

Lecture 9: Machine Learning II



Gabriel Poesia

Assistant Professor, CSE

University of Michigan, Ann Arbor

poesia@umich.edu

<https://gpoesia.com>

- 0. Administrivia
- 1. Recap
- 2. Linear classification
- 3. When does learning work?
- 4. Nonlinear features
- 5. Neural networks

Project proposals due tonight!

HW2 due next week (Wednesday 11:59pm)

RECAP

Supervised learning: learn a predictor f from (x, y) examples.

Three decisions: **hypothesis class** (e.g., $f_w(x) = w \cdot \phi(x)$), **loss function** (e.g., least squares), and **optimization** (e.g., SGD).

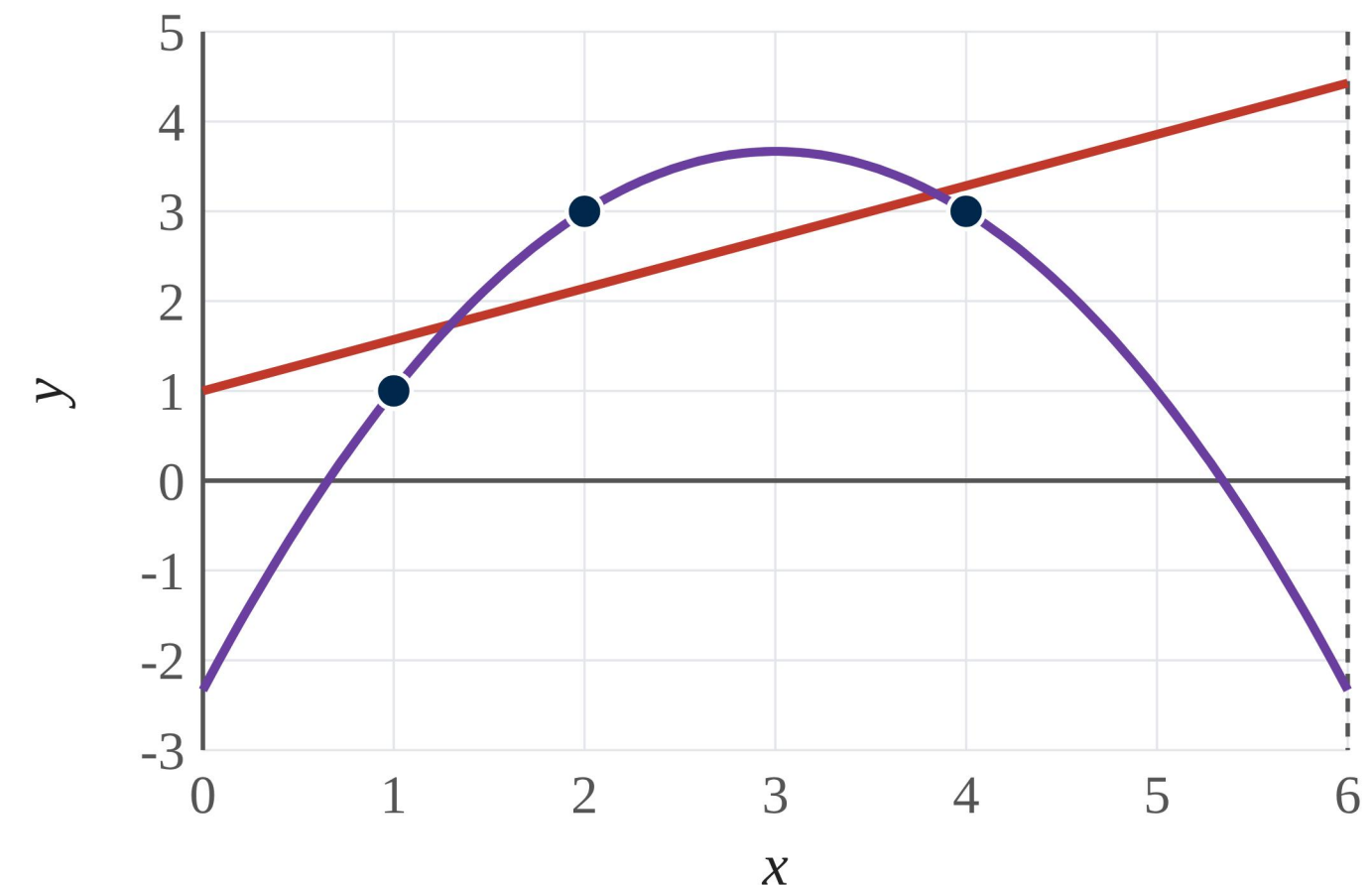
Binary classification: we can predict $\text{sign}(w \cdot \phi(x))$.

Counting mistakes (0-1 loss) has zero gradient almost everywhere.

Zero training loss isn't the goal: e.g., here the parabola **overfits**

Goal: low error on new examples,
 $\text{Err}(f) = \mathbb{E}_{(x,y) \sim \mathcal{D}} [\text{Loss}(x, y, f)]$.

Question we left: When does small training loss imply a small test error?



LINEAR CLASSIFICATION

A linear classifier

Last time: $y \in \{+1, -1\}$, and we predict the sign of the score.

Example: $x = [x_1, x_2]$, $\phi(x) = x$.

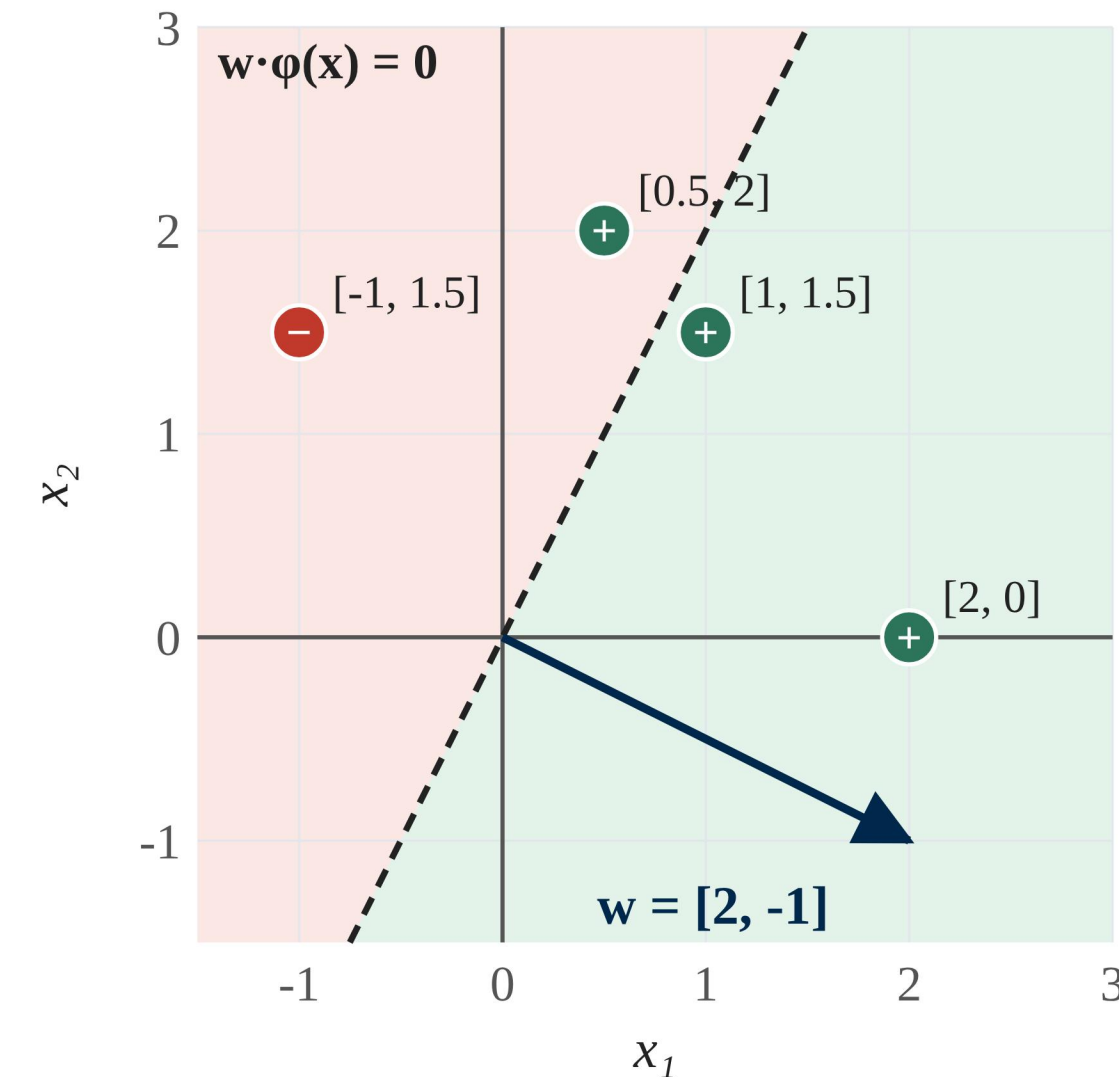
$$f_w(x) = \text{sign}(w \cdot \phi(x))$$

Say $w = [2, -1]$. $x = [2, 0]$: score 4 $\rightarrow +1$. $x = [0.5, 2]$: score $-1 \rightarrow -1$.

Decision boundary: where the score is 0.

A line perpendicular to w . The side w points to is $+1$.

$\mathcal{F} = \{f_w : w \in \mathbb{R}^2\}$: all lines through the origin.



Score and margin

DEFINITION (SCORE AND MARGIN)

The **score** $w \cdot \phi(x)$ is how confidently we predict $+1$.

The **margin** $(w \cdot \phi(x)) y$ is how correct we are.

With $w = [2, -1]$:

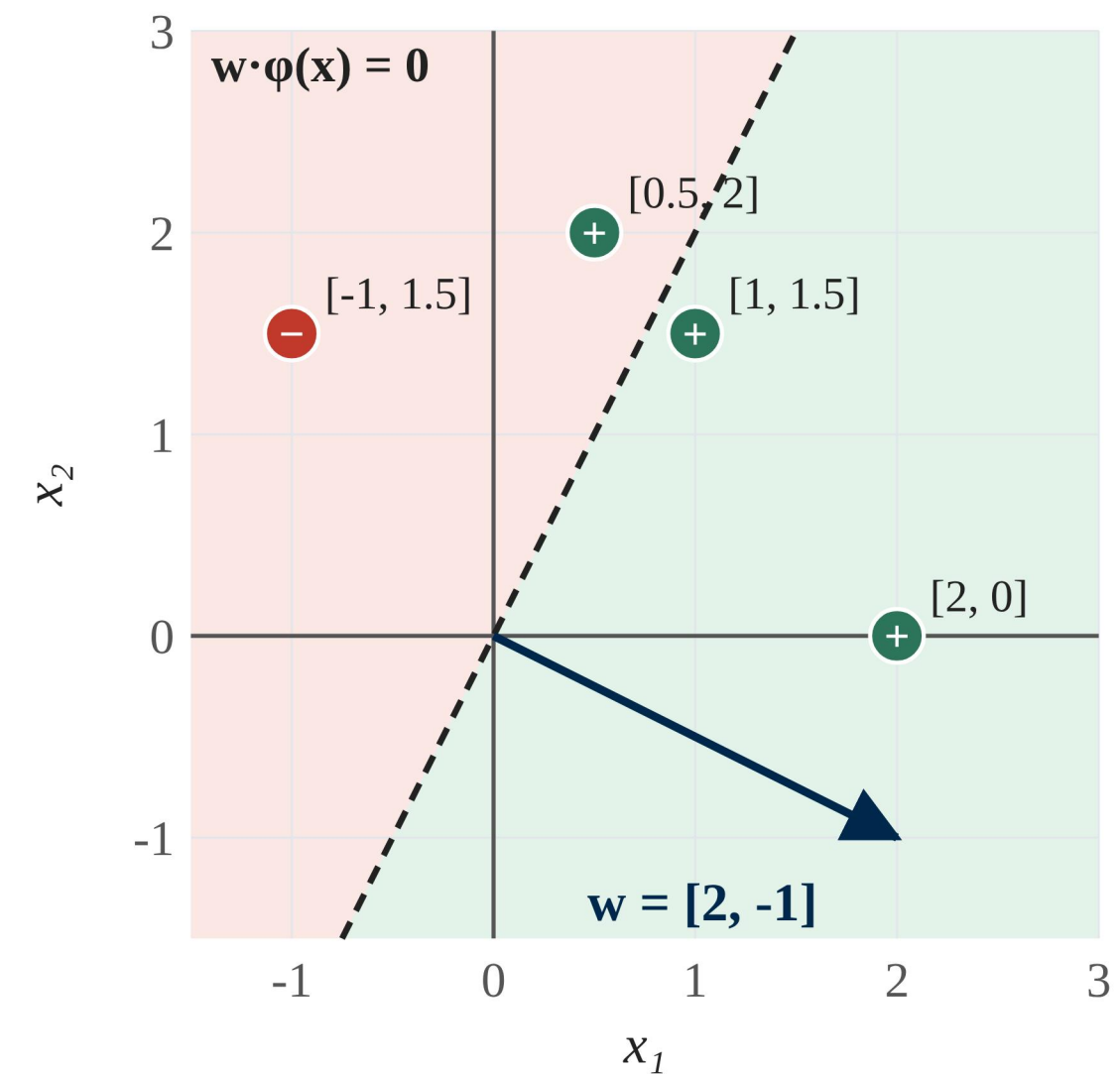
x	y	Score	Margin
$[2, 0]$	$+1$	4	4
$[1, 1.5]$	$+1$	0.5	0.5
$[0.5, 2]$	$+1$	-1	-1
$[-1, 1.5]$	-1	-3.5	3.5

Margin > 0 : correct. Margin < 0 : mistake.

Large margin $\rightarrow$ correct & far from the boundary.

Zero-one loss: $\mathbf{1}[f_w(x) \neq y] = \mathbf{1}[(w \cdot \phi(x)) y \leq 0]$

Here: $\text{TrainLoss}_{0-1}(w) = \frac{1}{4}$



Hinge loss

Zero-one loss vs. margin is a step function.

Gradient 0 $\rightarrow$ gradient descent / SGD won't move.

Fix: optimize a loss that tells us which way to go.

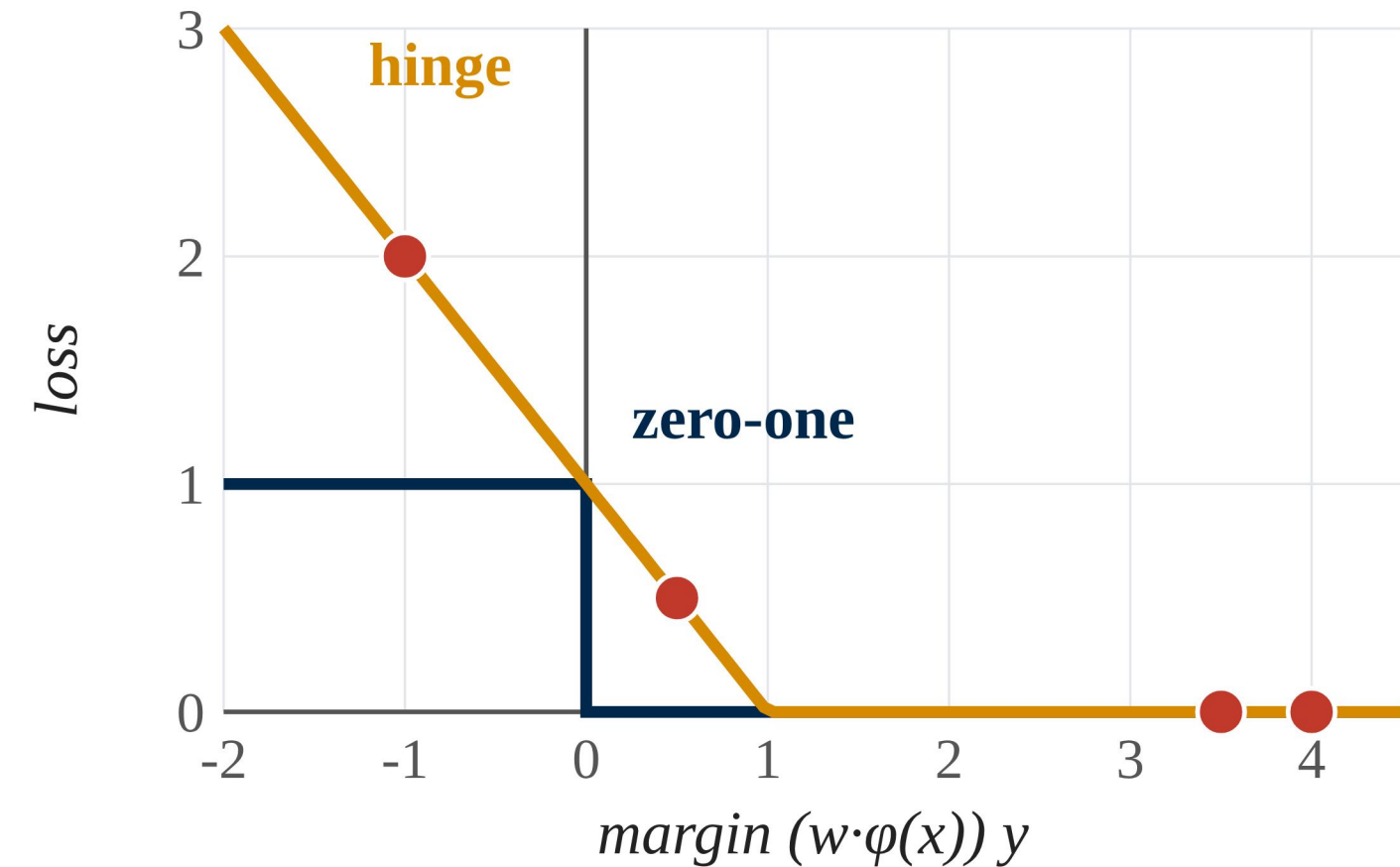
$$\text{Loss}_{\text{hinge}}(x, y, w) = \max\{1 - (w \cdot \phi(x)) y, 0\}$$

Margin ≥ 1 : zero loss. Below: grows linearly.

Hinge $\geq$ zero-one. So hinge = 0 $\Rightarrow$ no training errors.

On our examples (margins 4, 0.5, -1, 3.5):

$$\begin{aligned} \max\{1 - 4, 0\} &= 0 & \max\{1 - 0.5, 0\} &= 0.5 \\ \max\{1 + 1, 0\} &= 2 & \max\{1 - 3.5, 0\} &= 0 \end{aligned}$$



Gradient of the hinge loss

Margin < 1 : loss is $1 - (w \cdot \phi(x)) y$

In that case, gradient is $-y \nabla_w (w \cdot \phi(x)) = -\phi(x) y$

Margin ≥ 1 : loss is 0, gradient 0.

$$\nabla_w \text{Loss}_{\text{hinge}}(x, y, w) = \begin{cases} -\phi(x) y & \text{if } (w \cdot \phi(x)) y < 1 \\ 0 & \text{otherwise} \end{cases}$$

SGD step on an example with margin < 1 :

$$w \leftarrow w - \eta \nabla_w \text{Loss}_{\text{hinge}} = w - \eta(-\phi(x) y) = w + \eta y \phi(x)$$

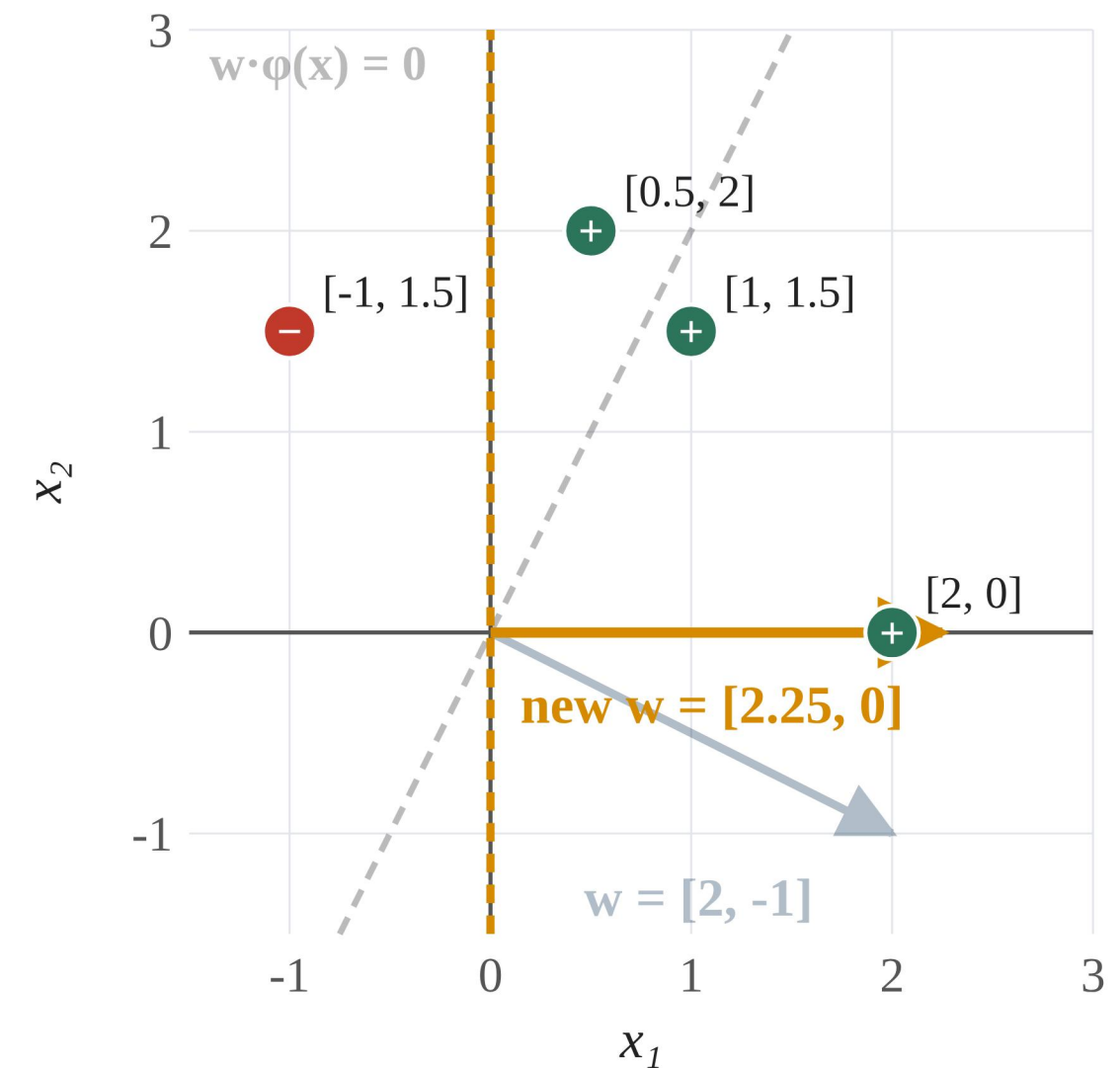
Moves w towards $\phi(x)$ if $y = +1$, away if $y = -1$.

Our mistake, $([0.5, 2], +1)$, with $\eta = 0.5$:

$$\begin{aligned} w &\leftarrow [2, -1] + 0.5 \cdot (+1) \cdot [0.5, 2] = [2, -1] + [0.25, 1] \\ &= [2.25, 0] \end{aligned}$$

After this update, hinge loss is 0

(Updating only on mistakes, i.e. margin ≤ 0 gives the classical perceptron algorithm)



Predicting probabilities

Sometimes we want more than a label: how likely is $y = +1$?

But the score is any real number, and a probability is in $[0, 1]$.

Idea: let the score be the **log-odds**:

$$\log \frac{P(y=+1|x)}{P(y=-1|x)} = w \cdot \phi(x)$$

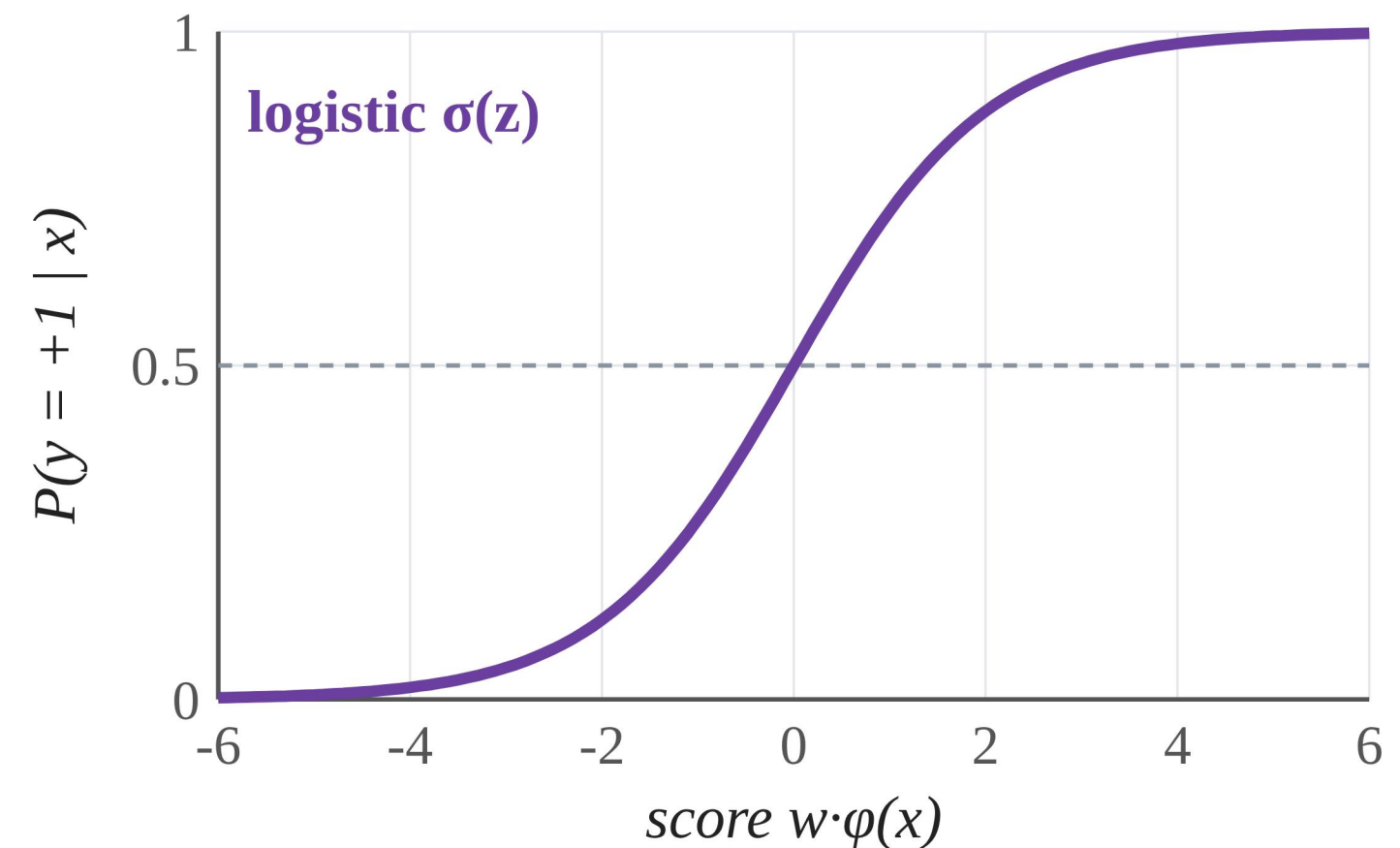
Solve for $p = P(y = +1 \mid x)$: $\frac{p}{1-p} = e^{w \cdot \phi(x)}$

$$p = \frac{1}{1+e^{-w \cdot \phi(x)}} = \sigma(w \cdot \phi(x))$$

The **logistic** (sigmoid) function σ squashes scores into $[0, 1]$.

And $P(y = -1 \mid x) = 1 - p = \sigma(-w \cdot \phi(x))$. So:

$$P_w(y \mid x) = \sigma((w \cdot \phi(x)) y) = \sigma(\text{margin})$$



If our model gives probabilities (given w), how do we choose the best model?

We can do maximum likelihood estimation, like before!
Minimize $-\log P_w(y \mid x)$.

With margin $m = (w \cdot \phi(x)) y$:

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

This is the **logistic loss**. As a function of the margin:

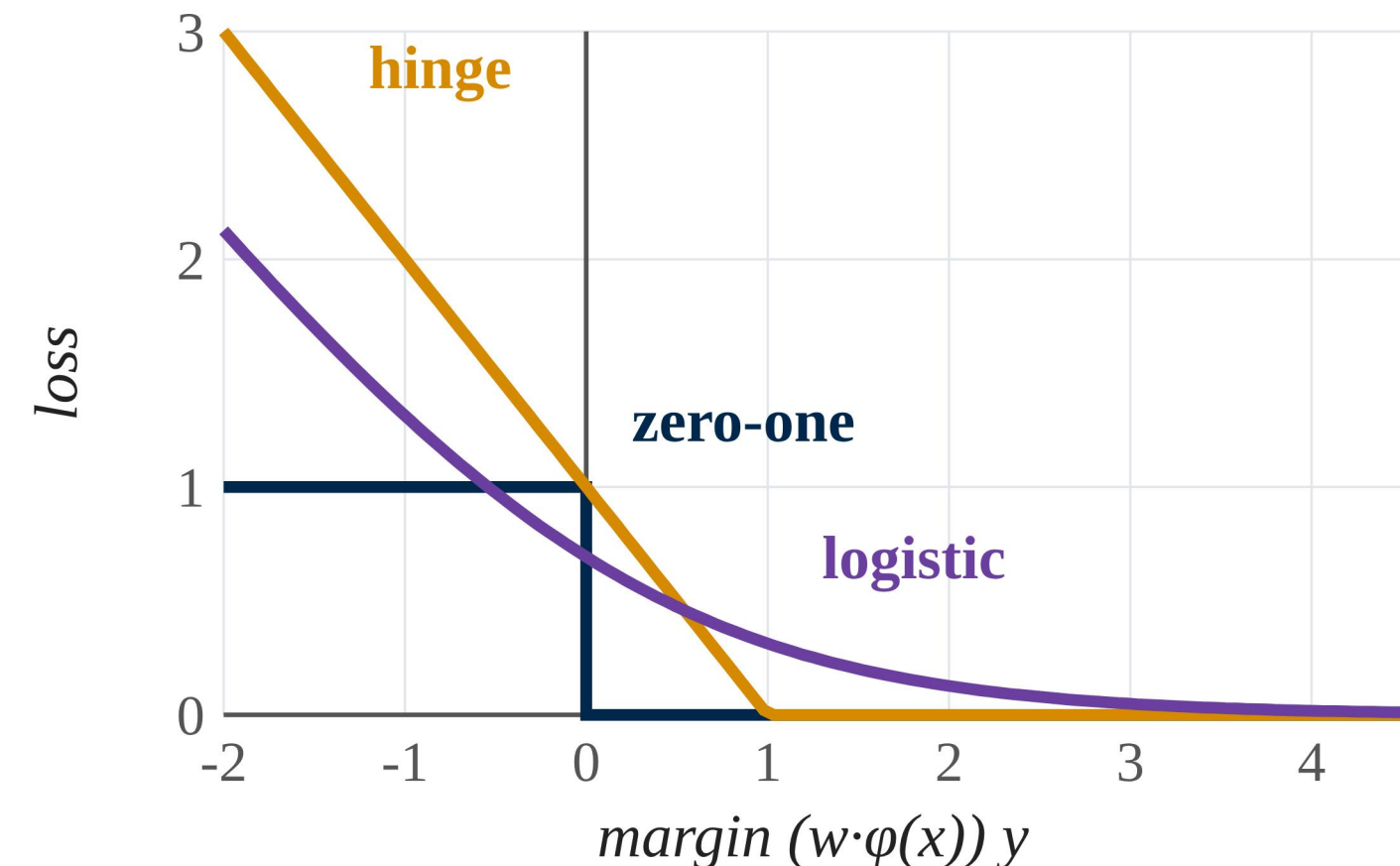
Like hinge: large for mistakes, small for large margins.

Unlike hinge: never 0. Always some push to increase the margin.

You'll derive the gradient and train this classifier in Homework 2.

Extends easily to K classes (softmax + cross-entropy).

That's the standard loss for neural classifiers. And for LLMs: next-token prediction!



Regression vs. classification

	Regression	Classification
Score	$w \cdot \phi(x)$	$w \cdot \phi(x)$
Prediction $f_w(x)$	the score	sign of the score
Compare with y	residual: score $- y$	margin: score $\times y$
Loss	squared, absolute deviation	zero-one (what we care about), hinge, logistic (easier to optimize for)
Optimization	(stochastic) gradient descent	(stochastic) gradient descent

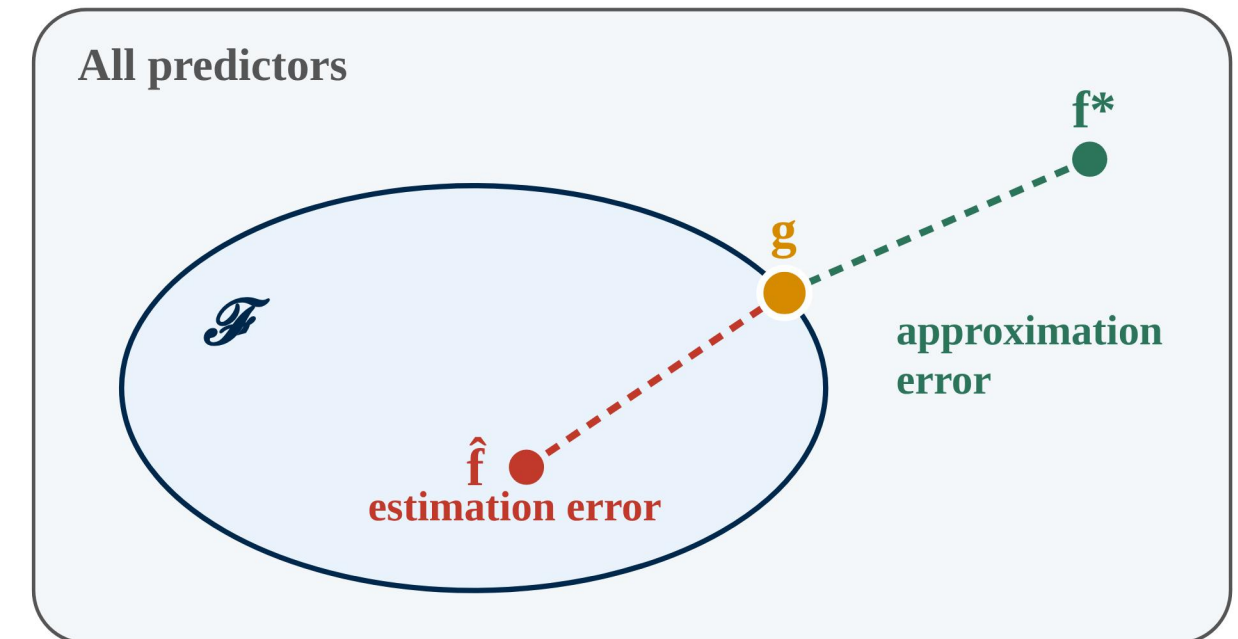
**WHEN DOES LEARNING
WORK?**

Two sources of error

Learning fails when we pick a predictor with high error on new examples.
Where does error come from?

Let's compare three predictors:

- f^* : the best possible predictor (lowest Err of any function)
- g : the best predictor in our hypothesis class $\mathcal{F}$
- $\hat{f}$: what we learn from the training data (e.g., lowest training loss in $\mathcal{F}$)



$$\begin{aligned}\text{Err}(\hat{f}) - \text{Err}(f^*) &= \text{Err}(\hat{f}) - \text{Err}(g) + \text{Err}(g) - \text{Err}(f^*) \\ &= \underbrace{\text{Err}(\hat{f}) - \text{Err}(g)}_{\text{estimation error}} + \underbrace{\text{Err}(g) - \text{Err}(f^*)}_{\text{approximation error}}\end{aligned}$$

Approximation error: how good is the best in $\mathcal{F}$? Only depends on $\mathcal{F}$.

Estimation error: how well did we pick within $\mathcal{F}$, from limited data?

How big should the hypothesis class be?

What if we make $\mathcal{F}$ larger (more features, ...)?

Approximation error goes down: a larger class has a better best predictor.

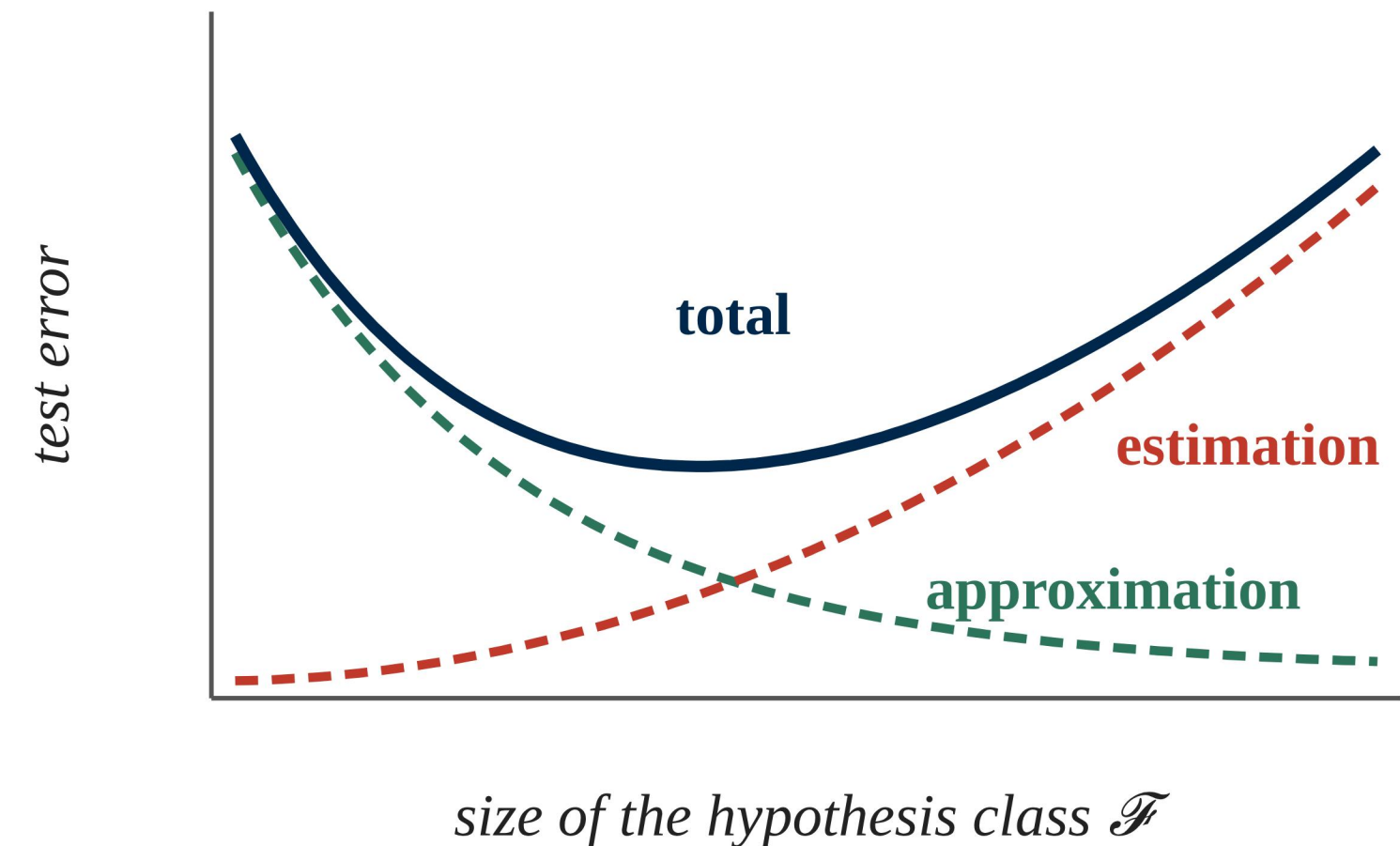
Estimation error goes up: more predictors that do well in training but don't generalize.

The sweet spot depends on how much data we have.

Keeping $\mathcal{F}$ small: fewer features, or penalizing large weights (**regularization**).

How do we reduce estimation error?

Intuitively: more compute (try hard to find best predictor) / more data (try hard to rule out bad predictors)



Empirical risk minimization

How much data do we need, for a given $\mathcal{F}$?

Let's analyze the idealized learning algorithm we've been using: lowest training loss in $\mathcal{F}$.

DEFINITION (EMPIRICAL RISK MINIMIZATION (ERM))

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \text{TrainLoss}(f)$$

Recall: the expected loss $\text{Err}(f)$ is the *risk*. The training loss is the *empirical* risk.

For a *fixed* f : $\text{TrainLoss}(f)$ averages n independent samples of the loss, each with mean $\text{Err}(f)$.

Law of large numbers: $\text{TrainLoss}(f) \rightarrow \text{Err}(f)$ as $n \rightarrow \infty$.

Q: So $\text{TrainLoss}(\hat{f}) \approx \text{Err}(\hat{f})$?

A: Not necessarily! $\hat{f}$ isn't fixed: we chose it *because* its training loss is low.

Lucky predictors

Say $\mathcal{F}$ has 1000 predictors, and each one guesses at random: 50% error on new examples.

We have $n = 10$ training examples.

One predictor gets all 10 right with probability $2^{-10} \approx .001$.

Q: What's the chance that *some* predictor gets all 10 right?

$$1 - (1 - 2^{-10})^{1000} \approx .62$$

ERM picks it: training loss 0, but 50% error on new examples!

More predictors $\rightarrow$ more likely that a bad one looks good by chance.

ERM will work well if *no* bad predictor looks good: $\text{TrainLoss}(f) \approx \text{Err}(f)$ for *all* $f \in \mathcal{F}$ (uniform convergence).

Then, whichever f ERM picks, we can trust its training loss. How much data does that take?

A simple setting

Let's answer this in the simplest setting:

- Binary classification, with the zero-one loss
- $\mathcal{F}$ is finite
- Some $f \in \mathcal{F}$ is perfect: $\text{Err}(f) = 0$ (the "realizable" case)
- The n training examples are drawn iid from $\mathcal{D}$

Here, ERM always finds some $\hat{f}$ with $\text{TrainLoss}(\hat{f}) = 0$. (The perfect one has zero training loss.)

But other predictors might have zero training loss too.

Q: Could ERM pick one that's bad on new examples?

A: With enough data, **probably not!**

Probably approximately correct

What can we guarantee?

We'll prove that ERM gives us a classifier that is:

- **Approximately correct:** $\text{Err}(\hat{f}) \leq \epsilon$.
- **With high probability:** with probability $\geq 1 - \delta$

More precisely, given a desired error rate ϵ and a desired probability $1 - \delta$, we can find a minimum number of training samples n to get this guarantee from ERM

This is the **Probably Approximately Correct (PAC) learning** framework, introduced by Leslie Valiant in 1984 (Turing Award in 2010)

How many examples n do we need, for a given ϵ , δ and $\mathcal{F}$?

THEOREM (PAC LEARNING FOR FINITE HYPOTHESIS CLASSES)

In our simple setting, for any $\epsilon, \delta \in (0, 1)$: if $n \geq \frac{1}{\epsilon} (\ln |\mathcal{F}| + \ln \frac{1}{\delta})$, then with probability at least $1 - \delta$, $\text{Err}(\hat{f}) \leq \epsilon$.

Proof (1/2): when can ERM go wrong?

Call f **bad** if $\text{Err}(f) > \epsilon$.

ERM only returns predictors with zero training loss (we assumed the realizable case, some perfect hypothesis in $\mathcal{F}$)

So it can only go wrong if some bad f has $\text{TrainLoss}(f) = 0$.

Fix one bad f . On one random example, it's correct with probability $1 - \text{Err}(f) < 1 - \epsilon$.

Examples are independent, so f gets all n right with probability:

$$P(\text{TrainLoss}(f) = 0) < (1 - \epsilon)^n$$

$(1 - \epsilon)^n$ is awkward to solve for n . Since $1 - \epsilon \leq e^{-\epsilon}$:

$$(1 - \epsilon)^n \leq e^{-\epsilon n}$$

There are at most $|\mathcal{F}|$ bad predictors. Union bound:

$$P(\text{ERM can go wrong}) \leq P(\text{some bad } f \text{ has } \text{TrainLoss}(f) = 0) \leq |\mathcal{F}| e^{-\epsilon n}$$

Proof (2/2): how much data?

So far: $P(\text{ERM can go wrong}) \leq |\mathcal{F}| e^{-\epsilon n}$.

This decays exponentially with n . We want it to be at most δ . How large does n need to be?

$$\begin{aligned} |\mathcal{F}| e^{-\epsilon n} &\leq \delta \\ \iff e^{-\epsilon n} &\leq \frac{\delta}{|\mathcal{F}|} && \text{divide by } |\mathcal{F}| \\ \iff -\epsilon n &\leq \ln \delta - \ln |\mathcal{F}| && \text{take } \ln \text{ (increasing)} \\ \iff \epsilon n &\geq \ln |\mathcal{F}| + \ln \frac{1}{\delta} && \text{multiply by } -1 \\ \iff n &\geq \frac{1}{\epsilon} \left(\ln |\mathcal{F}| + \ln \frac{1}{\delta} \right) && \text{divide by } \epsilon > 0 \end{aligned}$$

$\ln |\mathcal{F}|$: enough data to rule out every bad predictor. $\ln \frac{1}{\delta}$: the confidence we want.

So with this much data, with probability $\geq 1 - \delta$, no bad predictor has zero training loss.

$\hat{f}$ has zero training loss, so it's not bad: $\text{Err}(\hat{f}) \leq \epsilon$. ■

How much data do we need?

$$n \geq \frac{1}{\epsilon} \left(\ln |\mathcal{F}| + \ln \frac{1}{\delta} \right)$$

- Only logarithmic in $|\mathcal{F}|$: $10^6 \times$ more predictors costs $\approx 14/\epsilon$ more examples.
- High confidence is cheap: $\ln \frac{1}{\delta}$.
- E.g., $|\mathcal{F}| = 10^6$, $\epsilon = 0.05$, $\delta = 0.01$: 369 examples suffice.

Our memorizer from last time? If $\mathcal{F}$ has every function of x , $\ln |\mathcal{F}|$ grows with the number of possible inputs. We'd have to see almost all of them!

What about infinite classes?

Lines are an infinite class. Is the bound useless for them?

Trick: on a computer, each weight is a 32-bit float. d weights $\rightarrow$ at most 2^{32d} predictors:

$$\ln |\mathcal{F}| \leq 32d \ln 2 \approx 22d$$

So n grows *linearly* with the number of features: $n \geq \frac{1}{\epsilon} (22d + \ln \frac{1}{\delta})$.

The classical picture: more features $\rightarrow$ bigger $\mathcal{F} \rightarrow$ more data needed.

This is the simplest result in **statistical learning theory**. It extends in many directions:

- **No perfect predictor in $\mathcal{F}$** (non-realizable): similar argument. Estimation error shrinks like $\sqrt{\ln |\mathcal{F}|/n}$.
- **Better measures of complexity** than counting predictors: VC dimension (Vapnik & Chervonenkis, 1971), Rademacher complexity.
- **Regularization**: bounds that depend on the size of the weights, not just how many there are.

Neural networks often have more parameters than training examples. These bounds say little about them.

Yet they generalize! More in Module 3.

NONLINEAR FEATURES

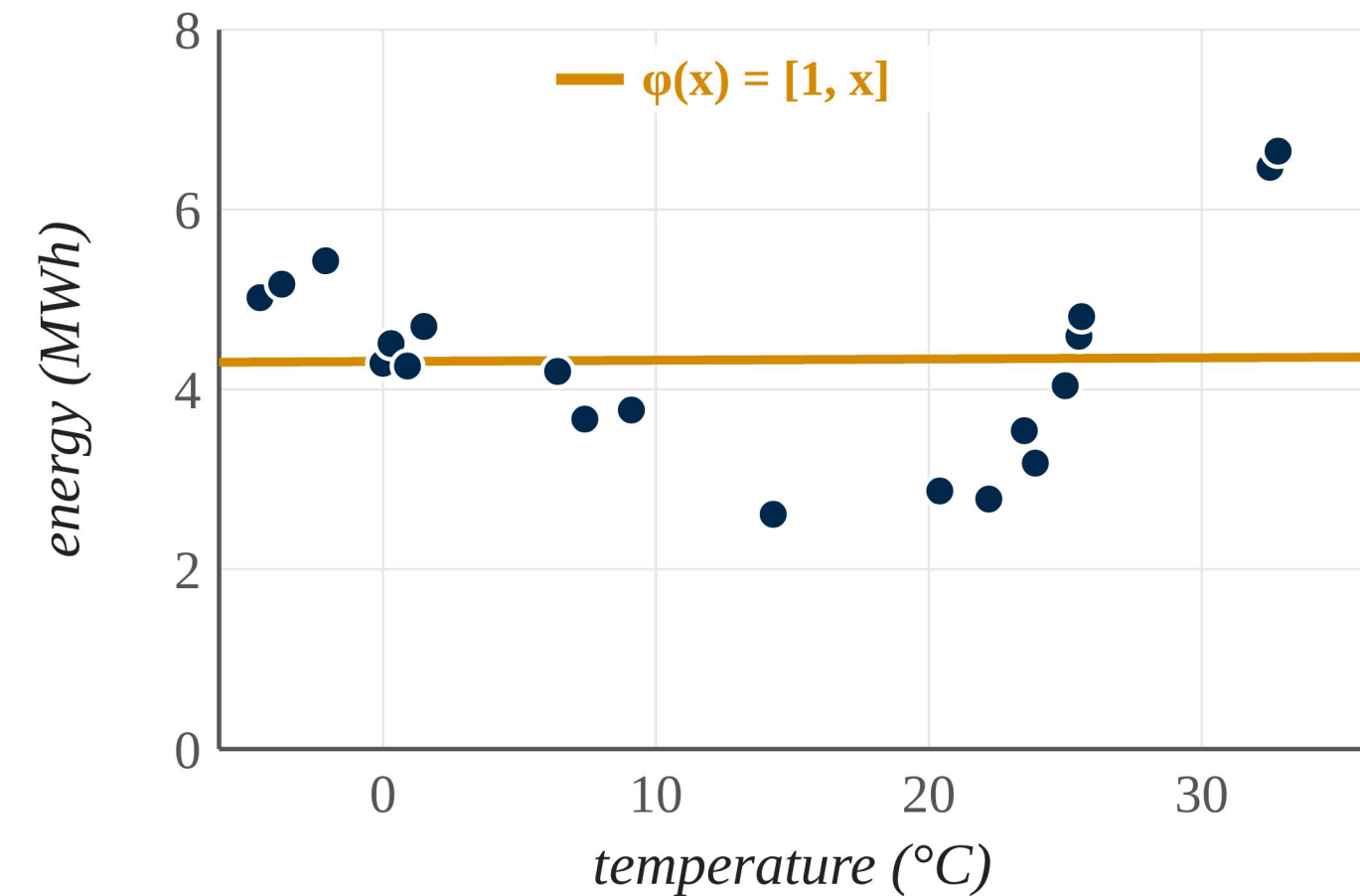
When the truth isn't a line

Let's predict a building's daily energy use from the outside temperature.

20 days of data. Cold days: heating. Hot days: cooling.

The best line is basically flat!

Would more data help?



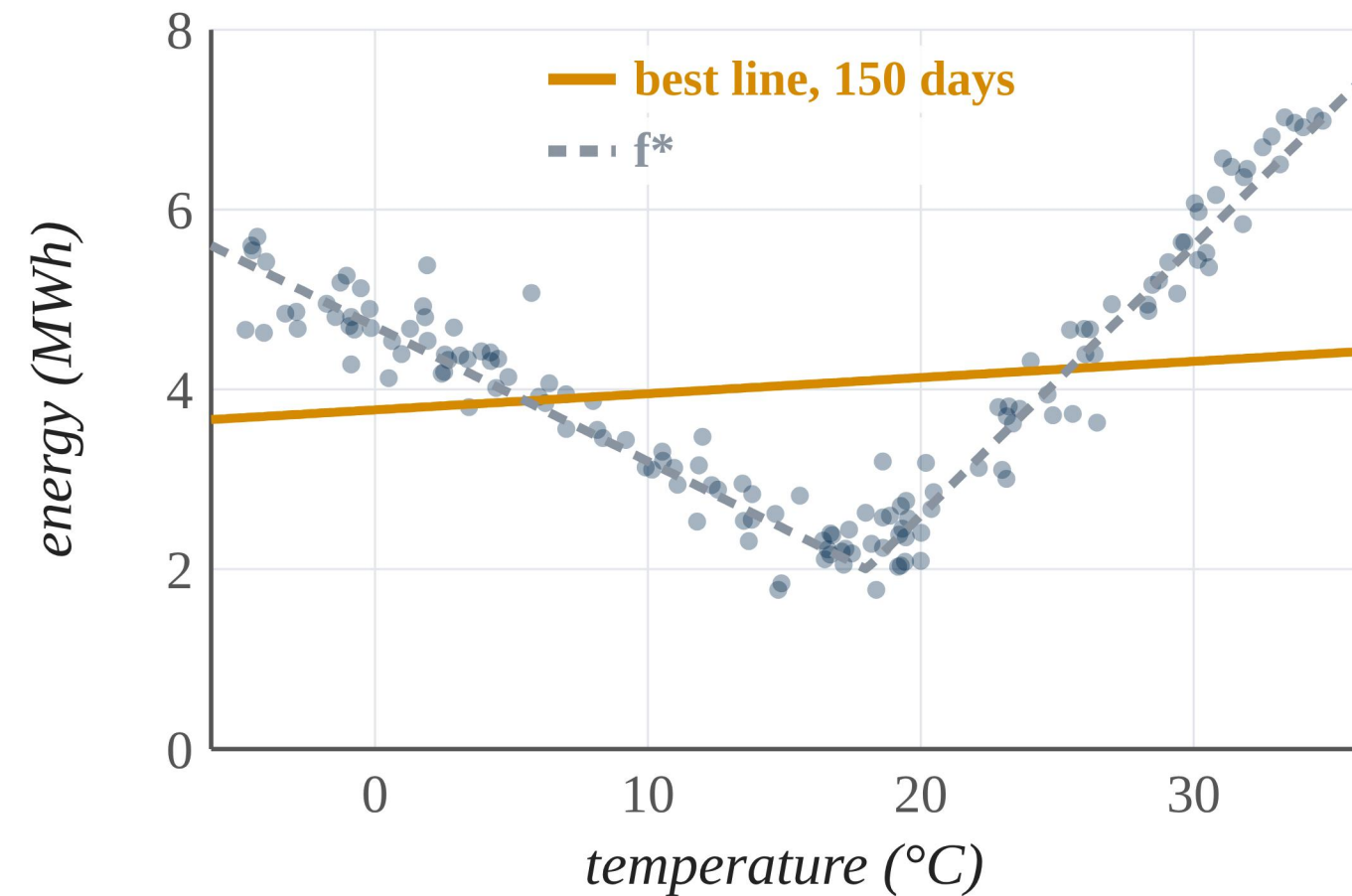
More data doesn't help

With 150 days, the best line is still bad.

The data comes from a curve f^* , plus noise. No line is close to it.

This is **approximation error**: more data won't fix it.

We need a bigger $\mathcal{F}$. Do we have to give up on linear predictors?



Quadratic features

Recall: $f_w(x) = w \cdot \phi(x)$, and we get to choose ϕ .

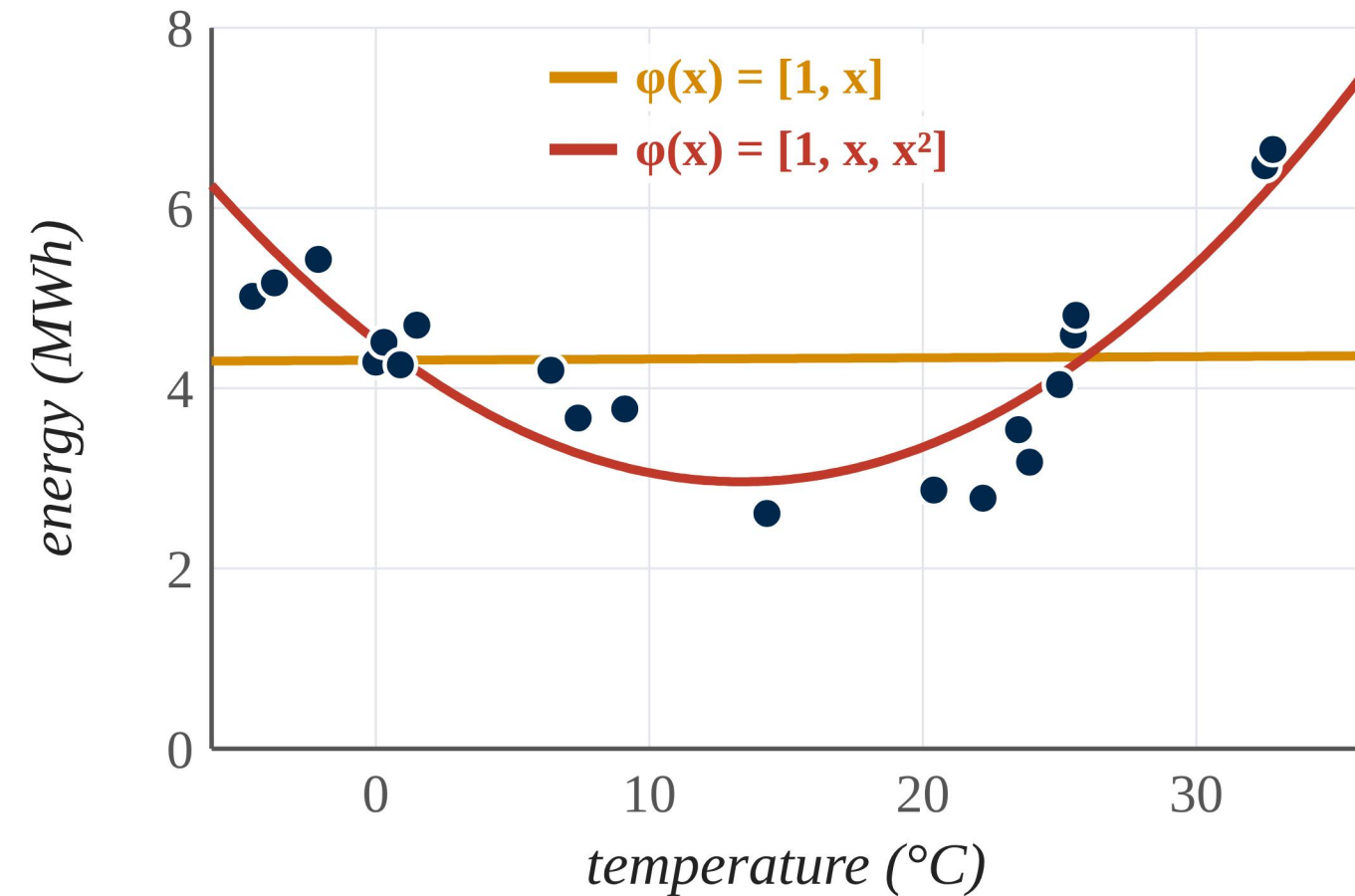
Let's add a feature: $\phi(x) = [1, x, x^2]$.

Now $f_w(x) = w_1 + w_2x + w_3x^2$: a parabola.

Same loss, same gradient descent. Only ϕ changed.

Includes every line ($w_3 = 0$), so it fits at least as well.

Last time, this ϕ overfit 3 points. With 20, it looks reasonable.



Conditional features

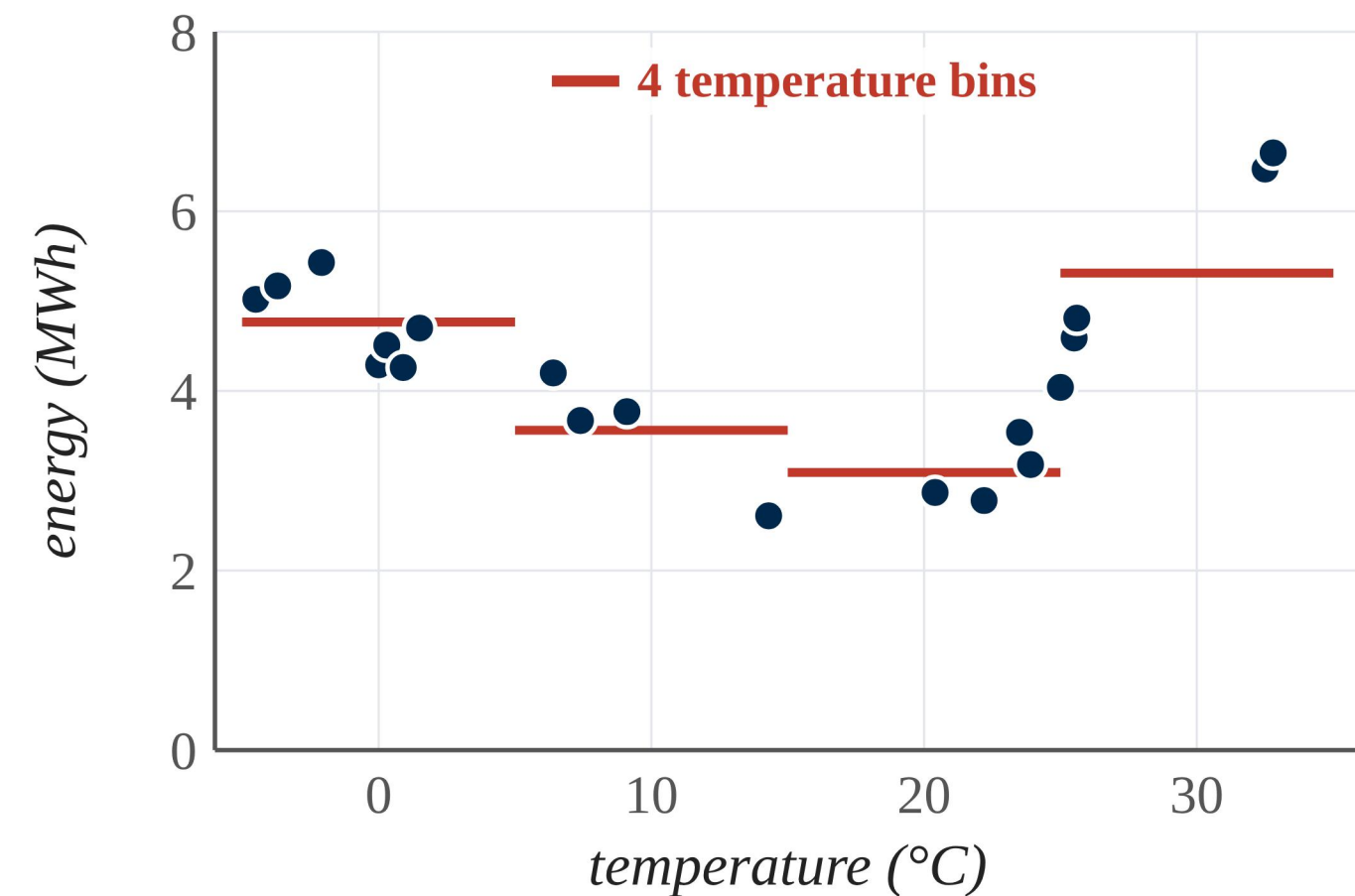
A parabola is symmetric around its minimum. Is that reasonable here?

Features can be anything we compute from x , including conditions:

$$\phi(x) = [\mathbf{1}[x < 5], \mathbf{1}[5 \leq x < 15], \\ \mathbf{1}[15 \leq x < 25], \mathbf{1}[x \geq 25]]$$

E.g., $\phi(20) = [0, 0, 1, 0]$. Exactly one is 1: the score is the weight of x 's bin.

We get a piecewise constant predictor.



Using what we know

Suppose an engineer tells us: system starts heating below 18° C and cooling above, and energy for each is roughly linear in °C.

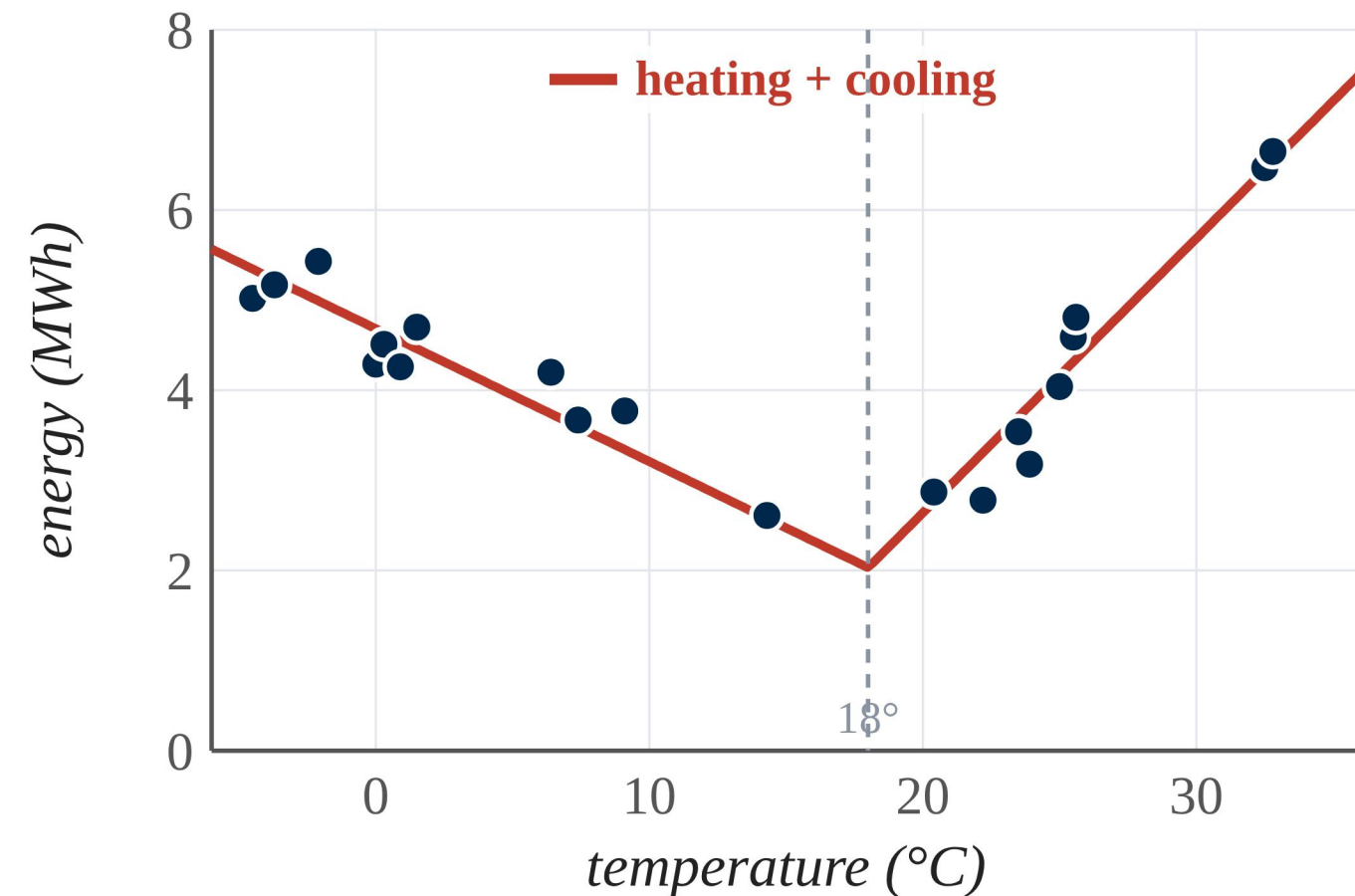
We could then use features like:

$$\phi(x) = [1, \max(18 - x, 0), \max(x - 18, 0)]$$

Least squares now fits the data very well.

Good features encode what we know about the problem.

A useless feature can just get a weight close to 0.



Linear in what?

We call these "linear predictors". But we just fit curves & other weird shapes!

$$f_w(x) = w \cdot \phi(x)$$

Linear in w ?	Yes
Linear in $\phi(x)$?	Yes
Linear in x ?	No! (x might not even be a vector)

E.g., spam filter: x is an email. Features: contains “free money”, number of links, sender is a contact, ...

Expressivity: through ϕ , the score can be very nonlinear in x .

Efficiency: the score is always linear in w . Learning only sees $\phi(x)$.

Squared, hinge, logistic losses: $\text{TrainLoss}(w)$ is **convex**. Gradient descent (with suitable step sizes) finds a global minimum.

Nonlinear decision boundaries

Same idea for classification. Take $x = [x_1, x_2]$ and

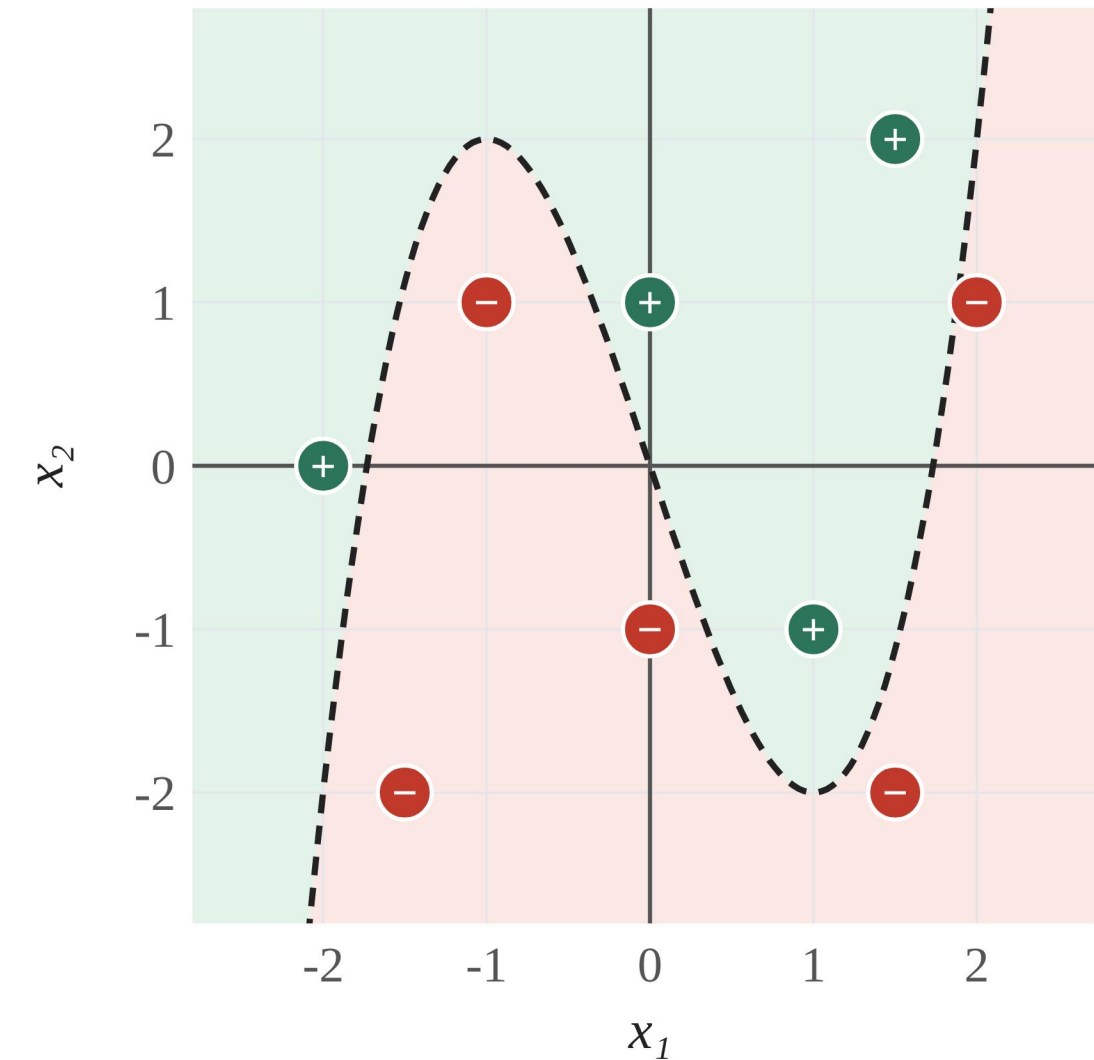
$$\phi(x) = [x_1, x_1^3, x_2], \quad w = [3, -1, 1]$$

Score: $3x_1 - x_1^3 + x_2 = x_2 - (x_1^3 - 3x_1)$

Predict +1 above the cubic $x_2 = x_1^3 - 3x_1$, and -1 below.

In feature space $\mathbb{R}^3$: the boundary is a plane.

In input space: a cubic curve.



Who chooses the features?

Designing ϕ is called **feature engineering**. For decades, it was much of the work in applied ML.
More features: bigger $\mathcal{F}$.

- Lower approximation error, if the features are useful
- But more data needed: in our bound, n grows like $22d/\epsilon$ for d features

All quadratic features $x_i x_j$ of k inputs: already $\approx k^2/2$.

We want a few good features. That takes knowledge about the problem.

What are good features for an image (a big matrix of pixels)?

Hard to say! Edges? Textures? Object parts? How do we detect them?

Can we learn the features from data, too?

NEURAL NETWORKS

Predicting car collisions

Two cars at positions x_1 and x_2 . Safe ($y = +1$), or collision ($y = -1$)?

Say: safe iff at least 1 apart. $y = \text{sign}(|x_1 - x_2| - 1)$

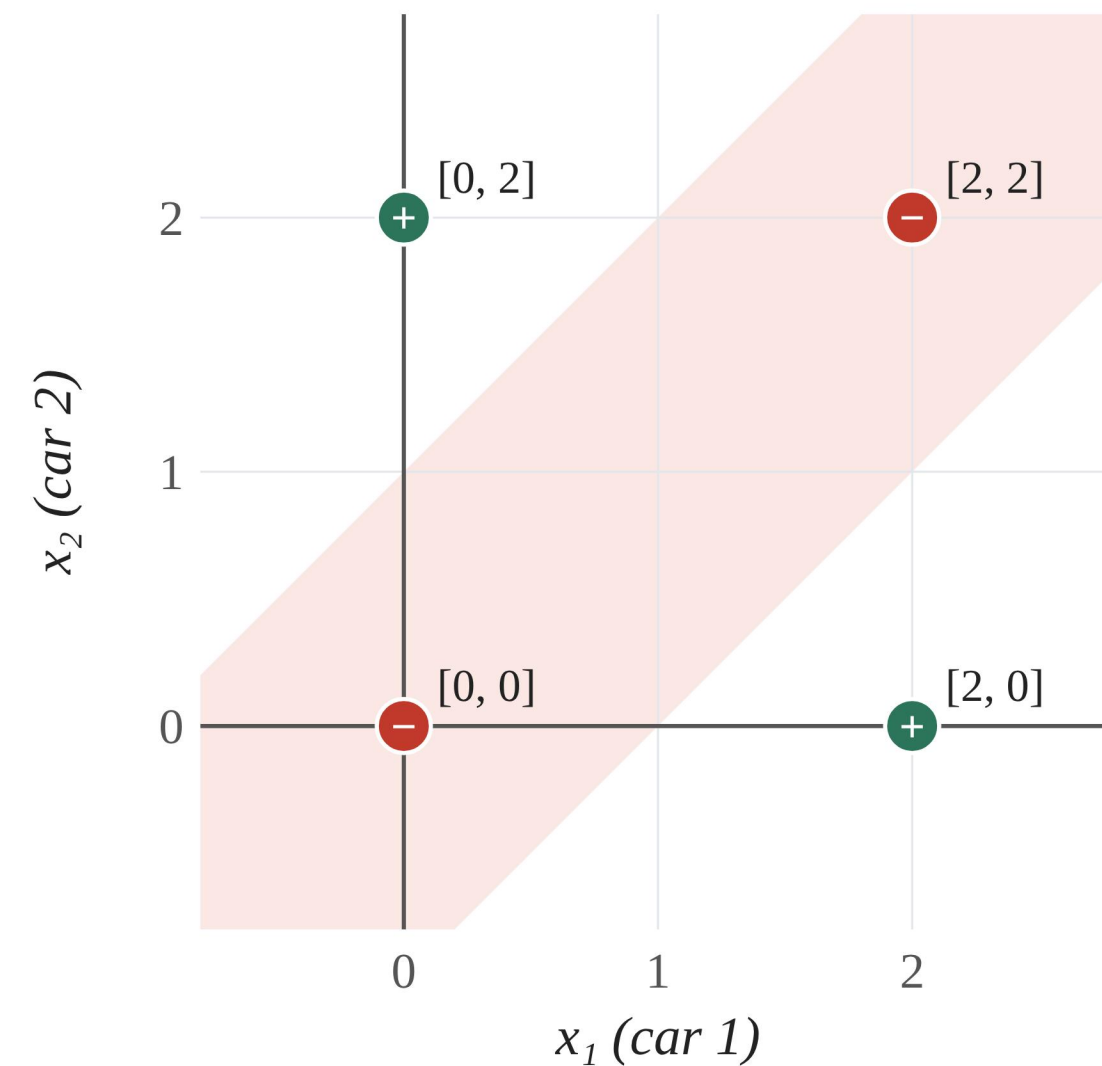
The learner doesn't know this, it only sees examples.

Q: Can a linear classifier with $\phi(x) = [1, x_1, x_2]$ fit these 4 examples?

A: No!

We could with feature engineering (e.g., $|x_1 - x_2|$), but let's avoid that

Let's start with just the "raw" inputs $\phi(x)$ above and try to combine linear pieces.



Decomposing the problem

Let's break it into two simpler questions.

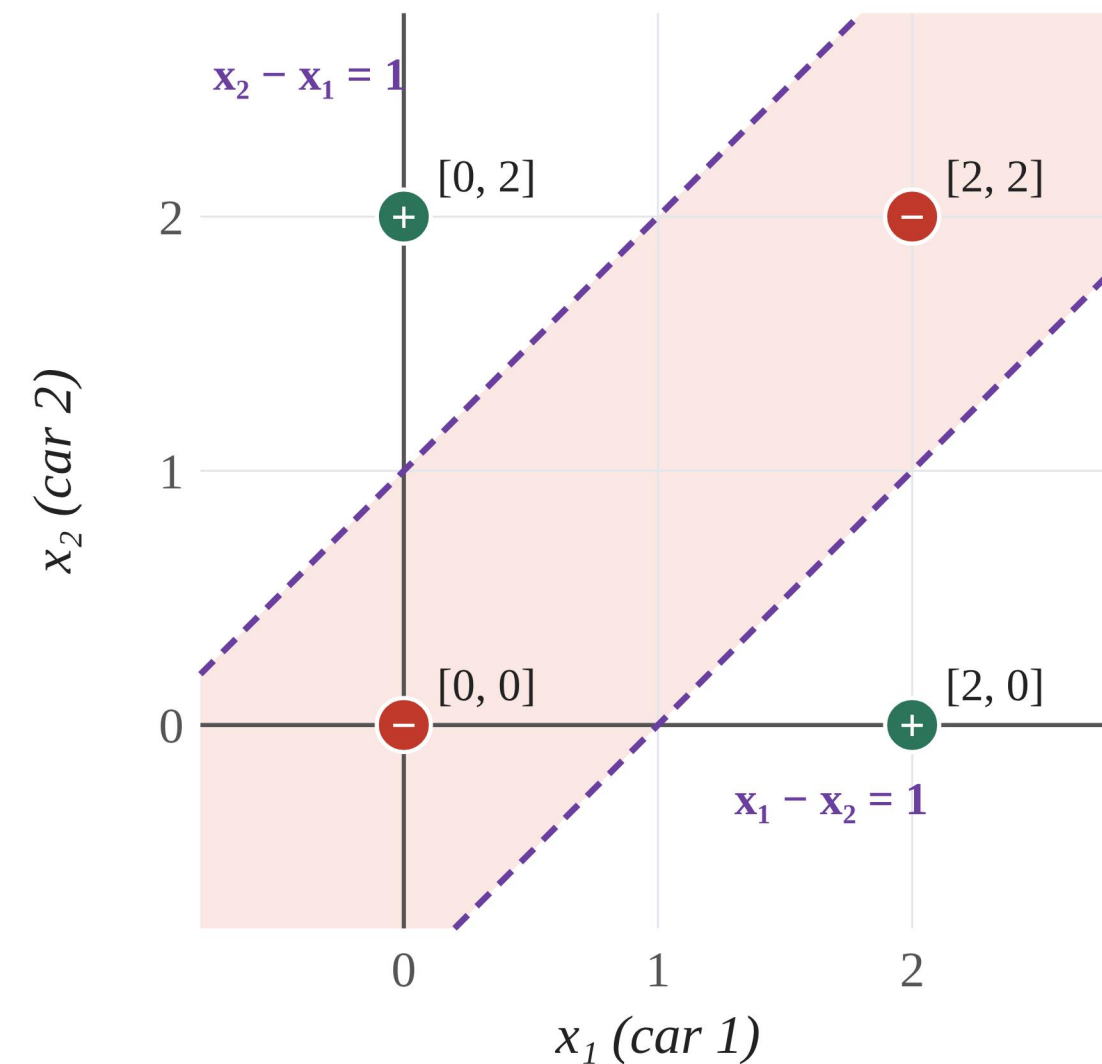
Car 1 far right of car 2? $h_1(x) = \mathbf{1}[x_1 - x_2 \geq 1]$

Car 2 far right of car 1? $h_2(x) = \mathbf{1}[x_2 - x_1 \geq 1]$

Safe if either: $f(x) = \text{sign}(h_1(x) + h_2(x) - \frac{1}{2})$

x	$h_1(x)$	$h_2(x)$	$f(x)$
$[0, 2]$	0	1	+1
$[2, 0]$	1	0	+1
$[0, 0]$	0	0	-1
$[2, 2]$	0	0	-1

Each h_j is a linear classifier. And f is a linear classifier on top of h_1, h_2 !



In vector notation

With $\phi(x) = [1, x_1, x_2]$, each h_j thresholds a dot product:

$$h_1(x) = \mathbf{1}[x_1 - x_2 \geq 1] = \mathbf{1}[v_1 \cdot \phi(x) \geq 0], \quad v_1 = [-1, +1, -1]$$

$$h_2(x) = \mathbf{1}[x_2 - x_1 \geq 1] = \mathbf{1}[v_2 \cdot \phi(x) \geq 0], \quad v_2 = [-1, -1, +1]$$

Stack v_1 and v_2 as the rows of a matrix V , and threshold each entry:

$$V = \begin{bmatrix} -1 & +1 & -1 \\ -1 & -1 & +1 \end{bmatrix}, \quad h(x) = \mathbf{1}[V\phi(x) \geq 0]$$

Prediction: $f(x) = \text{sign}(w \cdot h(x) - \frac{1}{2})$, with $w = [1, 1]$.

This is the idea of a 2-layer neural network:

- $\phi(x) = [1, x_1, x_2]$ is our input layer (nothing to learn)
- V : weights for our (only) hidden layer, $h(x)$
- w : weights for the output layer, $f(x)$

Here we just set V and w by hand.

In practice, we'll learn them using gradient-based optimization again

We could have many more hidden layers (forming a deep neural net)

Avoiding zero gradients

To learn V by gradient descent, we need $\nabla_V \text{Loss}$.

But $\mathbf{1}[v \cdot \phi(x) \geq 0]$ is flat almost everywhere. Same problem as the zero-one loss!

Fix: replace the threshold by an **activation function** σ with useful gradients.

Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$: smooth threshold. Small gradient when $|z|$ is large.

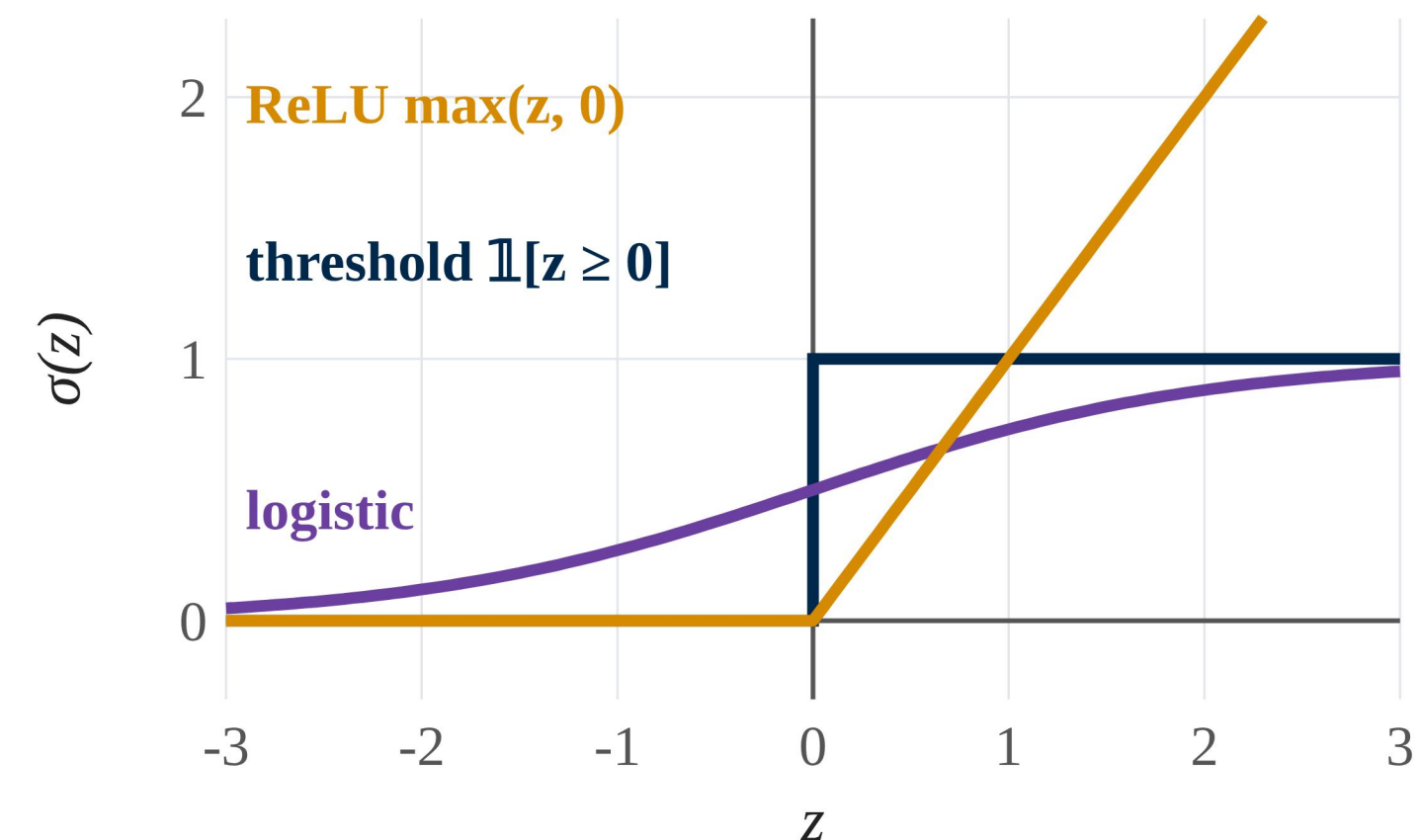
ReLU $\sigma(z) = \max(z, 0)$: gradient 1 when $z > 0$. Cheap, and typical choice (or modern variants like GELU, GLU, SwiGLU).

If we used linear $\sigma(z) = z$, then $w \cdot (V\phi(x)) = (V^\top w) \cdot \phi(x)$: linear again!

So many linear layers combined linearly can express the same as a single linear layer

To gain anything from many layers, we need σ to be non-linear

That's why $\sigma(z)$ is also called the **non-linearity**



Two-layer neural networks

Putting it together, with $V \in \mathbb{R}^{k \times d}$ and $w \in \mathbb{R}^k$:

$$h(x) = \sigma(V\phi(x))$$

$$\text{score} = w \cdot h(x)$$

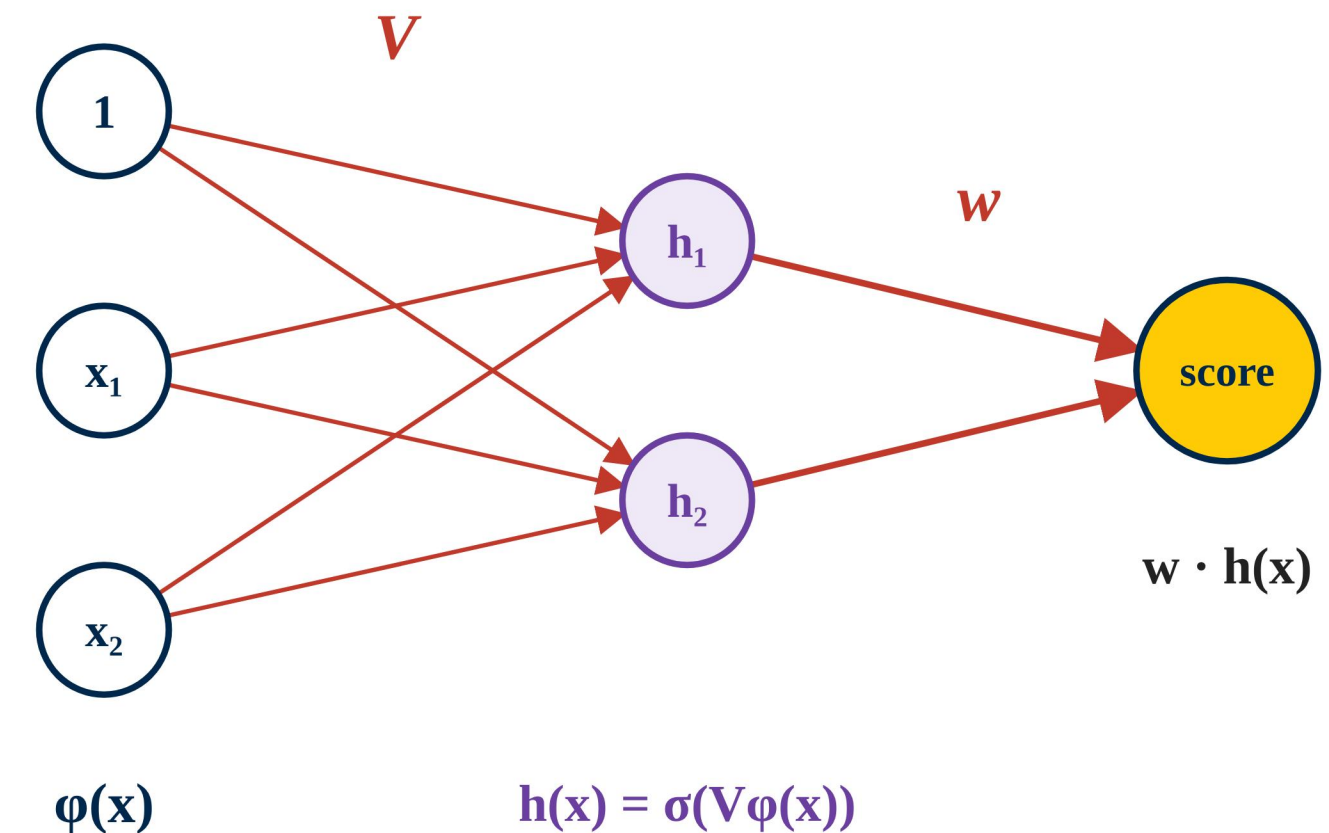
Regression: predict the score. Classification: its sign.
Same losses as before.

$$\mathcal{F} = \{f_{V,w} : V \in \mathbb{R}^{k \times d}, w \in \mathbb{R}^k\}$$

The last layer is a linear predictor, on features $h(x)$.

But now the features are **learned**, together with w .

(In practice, each layer also adds a bias term: $V\phi(x) + b$)



Example: energy use with ReLUs

We can go back to our energy consumption example and get away without feature engineering now

With $\phi(x) = [1, x]$ and ReLU, each hidden unit is $h_j(x) = \max(v_{j1} + v_{j2}x, 0)$.

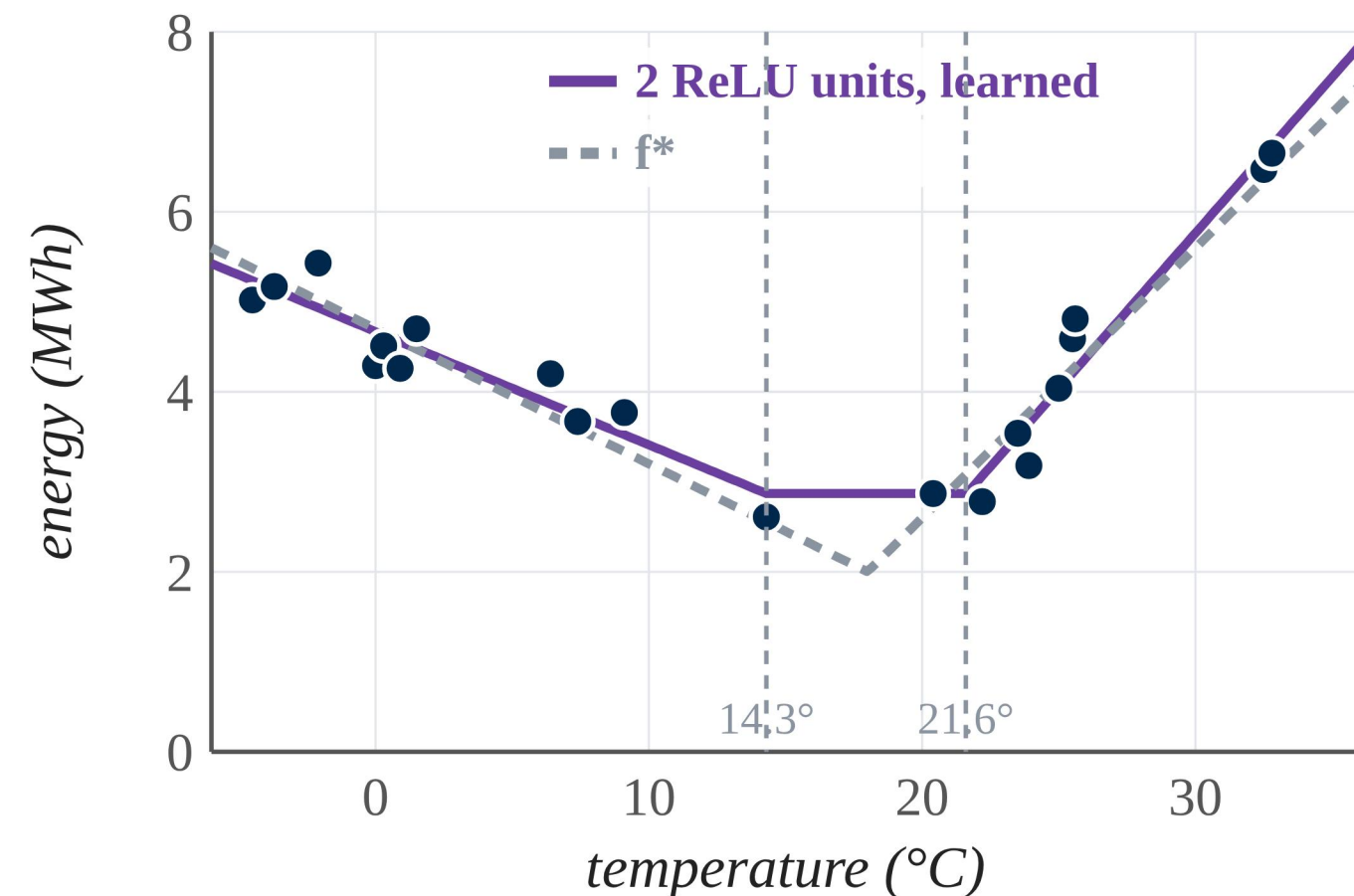
A 2-layer NN with 2 hidden units, trained by gradient descent on same data:

Leans two "linear on one side, zero on the other" features, like our heating and cooling features at 18C.

Q: Did it find the true curve f^* ?

A: Close, but can see some estimation error.

Not approximation error: optimal predictor is in the hypothesis class now



Deep neural networks

We can repeat the (linear map, activation) step:

$$\text{1 layer: score} = w \cdot \phi(x)$$

$$\text{2 layers: score} = w \cdot \sigma(V\phi(x))$$

$$\text{3 layers: score} = w \cdot \sigma(V_2 \sigma(V_1 \phi(x)))$$

The last layer is always a linear predictor on learned features.

Each layer builds on the previous layer's features. On images: edges $\rightarrow$ textures $\rightarrow$ object parts.

Intuition: each layer can learn increasingly high-level features using previous ones

Lots of open theoretical questions on why deep learning works so well

Next we'll see: backpropagation, optimization, deep learning architectures, generalization again.

- **Classification losses:** zero-one is what we care about. Hinge and logistic have useful gradients. Logistic gives probabilities.
- **Test error** = best possible error + approximation error + estimation error.
- **PAC learning:** for finite $\mathcal{F}$, $\frac{1}{\epsilon} (\ln |\mathcal{F}| + \ln \frac{1}{\delta})$ examples suffice.
- **Features:** $w \cdot \phi(x)$ is linear in w , but can be very nonlinear in x .
- **Neural networks** learn the features too: $w \cdot \sigma(V\phi(x))$, and deeper.

Next week: Module 3, deep learning.