Intro to ML I

CERN School of Computing 2023

Lukas Heinrich, TUM

Why ML?

Why ML for Fundamental Physics

In a way, this is what we do:

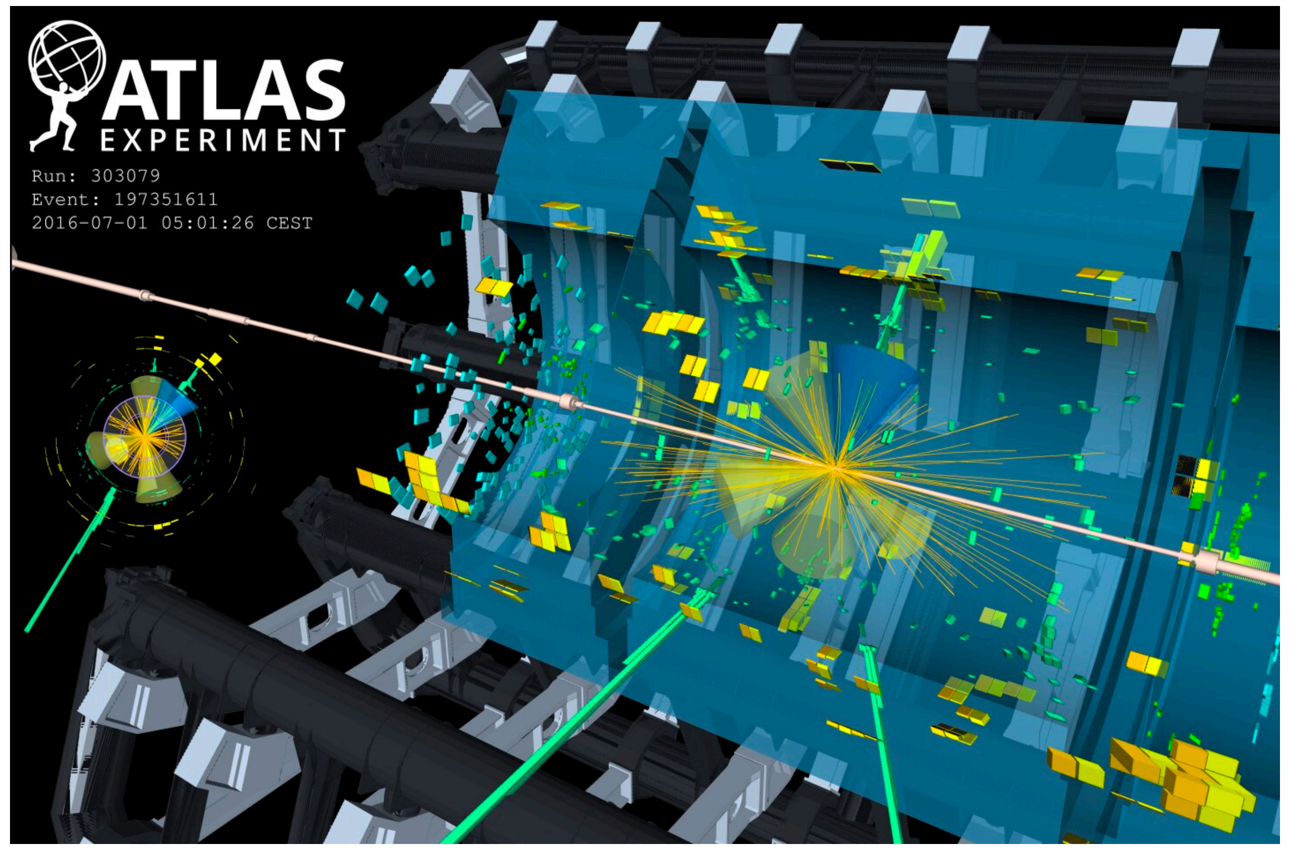
$$p(\text{theory}\,|\,\text{data}) = \frac{p(\text{data}\,|\,\text{theory})p(\text{theory})}{p(\text{data})}$$

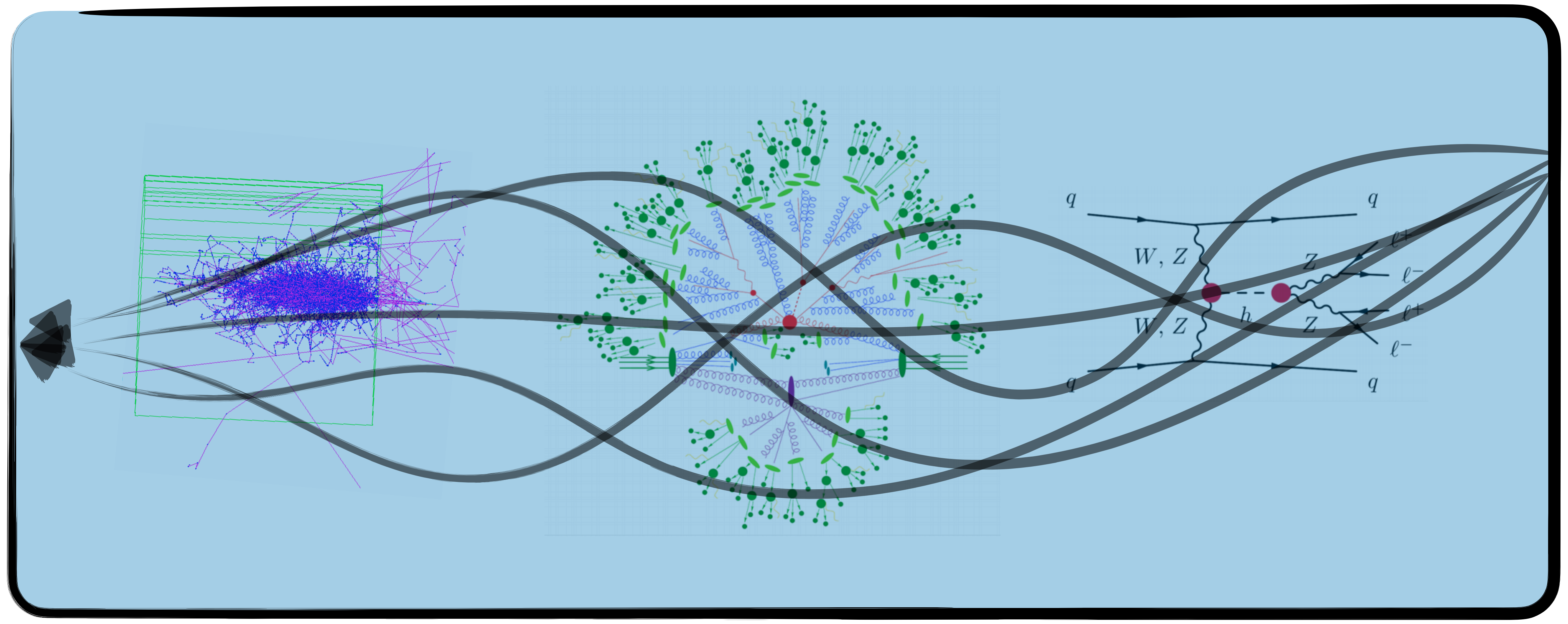
On the face of it: no ML to be seen

Turns out, this is not so easy
and ML can help a lot!

Complex Data

It’s often impossible to get closed-form predictions





$$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu} + i \bar{\Psi} \not{D} \psi + h.c. + \bar{\Psi}_i y_{ij} \psi_j \phi + h.c. + |D_\mu \phi|^2 - V(\phi)$$

z : intermediate unobserved physics

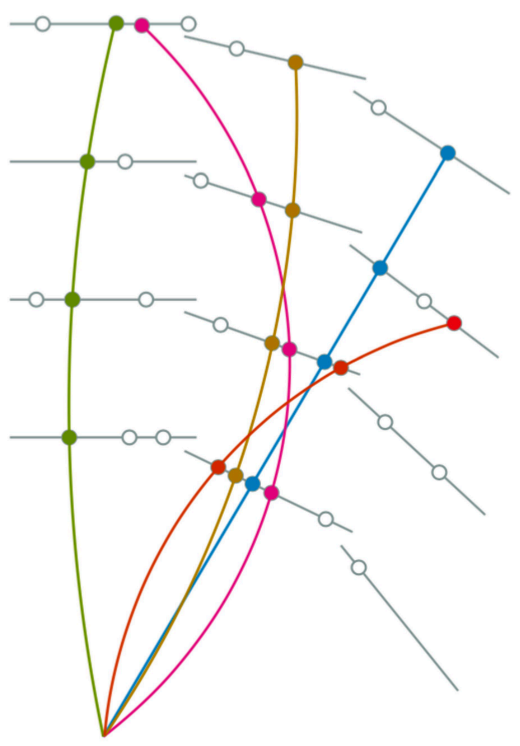
$$p(\text{data}|\text{theory}) = \int_Z p(\text{data}|z)p(z|\text{theory})$$

often completely intractable

Pattern Recognition

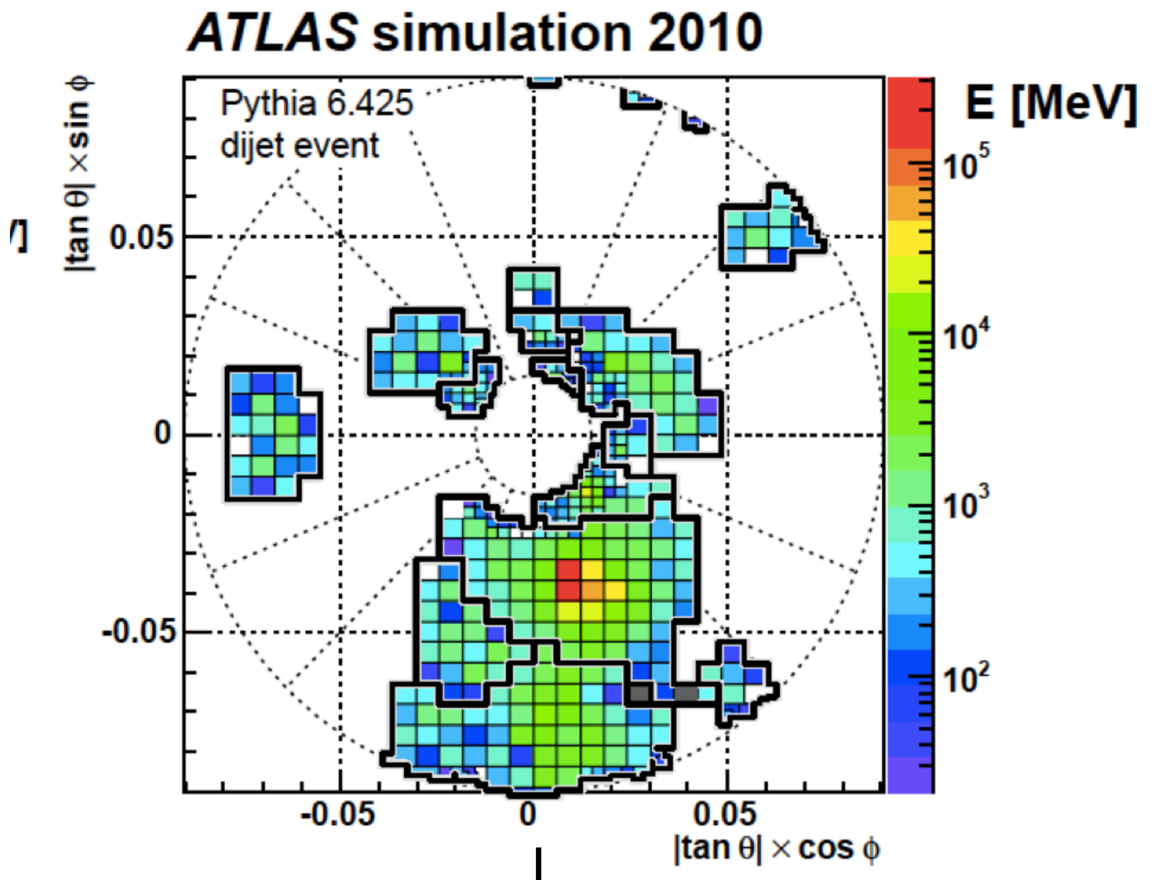
Goal: bring the data into a form that is easier to understand and interpret

Hits



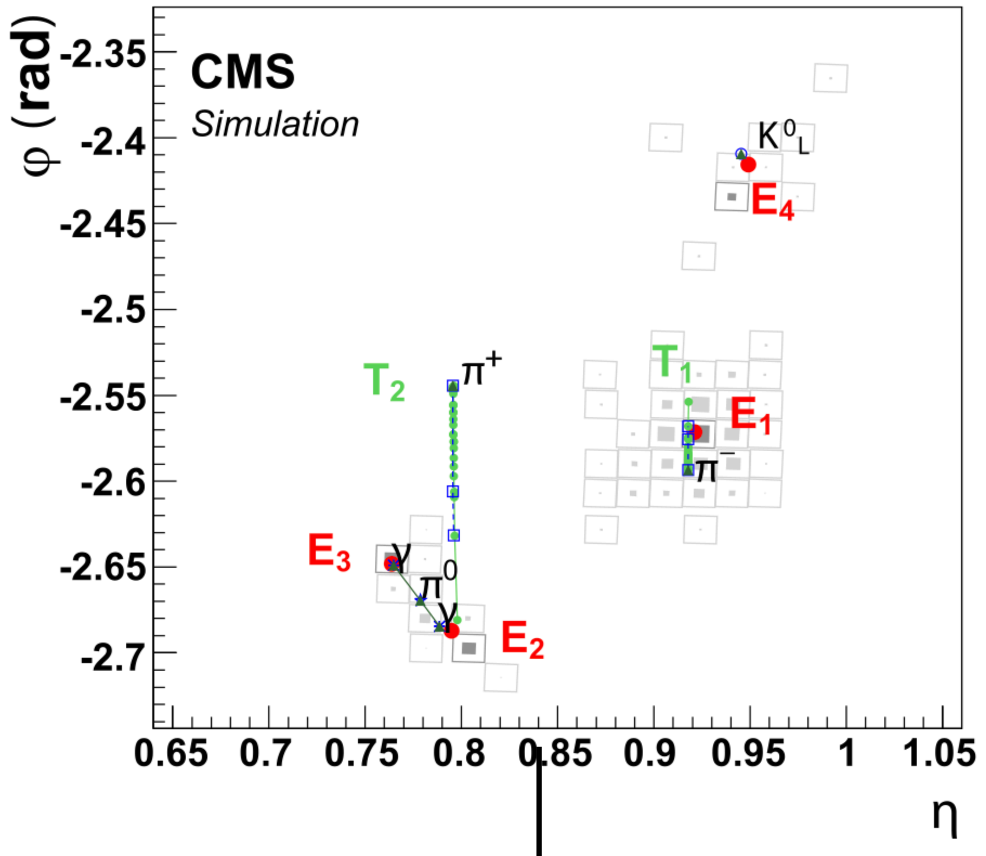
Tracks

Energy Cells



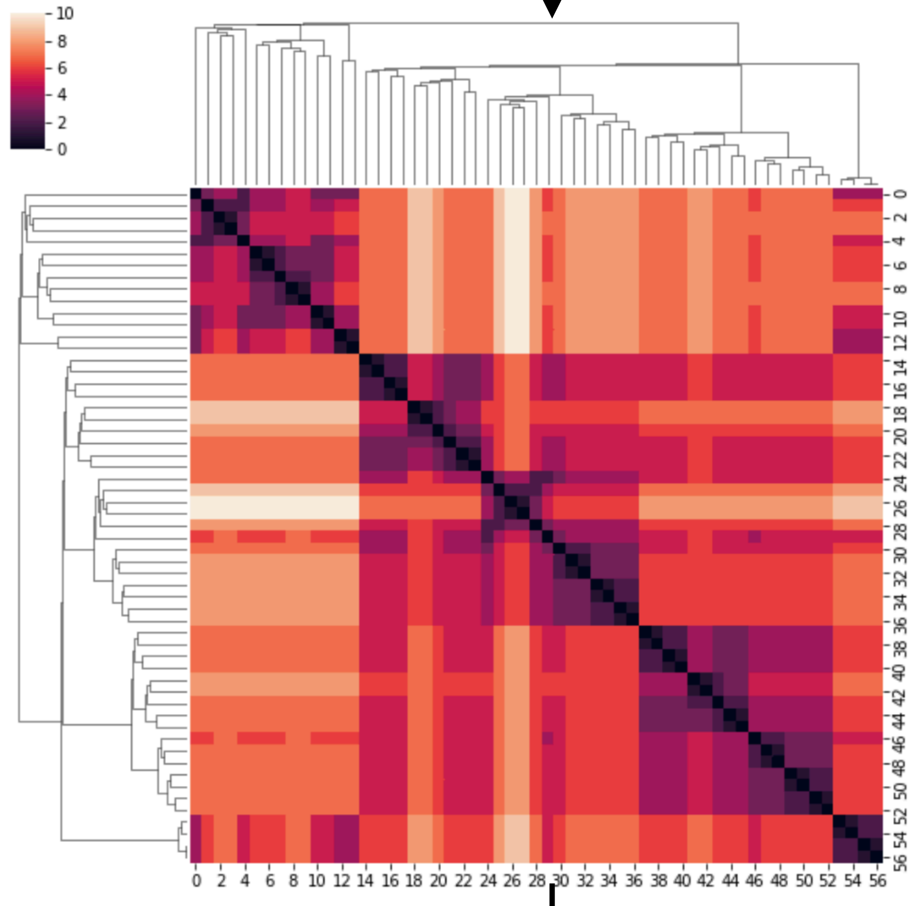
Energy Clusters

E Clusters & Tracks



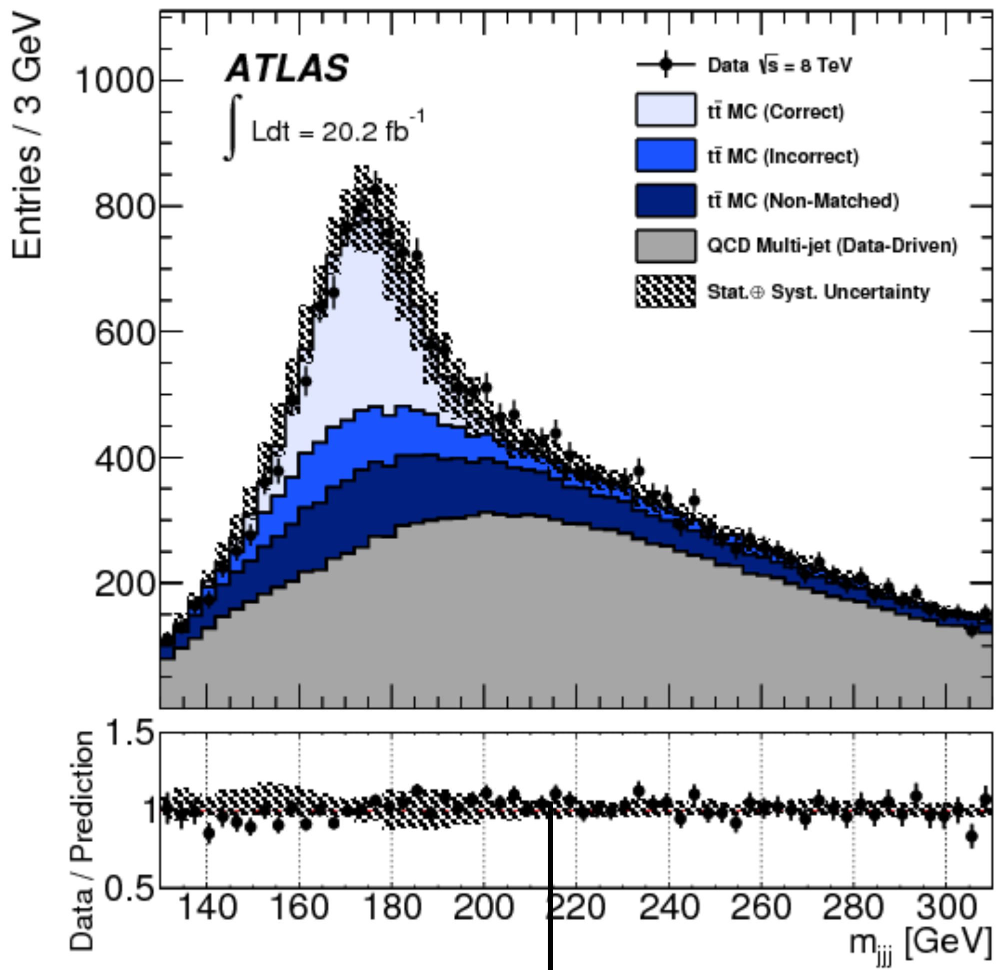
Particles

E Clusters



Jets (~ parton)

Particles

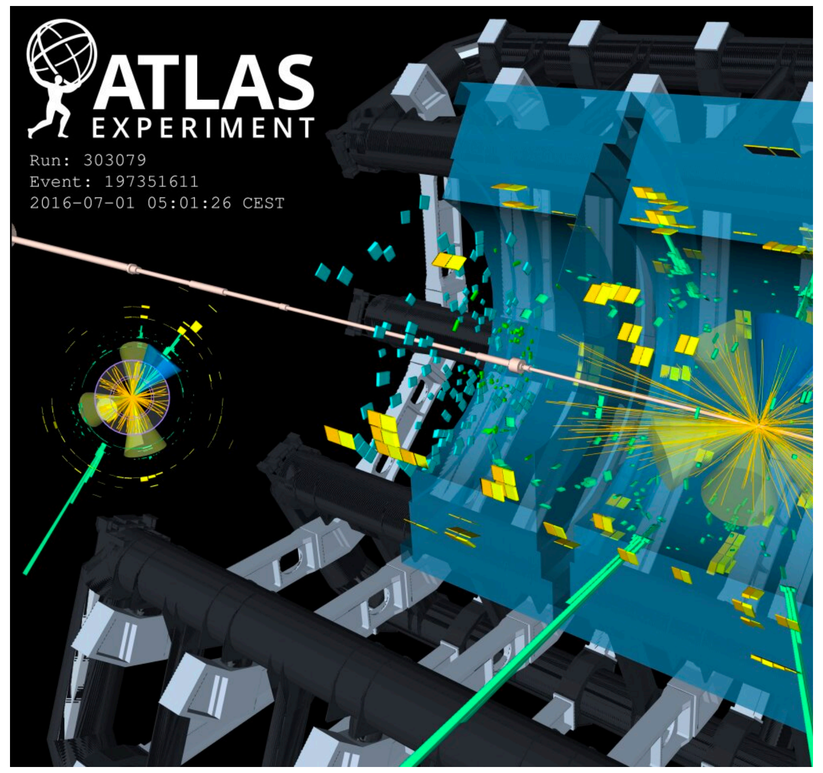


Resonances

Particle Physics in a Nutshell

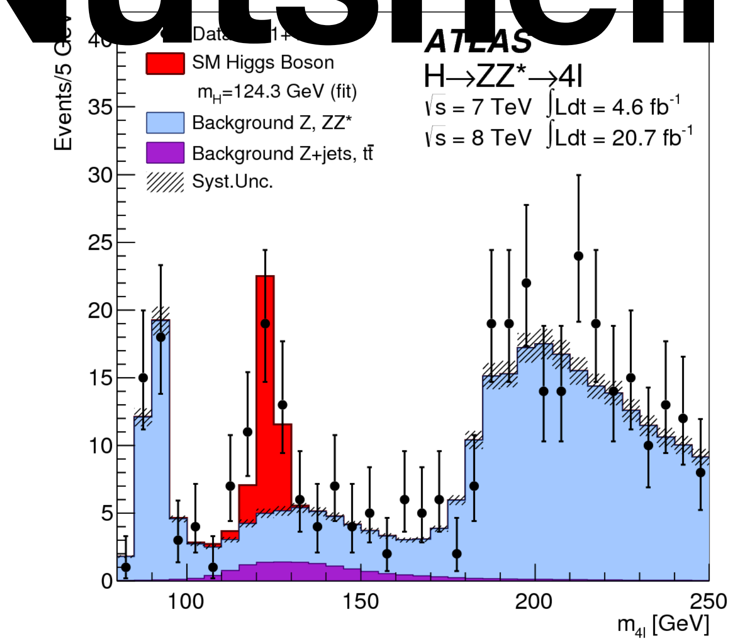
$$\begin{aligned}\mathcal{L} = &-\tfrac{1}{4} F_{\mu\nu} F^{\mu\nu} \\ &+ i\bar{\psi}\not{D}\psi + h.c. \\ &+ \bar{\psi}_i y_{ij} \psi_j \phi + h.c. \\ &+ |D_\mu \phi|^2 - V(\phi)\end{aligned}$$

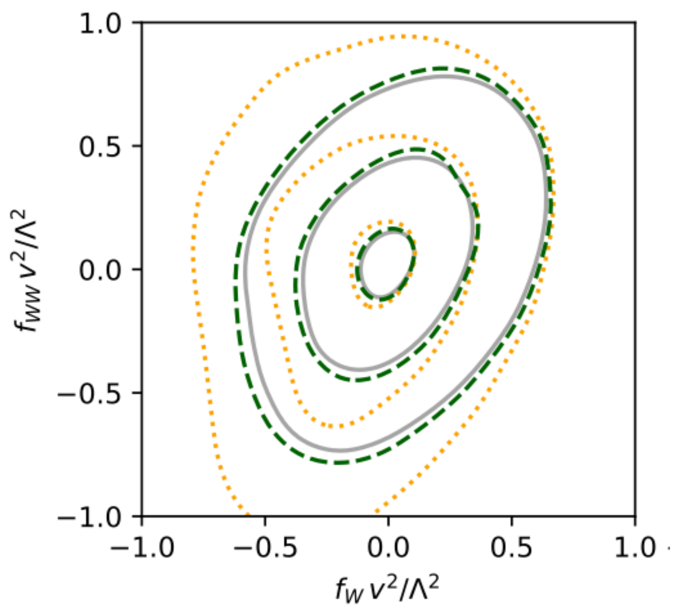
generate low-level, high-dim data from high-level concepts



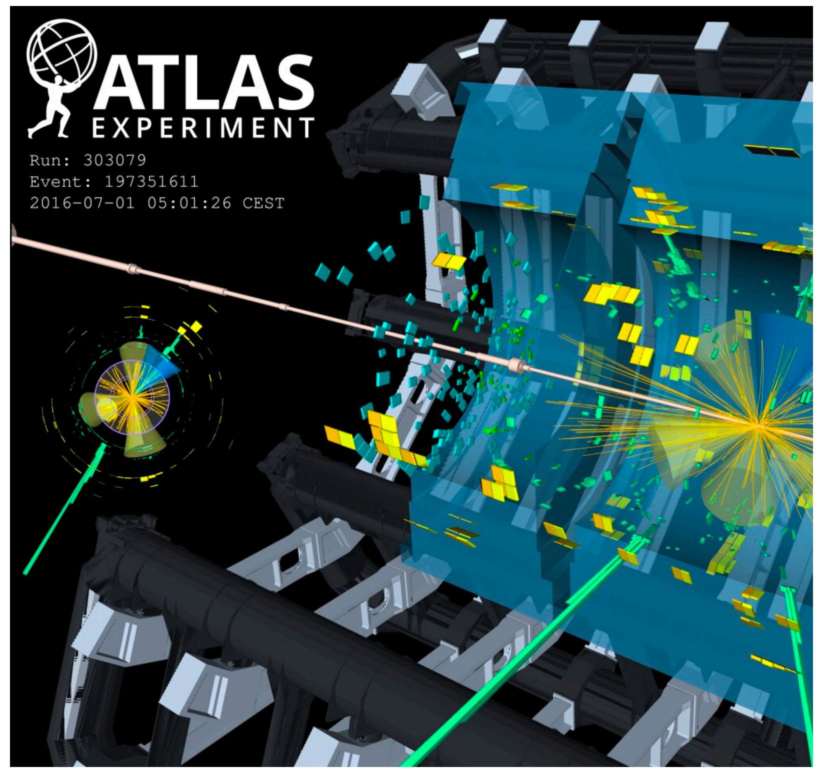
High-Level Concept

Low-Level Data





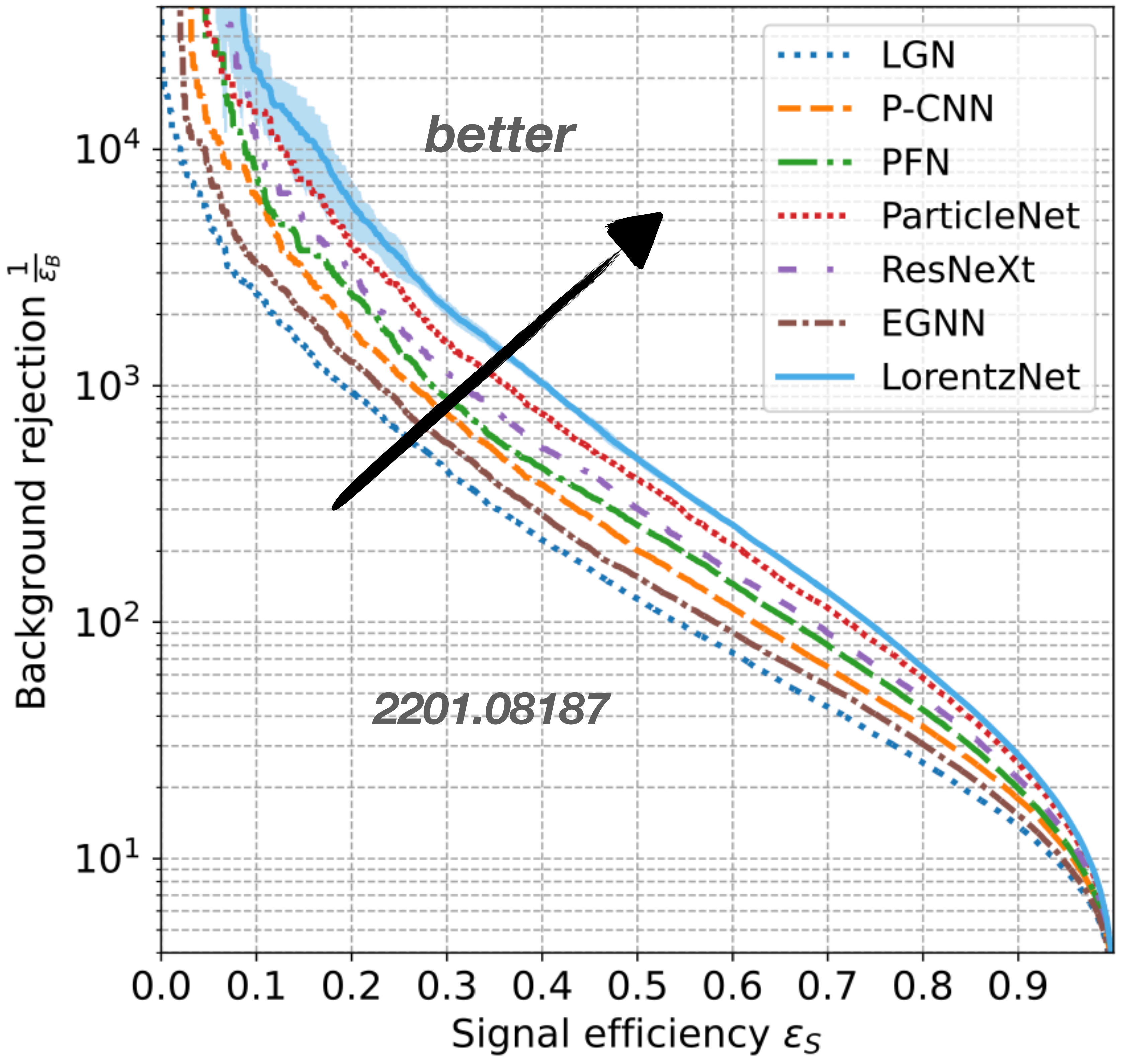
reconstruct high level concepts from low-level, high-dim data



Pattern Recognition

Not obvious what the most important patterns are to extract the most knowledge from the data.

It's an optimization problem (ML excels at this)



ML Systems are Good at Both

street style photo of a woman selling pho at a Vietnamese street market, sunset, shot on fujifilm

generate low-level, high-dim data from high-level concepts

High-Level Concept

Low-Level Data

This is a picture of Barack Obama.
His foot is positioned on the right side of the scale.
The scale will show a higher weight.

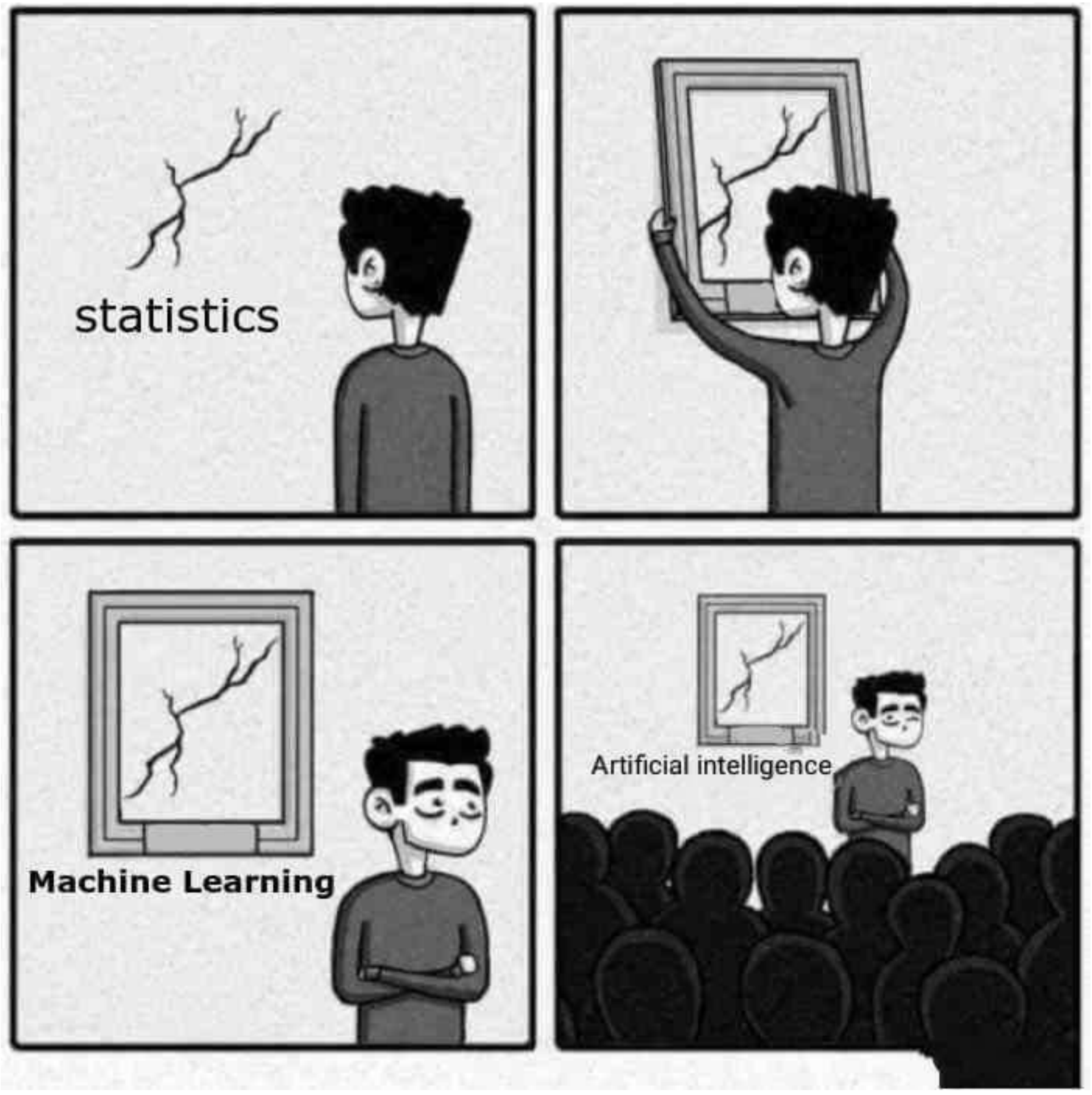
reconstruct high level concepts from low-level, high-dim data

What does it mean to learn?

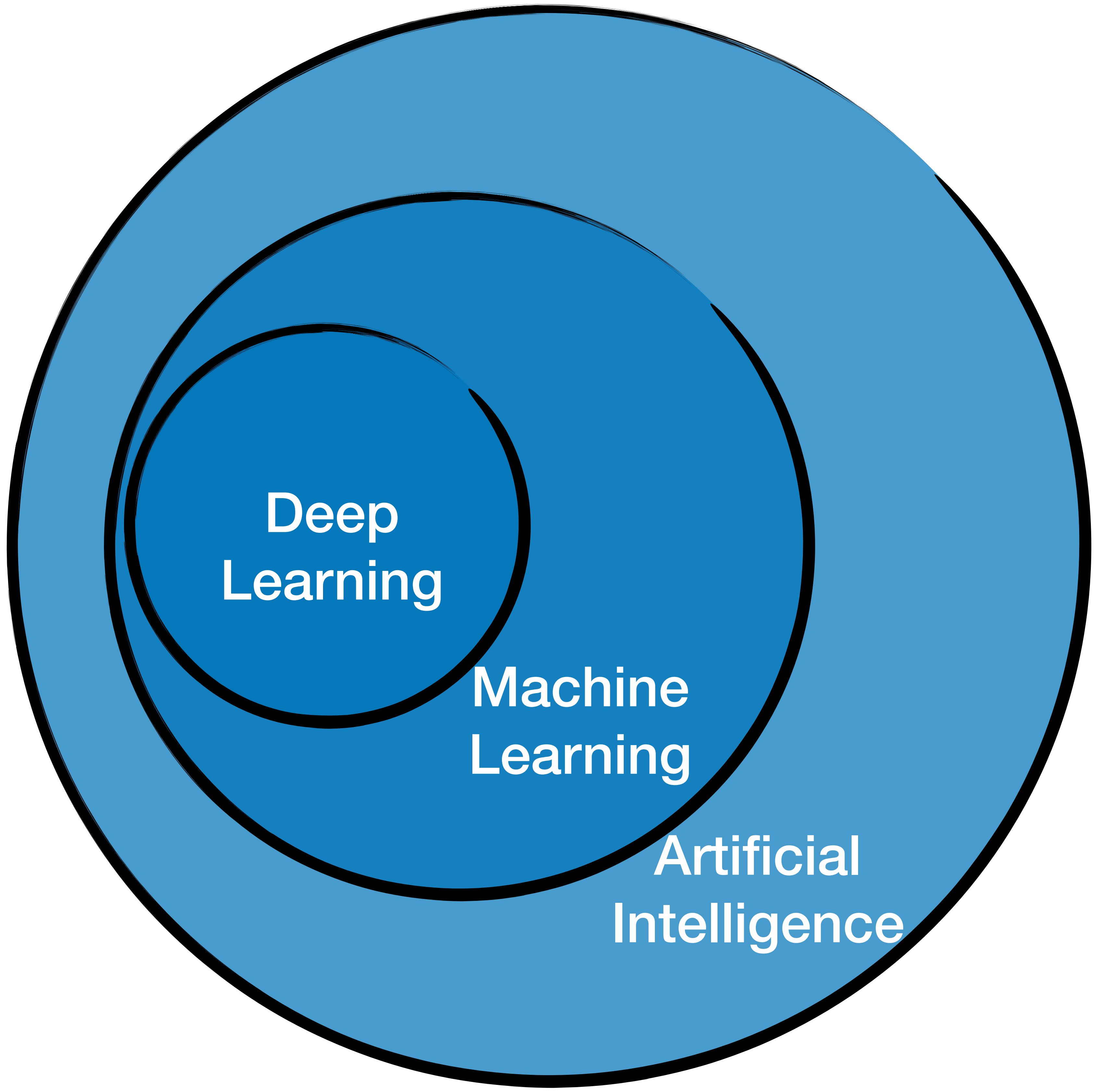
Defining the Terms

Colloquially the terms **"Artificial Intelligence"**, **"Machine Learning"** and **"Deep Learning"** are often used interchangably

Is there a difference?



Artificial Intelligence

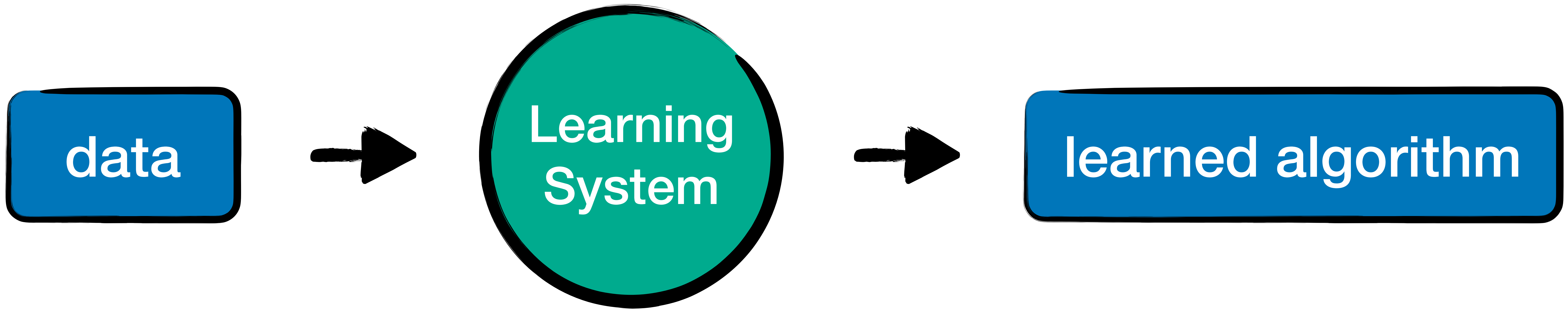


AI: make computers act in an “intelligent” way
(e.g. via rules, reasoning, symbol manipulation …)

ML: approach to AI that uses data to generate
the “intelligent” algorithms

DL: subset of ML that aims at complex pipelines,
work on low-level data (e.g. pixels)

Machine Learning



What kind of Algorithms ?

Two broad classes of algorithms we would like

learn to infer/predict
(unobserved data)

"Supervised Learning"

learn to describe
(the seen data)

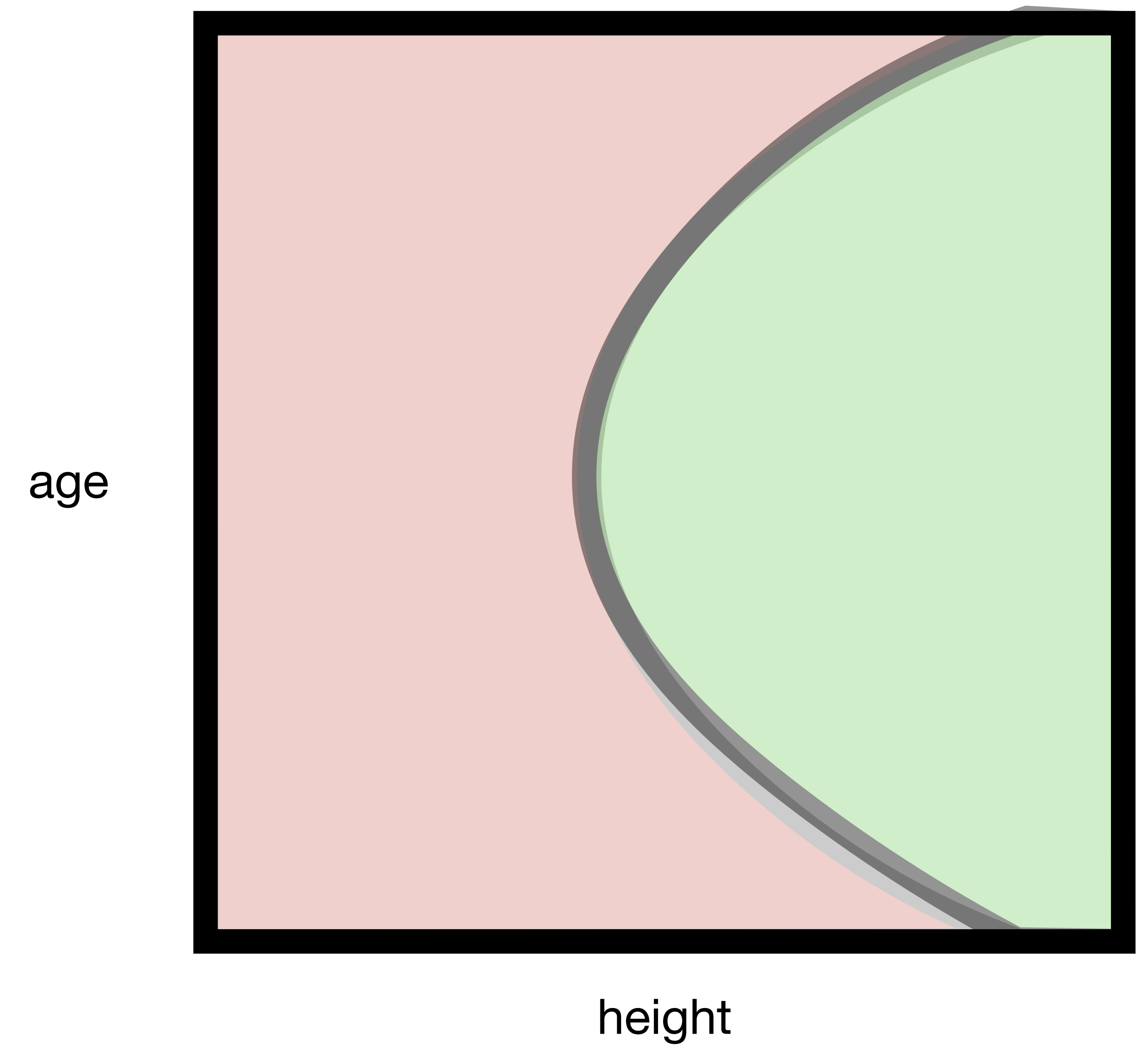
"Unsupervised Learning"

Example: Predicting Basketball Ability

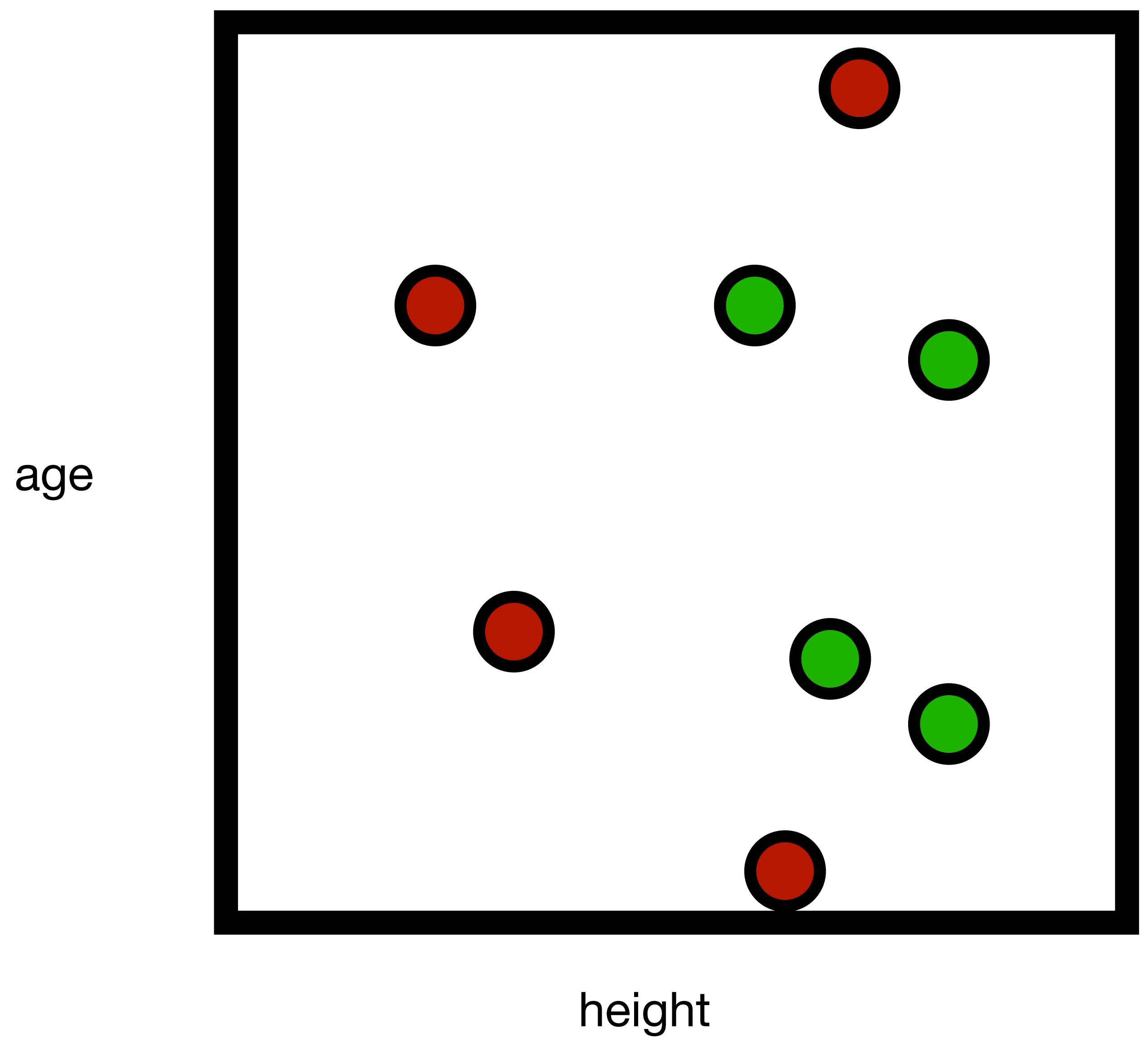
$$f : (x_{\text{age}}, x_{\text{height}}) \rightarrow [0,1]$$

height	age	Good 🏀
1.72m	26y	
1.59m	37y	
2.09m	17y	
...	...	...

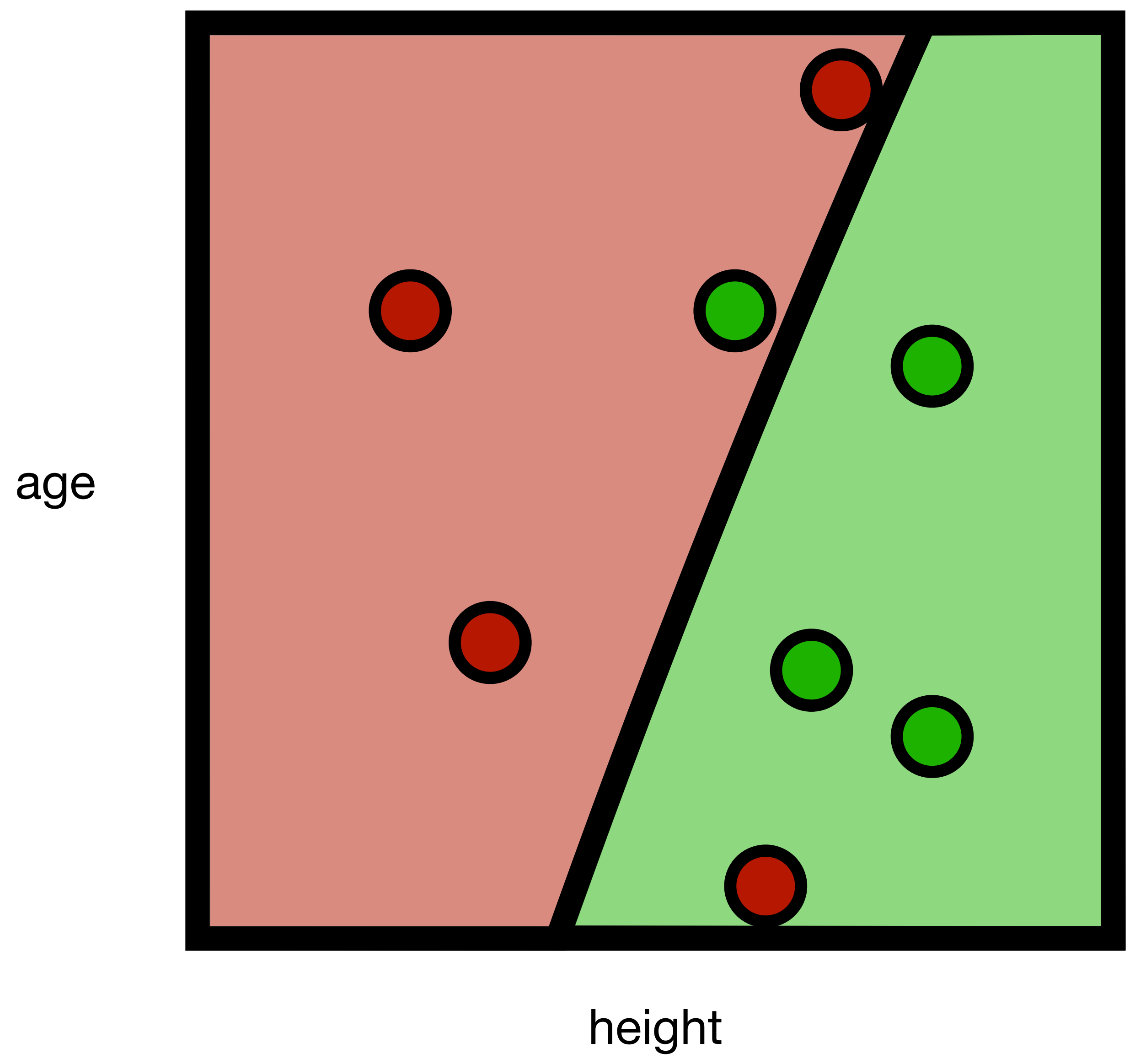
$$f^*(x \mid \mathscr{D})$$



True but unknown function



Empirical Data



Estimated Function

About the data…

Your connection to the algorithm is the data

- the most important thing in the ML lifecycle

Need to know:

- where does the **existing data** come from?
- where will the **new data** come from?

[src]

The dominant Paradigm: Statistical Learning

We **assume** the data is drawn i.i.d.

independent

identical

distributed

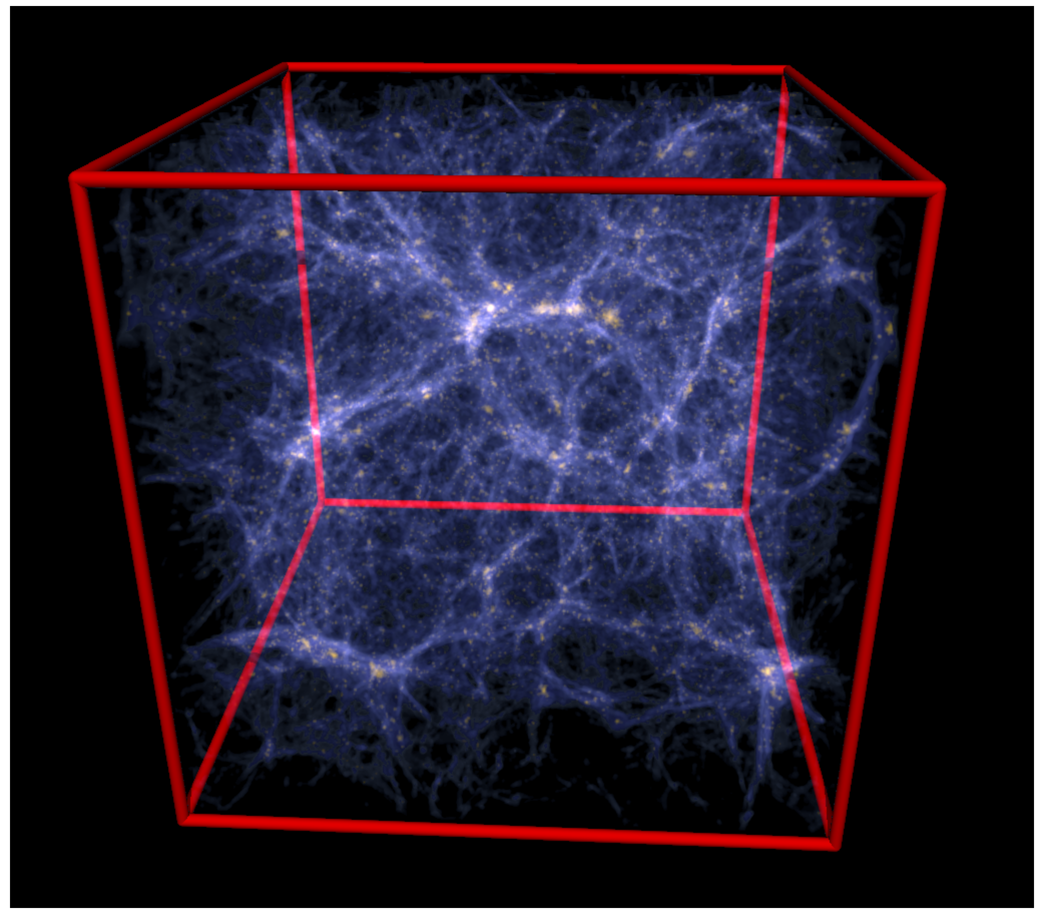
$$\text{data} = \{s_1, s_2, \ldots, s_n\} \qquad s \sim p(s)$$

We **assume** all **existing data and all future data** come from the same distribution.

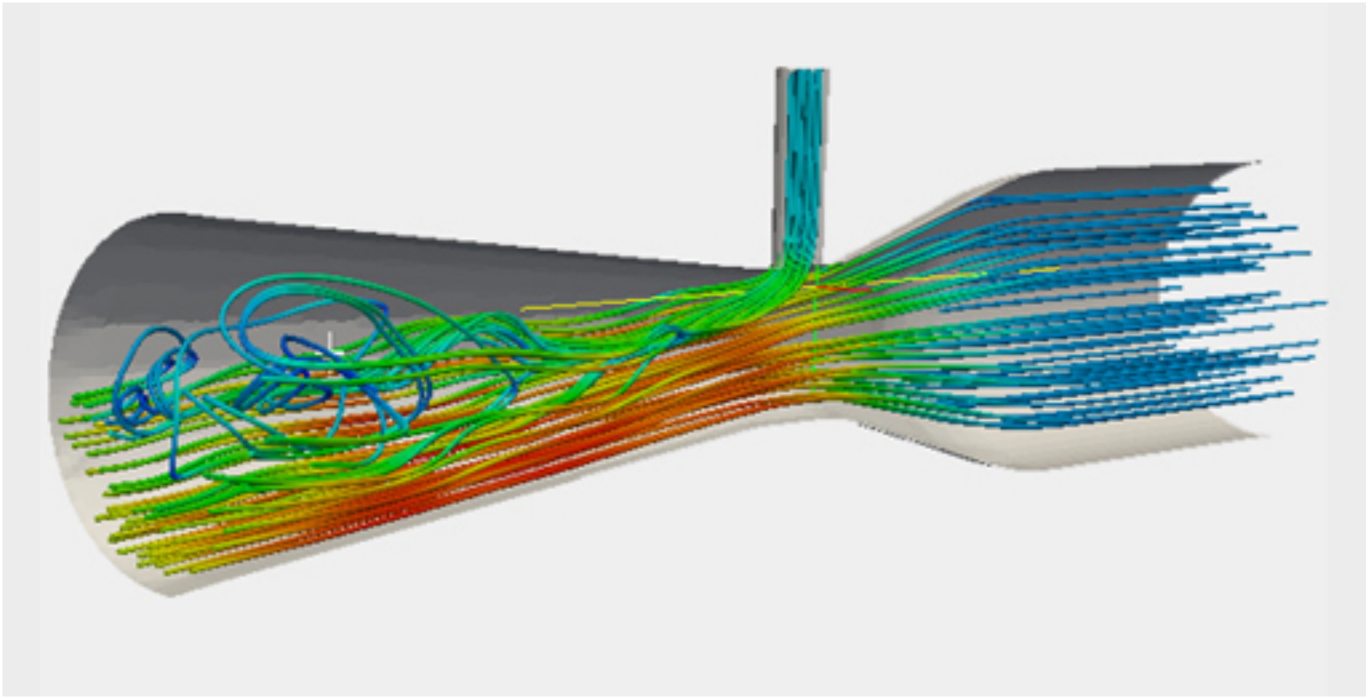
- Danger: "Out-of-Distribution" samples / Distribution Shift

Possible Data Sources

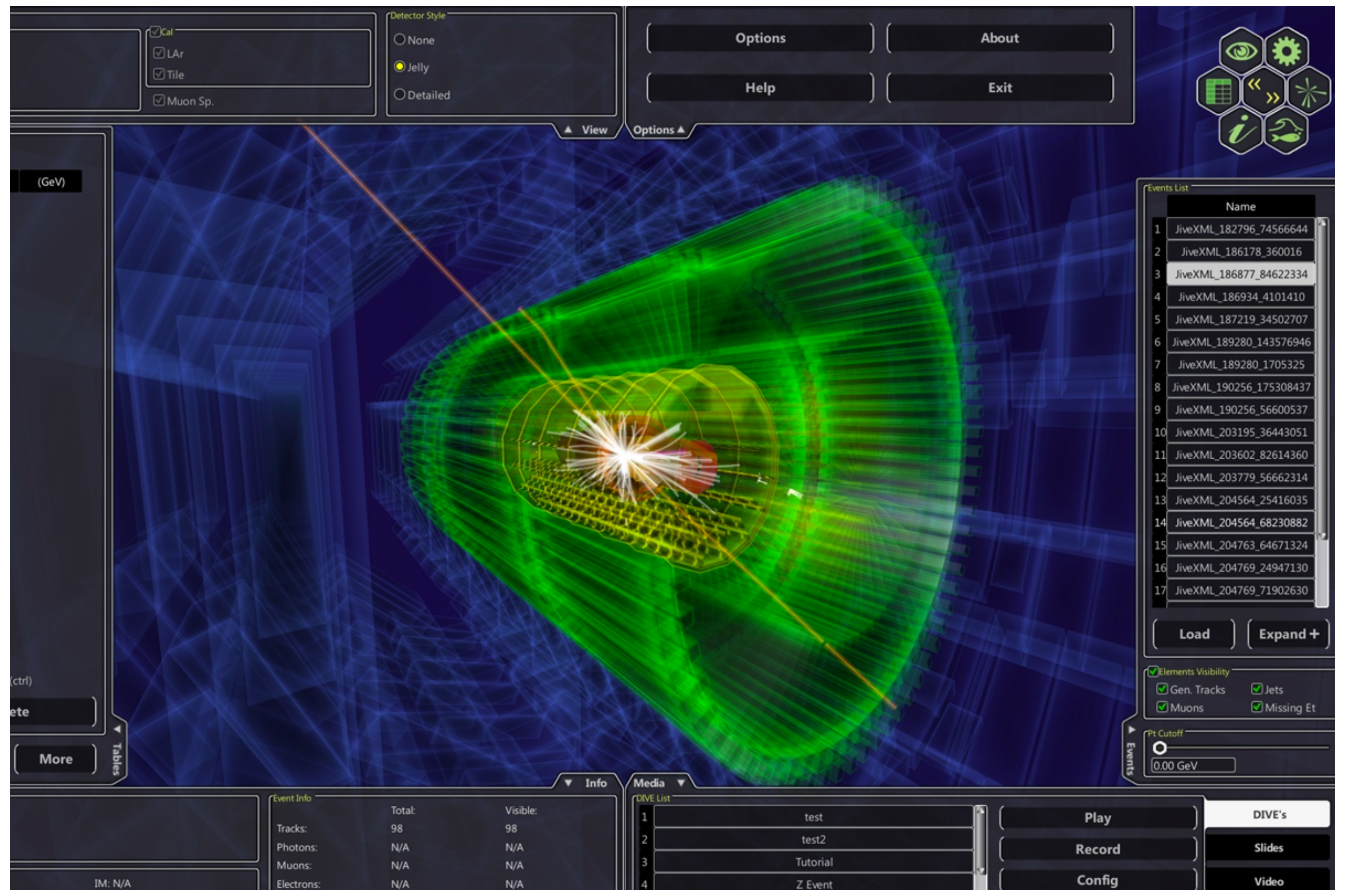
Huge advantage for ML in Science: We can actually often come close to this with our high-fidelity simulators.



simulated cosmology



simulated fluid dynamics



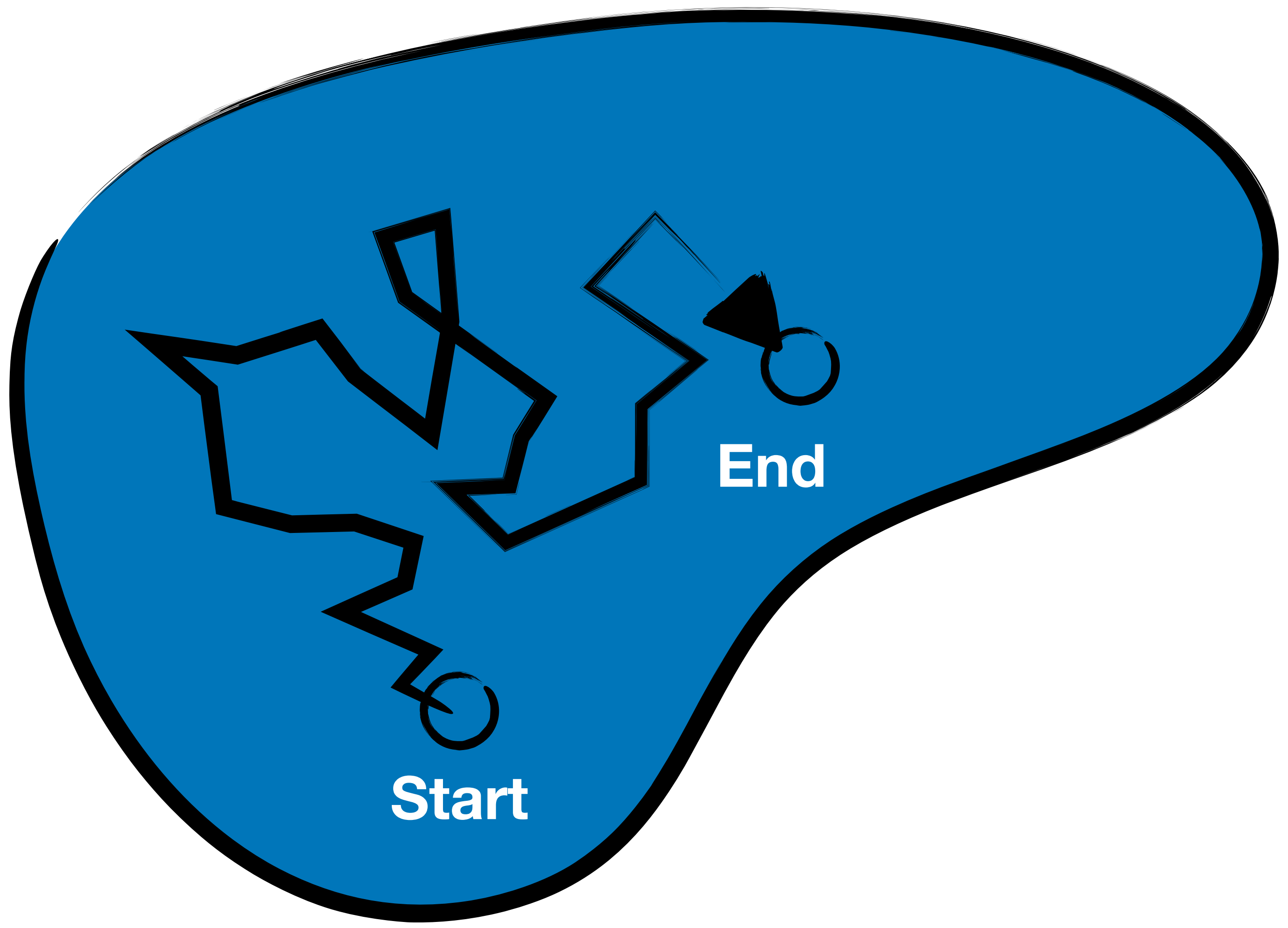
simulated particle physics

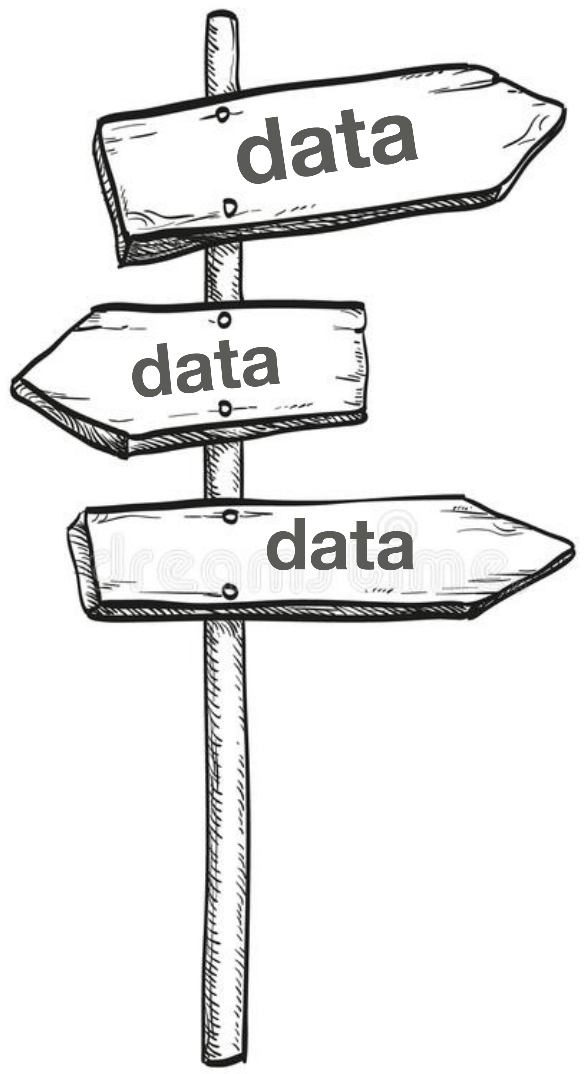
How do we learn?

Once we have data we need to turn it into an algorithm?

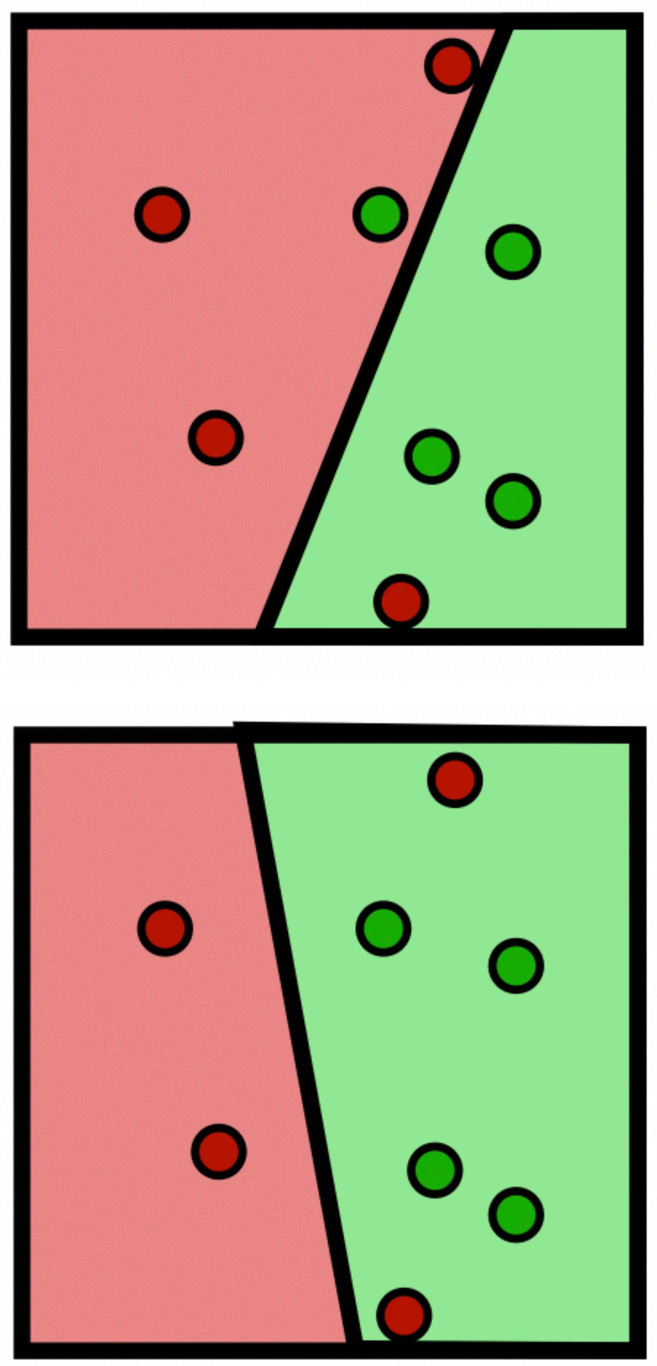
Idea: "Learning as Search" through a **Space of Programs**

- let the data guide you **to the best one**

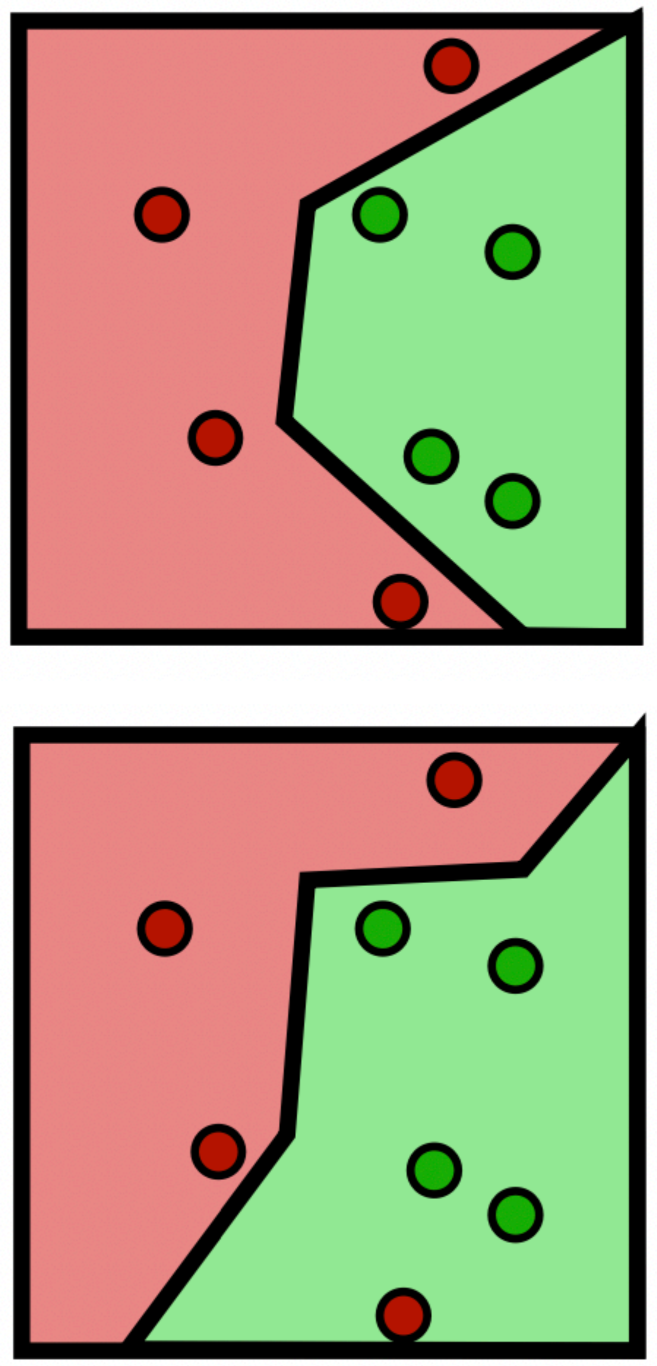




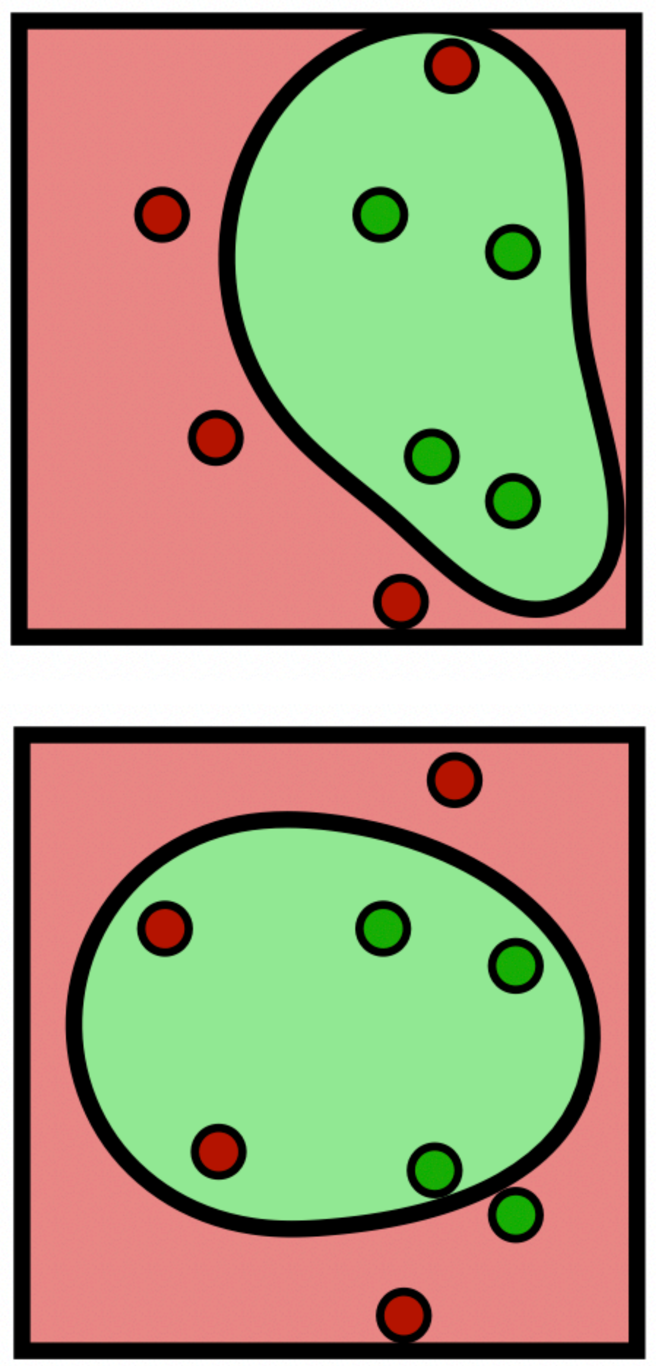
Examples



Linear Separators



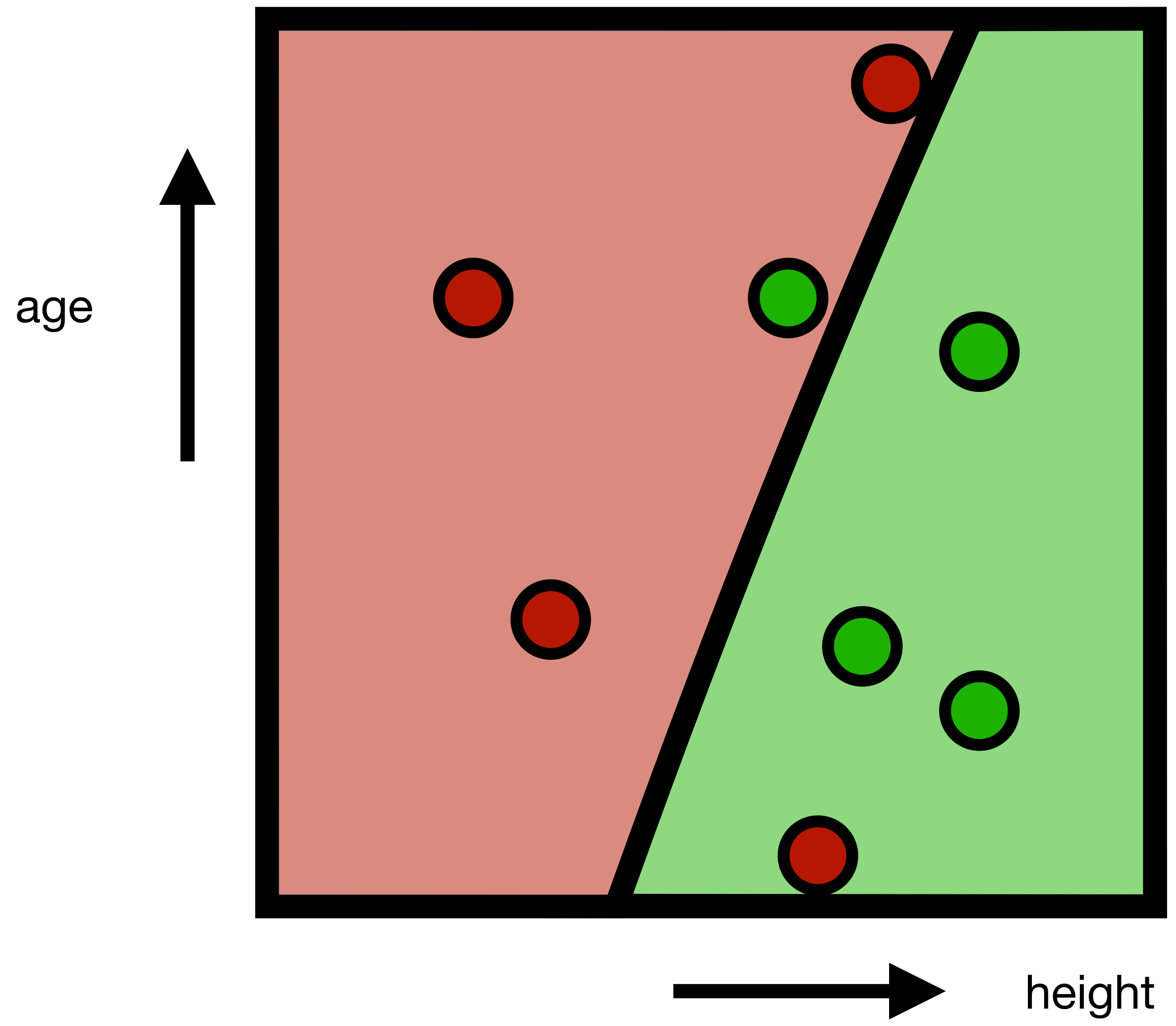
Piecewise Linear
Separators



Complex Curved
Areas

Assessing Performance

In order to start to learn, we need to be able to **assess the performance** of an algorithm: “risk” or “loss” (*lower* is better)



Algorithm mispredicts twice:
“risk” 2/8: 25%

Learning Goal
“Minimize the Risk”

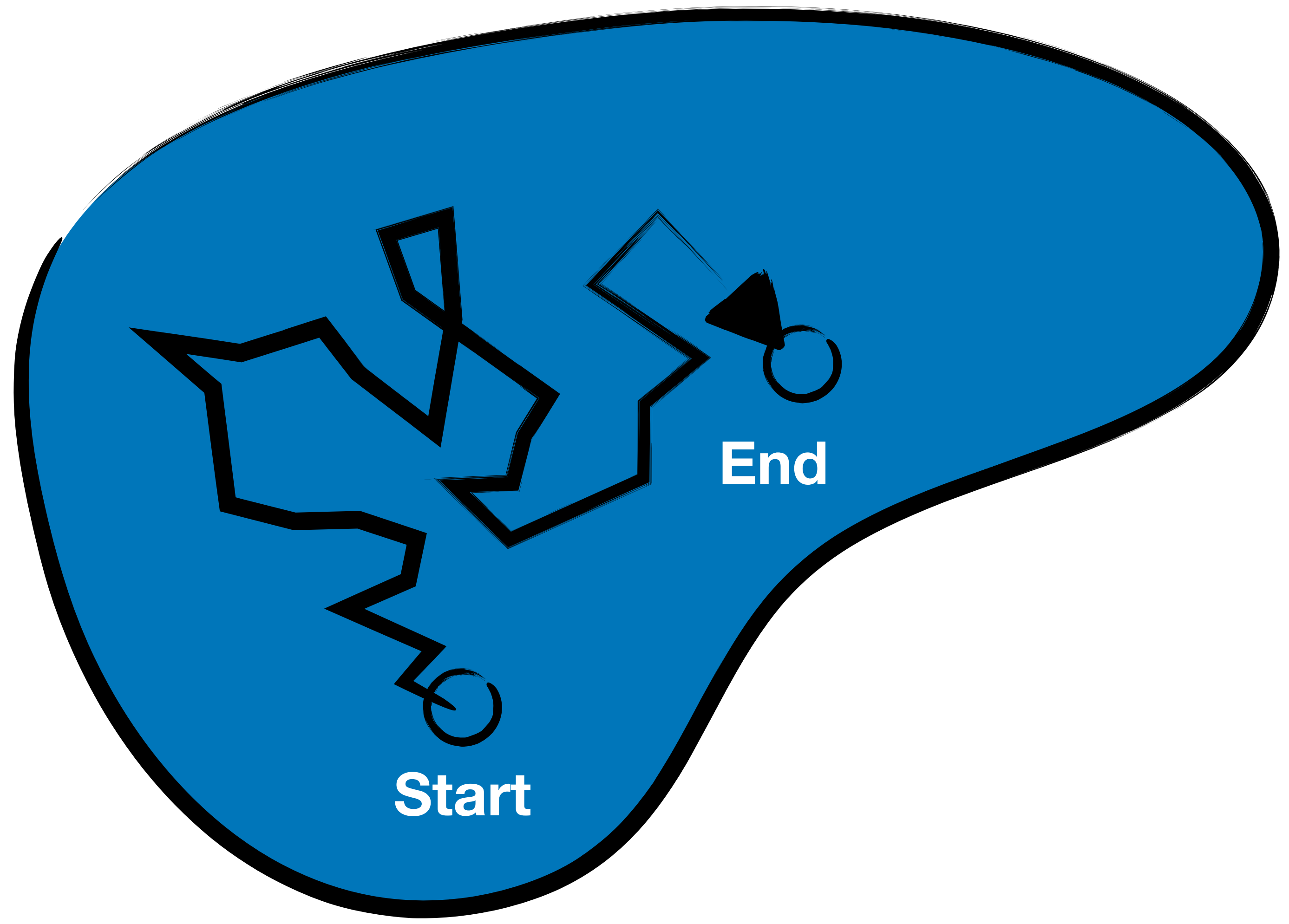
$$h^* = \text{argmin}_{\mathcal{H}} L(h)$$

Learning Algorithm

Usually we have no idea, which hypothesis is the best, we need to have a learning algorithm, that leads us there.

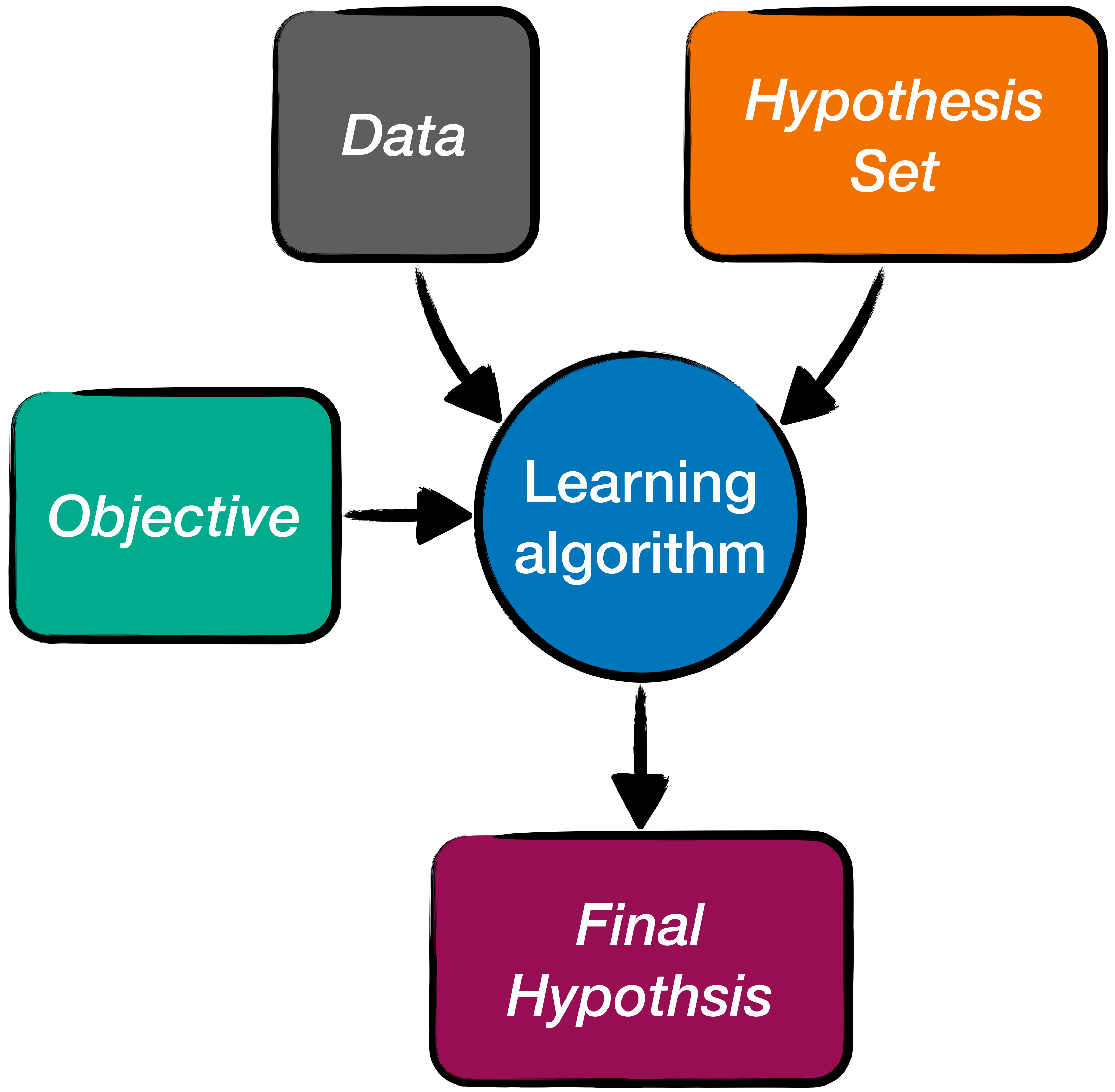
Various possibilities

- exhaustive search (discrete $\mathscr{H}$)
- closed form solutions (rare)
- iterative optimization (mostly used)



Summary: Learning Framework

- gather and **prepare data**
  to be consumed by the machine
- propose **search space of possible algorithms**
- Define what a “good” even means, i.e.
  a performance measure
- provide a **“learning algorithm”**
  to select the best one



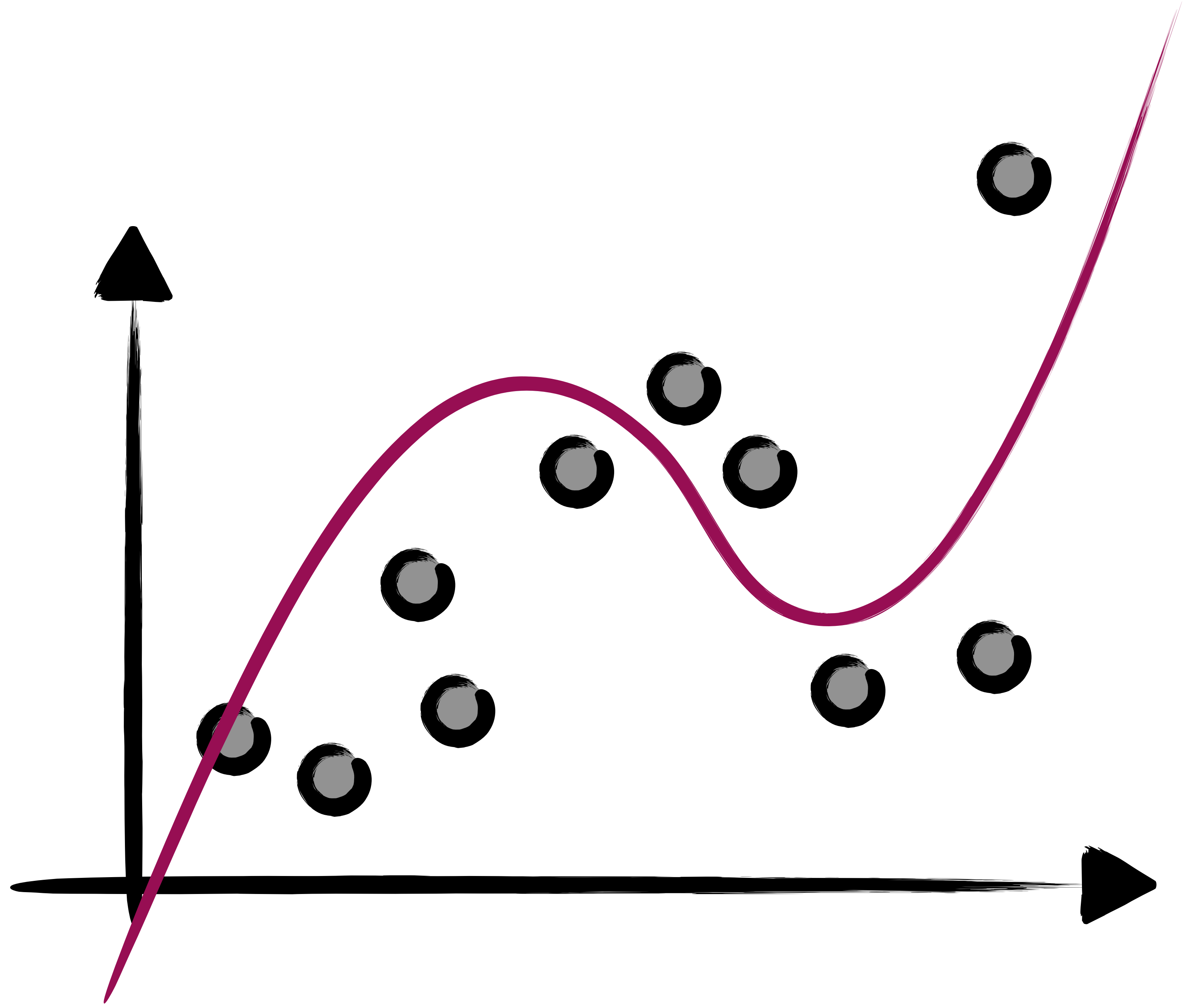
Example: Polynomial Regression

Hypothesis Set: Polynomials

$$(w_0, w_1, \ldots, w_n) \rightarrow y = f(x) = \sum_k w_k x^k$$

Risk: Mean Squared Error

$$\frac{1}{N} \sum_i (y - f_w(x))^2$$



Learning “Algorithm”: exact

$$w_{\text{best}} = (X^T X)^{-1} X^T y \qquad X_{ik} = x_i^k$$

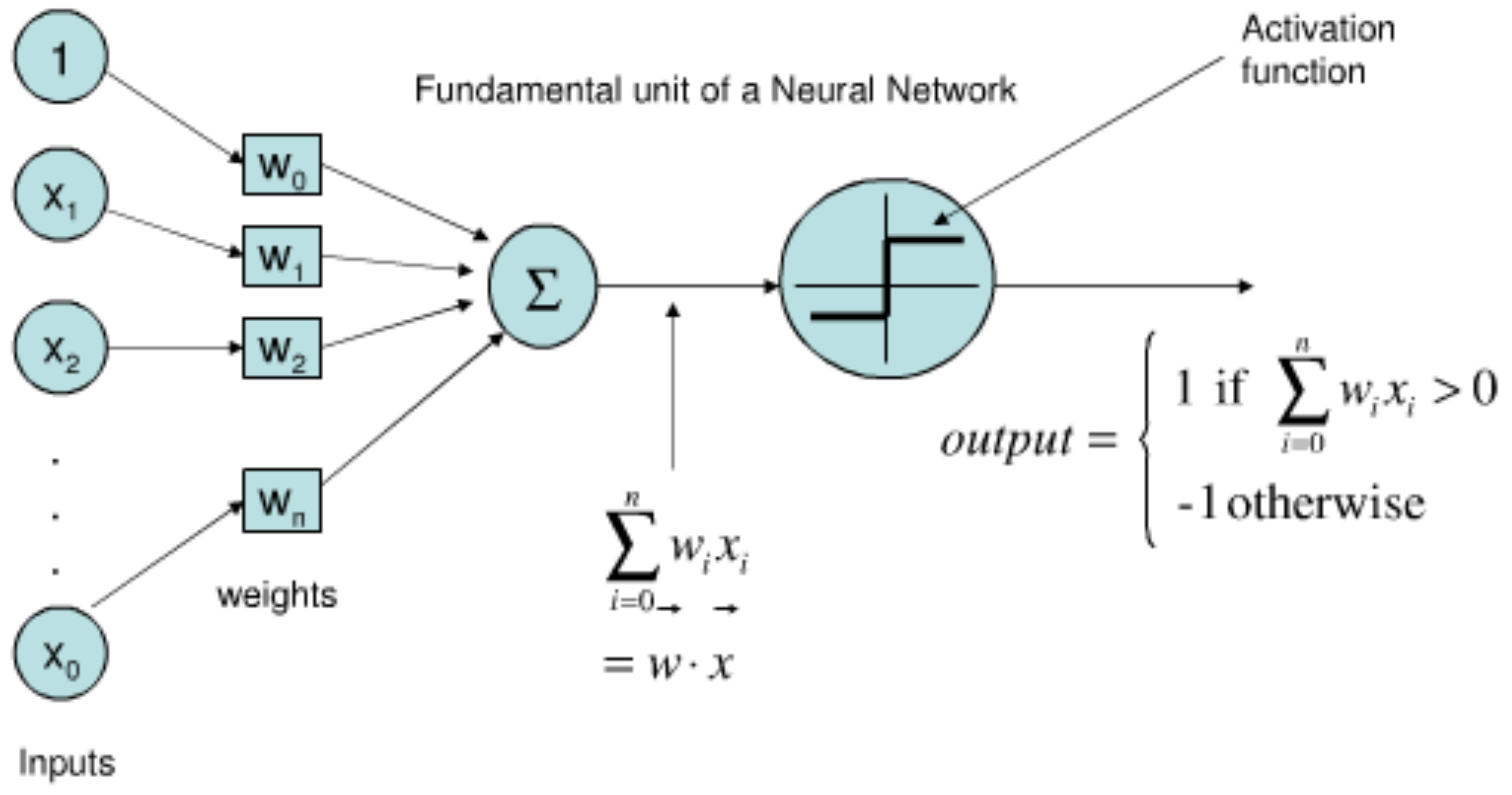
(i-th data point, k-th power)

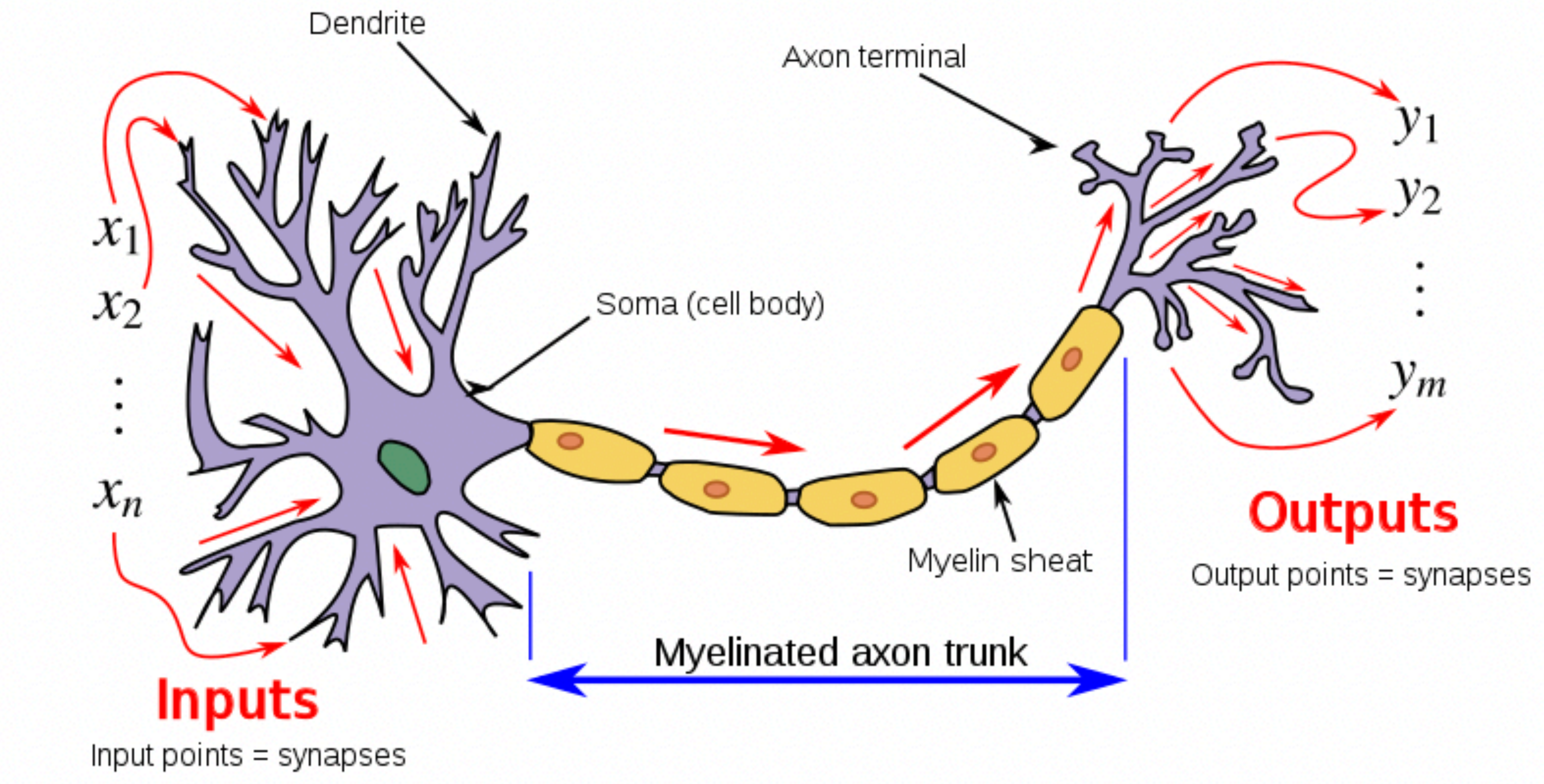
Neural Nets

Hypothesis Sets

Neural Nets are a a particularly interesting class to build hypothesis spaces with.

Build complexity by composing many very simple building blocks: the “artificial neuron”





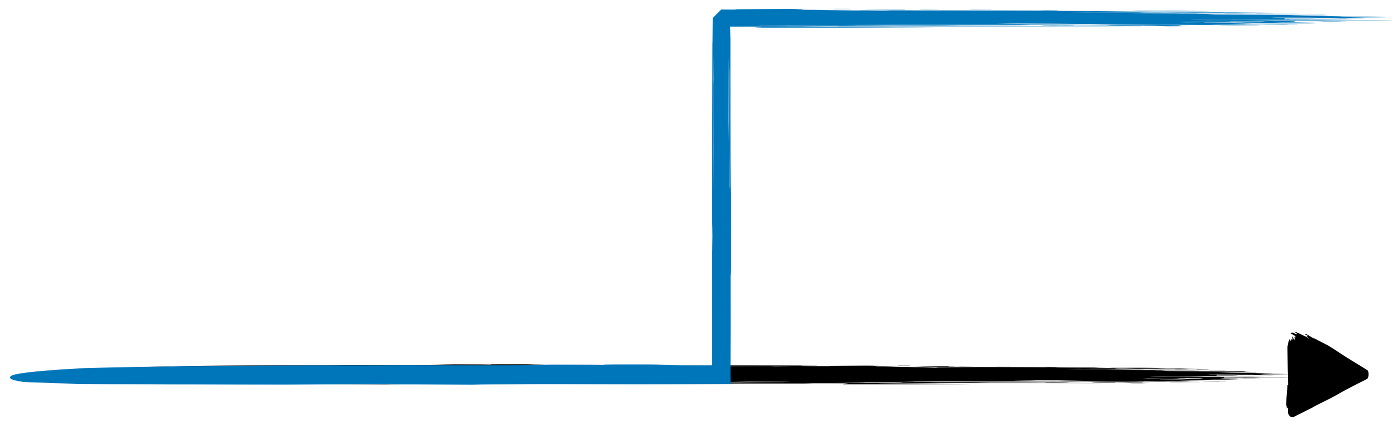
The Perceptron

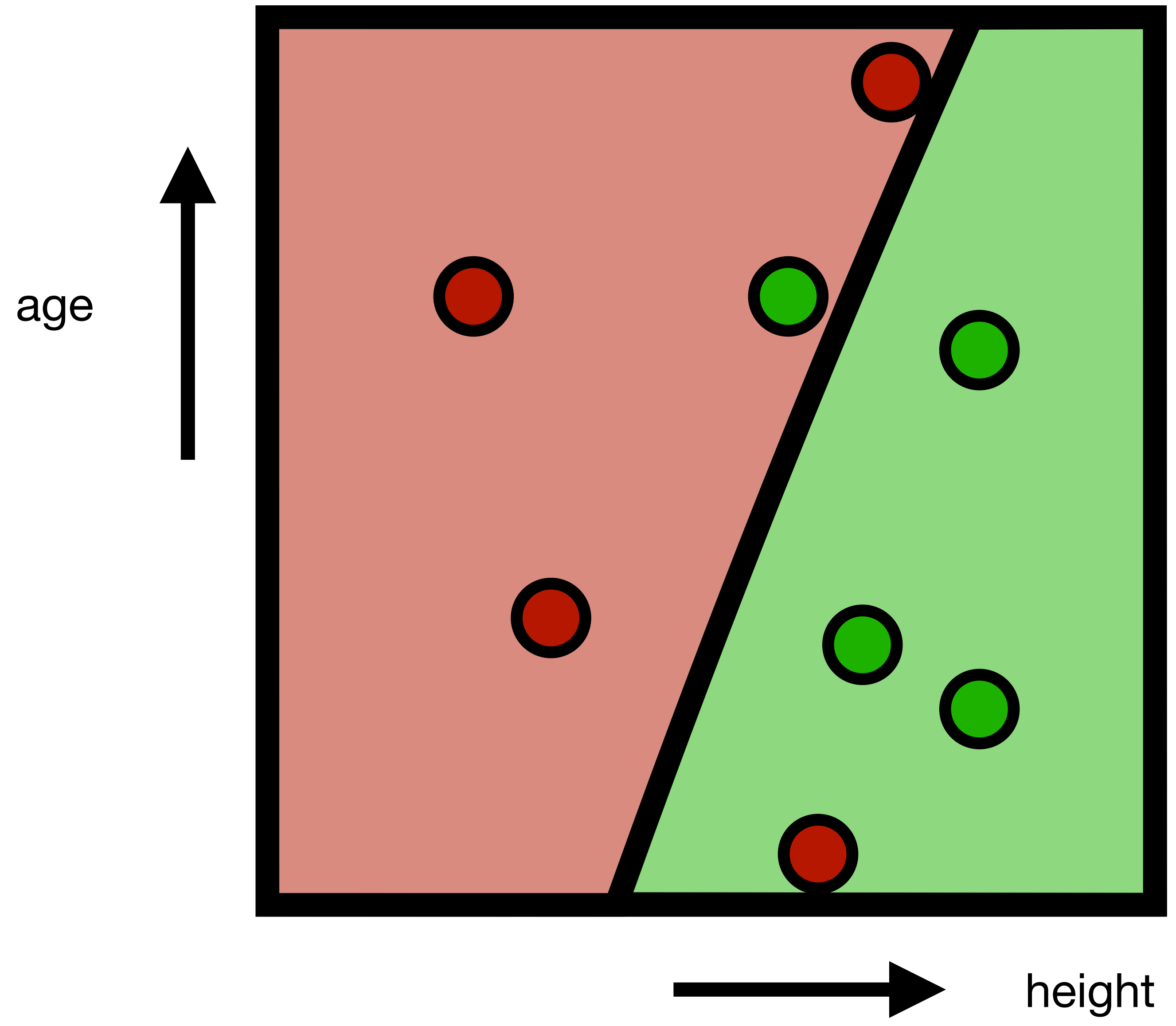
A single neuron, binary output & linear decision boundaries

weight

$$\theta\left(\sum_i w_i x_i + b\right)$$

bias

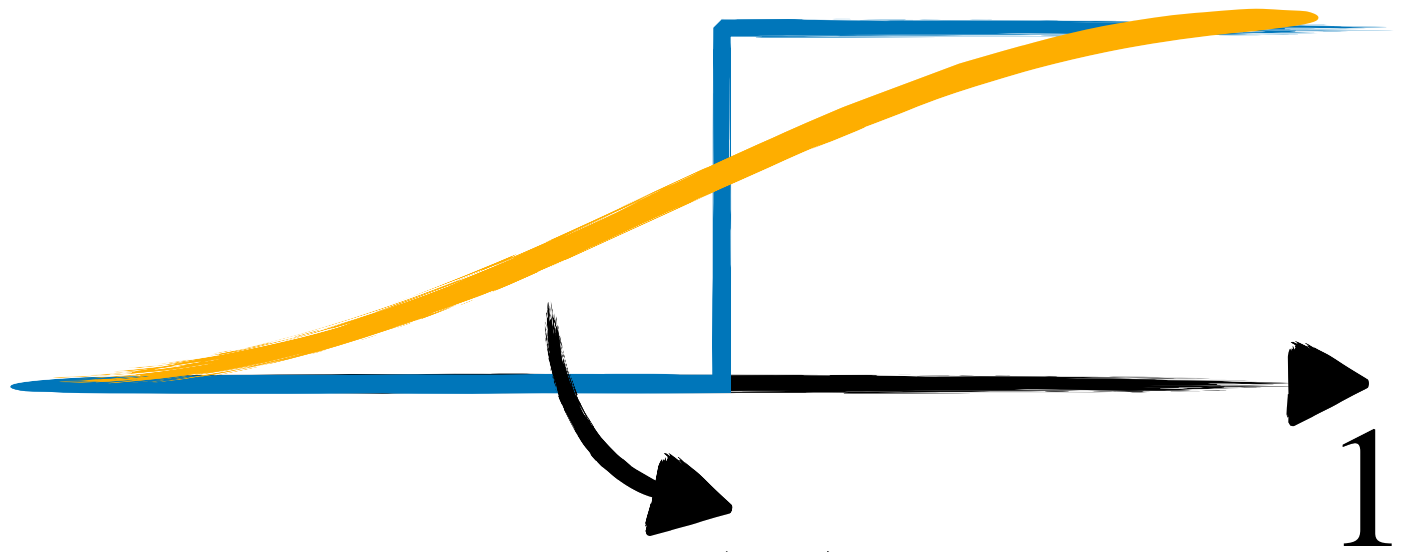




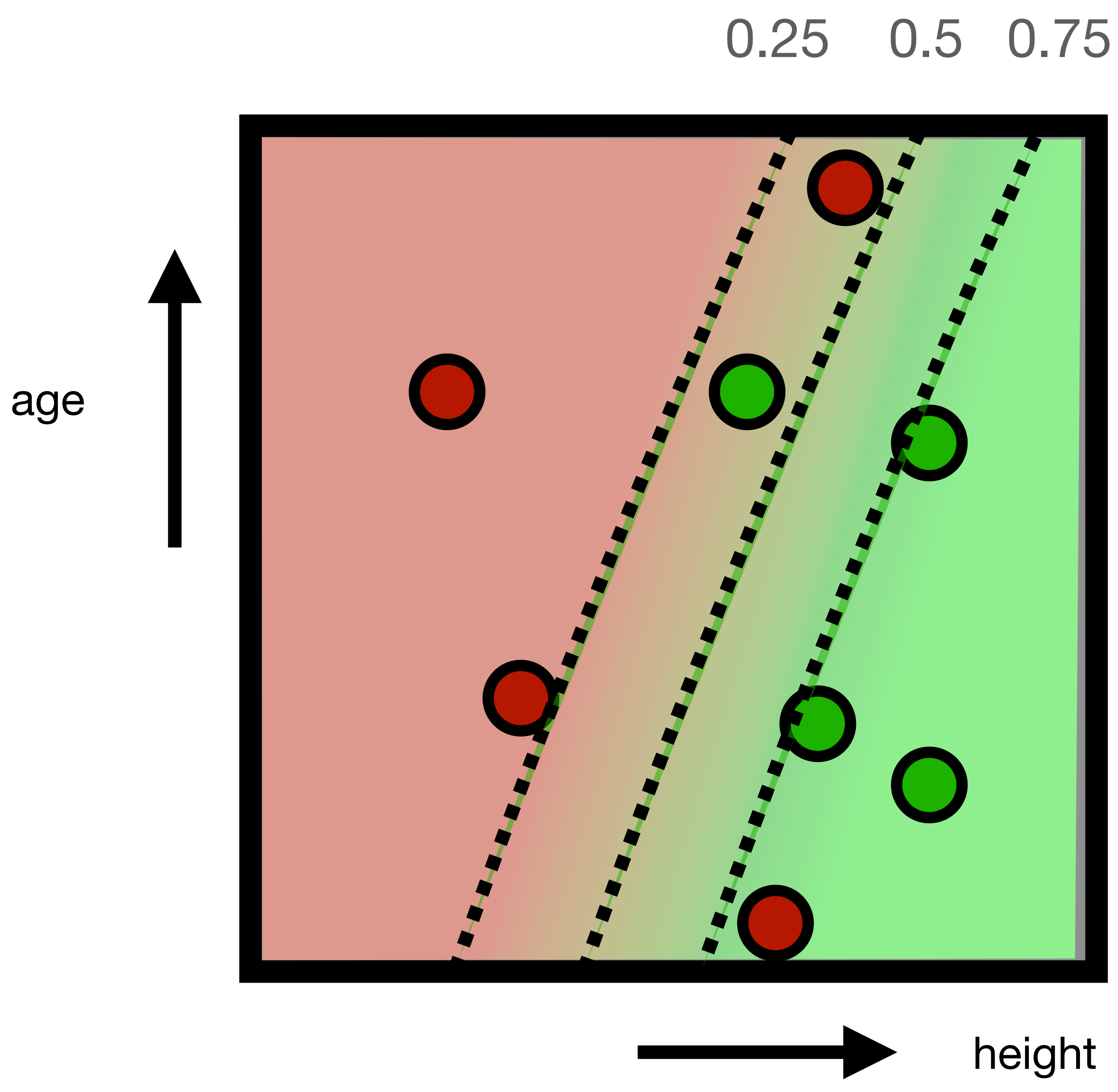
The Perceptron

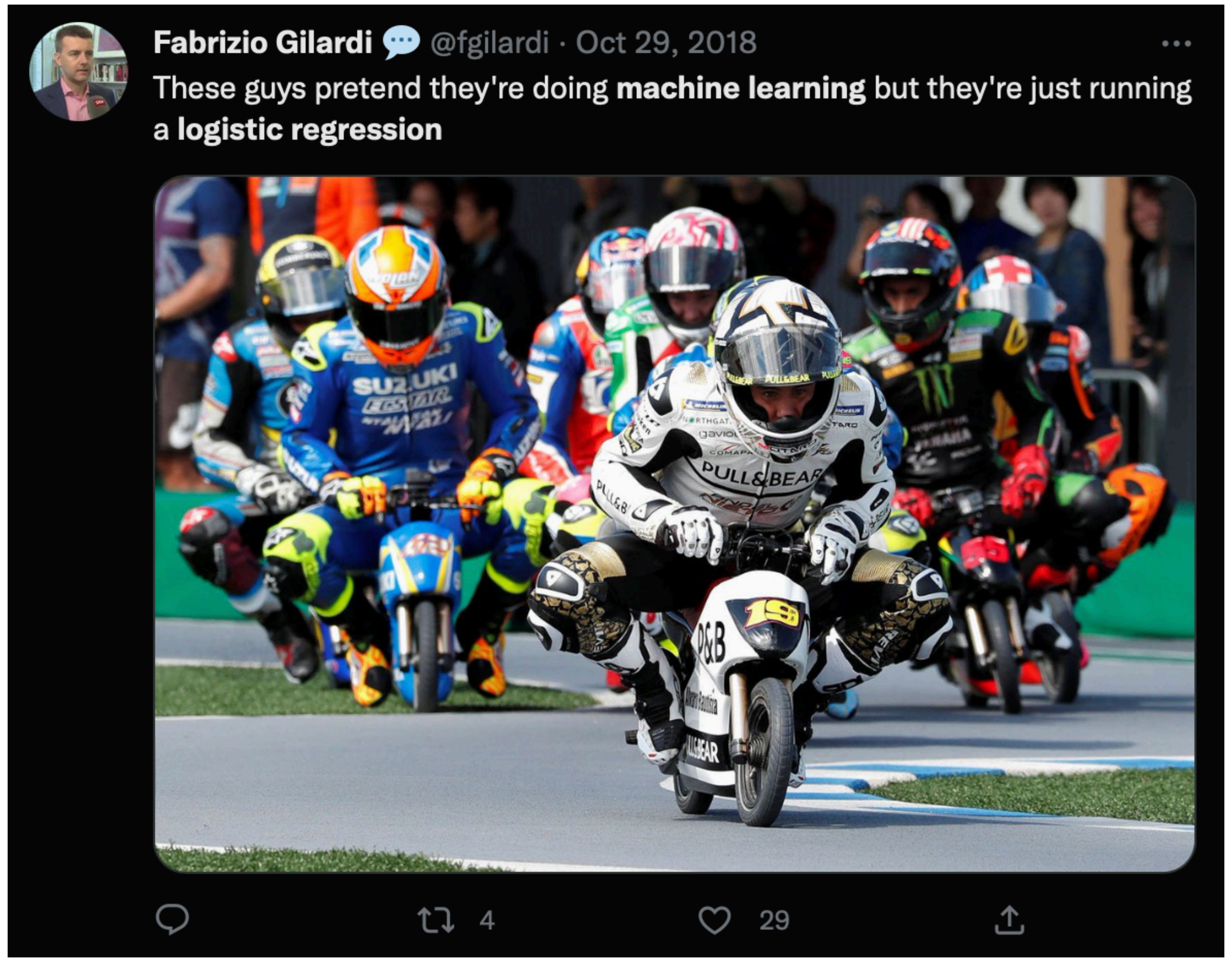
It may be preferable to get more of a probabilistic interpretation of the decision ($q(z = 1 \mid x)$), instead of a hard decision.

$$\sigma\left(\sum_i w_i x_i + b\right)$$



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

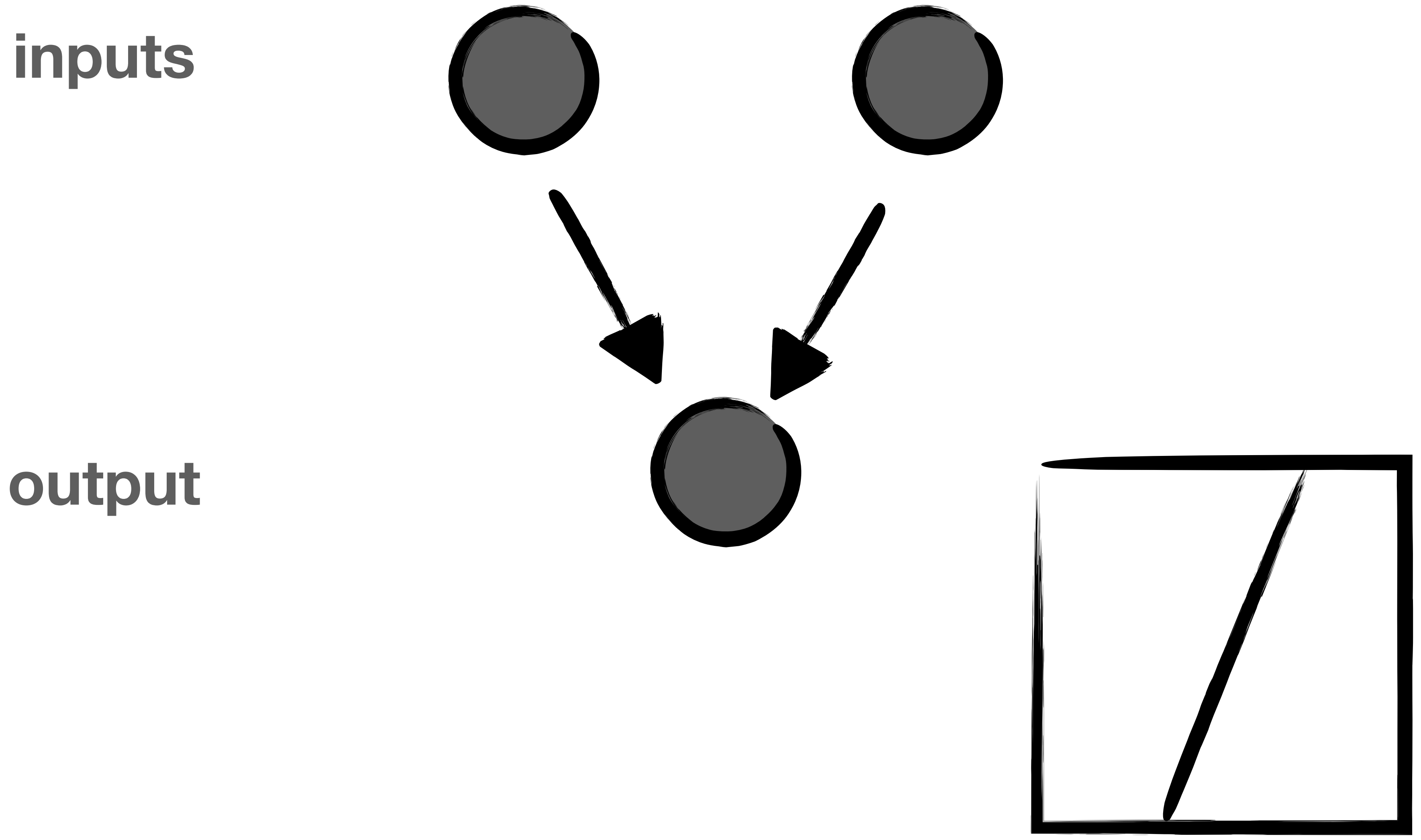


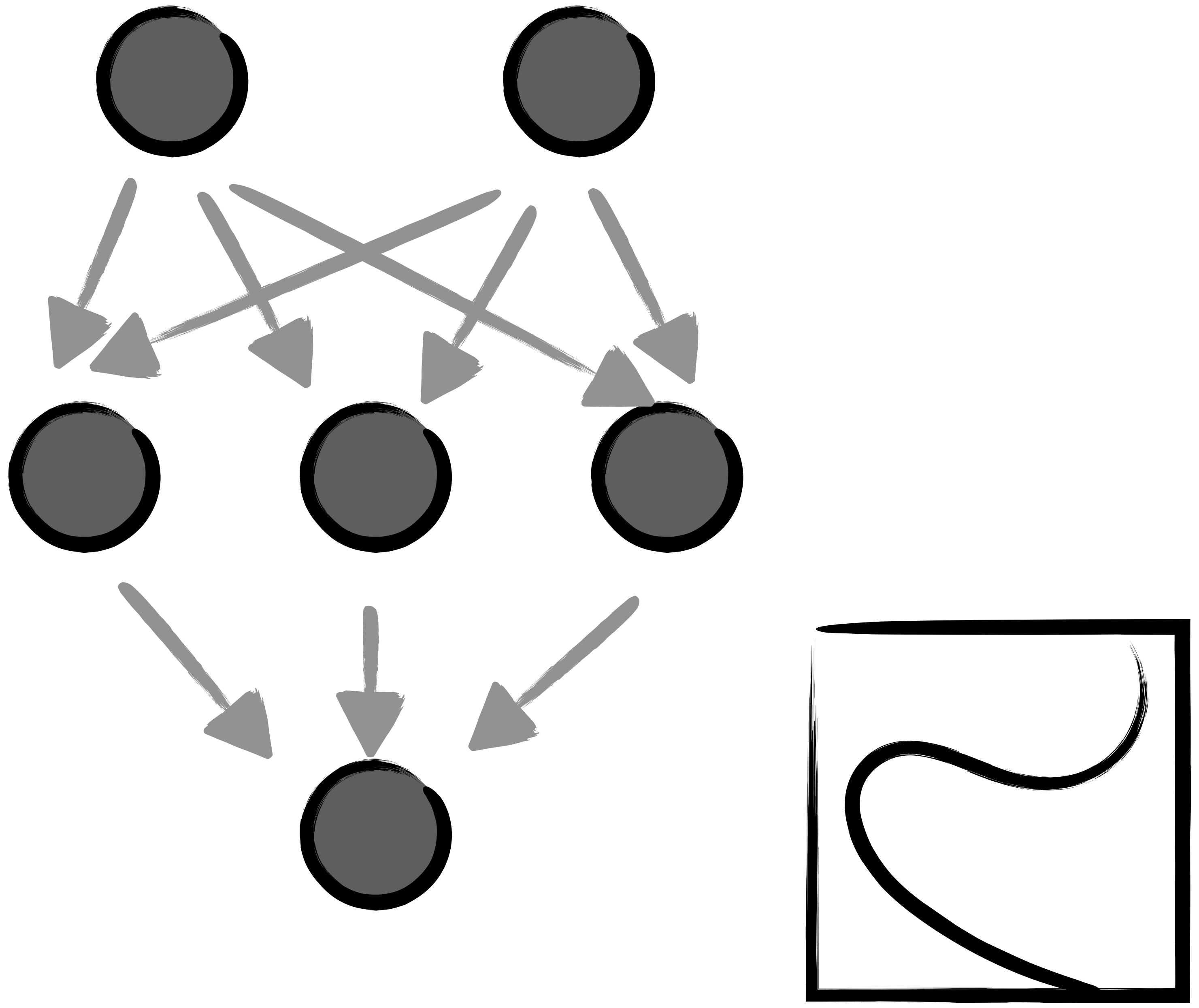


Beyond the Perceptron

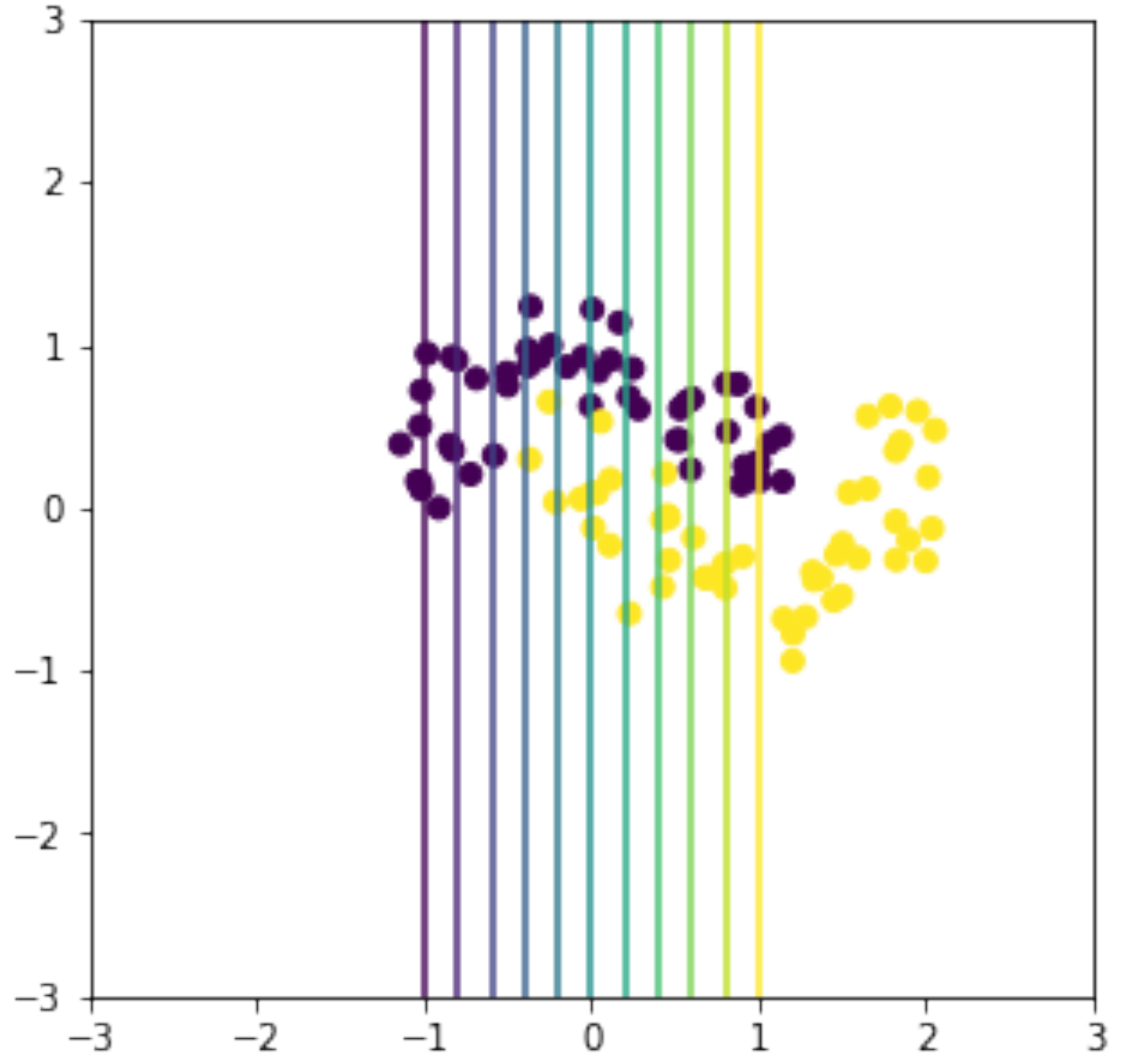
A bit boring, can we do something more complicated?

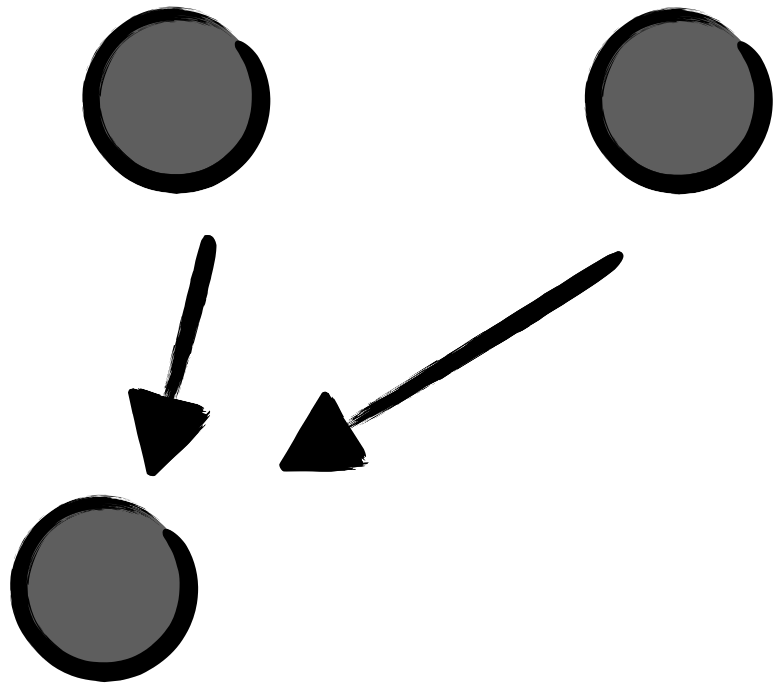
Instead of a **single neuron** we can combine the results of many!





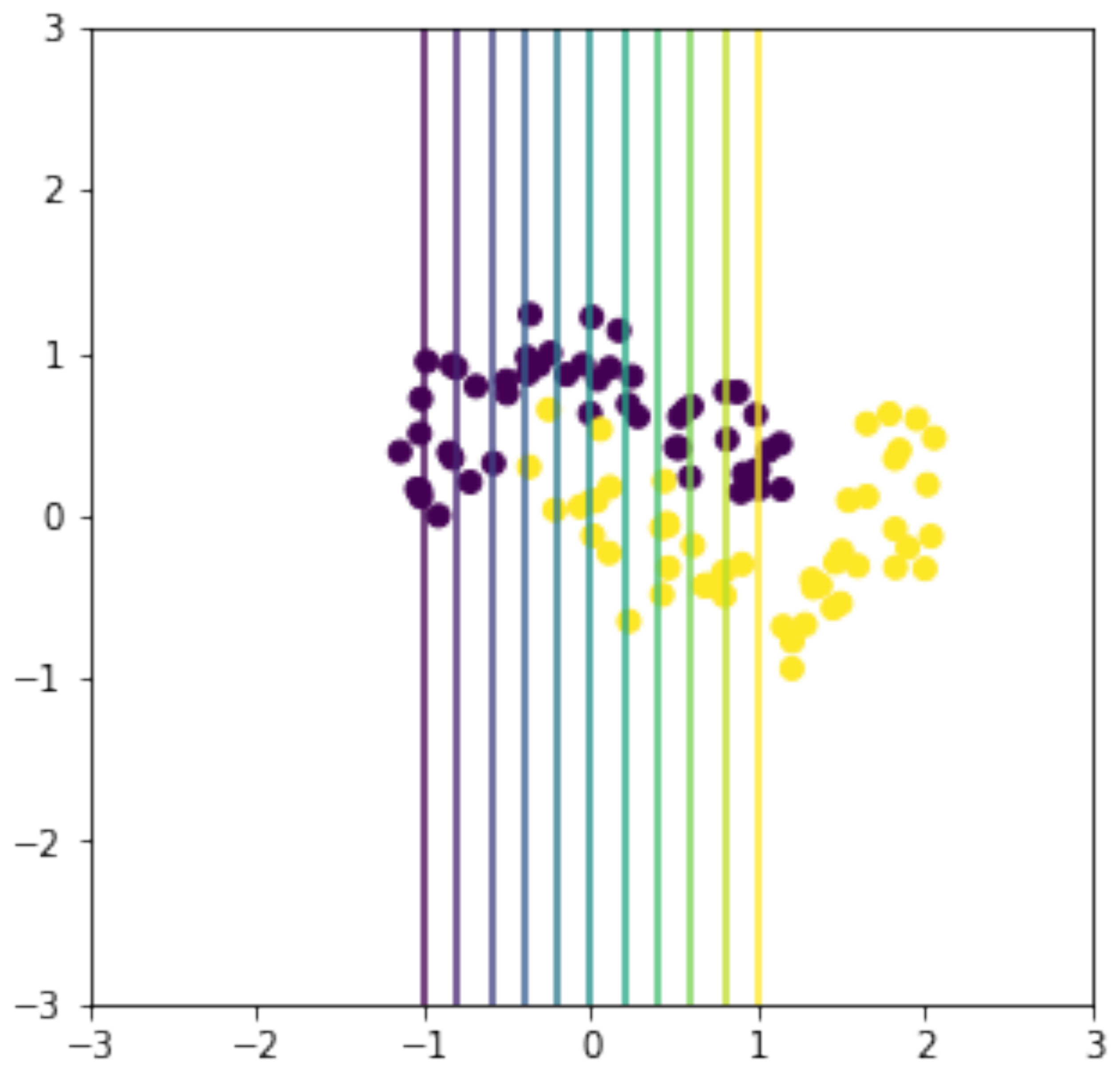
Going Complex

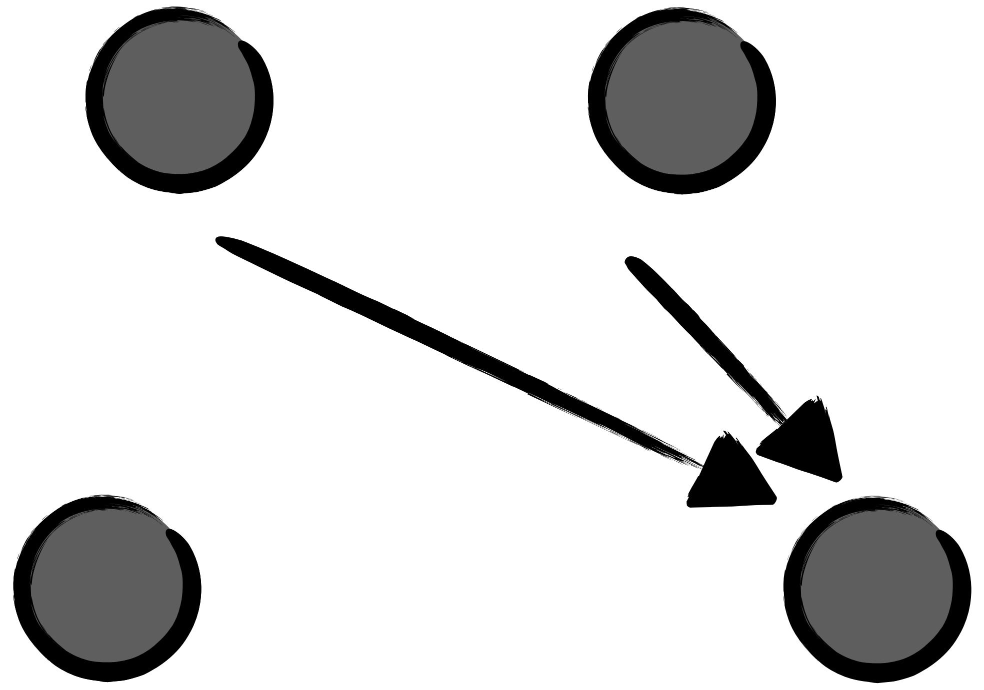


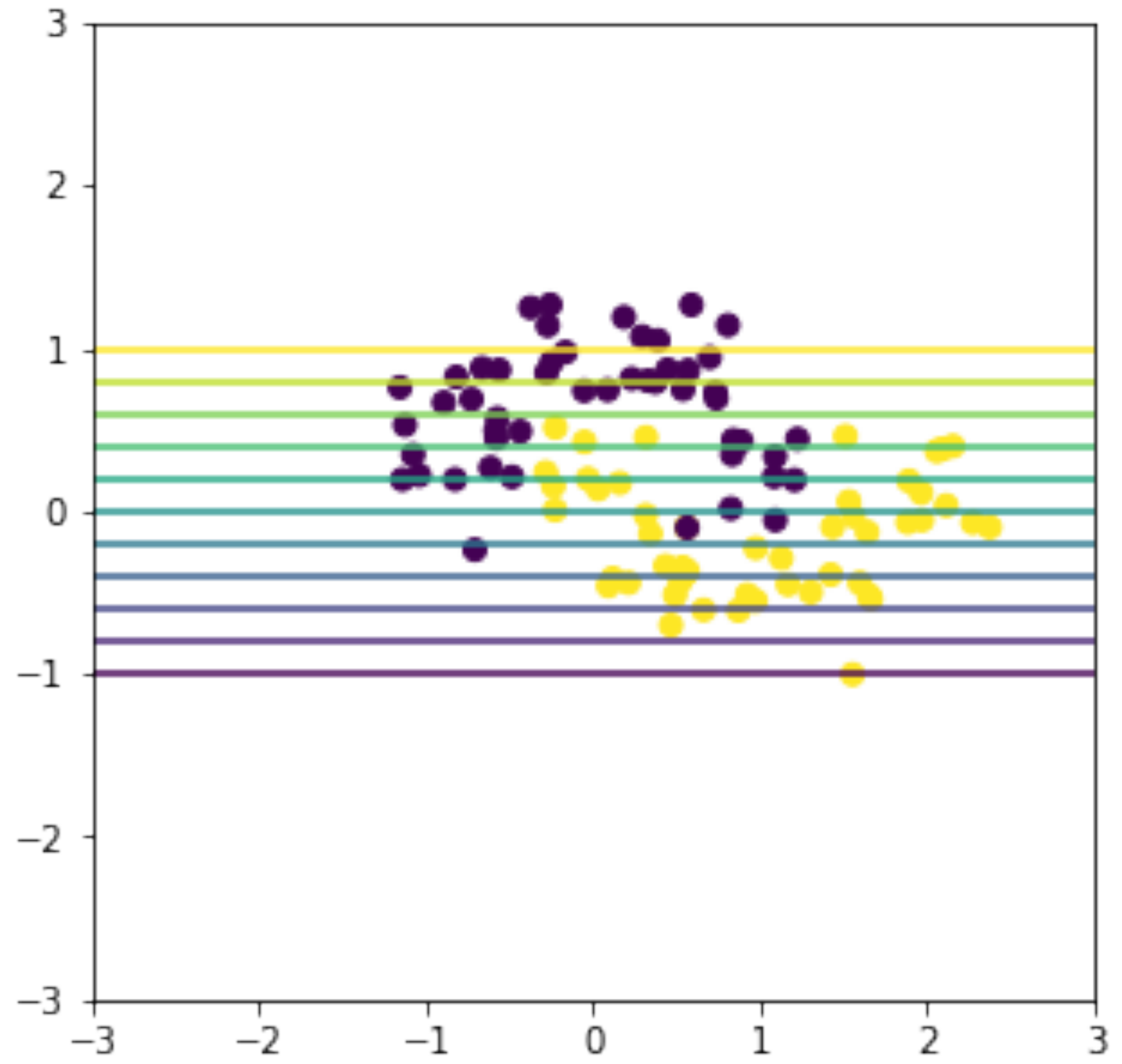


E.g. maybe combine these two decision boundaries?

Going Complex

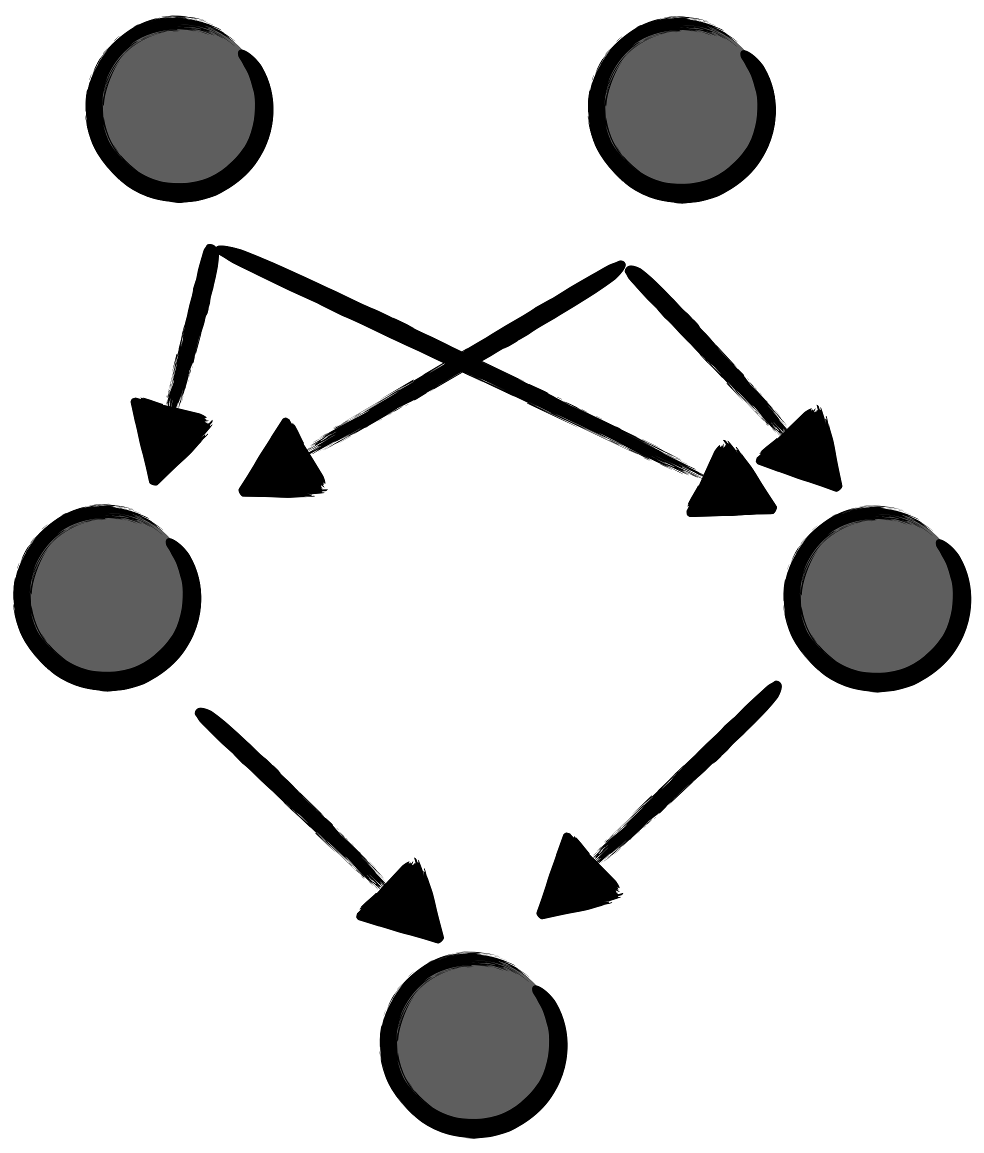






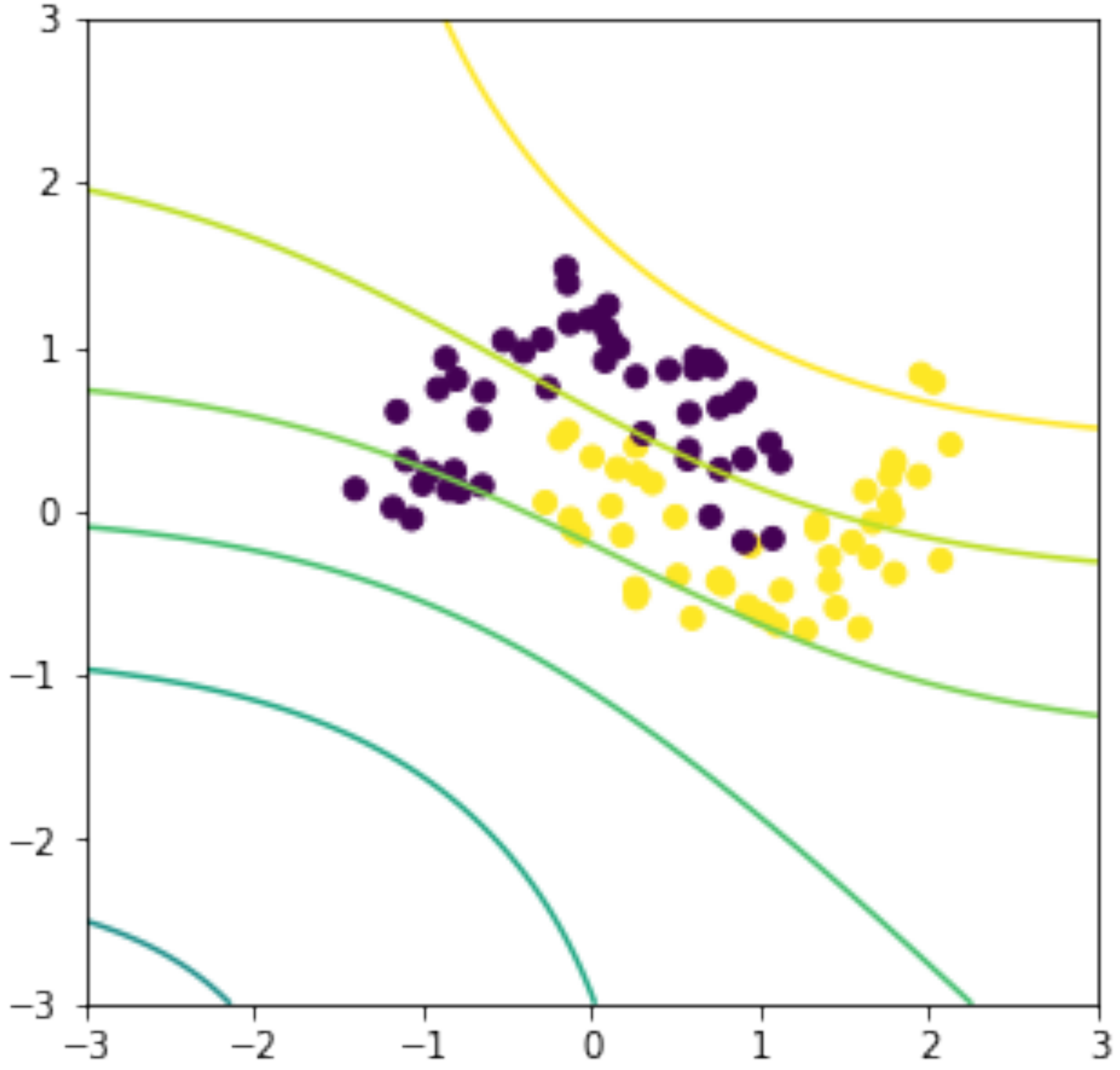
E.g. maybe combine these two decision boundaries?

Going Complex



E.g. maybe combine these two decision boundaries?

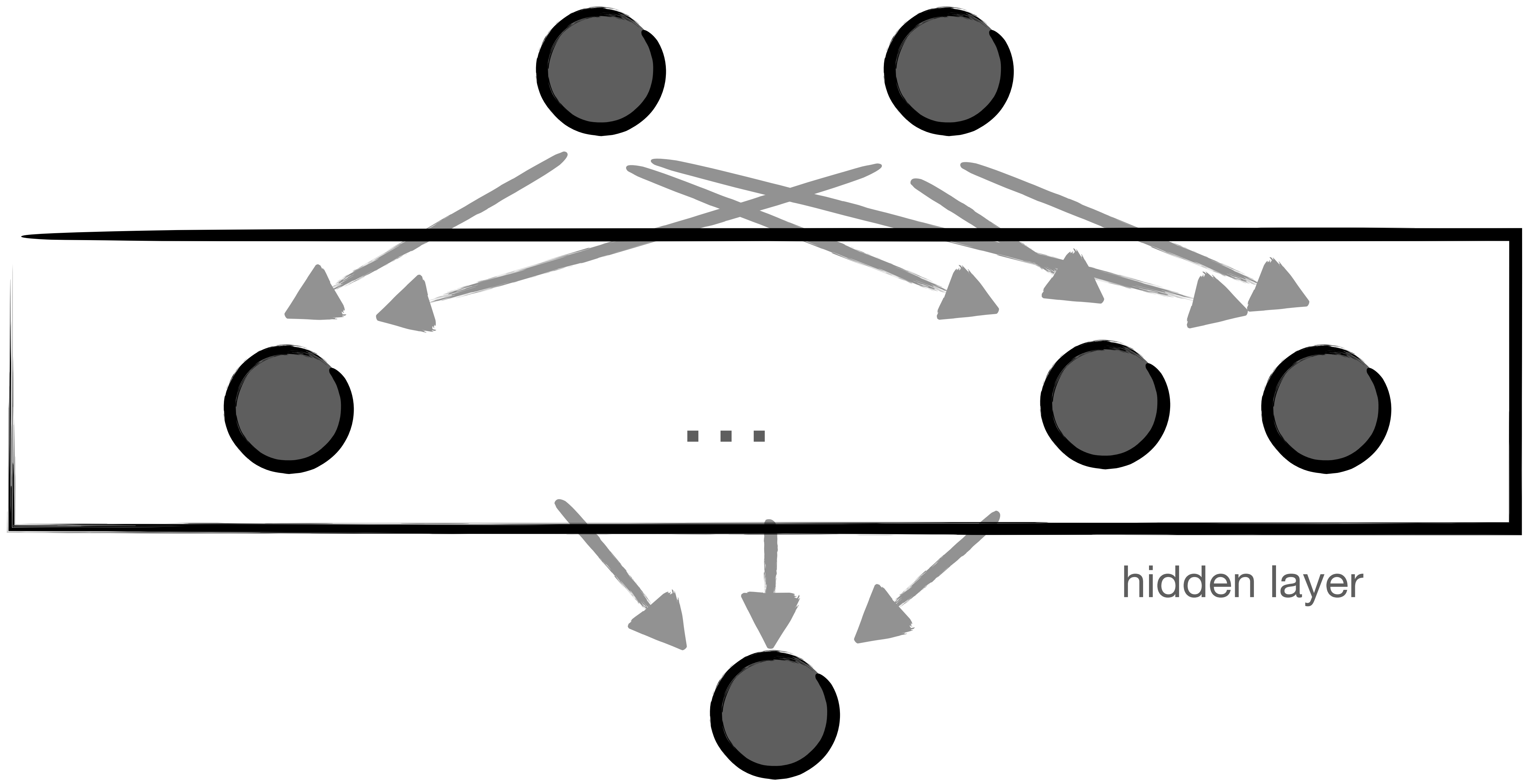
Success!

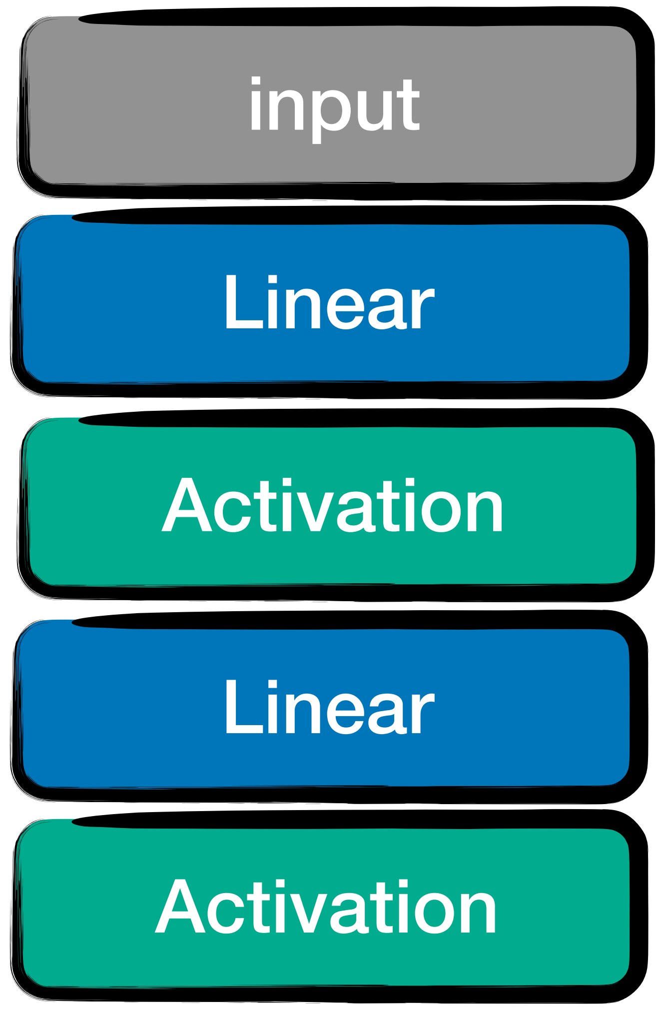


Linear Combination of non-Linear decisions yield complex decision boundaries

What do we gain?

By combining non-linearly activating neurons, things don't get only a little better. We gain a lot!



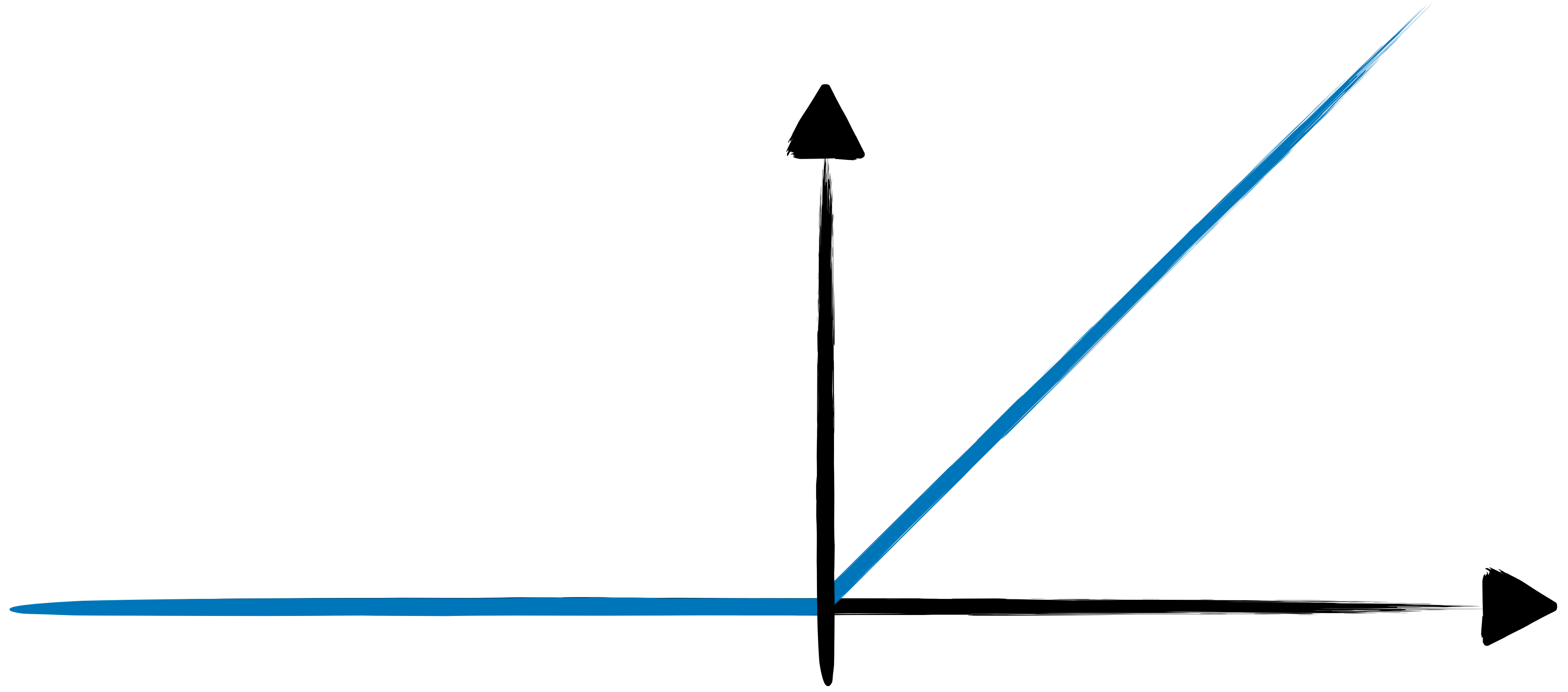


Neural Networks with a single hidden layer are universal function approximators!

Activation Functions

UFA is achieved with any non-linear activation function, not only a sigmoid like in the classic perceptron.

In practice, many use the simplest one you could think of:



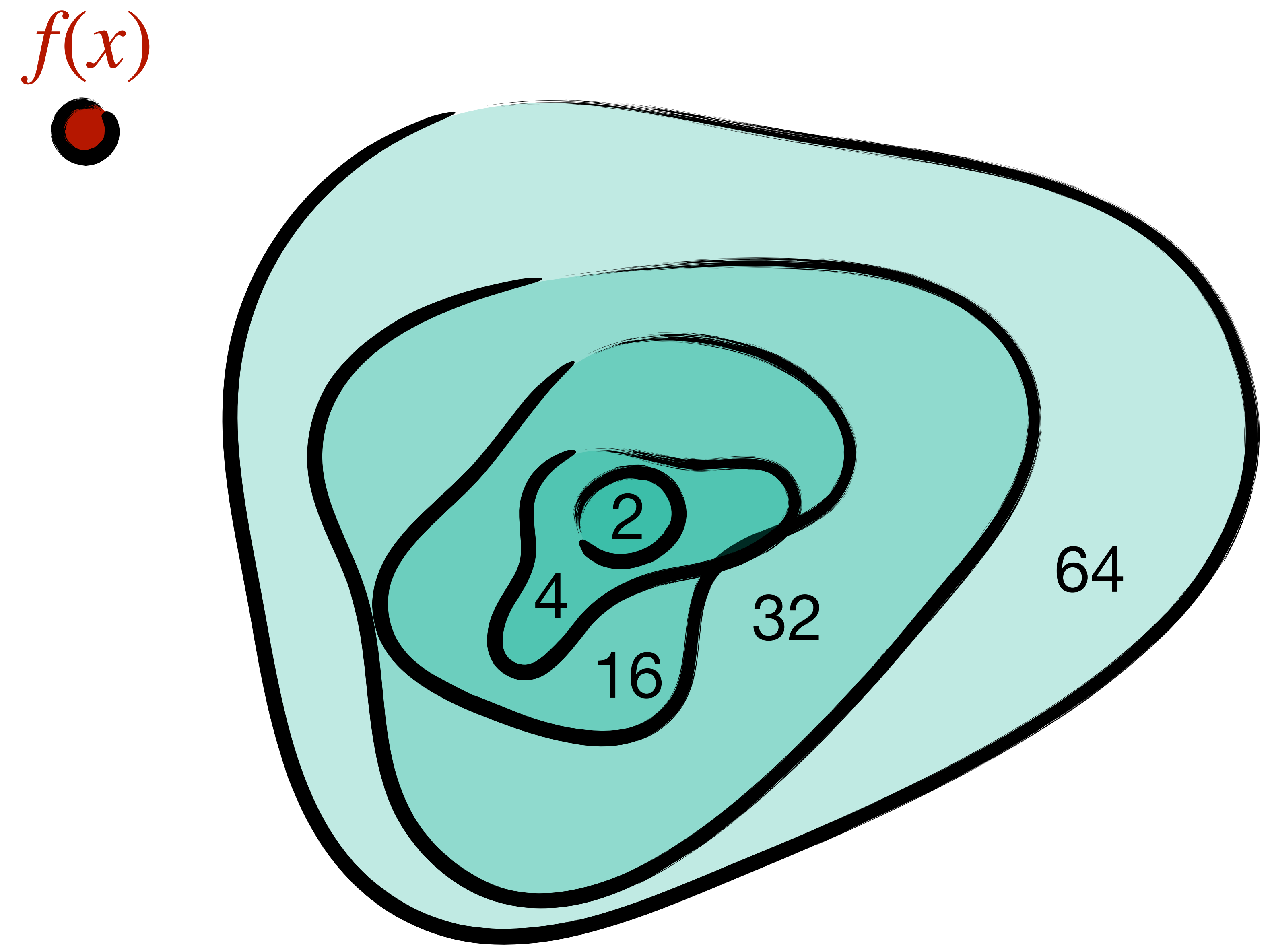
The rectifying linear unit (ReLU)

How big should we go?

With increasing size you get a better chance that the actual algorithm you are looking for lives within the hypothesis set.

Bias: the loss $L(h_{\text{min}})$ of the overall best function $\bar{h} \in \mathscr{H}$

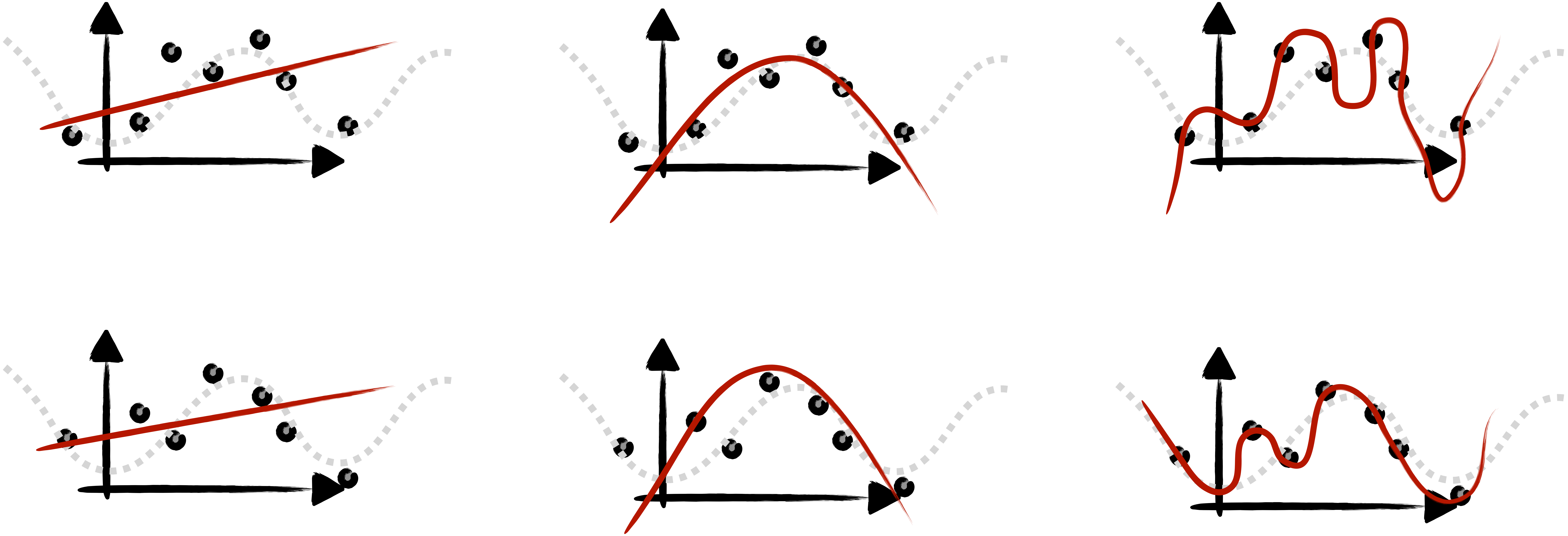
$$h_{\text{min}} = \bar{h} = \mathbb{E}_D h^*$$



An argument to make the hypothesis set as big as possible

But should we really?

"With four parameters I can fit an elephant, and with five I can make him wiggle his trunk."
- John von Neumann



Risk Functions

The Risk we want

In statistical learning we are interested in the **expected performance of the algorithm** on <u>future</u> data.

With assumption of i.i.d. distribution of data:

Performance of the hypothesis for a specific input

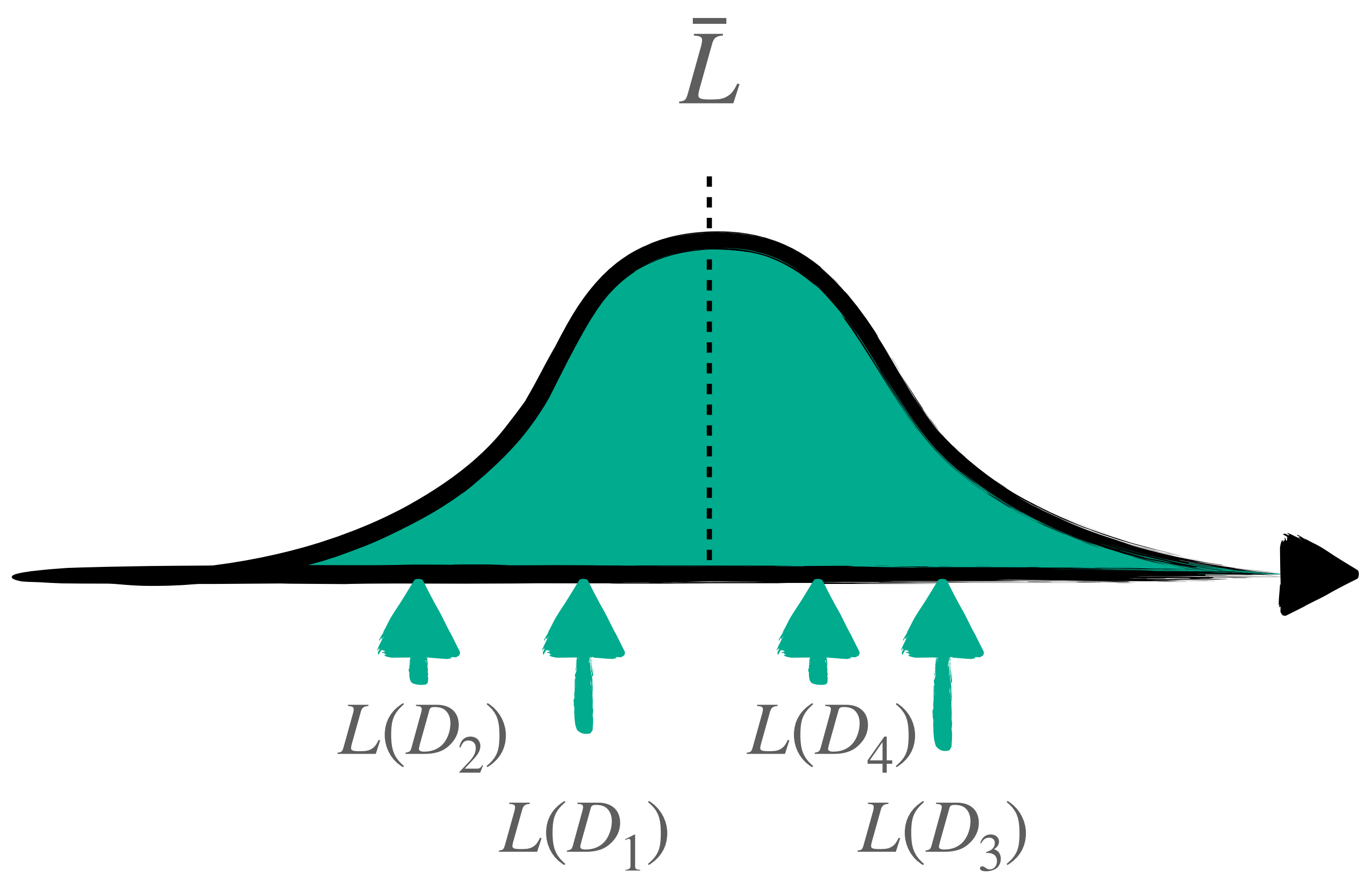
$$L(h) = \mathbb{E}_{p(s)} L(s, h)$$

Distribution of possible inputs

The Risk we can get

While we don't have $p(s)$, we do have samples $s \sim p(s)$
→ we can only estimate the risk **empirically as a proxy!**

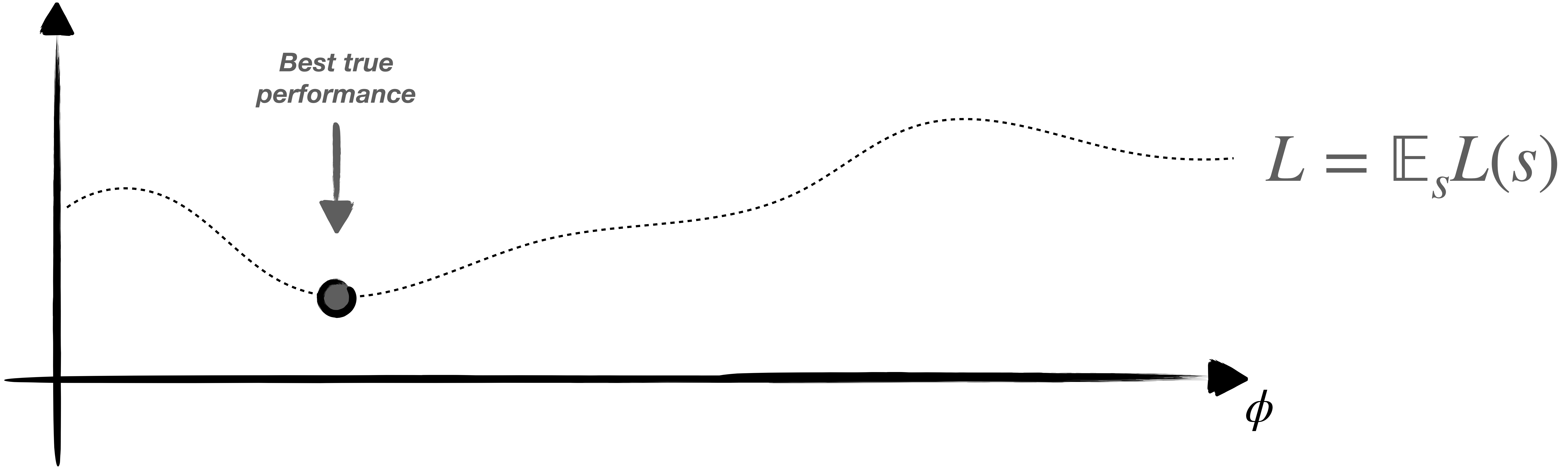
$$\bar{L} = \int_S p(s)L(s,h) \rightarrow \hat{L} = \frac{1}{N}\sum_i L(s_i, h)$$



This switch between what we want and what we can get has tricky consequences

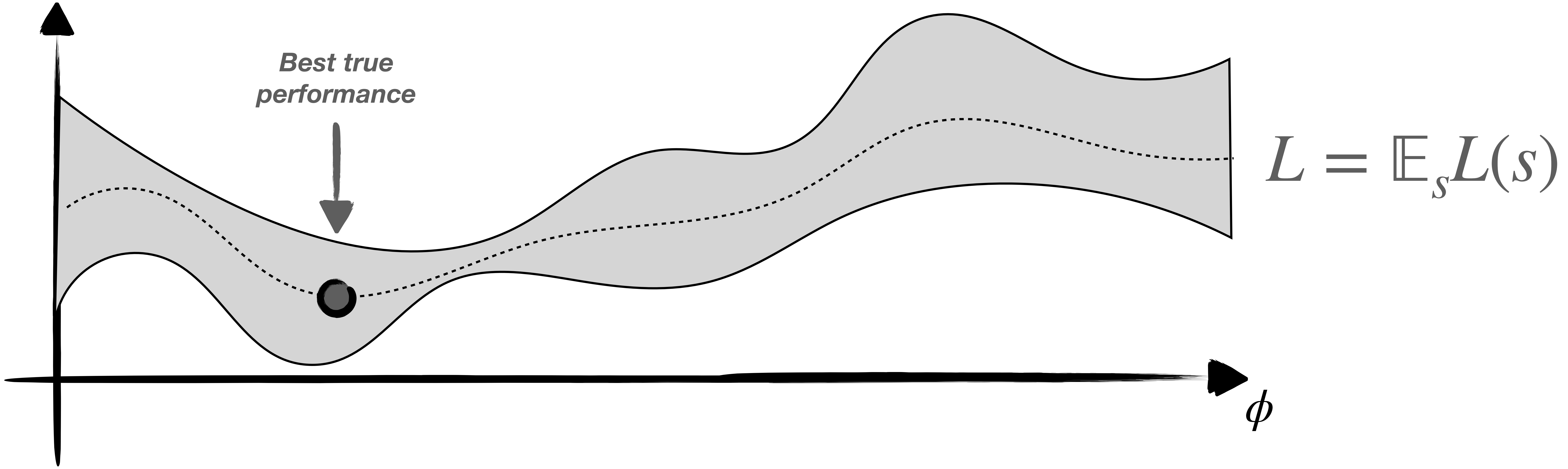
Empirical Risk Minimization

But, we have to keep in mind that it's just a proxy that depends **training dataset** we have!



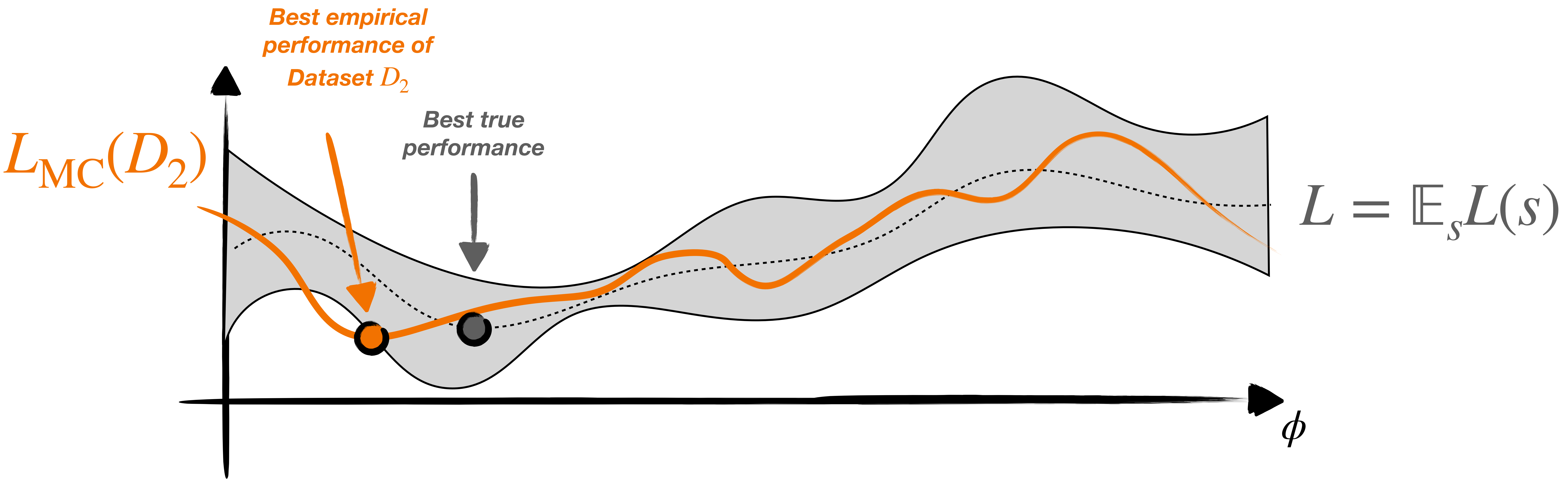
Empirical Risk Minimization

But, we have to keep in mind that it's just a proxy that depends **training dataset** we have!



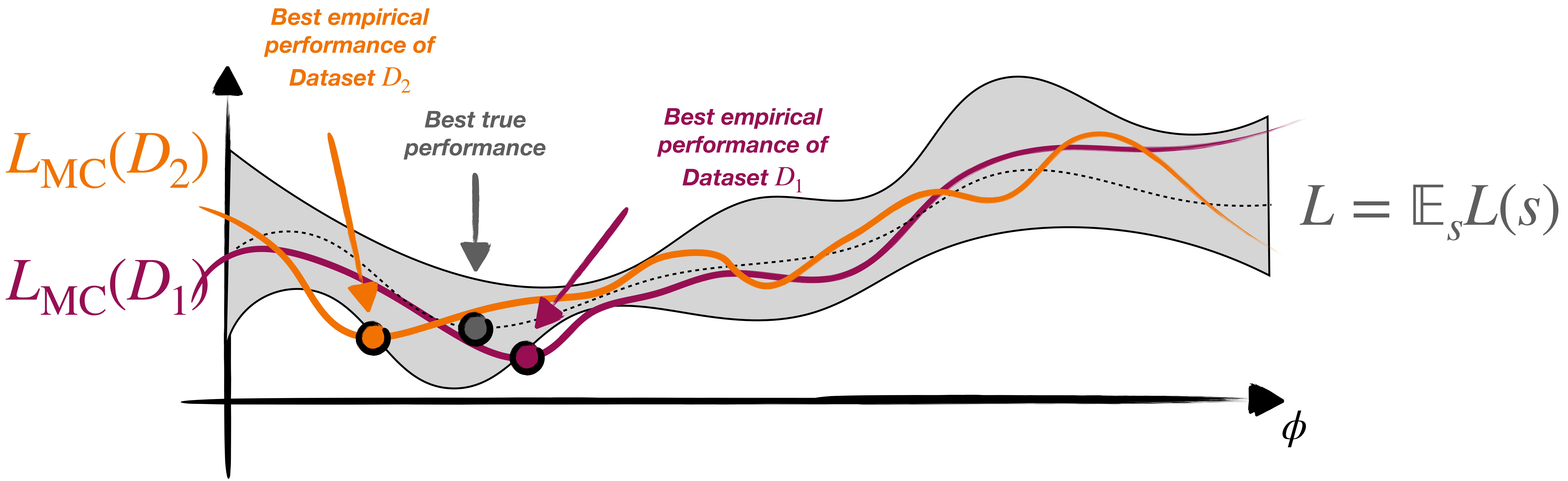
Empirical Risk Minimization

But, we have to keep in mind that it's just a proxy that depends **training dataset** we have!



Empirical Risk Minimization

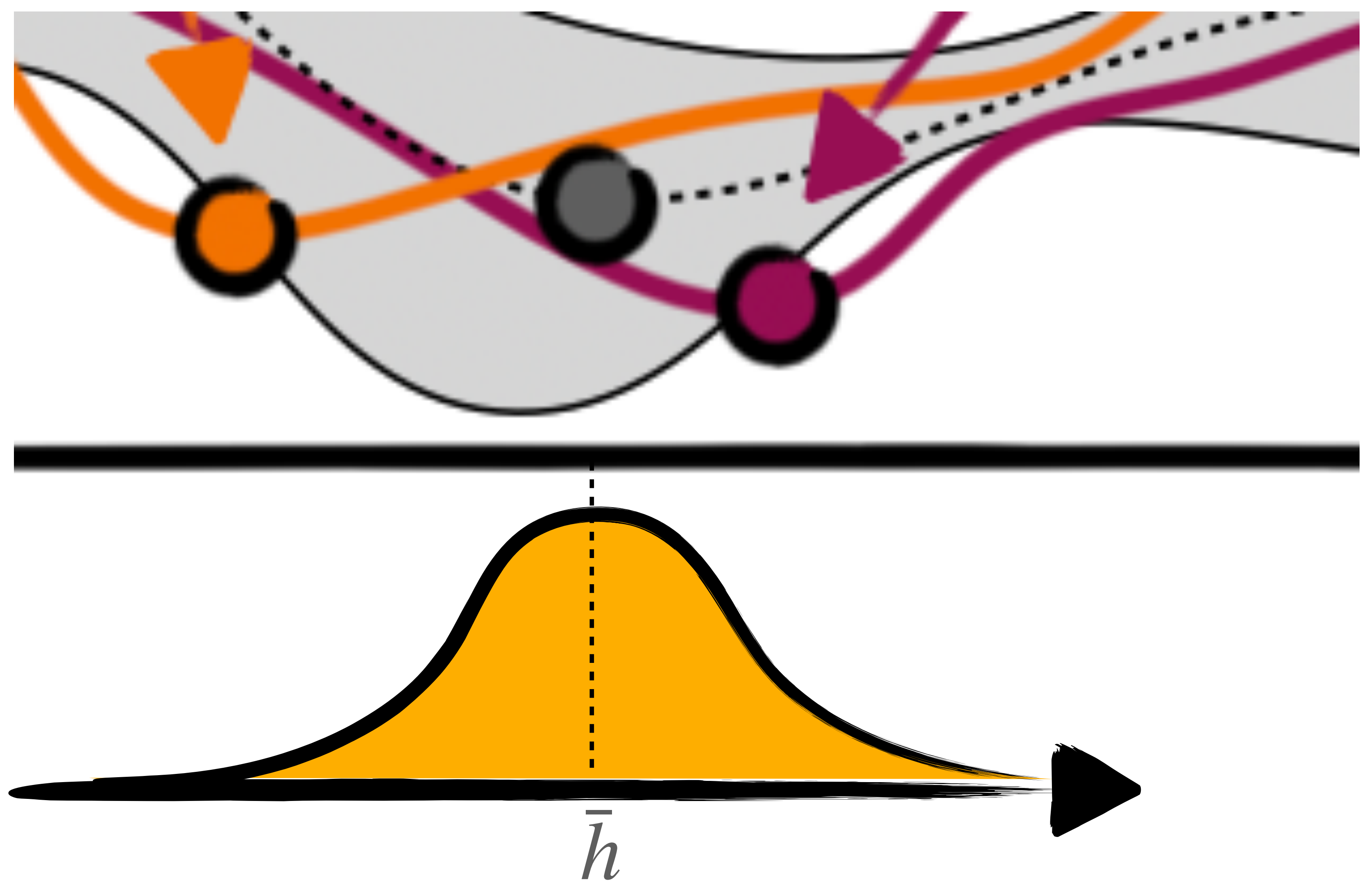
But, we have to keep in mind that it's just a proxy that depends **training dataset** we have!



Empirical Risk Minimization

In empirical risk minimization, the selected final hypothesis is **distributed** around the actual best hypothesis in the set

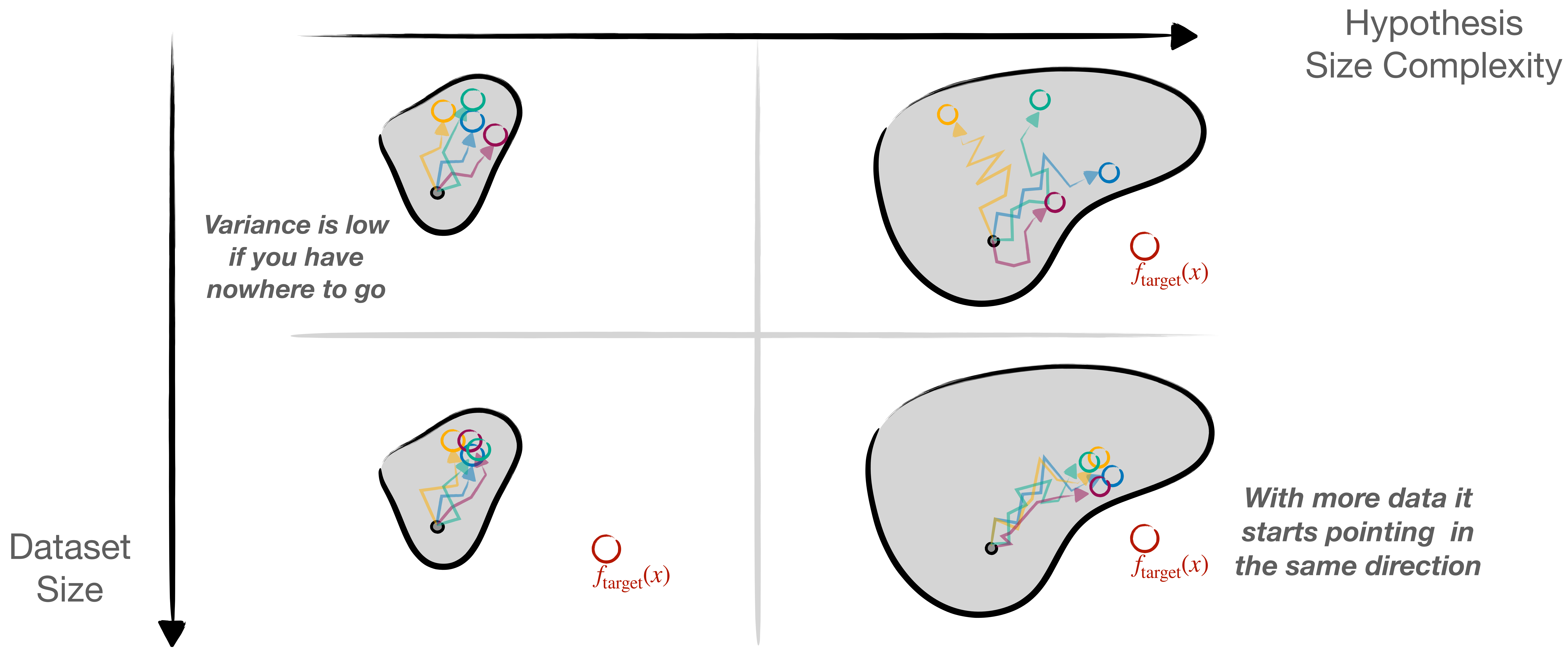
Variance: spread of the distribution of h^* in $\mathscr{H}$



True Loss of h^* will be worse than the best possible one in $\mathscr{H}$ (Bias)

$$\bar{L}_{h^*} = \bar{L}_{\text{bias}} + \Delta L_{\text{var}}$$

Increases with $\mathscr{H}$, decreases with N

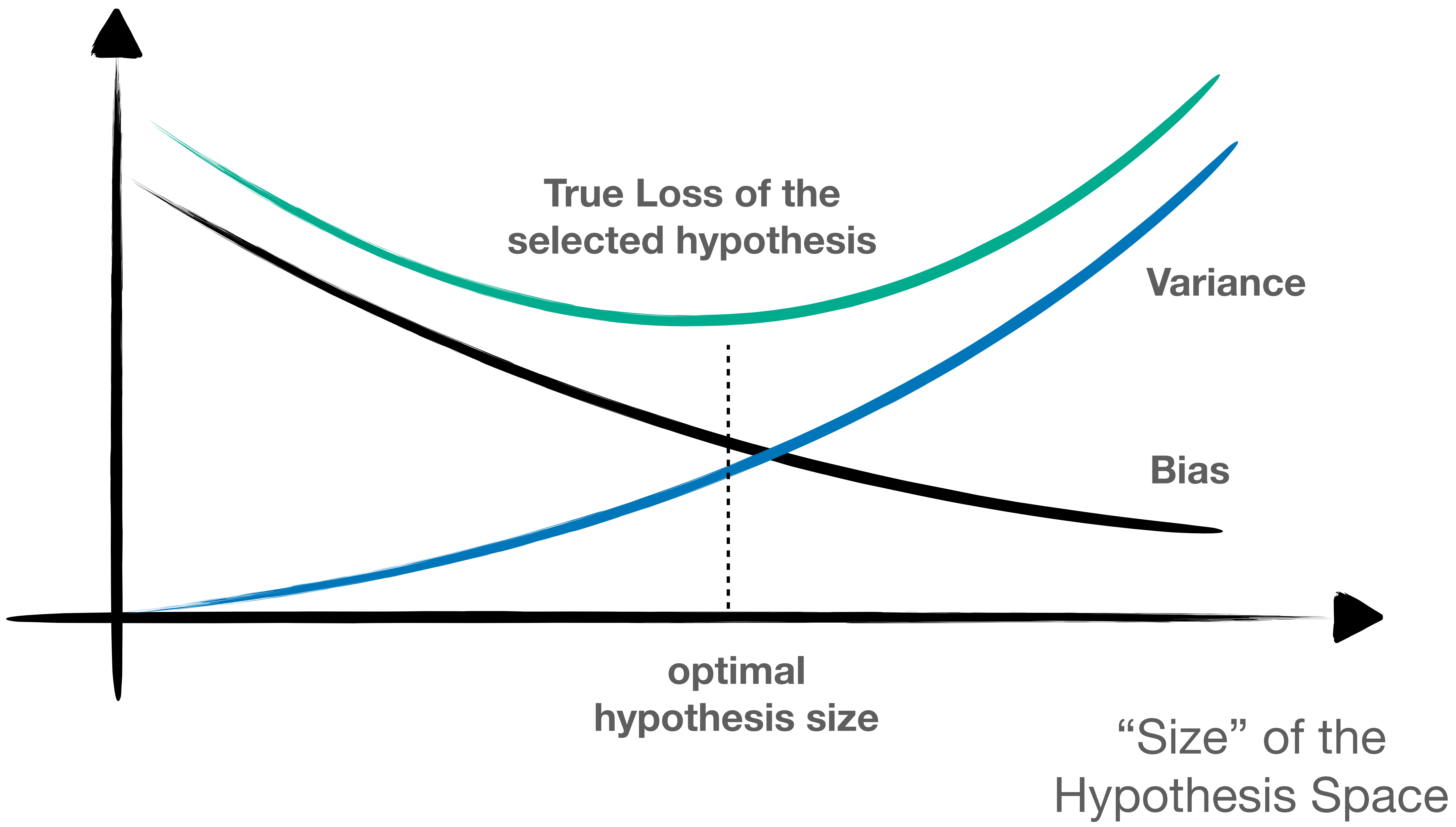


An argument to make the hypothesis set as small as possible

Bias Variance Tradeoff

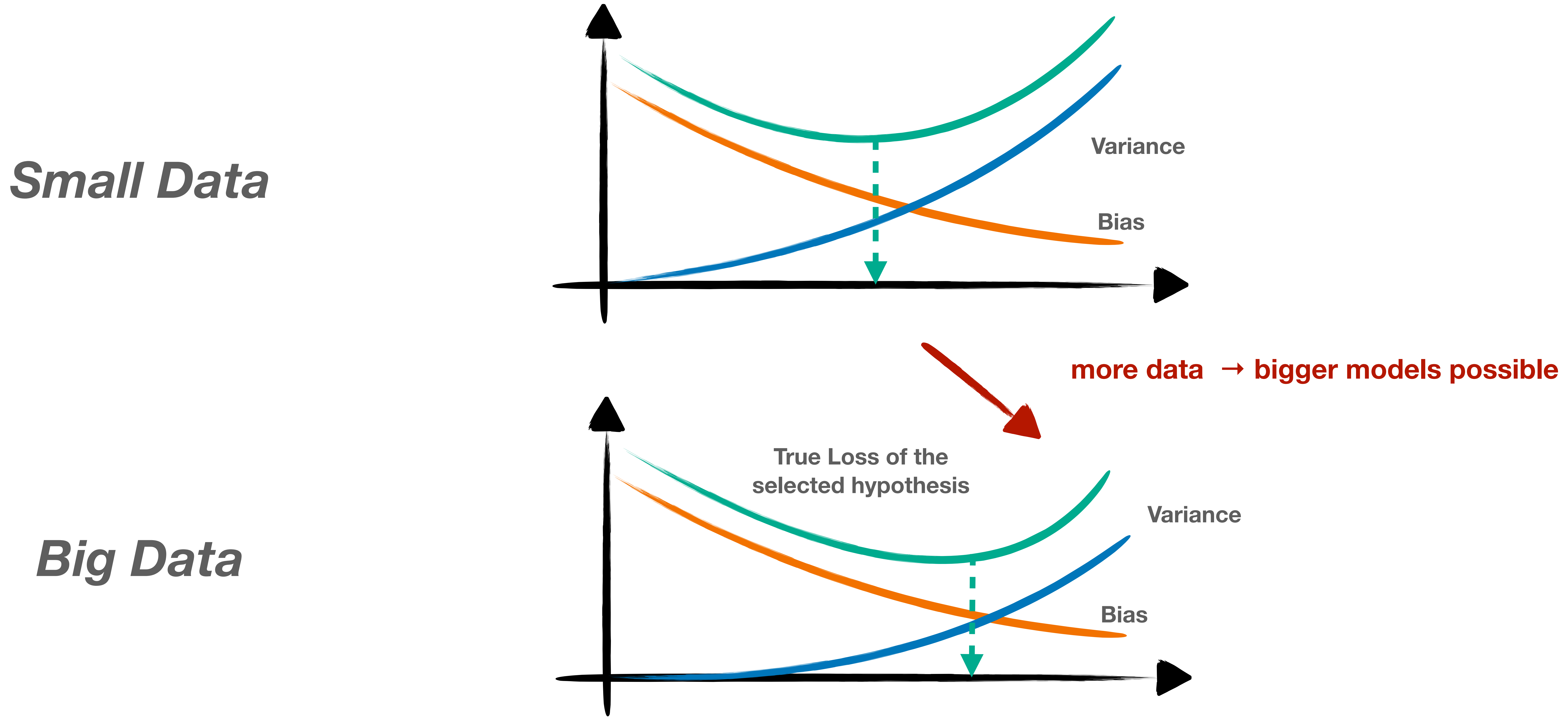
We have now two competing forces

- make the model space as big as possible: **reduce bias**
- constrain the model space: **reduce variance**



Big Networks require big data!

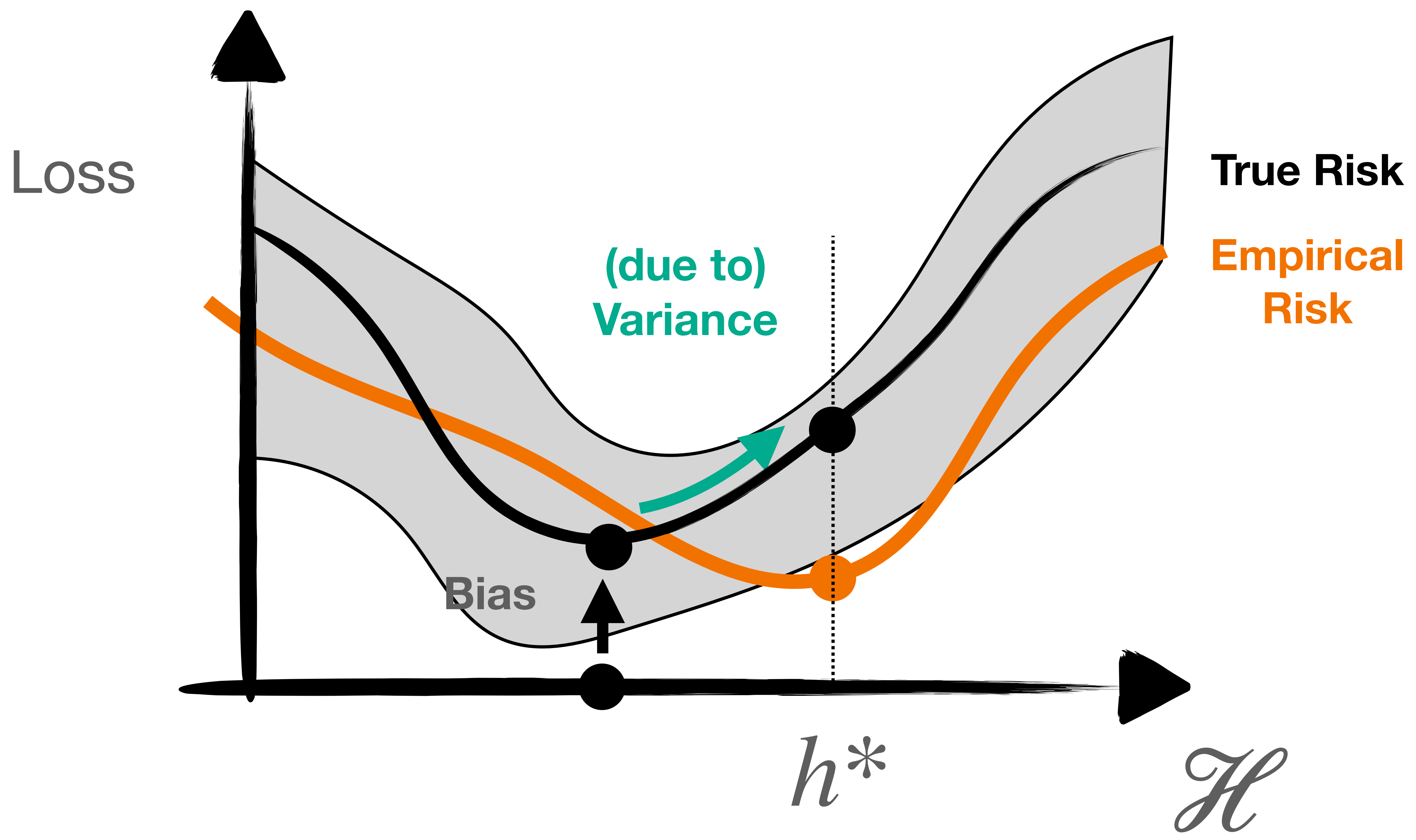
If you don't have enough of it, you simply cannot afford to train a billion parameter model!

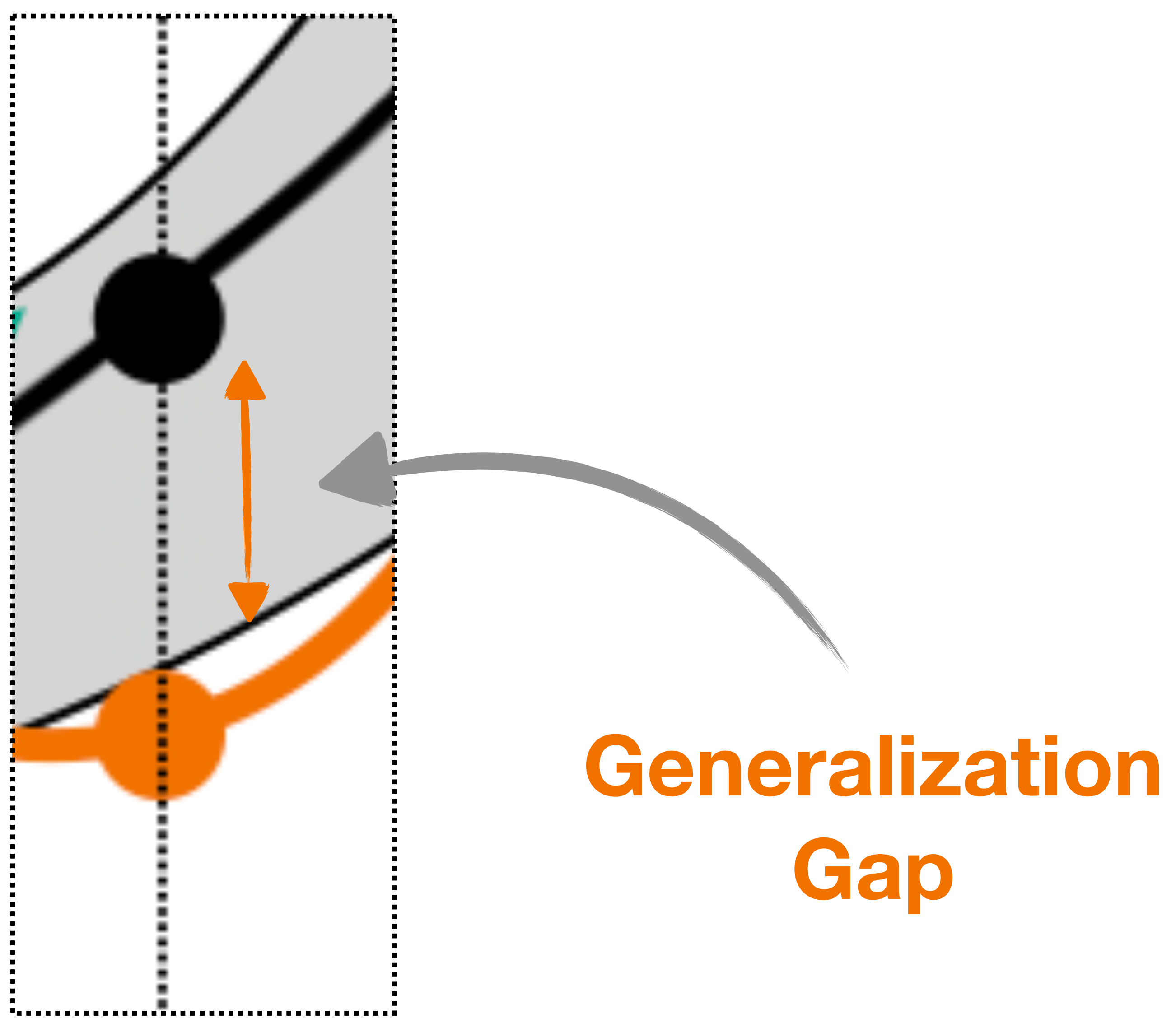


Empirical Risk Minimization

Both Bias and Variance talk about the true risk. But we found the final hypothesis through minimizing the empirical risk

What is the true risk of our hypothesis?





The value of we measure as empirical risk isn't reliable

Why?

The Data would be used for two things at once:

- selecting the hypothesis (based on risk)
- estimating the performance of the hypothesis

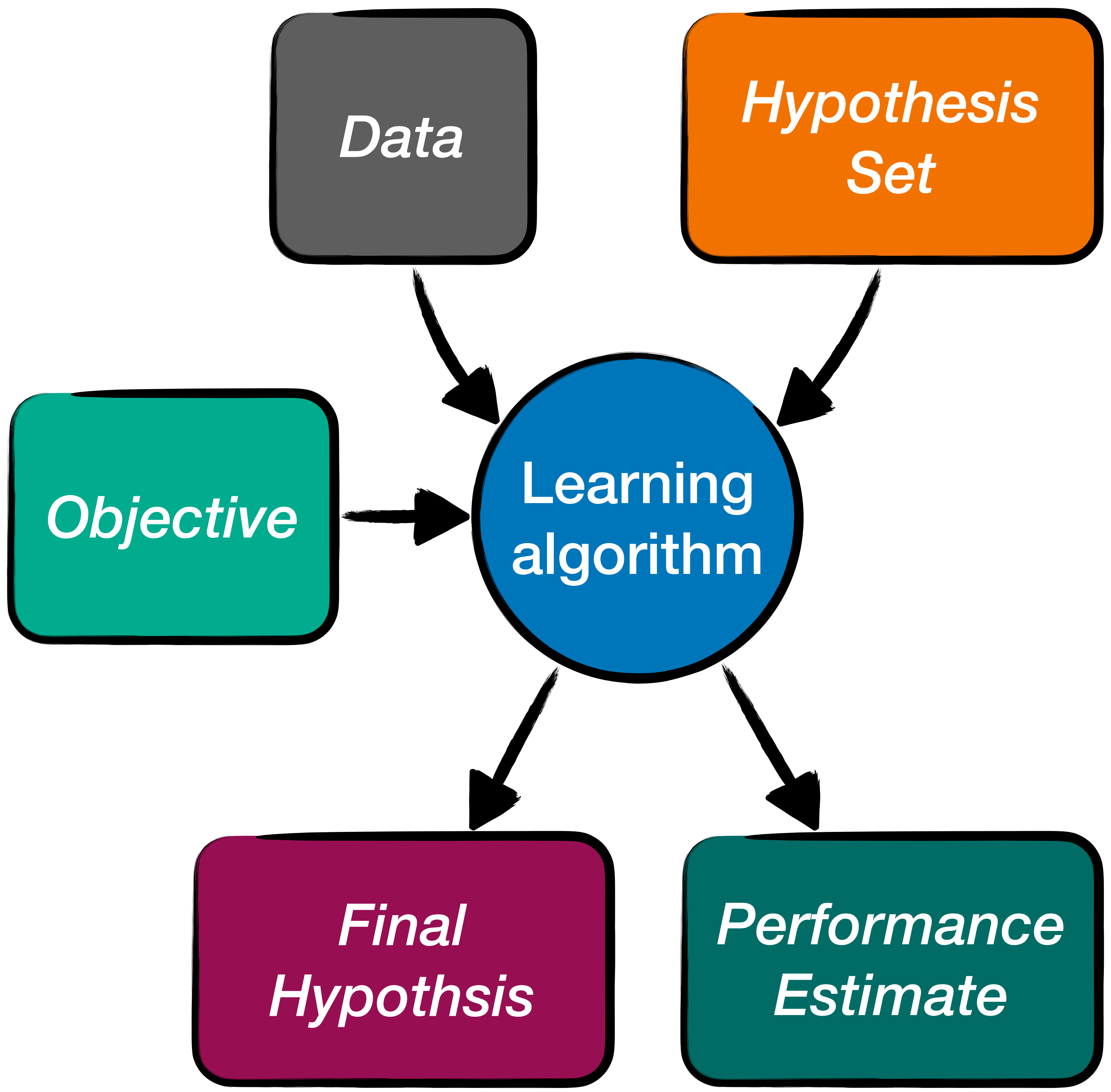
Once a metric becomes a target it ceases to be a good measure

Goodhart's Law

Charles Goodhart

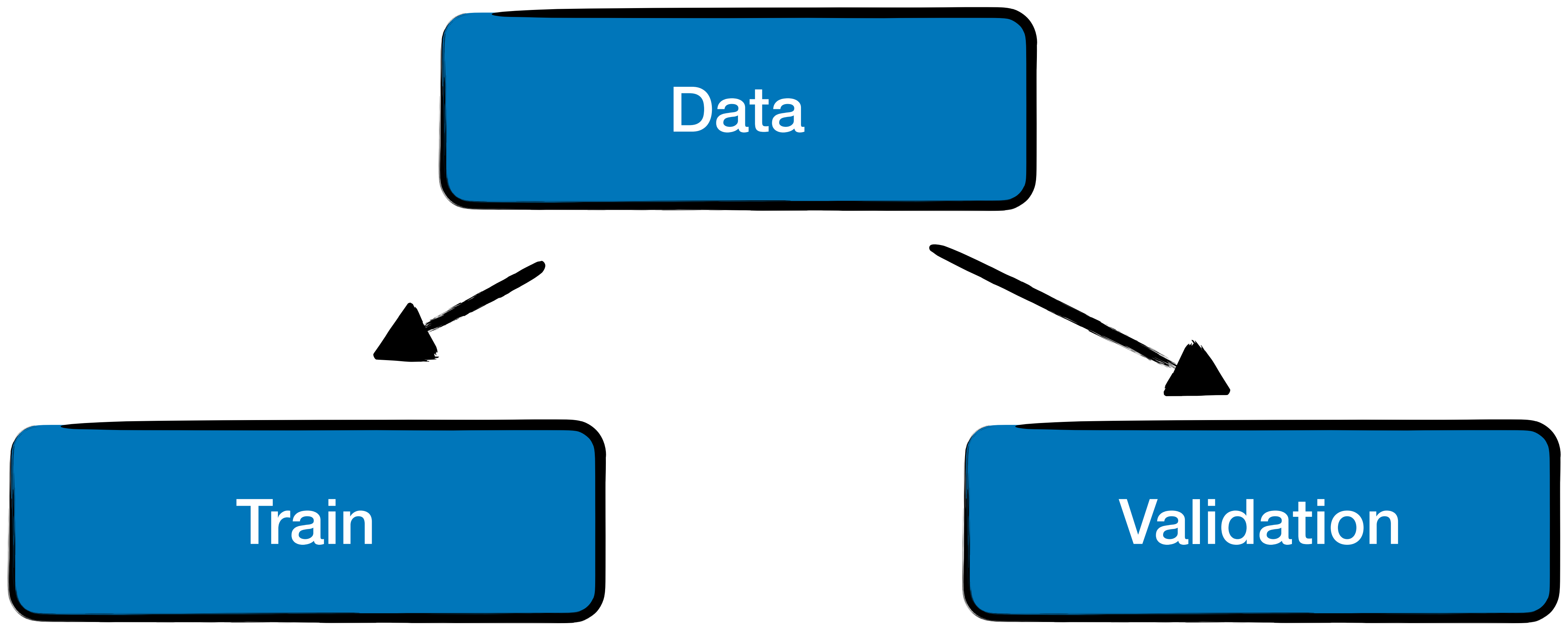
Upshot

We should thus adapt our learning framework to include a procedure to reliably estimate the generalization performance



Data Split

The data should we split into three categories
for a proper ML workflow



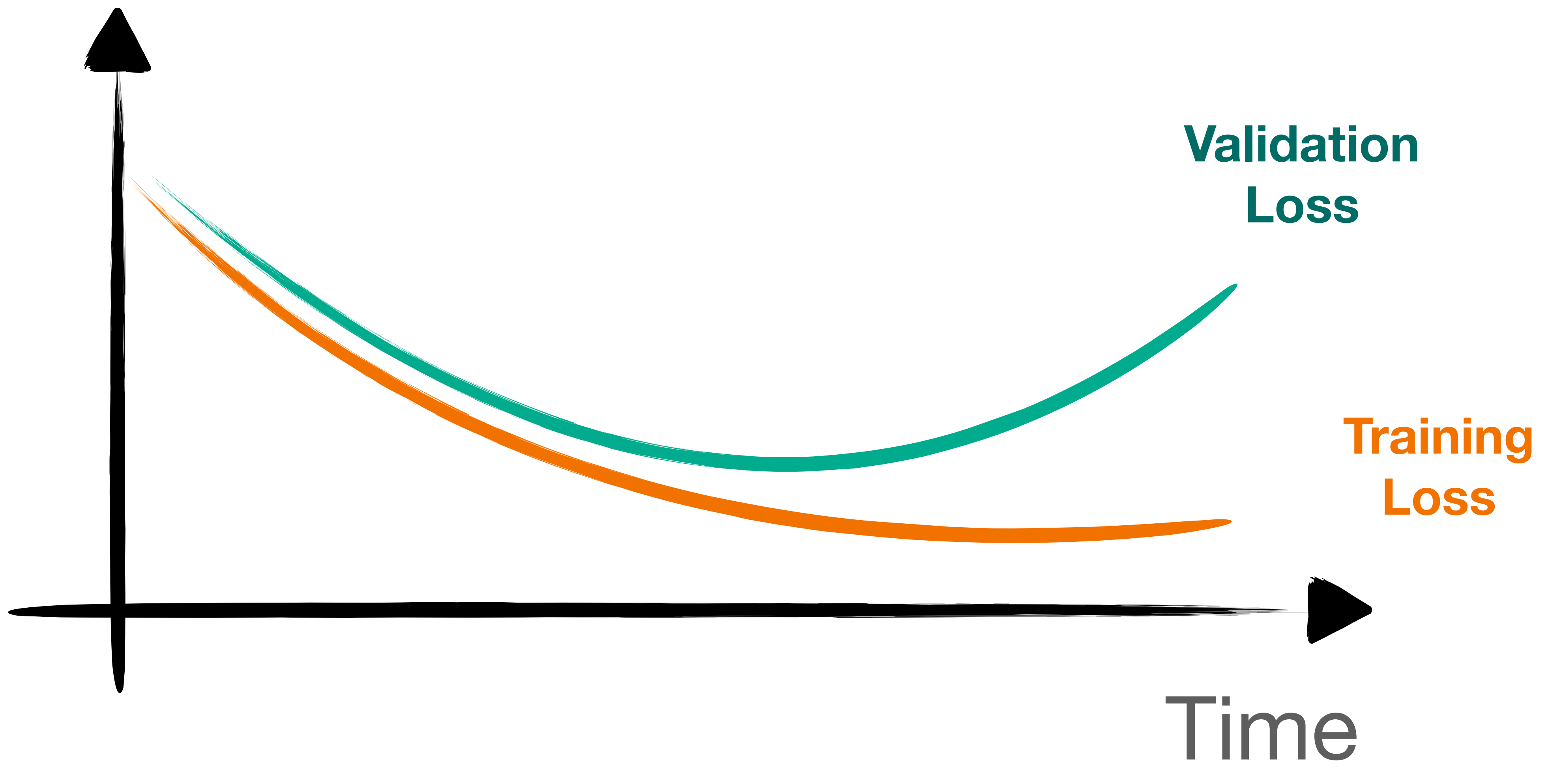
Use to pick a
$h^* \in \mathscr{H}$, i.e. ERM

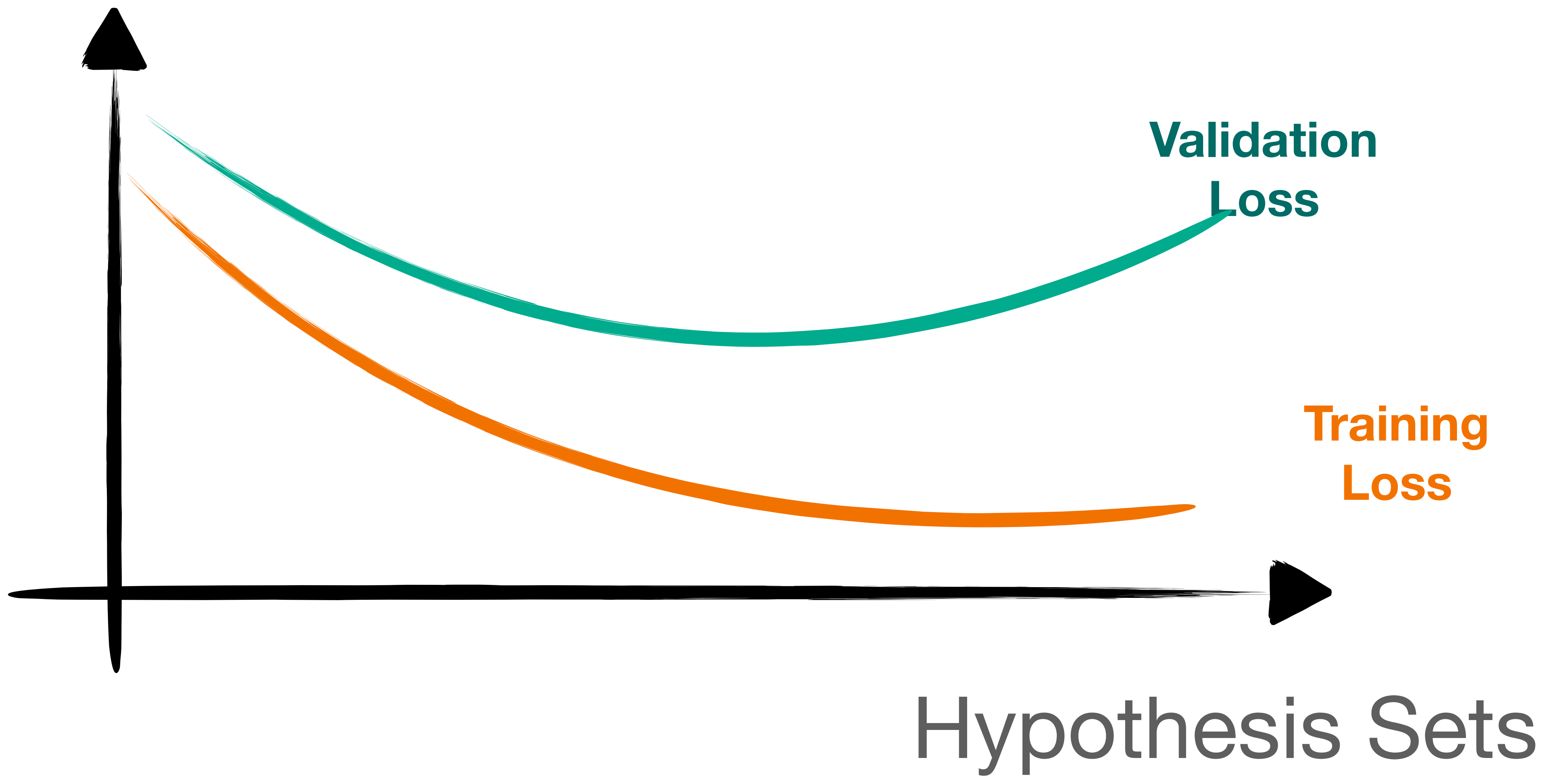
Use to produce a
performance estimate

Data Split

The validation data is (for now) independent of the selection procedure of $h*$, so it's a valid performance estimate again

We can monitor it **during training** and **across Hypothesis Sets**



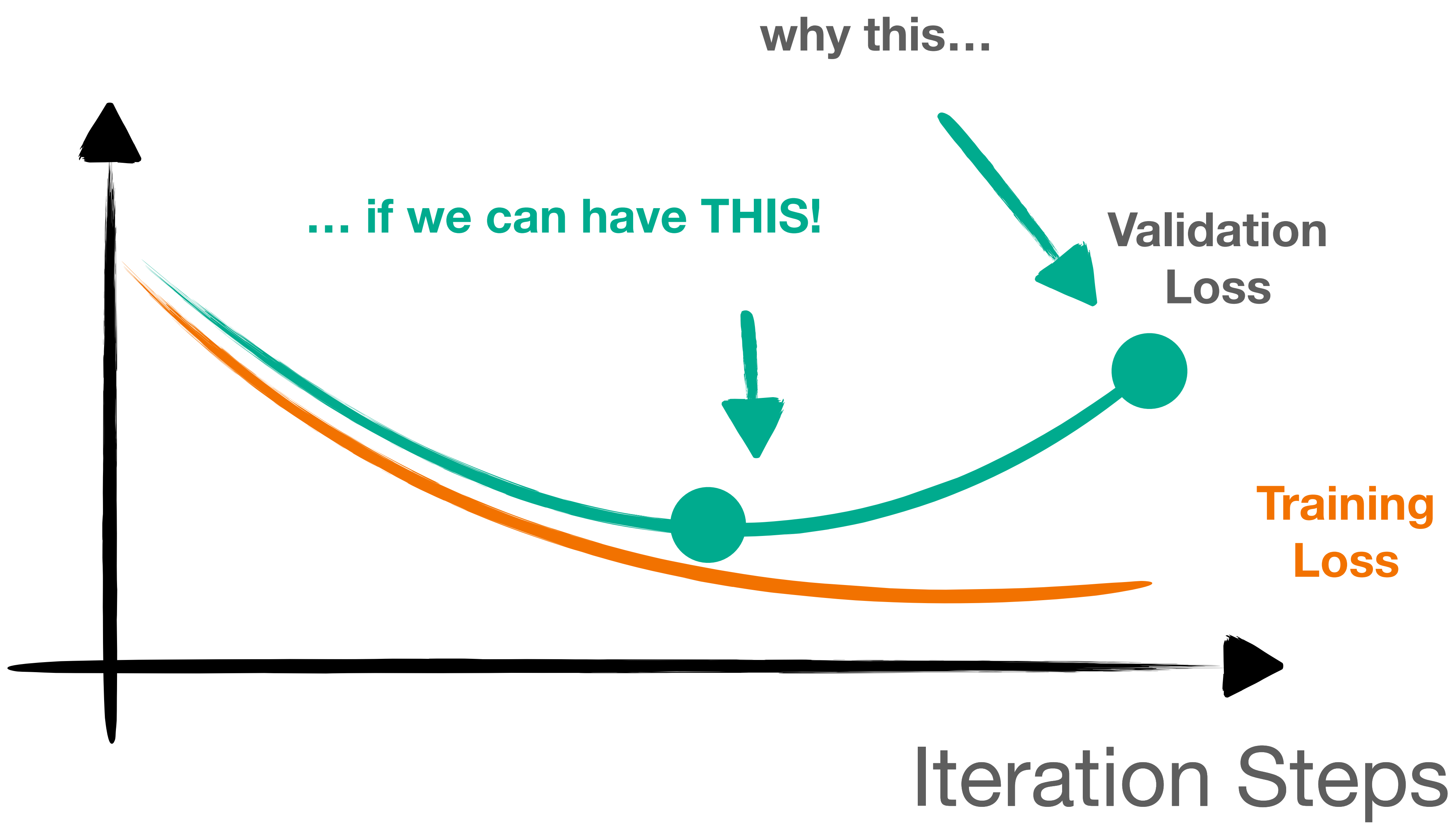


Note the x-axis, these are different (but related) plots

A Temptation

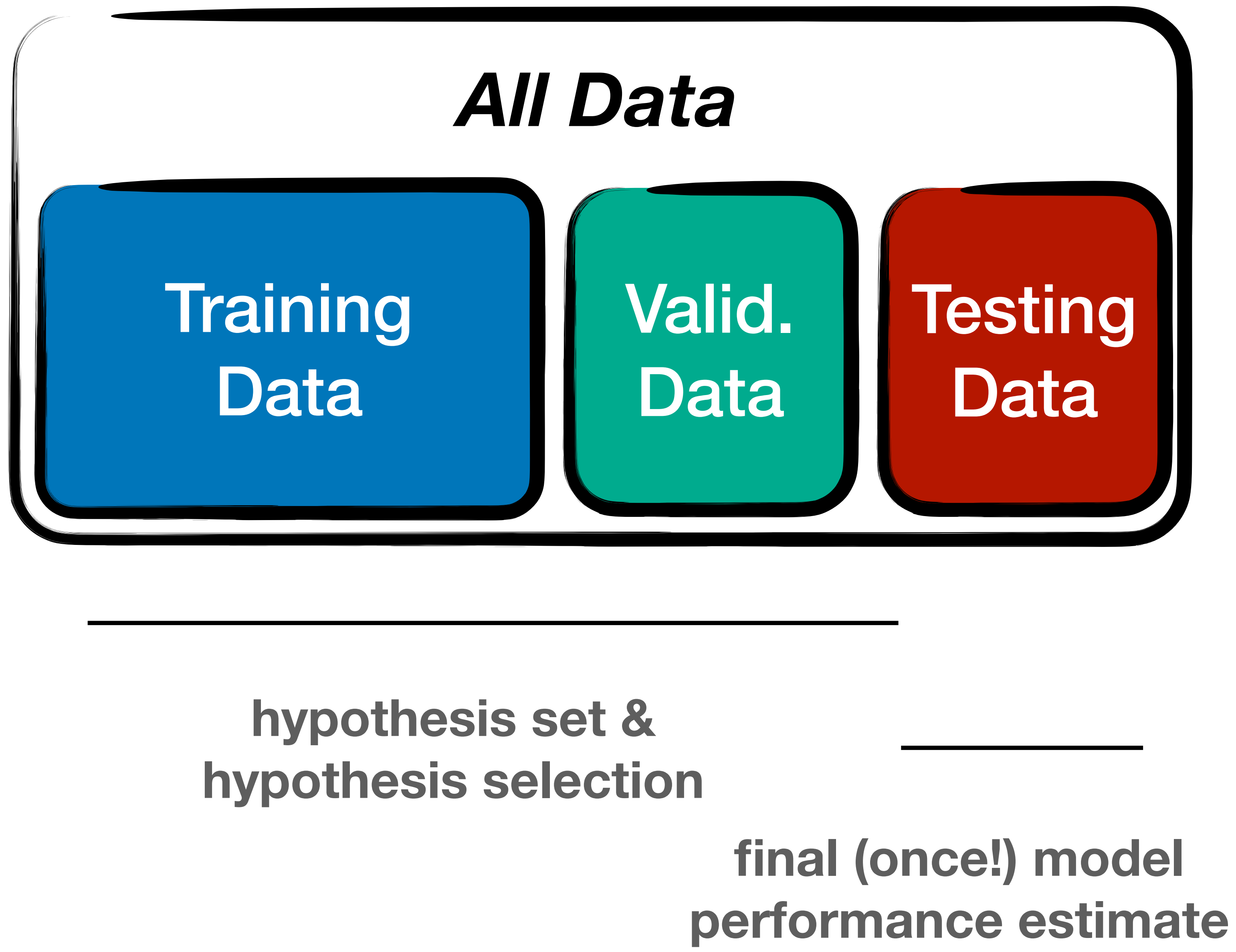
When monitoring the validation loss, **we're tempted to use it to flip-flop**. Instead of taking the final model from ERM, we could:

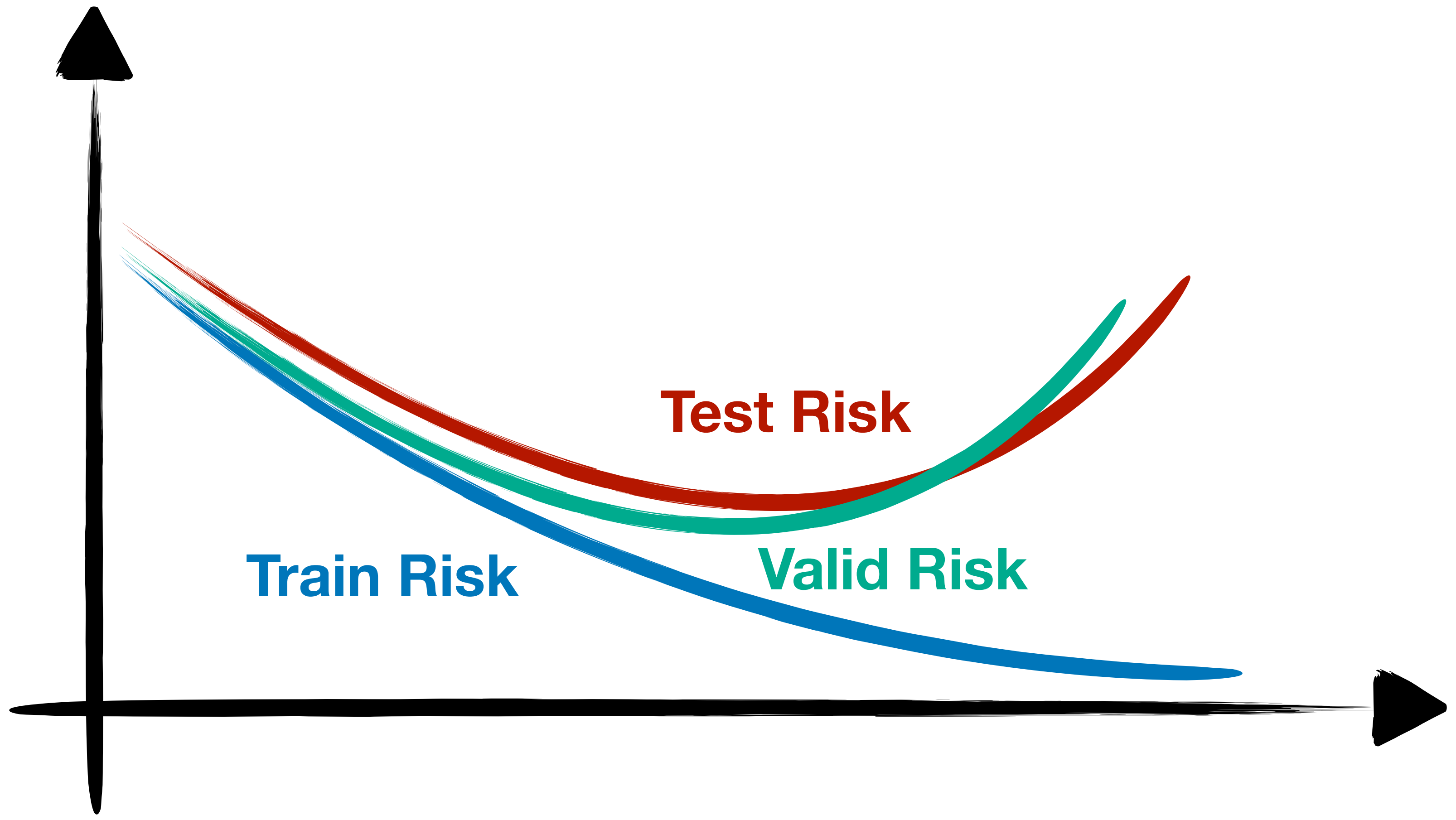
- take any other model from this run ("early stopping")
- switch to a better hypothesis set ("hyperparameter tuning")



Choosing the right Hypothesis Set

If we want to use the validation risk to select the model, we need to split the data **in three ways** to avoid double dipping

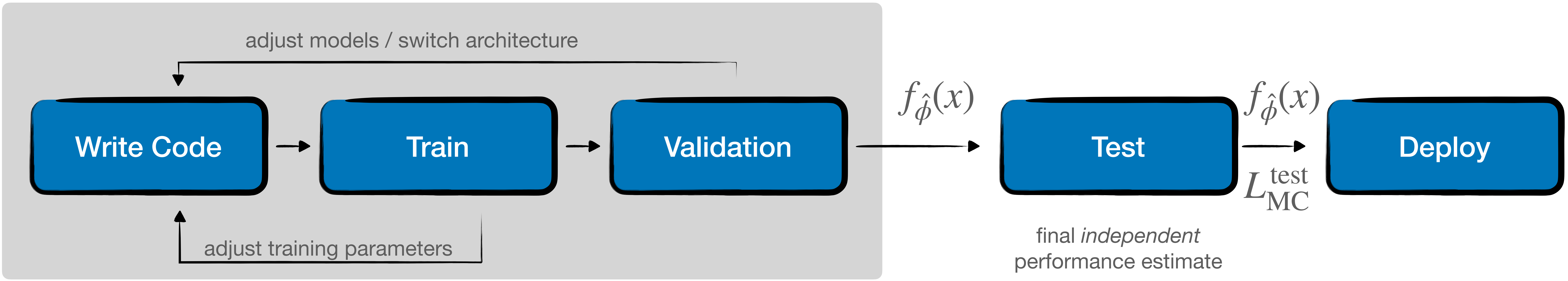




The ML Workflow

Training ML systems is a highly iterative process. Many small adjustments in e.g. training parameters, experiments with different models, ..

Overall it looks like this:



Optimization

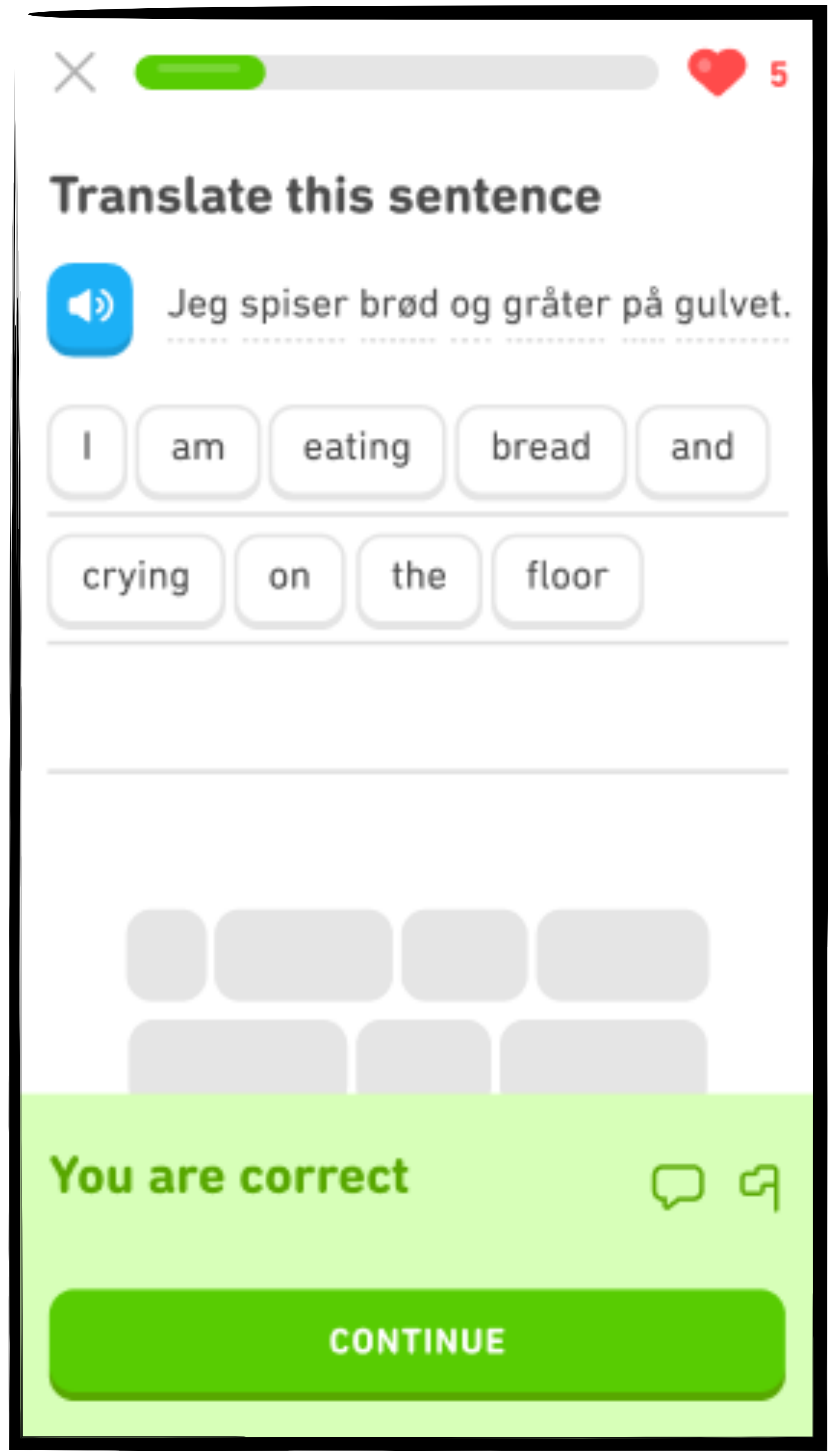
Iterative Optimization

Closed form solutions are rare, most often we use iterative optimization: improve

learn by revisiting the data often & adjusting

An Iterative Training Loop

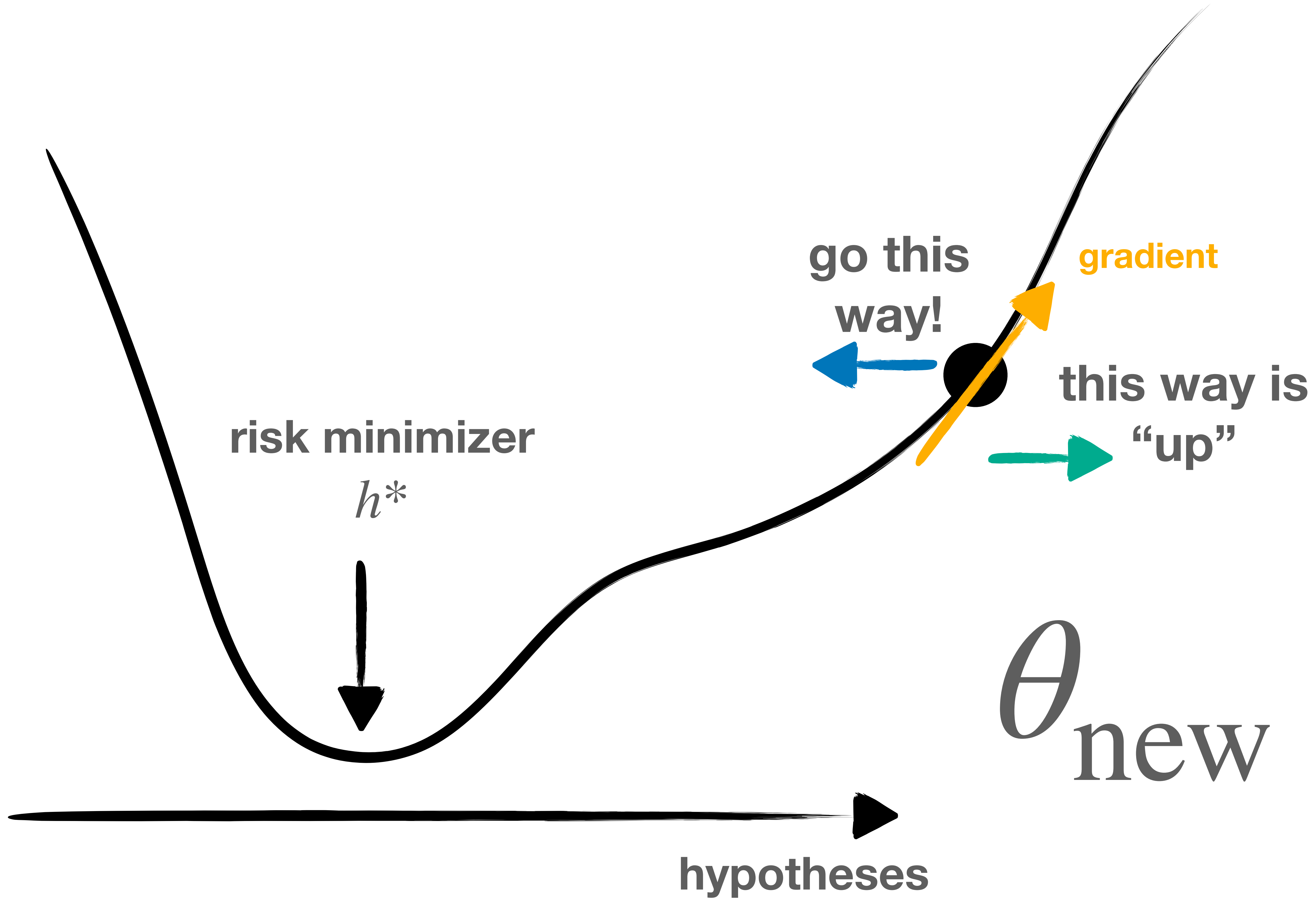
```
h = initial_guess()
for n in range(steps):
    examples ~ p(data)
    risk = evaluate(h, examples)
    adjustment = react(risk, h)
    h = new_hypo(h, adjustment)
```



If we can improve a little bit each time eventually we find a good solution

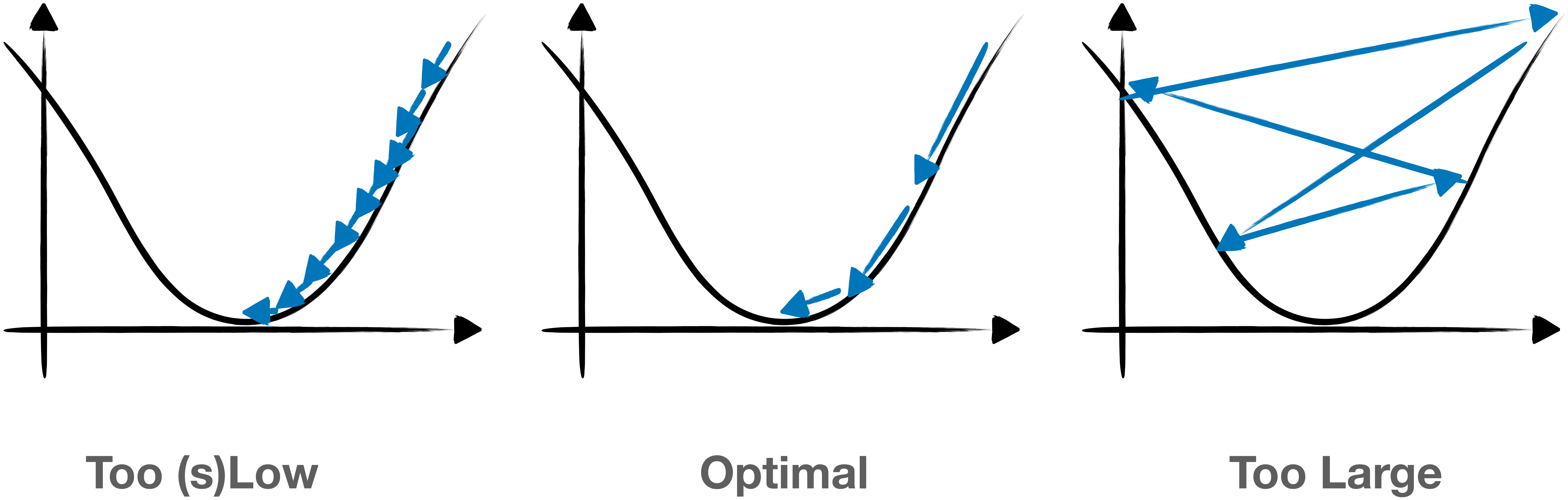
Gradient Descent

A natural idea is to minimizing the loss by walking downhill



$$\theta_{\text{new}} \leftarrow \theta - \lambda \boxed{\nabla_\theta L}$$

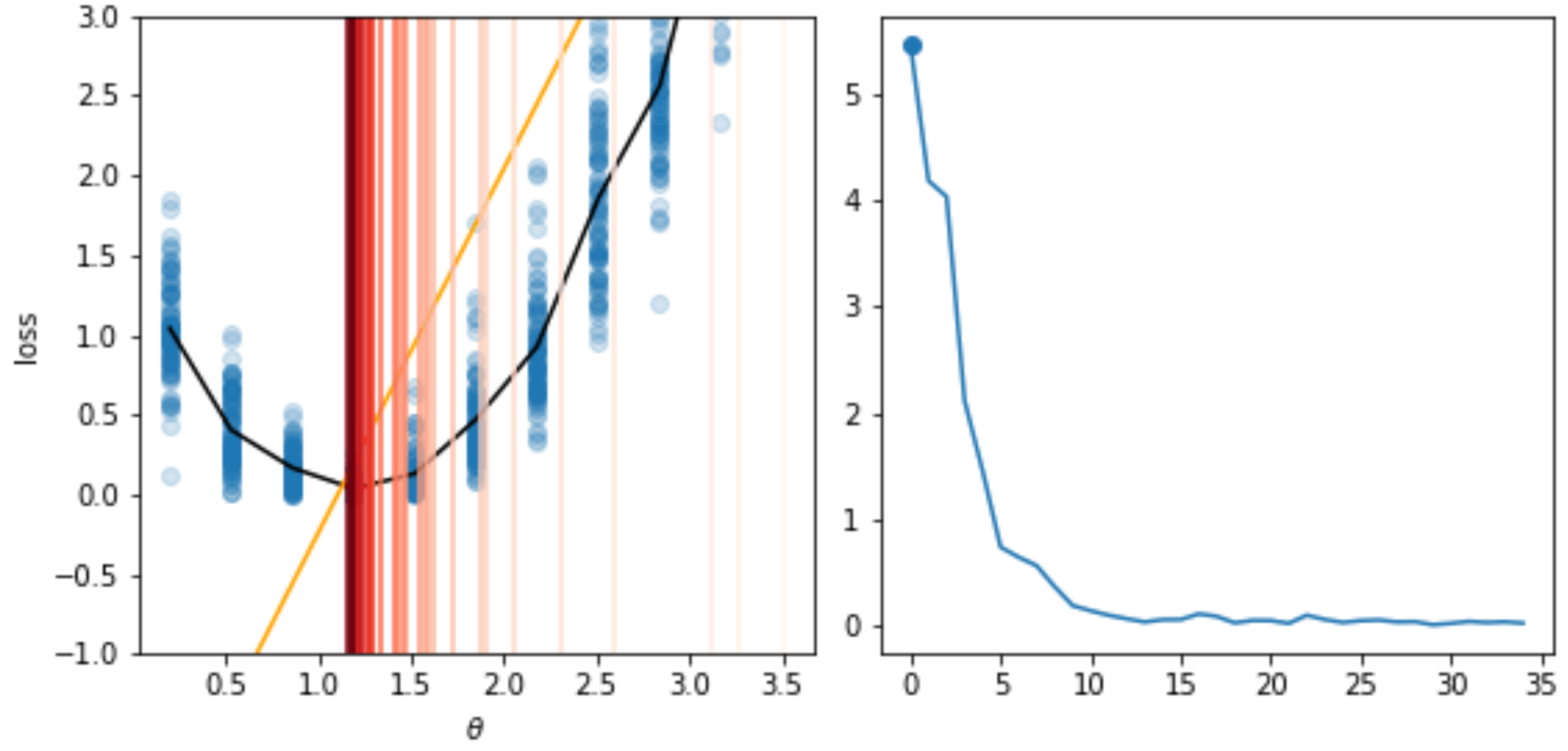
Tuning the Learning Rate

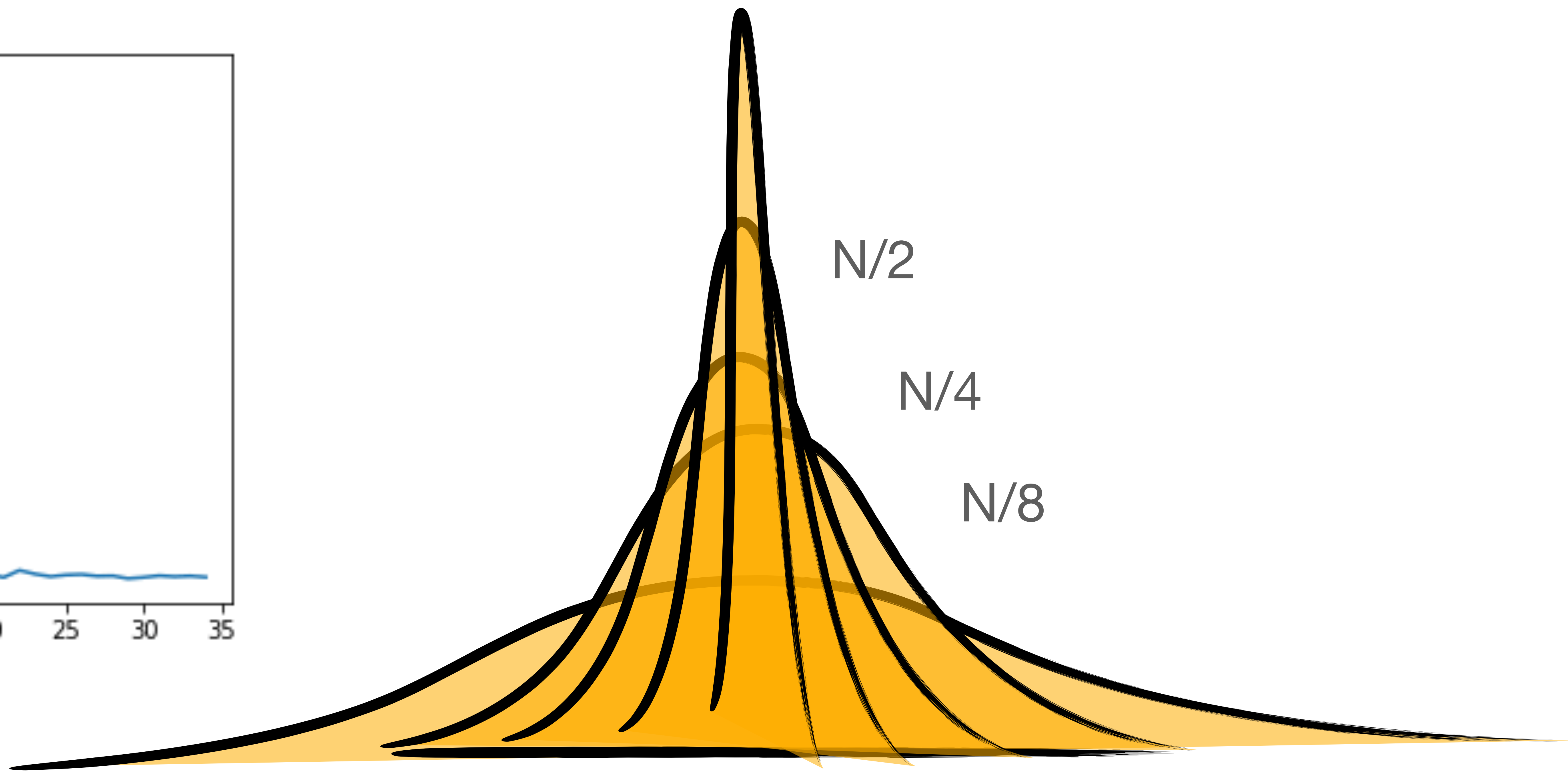


Stochastic Gradient Descent

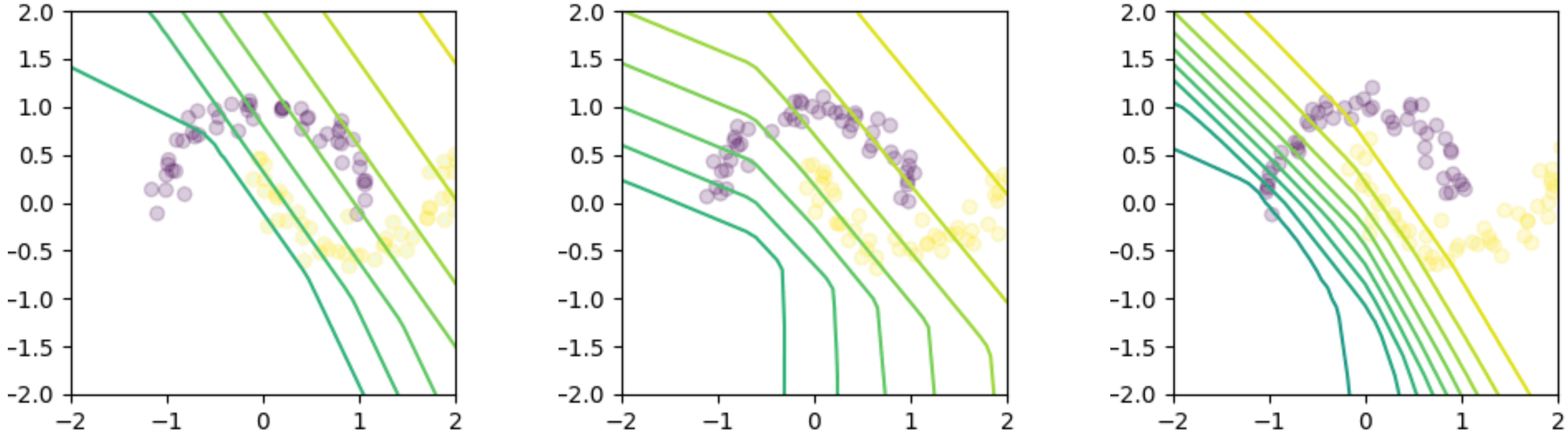
Evaluating the loss on a small "mini-batch" instead of the full data: useful noise to jump over e.g. local minima.

Remember: actual goal is generalization not training loss





Optimizing a simple Neural Net

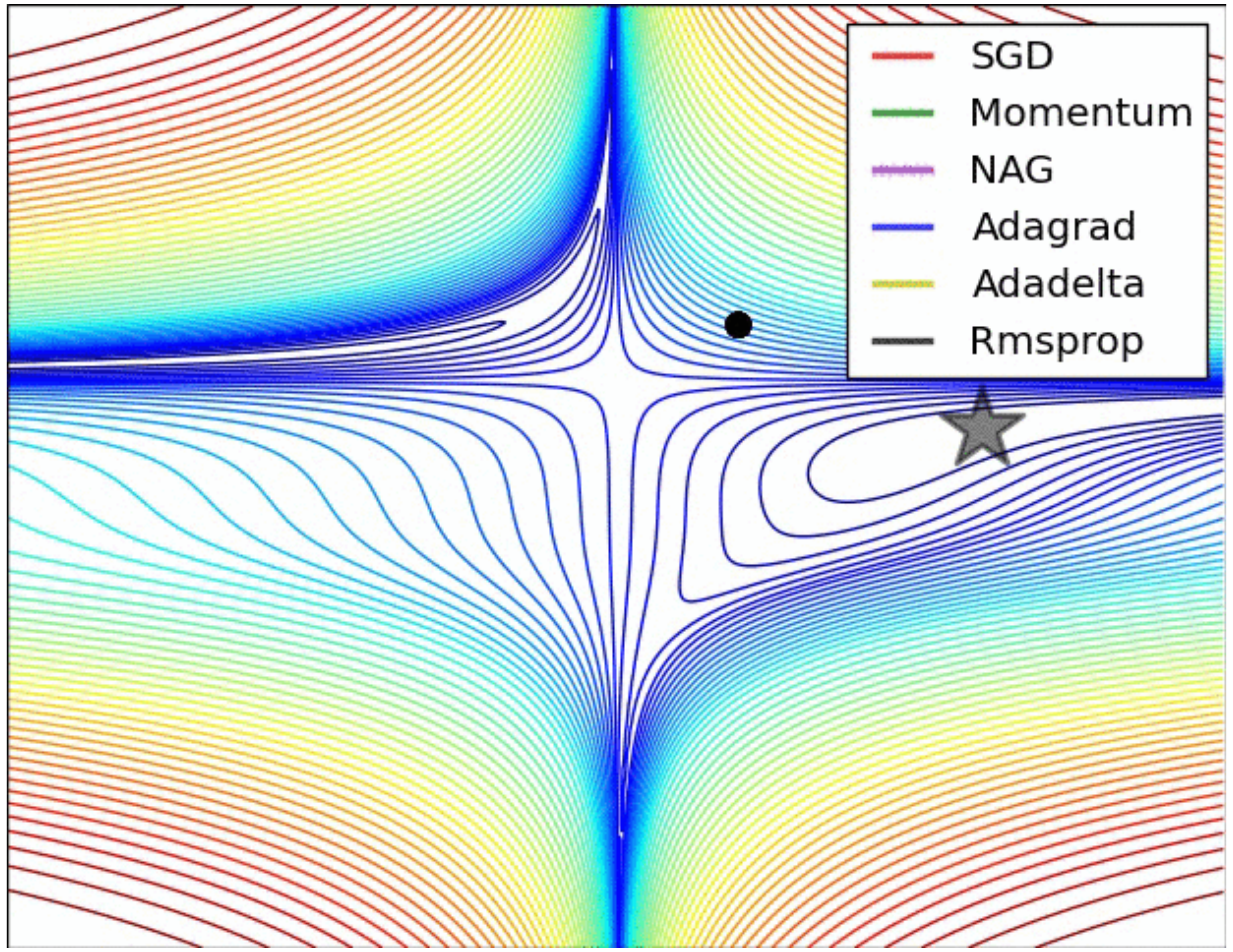


Optimizers

Many additional tricks & nuances in practical optimization algorithms to improve convergence for non-convex problems

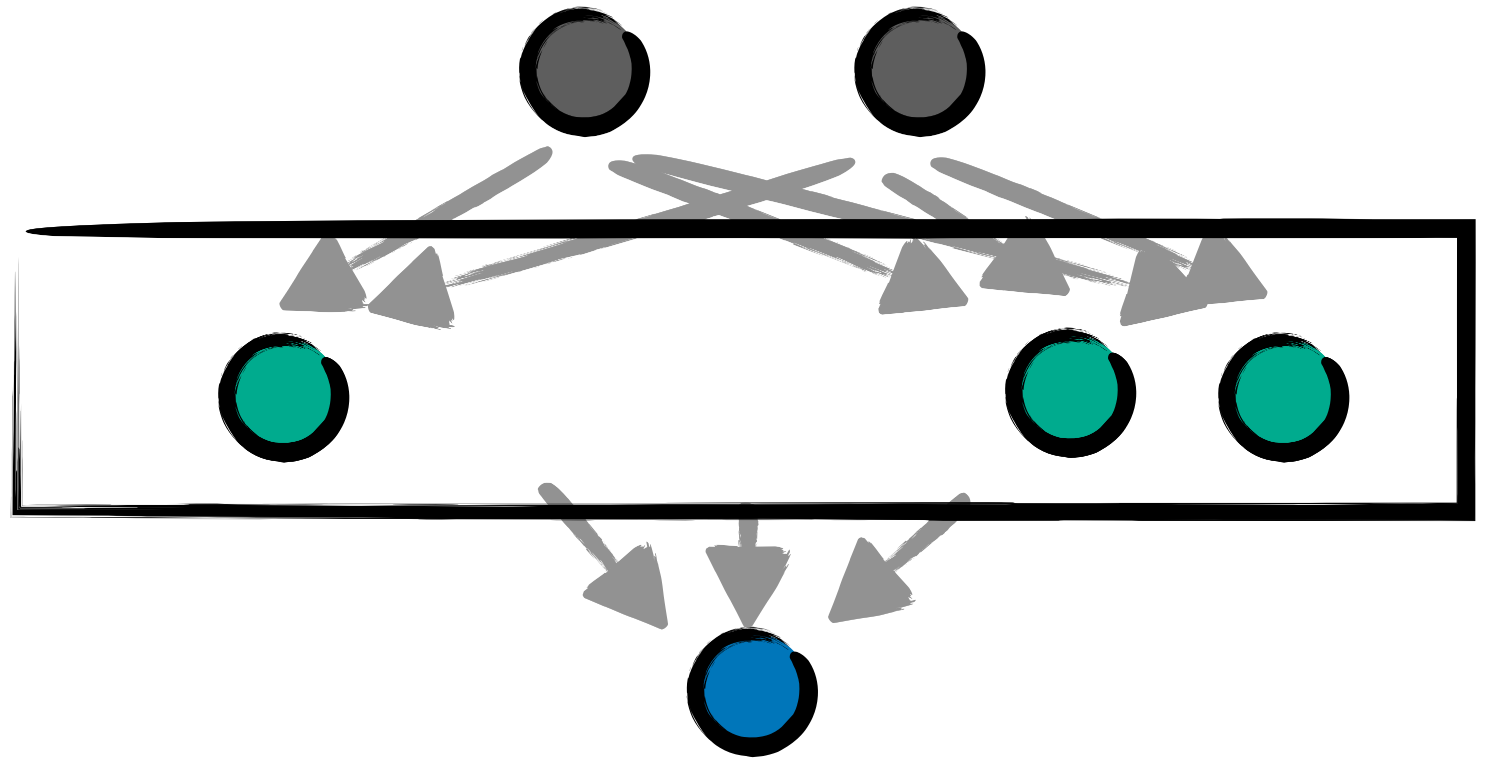
- **Momentum:**
  keep historical
- **Adaptive Learning Rate:**
  accelerate in flat areas

Adam Optimizer is a good default

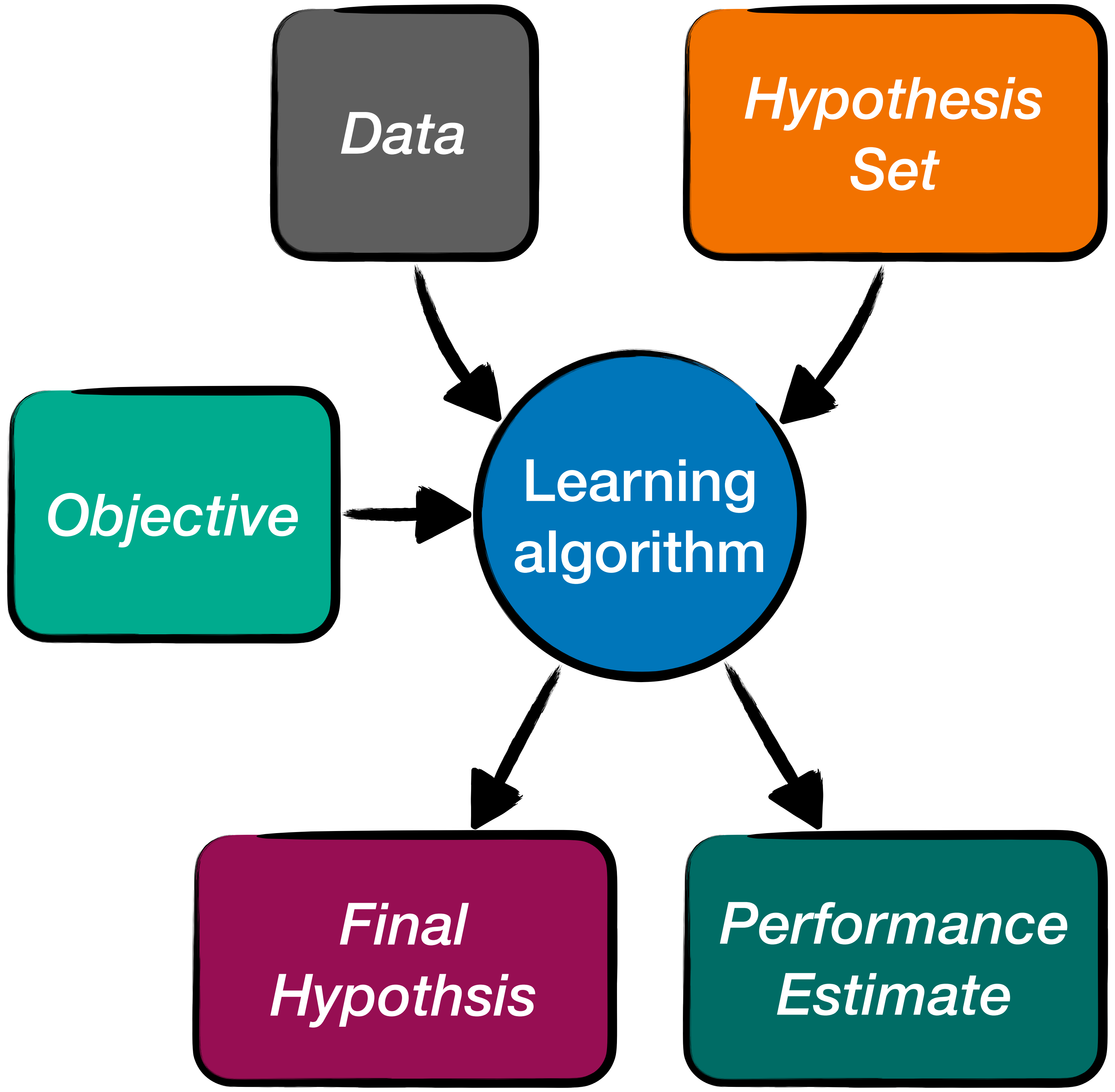


Summary

Neural Networks



Empirical Risk Minimization



Bias-Variance & Generalization