



Machine learning in chemistry

Berend Smit | @SmitBerend | berend.smit@epfl.ch
Kevin Jablonka | @kmjablonka | mail@kjablonka.com
Laboratory of Molecular Simulation (LSMO)

Organization

CH315 | Machine Learning



CH315 | Machine Learning

Learning objectives

In the three weeks we have for machine learning in this course we aim to

- give you a basic overview over the different "flavors" of ML (supervised, unsupervised, reinforcement)
- give you an idea how ML can be used in chemistry
- give you a basic understanding over the workflow of *supervised ML*
- familiarize you with the basic tooling to (**pandas** , **skLearn**) so that you can build models yourself
- provide you with the tools (e.g., reading documentation) and resources (e.g., Kaggle) to continue your learning journey

At the end of the course, you should be equipped to understand the general ideas of papers that use ML in chemistry.

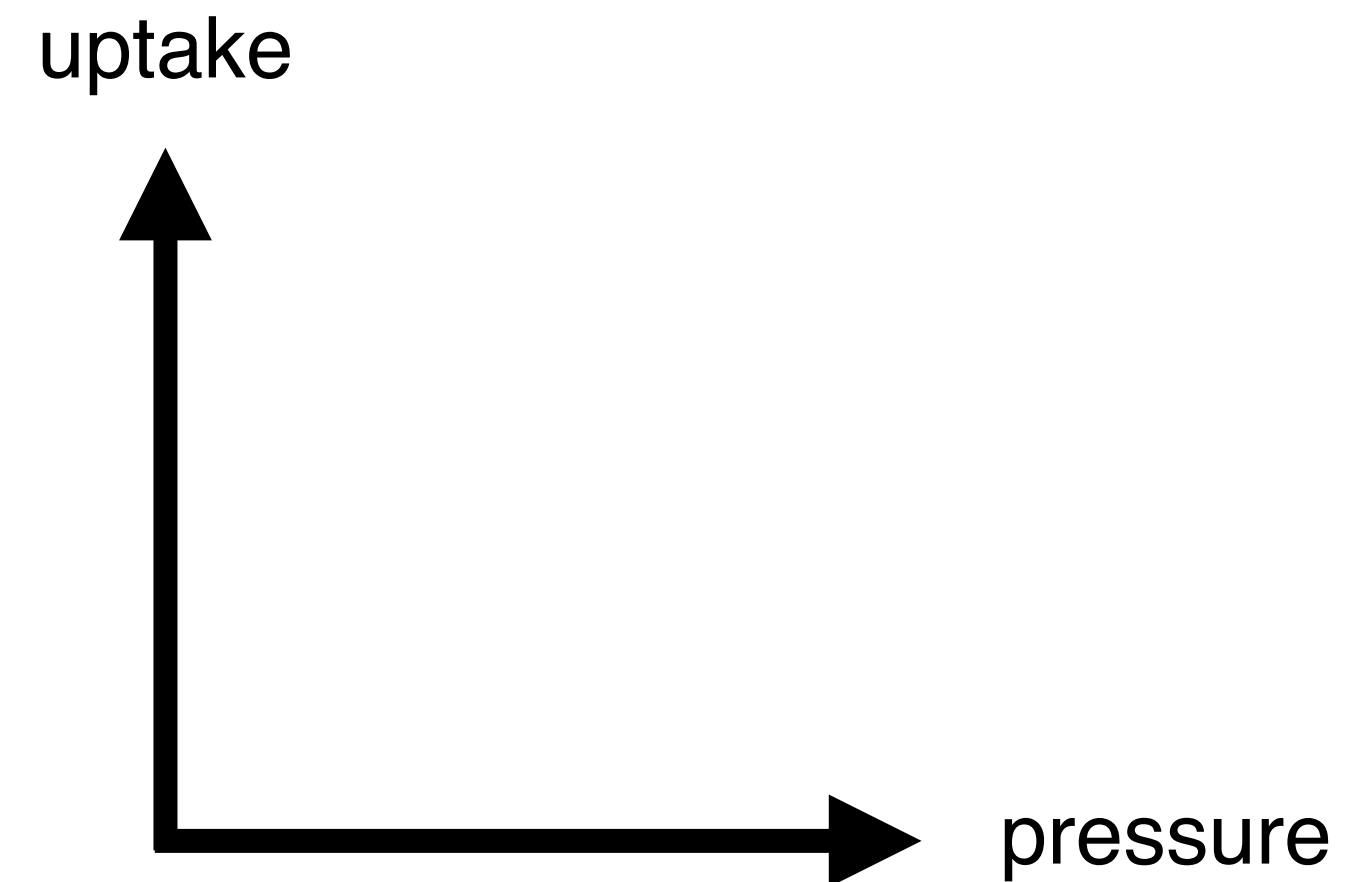
Organization

ML course organization

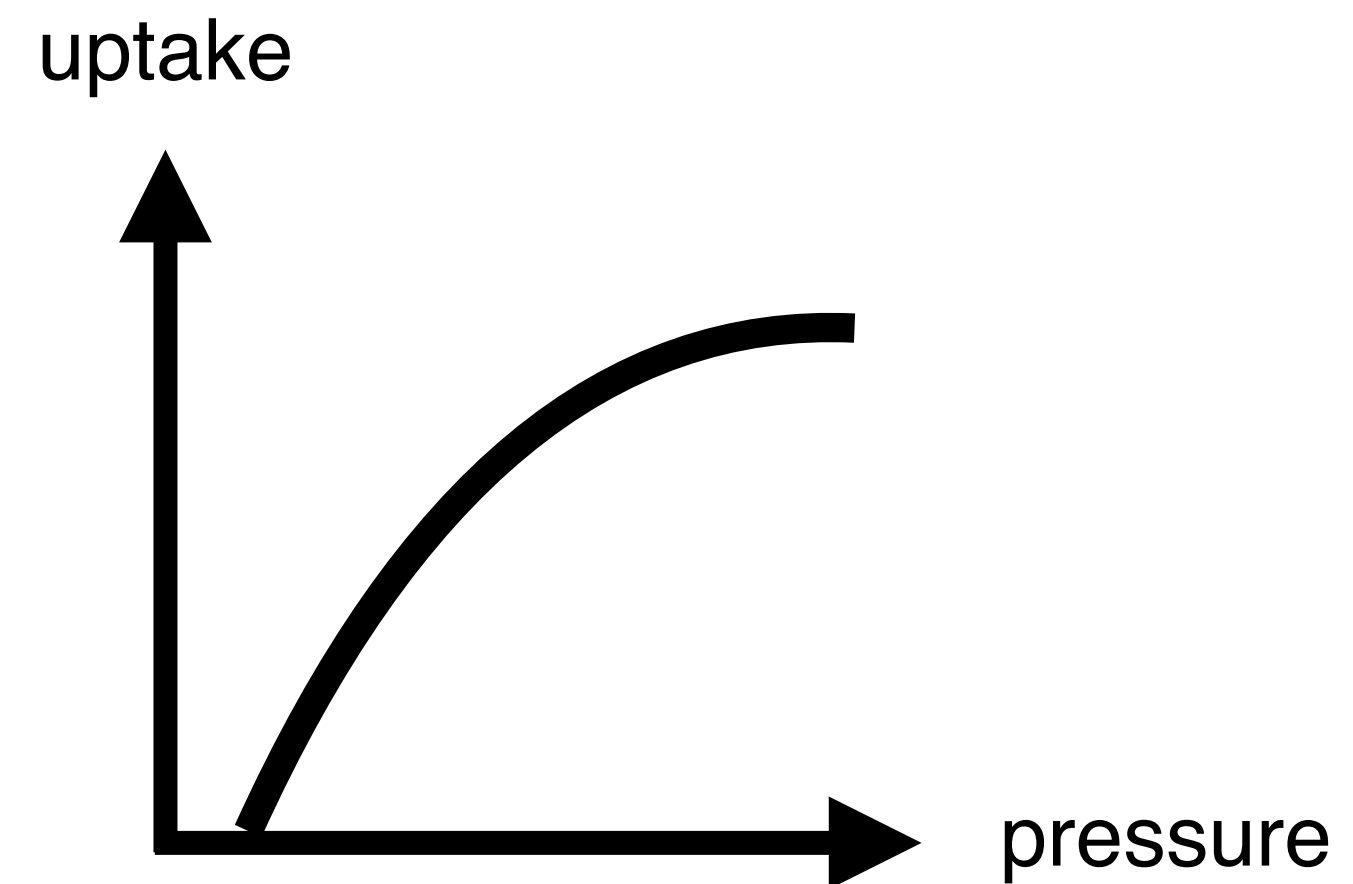
Date	Content	Room	Notes	Deliverables
December 10, 2021 10:15 AM-5:00 PM	Lecture, Quizzes, Theoretical Exercise	BCH1103	Kevin won't be available approx. 13:45 - 15:15	- Quizzes on Moodle filled until 15.12.2021 23:59 - PDF with solution of theoretical exercises uploaded on Moodle until 15.12.2021 23:59
December 17, 2021 10:15 AM-5:00 PM	Hands-on exercise and project	BCH1103		to be submitted with the project (see below)
December 24, 2021	Virtual Q&A for the project	Zoom	Time and Zoom link TBA!	- Notebook of hands-on and project uploaded as PDF until 06.01.2021 23:59 - Submission to Kaggle challenge until 06.01.2021

- Deliverables for this session:
 - Quizzes, theoretical exercise
- Deliverables for new year:
 - Hands-on/project notebook and submission to Kaggle challenge

Recall: Testing a MOF for carbon capture

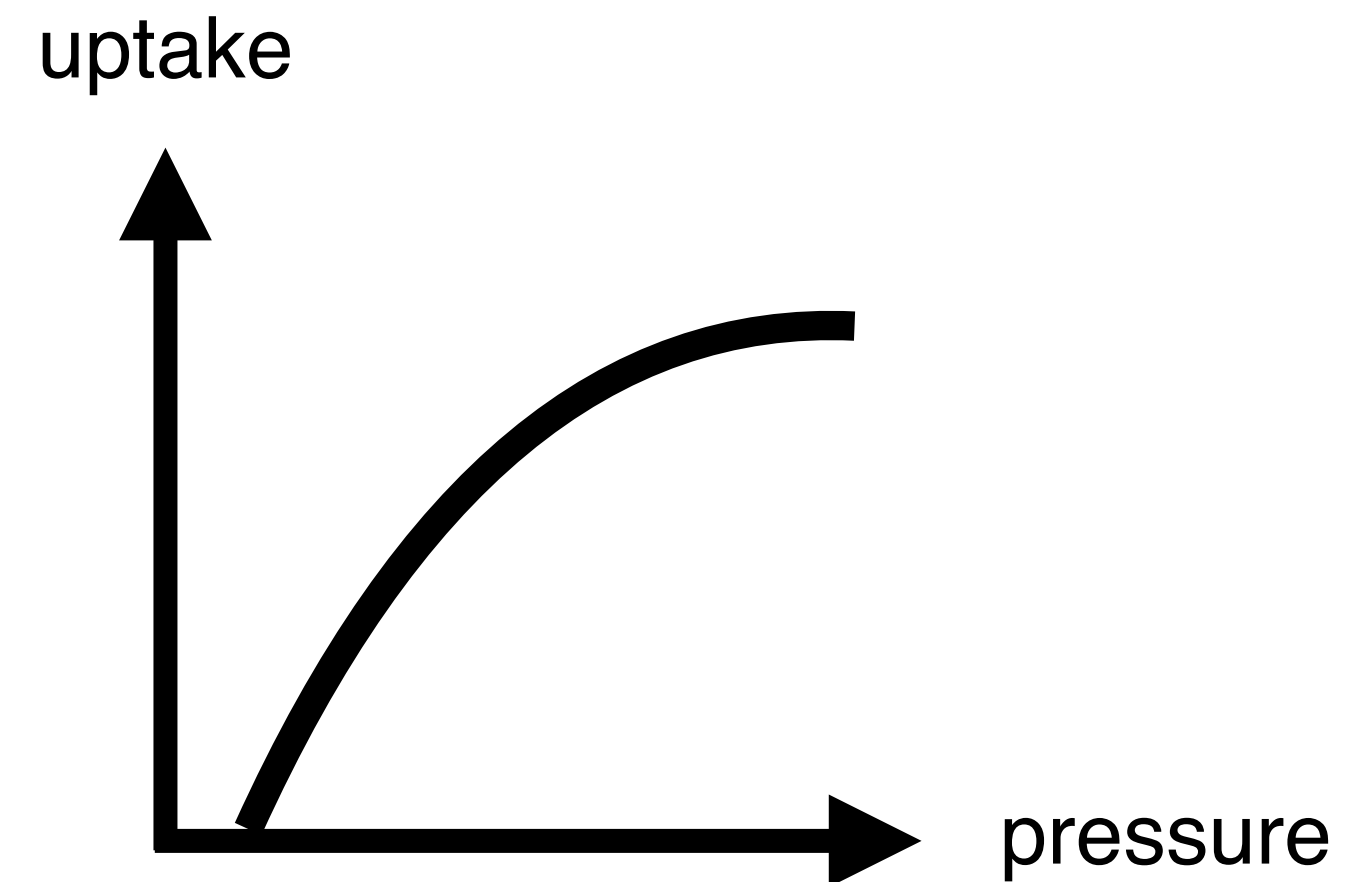
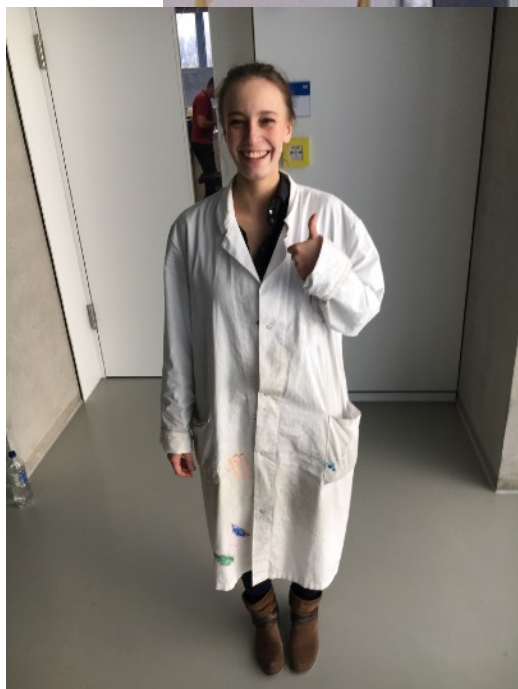
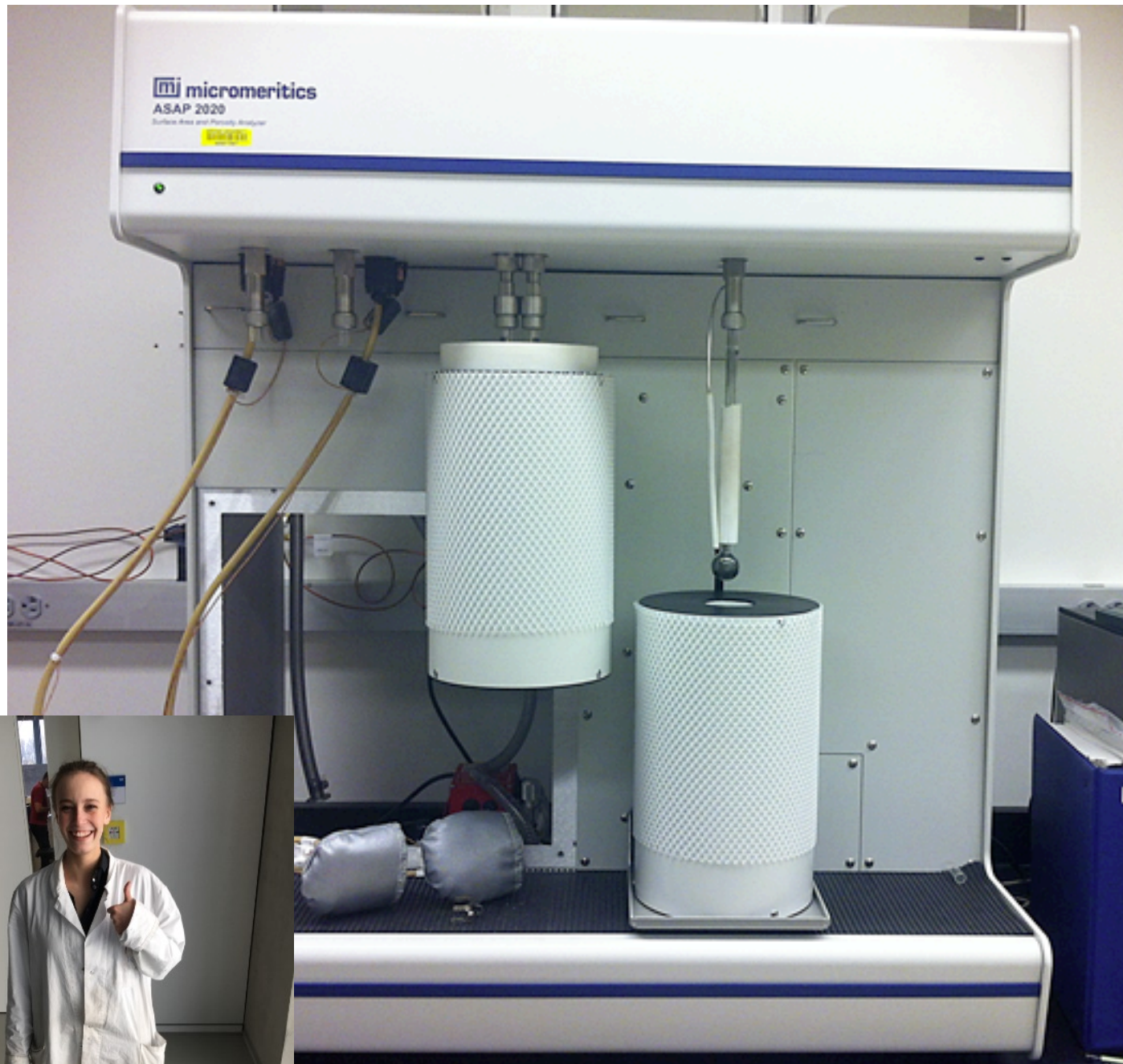


Recall: Testing a MOF for carbon capture



Recall: Testing a MOF for carbon capture

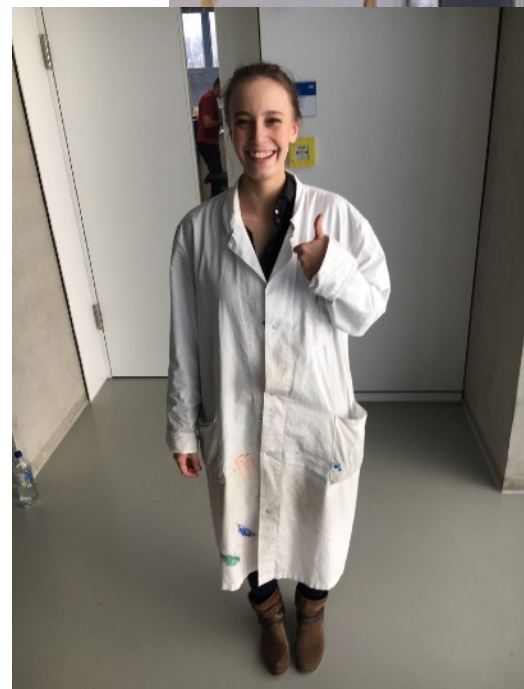
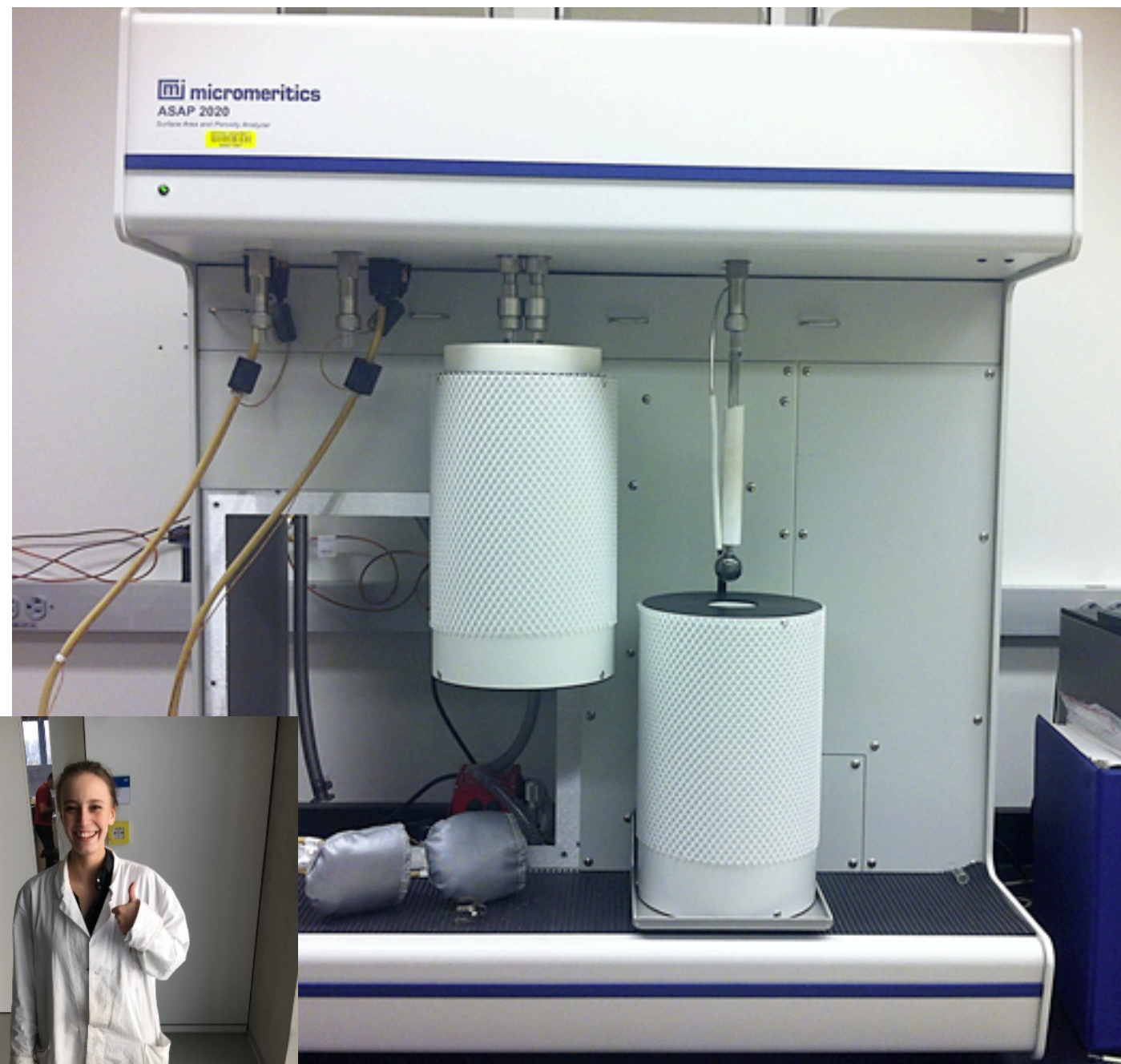
Experiment: Weeks/Months



Molecular Simulation **2013**, 39 (14–15), 1253–1292.

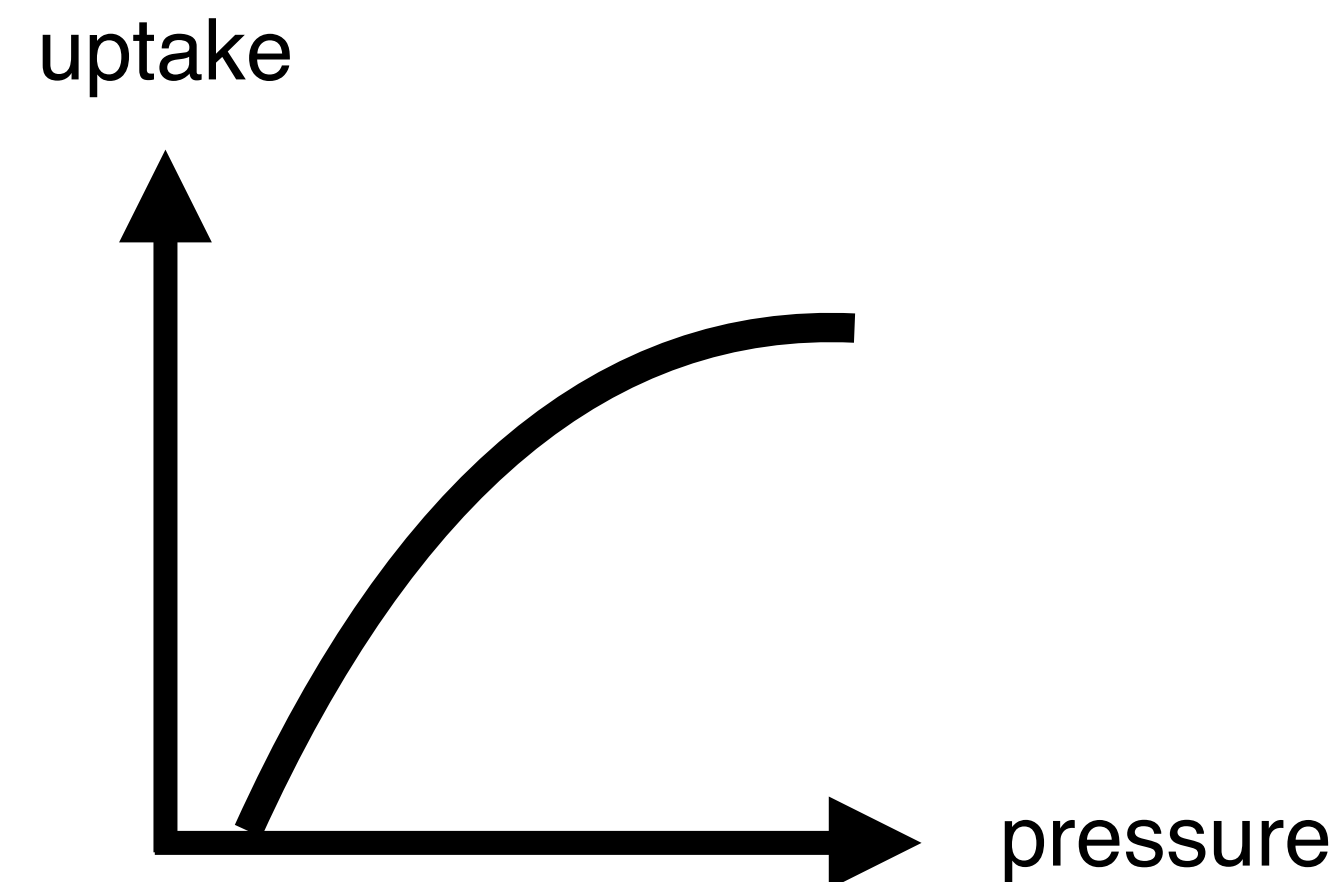
Recall: Testing a MOF for carbon capture

Experiment: Weeks/Months

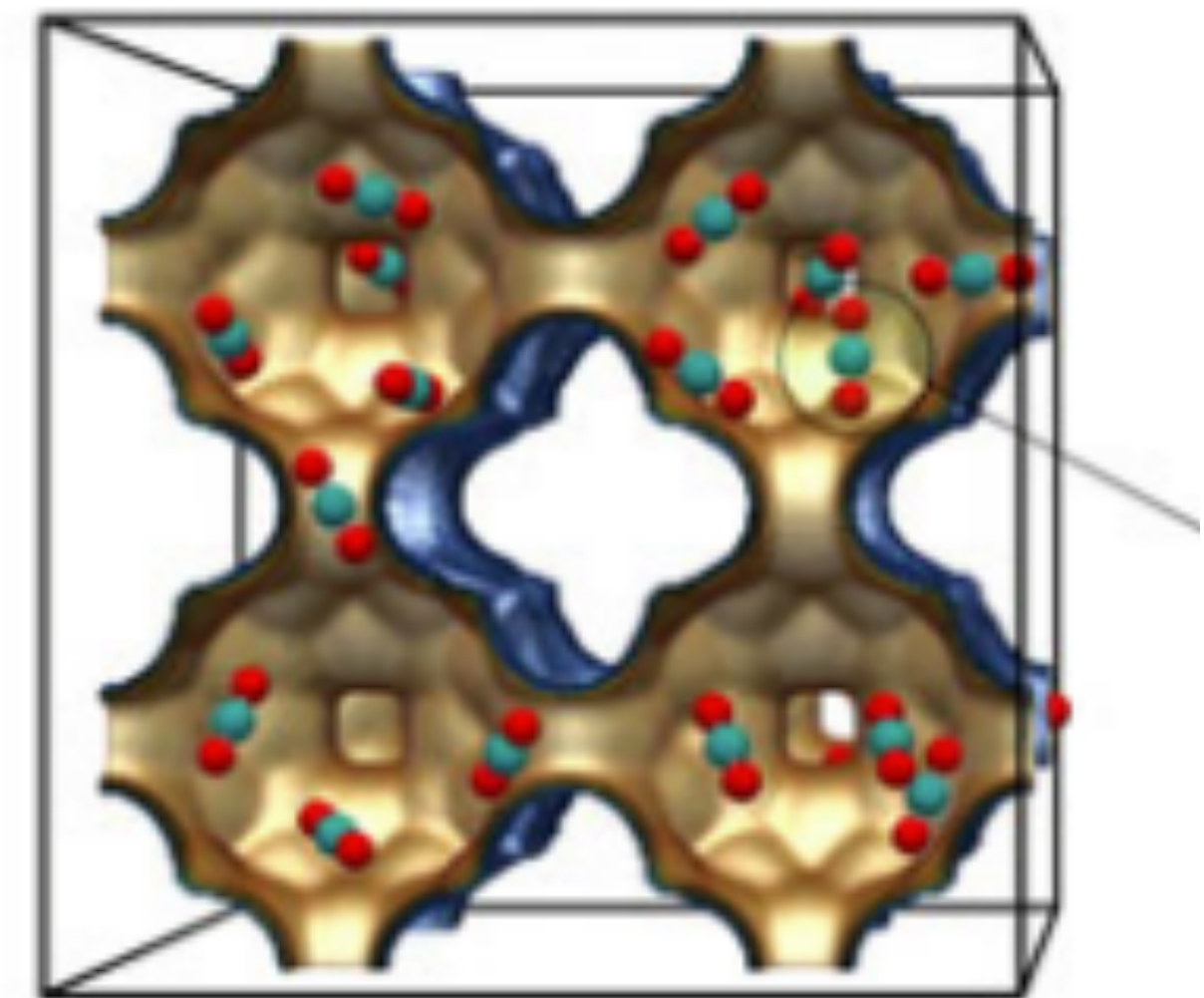


Molecular Simulation **2013**, 39 (14–15), 1253–1292.

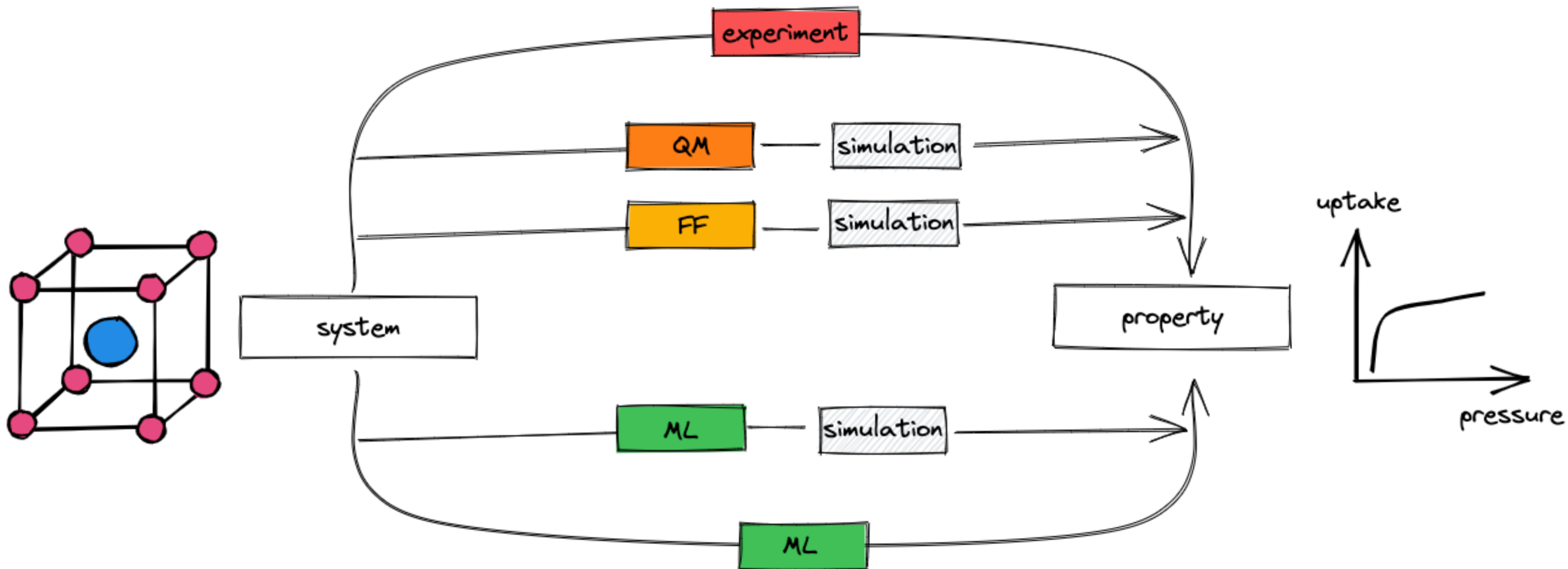
Computer simulations: Hours/Days



$$\frac{\text{acc}(o \rightarrow n)}{\text{acc}(n \rightarrow o)} = e^{-\beta [U_n(s^N; \mathbf{h}) - U_o(s^N; \mathbf{h})]}$$
$$\frac{\text{acc}(o \rightarrow n)}{\text{acc}(n \rightarrow o)} = e^{-\beta [U_n(s^{N+1}; \mathbf{h}) - U_o(s^N; \mathbf{h})]}$$



Where does ML fit in this picture?



SUPERVISED

UNSUPERVISED

REINFORCEMENT

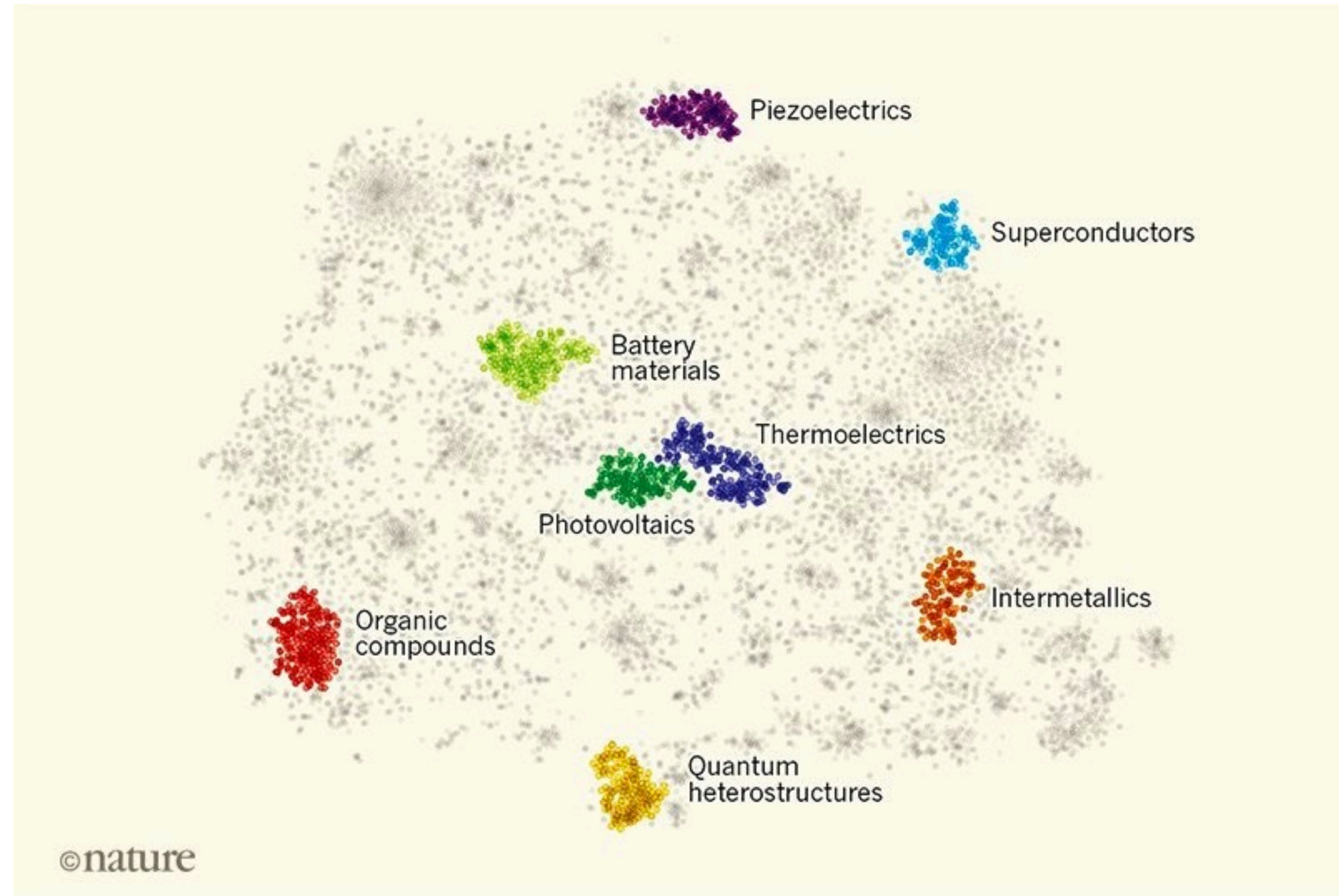
Unsupervised learning

Compressing and visualizing data



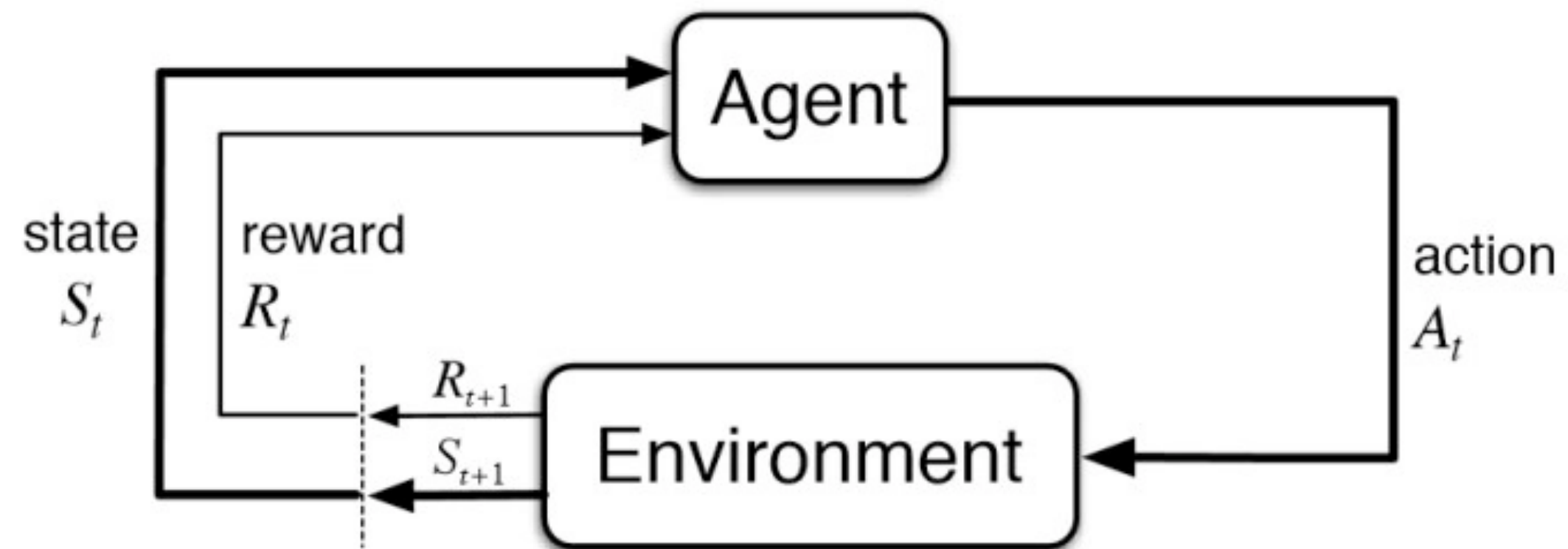
Unsupervised learning

*Finding patterns/
clusters in data*



Reinforcement learning

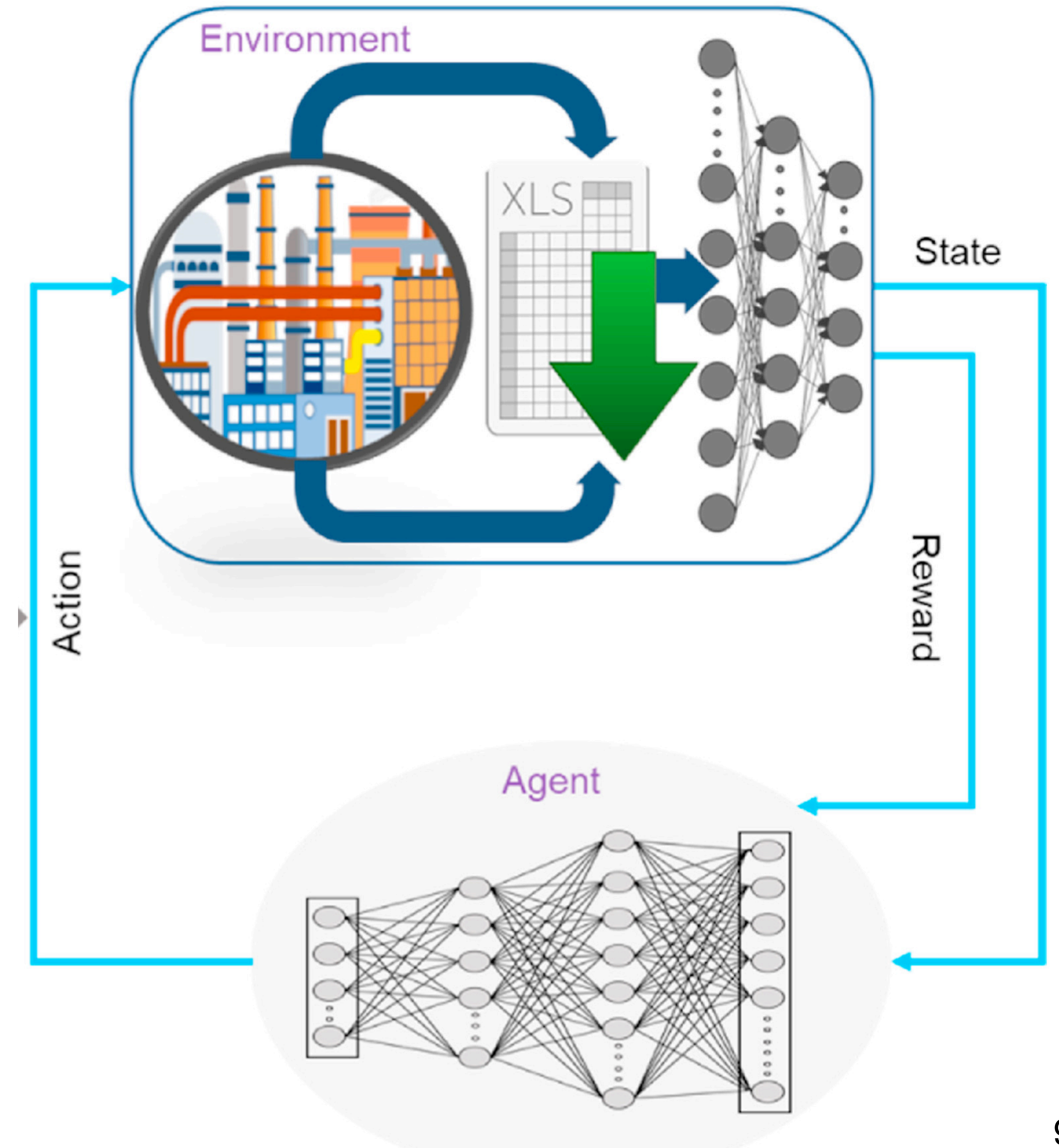
An agent in an environment



Reinforcement learning

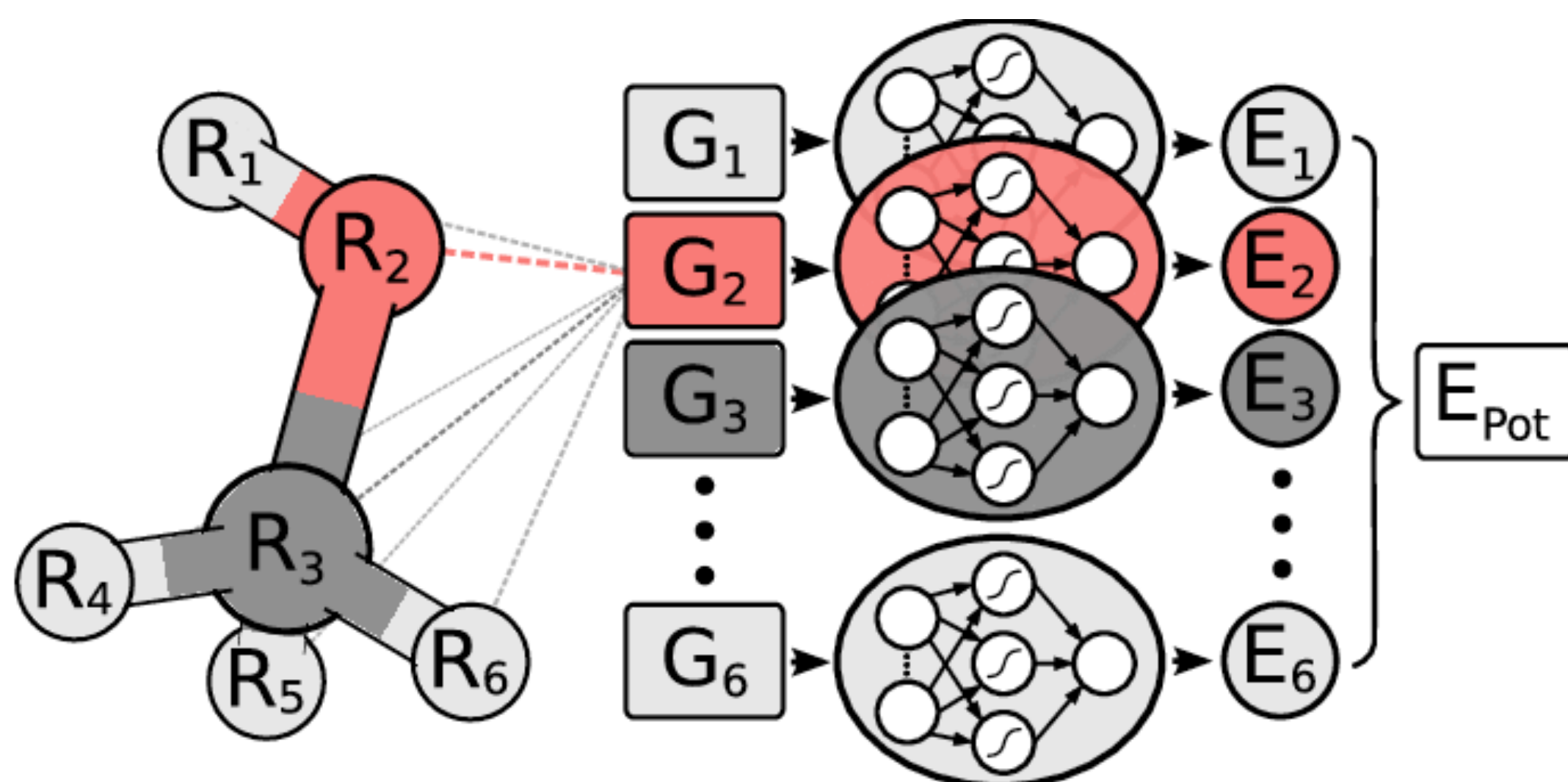
An agent in an environment

Deep Reinforcement Learning Optimization Framework



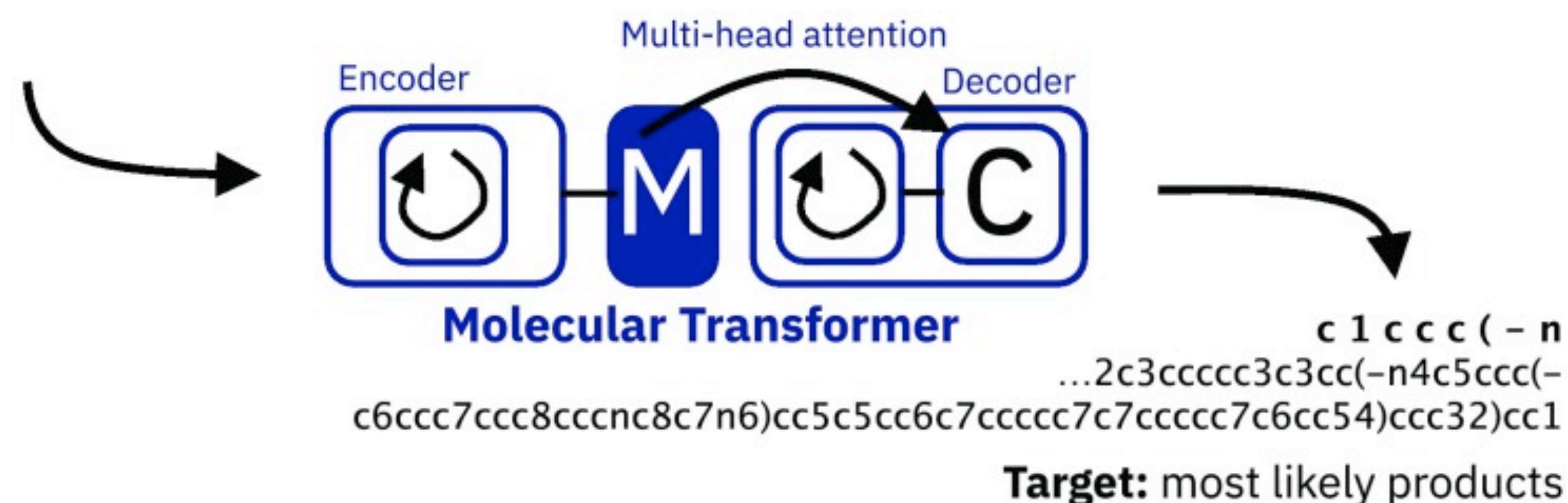
Supervised learning

Predicting given some features



Input: reactants-reagents (atom-wise tokenization)

Br c 1 c c c 2 ...c(c1)c1cc3c4cccc4c4cccc4c3cc1n2-c1ccc2c(c1)c1cccc1n2-c1cccc1.CCO.
Cc1cccc1.OB(O)c1ccc2ccc3ccnc3c2n1.c1ccc([PH](c2cccc2)(c2cccc2)[Pd]([PH](c2cccc2)
(c2cccc2)c2cccc2)([PH](c2cccc2)(c2cccc2)c2cccc2)[PH](c2cccc2)(c2cccc2)c2cccc2)cc1



SUPERVISED

Given X and Y
Predicting new Y

UNSUPERVISED

Given X
Trying to understand
structure in X

REINFORCEMENT

“Self-learn” in an
environment

SUPERVISED

Given X and Y
Predicting new Y

UNSUPERVISED

Given X
Trying to understand
structure in X

REINFORCEMENT

“Self-learn” in an
environment

Supervised ML workflow



Supervised ML workflow

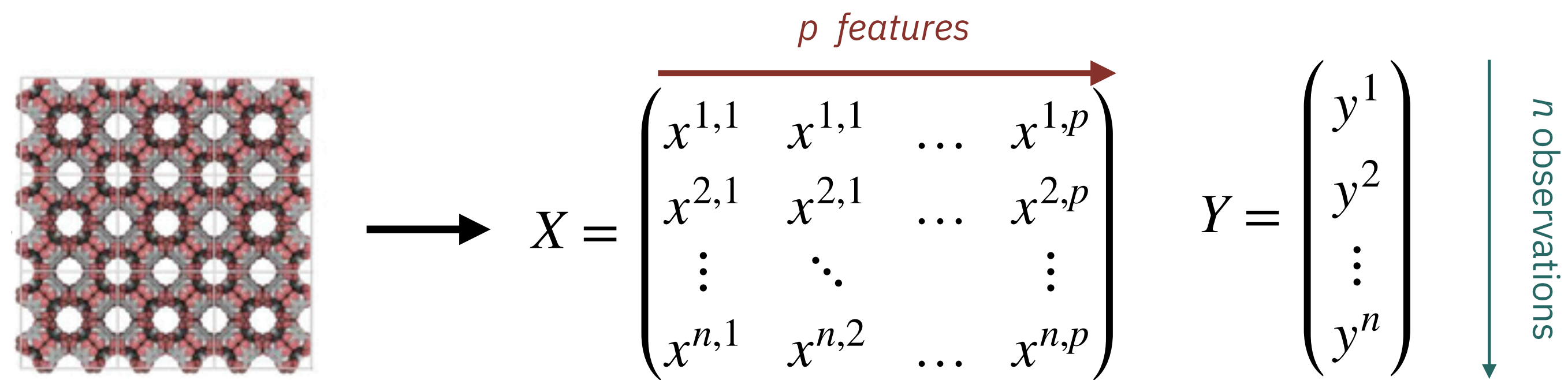
$$X = \begin{pmatrix} x^{1,1} & x^{1,1} & \dots & x^{1,p} \\ x^{2,1} & x^{2,1} & \dots & x^{2,p} \\ \vdots & \ddots & & \vdots \\ x^{n,1} & x^{n,2} & \dots & x^{n,p} \end{pmatrix} \quad Y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix}$$

p features (indicated by a red arrow above the matrix X)

n observations (indicated by a green arrow to the right of the matrix Y)



Supervised ML workflow



Intro

Learning Objectives:

- How does this module fit into the big picture?
- Different types of ML
- Supervised ML big picture

Cu1 Cu 0.04746(3) 0.70528(2) 0.5000
Cu2 Cu 0.06207(2) 0.60834(2) 0.5000
Cu3 Cu 0.377374(18) 0.327333(17) 0.145781(12)
Cu4 Cu 0.337355(17) 0.272144(17) 0.196102(12)
O1 O 0.11047(12) 0.62203(11) 0.46304(8)
O2 O 0.09801(12) 0.70448(11) 0.46334(7)
O3 O 0.17739(12) 0.81124(11) 0.37594(7)
O4 O 0.21888(12) 0.77657(11) 0.33090(7)
O5 O 0.27044(11) 0.59561(11) 0.21620(7)
O6 O 0.32403(11) 0.56455(10) 0.17643(7)
O7 O 0.35334(12) 0.38519(11) 0.17199(8)
O8 O 0.32649(12) 0.33741(11) 0.21710(7)
O9 O 0.14566(12) 0.28556(11) 0.33326(8)
O10 O 0.10519(11) 0.23780(11) 0.37266(7)
O11 O 0.00204(12) 0.30975(11) 0.46366(7)

1. Feeding structures into models

Learning Objectives:

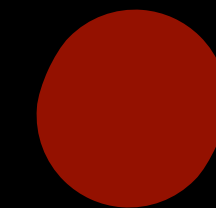
- Need for featurization
- Equivariances/invariances and approaches to incorporate them
- Locality approximation
- Categorical features
- Examples of descriptors

However, we only
have *little data*

all learnable functions

However, we only
have *little data*

all learnable functions



functions we want to learn

However, we only
have *little data*

all learnable functions

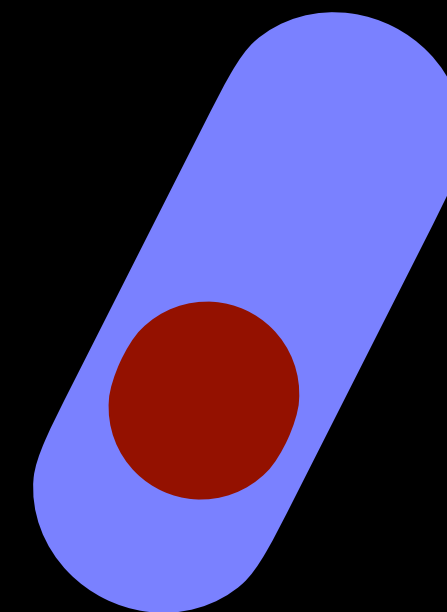
functions constrained by data

functions we want to learn

However, we only
have *little data*

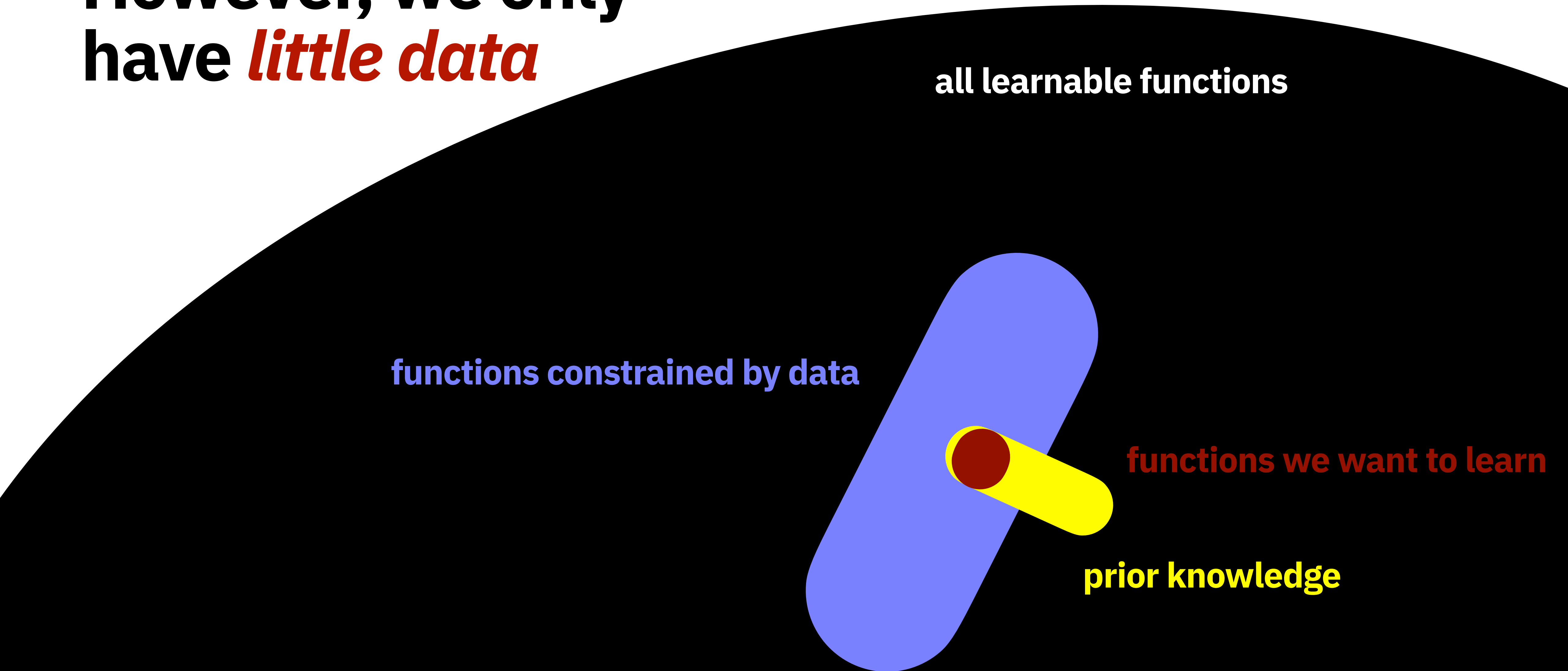
all learnable functions

functions constrained by data



functions we want to learn

However, we only
have *little data*



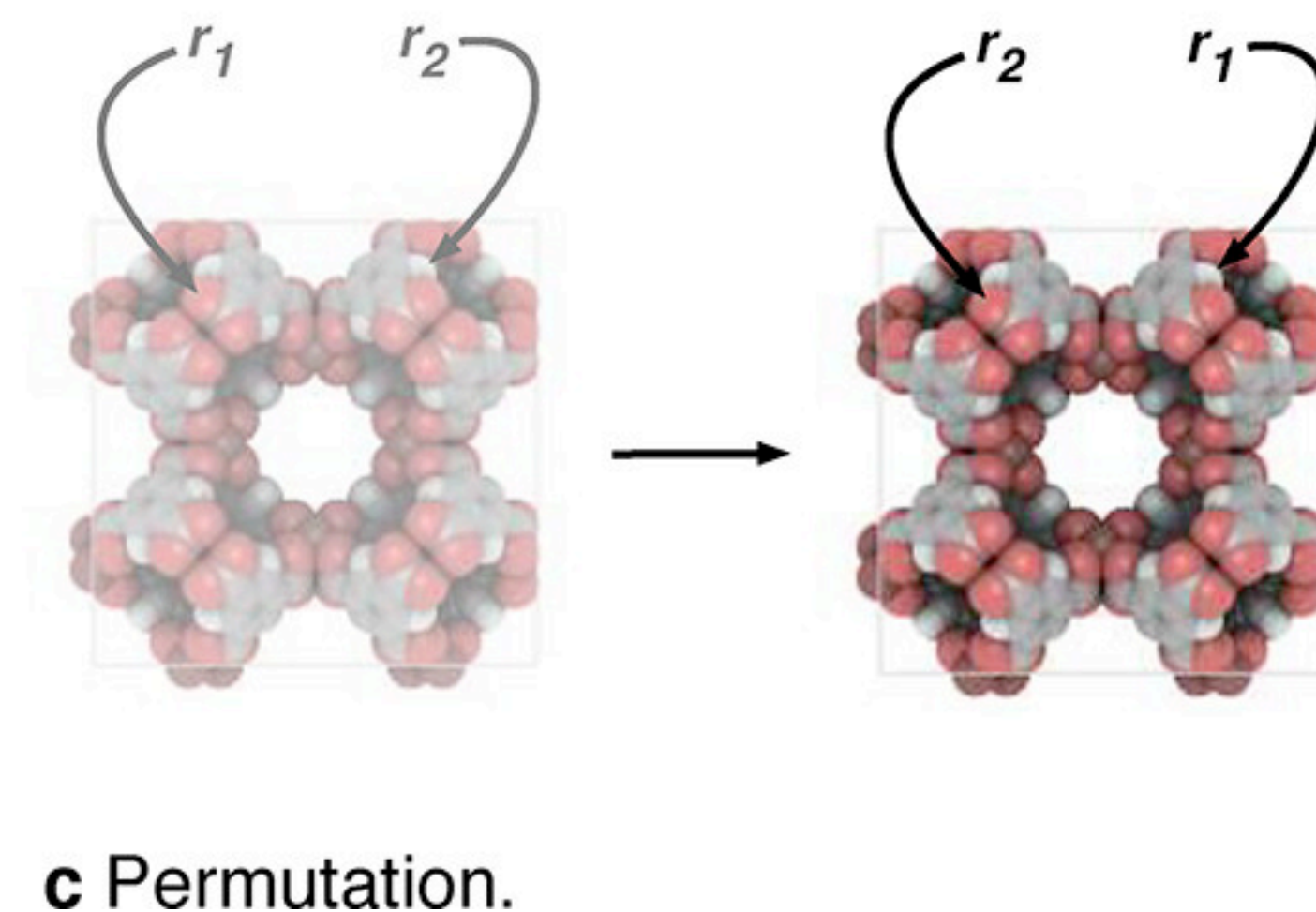
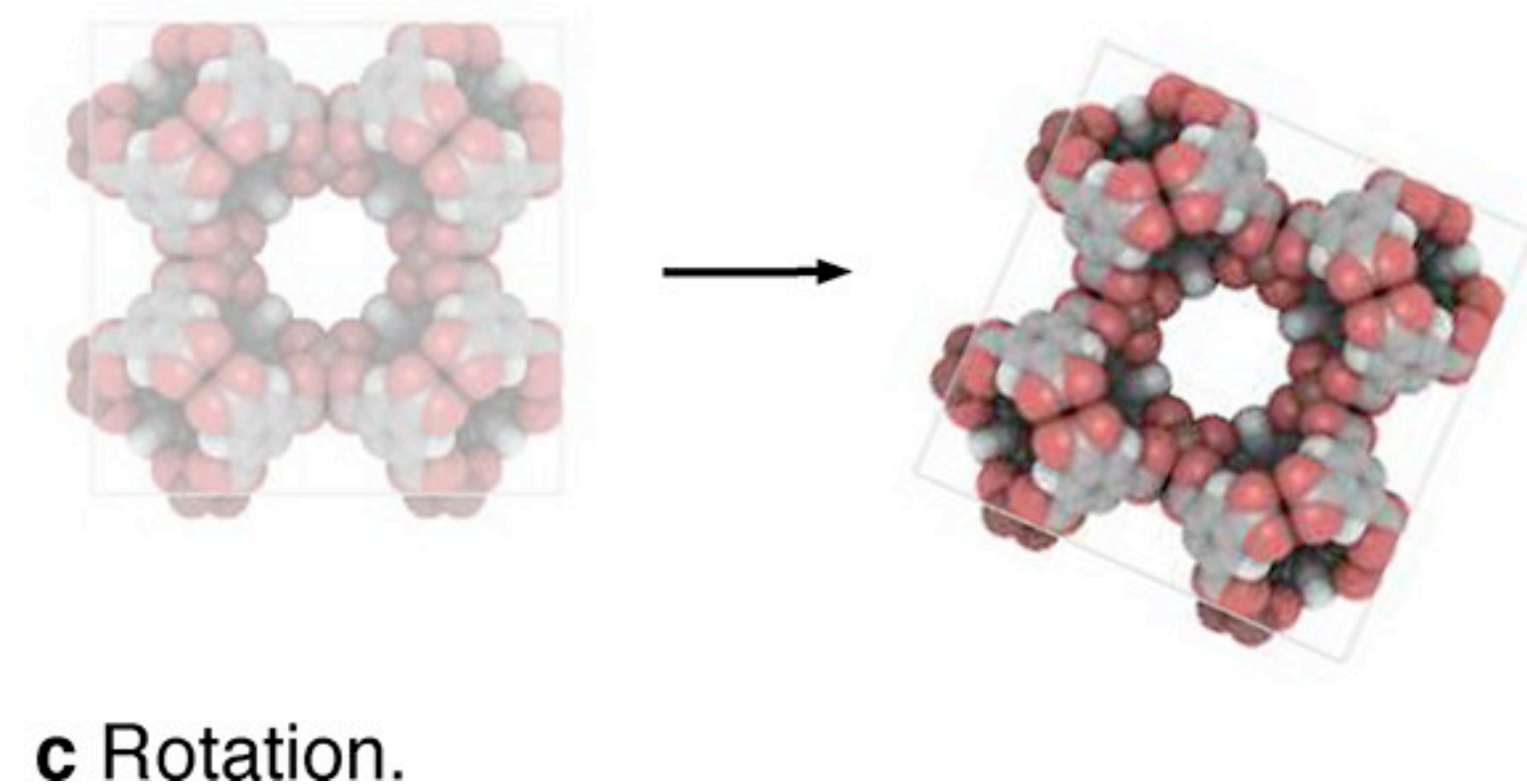
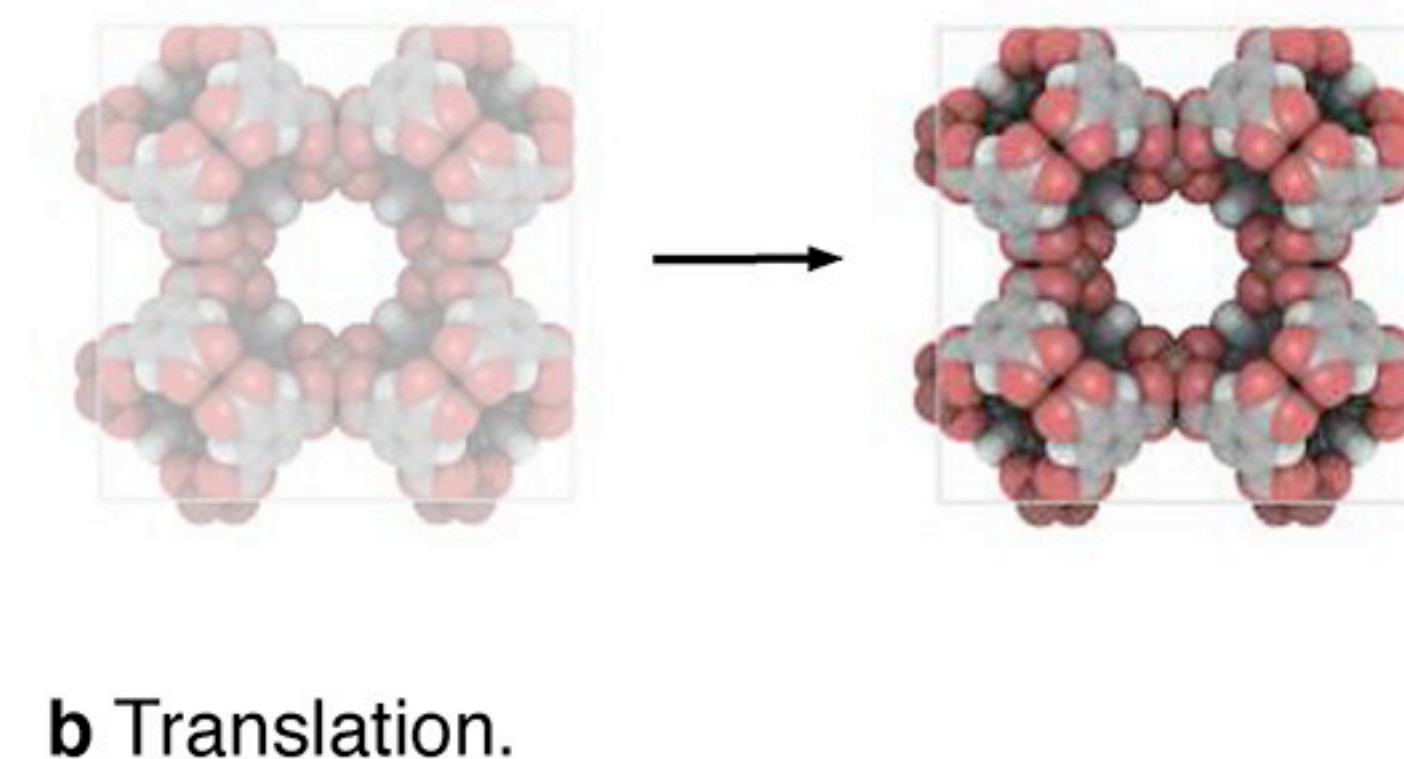
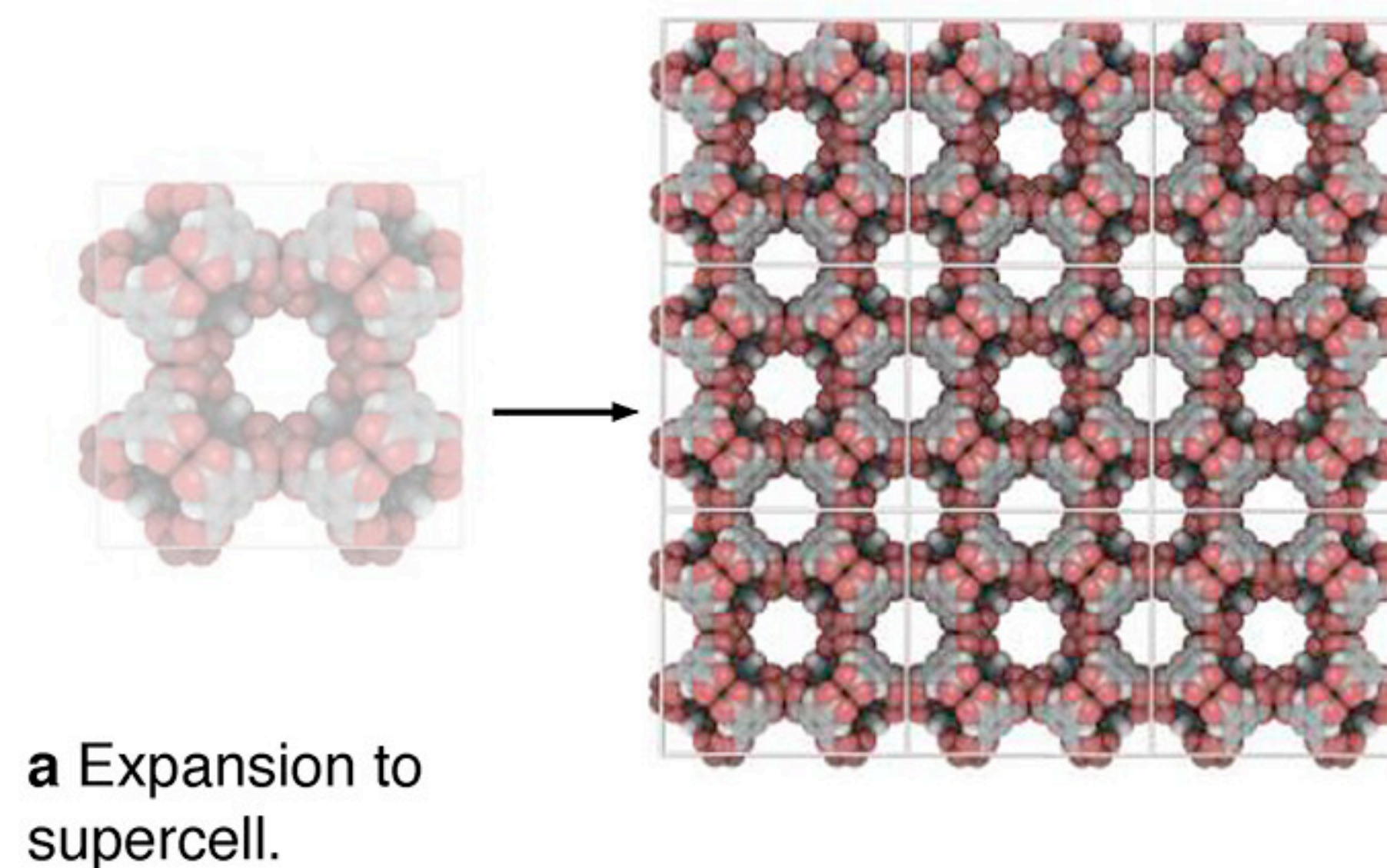
Inductive biases and wishlist

We would like to satisfy basic symmetries. Typically invariance/equivariance wr.t.

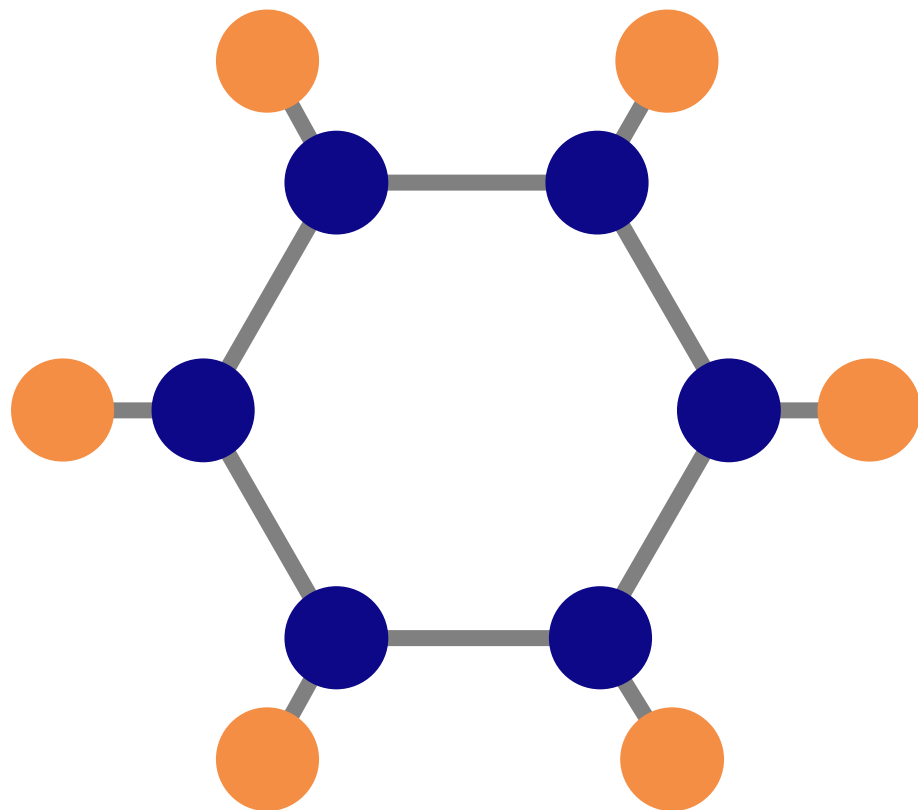
- translation
- rotation
- permutation
- expansion to supercell

Additionally, in some cases we want

- differentiability
- invertibility



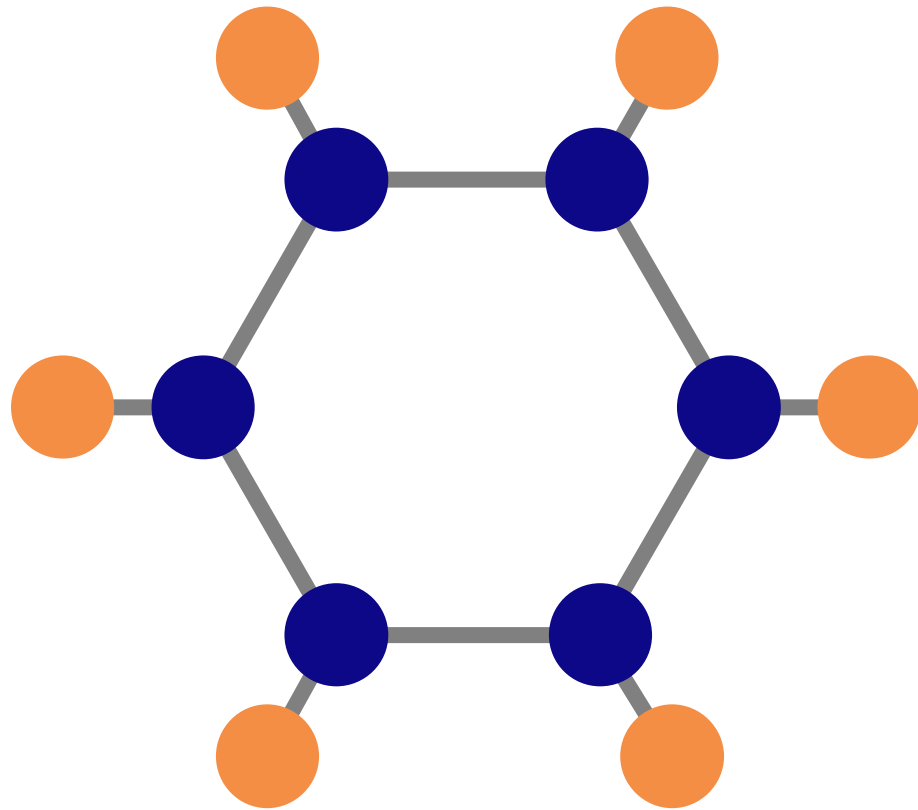
Three ways to make model symmetry aware



H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630

Coordinates are sensitive to translations, rotations, and permutation.

Three ways to make model symmetry aware



H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630

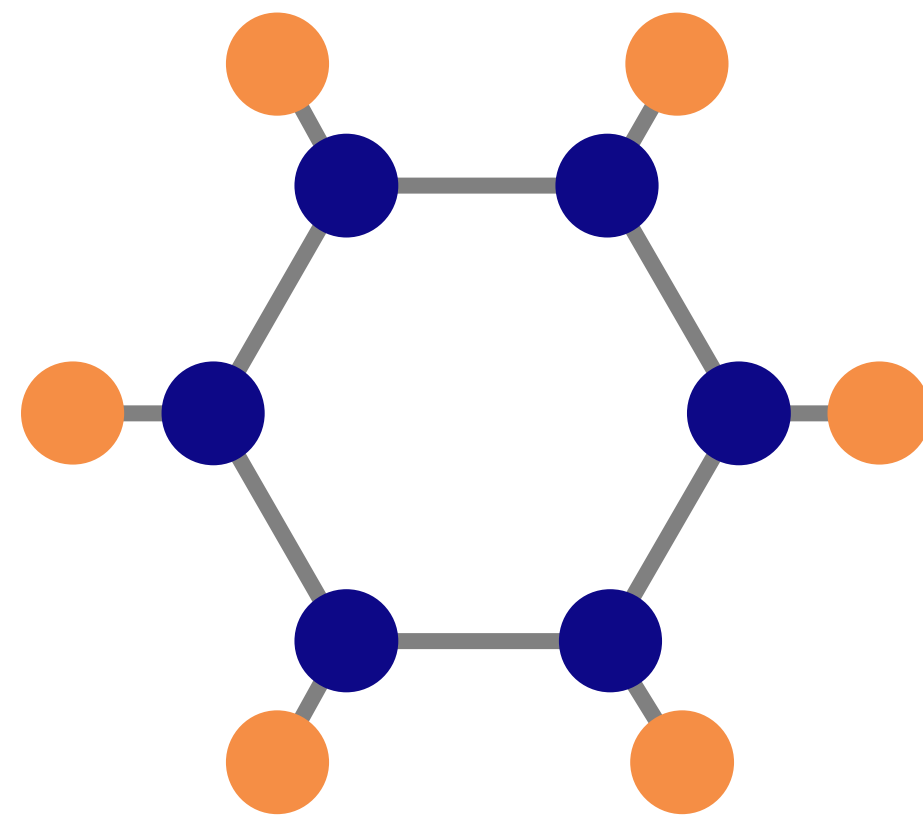
Coordinates are sensitive to translations, rotations, and permutation.

Data Augmentation

Throw data at the problem and see what you get!

22

Three ways to make model symmetry aware



H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630

Coordinates are sensitive to translations, rotations, and permutation.

Data Augmentation

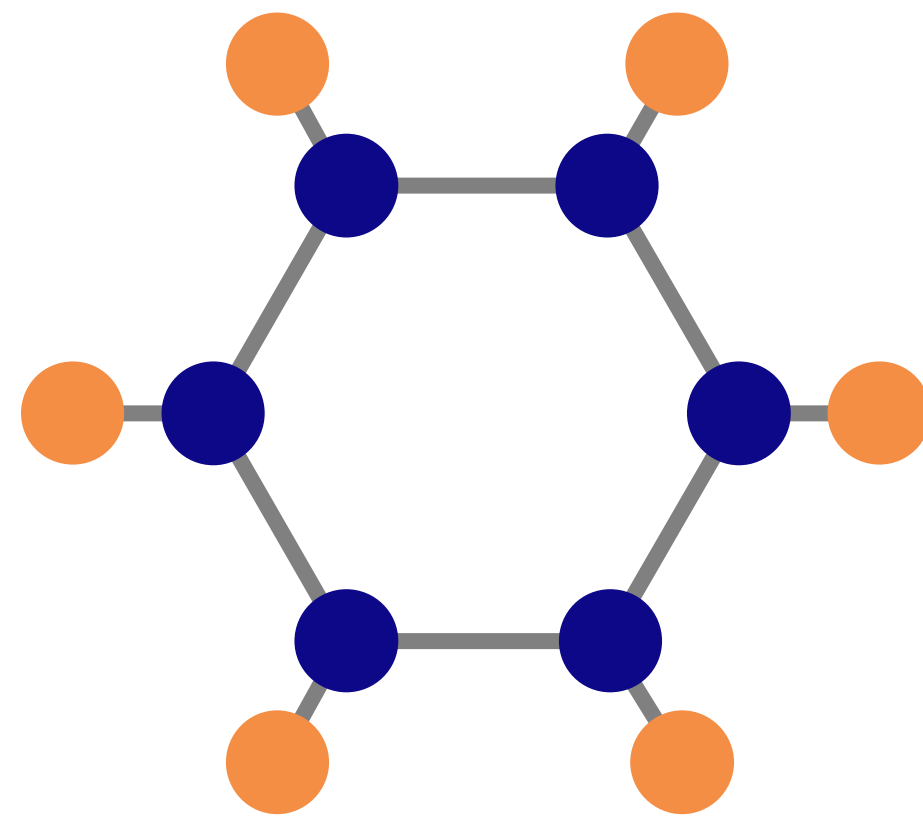
Throw data at the problem and see what you get!

Invariant Inputs Equivariant Inputs

Convert your data to invariant representations so the model can't possibly mess it up.

22

Three ways to make model symmetry aware



H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630

Coordinates are sensitive to translations, rotations, and permutation.

Data Augmentation

Throw data at the problem and see what you get!

Invariant Inputs Equivariant Inputs

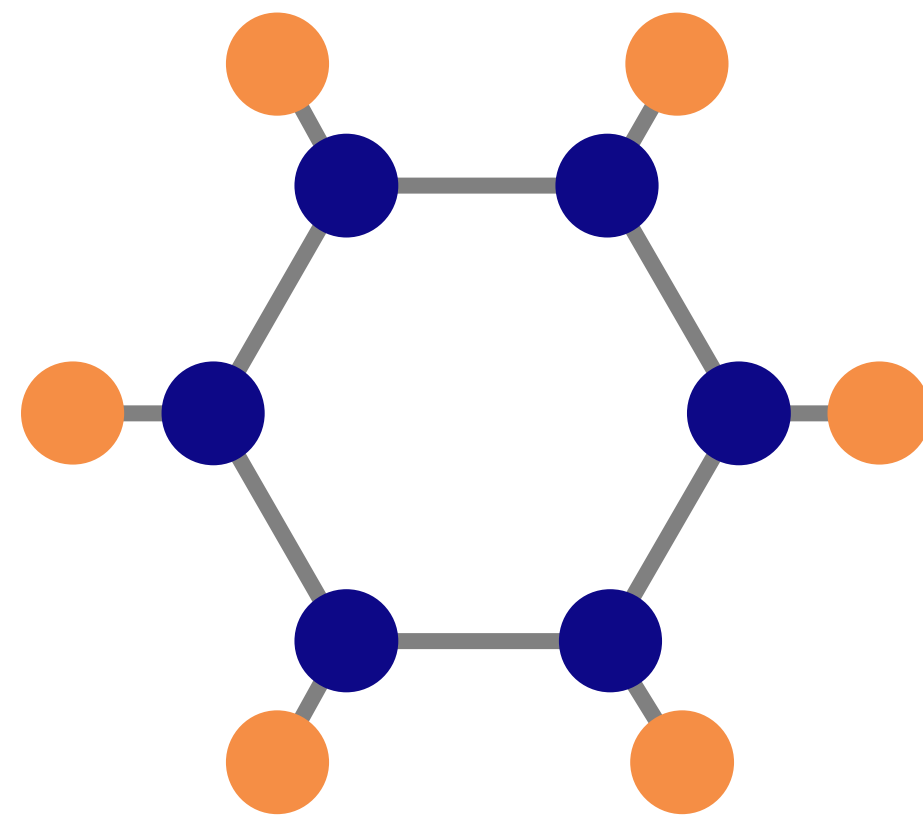
Convert your data to invariant representations so the model can't possibly mess it up.

22

Invariant models Equivariant models

Make a model that can naturally handle coordinates as input (equivariant neural networks)

Three ways to make model symmetry aware



H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630

Coordinates are sensitive to translations, rotations, and permutation.

Data Augmentation

Throw data at the problem and see what you get!

Invariant Inputs Equivariant Inputs

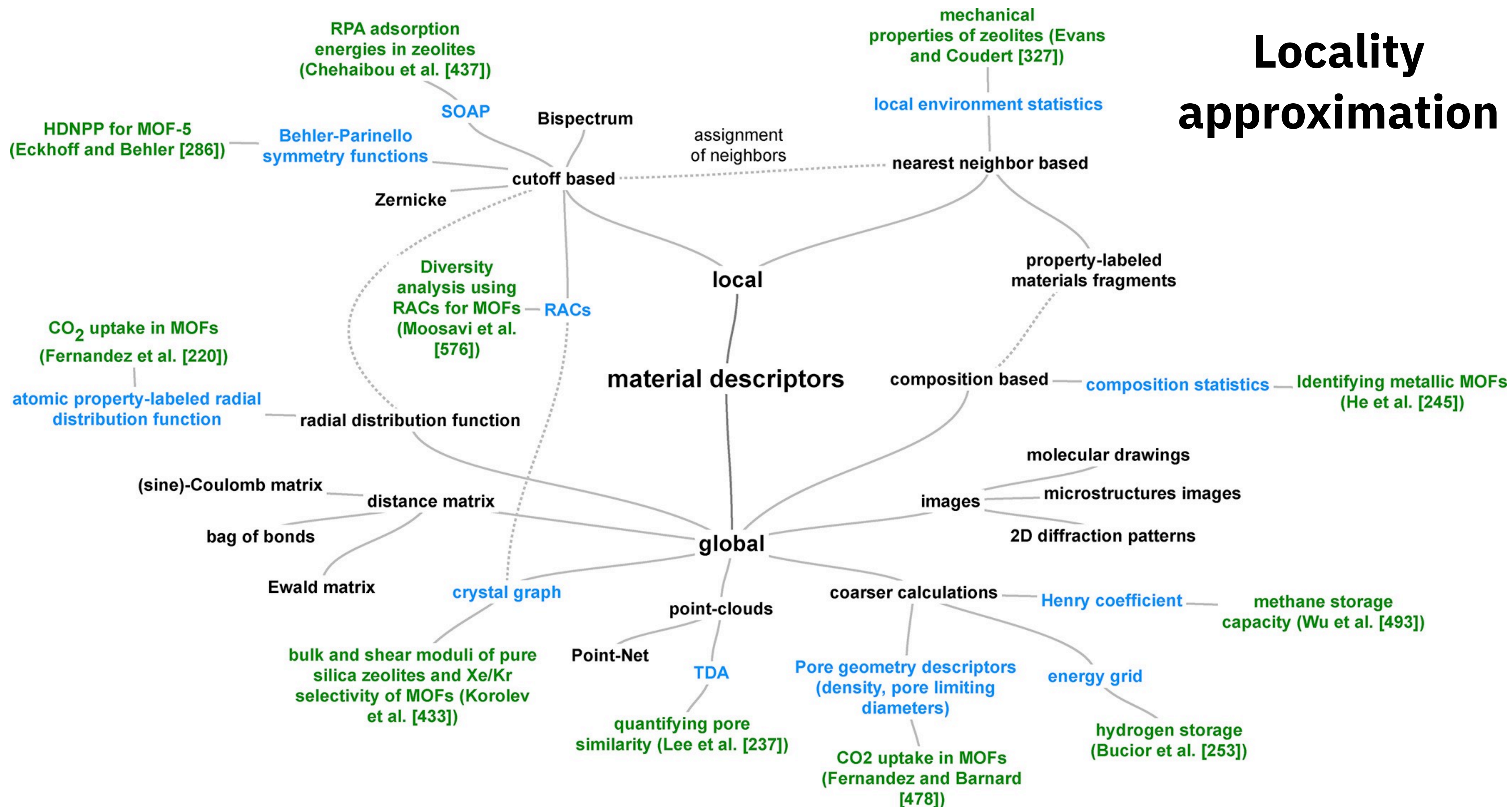
Convert your data to invariant representations so the model can't possibly mess it up.

22

Invariant models Equivariant models

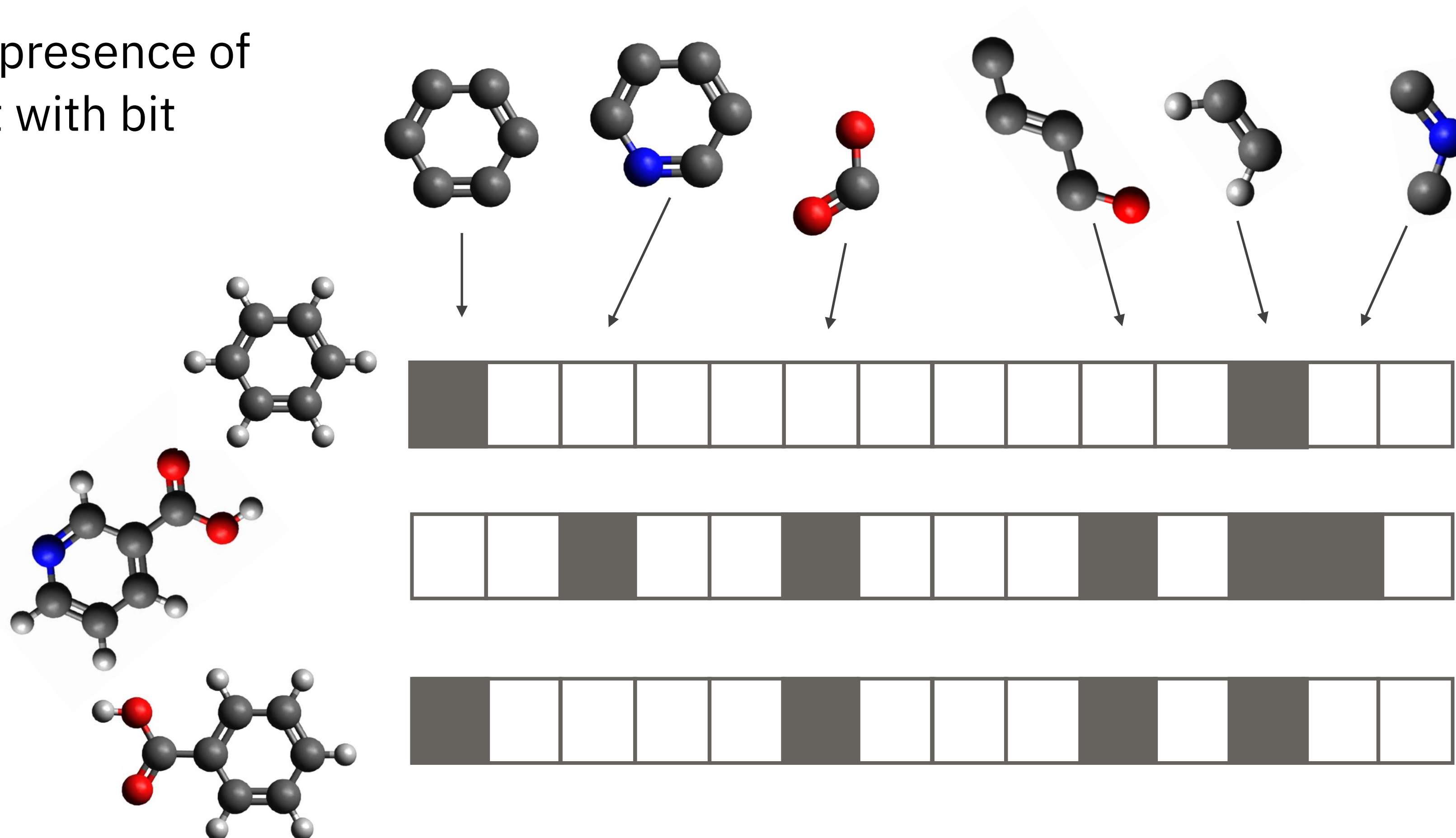
Make a model that can naturally handle coordinates as input (equivariant neural networks)

Locality approximation



One of the oldest approaches: Fragment based fingerprints

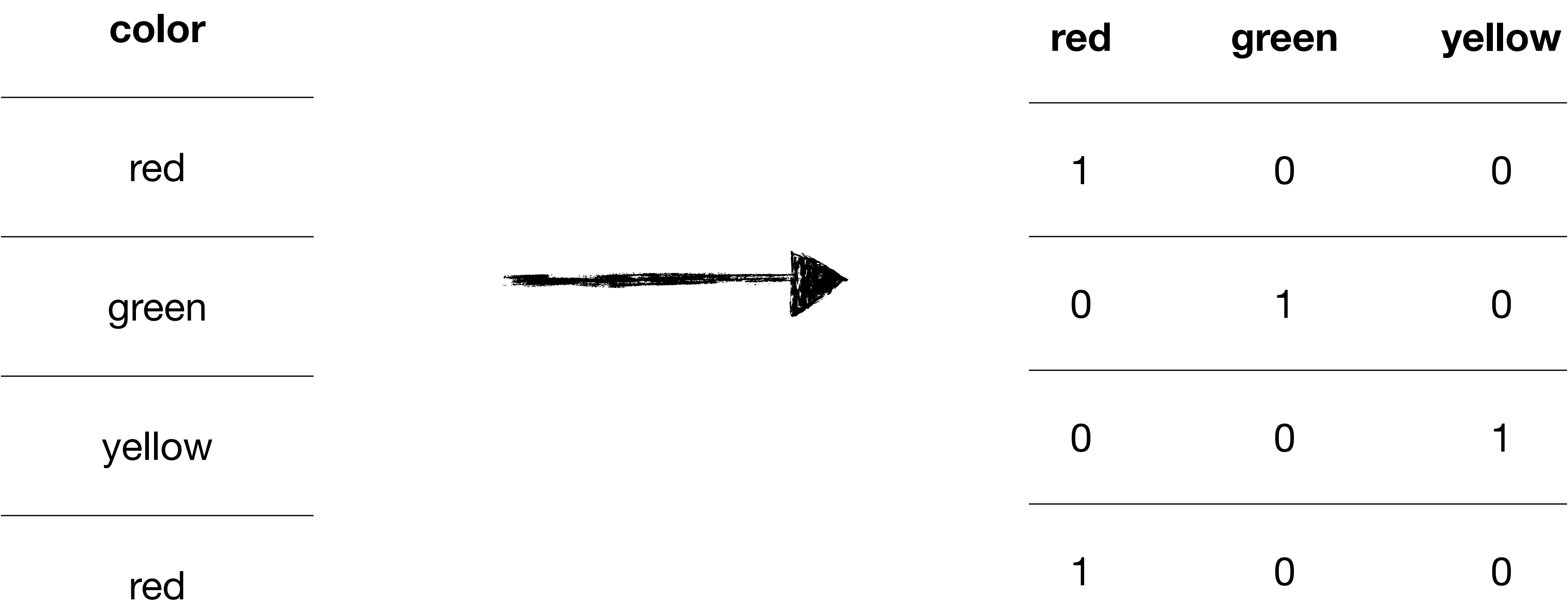
Indicate presence of
fragment with bit



Limitation of hard-coded categorical features

Typically, categories mapped into numbers using one-hot encoding

One-hot encoding equivalent to bit-fingerprints.



Limitation of hard-coded categorical features

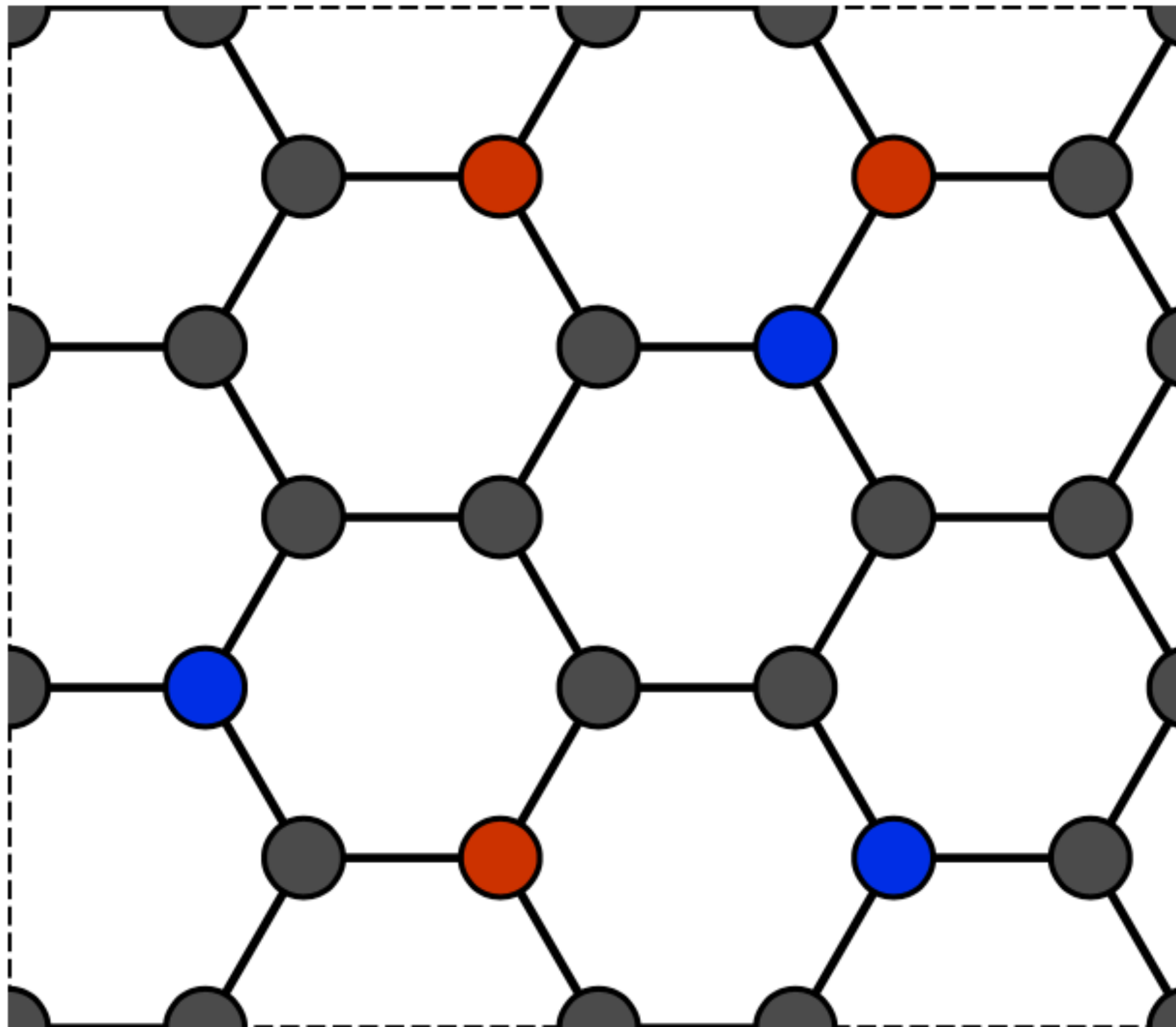
Typically, categories mapped into numbers using one-hot encoding

One-hot encoding equivalent to bit-fingerprints.

color		red	green	yellow
red		1	0	0
green		0	1	0
yellow		0	0	1
red		1	0	0

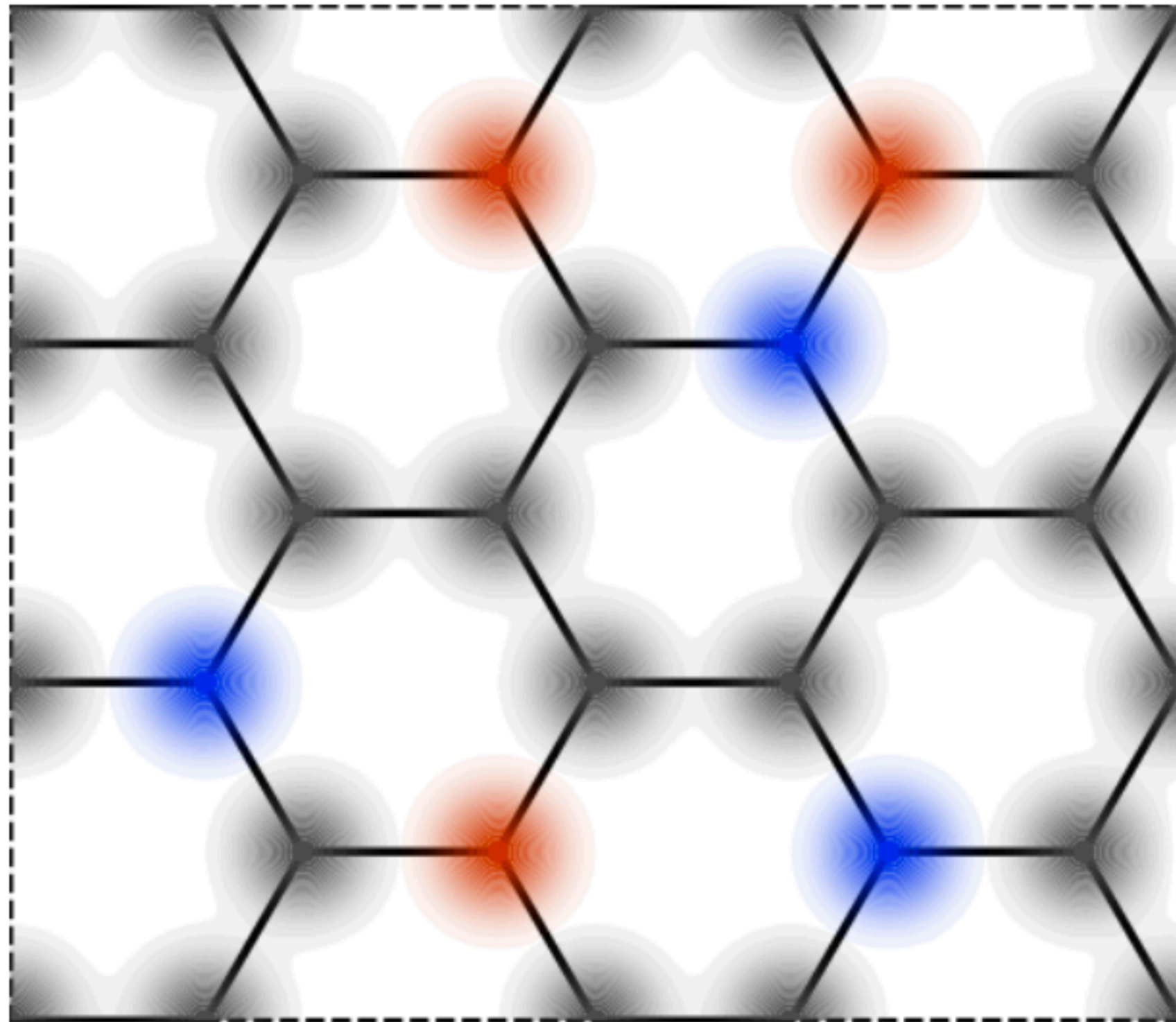
What do we do with out **blue** compound?

Symmetry functions



$$\begin{array}{l} \mathcal{A} \rightarrow \\ C \ 0.00 \ 0.00 \ 0.00 \\ C \ 0.00 \ 1.00 \ 0.00 \\ B \ 1.00 \ 2.00 \ 0.00 \\ \dots \end{array}$$

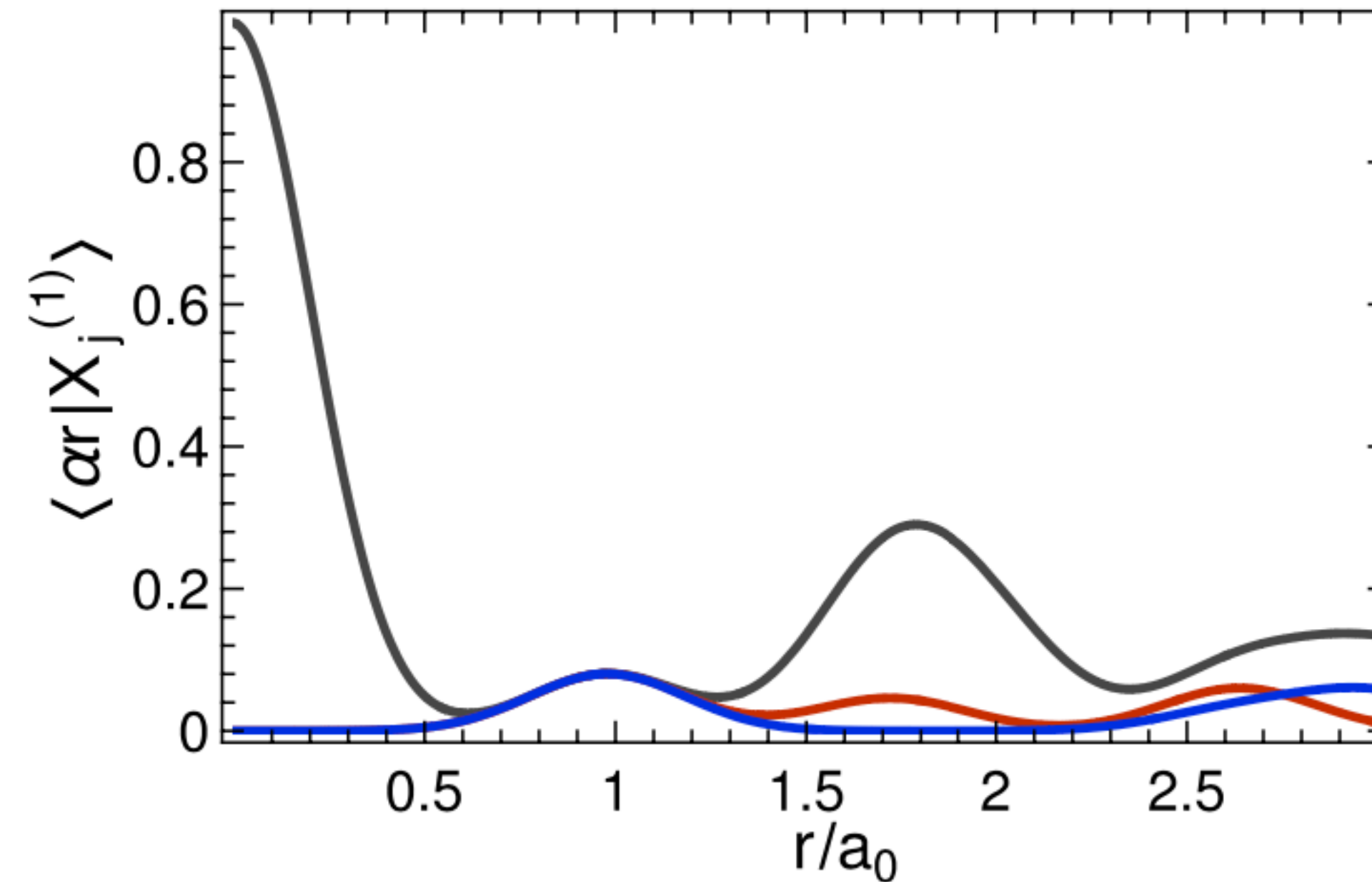
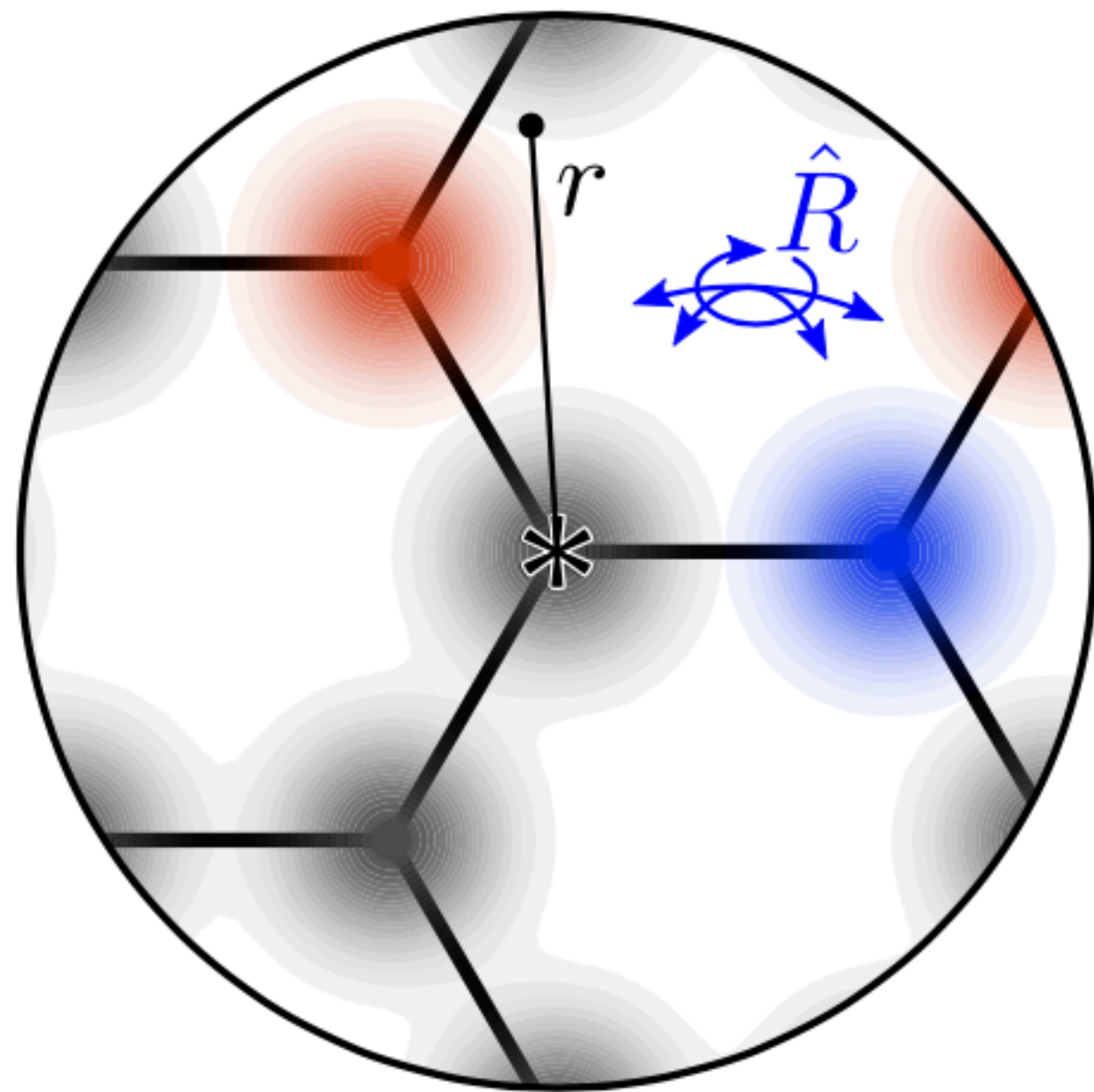
Symmetry functions



$$\langle a\mathbf{r}|\rho\rangle = \sum_i g(\mathbf{r} - \mathbf{r}_i)\delta_{aa_i}$$

$$\begin{aligned} |C\rangle & \blacksquare \\ |N\rangle & \blacksquare \\ |B\rangle & \blacksquare \end{aligned}$$

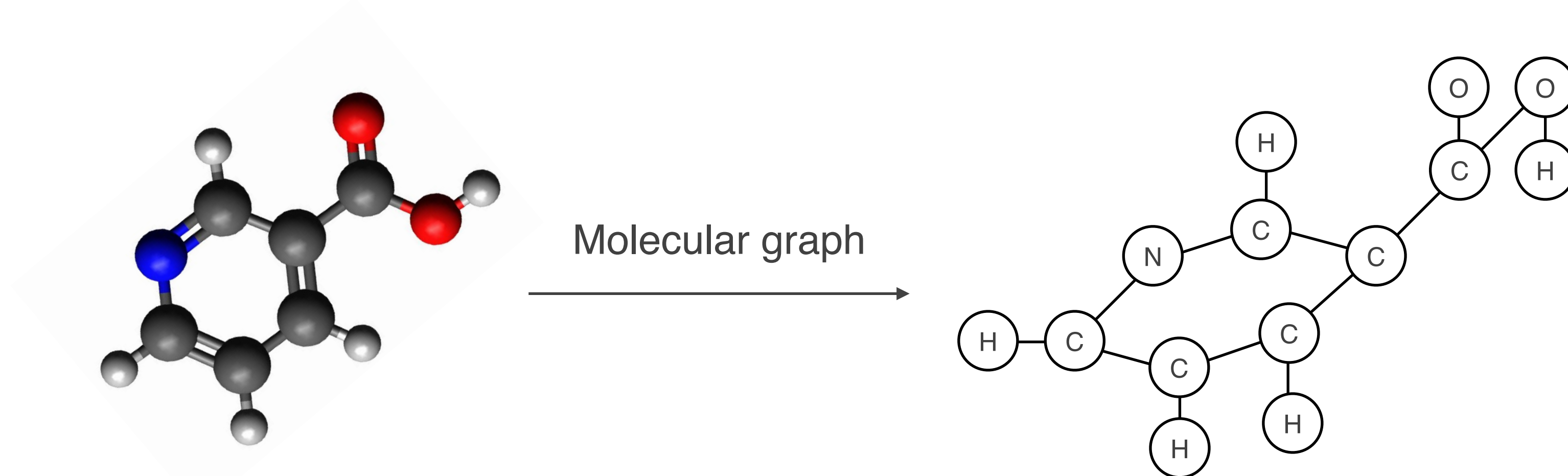
Symmetry functions



$$\langle ar | \overline{\rho_i^{\otimes 1}} \rangle = \int d\hat{R} \langle ar \hat{\mathbf{r}} | \hat{R} | \rho_i \rangle$$

Molecule graph summarization

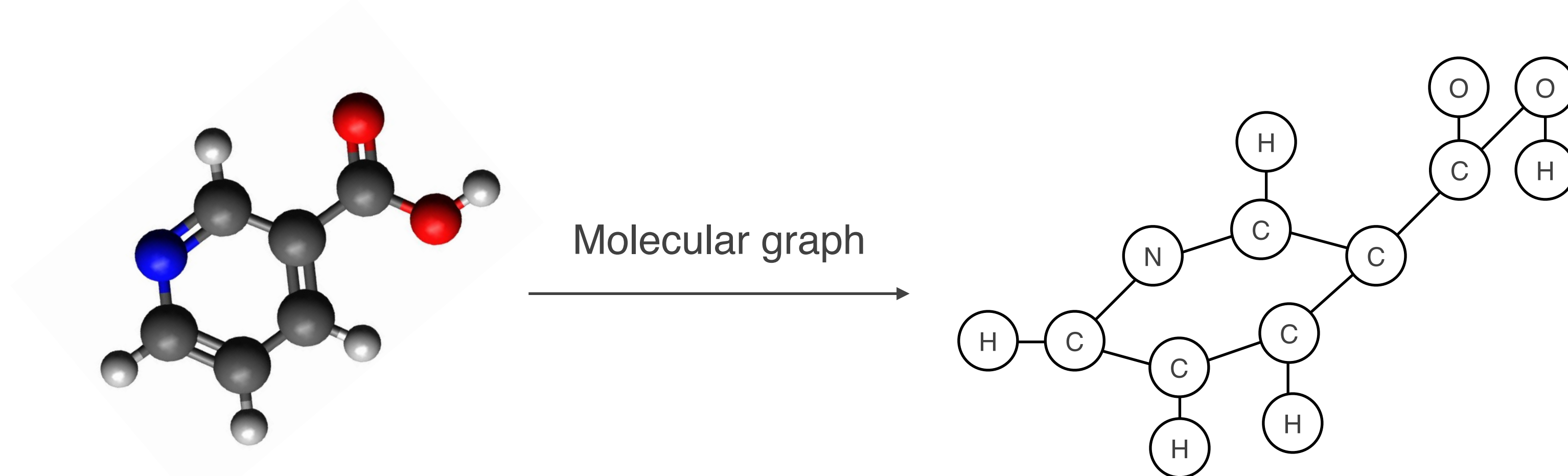
Create fix-length descriptor by capturing local correlations of molecule graph



$$[N]_{all} \chi_1^{diff} = \sum_i^{[N]} \sum_j^{all} (\chi_i - \chi_j) \delta(d_{i,j}, 1)$$

Molecule graph summarization

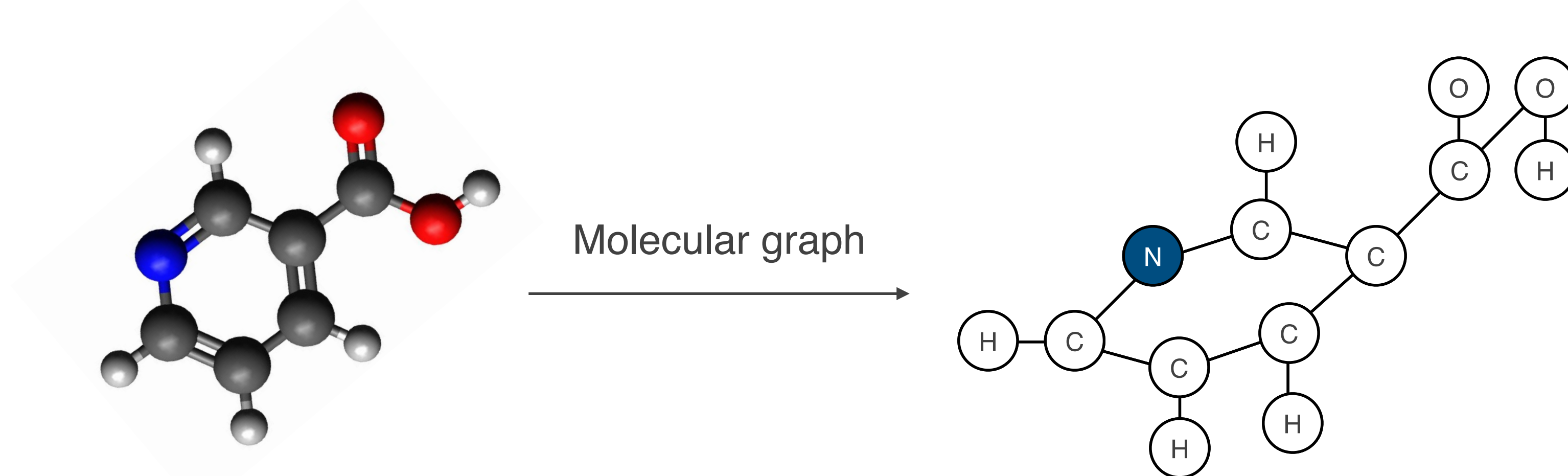
Create fix-length descriptor by capturing local correlations of molecule graph



$$\boxed{[N]}_{all} \chi_1^{diff} = \boxed{\sum_i^{[N]}} \sum_j^{all} (\chi_i - \chi_j) \boxed{\delta(d_{i,j}, 1)}$$

Molecule graph summarization

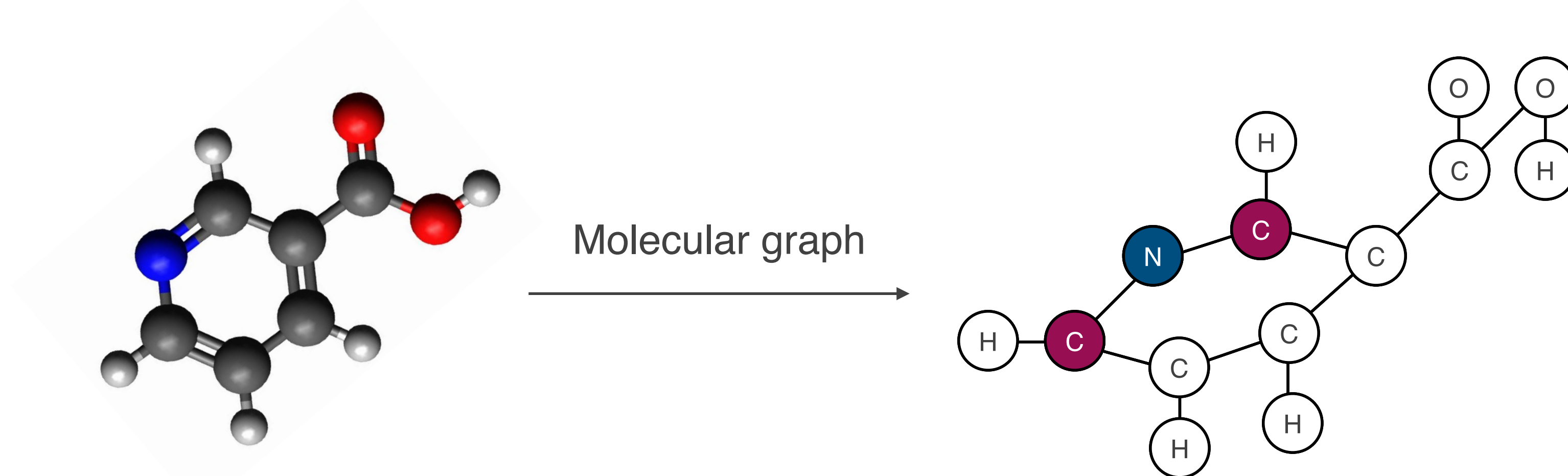
Create fix-length descriptor by capturing local correlations of molecule graph



$$\boxed{[N]}_{all} \chi_1^{diff} = \boxed{\sum_i^{[N]}} \sum_j^{all} (\chi_i - \chi_j) \boxed{\delta(d_{i,j}, 1)}$$

Molecule graph summarization

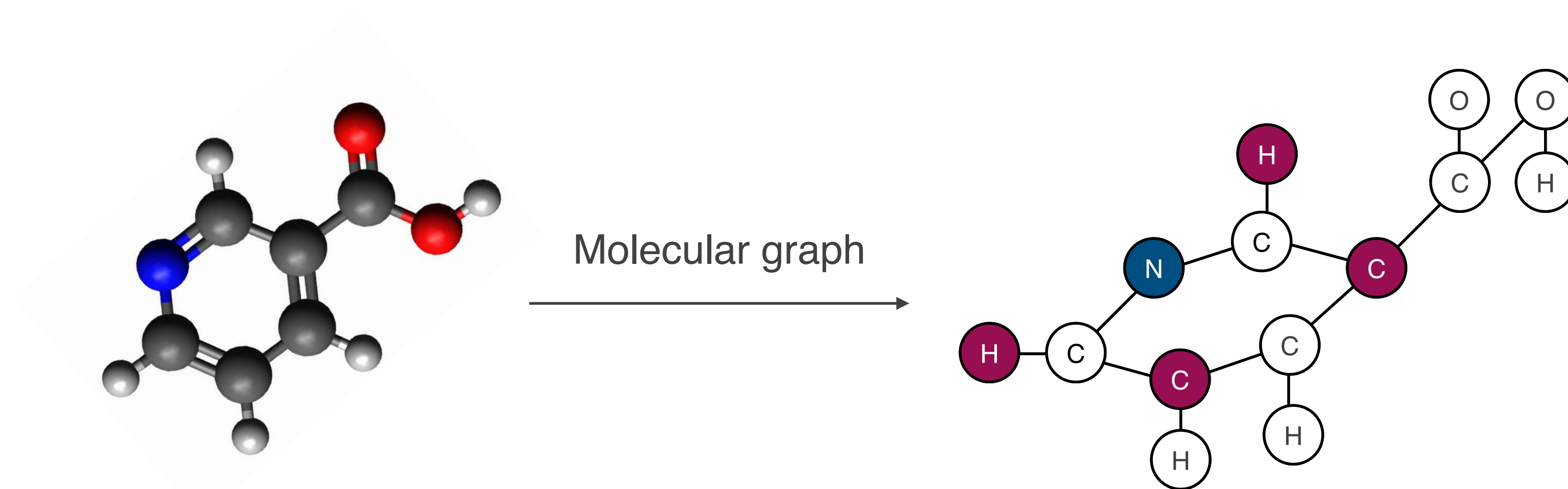
Create fix-length descriptor by capturing local correlations of molecule graph



$$\boxed{[N]}_{all} \chi_1^{diff} = \boxed{\sum_i^{[N]}} \sum_j^{all} (\chi_i - \chi_j) \boxed{\delta(d_{i,j}, 1)}$$

Molecule graph summarization

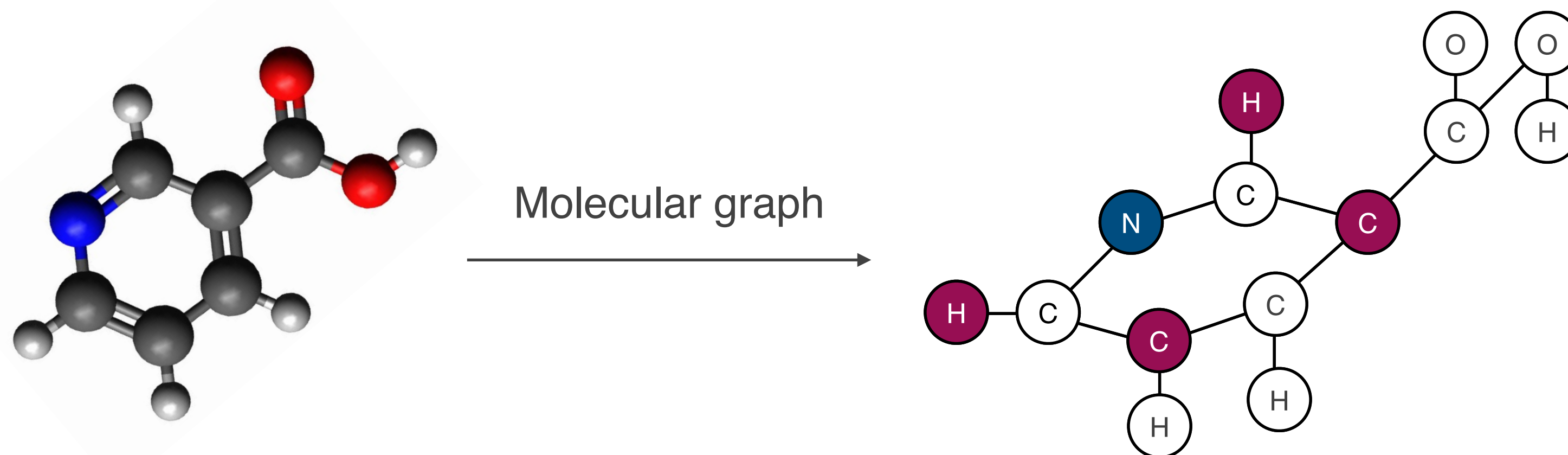
Create fix-length descriptor by capturing local correlations of molecule graph



$$[N]_{all} \chi_1^{diff} = \sum_i^{[N]} \sum_j^{all} (\chi_i - \chi_j) \delta(d_{i,j}, 2)$$

Molecule graph summarization

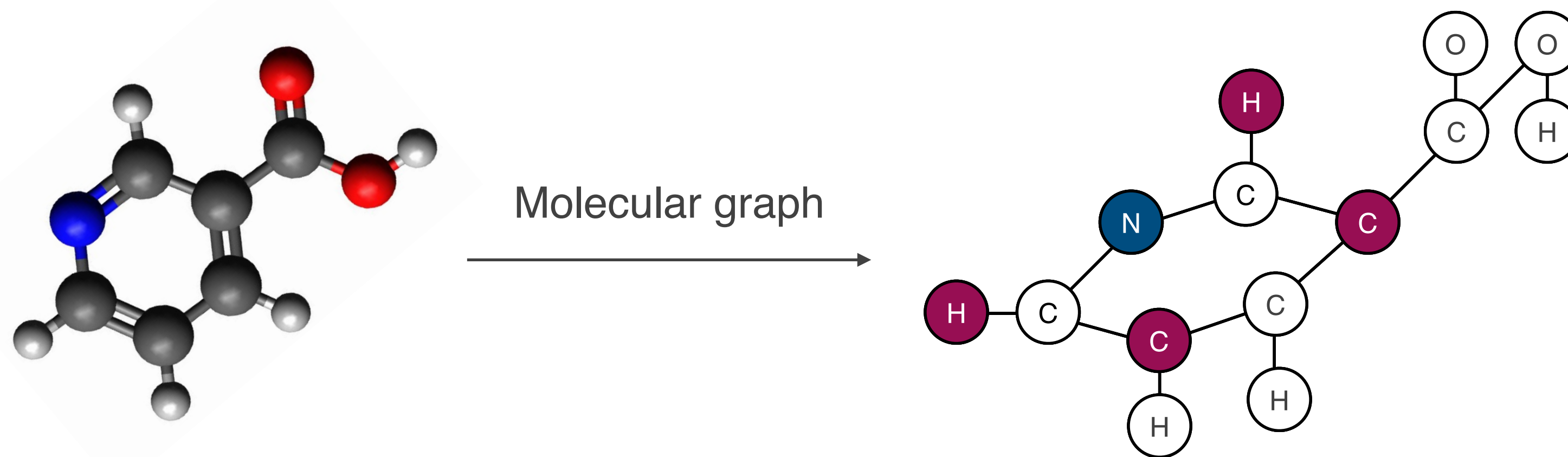
Create fix-length descriptor by capturing local correlations of molecule graph



$$\boxed{[N]}_{all} \chi_1^{diff} = \boxed{\sum_i^{[N]}} \sum_j^{all} (\chi_i - \chi_j) \boxed{\delta(d_{i,j}, 2)}$$

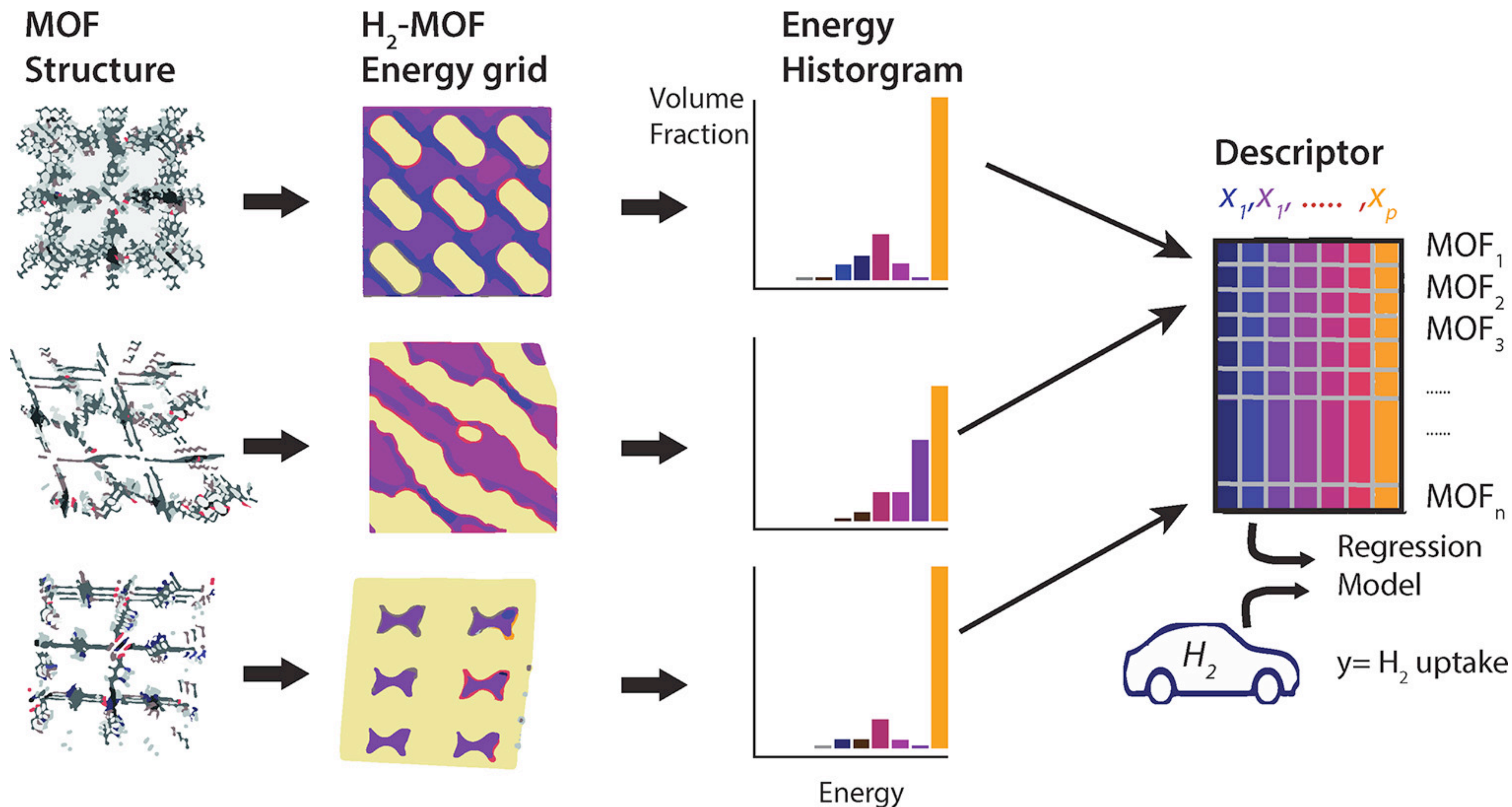
Molecule graph summarization

Create fix-length descriptor by capturing local correlations of molecule graph



$$\boxed{[N]}_{all} \chi_1^{diff} = \boxed{\sum_i^{[N]}} \sum_j^{all} (\chi_i - \chi_j) \boxed{\delta(d_{i,j}, 2)}$$

Cheaper surrogate



1. Feeding structures into models

Learning Objectives:

- Need for featurization
- Equivariances/invariances and approaches to incorporate them
- Locality approximation
- Categorical features
- Examples of descriptors

2. Training a basic model

Learning Objectives:

- Supervised ML workflow
- Empirical risk estimate
- Test data

Supervised ML workflow



Supervised ML workflow

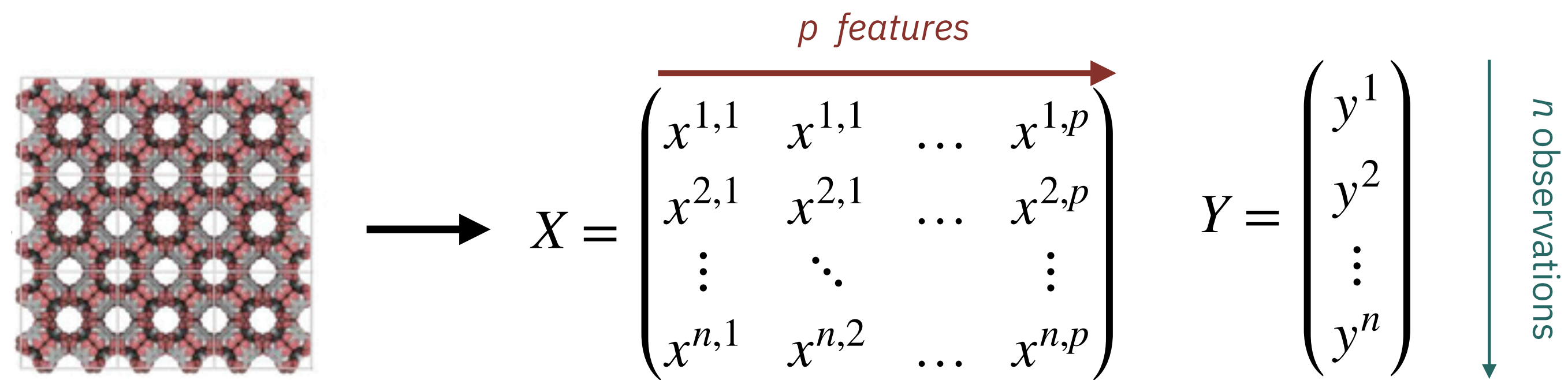
$$X = \begin{pmatrix} x^{1,1} & x^{1,1} & \dots & x^{1,p} \\ x^{2,1} & x^{2,1} & \dots & x^{2,p} \\ \vdots & \ddots & & \vdots \\ x^{n,1} & x^{n,2} & \dots & x^{n,p} \end{pmatrix} \quad Y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix}$$

p features (indicated by a red arrow above the matrix X)

n observations (indicated by a green arrow to the right of the matrix Y)



Supervised ML workflow



How do we measure progress towards our goal

Goal: Find good function f in function space $\mathcal{F}$

How do we measure progress towards our goal

Goal: Find good function f in function space $\mathcal{F}$

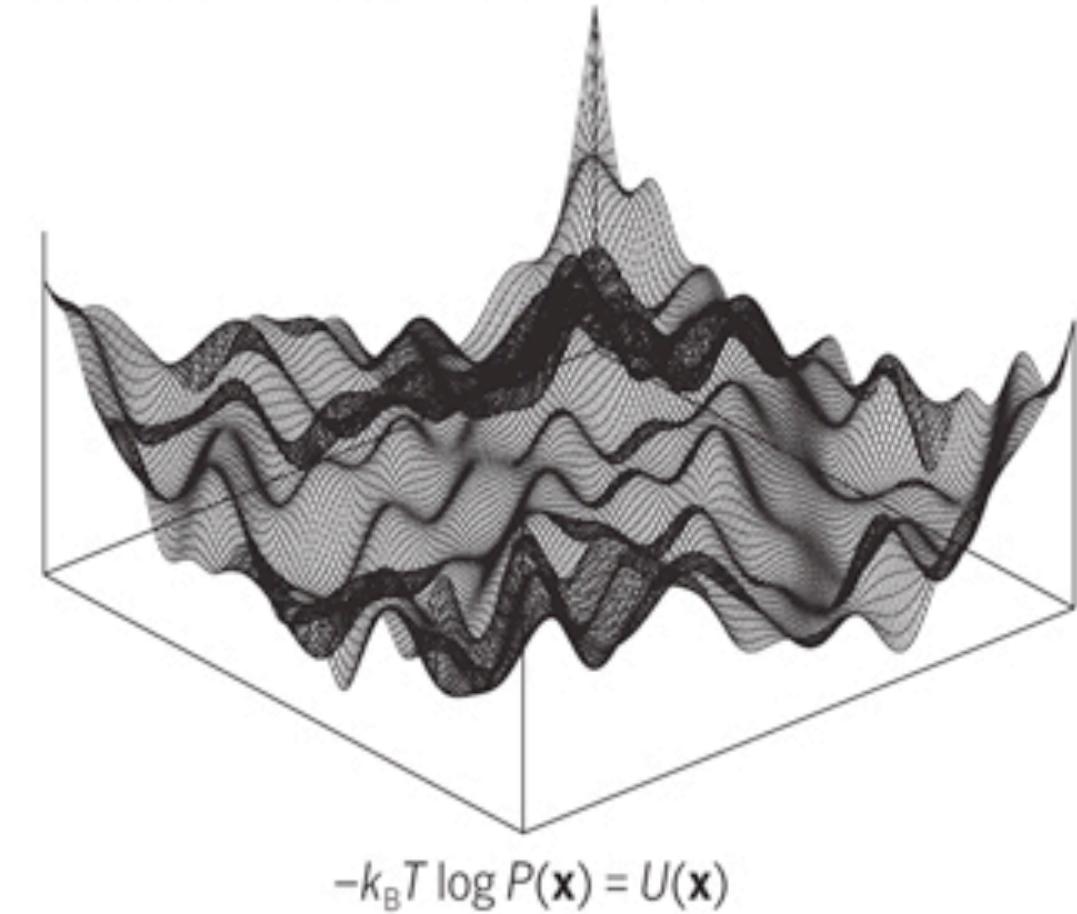
How: Minimize the expected risk of f $R(f) = E_Z[L(z, f)] = \int_Z L(z, f)p(z) \, dz$

How do we measure progress towards our goal

Goal: Find good function f in function space $\mathcal{F}$

How: Minimize the expected risk of f $R(f) = E_Z[L(z, f)] = \int_Z L(z, f)p(z) dz$

However $p(z)$ *unknown!*

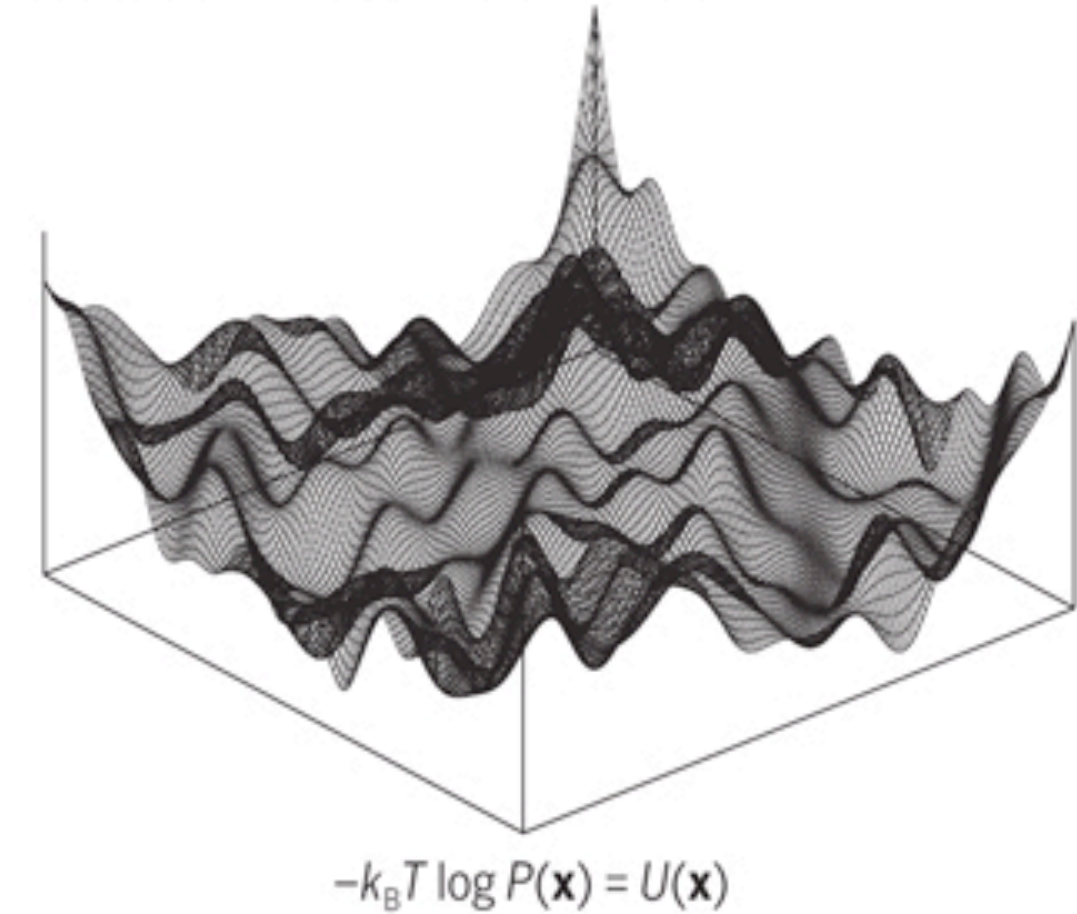


How do we measure progress towards our goal

Goal: Find good function f in function space $\mathcal{F}$

How: Minimize the expected risk of f $R(f) = E_Z[L(z, f)] = \int_Z L(z, f)p(z) dz$

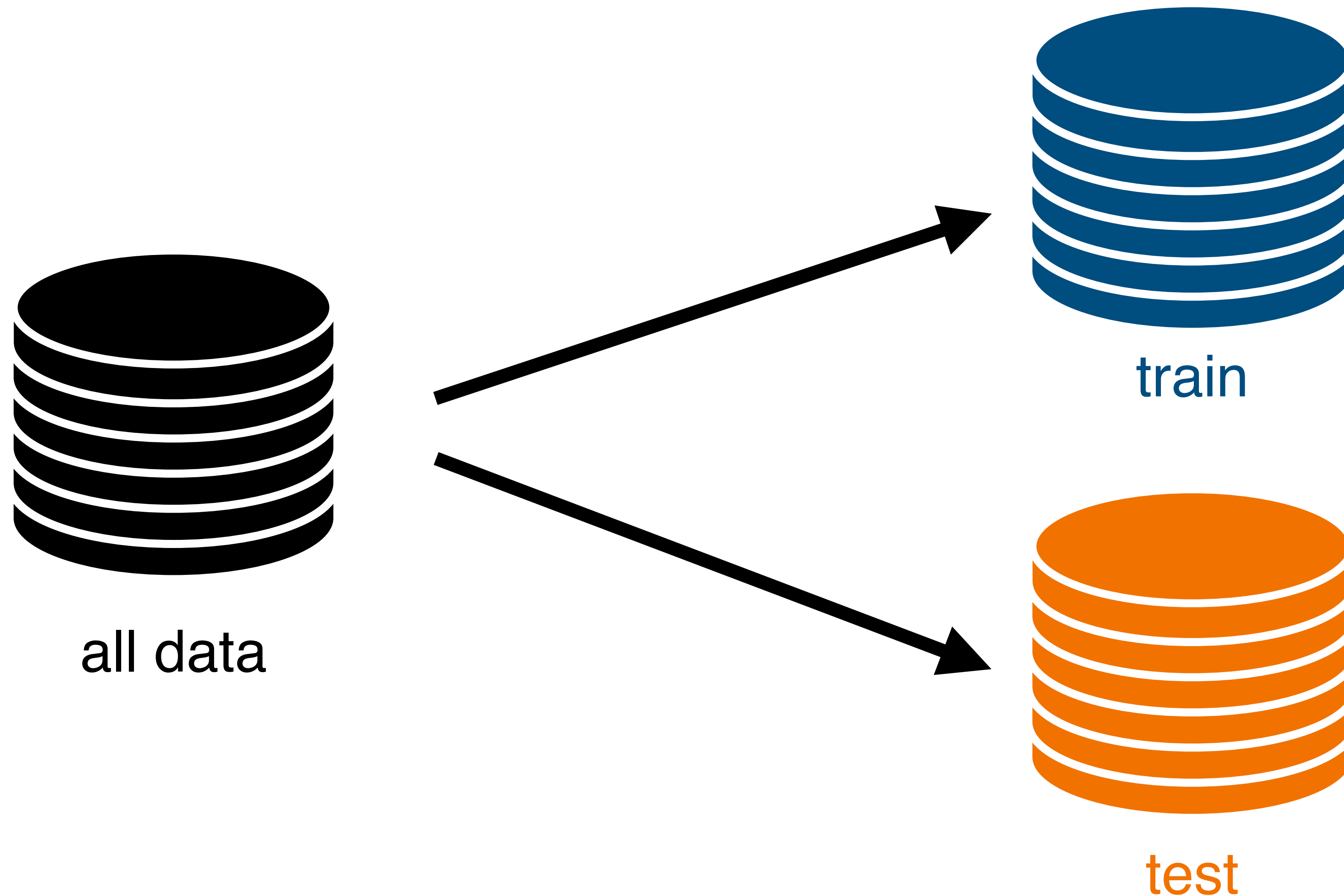
However $p(z)$ *unknown!*



Fix: Empirical Risk $\hat{R}(f) = \frac{1}{n} \sum_{i=1}^n L(z_i, f)$ Given a finite amount of training data, derive a relation for an infinite domain!

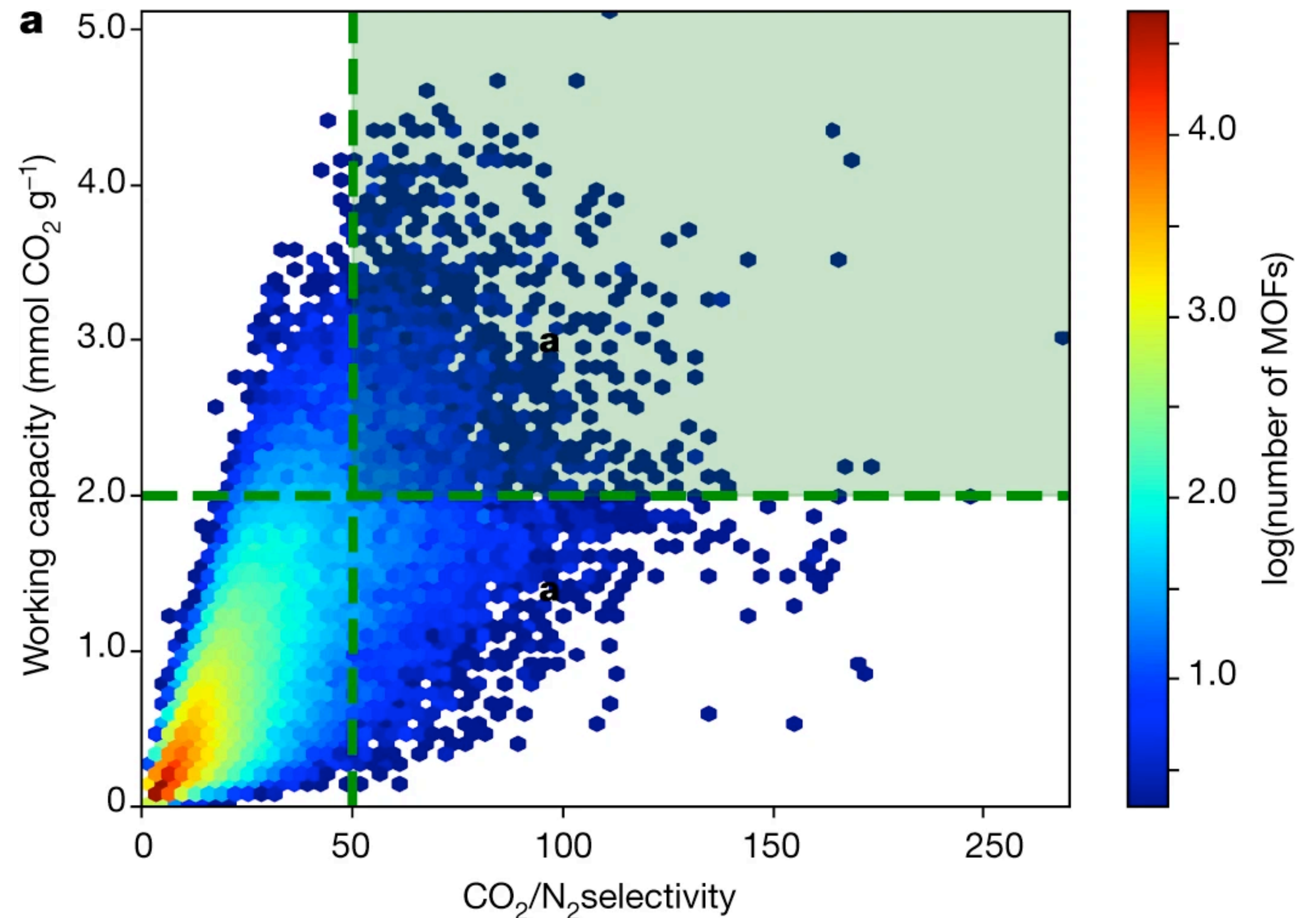
Selecting the datasets over which to compute the empirical risk is a key challenge!

First step: Put aside some test data you do not look at

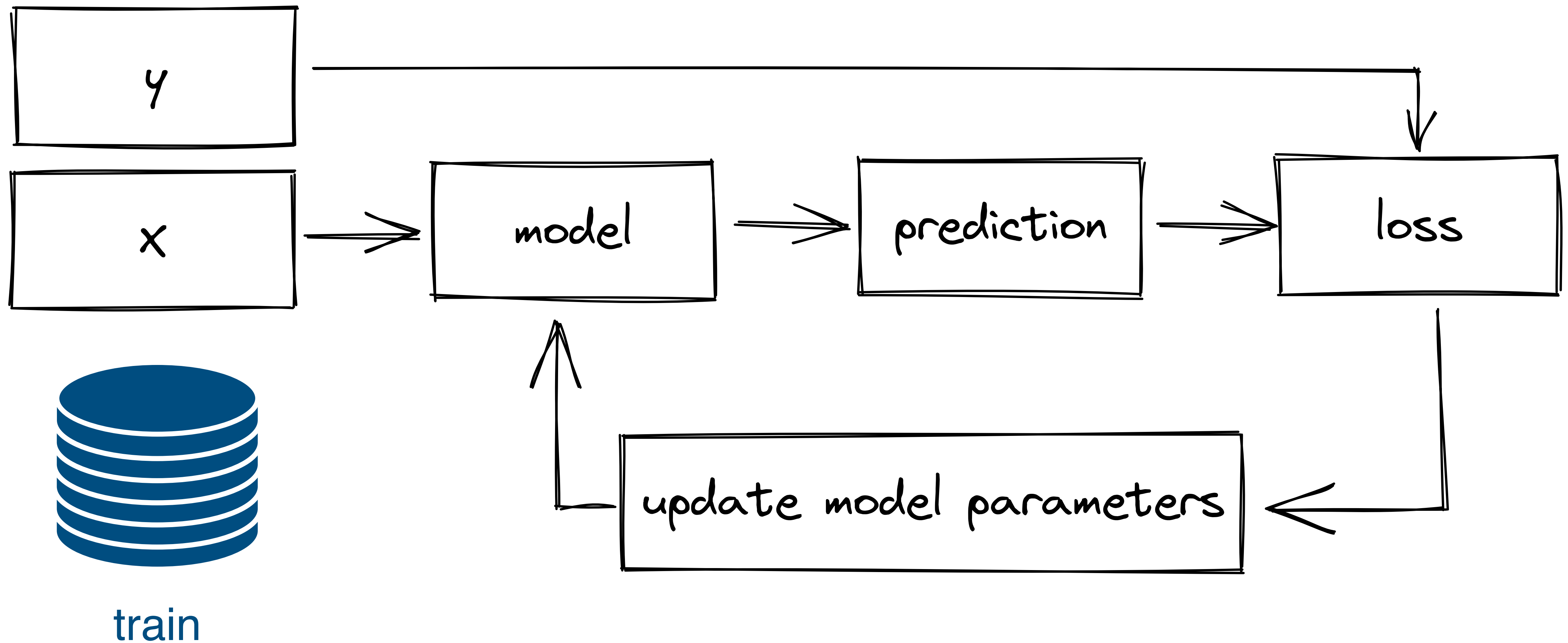


First step: Put aside some test data you do not look at

- A random sample from such a datasets might contain no or little top performing materials.
- Stratification can be an important technique.



Choose trainable weights that minimize empirical risk



2. Training a basic model

Learning Objectives:

- Supervised ML workflow
- Empirical risk estimate
- Test data

3. (Linear) Regression

Learning Objectives:

- Matrix formulation of linear regression
- Recasting non-linear regression in matrix notation by pre-computing features
- Gradient descent

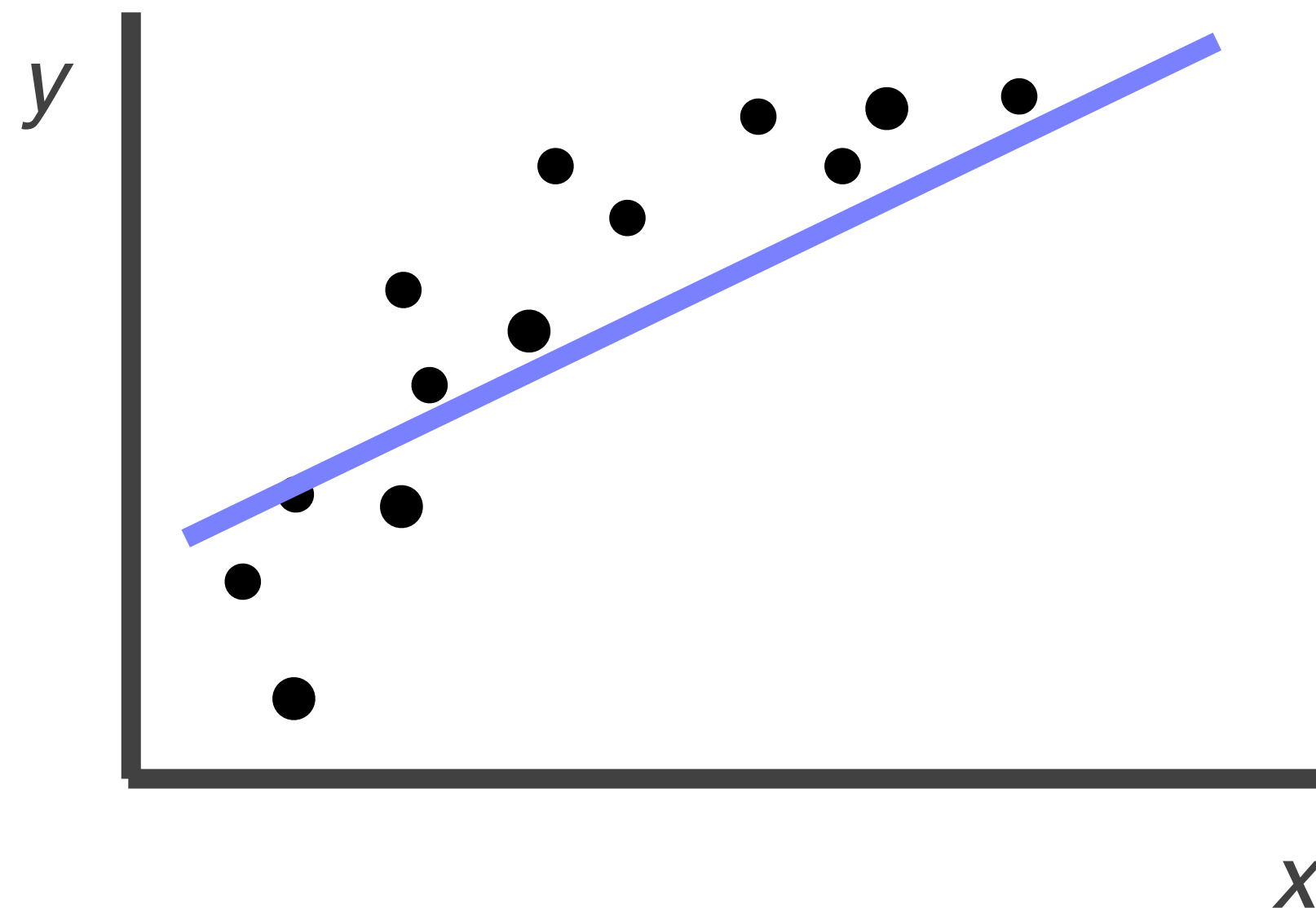
Simple linear regression

One feature

One point

$$y^* \approx f(x^*) = wx^* + b$$

All points



Simple linear regression

One feature

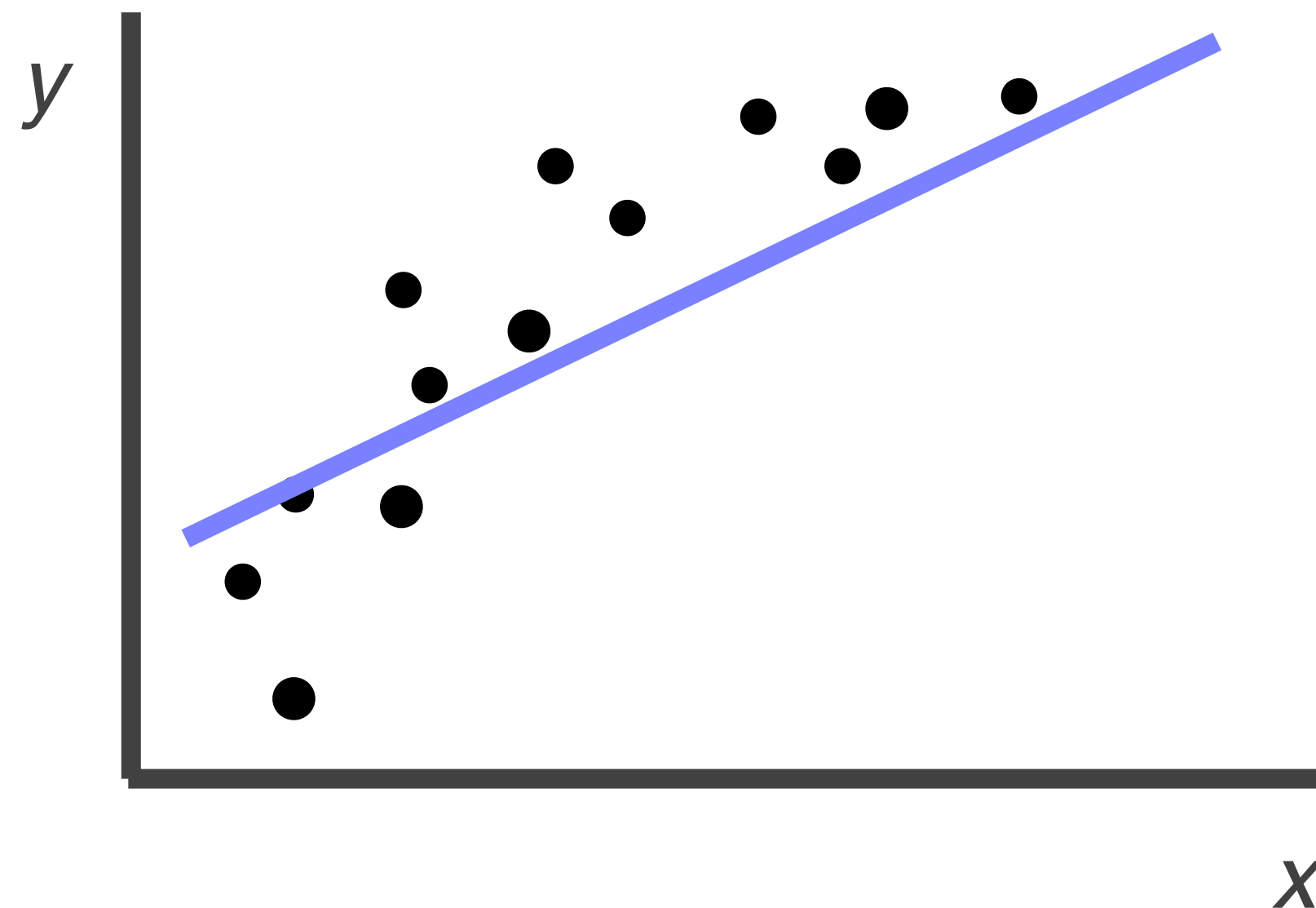
One point

$$y^* \approx f(x^*) = wx^* + b$$

alternative formulation

$$f(x^*) = x^*w = \begin{bmatrix} 1 & x^* \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

All points



Simple linear regression

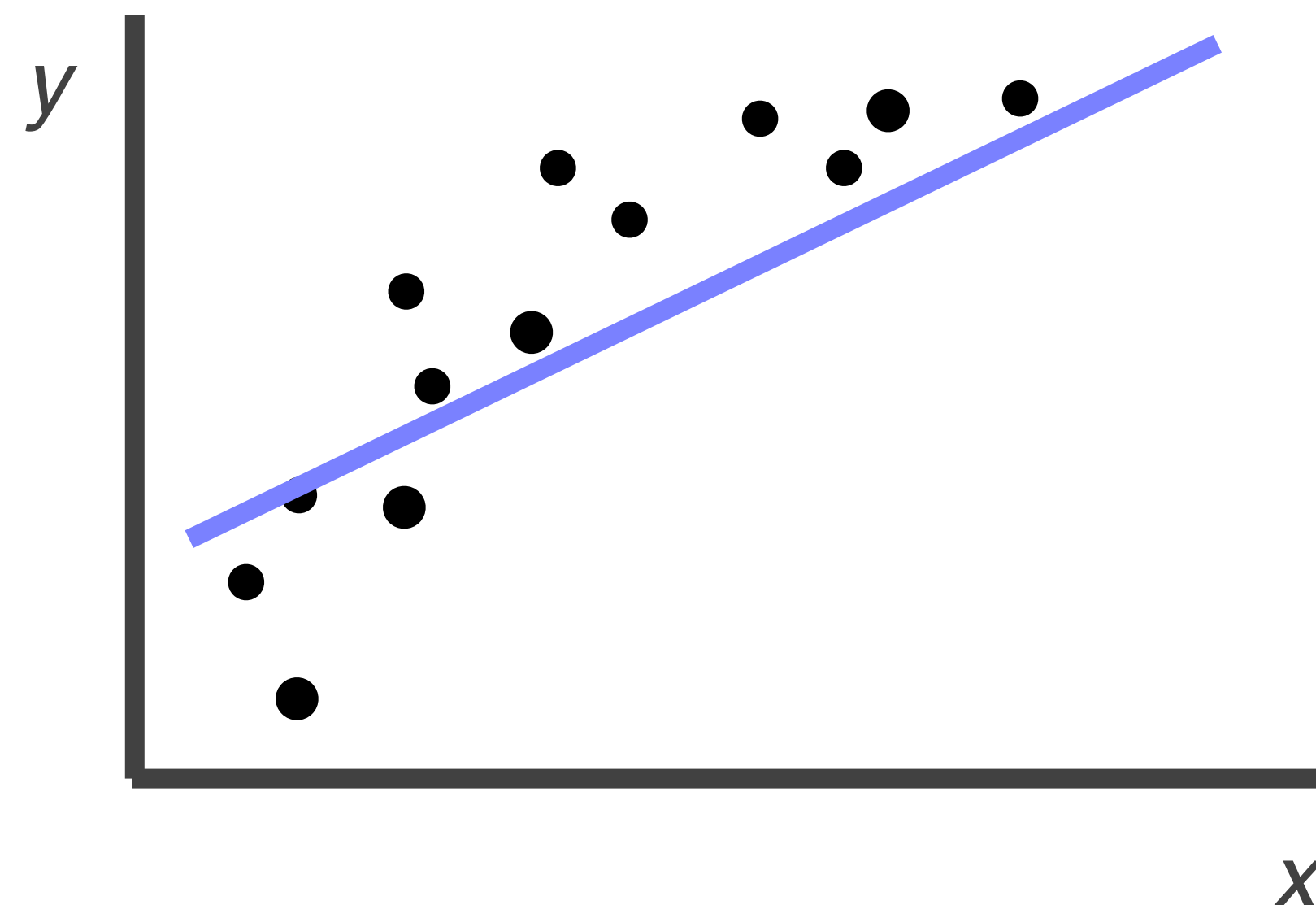
One feature

One point

$$y^* \approx f(x^*) = wx^* + b$$

alternative formulation

$$f(x^*) = x^*w = \begin{bmatrix} 1 & x^* \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$



All points

$$f(\mathbf{X}) = w\mathbf{X} + b$$

$$f(\mathbf{X}) = Xw = \begin{bmatrix} 1 & x^1 \\ \vdots & \vdots \\ 1 & x^m \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Simple linear regression

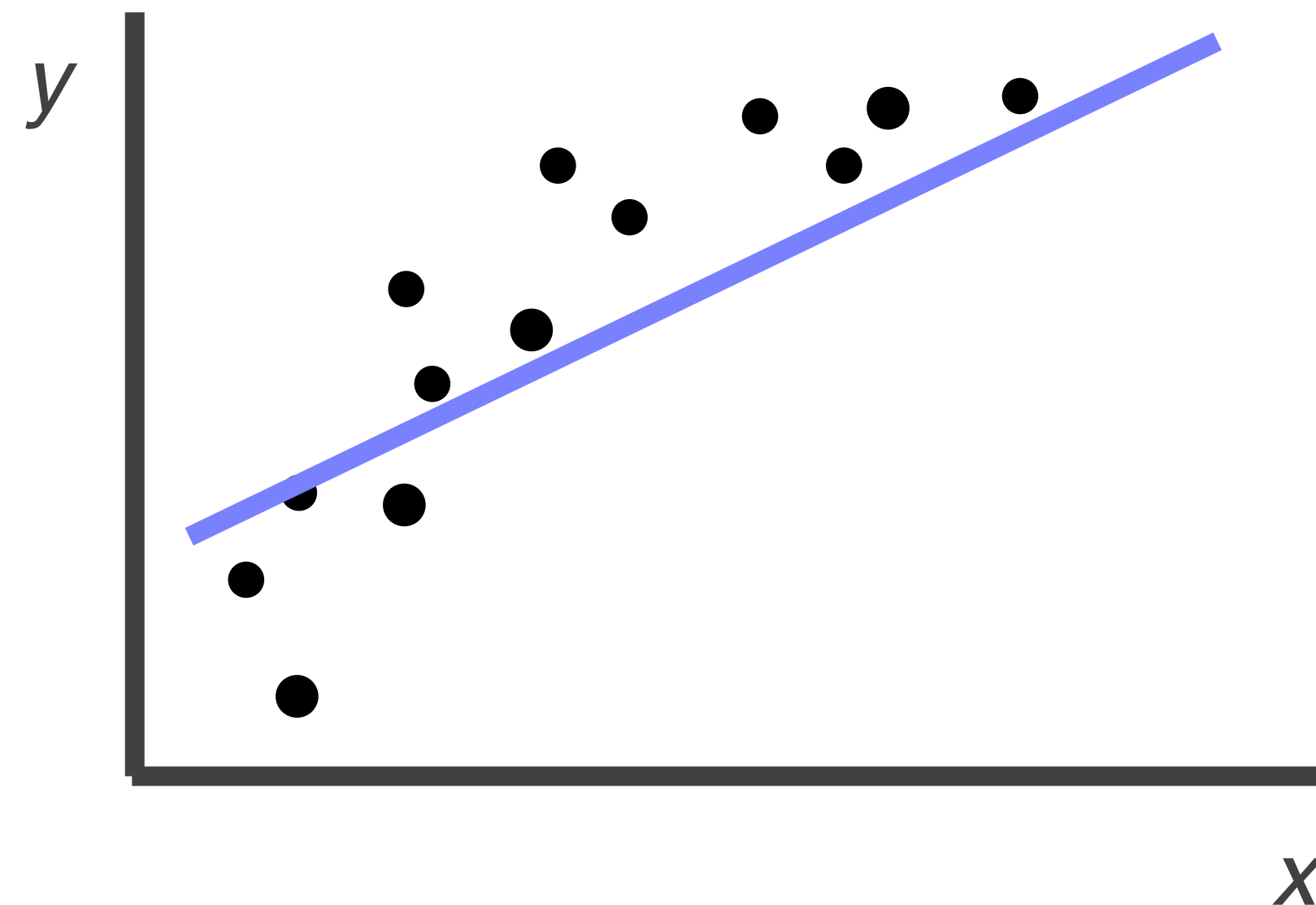
One feature

One point

$$y^* \approx f(x^*) = wx^* + b$$

alternative formulation

$$f(x^*) = x^*w = \begin{bmatrix} 1 & x^* \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$



All points

$$f(\mathbf{X}) = w\mathbf{X} + b$$

$$f(\mathbf{X}) = Xw = \begin{bmatrix} 1 & x^1 \\ \vdots & \vdots \\ 1 & x^m \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$



```
def linear_model(x, w, b):  
    return np.dot(x, w) + b
```

Multiple linear regression

Multiple features: One weight per feature

Single data point

All data points

Multiple linear regression

Multiple features: One weight per feature

Single data point

$$f(\mathbf{X}^*) = \mathbf{X}^* \mathbf{w} = \begin{bmatrix} 1 & x_0^* & x_1^* & \dots & x_p^* \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{bmatrix}$$

All data points

Multiple linear regression

Multiple features: One weight per feature

Single data point

$$f(\mathbf{X}^*) = \mathbf{X}^* \mathbf{w} = \begin{bmatrix} 1 & x_0^* & x_1^* & \dots & x_p^* \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{bmatrix}$$

All data points

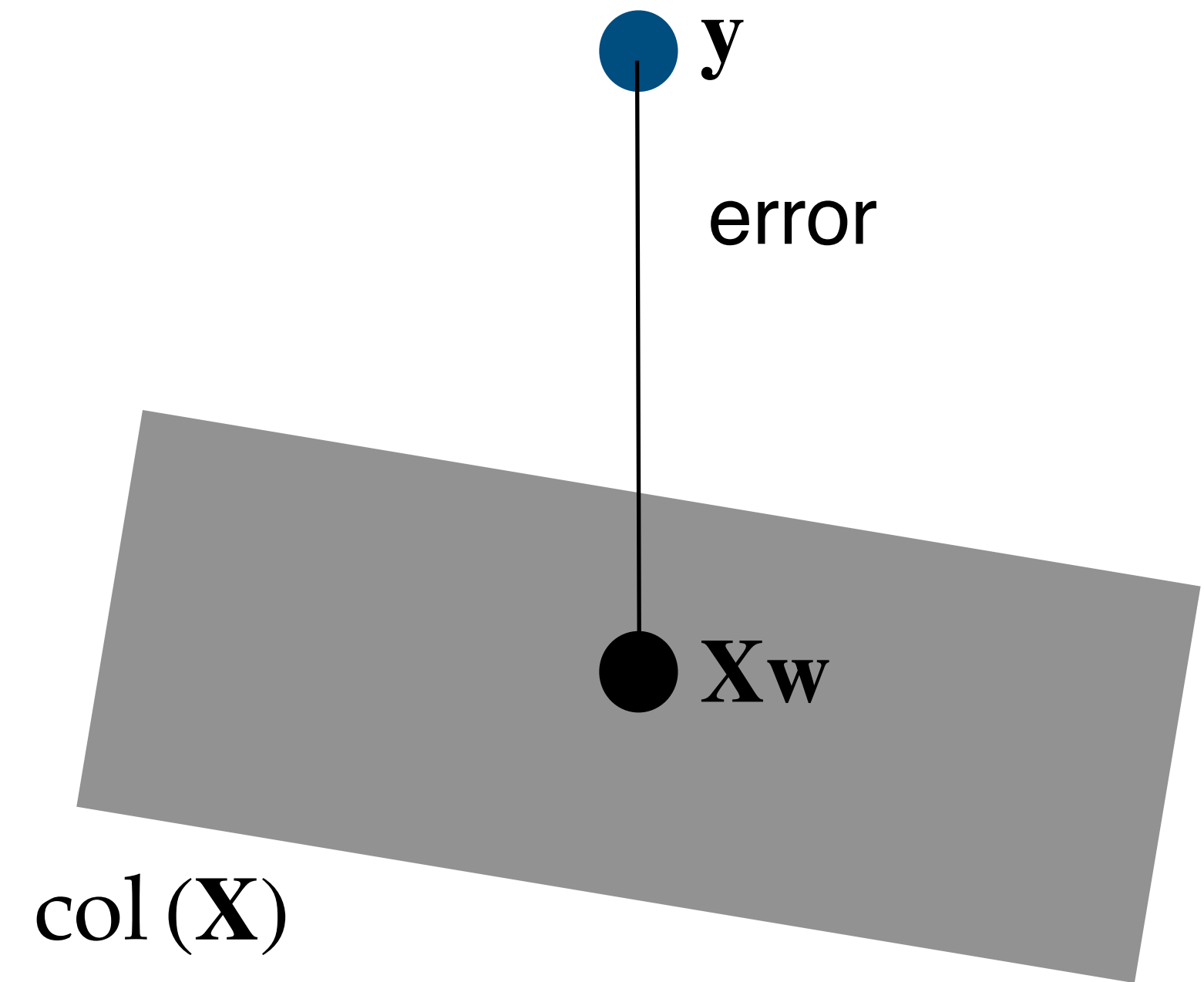
$$f(\mathbf{X}) = \mathbf{X} \mathbf{w} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \dots & x_p^1 \\ \vdots & \vdots & & \ddots & \\ 1 & x_1^n & x_2^n & \dots & x_p^n \end{bmatrix} \begin{bmatrix} w_0 \\ \vdots \\ w_p \end{bmatrix}$$

How do we find the adjustable parameters w ?

For a few models: Closed-form solution

Linear model: “normal equations”

Project into the column space of X



orthogonal projection will be lowest error

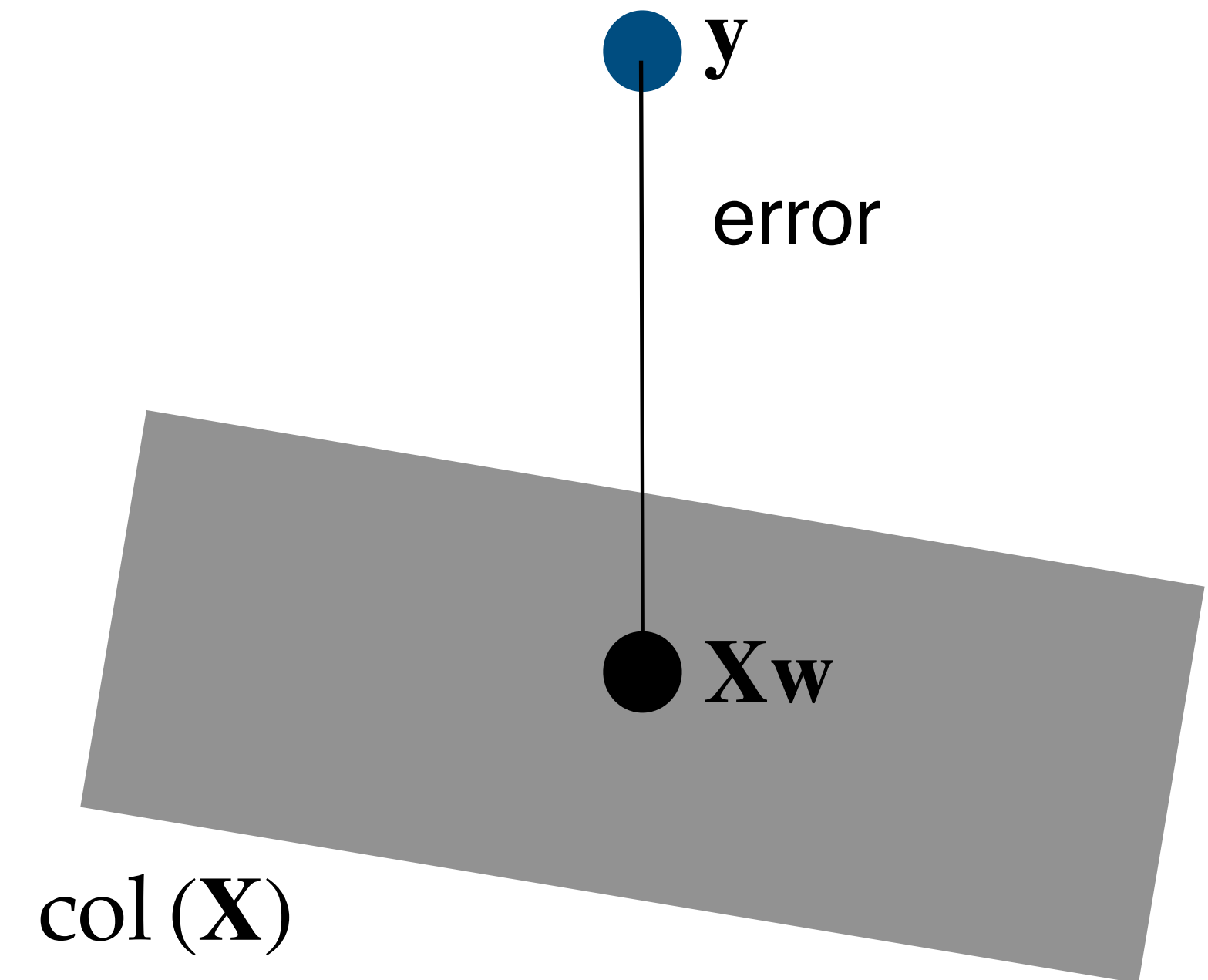
Note: There might be issues with invertability, e.g., if $p > n$ (*only square matrices are*)

How do we find the adjustable parameters w ?

For a few models: Closed-form solution

Linear model: “normal equations”

Project into the column space of X



orthogonal projection will be lowest error

$$X^T (y - Xw) = 0$$

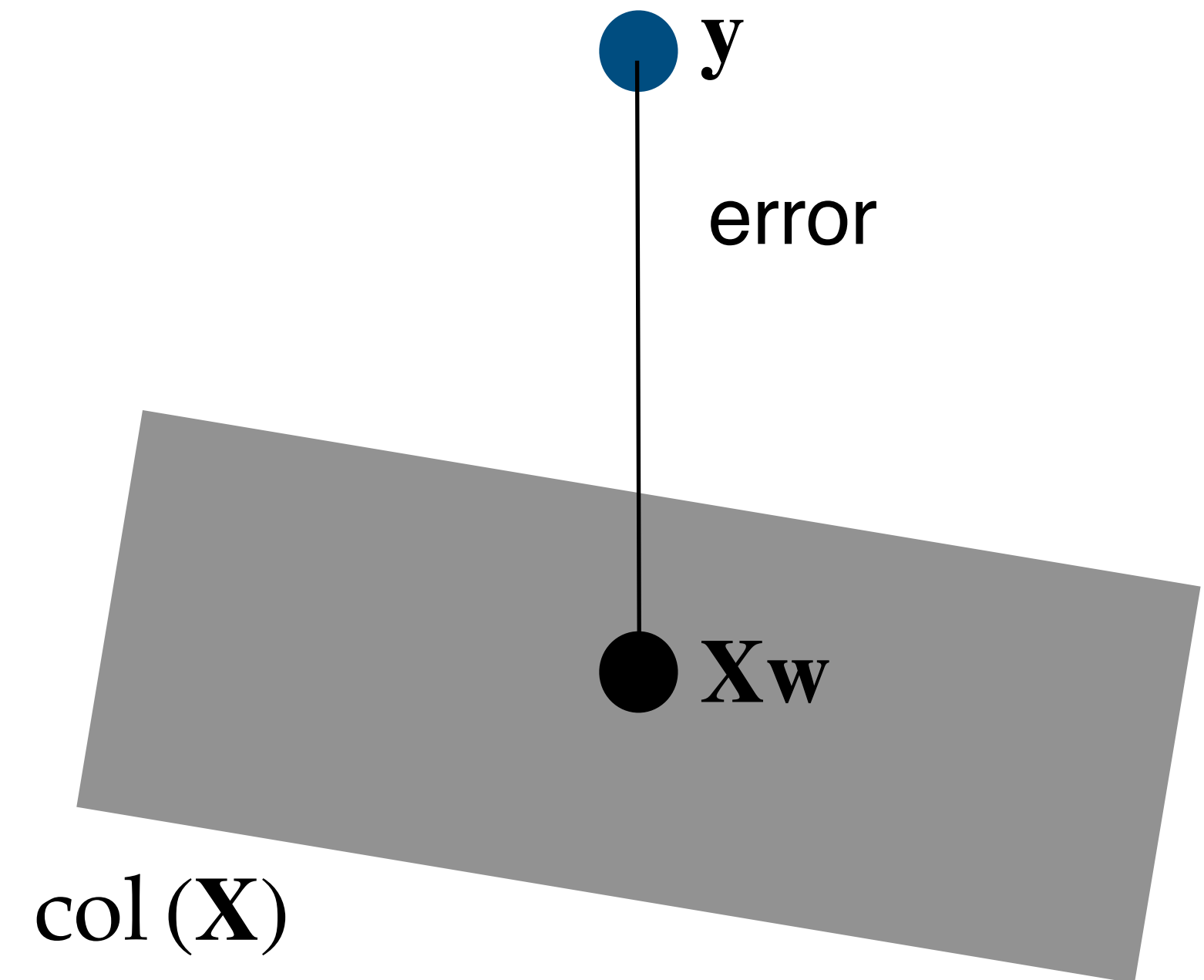
Note: There might be issues with invertability, e.g., if $p > n$ (*only square matrices are*)

How do we find the adjustable parameters w ?

For a few models: Closed-form solution

Linear model: “normal equations”

Project into the column space of X



orthogonal projection will be lowest error

$$X^T (y - Xw) = 0$$

$$X^T y = X^T X w$$

Note: There might be issues with invertability, e.g., if $p > n$ (*only square matrices are*)

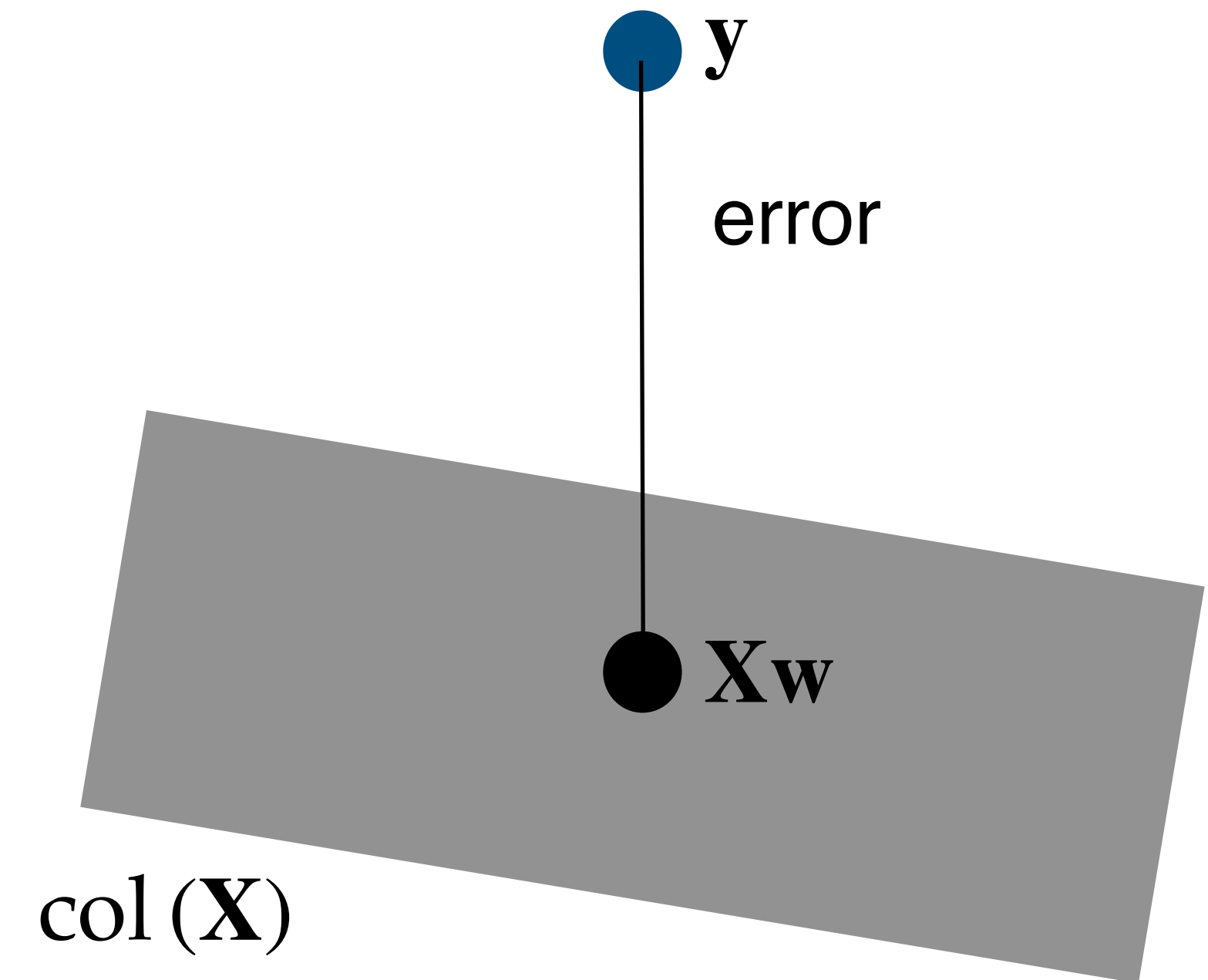
How do we find the adjustable parameters w ?

For a few models: Closed-form solution

Linear model: “normal equations”

Project into the column space of X

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$



```
def closed_form(X, y):  
    return np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
```

orthogonal projection will be lowest error

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

$$\mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \mathbf{w}$$

Note: There might be issues with invertability, e.g., if $p > n$ (only square matrices are)

How do we find the adjustable parameters w ?

For a *all* models: Numerical optimization

Step 1: Initialize and define a loss

```
# initialize our paramaters
w = np.random.normal(size=feature_dim)
b = 0.0

# define loss
def loss(y, labels):
    return np.mean((y - labels) ** 2)

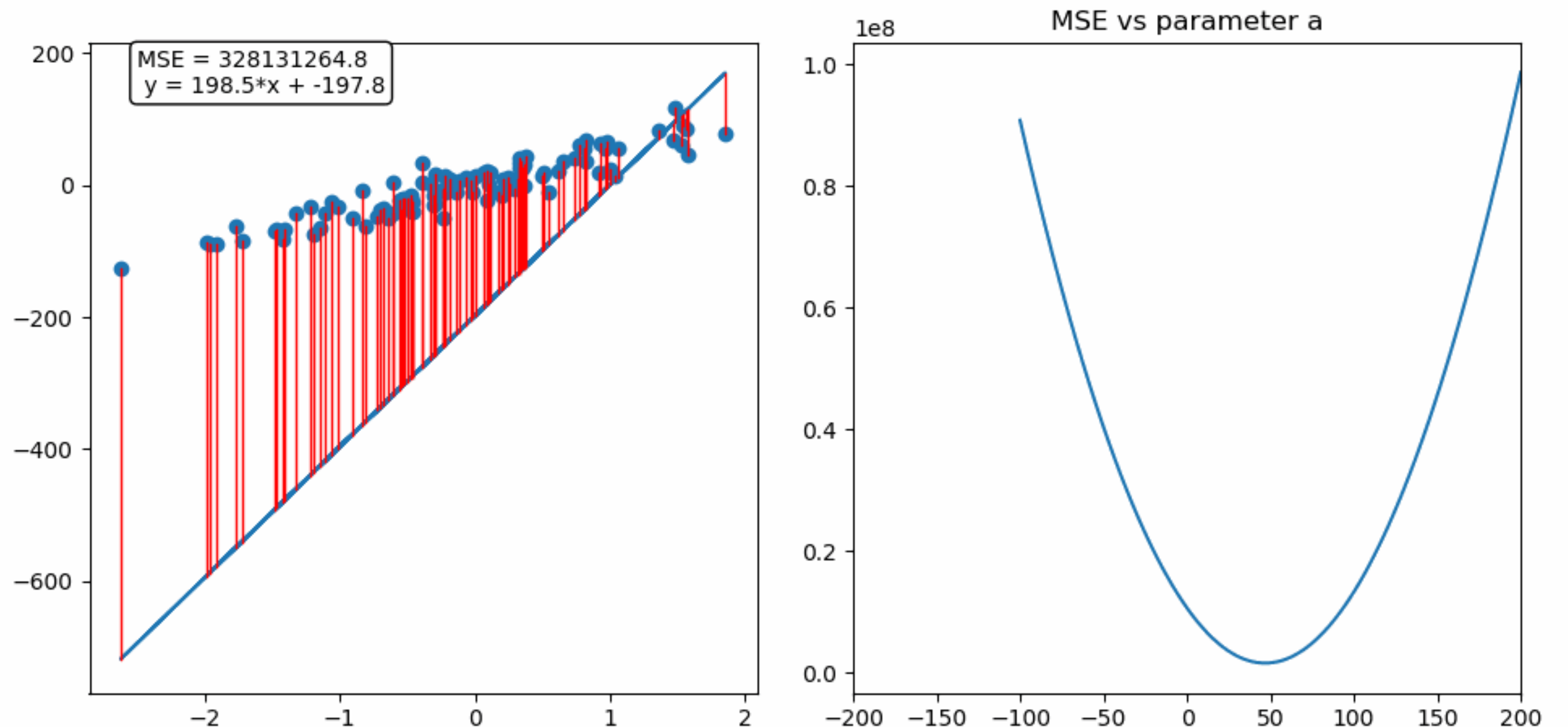
# test it out
y = linear_model(features, w, b)
loss(y, labels)
```

How do we find the adjustable parameters?

For a *all* models: Numerical optimization (Gradient descent)

Step 2: Go downhill

Use the gradient to
decide how to tune
parameters

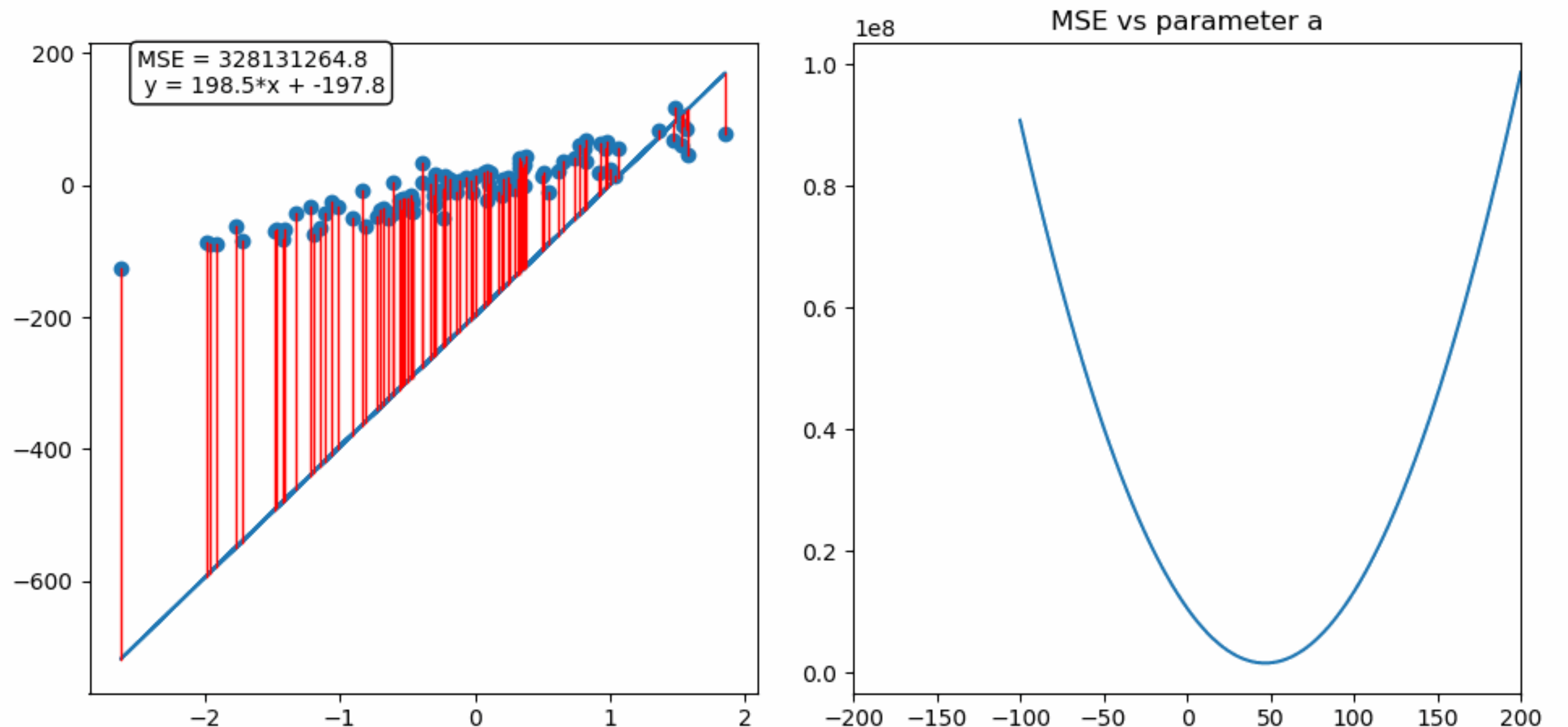


How do we find the adjustable parameters?

For a *all* models: Numerical optimization (Gradient descent)

Step 2: Go downhill

Use the gradient to
decide how to tune
parameters



How do we find the adjustable parameters?

For a *all* models: Numerical optimization

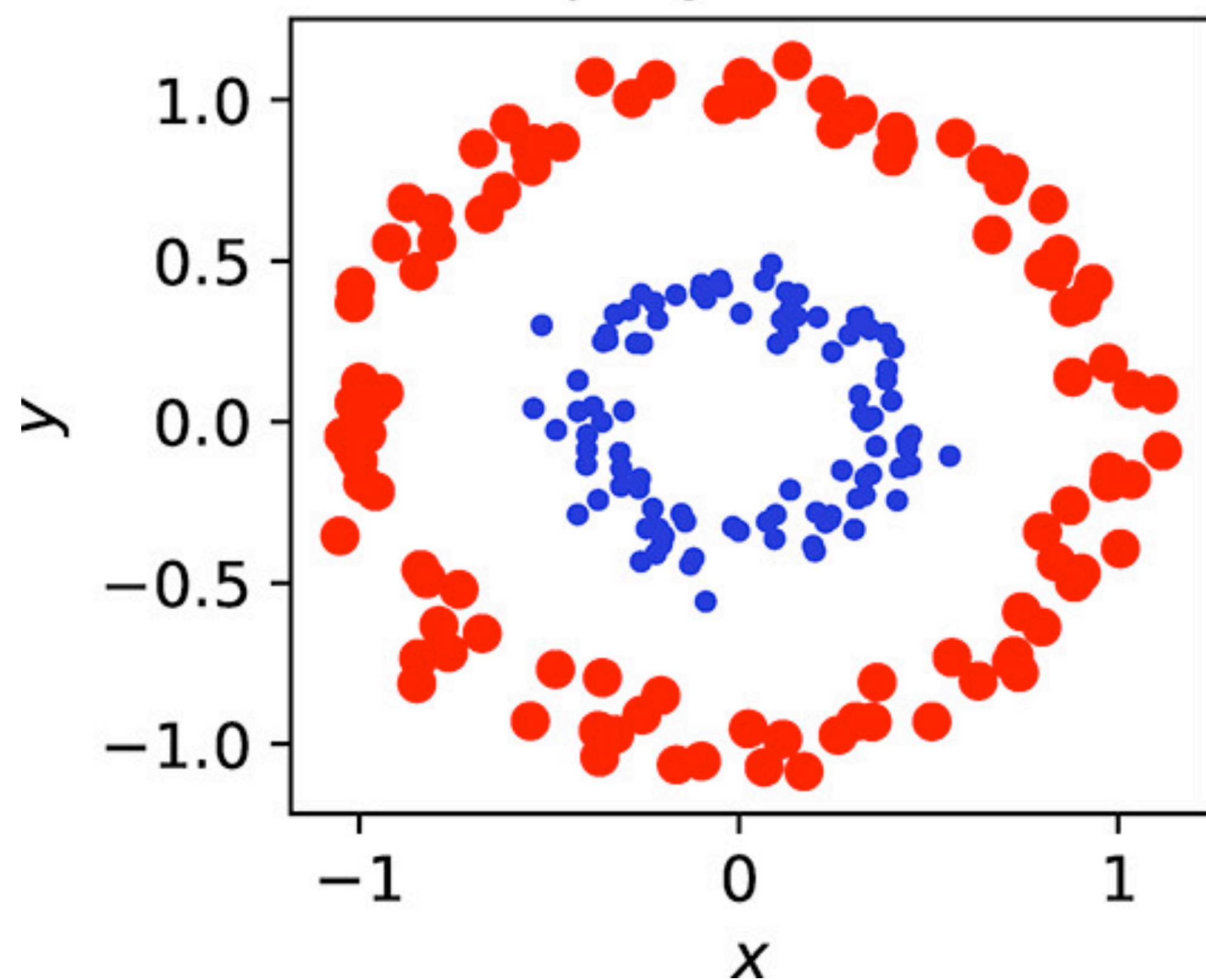
Step 2: Go downhill

Modern frameworks (e.g., jax)
or languages (e.g., Julia) can
automatically compute
gradients (autodiff)!

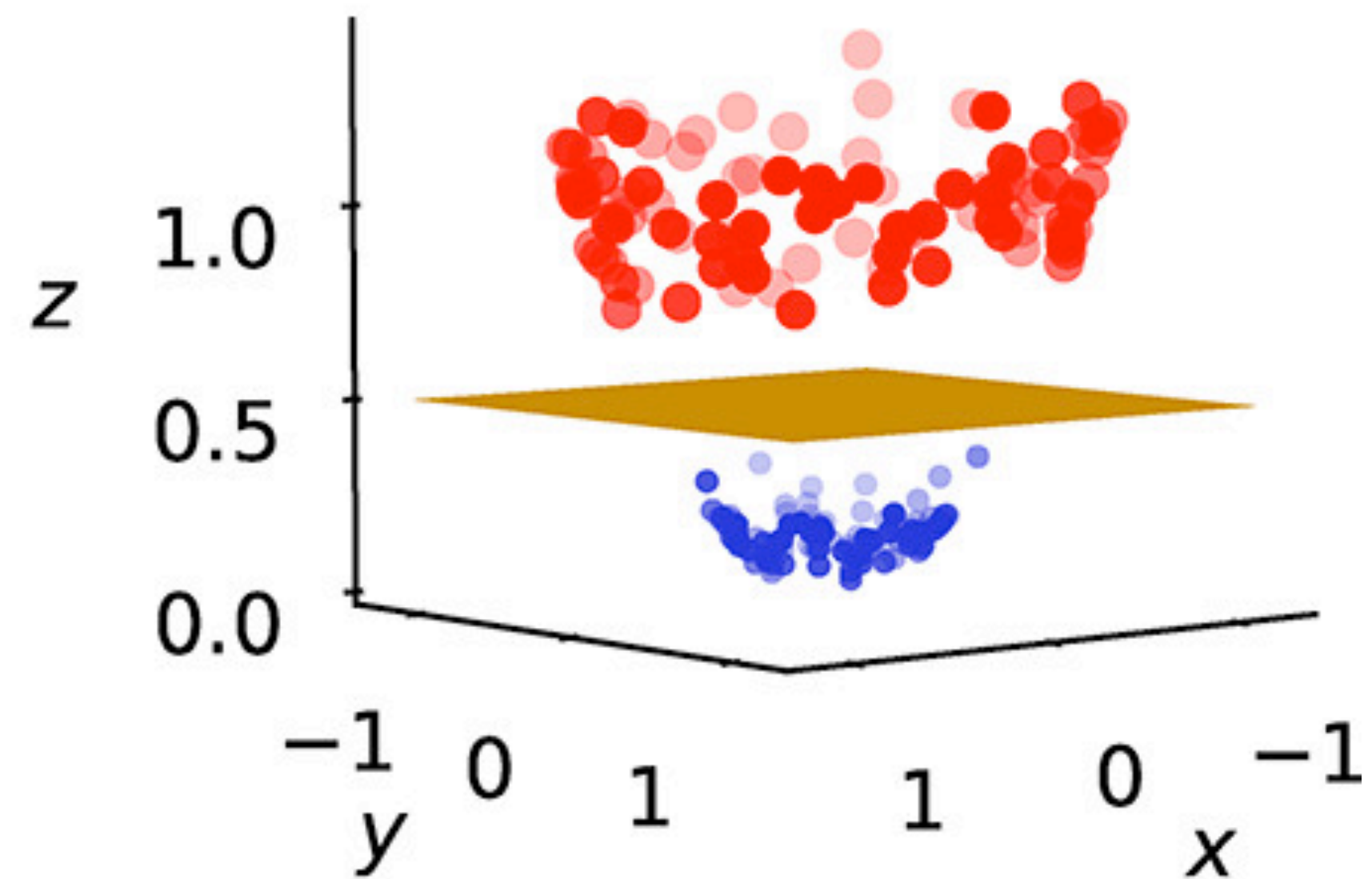
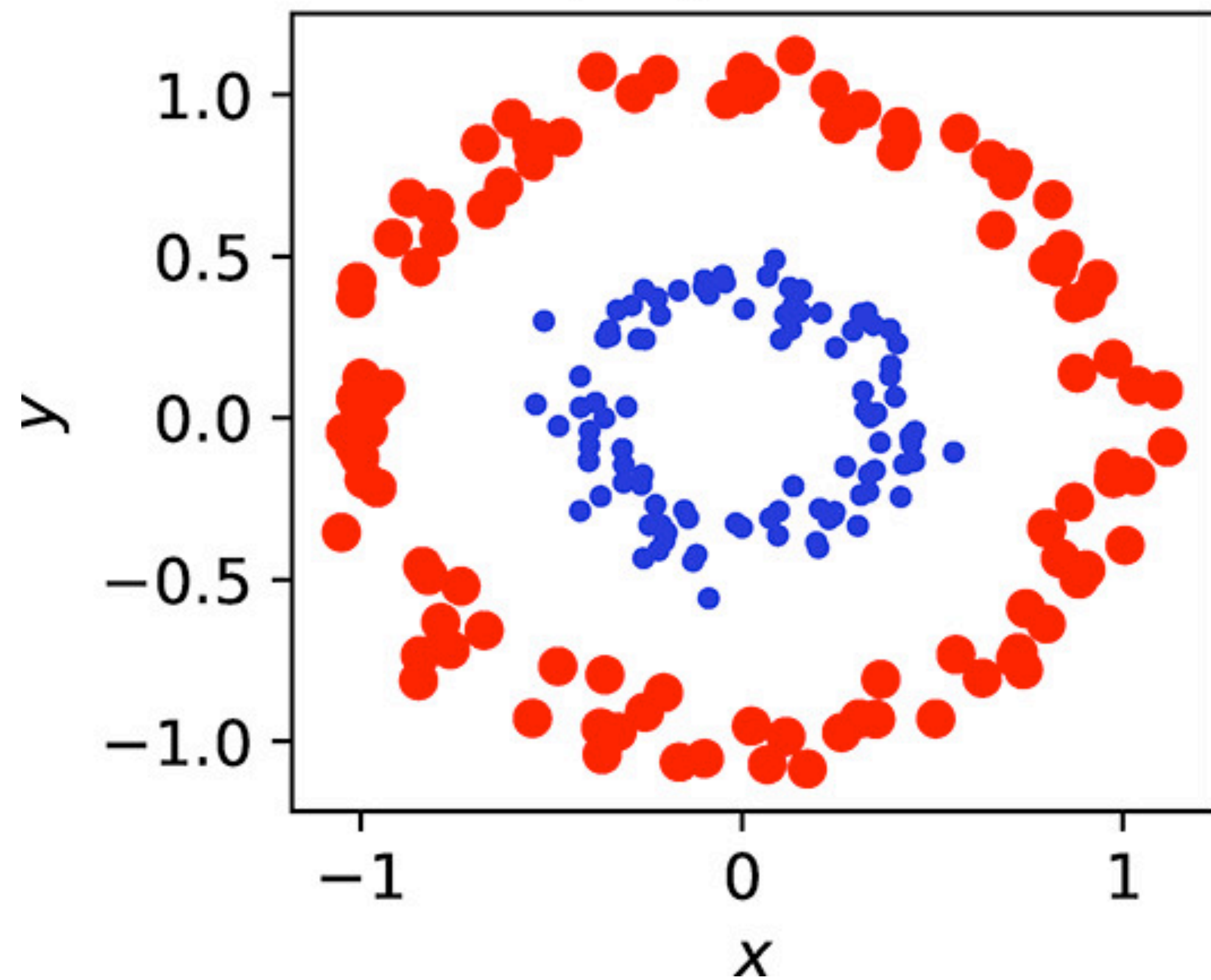
- Autodiff is *not* finite differences!
- No need to implement derivatives by hand!

```
def loss_wrapper(w, b, data):  
    features = data[0]  
    labels = data[1]  
    y = linear_model(features, w, b)  
    return loss(y, labels)  
  
loss_grad = jax.grad(loss_wrapper, (0, 1))  
  
data = (features, labels)  
  
for i in range(10):  
    grad = loss_grad(w, b, data)  
    w -= eta * grad[0]  
    b -= eta * grad[1]
```

How do we classify this data with a linear model?



How do we classify this data with a linear model?



By computing non-linear features!

$$[x_1^2 \quad x_2^2 \quad \sqrt{2}x_1x_2]$$

3. Regression

Learning Objectives:

- Matrix formulation of linear regression
- Recasting non-linear regression in matrix notation by pre-computing features
- Gradient descent

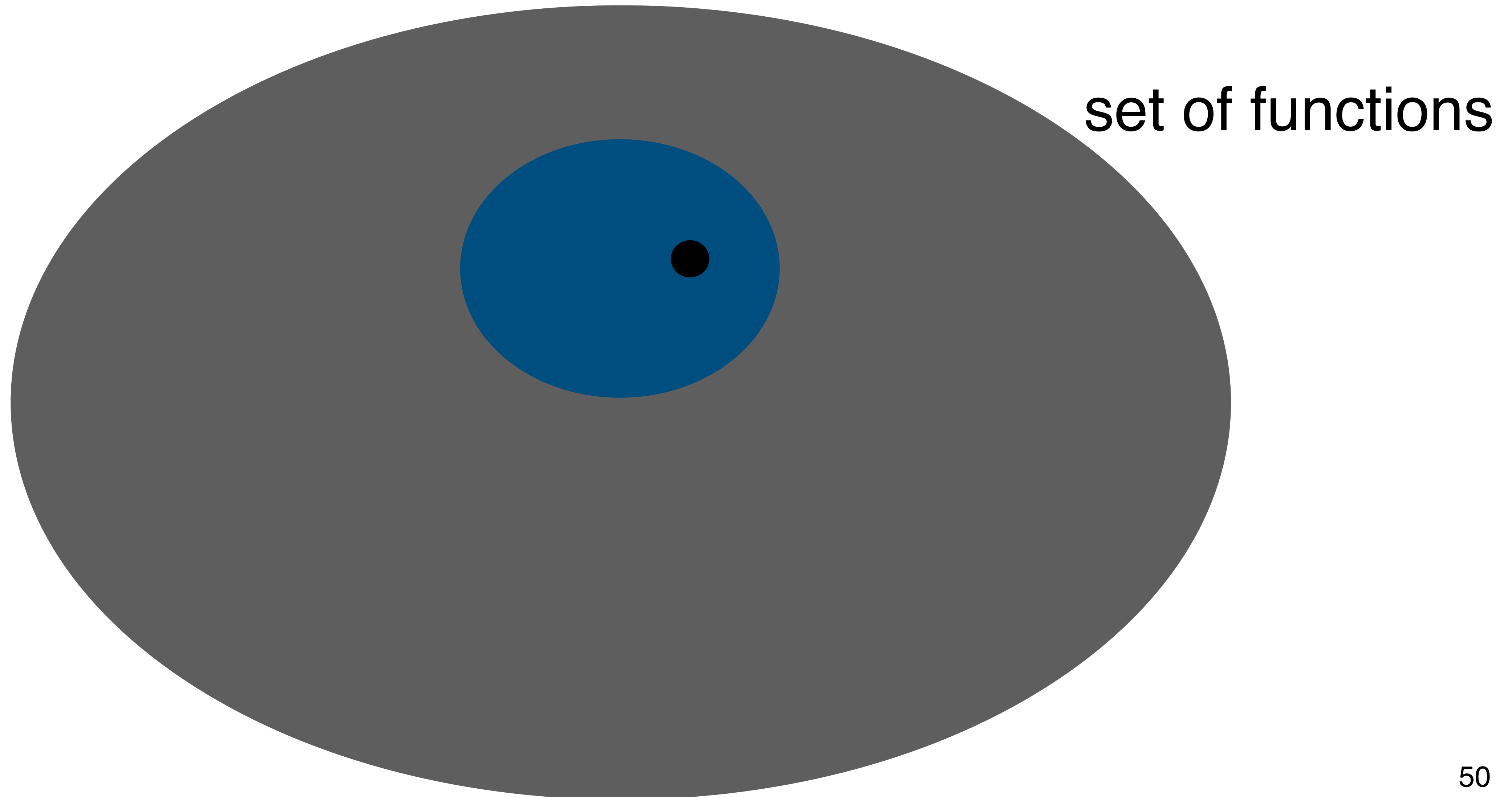
4. Bias-variance tradeoff

Learning Objectives:

- We can split the error into different contributions: bias, variance, irreducible
- We cannot reduce bias and variance at the same time

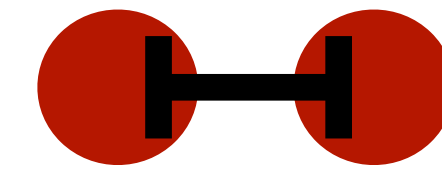
Error contributions

● ● optimal solution

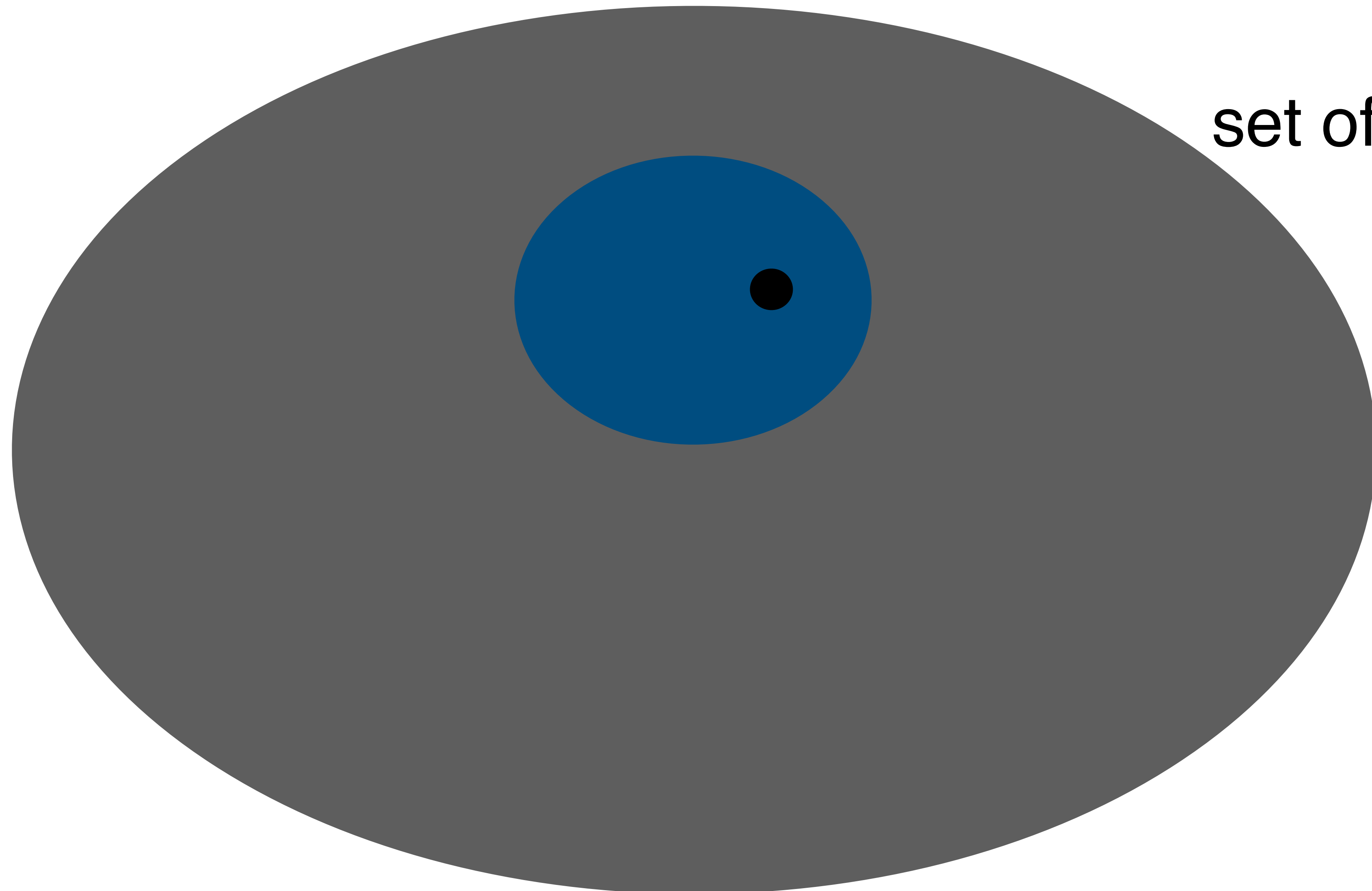


irreducible error

Error contributions



optimal solution



set of functions

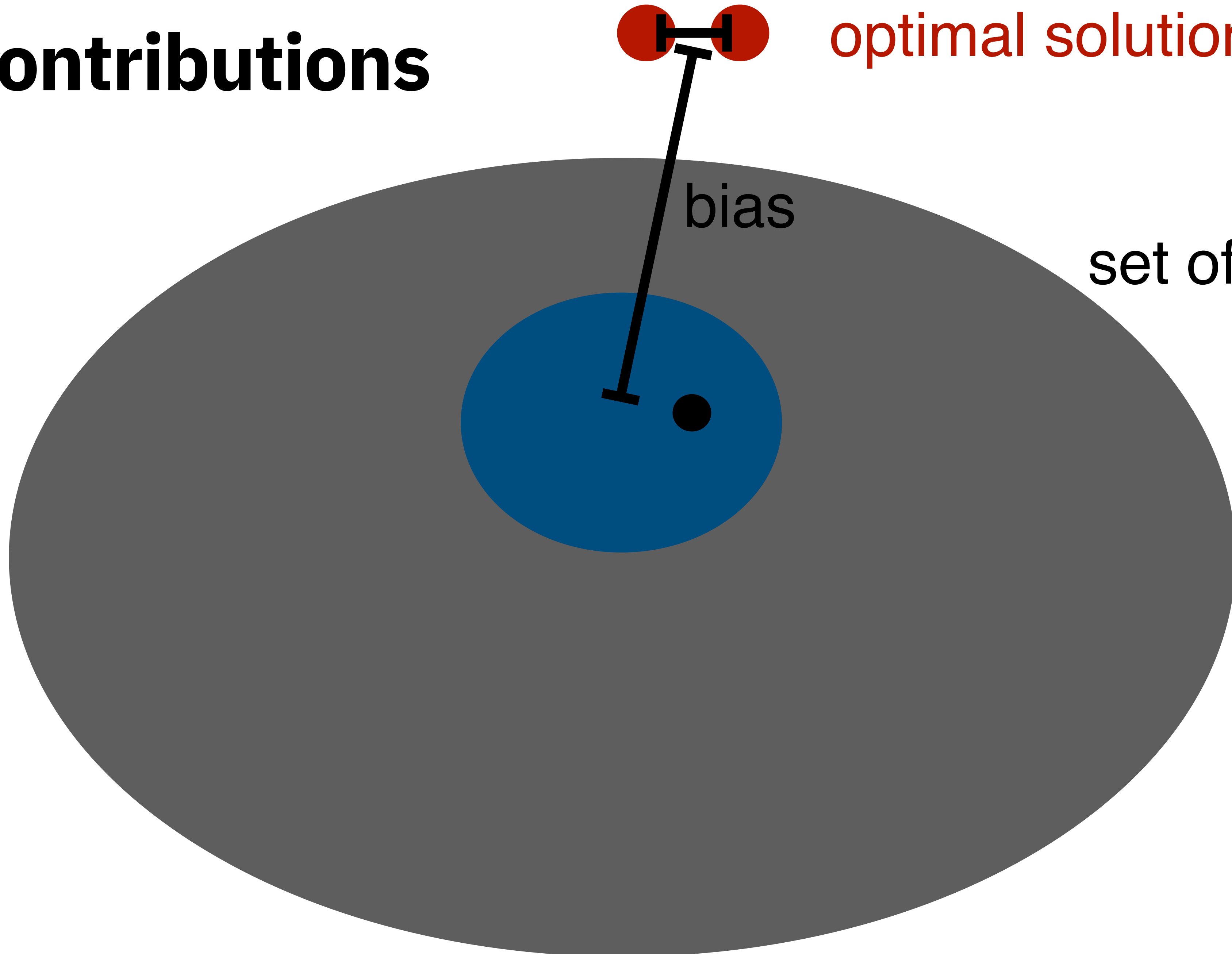
irreducible error

Error contributions

optimal solution

bias

set of functions



irreducible error

Error contributions

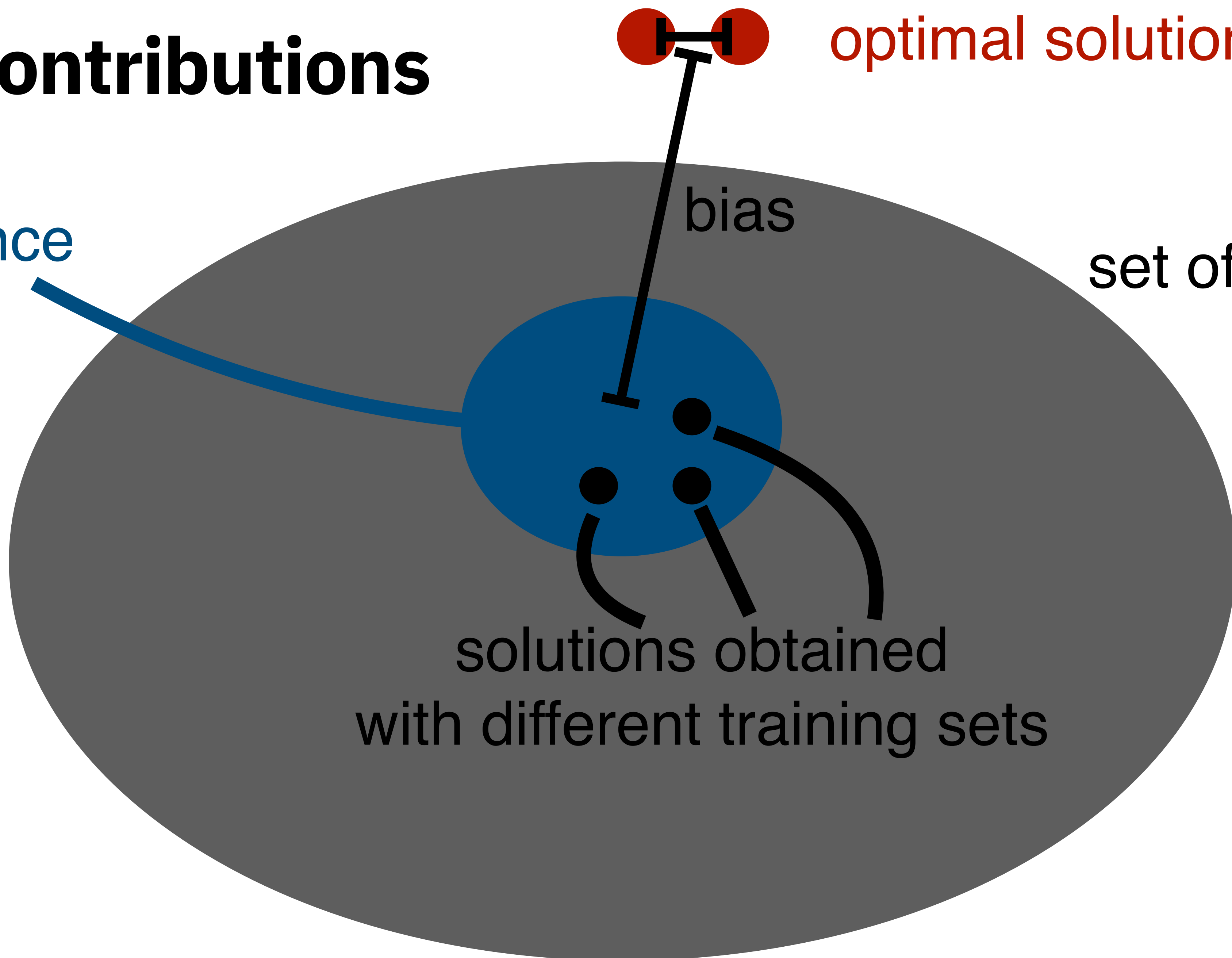
optimal solution

variance

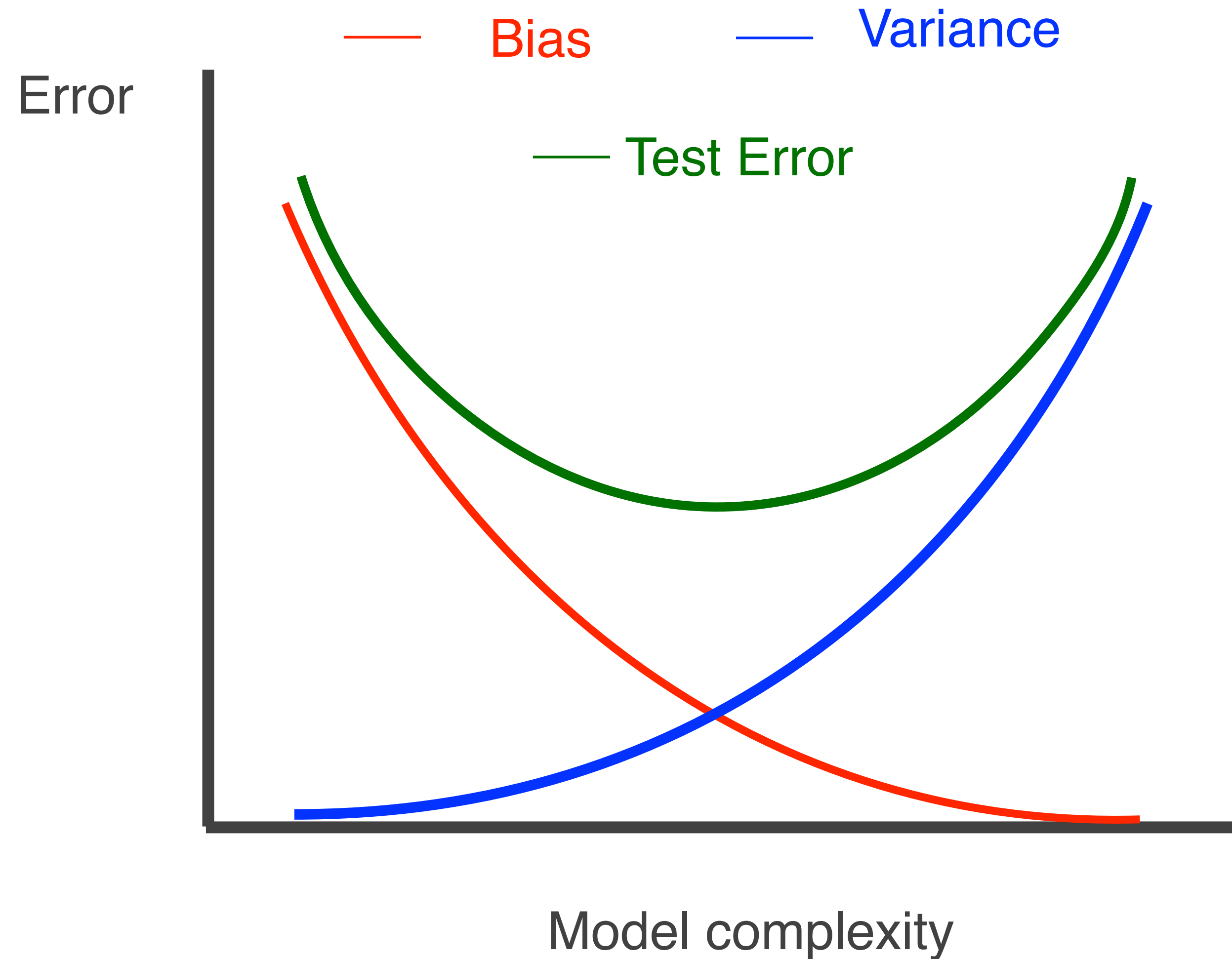
bias

set of functions

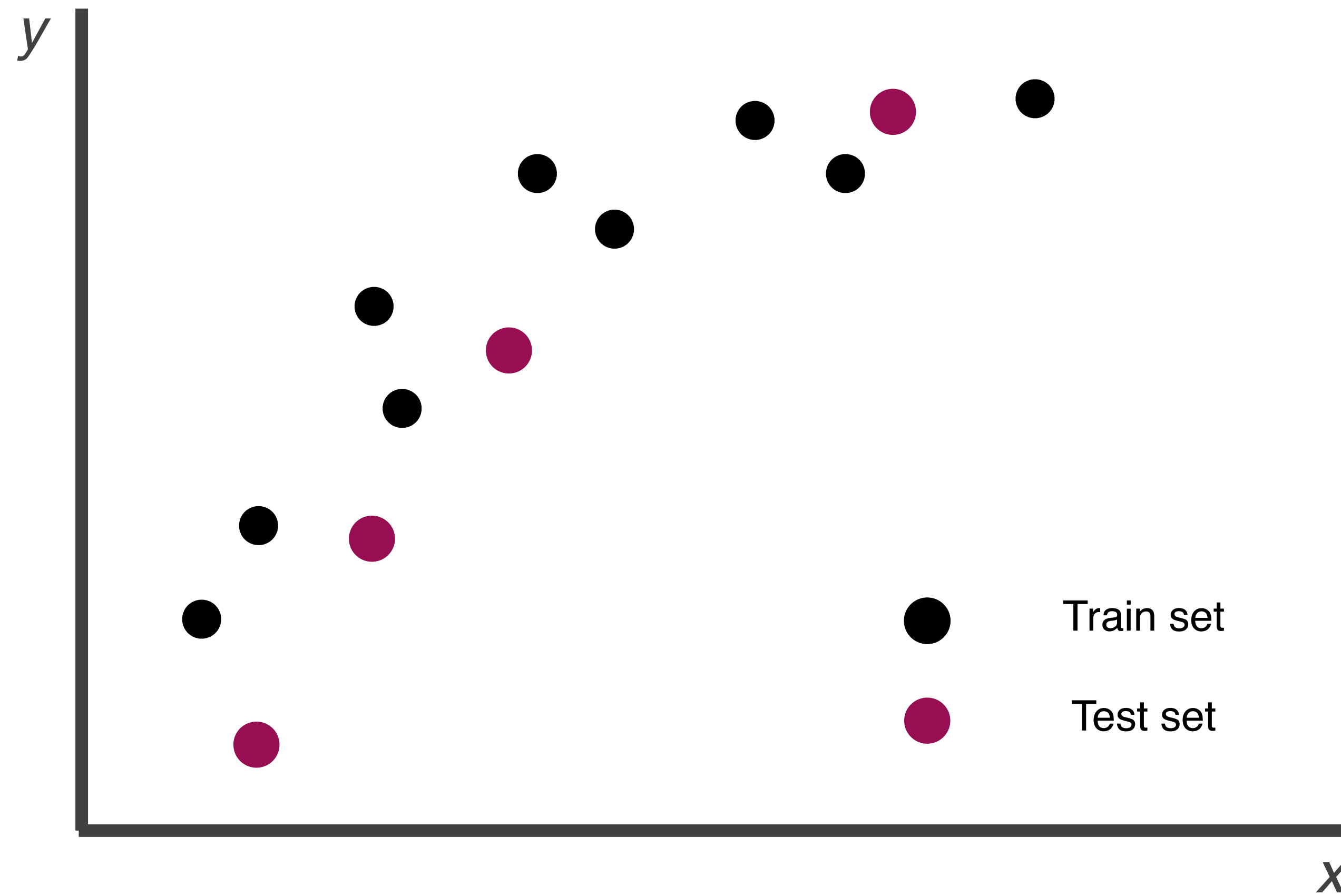
solutions obtained
with different training sets



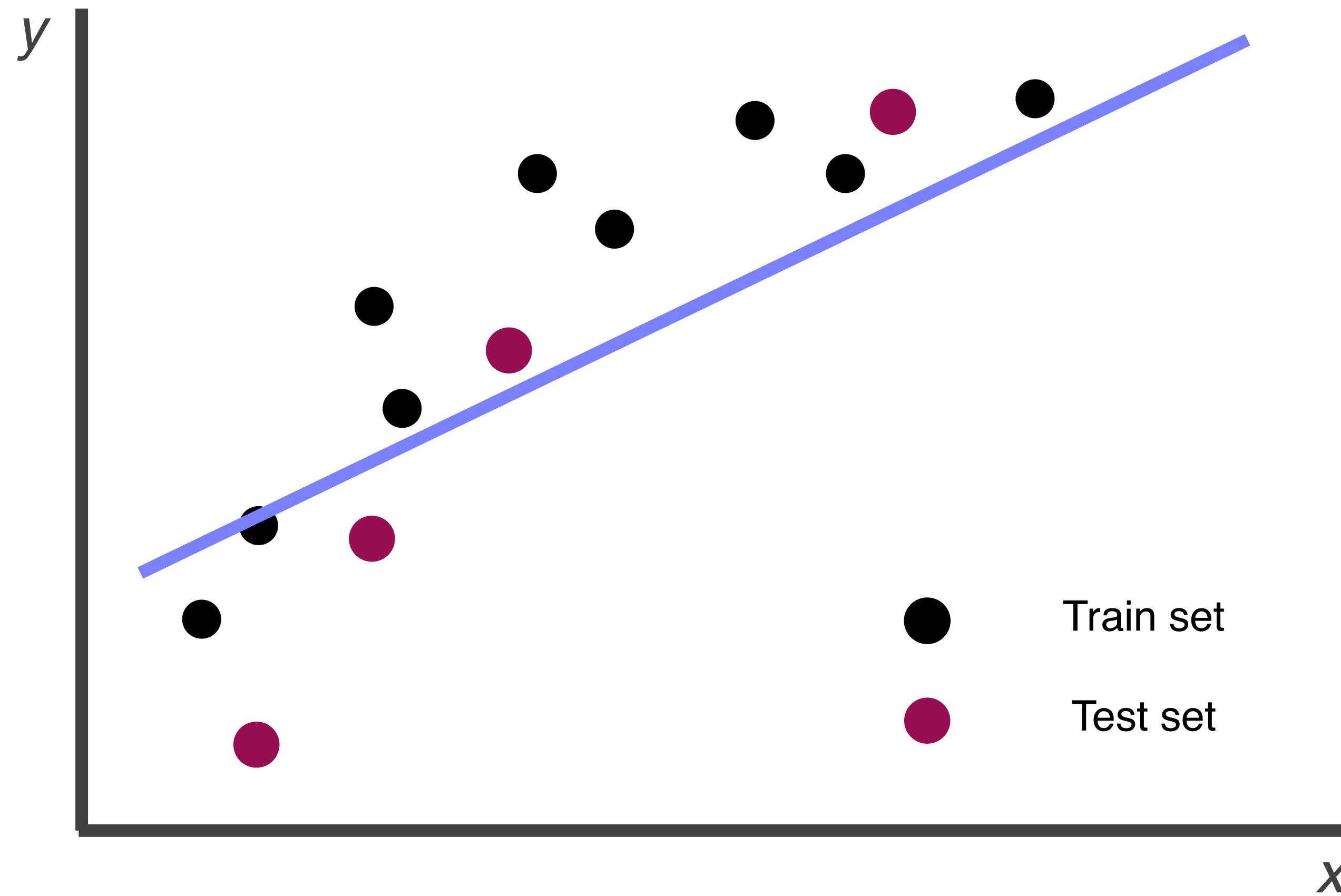
How do these error terms depend on the model complexity?



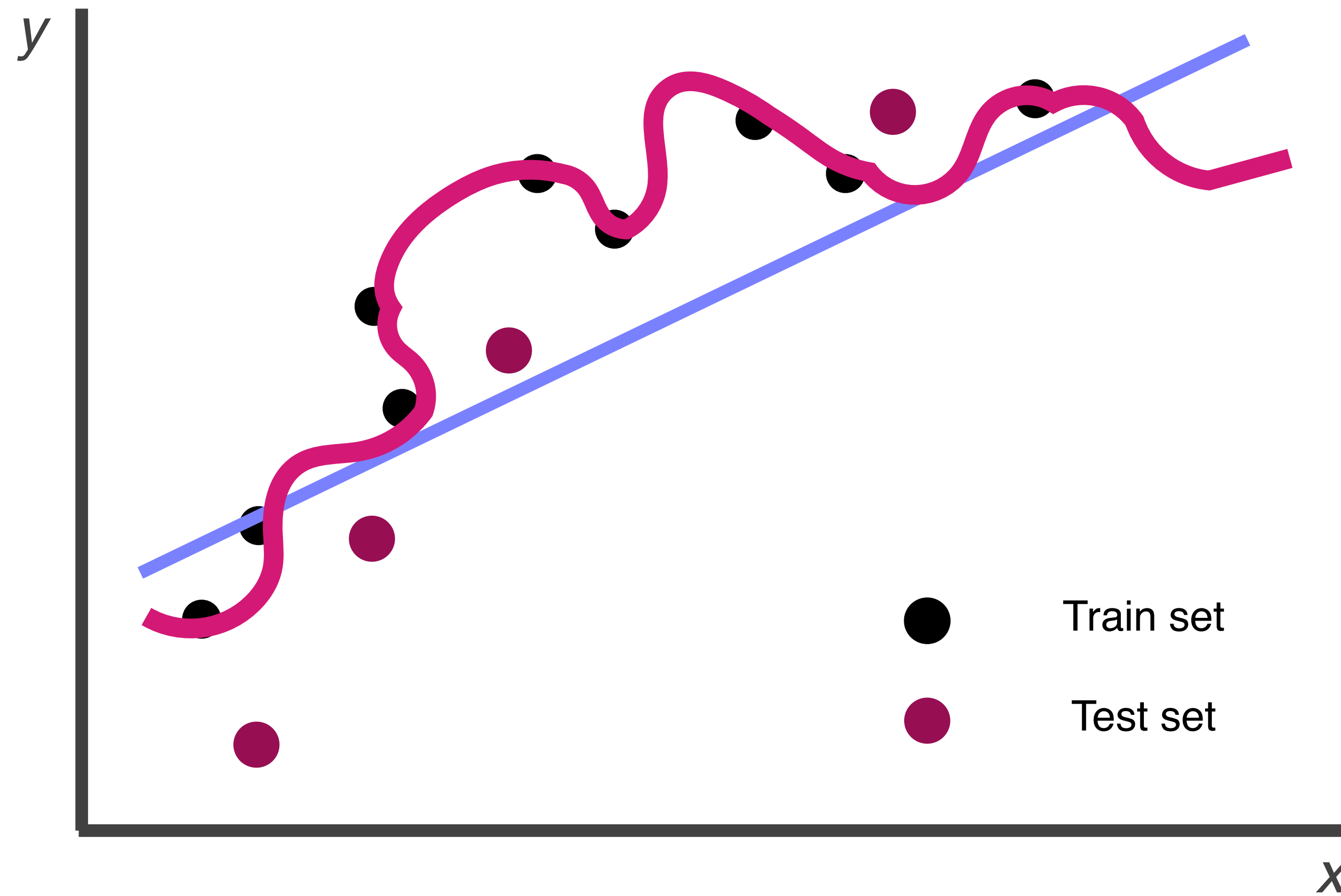
Overfitting vs. Underfitting



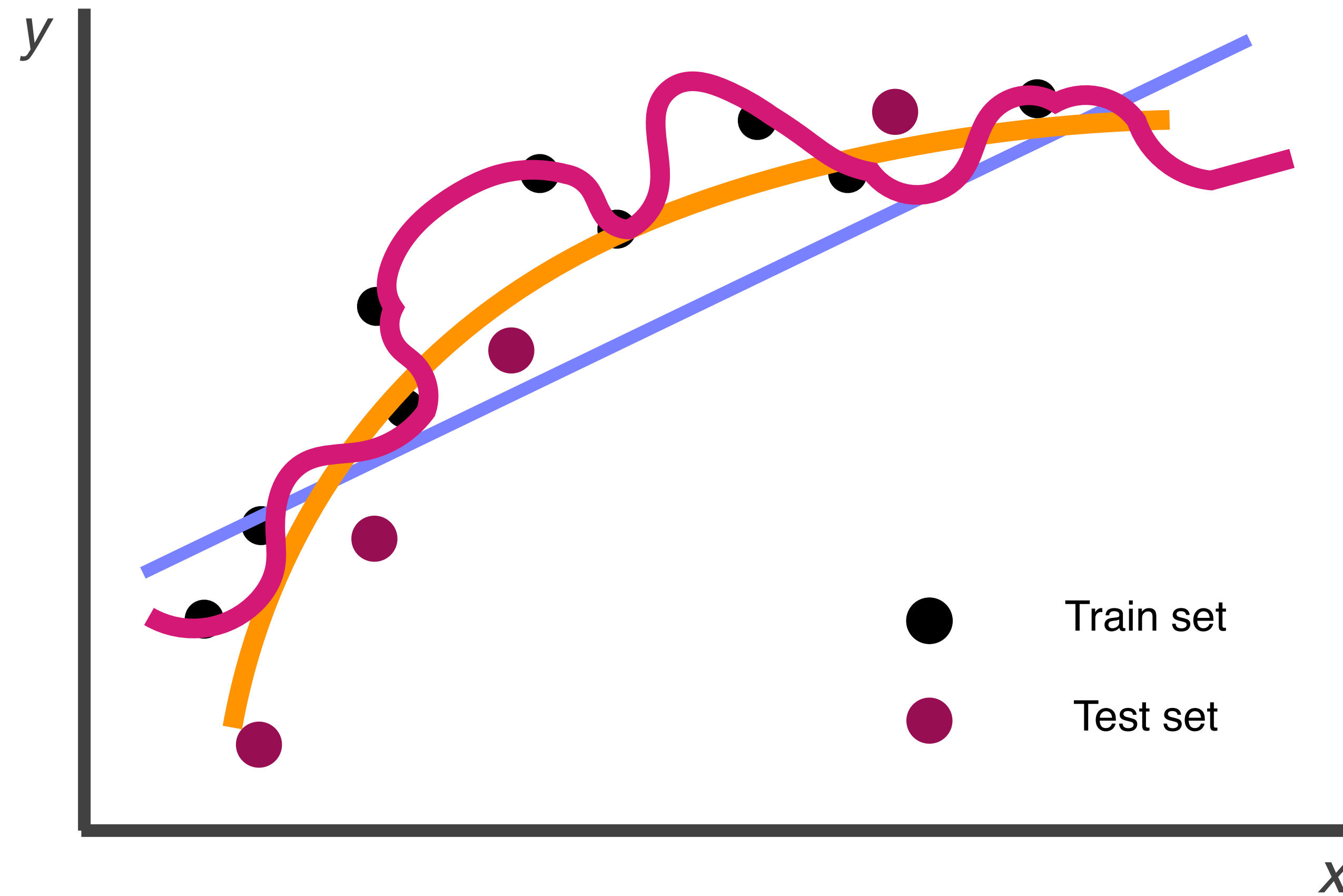
Overfitting vs. Underfitting



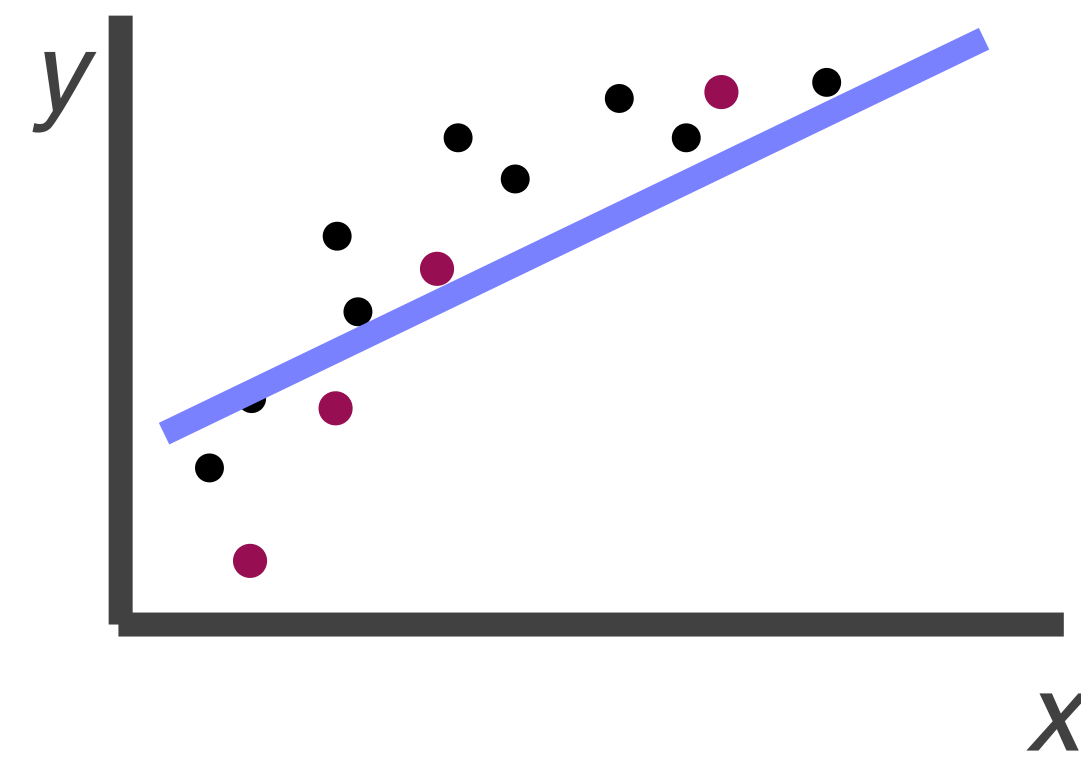
Overfitting vs. Underfitting



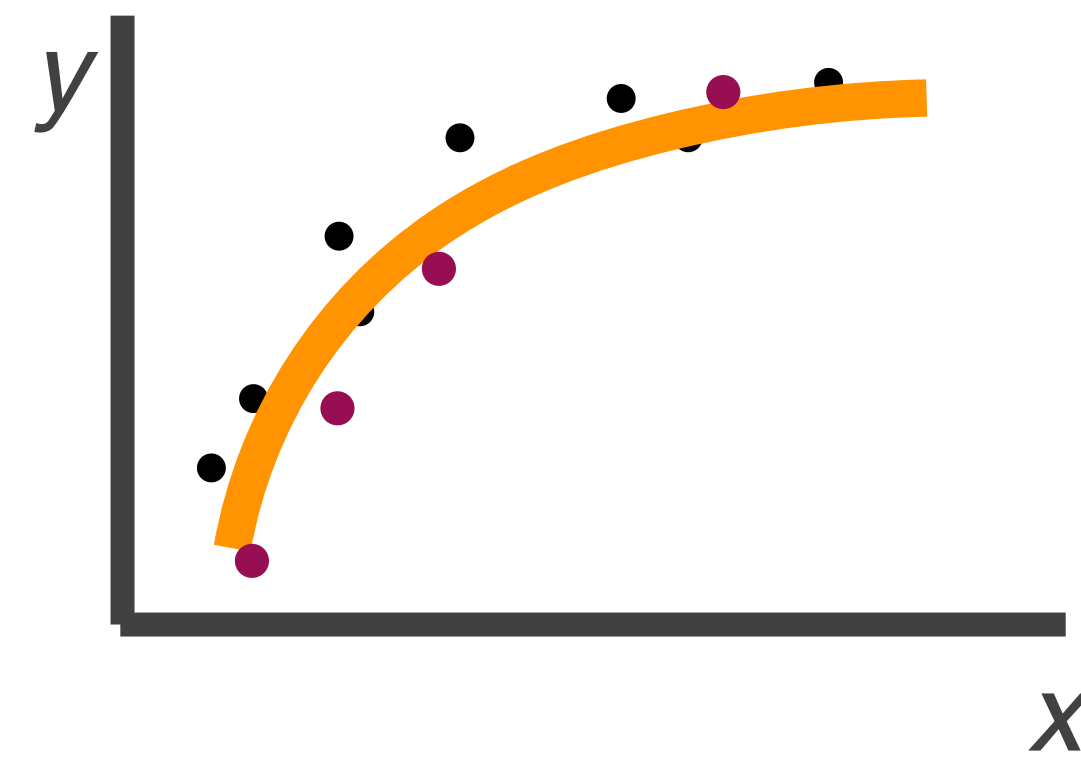
Overfitting vs. Underfitting



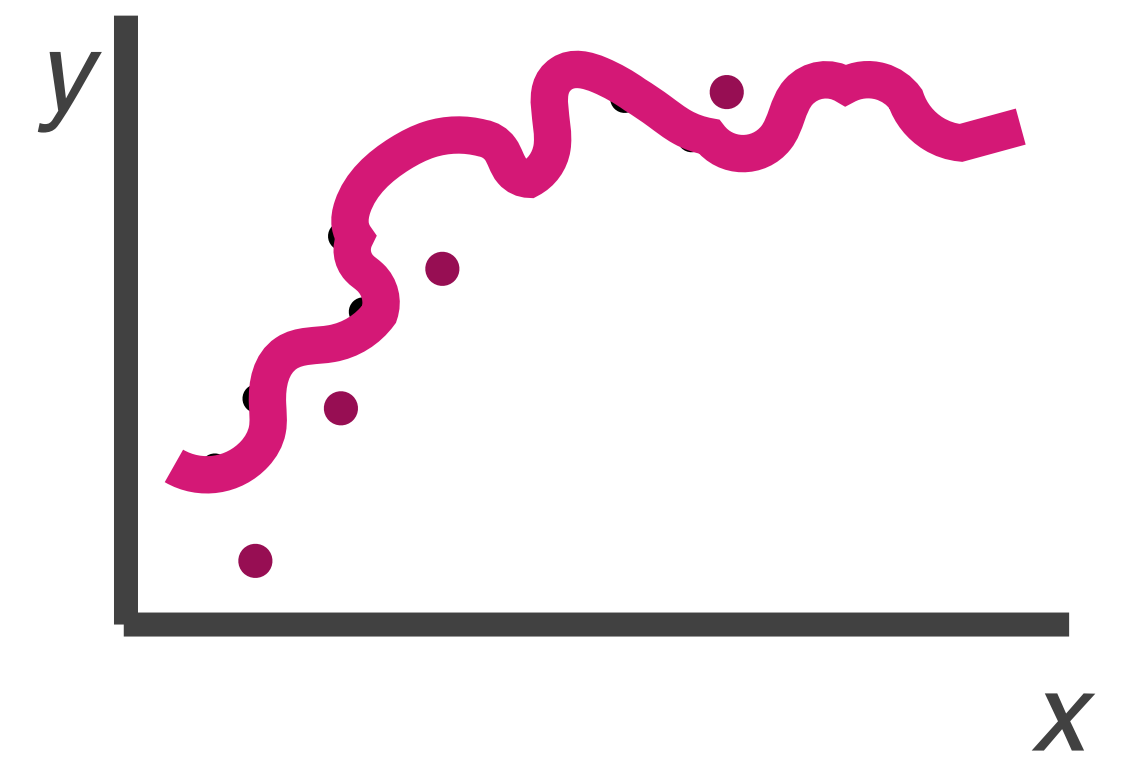
Model complexity and regularization



high bias
 $w_0 + w_1x$



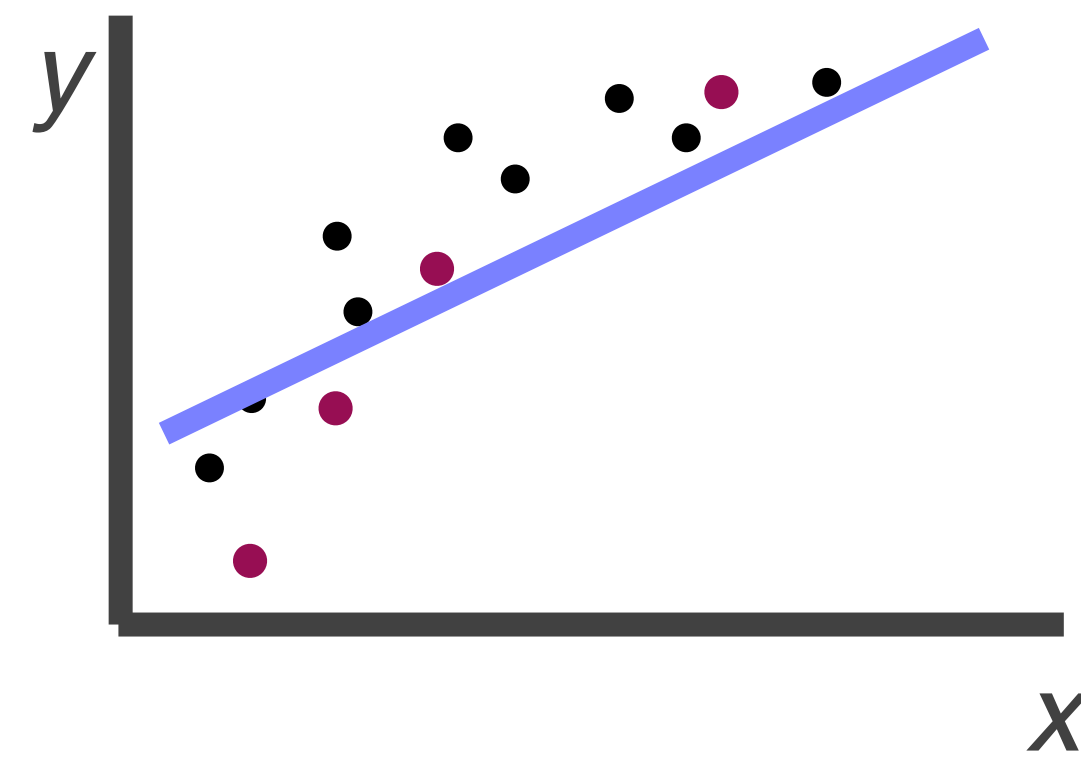
“just right”
 $w_0 + w_1x + w_2x^2$



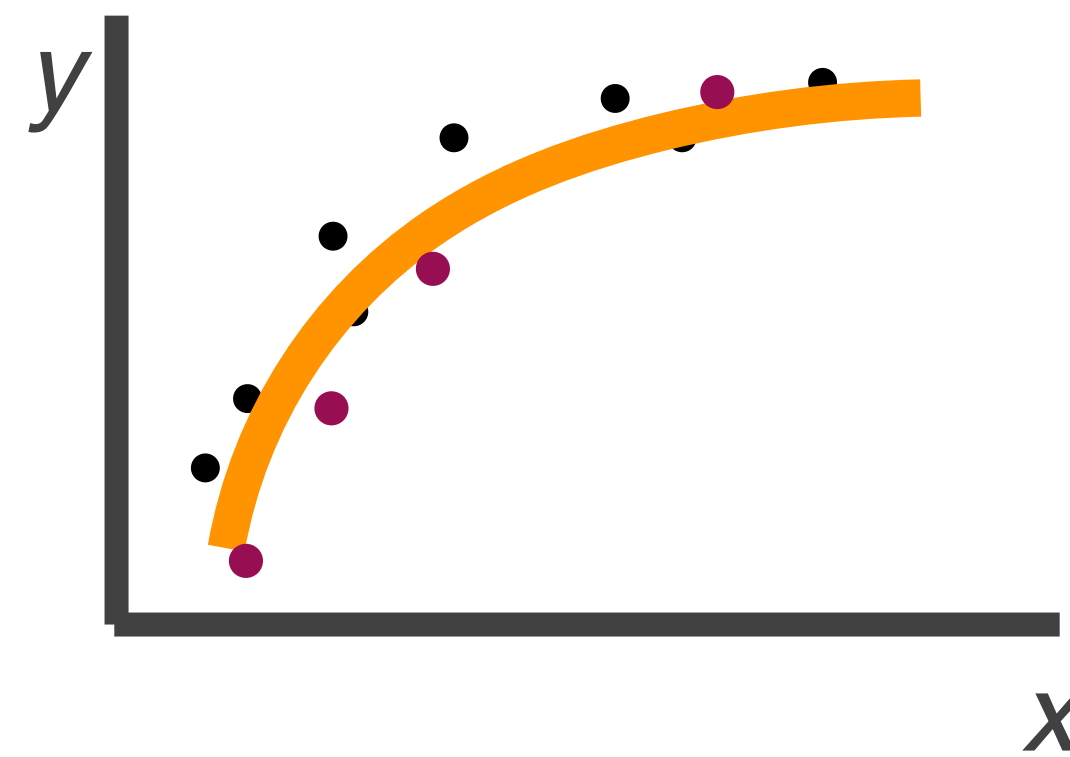
high variance
 $w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4$

Model complexity and regularization

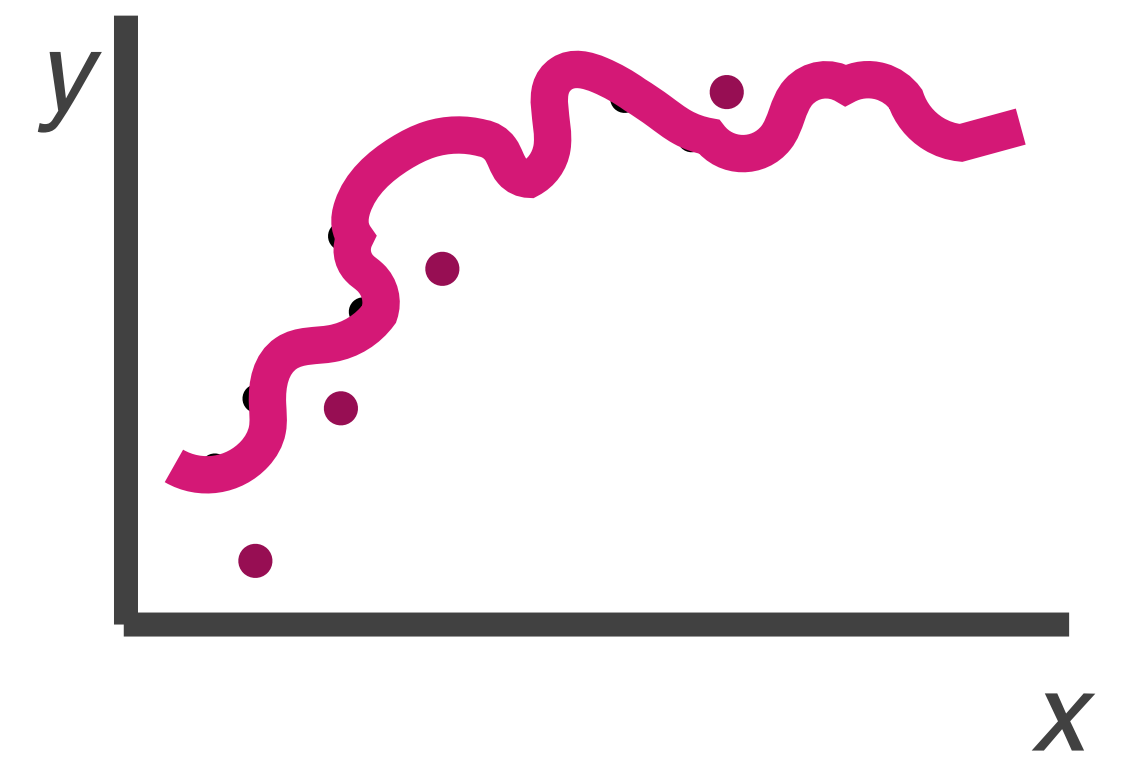
Penalize functions
with large parameters



high bias
 $w_0 + w_1x$



"just right"
 $w_0 + w_1x + w_2x^2$



high variance
 $w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4$

$$\mathcal{L} = \|\hat{y}_{ML} - y\|_2^2 + \lambda \|w\|_2^2$$

4. Bias-variance tradeoff

Learning Objectives:

- We can split the error into different contributions: bias, variance, irreducible
- We cannot reduce bias and variance at the same time
- We can penalize for model complexity with a regularization term

5. Hyperparameters

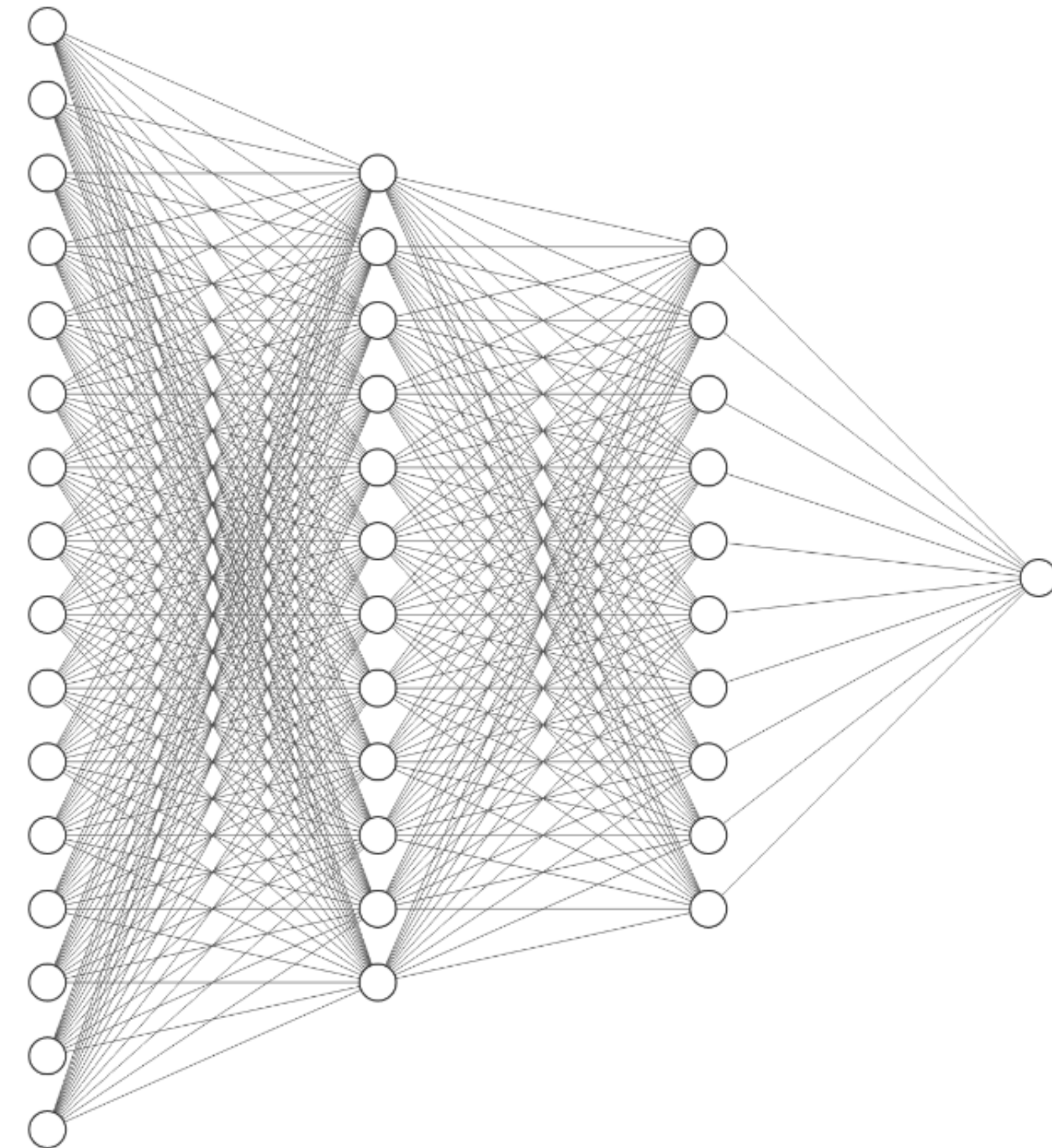
Learning Objectives:

- Hyperparameter are non-trainable parameters that need to be tuned
- (Nested) Cross-validation can be used to optimize them
- K-fold cross-validation

Hyperparameters

We cannot learn them,
but we need to optimize
them

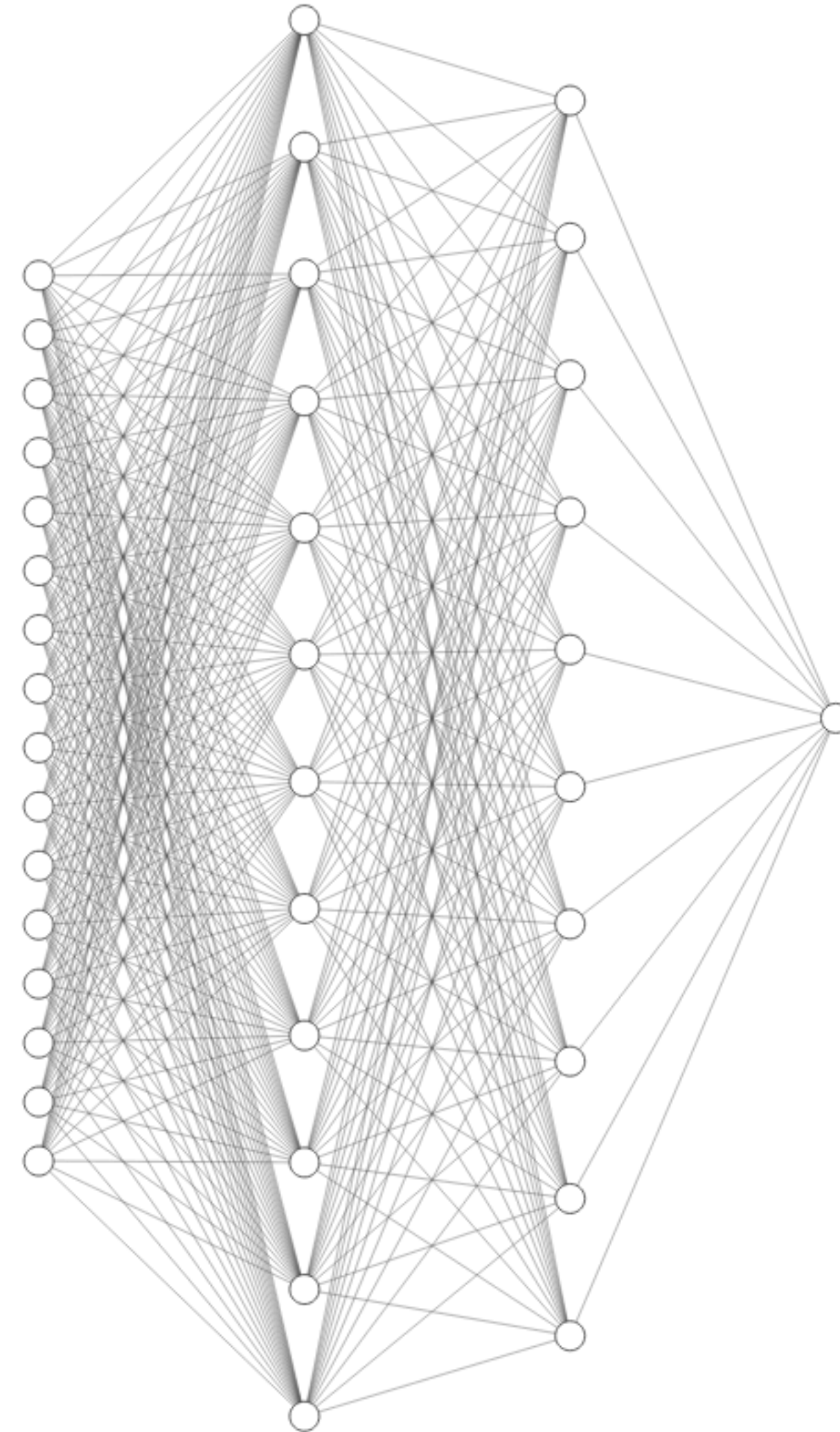
$$\mathcal{L} = \|\hat{y}_{ML} - y\|_2^2 + \lambda \|w\|_2^2$$



Hyperparameters

We cannot learn them,
but we need to optimize
them

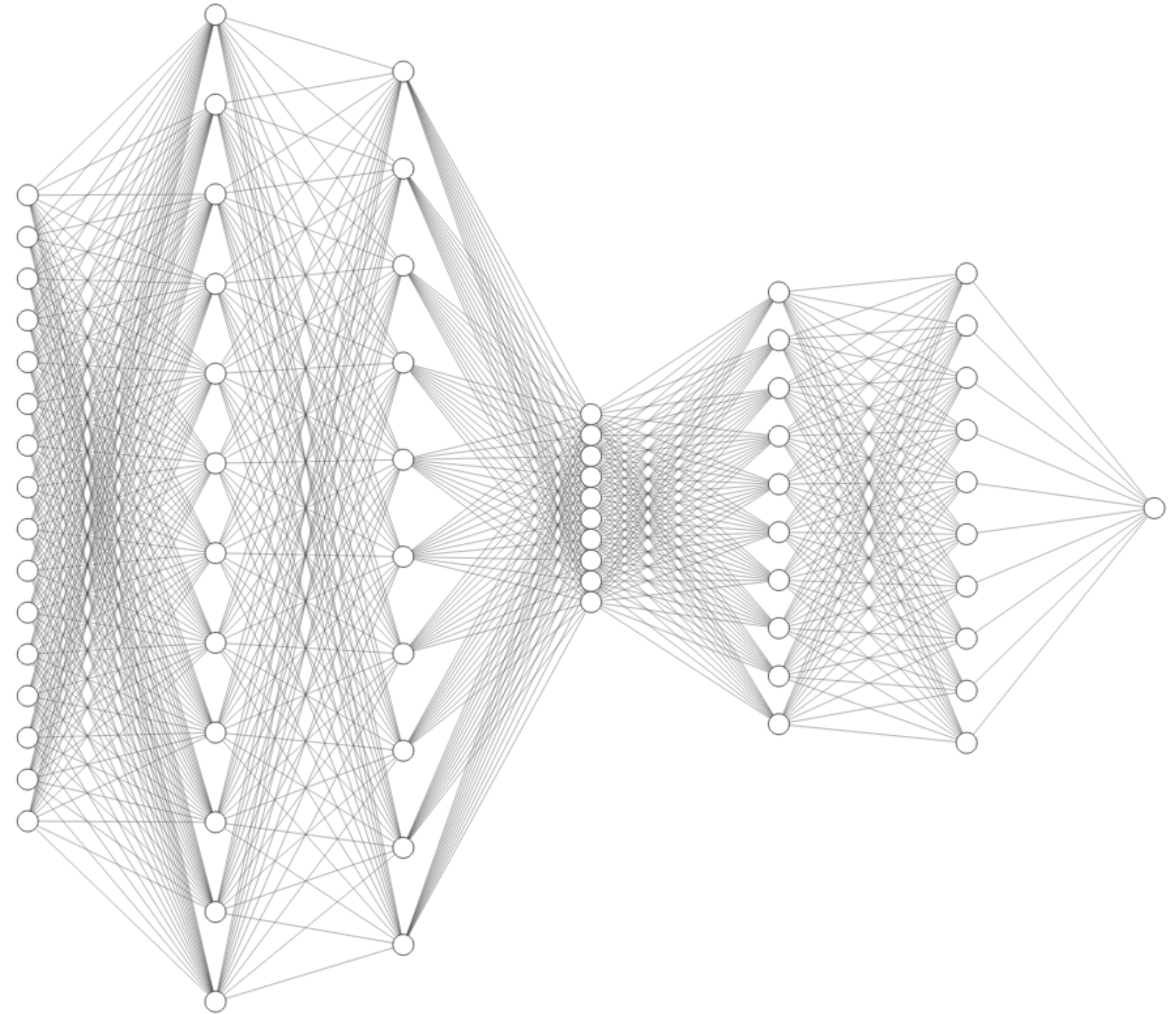
$$\mathcal{L} = \|\hat{y}_{ML} - y\|_2^2 + \lambda \|w\|_2^2$$



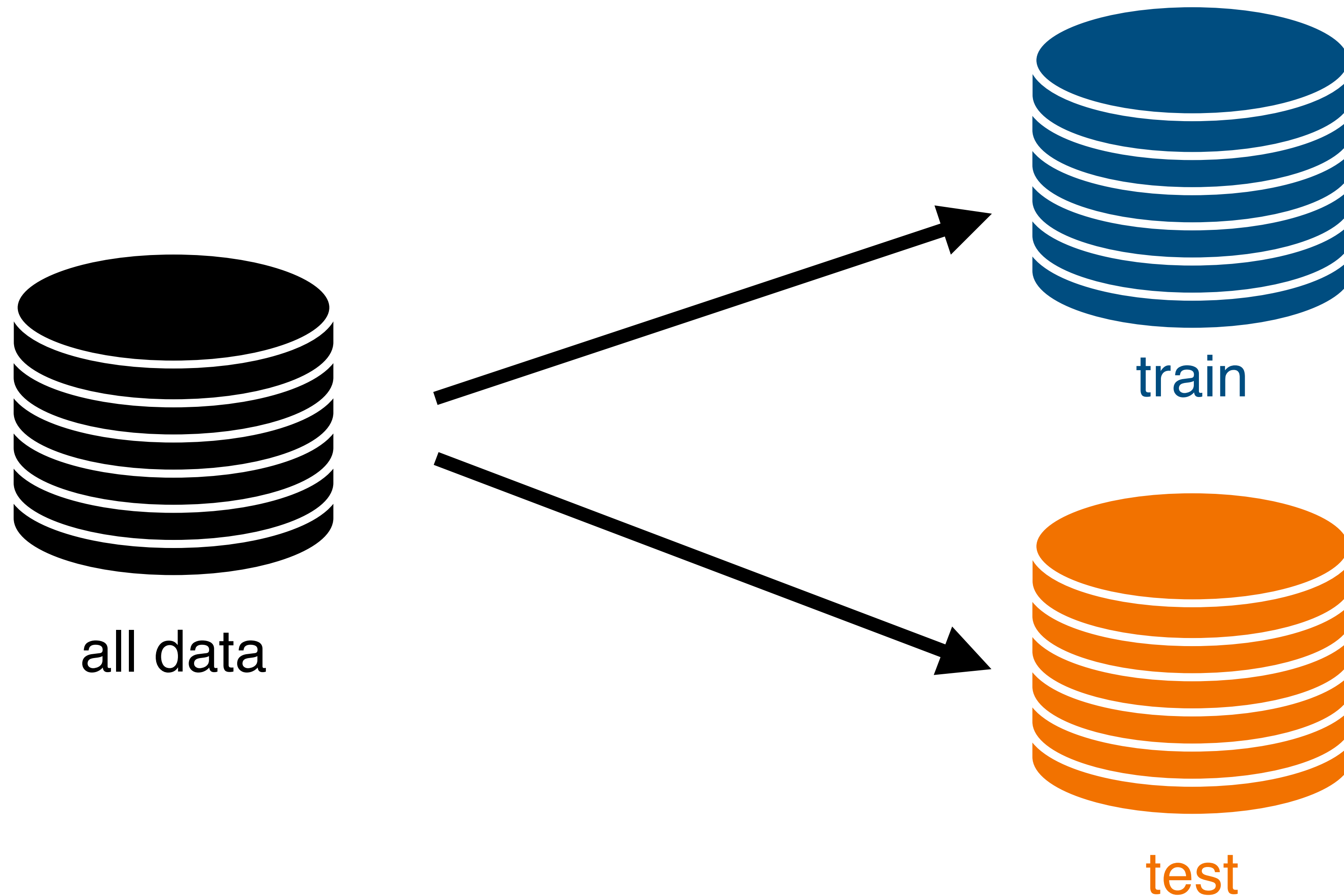
Hyperparameters

We cannot learn them,
but we need to optimize
them

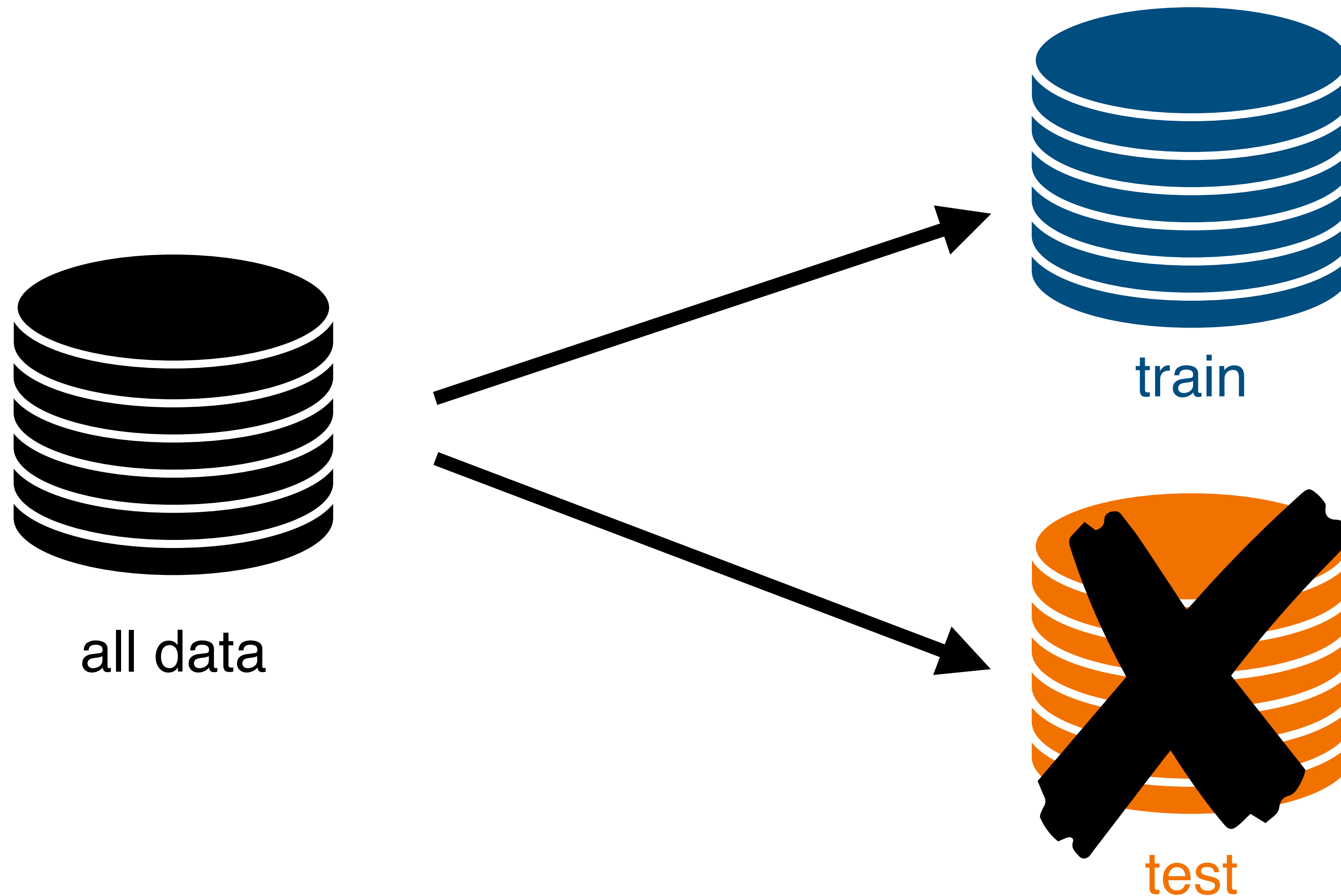
$$\mathcal{L} = \|\hat{y}_{ML} - y\|_2^2 + \lambda \|w\|_2^2$$



Recall: First step: Put aside some test data *you do not look at*

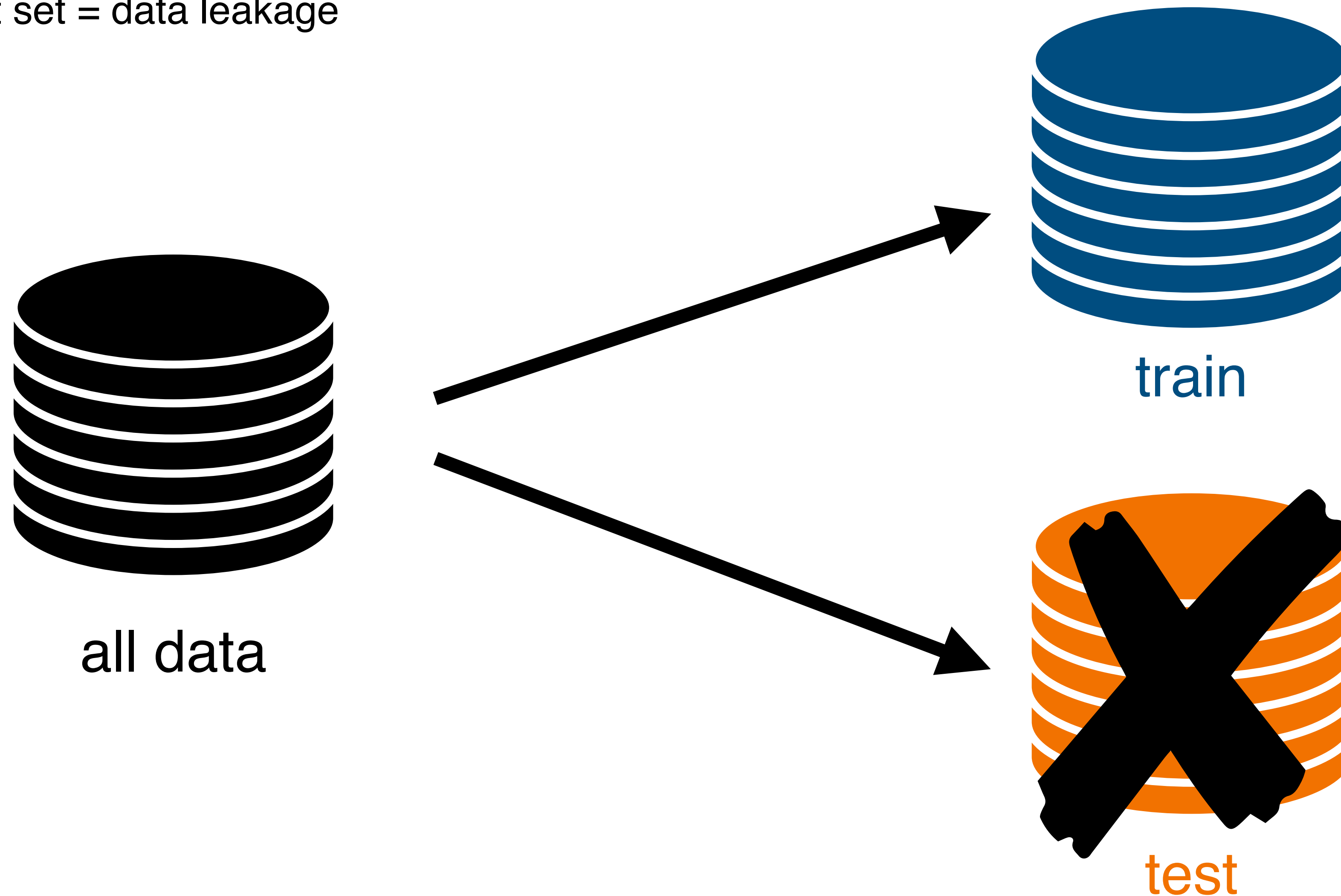


Recall: First step: Put aside some test data *you do not look at*



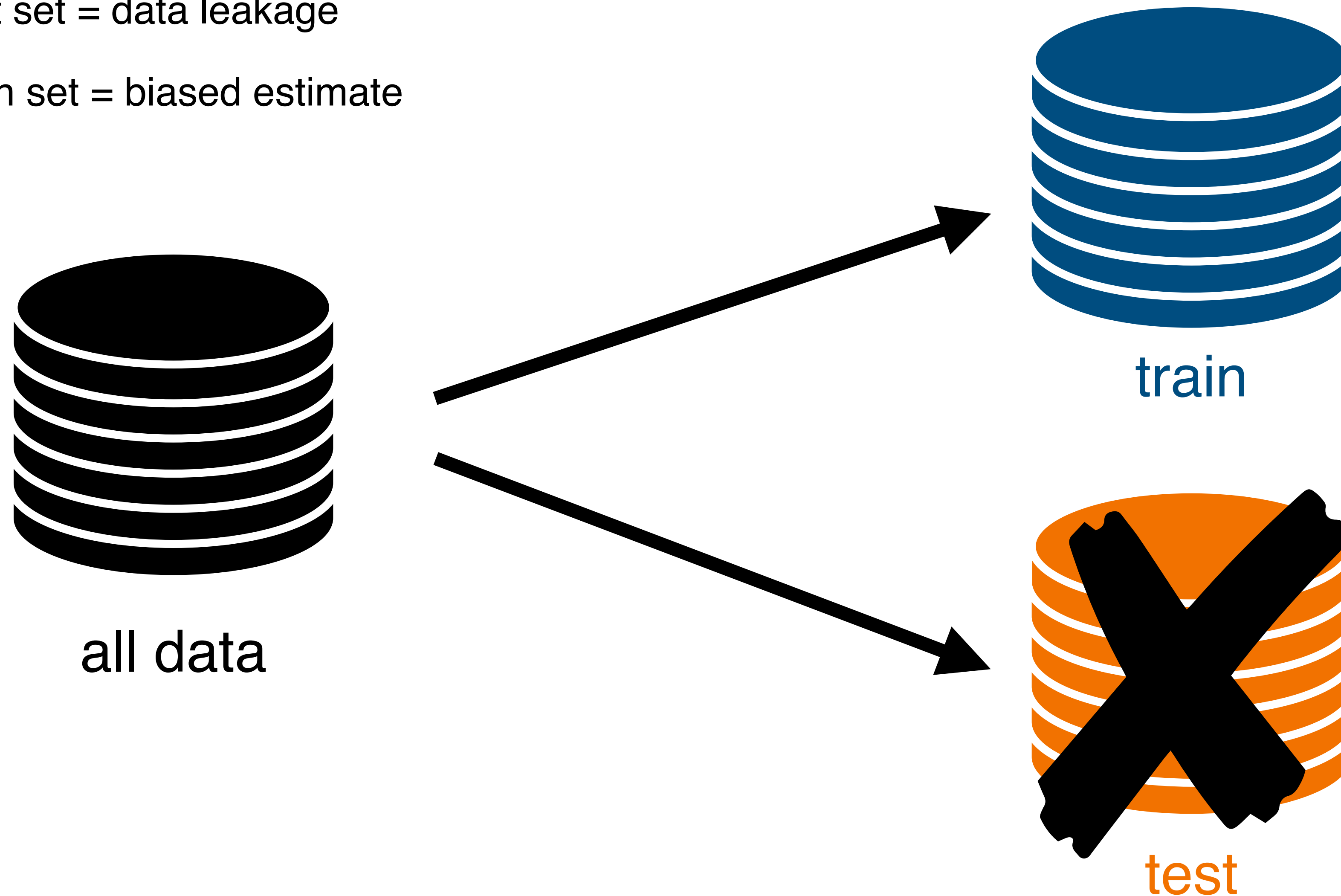
Recall: First step: Put aside some test data *you do not look at*

- Using test set = data leakage

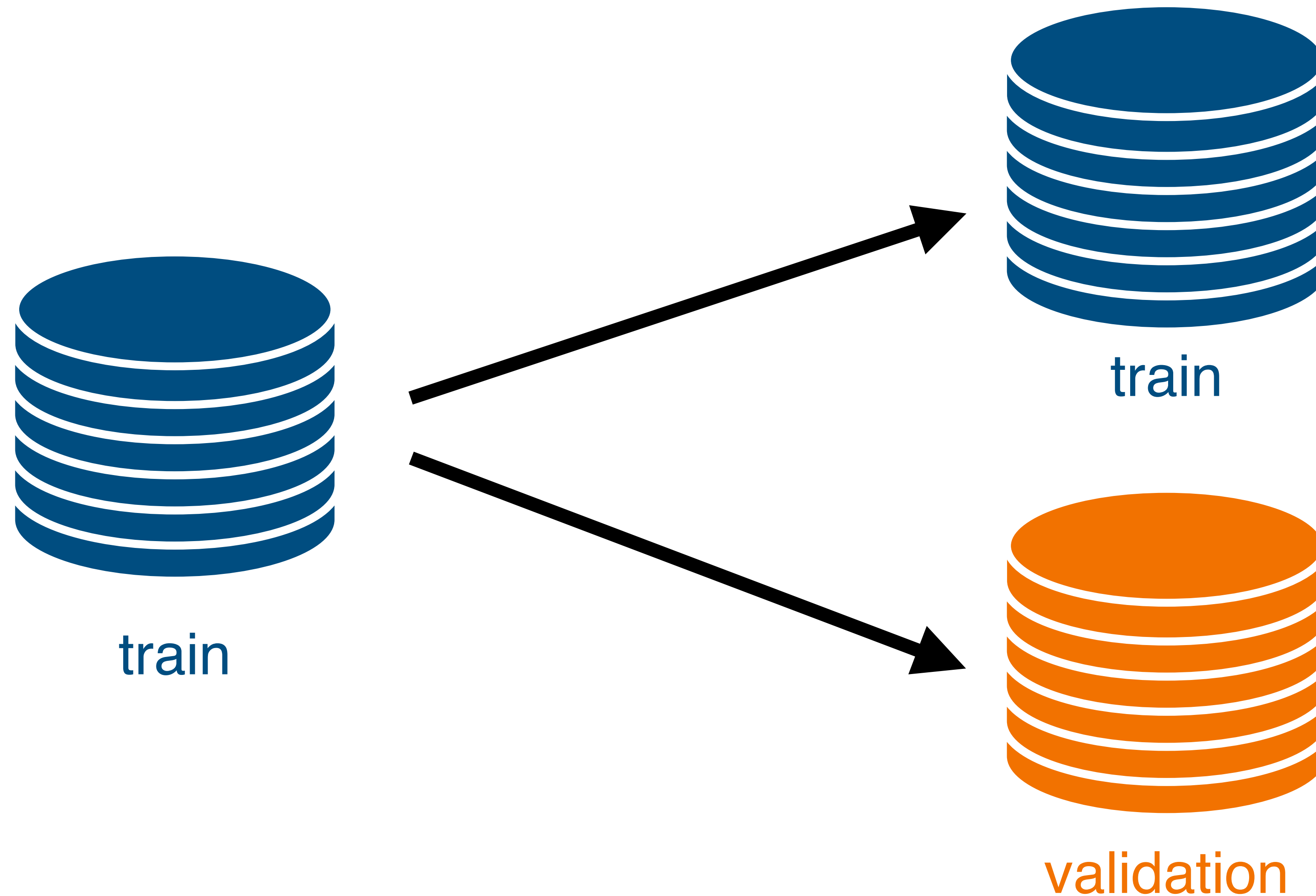


Recall: First step: Put aside some test data *you do not look at*

- Using test set = data leakage
- Using train set = biased estimate

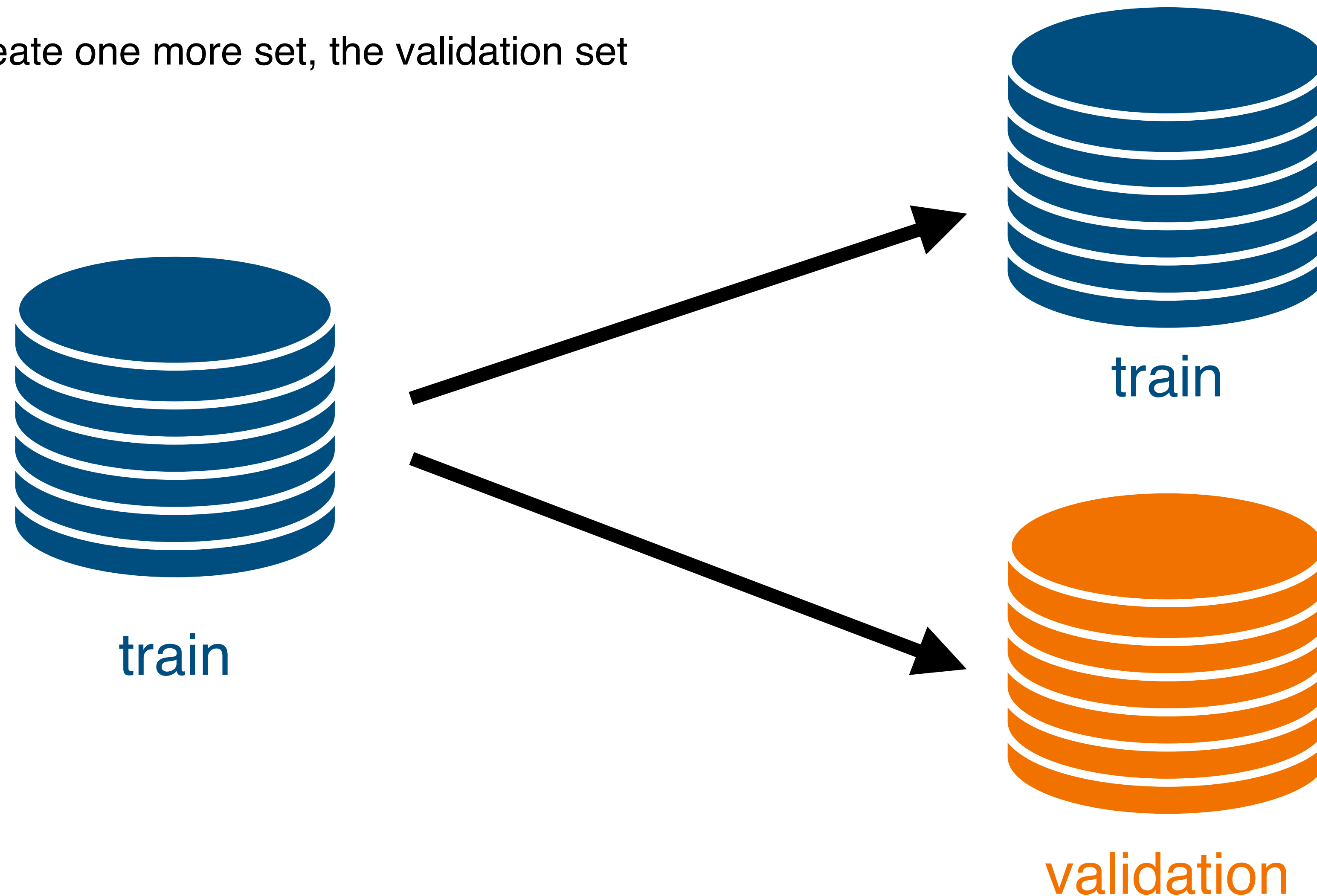


Hyperparameter tuning on *validation set*

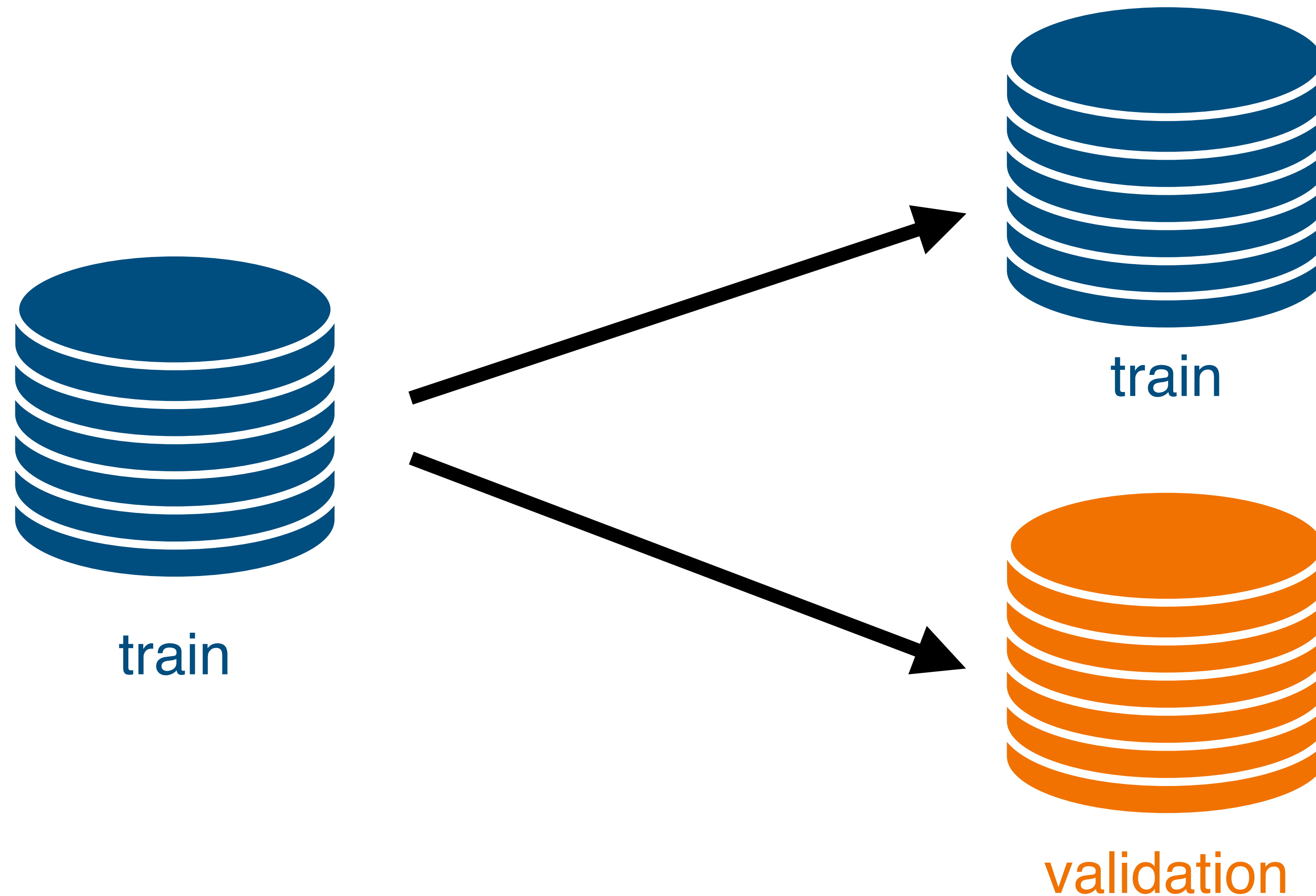


Hyperparameter tuning on *validation set*

Solution: Create one more set, the validation set



Hyperparameter tuning on *validation set*



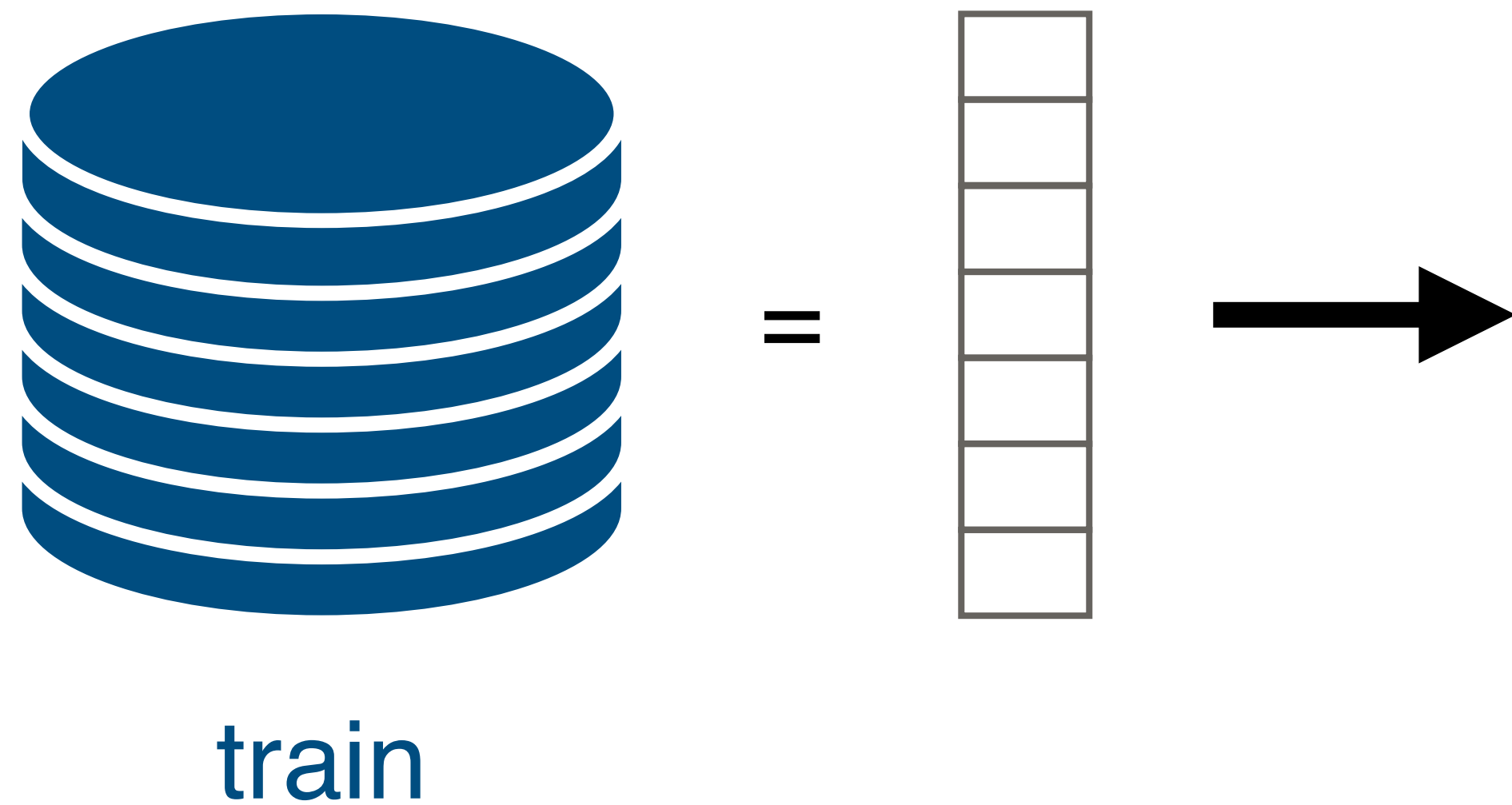
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost



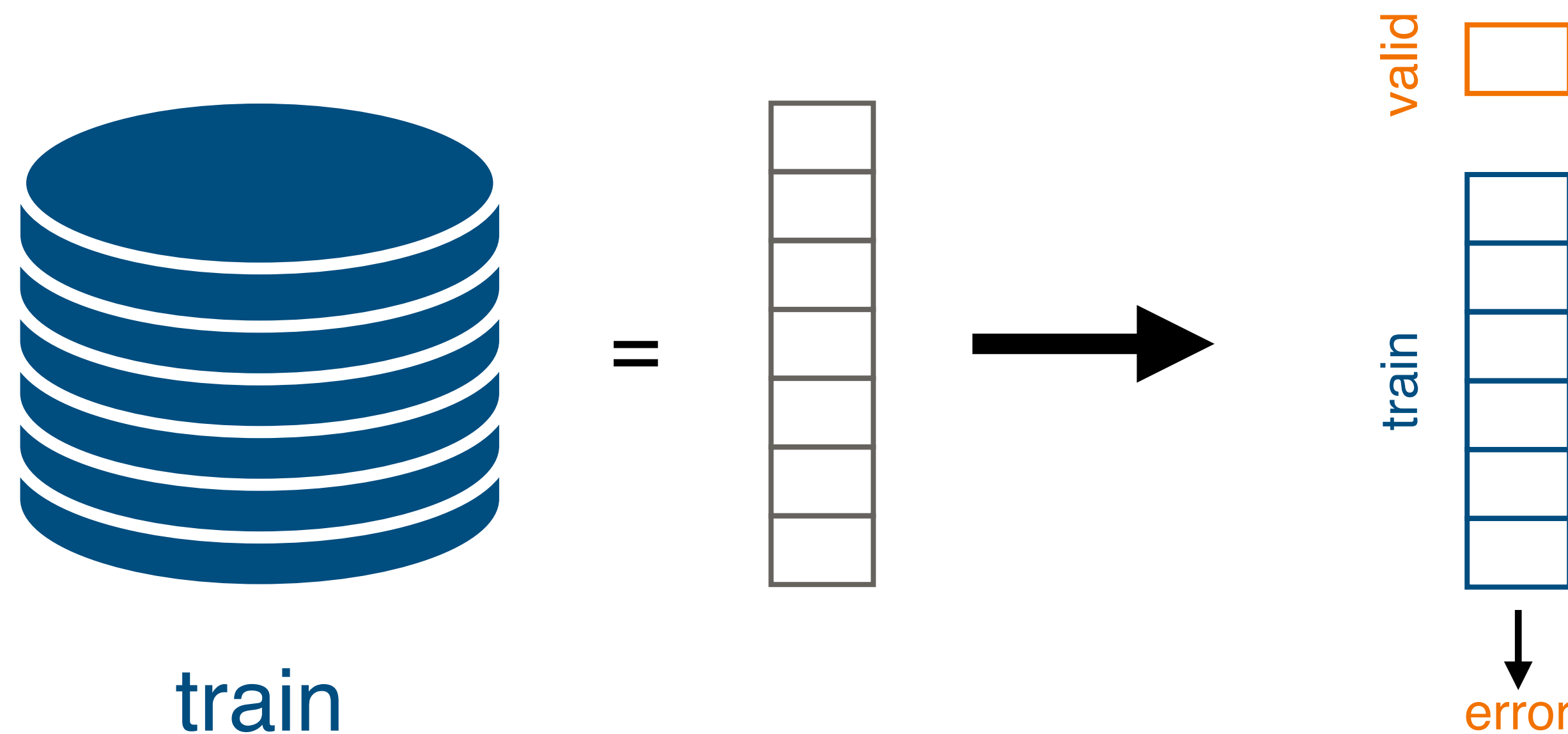
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost



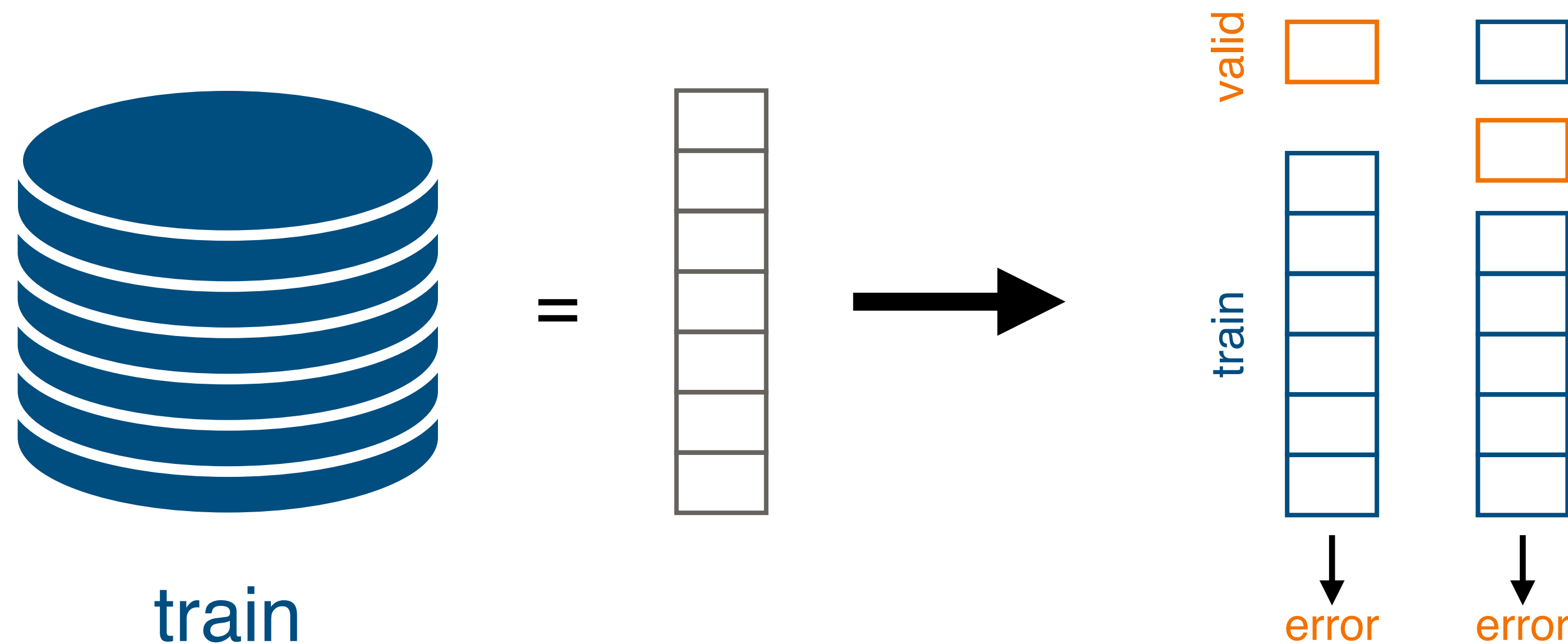
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost



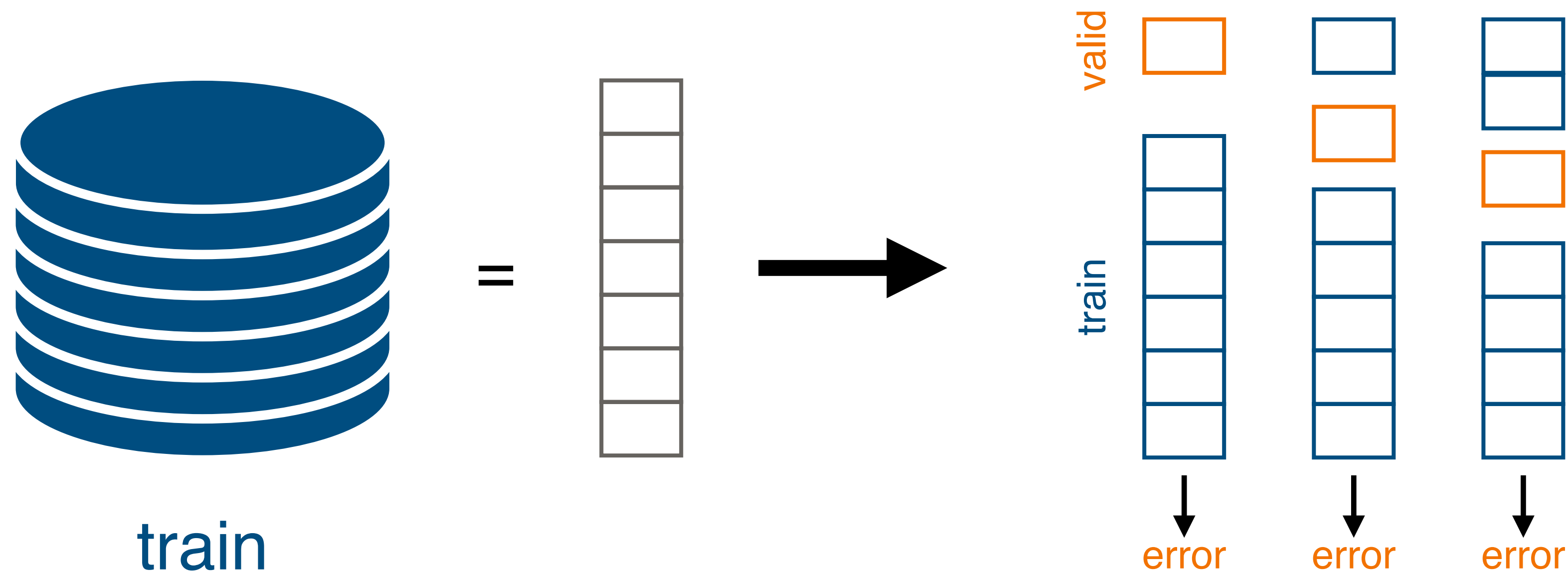
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost



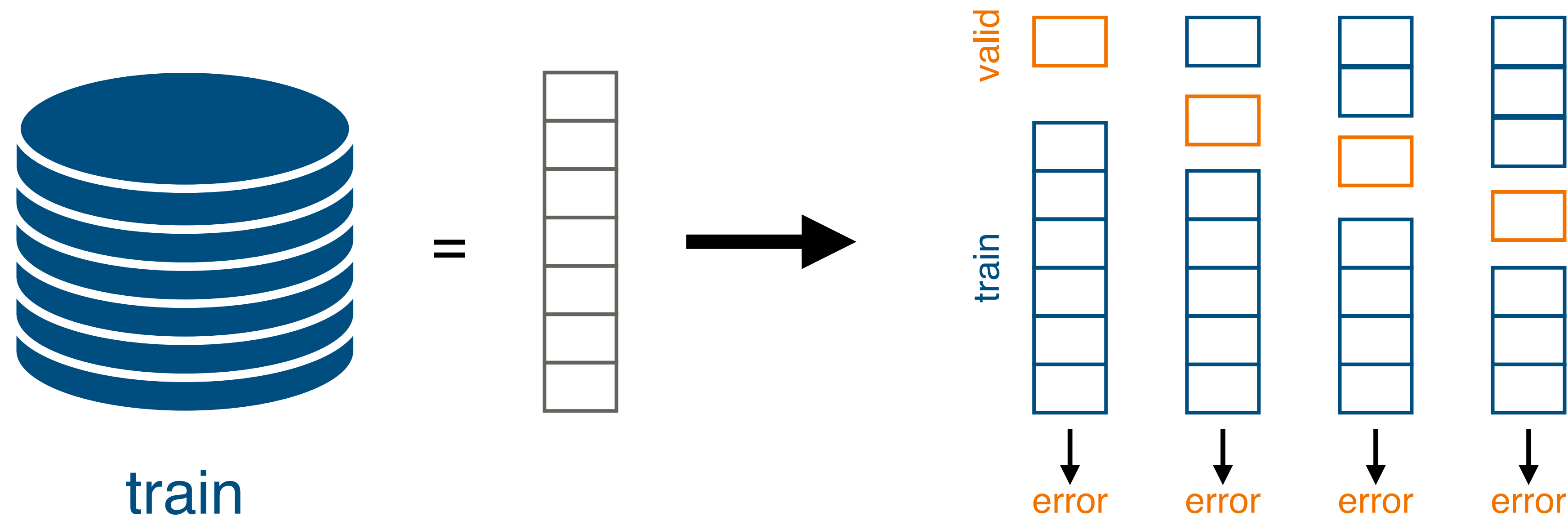
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost



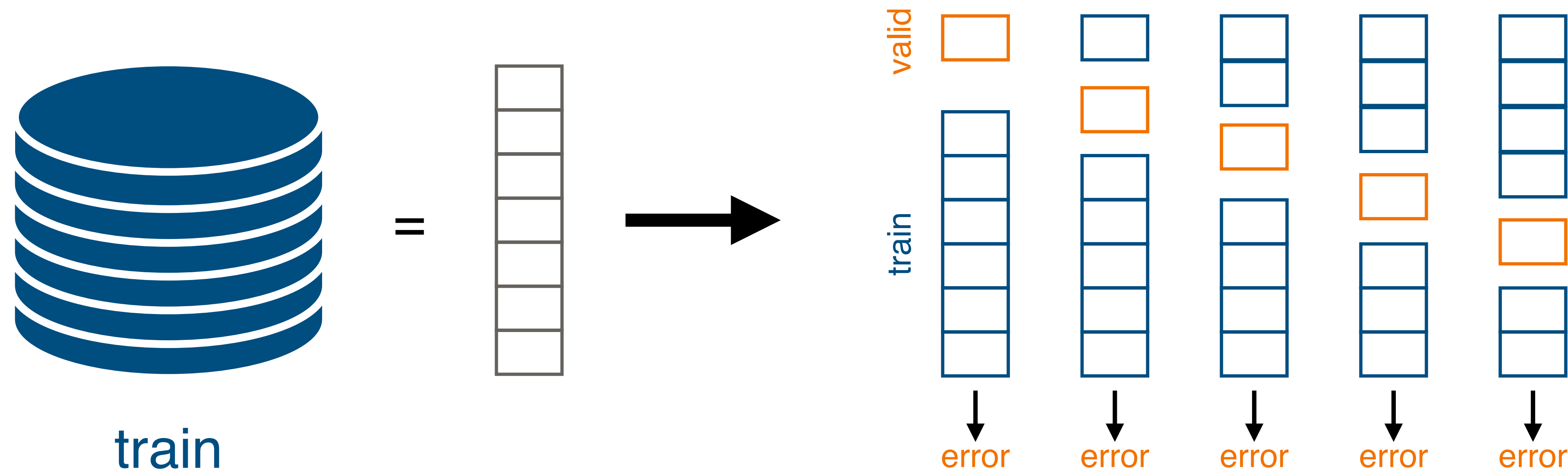
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost



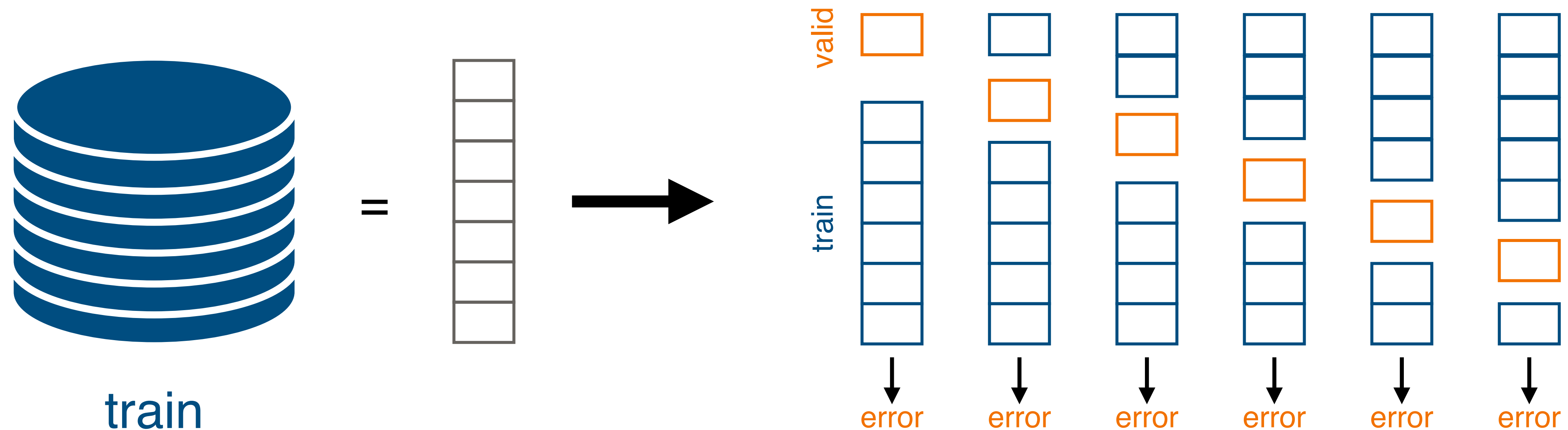
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost



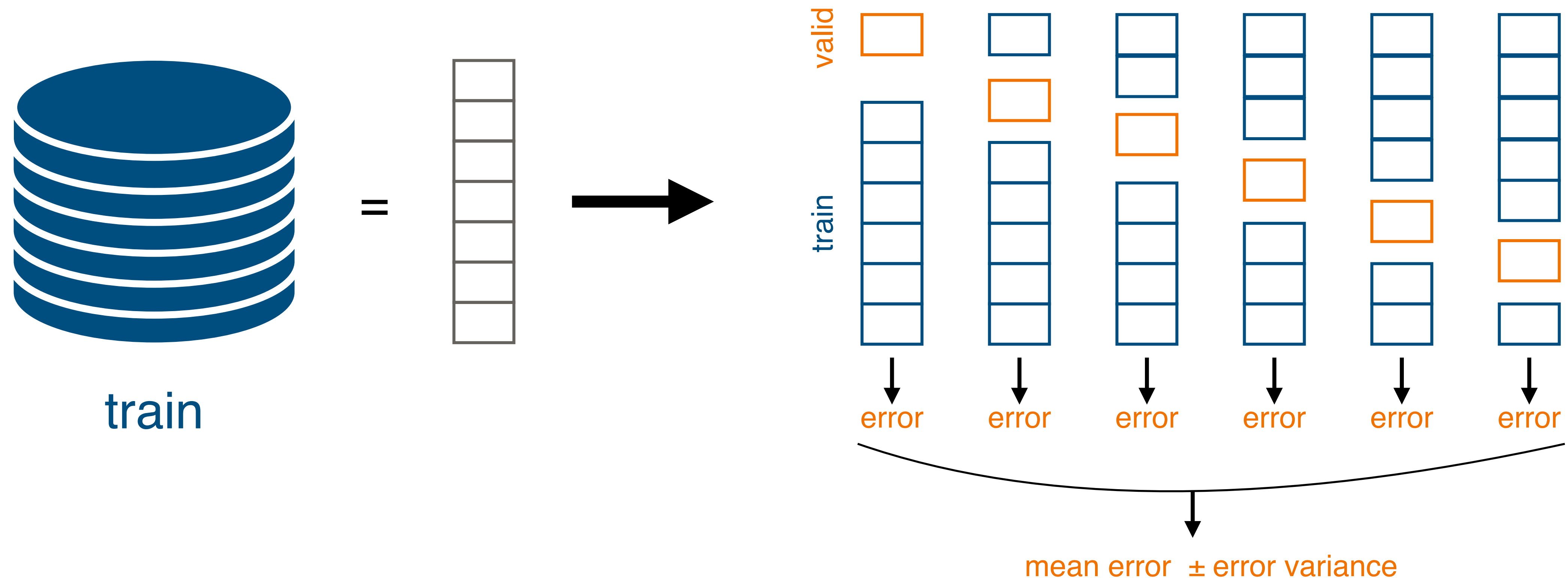
For small datasets: Build validation set using *k*-fold crossvalidation

Problem with simple split: Pessimistic bias and variance of the error estimate

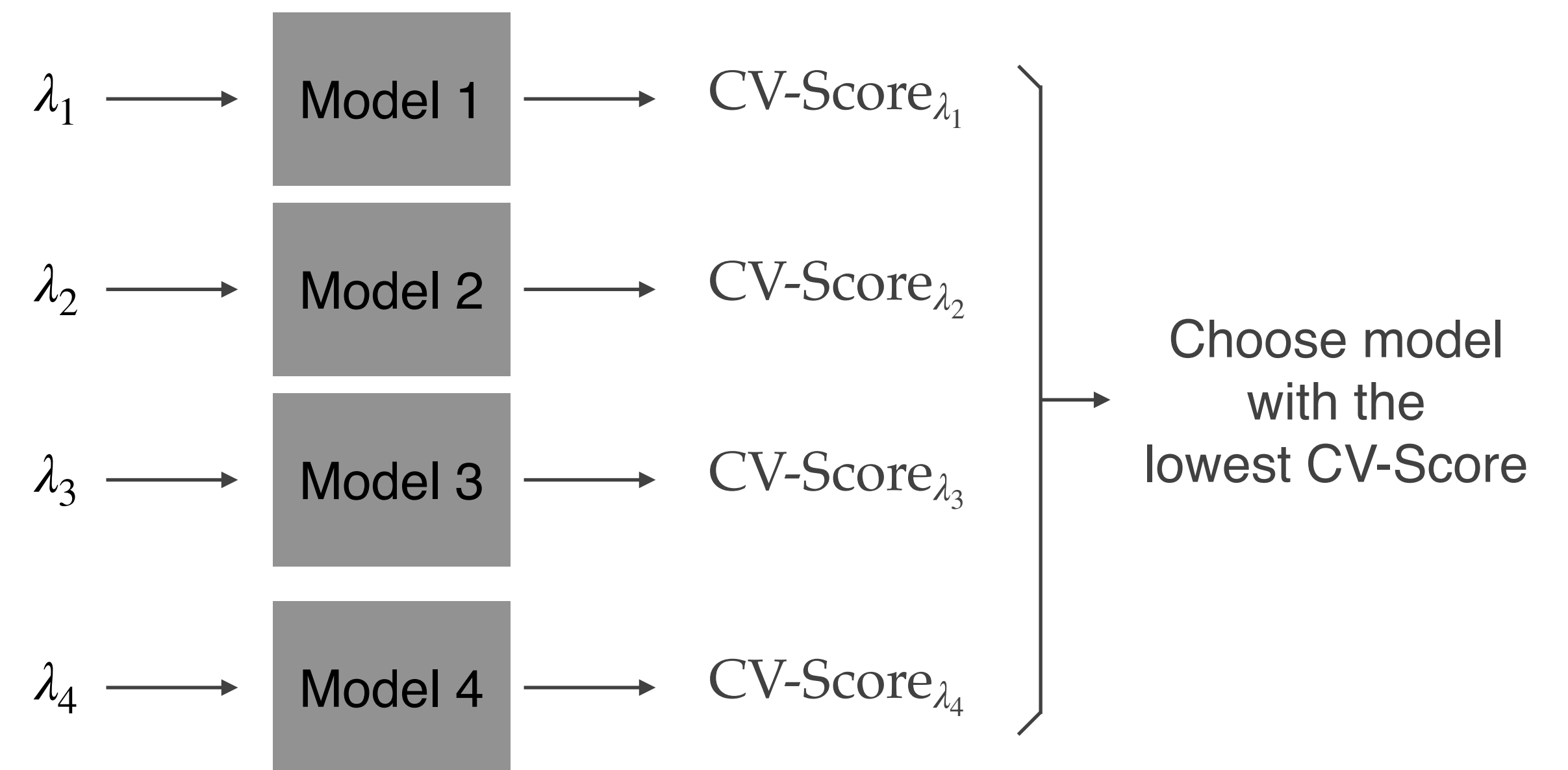
Doing split only once: High variance

Larger test set reduces the variance, but leads to underestimation (pessimistic bias)

Increasing *k* increases computational cost

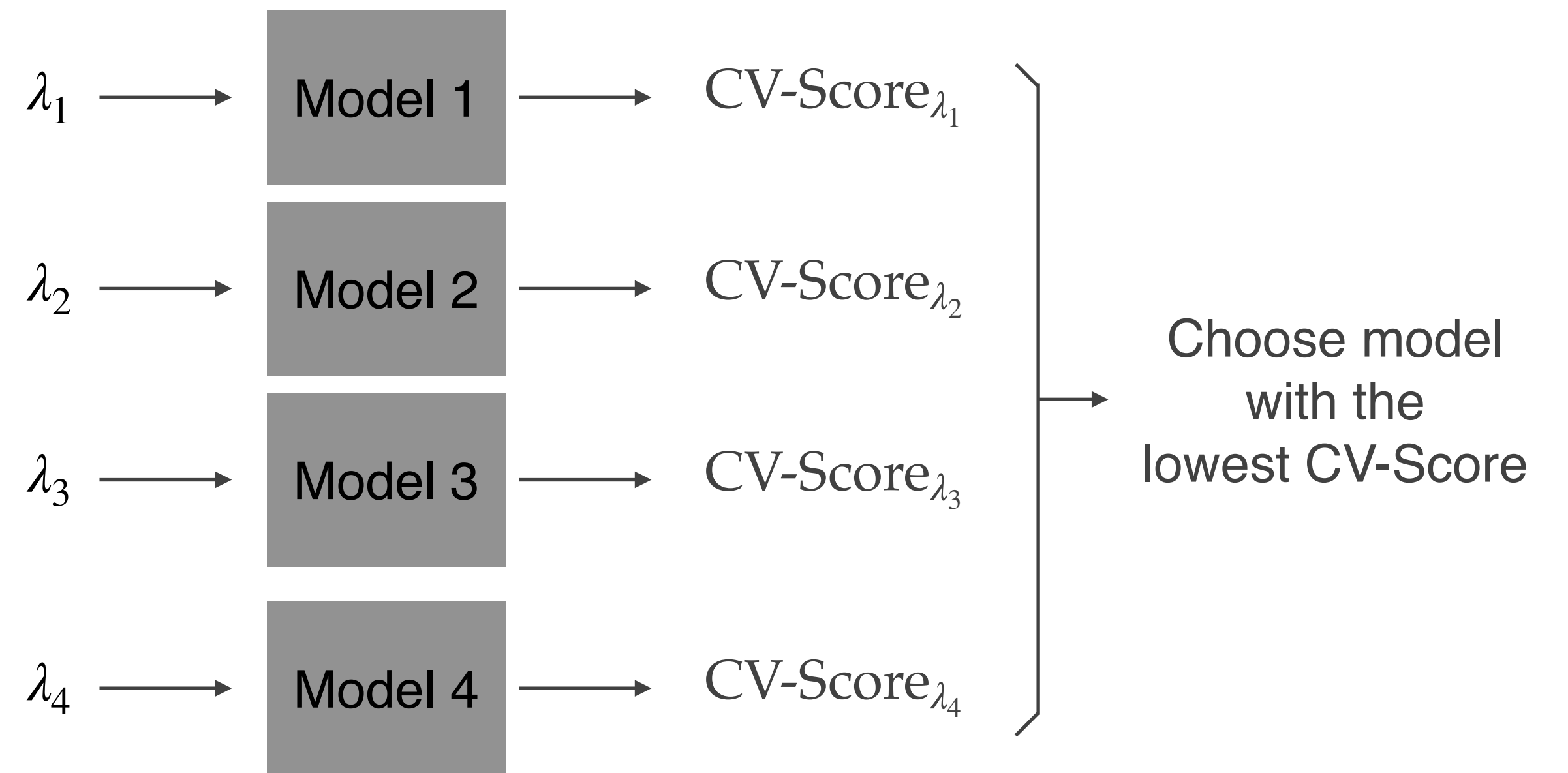


Using k-fold crossvalidation for hyperparameter tuning



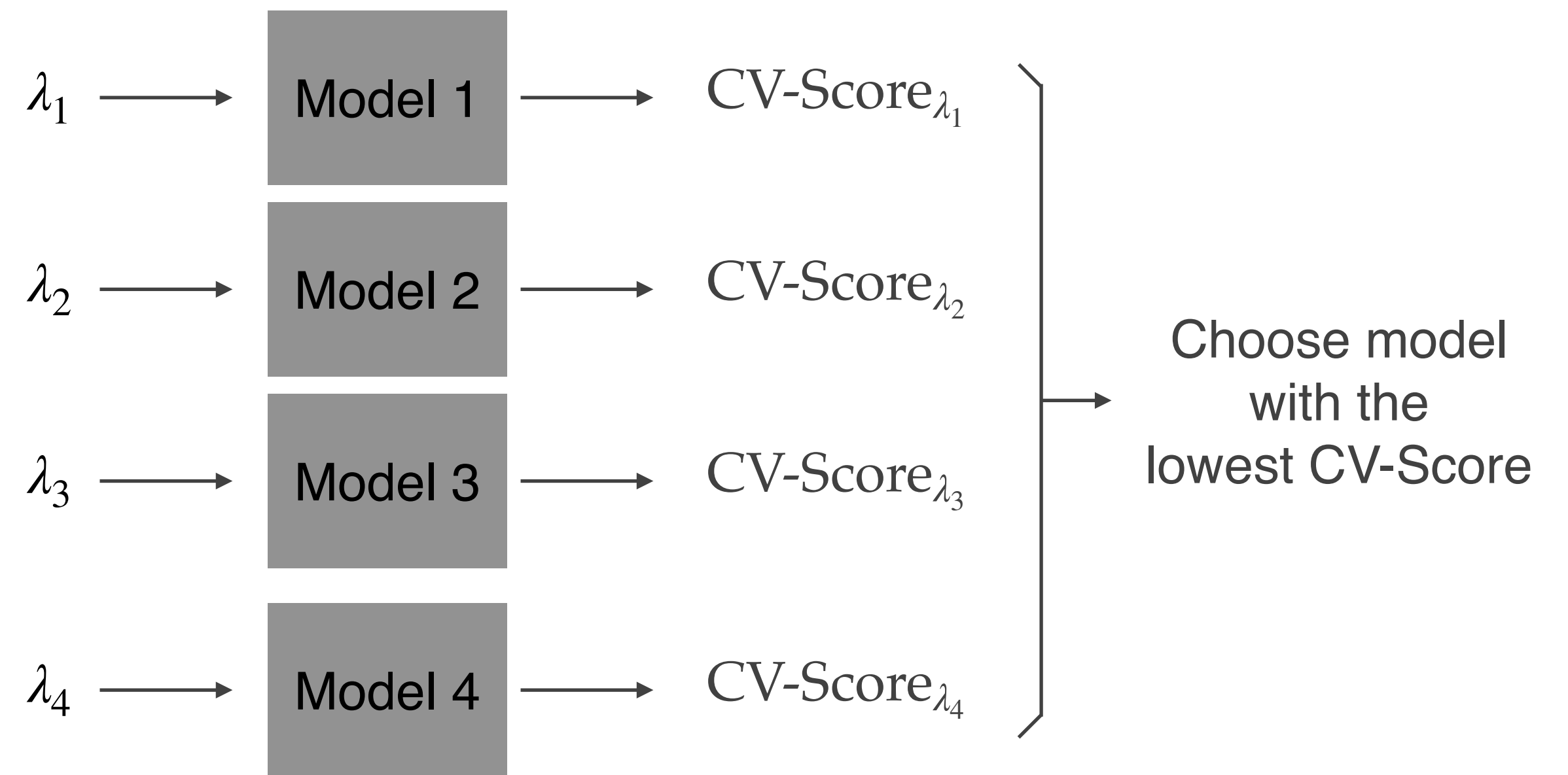
Using k-fold crossvalidation for hyperparameter tuning

- We test different hyperparameter settings



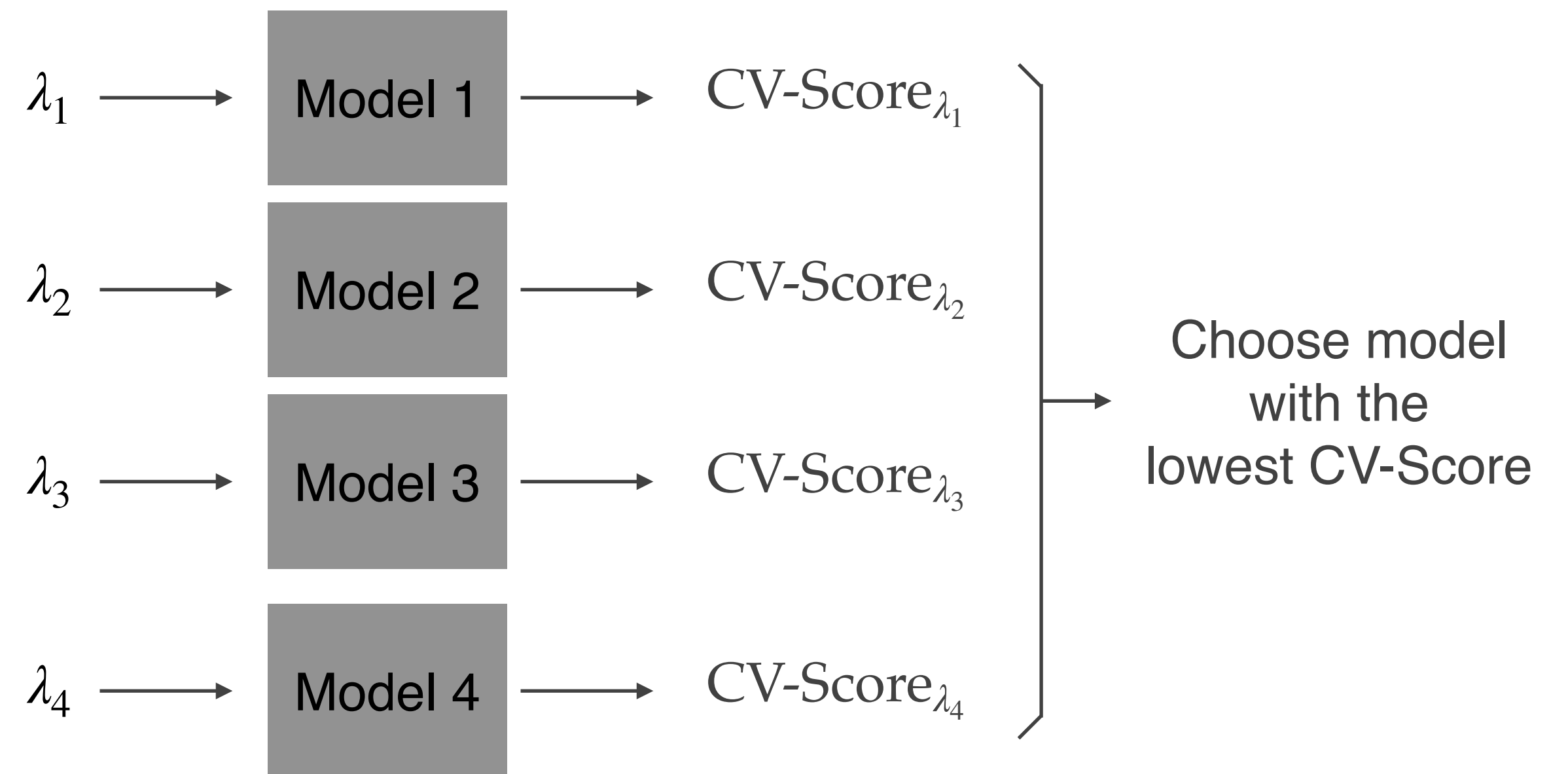
Using k-fold crossvalidation for hyperparameter tuning

- We test different hyperparameter settings
- Estimate the error rate using cross validation



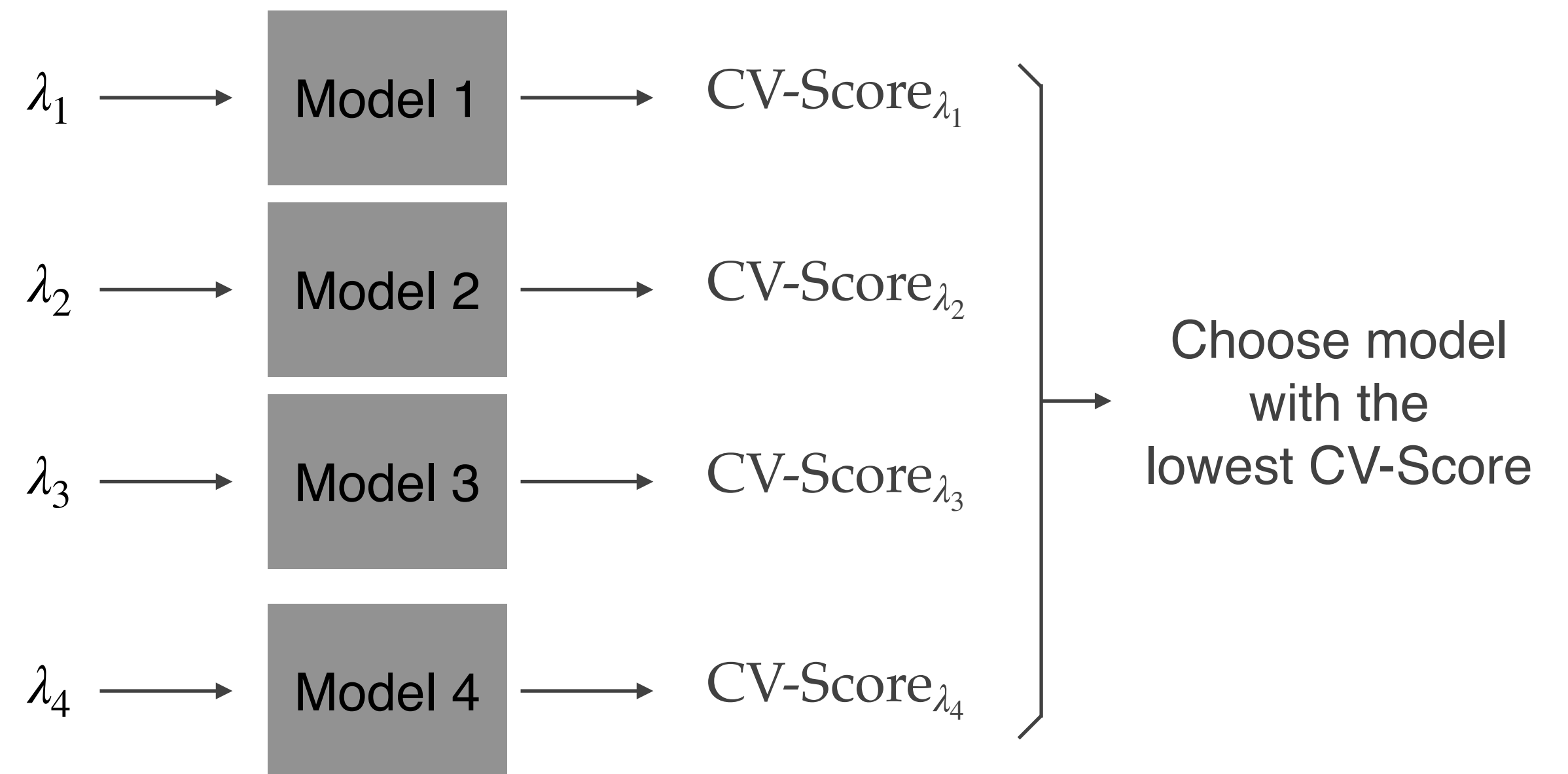
Using k-fold crossvalidation for hyperparameter tuning

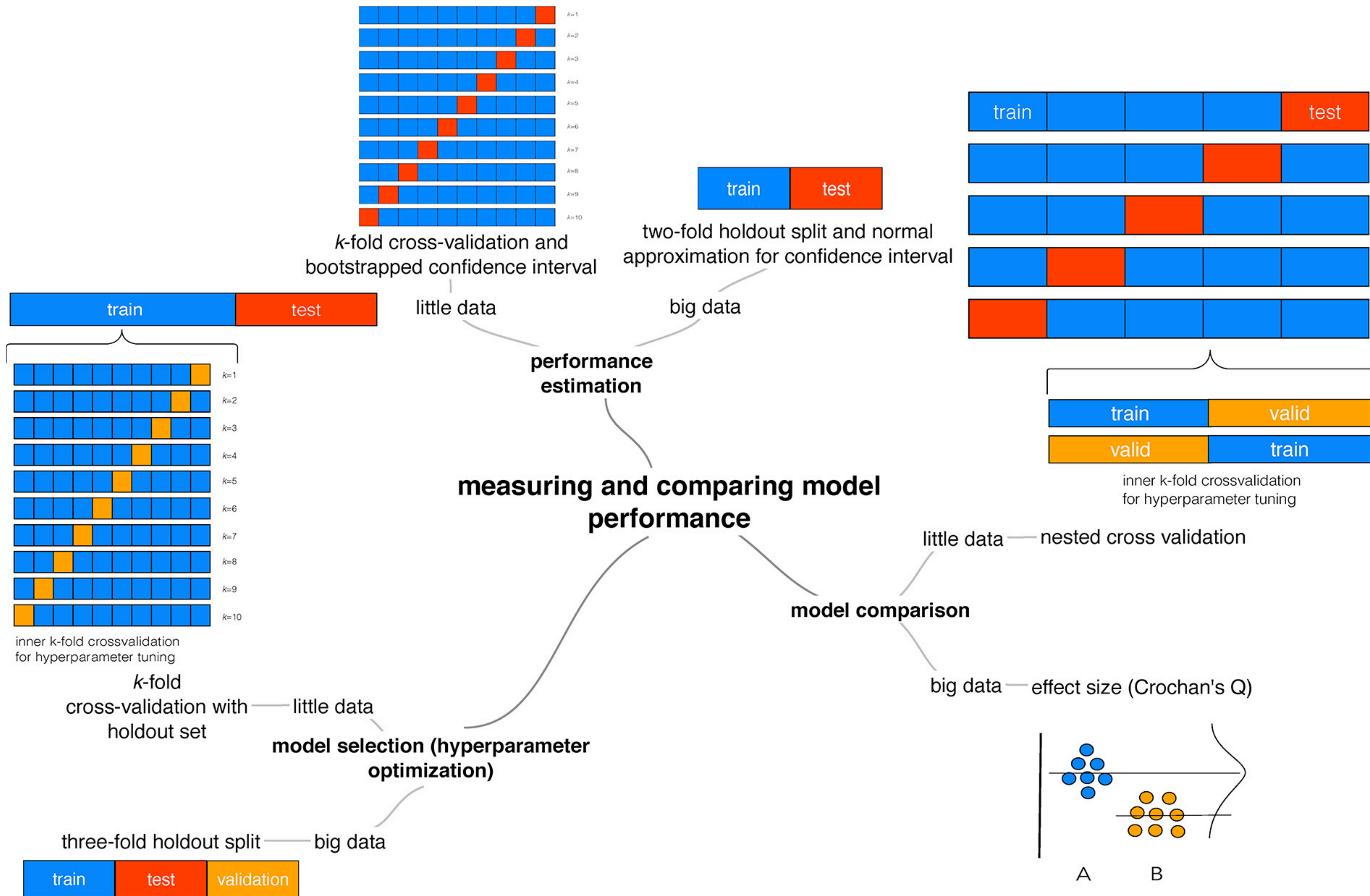
- We test different hyperparameter settings
- Estimate the error rate using cross validation
- Then pick the one with the lowest error



Using k-fold crossvalidation for hyperparameter tuning

- We test different hyperparameter settings
- Estimate the error rate using cross validation
- Then pick the one with the lowest error
- And only then (when we are done) get the error rate on the test set





5. Hyperparameters

Learning Objectives:

- Hyperparameter are non-trainable parameters that need to be tuned
- (Nested) Cross-validation can be used to optimize them
- K-fold cross-validation

6. Kernel learning

Learning Objectives:

- The kernel trick allows us to use the training data as basis
- We do not have to pre-compute non-linearities
- Kernel is a measure of similarity
- Kernel corresponds to infinite feature space (without computing all infinite polynomials!)

Features as basis set

(Non) Linear regression
primal picture

$$f(x) = \sum_i^N w_i b_i(x)$$

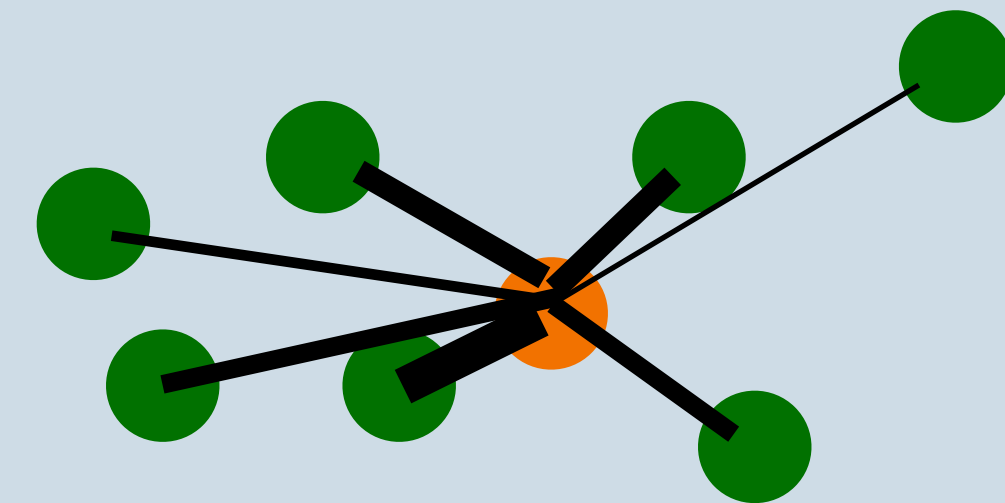
$b_i(x)$ basis functions

uptake = $w_1 \cdot \text{density}$
+ $w_2 \cdot \text{pore diameter}$
− $w_3 \text{ maximum charge}^2$

Training data as basis set

Kernel regression
dual picture

$$f(x) = \sum_i^N w_i K(x, x_i)$$

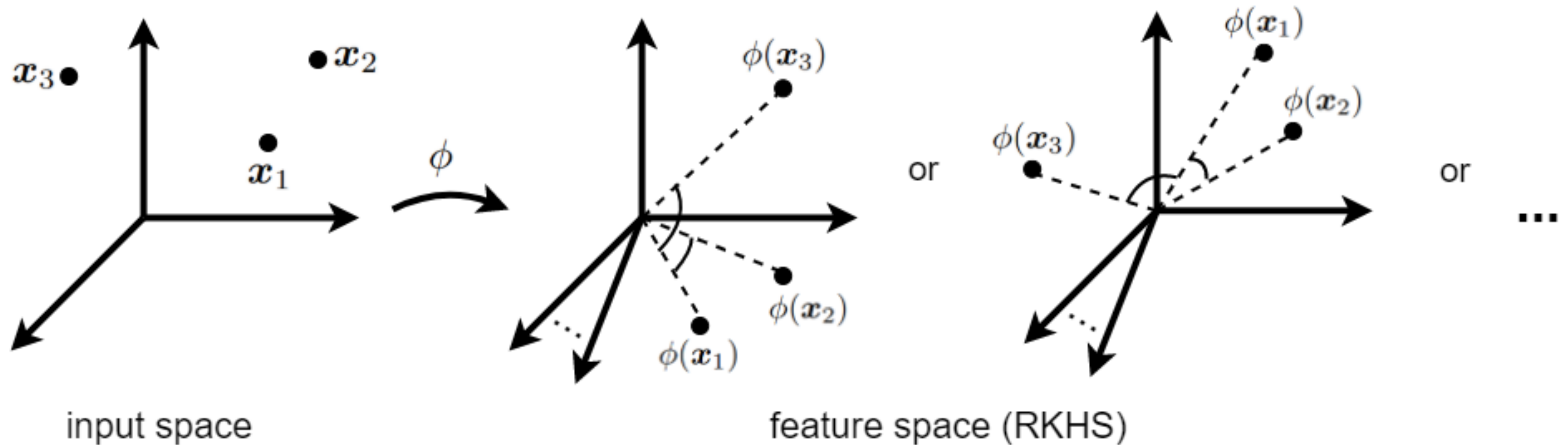


● train example
● query

Consider similarity to *all*
training instances

Using training data as basis set

Explicit location in feature space not necessarily known, *but relative similarity*



How do we get there?

Regularized linear regression

$$\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} \left(\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right)$$

How do we get there?

Regularized linear regression $\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} \left(\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right)$

Recall normal equation $\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$

How do we get there?

Regularized linear regression

$$\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} \left(\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right)$$

Recall normal equation

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Rewrite

$$\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

How do we get there?

Regularized linear regression $\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} (\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2)$

Recall normal equation $\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$

Rewrite $\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$

Define $\mathbf{a} := (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$

How do we get there?

Regularized linear regression

$$\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} (\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2)$$

Recall normal equation

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Rewrite

$$\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Define

$$\mathbf{a} := (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Using this definition

$$\mathbf{w} = \mathbf{X}^T \mathbf{a}$$

How do we get there?

Regularized linear regression

$$\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} \left(\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right)$$

Recall normal equation

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Rewrite

$$\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Define

$$\mathbf{a} := (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Using this definition

$$\mathbf{w} = \mathbf{X}^T \mathbf{a}$$

Rewrite inference equation

$$\hat{y}_{ML}(x^*) = x^* \mathbf{w} = \begin{bmatrix} x_1^* & \dots & x_p^* \end{bmatrix} \mathbf{X}^T \mathbf{a}$$

How do we get there?

Regularized linear regression

$$\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} \left(\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right)$$

Recall normal equation

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Rewrite

$$\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Define

$$\mathbf{a} := (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Using this definition

$$\mathbf{w} = \mathbf{X}^T \mathbf{a}$$

Rewrite inference equation

$$\begin{aligned} \hat{y}_{ML}(x^*) &= x^* \mathbf{w} = \begin{bmatrix} x_1^* & \dots & x_p^* \end{bmatrix} \mathbf{X}^T \mathbf{a} \\ &= \begin{bmatrix} x_1^* & \dots & x_p^* \end{bmatrix} \begin{bmatrix} x_1^1 & \dots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^p & \dots & x_n^p \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \end{aligned}$$

How do we get there?

Regularized linear regression

$$\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L} = \arg \min_{\mathbf{w}} \left(\|\hat{\mathbf{y}}_{ML} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right)$$

Recall normal equation

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Rewrite

$$\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Define

$$\mathbf{a} := (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Using this definition

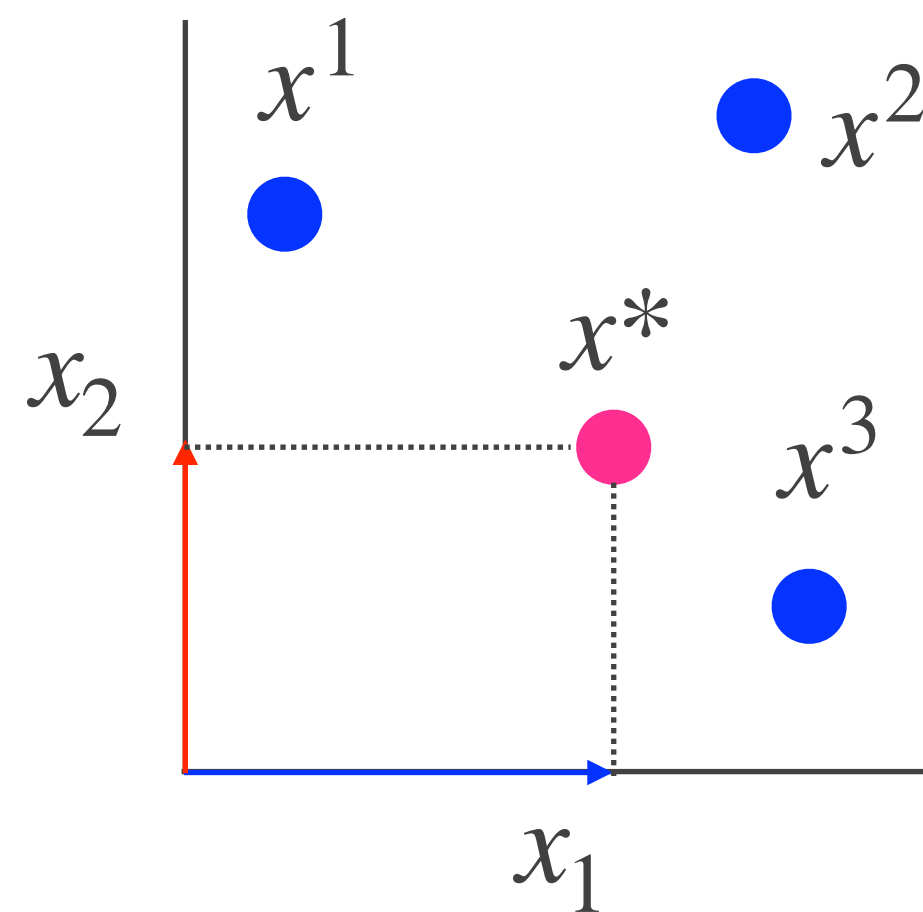
$$w = X^T a$$

Rewrite inference equation

$$\begin{aligned} \hat{y}_{ML}(x^*) &= x^* w = \begin{bmatrix} x_1^* & \dots & x_p^* \end{bmatrix} X^T a \\ &= \begin{bmatrix} x_1^* & \dots & x_p^* \end{bmatrix} \begin{bmatrix} x_1^1 & \dots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^p & \dots & x_n^p \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \\ &= \sum_{i=1}^n x^* x_i^T a_i = \sum_{i=1}^n K(x^*, x_i) a_i \end{aligned}$$

Contrasting Feature Basis and Data Basis

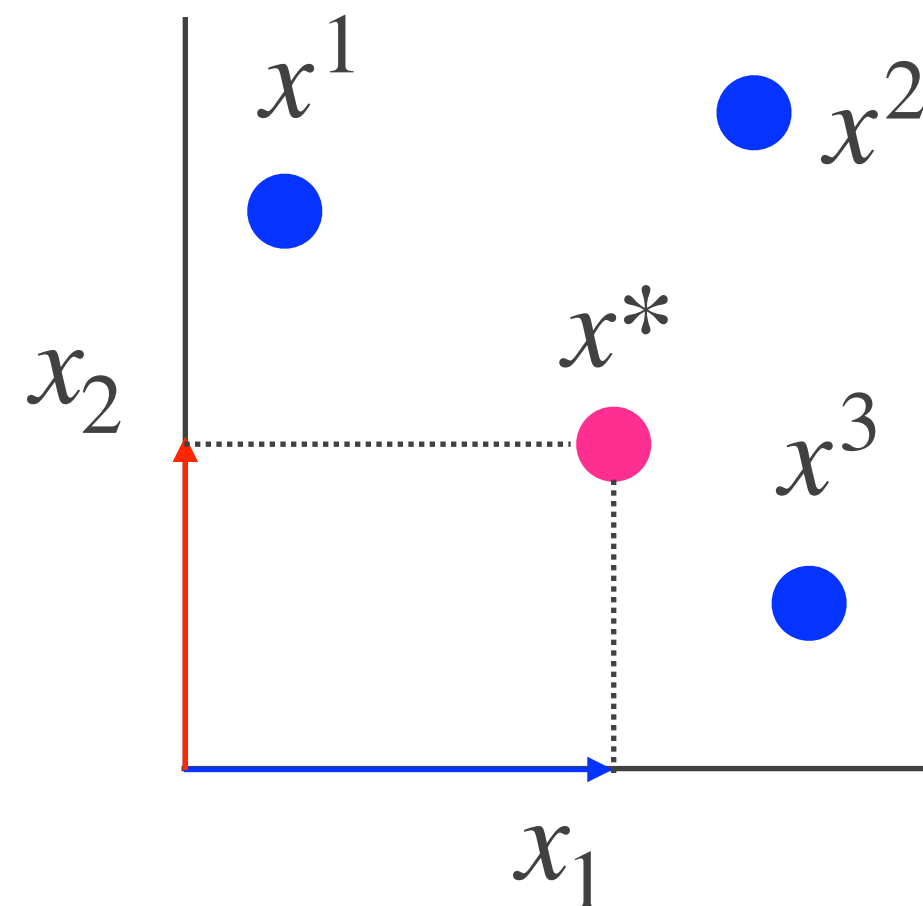
$$\begin{aligned}\hat{y}_{ML}(x^*) &= x^* w = \begin{bmatrix} x_1^* & x_2^* \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = x_1^* w_1 + x_2^* w_2 \\ &= \begin{bmatrix} x_1^* & x_2^* \end{bmatrix} \begin{bmatrix} x_1 & 0 \\ 0 & x_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}\end{aligned}$$



We describe the query point using the features

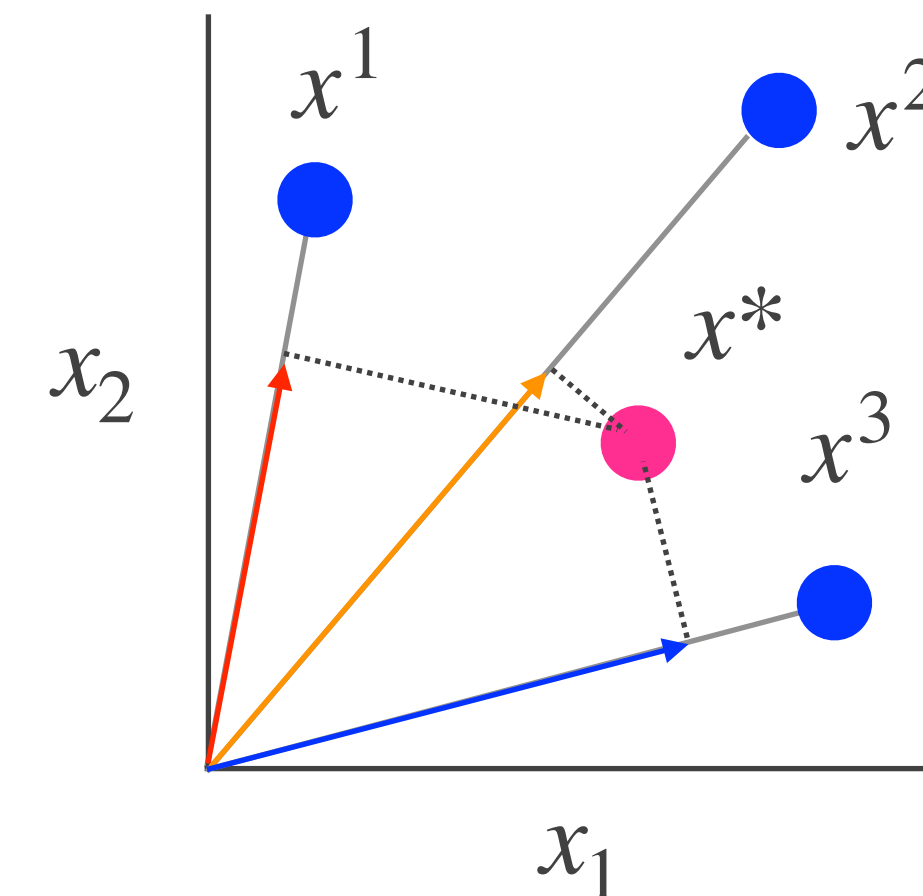
Contrasting Feature Basis and Data Basis

$$\begin{aligned}\hat{y}_{ML}(x^*) &= x^* w = \begin{bmatrix} x_1^* & x_2^* \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = x_1^* w_1 + x_2^* w_2 \\ &= \begin{bmatrix} x_1^* & x_2^* \end{bmatrix} \begin{bmatrix} x_1 & 0 \\ 0 & x_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}\end{aligned}$$



We describe the query point using the features

$$\begin{aligned}\hat{y}_{ML}(x^*) &= \begin{bmatrix} 1 & x_1^* & \dots & x_d^* \end{bmatrix} X^T a \\ &= \begin{bmatrix} x_1^* & x_2^* \end{bmatrix} \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 \\ x_2^1 & x_2^2 & x_1^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_3 \end{bmatrix}\end{aligned}$$



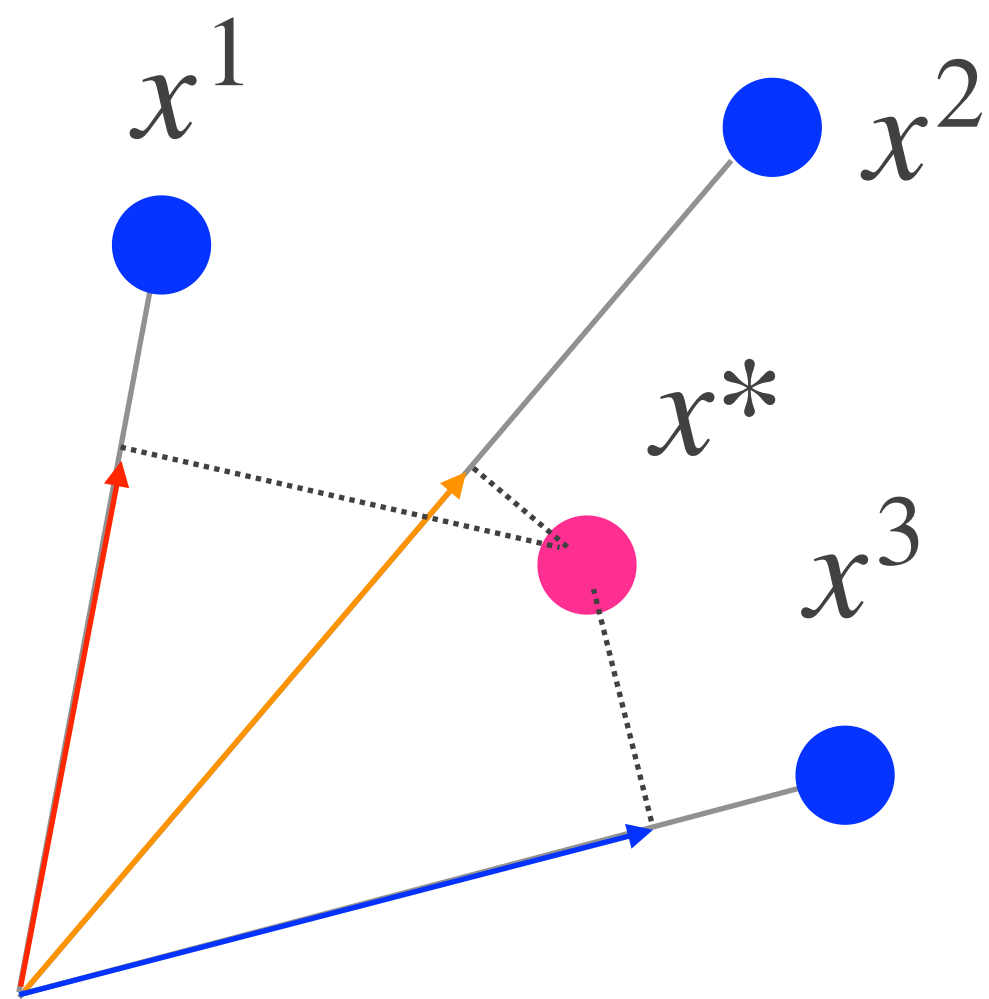
We describe the query point using the other points

Examples

Kernel is a notion of similarity, i.e. distance in feature space: $\hat{y}_{ML}(x^*) = \sum_{i=1}^n a_i K(x^*, x^i)$

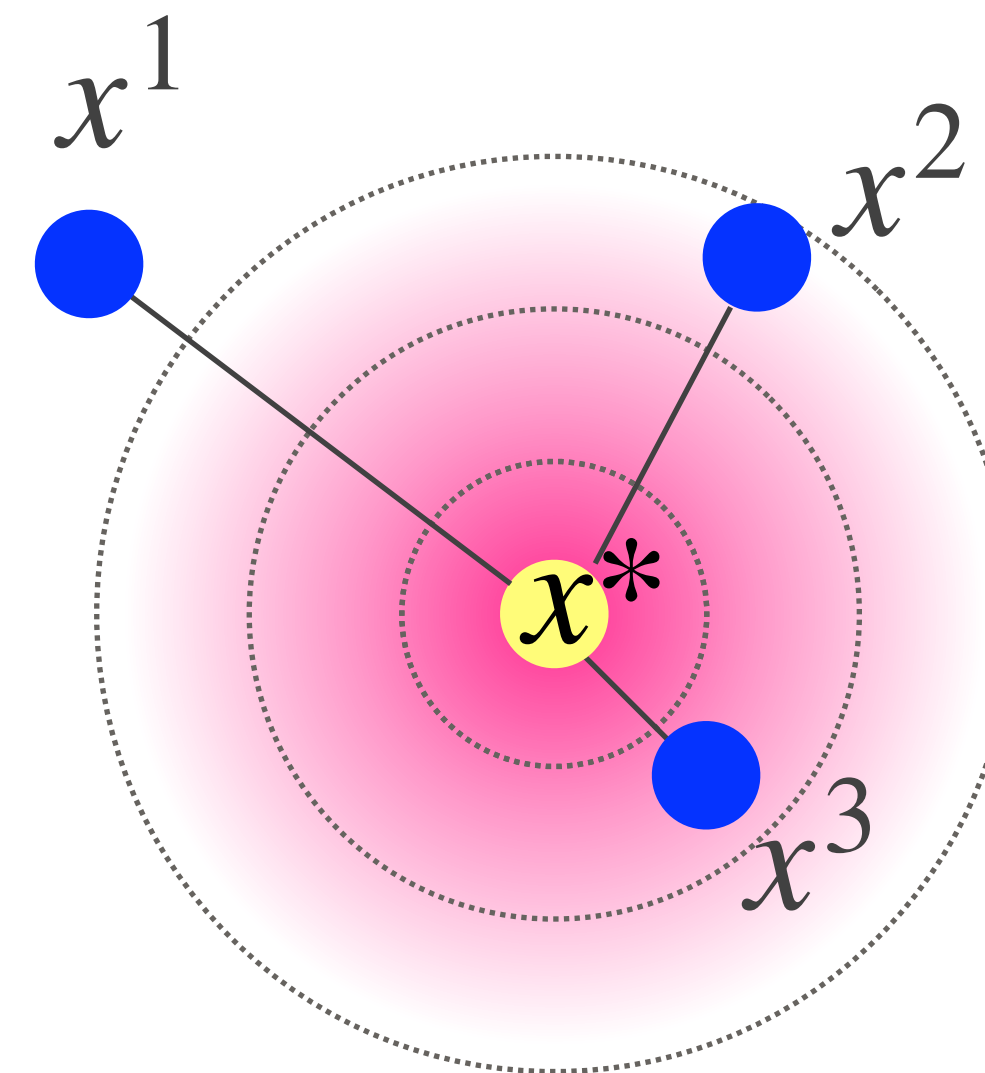
Linear basis

$$K(x^*, x_i) = \langle x^*, x_i \rangle$$



Radial basis

$$K(x^*, x_i) = \exp(-\gamma \|x^* - x_i\|_2^2)$$



***Extra:* Infinite dimensional feature space via kernels**

$$\begin{aligned}k_{\text{RBF}}(\mathbf{x}, \mathbf{y}) &= \exp \left(-\frac{1}{2} \|\mathbf{x} - \mathbf{y}\|^2 \right) \\&= \exp \left(-\frac{1}{2} \langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle \right) \\&\triangleq \exp \left(-\frac{1}{2} [\langle \mathbf{x}, \mathbf{x} - \mathbf{y} \rangle - \langle \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle] \right) \\&\triangleq \exp \left(-\frac{1}{2} [\langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{x}, \mathbf{y} \rangle - [\langle \mathbf{y}, \mathbf{x} \rangle - \langle \mathbf{y}, \mathbf{y} \rangle]] \right) \\&= \exp \left(-\frac{1}{2} [\langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle] \right) \\&= \exp \left(-\frac{1}{2} \|\mathbf{x}\|^2 \right) \exp \left(-\frac{1}{2} \|\mathbf{y}\|^2 \right) \exp(-2\langle \mathbf{x}, \mathbf{y} \rangle)\end{aligned}$$

$$\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$$

***Extra:* Infinite dimensional feature space via kernels**

$$\begin{aligned}k_{\text{RBF}}(\mathbf{x}, \mathbf{y}) &= \exp \left(-\frac{1}{2} \|\mathbf{x} - \mathbf{y}\|^2 \right) \\&= \exp \left(-\frac{1}{2} \langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle \right) \\&\triangleq \exp \left(-\frac{1}{2} [\langle \mathbf{x}, \mathbf{x} - \mathbf{y} \rangle - \langle \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle] \right) \\&\triangleq \exp \left(-\frac{1}{2} [\langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{x}, \mathbf{y} \rangle - [\langle \mathbf{y}, \mathbf{x} \rangle - \langle \mathbf{y}, \mathbf{y} \rangle]] \right) \\&= \exp \left(-\frac{1}{2} [\langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle] \right) \\&= \exp \left(-\frac{1}{2} \|\mathbf{x}\|^2 \right) \exp \left(-\frac{1}{2} \|\mathbf{y}\|^2 \right) \exp(-2\langle \mathbf{x}, \mathbf{y} \rangle)\end{aligned}$$

$$C \equiv \exp \left(-\frac{1}{2} \|\mathbf{x}\|^2 \right) \exp \left(-\frac{1}{2} \|\mathbf{y}\|^2 \right)$$

$$e^{f(x)} = \sum_{r=0}^{\infty} \frac{[f(x)]^r}{r!}$$

$$\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$$

***Extra:* Infinite dimensional feature space via kernels**

$$\begin{aligned}k_{\text{RBF}}(\mathbf{x}, \mathbf{y}) &= \exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2\right) \\&= \exp\left(-\frac{1}{2}\langle\mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y}\rangle\right) \\&\triangleq \exp\left(-\frac{1}{2}[\langle\mathbf{x}, \mathbf{x} - \mathbf{y}\rangle - \langle\mathbf{y}, \mathbf{x} - \mathbf{y}\rangle]\right) \\&\triangleq \exp\left(-\frac{1}{2}[\langle\mathbf{x}, \mathbf{x}\rangle - \langle\mathbf{x}, \mathbf{y}\rangle - [\langle\mathbf{y}, \mathbf{x}\rangle - \langle\mathbf{y}, \mathbf{y}\rangle]]\right) \\&= \exp\left(-\frac{1}{2}[\langle\mathbf{x}, \mathbf{x}\rangle + \langle\mathbf{y}, \mathbf{y}\rangle - 2\langle\mathbf{x}, \mathbf{y}\rangle]\right) \\&= \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{y}\|^2\right) \exp(-2\langle\mathbf{x}, \mathbf{y}\rangle)\end{aligned}$$

$$\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$$

$$C \equiv \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{y}\|^2\right)$$

$$e^{f(x)} = \sum_{r=0}^{\infty} \frac{[f(x)]^r}{r!}$$

$$\begin{aligned}k_{\text{RBF}}(\mathbf{x}, \mathbf{y}) &= C \exp(-2\langle\mathbf{x}, \mathbf{y}\rangle) \\&= C \sum_{r=0}^{\infty} \frac{\langle\mathbf{x}, \mathbf{y}\rangle^r}{r!} \\&= C \sum_r \frac{k_{\text{poly}(r)}(\mathbf{x}, \mathbf{y})}{r!}\end{aligned}$$

6. Kernel learning

Learning Objectives:

- The kernel trick allows us to use the training data as basis
- We do not have to pre-compute non-linearities
- Kernel is a measure of similarity
- Kernel corresponds to infinite feature space (without computing all infinite polynomials!)

7. Feature importance

Learning Objectives:

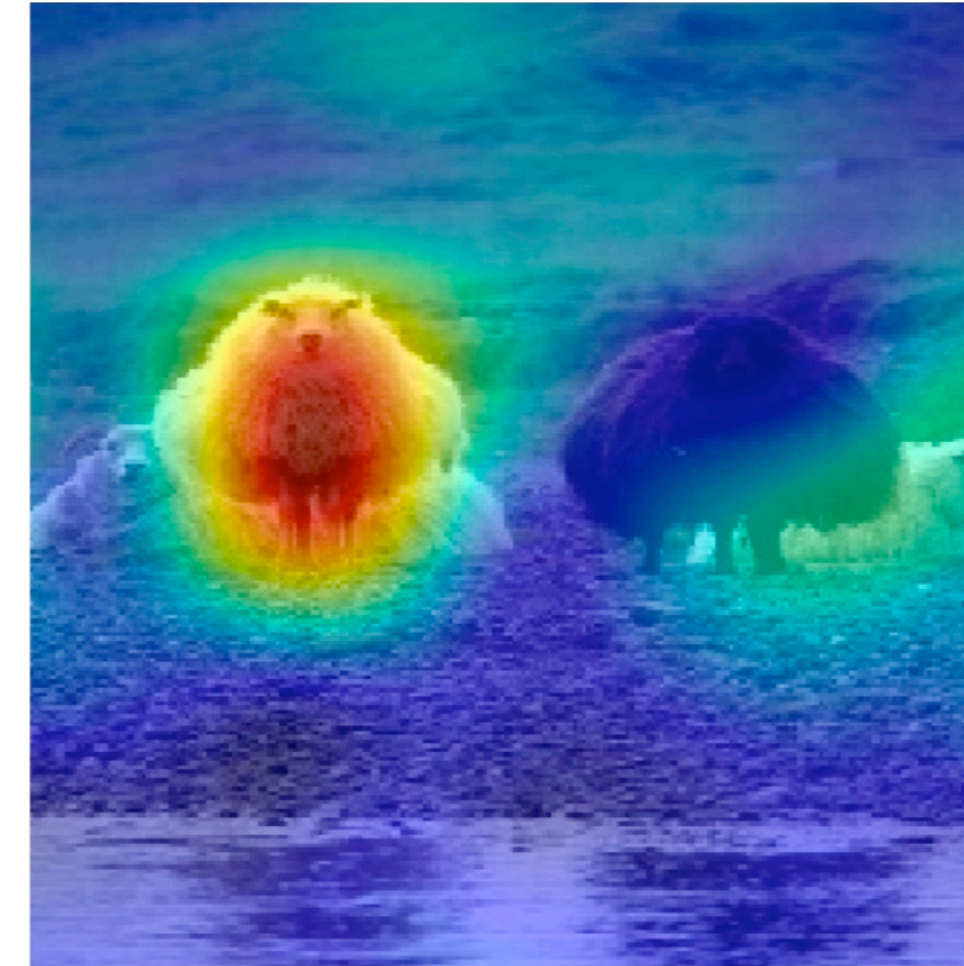
- Motivation for explainable ML
- Permutation feature importance

Motivation

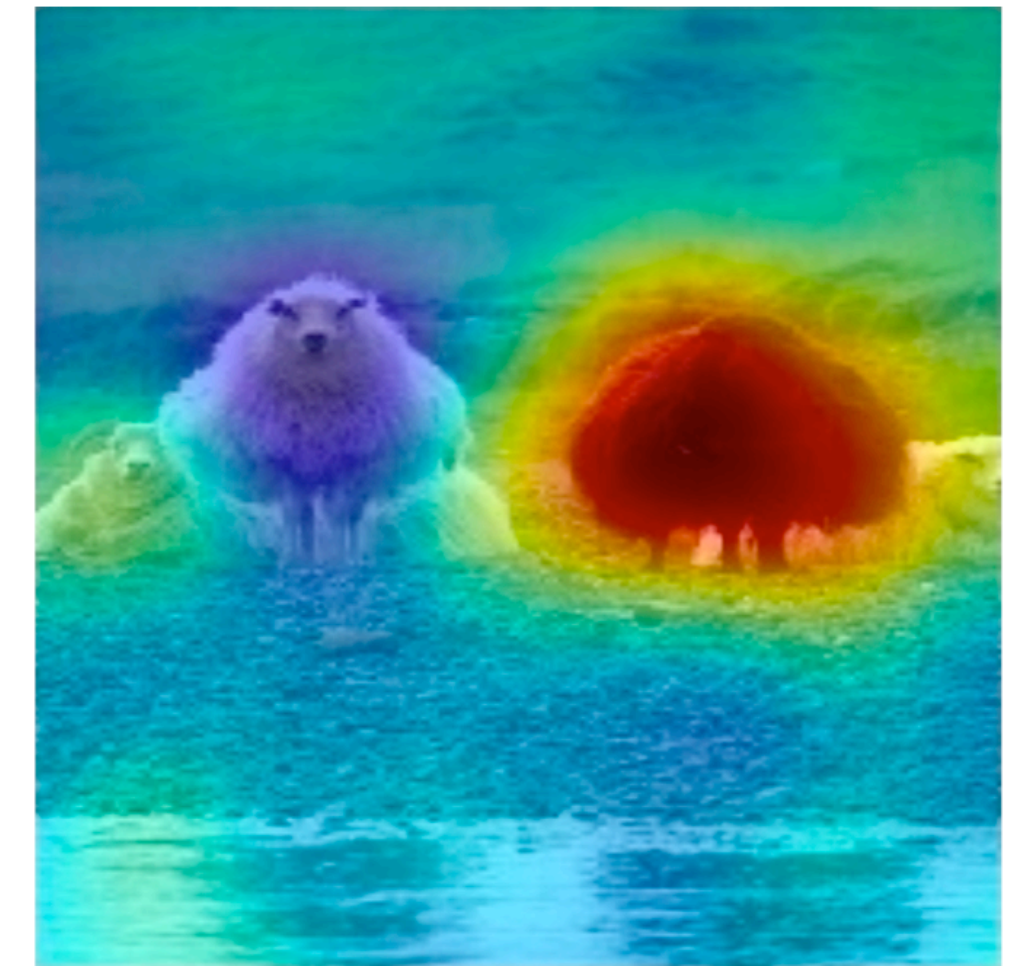
Debugging models



(a) Sheep - 26%, Cow - 17%



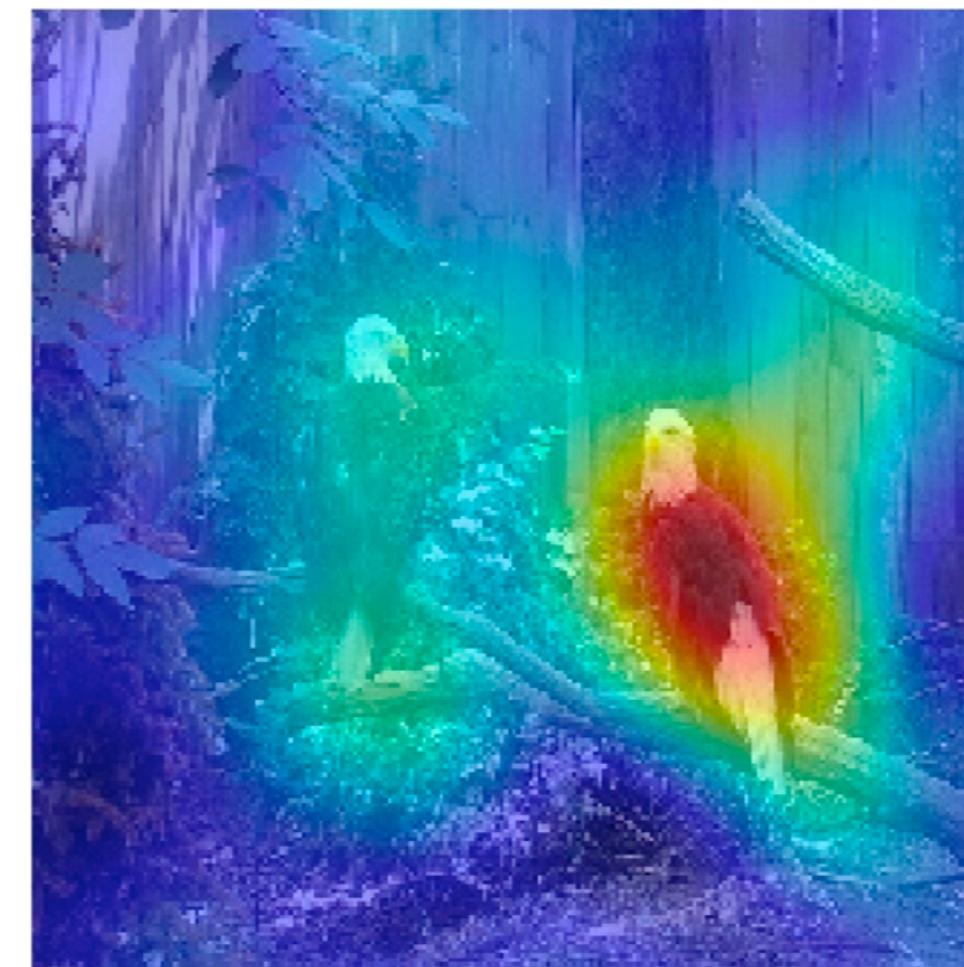
(b) Importance map of '*sheep*'



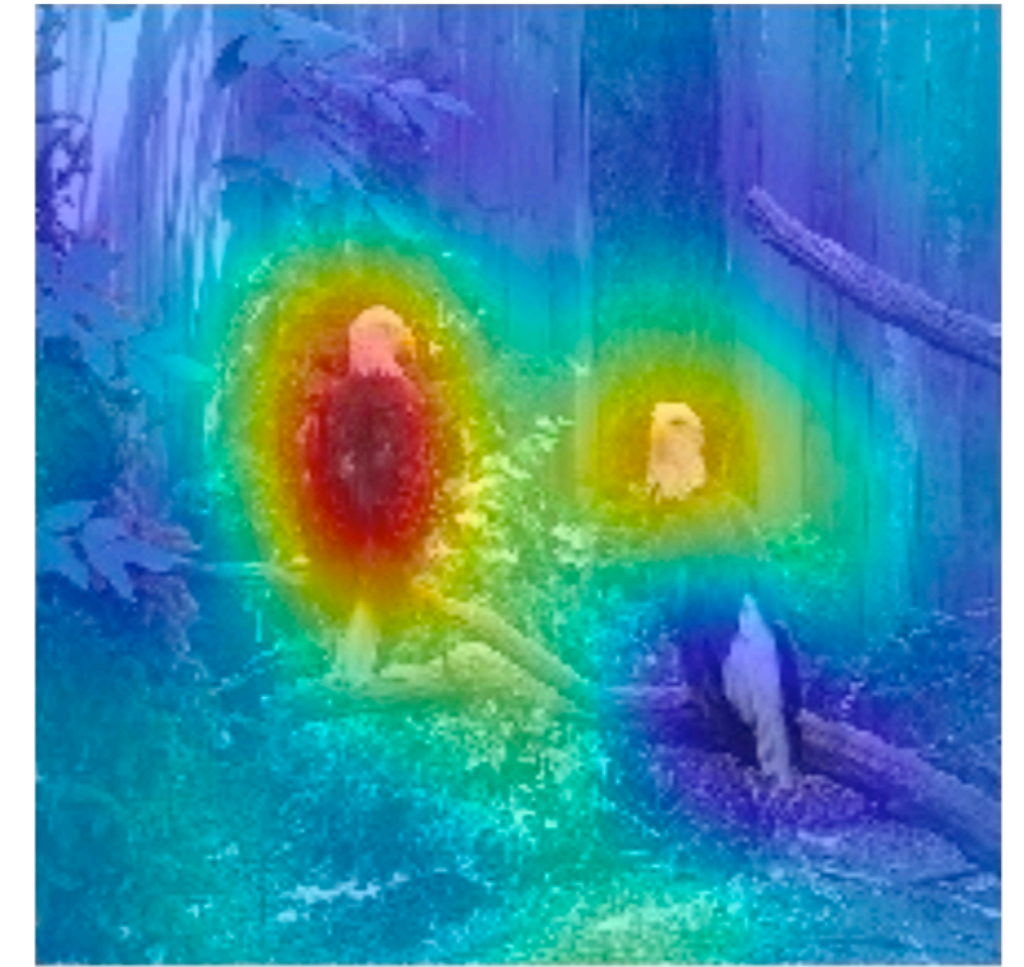
(c) Importance map of '*cow*'



(d) Bird - 100%, Person - 39%



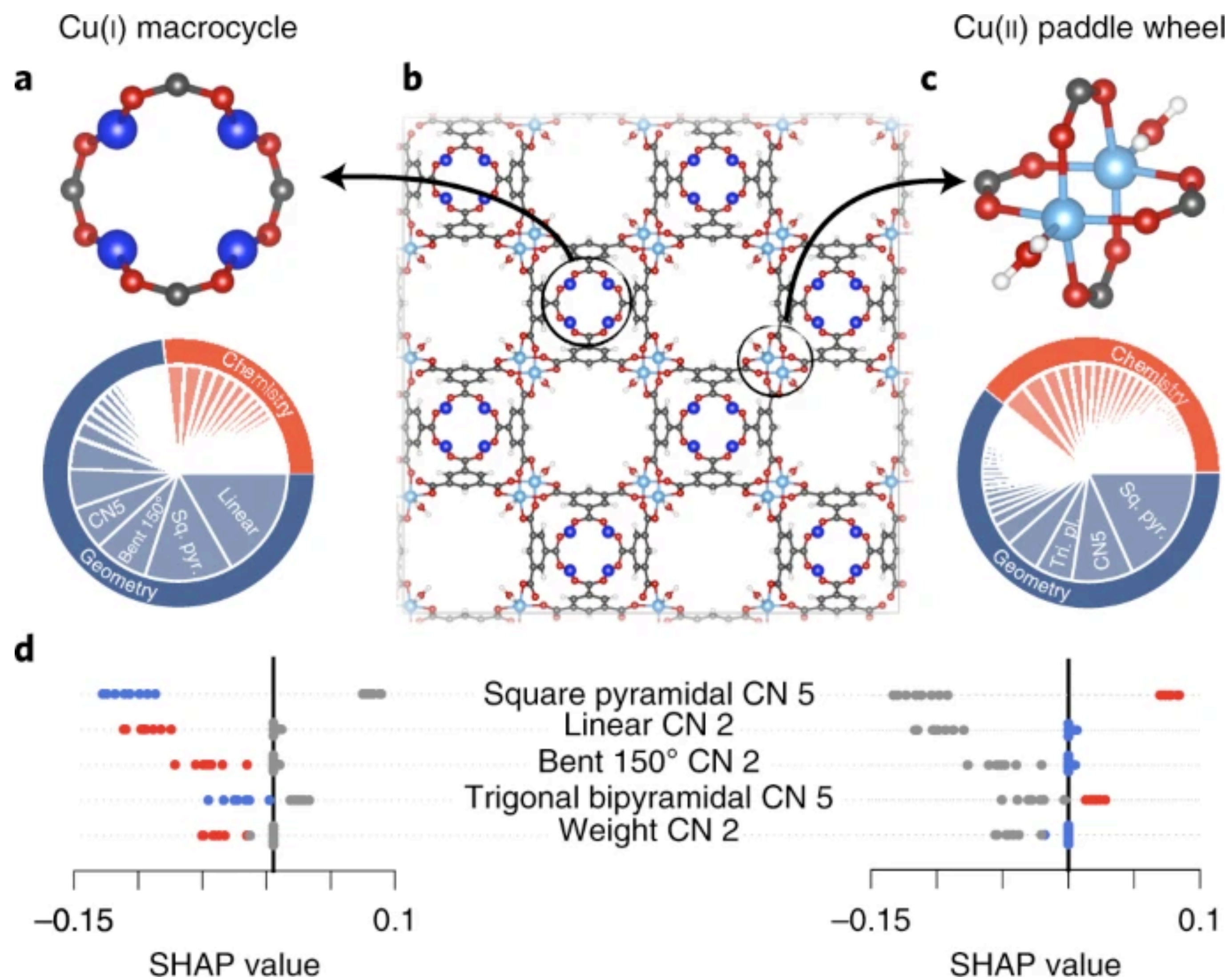
(e) Importance map of '*bird*'



(f) Importance map of '*person*'

Motivation

Chemical insights into system



Permutation feature importance

Look at performance drop when trained on shuffled features

0. Build the model

$$X = \begin{pmatrix} x^{1,1} & x^{1,1} & \dots & x^{1,p} \\ x^{2,1} & x^{2,1} & \dots & x^{2,p} \\ \vdots & \ddots & & \vdots \\ x^{n,1} & x^{n,1} & \dots & x^{n,p} \end{pmatrix} y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix} \hat{y}_{ML} = f(X), \mathcal{L}(y, \hat{y}_{ML})$$

Permutation feature importance

Look at performance drop when trained on shuffled features

0. Build the model

$$X = \begin{pmatrix} x^{1,1} & x^{1,1} & \dots & x^{1,p} \\ x^{2,1} & x^{2,1} & \dots & x^{2,p} \\ \vdots & \ddots & & \vdots \\ x^{n,1} & x^{n,1} & \dots & x^{n,p} \end{pmatrix} \quad y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix} \quad \hat{y}_{ML} = f(X), \mathcal{L}(y, \hat{y}_{ML})$$

1. Estimate the model error

$$e^{\text{orig}} = \mathcal{L}(y, \hat{y}_{ML})$$

Permutation feature importance

Look at performance drop when trained on shuffled features

0. Build the model

$$X = \begin{pmatrix} x^{1,1} & x^{1,1} & \dots & x^{1,p} \\ x^{2,1} & x^{2,1} & \dots & x^{2,p} \\ \vdots & \ddots & & \vdots \\ x^{n,1} & x^{n,1} & \dots & x^{n,p} \end{pmatrix} \quad y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix} \quad \hat{y}_{ML} = f(X), \mathcal{L}(y, \hat{y}_{ML})$$

1. Estimate the model error

$$e^{\text{orig}} = \mathcal{L}(y, \hat{y}_{ML})$$

2. For each feature $j=1,\dots,p$:

- Generate permuted feature matrix for feature (j)

$$X_j^{\text{perm}} = \begin{pmatrix} 1 & x^{1,1} & \dots & x^{50,j} & \dots & x^{1,p} \\ 1 & x^{2,1} & \dots & x^{n,j} & \dots & x^{2,p} \\ 1 & \vdots & \dots & \vdots & \ddots & \vdots \\ 1 & x^{n,1} & \dots & x^{2,j} & \dots & x^{n,p} \end{pmatrix}$$

Permutation feature importance

Look at performance drop when trained on shuffled features

0. Build the model

$$X = \begin{pmatrix} x^{1,1} & x^{1,1} & \dots & x^{1,p} \\ x^{2,1} & x^{2,1} & \dots & x^{2,p} \\ \vdots & \ddots & & \vdots \\ x^{n,1} & x^{n,1} & \dots & x^{n,p} \end{pmatrix} \quad y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix} \quad \hat{y}_{ML} = f(X), \mathcal{L}(y, \hat{y}_{ML})$$

1. Estimate the model error

$$e^{\text{orig}} = \mathcal{L}(y, \hat{y}_{ML})$$

2. For each feature $j=1,\dots,p$:

- Generate permuted feature matrix for feature (j)
- Estimate the error with permuted features

$$X_j^{\text{perm}} = \begin{pmatrix} 1 & x^{1,1} & \dots & x^{50,j} & \dots & x^{1,p} \\ 1 & x^{2,1} & \dots & x^{n,j} & \dots & x^{2,p} \\ 1 & \vdots & \dots & \vdots & \ddots & \vdots \\ 1 & x^{n,1} & \dots & x^{2,j} & \dots & x^{n,p} \end{pmatrix}$$

$$e_j^{\text{perm}} = \mathcal{L}(y, f(x_j^{\text{perm}}))$$

Permutation feature importance

Look at performance drop when trained on shuffled features

0. Build the model

$$X = \begin{pmatrix} x^{1,1} & x^{1,1} & \dots & x^{1,p} \\ x^{2,1} & x^{2,1} & \dots & x^{2,p} \\ \vdots & \ddots & & \vdots \\ x^{n,1} & x^{n,1} & \dots & x^{n,p} \end{pmatrix} \quad y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix} \quad \hat{y}_{ML} = f(X), \mathcal{L}(y, \hat{y}_{ML})$$

1. Estimate the model error

$$e^{\text{orig}} = \mathcal{L}(y, \hat{y}_{ML})$$

2. For each feature $j=1,\dots,p$:

- Generate permuted feature matrix for feature (j)
- Estimate the error with permuted features
- Get the the performance difference

$$X_j^{\text{perm}} = \begin{pmatrix} 1 & x^{1,1} & \dots & x^{50,j} & \dots & x^{1,p} \\ 1 & x^{2,1} & \dots & x^{n,j} & \dots & x^{2,p} \\ 1 & \vdots & \dots & \vdots & \ddots & \vdots \\ 1 & x^{n,1} & \dots & x^{2,j} & \dots & x^{n,p} \end{pmatrix}$$

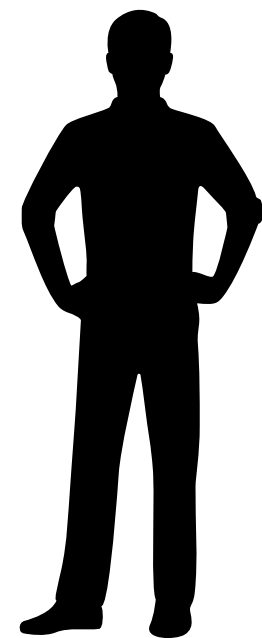
$$e_j^{\text{perm}} = \mathcal{L}(y, f(x_j^{\text{perm}}))$$

$$e_j^{\text{perm}} - e_j^{\text{orig}}$$

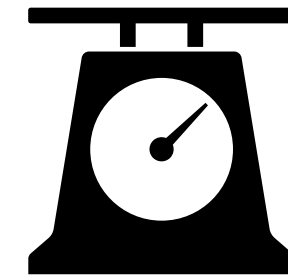
Permutation feature importance

Caveats with correlated features

Might generate unreasonable data points



height



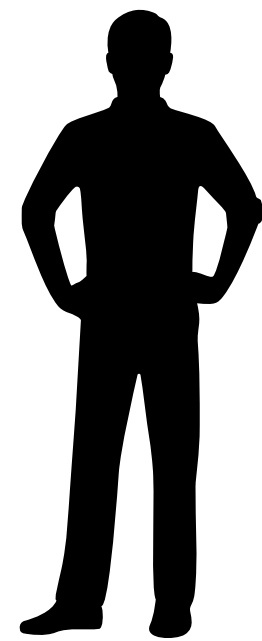
weight

Permutation feature importance

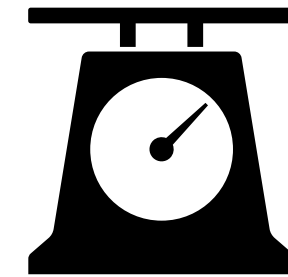
Caveats with correlated features

Might generate unreasonable data points

Importance of correlated features might be split



height



weight

```
data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
print("Accuracy on test data: {:.2f}".format(clf.score(X_test, y_test)))

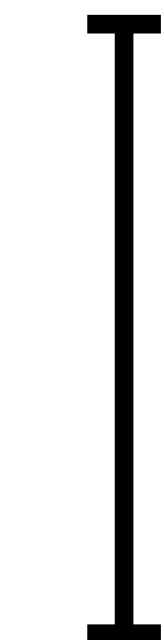
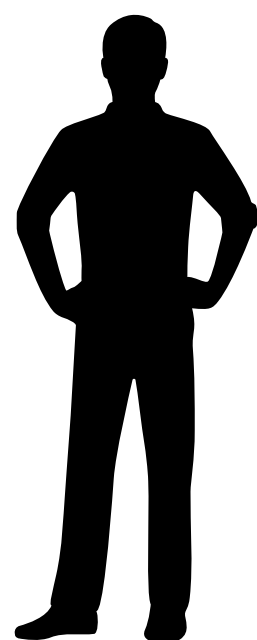
>> Accuracy on test data: 0.97
```

Permutation feature importance

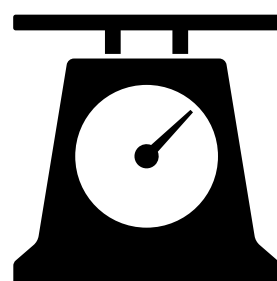
Caveats with correlated features

Might generate unreasonable data points

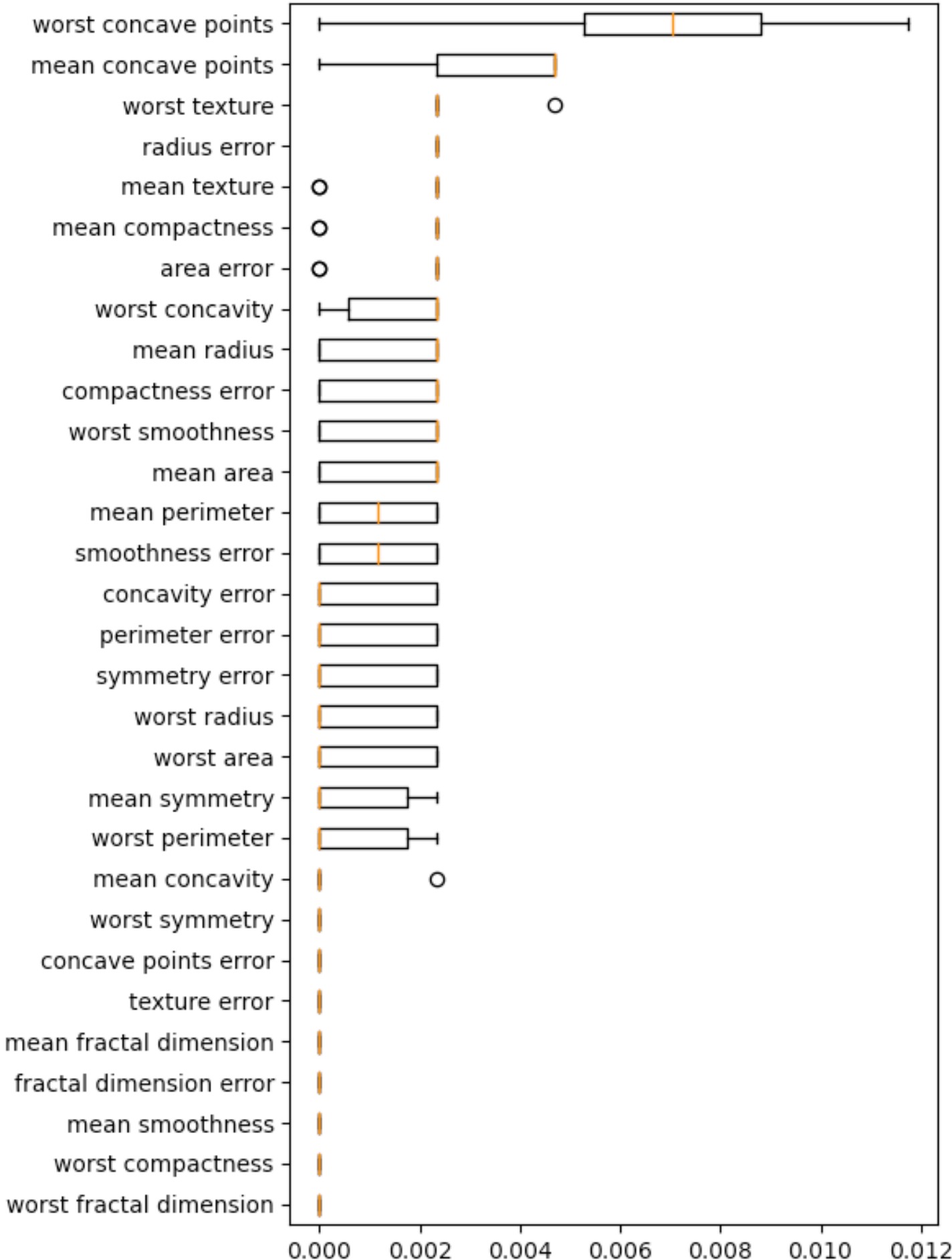
Importance of correlated features might be split



height



weight



7. Feature importance

Learning Objectives:

- Motivation for explainable ML
- Permutation feature importance

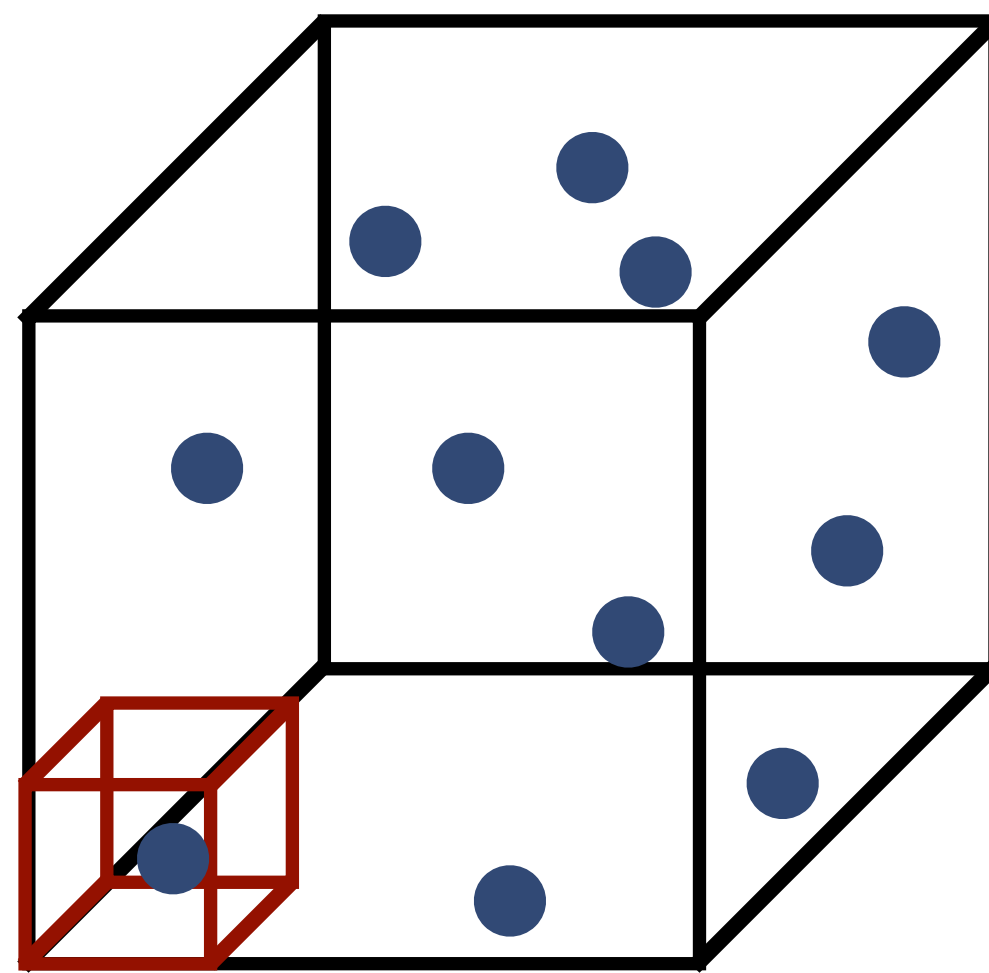
8. Feature Selection

Learning Objectives:

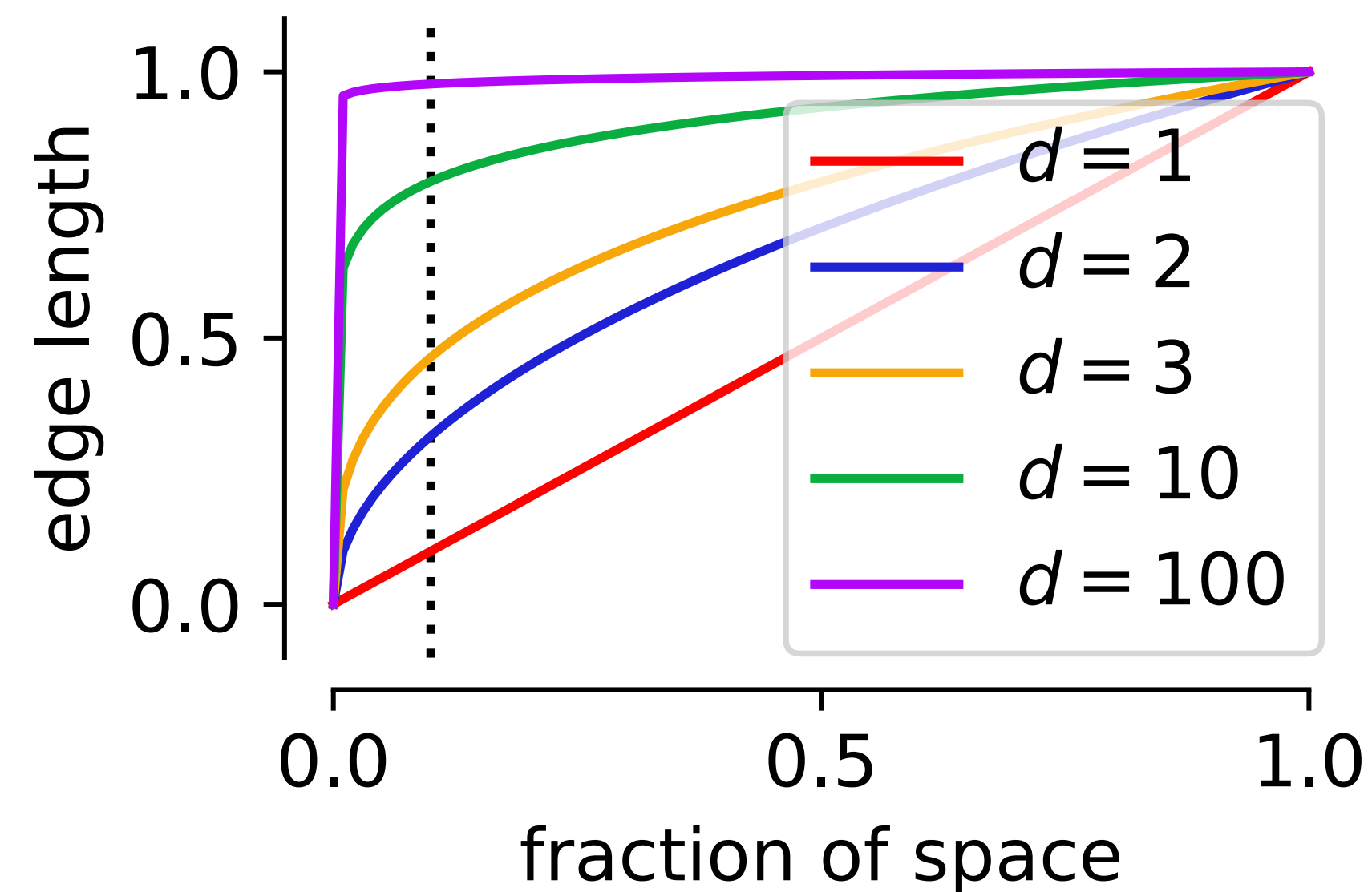
- Curse of dimensionality: No locality in high-dimensional spaces
- Filter heuristics
- LASSO

No locality in high dimensions

$$\text{edgelen}gth = \text{fraction of space}^{1/\text{dimensionality}}$$



unit cube

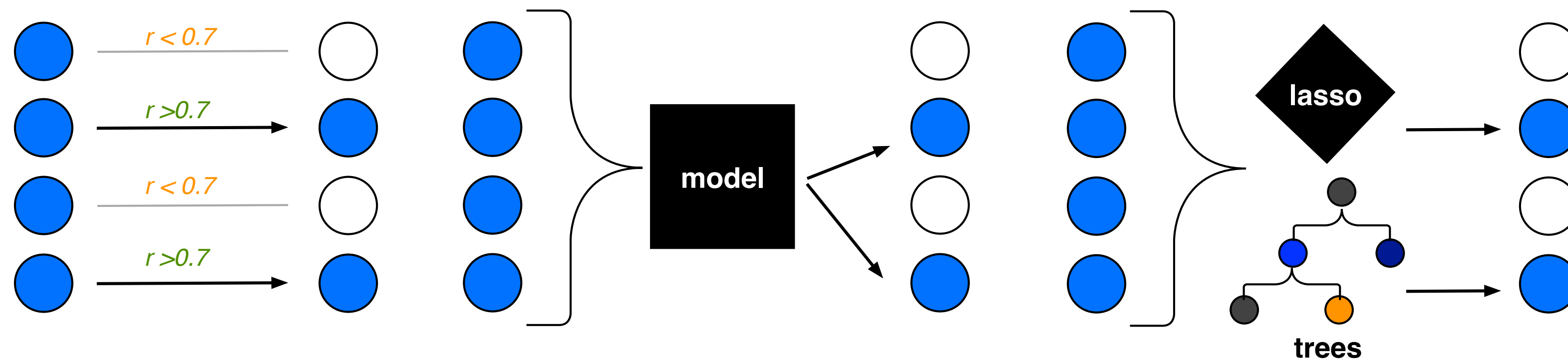


Methods that are based on similarity (k NN/KRR) might fail in high dimensional spaces!

We want to **reduce the dimensionality** of our feature matrix!

Feature selection

Reduce dimensionality of feature space by feature selection (compression)



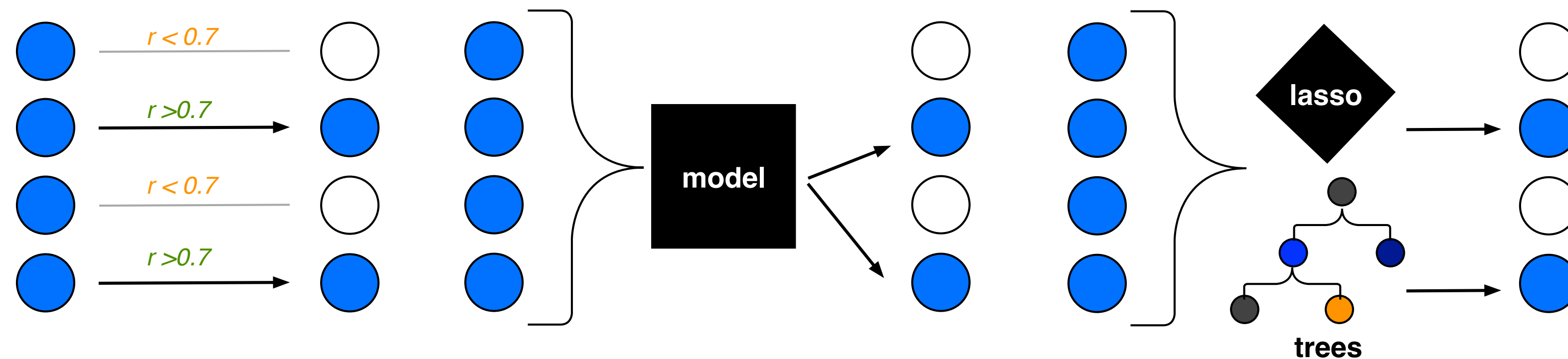
a Univariate filters.

b Wrapper methods.

c Shrinkage or direct.

Feature selection

Reduce dimensionality of feature space by feature selection (compression)

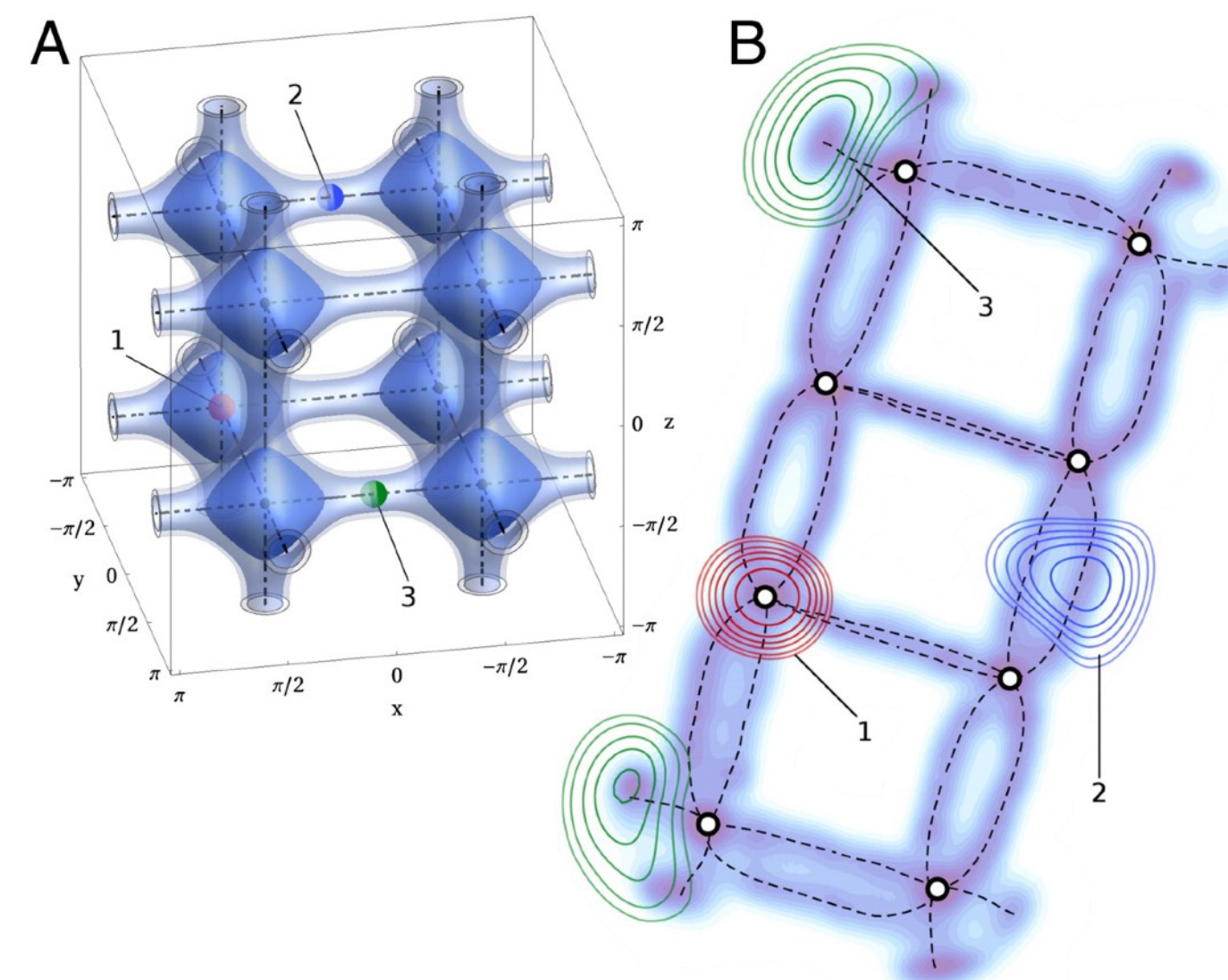


a Univariate filters.

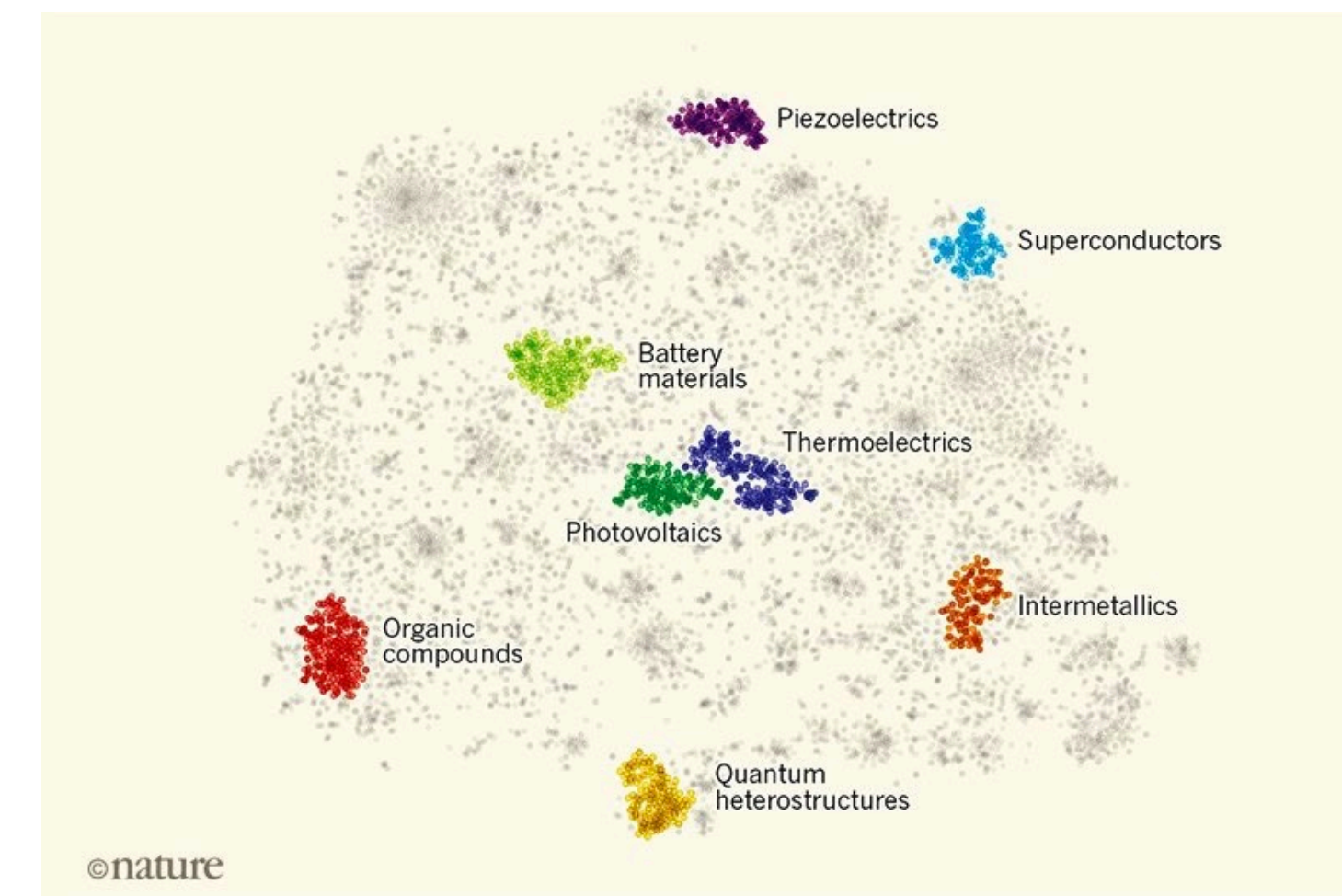
b Wrapper methods.

c Shrinkage or direct.

Reduce size of feature space by dimensionality reduction (feature projection)



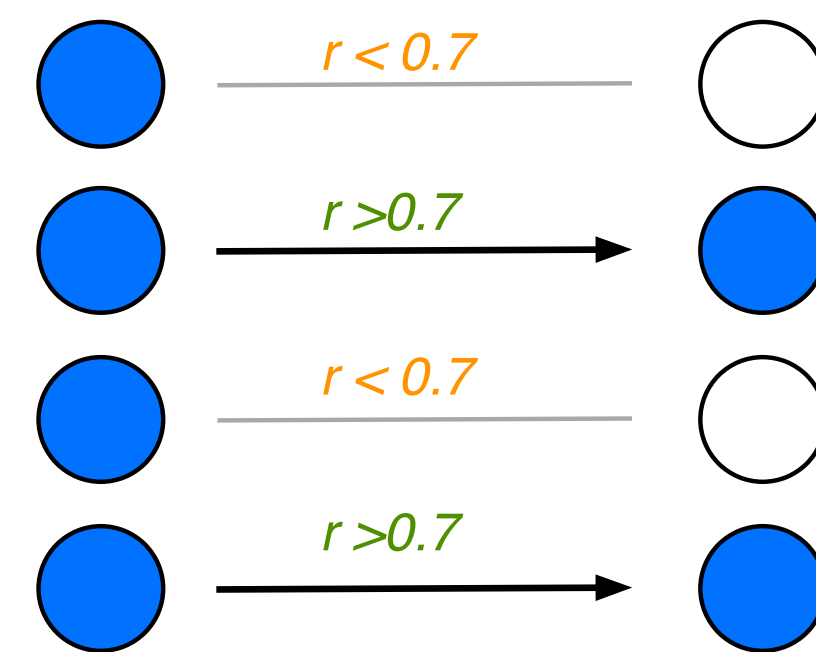
Visualize data and Materials Cartography



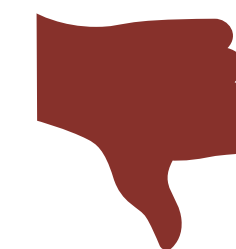
Feature projection

Janet, J. P.; Kulik, H. J. *J. Phys. Chem. A* **2017**, 121 (46), 8939–8954.
 Tribello, G. A.; Ceriotti, M.; Parrinello, M. *PNAS* **2012**, 109 (14), 5196–5201.

Feature Selection: Filter Methods

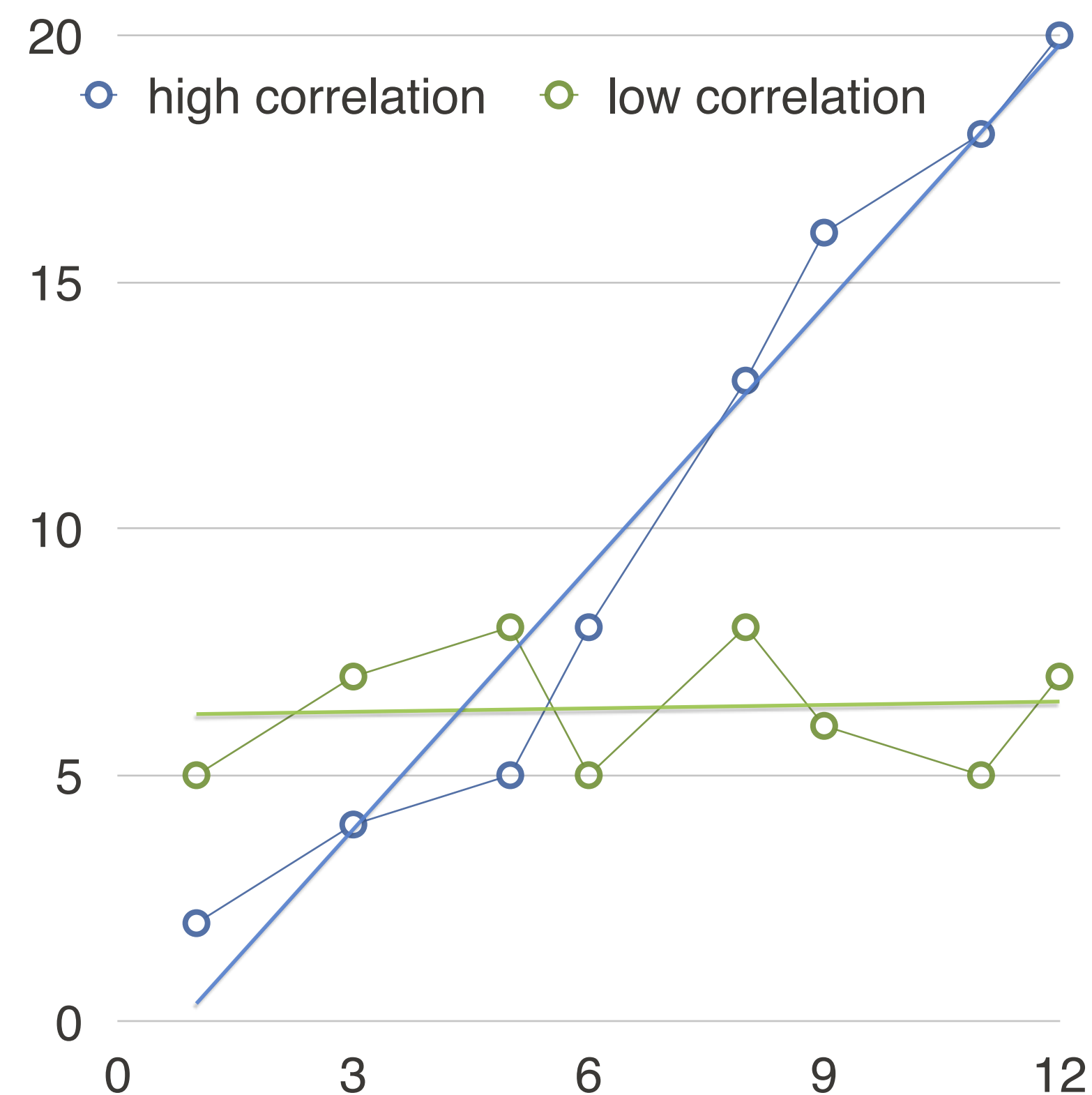


Easy and cheap

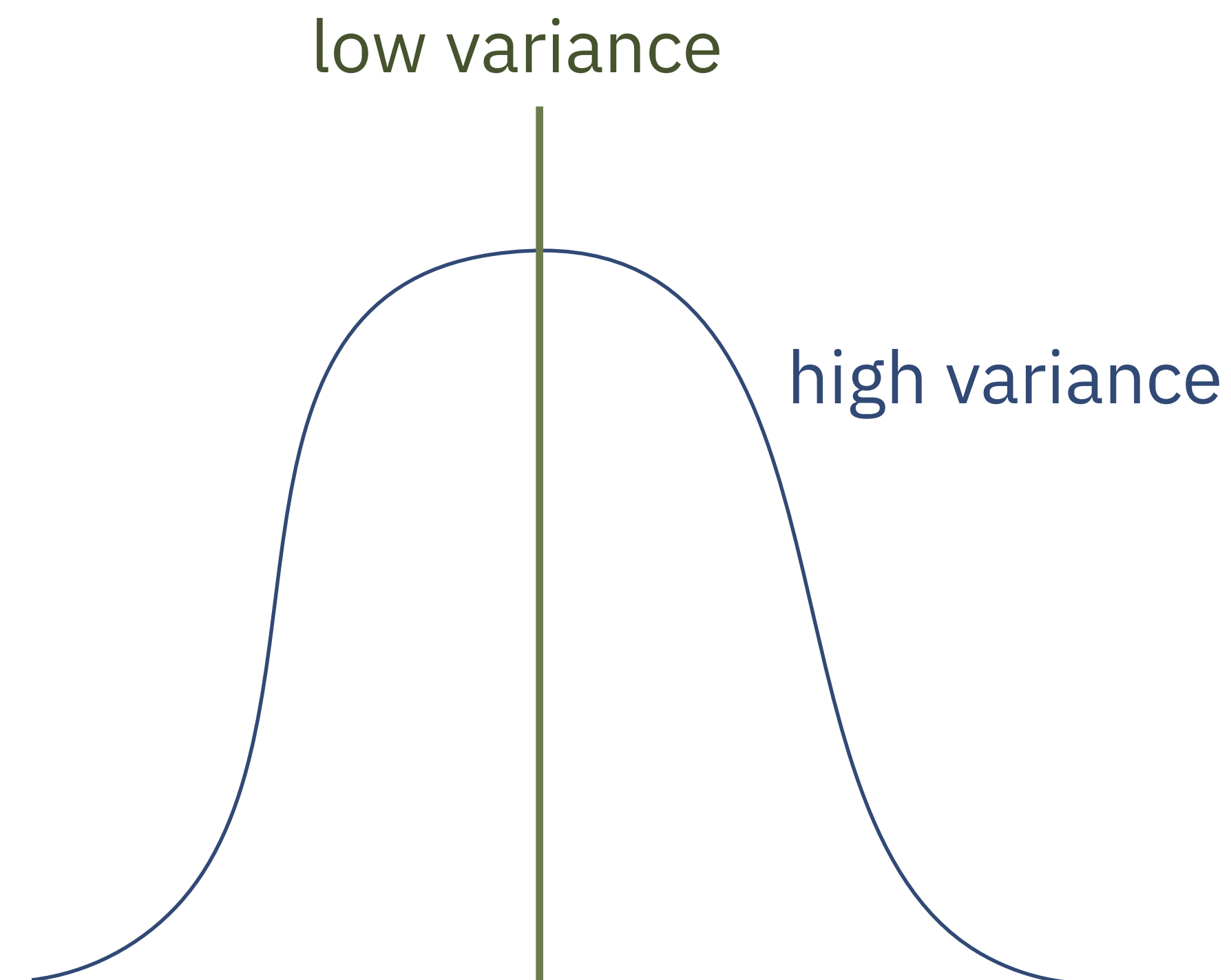


Interaction effects are not considered

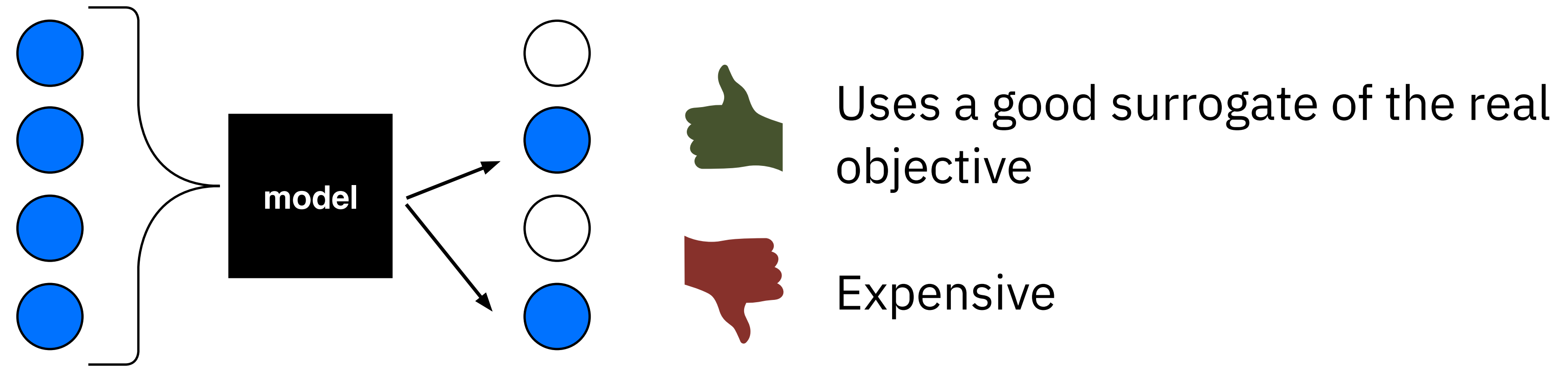
correlation threshold



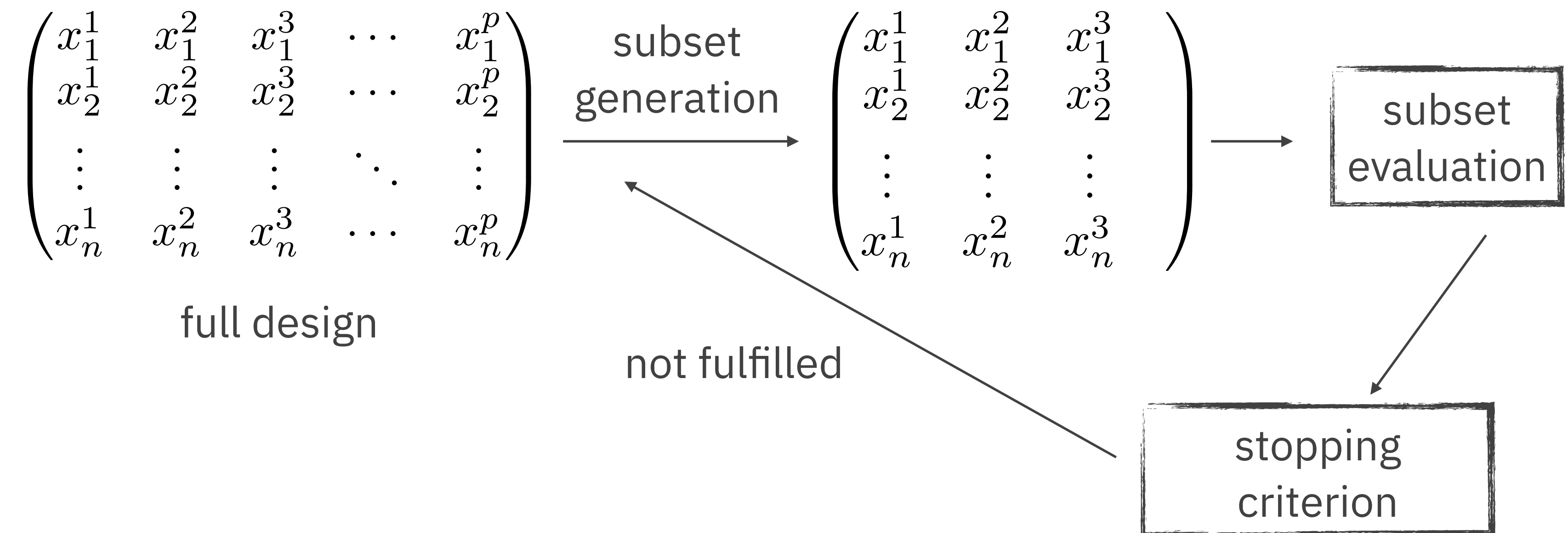
variance threshold



Feature Selection: Wrapper Methods



For example recursive feature addition or elimination



Feature Selection: Relaxing best subset selection

The basic problem: Best subset selection

$$\min_{\beta \in \mathbb{R}^p} \|y - X\beta\|^2 \text{ subject to } \|\beta\|_0 \leq k \quad \|\beta\|_0 = \sum_{j=1}^p 1\{\beta_j \neq 0\}$$

But this is our hard problem (NP hard) ...

... hence we relax the constraint to have a problem that is convex

... the Lasso gives use sparsity as the most feasible approximation to l_0

Feature Selection: Relaxing best subset selection

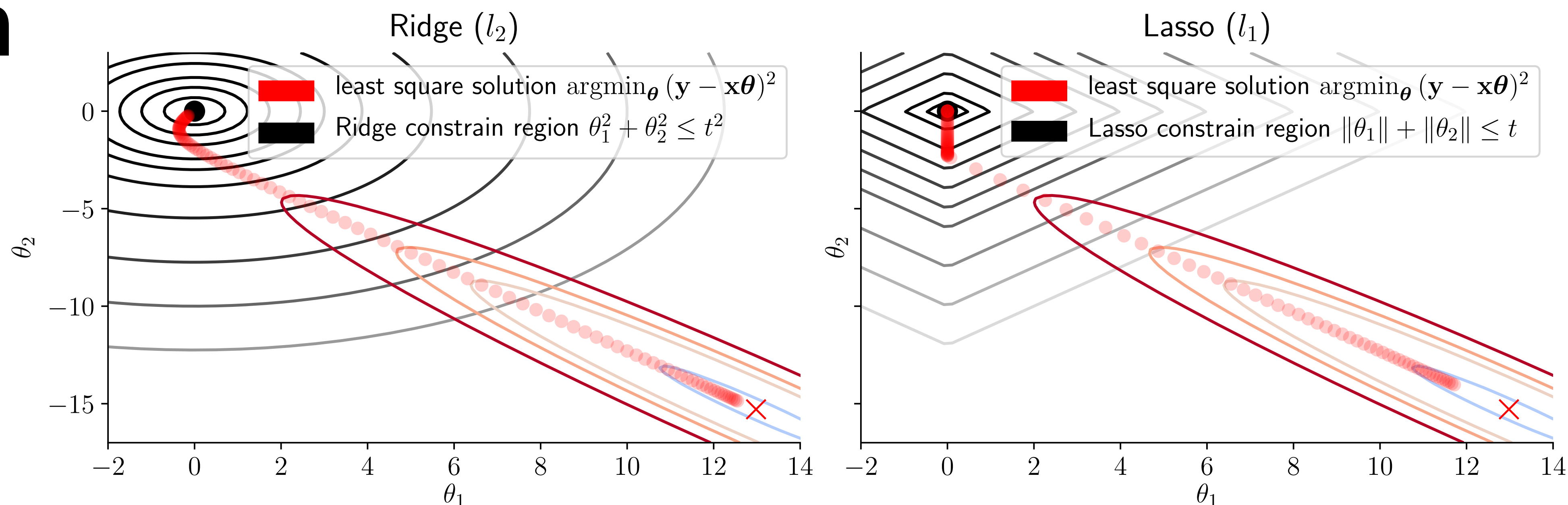
The basic problem: Best subset selection

$$\min_{\beta \in \mathbb{R}^p} \|y - X\beta\|^2 \text{ subject to } \|\beta\|_0 \leq k \quad \|\beta\|_0 = \sum_{j=1}^p 1\{\beta_j \neq 0\}$$

But this is our hard problem (NP hard) ...

... hence we relax the constraint to have a problem that is convex

... the Lasso gives use sparsity as the most feasible approximation to l_0



Feature Selection: LASSO in practice

primary features

$$\text{primary features} = \{r_A, r_B, n_a, n_b, \dots\} \quad \hat{R} = \{+, -, \cdot, \exp, \lg, ^{-1}, ^2, ^3, \sqrt{\cdot}, \|\cdot\|\}$$

Ghiringhelli, *et al. Phys. Rev. Lett.* **2015**, 114 (10), 105503.

Ouyang, R.; *et al. Phys. Rev. Materials* **2018**, 2 (8), 083802.

Bartel, C.; *et al.*, *M. Sci. Adv.* **2019**, 5 (2), eaav0693.

Feature Selection: LASSO in practice

primary features $\longrightarrow$ **billions** of feature candidates

primary features = $\{r_A, r_B, n_a, n_b, \dots\}$ $\hat{R} = \{+, -, \cdot, \exp, \lg, ^{-1}, ^2, ^3, \sqrt{\cdot}, \|\cdot\|\}$

Feature Selection: LASSO in practice

primary features $\longrightarrow$ **billions** of feature candidates $\xrightarrow{\text{(SI) +Lasso}}$ subset of features

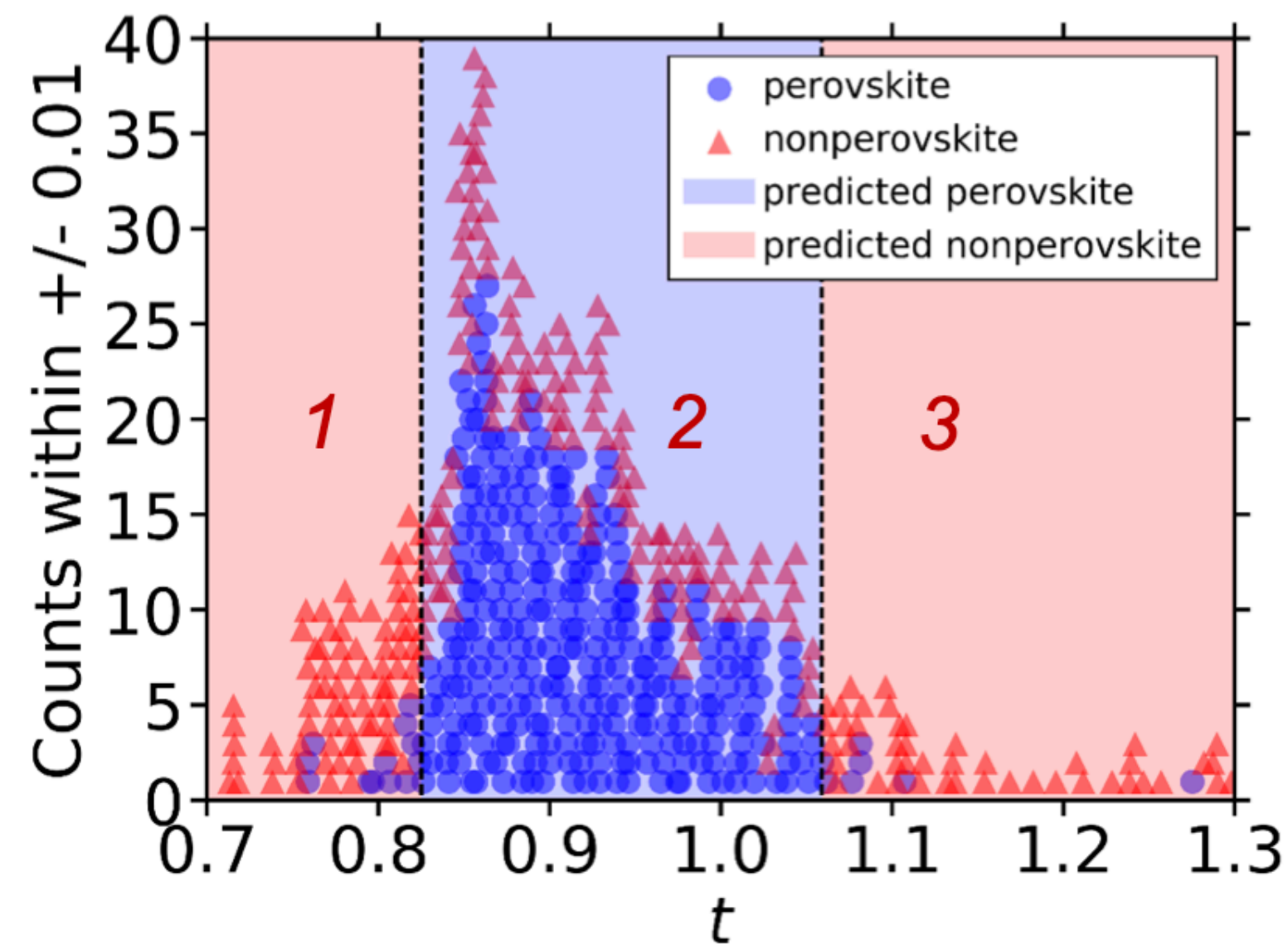
primary features = $\{r_A, r_B, n_a, n_b, \dots\}$ $\hat{R} = \{+, -, \cdot, \exp, \lg, ^{-1}, ^2, ^3, \sqrt{\cdot}, \|\cdot\|\}$

Feature Selection: LASSO in practice

primary features $\longrightarrow$ **billions** of feature candidates $\xrightarrow{\text{(SI) + Lasso}}$ subset of features

primary features = $\{r_A, r_B, n_a, n_b, \dots\}$ $\hat{R} = \{+, -, \cdot, \exp, \lg, ^{-1}, ^2, ^3, \sqrt{\cdot}, \|\cdot\|\}$

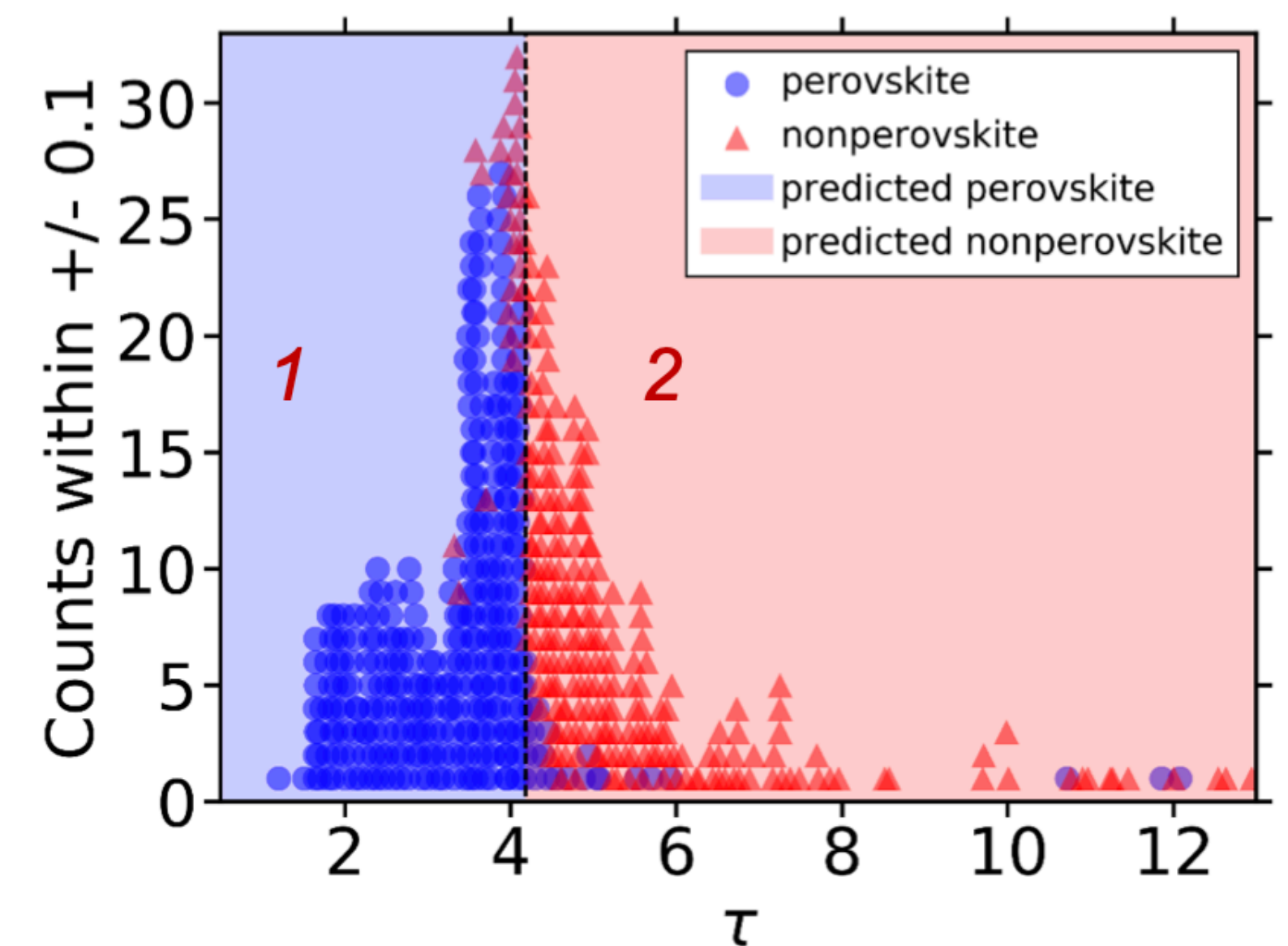
$$t = \frac{r_A + r_X}{\sqrt{2}(r_B + r_X)}$$



$0.825 < t < 1.059 \rightarrow$ perovskite

74% accuracy

$$\tau = \frac{r_X}{r_B} - n_A \left(n_A - \frac{r_A/r_B}{\ln r_A/r_B} \right)$$



$\tau < 4.18 \rightarrow$ perovskite

92% accuracy

Ghiringhelli, *et al. Phys. Rev. Lett.* **2015**, 114 (10), 105503.

Ouyang, R.; *et al. Phys. Rev. Materials* **2018**, 2 (8), 083802.

Bartel, C.; *et al., M. Sci. Adv.* **2019**, 5 (2), eaav0693.

8. Feature Selection

Learning Objectives:

- Curse of dimensionality: No locality in high-dimensional spaces
- Filter heuristics
- LASSO

9. Feature Projection

Learning Objectives:

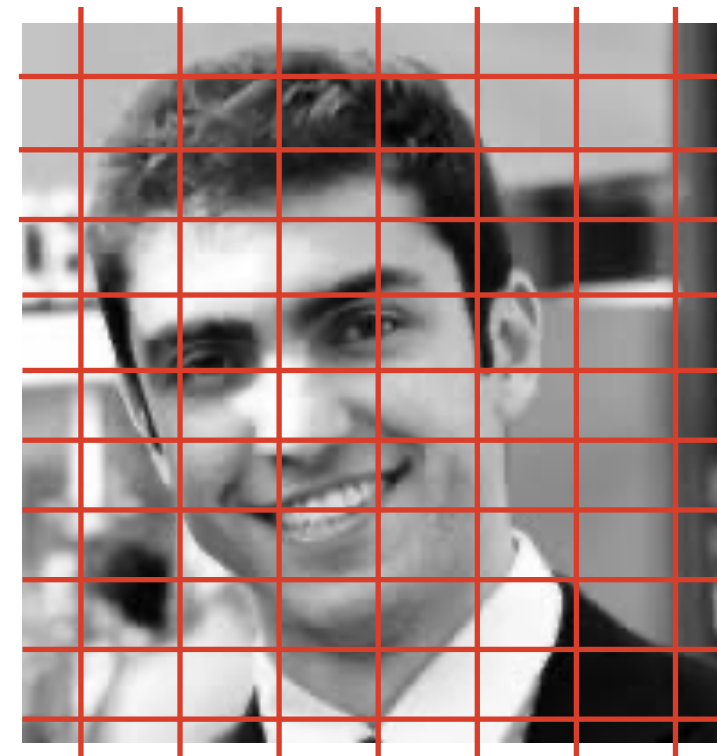
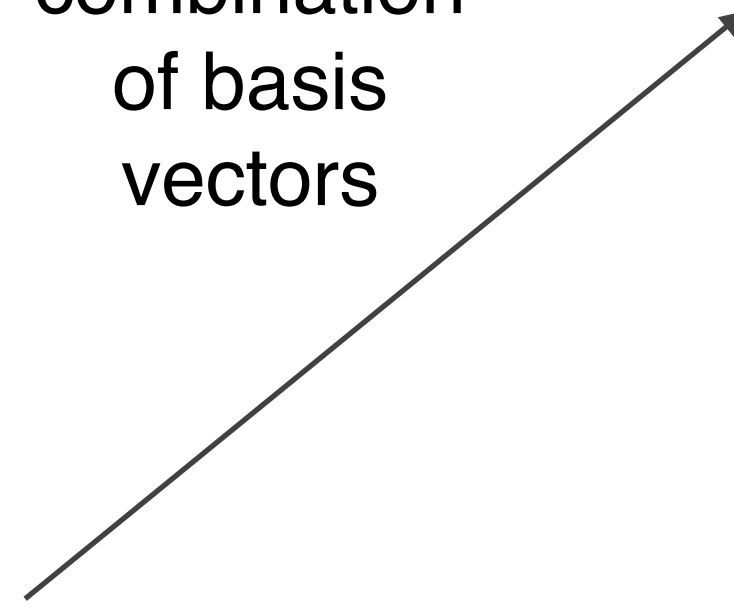
- Principal component analysis, t -SNE: Principles and pitfalls
- Projecting high dimensional data into a lower dimensional space

Feature Projection: Projecting High-Dimensional Data

Eigenfaces: Principal Component Analysis on face images to get “basis vectors” of face image space



Linear
combination
of basis
vectors

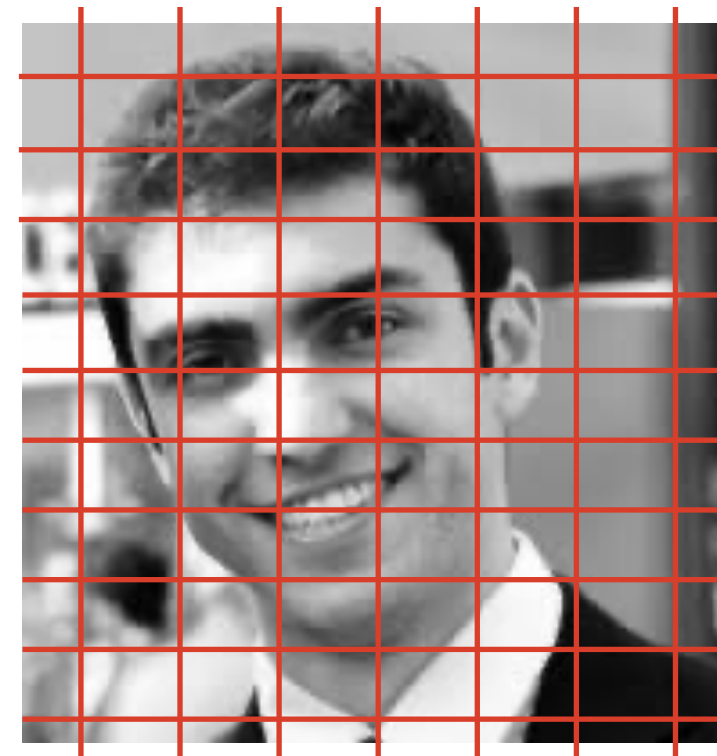
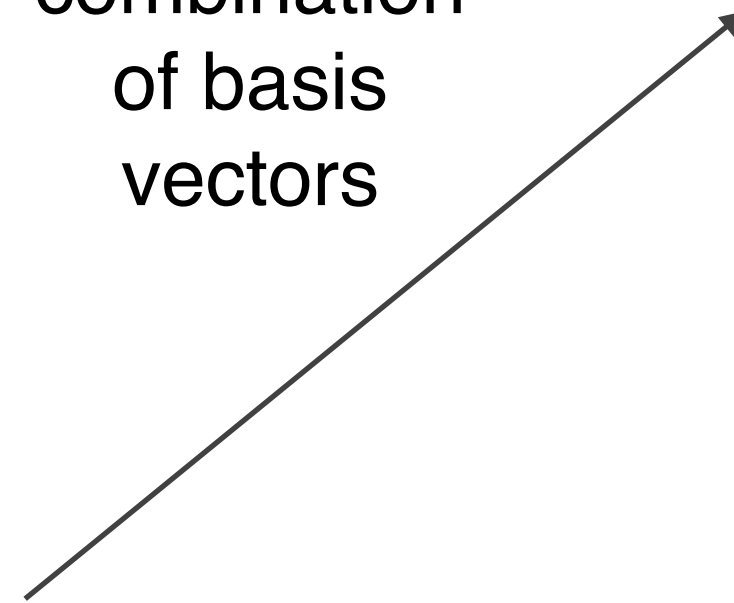


Feature Projection: Projecting High-Dimensional Data

Eigenfaces: Principal Component Analysis on face images to get “basis vectors” of face image space



Linear
combination
of basis
vectors

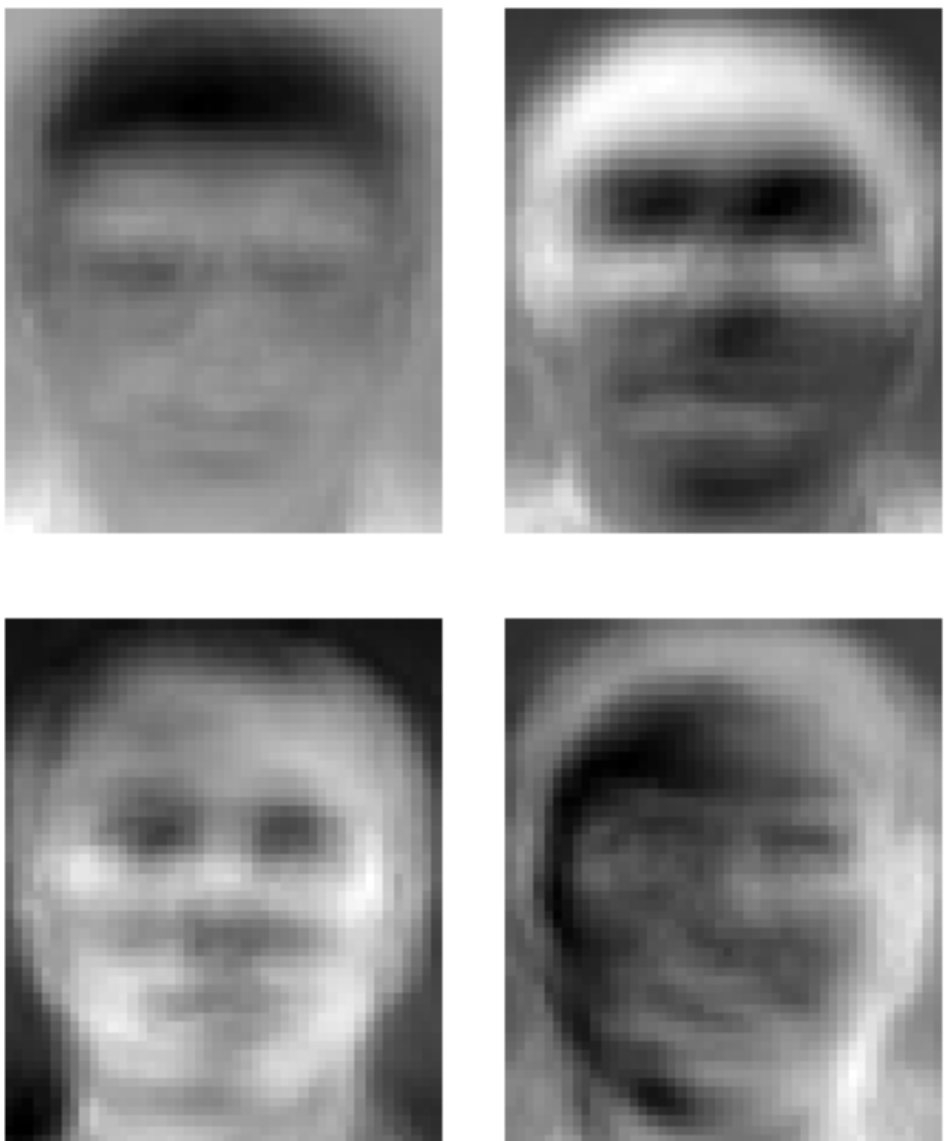


One image
is one column

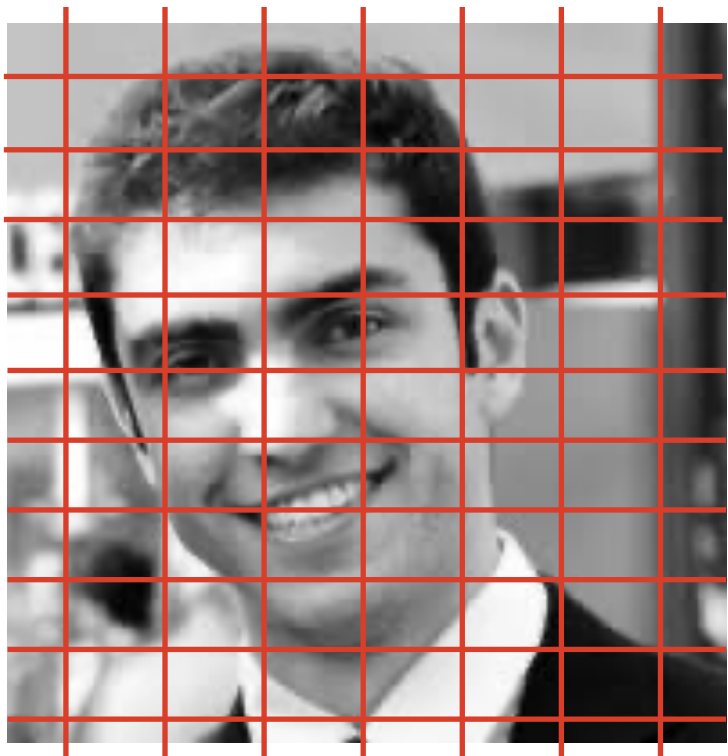
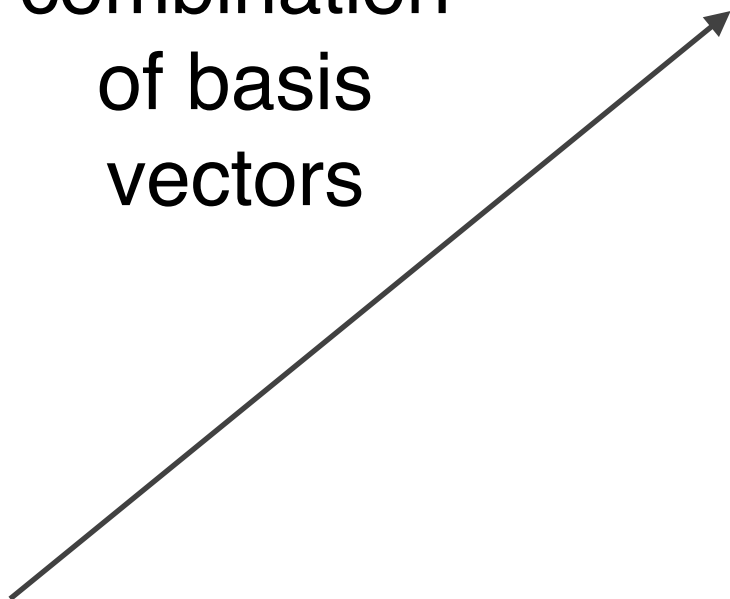


Feature Projection: Projecting High-Dimensional Data

Eigenfaces: Principal Component Analysis on face images to get “basis vectors” of face image space



Linear combination of basis vectors



One image is one column



faces

pixels

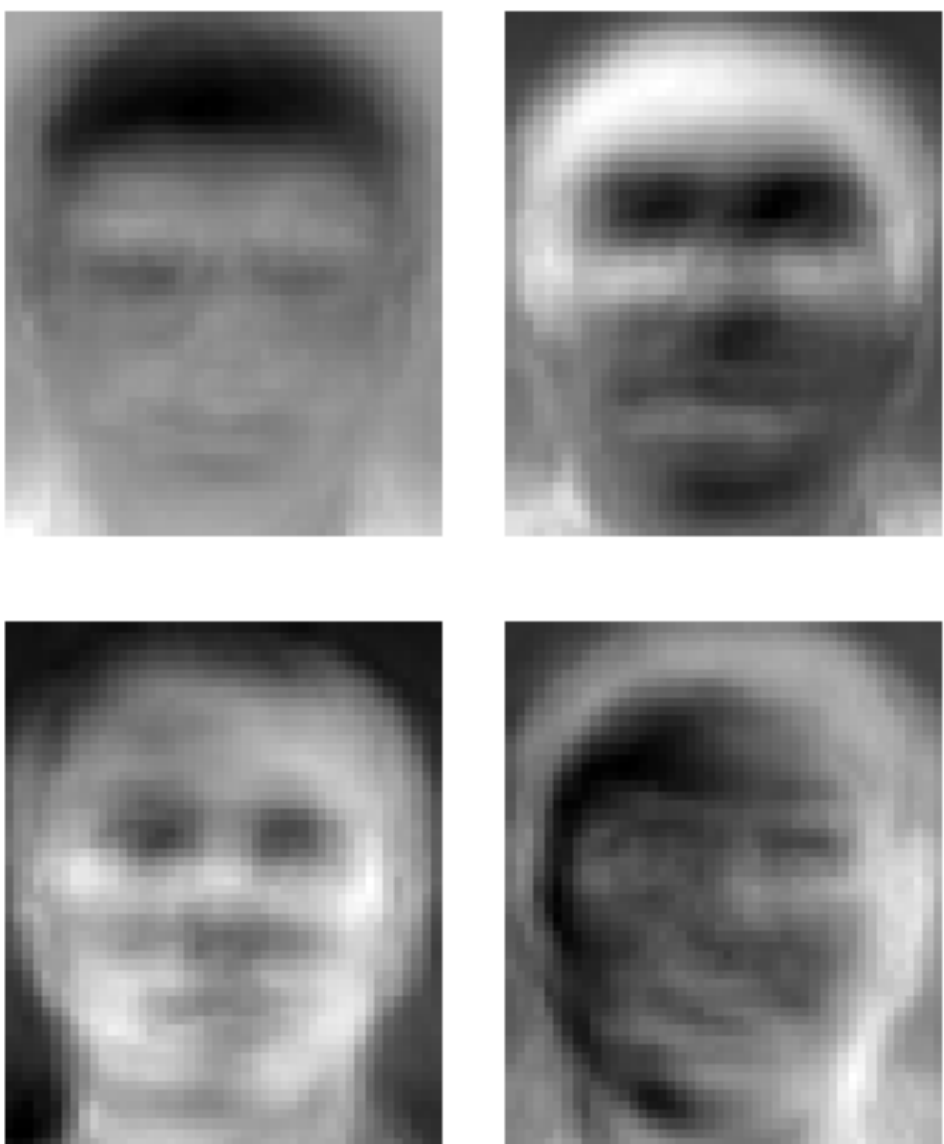


many faces

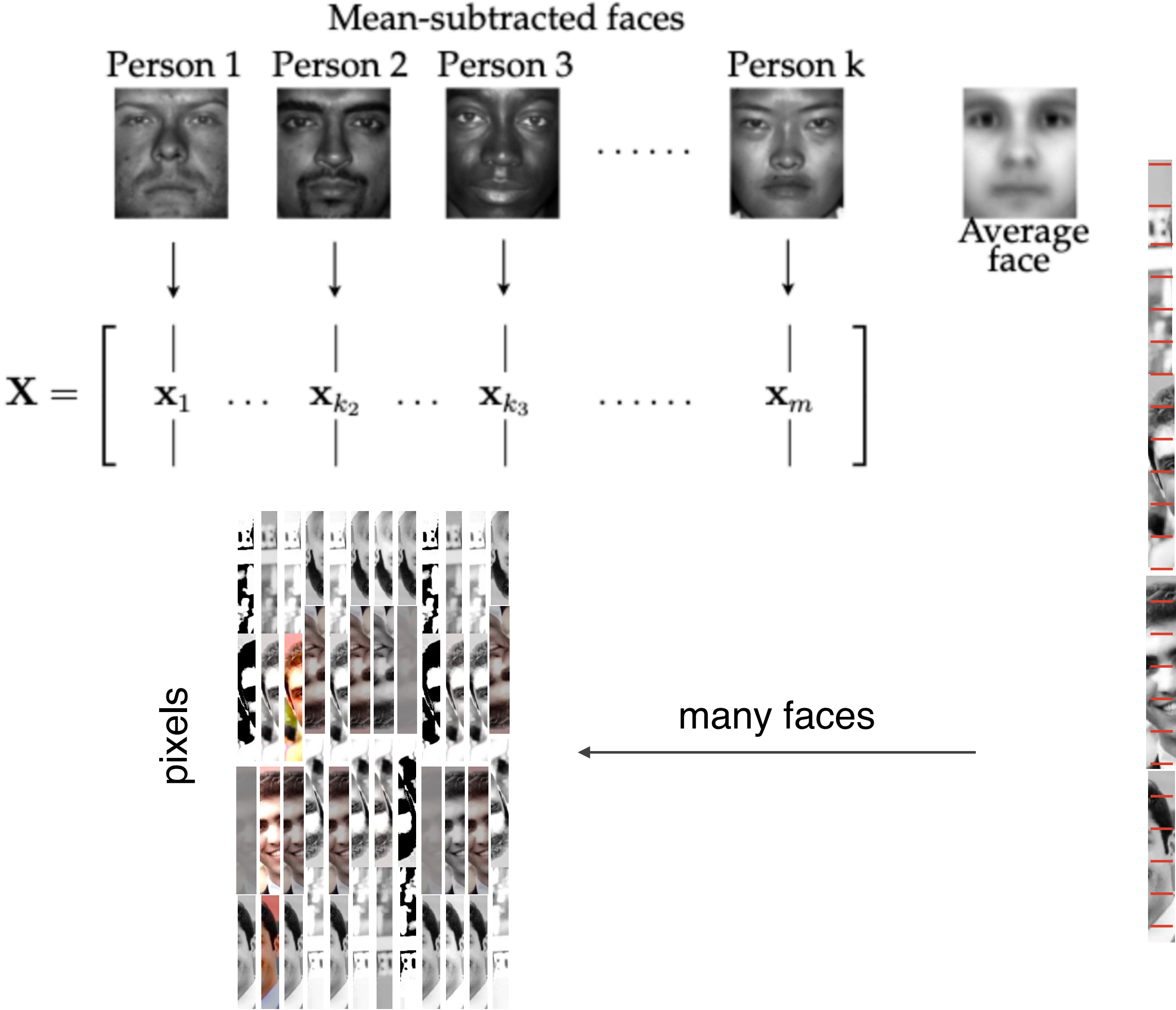
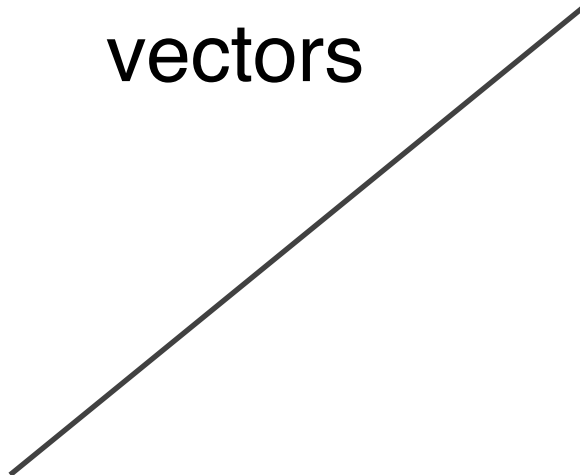


Feature Projection: Projecting High-Dimensional Data

Eigenfaces: Principal Component Analysis on face images to get “basis vectors” of face image space

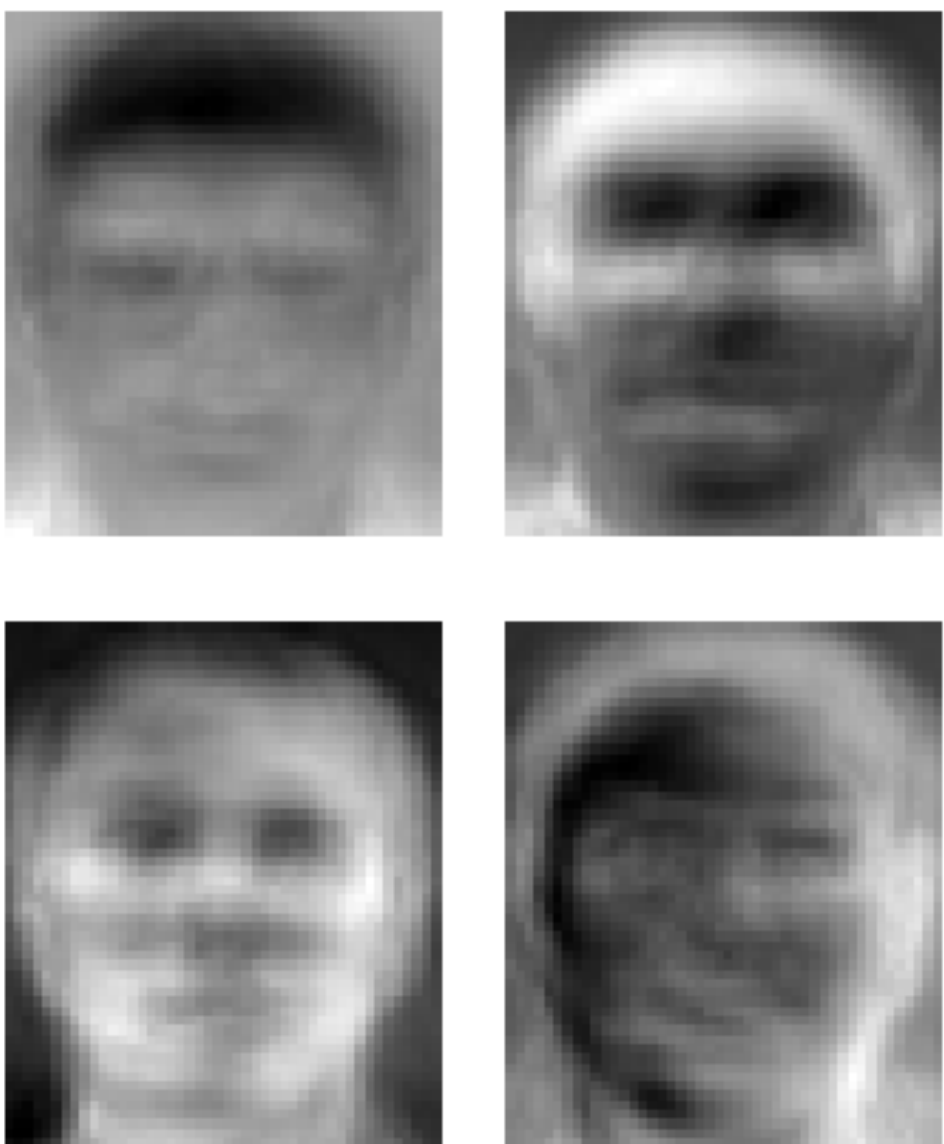


Linear combination of basis vectors



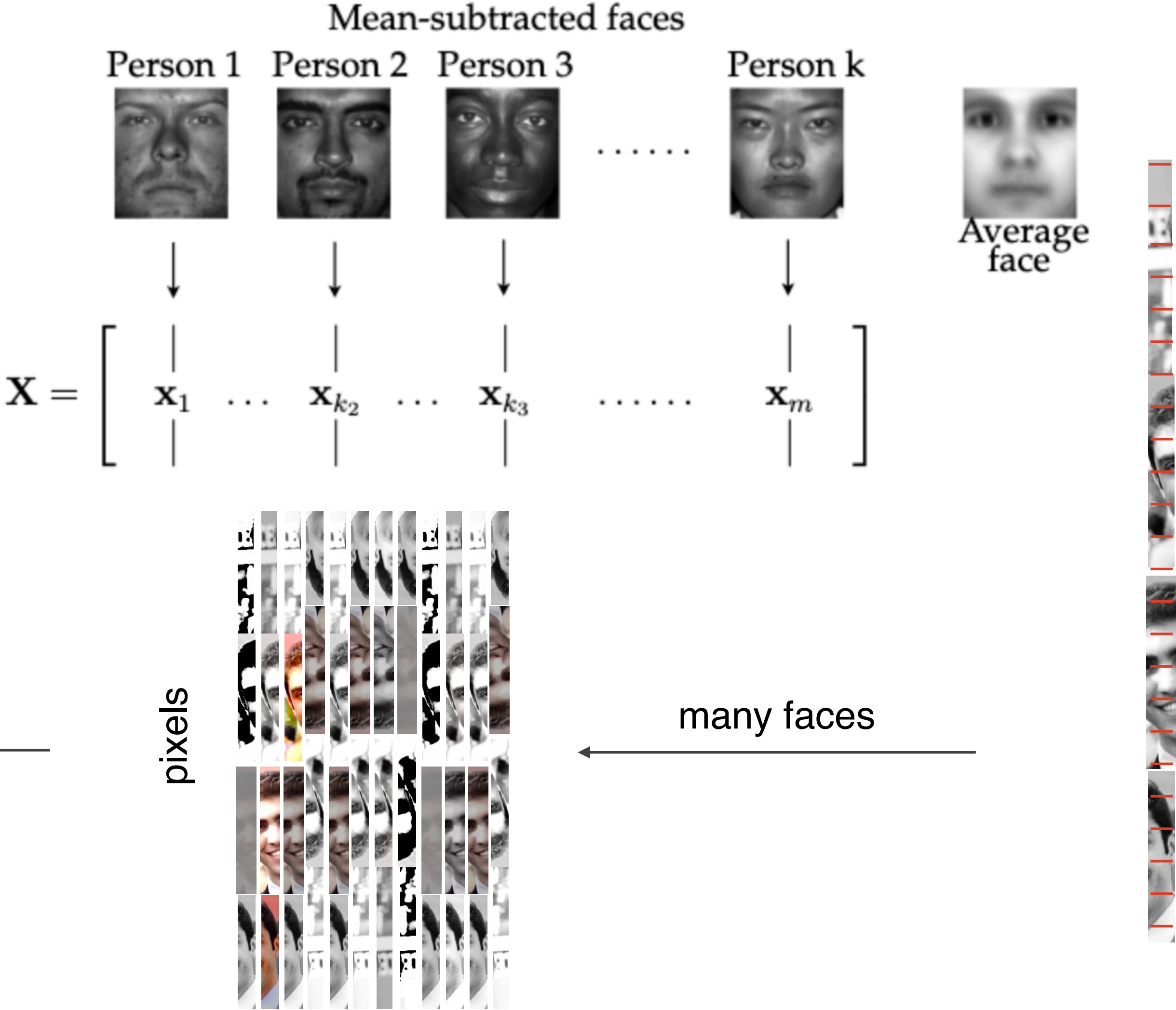
Feature Projection: Projecting High-Dimensional Data

Eigenfaces: Principal Component Analysis on face images to get “basis vectors” of face image space



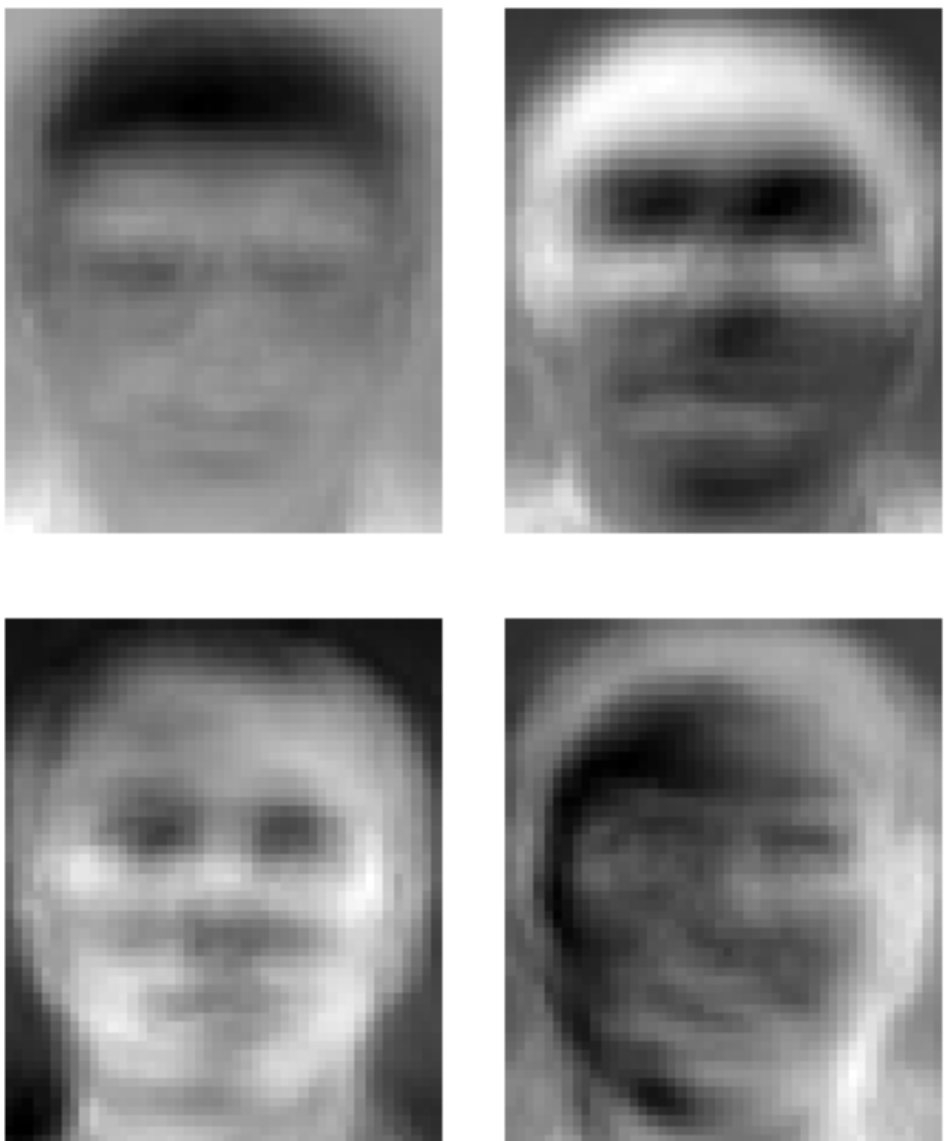
Linear combination of basis vectors

PCA
Find basis vectors

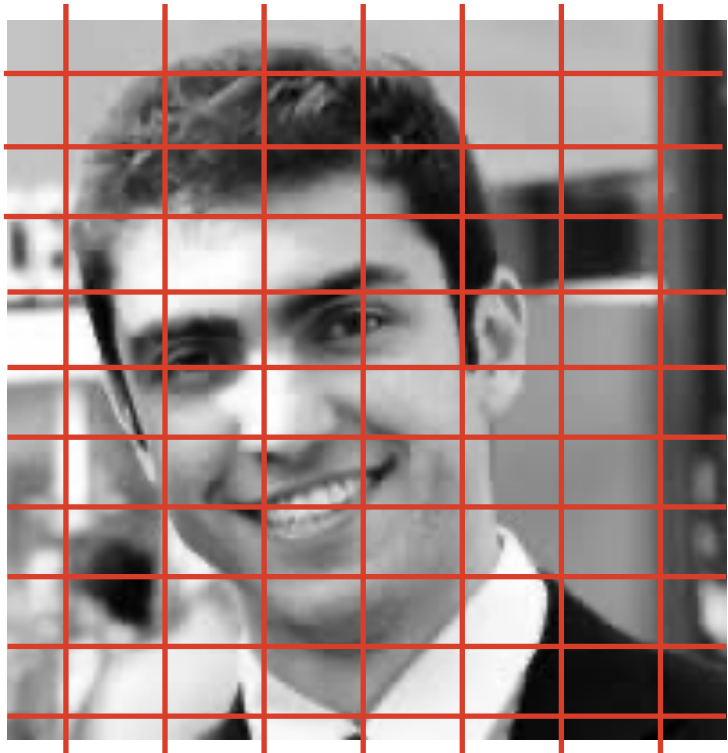


Feature Projection: Projecting High-Dimensional Data

Eigenfaces: Principal Component Analysis on face images to get “basis vectors” of face image space



Linear combination of basis vectors



faces



pixels

PCA

Find basis vectors

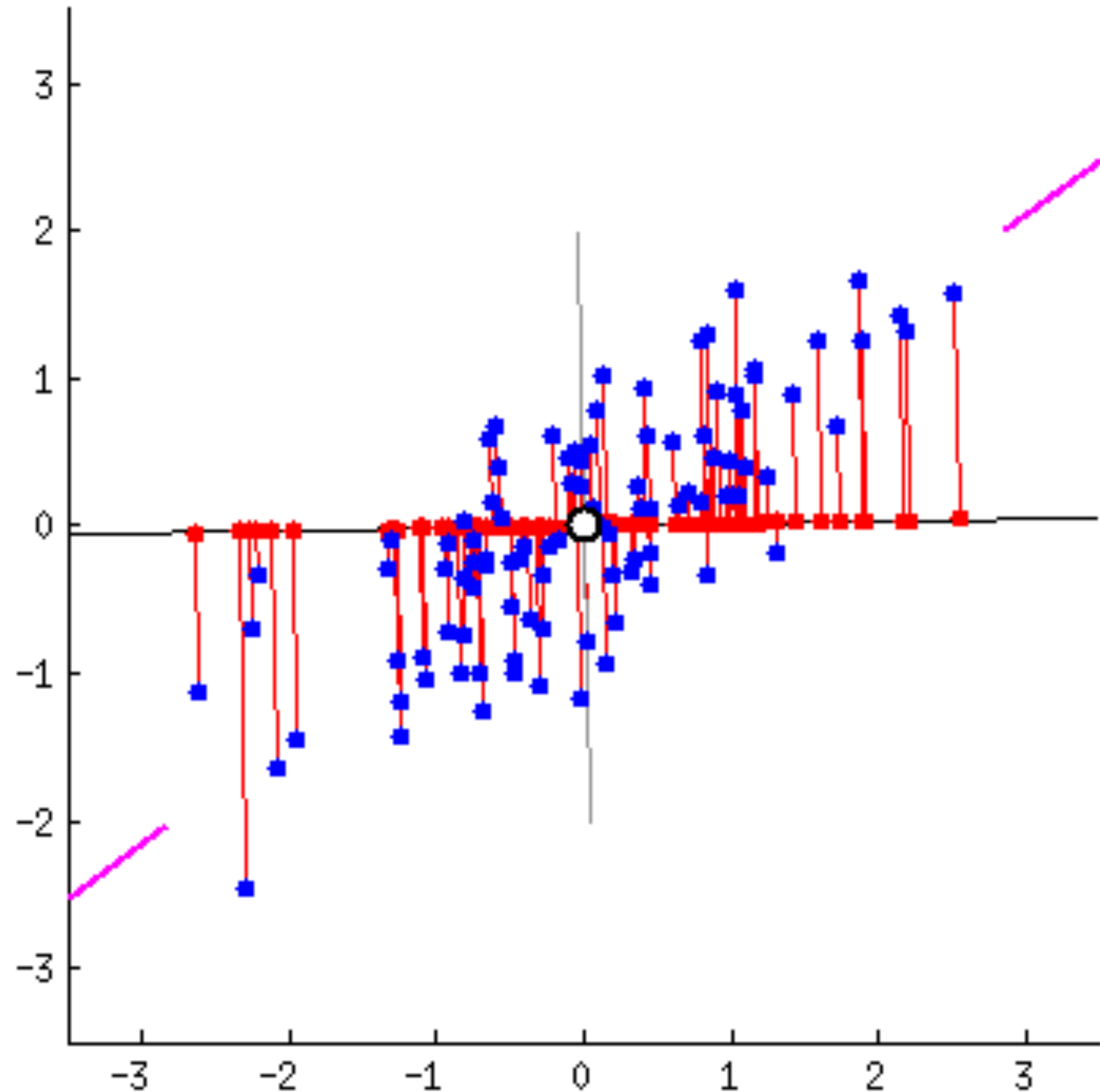
One image is one column

many faces



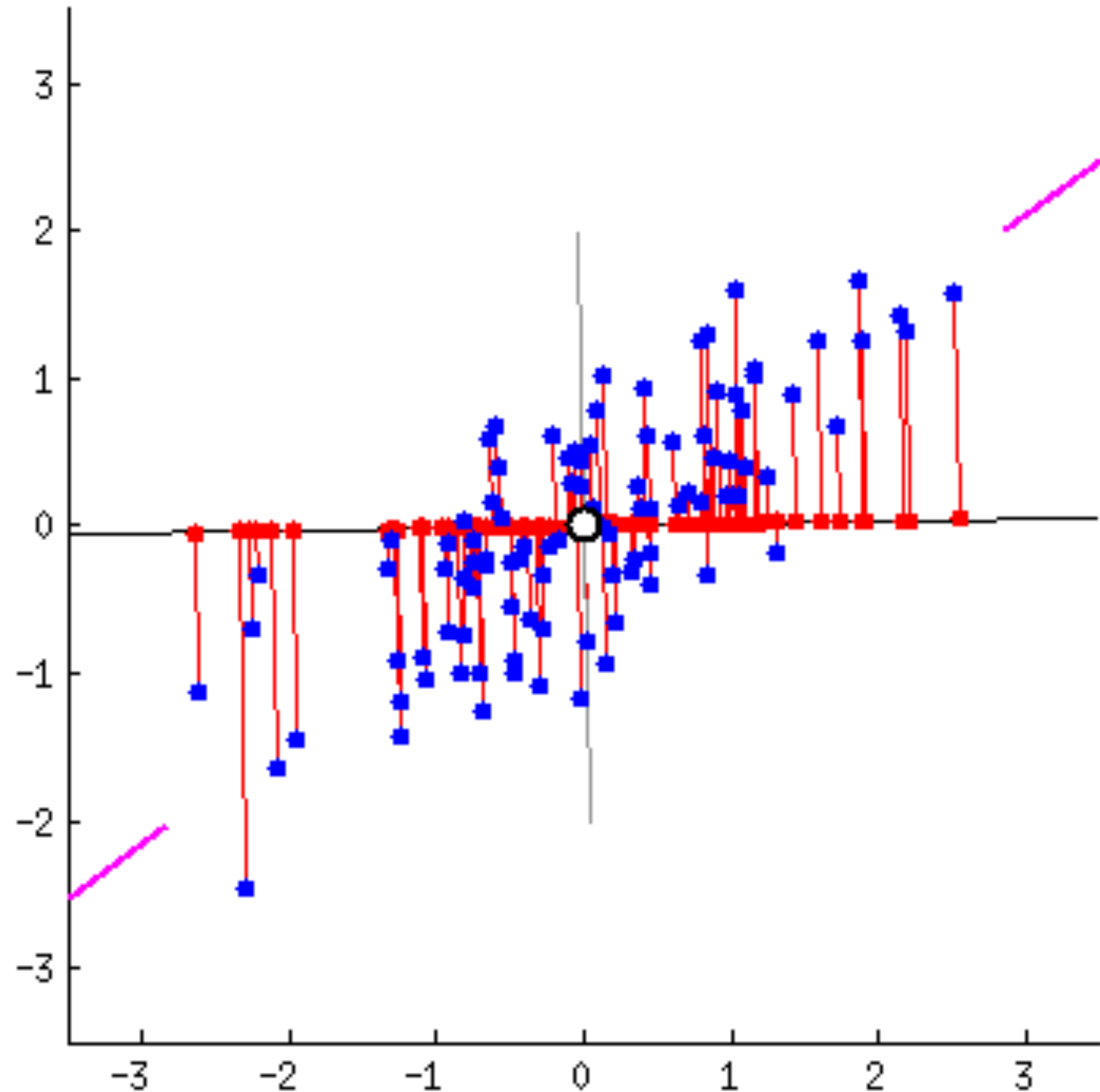
PCA Intuition

Finding the line that captures most of the variance, i.e., maximum decorrelation



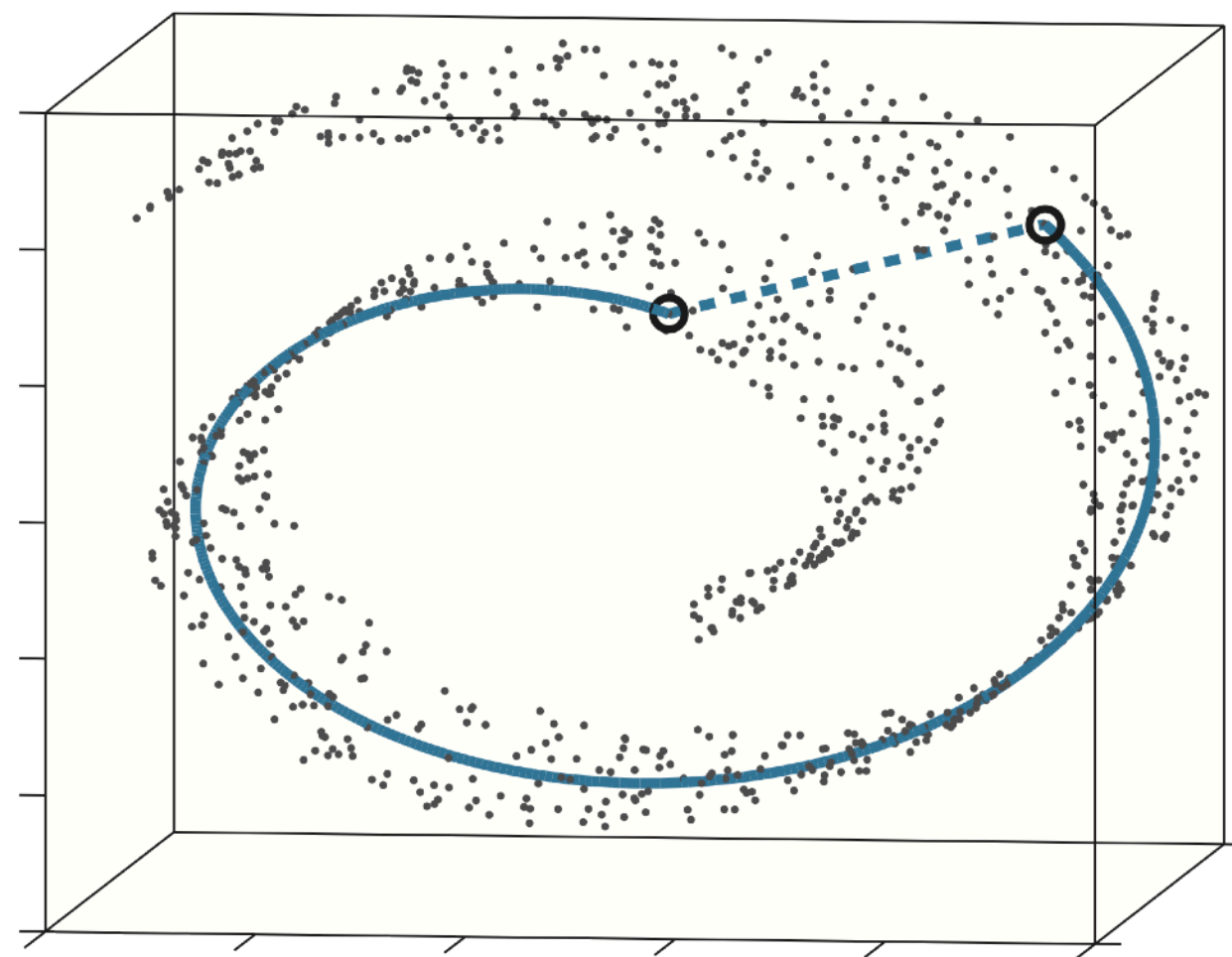
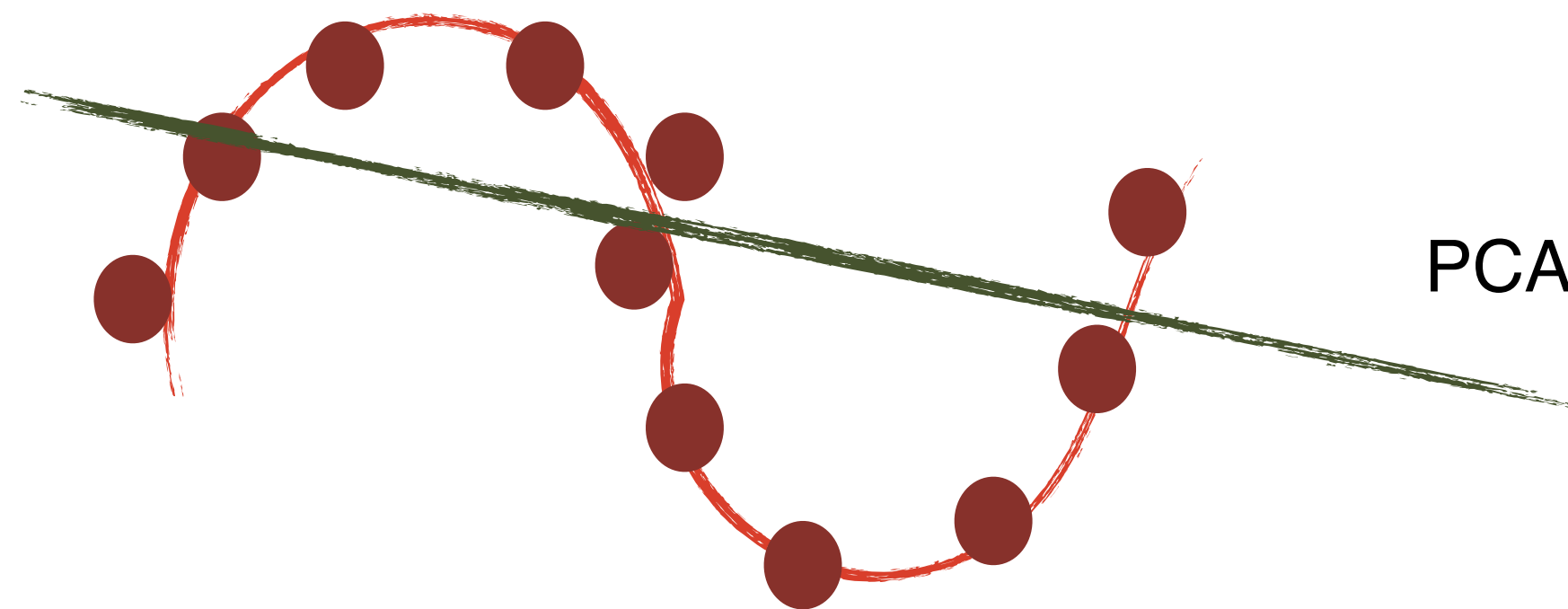
PCA Intuition

Finding the line that captures most of the variance, i.e., maximum decorrelation



Caveats with PCA

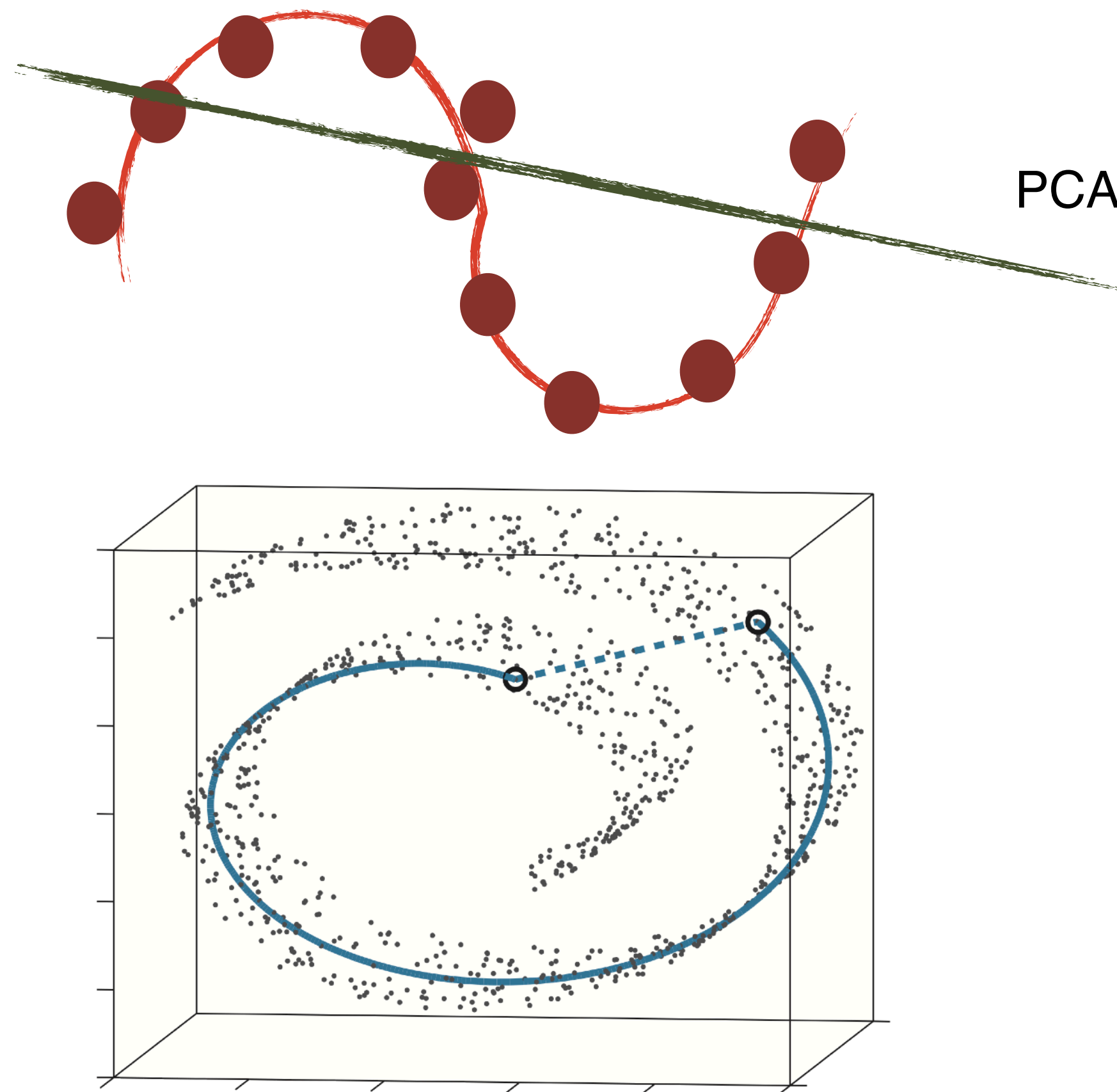
Non-linearity



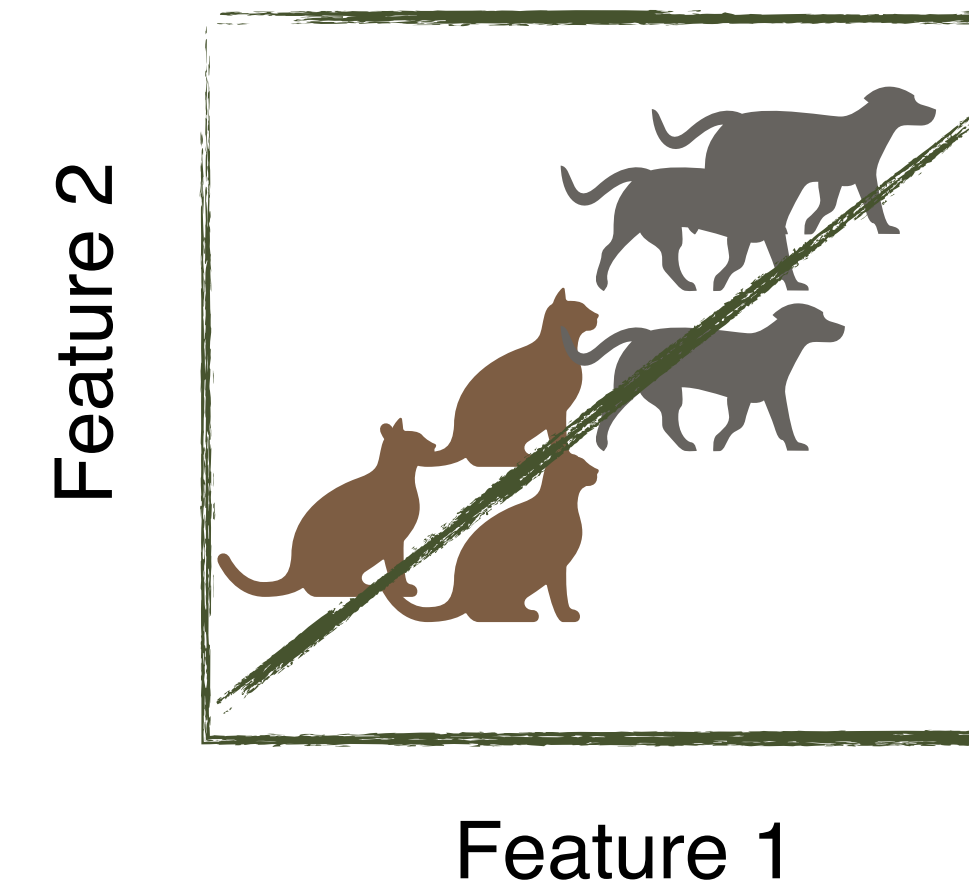
- Data is linearly uncorrelated
- But there is still a non-linear dependence

Caveats with PCA

Non-linearity



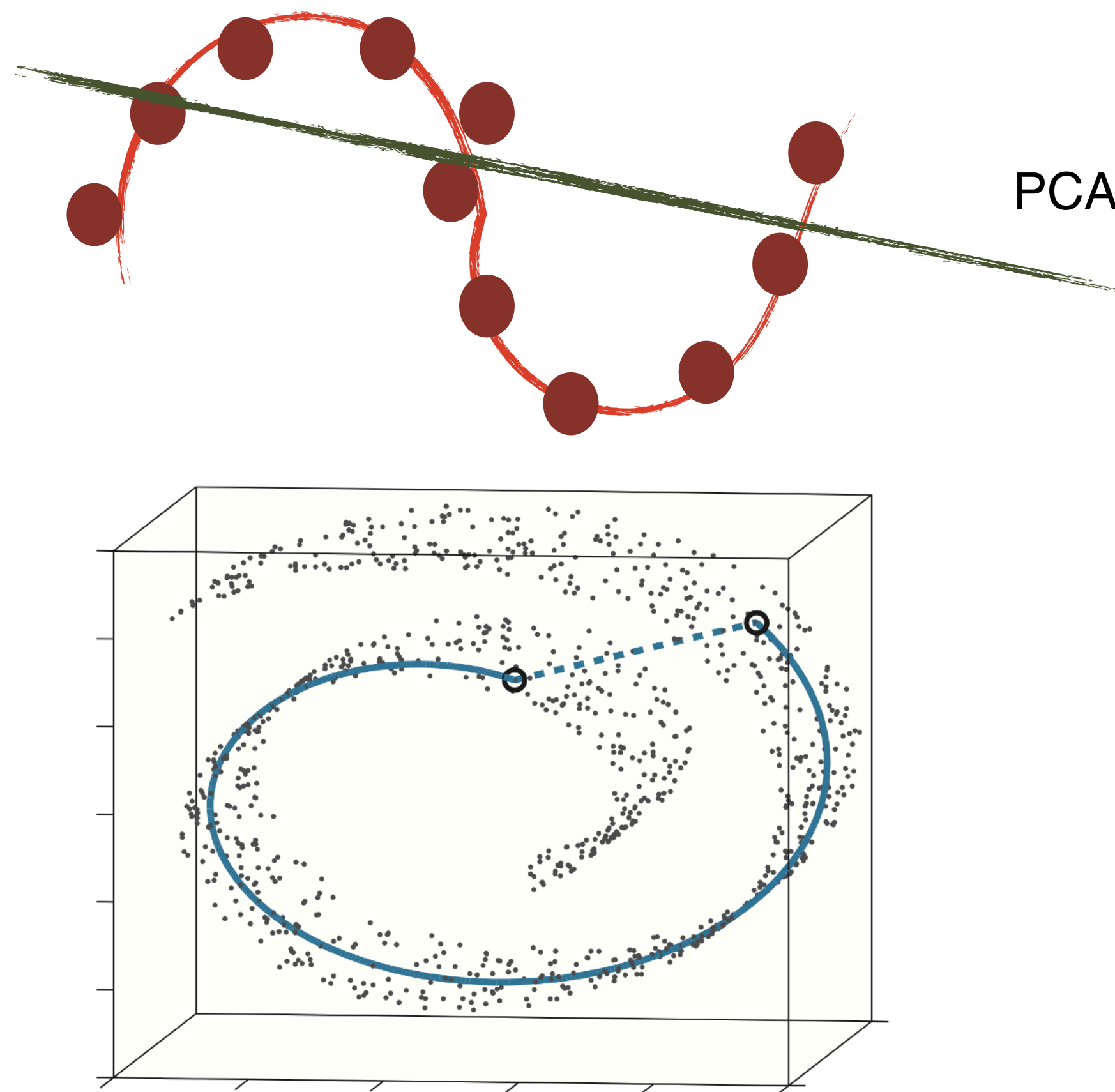
Higher variance feature is not more discriminative



- Data is linearly uncorrelated
- But there is still a non-linear dependence

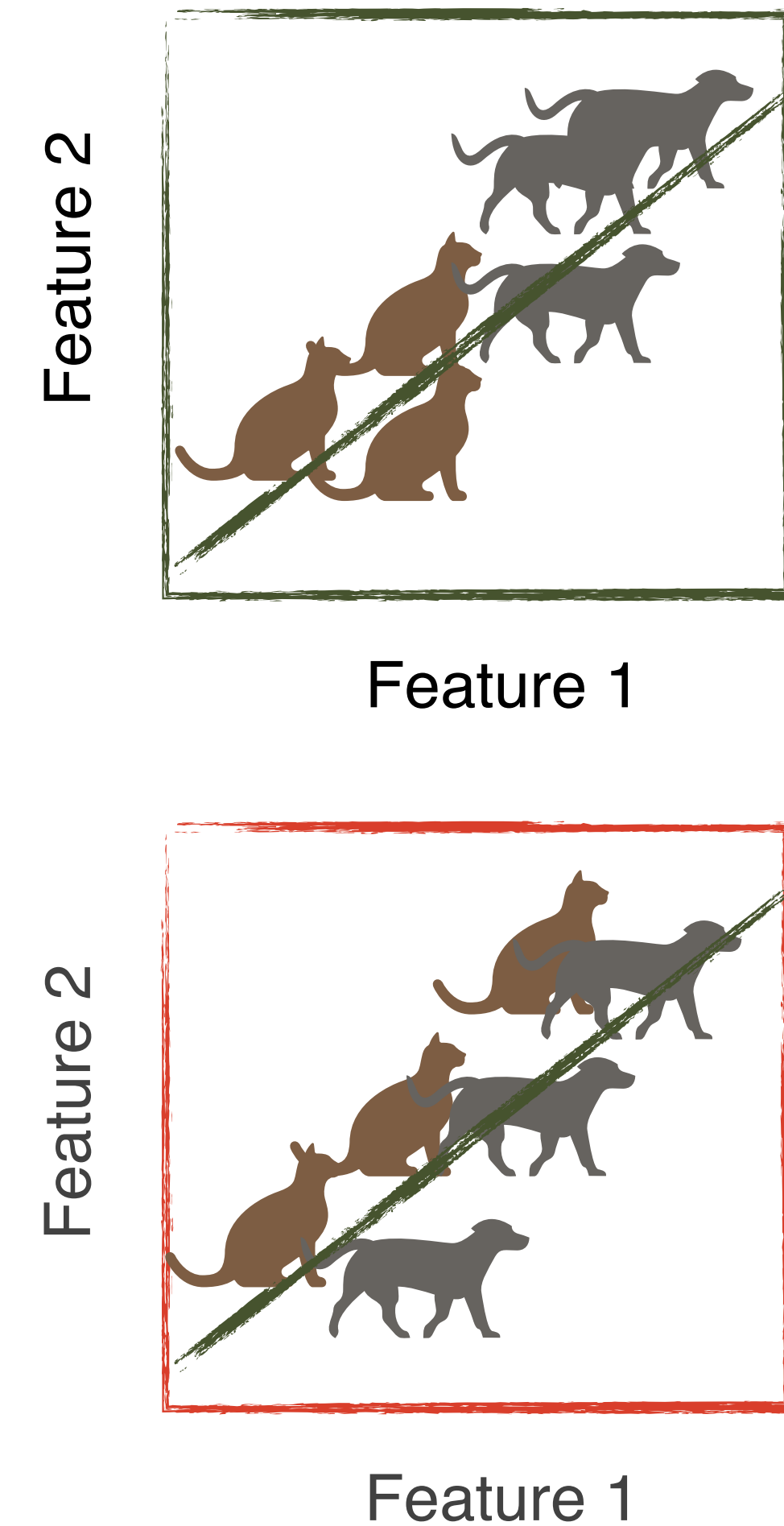
Caveats with PCA

Non-linearity



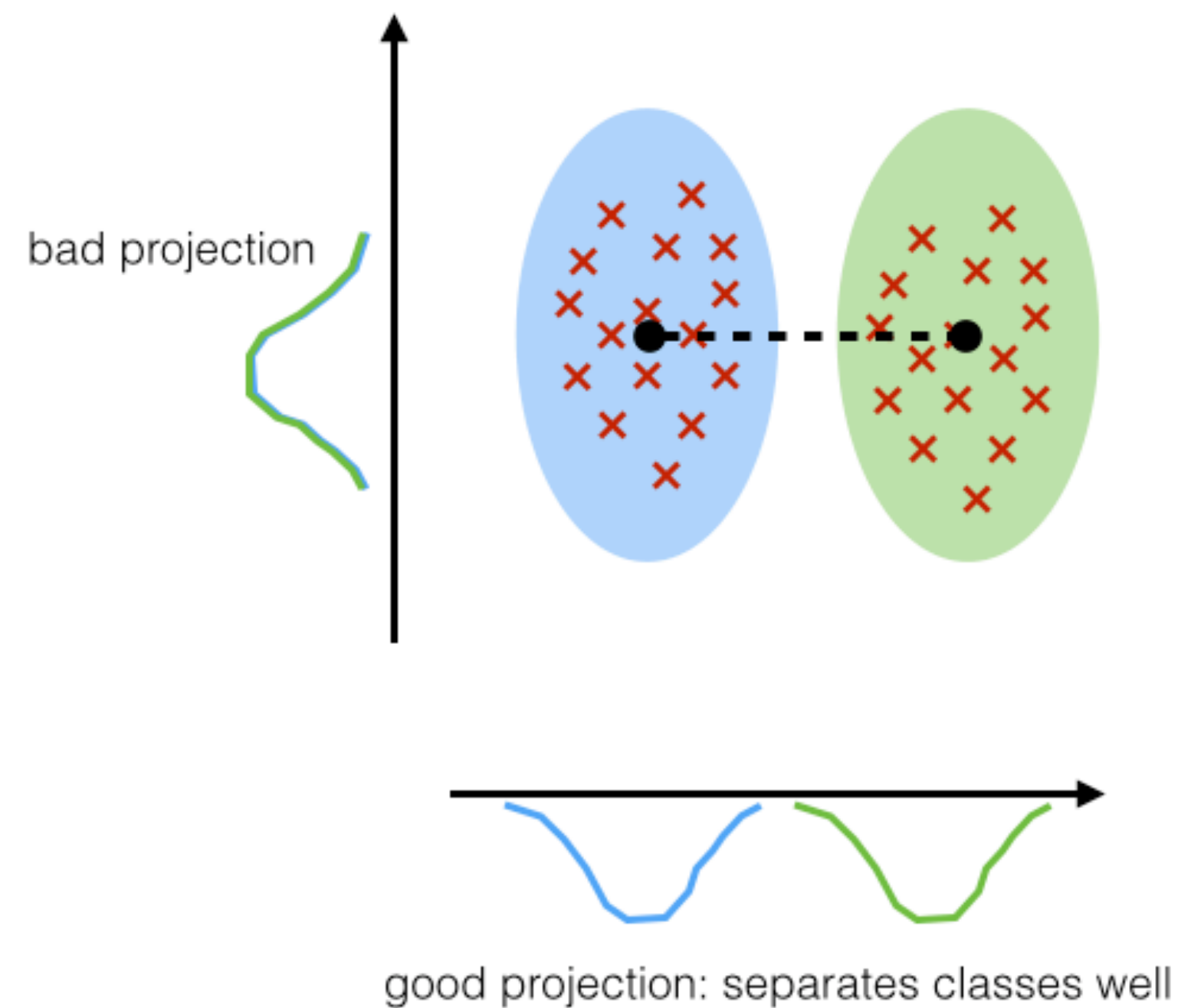
- Data is linearly uncorrelated
- But there is still a non-linear dependence

Higher variance feature is not more discriminative



Other member of dimensionality reduction zoo

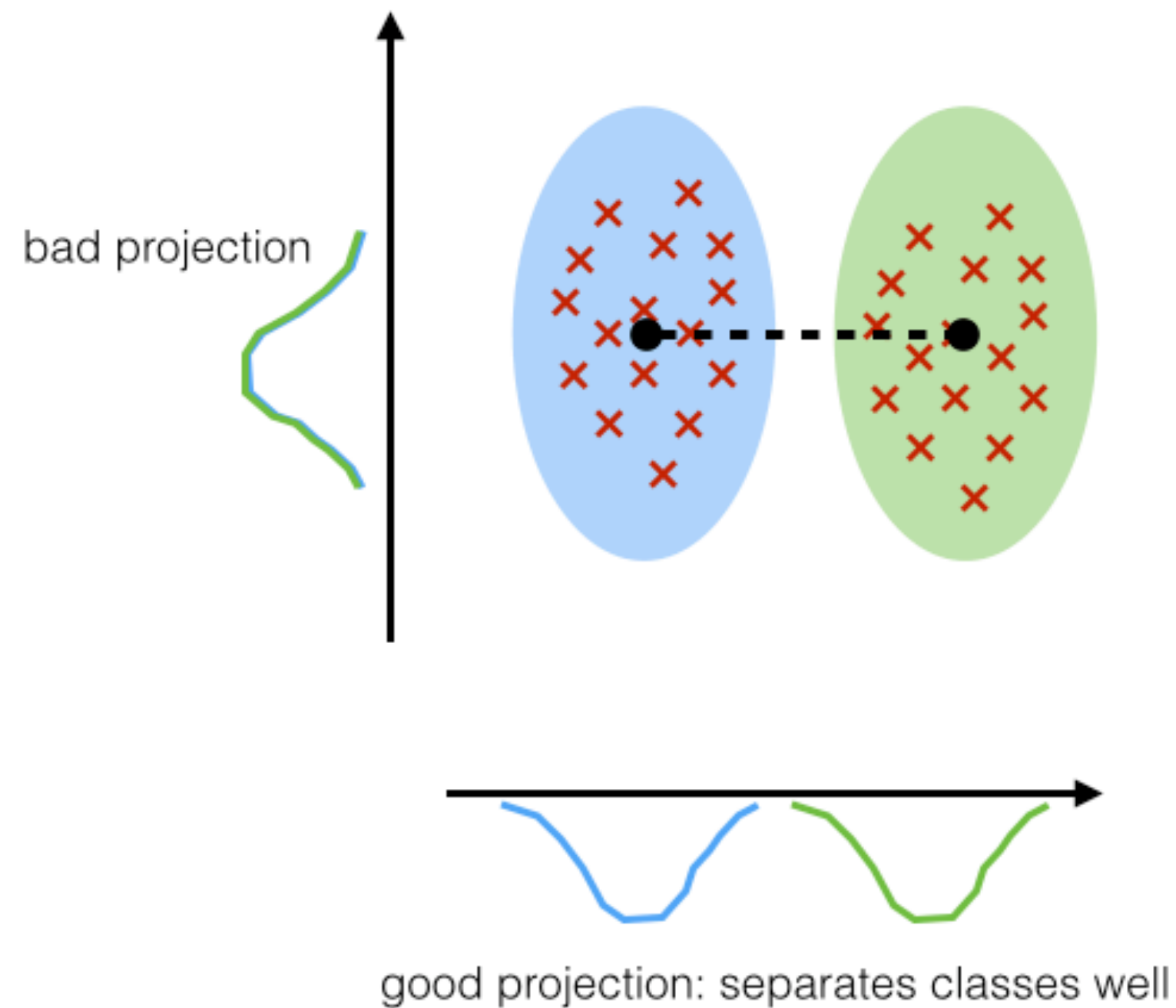
Linear Discriminant Analysis (LDA)



- math like for PCA
- maximizing component axes for class

Other member of dimensionality reduction zoo

Linear Discriminant Analysis (LDA)

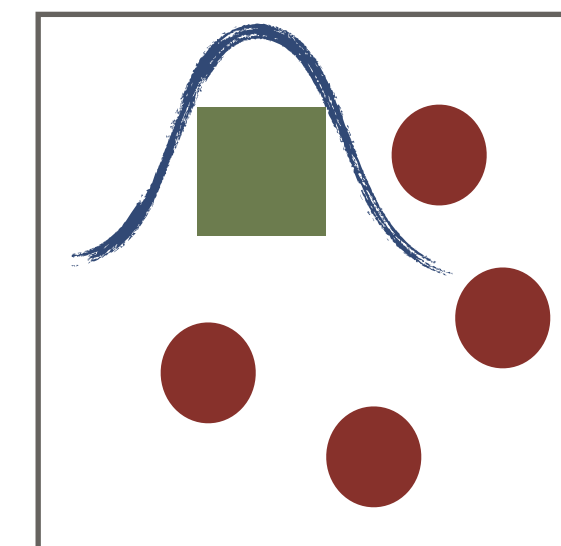


- math like for PCA
- maximizing component axes for class

t-distributed stochastic neighbor embedding (*t*-SNE)

- Non-linear
- Conditional probabilities that represent similarities
- Sensitive to perplexity (number of neighbors)

High Dimensional

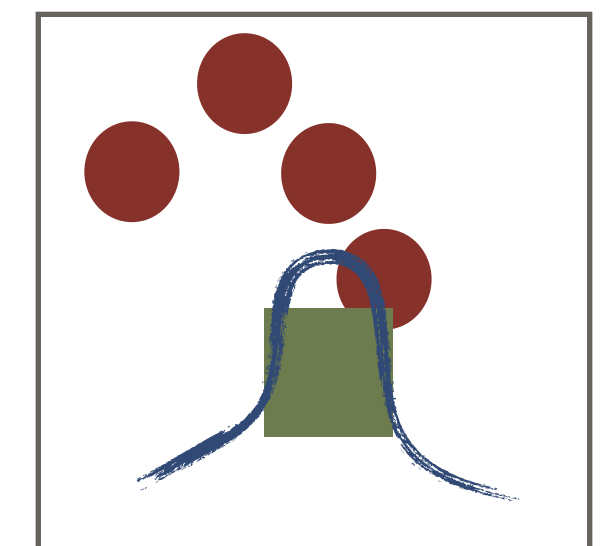


Gaussian
similarity

minimize
mismatch
between
conditional
distributions



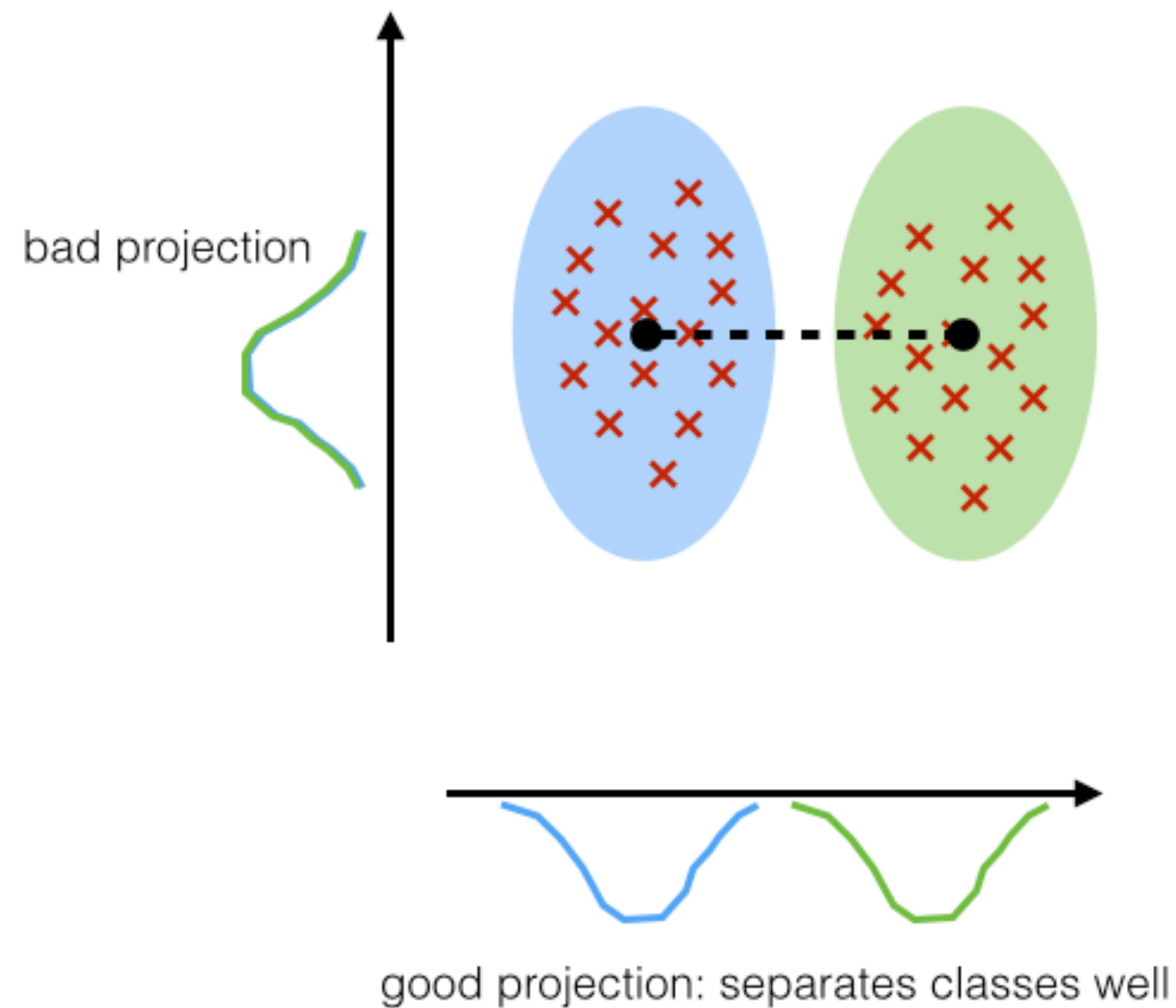
Low Dimensional



student-t
similarity

Other member of dimensionality reduction zoo

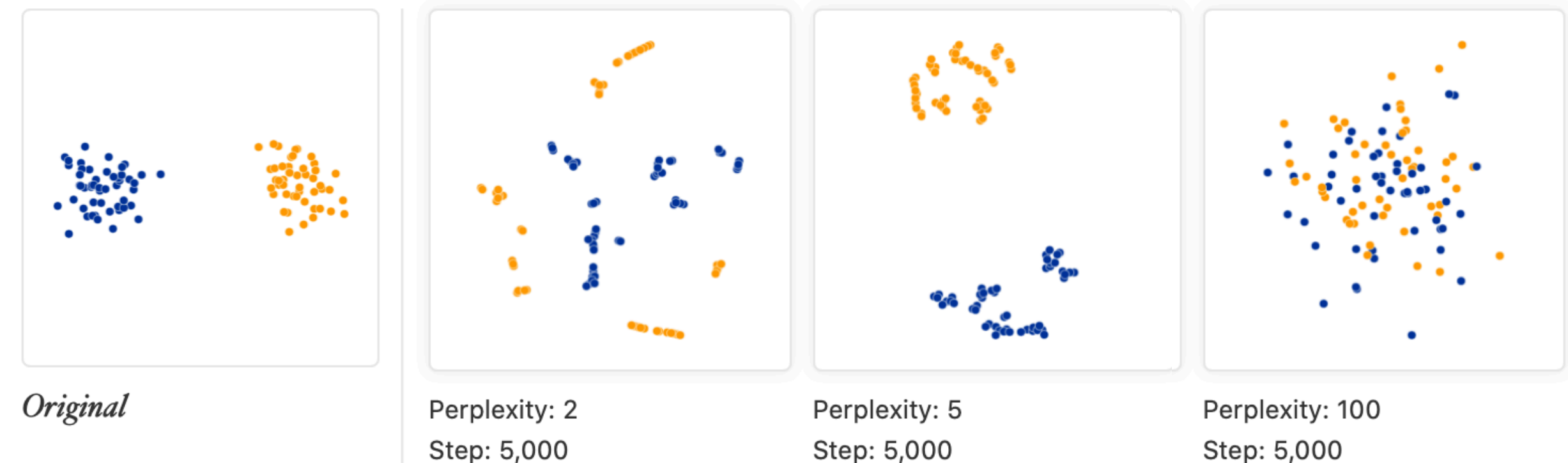
Linear Discriminant Analysis (LDA)



- math like for PCA
- maximizing component axes for class

t-distributed stochastic neighbor embedding (*t*-SNE)

- Non-linear
- Conditional probabilities that represent similarities
- Sensitive to perplexity (number of neighbors)



9. Feature Projection

Learning Objectives:

- Principal component analysis, t-SNE: Principles and pitfalls
- Projecting high dimensional data into a lower dimensional space

10. Feature Learning

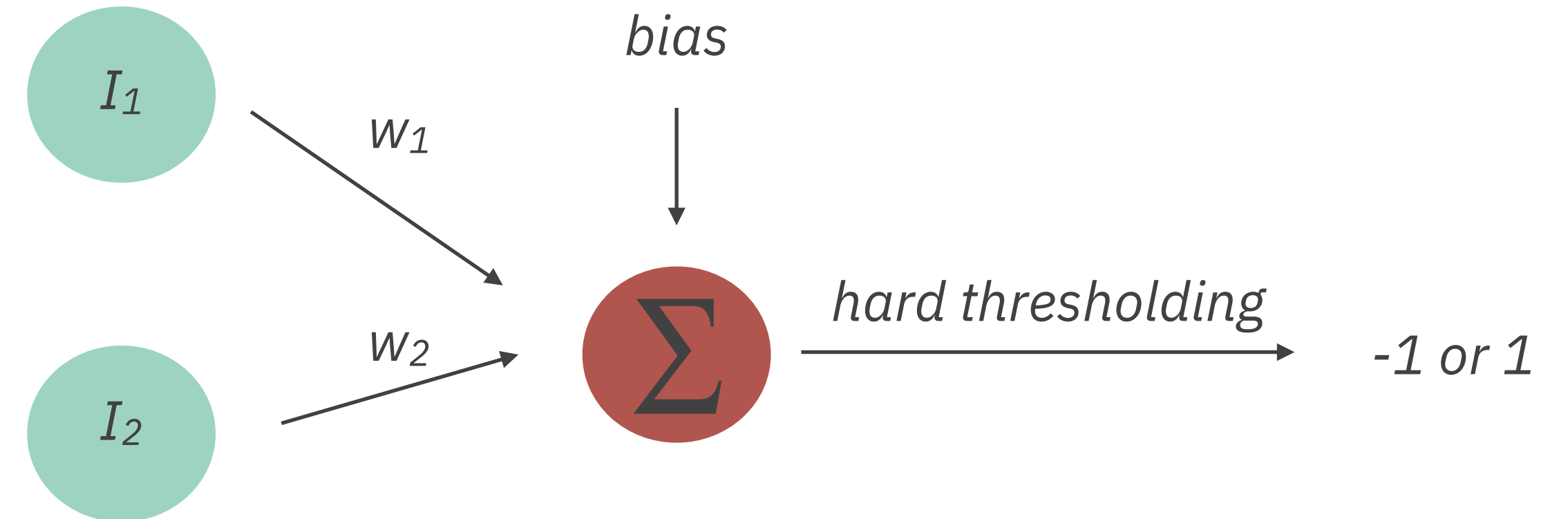
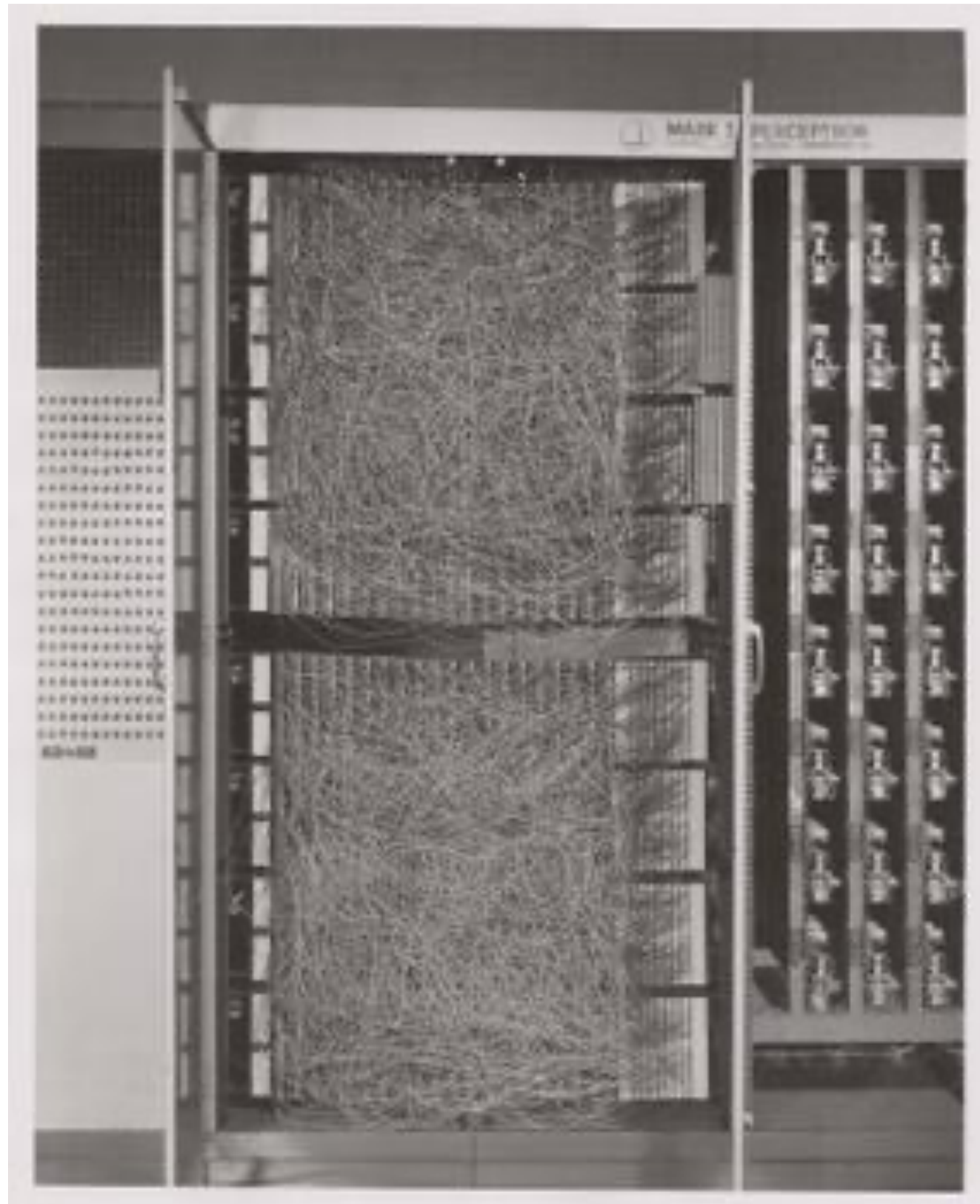
Learning Objectives:

- Multiple layers with non-linearities can act as universal function approximations
- Intermediate layers learn predictive representations
- Symmetry of problem guides architecture choice

Neural Networks: Perceptron (*“may eventually be able to learn, make decisions, and translate languages”*)

The Perceptron (Rosenblatt (1957))

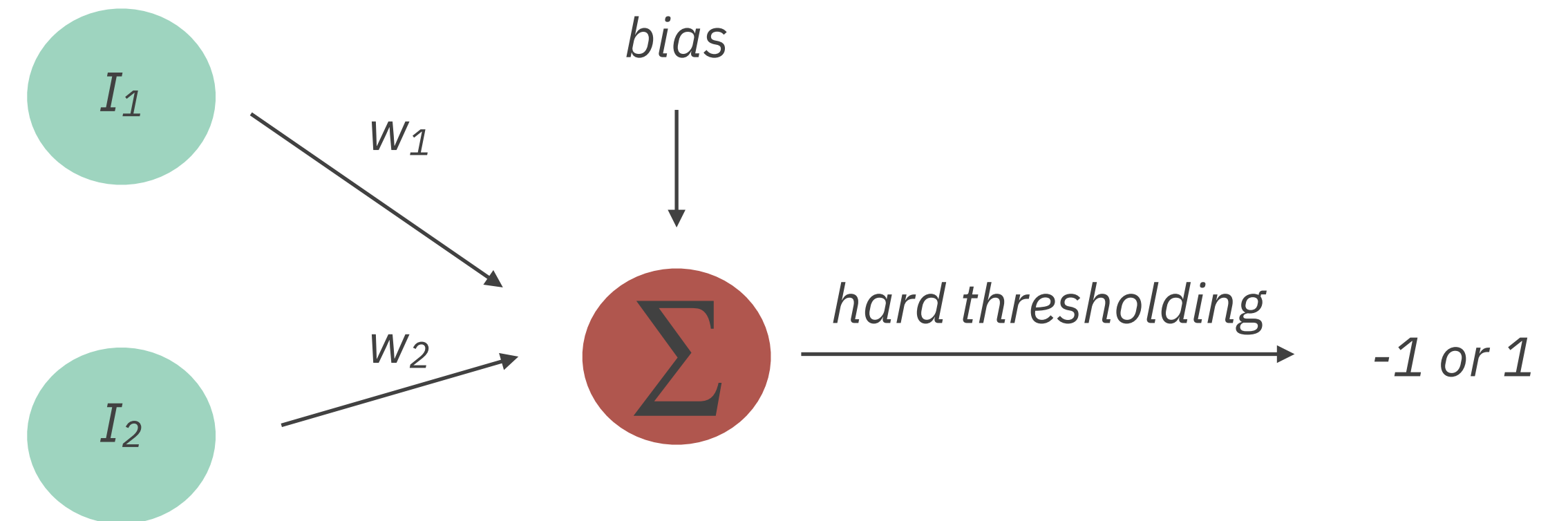
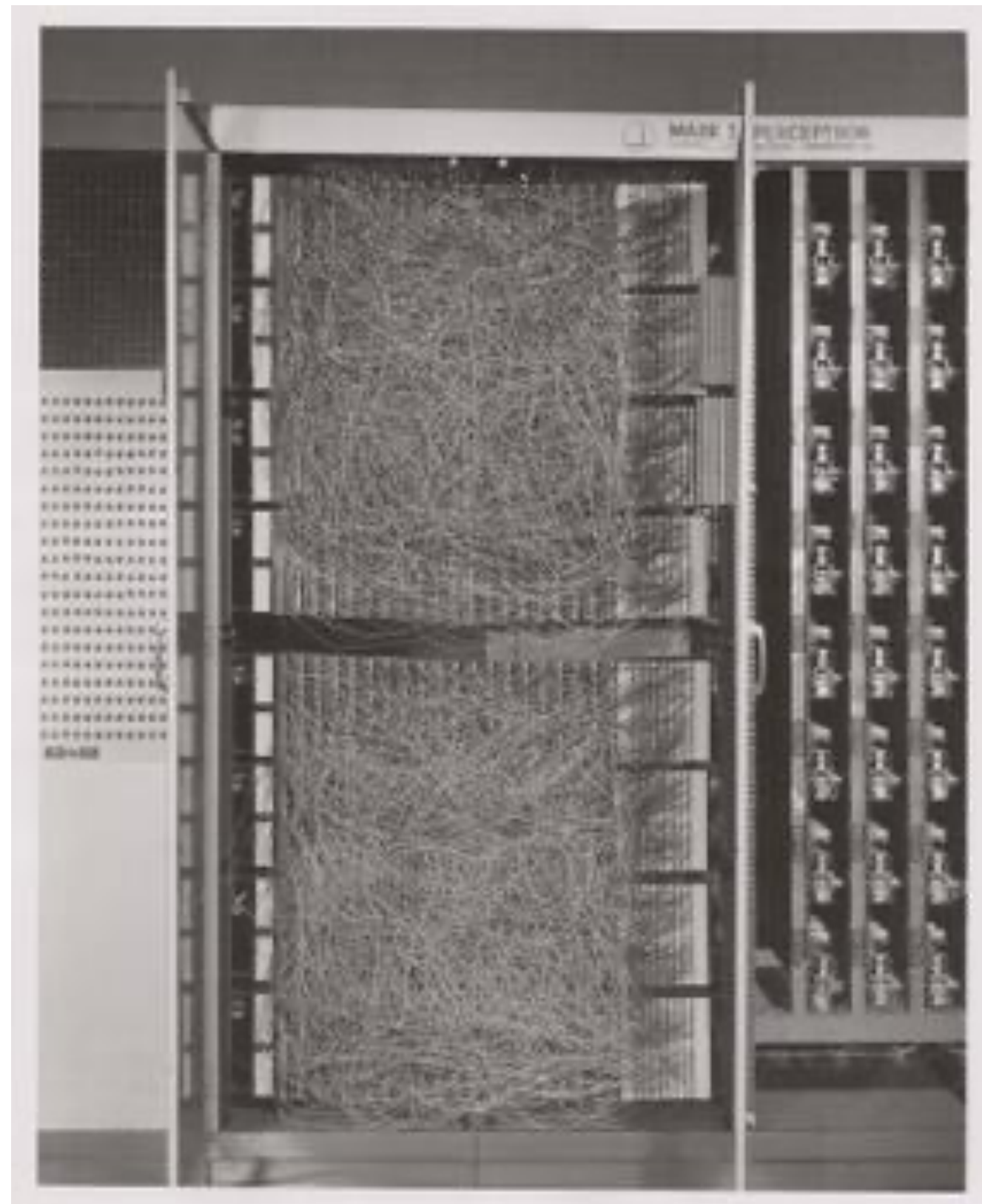
Mark 1 built for image recognition



Neural Networks: Perceptron (*“may eventually be able to learn, make decisions, and translate languages”*)

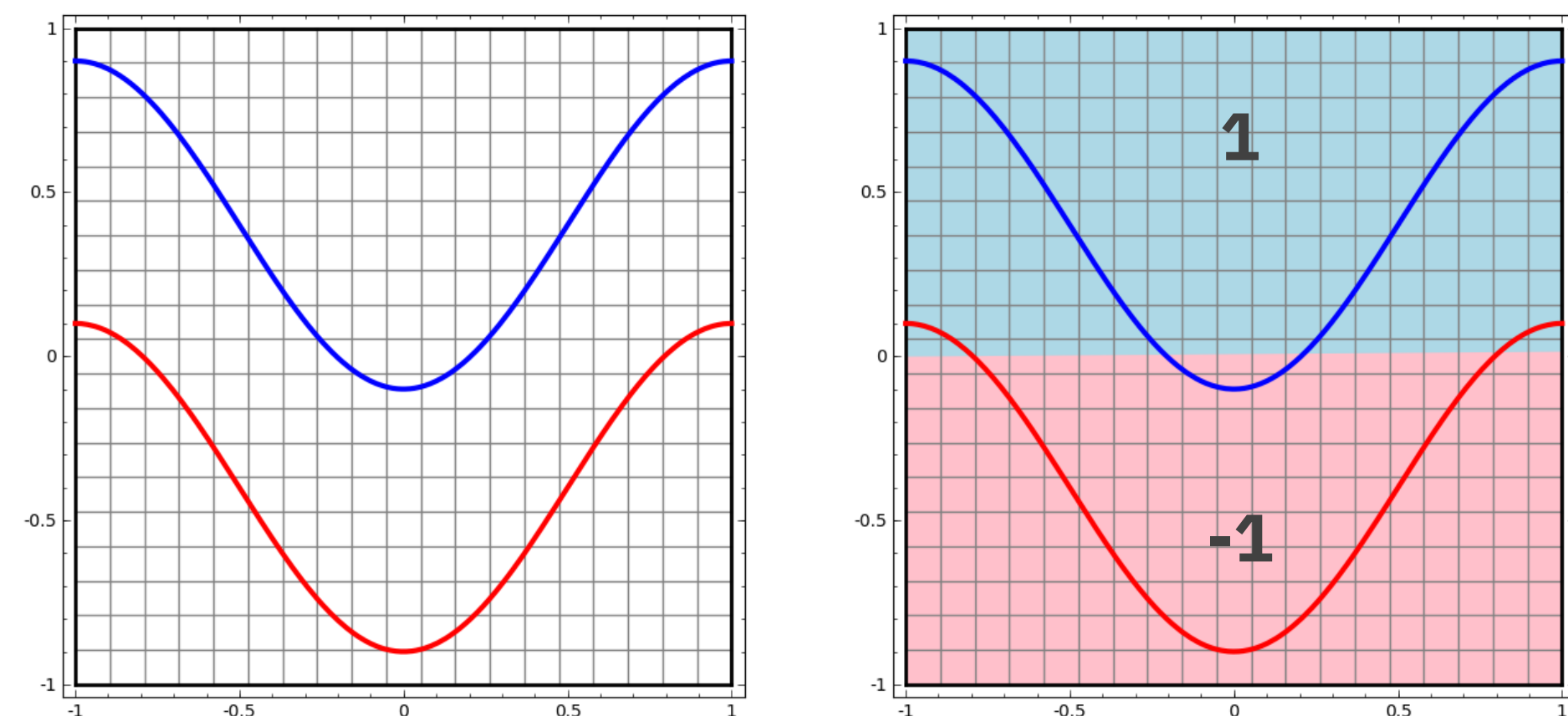
The Perceptron (Rosenblatt (1957))

Mark 1 built for image recognition

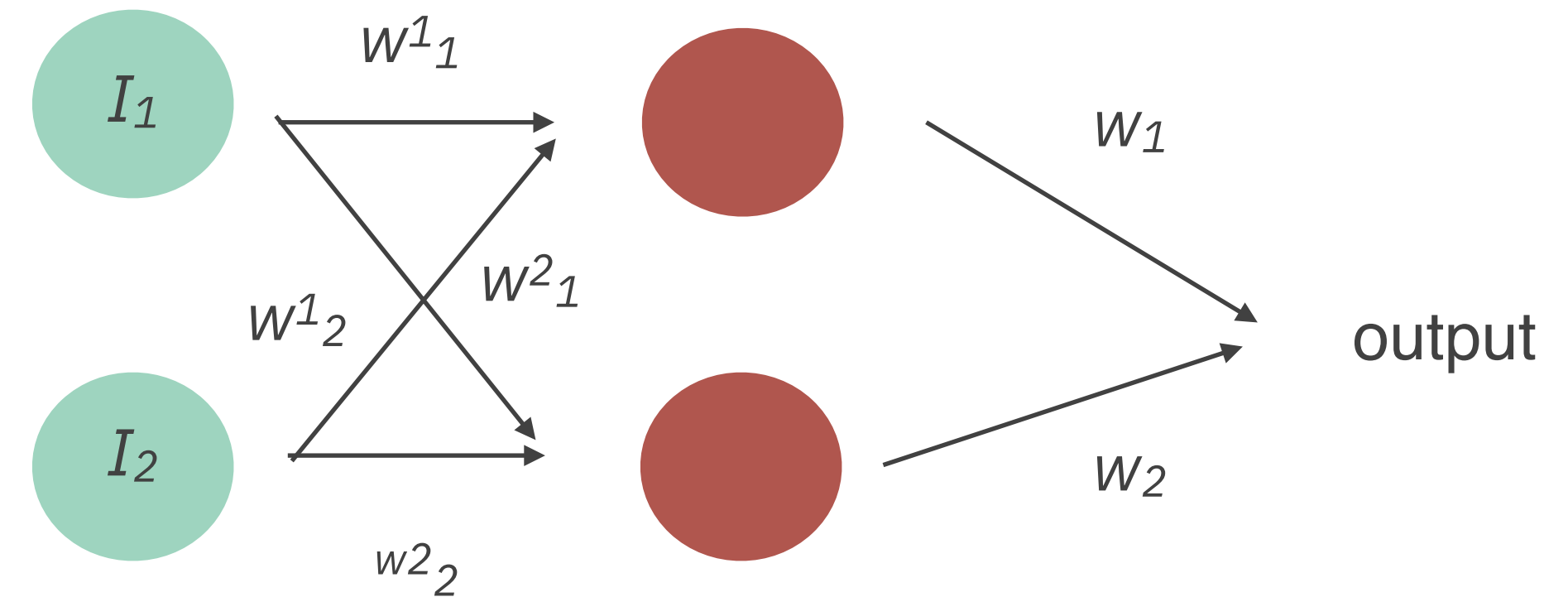


Try to classify which points belong to which curve.

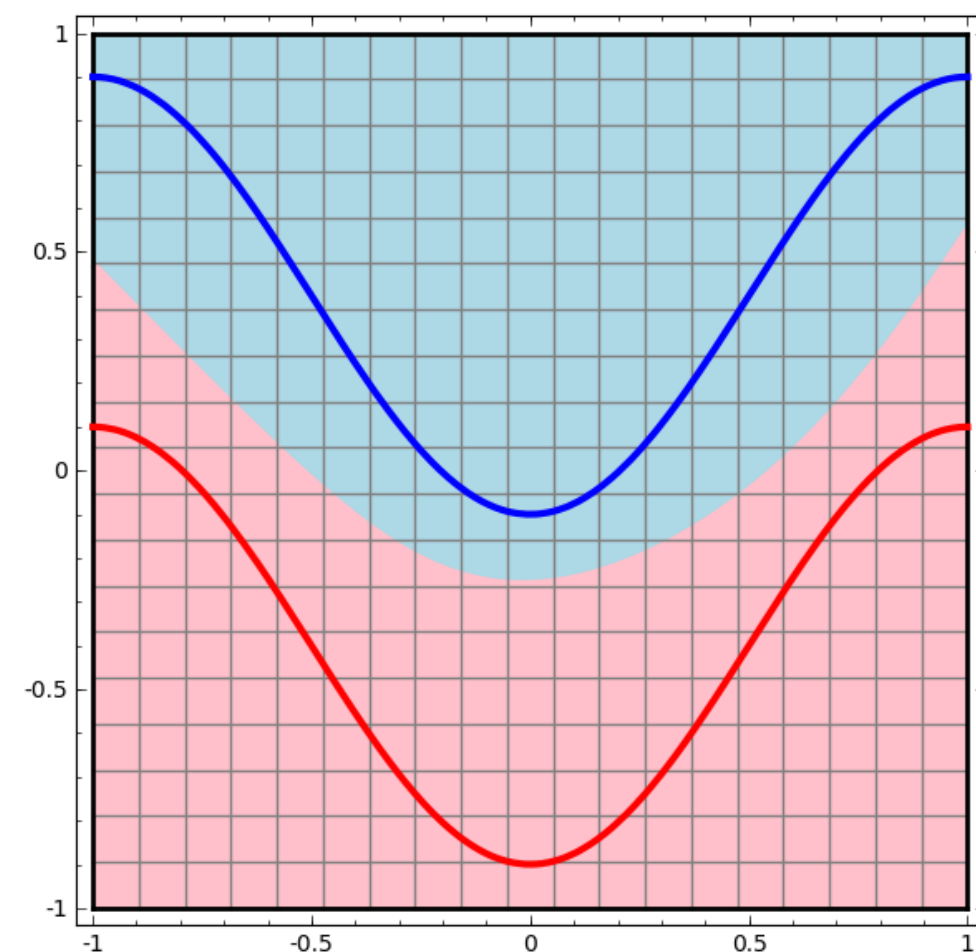
Decision boundary: $w_1x_1 + w_2x_2 + b = 0$



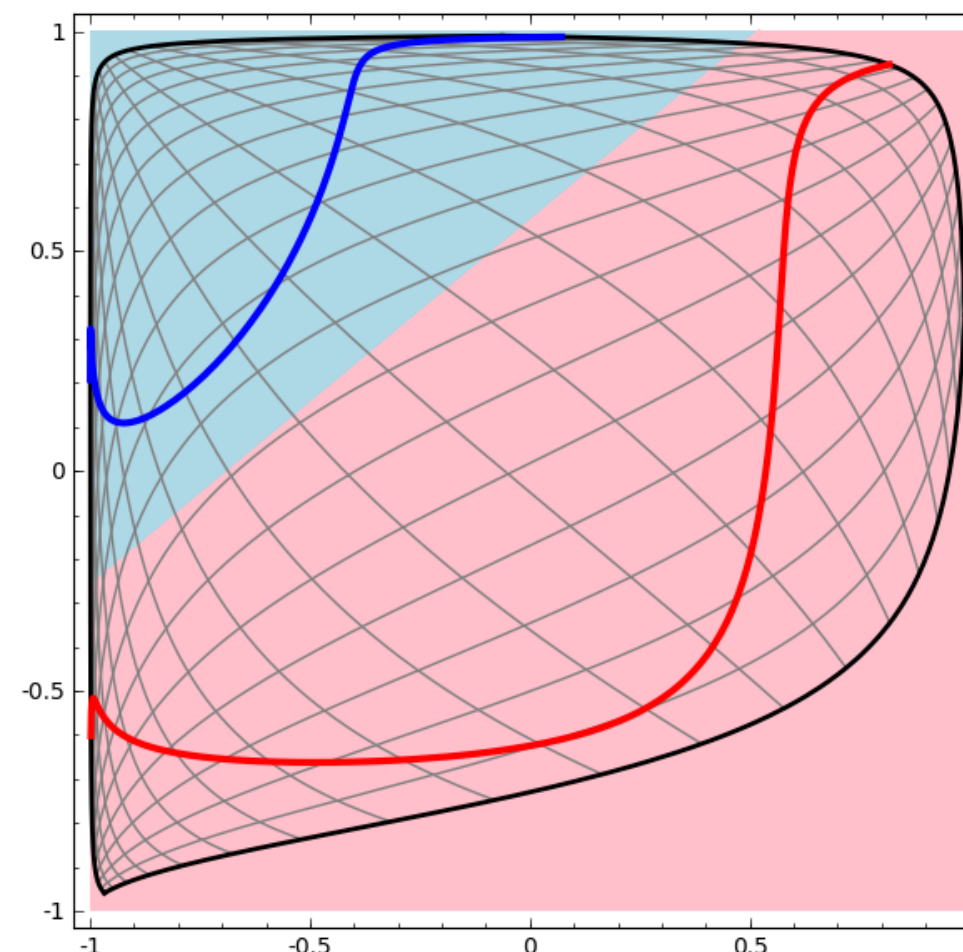
Neural Networks: Representation learning



transform
representation
such that it is linearly
separable

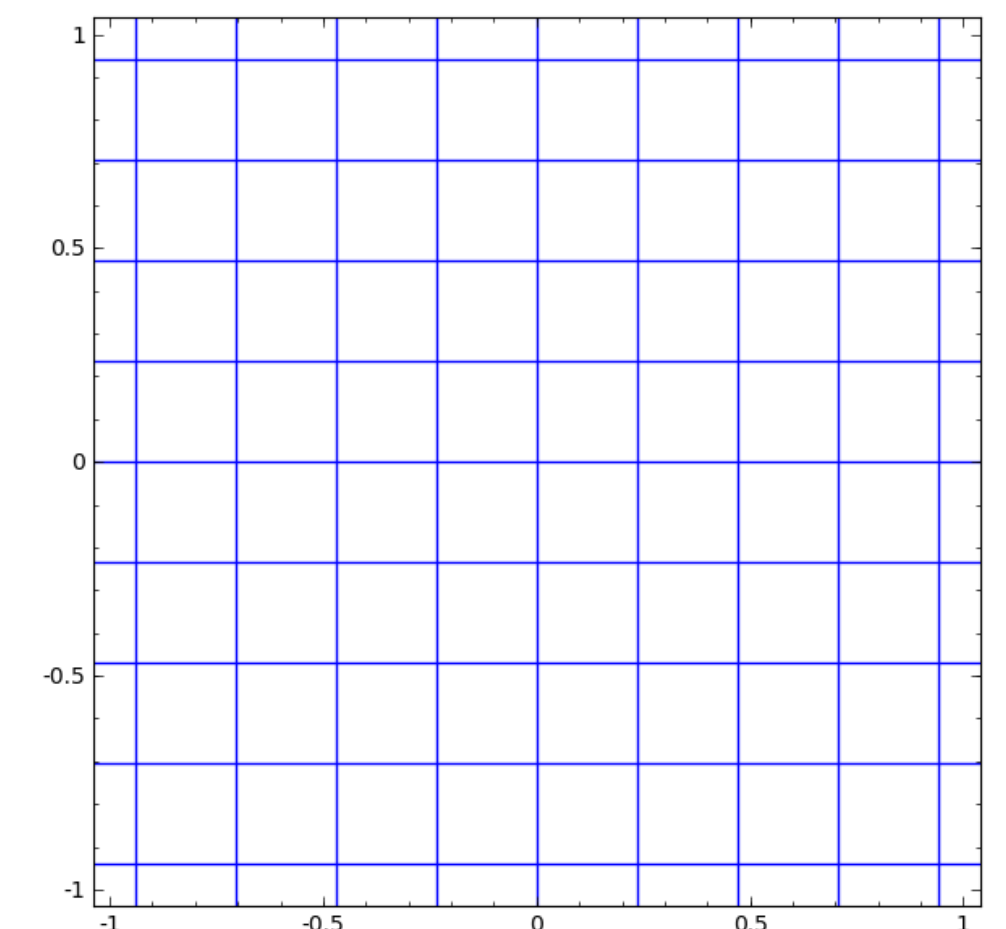


hidden layer, e.g. tanh



in $\tanh(\mathbf{W}\mathbf{x}+\mathbf{b})$:

- rotate ($\mathbf{W}$)
- Translate ($\mathbf{b}$)
- point-wise application of tanh

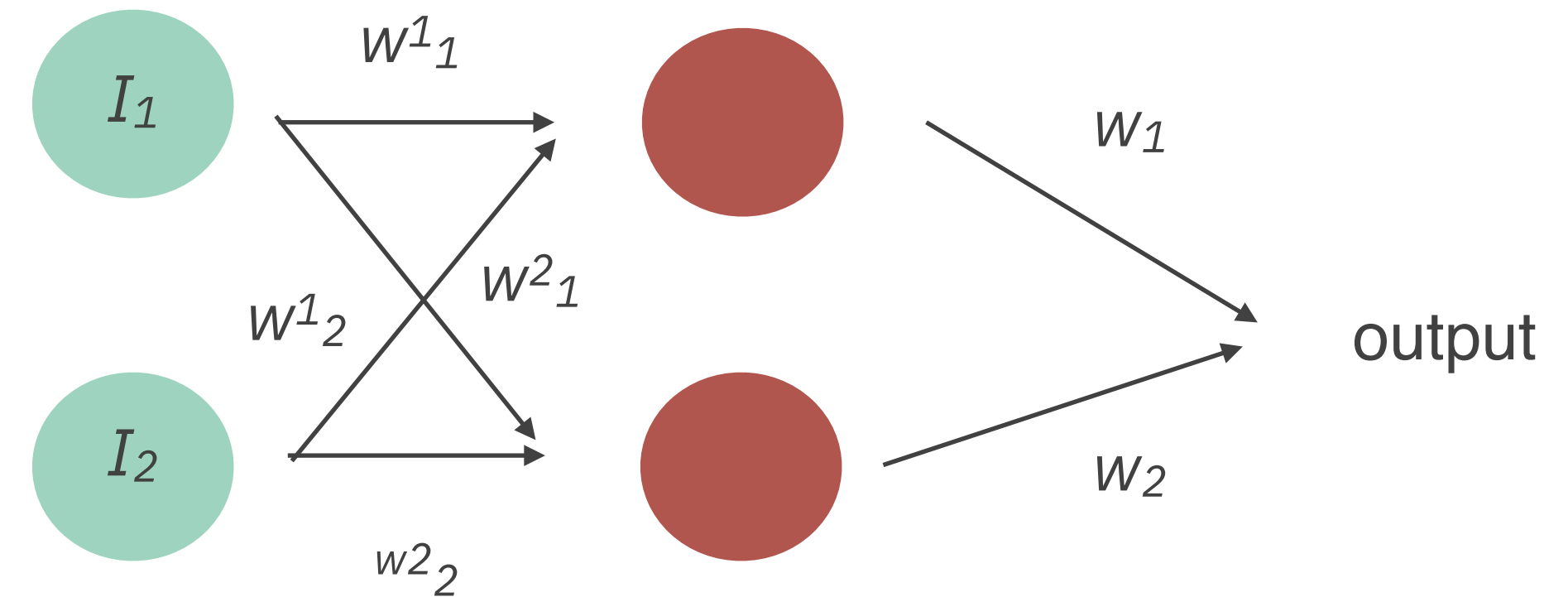


Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*. The MIT Press: Cambridge, Massachusetts, 2016.

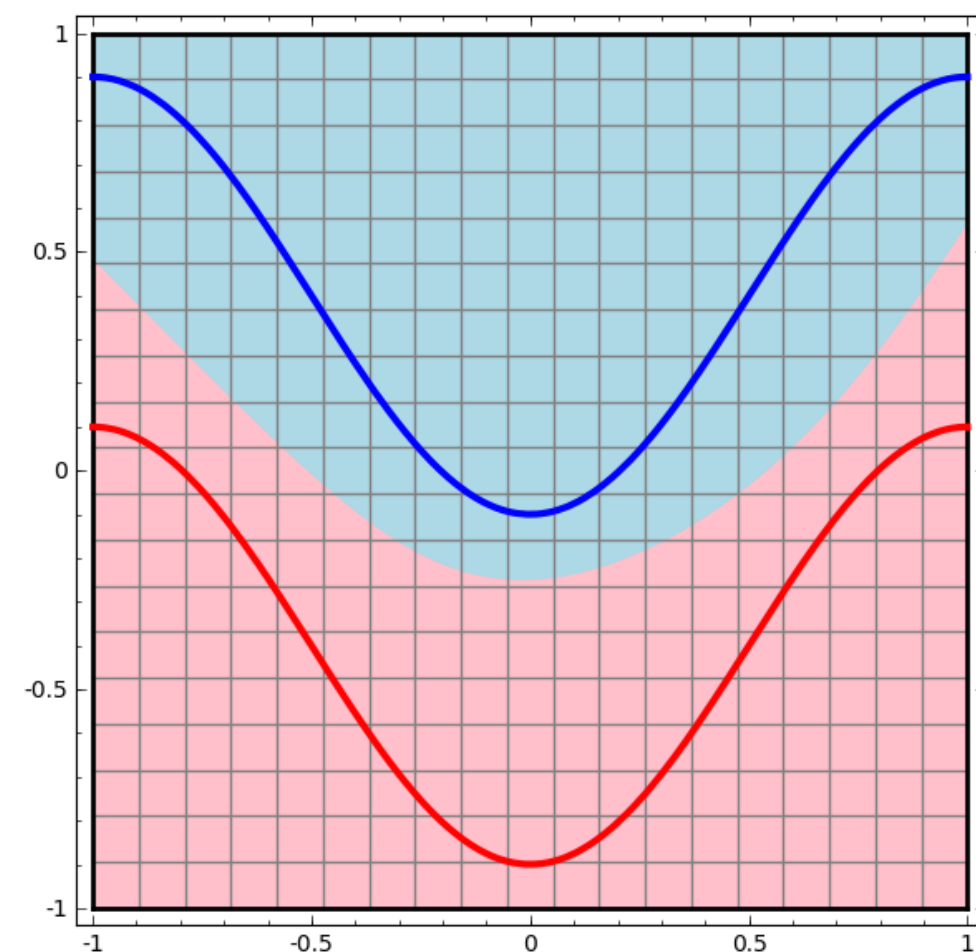
LeCun, Y.; Bengio, Y.; Hinton, G. *Nature* **2015**, 521 (7553), 436–444.

<http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/>

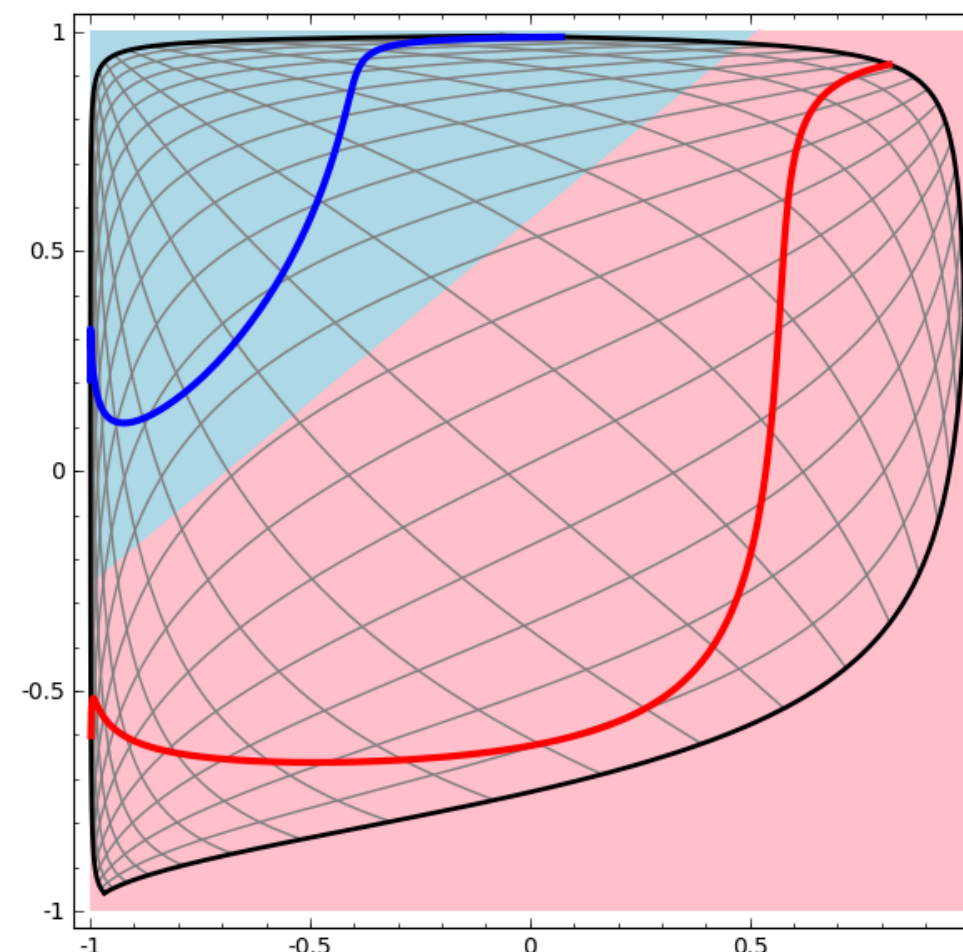
Neural Networks: Representation learning



transform
representation
such that it is linearly
separable

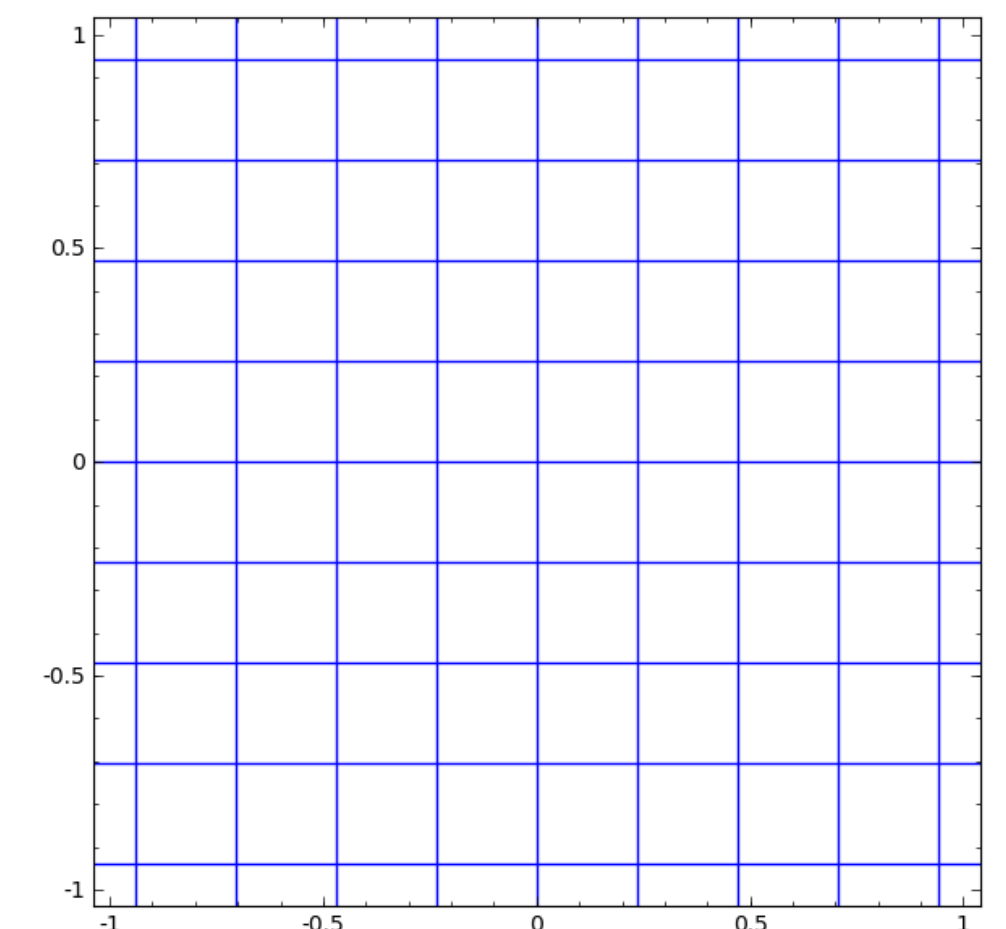


hidden layer, e.g. tanh



in $\tanh(\mathbf{W}\mathbf{x}+\mathbf{b})$:

- rotate ($\mathbf{W}$)
- Translate ($\mathbf{b}$)
- point-wise application of tanh



Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*. The MIT Press: Cambridge, Massachusetts, 2016.

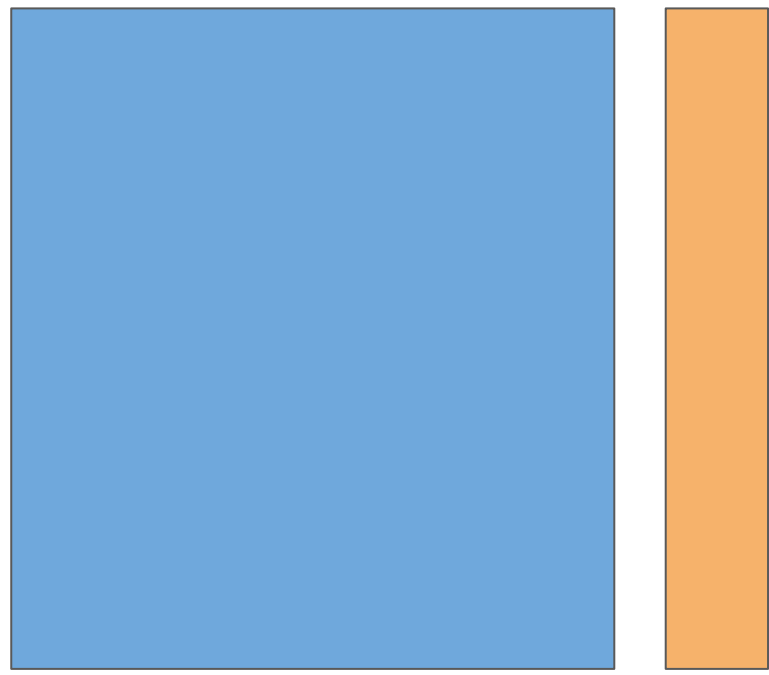
LeCun, Y.; Bengio, Y.; Hinton, G. *Nature* **2015**, 521 (7553), 436–444.

<http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/>

The right architecture for your symmetry

Arrays

⇒ **Dense NN**

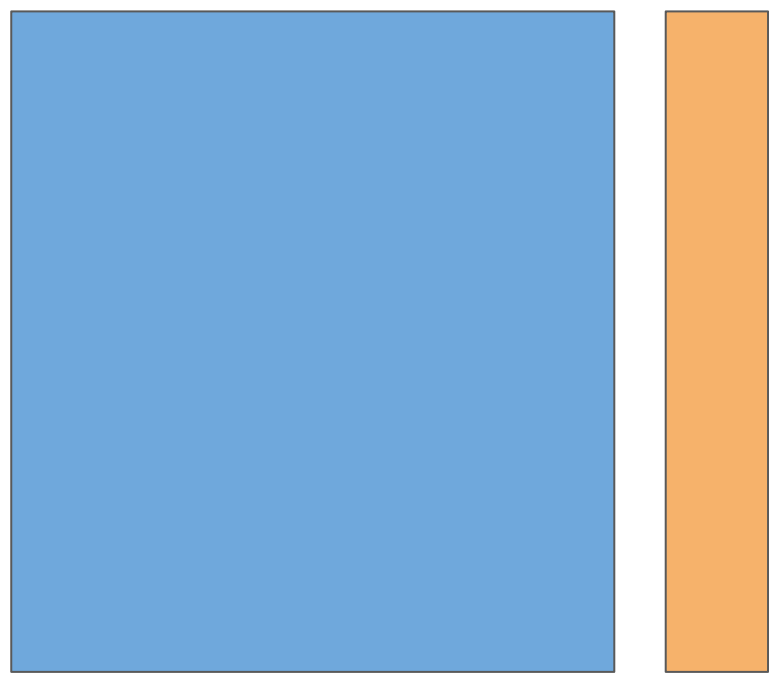


Components are
independent.

The right architecture for your symmetry

Arrays

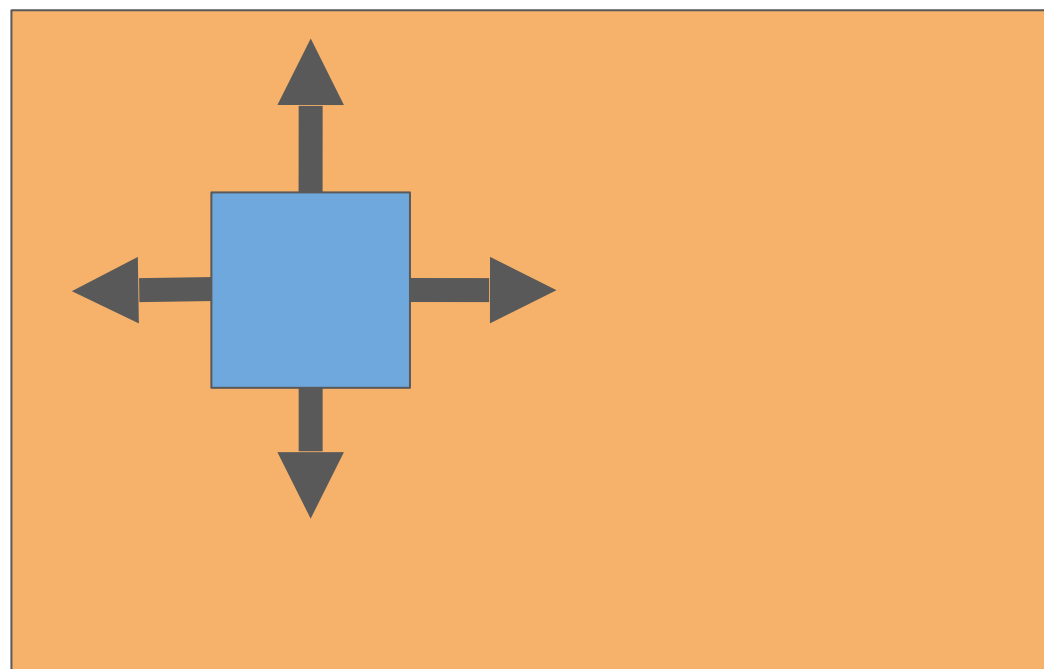
⇒ **Dense NN**



Components are independent.

2D images

⇒ **Convolutional NN**

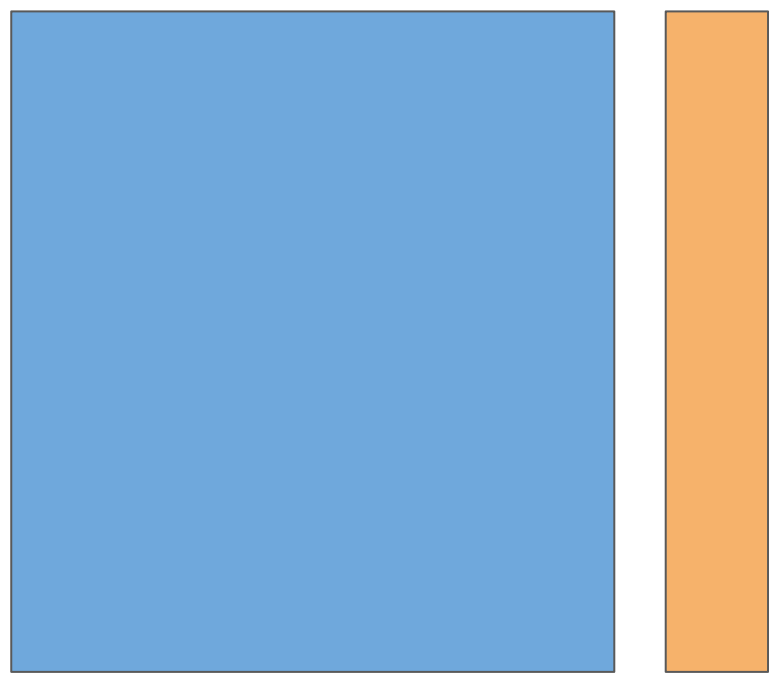


The same features can be found anywhere in an image.
Locality.

The right architecture for your symmetry

Arrays

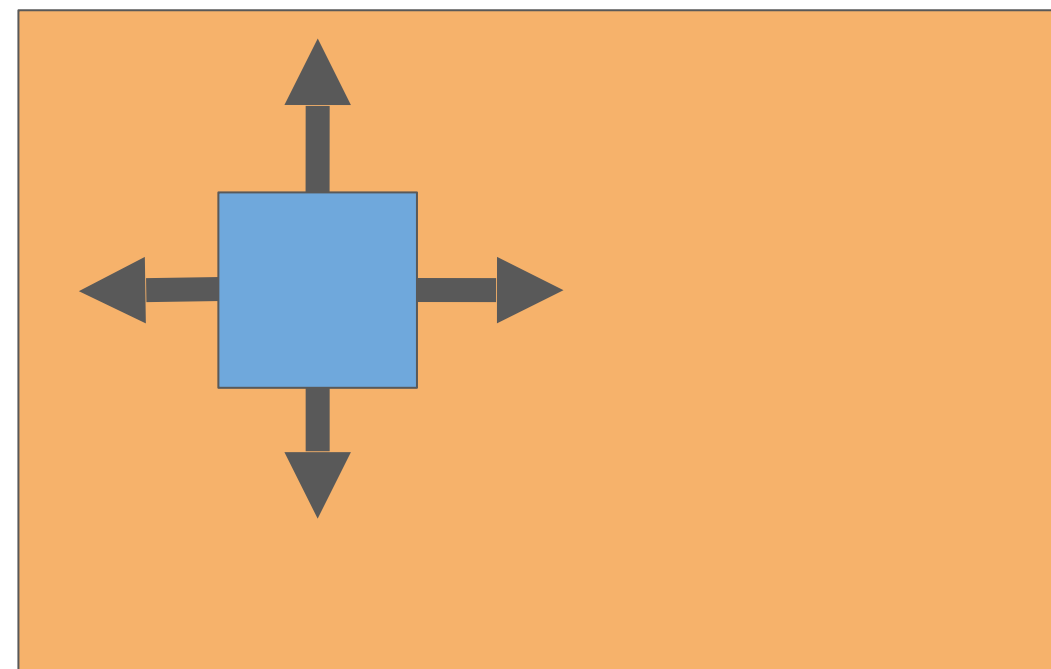
⇒ **Dense NN**



Components are independent.

2D images

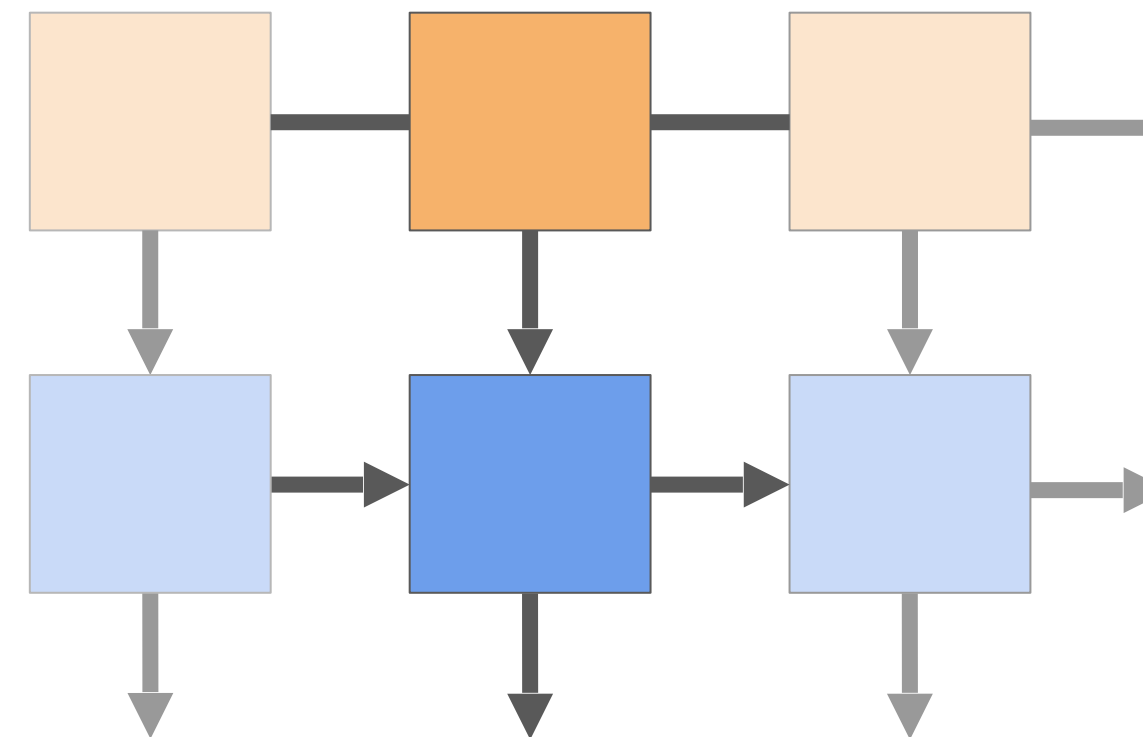
⇒ **Convolutional NN**



The same features can be found anywhere in an image. Locality.

Text

⇒ **Recurrent NN**

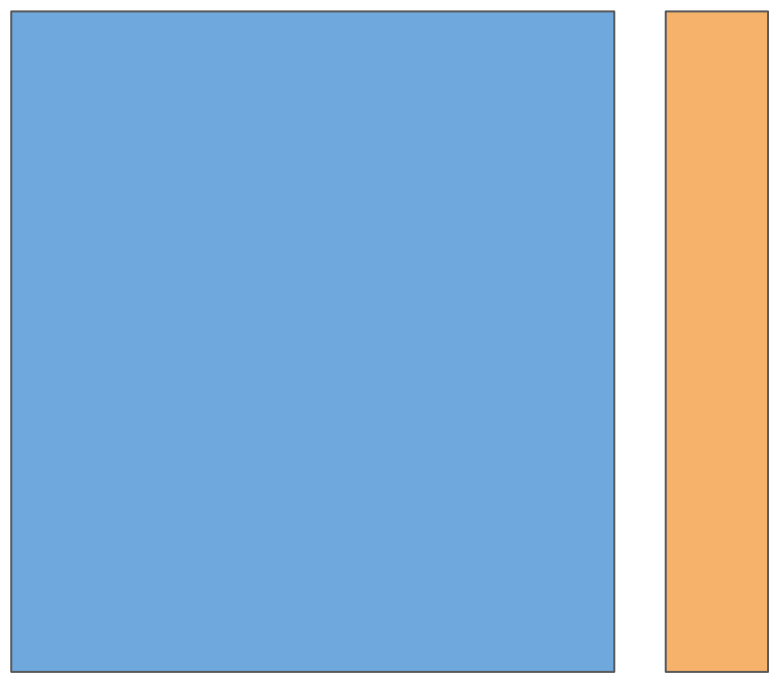


Sequential data. Next input/output depends on input/output that has come before.

The right architecture for your symmetry

Arrays

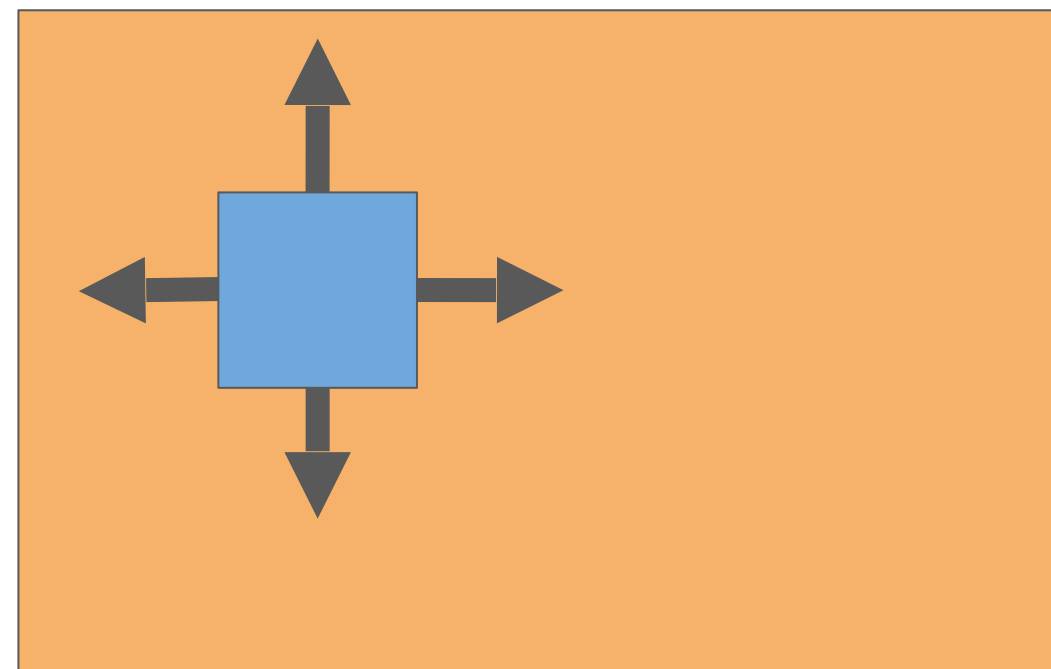
⇒ **Dense NN**



Components are independent.

2D images

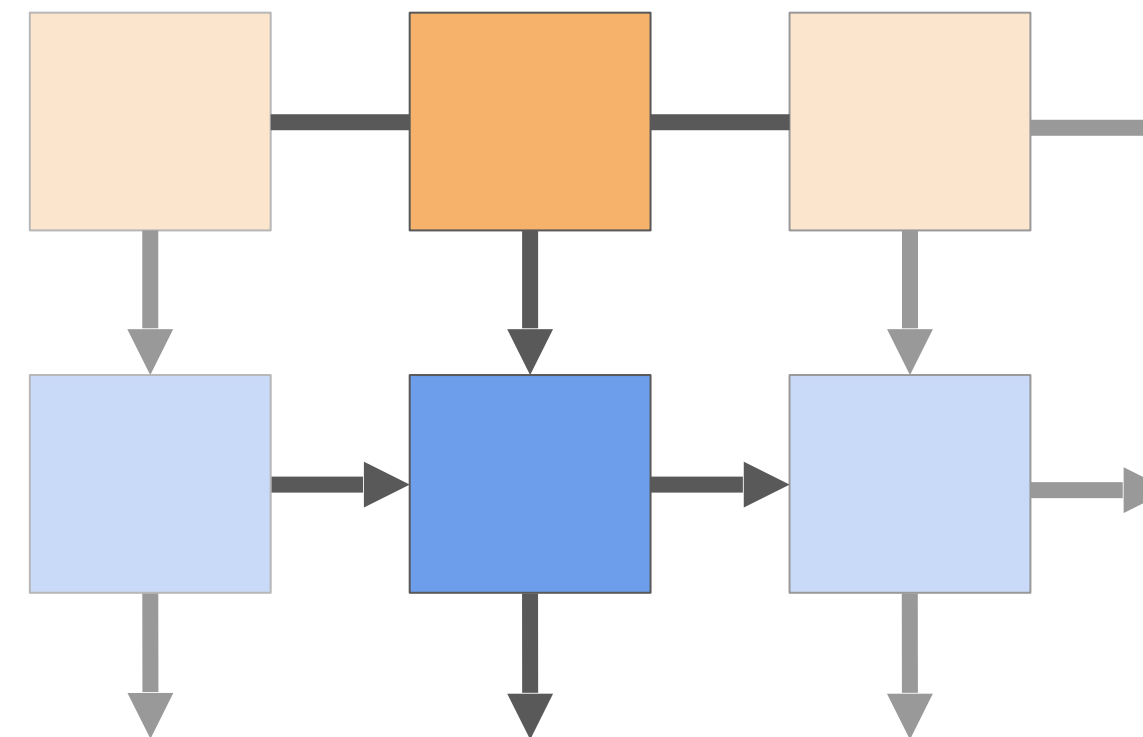
⇒ **Convolutional NN**



The same features can be found anywhere in an image. Locality.

Text

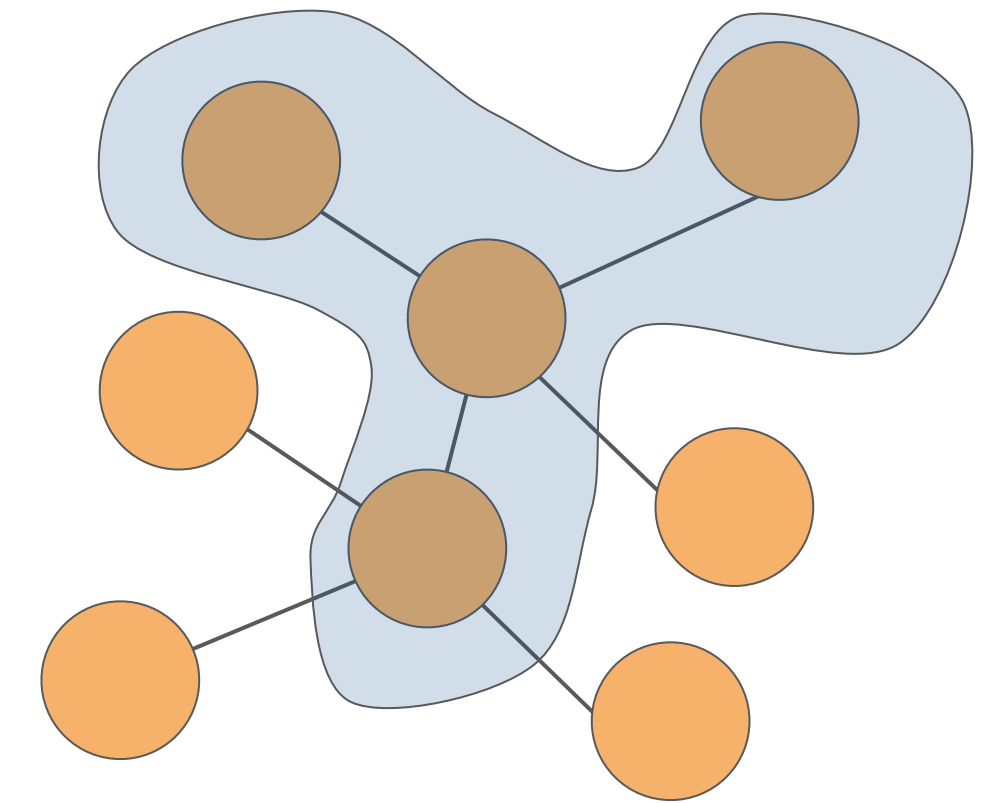
⇒ **Recurrent NN**



Sequential data. Next input/output depends on input/output that has come before.

Graph

⇒ **Graph (Conv.) NN**

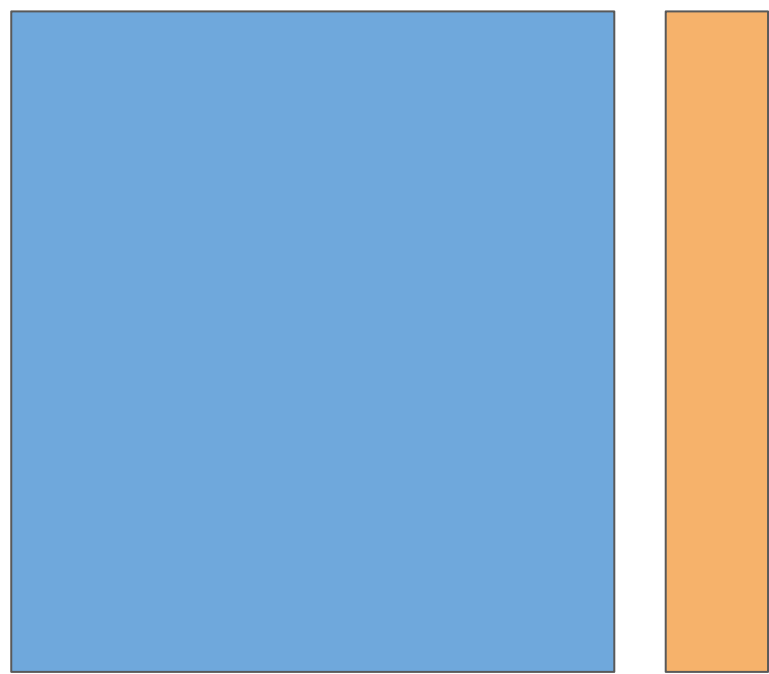


Non-Euclidean data. Nodes have features and network passes messages between nodes connected via edges.

The right architecture for your symmetry

Arrays

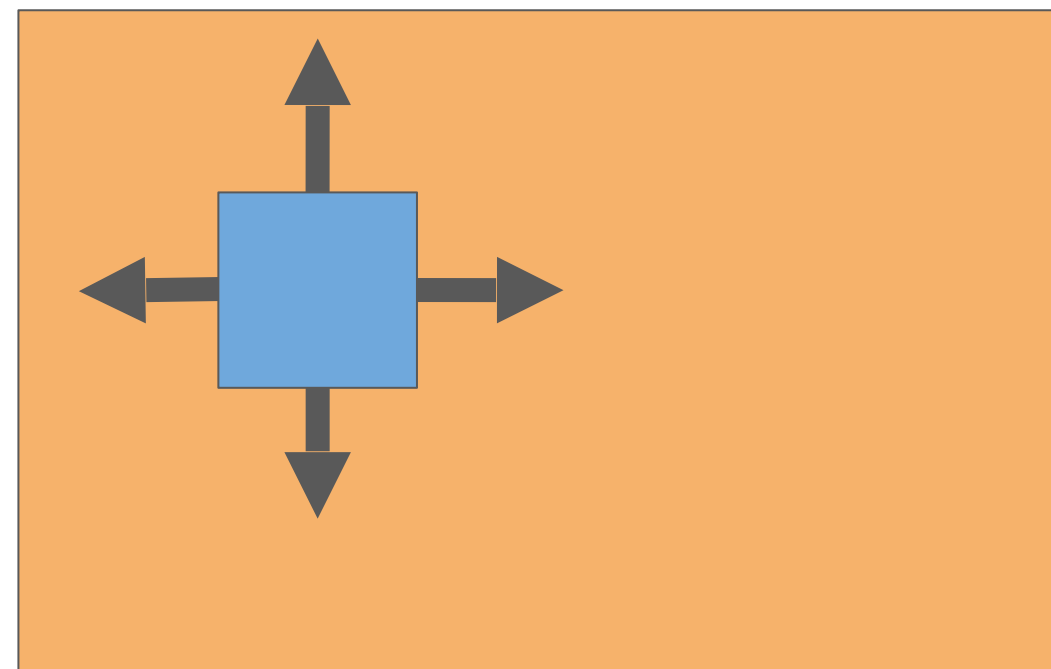
⇒ **Dense NN**



Components are independent.

2D images

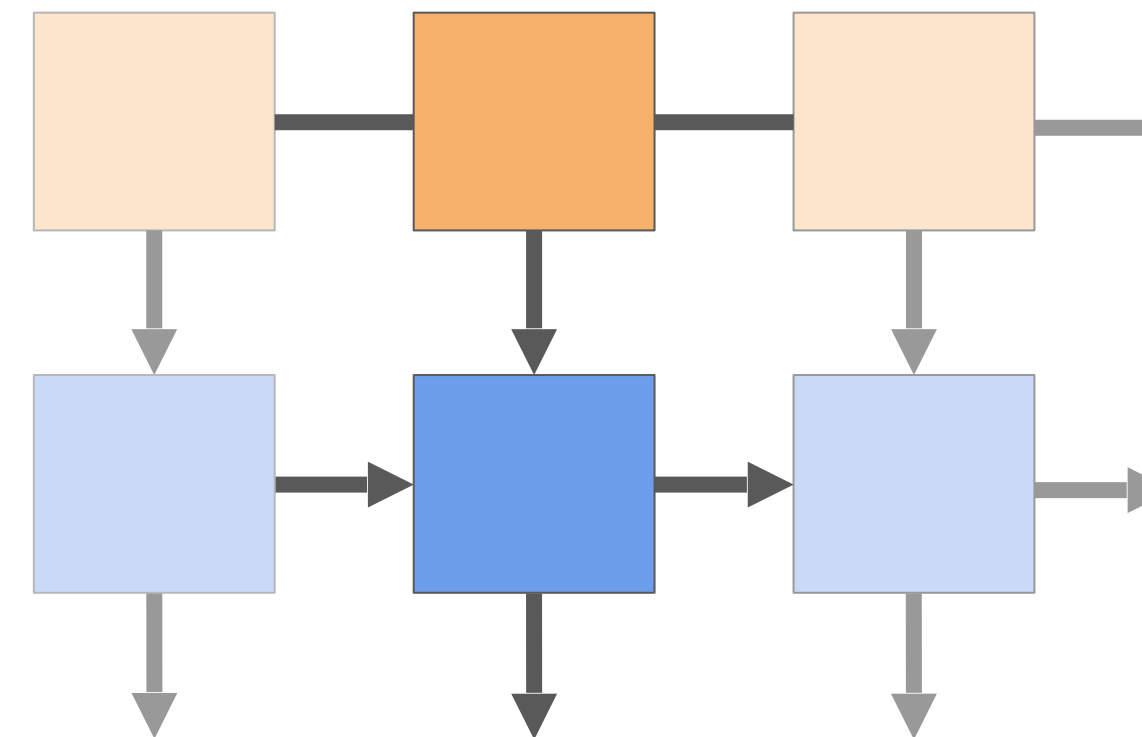
⇒ **Convolutional NN**



The same features can be found anywhere in an image. Locality.

Text

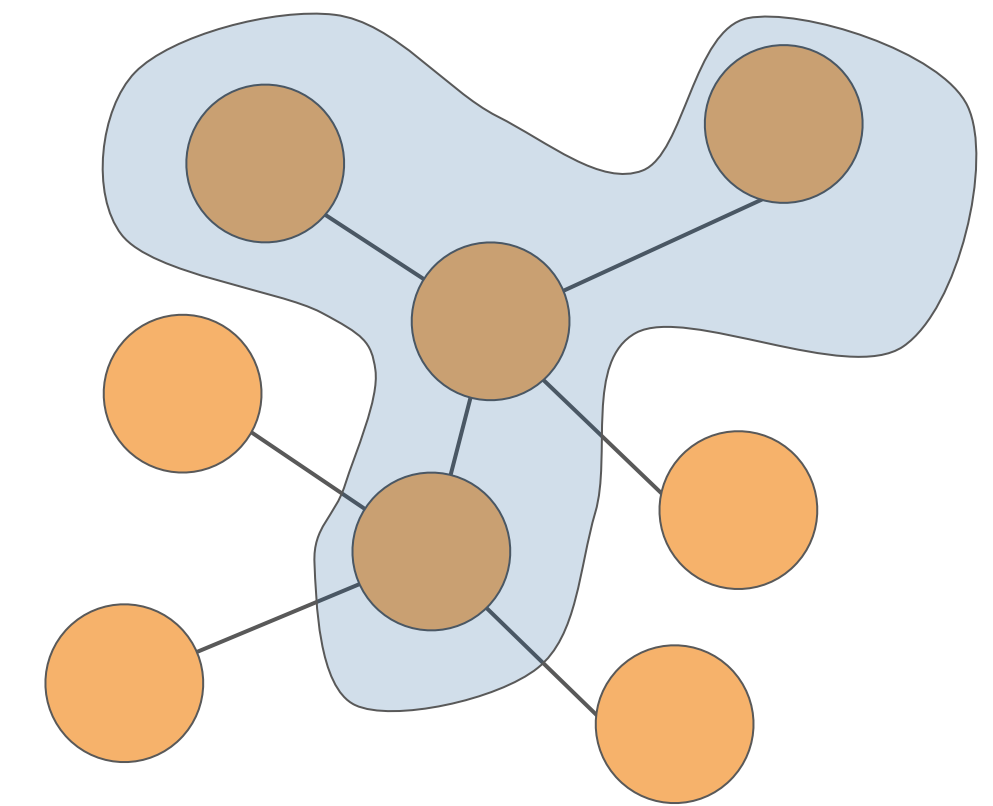
⇒ **Recurrent NN**



Sequential data. Next input/output depends on input/output that has come before.

Graph

⇒ **Graph (Conv.) NN**

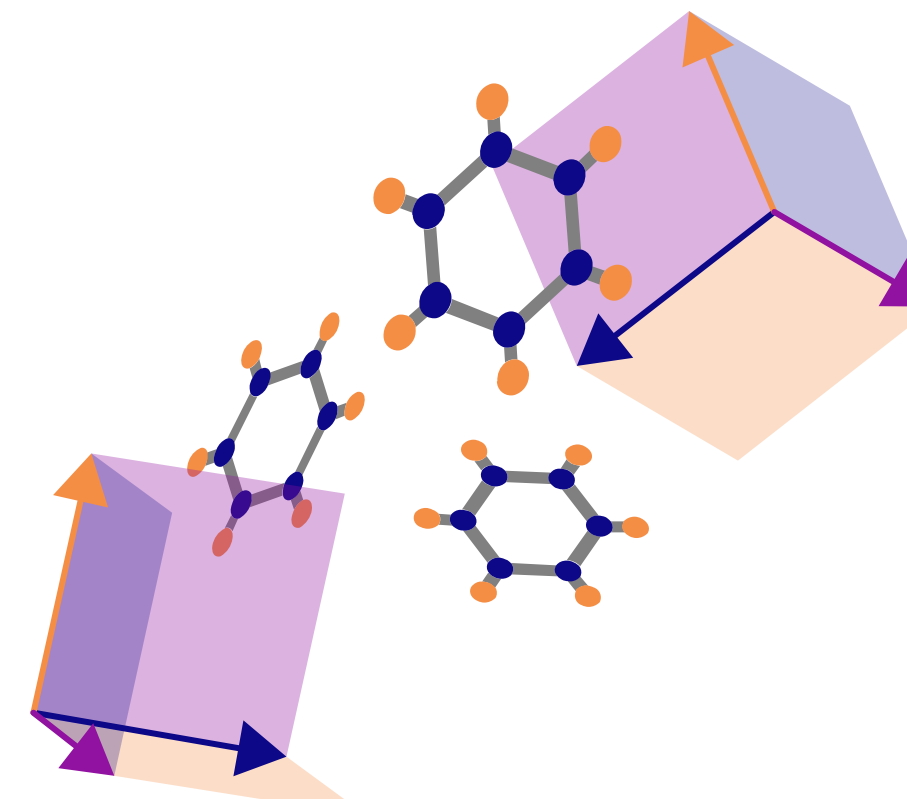


Non-Euclidean data. Nodes have features and network passes messages between nodes connected via edges.

3D physical data

⇒ **Euclidean NN**

Data in 3D Euclidean space.



10. Feature Learning

Learning Objectives:

- Multiple layers with non-linearities can act as universal function approximations
- Intermediate layers learn predictive representations
- Symmetry of problem guides architecture choice

Applications

Learning Objectives:

- ML “force fields”
- ML for efficient sampling
- Using natural language processing for materials discovery

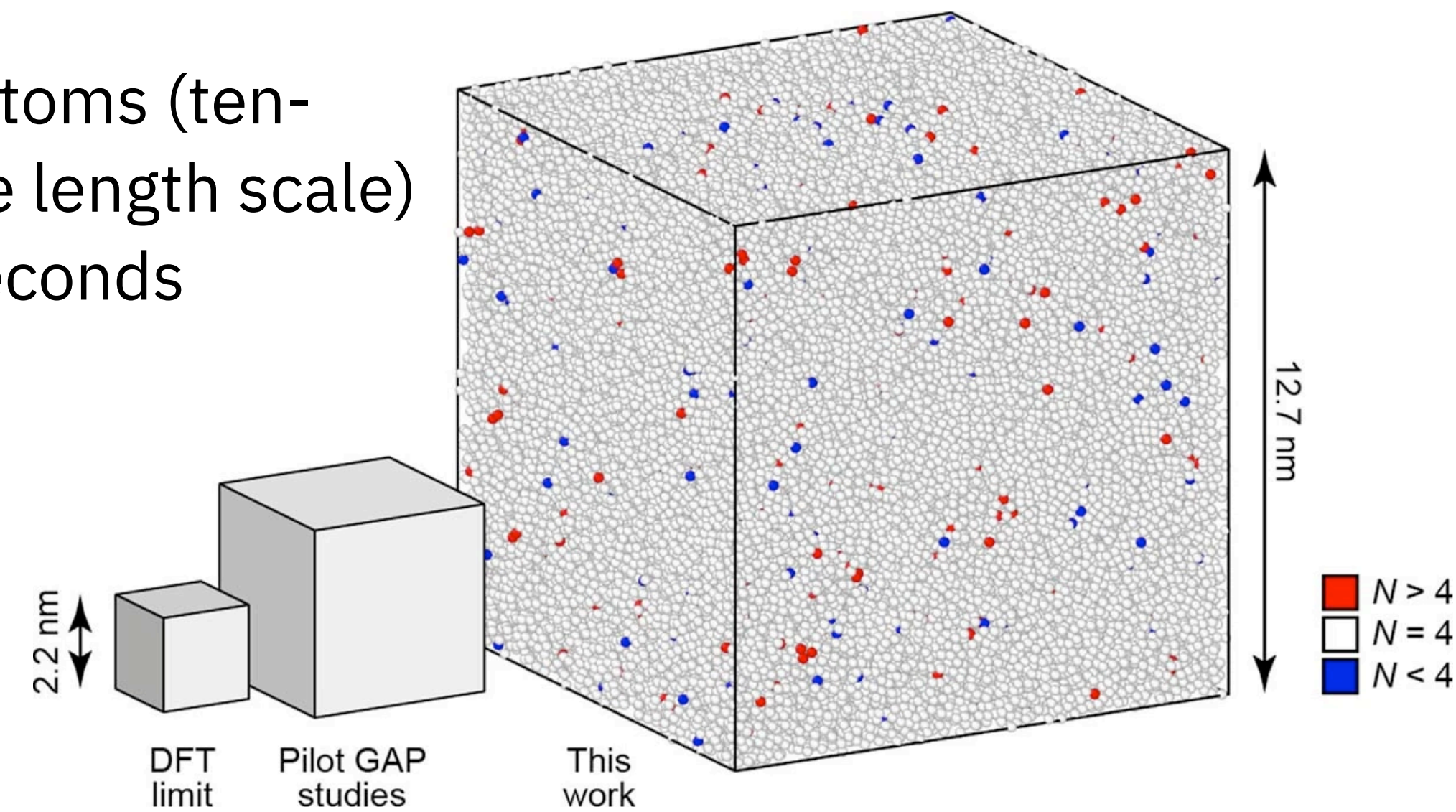
Machine learned force fields

- High accuracy for long time scales and/or large systems
- ML potentials are close to being a “commodity” technique



Nature **589**, 59–64 2021.

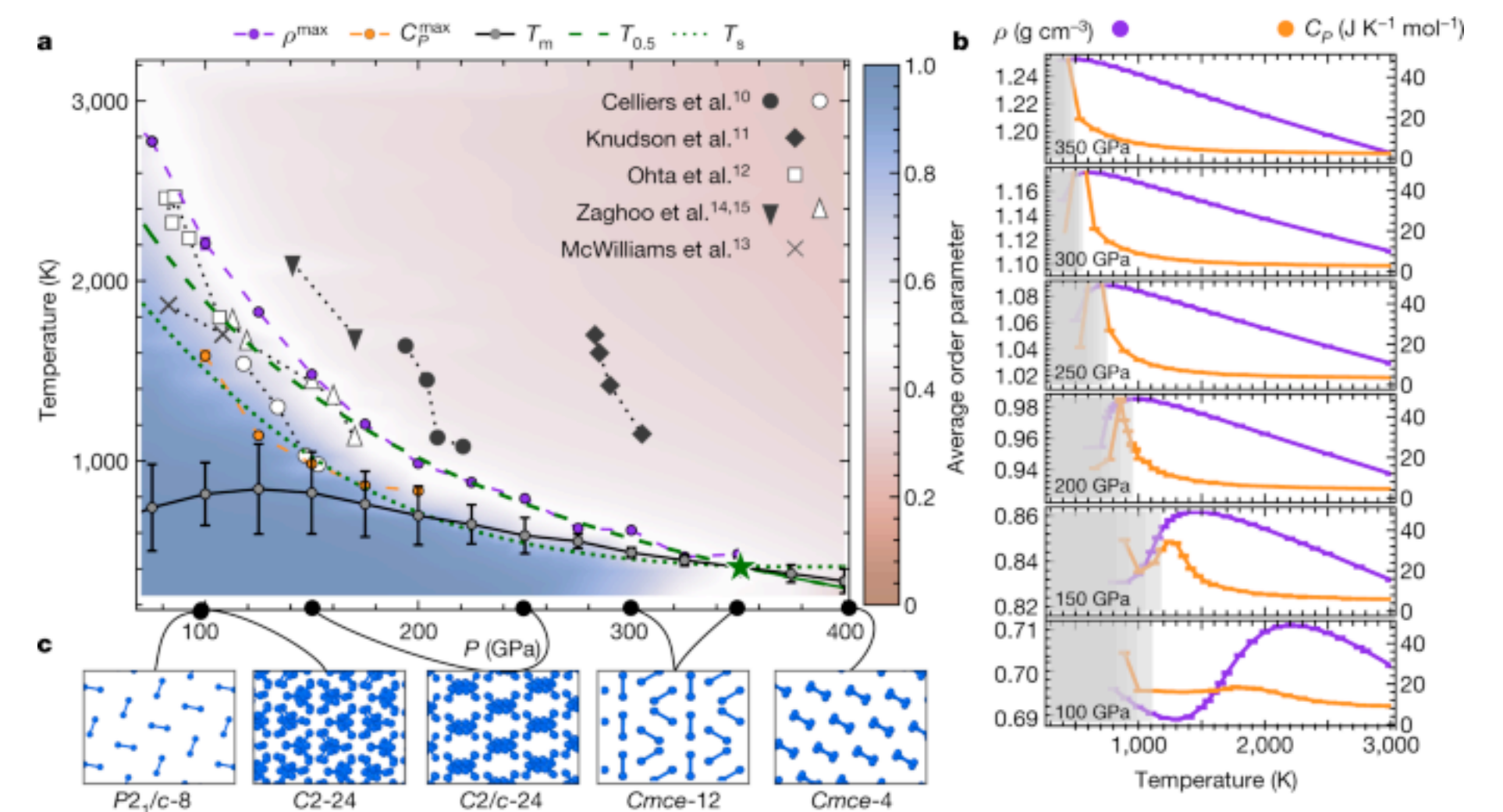
100,000 atoms (ten-nanometre length scale) for nanoseconds



Hydrogen in planets
(under high pressure):

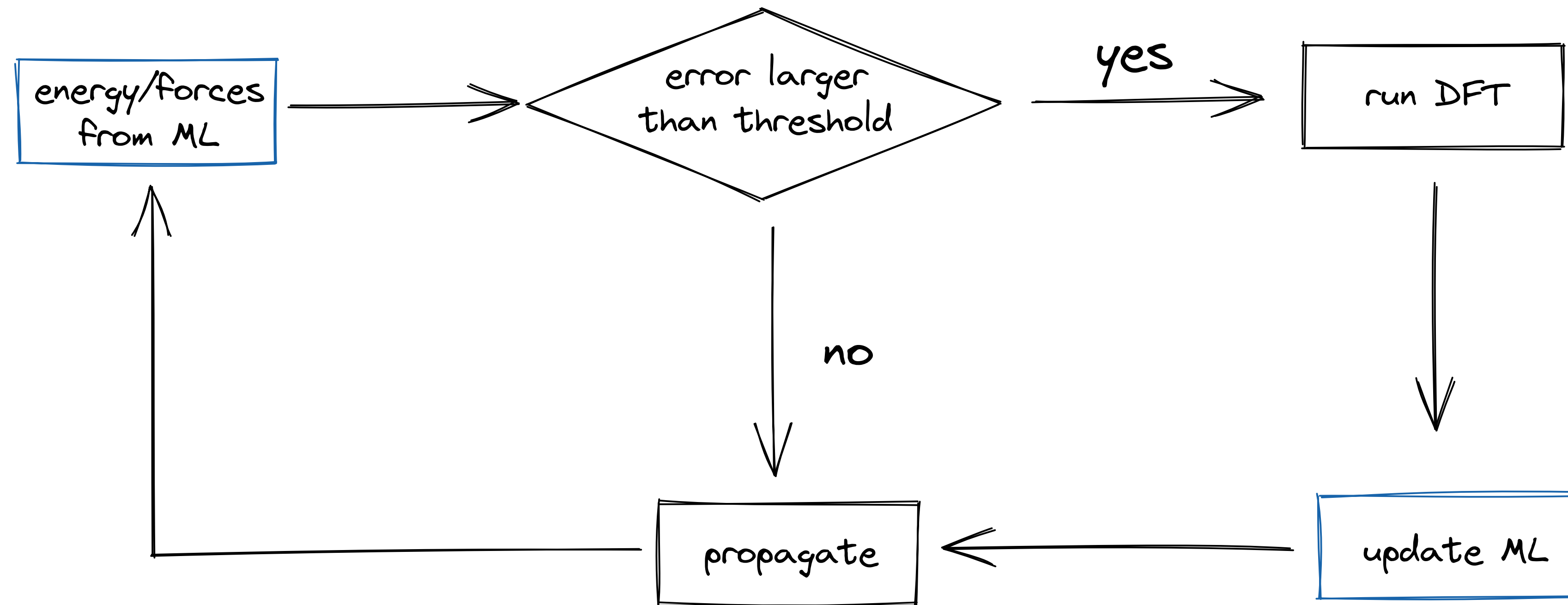
Thousands of atoms for nanoseconds and milli-electronvolt energy differences

Nature **585**, 217–220 2020.



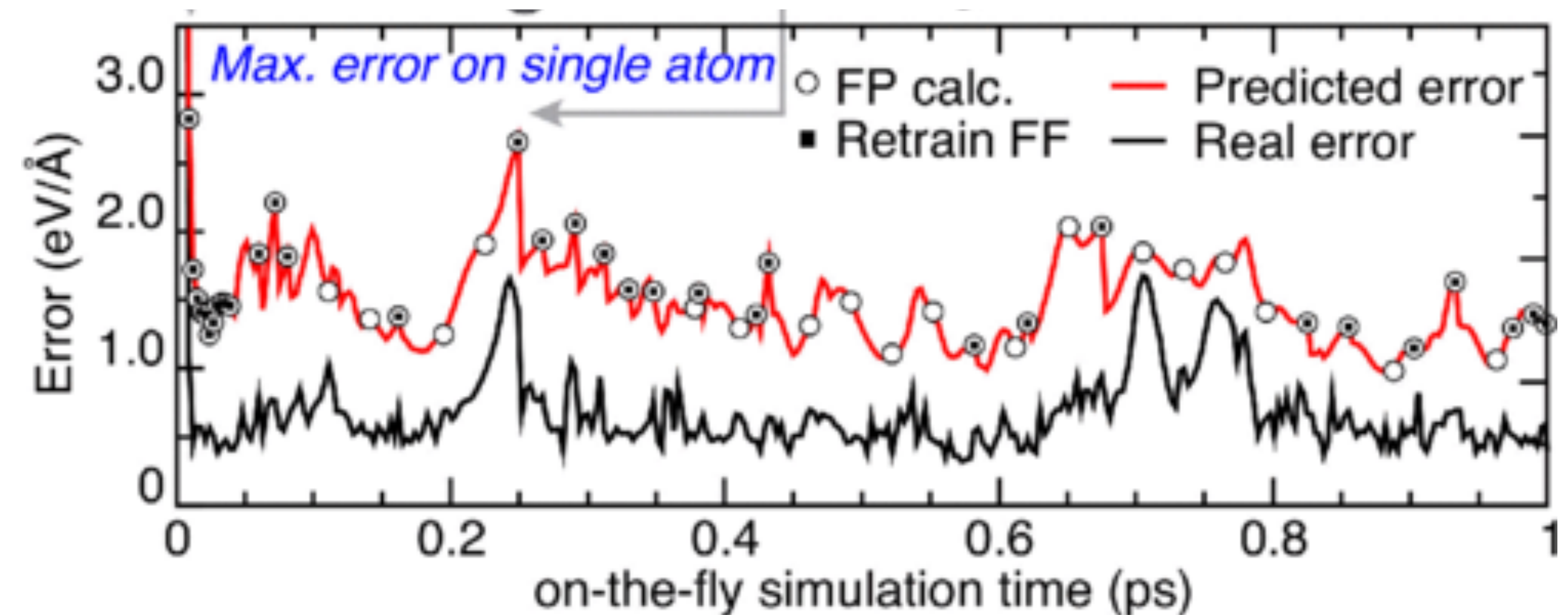
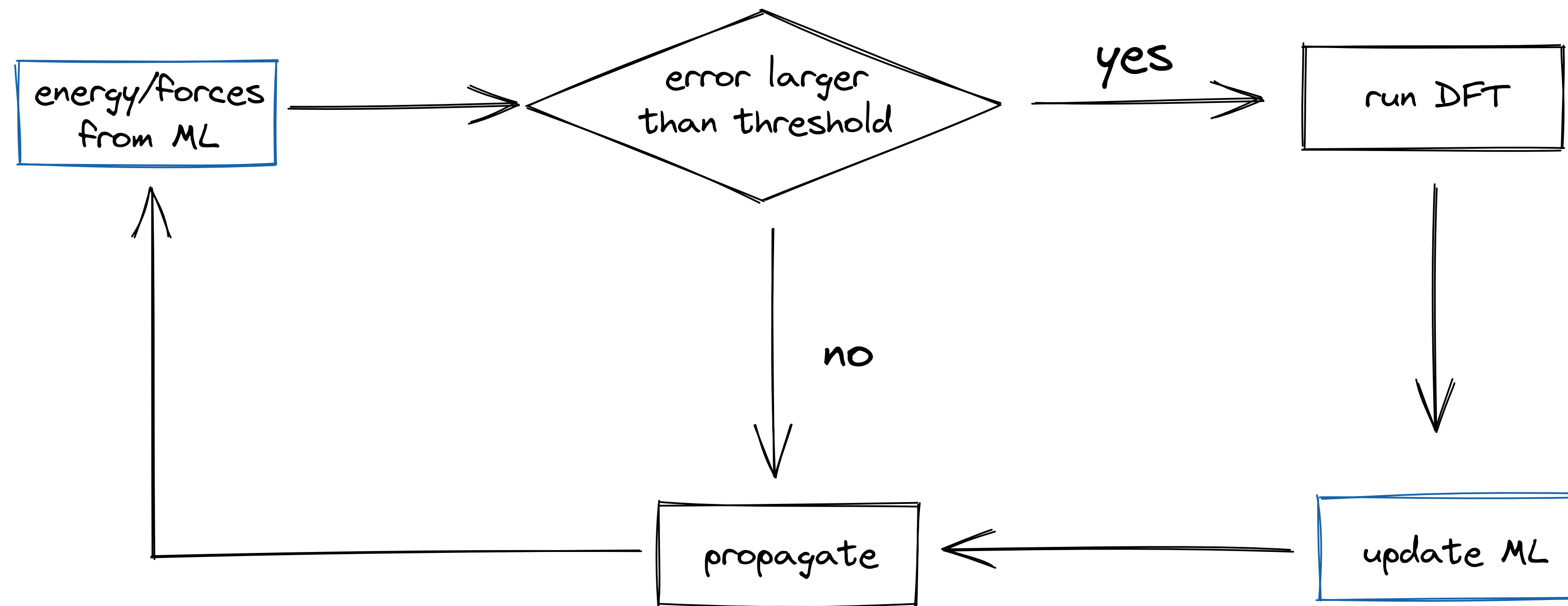
Machine learned force fields

DFT accuracy with only 1% DFT!

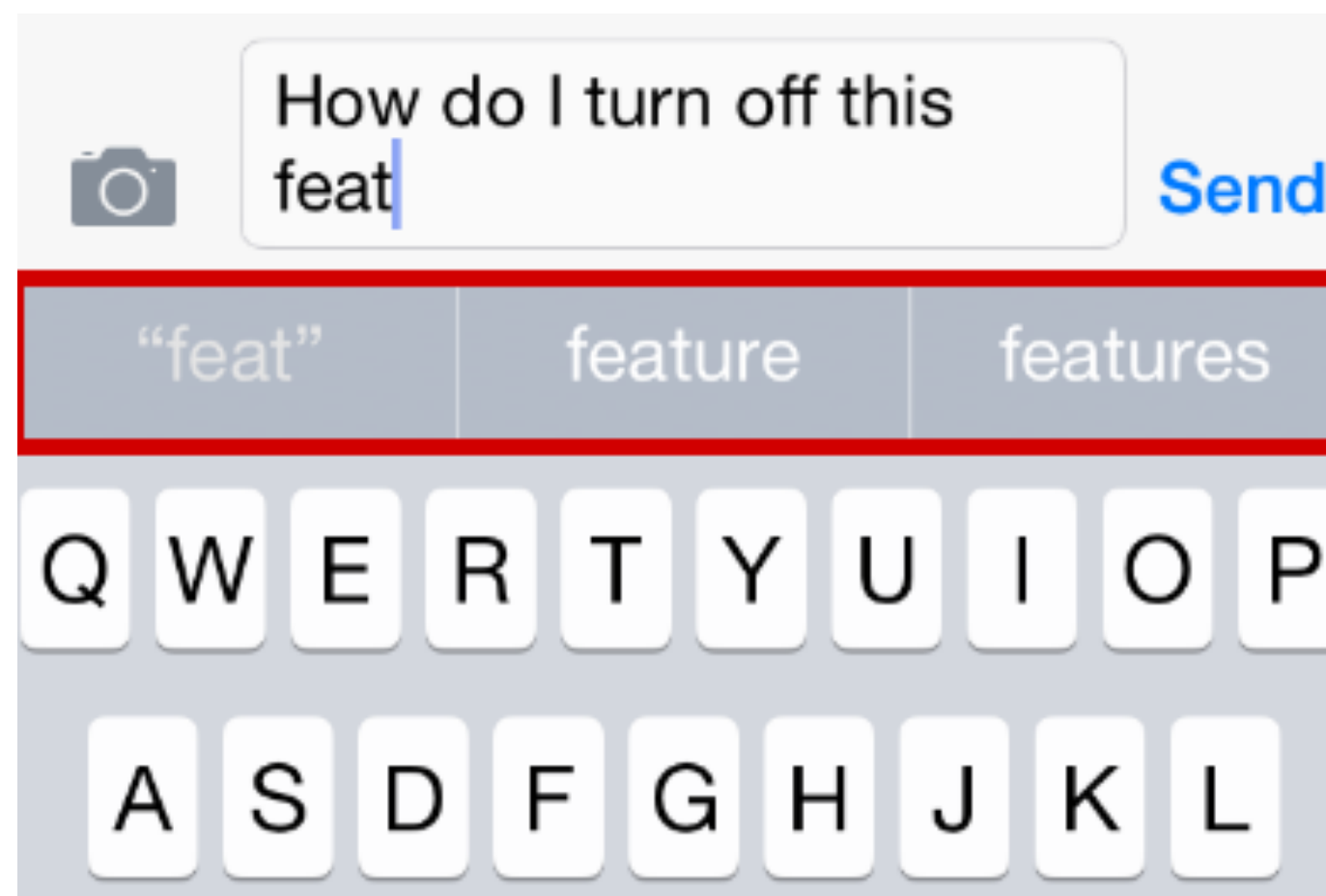


Machine learned force fields

DFT accuracy with only 1% DFT!



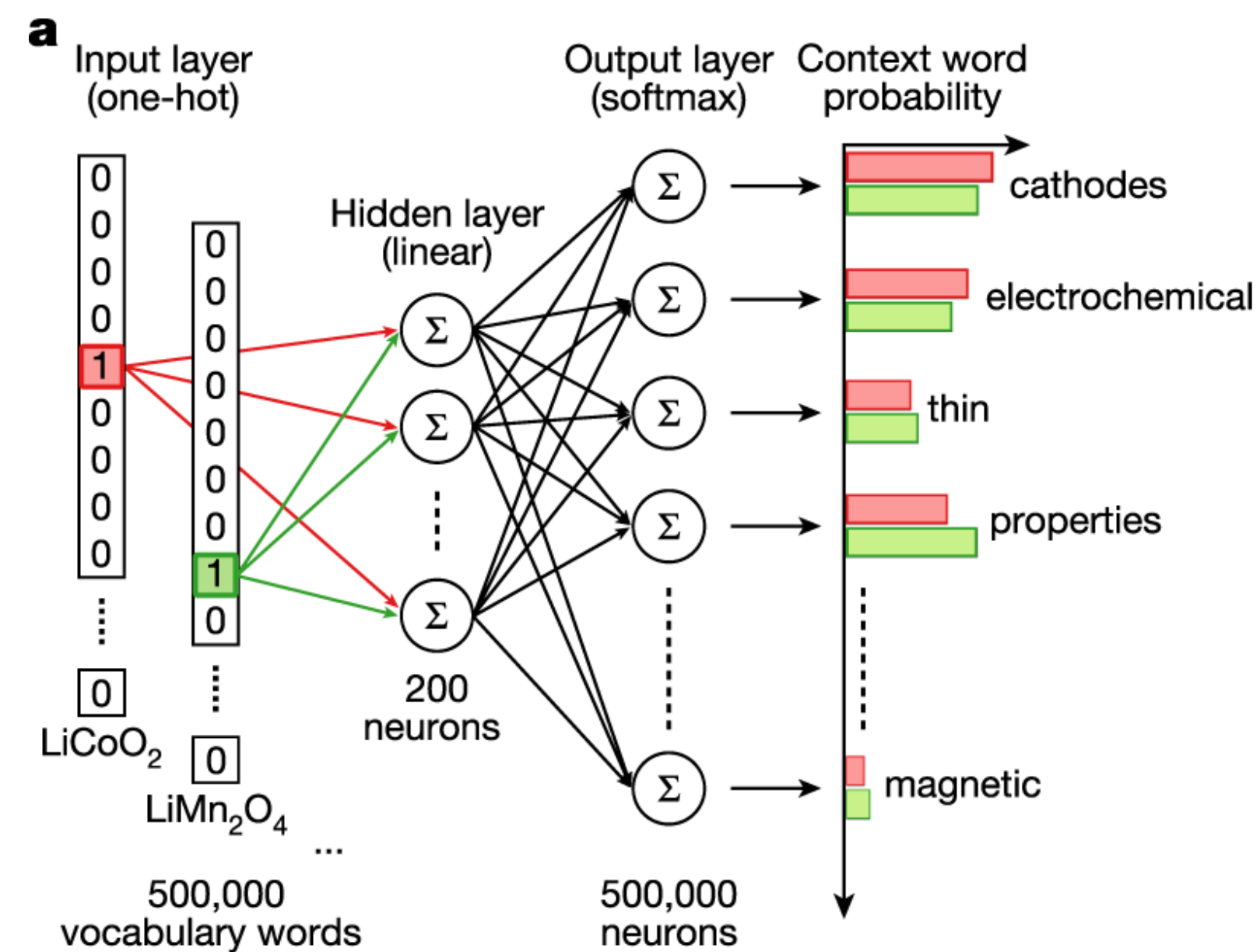
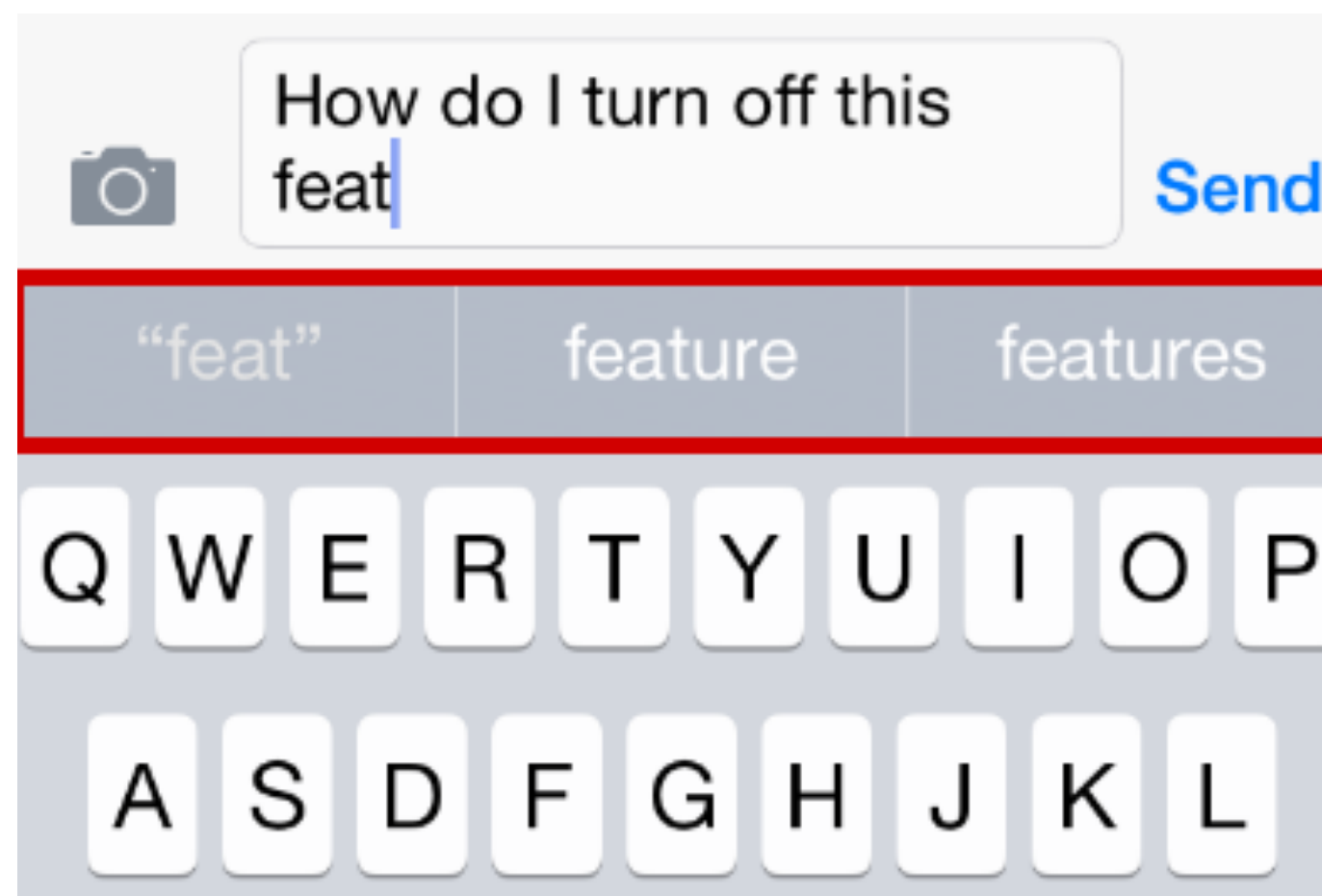
Unsupervised word embeddings



Tshitoyan, V.; *et al.*. *Nature* **2019**, 571 (7763), 95.
Perspective: Isayev, O. *Nature* **2019**, 571 (7763), 42–43.

Unsupervised word embeddings

single layer NN is trained to predict all context words for the given target word.

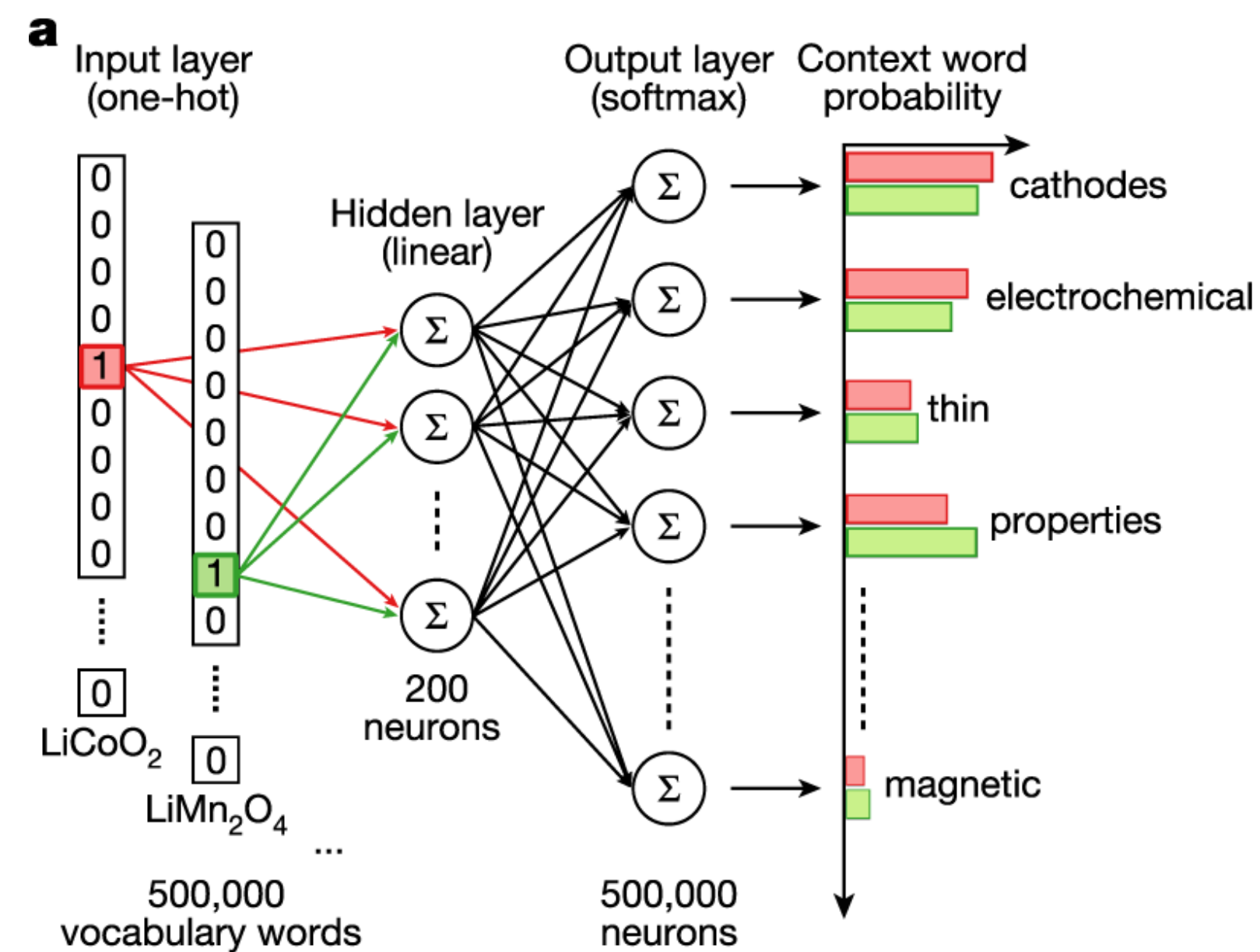
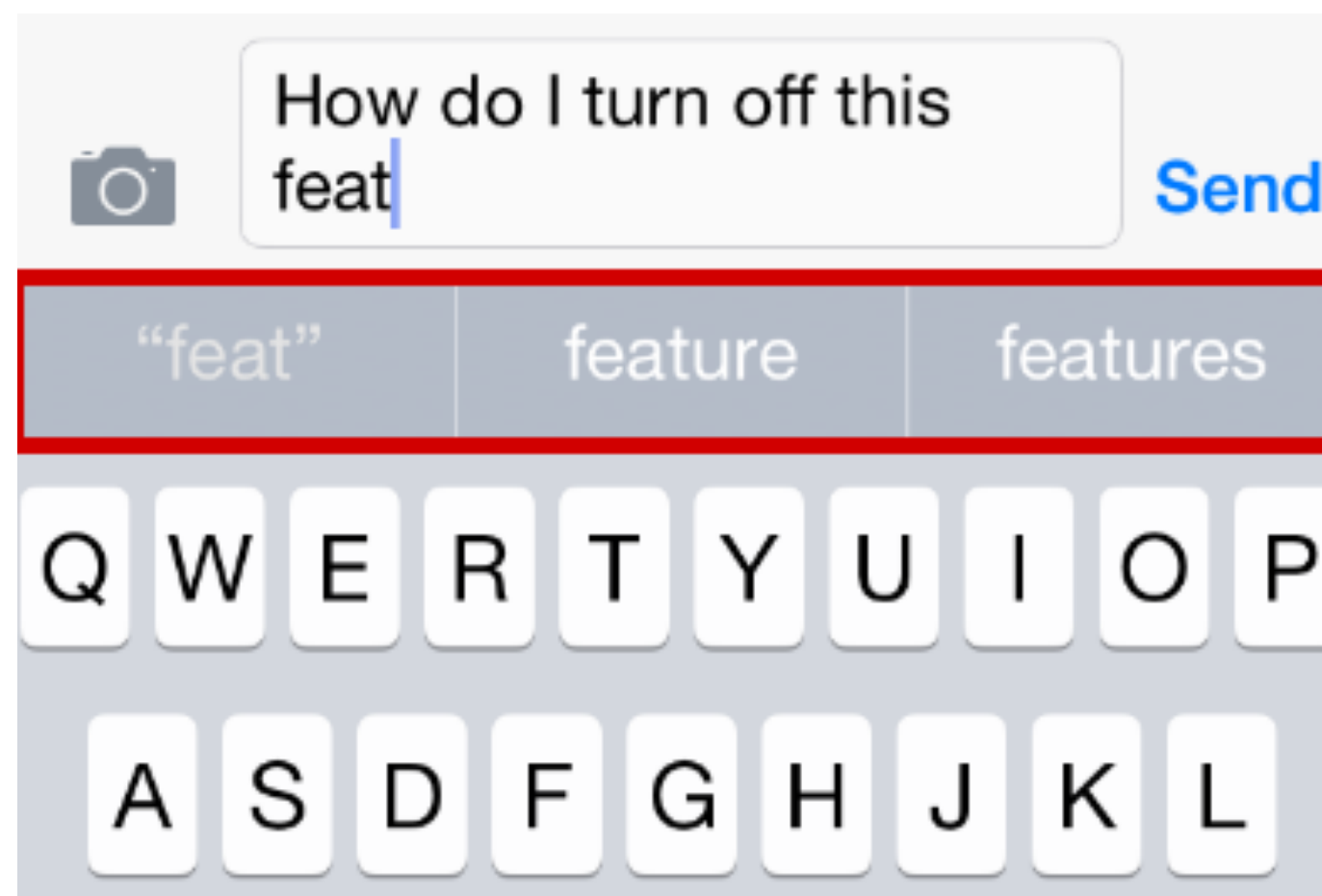


for similar words the context words are the same

Unsupervised word embeddings

ferromagnetic-NiFe + IrMn $\approx$ antiferromagnetic

single layer NN is trained to predict all context words for the given target word.



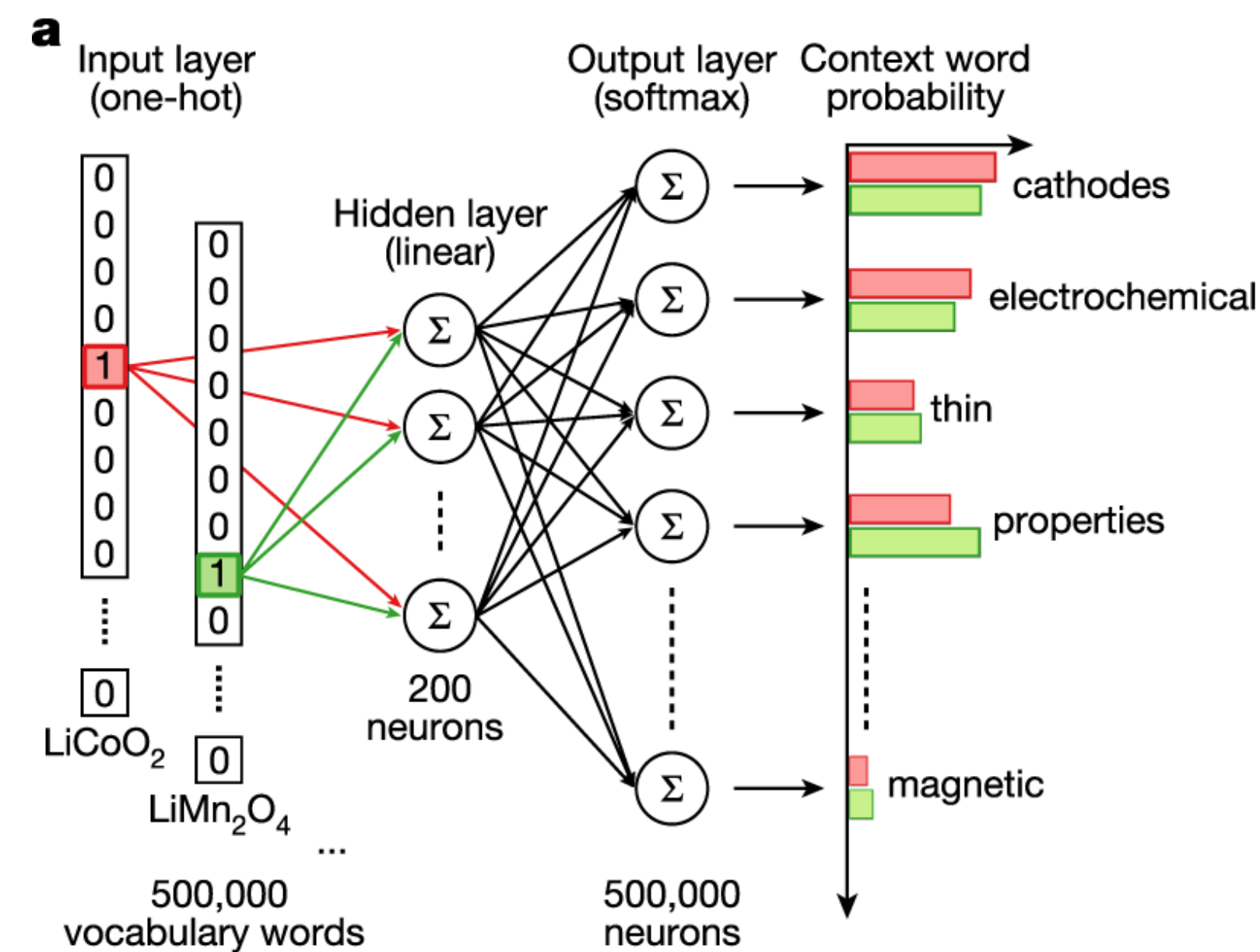
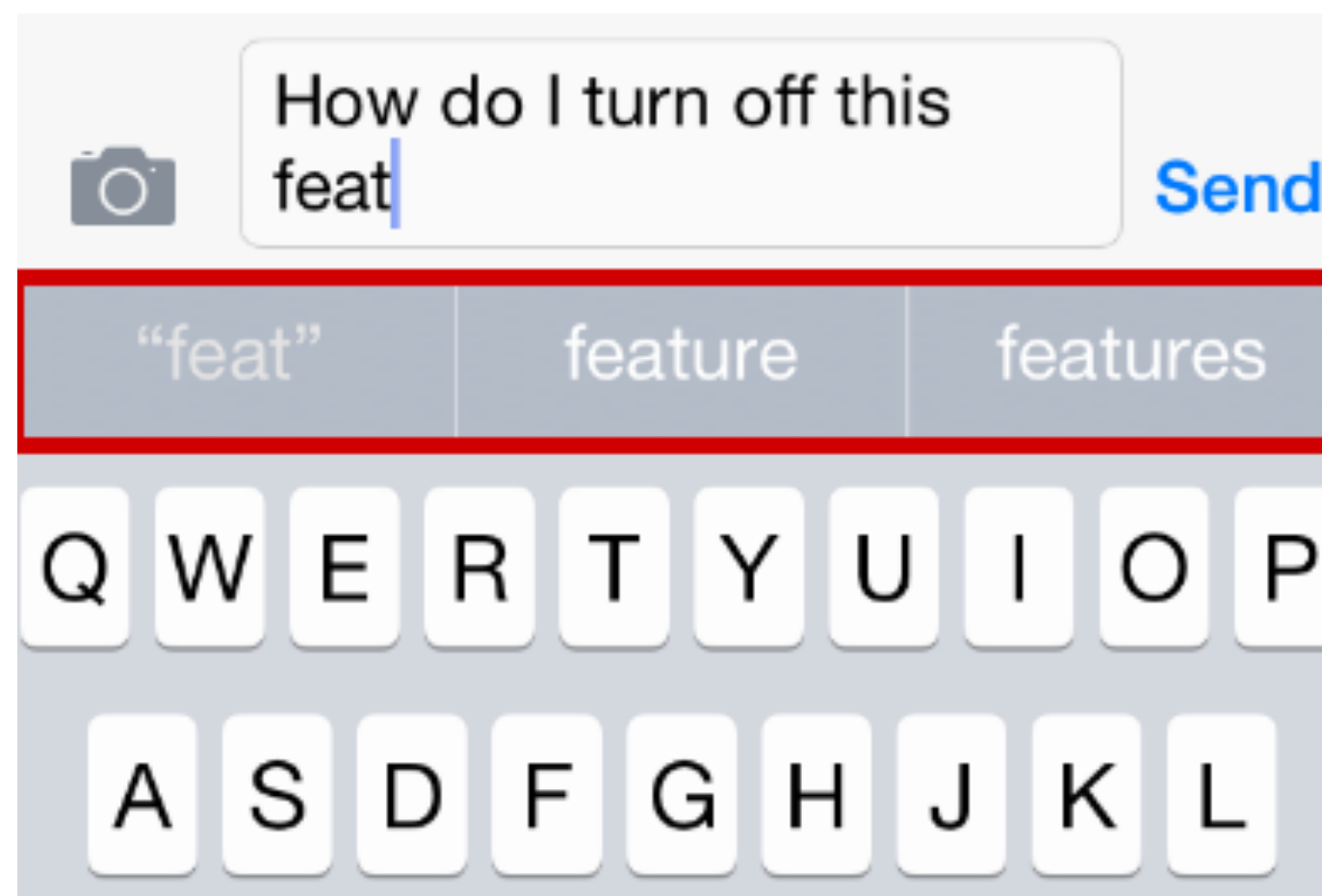
for similar words the context words are the same

Unsupervised word embeddings

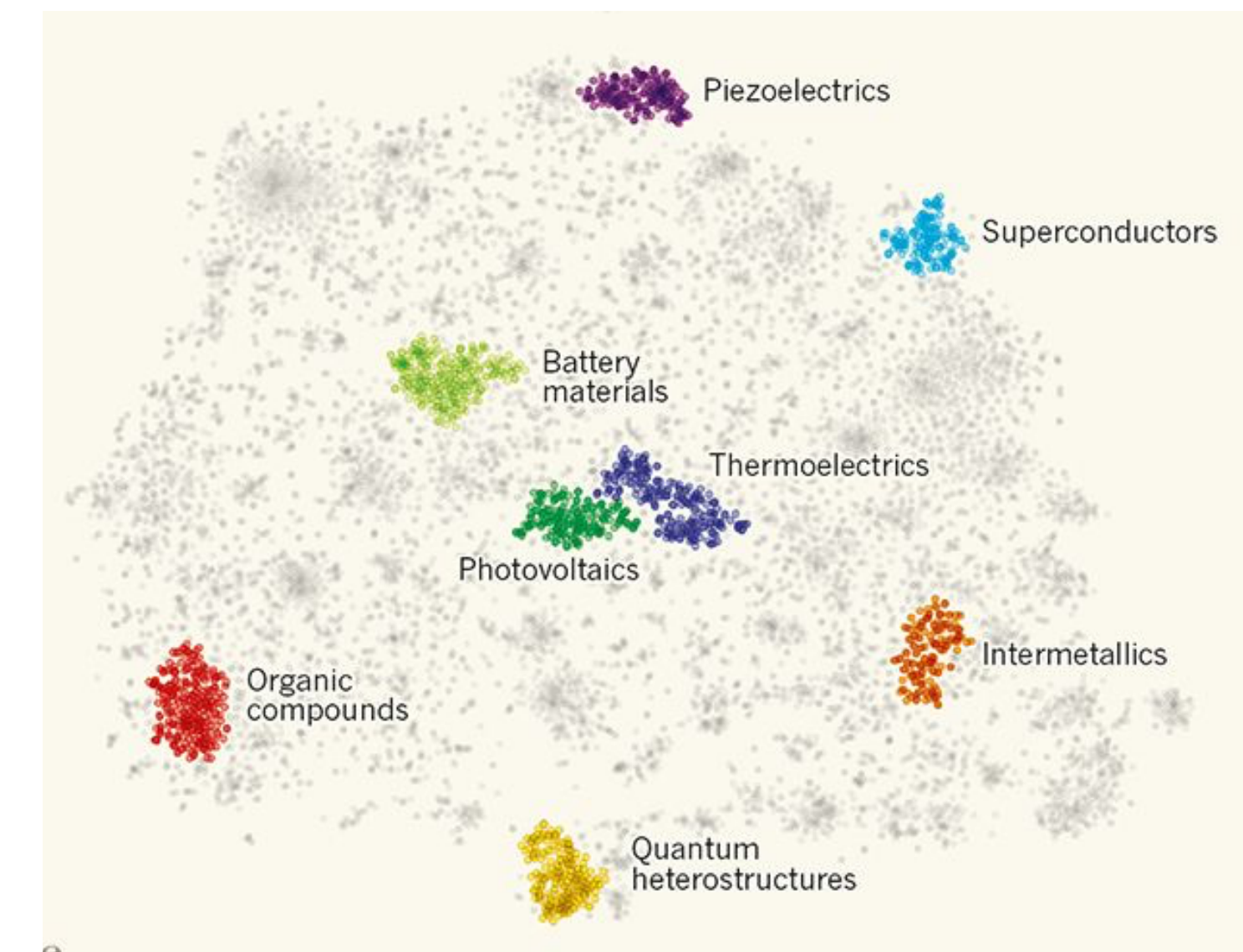
ferromagnetic-NiFe + IrMn $\approx$ antiferromagnetic

single layer NN is trained to predict all context words for the given target word.

Projection of embeddings onto two dimensions (t-SNE).



for similar words the context words are the same



Unsupervised word embeddings

Embeddings can also be used for predictions.

a

Cosine similarity to
'thermoelectric'

1.	Bi_2Te_3	✓
2.	MgAgSb	✓
3.	PbTe	✓
	...	✓
326.	Li_2CuSb	?
	...	✓
328.	In_4Te_3	✓
	...	✓
345.	$\text{Cu}_3\text{Nb}_2\text{O}_8$	?
	...	✓

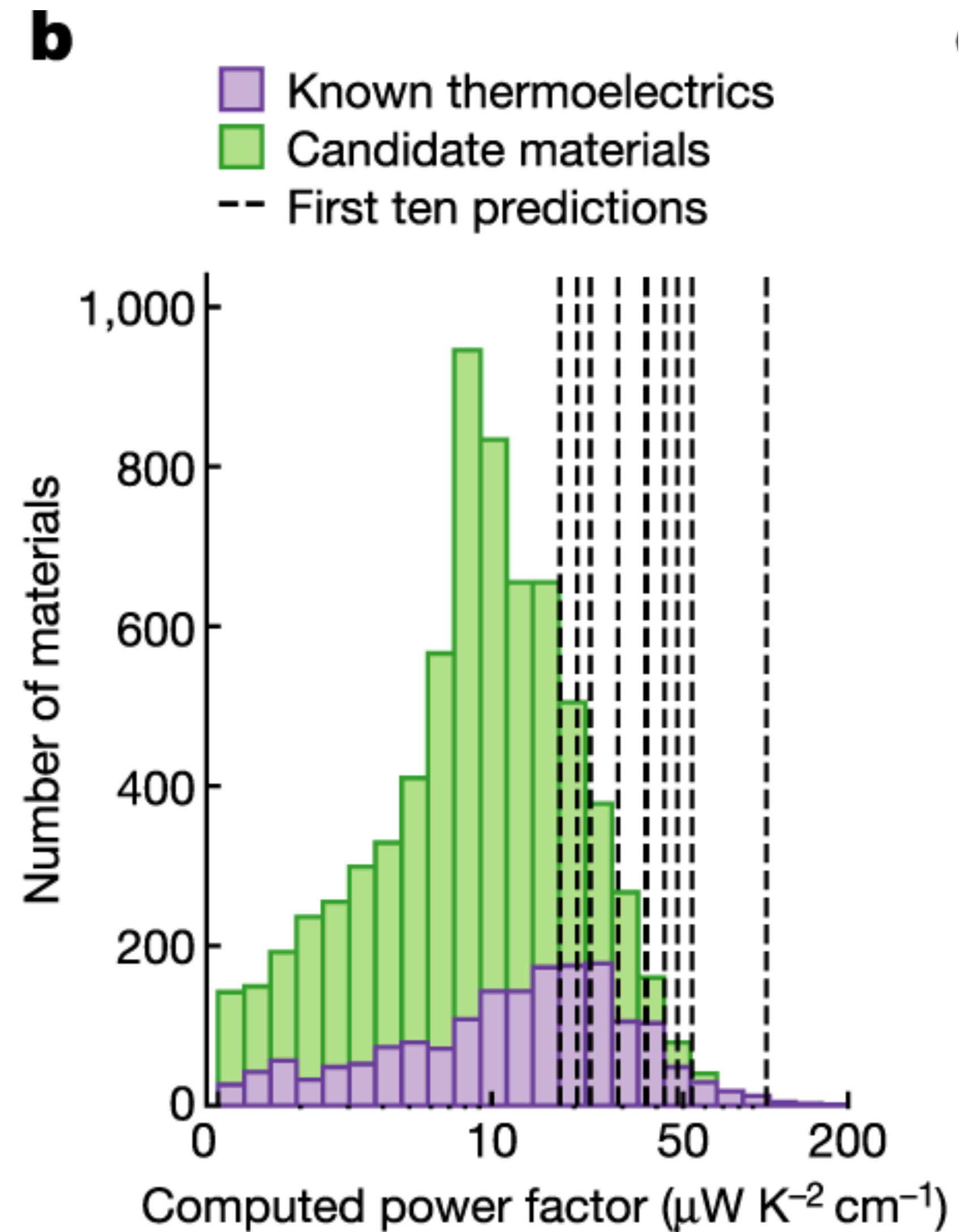
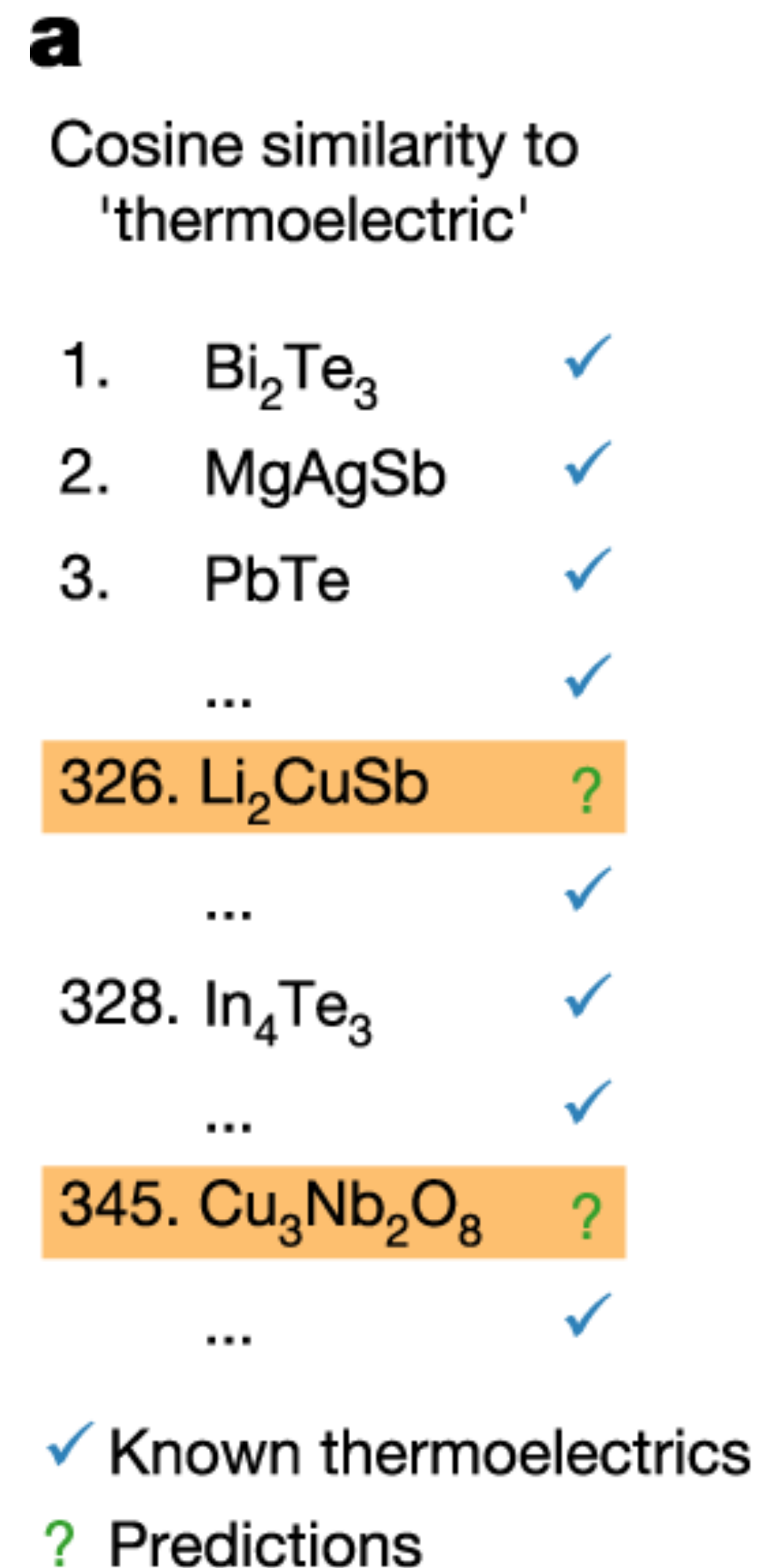
✓ Known thermoelectrics

? Predictions

Unsupervised word embeddings

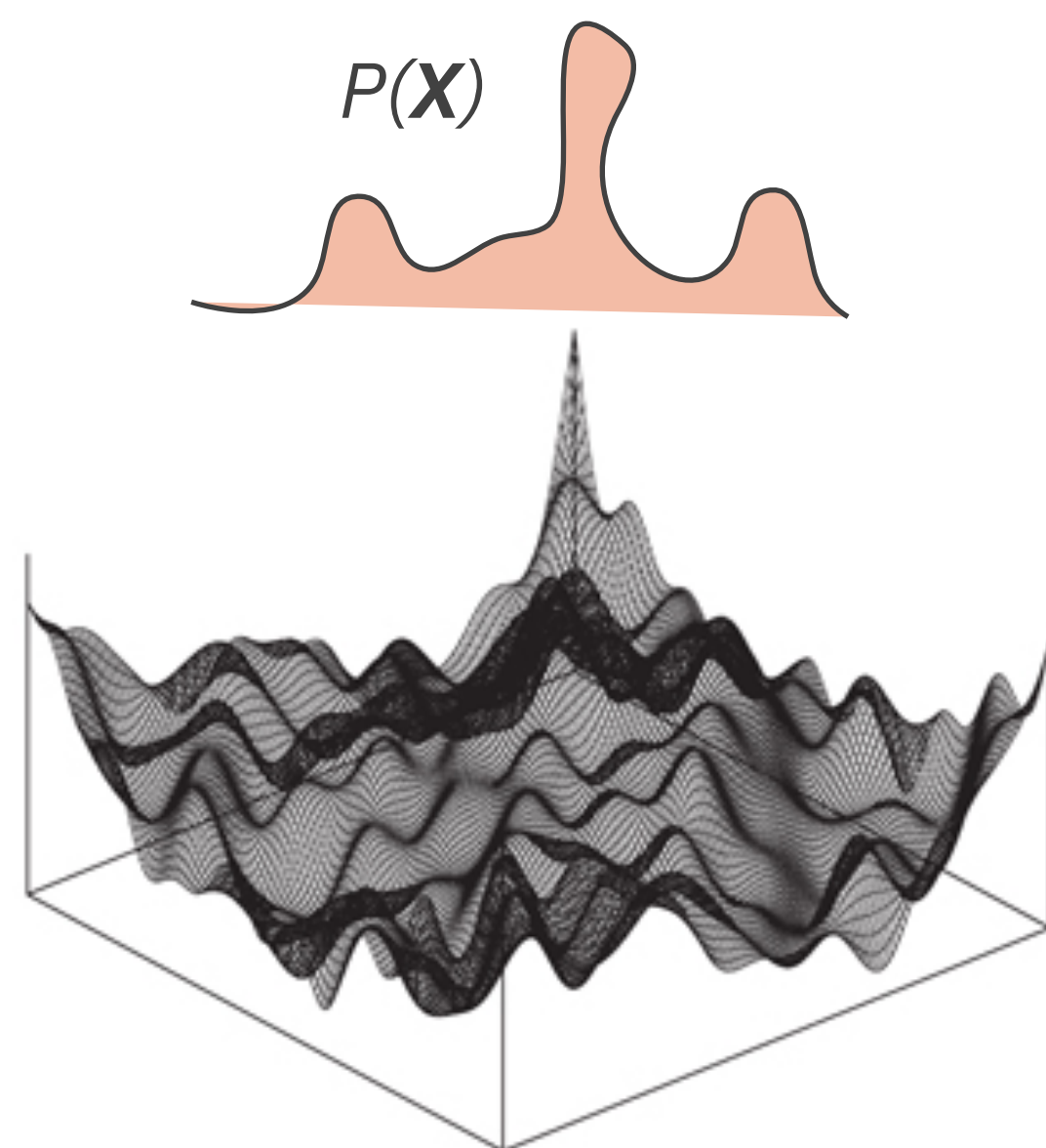
Embeddings can also be used for predictions.

- Top ten predictions even slightly higher than known average
- Better rank correlation with experiments than DFT



Boltzmann samplers

Molecular Simulations: $U(\mathbf{X})$ given, need approach to sample $P(\mathbf{X})$

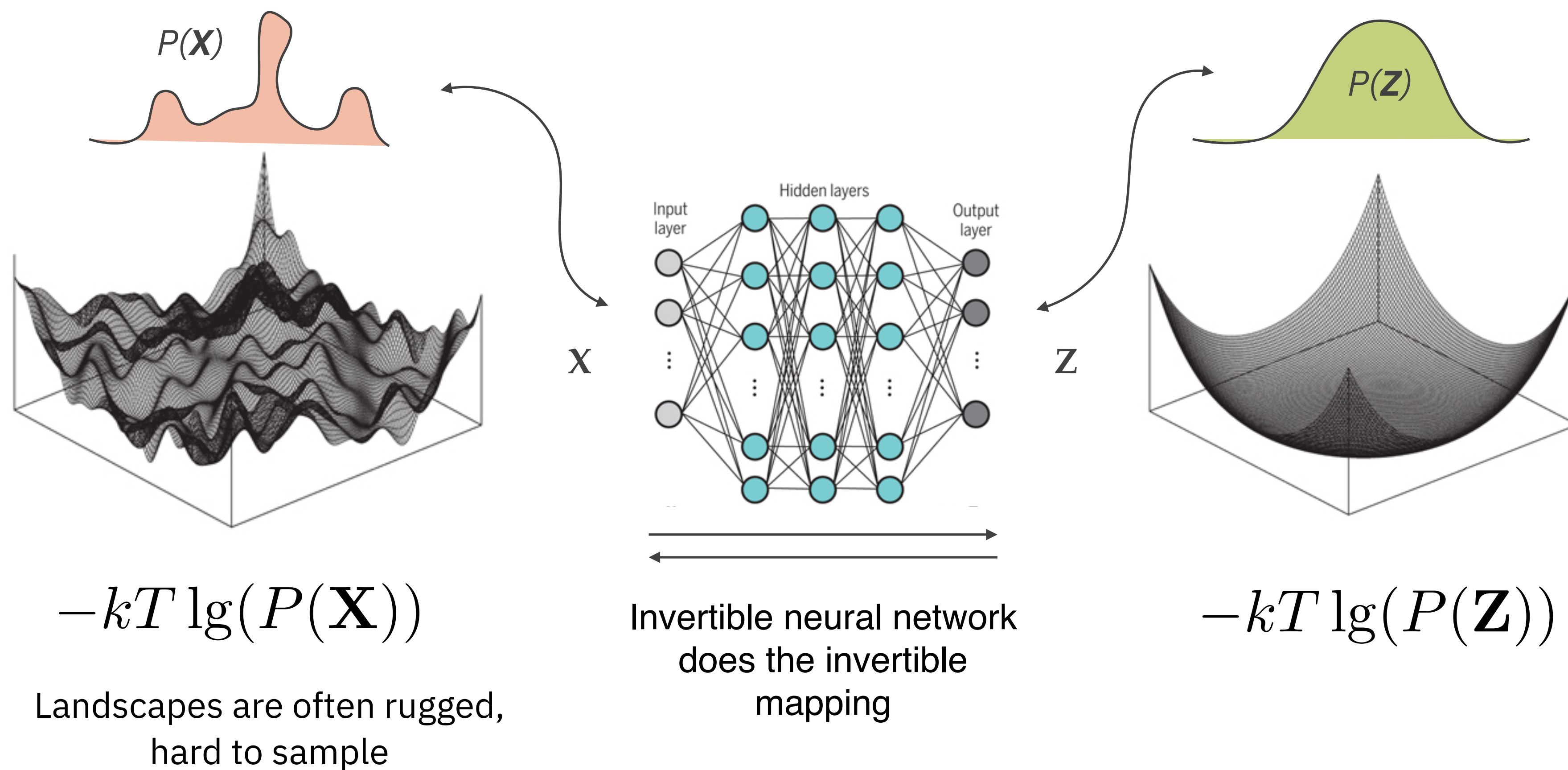


$$-kT \lg(P(\mathbf{X}))$$

Landscapes are often rugged,
hard to sample

Boltzmann samplers

Molecular Simulations: $U(\mathbf{X})$ given, need approach to sample $P(\mathbf{X})$



Applications

Learning Objectives:

- ML “force fields”
- ML for efficient sampling
- Using natural language processing for materials discovery