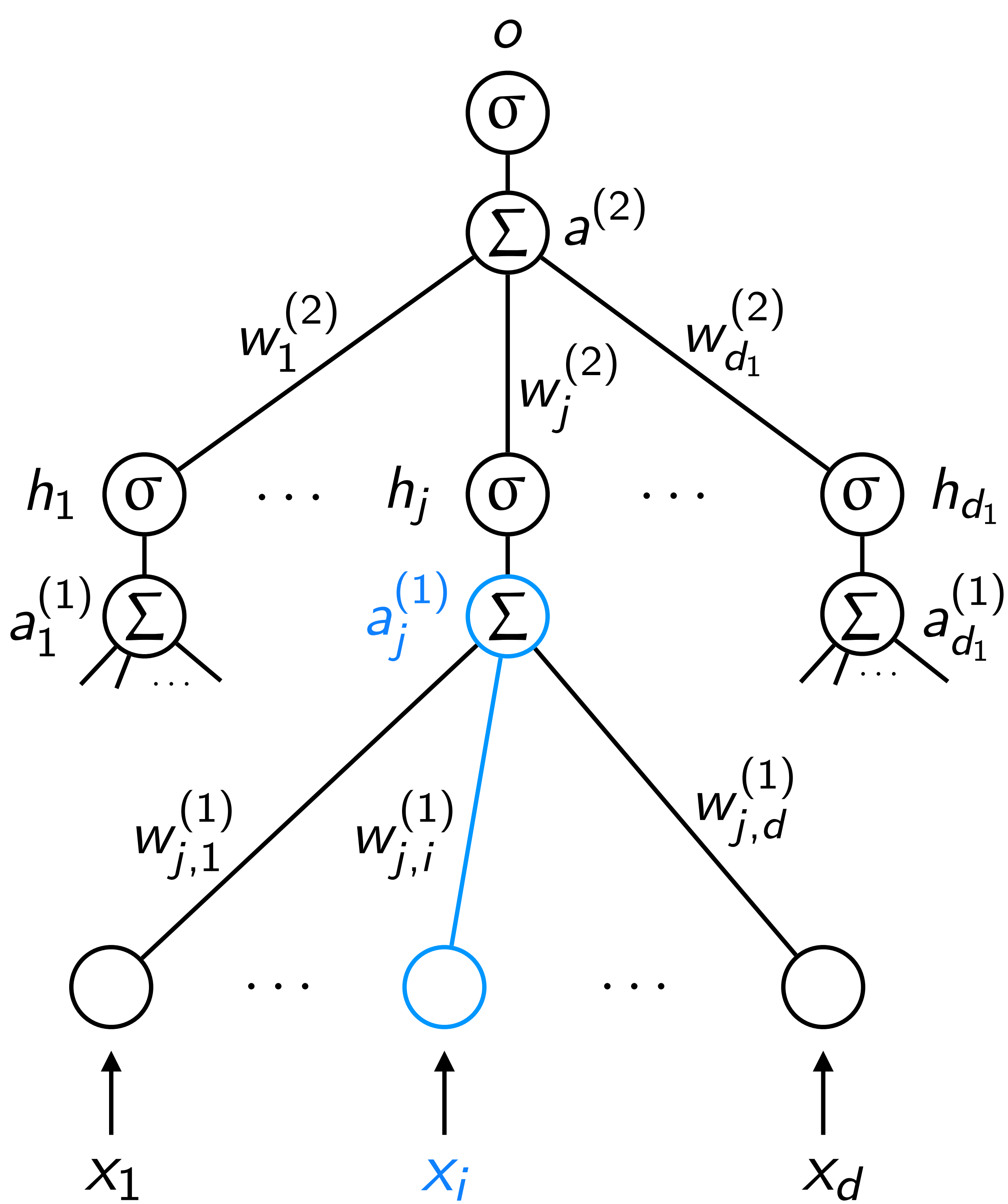
$$\frac{\partial E(w)}{\partial w_{j,i}^{(1)}}$$

$$= \frac{\partial E(w)}{\partial o} \frac{\partial o}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$= \delta \, w_j^{(2)} h_j (1 - h_j) \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$\boxed{\frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}} = x_i}$$

Backpropagation - weights to hidden layer



Next, we compute the gradient updates for the weights going to the hidden layer:

$$\frac{\partial E(w)}{\partial w_{j,i}^{(1)}}$$

$$= \frac{\partial E(w)}{\partial o} \frac{\partial o}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$= \delta\, w_j^{(2)} h_j (1 - h_j) x_i$$

How to use backprop to train neural networks

(1) Initialization: use random initialization (this is important!)

randomness is important to:

- to avoid problematic (all zero) gradients
- break symmetry (think about why)

(2) Loop over training examples repeatedly (use SGD)

(i) forward propagation
(ii) backprop

(3) Stop (there are different choices of stopping criteria)

Universal Approximation Theorem

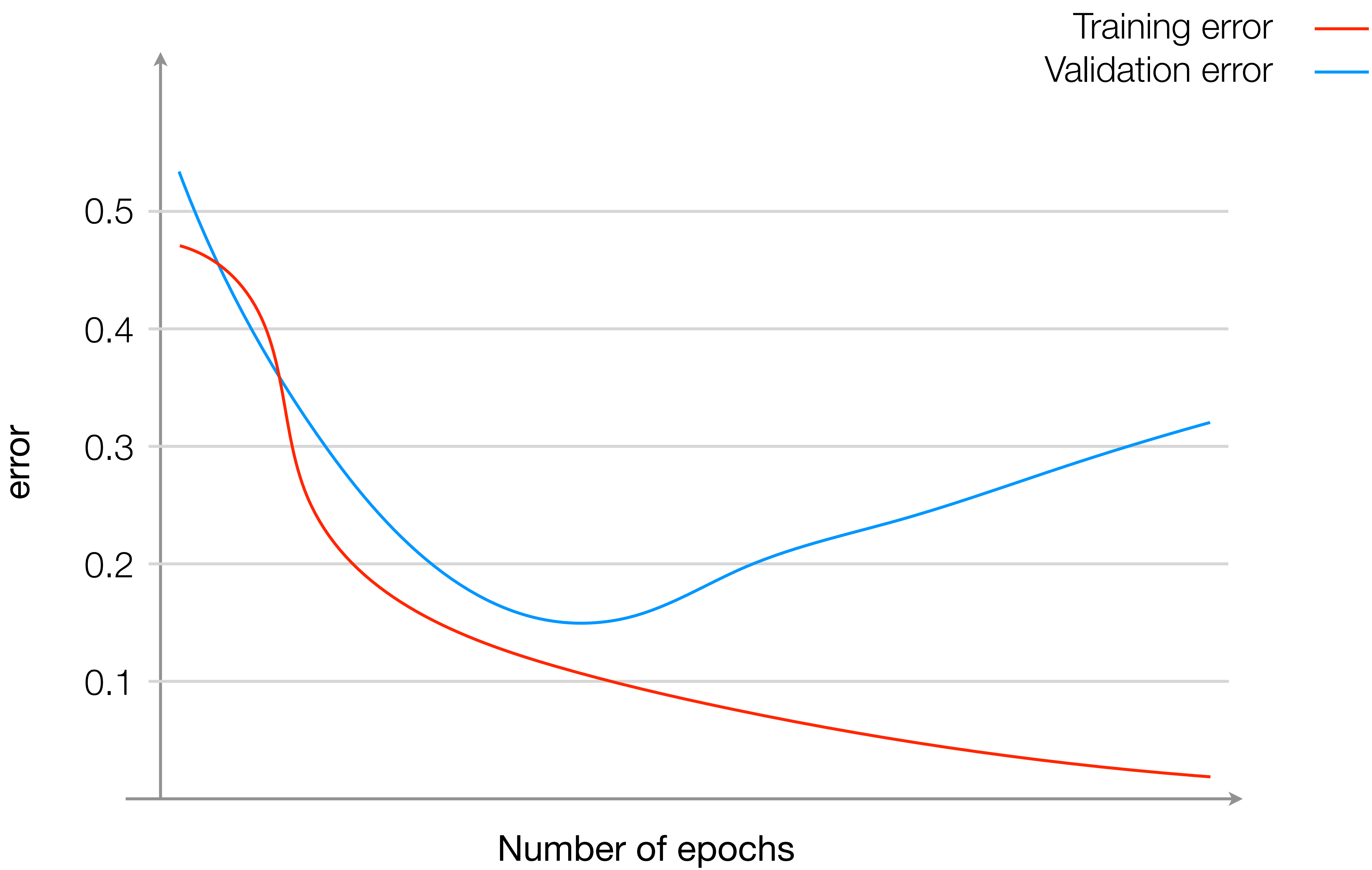
Any bounded, continuous function over the input space $[0, 1]^d$ can be approximated arbitrarily well (i.e. with arbitrarily small error) using a neural network with only one hidden layer (with a number of hidden nodes depending on the function).

The hidden units have sigmoid activation functions, while the output unit is linear (no activation function).

(Cybenko, 1989)

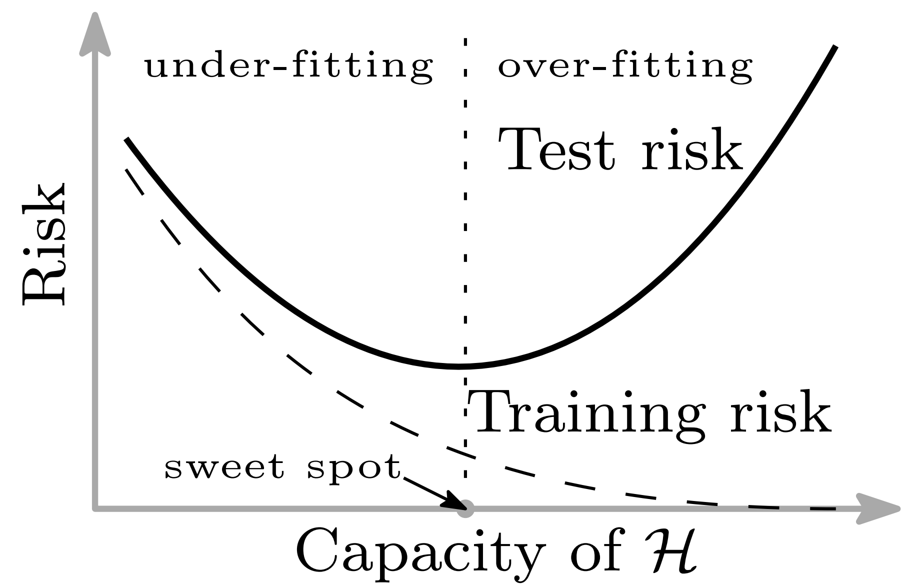
A similar result can be proved for other activation functions
(Hornik, 1991)

Early Stopping

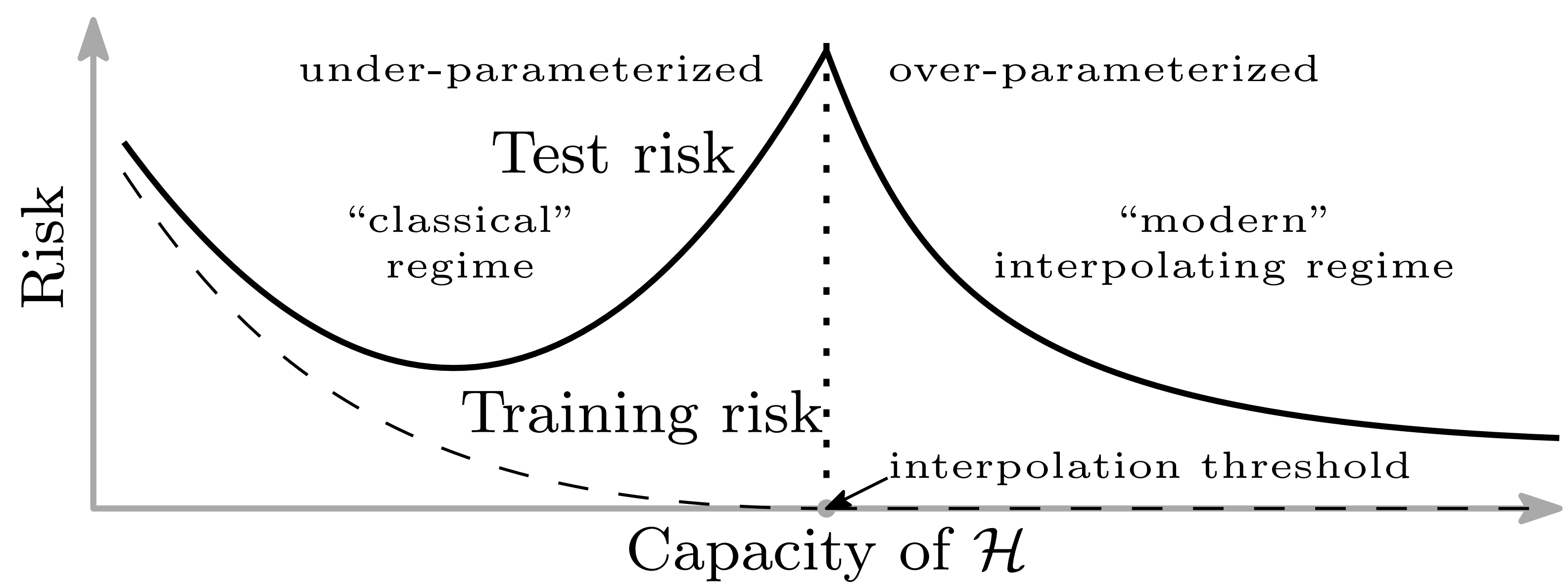


Double descent

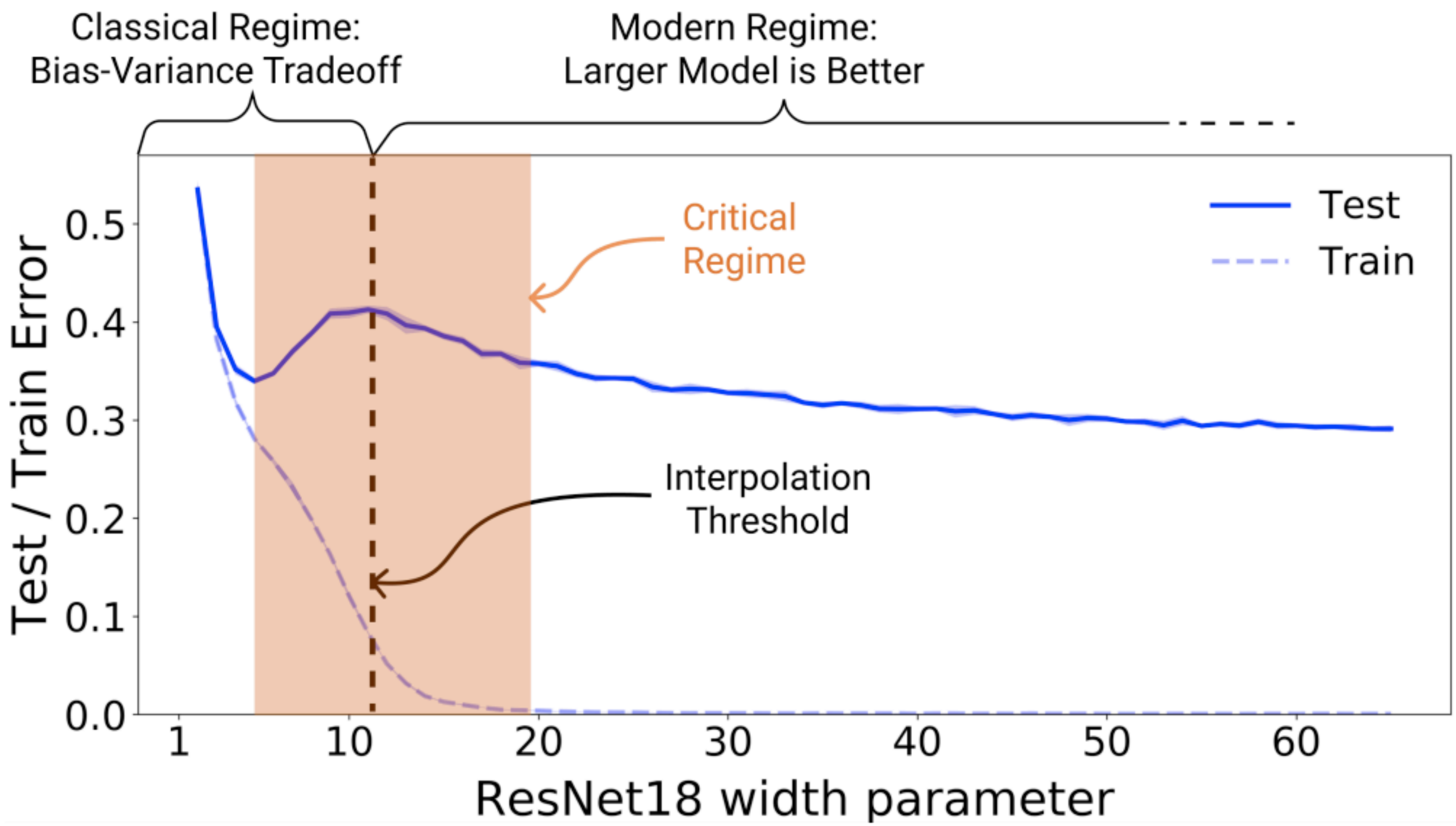
Classical view of learning



Double descent curve



Double descent



More reading: https://windowsontheory.org/2019/12/05/deep-double-descent

L2-Regularization ("Weight decay")

For a network with an input layer and L additional layers (including the output layer)

Weights matrix feeding into layer l

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - o_i)^2 + \frac{\lambda}{2}\sum_{\ell=1}^{L}\left\| W^{(\ell)} \right\|_F^2$$

Frobenius norm of $W^{(\ell)}$, defined as $\displaystyle\sum_{i=1}^{d_{\ell-1}}\sum_{j=1}^{d_\ell}(W_{ij}^{(\ell)})^2$

Why is this helpful?

Tends to shrink the weights (spreads them out more)

There is theoretical support; this is a form of "capacity control"

L1-Regularization

For a network with an input layer and L additional layers (including the output layer)

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - o_i)^2 + \lambda \sum_{\ell=1}^{L}\sum_{i=1}^{d_{\ell-1}}\sum_{j=1}^{d_\ell}\left|W_{ij}^{(\ell)}\right|$$

Why is this helpful?

Tends to shrink the weights and make many of them zero (a thinner network)

There is even stronger theoretical support for this (again, capacity control)

For valid generalization, the size of the weights is more important than the size of the network

Peter L. Bartlett
Department of Systems Engineering
Research School of Information Sciences and Engineering
Australian National University
Canberra, 0200 Australia
Peter.Bartlett@anu.edu.au

Abstract

This paper shows that if a large neural network is used for a pattern classification problem, and the learning algorithm finds a network with small weights that has small squared error on the training patterns, then the generalization performance depends on the size of the weights rather than the number of weights. More specifically, consider an ℓ-layer feed-forward network of sigmoid units, in which the sum of the magnitudes of the weights associated with each unit is bounded by A. The misclassification probability converges to an error estimate (that is closely related to squared error on the training set) at rate $O((cA)^{\ell(\ell+1)/2}\sqrt{(\log n)/m})$ ignoring log factors, where m is the number of training patterns, n is the input dimension, and c is a constant. This may explain the generalization performance of neural networks, particularly when the number of training examples is considerably smaller than the number of weights. It also supports heuristics (such as weight decay and early stopping) that attempt to keep the weights small during training.

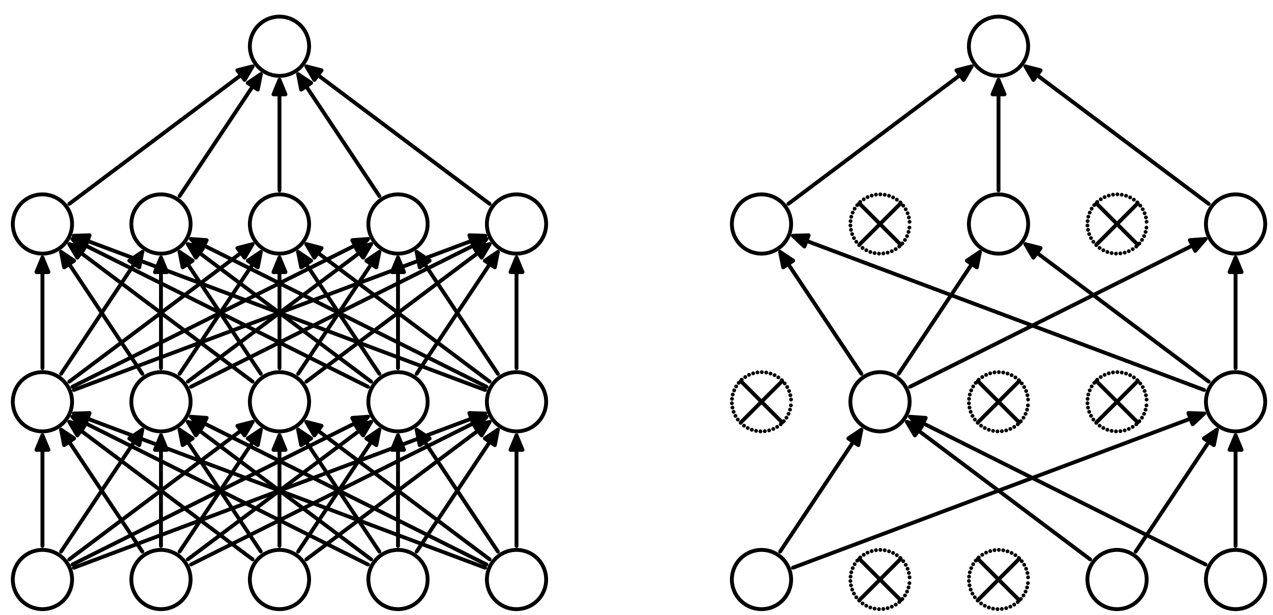
[Neural Information Processing Systems, 1996]

(Srivastava et al., 2014)
"Dropout: A Simple Way to Prevent Neural Networks from Overfitting"

Dropout

During training

- In each iteration/update step, train a "thinned" version of the network:
  - "Thinning" means we randomly, independently drop out (remove) each node (along with all its incoming and outgoing edges) with probability ***p***
  - Do forward and backpropagation on the thinned network



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Dropout

- Suppose that the original network has *m* nodes and each node is independently dropped with probability *p*
- How many possible thinned networks are there?

Dropout

- Suppose that the original network has *m* nodes and each node is independently dropped with probability *p*
- How many possible thinned networks are there? 2^m

Dropout

- Suppose that the original network has *m* nodes and each node is independently dropped with probability *p*
- How many possible thinned networks are there? 2^m
- This is equivalent to drawing the thinned network from a set of 2^m thinned networks (a binary choice for each node)
- The number of nodes in a thinned network follows a binomial distribution with success probability *p*

Dropout

(Srivastava et al., 2014)
"Dropout: A Simple Way to Prevent Neural Networks from Overfitting"

At test time

- Use the original network, but scale each weight by 1 - *p*. Why 1 - *p*? (its expected value)

Dropout reduces overfitting!

Mathy intuition:

- Dropout is like training with input noise
- Forces network to be robust to perturbations
- Network responds by spreading out its weight (better not rely on any node or connection too much!)

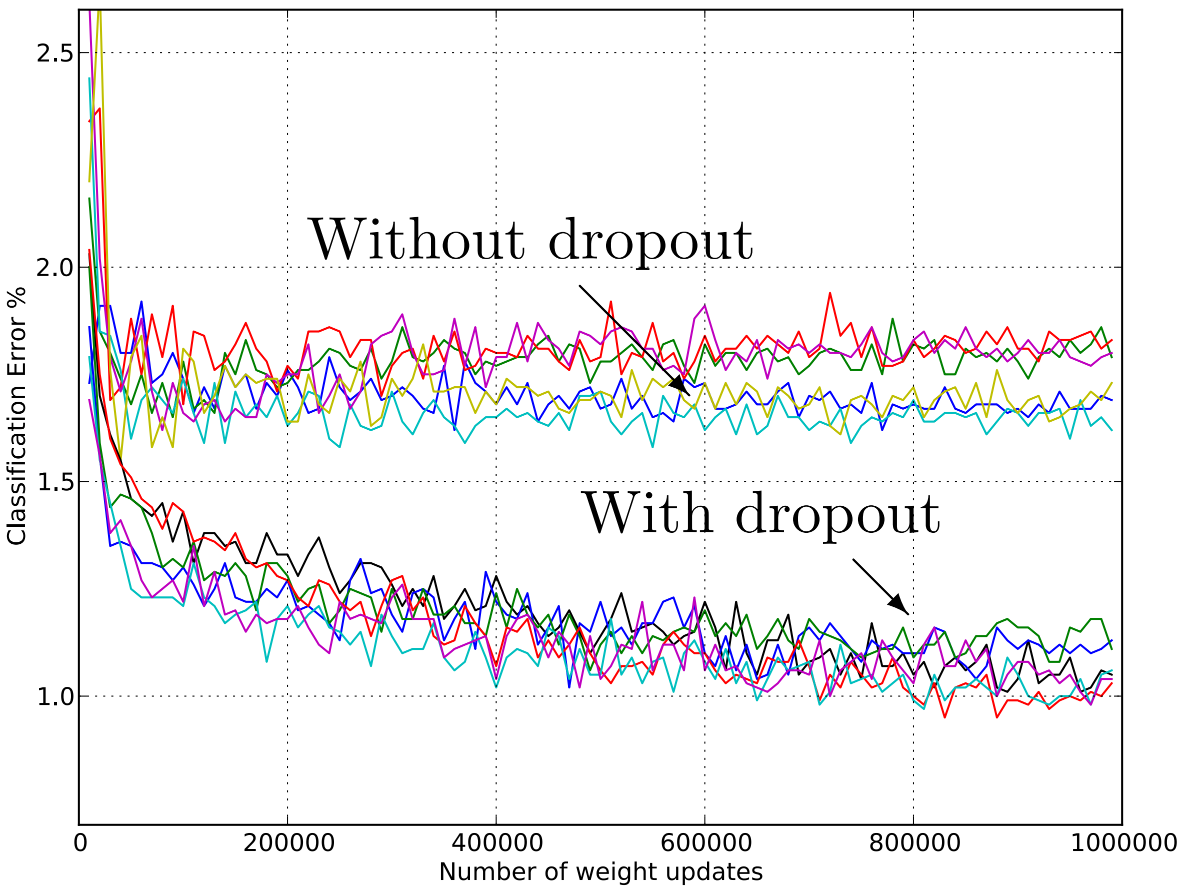


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

ML Hipster
@ML_Hipster

Follow

Spider ANN spider ANN
Autoencoded deep spider ANN
Trains on images any size
Classify them as good as eyes
Drop out!
Backprop that spider ANN

3:07 AM - 11 Sep 2014

50 Retweets **43** Likes

50 43

Q: How many machine learners does it take to change a lightbulb?
A: Isn't that a database thing?

62 56

ML Hipster @ML_Hipster · 5 Apr 2014
This ML algorithm is rated R. It contains: Strong Assumptions, Frequent Sampling, and Matrix Inversions. Restricted to masochistic users.

1 79 97

ML Hipster @ML_Hipster · 30 Mar 2014
The simplest explanation is that when Occam tried to grow a beard it looked like crap.

1 89 83