UNIVERSITÄT
HEIDELBERG

Machine Learning for Biochemistry

L2, Structural Bioinformatics

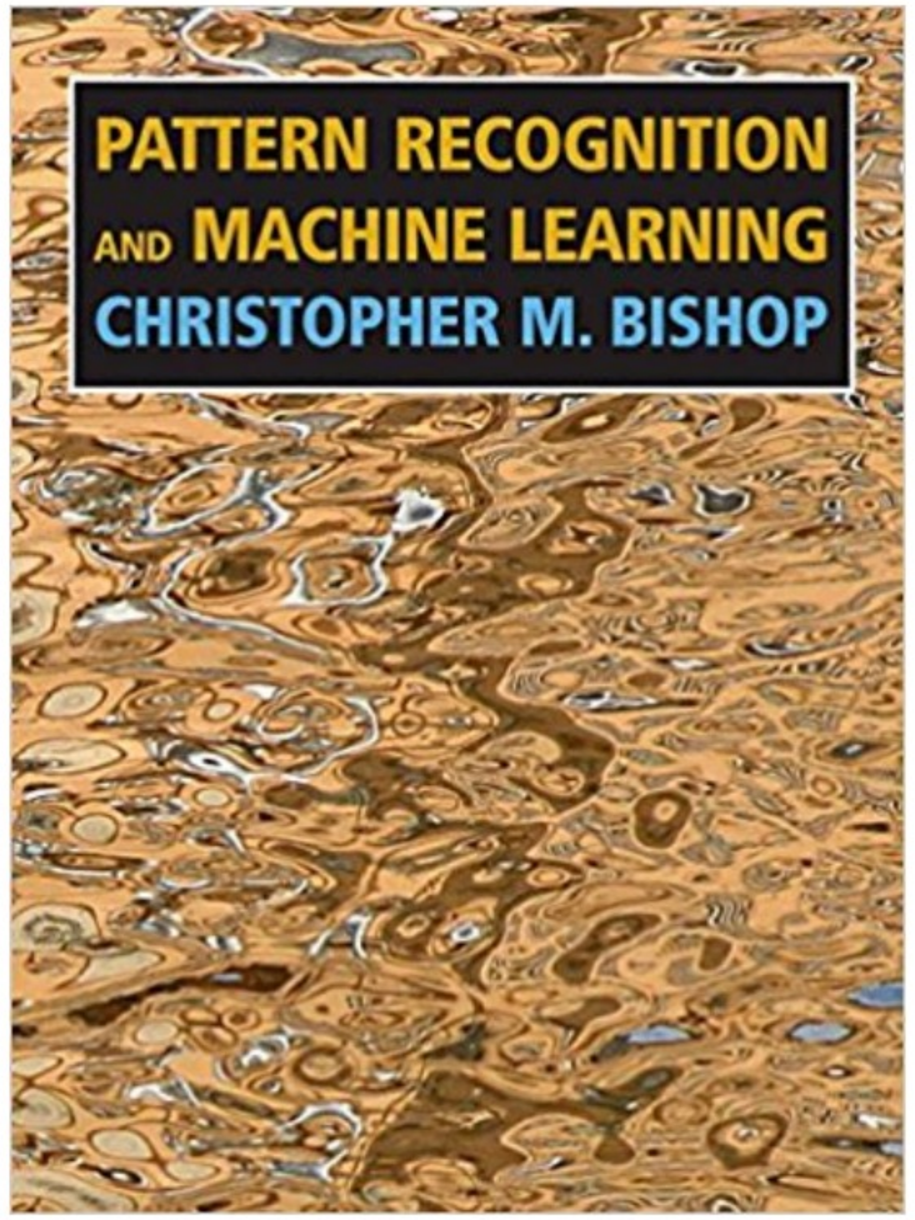
WiSe 2023/24, Heidelberg University

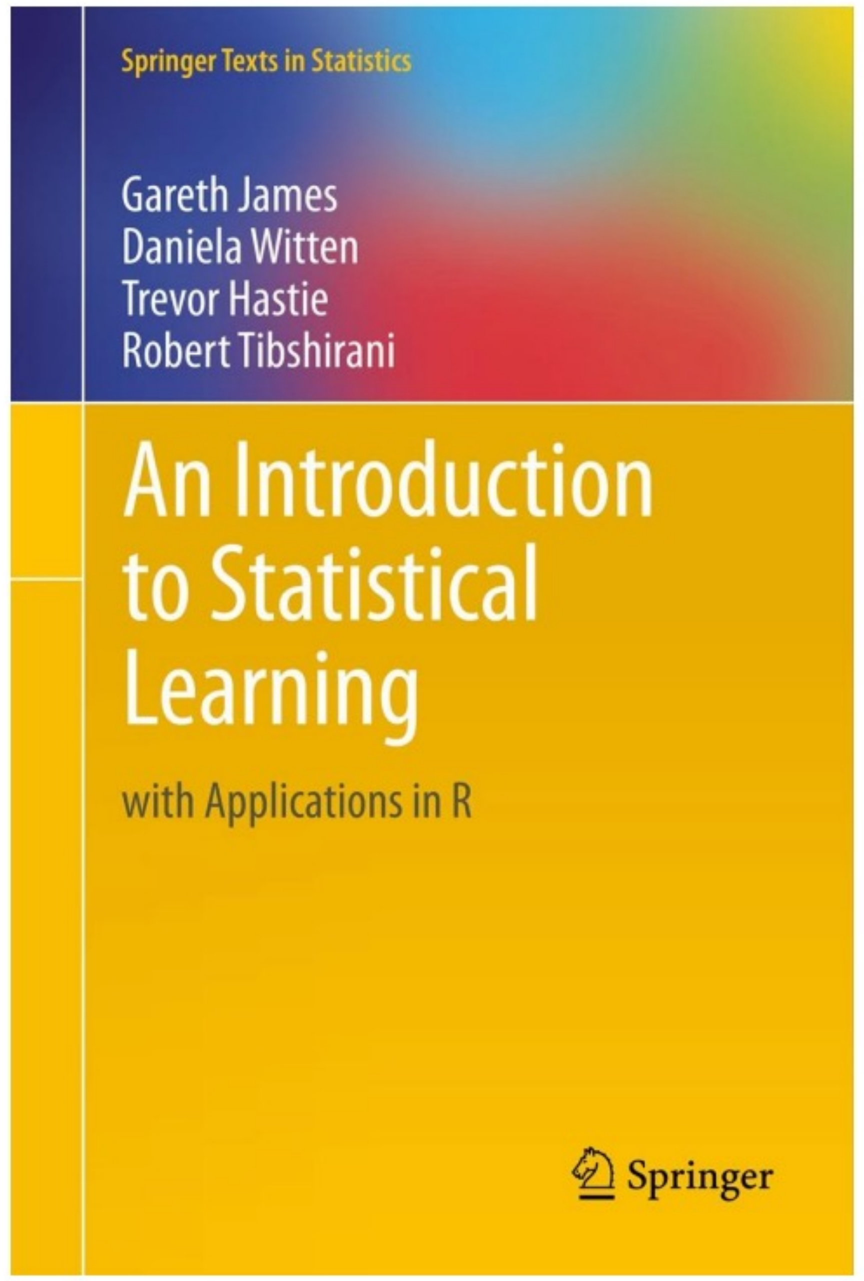
Overview

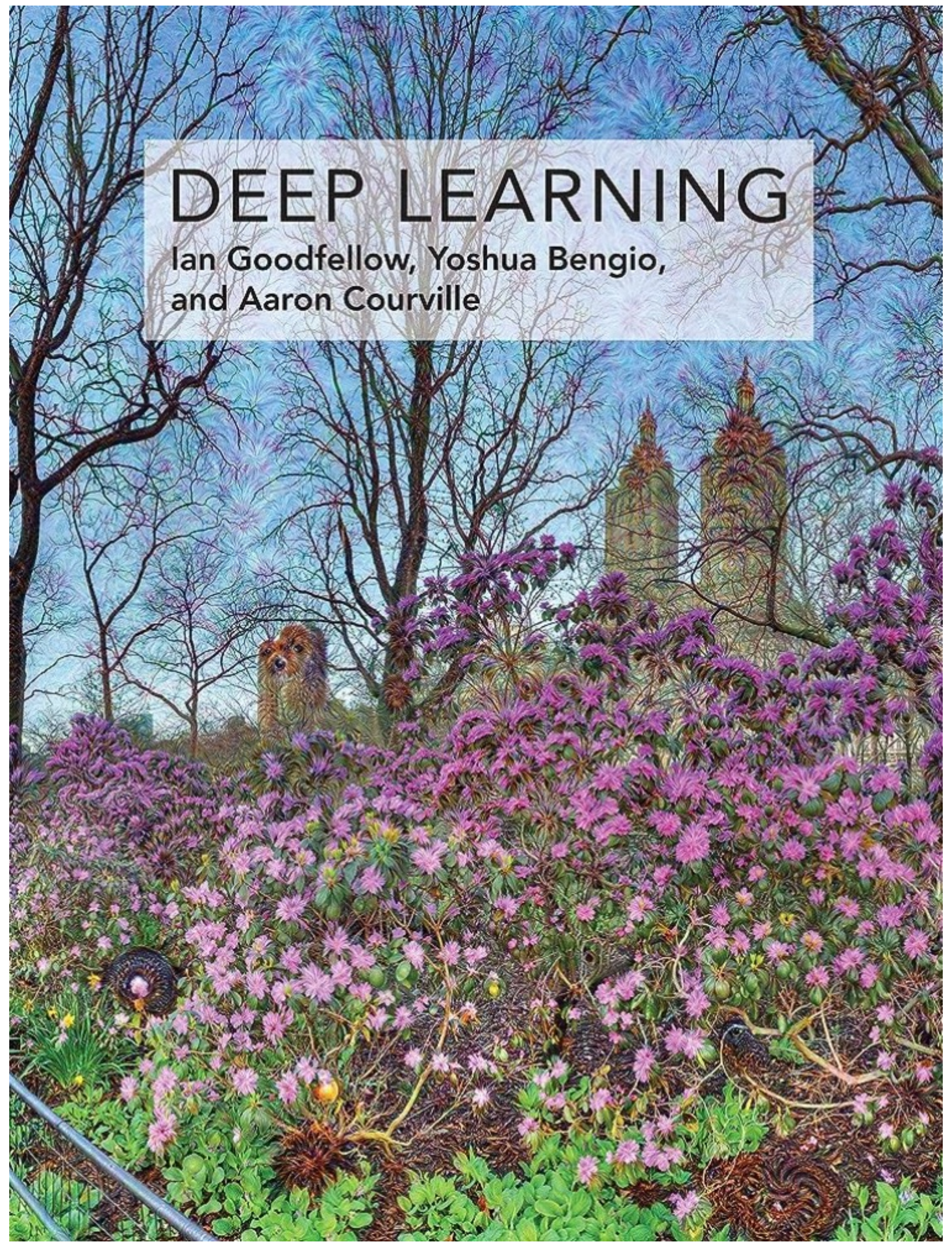
1. Types of Machine Learning
2. Linear Models
3. Gradient Descent
4. Deep Learning
5. Outlook for what’s to come

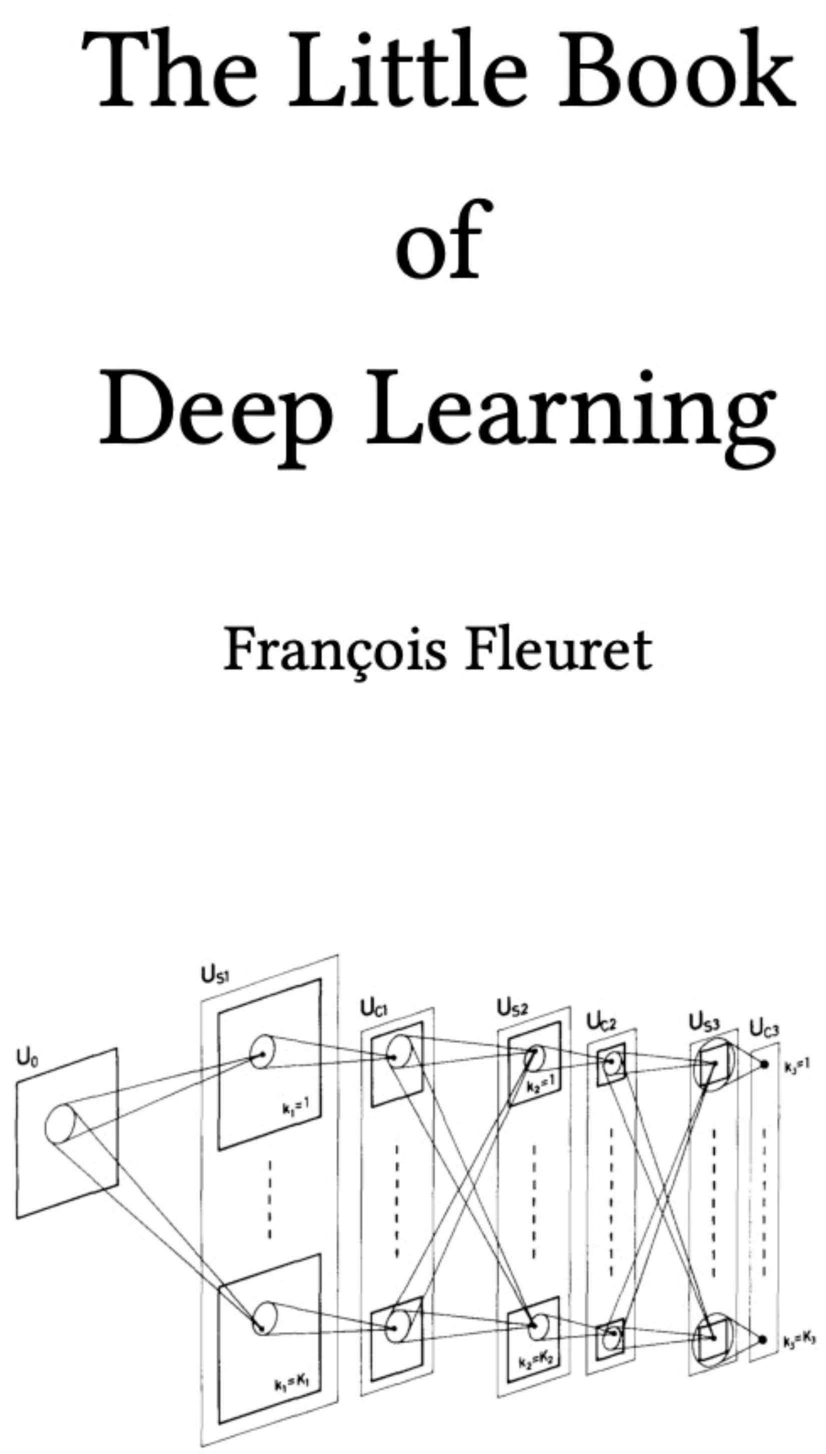
Book Recommendations

The Classics



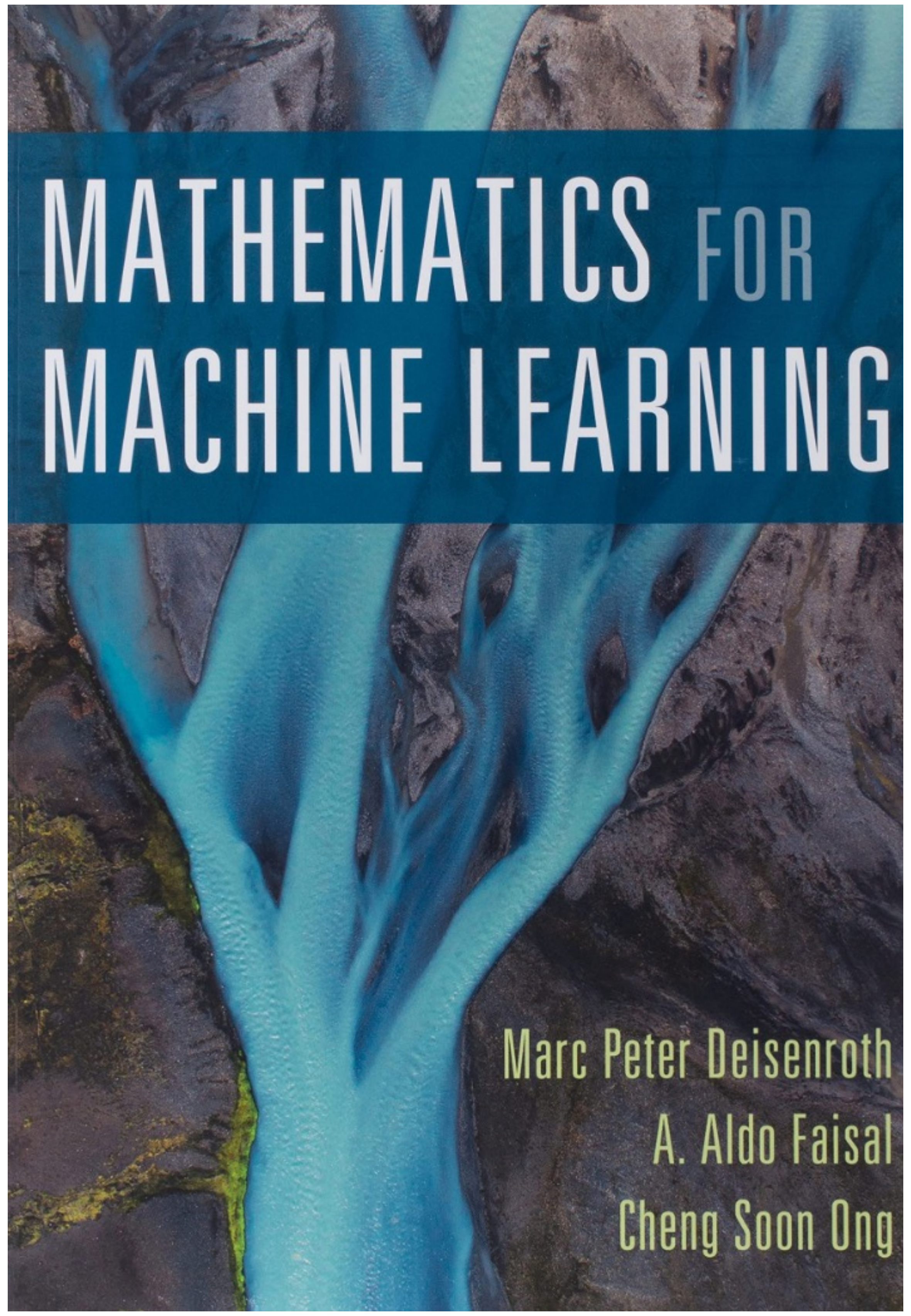


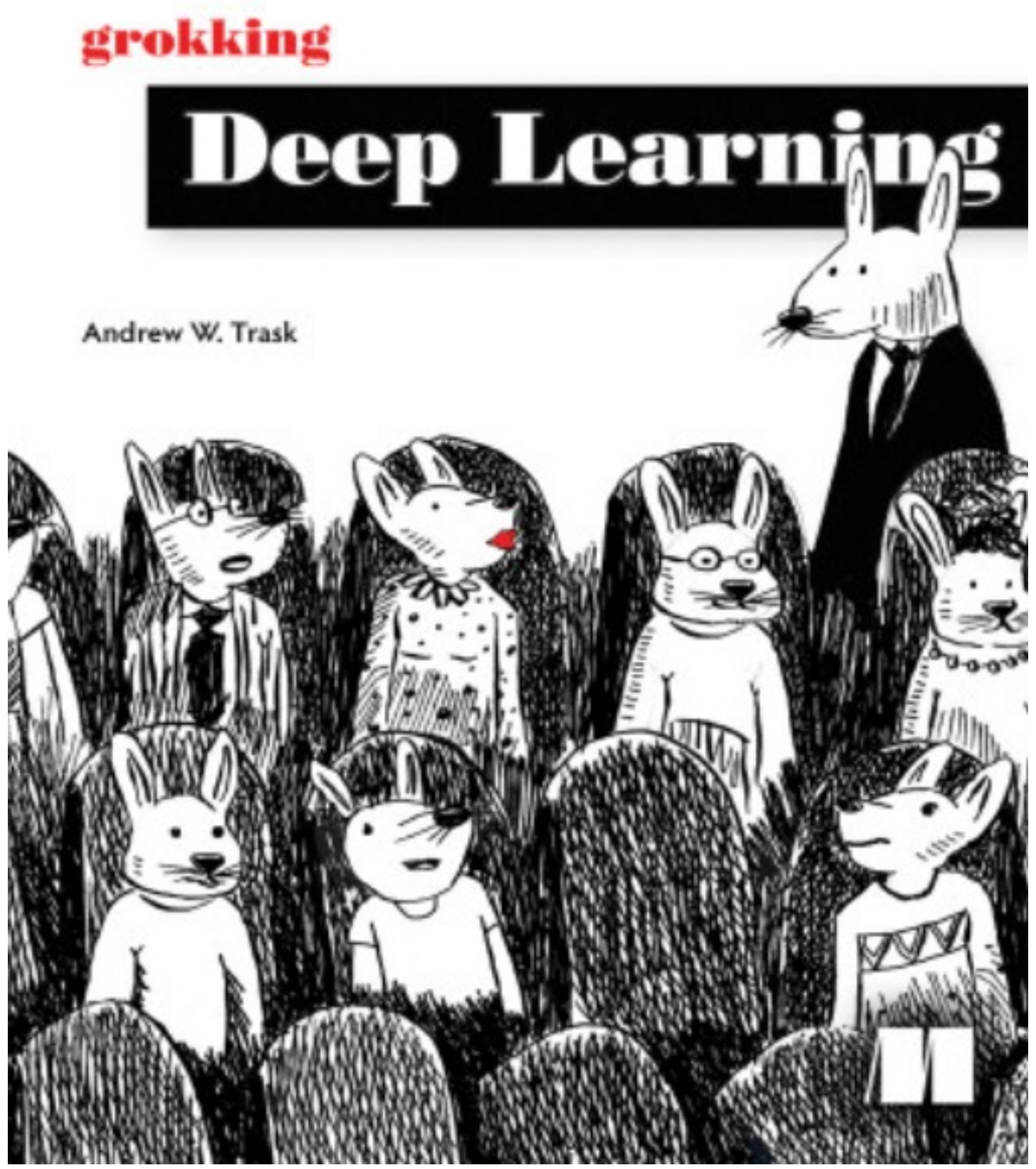


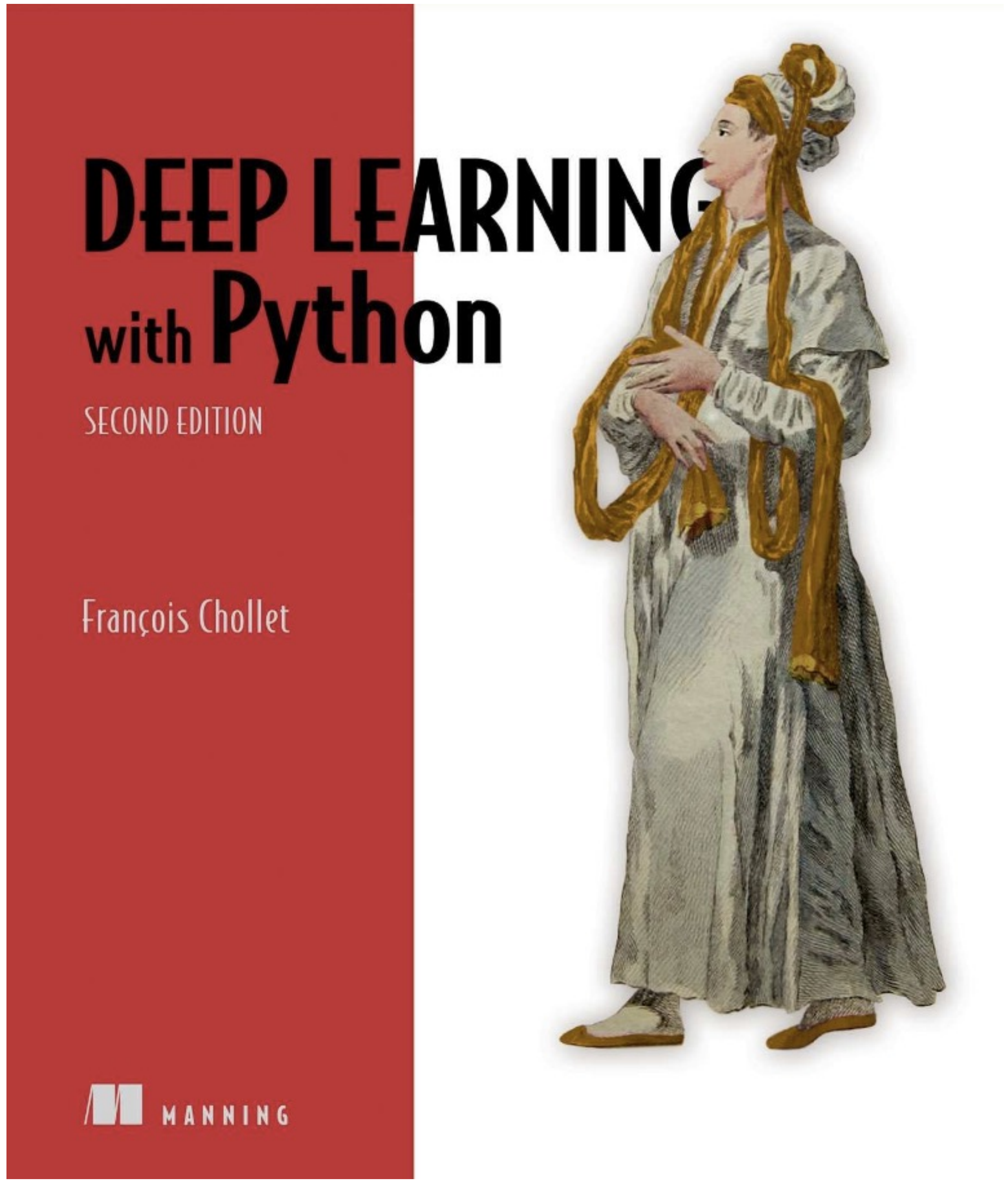


Book Recommendations

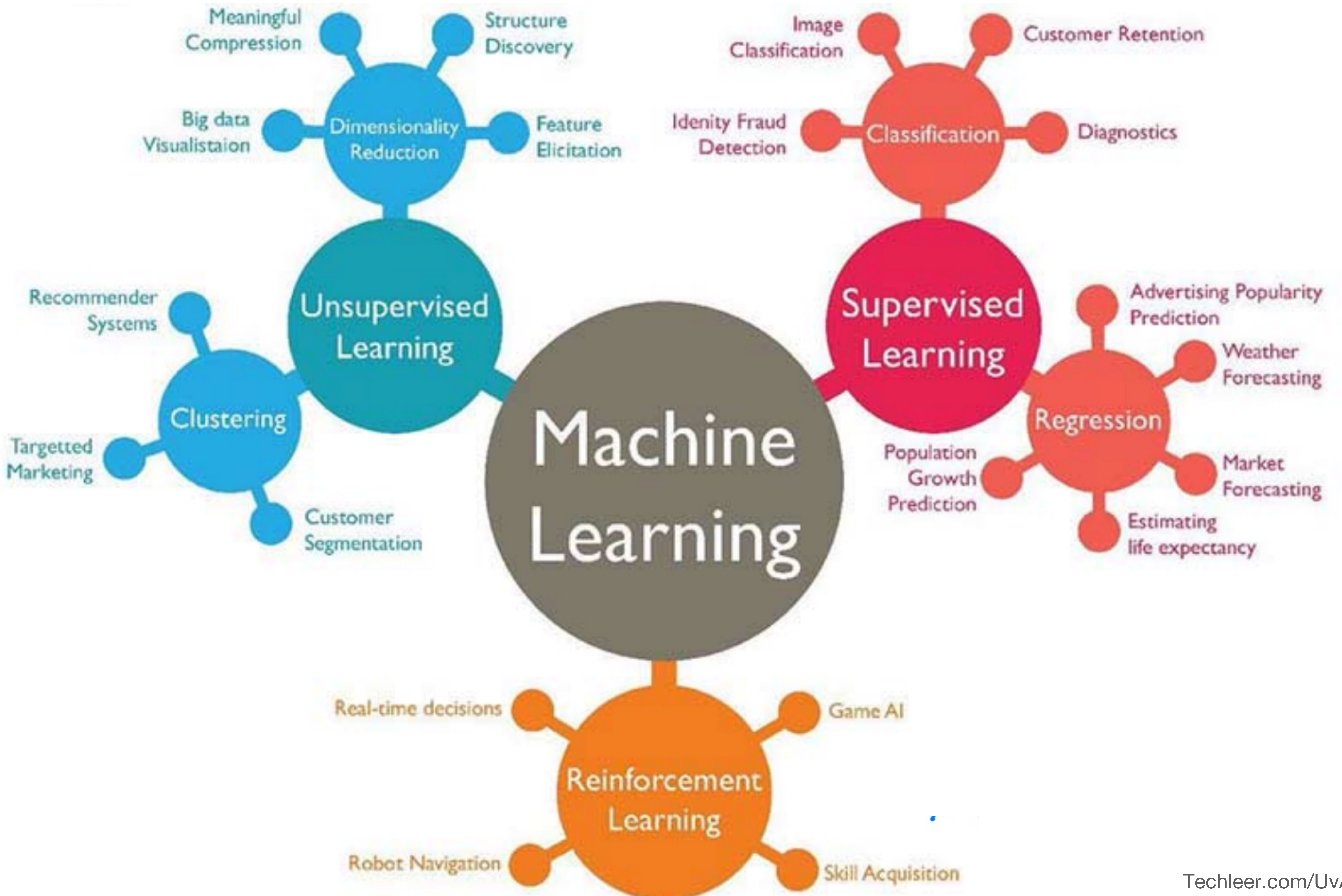
Background and more introductory books





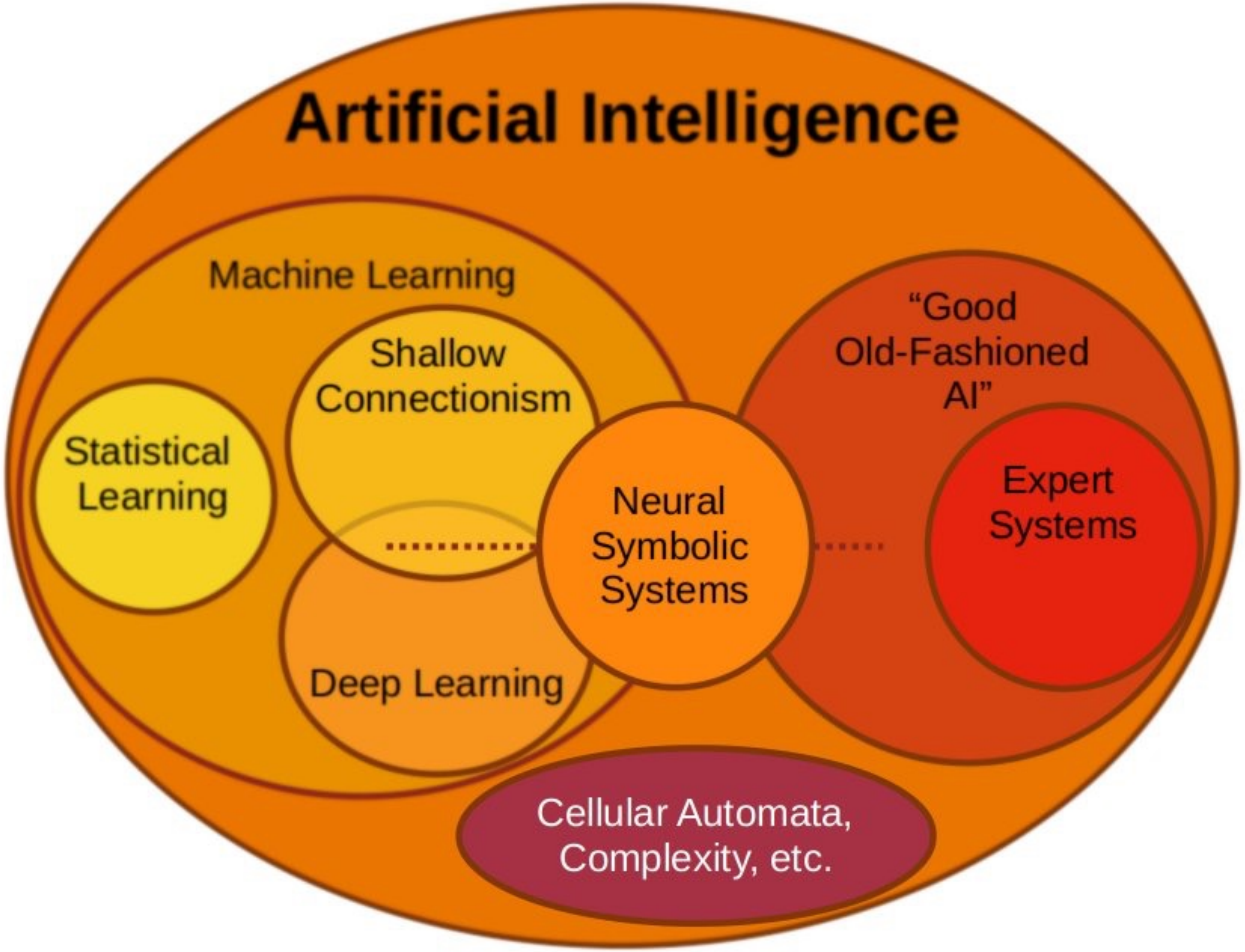


1 Types of Machine Learning



Techleer.com/UvA ML1 Course

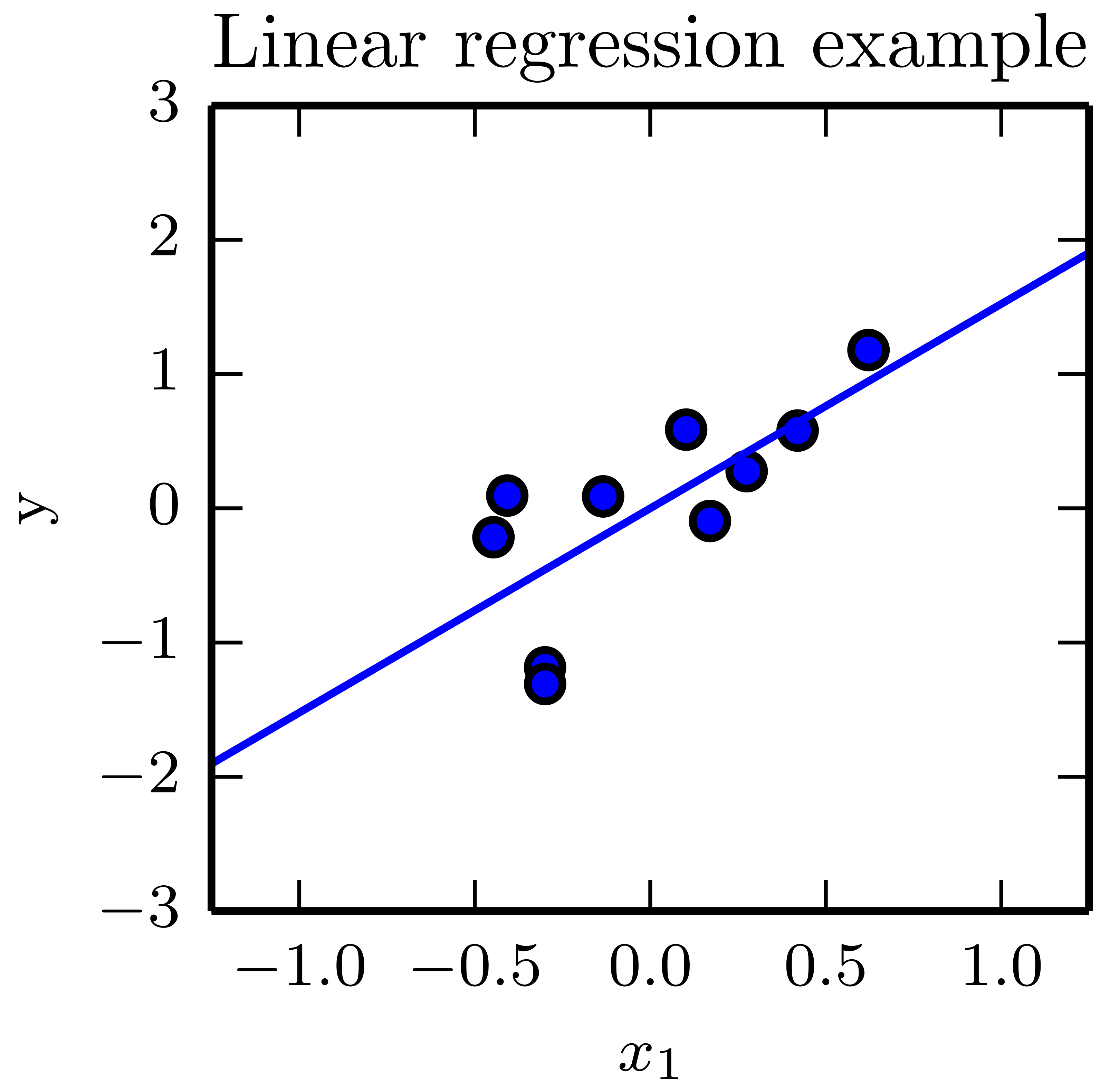
AI is more than just Deep Learning



2 Linear Models

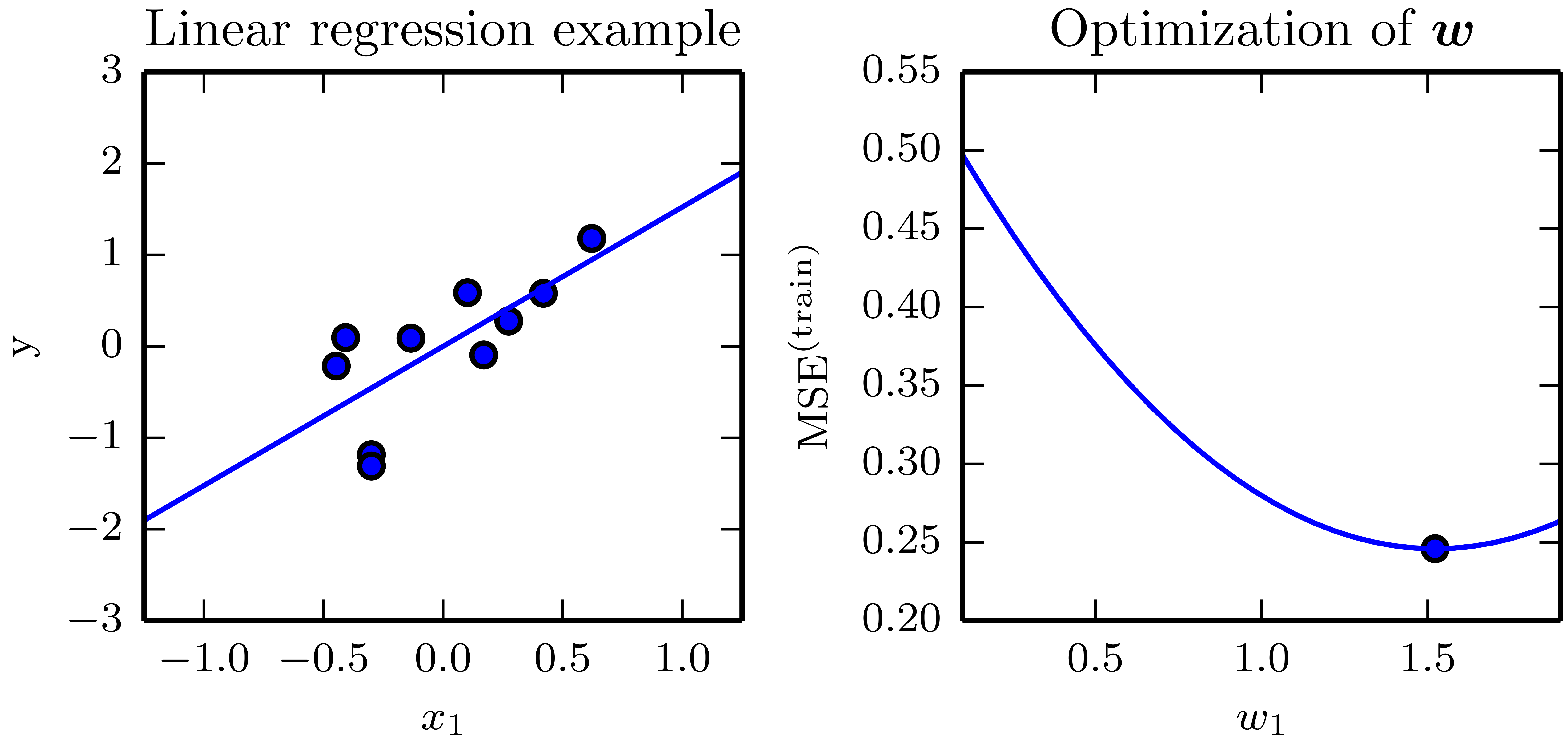
What is a linear model?

Adjust your free parameter based on some loss



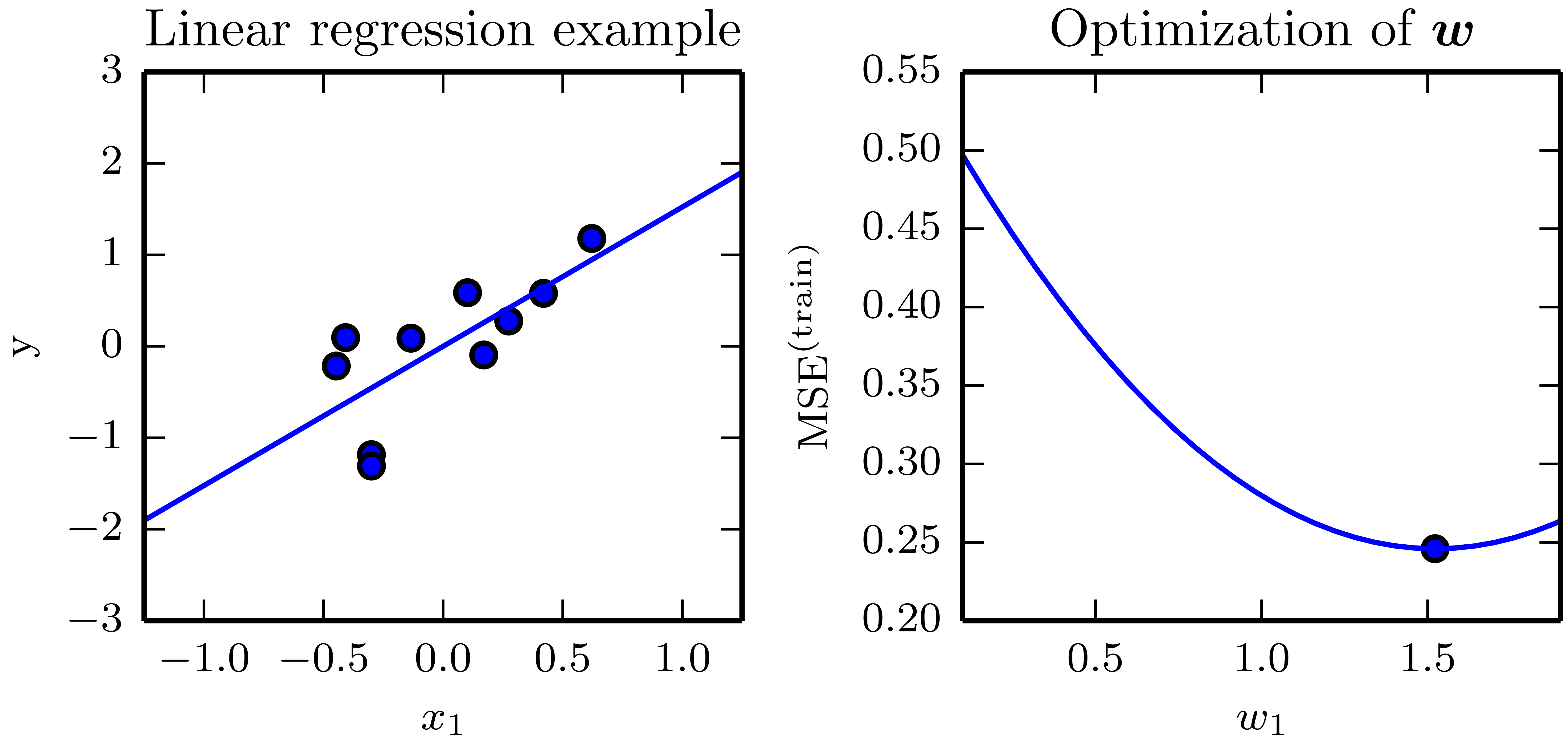
What is a linear model?

Adjust your free parameter based on some loss



What is a linear model?

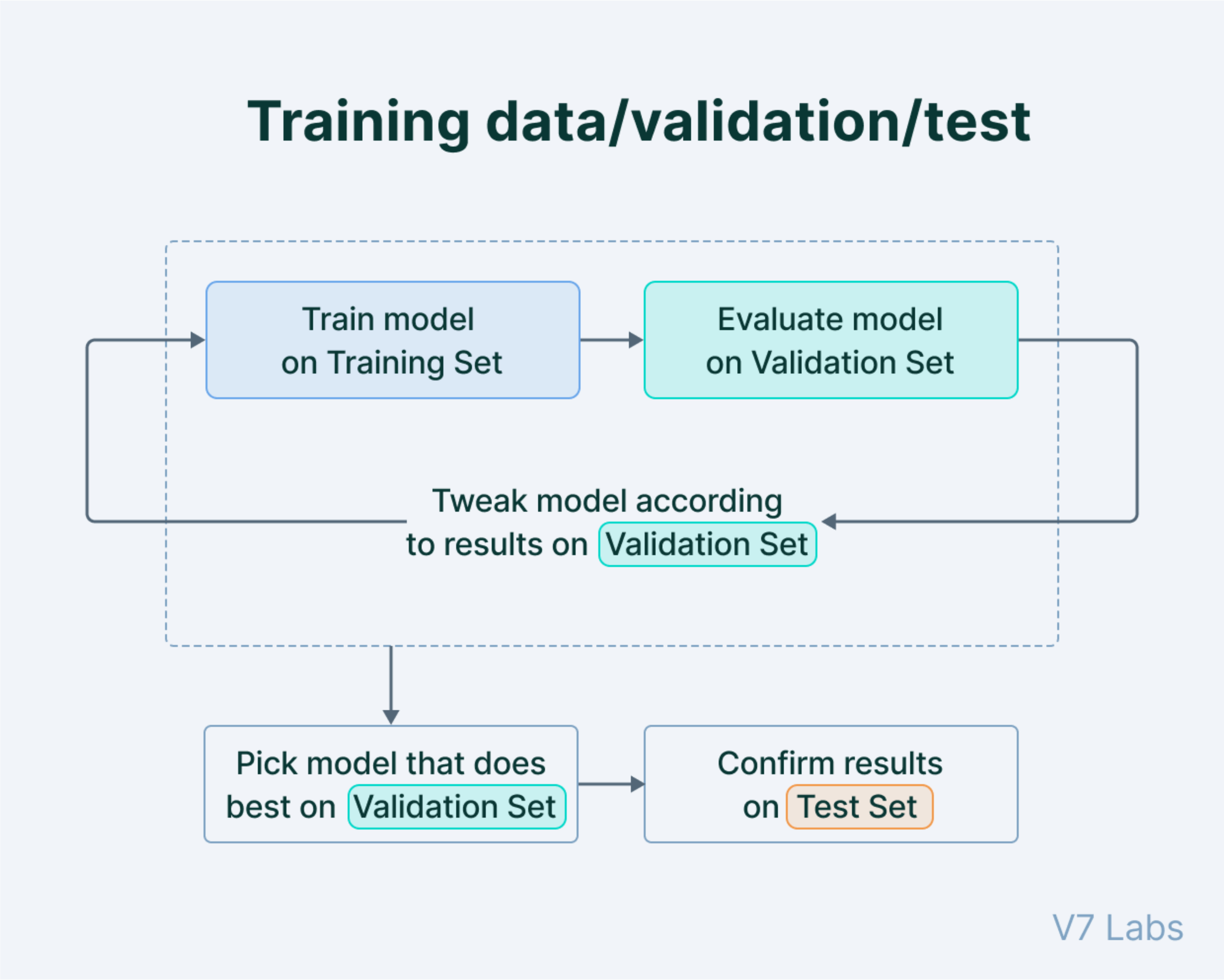
Adjust your free parameter based on some loss



MSE (Mean-squared error): $\mathrm{L}[\mathrm{f}] := \sum_{x \in X} (f(x;\theta) - y(x))^2$

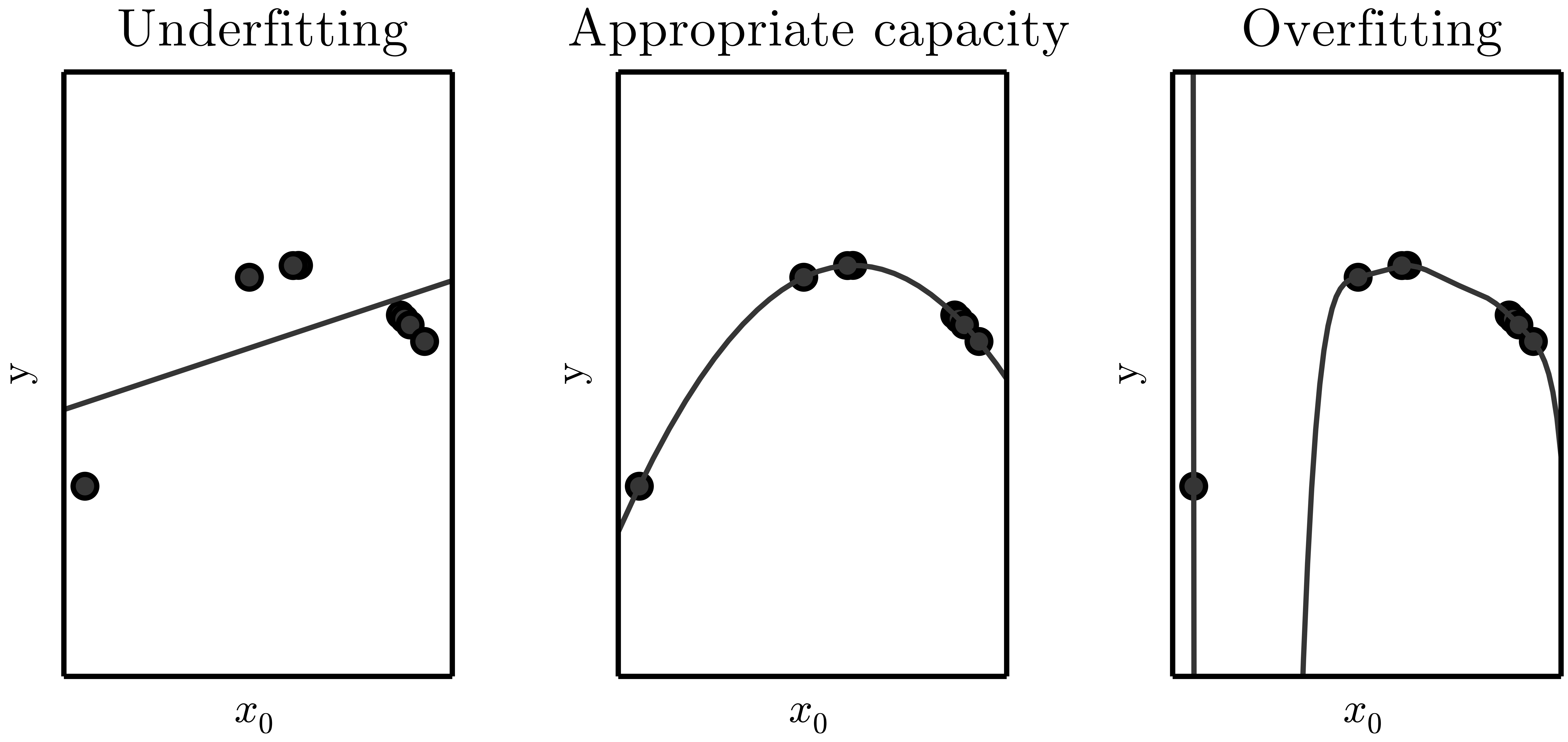
What is a train versus a test dataset?

Evaluate how well your model generalises



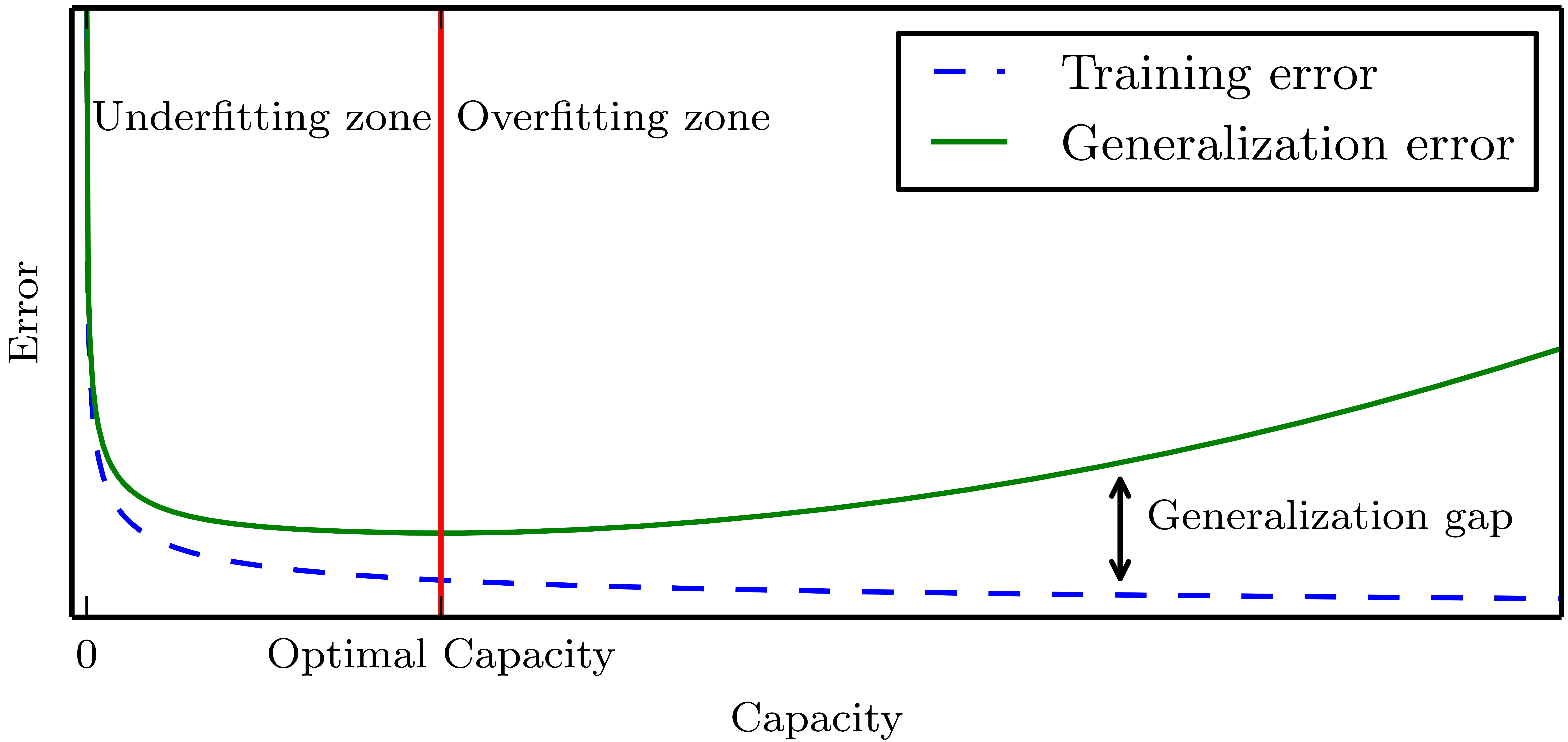
Why limit yourself to linear models?

Varying the degree of basis function results in different fits



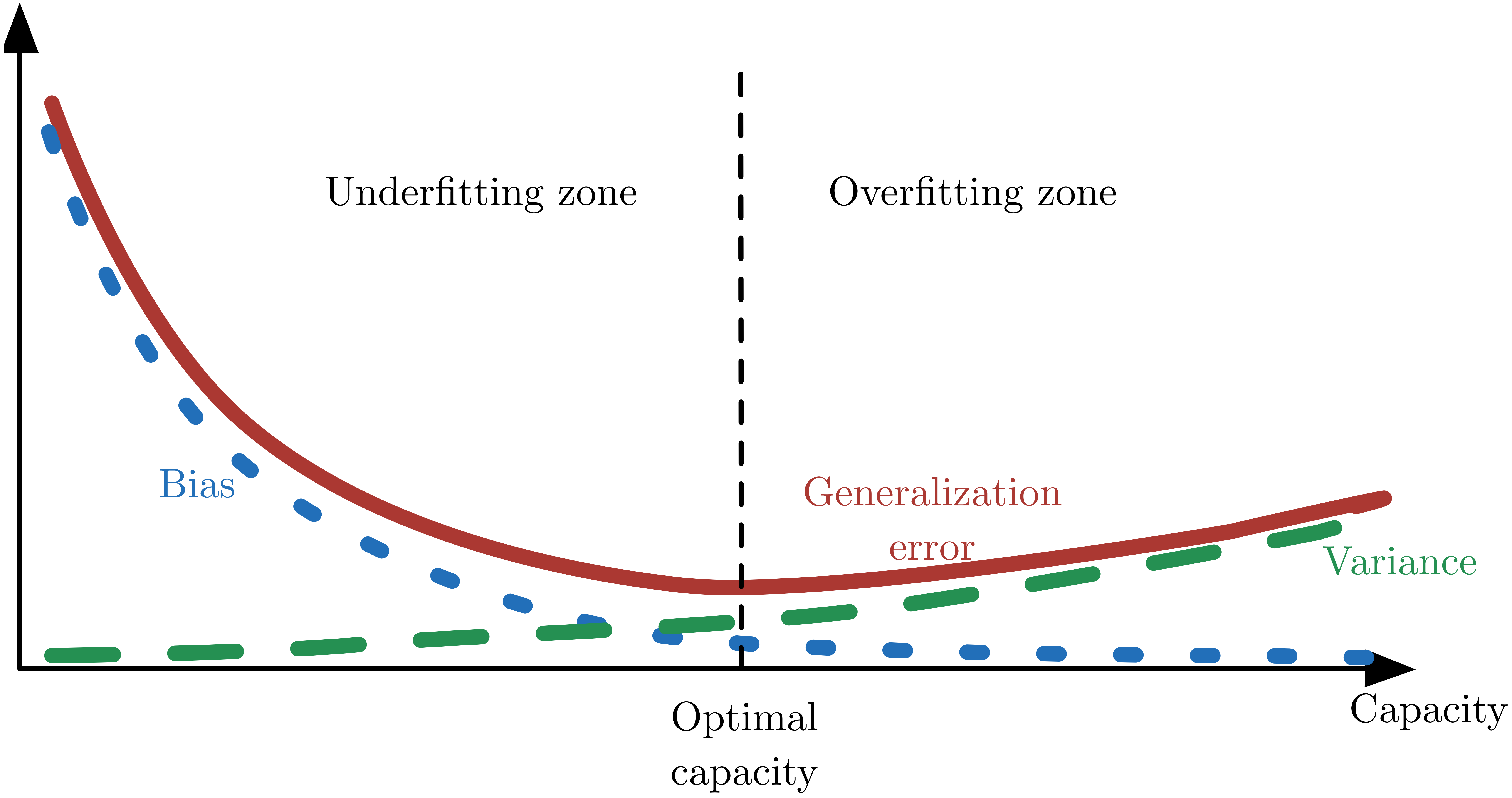
Always look at your test set!

Machine learning models like to overfit your data



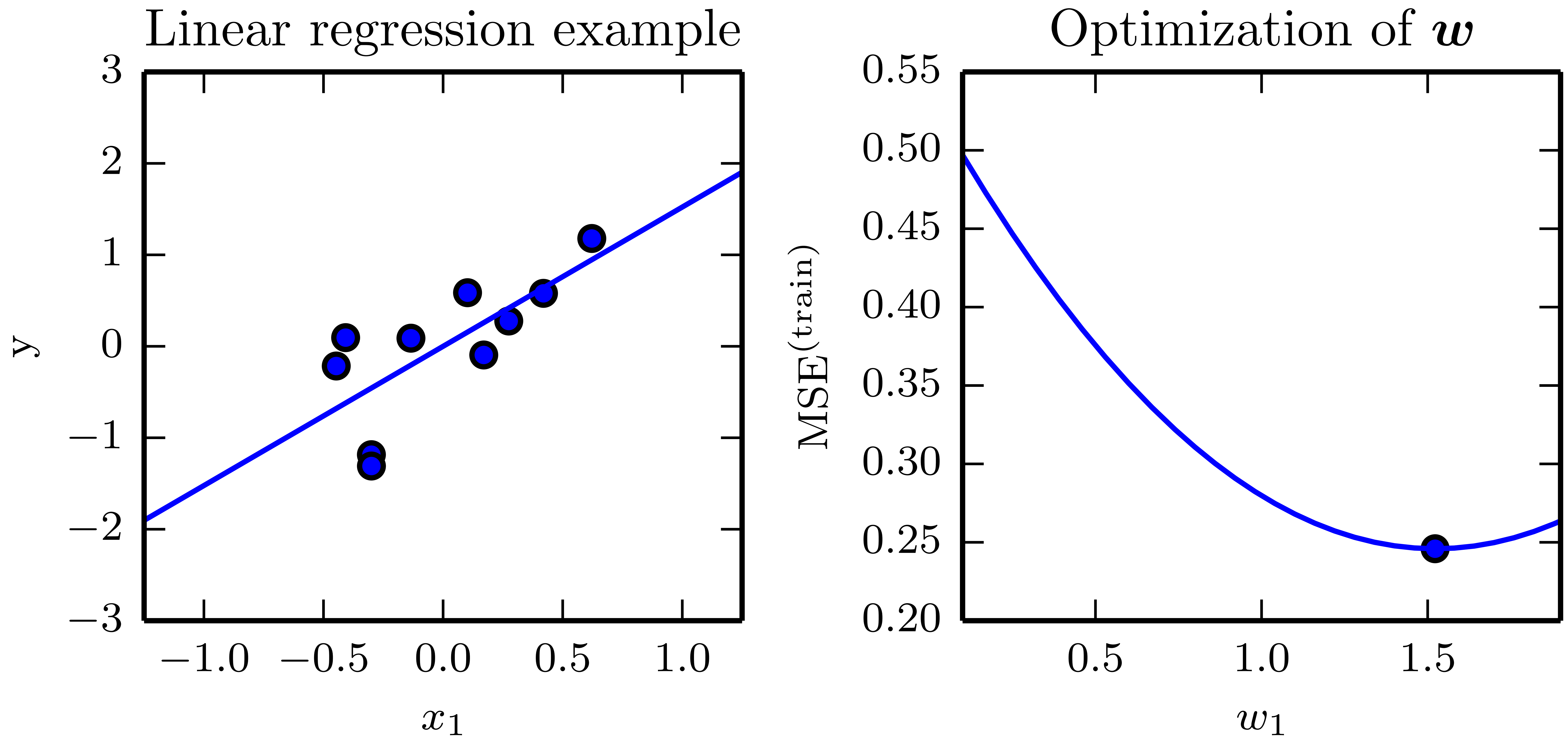
The Bias-Variance Trade-Off

B



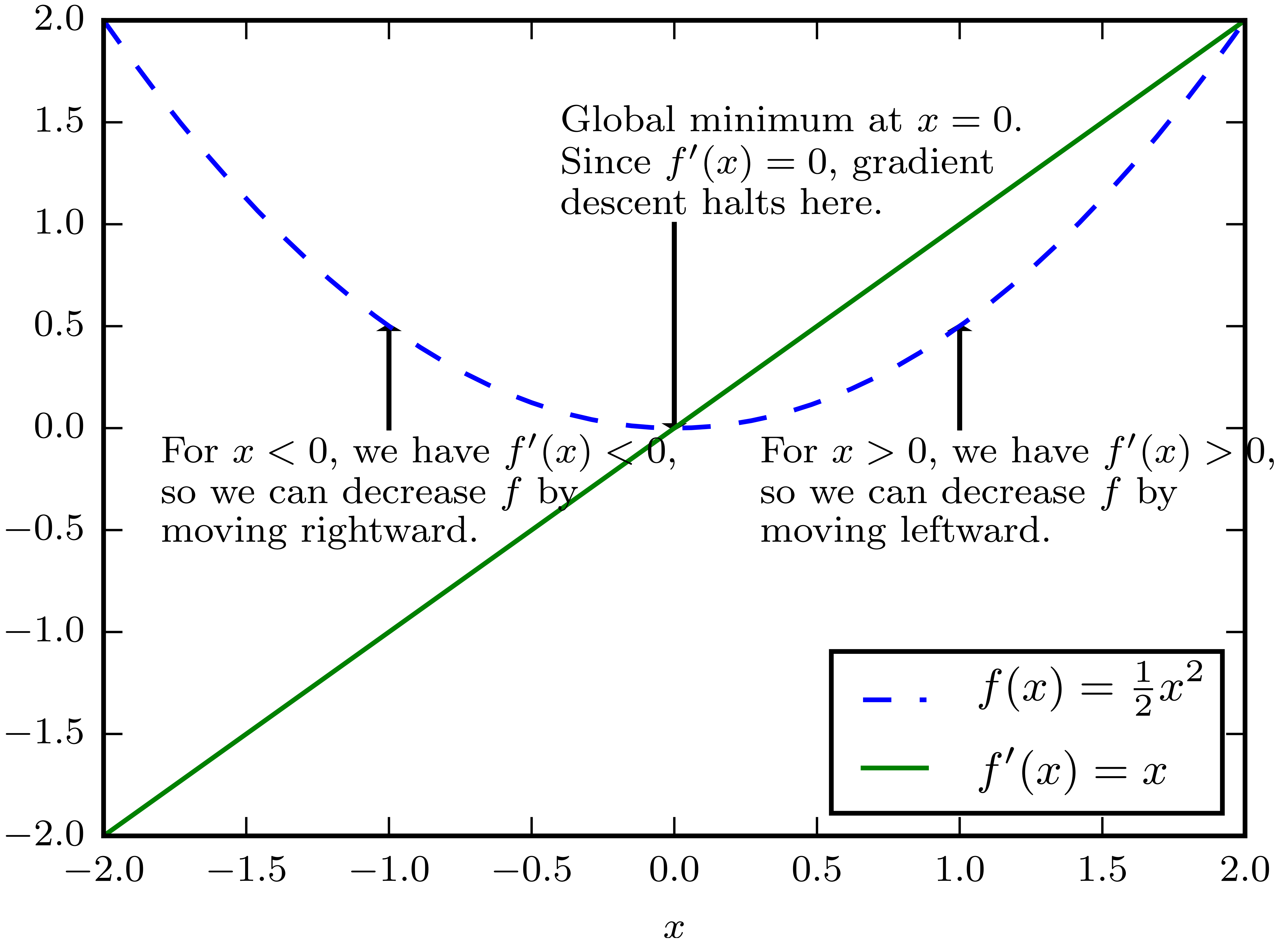
3 Gradient Descent

How do we optimize our model?



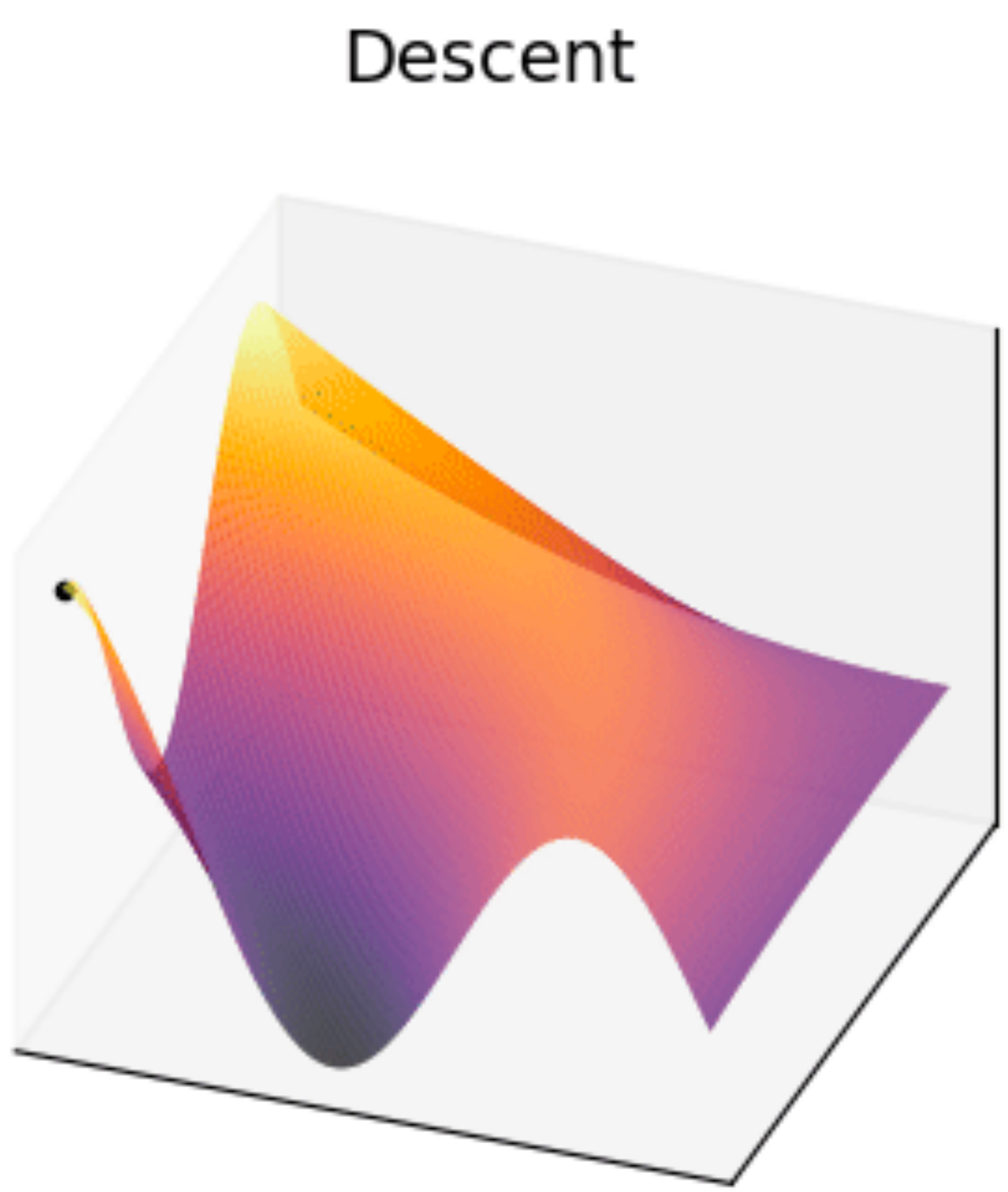
How do we optimize our model?

Follow the gradient



How can I imagine that?

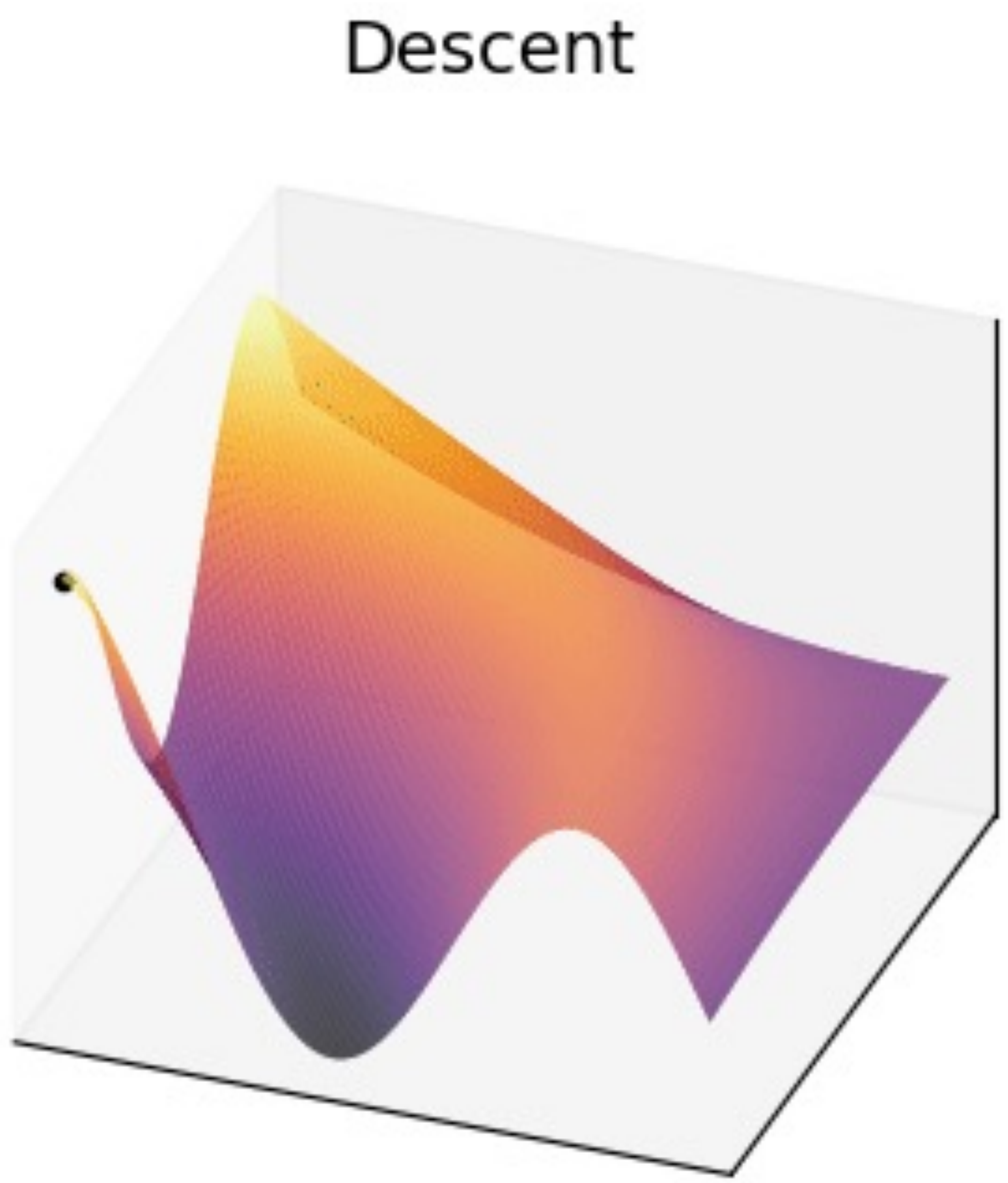
Think about a ball rolling down the loss landscape



Animations: Nick Gale

How can I imagine that?

Think about a ball rolling down the loss landscape

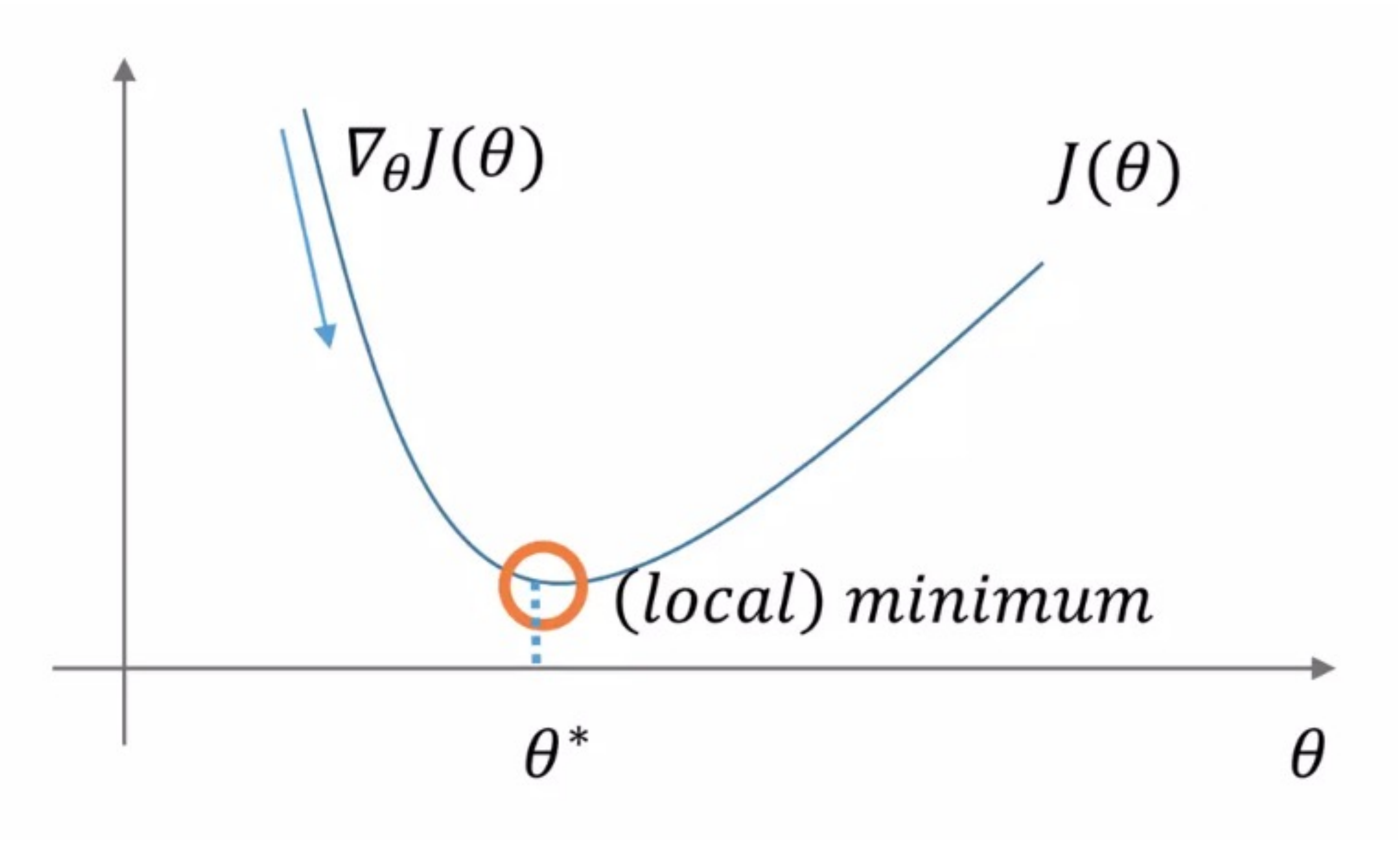


Animations: Nick Gale

Gradient Descent in formulas

Going in the opposite direction

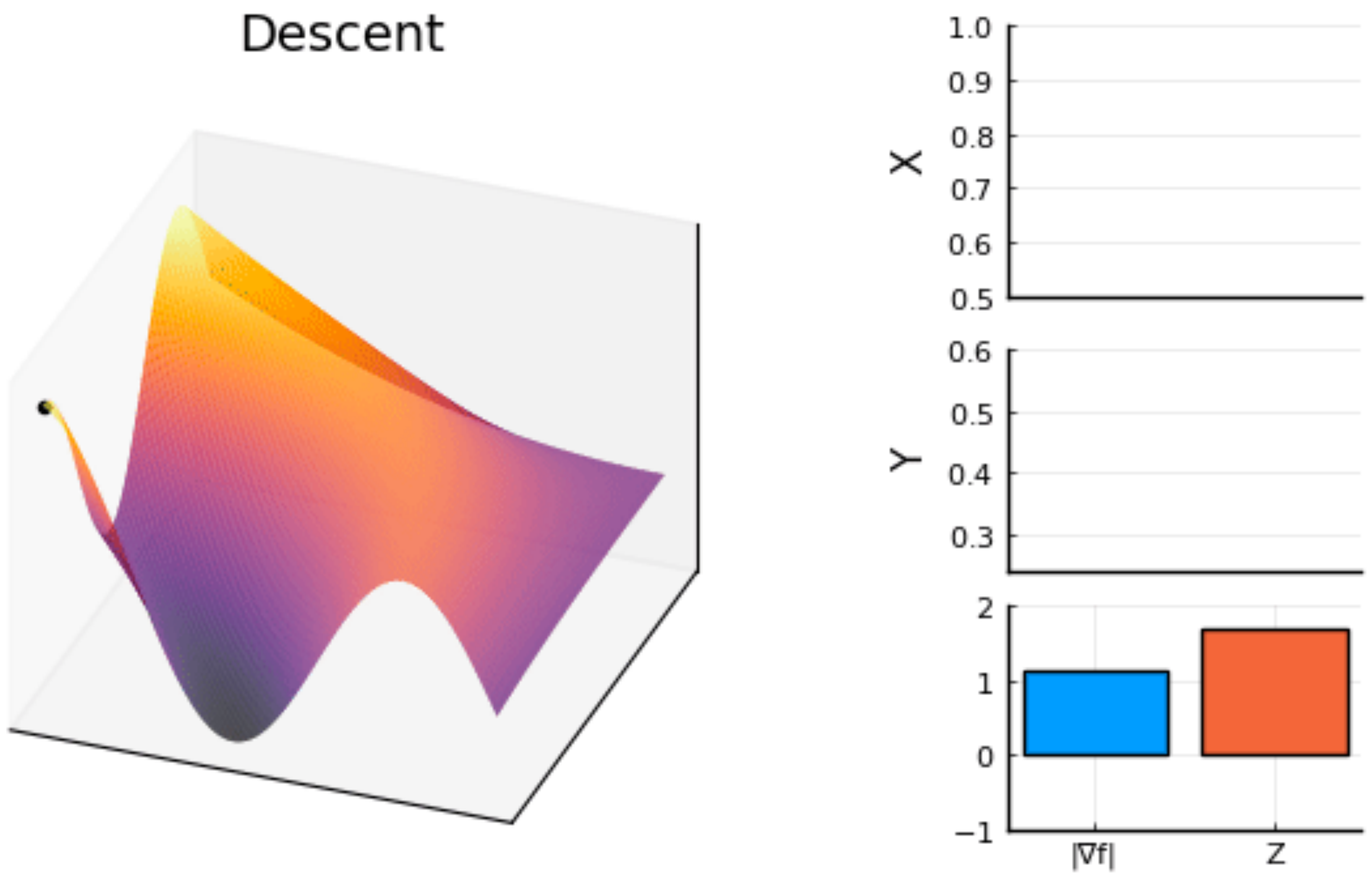
$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$



https://www.ruder.io/optimizing-gradient-descent/

Going down the gradient

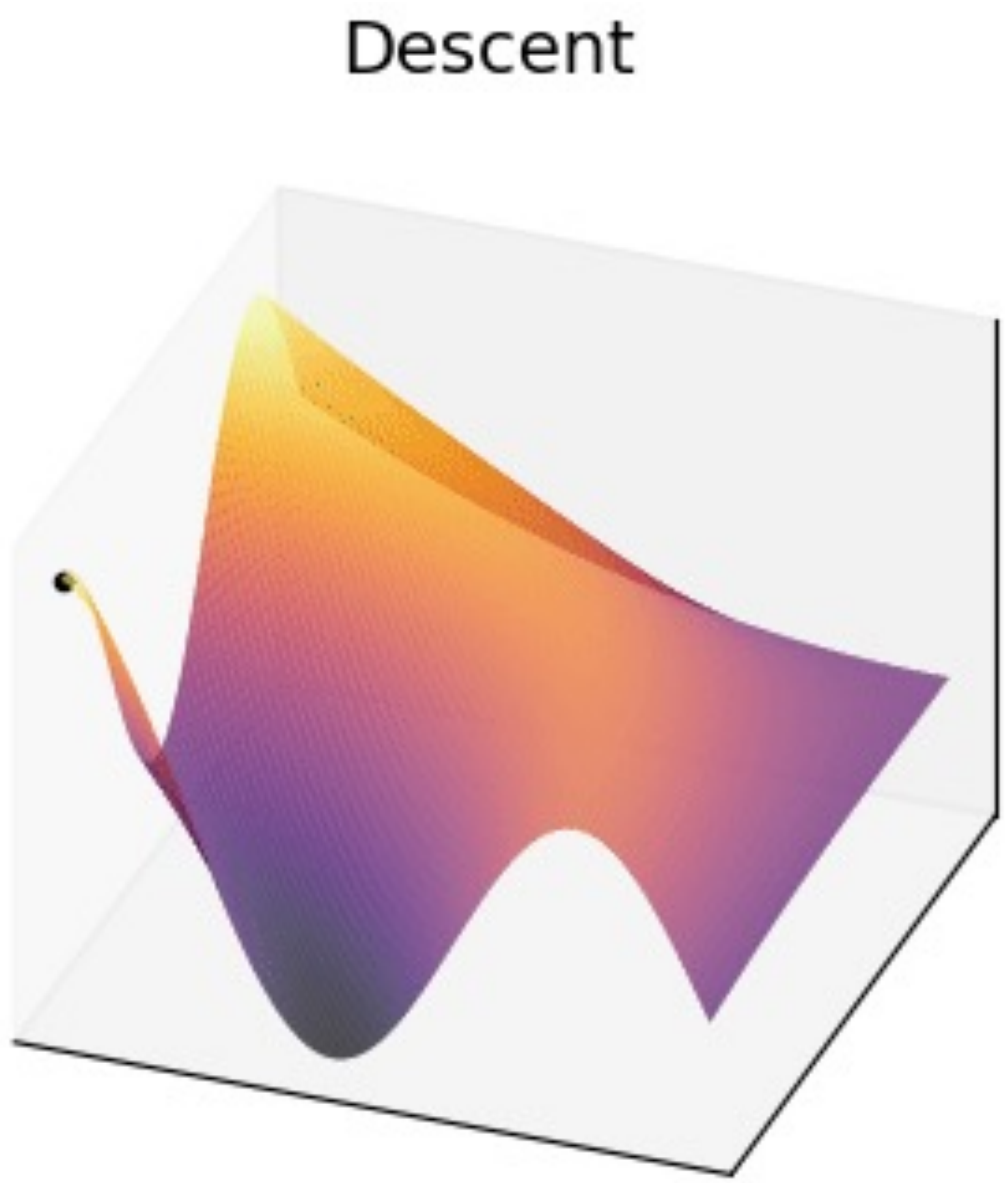
Local Minima can become a problem

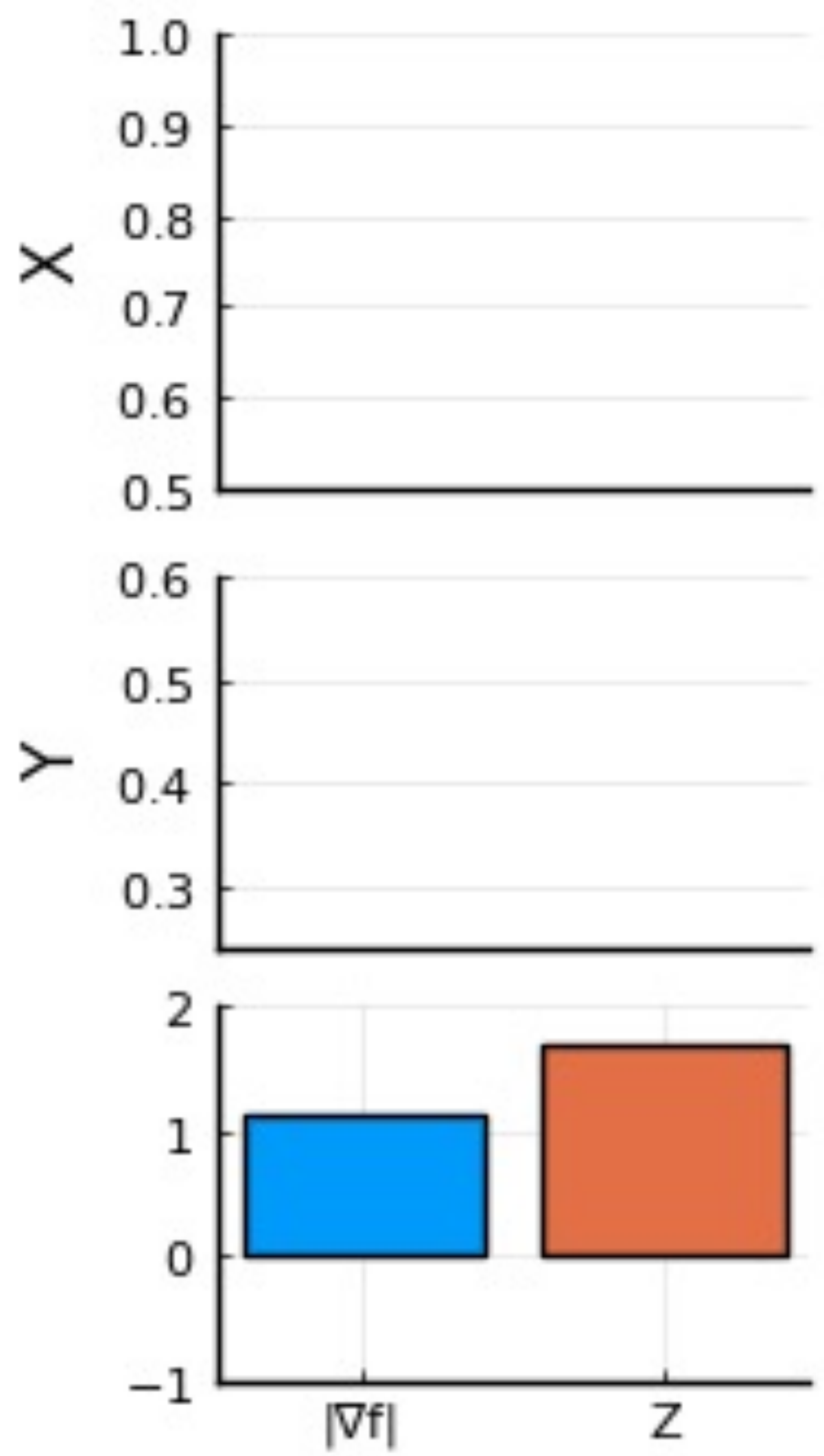


Animations: Nick Gale

Going down the gradient

Local Minima can become a problem

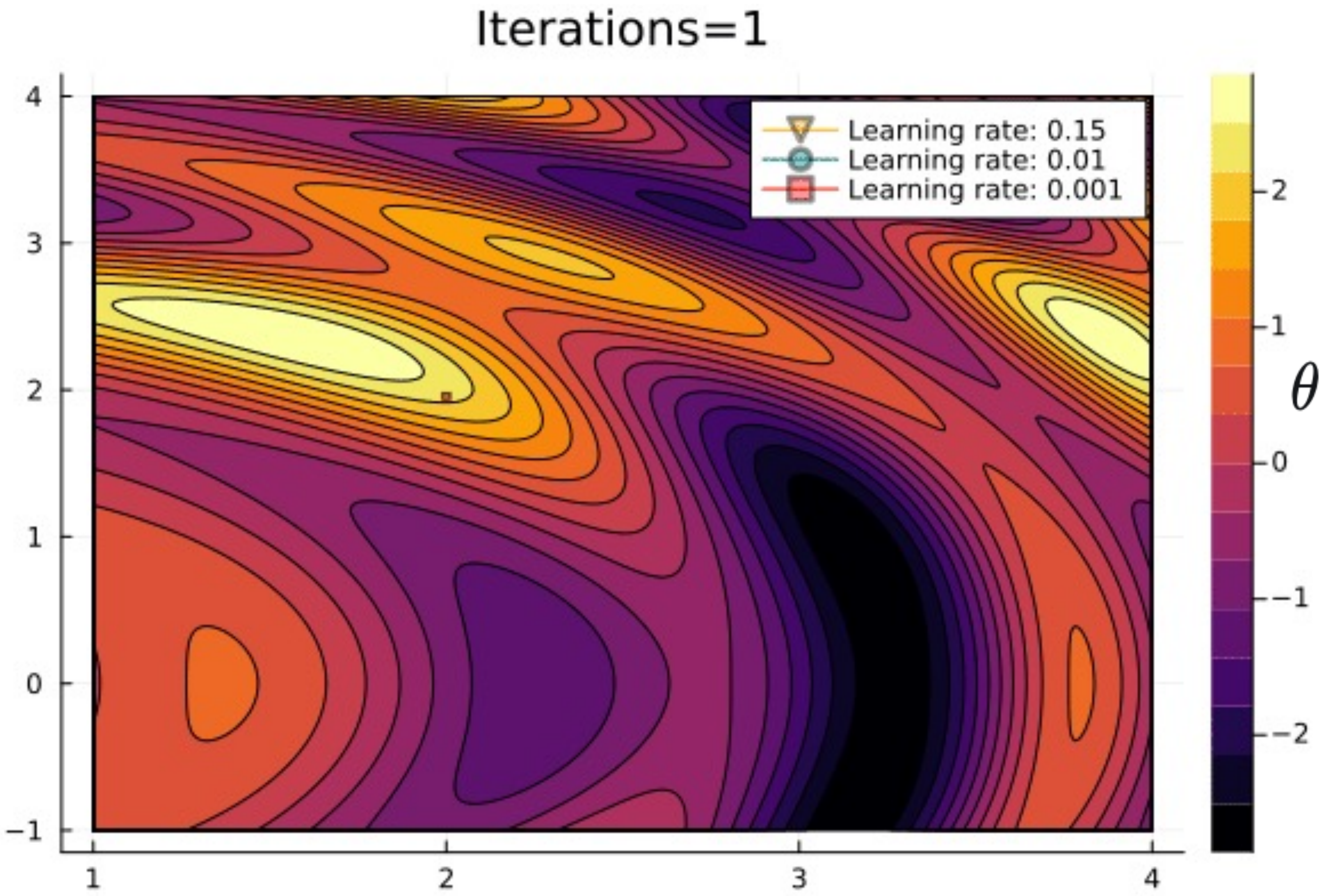




Animations: Nick Gale

Learning Rate

The first hyperparameter you should check if trouble arises



$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

Animations: Nick Gale

Vanilla Gradient Descent is slow

Speed it up by only looking at one data point at a time

Vanilla GD

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

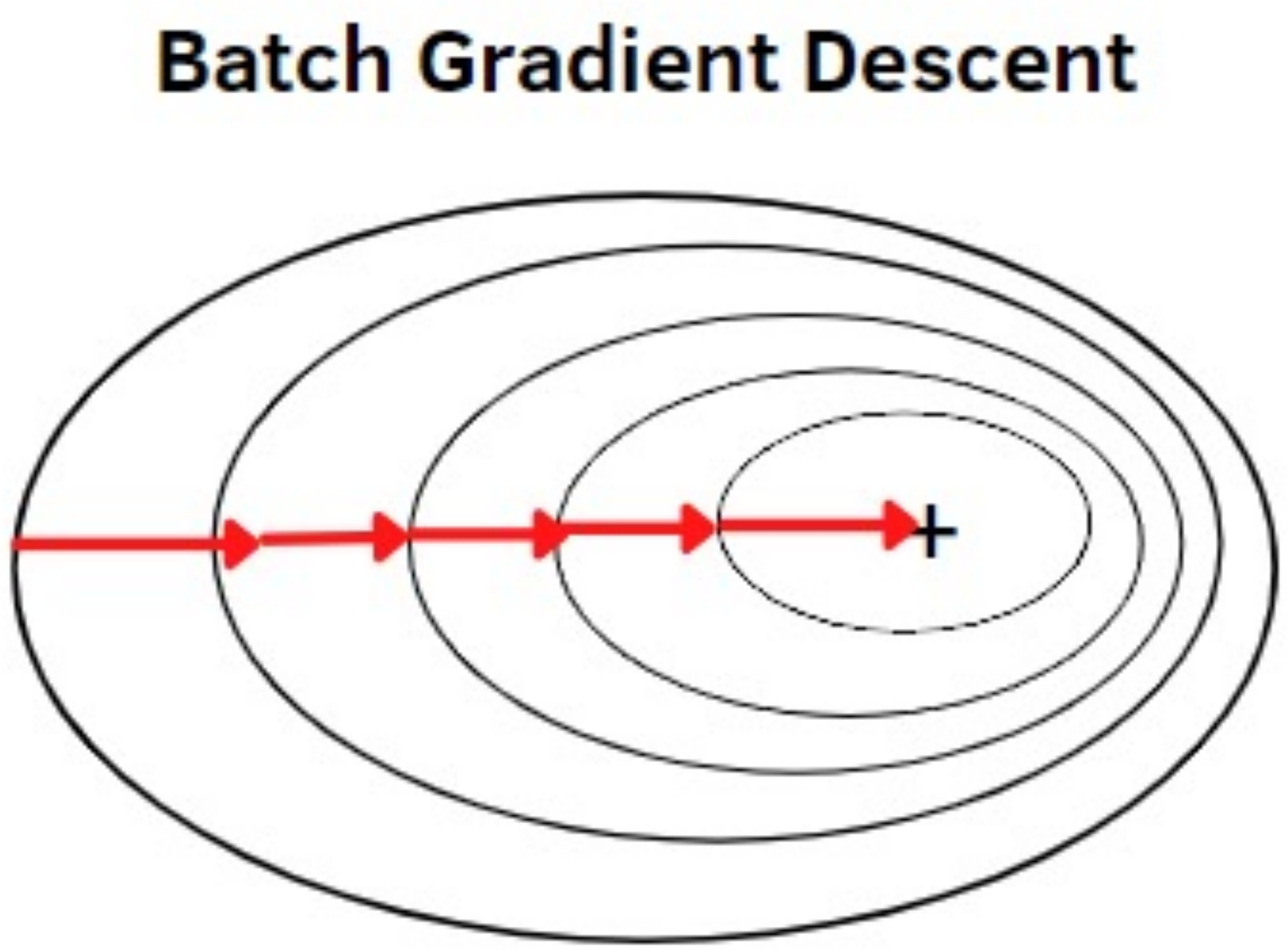
$$\Downarrow$$

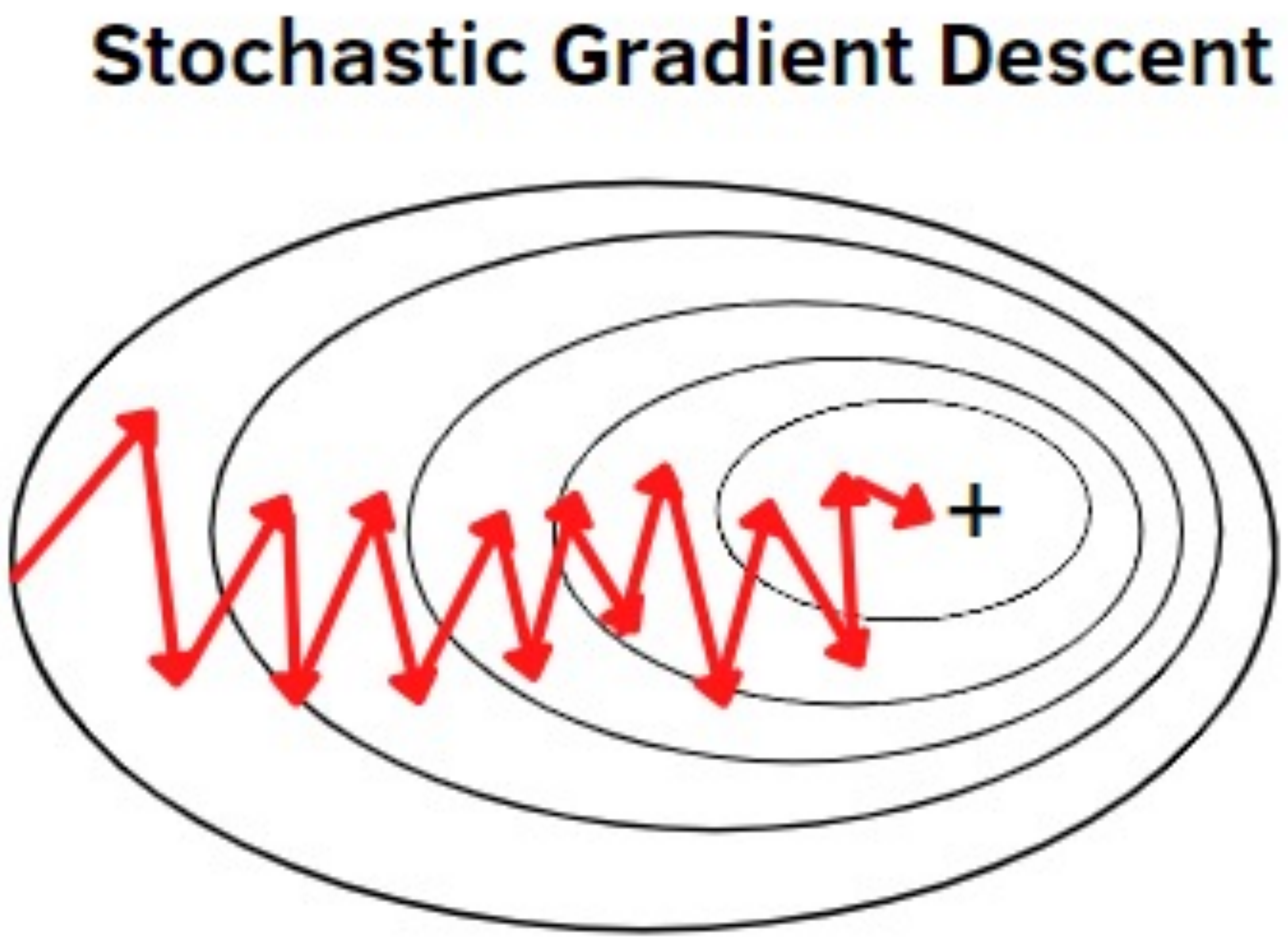
Stochastic GD

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

How can I imagine that?

Planned and cautious versus spontaneous and chaotic





Mini-batch GD: The best of both worlds

Only look at a subset of your data for each update step

Stochastic GD $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

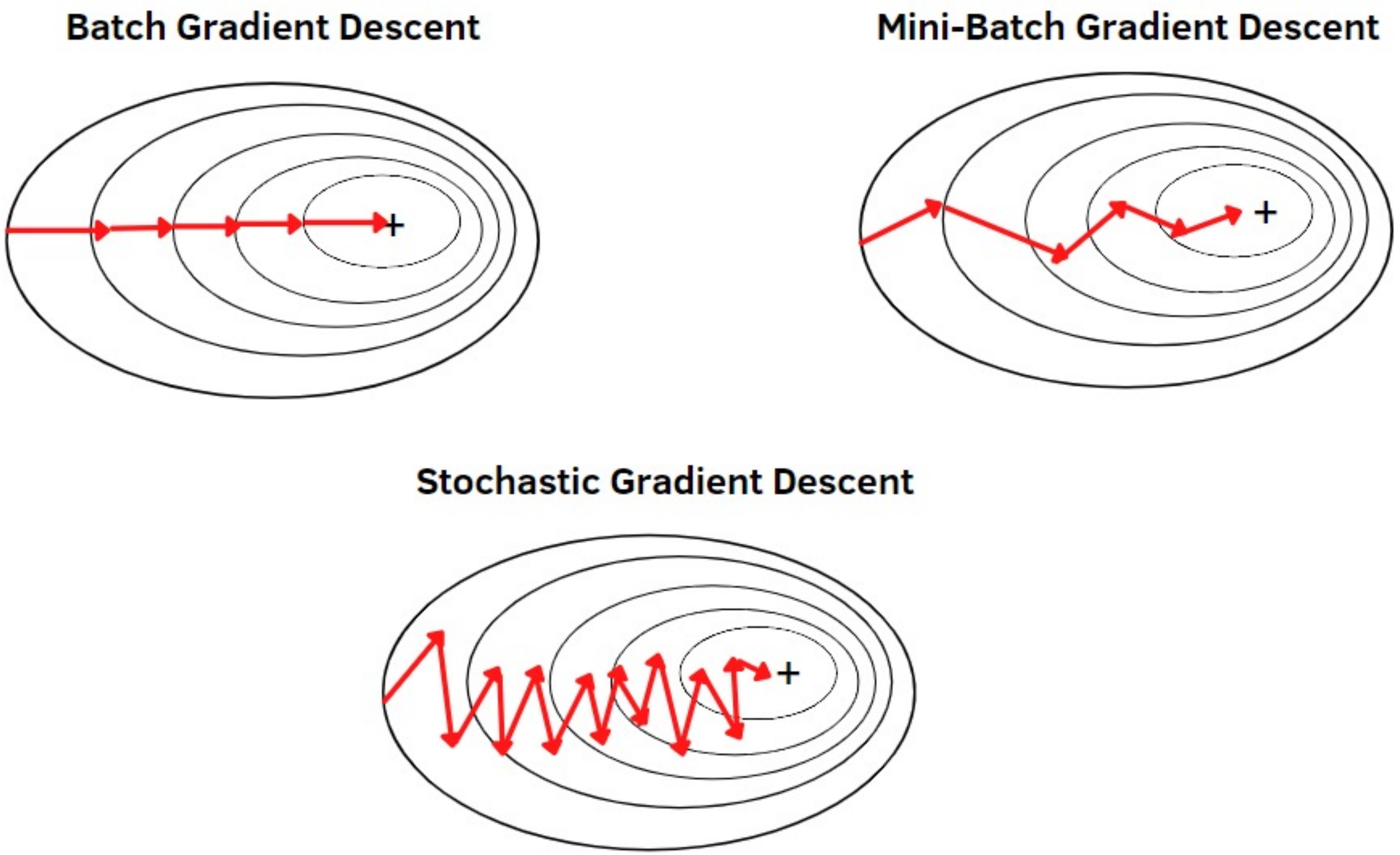
↓

Mini-batch GD $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

https://www.ruder.io/optimizing-gradient-descent/

Mini-batch GD: The best of both worlds

Only look at a subset of your data for each update step



analyticsvidhya.com

Which GD to choose?

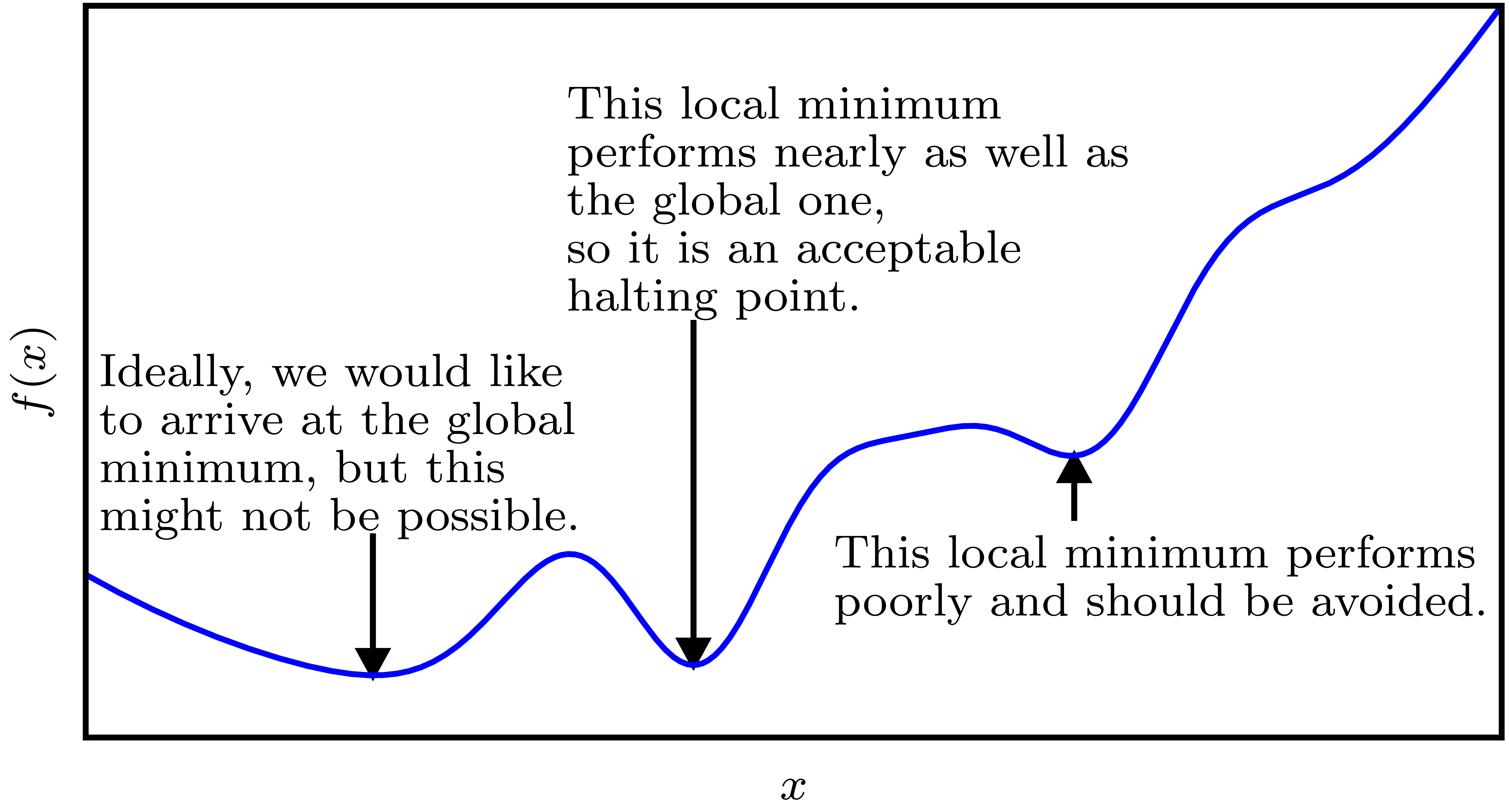
In practice, mini-batch is often a good choice

Method	Accuracy	Update Speed	Memory Usage	Online Learning
Batch gradient descent	Good	Slow	High	No
Stochastic gradient descent	Good (with annealing)	High	Low	Yes
Mini-batch gradient descent	Good	Medium	Medium	Yes

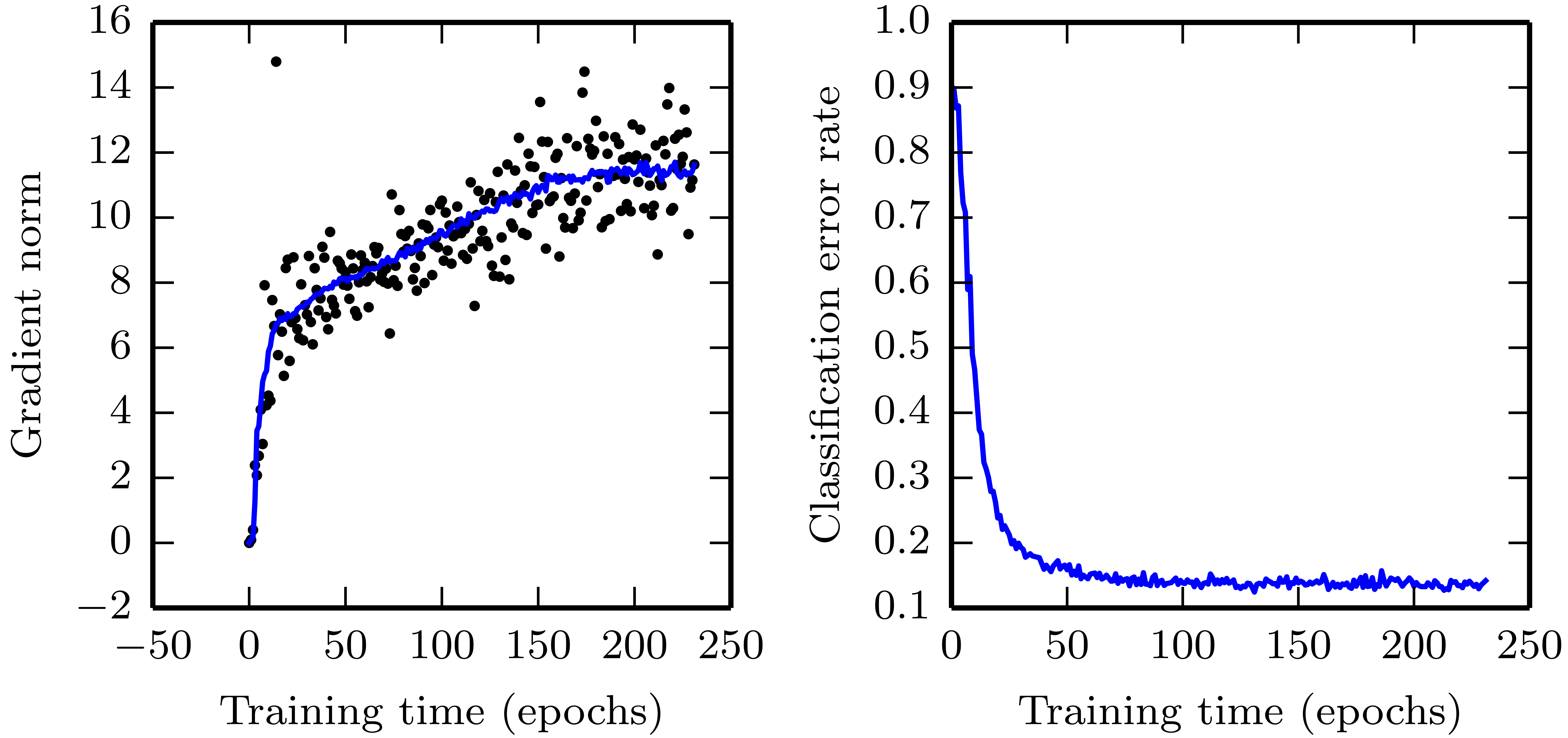
https://www.ruder.io/optimizing-gradient-descent/

Real-life optimisation is hard

We do not expect to find the global minimum in most cases

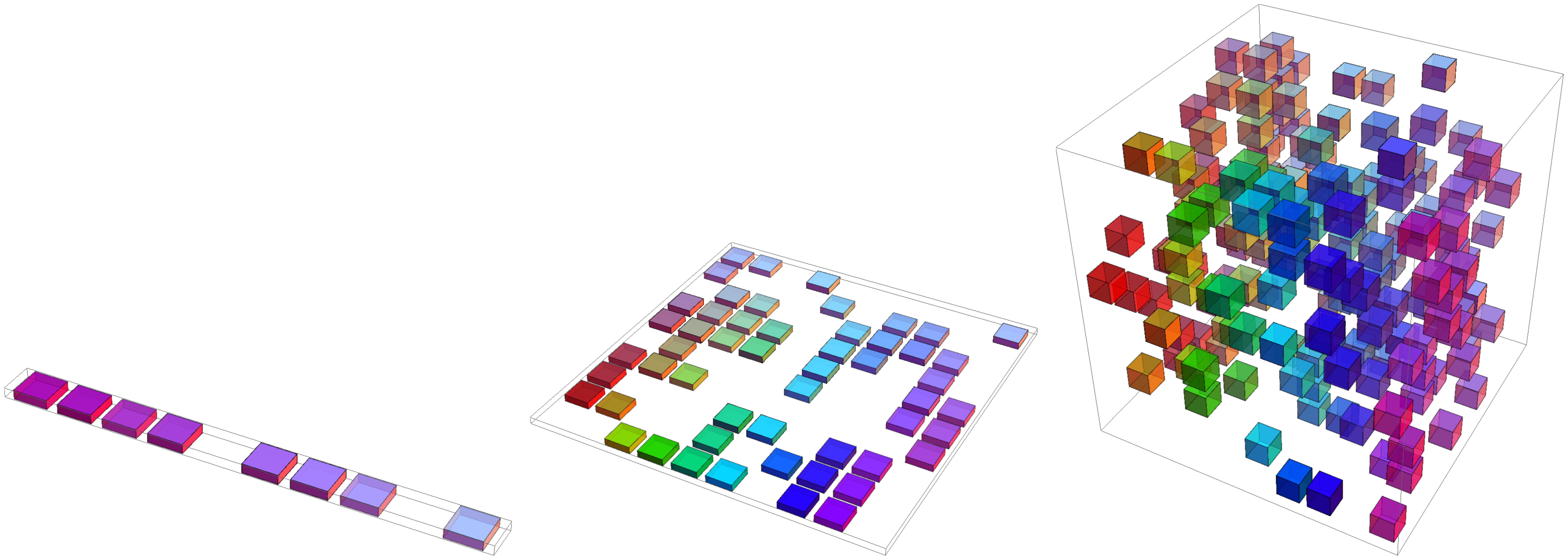


We usually don't even reach a local minimum



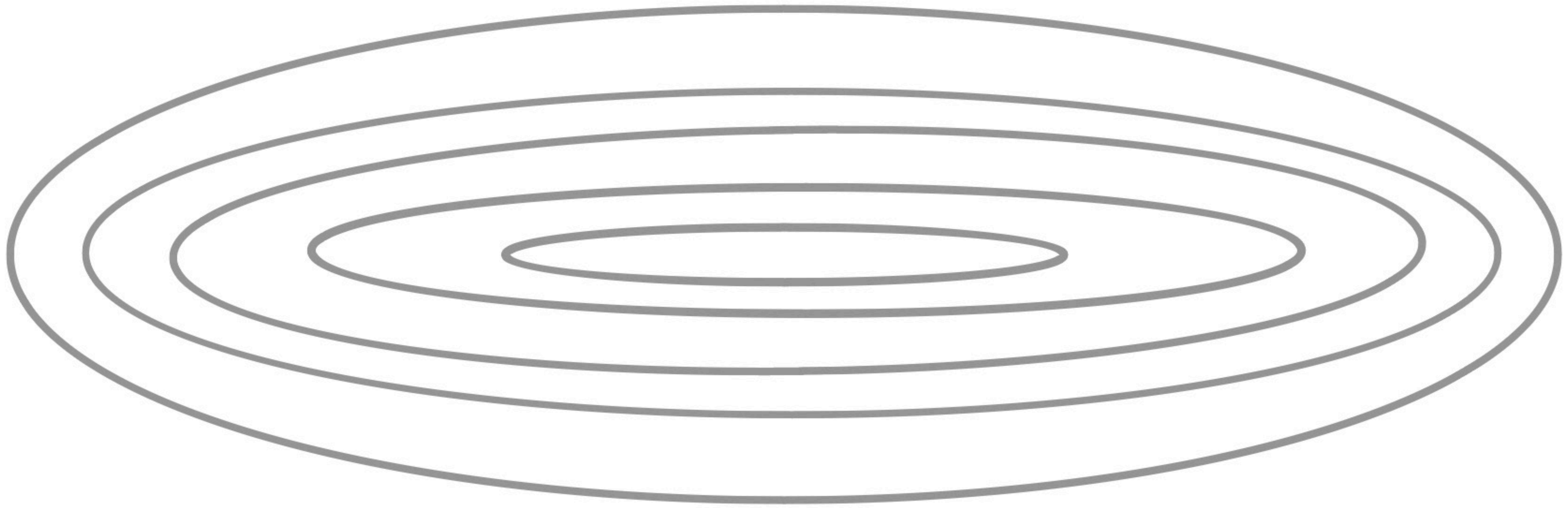
Curse of Dimensionality

The higher-dimensional the data, the more we need of it



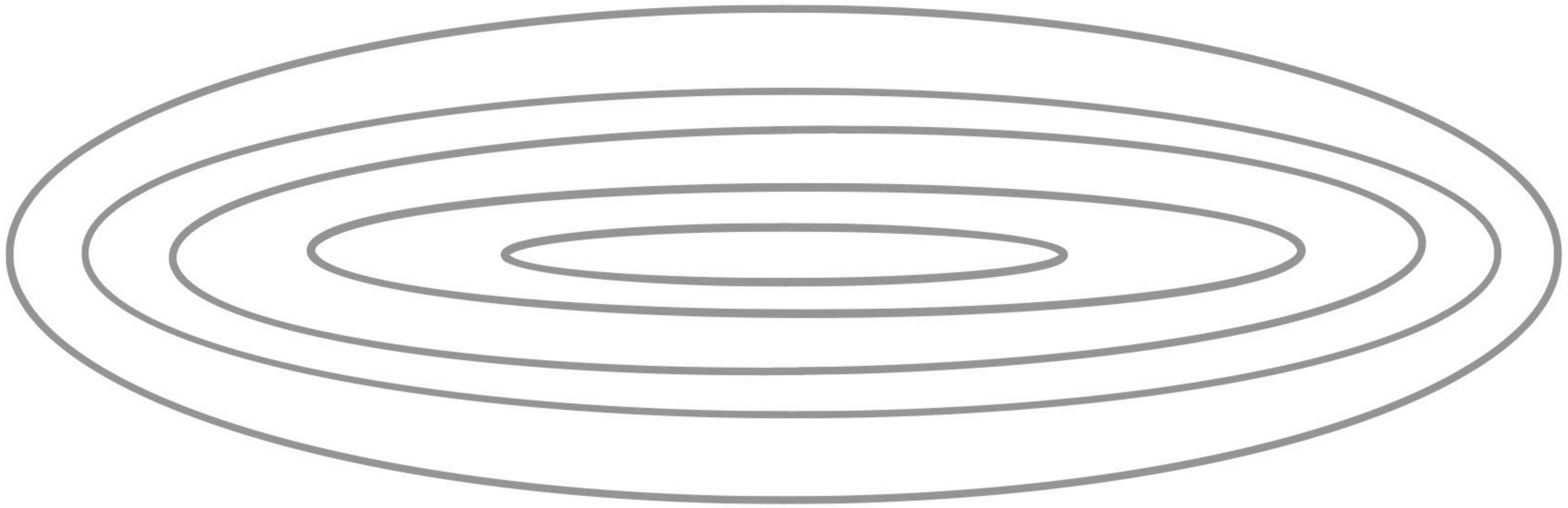
Momentum Optimisers

Have a memory of the past to overcome dire times



Momentum Optimisers

Have a memory of the past to overcome dire times



Momentum Optimisers

Have a memory of the past to overcome dire times

Vanilla GD

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

$$\downarrow$$

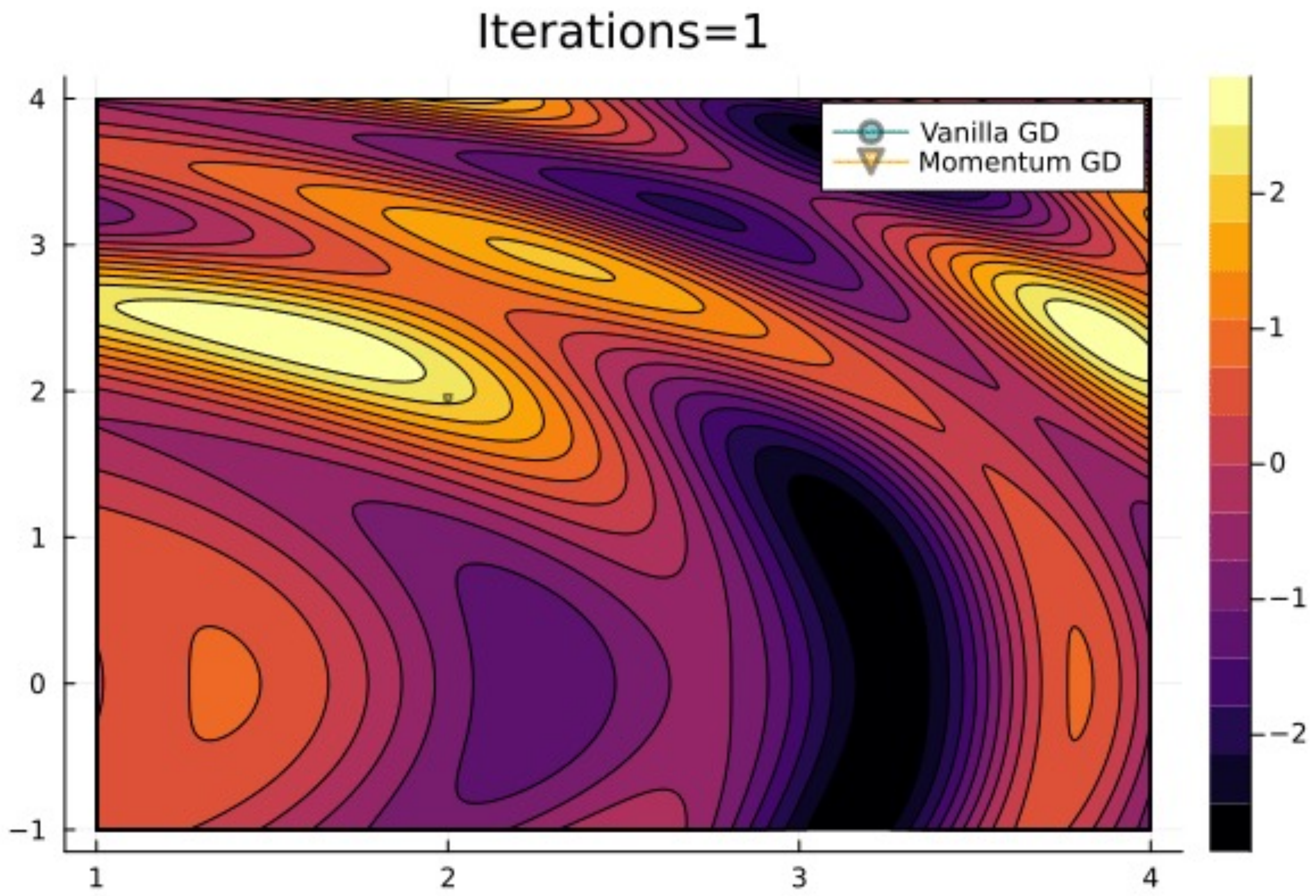
Momentum GD

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

https://www.ruder.io/optimizing-gradient-descent/

Momentum

Speeding things up when the gradient becomes shallow



Animations: Nick Gale

A Zoo of Momentum Methods…

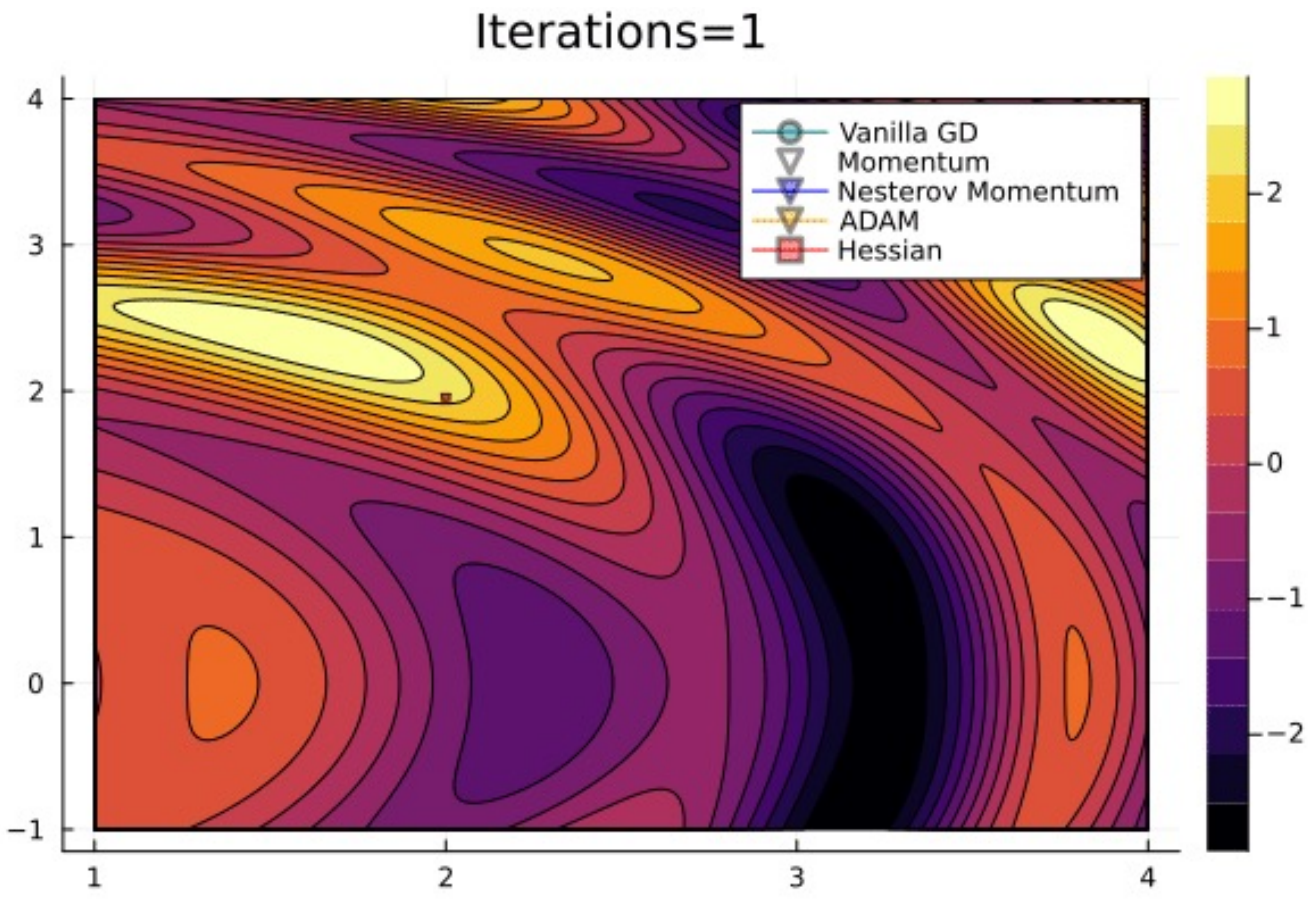
ADAM is the standard default choice

Method	Update equation
SGD	$g_t = \nabla_{\theta_t} J(\theta_t)$ $\Delta\theta_t = -\eta \cdot g_t$ $\theta_t = \theta_t + \Delta\theta_t$
Momentum	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta g_t$
NAG	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
Adagrad	$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
Adadelta	$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
RMSprop	$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Adam	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

https://www.ruder.io/optimizing-gradient-descent/

A Zoo of Momentum Methods…

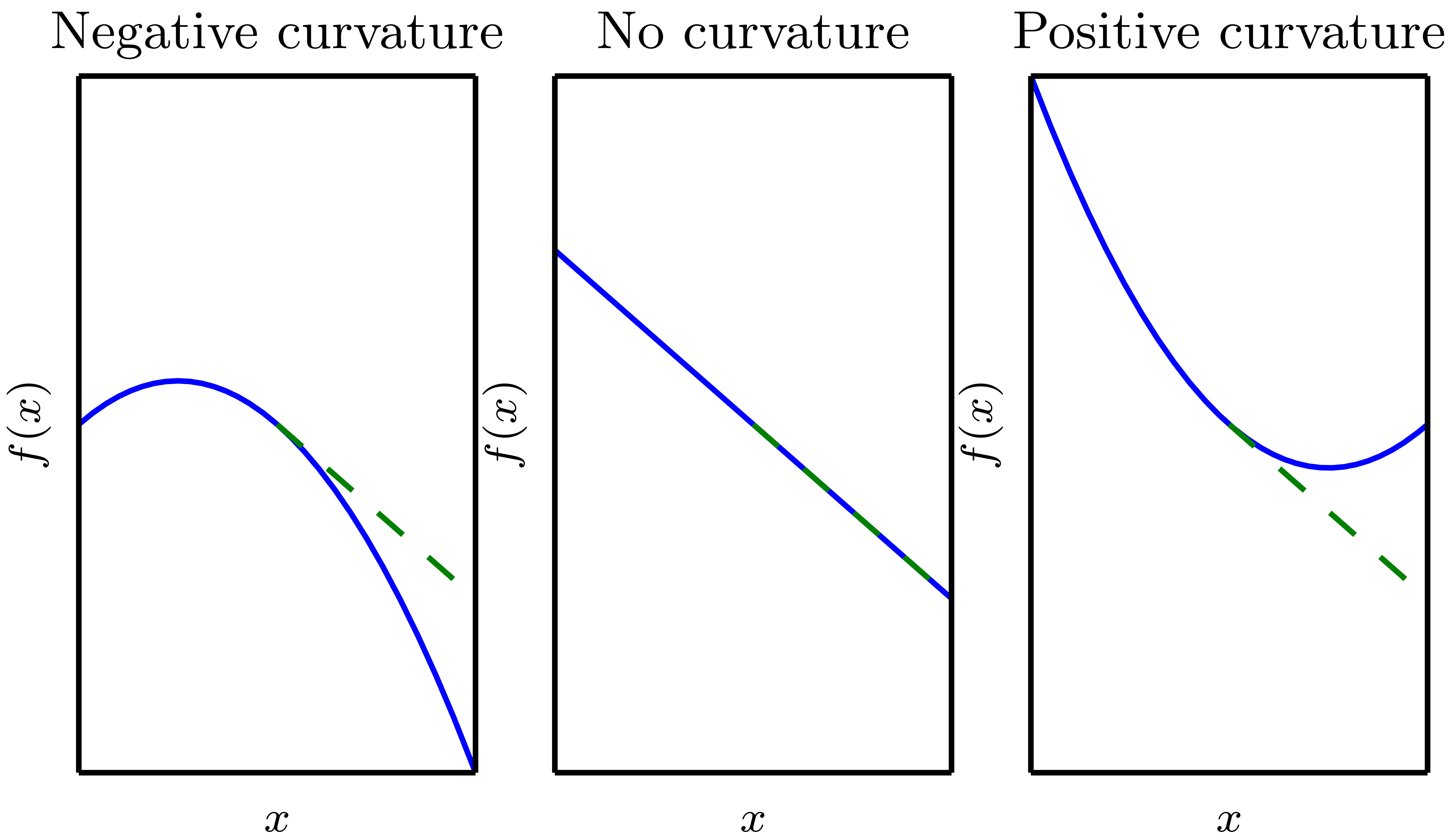
ADAM is the standard default choice



Animations: Nick Gale

Back to learning rate: Is there an optimum?

Curvature helps out



Newton's method: Using curvature

The Hessian tells you your optimal step size

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}. \tag{4.9}$$

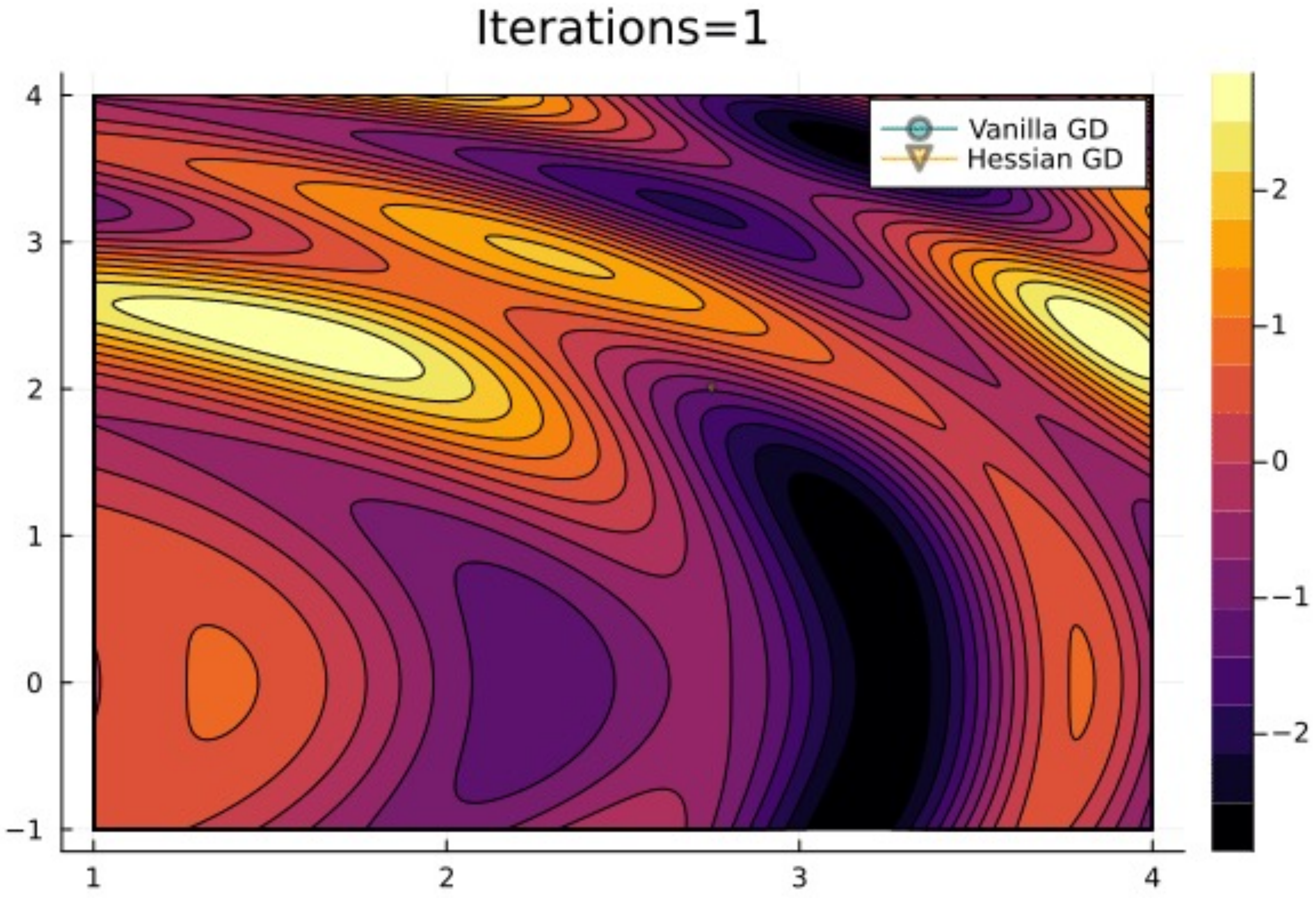
$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}. \tag{4.10}$$

Big gradients speed you up

Big eigenvalues slow you down if you align with their eigenvectors

Hessian

Each step is better, but evaluating each one is very expensive

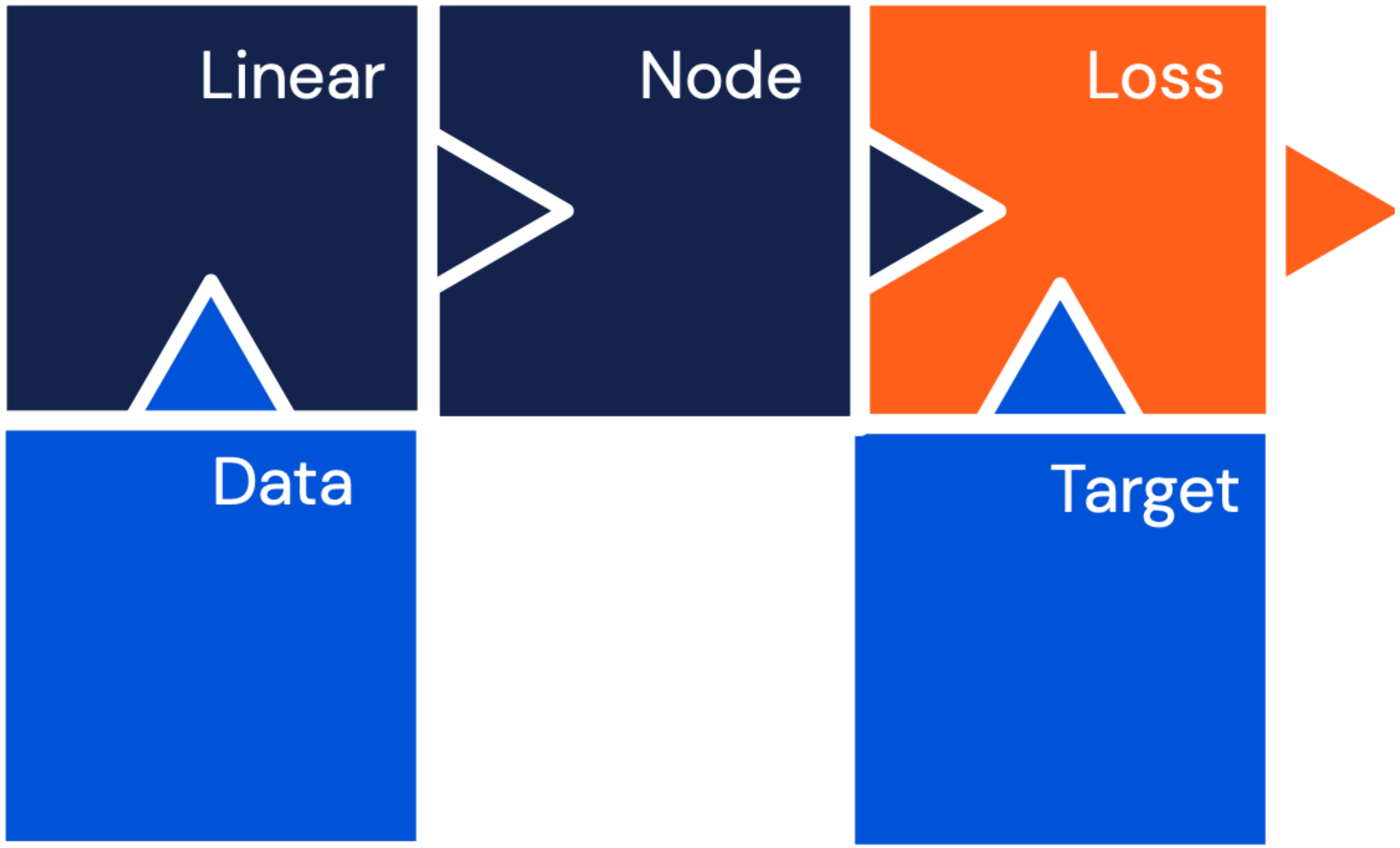


Animations: Nick Gale

4 Deep Learning

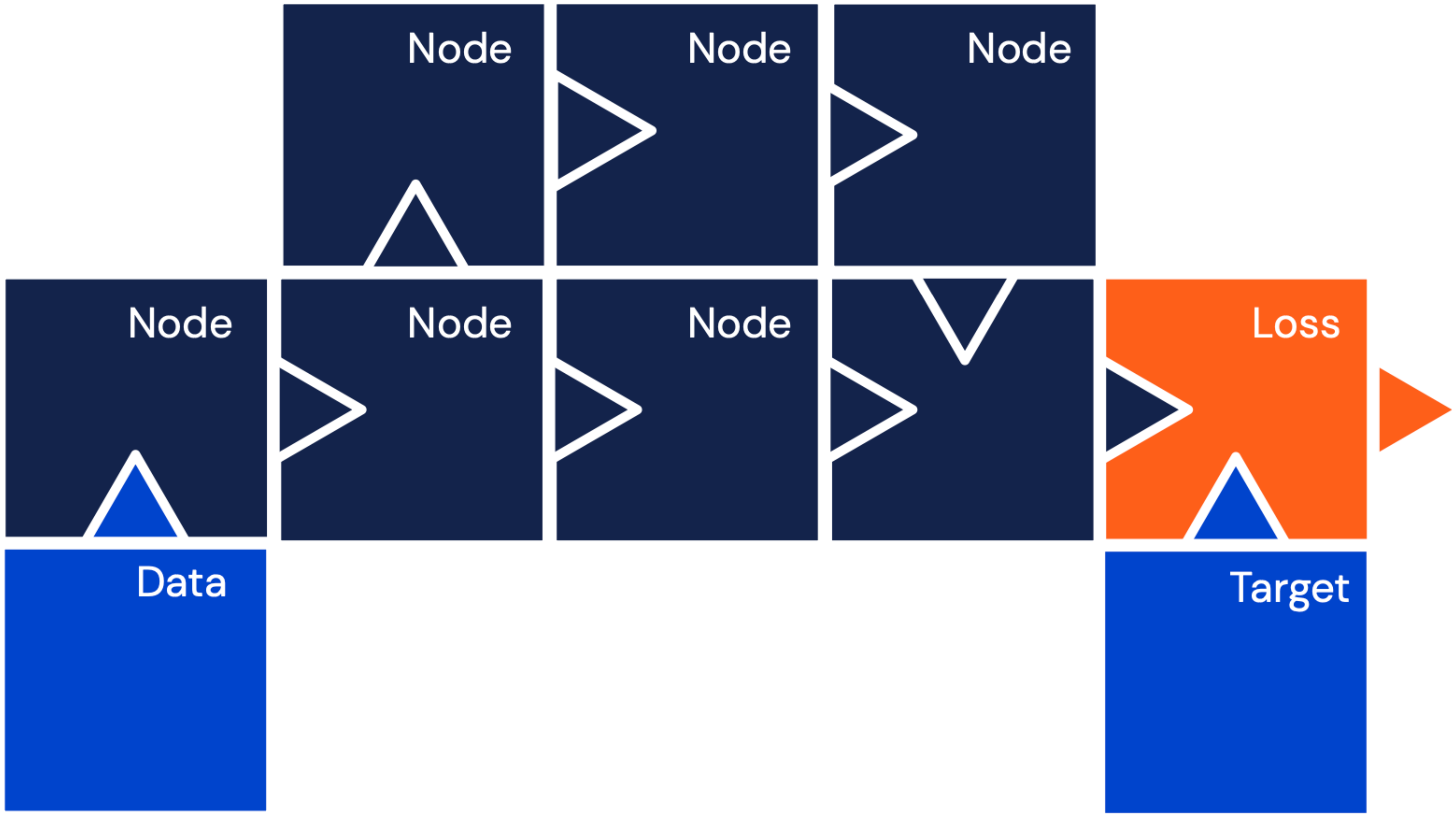
Deep Learning as LEGO for adults

How to build your best model



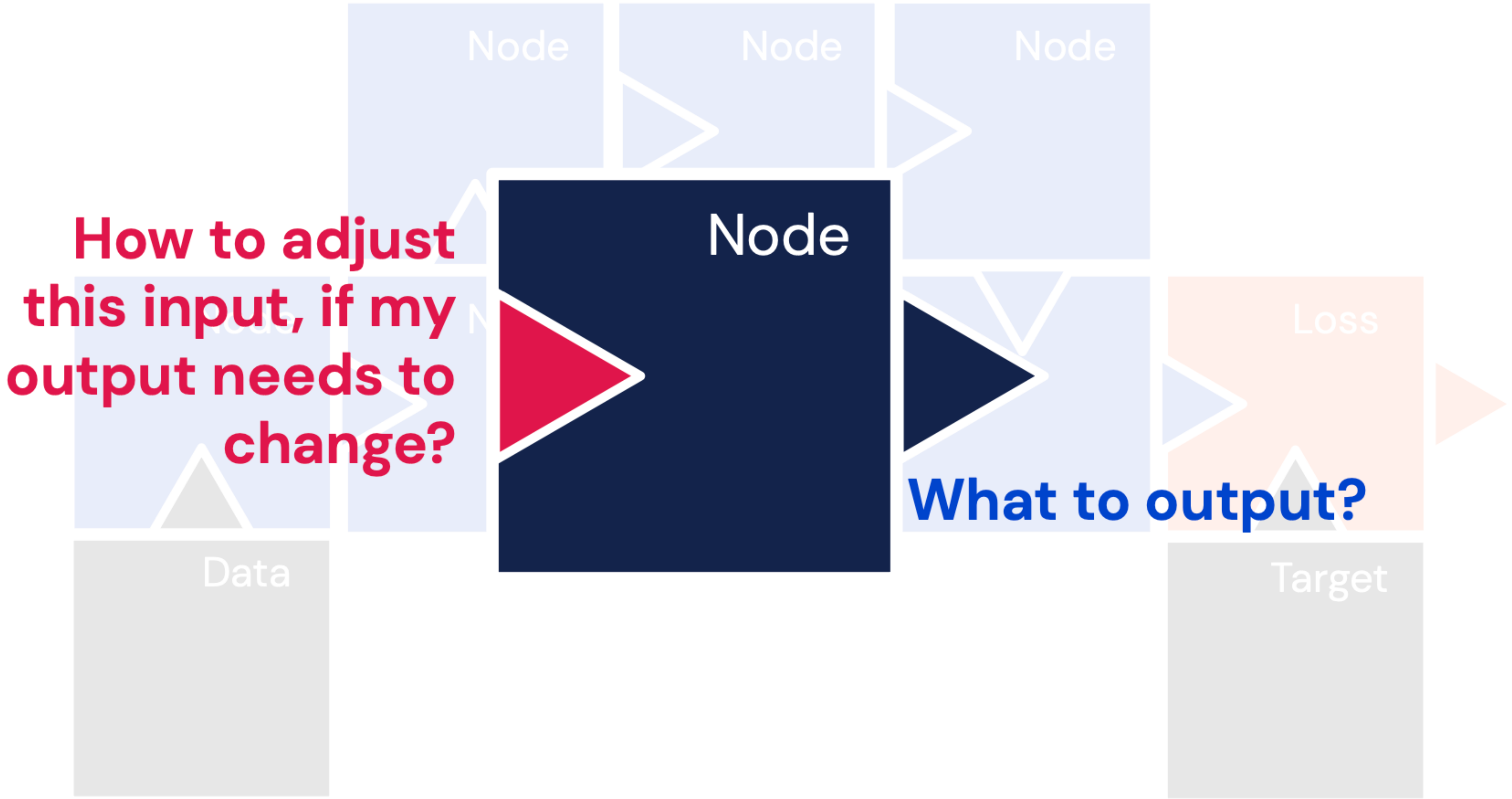
Deep Learning as LEGO for adults

How to build your best model



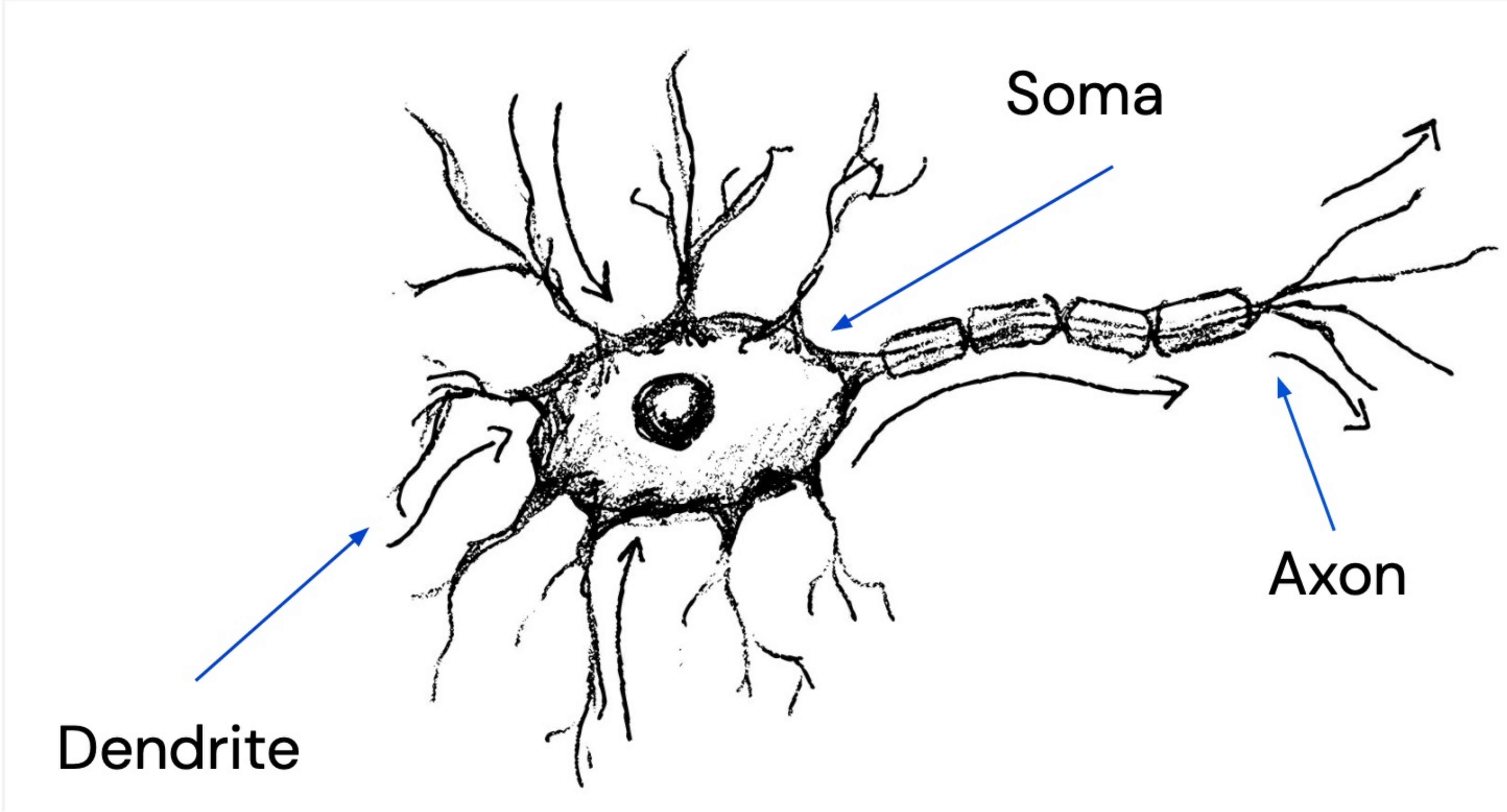
A

B

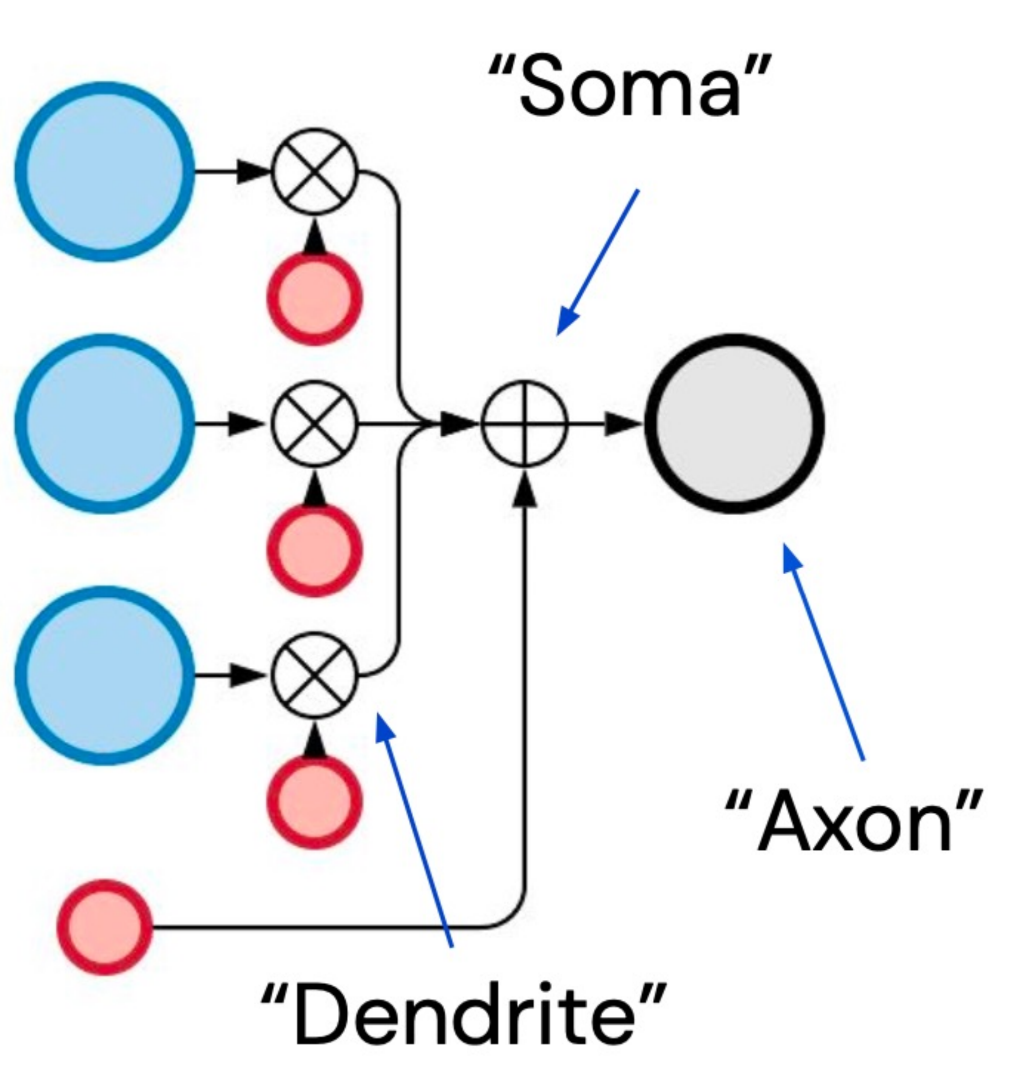

Node
Node
Node
Node
How to adjust this input, if my output needs to change?
What to output?
Loss
Data
Target

Artificial Neurons

Not biologically plausible, but still very useful



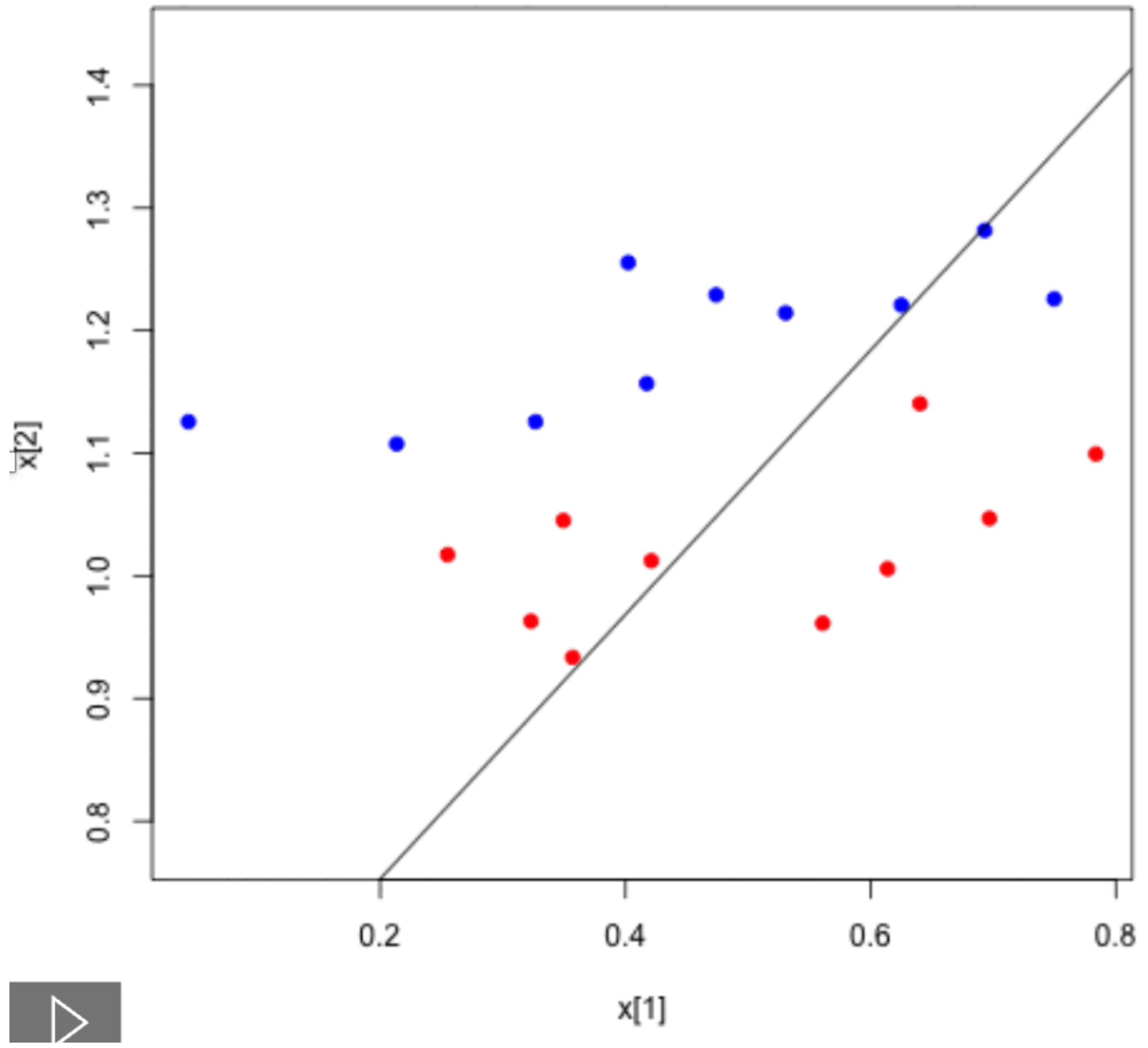
$\neq$



$$\sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i + b$$

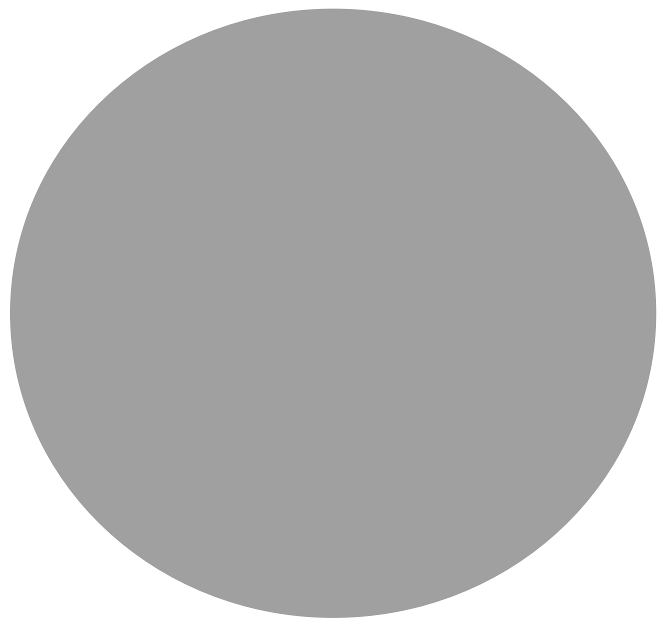
$$\sum_{i=0}^{d} \mathbf{w}_i \mathbf{x}_i \qquad \mathbf{x}_0 := 1$$

The Perceptron (Rosenblatt, 1958)



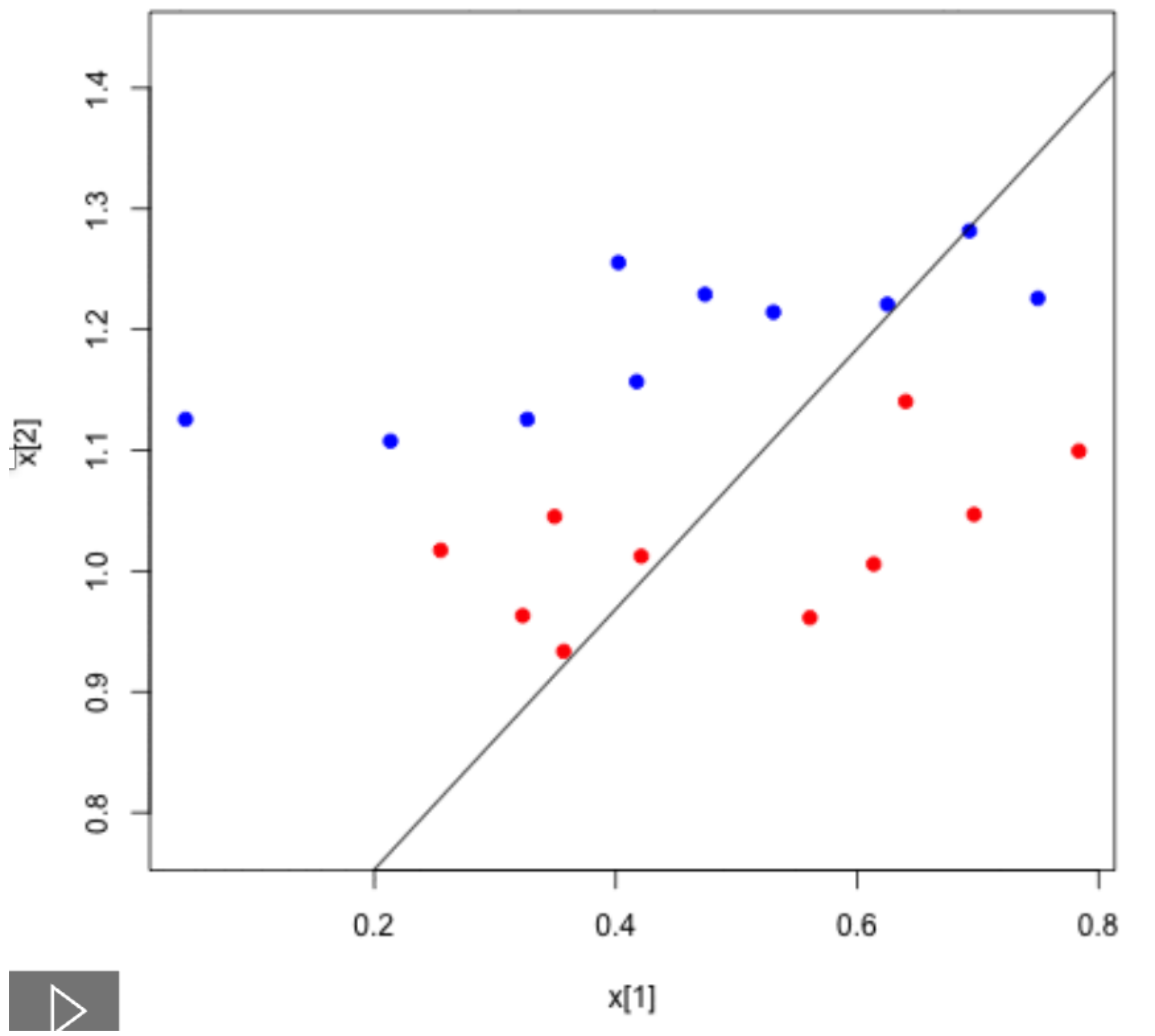
Not biologically plausible, but still very useful

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

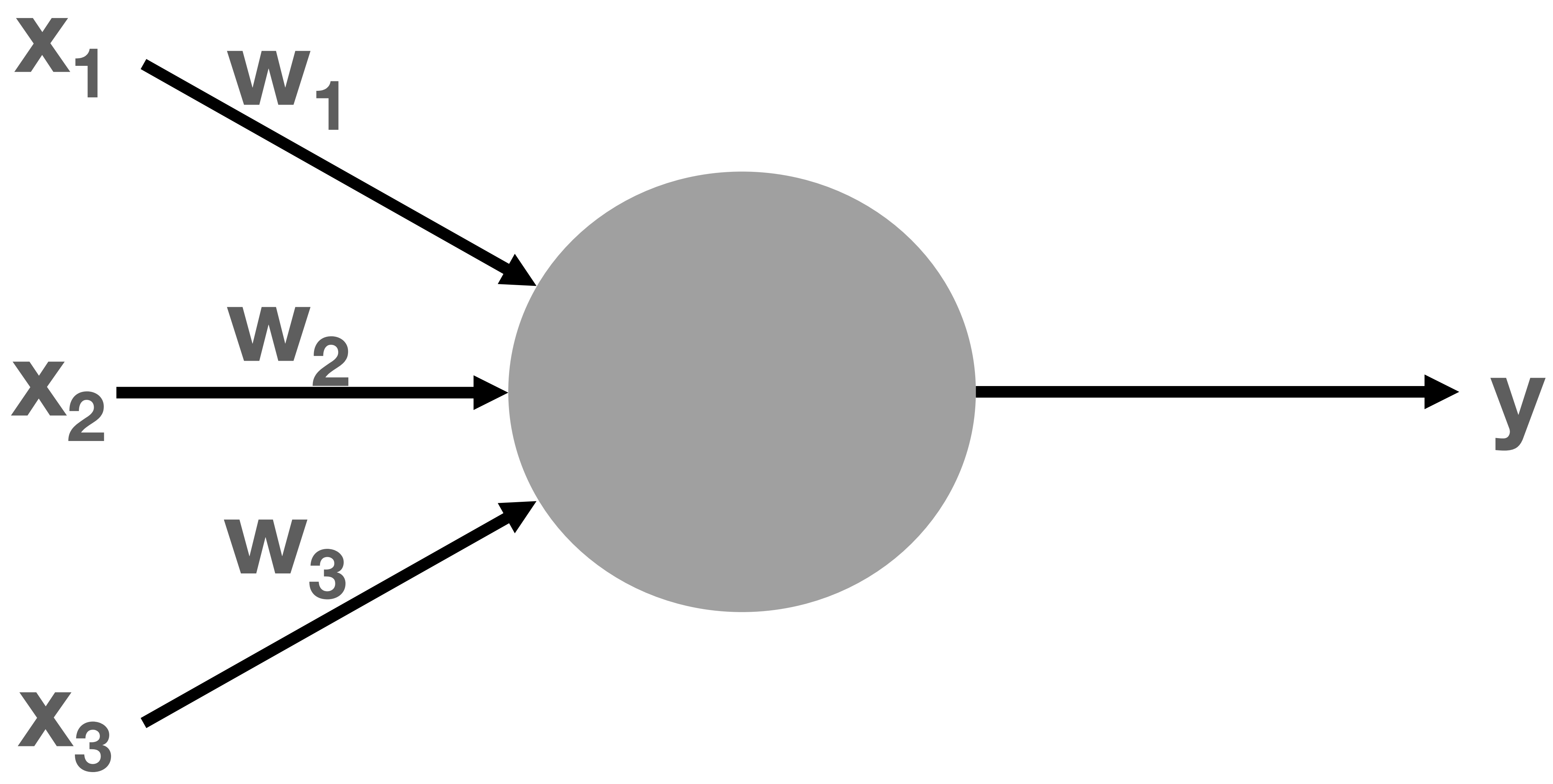


The Perceptron (Rosenblatt, 1958)

Not biologically plausible, but still very useful

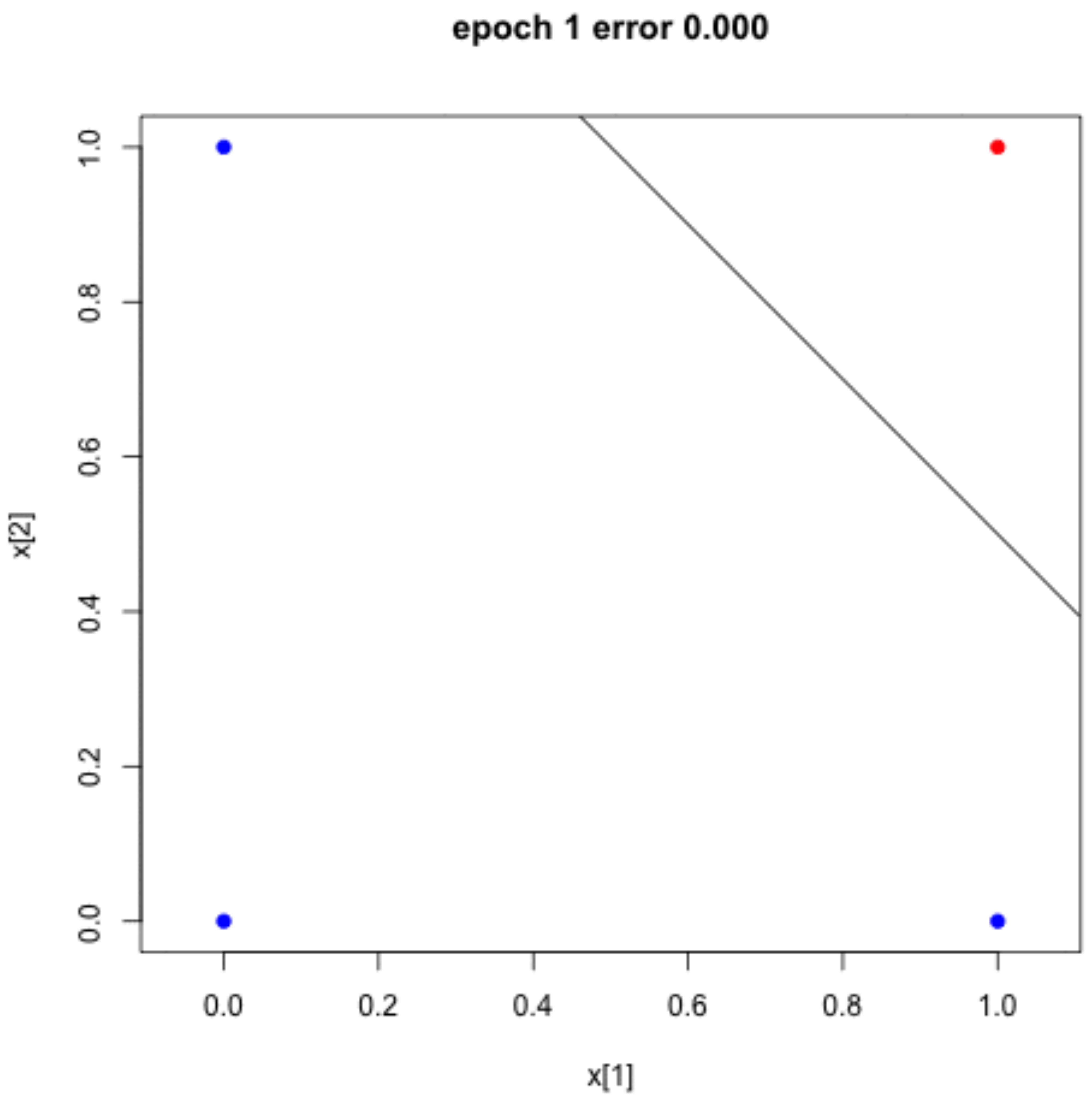


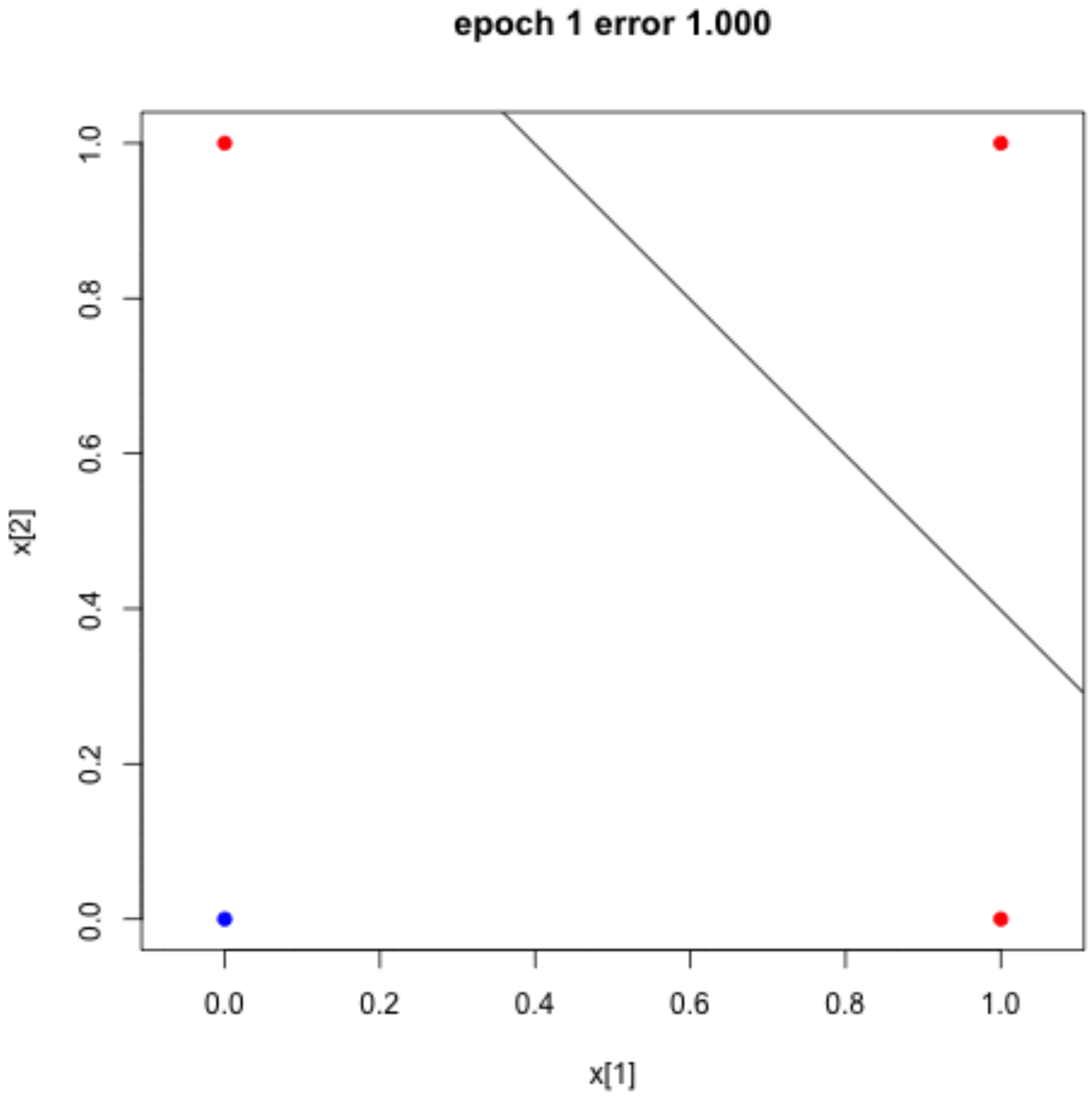
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$



The perceptron can solve AND and OR

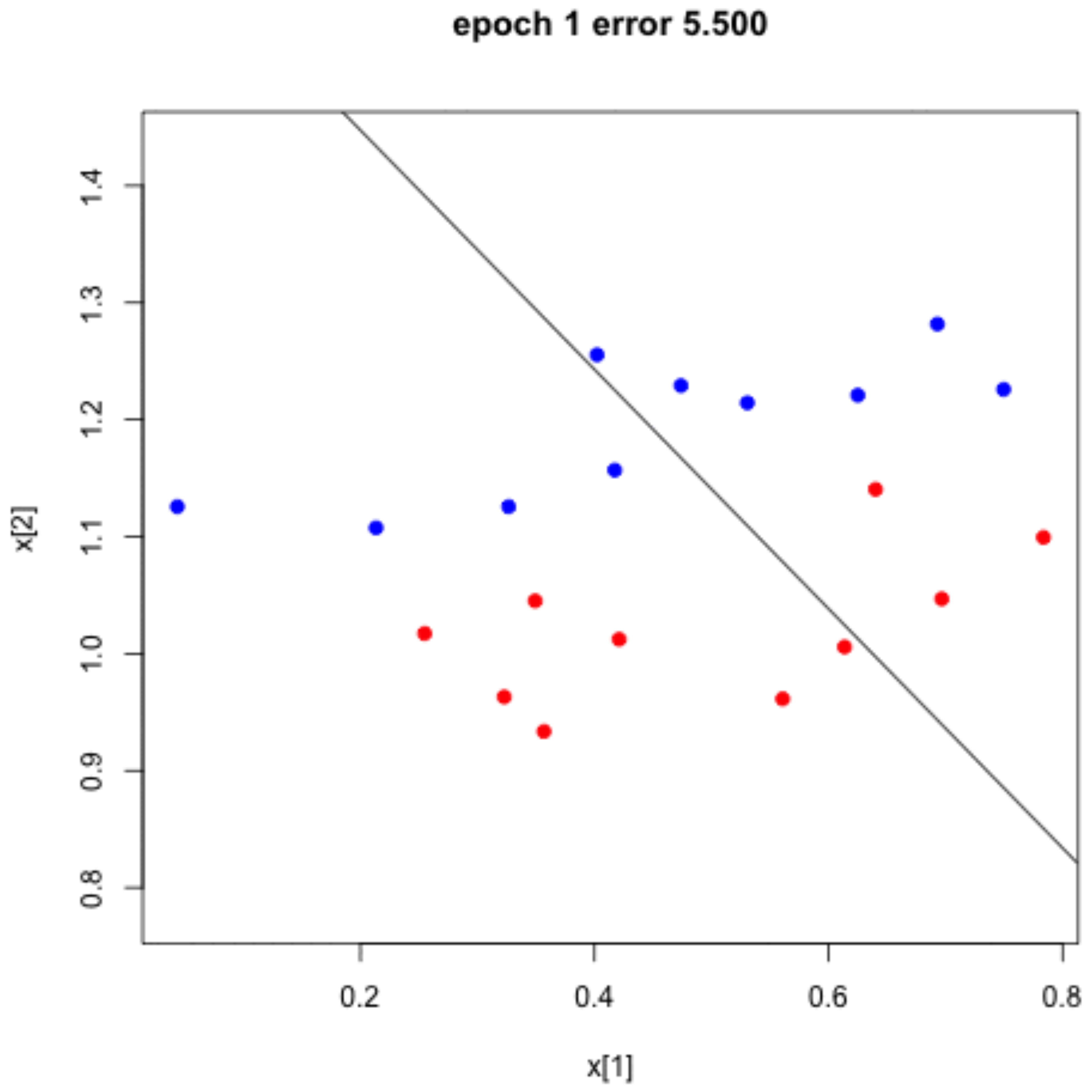
Linear problems can be solved





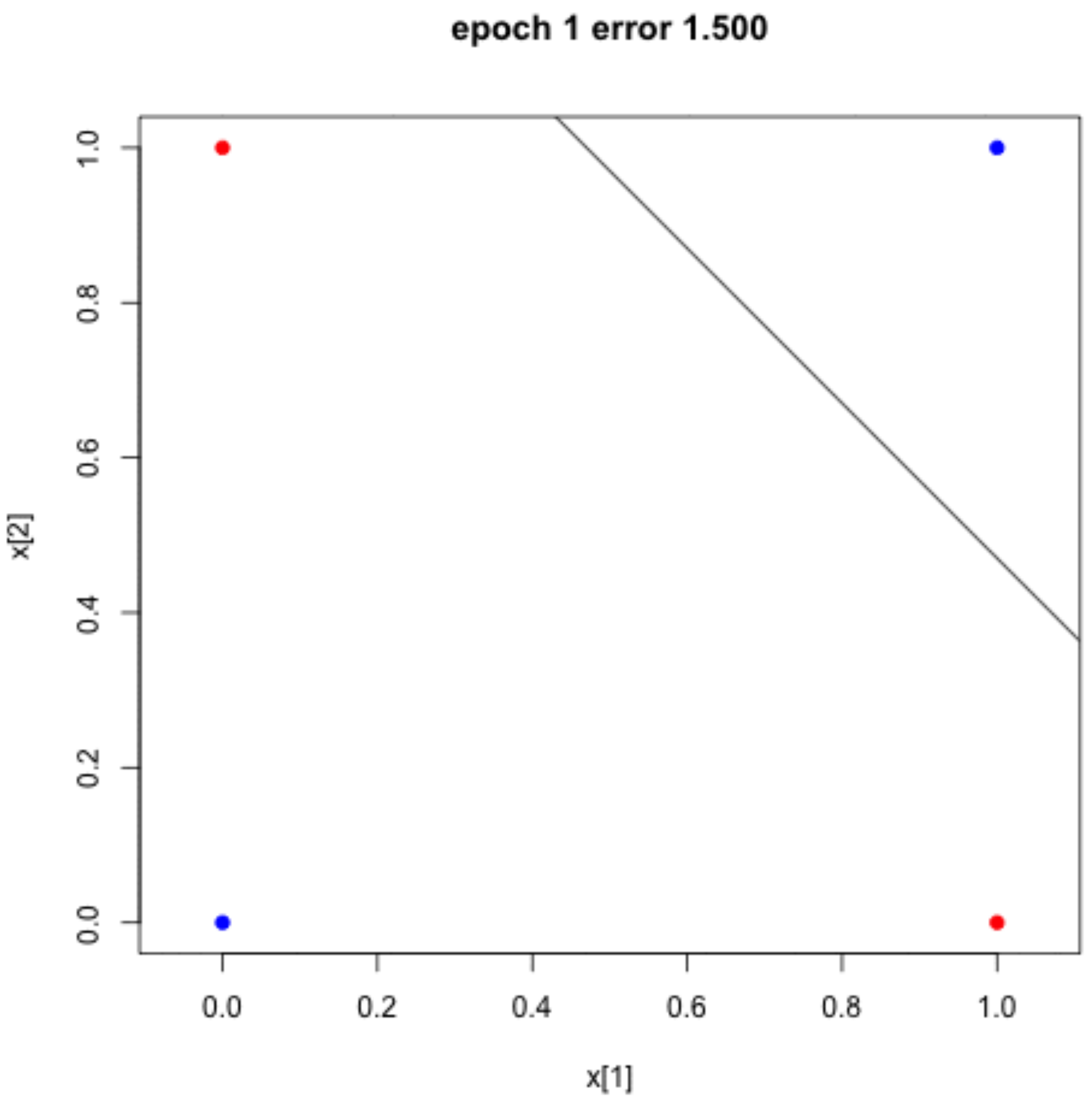
Perceptron on linearly separable data

Linear problems can be solved



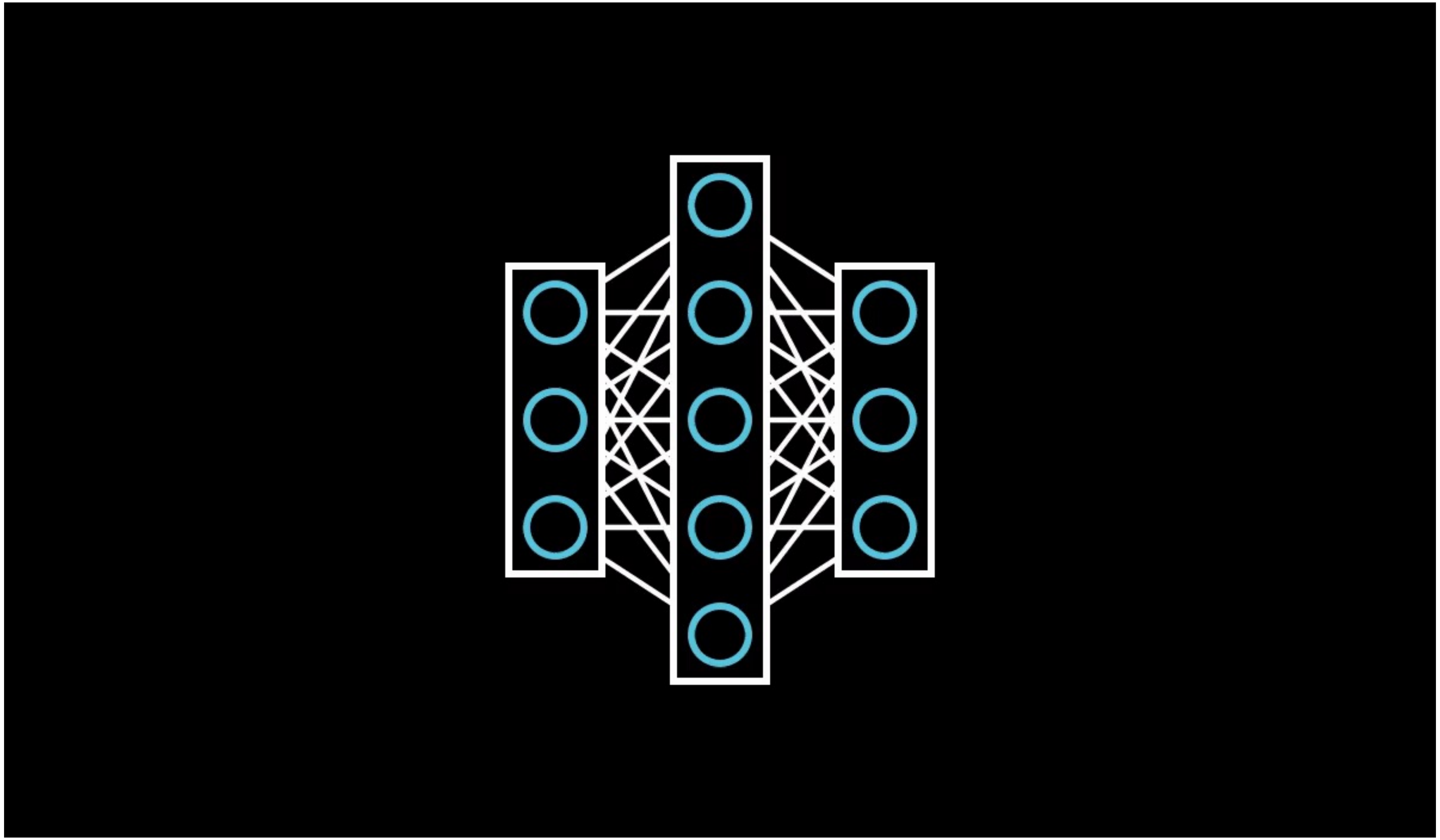
The perceptron cannot solve XOR

Minsky & Papert, 1969: Dawn of the first AI winter



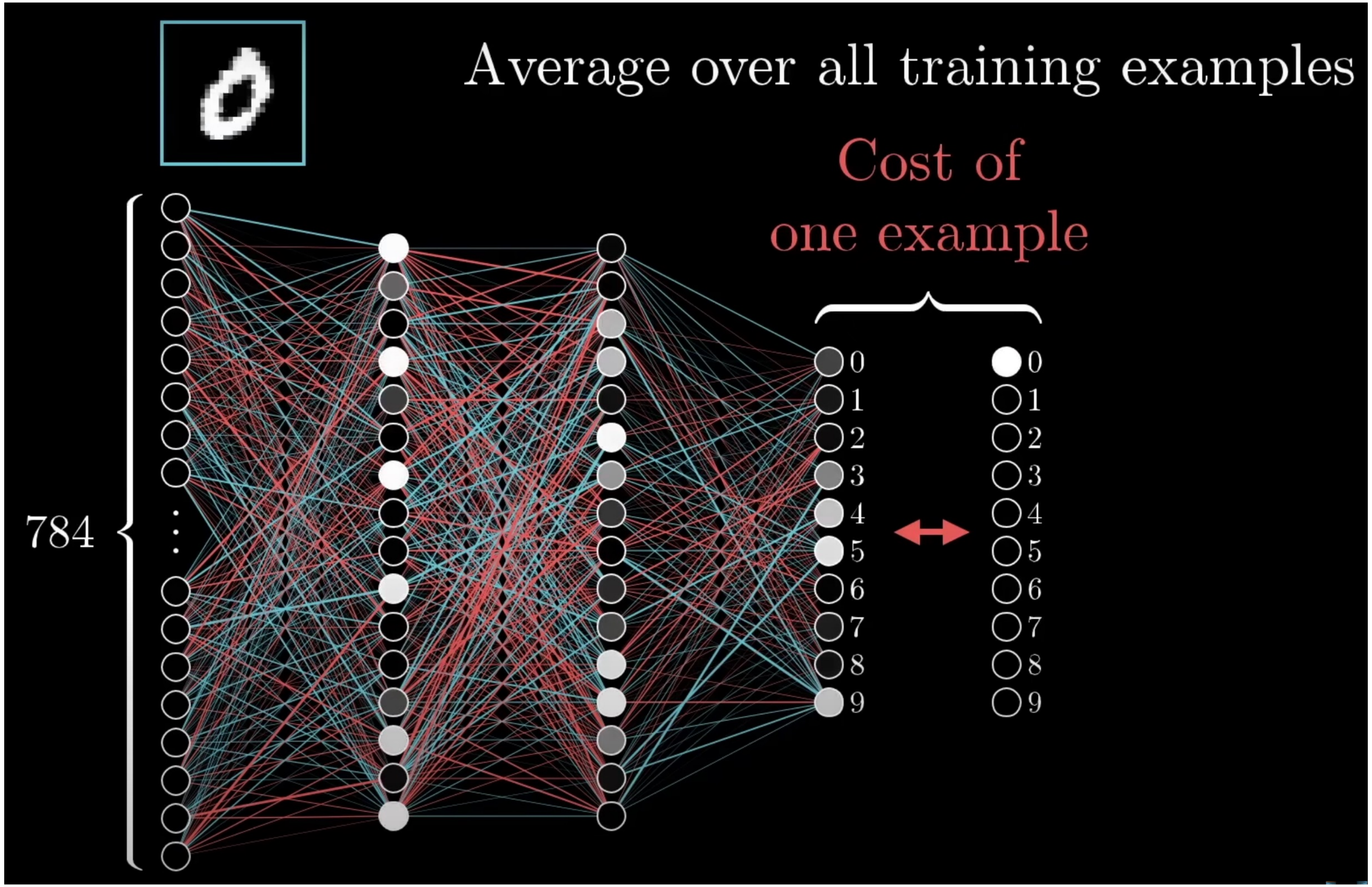
The secret sauce (1/2): Multilayer Perceptrons

Backpropagation allowed the use of deeper networks



The secret sauce (1/2): Multilayer Perceptrons

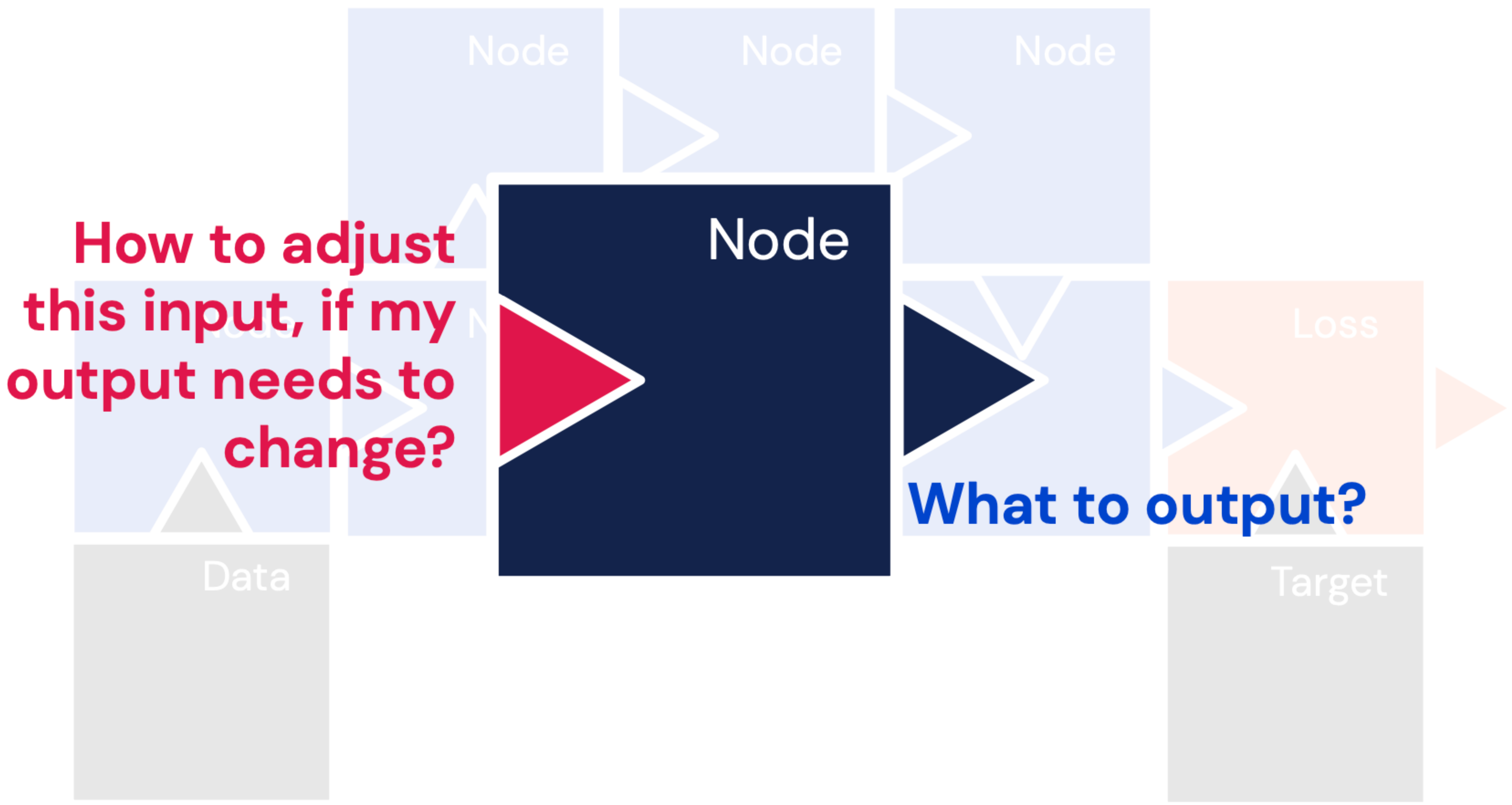
Backpropagation allowed the use of deeper networks



YouTube, 3B1B

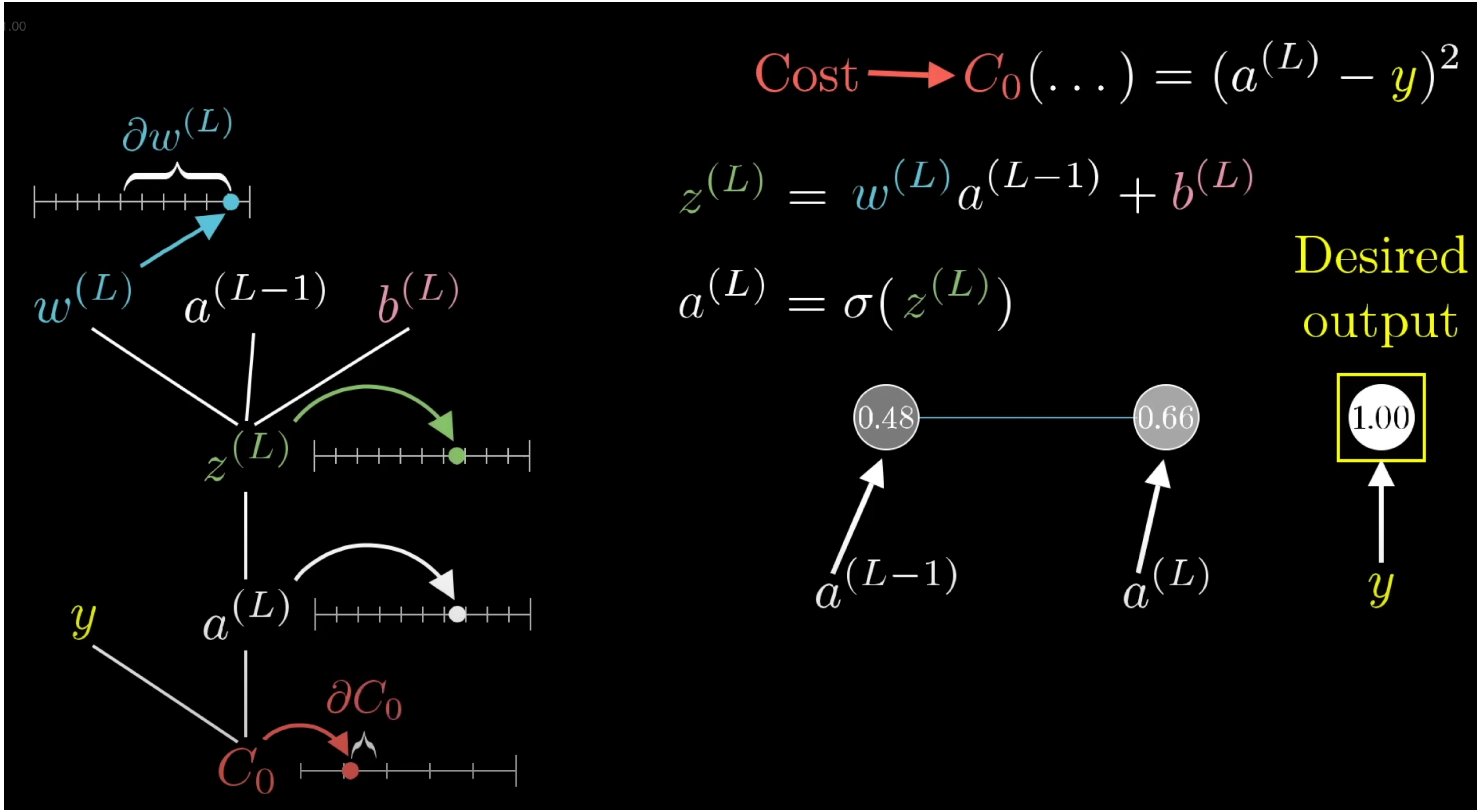
How do we tune hidden neurons

Backpropagation allowed the use of deeper networks



The secret sauce (1/2): Multilayer Perceptrons

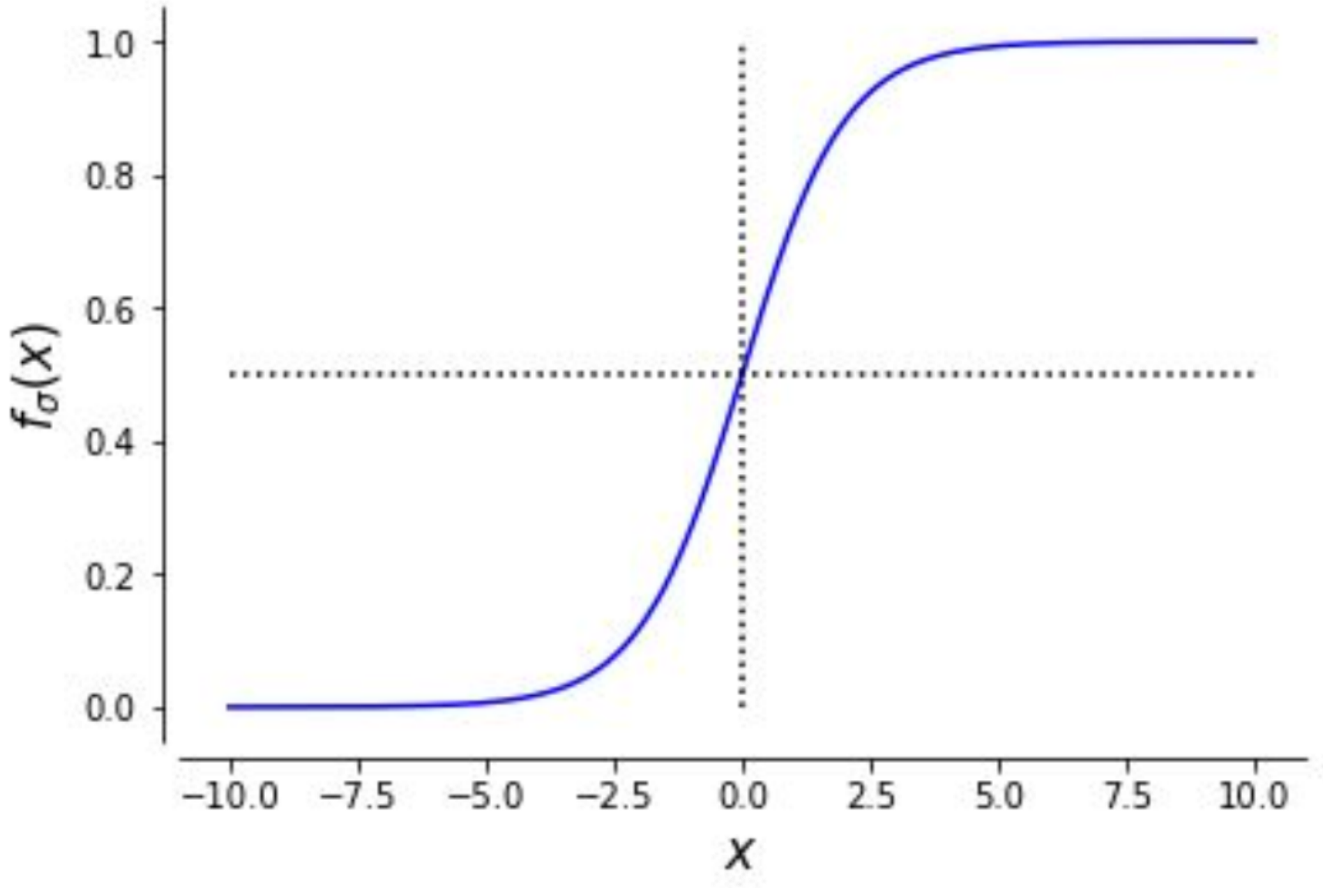
Backpropagation allowed the use of deeper networks



YouTube, 3B1B

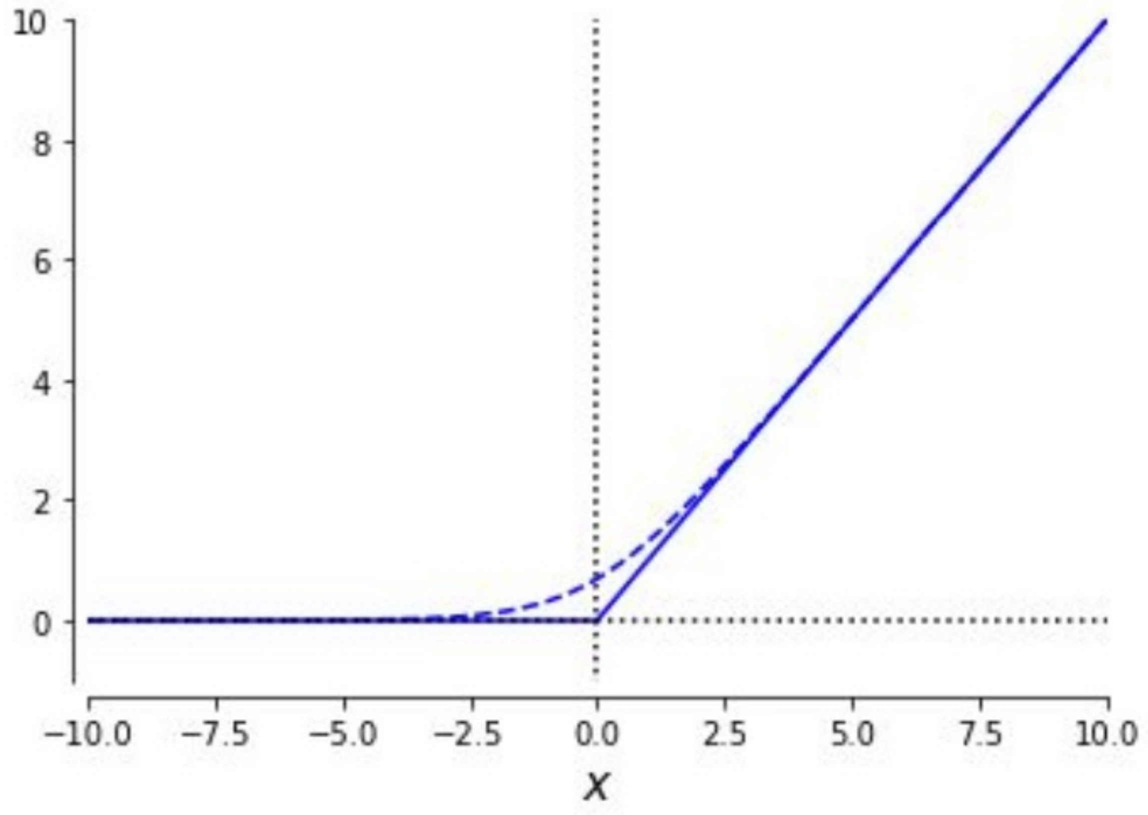
The secret sauce (2/2): Activation functions

Non-linearities allow us to solve non-linear problems



$$f_\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}}$$

$$f_\sigma(\mathbf{x}) = \frac{e^{\mathbf{x}}}{e^{\mathbf{x}} + 1}$$



$$f_{\text{relu}}(\mathbf{x}) = \max(0, \mathbf{x})$$

$$f_{\text{sp}}(\mathbf{x}) = \log(1 + e^{\mathbf{x}})$$

Activation functions are often called **non-linearities**.
Activation functions are applied **point-wise**.

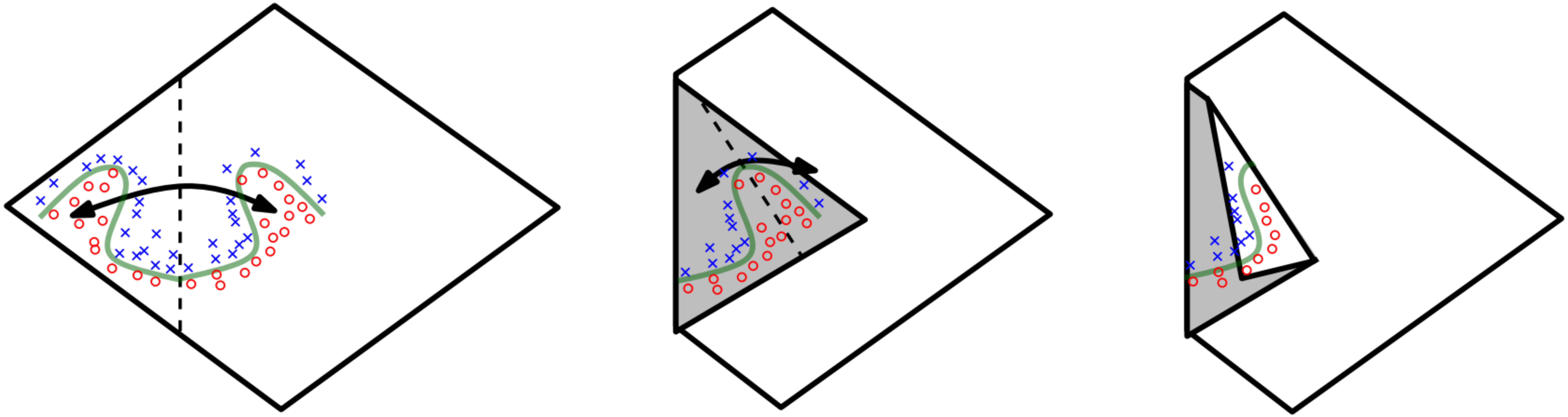
One of the **most commonly used activation functions.**
Made **math analysis** of networks **much simpler.**

Solve XOR with just 2 hidden neurons

- Hidden layers bend and deform input space
- Last linear model does linear classification
- Non-linear transformations are key for deep learning!
- Our network became a feature extractor!

What is deep lerning doing?

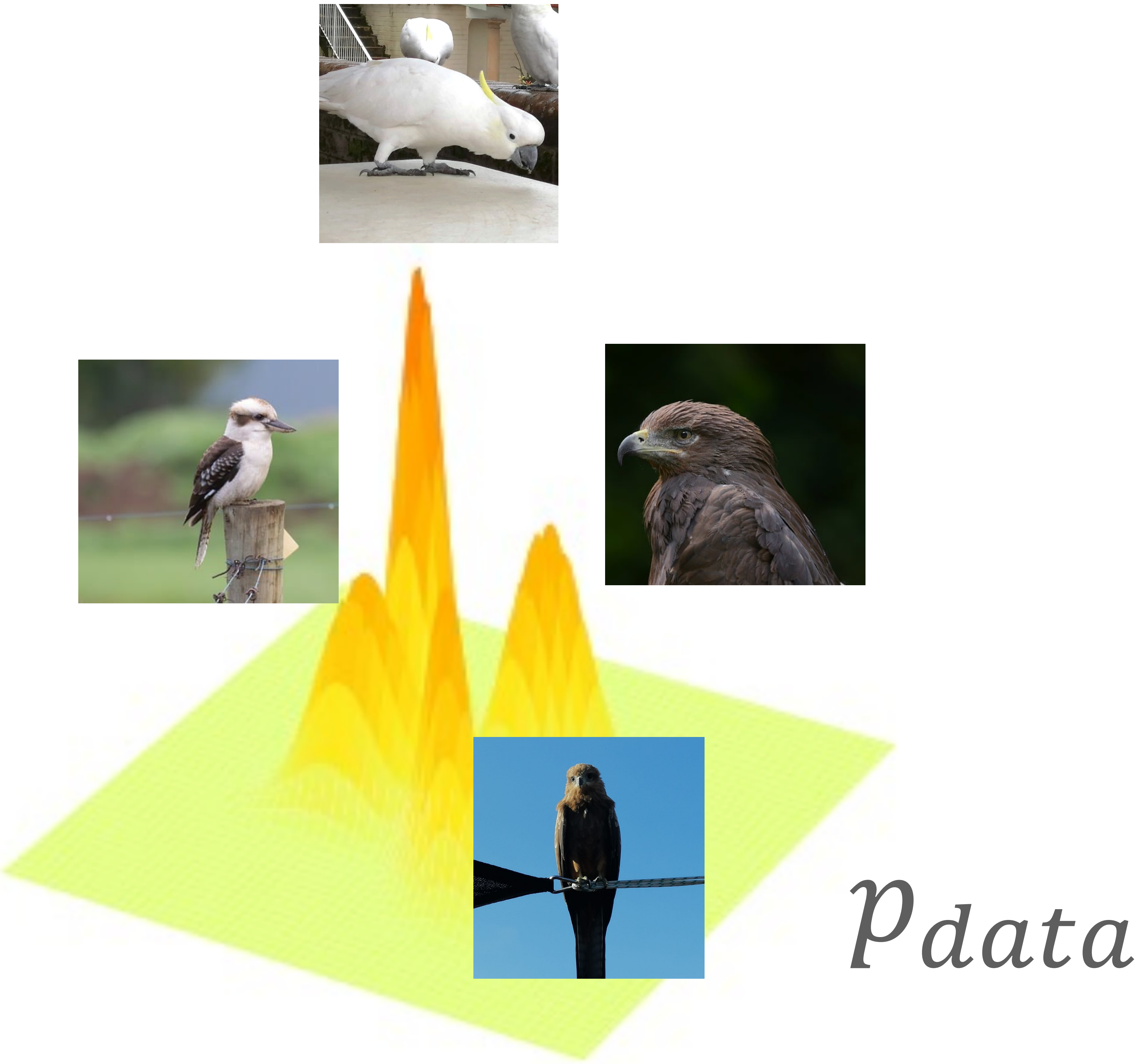
Transforming data non-linearly into a better representation



Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, Yoshua Bengio. **On the Number of Linear Regions of Deep Neural Networks** Arxiv (2014)

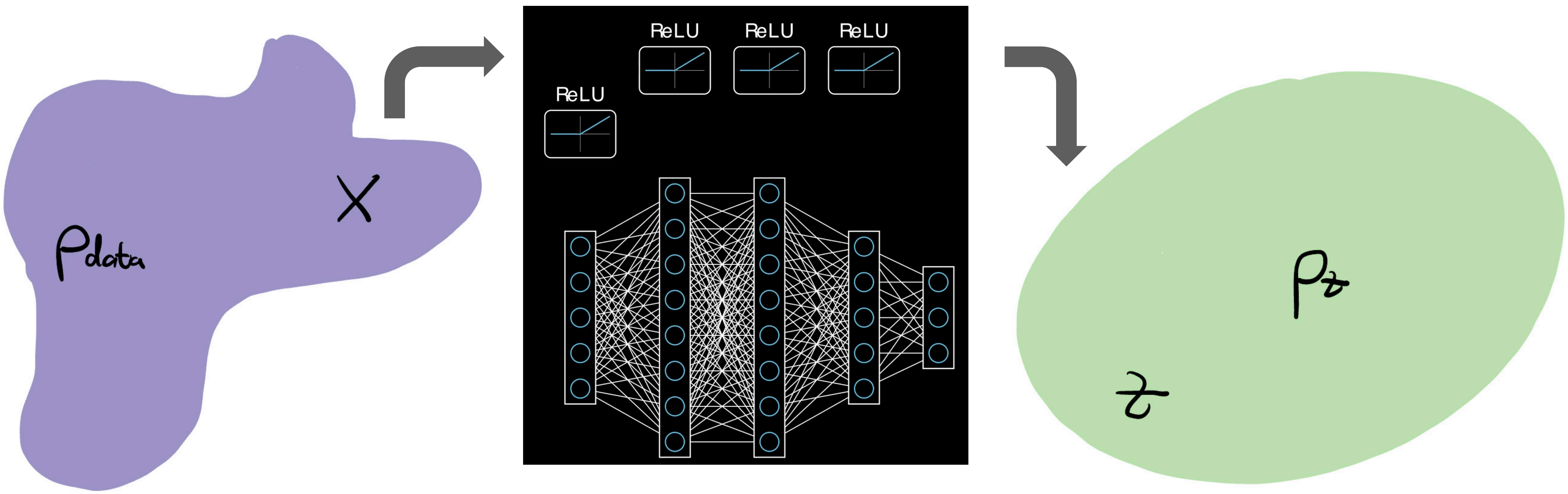
What is deep lerning doing?

Transforming data non-linearly into a better representation



What is deep lerning doing?

Transforming data non-linearly into a better representation



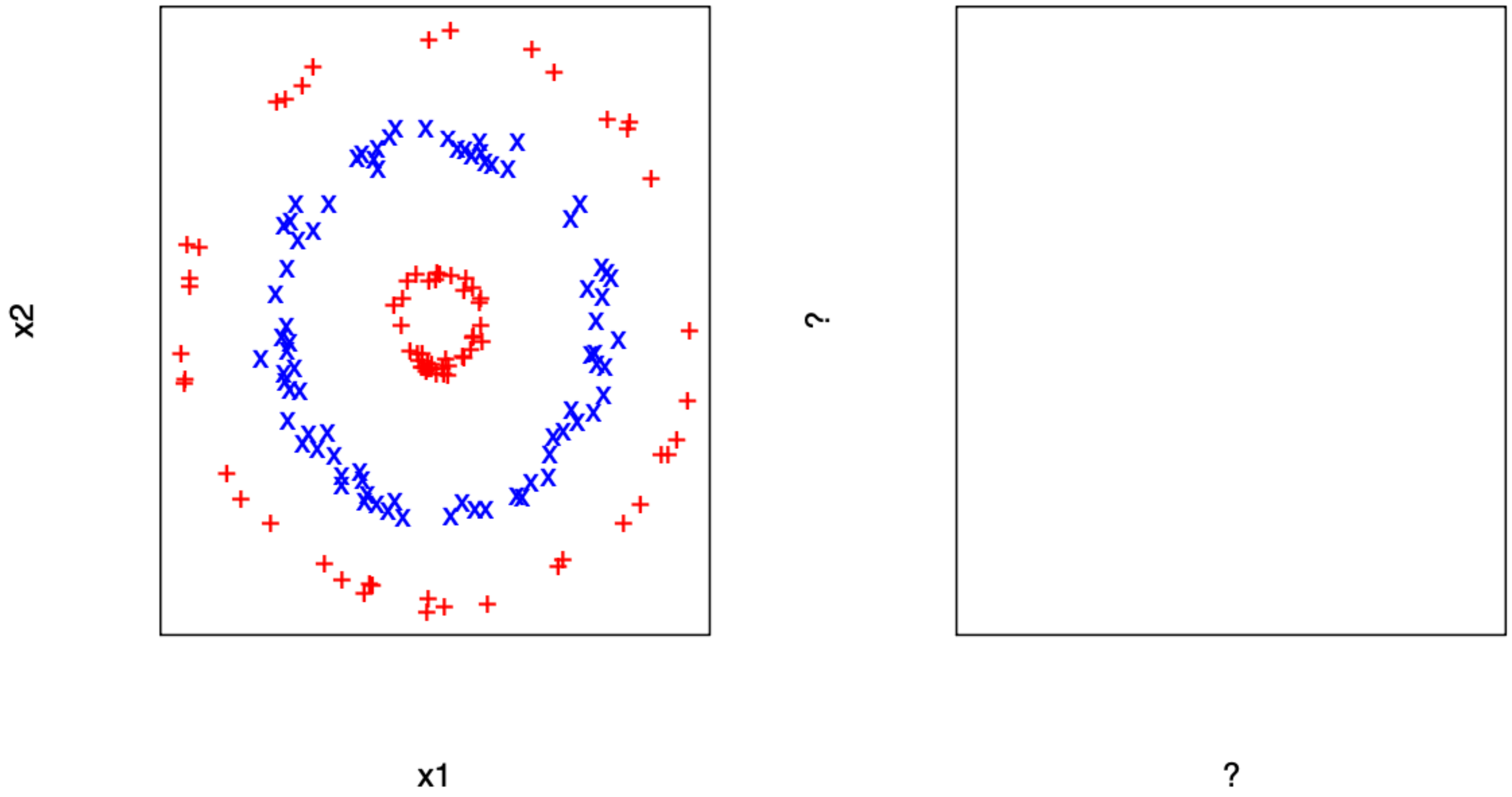
Let’s try it ourselves

See the power of hidden layers

http://playground.tensorflow.org/

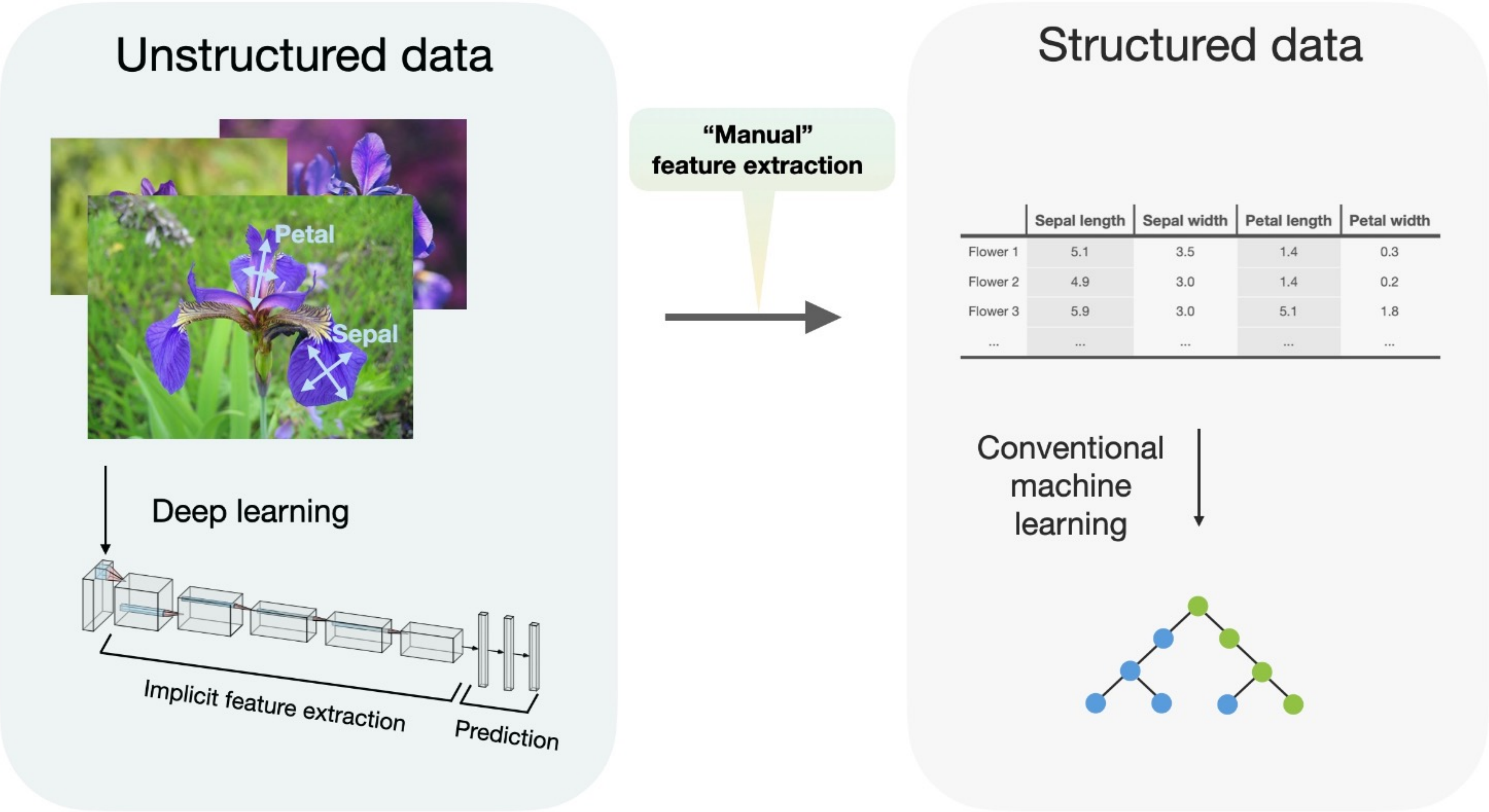
Feature/Representation engineering is hard

The Deep Learning way: Let the network figure it out for you



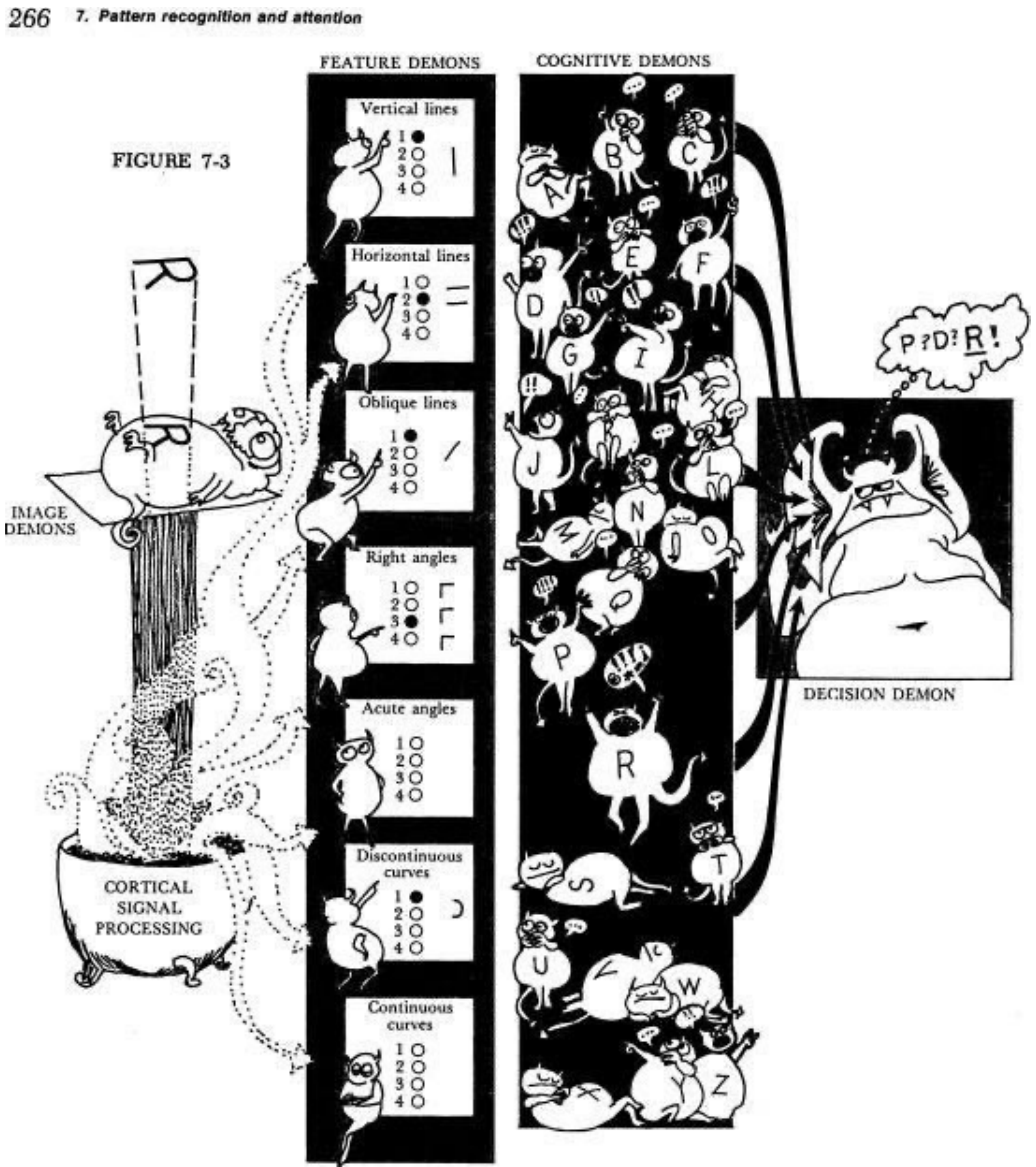
Deep Learning performs feature extraction

The deeper the network, the more complex the patterns can be



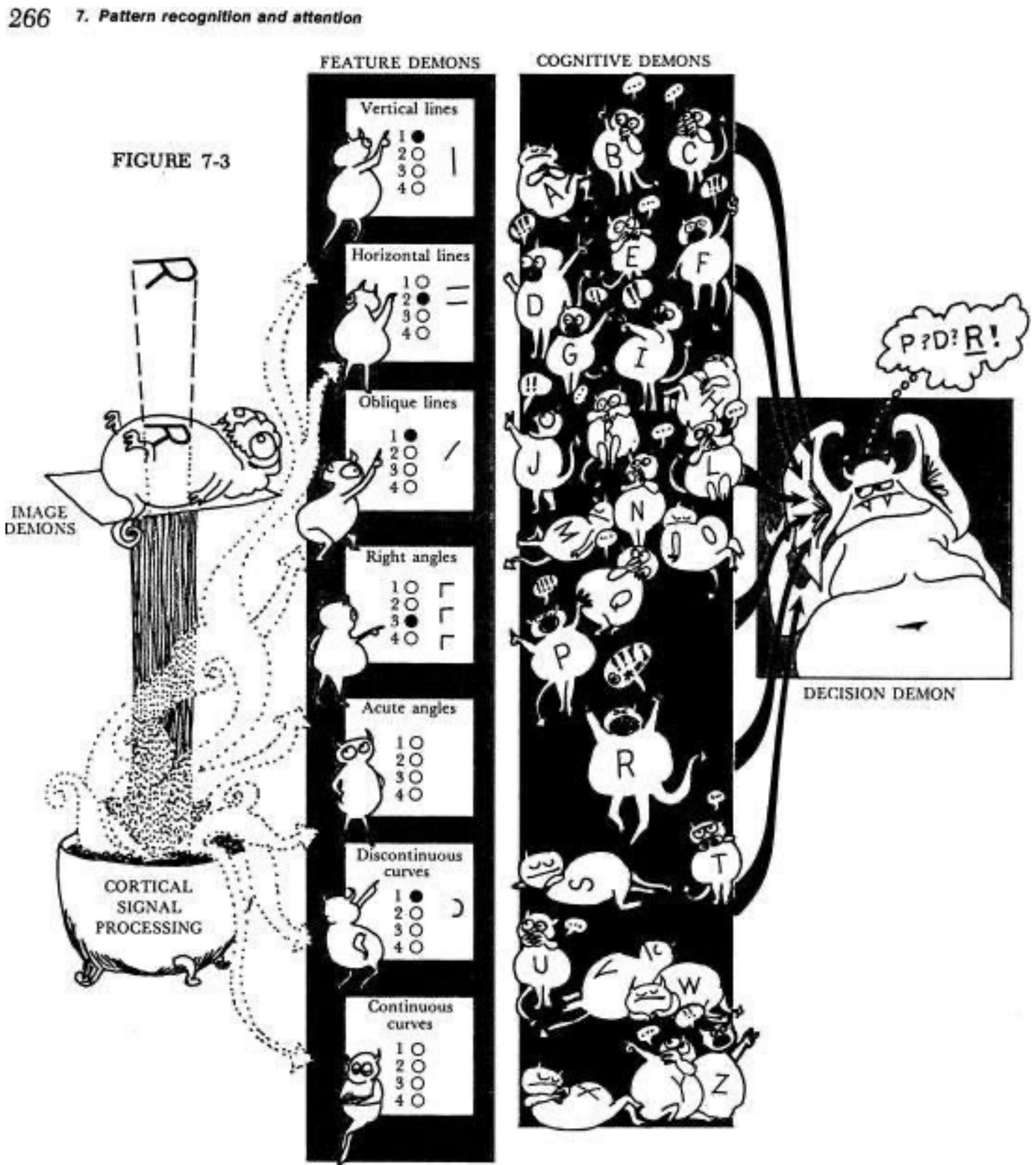
The demons in the pandemonium

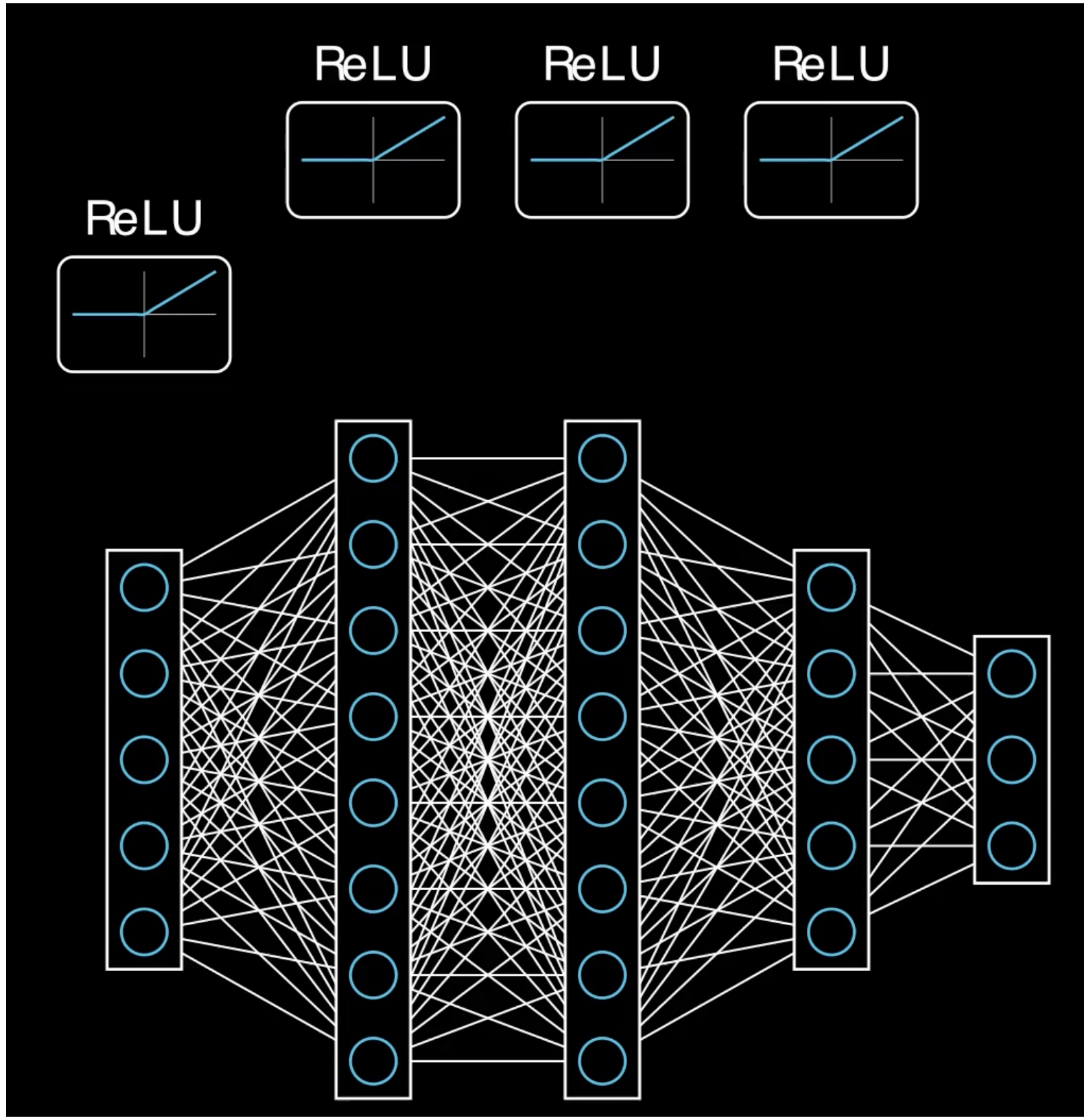
Selfridge, 1959: Pattern recognition triggers downstream cognition



The demons in the pandemonium

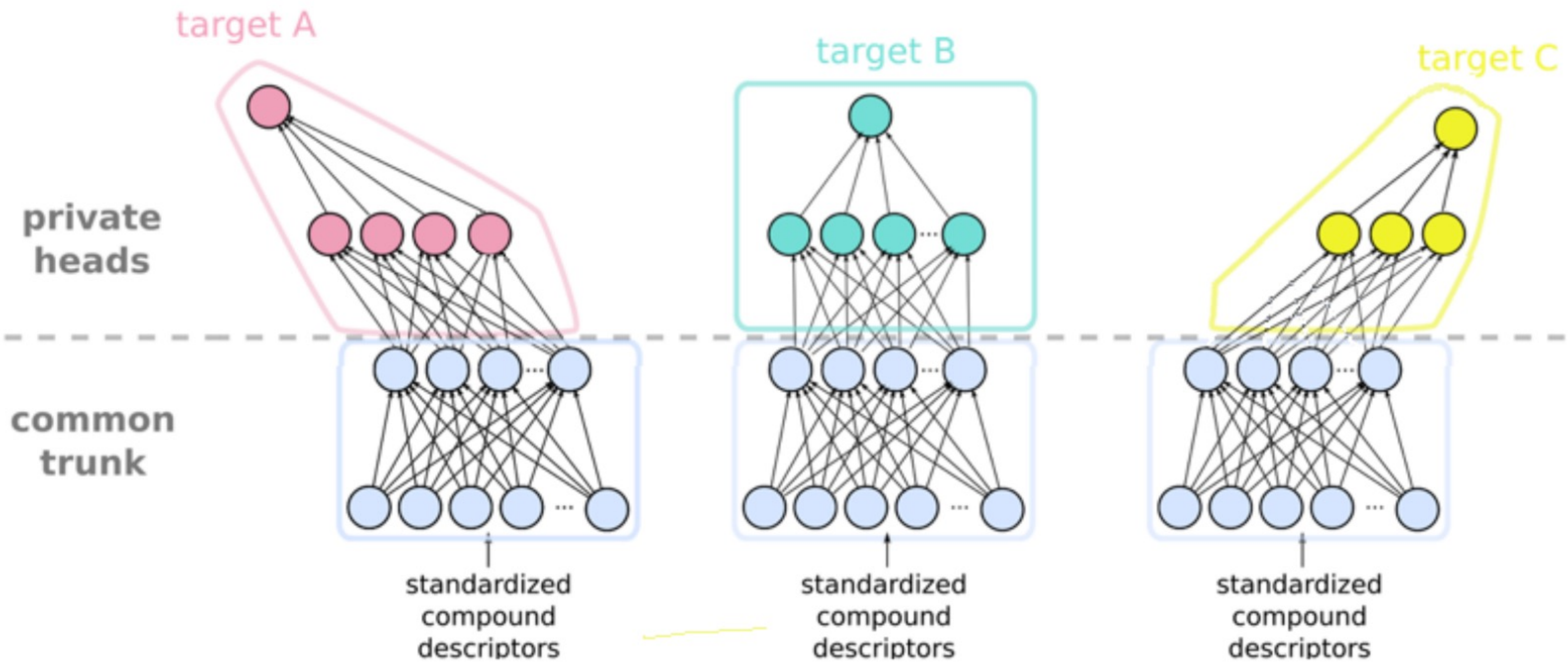
Trunk versus head





Deep Lerning as feature extractors

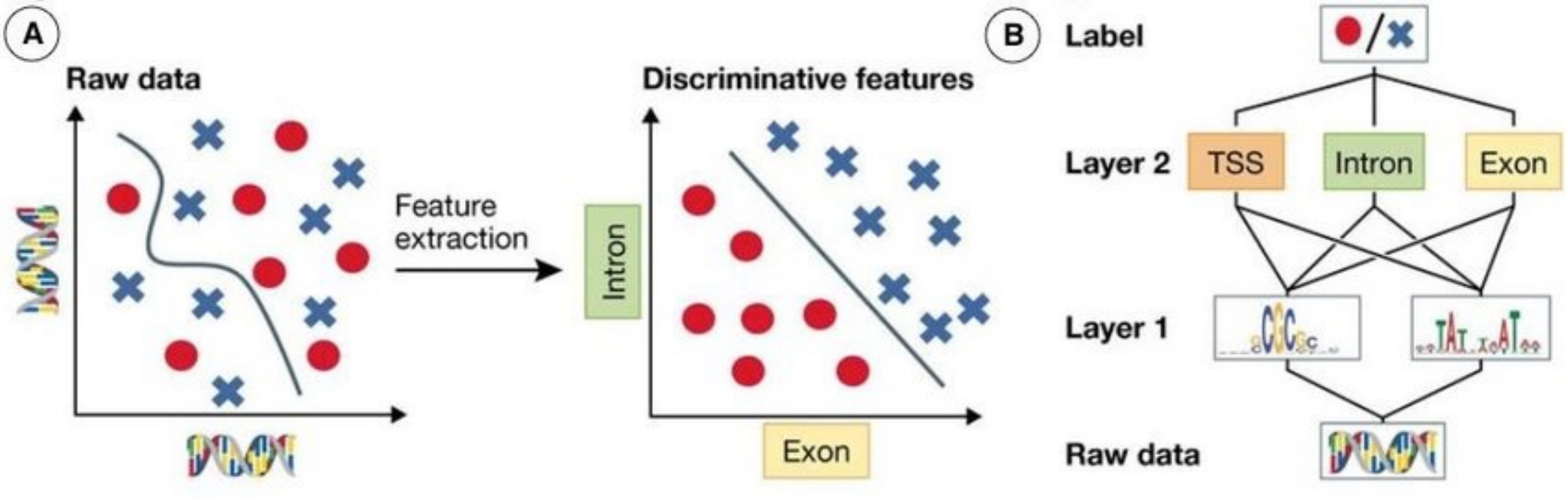
Trunk versus head



Pejo, Balazs, et al. "Collaborative Drug Discovery: Inference-level Data Protection Perspective." *arXiv preprint arXiv:2205.06506* (2022).

Deep Lerning as feature extractors

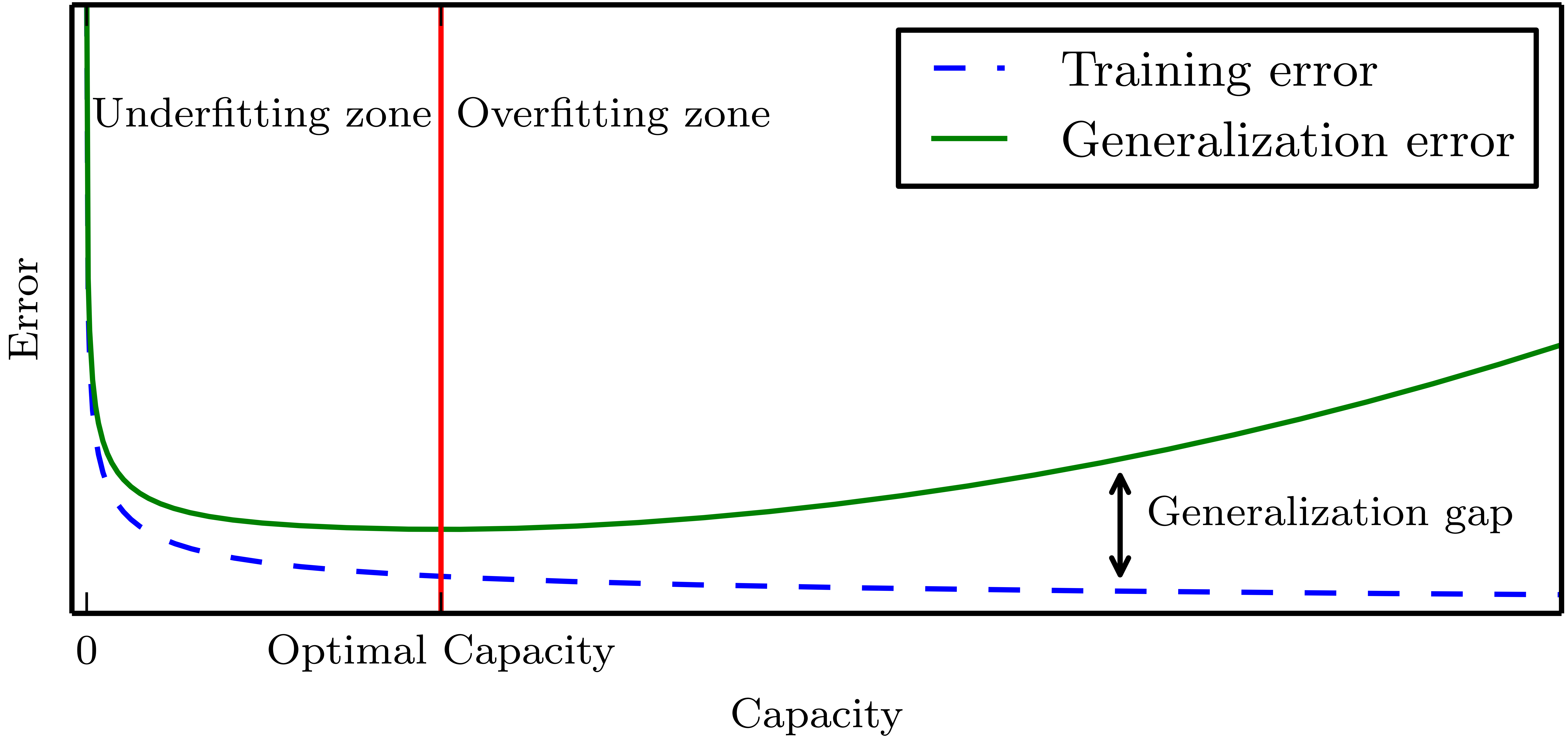
Trunk versus head



Adapted under CC BY 3.0 license from: Kalinin, Alexandr A., et al. "Deep learning in pharmacogenomics: from gene regulation to patient stratification." *Pharmacogenomics* 19.7 (2018): 629-650.

Regularisation: How do we avoid overfitting?

Machine learning models like to overfit your data



Regularisation: How do we avoid overfitting?

Machine learning models like to overfit your data

Explicit Regularisation

-> Add reg. term to loss function:

- $\sum_{w \in \theta} ||w||_1$
- $\sum_{w \in \theta} ||w||_2$
- $\sum_{w \in \theta} ||w||_\infty$
- $\sum_x p(x) \log(x)$

Implicit Regularisation

-> Various hacks to improve optimisation

- data augmentation
- dropout
- early stopping
- label smoothing
- Batch/Layer Normalisation

🥡 Takeaway 🥡

Neural networks perform **implicit feature extraction** and are optimized via **gradient descent.**

Outlook for Part II

1. Images: Convolutional Neural Networks
2. Sequences: RNNs + Transformers
3. Current developments