From Statistical to Deep Learning

A Primer

Frederic Koriche

Centre de Recherche en Informatique de Lens, CNRS
Université d'Artois, France



ia2-2026 — Touquet-Paris-Plage, France — 19-23 October, 2026

Outline

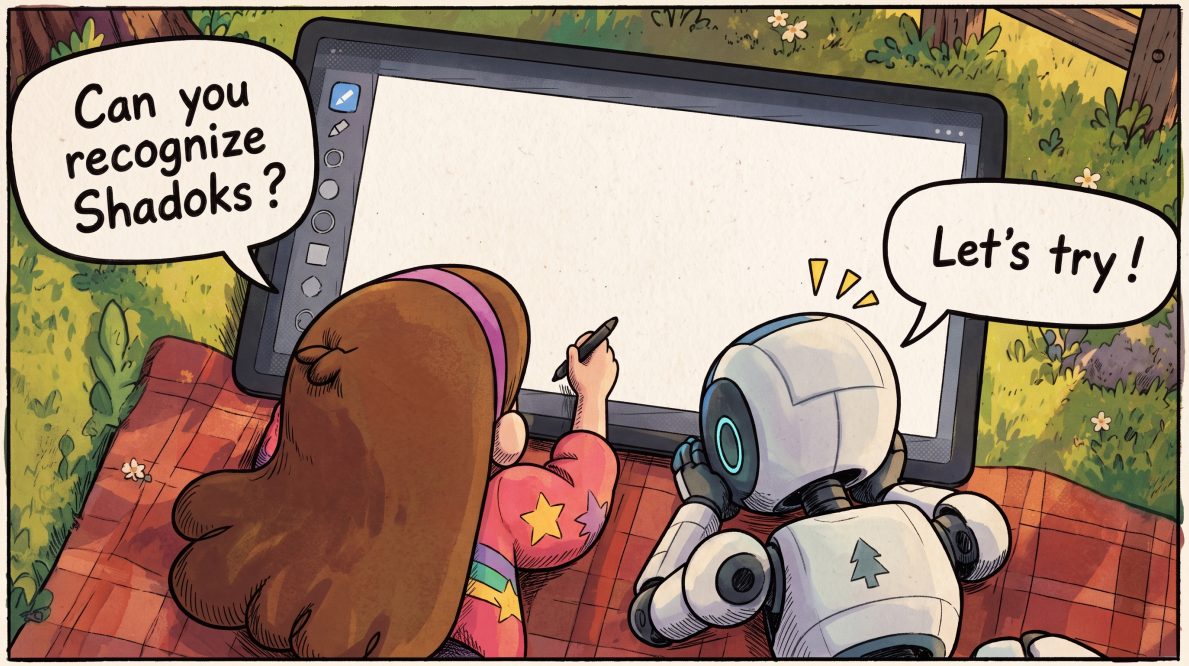
1 A Gentle Start

2 Statistical Learning

3 Deep Learning

4 Perspectives

What is "Machine Learning"?



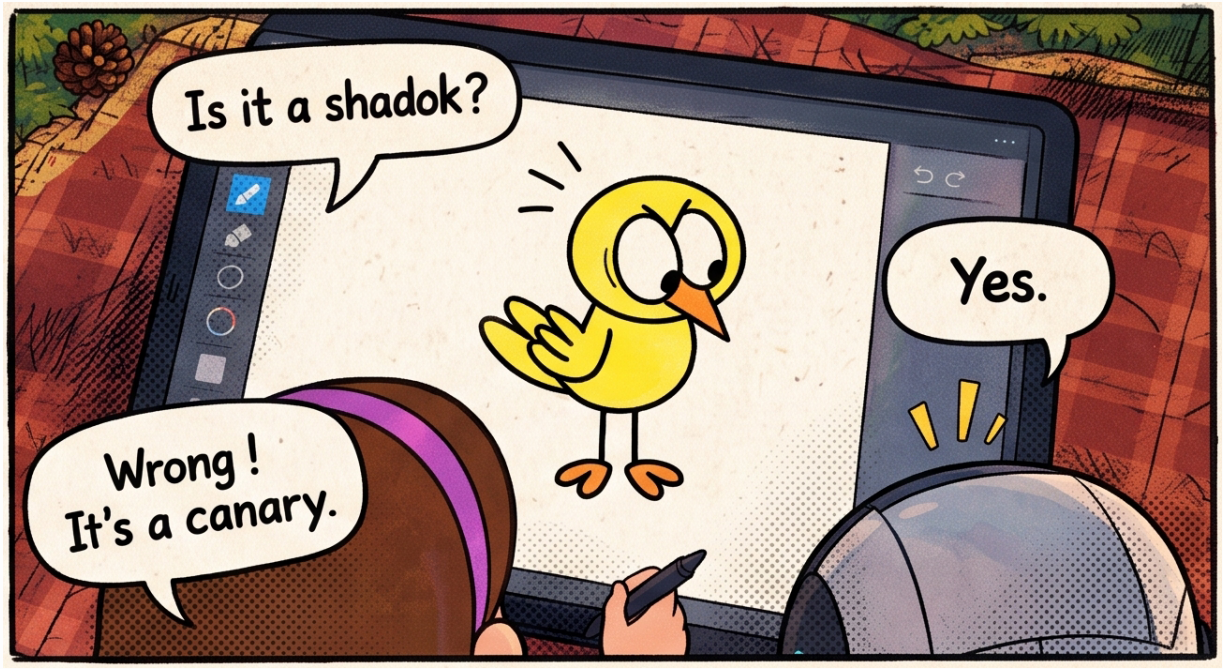
Consider the problem of recognizing "shadoks" from drawings.

What is "Machine Learning"?



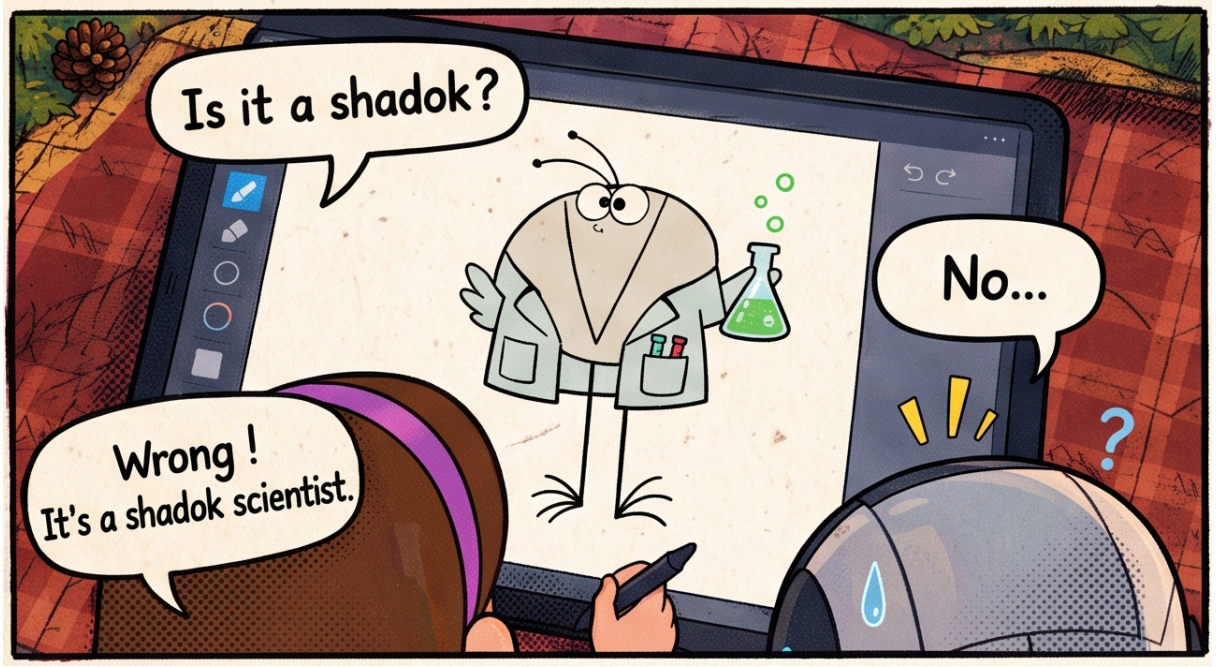
During each round, the algorithm is presented with a drawing and is asked to predict whether it is a "shadok" or not.
After making its prediction, the algorithm is informed of the correct answer.

What is "Machine Learning"?



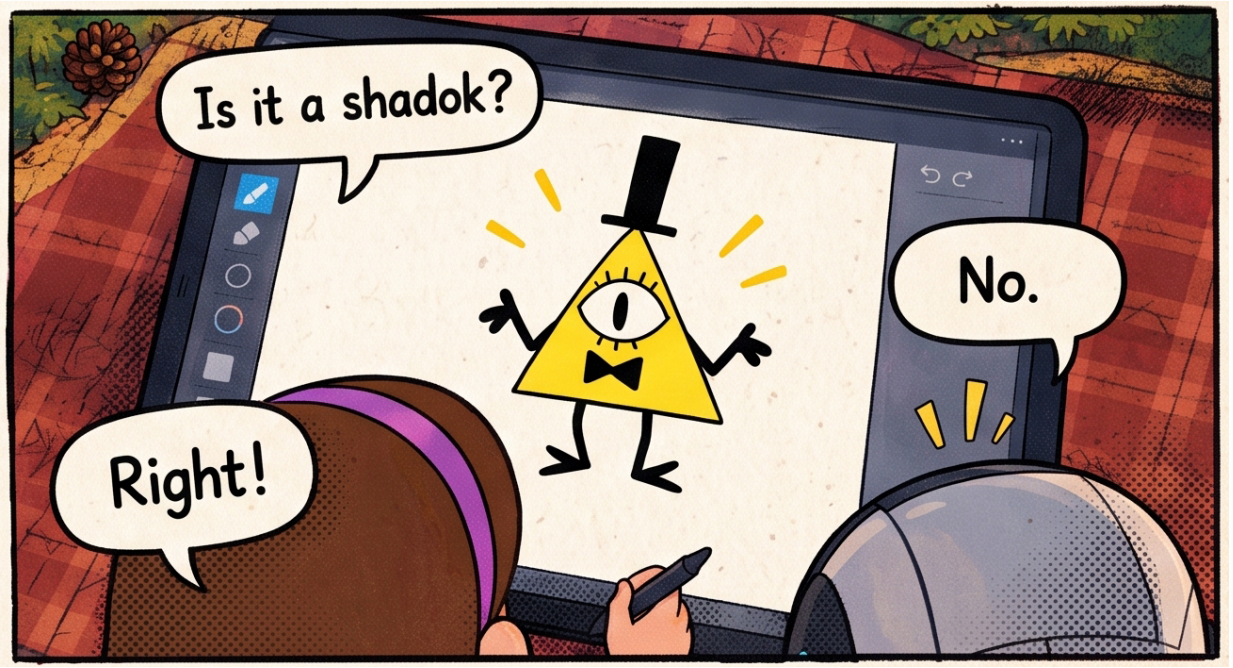
During each round, the algorithm is presented with a drawing and is asked to predict whether it is a "shadok" or not.
After making its prediction, the algorithm is informed of the correct answer.

What is "Machine Learning"?



During each round, the algorithm is presented with a drawing and is asked to predict whether it is a "shadok" or not.
After making its prediction, the algorithm is informed of the correct answer.

What is "Machine Learning"?



The algorithm learns to classify shadoks if it makes fewer and fewer prediction mistakes over time.

Defining a Learning Problem

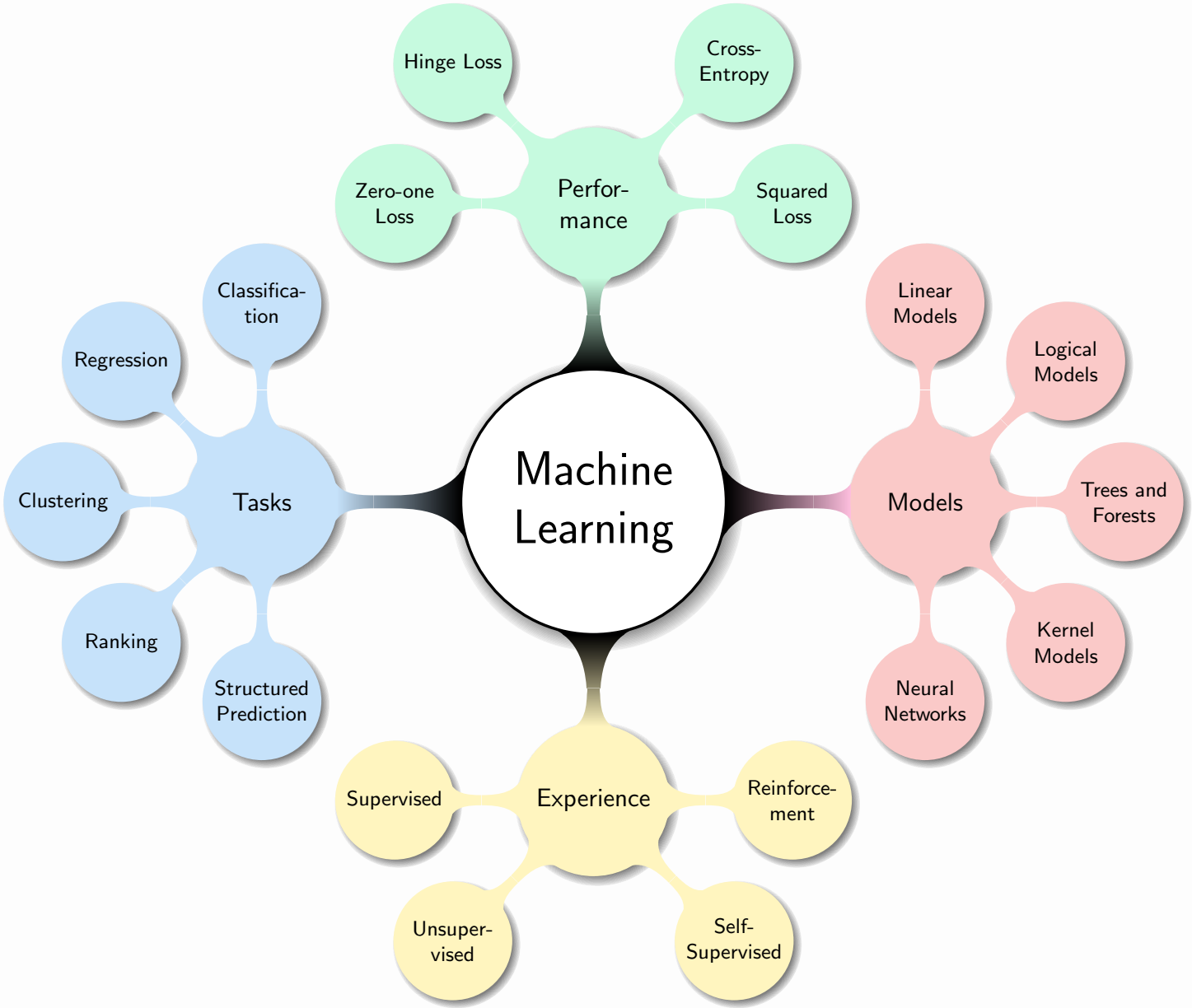
Tom Mitchell (1997)

Machine Learning is *the study of computer algorithms that improve automatically through experience*.

To fully specify a learning problem, we must identify four core dimensions:

- Task: What is the objective? (*e.g., Binary classification*)
- Experience: What is the feedback? (*e.g., Supervised learning*)
- Model: What is the search space? (*e.g., Linear classifiers*)
- Performance: How do we evaluate success? (*e.g., Zero-one loss*)

The Machine Learning Universe



Outline

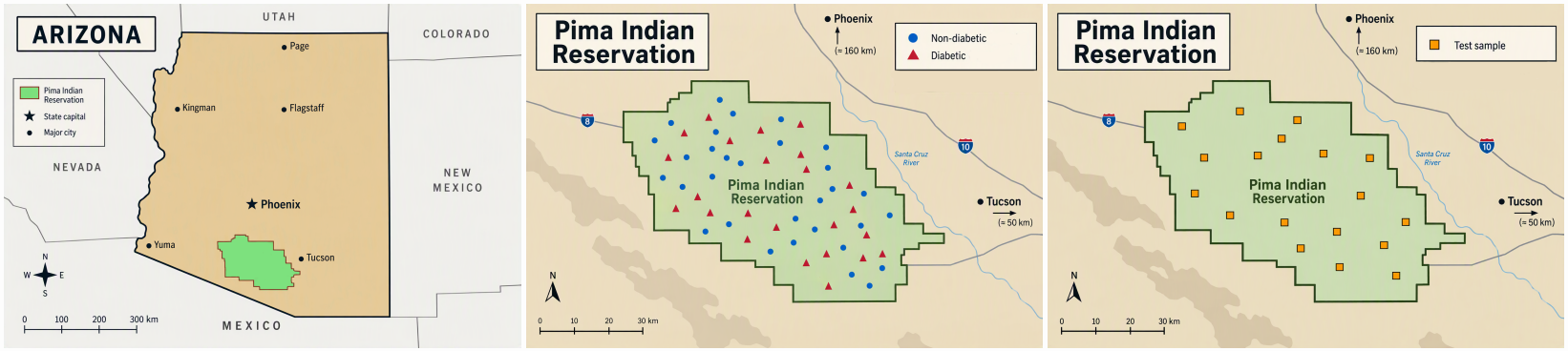
1 A Gentle Start

2 Statistical Learning
- The Framework
- Learnability
- Uniform Convergence
- Sample Complexity
- Computational Complexity

3 Deep Learning

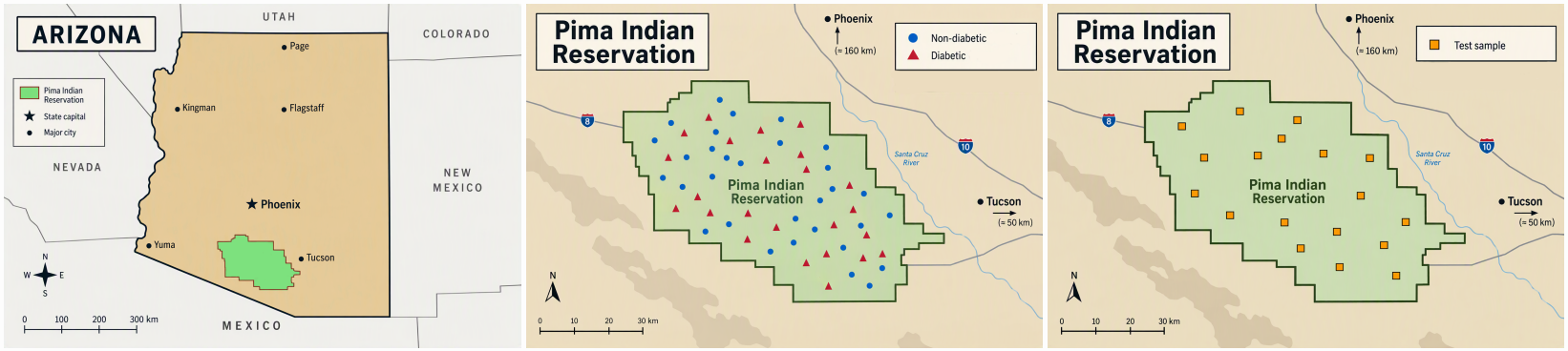
4 Perspectives

A Running Example: Pima Indians Diabetes



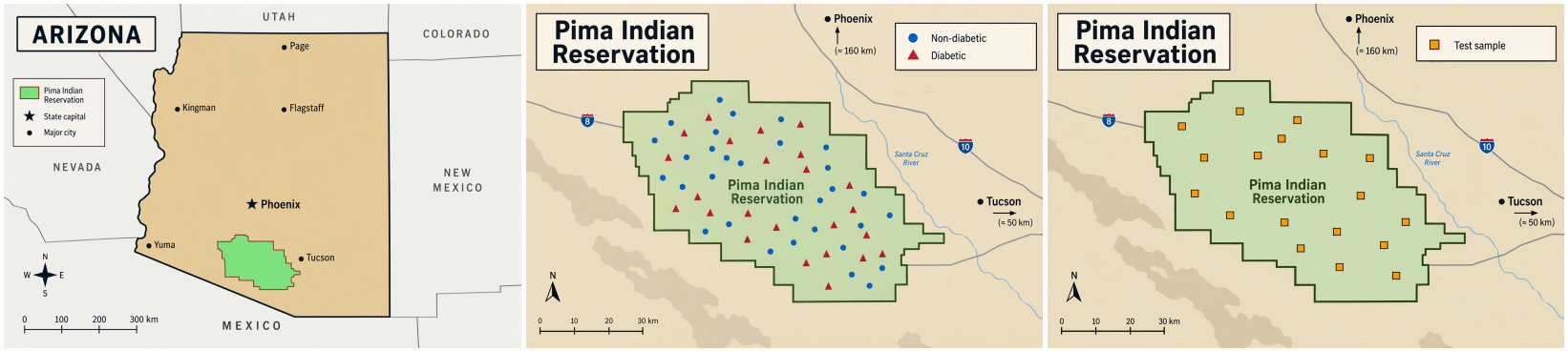
The environment sets a fixed but unknown probability distribution $\mathcal{D}$ over patients (features and diagnosis).

A Running Example: Pima Indians Diabetes



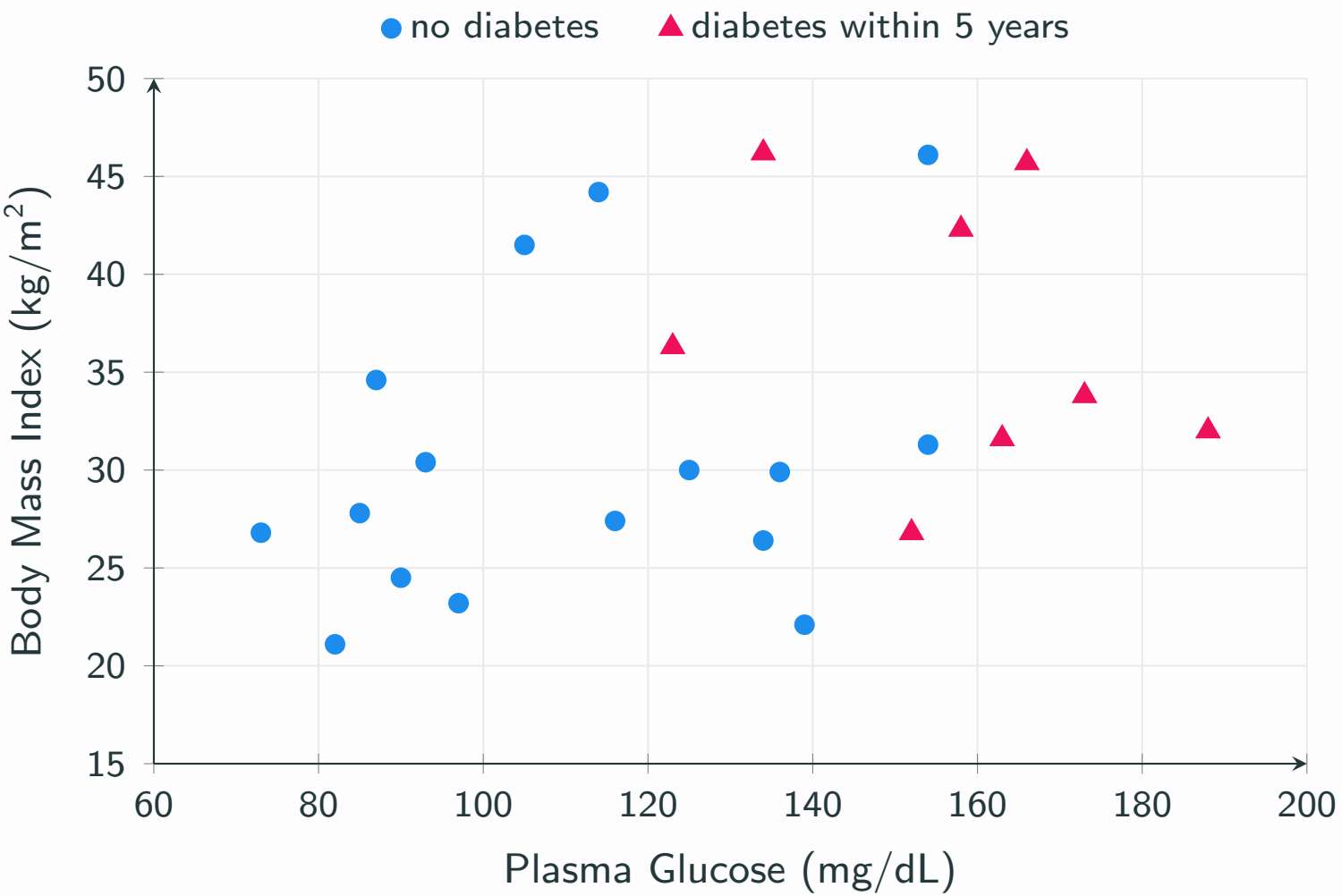
During the training phase, we collect a set S sampled independently and identically (i.i.d.) from $\mathcal{D}$, from which we learn a hypothesis or model h.

A Running Example: Pima Indians Diabetes



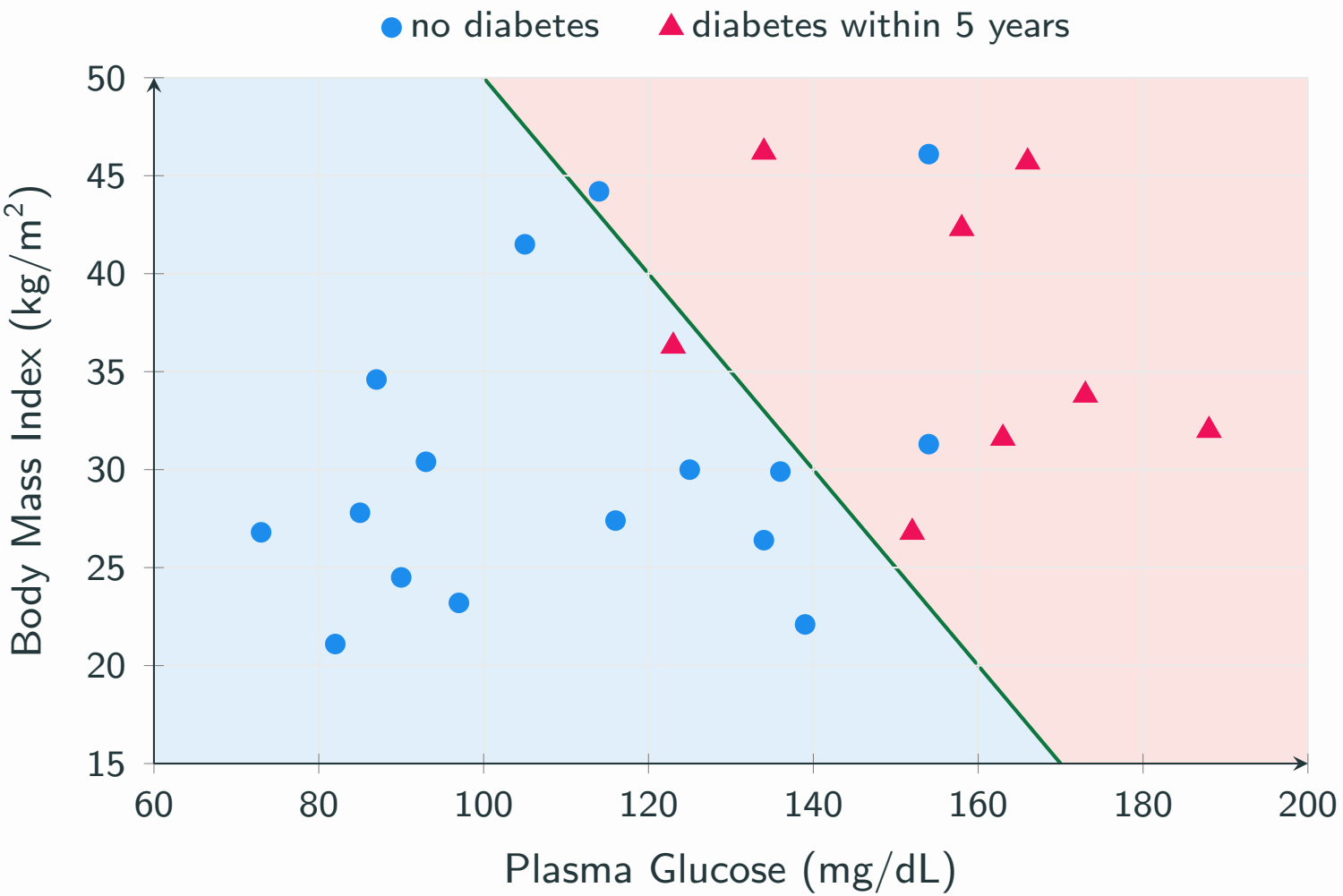
During the test phase, we check whether h accurately classifies *new*, *unseen* patients drawn from the same distribution $\mathcal{D}$.

A Running Example: Pima Indians Diabetes



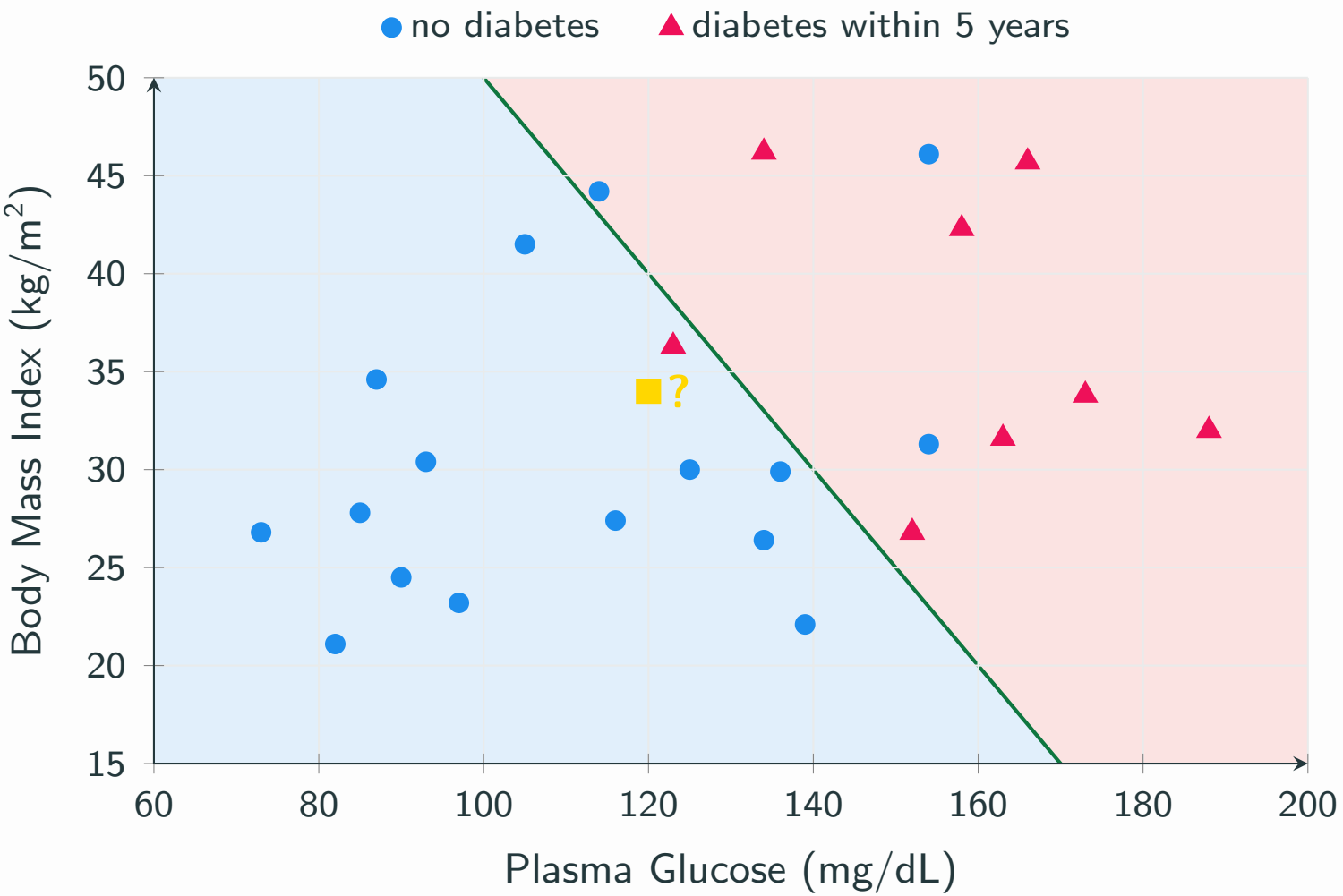
Concretely, let S be a set of 24 patients.

A Running Example: Pima Indians Diabetes



Let h be the linear hypothesis $h(\boldsymbol{x}) = \operatorname{sgn}(x_{\mathrm{PG}} + 2\,x_{\mathrm{BMI}} - 200)$.

A Running Example: Pima Indians Diabetes



The hypothesis h predicts that this new patient will not develop diabetes within 5 years.

Formal Setting

A statistical learning problem is a triplet $(\mathcal{Z}, \mathcal{H}, \ell)$, such that:

- $\mathcal{Z}$ is a pair $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}$ is the *instance space* and $\mathcal{Y}$ is the *label space*. $\longleftarrow$ Task
- $\mathcal{H}$ is a set of candidate functions h from $\mathcal{X}$ to $\mathcal{Y}$. $\longleftarrow$ Models
- ℓ is a loss function from $\mathcal{Y} \times \mathcal{Y}$ to $\mathbb{R}_+$. $\longleftarrow$ Performance

Formal Setting

A statistical learning problem is a triplet $(\mathcal{Z}, \mathcal{H}, \ell)$, such that:

- $\mathcal{Z}$ is a pair $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}$ is the *instance space* and $\mathcal{Y}$ is the *label space*. $\longleftarrow$ Task
- $\mathcal{H}$ is a set of candidate functions h from $\mathcal{X}$ to $\mathcal{Y}$. $\longleftarrow$ Models
- ℓ is a loss function from $\mathcal{Y} \times \mathcal{Y}$ to $\mathbb{R}_+$. $\longleftarrow$ Performance

A statistical learning algorithm is a mapping A that:

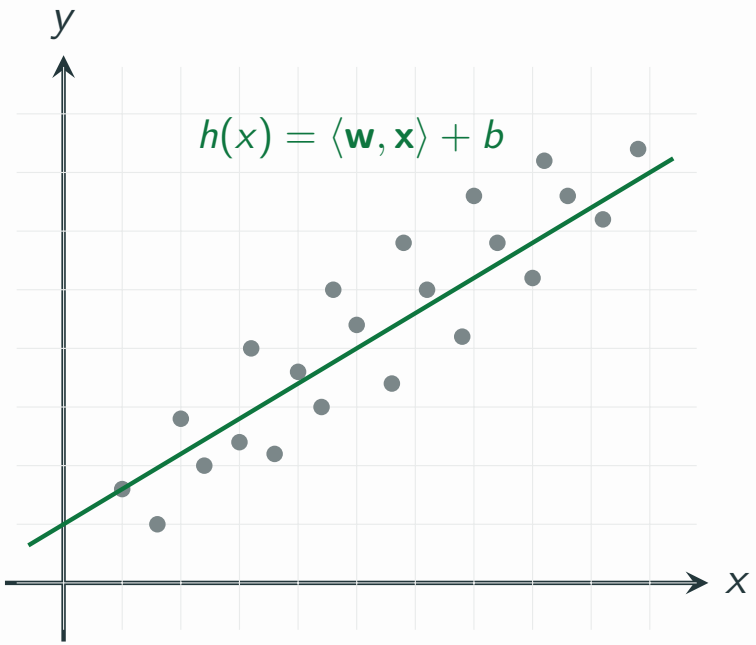
- Takes as input a finite set S of examples drawn independently and identically (i.i.d.) from a fixed, but *unknown*, distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. $\longleftarrow$ Experience
- Returns as output a hypothesis $A(S)$ in $\mathcal{H}$. $\longleftarrow$ Solution

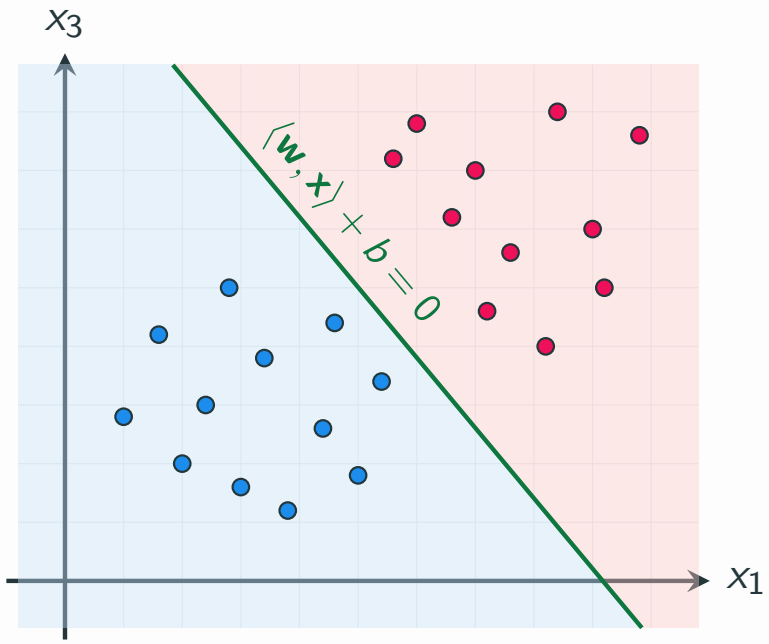
Tasks

Task Name	Instance Space $\mathcal{X}$	Label Space $\mathcal{Y}$
Boolean Classification	$\{0,1\}^d$	$\{0,1\}$
Binary Classification	$\mathbb{R}^d$	$\{-1,+1\}$
Multilabel Classification	$\mathbb{R}^d$	$\{-1,+1\}^k$
Label Ranking	$\mathbb{R}^d$	$\mathfrak{S}_k$
Regression	$\mathbb{R}^d$	$\mathbb{R}$
Ordinal Regression	$\mathbb{R}^d$	$\{1,2,\ldots,k\}$
Probabilistic Regression	$\mathbb{R}^d$	Δ^k

$\mathfrak{S}_k$ denotes the symmetric group containing all permutations of k elements.
Δ^k denotes the k-dimensional simplex.

Hypothesis Classes: Linear Models





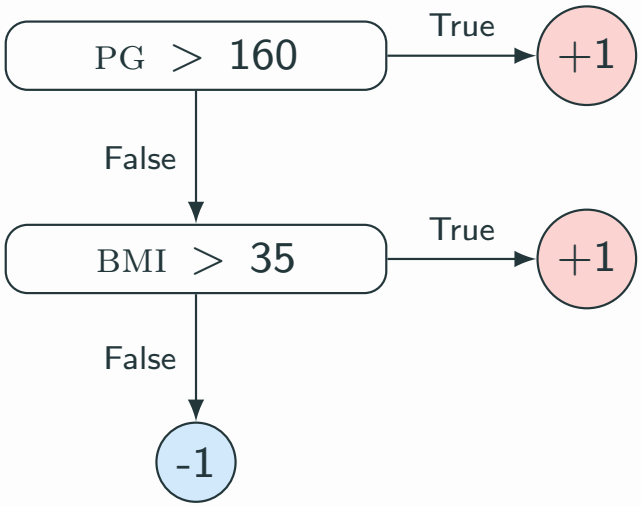
- Linear Regressors (left):

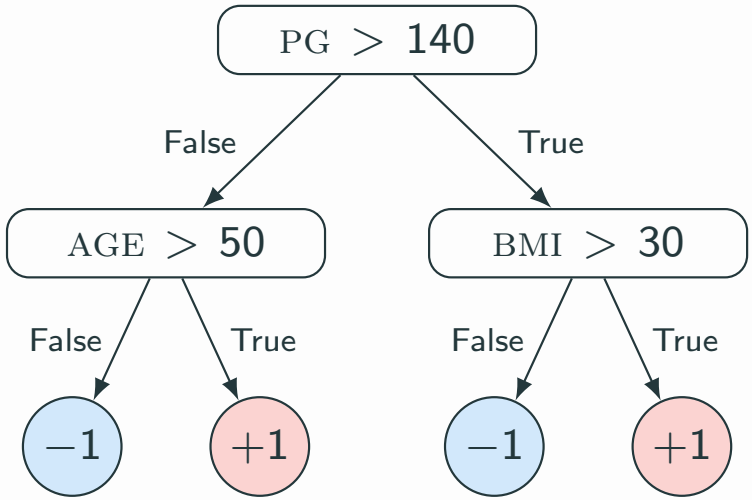
$$\mathcal{H}_{\text{lin}} = \left\{ h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \;\middle|\; \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \right\}$$

- Linear Classifiers (right):

$$\mathcal{H}_{\text{half}} = \left\{ h_{\mathbf{w},b}(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \;\middle|\; \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \right\}$$

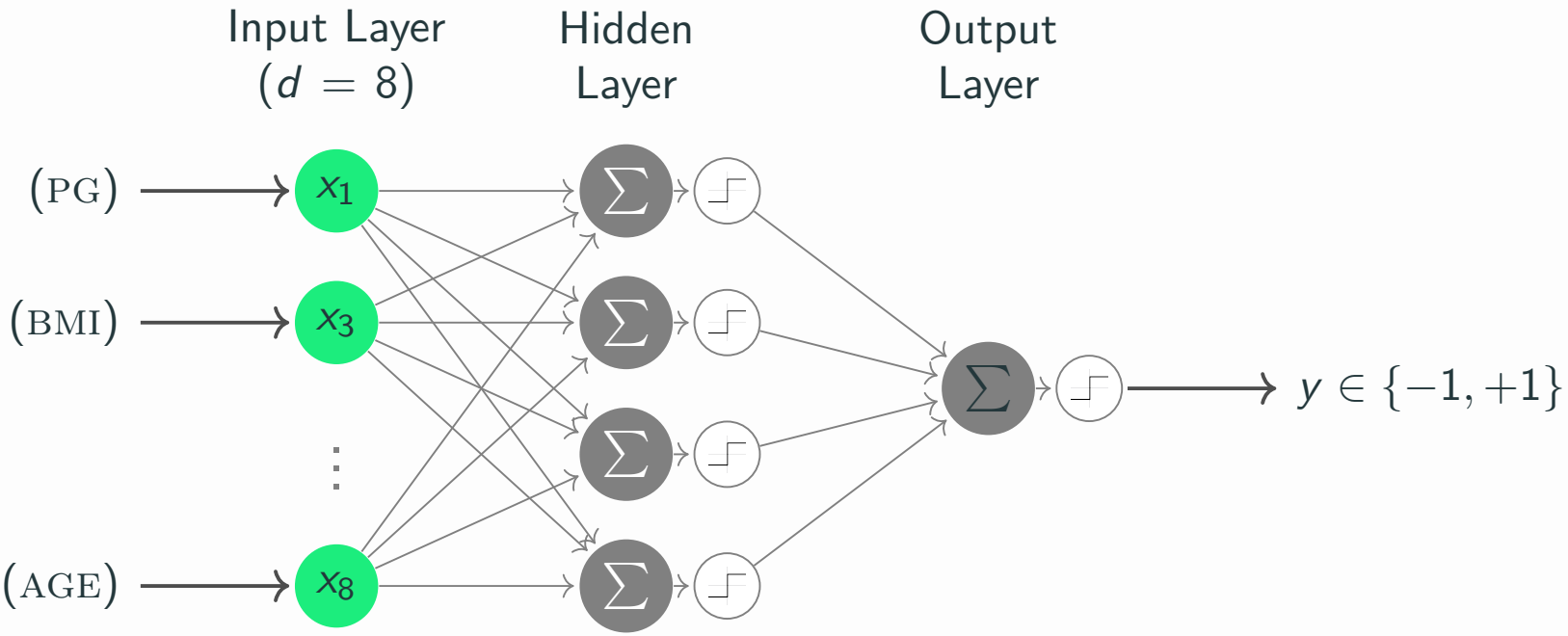
Hypothesis Classes: Logical and Tree Models





- Decision lists (left): hypotheses in $\mathcal{H}_{\mathrm{DL}}$ are represented by *if-then-else* rules over the features. The model outputs the label of the first condition that evaluates to True.
- Decision trees (right): hypotheses in $\mathcal{H}_{\mathrm{DT}}$ partition the instance space into mutually exclusive regions via hierarchical tests, assigning a label (leaf) to each region.

Hypothesis Classes: Neural Networks



Hypotheses in $\mathcal{H}_{\text{NN}}$ are represented using a directed graph of neurons, structured into layers. Each unit computes a linear combination of its inputs followed by a non-linear activation function.

Loss Functions

Name	**Typical Task**	**Definition** $\ell(\hat{y}, y)$
Zero-One Loss	Boolean Classification	$\mathbb{1}[\hat{y} \neq y]$
Hinge Loss	Binary Classification	$\max(0, 1 - y\hat{y})$
Hamming Loss	Multilabel Classification	$\sum_{i=1}^{k} \mathbb{1}[\hat{y}_i \neq y_i]$
Spearman Loss	Label Ranking	$\sum_{i=1}^{k} \lvert\hat{y}_i - y_i\rvert$
Squared Loss	Regression	$(\hat{y} - y)^2$
Absolute Loss	Ordinal Regression	$\lvert\hat{y} - y\rvert$
Cross-Entropy	Probabilistic Regression	$-\sum_{i=1}^{k} y_i \log(\hat{y}_i)$

For the zero-one loss, $\mathbb{1}[E]$ returns 1 if the event E is true, and 0 otherwise.
For the hinge loss, $\hat{y} \in \mathbb{R}$ is the real-valued score *before* thresholding.

Outline

1 A Gentle Start

2 Statistical Learning
- The Framework
- Learnability
- Uniform Convergence
- Sample Complexity
- Computational Complexity

3 Deep Learning

4 Perspectives

True Risk vs. Empirical Risk

For a learning problem $(\mathcal{Z}, \mathcal{H}, \ell)$,

- Let $\mathcal{D}$ be a fixed but *unknown* distribution over $\mathcal{Z}$, and
- Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be a training set drawn i.i.d. from $\mathcal{D}$.

True Risk vs. Empirical Risk

For a learning problem $(\mathcal{Z}, \mathcal{H}, \ell)$,

- Let $\mathcal{D}$ be a fixed but *unknown* distribution over $\mathcal{Z}$, and
- Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be a training set drawn i.i.d. from $\mathcal{D}$.

A True Risk Minimizer is any hypothesis $h \in \mathcal{H}$ that minimizes the expected loss:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h(\mathbf{x}), y)] \quad \longleftarrow \text{What we want}$$

True Risk vs. Empirical Risk

For a learning problem $(\mathcal{Z}, \mathcal{H}, \ell)$,

- Let $\mathcal{D}$ be a fixed but *unknown* distribution over $\mathcal{Z}$, and
- Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be a training set drawn i.i.d. from $\mathcal{D}$.

A True Risk Minimizer is any hypothesis $h \in \mathcal{H}$ that minimizes the expected loss:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h(\mathbf{x}), y)] \quad \longleftarrow \text{What we want}$$

An Empirical Risk Minimizer (ERM) is any hypothesis $h \in \mathcal{H}$ that minimizes the observable loss:

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i) \quad \longleftarrow \text{What we have}$$

Empirical Risk Minimization

Is the Empirical Risk Minimizer a good estimate of the True Risk Minimizer?

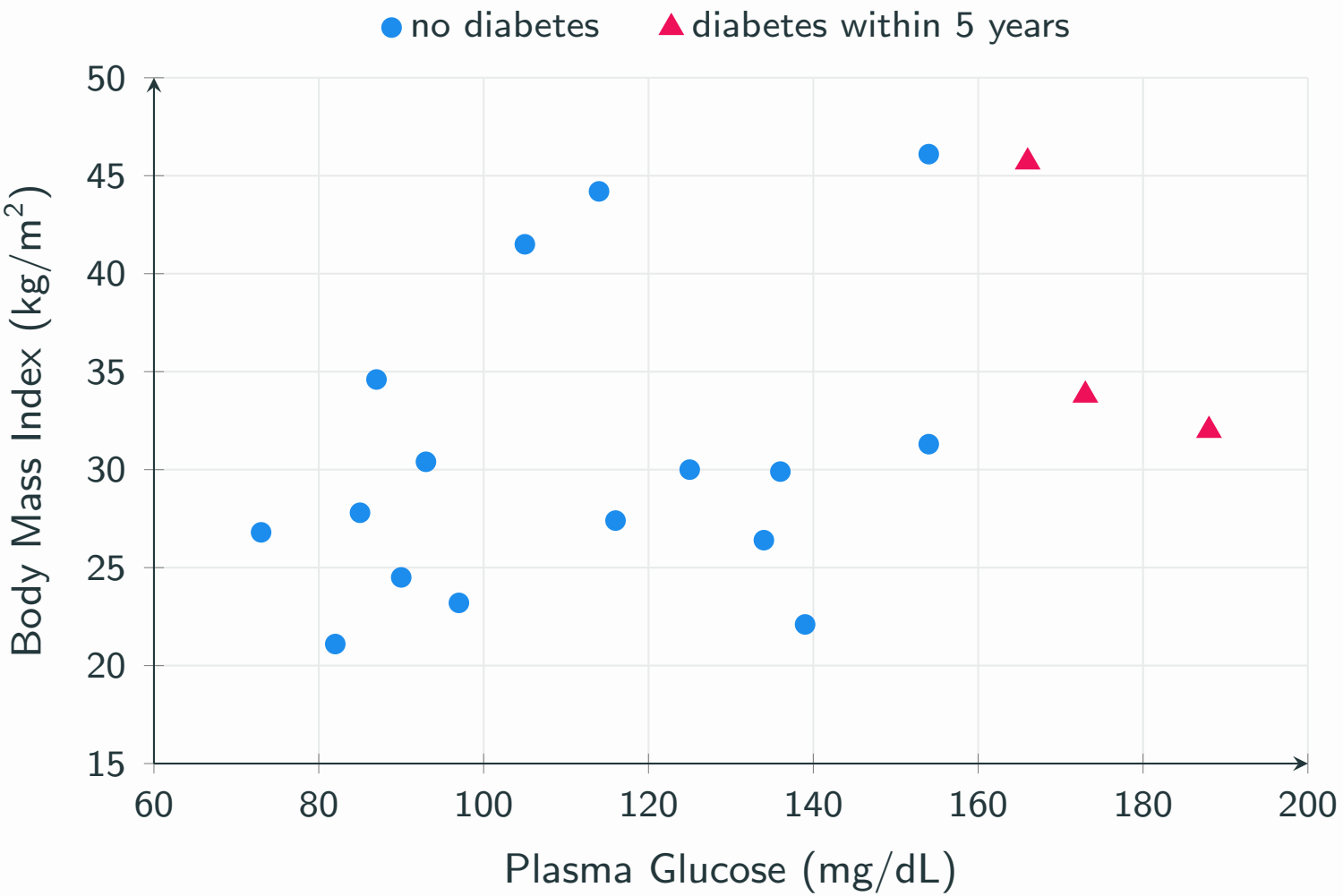
Empirical Risk Minimization

Is the Empirical Risk Minimizer a good estimate of the True Risk Minimizer?

Since S is the only experience we have, there are two sources of uncertainty:

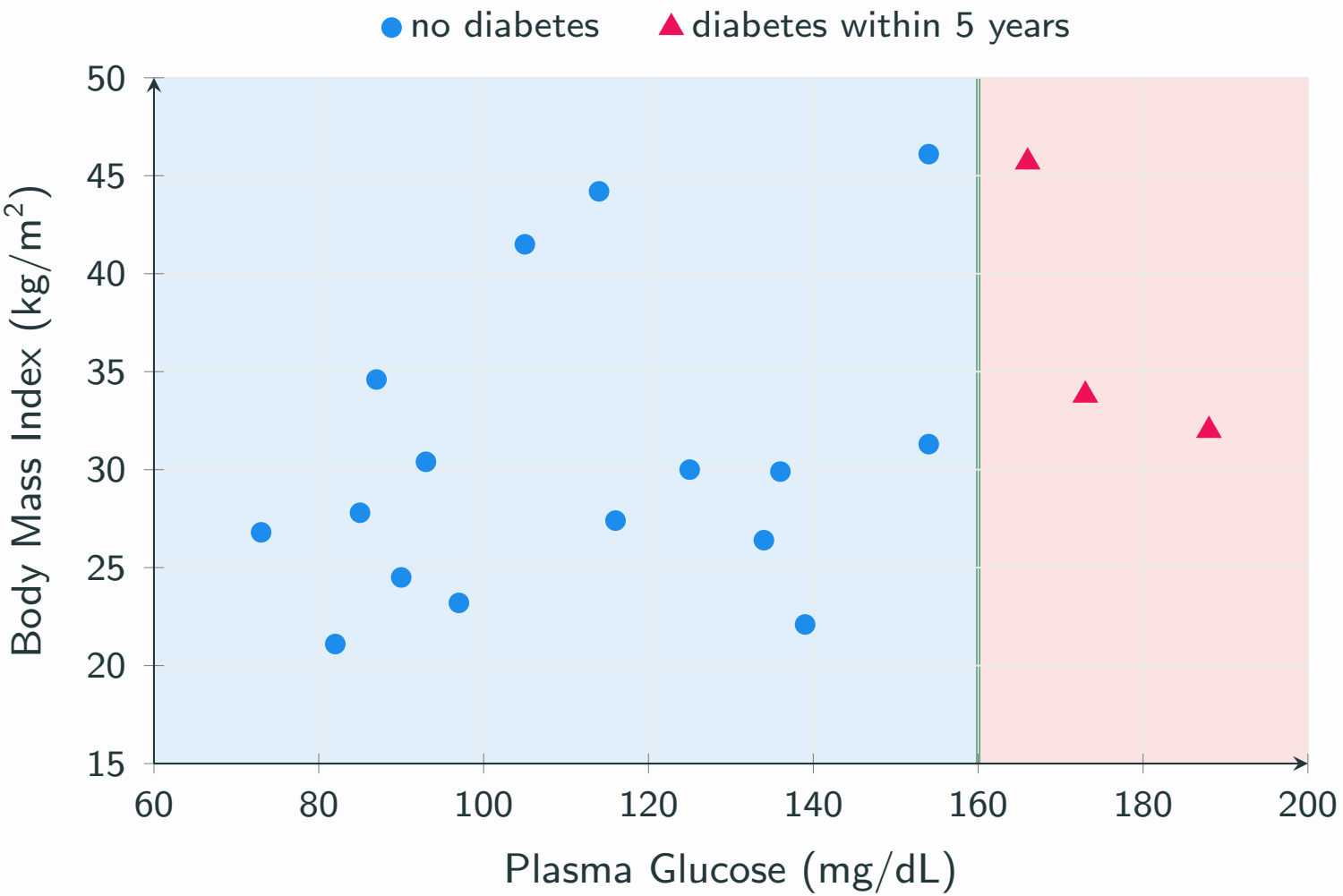
- Confidence: Does S faithfully reflect the underlying distribution $\mathcal{D}$?
- Accuracy: Does S leave only a few hypotheses in $\mathcal{H}$ indistinguishable?

The Confidence Issue



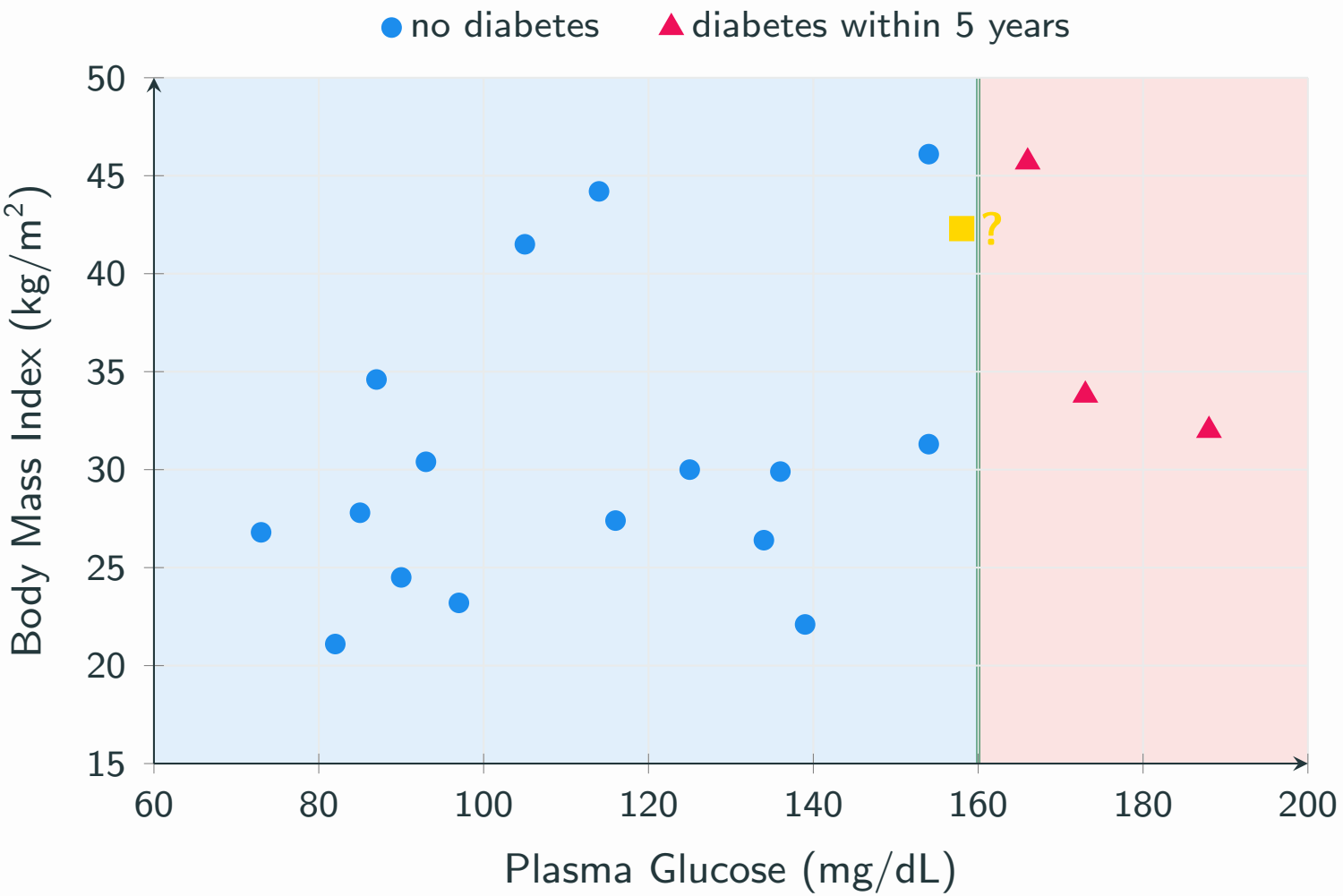
By sheer bad luck, the few diabetic patients drawn in S all happen to have extreme glucose levels.

The Confidence Issue



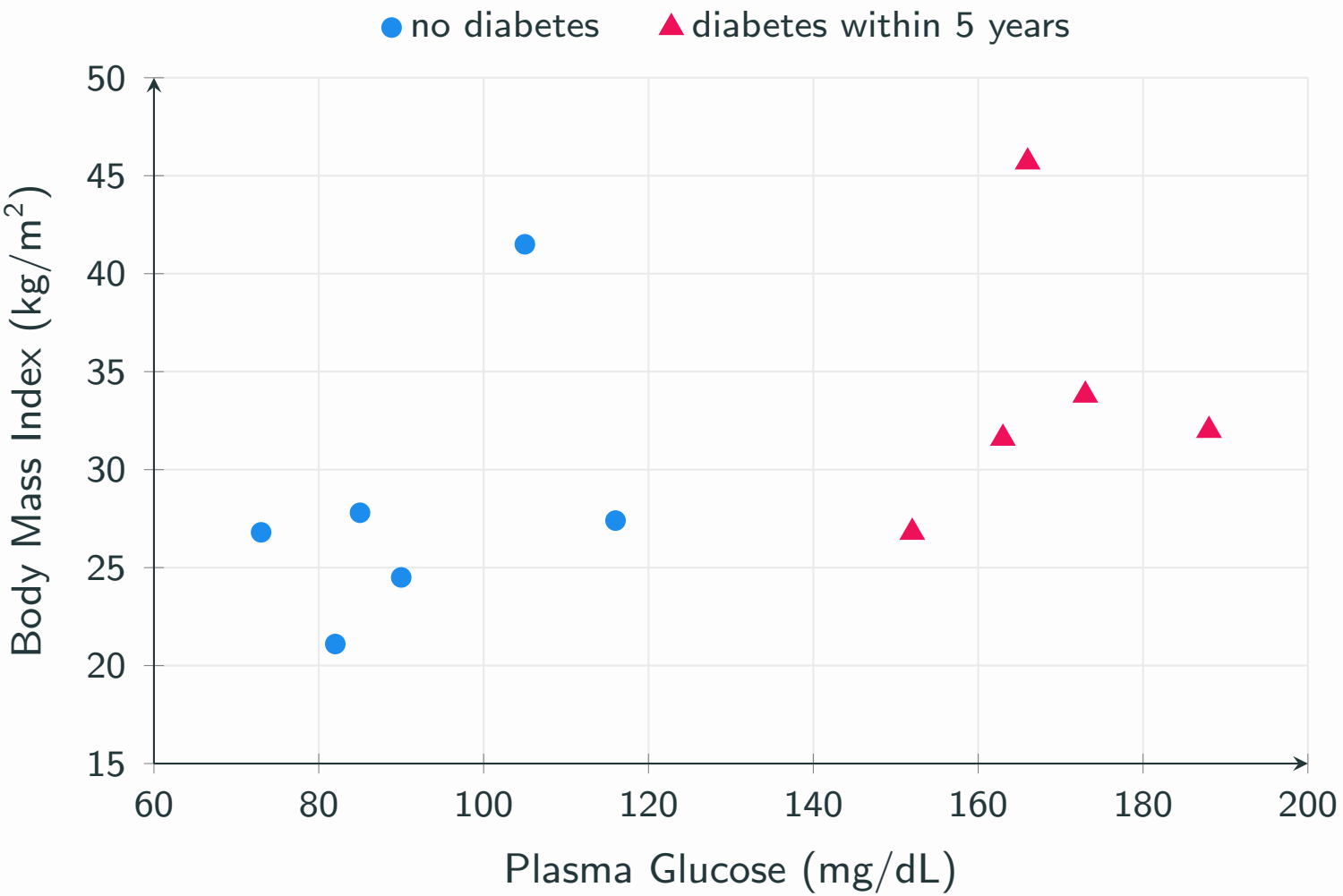
The ERM principle minimizes error on S by generating a dangerously simplistic hypothesis h: $x_{\text{PG}} > 160$.

The Confidence Issue



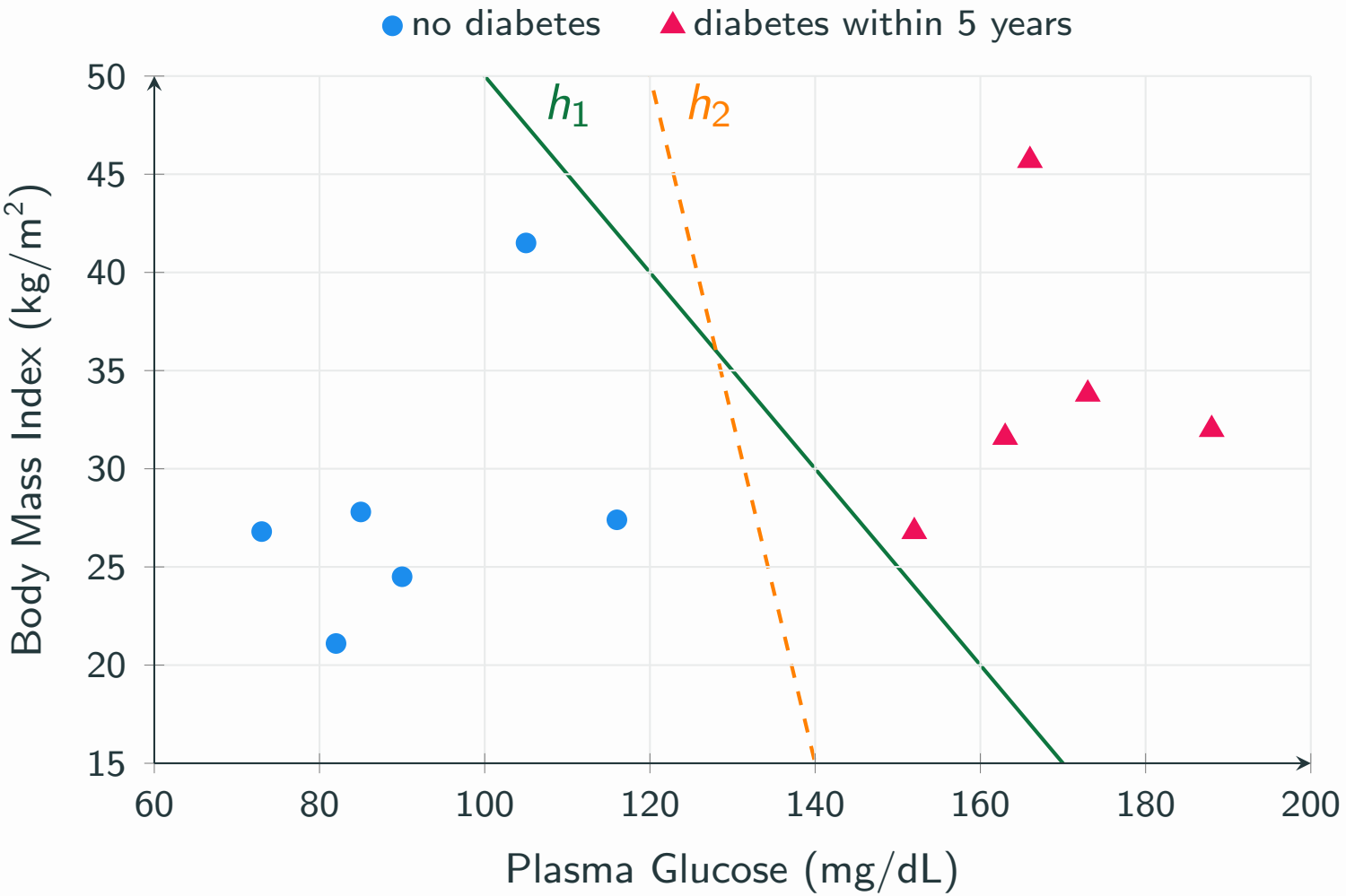
A new patient arrives. Because of the skewed training data, h incorrectly predicts they are safe.

The Accuracy Issue



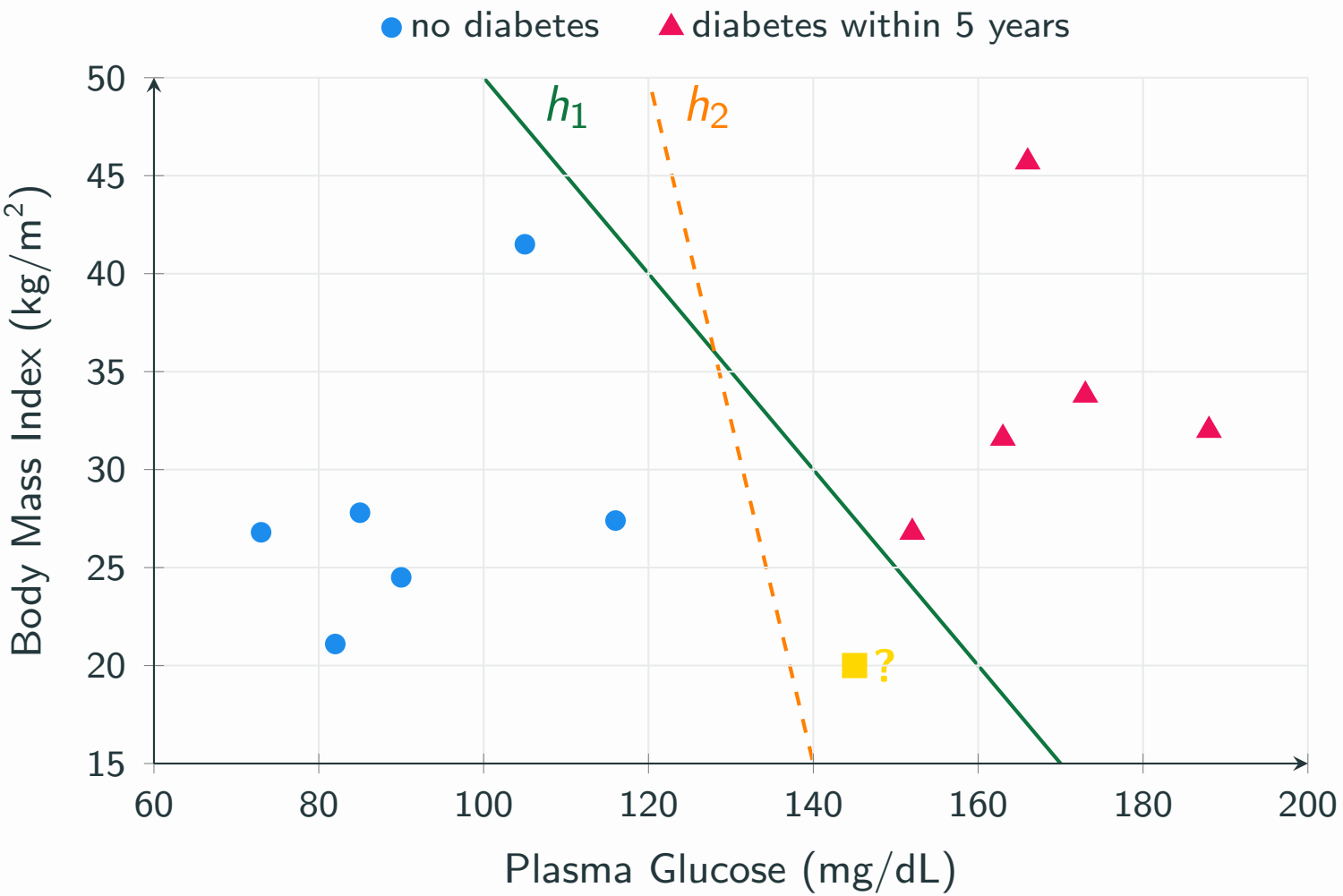
Suppose our sample S reflects both profiles correctly, but it lacks *density*.

The Accuracy Issue



Because S is sparse, multiple hypotheses (like h_1 and h_2) can perfectly separate the training data, both achieving $L_S(h) = 0$.

The Accuracy Issue



For a new patient in the empty space, h_1 predicts "Safe" while h_2 predicts "Diabetes". We need a larger sample size m to shrink this ambiguity gap.

Agnostic PAC Learning

A class $\mathcal{H}$ is learnable if there is an algorithm A and a function $m_{\mathcal{H}}$ such that, for every $\varepsilon, \delta \in (0,1)$ and every distribution $\mathcal{D}$, if S consists of $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ examples drawn i.i.d. from $\mathcal{D}$, then

$$L_{\mathcal{D}}(A(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon \quad \text{with probability at least } 1 - \delta$$

Learnability

Agnostic PAC Learning

A class $\mathcal{H}$ is learnable if there is an algorithm A and a function $m_{\mathcal{H}}$ such that, for every $\varepsilon, \delta \in (0,1)$ and every distribution $\mathcal{D}$, if S consists of $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ examples drawn i.i.d. from $\mathcal{D}$, then

$$L_{\mathcal{D}}(A(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon \quad \text{with probability at least } 1 - \delta$$

- Probably: with confidence $1 - \delta$
- Approximately: up to accuracy ε
- Correct: compared to the best hypothesis in $\mathcal{H}$

Agnostic PAC Learning

A class $\mathcal{H}$ is learnable if there is an algorithm A and a function $m_{\mathcal{H}}$ such that, for every $\varepsilon, \delta \in (0,1)$ and every distribution $\mathcal{D}$, if S consists of $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ examples drawn i.i.d. from $\mathcal{D}$, then

$$L_{\mathcal{D}}(A(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon \quad \text{with probability at least } 1 - \delta$$

- Probably: with confidence $1 - \delta$
- Approximately: up to accuracy ε
- Correct: compared to the best hypothesis in $\mathcal{H}$

Agnostic: no hypothesis in $\mathcal{H}$ needs to be perfect (zero true risk).

Sample complexity: the smallest such function $m_{\mathcal{H}}(\varepsilon, \delta)$.

Outline

1 A Gentle Start

2 Statistical Learning
- The Framework
- Learnability
- Uniform Convergence
- Sample Complexity
- Computational Complexity

3 Deep Learning

4 Perspectives

Uniform Convergence

A training set S is ε-representative for $\mathcal{H}$ if $|L_S(h) - L_{\mathcal{D}}(h)| \leq \varepsilon$ for all $h \in \mathcal{H}$.

Uniform Convergence

A class $\mathcal{H}$ has the uniform convergence property if there is a function $m_{\mathcal{H}}^{\mathrm{UC}}$ such that, for every $\varepsilon, \delta \in (0,1)$ and every distribution $\mathcal{D}$, if S consists of $m \geq m_{\mathcal{H}}^{\mathrm{UC}}(\varepsilon, \delta)$ examples drawn i.i.d. from $\mathcal{D}$, then S is ε-representative with probability at least $1 - \delta$.

Uniform Convergence

A training set S is ε-representative for $\mathcal{H}$ if $|L_S(h) - L_{\mathcal{D}}(h)| \leq \varepsilon$ for all $h \in \mathcal{H}$.

Uniform Convergence

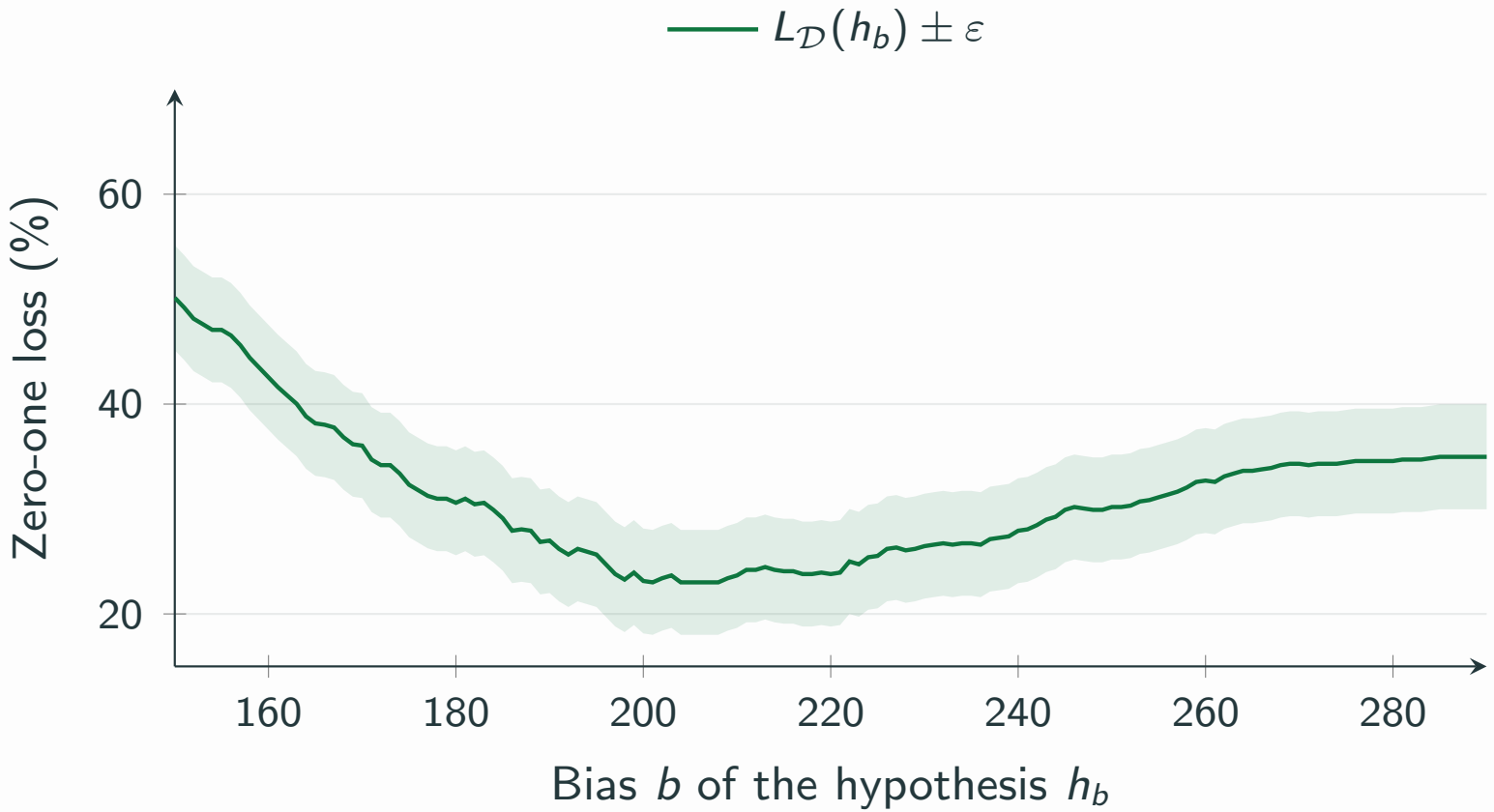
A class $\mathcal{H}$ has the uniform convergence property if there is a function $m_{\mathcal{H}}^{\mathrm{UC}}$ such that, for every $\varepsilon, \delta \in (0,1)$ and every distribution $\mathcal{D}$, if S consists of $m \geq m_{\mathcal{H}}^{\mathrm{UC}}(\varepsilon, \delta)$ examples drawn i.i.d. from $\mathcal{D}$, then S is ε-representative with probability at least $1 - \delta$.

From Uniform Convergence to Learnability

If S is $\frac{\varepsilon}{2}$-representative, then any ERM algorithm learns $\mathcal{H}$ with

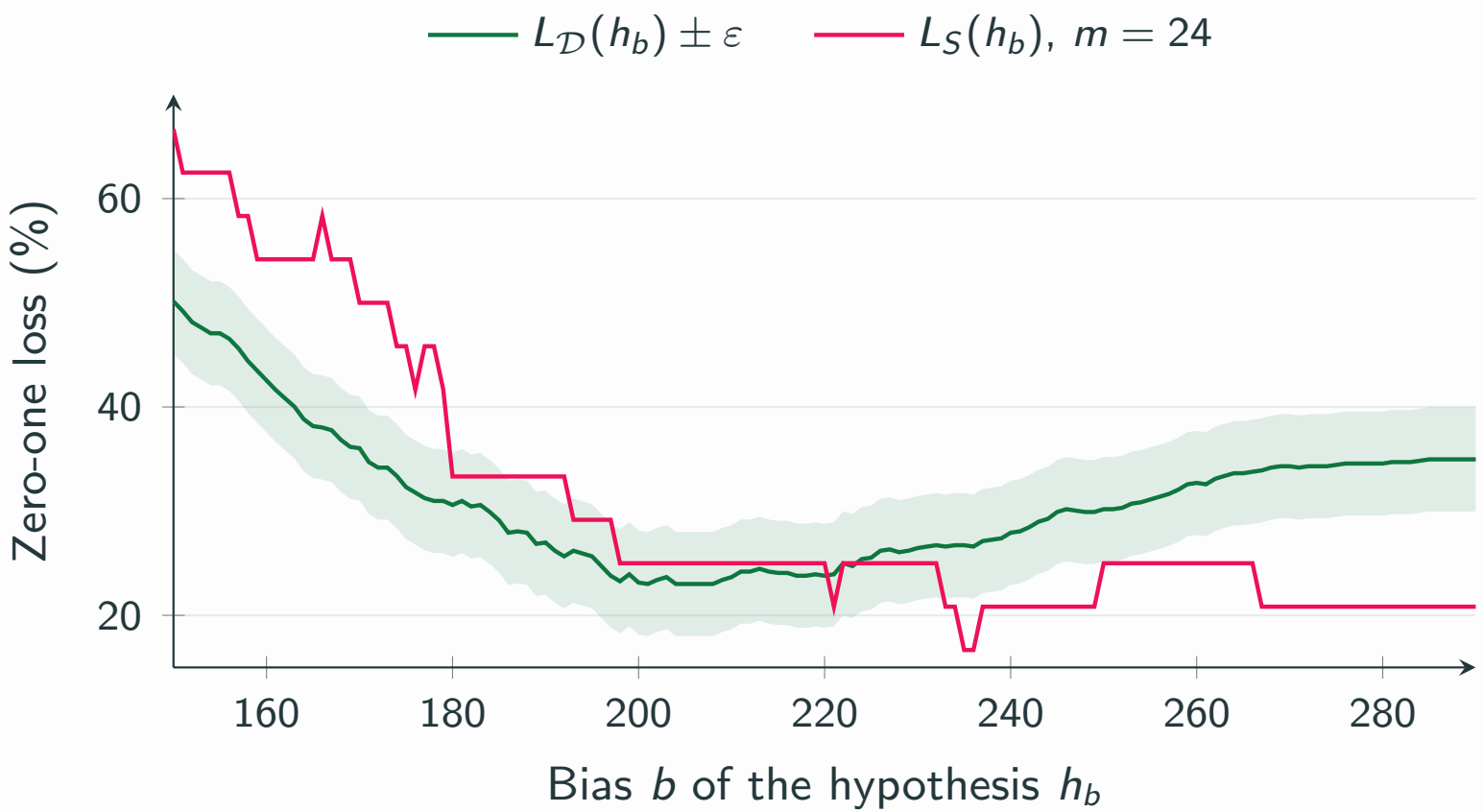
$$m_{\mathcal{H}}(\varepsilon, \delta) \leq m_{\mathcal{H}}^{\mathrm{UC}}(\varepsilon/2, \delta)$$

Uniform Convergence on Pima



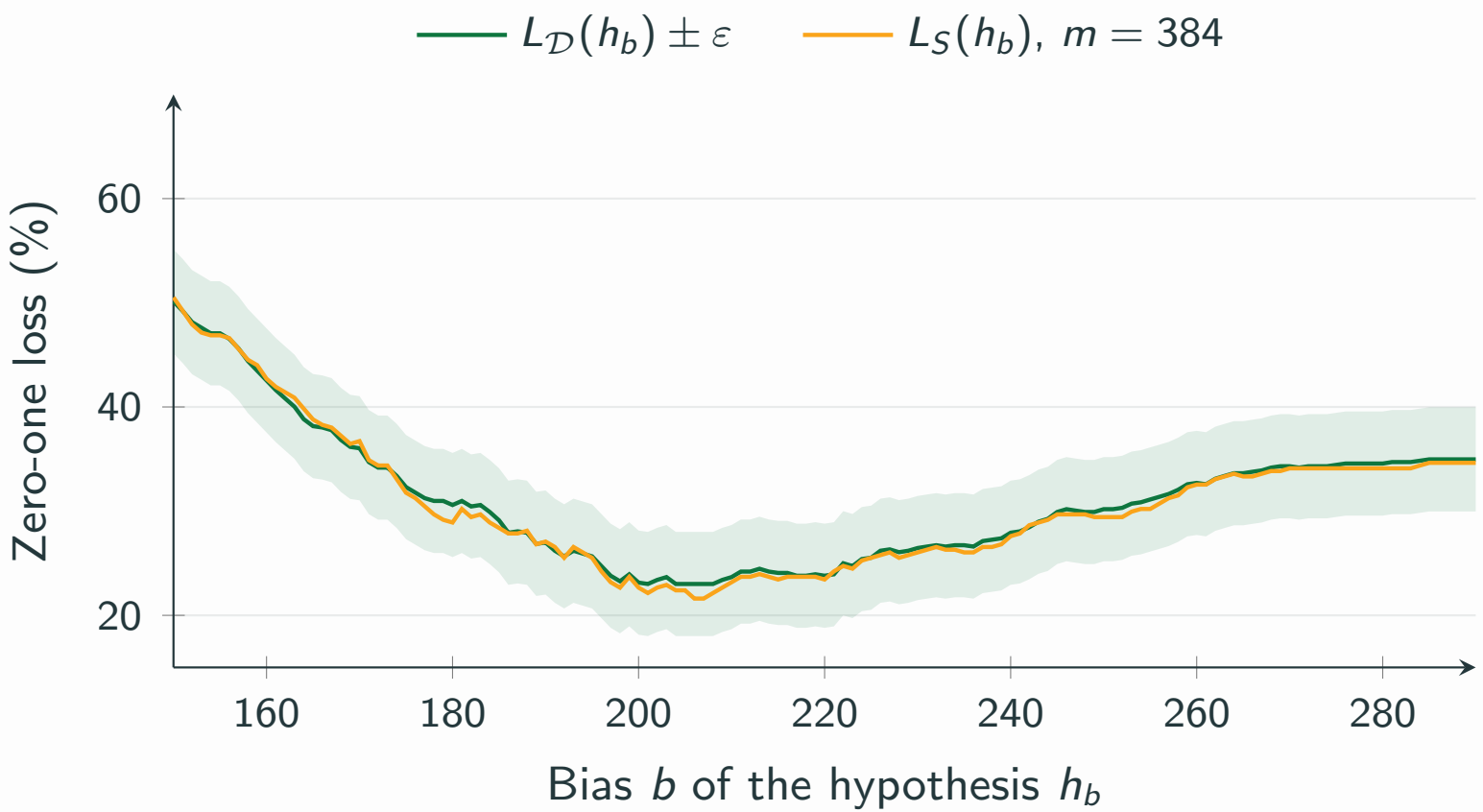
Let $\mathcal{D}$ be the uniform distribution over the 752 Pima patients, and let $\mathcal{H}$ the class of functions $h_b(\boldsymbol{x}) = \operatorname{sgn}(x_{\text{PG}} + 2\,x_{\text{BMI}} - b)$. The best threshold is $b^* \approx 200$, with $L_{\mathcal{D}}(h_{b^*}) \approx 23\%$.

Uniform Convergence on Pima



With $m = 24$ patients and $\varepsilon = 5\%$, L_S leaves the ε-band: S is not representative. ERM picks $b \approx 235$, with a true risk of 27%.

Uniform Convergence on Pima



With $m = 384$ patients, L_S stays inside the band for all b: S is ε-representative, and ERM picks $b \approx 206$, with a true risk of 23%.

Outline

1 A Gentle Start

2 Statistical Learning
- The Framework
- Learnability
- Uniform Convergence
- Sample Complexity
- Computational Complexity

3 Deep Learning

4 Perspectives

Vapnik-Chervonenkis Dimension

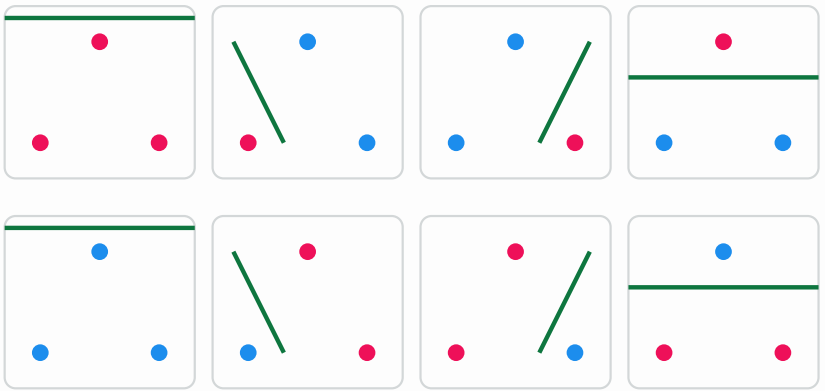
VC Dimension

A finite set $C \subset \mathcal{X}$ is shattered by $\mathcal{H}$ if $\mathcal{H}$ realizes all $2^{|C|}$ labelings of C. The VC dimension $\text{VCdim}(\mathcal{H})$ is the size of the largest set shattered by $\mathcal{H}$.

Vapnik-Chervonenkis Dimension

VC Dimension

A finite set $C \subset \mathcal{X}$ is shattered by $\mathcal{H}$ if $\mathcal{H}$ realizes all $2^{|C|}$ labelings of C. The VC dimension $\mathrm{VCdim}(\mathcal{H})$ is the size of the largest set shattered by $\mathcal{H}$.

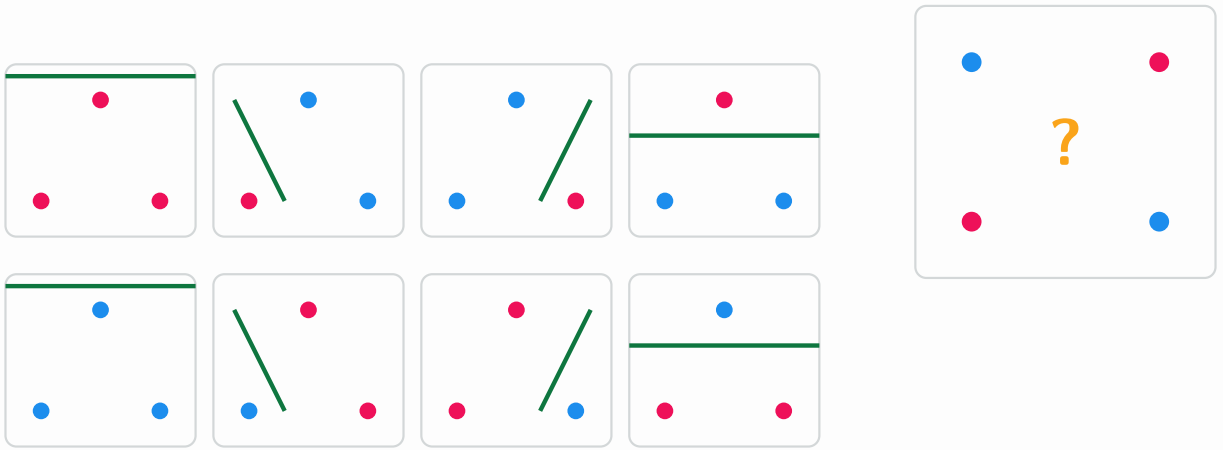


Halfspaces in the plane (*e.g.*, Pima with PG and BMI) shatter these 3 points: each of the $2^3 = 8$ labelings is realized by a line. Hence $\mathrm{VCdim}(\mathcal{H}_{\text{half}}) \geq 3$.

Vapnik-Chervonenkis Dimension

VC Dimension

A finite set $C \subset \mathcal{X}$ is shattered by $\mathcal{H}$ if $\mathcal{H}$ realizes all $2^{|C|}$ labelings of C. The VC dimension $\mathrm{VCdim}(\mathcal{H})$ is the size of the largest set shattered by $\mathcal{H}$.



But no set of 4 points can be shattered: for 4 points in convex position, the "XOR" labeling is not linearly separable (and otherwise, one point lies inside the triangle of the others). Hence $\mathrm{VCdim}(\mathcal{H}_{\text{half}}) = 3$.

VC Dimension of Common Classes

Hypothesis class	**Parameters**	$\mathrm{VCdim}(\mathcal{H})$
Thresholds $\mathbb{1}[x \geq \theta]$ on $\mathbb{R}$	1	1
Intervals $\mathbb{1}[a \leq x \leq b]$ on $\mathbb{R}$	2	2
Axis-aligned rectangles in $\mathbb{R}^2$	4	4
Halfspaces in $\mathbb{R}^d$	$d+1$	$d+1$
Finite classes	—	$\leq \log_2 \lvert\mathcal{H}\rvert$
Sine classifiers $\mathbb{1}[\sin(\theta x) \geq 0]$ on $\mathbb{R}$	1	∞

The VC dimension often matches the number of parameters, but not always: a single parameter can shatter arbitrarily large sets.

Putting Everything Together

Fundamental Theorem of Statistical Learning

For any class $\mathcal{H}$ of binary classifiers and the zero-one loss, the following are equivalent:

$$\mathcal{H} \text{ has the uniform convergence property } \Leftrightarrow \text{ ERM learns } \mathcal{H}$$
$$\Leftrightarrow \mathcal{H} \text{ is agnostic PAC learnable } \Leftrightarrow \text{VCdim}(\mathcal{H}) < \infty$$

Moreover, the sample complexity is

$$m_{\mathcal{H}}(\varepsilon, \delta) = \Theta\left(\frac{\text{VCdim}(\mathcal{H}) + \log(1/\delta)}{\varepsilon^2}\right).$$

Putting Everything Together

Fundamental Theorem of Statistical Learning

For any class $\mathcal{H}$ of binary classifiers and the zero-one loss, the following are equivalent:

$$\mathcal{H} \text{ has the uniform convergence property } \Leftrightarrow \text{ ERM learns } \mathcal{H}$$
$$\Leftrightarrow \mathcal{H} \text{ is agnostic PAC learnable } \Leftrightarrow \text{VCdim}(\mathcal{H}) < \infty$$

Moreover, the sample complexity is

$$m_{\mathcal{H}}(\varepsilon, \delta) = \Theta\left(\frac{\text{VCdim}(\mathcal{H}) + \log(1/\delta)}{\varepsilon^2}\right).$$

- Upper bound: with that many examples, *any* ERM succeeds, for *every* distribution.

Putting Everything Together

Fundamental Theorem of Statistical Learning

For any class $\mathcal{H}$ of binary classifiers and the zero-one loss, the following are equivalent:

$$\mathcal{H} \text{ has the uniform convergence property } \Leftrightarrow \text{ ERM learns } \mathcal{H}$$
$$\Leftrightarrow \mathcal{H} \text{ is agnostic PAC learnable } \Leftrightarrow \text{VCdim}(\mathcal{H}) < \infty$$

Moreover, the sample complexity is

$$m_{\mathcal{H}}(\varepsilon, \delta) = \Theta\left(\frac{\text{VCdim}(\mathcal{H}) + \log(1/\delta)}{\varepsilon^2}\right).$$

- Upper bound: with that many examples, *any* ERM succeeds, for *every* distribution.
- Lower bound: with fewer examples, for every learning algorithm, there exists a distribution on which it fails.

Outline

1 A Gentle Start

2 Statistical Learning
- The Framework
- Learnability
- Uniform Convergence
- Sample Complexity
- Computational Complexity

3 Deep Learning

4 Perspectives

Computing Empirical Risk Minimizers

Representational Assumptions

Each hypothesis $h \in \mathcal{H}$ is represented by a finite string of bits, such that:

- The length of the string is polynomial in the dimension d (number of features) of the instance space $\mathcal{X}$.
- Evaluating $h(\mathbf{x})$ for any instance $\mathbf{x} \in \mathcal{X}$ can be done in time polynomial in d.

The ERM Problem

Given: A hypothesis class $\mathcal{H}$, a loss function ℓ, and a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$.

Find: A hypothesis $h \in \mathcal{H}$ that minimizes the empirical risk:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i)$$

Linear Hypotheses Can be Hard to Learn

Classification

Let $\mathcal{H}_{\text{half}}$ be the class of halfspaces.

- If ℓ is the **hinge loss** function, then the ERM problem is solvable in polynomial time. $\longleftarrow$ Convex!
- If ℓ is the **zero-one loss** function, then the ERM problem is generally NP-hard. $\longleftarrow$ Nonconvex!

Linear Hypotheses Can be Hard to Learn

Classification

Let $\mathcal{H}_{\text{half}}$ be the class of halfspaces.

- If ℓ is the **hinge loss** function, then the ERM problem is solvable in polynomial time. $\longleftarrow$ Convex!
- If ℓ is the **zero-one loss** function, then the ERM problem is generally NP-hard. $\longleftarrow$ Nonconvex!

Regression

Let ℓ be the **squared loss** function.

- If $\mathcal{H}_{\text{lin}}$ is the class of dense linear functions, then the ERM problem is solvable in polynomial time. $\longleftarrow$ Convex!
- If $\mathcal{H}_{k-\text{lin}}$ is the class of k-sparse linear functions, then the ERM problem is generally NP-hard. $\longleftarrow$ Nonconvex!

Outline

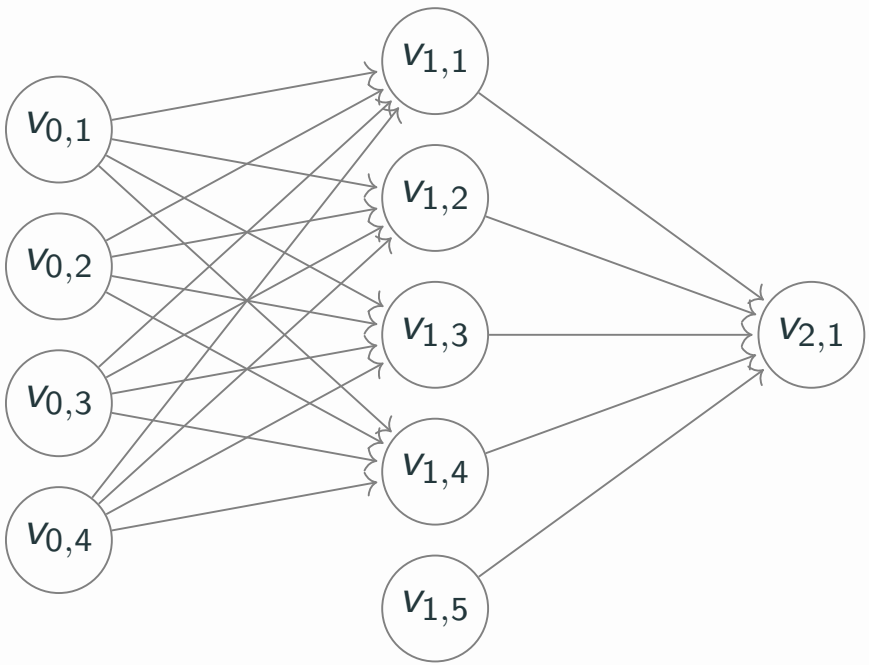
1 A Gentle Start

2 Statistical Learning

3 Deep Learning
- Neural Architectures
- Expressive Power
- Learning via Uniform Convergence
- Learning via Stability
- Stochastic Gradient Descent

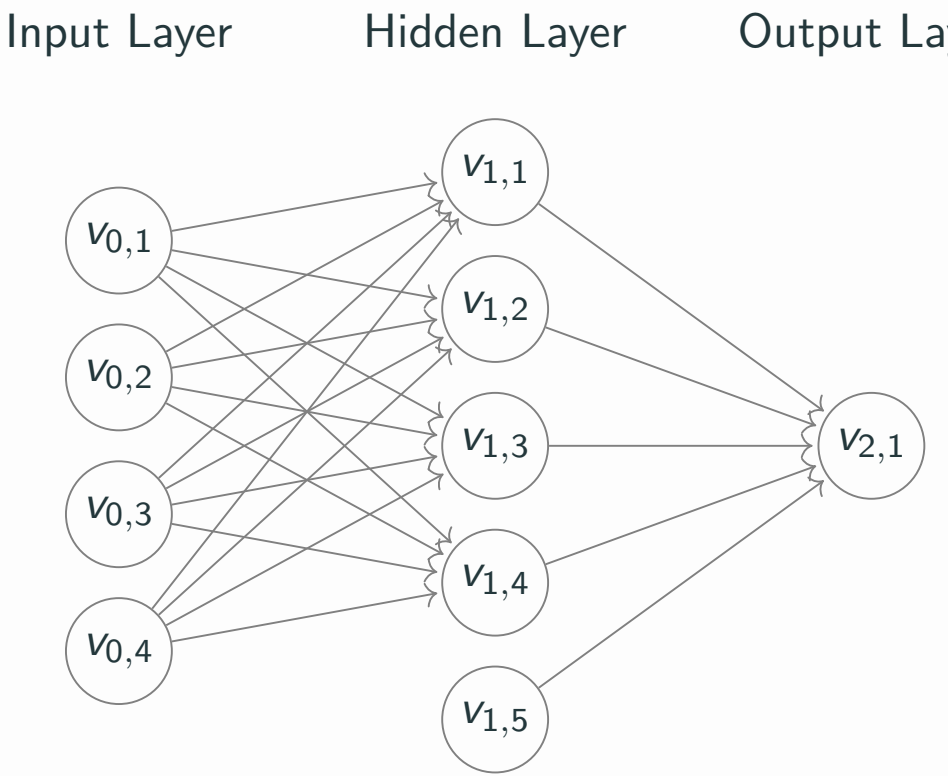
4 Perspectives

Feedforward Neural Networks



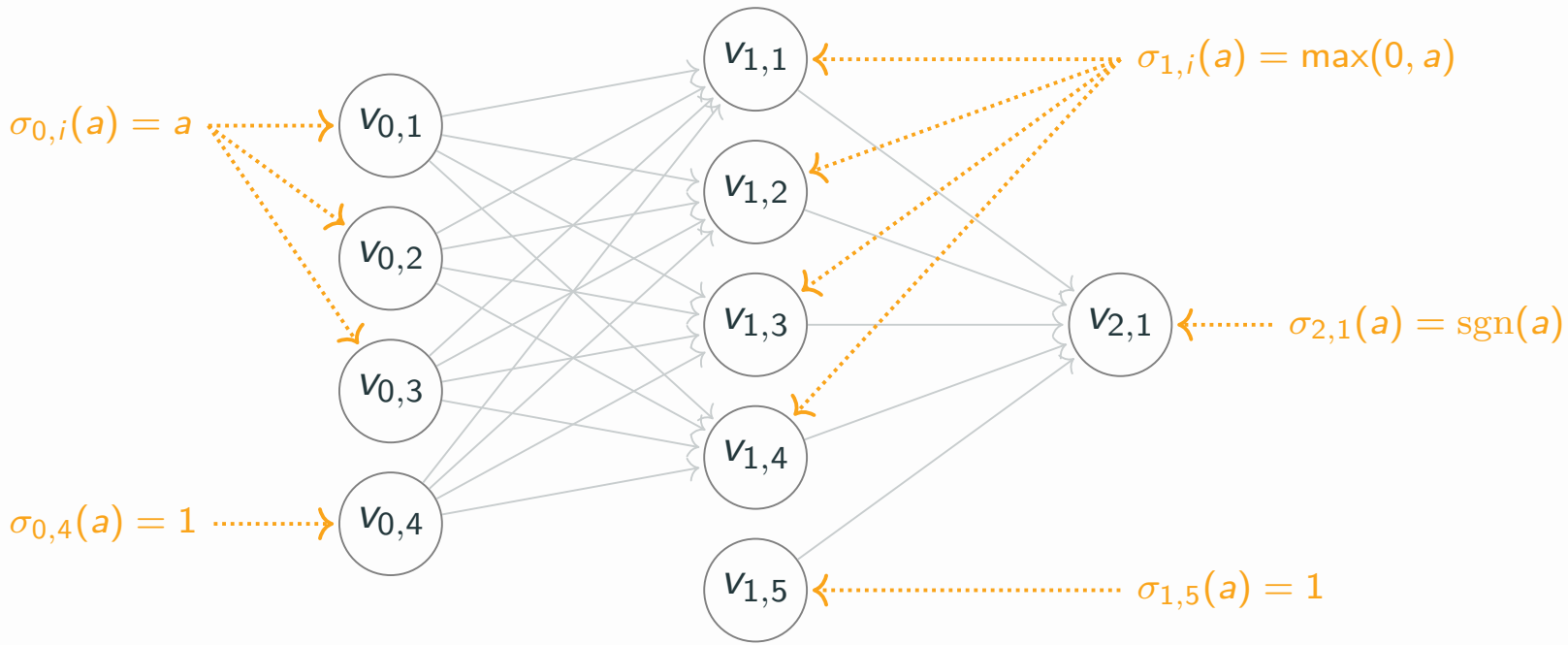
The *structure* of a feedforward neural network is a directed acyclic graph $G = (V, E)$. The vertex set V is partitioned into subsets $V = \bigcup_{l=0}^{L} V_l$, called layers. The edge set E is partitioned into subsets $E = \bigcup_{l=0}^{L-1} E_l$, where $E_l \subseteq V_l \times V_{l+1}$.

Feedforward Neural Networks



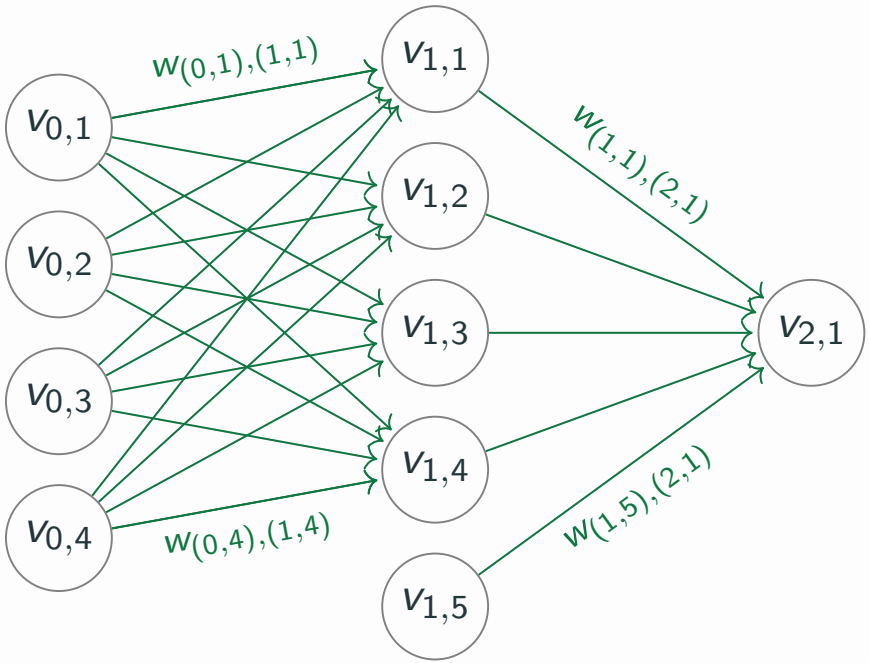
V_0 and V_L are called the *input layer* and the *output layer*, respectively. $V_1, \ldots, V_{L-1}$ are the *hidden layers*. The **depth** of the network is L, its **width** is $\max_i |V_i|$, and its **size** is $|V|$.

Feedforward Neural Networks



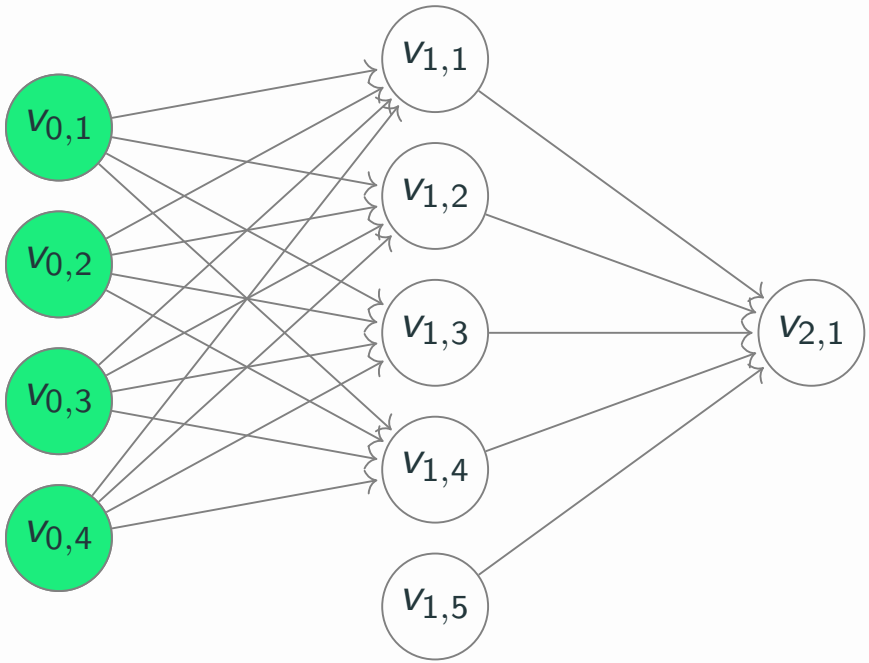
Each vertex v is labeled with an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. Examples are the identity function $\sigma(a) = a$, the ReLU function $\sigma(a) = \max(0, a)$, and the sign function $\sigma(a) = \operatorname{sgn}(a)$. Bias nodes use the constant function $\sigma(a) = 1$.

Feedforward Neural Networks



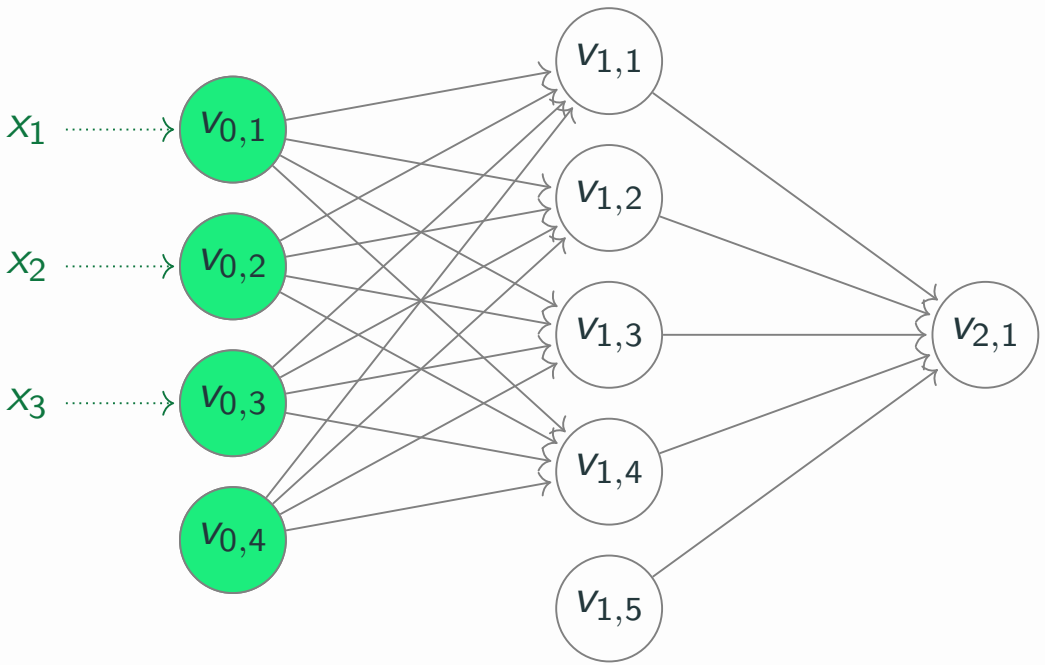
The *parameters* of the network are defined by a weight function $w : E \to \mathbb{R}$.

Feedforward Neural Networks



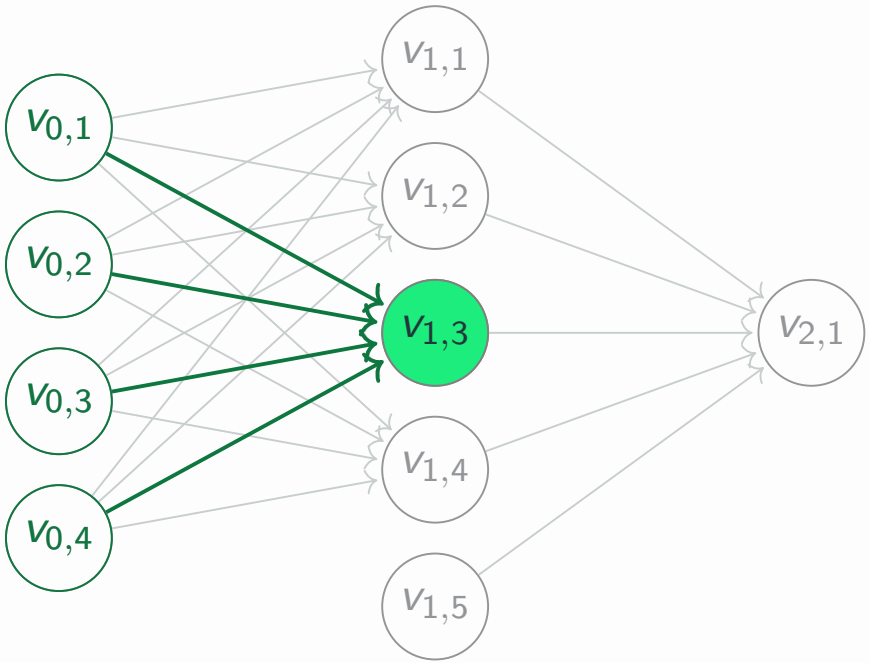
The input layer V_0 has $d+1$ nodes. For the first d nodes $\sigma_{0,i}(a) = a$, and for the last node $d+1$ (*bias*), $\sigma_{0,d+1}(a) = 1$.

Feedforward Neural Networks



Let $o_{l,i}(\boldsymbol{x})$ denote the output of $v_{l,i}$ given $\boldsymbol{x}$ as input. Thus, $o_{0,i}(\boldsymbol{x}) = x_i$ for $i \in \{1, \ldots, d\}$, and $o_{0,d+1}(\boldsymbol{x}) = 1$.

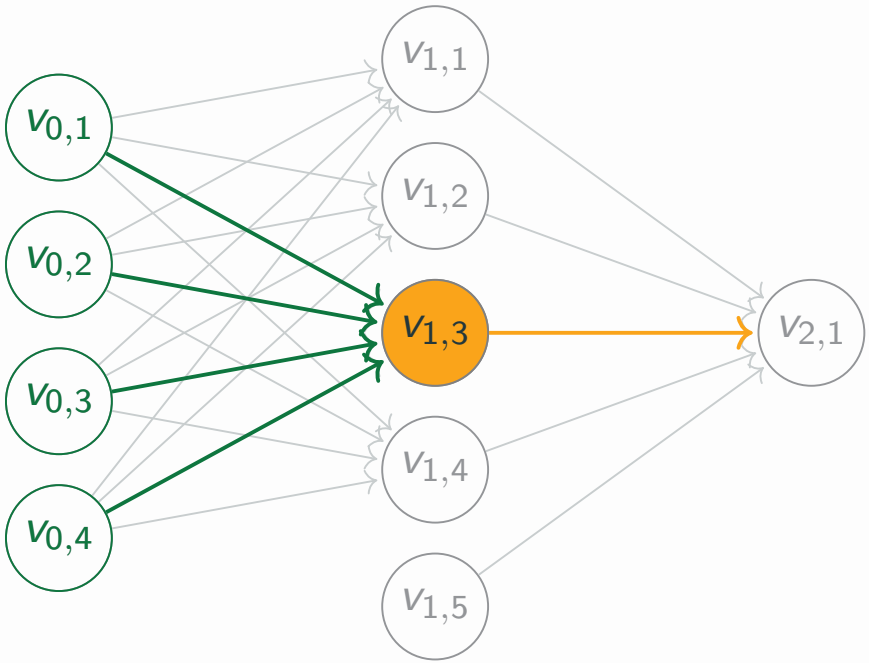
Feedforward Neural Networks



The input of any hidden neuron is the weighted sum of the outputs of the neurons connected to it. For $v_{1,3}$, the input is:

$$a_{1,3}(\boldsymbol{x}) = \sum_{i=1}^{4} w_{(0,i),(1,3)} o_{0,i}(\boldsymbol{x})$$

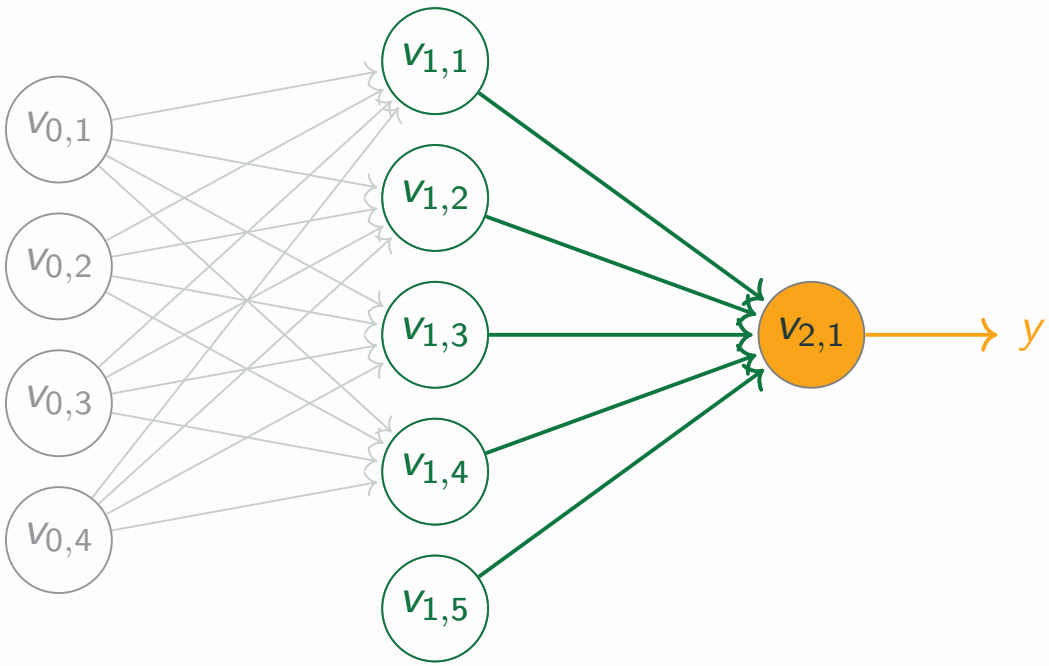
Feedforward Neural Networks



. . . and the output is therefore:

$$o_{1,3}(\boldsymbol{x}) = \sigma_{1,3}(a_{1,3}(\boldsymbol{x}))$$

Feedforward Neural Networks



By recursively applying the same operations, the final output is:

$$y = o_{2,1}(\boldsymbol{x}) = \sigma_{2,1}\left(\sum_{i=1}^{5} w_{(1,i),(2,1)} o_{1,i}(\boldsymbol{x})\right)$$

Outline

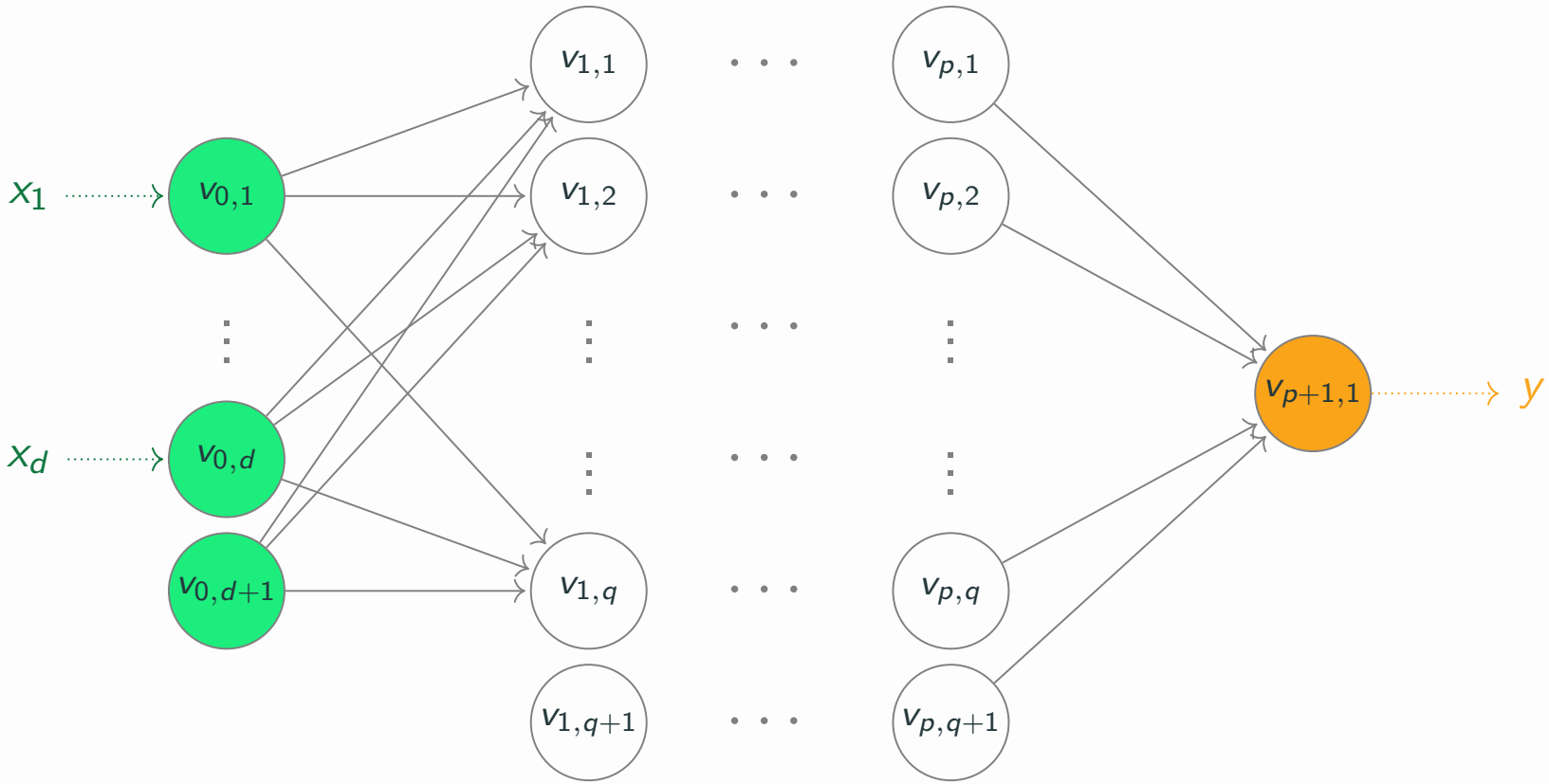
1 A Gentle Start

2 Statistical Learning

3 Deep Learning
- Neural Architectures
- Expressive Power
- Learning via Uniform Convergence
- Learning via Stability
- Stochastic Gradient Descent

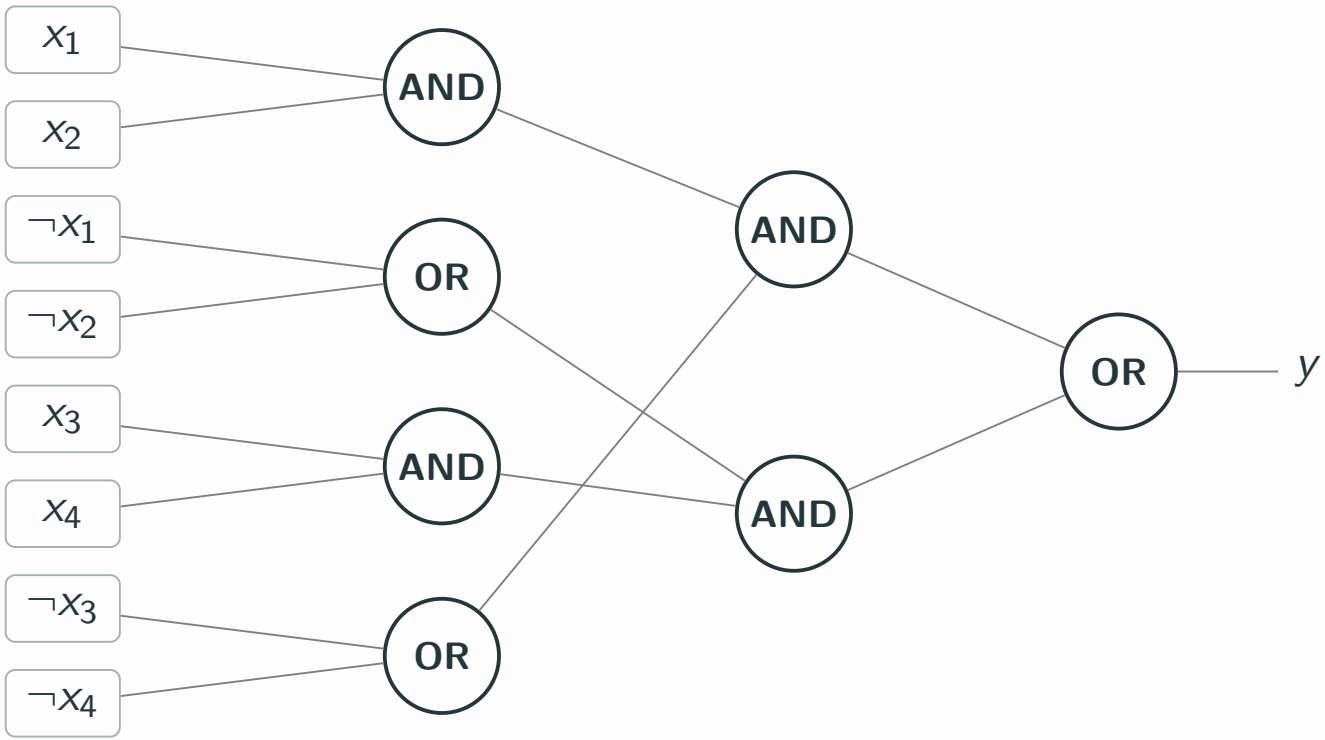
4 Perspectives

Expressing Boolean Functions



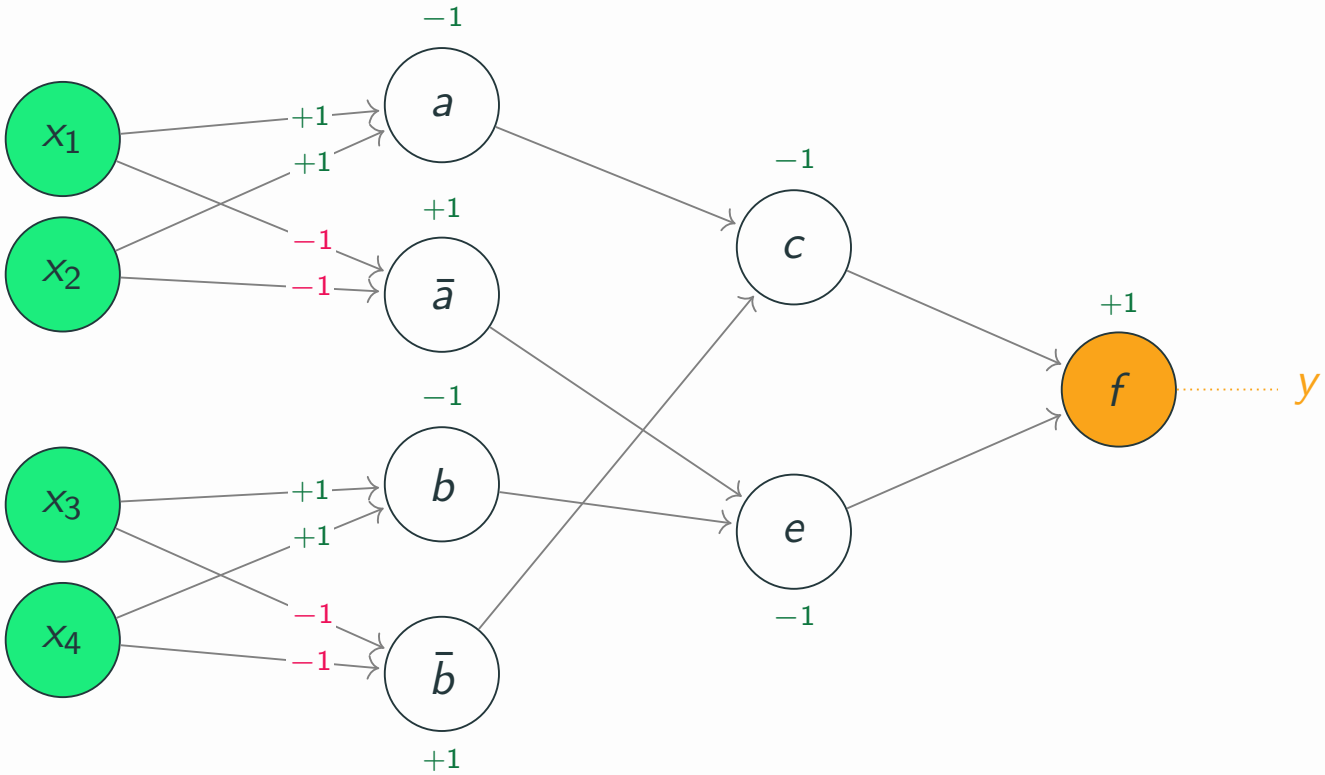
Any Boolean function $f : \{-1, +1\}^d \to \{-1, +1\}$ can be computed by a network of linear threshold units. But no architecture, whatever its depth, can express all Boolean functions with a size subexponential in d.

Representing Boolean Circuits



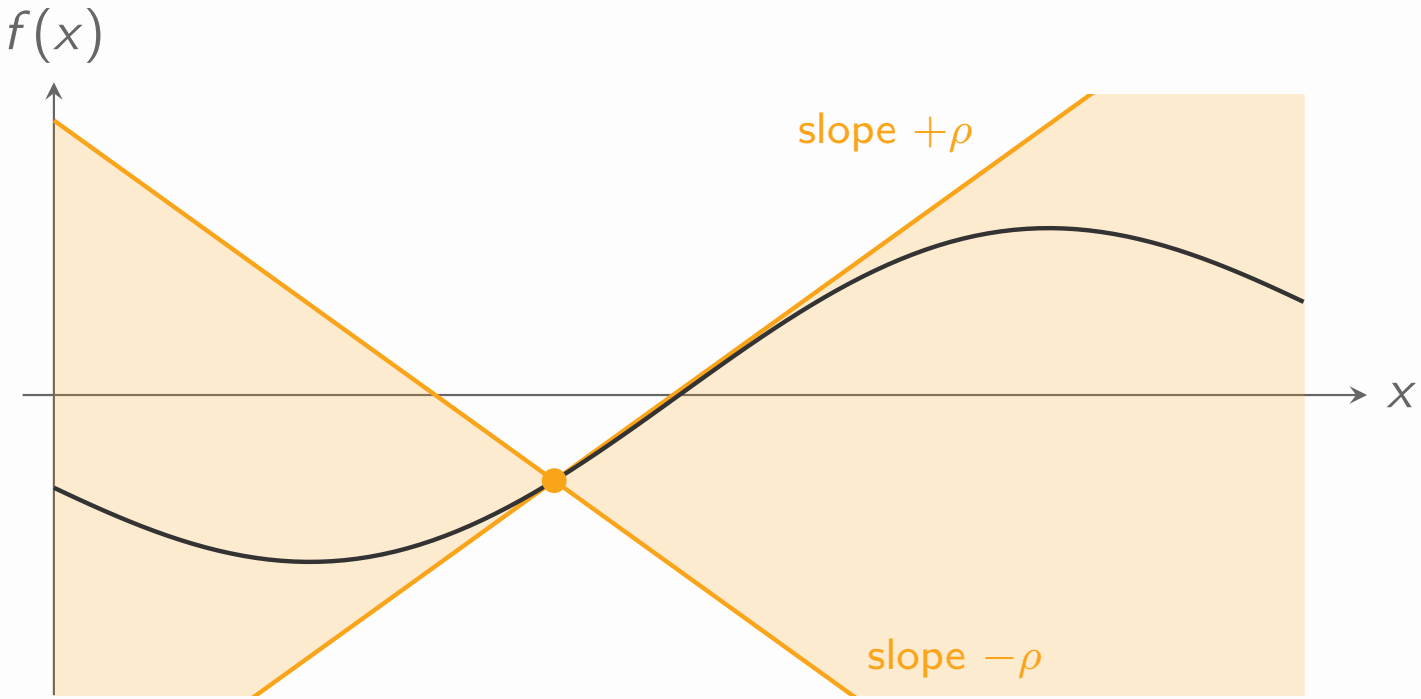
A Boolean circuit in negation normal form: AND and OR gates, with literals (variables or their negations) at the leaves. With 7 gates and depth 3, it computes $y = (x_1 \wedge x_2) \oplus (x_3 \wedge x_4)$.

Representing Boolean Circuits



$\mathrm{AND}(z_1, z_2) = \mathrm{sgn}(z_1 + z_2 - 1)$; $\mathrm{OR}(z_1, z_2) = \mathrm{sgn}(z_1 + z_2 + 1)$;
$\neg x_i = -x_i$ is a negative weight.
Thus, the network inherits the size and depth of the circuit.

Expressing Lipschitz Functions

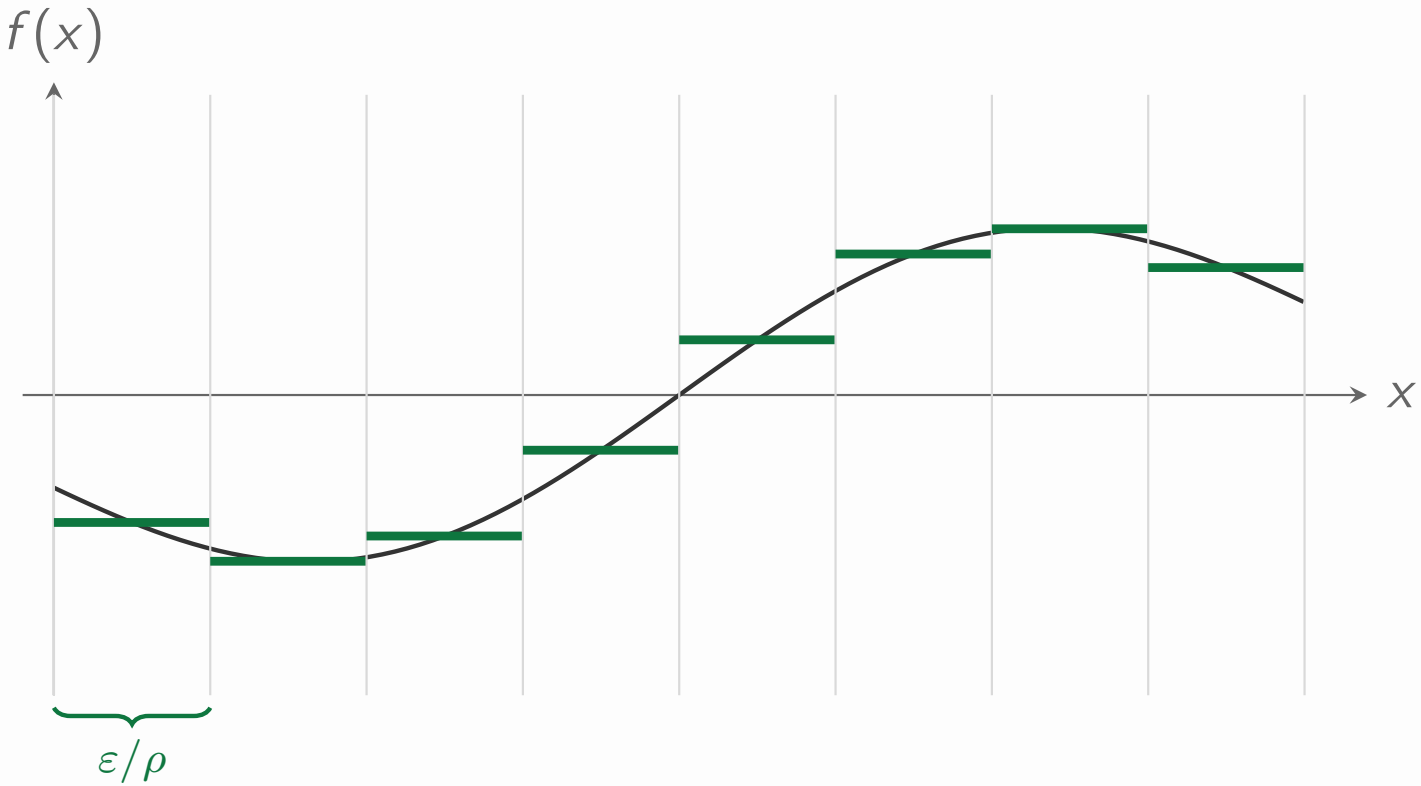


A function f is ρ-Lipschitz if

$$|f(\boldsymbol{x}) - f(\boldsymbol{x}')| \leq \rho \, \|\boldsymbol{x} - \boldsymbol{x}'\|_2$$

Its slope is bounded by ρ, so around every point, its graph stays inside the cone of slopes $\pm\rho$.

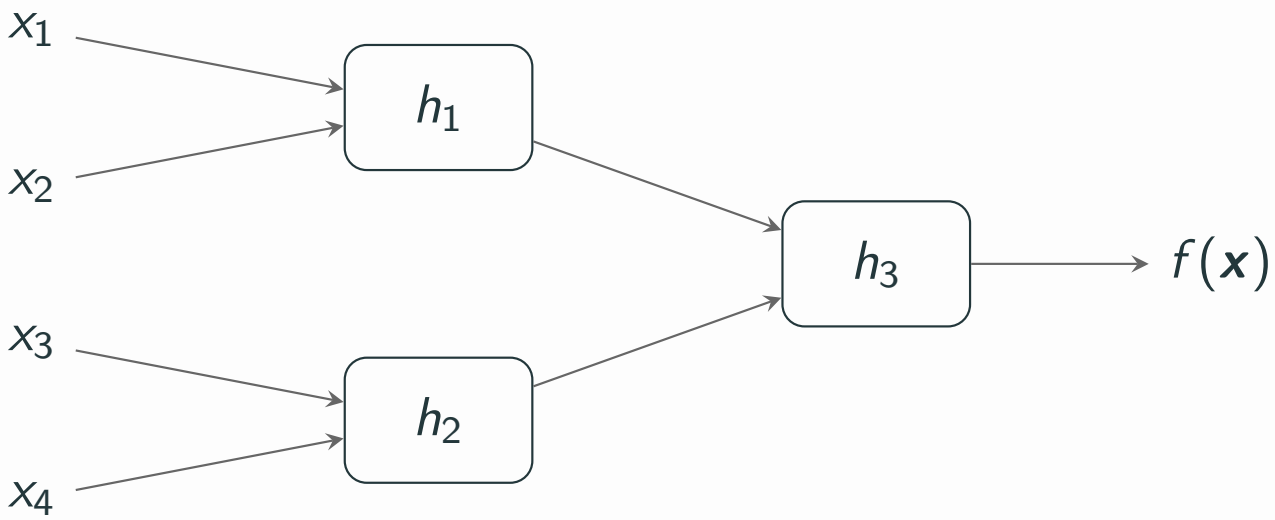
Expressing Lipschitz Functions



Split the domain into cells of width ε/ρ: on each cell, f is constant up to ε. A network with one threshold neuron per cell approximates f.

But in dimension d, there are $(\rho/\varepsilon)^d$ cells, and no architecture of size subexponential in d can approximate all ρ-Lipschitz functions.

Expressing Decomposable Functions

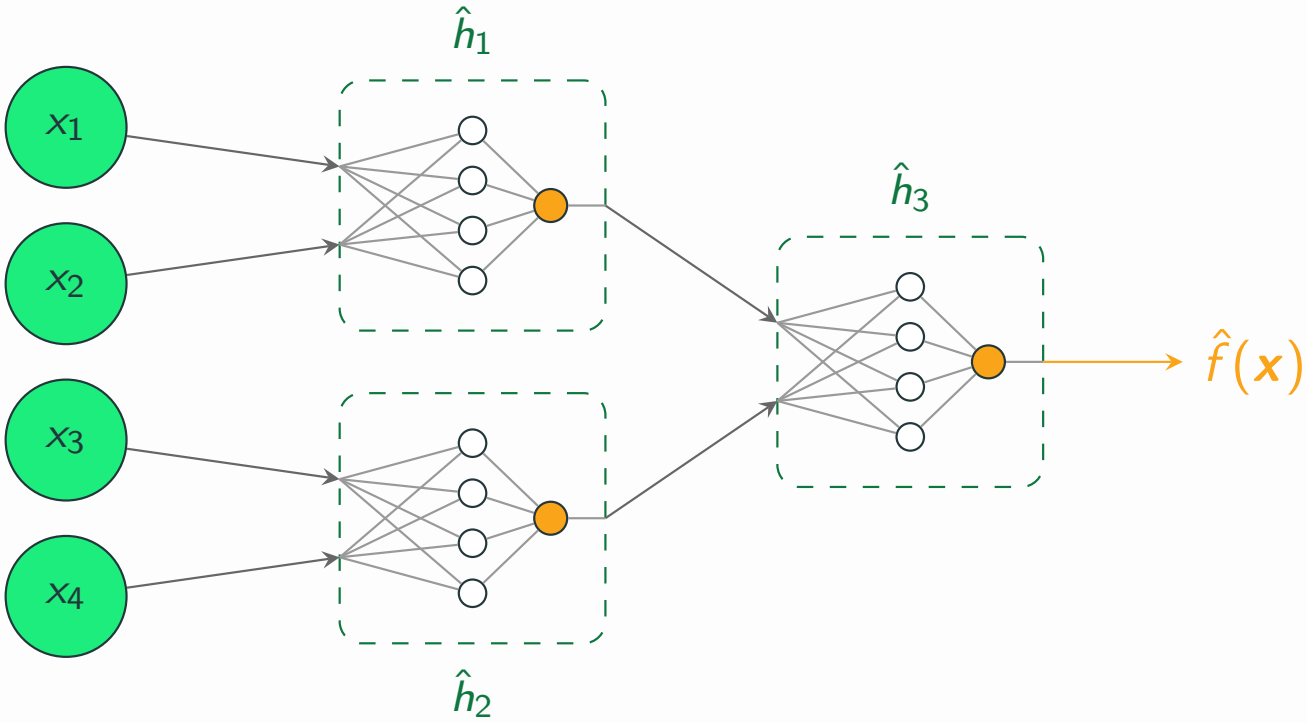


Now, consider a decomposable function:

$$f(\boldsymbol{x}) = h_3\big(h_1(x_1, x_2),\ h_2(x_3, x_4)\big)$$

where each h_i is ρ-Lipschitz and depends on only 2 variables.

Expressing Decomposable Functions



Each h_i is approximated by a shallow subnetwork $\hat{h}_i$ of size $(\rho/\varepsilon)^2$. The deep network mirrors the tree: its size is $\mathcal{O}\left(d\,(\rho/\varepsilon)^2\right)$ instead of $(\rho/\varepsilon)^d$.

Outline

1 A Gentle Start

2 Statistical Learning

3 Deep Learning
- Neural Architectures
- Expressive Power
- Learning via Uniform Convergence
- Learning via Stability
- Stochastic Gradient Descent

4 Perspectives

Sample Complexity of Neural Networks

VC Dimension of Neural Networks

Let $\mathcal{H}_{\text{NN}}$ be the class of binary classifiers computed by a feedforward network over a graph $G = (V, E)$ of depth L. Then:

$$\text{VCdim}(\mathcal{H}_{\text{NN}}) = \begin{cases} \mathcal{O}(|E| \log |E|) & \text{with sign activations,} \\ \mathcal{O}(|E|\, L \log |E|) & \text{with ReLU activations.} \end{cases}$$

Sample Complexity of Neural Networks

VC Dimension of Neural Networks

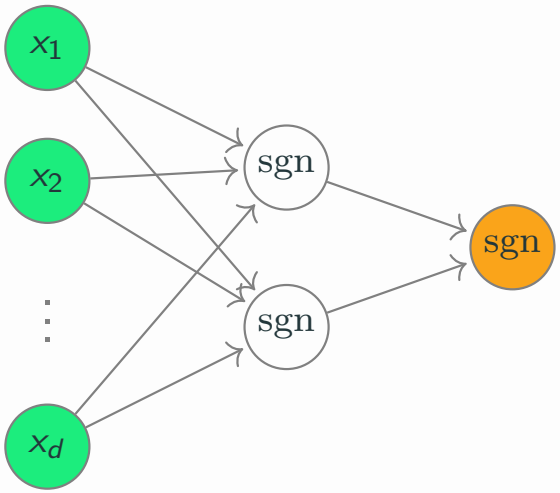
Let $\mathcal{H}_{\text{NN}}$ be the class of binary classifiers computed by a feedforward network over a graph $G = (V, E)$ of depth L. Then:

$$\text{VCdim}(\mathcal{H}_{\text{NN}}) = \begin{cases} \mathcal{O}(|E| \log |E|) & \text{with sign activations,} \\ \mathcal{O}(|E|\, L \log |E|) & \text{with ReLU activations.} \end{cases}$$

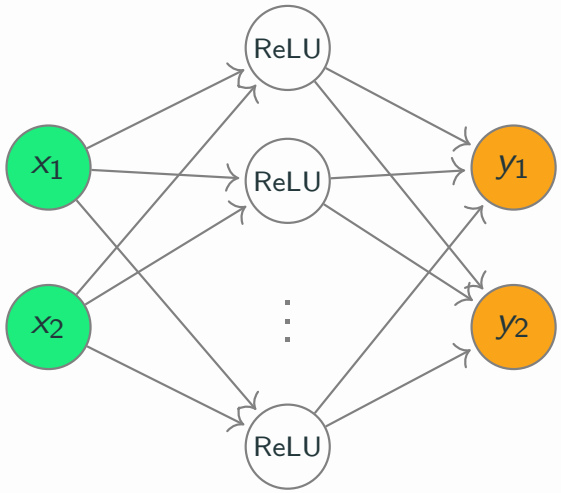
Network	**Dataset**	$\lvert E\rvert$	L	**Required** m	**Available** m
MLP 1×512	CIFAR-10	$1.2 \cdot 10^6$	2	$5 \cdot 10^9$	$5 \cdot 10^4$
AlexNet (2012)	ImageNet	$6.1 \cdot 10^7$	8	10^{12}	$1.3 \cdot 10^6$
VGG-16 (2014)	ImageNet	$1.4 \cdot 10^8$	16	$6 \cdot 10^{12}$	$1.3 \cdot 10^6$
ResNet-50 (2015)	ImageNet	$2.6 \cdot 10^7$	50	$3 \cdot 10^{12}$	$1.3 \cdot 10^6$

Required $m \approx \text{VCdim}/\varepsilon^2$ for $\varepsilon = 0.1$ (orders of magnitude, ignoring constants and δ). With bias nodes, $|E|$ is the number of parameters.

Computational Complexity of Neural Networks



d 2 neurons $\{-1, +1\}$

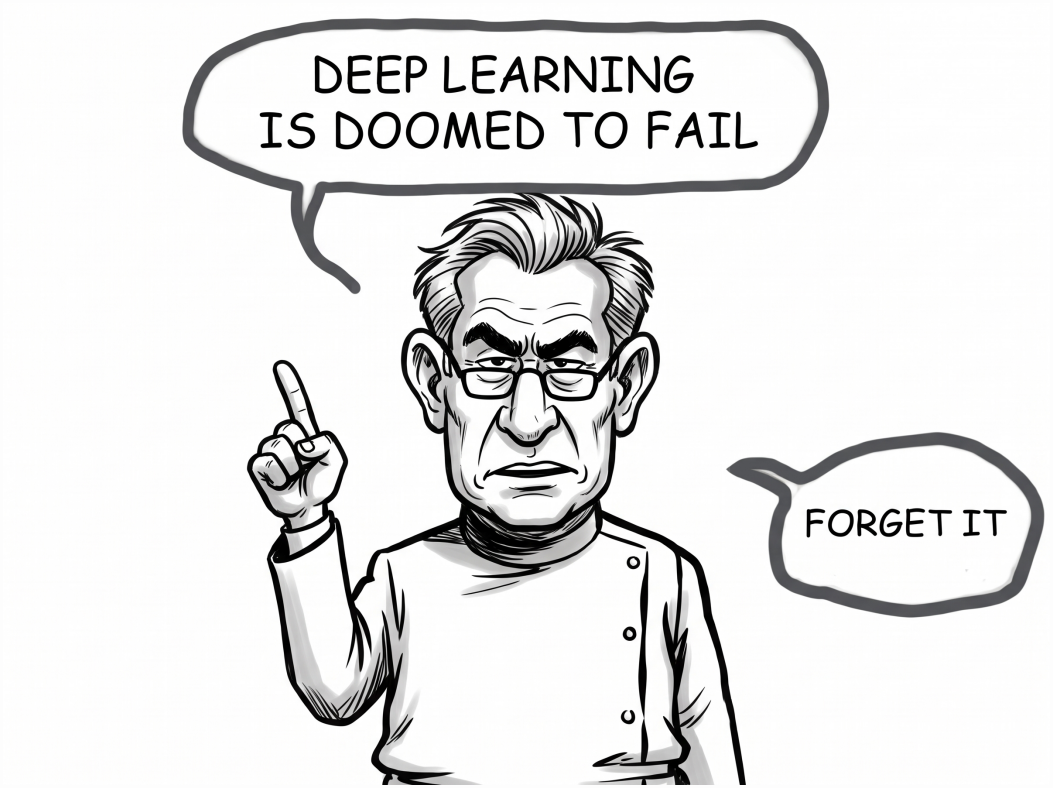


$\mathbb{R}^2$ width $\mathbb{R}^2$

Training Small Networks is Hard

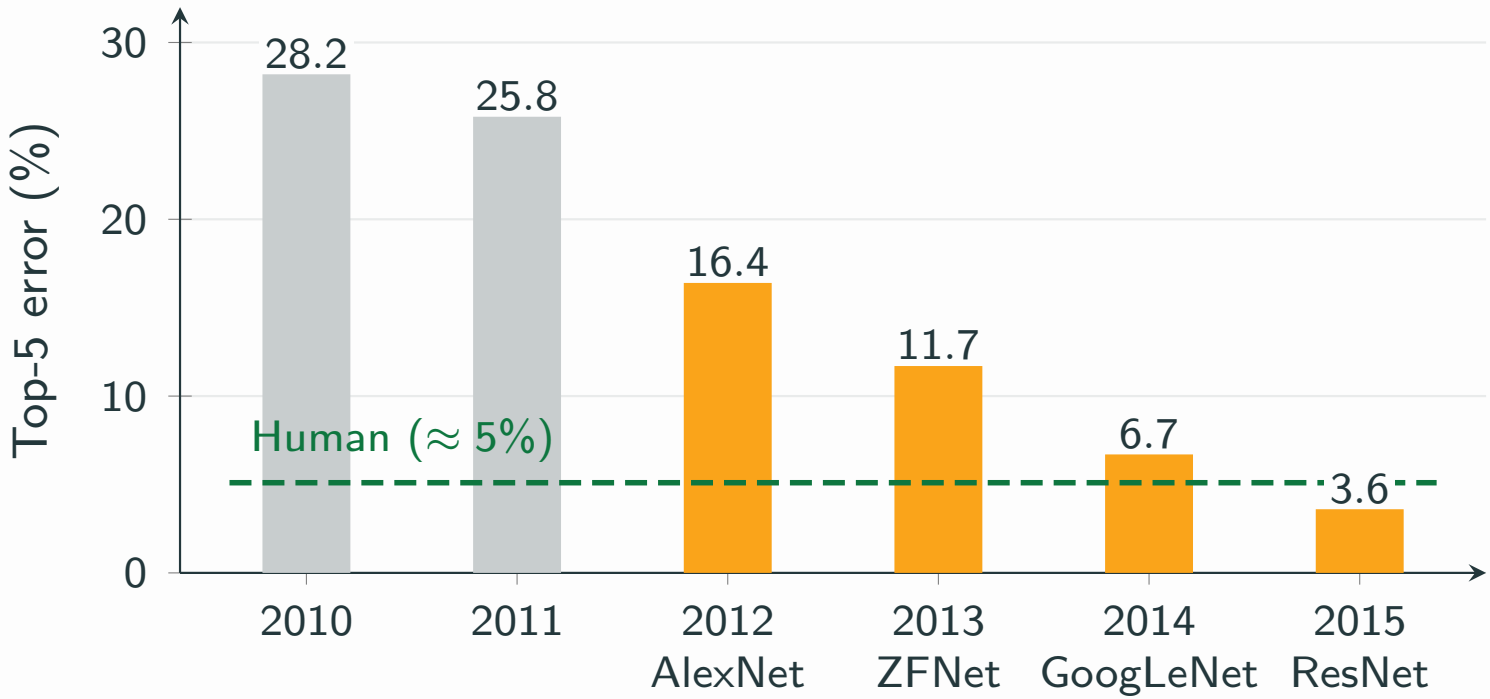
- **Classification (left):** deciding whether *two* hidden sign neurons can fit a training set is NP-complete.
- **Regression (right):** deciding whether a ReLU network can fit a training set is $\exists\mathbb{R}$-complete, hence NP-hard.

The Big Question



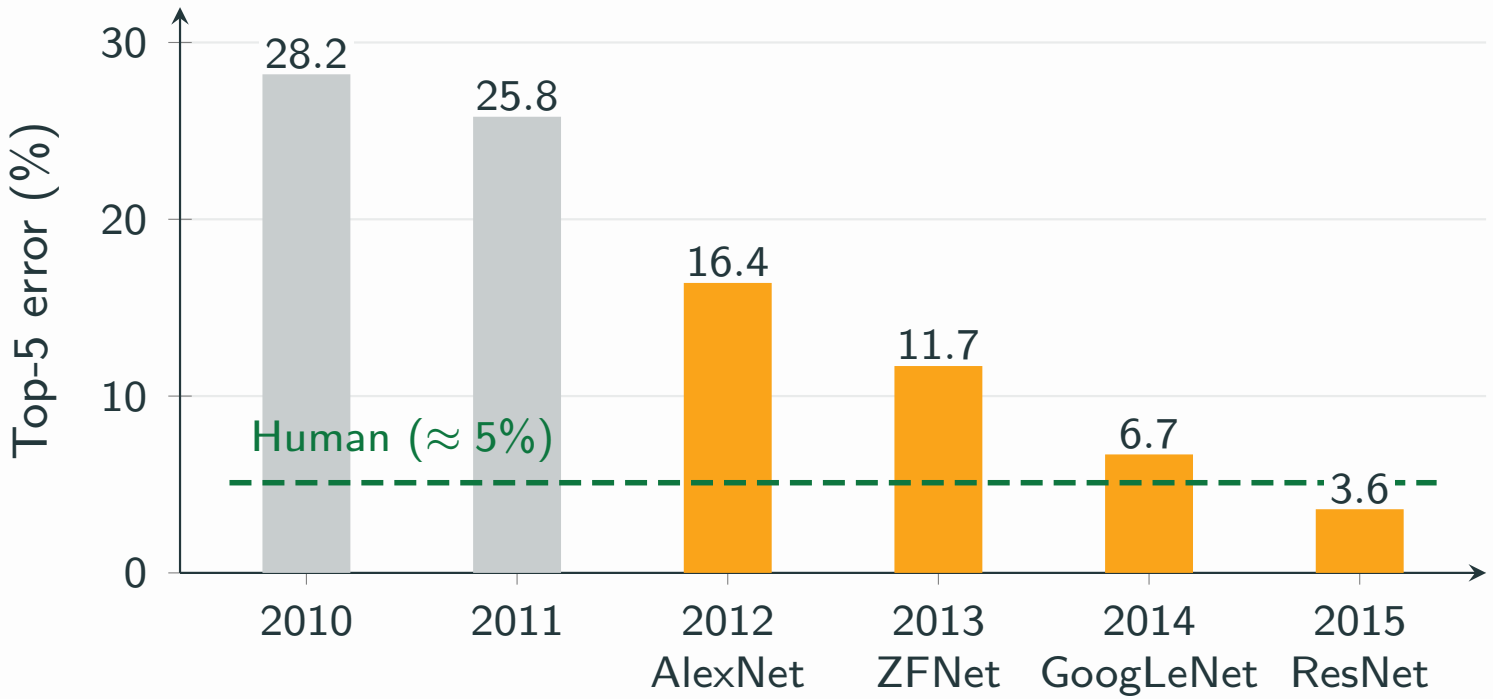
Far too many parameters for the available data, and an intractable training problem: according to theory, deep learning should not work.

The Big Question



Yet, on the ImageNet challenge, deep networks (orange) crushed all previous methods, and even outperformed humans in 2015.

The Big Question



How can we explain this success?

Outline

1 A Gentle Start

2 Statistical Learning

3 Deep Learning
- Neural Architectures
- Expressive Power
- Learning via Uniform Convergence
- Learning via Stability
- Stochastic Gradient Descent

4 Perspectives

Learning via Stability

A learning algorithm is stable if a small perturbation in the training set does not significantly change the prediction.

Learning via Stability

A learning algorithm is stable if a small perturbation in the training set does not significantly change the prediction.

Replacing One Example:
Let $S = (z_1, \ldots, z_{i-1}, z_i, z_{i+1}, \ldots, z_m)$ be a training set, and let i be an index drawn from the uniform distribution $\mathcal{U}$ over $\{1, \ldots, m\}$, and let

$$S^{(i)} = (z_1, \ldots, z_{i-1}, z', z_{i+1}, \ldots, z_m)$$

where z' is a fresh example drawn from $\mathcal{D}$.

Learning via Stability

On-Average Stability

A learning algorithm A is stable with rate $\epsilon(m)$ if:

$$\mathbb{E}_{(S,\boldsymbol{z}')\sim\mathcal{D}^{m+1},\, i\sim\mathcal{U}}\left[\,\ell(A(S^{(i)}),\boldsymbol{z}_i)-\ell(A(S),\boldsymbol{z}_i)\,\right]\;\leq\;\epsilon(m).$$

Learning via Stability

On-Average Stability

A learning algorithm A is stable with rate $\epsilon(m)$ if:

$$\mathbb{E}_{(S,\boldsymbol{z}')\sim\mathcal{D}^{m+1},\, i\sim\mathcal{U}}\left[\,\ell(A(S^{(i)}),\boldsymbol{z}_i)-\ell(A(S),\boldsymbol{z}_i)\,\right]\;\leq\;\epsilon(m).$$

Stability Means Generalization

If A is stable with rate $\epsilon(m)$, then

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[\,L_{\mathcal{D}}(A(S))-L_S(A(S))\,\right]\;\leq\;\epsilon(m).$$

While uniform convergence is a property of the hypothesis class, stability is a property of the learning algorithm.

Stability via Regularization

The Regularized Loss Minimization (RLM) principle replaces ERM by

$$A(S) \;=\; \operatorname*{argmin}_{\mathbf{w}} \; L_S(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \qquad \longleftarrow \text{stabilizer}$$

Stability of RLM

If ℓ is convex and ρ-Lipschitz, RLM is stable with rate $2\rho^2/(\lambda m)$:

$$\mathbb{E}_{S\sim\mathcal{D}^m}\big[\, L_{\mathcal{D}}(A(S)) - L_S(A(S)) \,\big] \;\leq\; \frac{2\rho^2}{\lambda m}.$$

Stability via Regularization

The Regularized Loss Minimization (RLM) principle replaces ERM by

$$A(S) \;=\; \operatorname*{argmin}_{\mathbf{w}} \; L_S(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \qquad \longleftarrow \text{stabilizer}$$

Stability of RLM

If ℓ is convex and ρ-Lipschitz, RLM is stable with rate $2\rho^2/(\lambda m)$:

$$\mathbb{E}_{S\sim\mathcal{D}^m}\big[\, L_{\mathcal{D}}(A(S)) - L_S(A(S)) \,\big] \;\le\; \frac{2\rho^2}{\lambda m}.$$

Loss	$\ell(\mathbf{w},(x,y))$	**Convex**	ρ-**Lipschitz**
Hinge	$\max(0,\, 1 - y\langle \mathbf{w}, x\rangle)$	✓	✓
Logistic	$\log(1 + e^{-y\langle \mathbf{w}, x\rangle})$	✓	✓
Squared	$(\langle \mathbf{w}, x\rangle - y)^2$	✓	✓ (bounded domains)
Zero-one	$\mathbb{1}[\operatorname{sgn}\langle \mathbf{w}, x\rangle \neq y]$	×	×

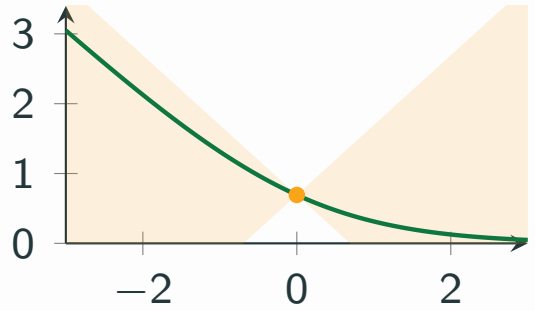
Regularized Logistic Regression

Logistic regression is a linear model that predicts a *probability*:

$$\mathbb{P}[y = +1 \mid \boldsymbol{x}] = \frac{1}{1 + e^{-\langle \mathbf{w}, \boldsymbol{x} \rangle}}$$

trained with the logistic loss on the score $\hat{y} = \langle \mathbf{w}, \boldsymbol{x} \rangle$:

$$\ell(\hat{y}, y) = \log\left(1 + e^{-y\,\hat{y}}\right)$$



$t = y\,\hat{y}$

Convex, with slopes in $[-1, 1]$: the graph never leaves the cone.

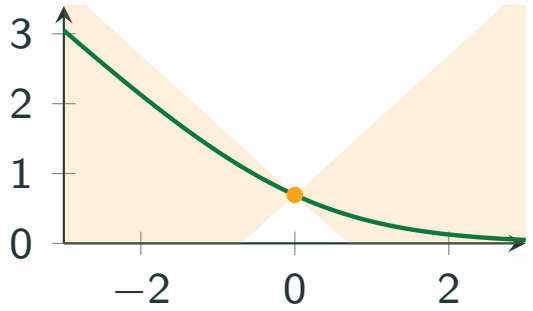
Regularized Logistic Regression

Logistic regression is a linear model that predicts a *probability*:

$$\mathbb{P}[y = +1 \mid \boldsymbol{x}] = \frac{1}{1 + e^{-\langle \mathbf{w}, \boldsymbol{x} \rangle}}$$

trained with the logistic loss on the score $\hat{y} = \langle \mathbf{w}, \boldsymbol{x} \rangle$:

$$\ell(\hat{y}, y) = \log\left(1 + e^{-y\,\hat{y}}\right)$$



$t = y\,\hat{y}$

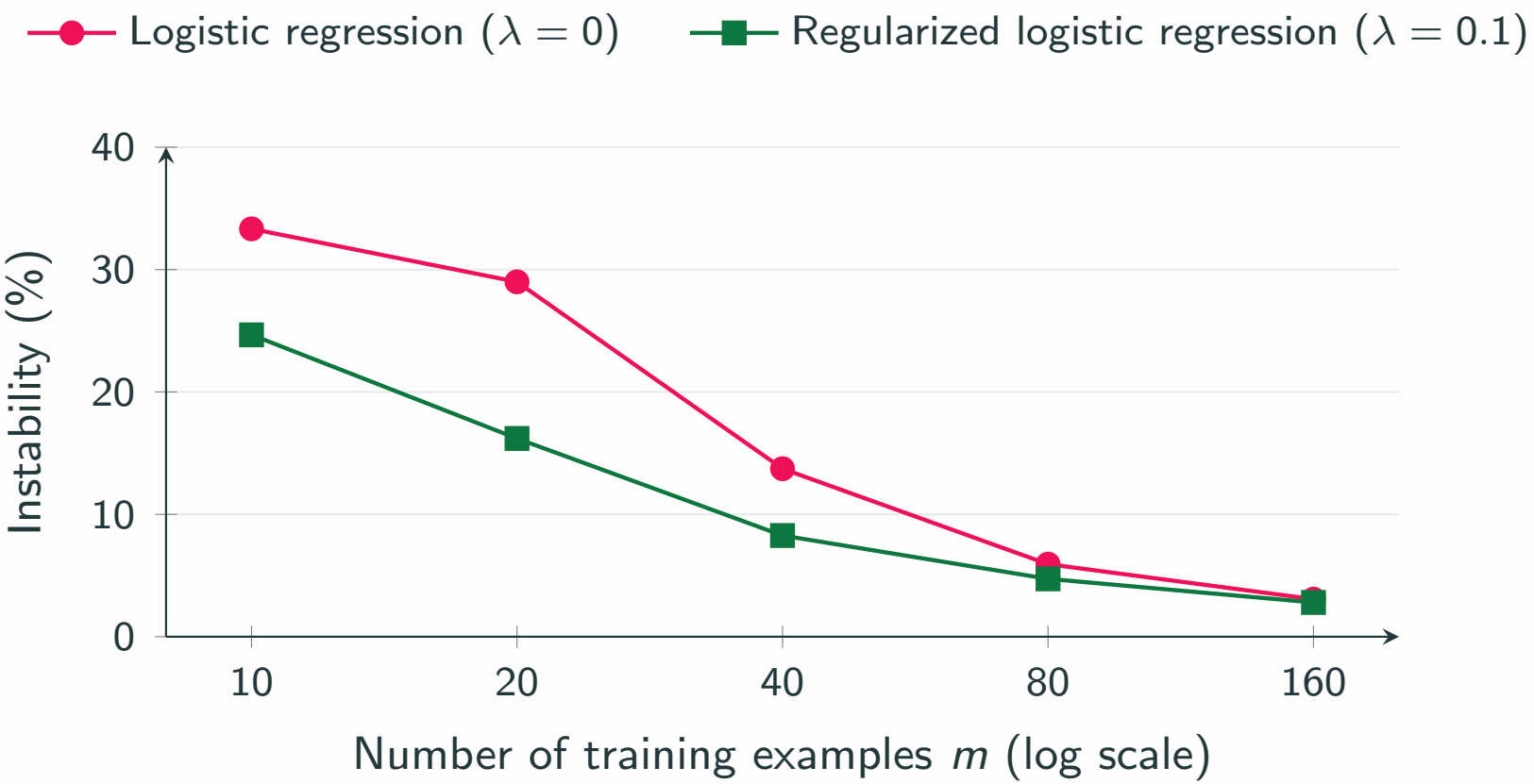
Convex, with slopes in $[-1, 1]$: the graph never leaves the cone.

Regularized logistic regression is RLM with this loss:

$$A(S) = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 + e^{-y_i \langle \mathbf{w}, \boldsymbol{x}_i \rangle}\right) + \lambda \|\mathbf{w}\|_2^2.$$

The problem is convex (solvable in polynomial time), and RLM is stable with rate $2\rho^2/(\lambda m)$.

Regularized Logistic Regression on Pima



Pima with all 8 features. For each m, we measure the instability $\mathbb{E}\left[\ell(A(S^{(i)}), \boldsymbol{z}_i) - \ell(A(S), \boldsymbol{z}_i)\right]$ using the zero-one loss and 1500 trials.

Regularized Logistic Regression on Pima



With $m = 20$ patients, $\lambda = 0$ reports a training error of 3% but has a true risk of 32%; $\lambda = 0.1$ reports 14% for a true risk of 29%. Stability makes the training error a reliable estimate of the true risk.

Outline

1 A Gentle Start

2 Statistical Learning

3 Deep Learning
- Neural Architectures
- Expressive Power
- Learning via Uniform Convergence
- Learning via Stability
- Stochastic Gradient Descent

4 Perspectives

Stochastic Gradient Descent (SGD)

Input: training set S, step size η, number of steps T

Initialize: $\mathbf{w}^{(1)} = \mathbf{0}$

for $t = 1, \ldots, T$ **do**

  draw $\boldsymbol{z}_t$ uniformly at random from S

  compute $\mathbf{v}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$

  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \, \mathbf{v}_t$

Output: $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

SGD minimizes $L_S(\mathbf{w}) = \frac{1}{m} \sum_i \ell(\mathbf{w}, \boldsymbol{z}_i)$ *without* ever computing its full gradient.

Stochastic Gradient Descent (SGD)

Input: training set S, step size η, number of steps T

Initialize: $\mathbf{w}^{(1)} = \mathbf{0}$

for $t = 1, \ldots, T$ **do**

  draw $\boldsymbol{z}_t$ uniformly at random from S ..... a single random example

  compute $\mathbf{v}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$

  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \, \mathbf{v}_t$

Output: $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

Since $\boldsymbol{z}_t$ is uniform over S, the stochastic gradient is unbiased:

$$\mathbb{E}[\mathbf{v}_t] = \nabla L_S(\mathbf{w}^{(t)})$$

Stochastic Gradient Descent (SGD)

Input: training set S, step size η, number of steps T

Initialize: $\mathbf{w}^{(1)} = \mathbf{0}$

for $t = 1, \ldots, T$ **do**

  draw $\boldsymbol{z}_t$ uniformly at random from S

  compute $\mathbf{v}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$ ..... the gradient is cheap

  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\, \mathbf{v}_t$

Output: $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

Since $\boldsymbol{z}_t$ is uniform over S, the stochastic gradient is unbiased:

$$\mathbb{E}[\mathbf{v}_t] = \nabla L_S(\mathbf{w}^{(t)})$$

Stochastic Gradient Descent (SGD)

Input: training set S, step size η, number of steps T

Initialize: $\mathbf{w}^{(1)} = \mathbf{0}$

for $t = 1, \ldots, T$ **do**

  draw $\boldsymbol{z}_t$ uniformly at random from S

  compute $\mathbf{v}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$

  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\, \mathbf{v}_t$ ..... a step against the gradient

Output: $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

Each step costs as much as one example, so SGD scales to millions of examples and parameters.

Stochastic Gradient Descent (SGD)

Input: training set S, step size η, number of steps T

Initialize: $\mathbf{w}^{(1)} = \mathbf{0}$

for $t = 1, \ldots, T$ **do**

  draw $\boldsymbol{z}_t$ uniformly at random from S

  compute $\mathbf{v}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$

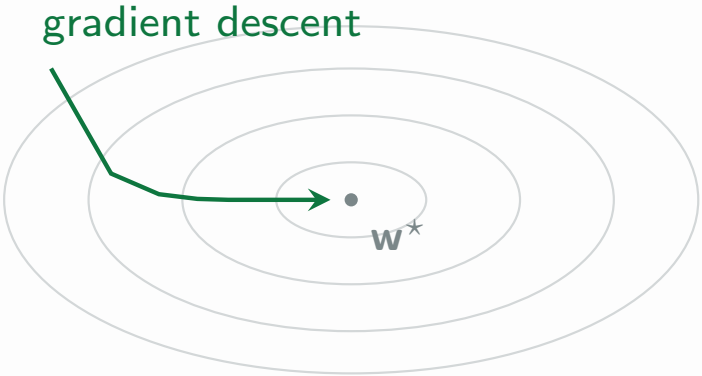
  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\, \mathbf{v}_t$

Output: $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$ ..... averaging reduces the noise

Each step costs as much as one example, so SGD scales to millions of examples and parameters.

SGD for Linear Models

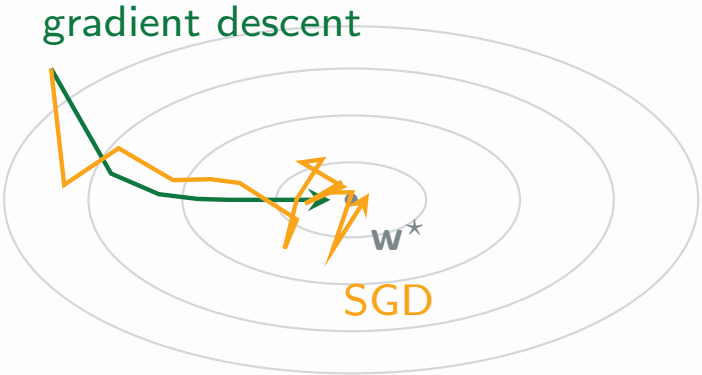
We predict with a linear model $\hat{y} = \langle \mathbf{w}, \boldsymbol{x} \rangle$ and a convex loss $\ell(\hat{y}, y)$ (*e.g.*, the logistic loss), and minimize $L_S(\mathbf{w}) = \frac{1}{m} \sum_i \ell(\langle \mathbf{w}, \boldsymbol{x}_i \rangle, y_i)$.



Gradient descent follows the steepest slope of L_S, but each step requires the whole training set.

SGD for Linear Models

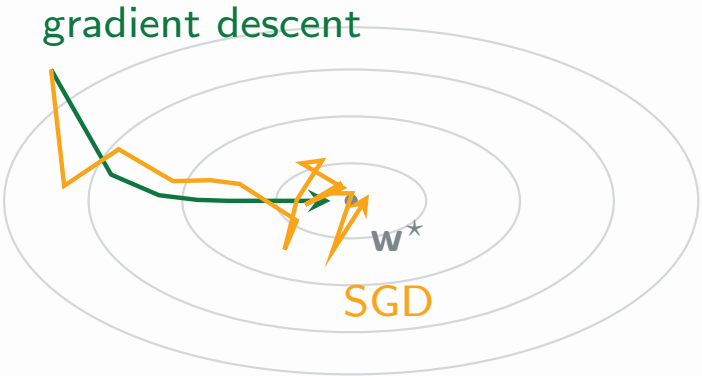
We predict with a linear model $\hat{y} = \langle \mathbf{w}, \boldsymbol{x} \rangle$ and a convex loss $\ell(\hat{y}, y)$ (*e.g.*, the logistic loss), and minimize $L_S(\mathbf{w}) = \frac{1}{m} \sum_i \ell(\langle \mathbf{w}, \boldsymbol{x}_i \rangle, y_i)$.



SGD zigzags, because each step uses a single example; yet, *on average*, it goes downhill.

SGD for Linear Models

We predict with a linear model $\hat{y} = \langle \mathbf{w}, \boldsymbol{x} \rangle$ and a convex loss $\ell(\hat{y}, y)$ (*e.g.*, the logistic loss), and minimize $L_S(\mathbf{w}) = \frac{1}{m} \sum_i \ell(\langle \mathbf{w}, \boldsymbol{x}_i \rangle, y_i)$.



Convergence of SGD

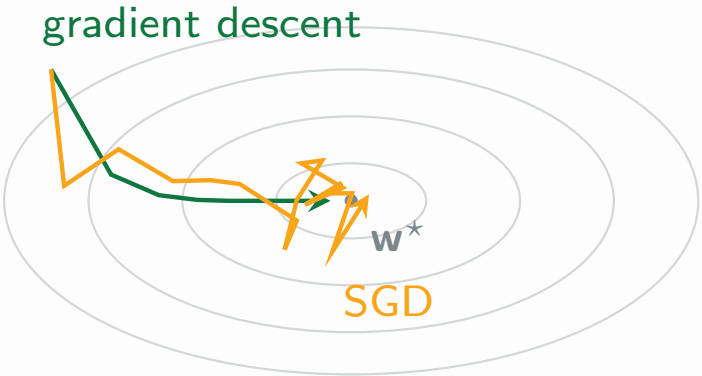
If ℓ is convex and ρ-Lipschitz, $\mathbf{w}^\star$ minimizes L_S with $\|\mathbf{w}^\star\| \leq B$, and $\eta = B/(\rho\sqrt{T})$, then

$$\mathbb{E}\big[L_S(\bar{\mathbf{w}})\big] - L_S(\mathbf{w}^\star) \;\leq\; \frac{B\rho}{\sqrt{T}}.$$

The noise only slows down convergence: the error decreases as $1/\sqrt{T}$.

SGD for Linear Models

We predict with a linear model $\hat{y} = \langle \mathbf{w}, \boldsymbol{x} \rangle$ and a convex loss $\ell(\hat{y}, y)$ (*e.g.*, the logistic loss), and minimize $L_S(\mathbf{w}) = \frac{1}{m} \sum_i \ell(\langle \mathbf{w}, \boldsymbol{x}_i \rangle, y_i)$.



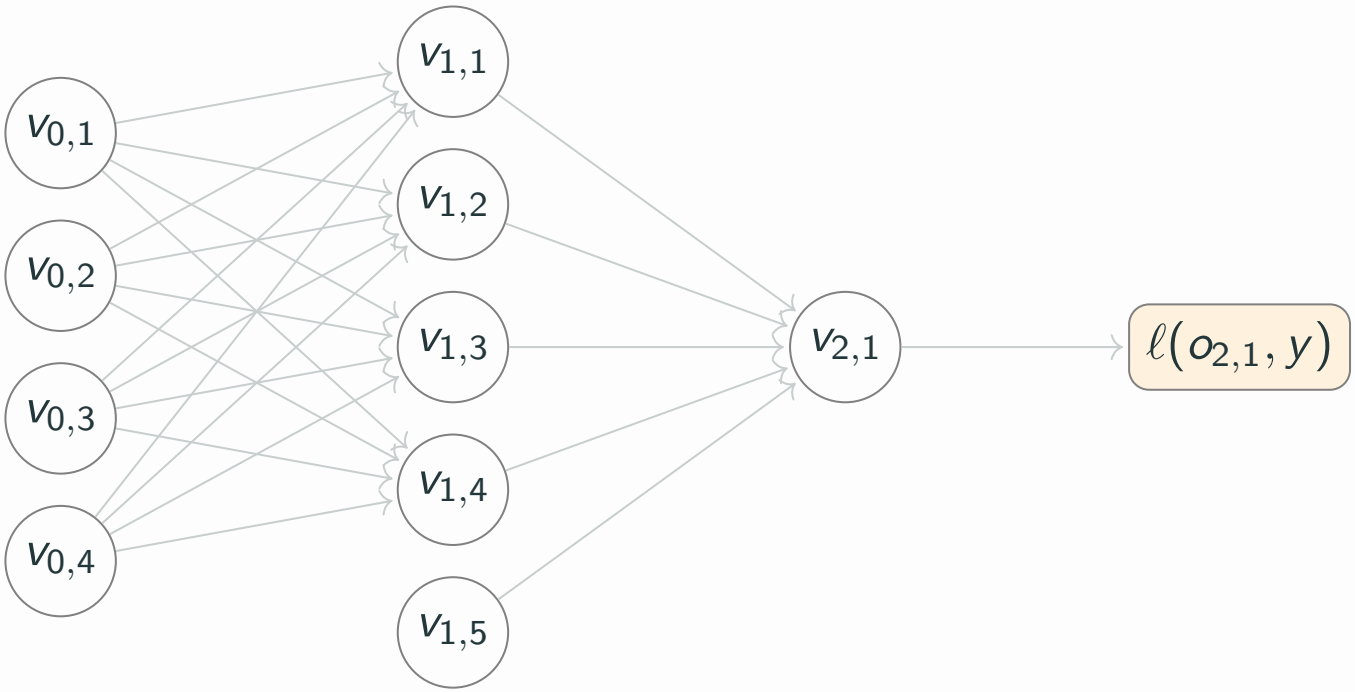
Convergence of SGD

If ℓ is convex and ρ-Lipschitz, $\mathbf{w}^\star$ minimizes L_S with $\|\mathbf{w}^\star\| \leq B$, and $\eta = B/(\rho\sqrt{T})$, then

$$\mathbb{E}\big[L_S(\bar{\mathbf{w}})\big] - L_S(\mathbf{w}^\star) \;\leq\; \frac{B\rho}{\sqrt{T}}.$$

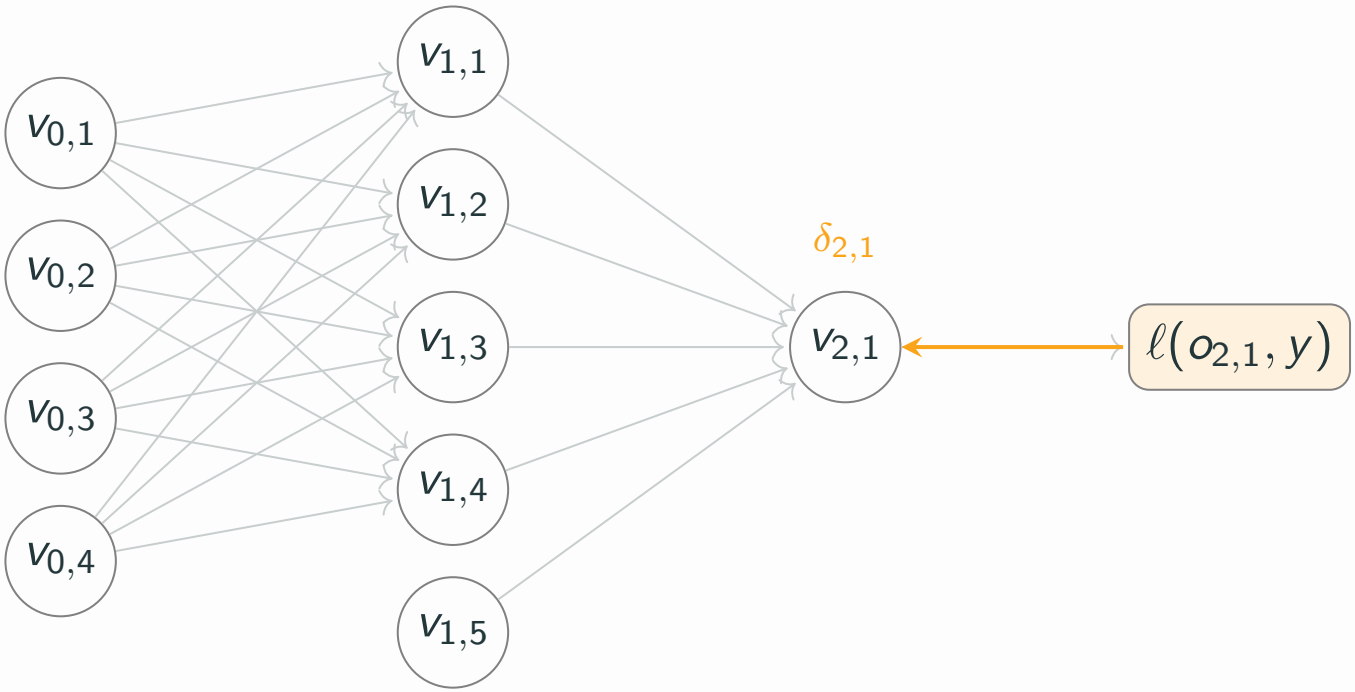
One pass over the data: with a fresh example at each step, SGD directly minimizes the *true* risk $L_{\mathcal{D}}$. Hence $m = B^2\rho^2/\varepsilon^2$ examples suffice: this depends on the radius B, not on the number of parameters.

SGD for Neural Networks: Backpropagation



To apply SGD to a network, we need the gradient of $\ell(o_{2,1}(\boldsymbol{x}), y)$ with respect to *every* weight w_e. The forward pass computes all inputs $a_{l,i}$ and outputs $o_{l,i}$, as before.

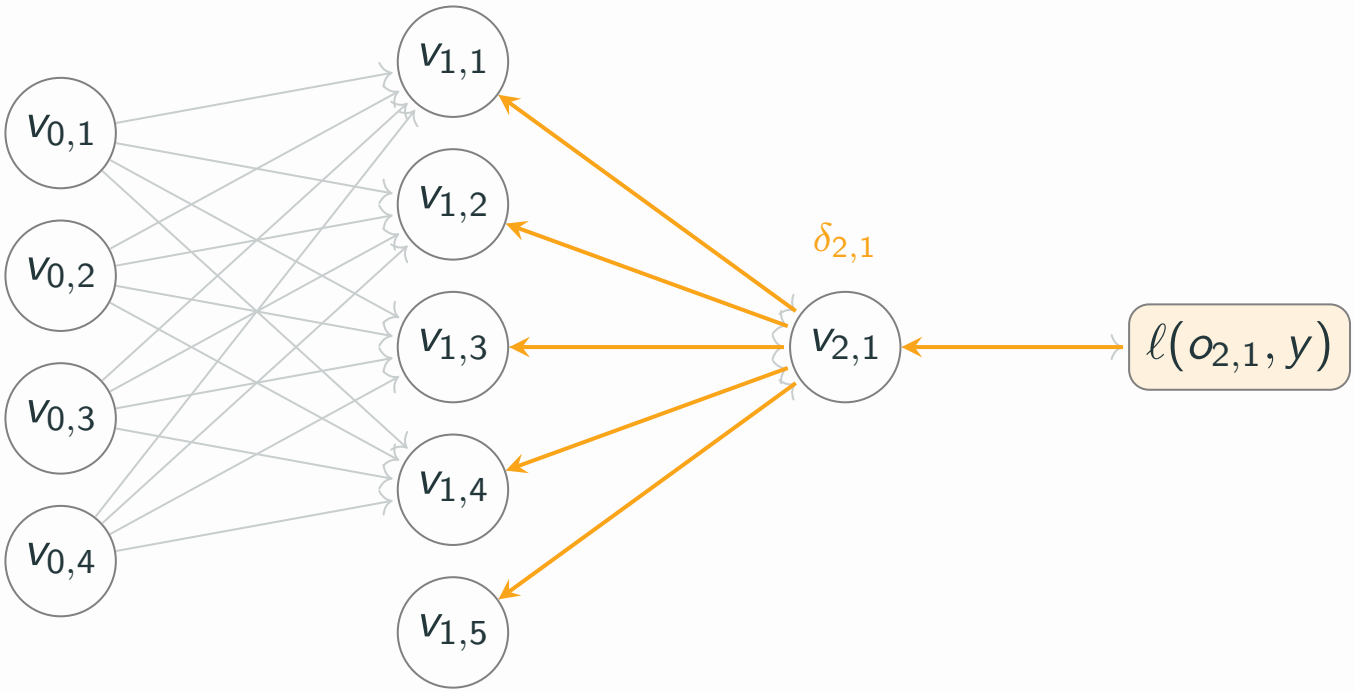
SGD for Neural Networks: Backpropagation



The backward pass starts from the loss. At the output neuron, let

$$\delta_{2,1} = \frac{\partial \ell}{\partial a_{2,1}} = \ell'(o_{2,1}, y)\, \sigma'_{2,1}(a_{2,1})$$

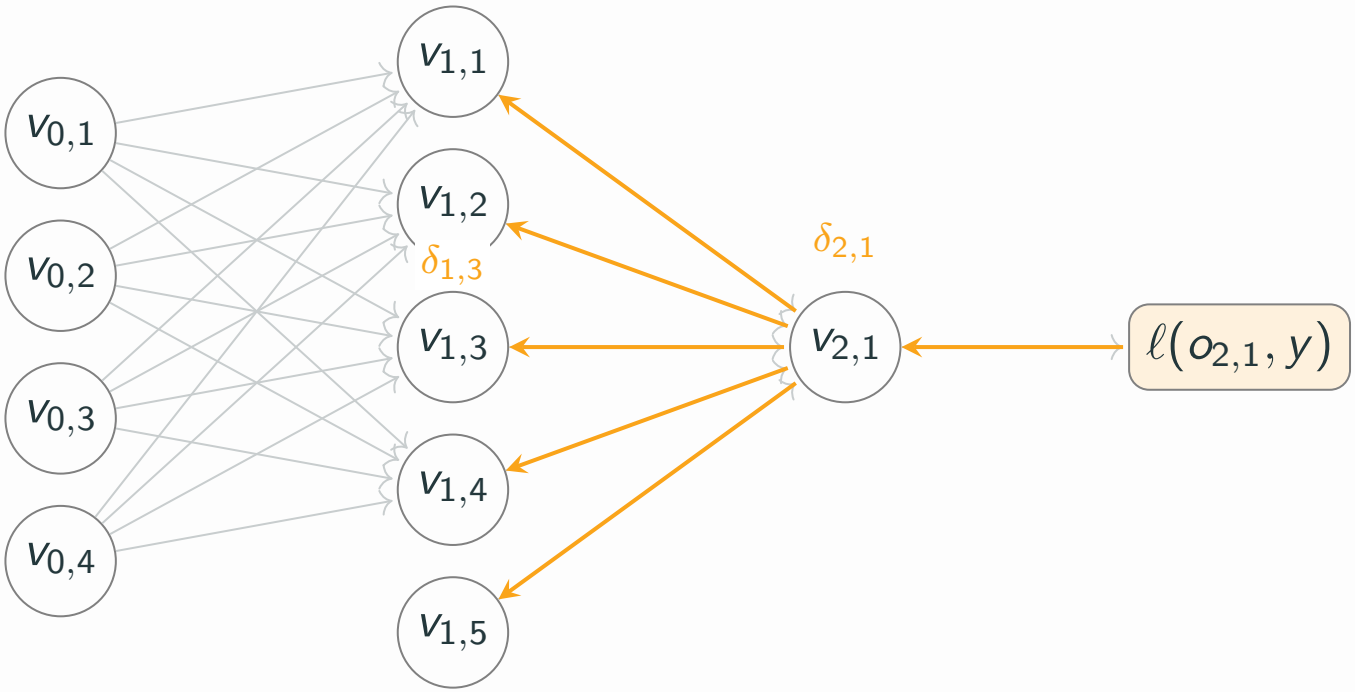
SGD for Neural Networks: Backpropagation



By the chain rule, the gradient for each weight entering the output neuron is

$$\frac{\partial \ell}{\partial w_{(1,i),(2,1)}} = \delta_{2,1}\, o_{1,i}$$

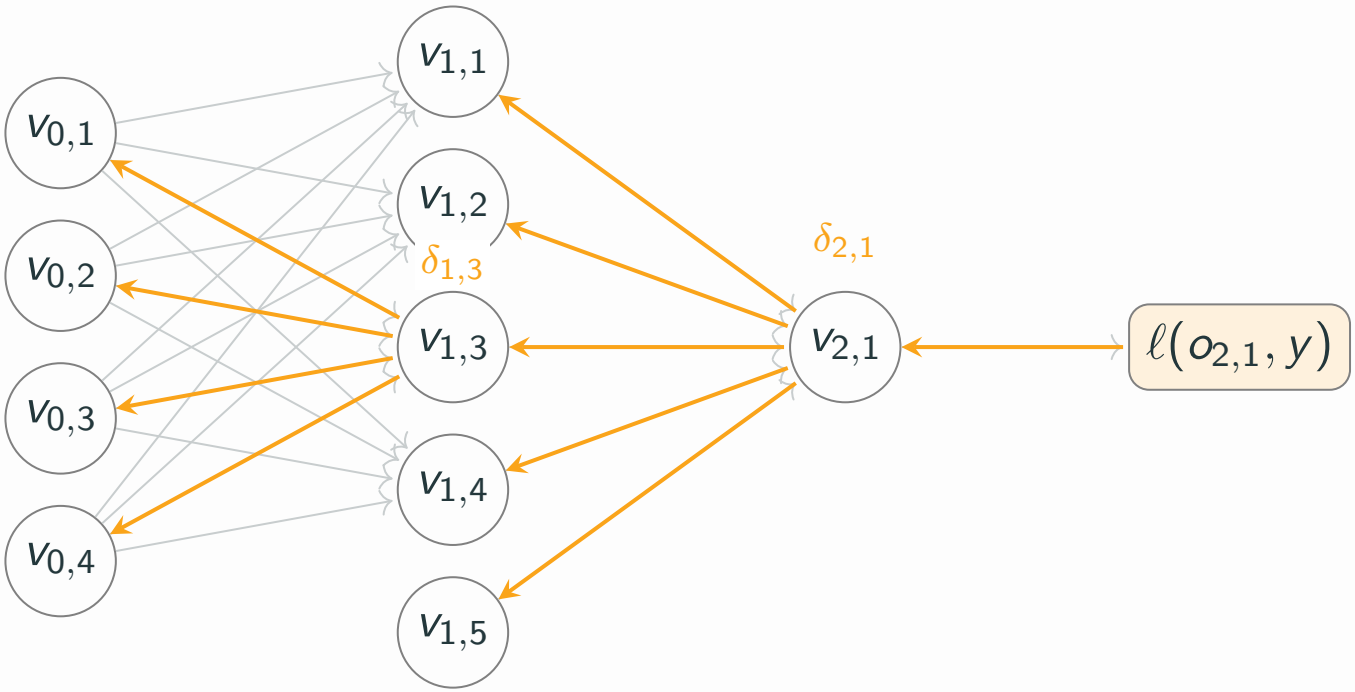
SGD for Neural Networks: Backpropagation



The error signal flows backwards to the hidden neurons:

$$\delta_{1,3} = \sigma'_{1,3}(a_{1,3})\ w_{(1,3),(2,1)}\ \delta_{2,1}$$

SGD for Neural Networks: Backpropagation



Therefore:

$$\frac{\partial \ell}{\partial w_{(0,i),(1,3)}} = \delta_{1,3}\, o_{0,i}$$

Each edge is used once forward and once backward: all gradients are computed in time $\mathcal{O}(|E|)$.

SGD for Neural Networks: Runtime

For neural networks, the loss is not convex in the weights:

- SGD is only guaranteed to approach a *stationary point* ($\nabla L_S \approx \mathbf{0}$), which may be a local minimum or a saddle point;
- yet, for large (over-parameterized) networks, SGD often reaches $L_S = 0$ in practice, despite NP-hardness in the worst case.

SGD for Neural Networks: Runtime

For neural networks, the loss is not convex in the weights:

- SGD is only guaranteed to approach a *stationary point* ($\nabla L_S \approx \mathbf{0}$), which may be a local minimum or a saddle point;
- yet, for large (over-parameterized) networks, SGD often reaches $L_S = 0$ in practice, despite NP-hardness in the worst case.

Note: backpropagation requires differentiable activations. The sign function is replaced by the identity at the output (with the logistic loss), and the ReLU is differentiable everywhere except at 0.

SGD for Neural Networks: Stability

Stability of SGD

If the loss is ρ-Lipschitz and β-smooth (its curvature is bounded by β), then SGD with steps $\eta_t \leq c/t$ is stable with rate

$$\epsilon(m) \;=\; \mathcal{O}\left(\frac{1}{m} T^{\frac{\beta c}{\beta c+1}}\right).$$

SGD for Neural Networks: Stability

Stability of SGD

If the loss is ρ-Lipschitz and β-smooth (its curvature is bounded by β), then SGD with steps $\eta_t \leq c/t$ is stable with rate

$$\epsilon(m) = \mathcal{O}\left(\frac{1}{m} T^{\frac{\beta c}{\beta c+1}}\right).$$

- Implicit regularization: fewer steps (T) or smaller steps (c) give more stability, like λ in RLM (*e.g.*, early stopping).

SGD for Neural Networks: Stability

Stability of SGD

If the loss is ρ-Lipschitz and β-smooth (its curvature is bounded by β), then SGD with steps $\eta_t \leq c/t$ is stable with rate

$$\epsilon(m) = \mathcal{O}\left(\frac{1}{m} T^{\frac{\beta c}{\beta c+1}}\right).$$

- Implicit regularization: fewer steps (T) or smaller steps (c) give more stability, like λ in RLM (*e.g.*, early stopping).
- Role of the loss: a Lipschitz and smooth loss (*e.g.*, logistic) keeps ρ and β small.

SGD for Neural Networks: Stability

Stability of SGD

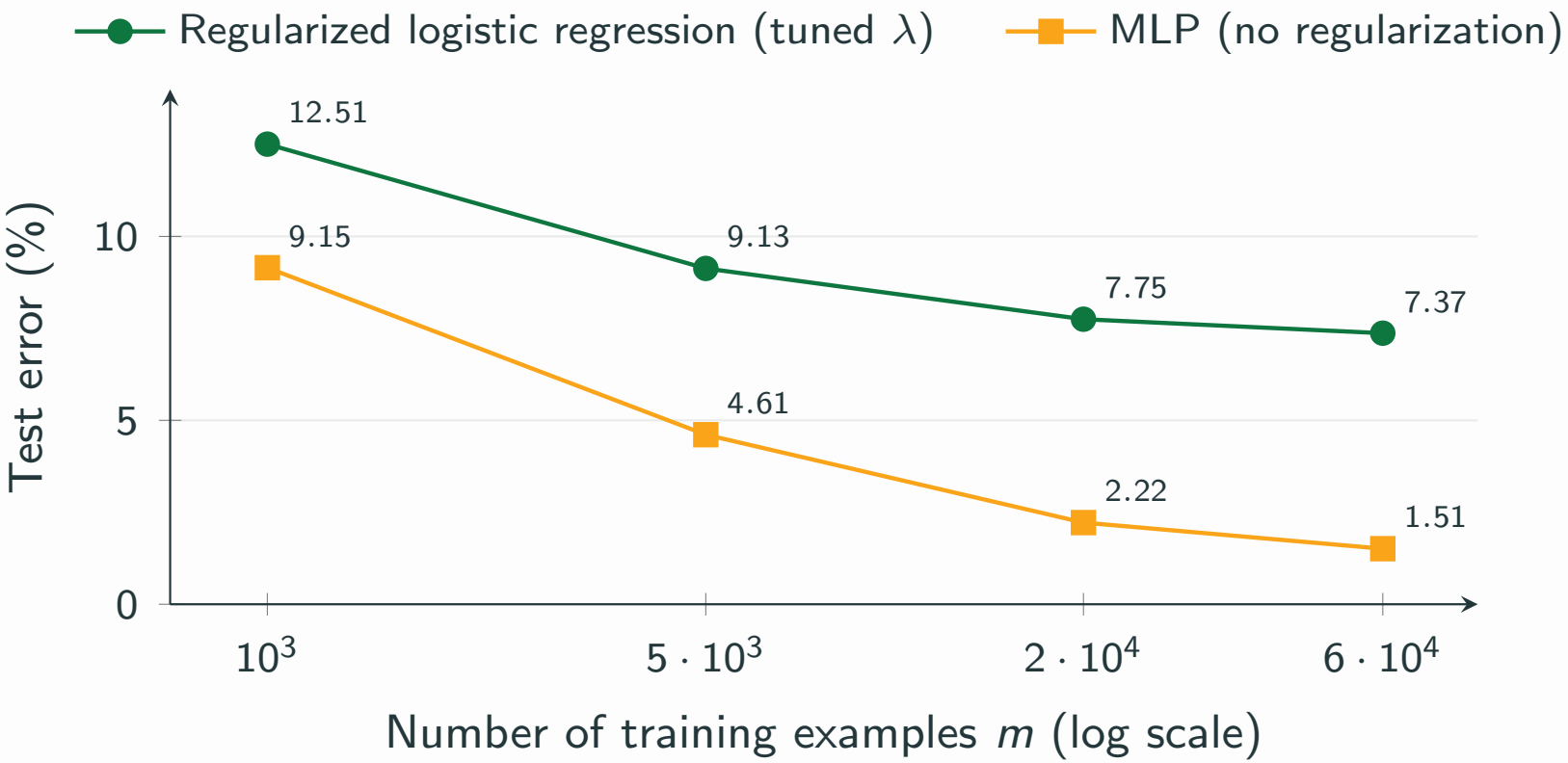
If the loss is ρ-Lipschitz and β-smooth (its curvature is bounded by β), then SGD with steps $\eta_t \leq c/t$ is stable with rate

$$\epsilon(m) = \mathcal{O}\left(\frac{1}{m} T^{\frac{\beta c}{\beta c+1}}\right).$$

- Implicit regularization: fewer steps (T) or smaller steps (c) give more stability, like λ in RLM (*e.g.*, early stopping).
- Role of the loss: a Lipschitz and smooth loss (*e.g.*, logistic) keeps ρ and β small.
- Role of the architecture: for a network, ρ and β depend on the weights and the depth; initialization, normalization and residual connections keep them under control.

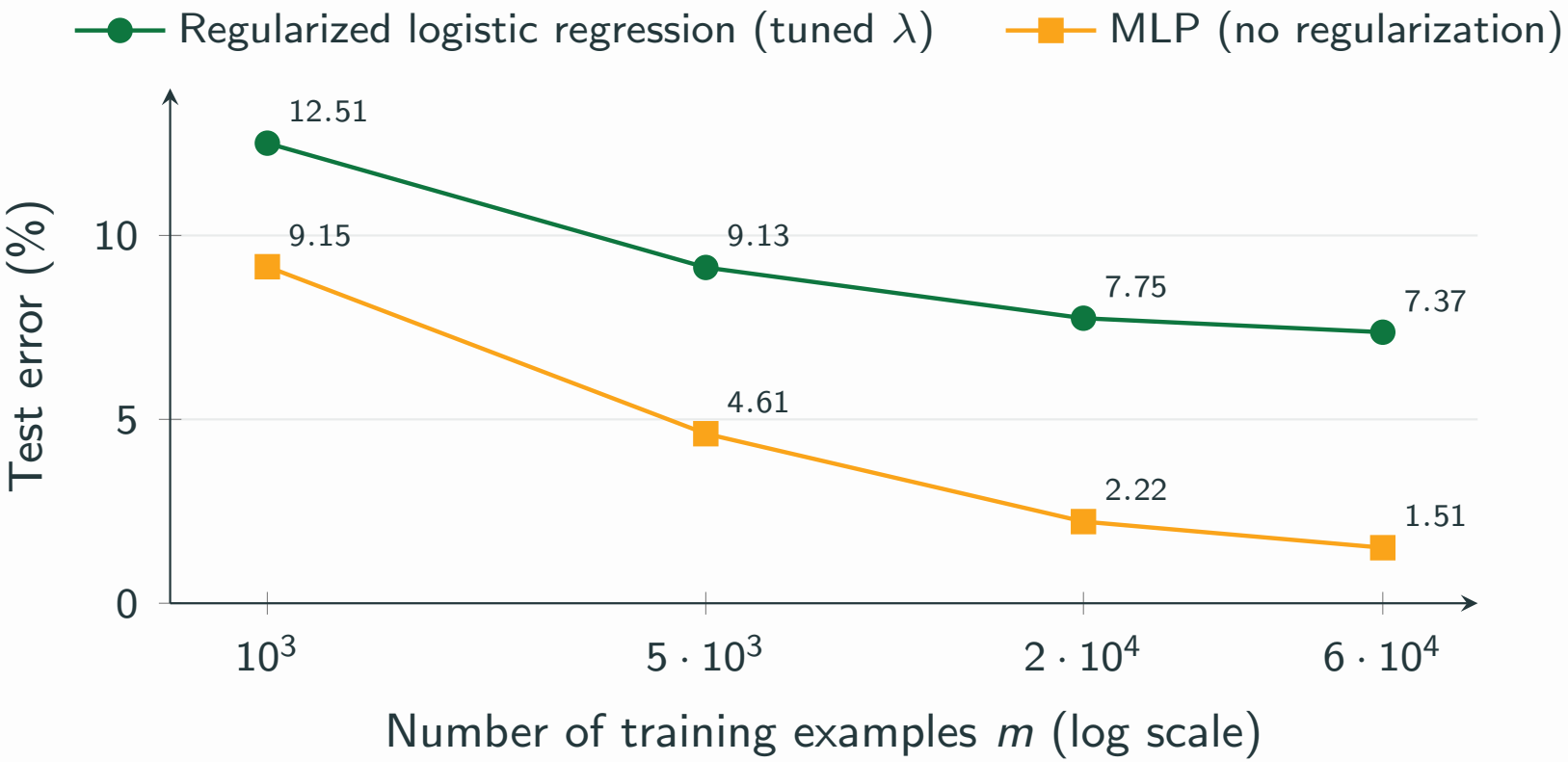
Caveat: ReLU networks are not smooth at 0; the theorem idealizes their behavior.

Comparisons on MNIST



- MLP: 2 hidden layers of 512 ReLU neurons, hence $|E| \approx 6.7 \cdot 10^5$ weights. *No regularization*: it is trained until $L_S = 0$.
- Logistic regression: λ is tuned *on the test set*, which favors it.

Comparisons on MNIST



With only $m = 1\,000$ examples, *i.e.*, 670 times fewer than weights, the MLP already wins.

Thus, the MLP is stable on MNIST: the architecture and the algorithm select, among all hypotheses with $L_S = 0$, one that generalizes.

Outline

1 A Gentle Start

2 Statistical Learning

3 Deep Learning

4 Perspectives
- Algorithms
- Architectures

Algorithms: Beyond SGD

Problem	Techniques	Theory
A single step size for parameters with gradients of very different scales	Momentum, Adam / AdamW (today's default), warm-up and decay of the step size	Convergence of adaptive methods in non-convex optimization
Very deep networks: vanishing or exploding gradients	Residual connections, normalization layers (BatchNorm, LayerNorm), careful initialization	Signal propagation, geometry of the loss landscape
Overfitting	Weight decay, dropout, data augmentation, early stopping	Stability, implicit bias

Algorithms: Beyond SGD

Problem	Techniques	Theory
A single step size for parameters with gradients of very different scales	Momentum, Adam / AdamW (today's default), warm-up and decay of the step size	Convergence of adaptive methods in non-convex optimization
Very deep networks: vanishing or exploding gradients	Residual connections, normalization layers (BatchNorm, LayerNorm), careful initialization	Signal propagation, geometry of the loss landscape
Overfitting	Weight decay, dropout, data augmentation, early stopping	Stability, implicit bias

Optimizers remain an active research topic: new methods (*e.g.*, Lion, Shampoo, Muon) keep challenging Adam on large-scale training.

The Adam Algorithm

for $t = 1, \ldots, T$ **do**

compute $\mathbf{g}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$

$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$ ..... **momentum**: average of gradients

$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$

$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{m}_t / (\sqrt{\mathbf{v}_t} + \varepsilon)$

The Adam Algorithm

for $t = 1, \ldots, T$ **do**

compute $\mathbf{g}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$

$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$

$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$ ..... average of *squared* gradients

$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{m}_t / (\sqrt{\mathbf{v}_t} + \varepsilon)$

The Adam Algorithm

for $t = 1, \ldots, T$ **do**

compute $\mathbf{g}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$

$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\, \mathbf{g}_t$

$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\, \mathbf{g}_t^2$

$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\, \mathbf{m}_t / (\sqrt{\mathbf{v}_t} + \varepsilon)$ ..... one step size *per coordinate*

The Adam Algorithm

for $t = 1, \ldots, T$ **do**

$$\textbf{compute } \mathbf{g}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{m}_t / (\sqrt{\mathbf{v}_t} + \varepsilon)$$



The Adam Algorithm

for $t = 1, \ldots, T$ **do**

$$\textbf{compute } \mathbf{g}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\, \mathbf{g}_t$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\, \mathbf{g}_t^2$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\, \mathbf{m}_t / (\sqrt{\mathbf{v}_t} + \varepsilon)$$



On an elongated valley (same 14 steps), SGD zigzags across the steep direction and crawls along the flat one; Adam rescales each direction and heads to the minimum.

The Adam Algorithm

for $t = 1, \ldots, T$ **do**

$$\textbf{compute } \mathbf{g}_t = \nabla_{\mathbf{w}} \ell(\mathbf{w}^{(t)}, \boldsymbol{z}_t)$$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\, \mathbf{g}_t$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\, \mathbf{g}_t^2$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\, \mathbf{m}_t / (\sqrt{\mathbf{v}_t} + \varepsilon)$$



On an elongated valley (same 14 steps), SGD zigzags across the steep direction and crawls along the flat one; Adam rescales each direction and heads to the minimum.

AdamW adds a decoupled weight decay $-\eta\lambda\mathbf{w}^{(t)}$: a regularizer, as in RLM.

Outline

1 A Gentle Start

2 Statistical Learning

3 Deep Learning

4 Perspectives
- Algorithms
- Architectures

Architectures: Beyond MLP

An architecture encodes a symmetry of the data: the right symmetry reduces what has to be learned *(e.g., CNNs reach less than* 1% *error on MNIST, vs.* 1.5% *for our MLP)*.

Architecture	**Data**	**Encoded symmetry**	**In science**
CNN	Images, signals	Translation (shared local filters)	Medical imaging
GNN	Graphs, molecules, meshes	Node permutation (message passing)	Chemistry, weather forecasting
Transformer	Sequences, almost anything	Almost none (*attention* learns interactions)	Language, protein folding

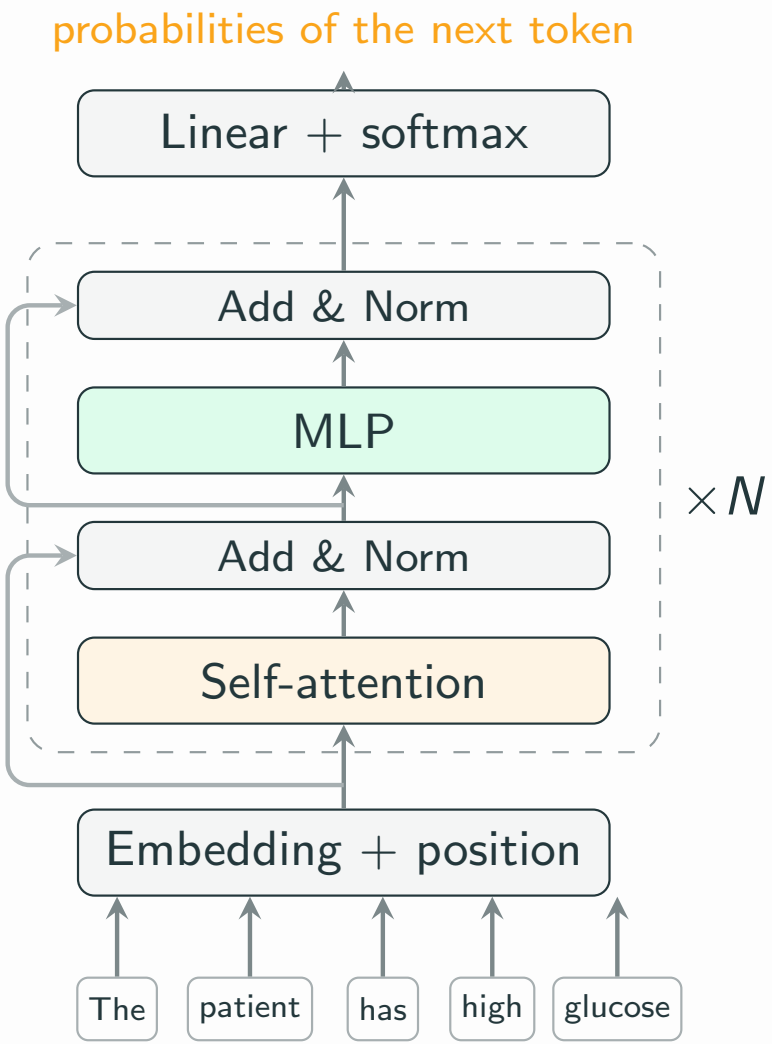
Architectures: Beyond MLP

An architecture encodes a symmetry of the data: the right symmetry reduces what has to be learned *(e.g., CNNs reach less than* 1% *error on MNIST, vs.* 1.5% *for our MLP)*.

Architecture	**Data**	**Encoded symmetry**	**In science**
CNN	Images, signals	Translation (shared local filters)	Medical imaging
GNN	Graphs, molecules, meshes	Node permutation (message passing)	Chemistry, weather forecasting
Transformer	Sequences, almost anything	Almost none (*attention* learns interactions)	Language, protein folding

Example: AlphaFold (Nobel Prize in Chemistry, 2024) predicts the 3D structure of proteins with attention and geometric biases.

The Transformer Architecture

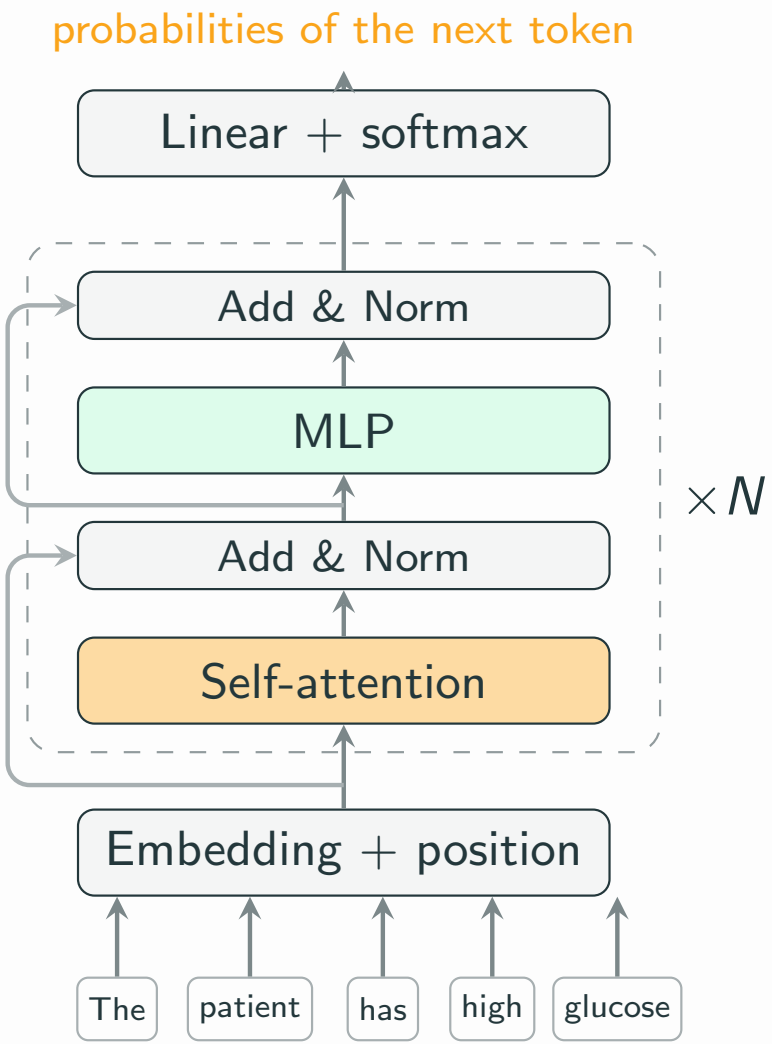


A transformer is a stack of N identical blocks, applied to a sequence of tokens (words, amino acids, image patches, ...).

Each token is first mapped to a vector, with its position in the sequence.

Trained to predict the next token, it needs no labels: this is self-supervision.

The Transformer Architecture

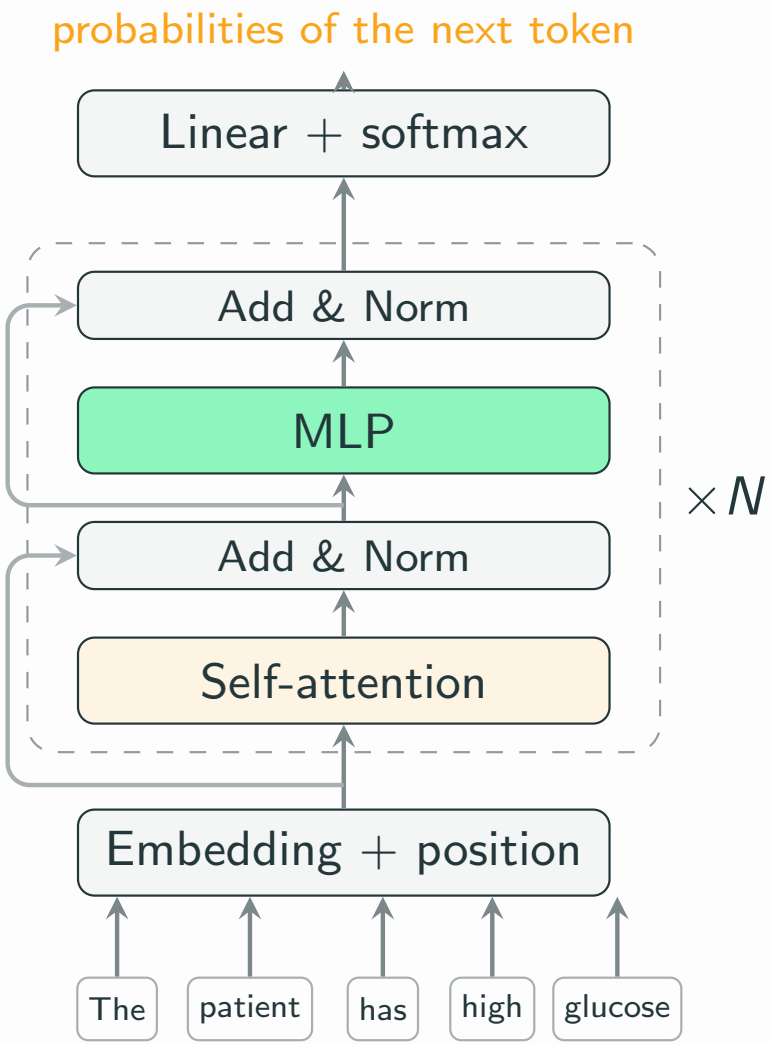


Self-attention mixes information *across* tokens: each token computes how relevant every other token is, and takes a weighted average:

$$\mathrm{softmax}\big(QK^\top/\sqrt{d}\,\big)\,V.$$

The weights are *learned from data*, not fixed by the architecture.

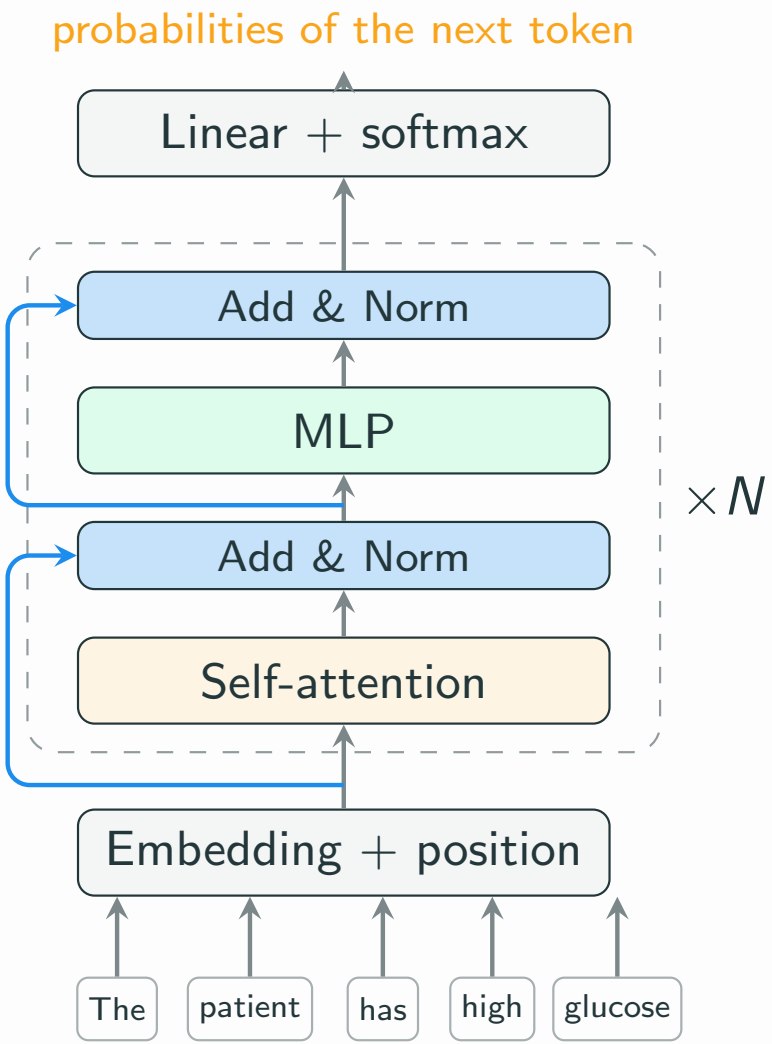
The Transformer Architecture



The **MLP** is exactly the network of this course: one hidden layer of ReLU-like units, applied to *each token separately*.

It holds most of the parameters of the model.

The Transformer Architecture



Residual connections and **normalization** keep the gradients under control through dozens of blocks.

Hence a transformer is: *attention* to mix tokens, *MLPs* to compute, and *stabilizers* to train deep.

Open Questions and Take-Home Messages

Open questions: implicit bias of SGD, tighter stability bounds, why architectures fit data so well.

Trustworthy deep learning: explainability, robustness, uncertainty, crucial in science.

Open Questions and Take-Home Messages

Open questions: implicit bias of SGD, tighter stability bounds, why architectures fit data so well.

Trustworthy deep learning: explainability, robustness, uncertainty, crucial in science.

Take-Home Messages

1. **Classical theory** (VC dimension, uniform convergence) tells us *when* learning is guaranteed, for *every* distribution.
2. **Stability** shows that generalization also depends on the *algorithm*: regularization and SGD *can* make learning stable, even with huge models.
3. **Architectures** that match the structure of the data make the difference in practice.

References: Statistical Learning

Textbook

S. Shalev-Shwartz and S. Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press. (Free online.)

PAC learning

- L. G. Valiant (1984). *A theory of the learnable*. Communications of the ACM, 27(11):1134–1142.
- D. Haussler (1992). *Decision theoretic generalizations of the PAC model for neural net and other learning applications*. Information and Computation, 100(1):78–150.

Uniform convergence and VC dimension

- V. N. Vapnik and A. Y. Chervonenkis (1971). *On the uniform convergence of relative frequencies of events to their probabilities*. Theory of Probability and its Applications, 16(2):264–280.
- A. Blumer, A. Ehrenfeucht, D. Haussler and M. K. Warmuth (1989). *Learnability and the Vapnik-Chervonenkis dimension*. Journal of the ACM, 36(4):929–965.

References: Deep Learning

Textbook

I. Goodfellow, Y. Bengio and A. Courville (2016). *Deep Learning*. MIT Press. (Free online.)

Expressivity

- G. Cybenko (1989). *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals and Systems, 2(4):303–314.
- T. Poggio et al. (2017). *Why and when can deep-but-not-shallow networks avoid the curse of dimensionality: a review*. International Journal of Automation and Computing, 14(5):503–519.

Sample and computational complexity

- P. L. Bartlett, N. Harvey, C. Liaw and A. Mehrabian (2019). *Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks*. Journal of Machine Learning Research, 20(63):1–17.
- A. L. Blum and R. L. Rivest (1992). *Training a 3-node neural network is NP-complete*. Neural Networks, 5(1):117–127.
- D. Bertschinger et al. (2023). *Training fully connected neural networks is* $\exists\mathbb{R}$*-complete*. Advances in Neural Information Processing Systems (NeurIPS).

References: Deep Learning

Stability

- S. Shalev-Shwartz, O. Shamir, N. Srebro and K. Sridharan (2010). *Learnability, stability and uniform convergence*. Journal of Machine Learning Research, 11:2635–2670.
- M. Hardt, B. Recht and Y. Singer (2016). *Train faster, generalize better: stability of stochastic gradient descent*. International Conference on Machine Learning (ICML).

SGD and backpropagation

- H. Robbins and S. Monro (1951). *A stochastic approximation method*. The Annals of Mathematical Statistics, 22(3):400–407.
- D. E. Rumelhart, G. E. Hinton and R. J. Williams (1986). *Learning representations by back-propagating errors*. Nature, 323:533–536.

Perspectives: Algorithms and architectures

- D. P. Kingma and J. Ba (2015). *Adam: a method for stochastic optimization*. International Conference on Learning Representations (ICLR).
- K. He, X. Zhang, S. Ren and J. Sun (2016). *Deep residual learning for image recognition*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- A. Vaswani et al. (2017). *Attention is all you need*. Advances in Neural Information Processing Systems (NeurIPS).
- J. Jumper et al. (2021). *Highly accurate protein structure prediction with AlphaFold*. Nature, 596:583–589.