Machine learning for phylogenomics

Laurent Jacob Luc Blassel

November 6th 2024

Roadmap

- Neural density inference

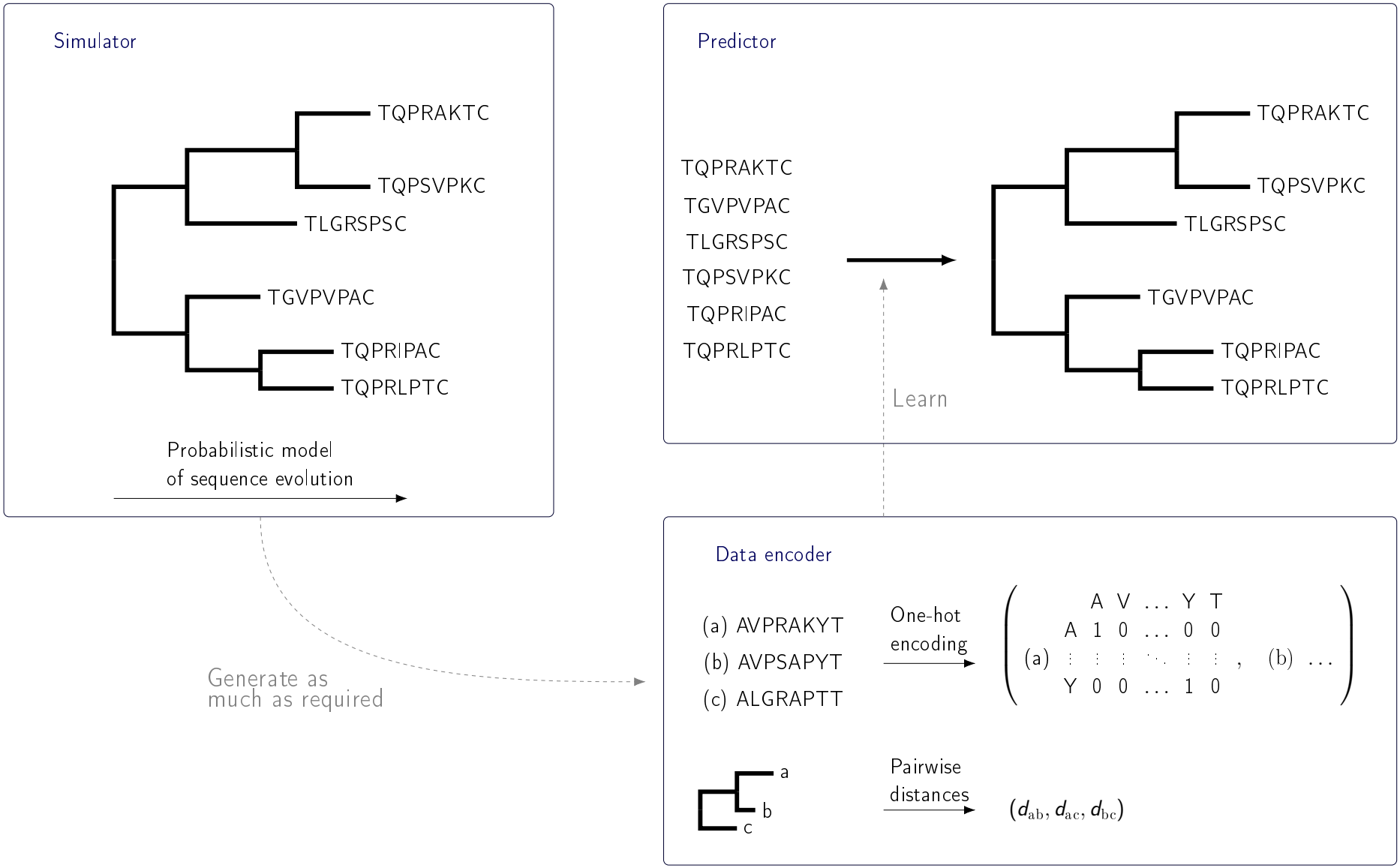
  Inference with supervised learning on simulations
- Machine learning

  Purpose, approximation vs estimation error
- Neural networks

  History, fundamentals
- Geometric deep learning

  Leveraging symmetries
- Phyloformer

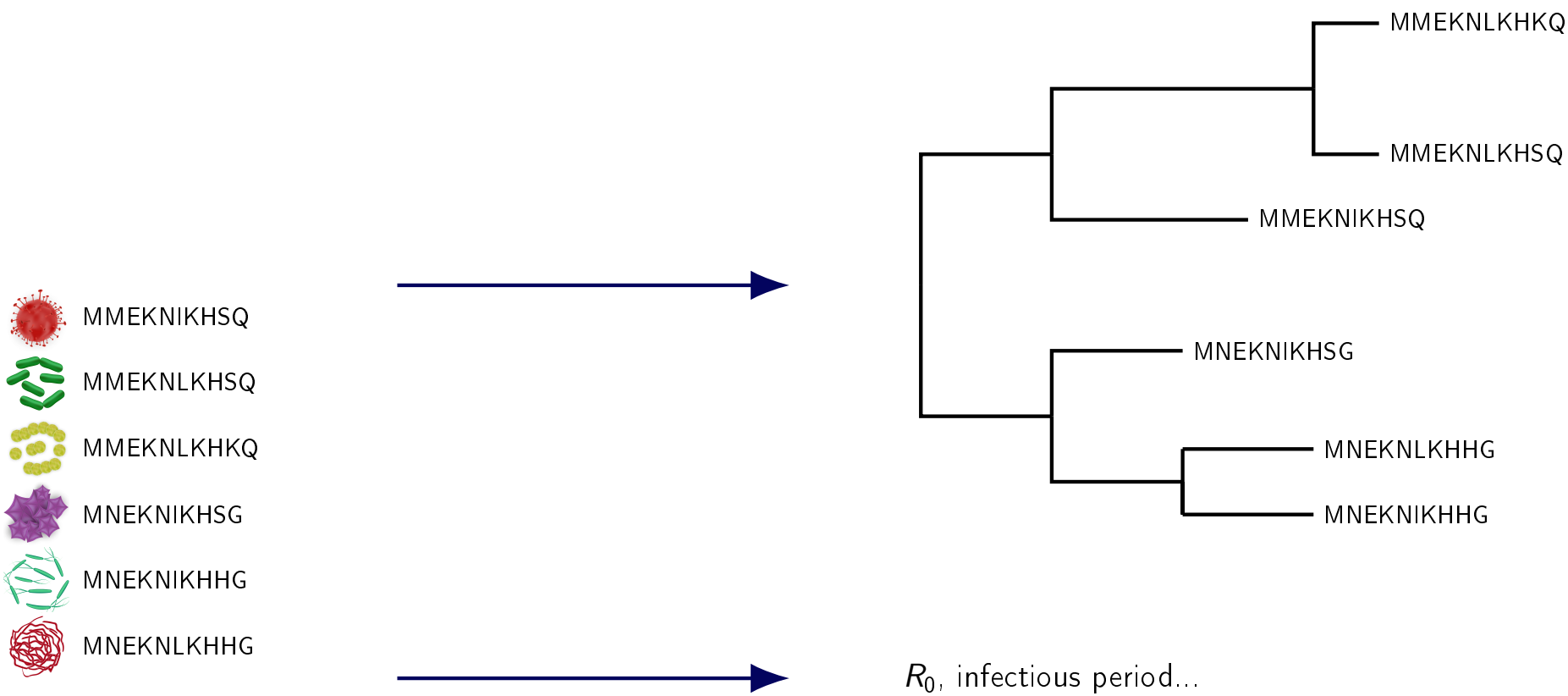
General idea



Part I

Neural density inference: a likelihood-free approach

Inference in evolutionary genomics



- **Observe** homologous sequences.
- **Infer** their evolutionary history: phylogeny, reproduction number...

Statistical inference

Model $p(\text{sequences}\|\text{tree})$		Point estimate $\widehat{\text{tree}}$
Observed sequences	$\rightarrow$	or
prior $p(\text{tree})$ (optional)		posterior $p(\text{tree}\|\text{sequences})$

Likelihood-based inference

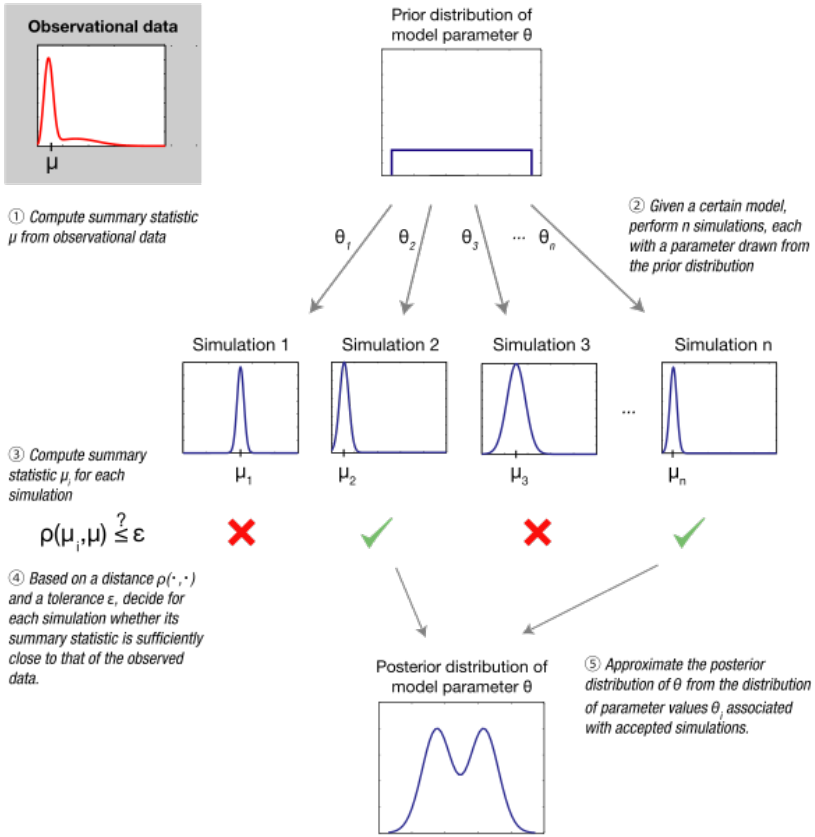
- Maximum likelihood: $\widehat{\text{tree}} = \arg\max_{\text{tree}} p(\text{sequences}|\text{tree})$.
- Estimate or sample from the posterior $p(\text{tree}|\text{sequences})$ (typically also involves computing $p(\text{sequences}|\text{tree})$).

Likelihood-**free** inference

- Realistic models for sequence evolution: computing $p(\text{sequences}|\text{tree})$ is expensive or impossible.
- But *sampling* from it can be cheap.

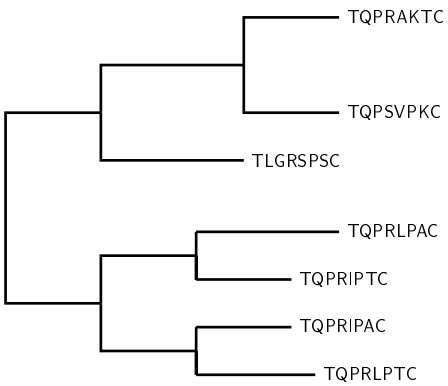
Likelihood-**free** inference

- Idea: perform inference by sampling, and not evaluating $p(\text{sequences}|\text{tree})$.
- Example: Approximate Bayesian Computation (ABC)

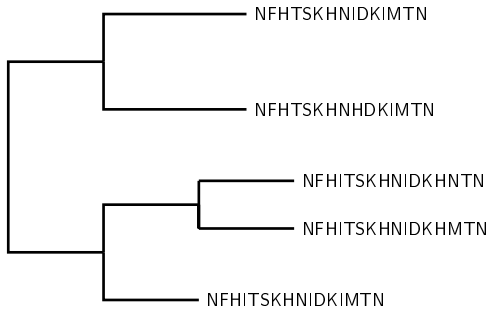


From Sunnåker *et al.* 2013

Neural inference

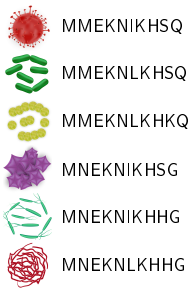


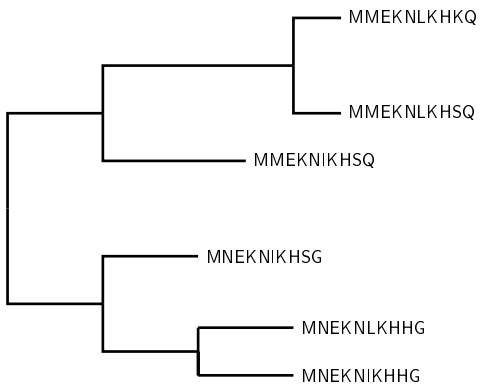
. . .



Simulate examples of:
Trees
Evolved sequences

Learn



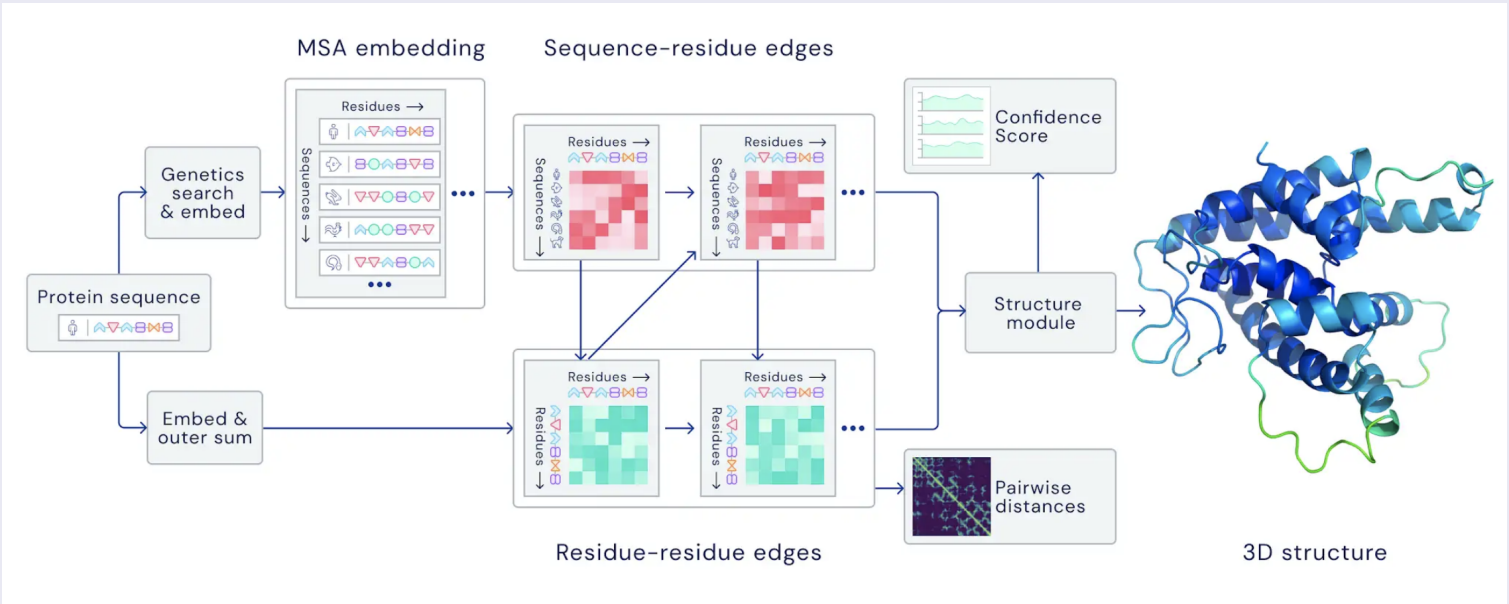


Compared to ABC:

- No rejection.
- No summary statistics.

Unusual setting for supervised learning

Ordinarily used for induction on real-world data



(adapted from Jumper *et al.*, 2021)

Common misconceptions

- Proxy "before we get real data"?
- "What if your model is off"?

From this point

What do we mean precisely by "learning" the sequence $\rightarrow$ tree function?
What difficulties does it entail?

- Machine learning

  Purpose, approximation vs estimation error
- Neural networks

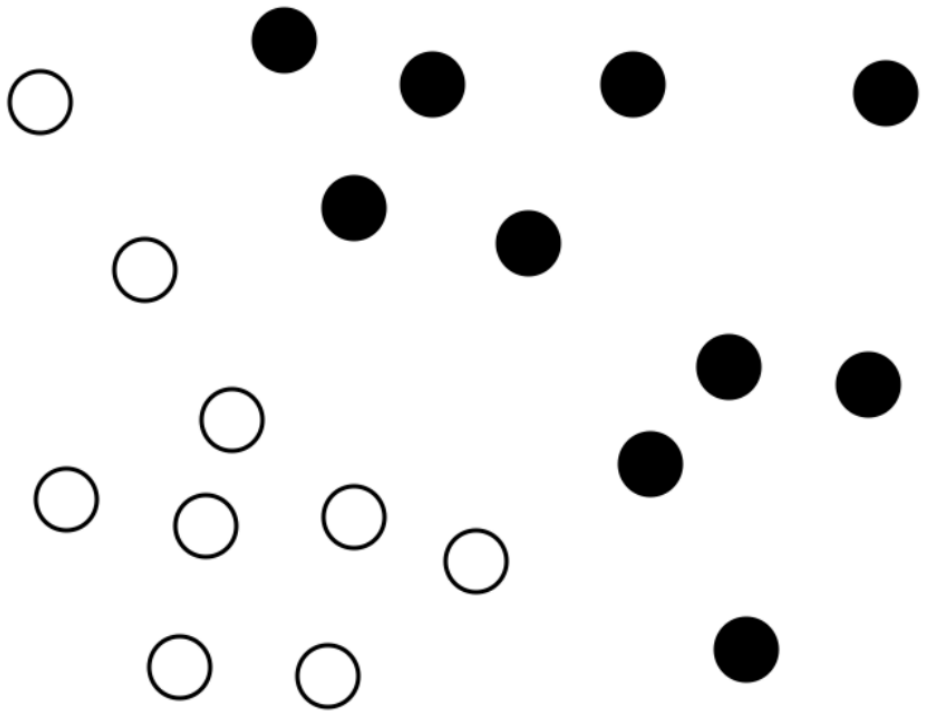
  History, fundamentals
- Geometric deep learning

  Leveraging symmetries
- Phyloformer

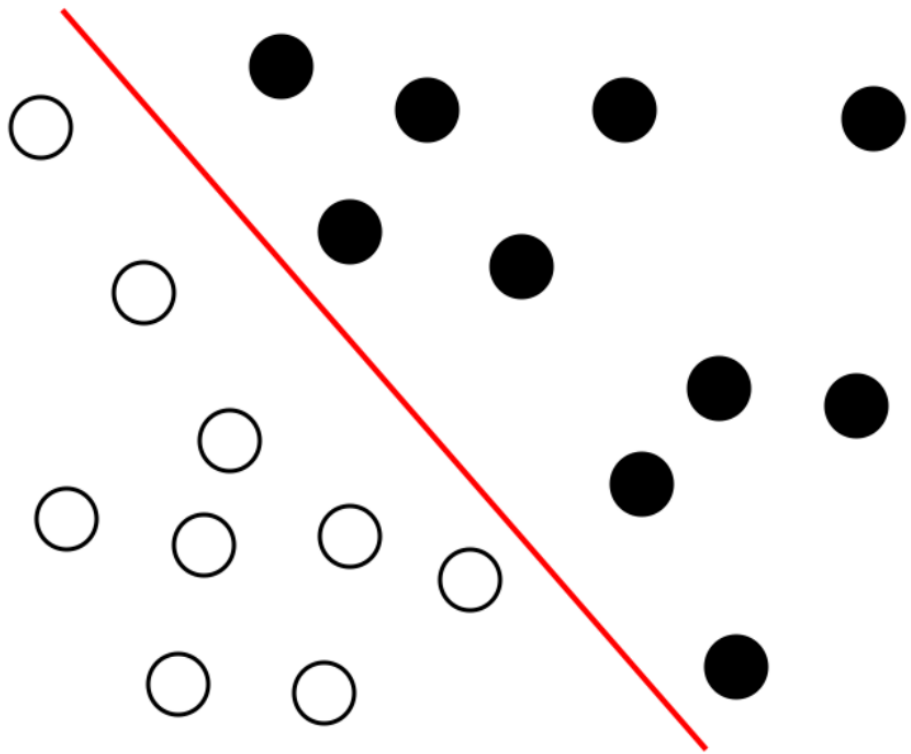
Part II

Machine learning: approximation vs estimation error

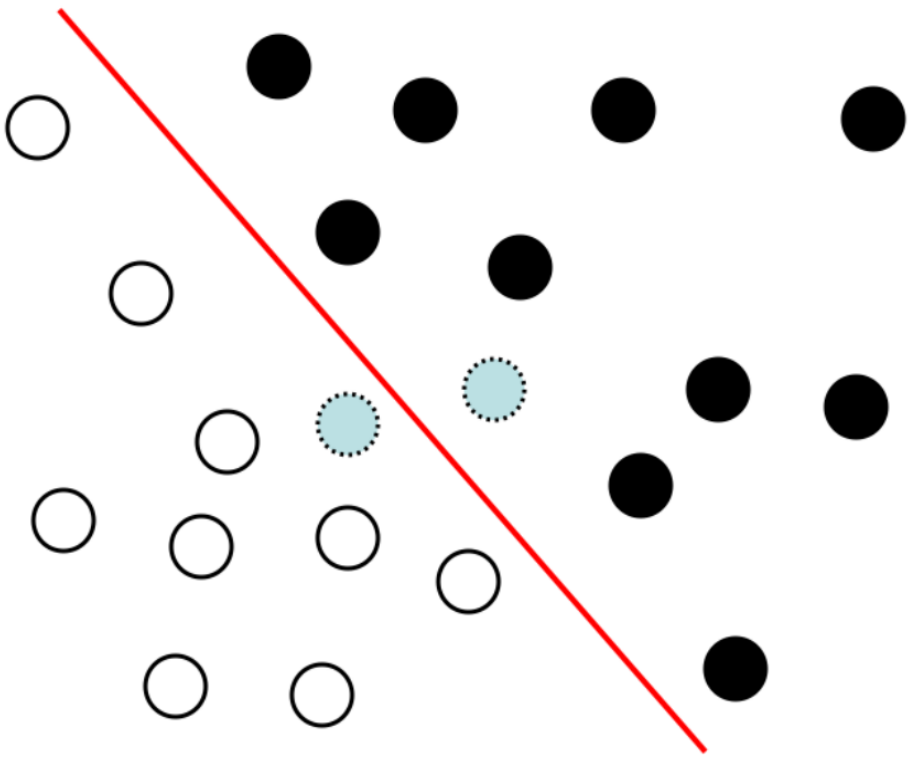
Learning from data



Learning from data

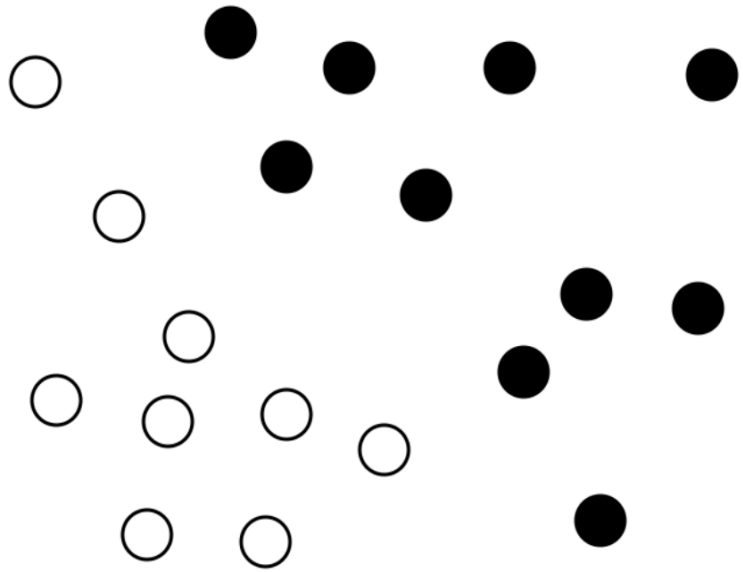


Learning from data



Linear regression

We observe samples in p dimensions:

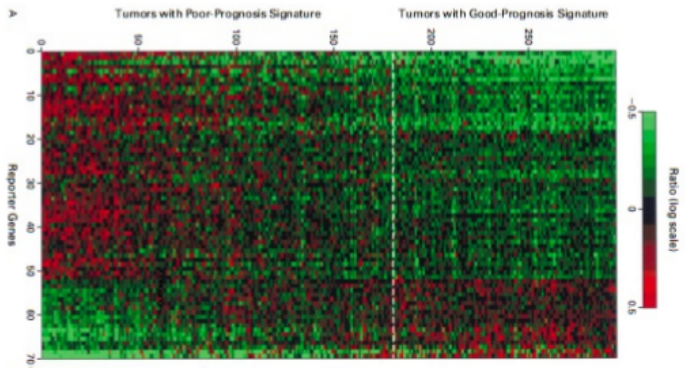


One sample = two coordinates ($p = 2$).

$x = (x_1, x_2)$

Linear regression

We observe samples in p dimensions:

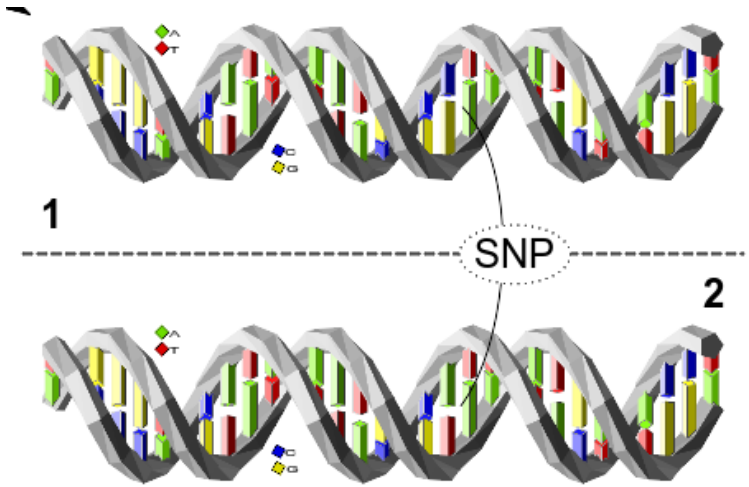


One sample = the expression of $40,000$ genes ($p = 40,000$), or here a subset of 70.

$x = (\text{expr}_1, \ldots, \text{expr}_{40,000})$

Linear regression

We observe samples in p dimensions:

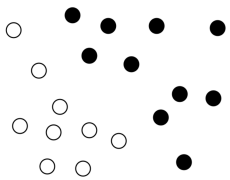


One sample = presence/absence of one among $1,000,000$ mutations in the genome ($p = 1,000,000$).

$x = (\text{SNP}_1, \ldots\ldots\ldots\ldots\ldots\ldots, \text{SNP}_{1,000,000})$

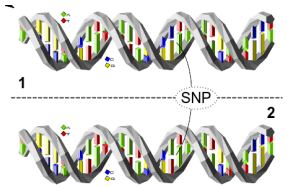
Linear regression

We observe n such samples and store them in an $X \in \mathbb{R}^{n,p}$ matrix



$$X_{\text{dots}} = \begin{pmatrix} x_1^1 & x_2^1 \\ \vdots & \vdots \\ \vdots & \vdots \\ x_1^{19} & x_2^{19} \end{pmatrix}$$

$n = 19$

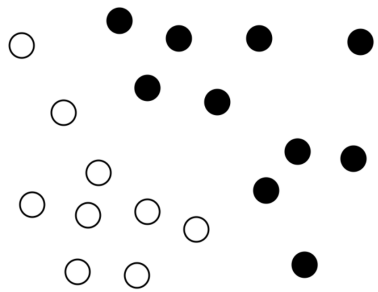


$$X_{\text{SNPs}} = \begin{pmatrix} \text{SNP}_1^1 & \dots & \text{SNP}_{100,000,000}^1 \\ \vdots & & \vdots \\ \text{SNP}_1^{200} & \dots & \text{SNP}_{1,000,000}^{200} \end{pmatrix}$$

$n = 200$

Linear regression

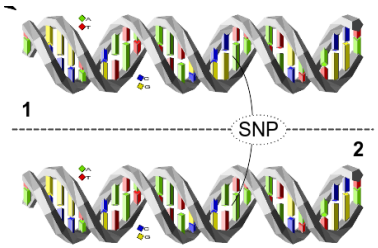
We also observe labels stored in a $Y \in \mathbb{R}^n$ vector.



$$X_{\text{dots}} = \begin{pmatrix} x_1^1 & x_2^1 \\ \vdots & \vdots \\ \vdots & \vdots \\ x_1^{19} & x_2^{19} \end{pmatrix} \quad Y_{\text{dots}} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Linear regression

We also observe labels stored in a $Y \in \mathbb{R}^n$ vector.



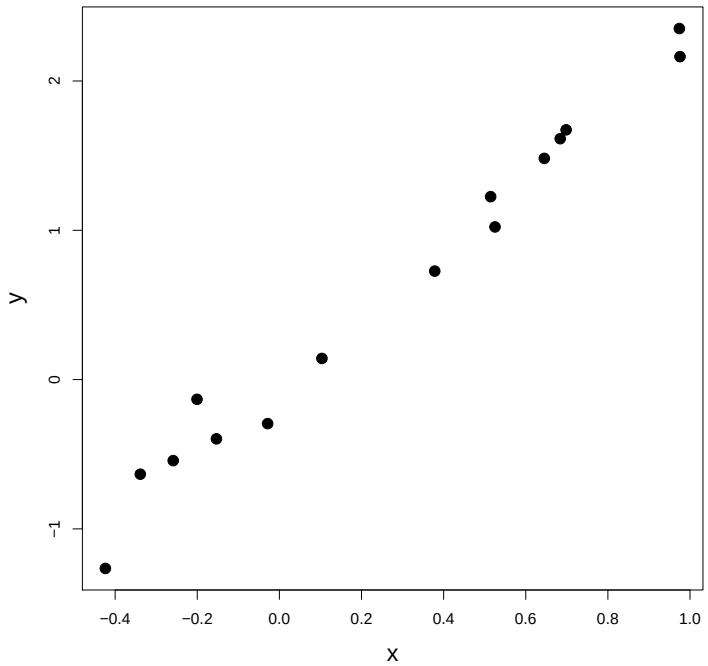
$$X_{\text{SNPs}} = \begin{pmatrix} \text{SNP}_1^1 & \dots & \text{SNP}_{100,000,000}^1 \\ \vdots & & \vdots \\ \text{SNP}_1^{200} & \dots & \text{SNP}_{1,000,000}^{200} \end{pmatrix} \quad Y_{\text{SNPs}} = \begin{pmatrix} \text{sick} \\ \vdots \\ \text{healthy} \end{pmatrix}$$

Linear regression

Design a linear function which takes a sample as input and predicts its label:

$$\theta^\top x = \theta_1 x_1 + ... + \theta_p x_p \simeq y,$$

for any sample $x = (x_1, \ldots, x_p)$.



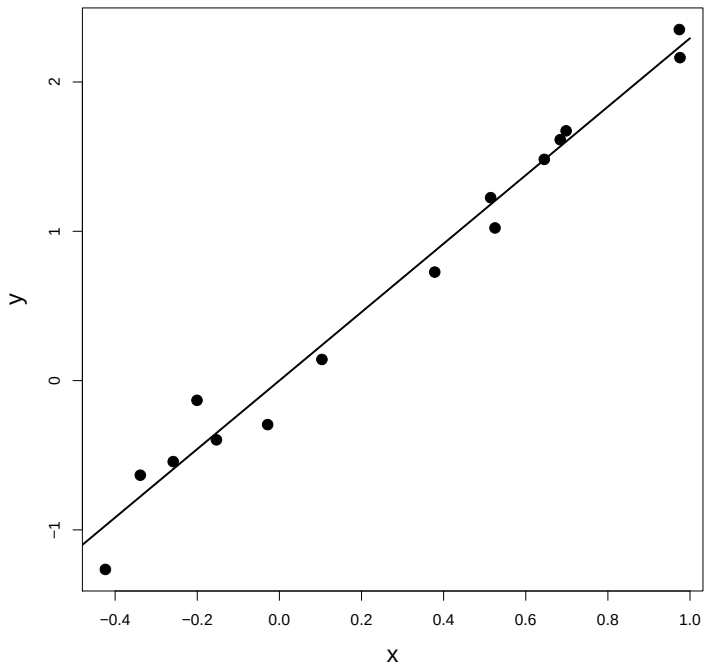
For $p = 1$, θ is just the slope of the line ($y \simeq \theta \times x$).

Linear regression

Design a linear function which takes a sample as input and predicts its label:

$$\theta^\top x = \theta_1 x_1 + ... + \theta_p x_p \simeq y,$$

for any sample $x = (x_1, \ldots, x_p)$.



For $p = 1$, θ is just the slope of the line ($y \simeq \theta \times x$).

Linear regression

- Design a linear function of a sample which predicts its label:

$$\theta^\top x = \theta_1 x_1 + \ldots + \theta_p x_p \simeq y,$$

  for any sample $x = (x_1, \ldots, x_p)$.
- Popular option: choose weights θ_j which make $(\theta^\top x - y)^2$ as small as possible across samples:

$$\hat{\theta} = \arg\min_\theta \sum_{i=1}^{n} (y_i - \theta^\top x_i)^2 = \arg\min_\theta \| Y - X\theta \|^2.$$

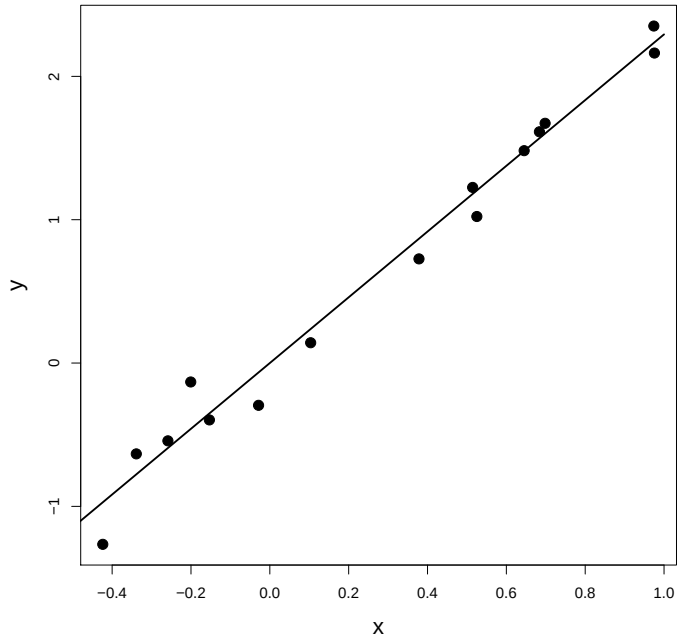
- If $X^\top X$ is invertible, closed form for $\hat{\theta}$:

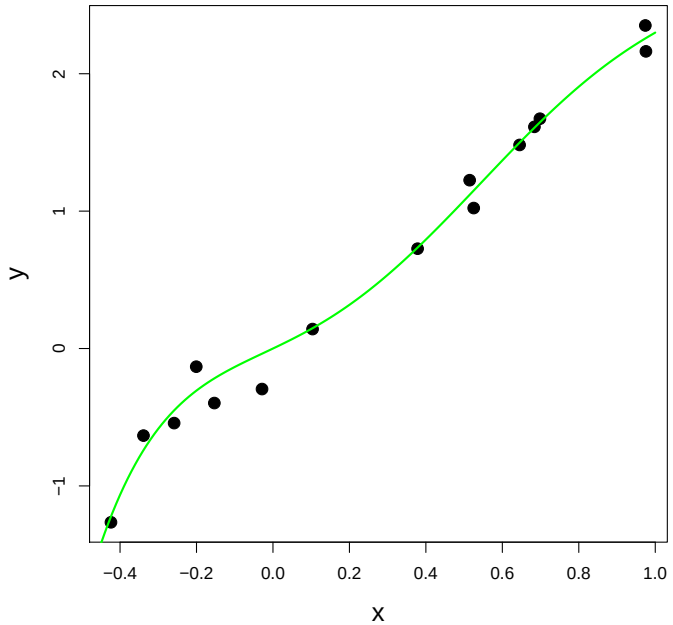
$$\hat{\theta} = (X^\top X)^{-1} X^\top Y.$$

- Can be used to learn polynomial functions, how?

Using linear regression to learn non-linear functions

A polynomial function in **one dimension** can be expressed as a linear function in d **dimensions**, using $\tilde{x} = (x, x^2, \ldots, x^d)$:





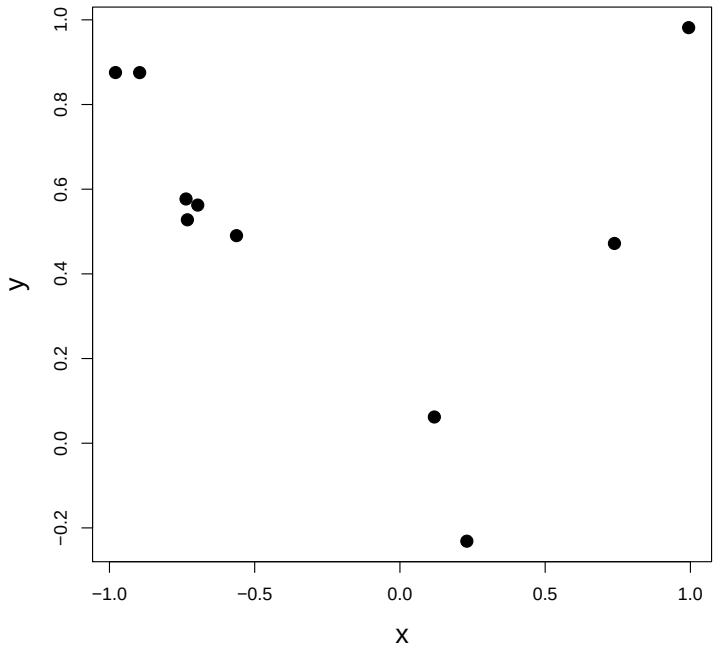
$y \simeq \theta x$

$p = 1$: θ is the slope of the line.

$y \simeq \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 = \theta^\top \tilde{x}$

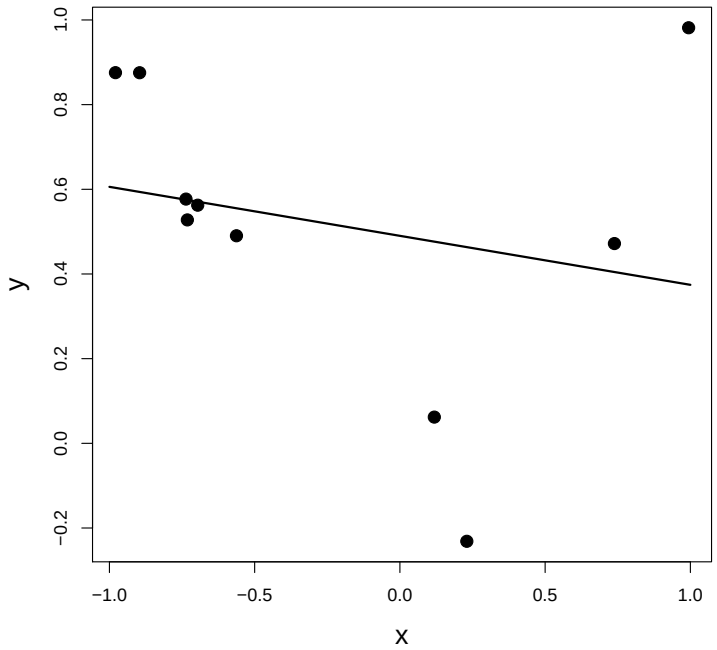
$p = 5$: 1D polynomial or 5D linear function.

Approximation vs estimation: intuition



- We observe 10 couples (x_i, y_i).
- We want to estimate y from x.
- Strategy: find f such that $f(x_i)$ is close to y_i.

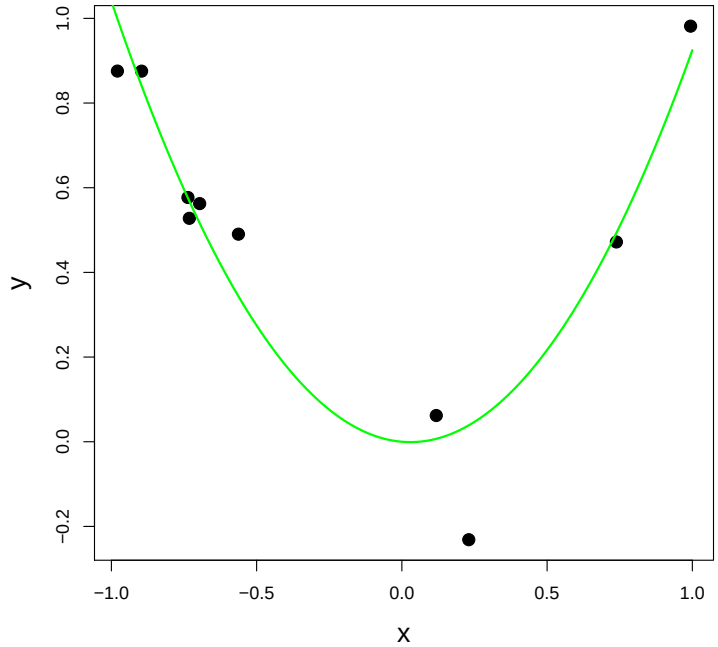
Approximation vs estimation: intuition



Find f as a line

$$\min_{f(x)=ax+b} \|Y - f(X)\|^2$$

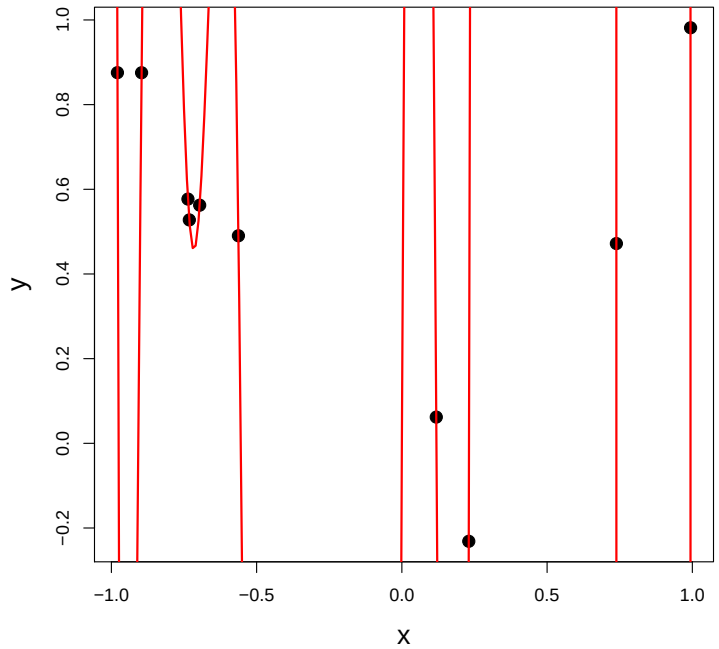
Approximation vs estimation: intuition



Find f as a quadratic function

$$\min_{f(x)=ax+bx^2} \|Y - f(X)\|^2$$

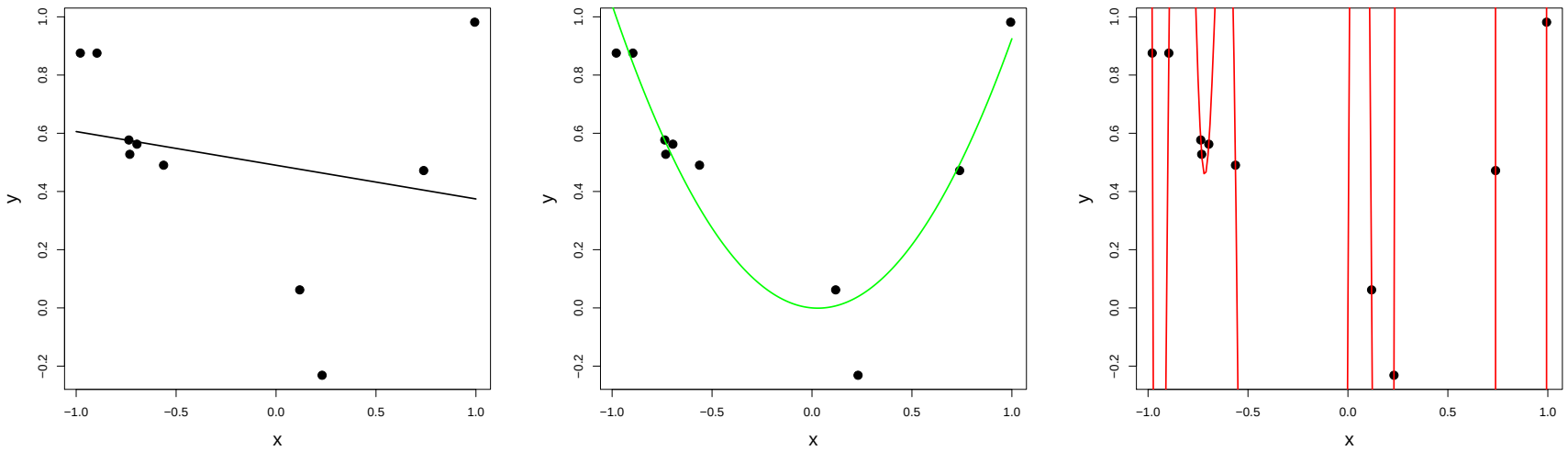
Approximation vs estimation: intuition



Find f as a polynomial of degree 10

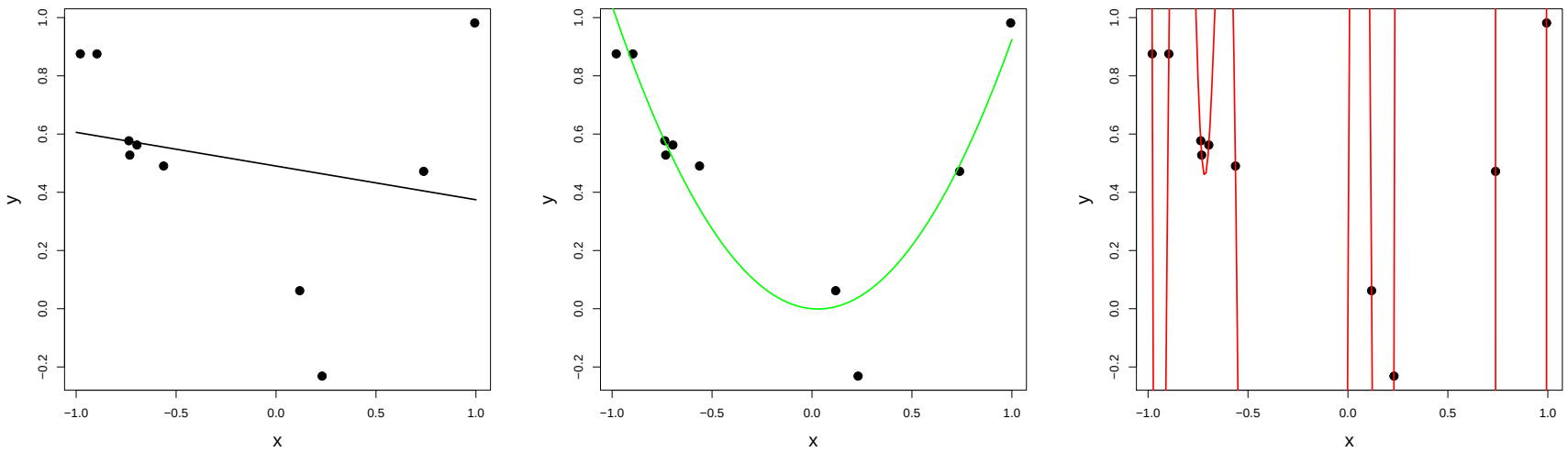
$$\min_{f(x)=\sum_{j=1}^{10} a_j x^j} \|Y - f(X)\|^2$$

Approximation vs estimation: intuition



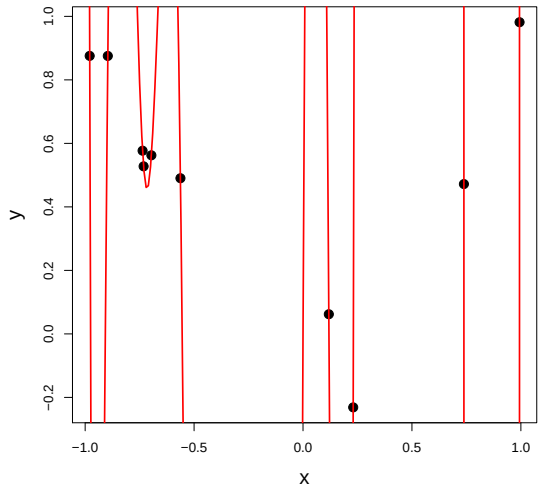
Which function would you trust to predict y corresponding to $x = 0.5$?

Approximation vs estimation: intuition



- Reminder: we aim at "finding f such that $f(x_i)$ is close to y_i".
- With the polynomial of degree 10, $f(x_i) - y_i = 0$ for all 10 points.
- There is something wrong with our objective.

Approximation vs estimation: intuition

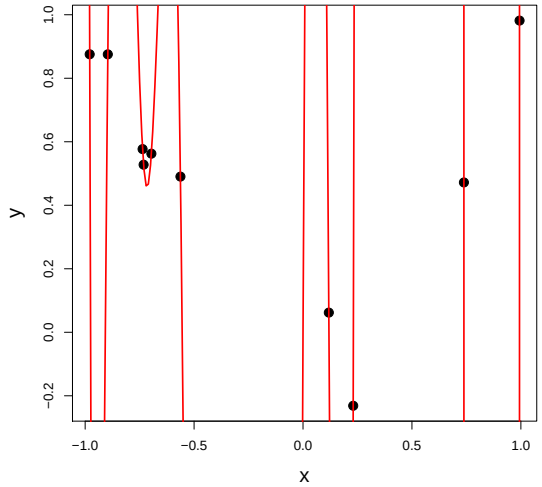


More precisely:

- If we allow any function f, we can find **a lot** of perfect solutions.
- Our actual goal is to estimate y for new points x from the same population :

$$\min_f \mathbb{E}_{(X,Y)} \| Y - f(X) \|^2$$

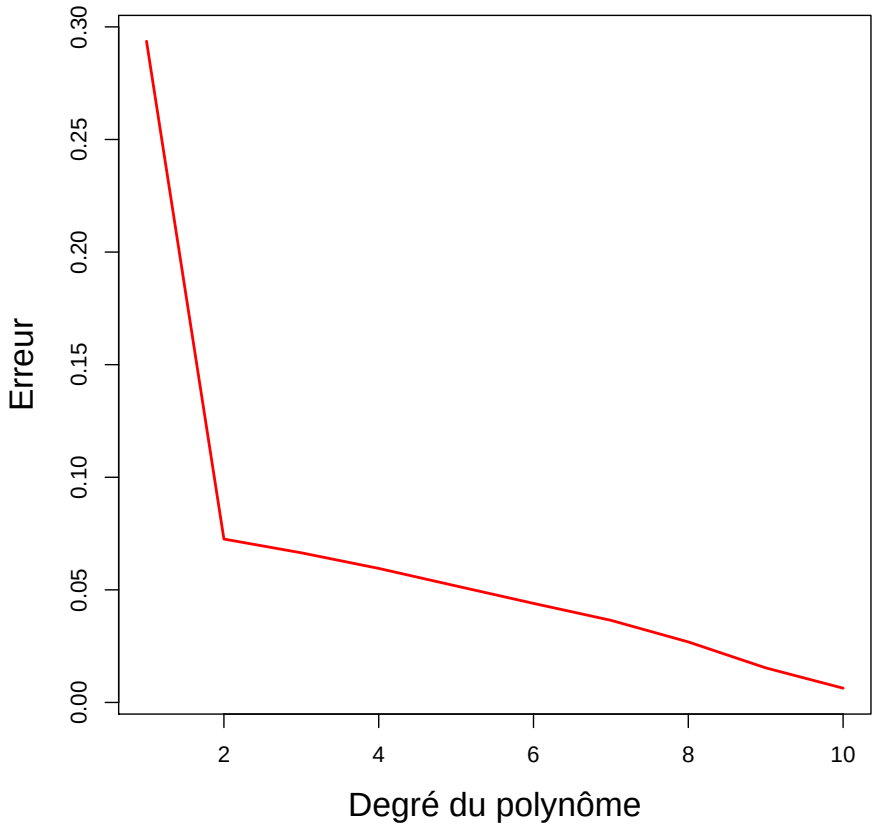
Approximation vs estimation: intuition



Even more precisely :

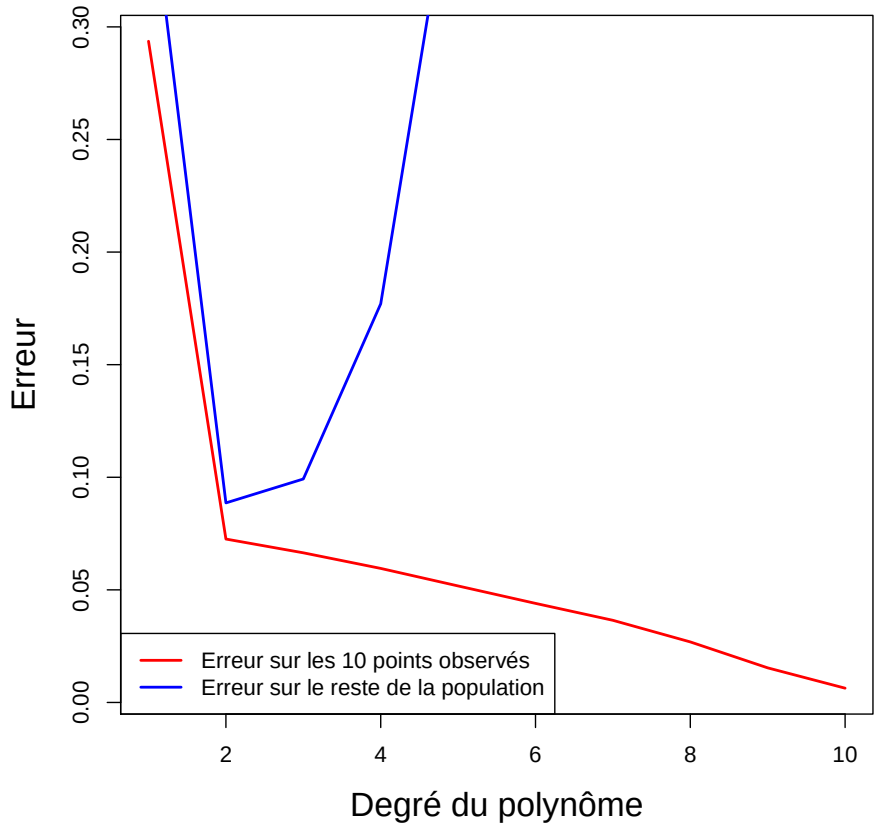
- We did not take into account the fact that our 10 points are a subsample from the population.
- If we sample 10 new points from the same population, the complex functions are likely to change more than the simple ones.
- Consequence: these functions will probably generalize less well to the rest of the population.

Overfitting



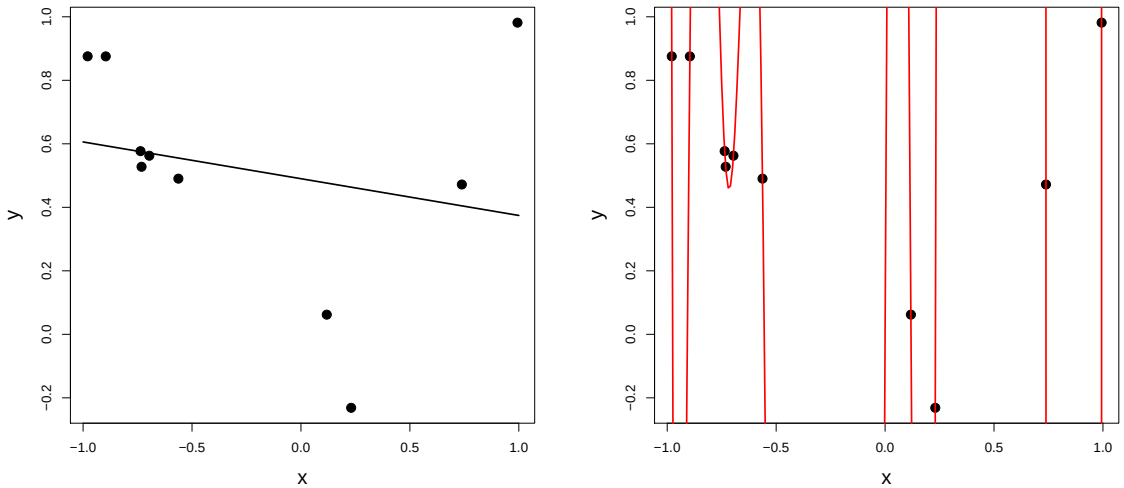
- When the degree increases, the error $\|y - f(x)\|^2$ over the 10 observations always decreases.
- Over the rest of the population, the error decreases, **then increases**.

Overfitting



- When the degree increases, the error $\|y - f(x)\|^2$ over the 10 observations always decreases.
- Over the rest of the population, the error decreases, **then increases**.

Overfitting



This suggests the existence of a **tradeoff** between two types of errors:

Approximation Sets of functions which are too simple cannot contain functions which explain the data well enough.

Estimation Sets of functions which are too rich may contain functions which are too specific to the observed sample.

Note: the latter is often less intuitive.

Dealing with the trade-off

Antagonistic errors...

... but some choices will lead to better trade-offs.

Structural risk minimization (Vapnik and Chervonenkis, 1974)

1. Define families of functions of increasing complexity

   Ability to increase/control complexity?

2. Minimize the empirical risk over each family

   Ability to optimize over the family?

3. Retain best generalization performance

   Ability to estimate this generalization?

We will see how deep learning deals with 1–2.
Before that, a few words on 3.

Are we back where we started?

1. Define nested function sets of increasing complexity.
2. Minimize the empirical risk over each family.
3. **Choose the solution giving the best generalization performances.**

- Best generalization means lowest (population) risk

$$R(f) = \int_{\mathcal{X}\times\mathcal{Y}} L(y, f(x)) d\mathbb{P} = \mathrm{E}[L(y, f(x))],$$

where L is some loss function quantifying the error.

- But the very reason we need all this is that we don't have access to R!
- We need to estimate it as well.

Validation/Hold out procedure

- Split available data into training and test sets.

Validation/Hold out procedure

- Split available data into training and test sets.

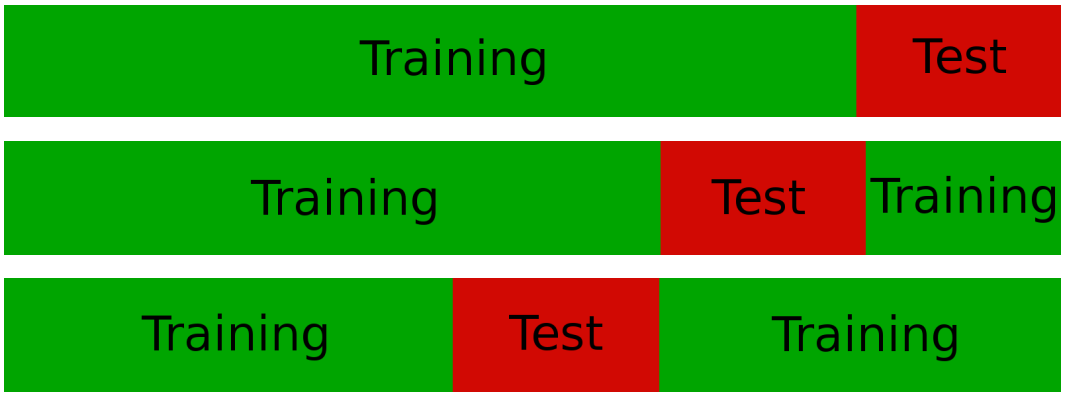
- Formally:

$$\hat{R}^{\mathrm{HO}}\left(\hat{f}; D_n; I^{(t)}\right) = \frac{1}{n_v} \sum_{i \in D_n^{(v)}} L\left(y_i, \hat{f}_{D_n^{(t)}}(x_i)\right).$$

- D_n: full set of n available data points. $I^{(t)}$: subset of indices used for training. $D_n^{(t)}$ (resp. $D_n^{(v)}$): set of data points restricted to training indices (resp. its complement).
- $\hat{f}$ denotes the learning algorithm whose risk we want to estimate. $\hat{f}_{D_n^{(t)}}$ is the function learnt by applying this algorithm to training data $D_n^{(t)}$.

Cross validation

- Idea: averaging several hold out estimators of the risk corresponding to different data splits:



...

- Formally:

$$\hat{R}^{\mathrm{CV}}\left(\hat{f}; D_n; \left(I_j^{(t)}\right)_{1\leq j\leq B}\right) = \frac{1}{B}\sum_{j=1}^{B} \hat{R}^{\mathrm{HO}}\left(\hat{f}; D_n; I_j^{(t)}\right),$$

where $I_1^{(t)}, \ldots, I_B^{(t)}$ are non-empty proper subsets of $\{1, \ldots, n\}$.

Cross validation procedures

- CV estimators differ in how they define $I_1^{(t)}, \ldots, I_B^{(t)}$.
- Most common: **V-fold CV**. Partition D_n into V sets of approximately equal cardinality $\frac{n}{V}$.
- **Leave-one-out** CV: V-fold with $V = n$.
- Monte-Carlo CV, leave-p-out CV...

Deep learning methods often use hold out (training is too costly).

Model selection vs assessment

- Validation was historically first used for **model assessment**: estimate the generalization error of a given algorithm $\hat{f}$.

Model selection vs assessment

- Validation was historically first used for **model assessment**: estimate the generalization error of a given algorithm $\hat{f}$.
- There are lots of methods to solve the same inference problem. Most of them have options/hyperparameters.

Model selection vs assessment

- Validation was historically first used for **model assessment**: estimate the generalization error of a given algorithm $\hat{f}$.
- There are lots of methods to solve the same inference problem. Most of them have options/hyperparameters.
- We also need a tool for **model selection**: choose best method or best class of hypothesis for a particular problem.

Model selection vs assessment

- Validation was historically first used for **model assessment**: estimate the generalization error of a given algorithm $\hat{f}$.
- There are lots of methods to solve the same inference problem. Most of them have options/hyperparameters.
- We also need a tool for **model selection**: choose best method or best class of hypothesis for a particular problem.
- Validation is commonly used for model selection as well. However, some **care** is necessary when doing both (which is often the case).

An example: selection bias in gene extraction

- 2002 paper by C. Ambroise and G. McLachlan: *Selection bias in gene extraction on the basis of microarray gene-expression data*.
- At the time, several paper using microarrays for cancer diagnosis claimed 0% generalization error estimated by cross validation:
  - Xiong et al.(2001), Mol Genet Metab, *Feature (Gene) Selection in Gene Expression-Based Tumor Classification*.
  - Zhang et al. (2001), Proc Natl Acad Sci USA, *Recursive partitioning for tumor classification with gene expression microarray data*.
  - Guyon et al. (2002), Mach Learn, *Gene Selection for Cancer Classification using Support Vector Machines*.

An example: selection bias in gene extraction

- General procedure was:
  1. Select a few genes which are good predictors over all samples.
  2. Perform cross validation to estimate the generalization error of a method using these genes.

  Exercise: What is wrong with this procedure? What should be done instead

An example: selection bias in gene extraction

- General procedure was:
  1. Select a few genes which are good predictors over all samples.
  2. Perform cross validation to estimate the generalization error of a method using these genes.

  Exercise: What is wrong with this procedure? What should be done instead
- The samples used to estimate the generalization error were used to select predictive genes.

An example: selection bias in gene extraction

- General procedure was:
  1. Select a few genes which are good predictors over all samples.
  2. Perform cross validation to estimate the generalization error of a method using these genes.

  Exercise: What is wrong with this procedure? What should be done instead
- The samples used to estimate the generalization error were used to select predictive genes.
- The predictive genes are optimal for the samples used to estimate the generalization error, which leads to an over-optimistic assessment regarding what will happen for actually new samples.

An example: selection bias in gene extraction

- General procedure was:
  1. Select a few genes which are good predictors over all samples.
  2. Perform cross validation to estimate the generalization error of a method using these genes.

  Exercise: What is wrong with this procedure? What should be done instead
- The samples used to estimate the generalization error were used to select predictive genes.
- The predictive genes are optimal for the samples used to estimate the generalization error, which leads to an over-optimistic assessment regarding what will happen for actually new samples.
- Picking a gene set is **model selection**, computing the generalization error of the estimator built over these genes is **model assessment**.

Model selection vs assessment

- Same thing goes for selecting a regularization parameter or a method: cross-validation error over D_n gives you an estimate of your best option (**model selection**), but it doesn't tell you how your best option will perform on new data (**model assessment**).

Model selection vs assessment

- Same thing goes for selecting a regularization parameter or a method: cross-validation error over D_n gives you an estimate of your best option (**model selection**), but it doesn't tell you how your best option will perform on new data (**model assessment**).
- **[Exercise:]** what would be an acceptable procedure to select a regularization parameter and estimate the resulting generalization error using a dataset D_n?

Model selection vs assessment

- Same thing goes for selecting a regularization parameter or a method: cross-validation error over D_n gives you an estimate of your best option (**model selection**), but it doesn't tell you how your best option will perform on new data (**model assessment**).
- **[Exercise:]** what would be an acceptable procedure to select a regularization parameter and estimate the resulting generalization error using a dataset D_n?
  - Train/Validation/Test split.
  - Double cross-validation.

Model selection vs assessment

- Principle: the data you use to estimate the generalization error of an algorithm cannot be used in any way to build the estimator. But it is easy to get confused, and difficult to strictly follow this principle when data is scarce.

Model selection vs assessment

- Principle: the data you use to estimate the generalization error of an algorithm cannot be used in any way to build the estimator. But it is easy to get confused, and difficult to strictly follow this principle when data is scarce.
- Maybe even more important than choosing best type of CV. Still results in many mistakes today.

Model selection vs assessment

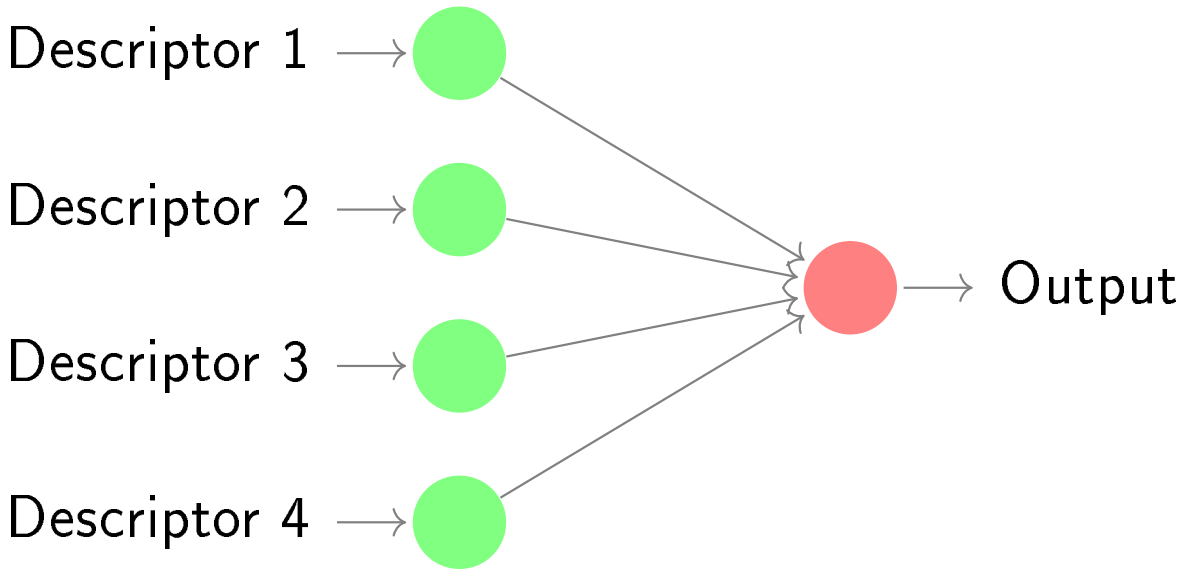
- Principle: the data you use to estimate the generalization error of an algorithm cannot be used in any way to build the estimator. But it is easy to get confused, and difficult to strictly follow this principle when data is scarce.
- Maybe even more important than choosing best type of CV. Still results in many mistakes today.
- Other frequent source of mistakes in CV: duplicate/non i.i.d. samples.

Part III

Neural networks

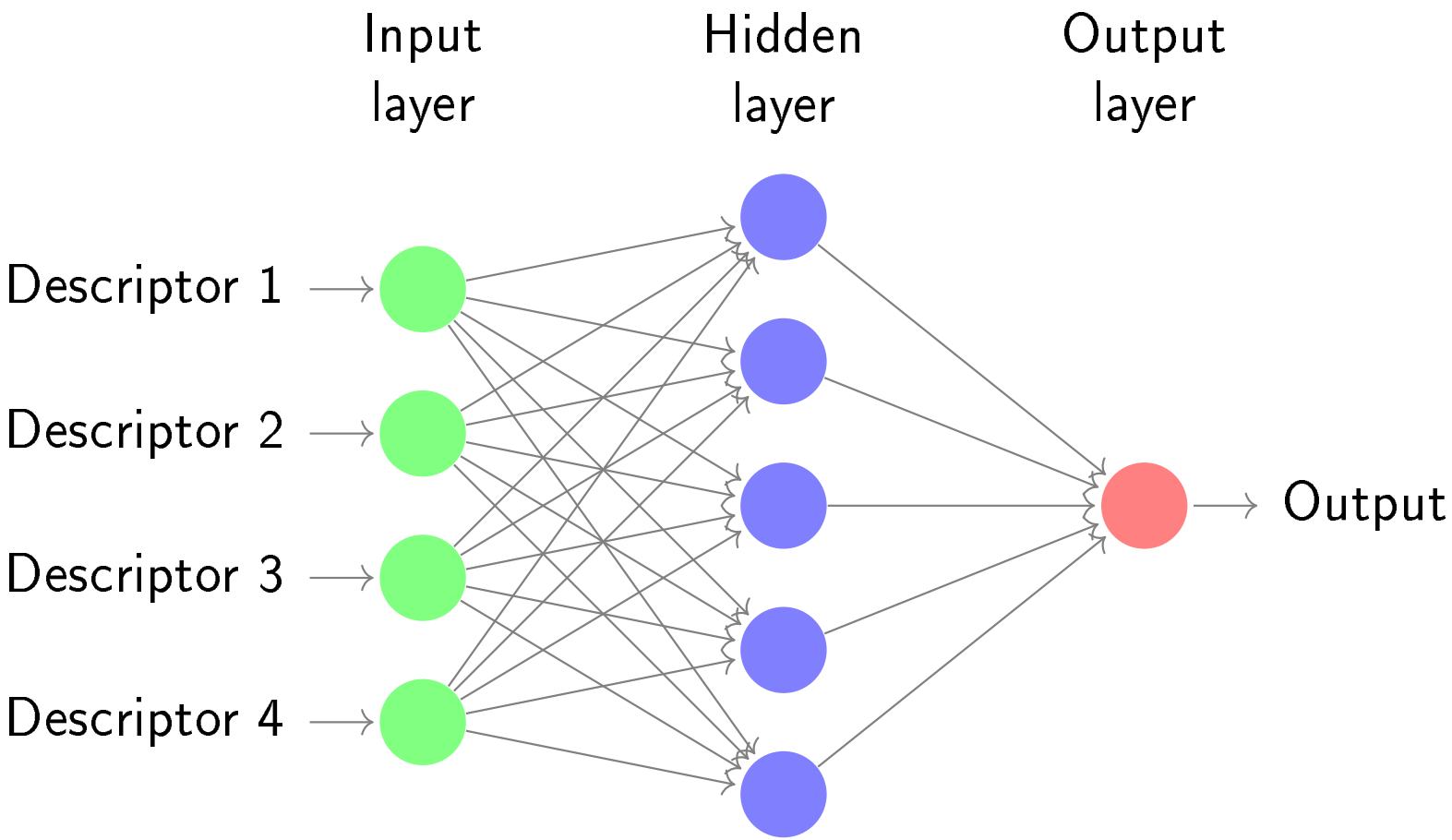
Used the geometric deep learning blog posts of Bronstein et al.

The perceptron



- Output is a linear combination of inputs, followed by a point non-linearity.
- McCullochs and Pitts (1943), Rosenblatt (1958).

The multilayer perceptron



(Already in Rosenblatt (1958))

Approximation properties of MLPs

The multilayer perceptron defines a family of functions

$$f(x) = \sigma_k(A_k\sigma_{k-1}(A_{k-1}\dots\sigma_2(A_2\sigma_1(A_1x))\dots)),$$

Parameterized by weights $A_1, A_2, \dots, A_k$.

The Perceptron book (Minsky and Papert, 1969)

- Criticizes the expressive power of perceptron (cannot learn simple logical functions such as XOR).
- Limited to single layer perceptron.

Universal approximation theorems (Cybenko 1989, Hornik 1991)

- Approximate any continuous functions to any desired accuracy.
- True even with a single hidden layer!
- An attractive feature for learning.

Back to our objective

Structural risk minimization (Vapnik and Chervonenkis, 1974)

1. Define families of functions of increasing complexity

   Ability to increase/control complexity?

2. Minimize the empirical risk over each family

   Ability to optimize over the family?

3. Retain best generalization performance

   Validation

- Multi-layer perceptrons are attractive for point 1,
- but we need a way to optimize it.
- We also need a way to control the complexity.

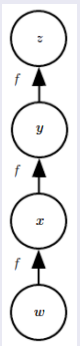
Optimizing a multi-layer perceptron

Empirical risk minimization $\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))$

- Solved analytically (e.g. linear regression) or numerically.
- Latter approach requires a descent direction.

Backpropagation (Rumelhart, Hinton and Williams, 1986)

- Relies on chain rule to express partial derivatives.
- Strategy: store all activations (forward pass) before computing derivatives (backward pass).
- Avoids recomputing quantities, but high memory cost!



$$\begin{aligned}
&\frac{\partial z}{\partial w} \\
=&\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \\
=&f'(y)f'(x)f'(w) \\
=&f'(f(f(w)))f'(f(w))f'(w).
\end{aligned}$$

From Deep Learning, Goodfellow et al. 2016, Figure 6.9

Controlling the complexity

Before deep learning: explicit regularization

- Often by some norm of the function (ℓ_p, structured...).
- Enforces some prior knowledge (smoothness, sparsity...).
- SRM strategy: find small sets that contain good candidates.

Deep learning typically relies on implicit regularization

- Stochastic gradient descent.
- Early stopping.
- Dropout.

Geometric deep learning provides some explicit, prior-based regularization.

Structural risk minimization checkup

With neural networks, we now know how to

1. Define families of functions of increasing complexity
   Approximation theorems + geometric deep learning
2. Minimize the empirical risk over each family
   Backpropagation+SGD
3. Retain best generalization performance
   Validation

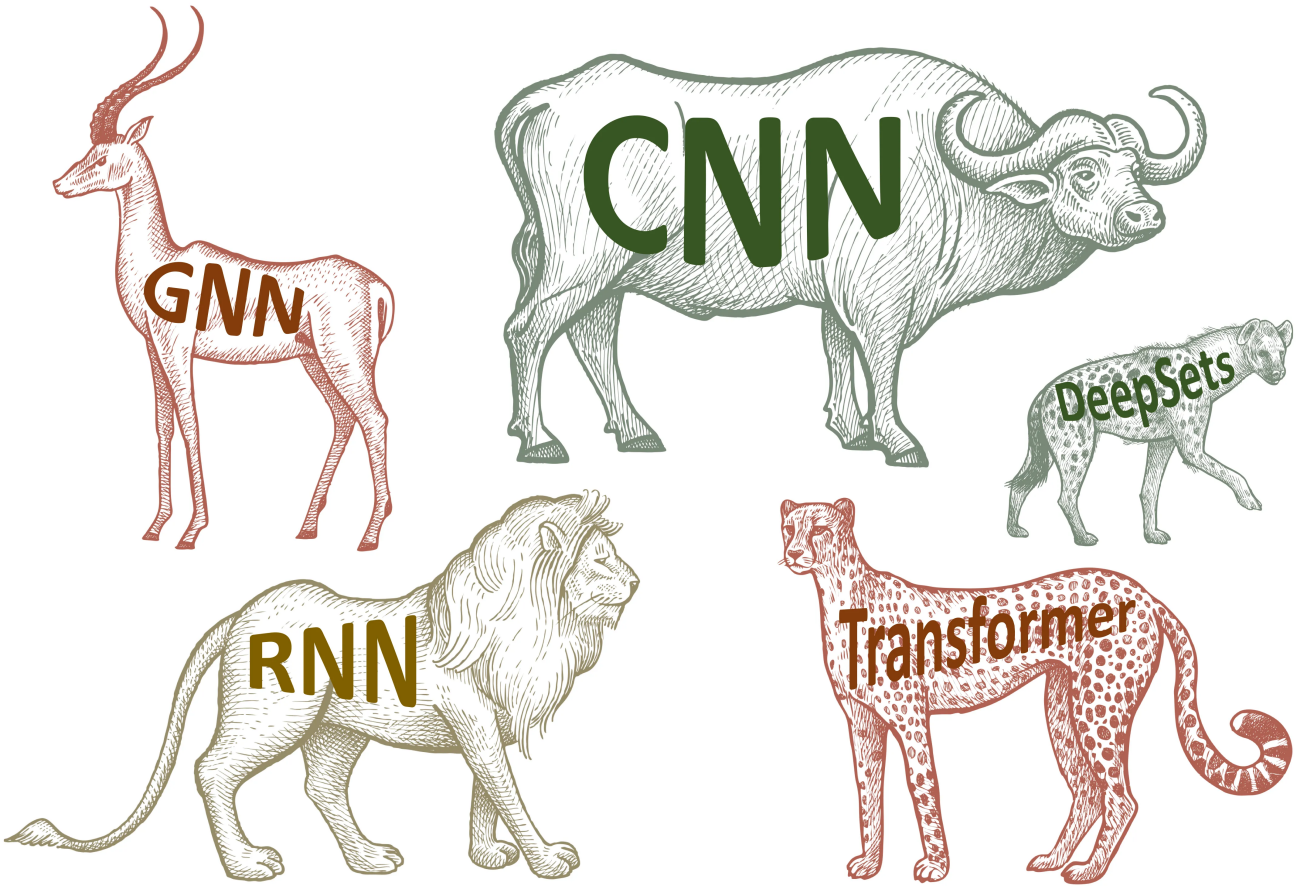
(once we have geometric deep learning)

Part IV

Geometric deep learning

Used the geometric deep learning blog posts of Bronstein et al.

Geometric deep learning (Bronstein *et al.*, 2021)



We believe that the current state of affairs in the field of deep (representation) learning is reminiscent of the situation of geometry in the nineteenth century

The Erlangen program

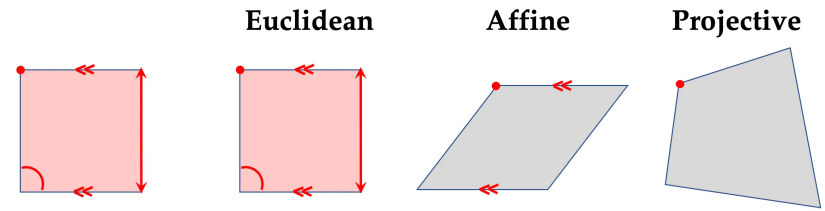
Geometry in the 19th century

- Introduction of several non-Euclidean geometries,
- but they all seem independant.

Felix Klein, 1872: every geometry is characterized by

- Its symmetry group (transformations by which an object is invariant).
- The set of objects that are invariant under these symmetries.

Induces a structure across geometries



	Euclidean	Affine	Projective
angle	+	—	—
distance	+	—	—
area	+	—	—
parallelism	+	+	—
intersection	+	+	+

How does it relate to deep learning?

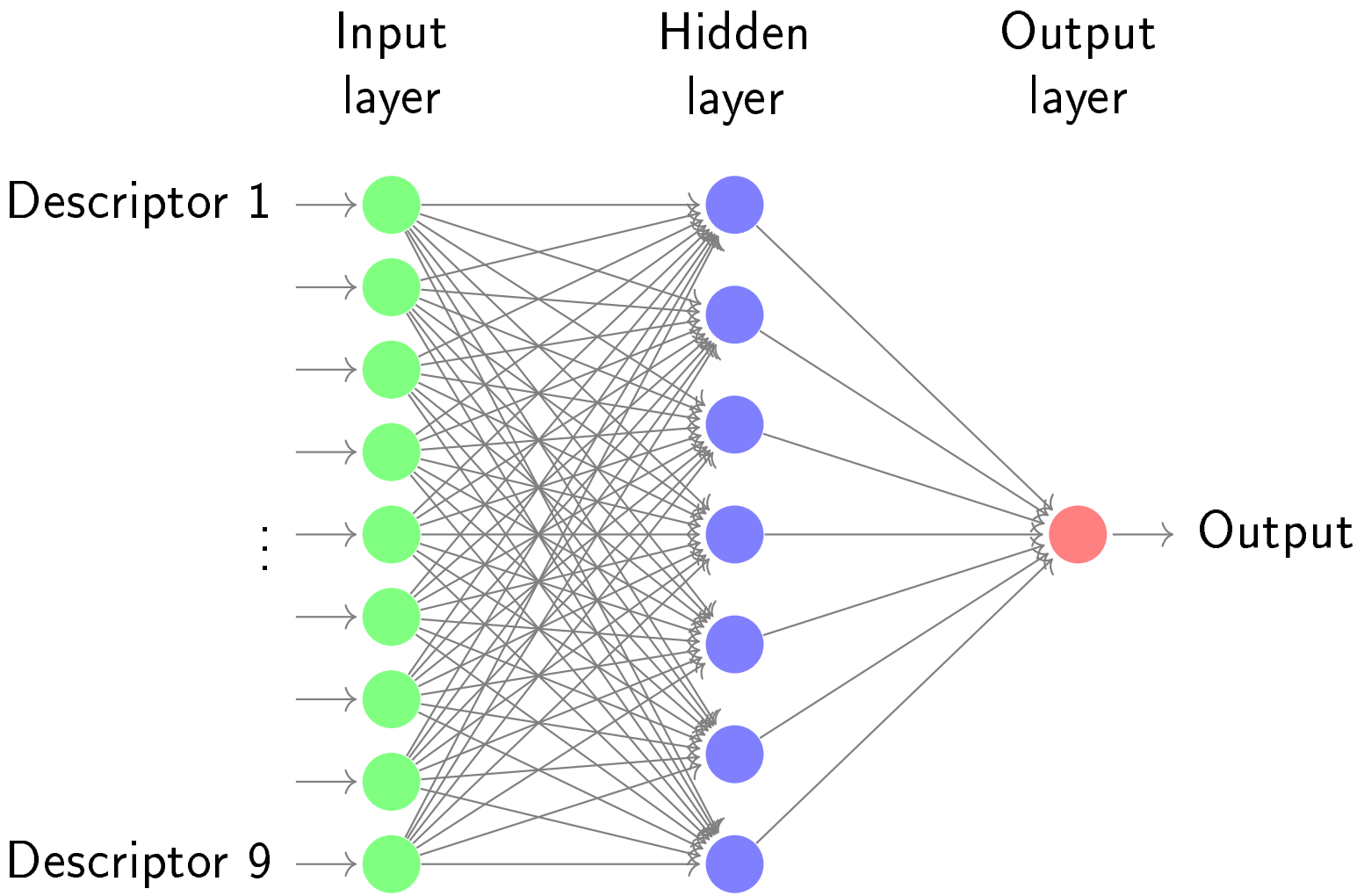
Curse of dimensionality

- MLPs can approximate any continuous function.
- But number of samples required to learn from such a large space grows with the dimension.
- Accounting for symmetries in the data saves us from this situation.

Invariant neural networks

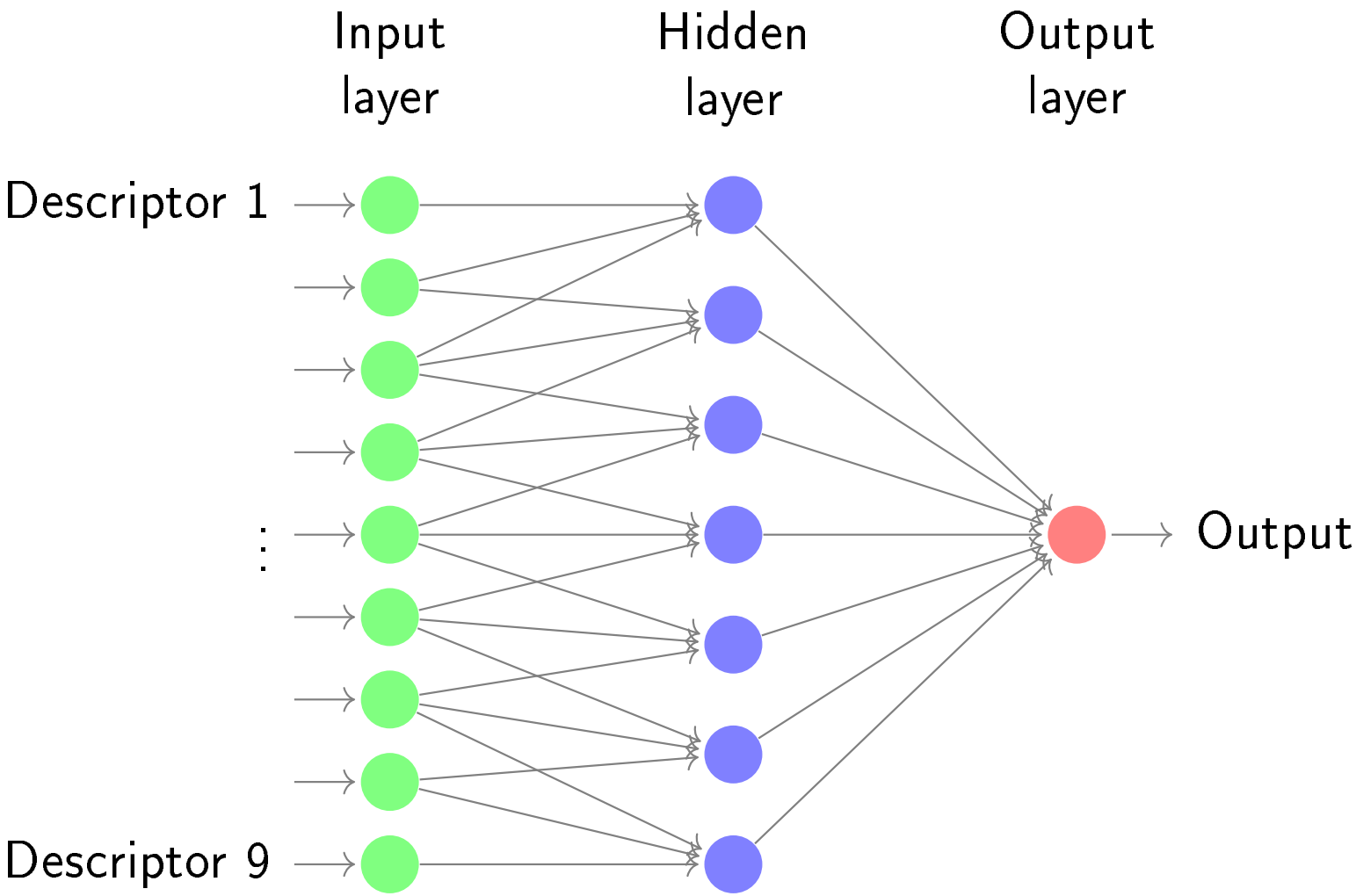
- Specialized architectures,
  invariant to some transformation of the input.
- Like in geometry, emerged independently and met success
  long before a theoretical framework was proposed.
- Typical symmetries in genomics: translation, reverse complement, permutation.

From fully connected to convolutional neural networks



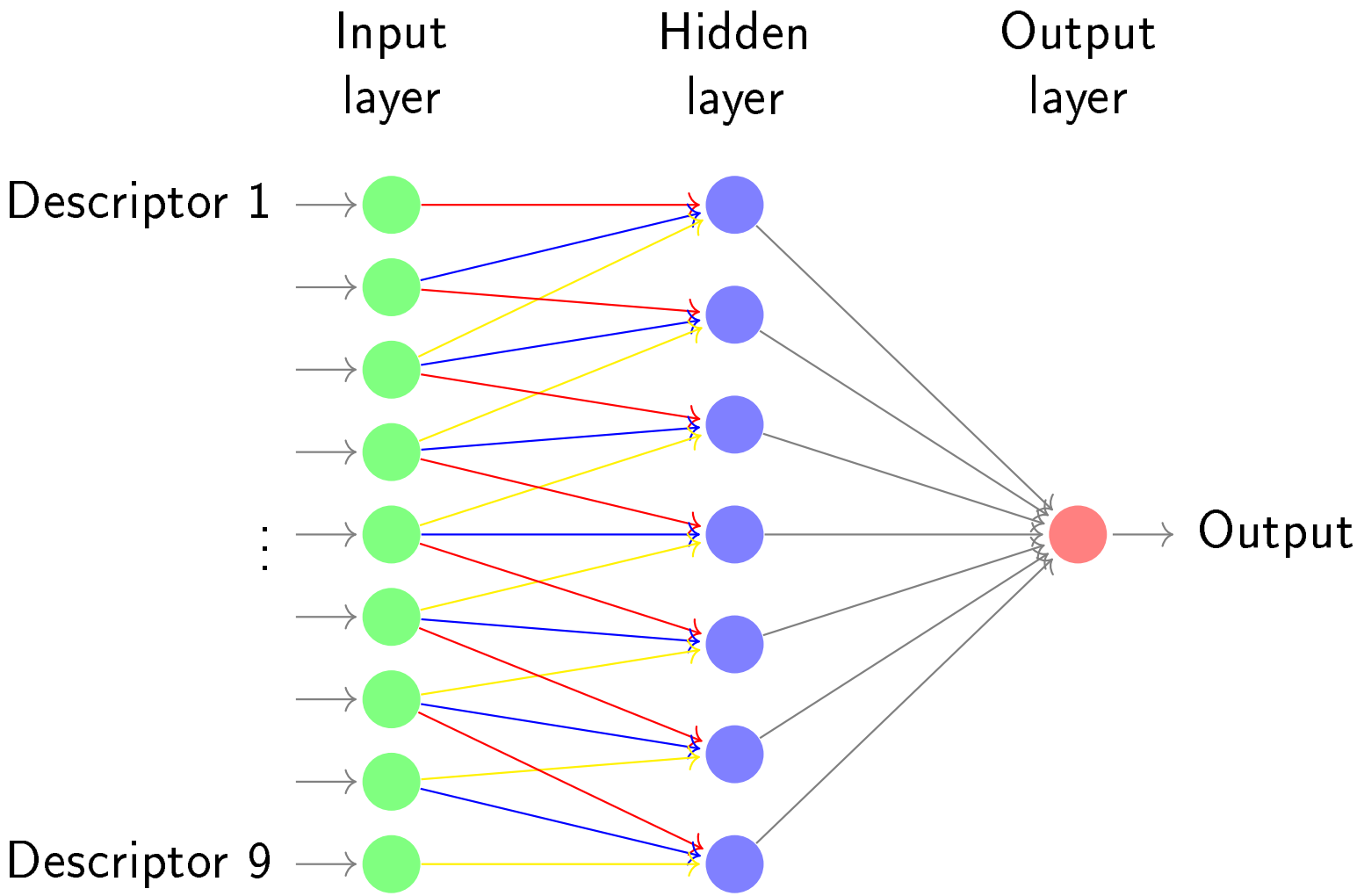
CNNs (Le Cun *et al.*, 1989) are instances of MLPs.

From fully connected to convolutional neural networks



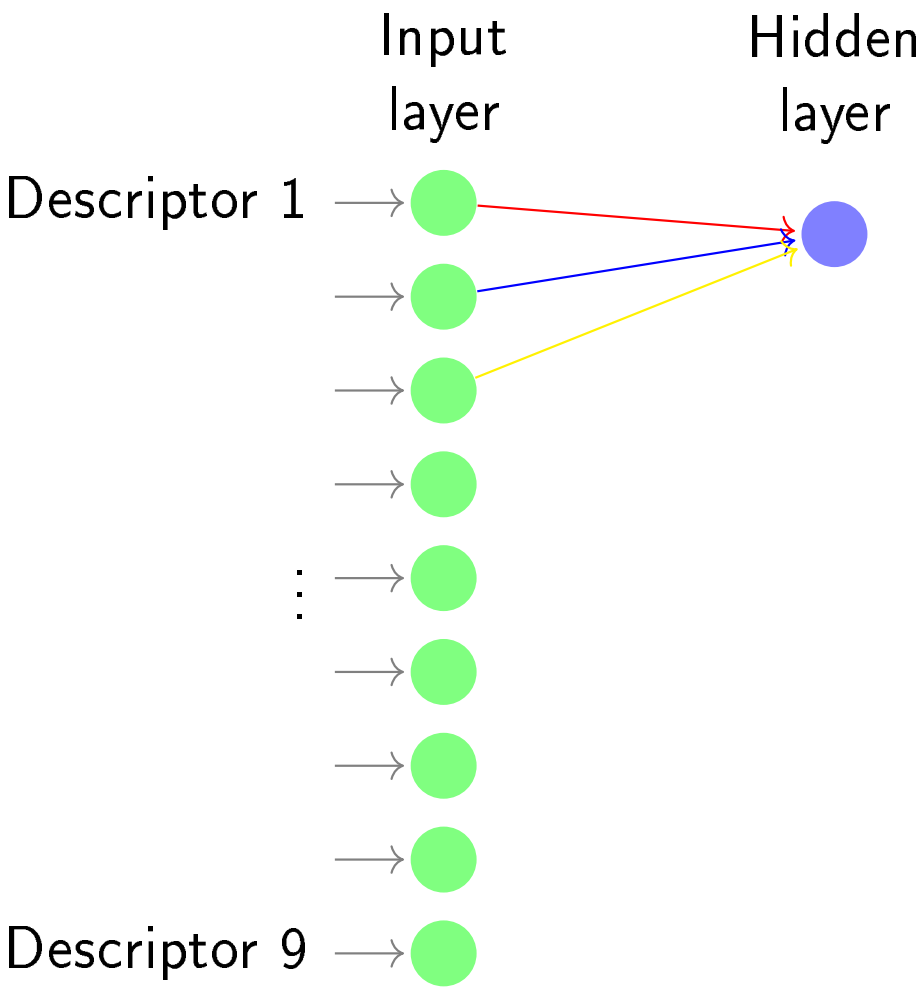
Only input "local" descriptors to hidden nodes (instance of regularization).

From fully connected to convolutional neural networks



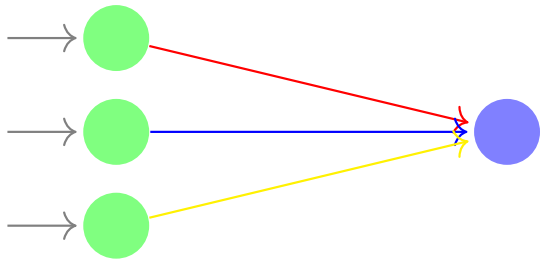
In addition, hidden nodes share their weight (further regularization).

From fully connected to convolutional neural networks



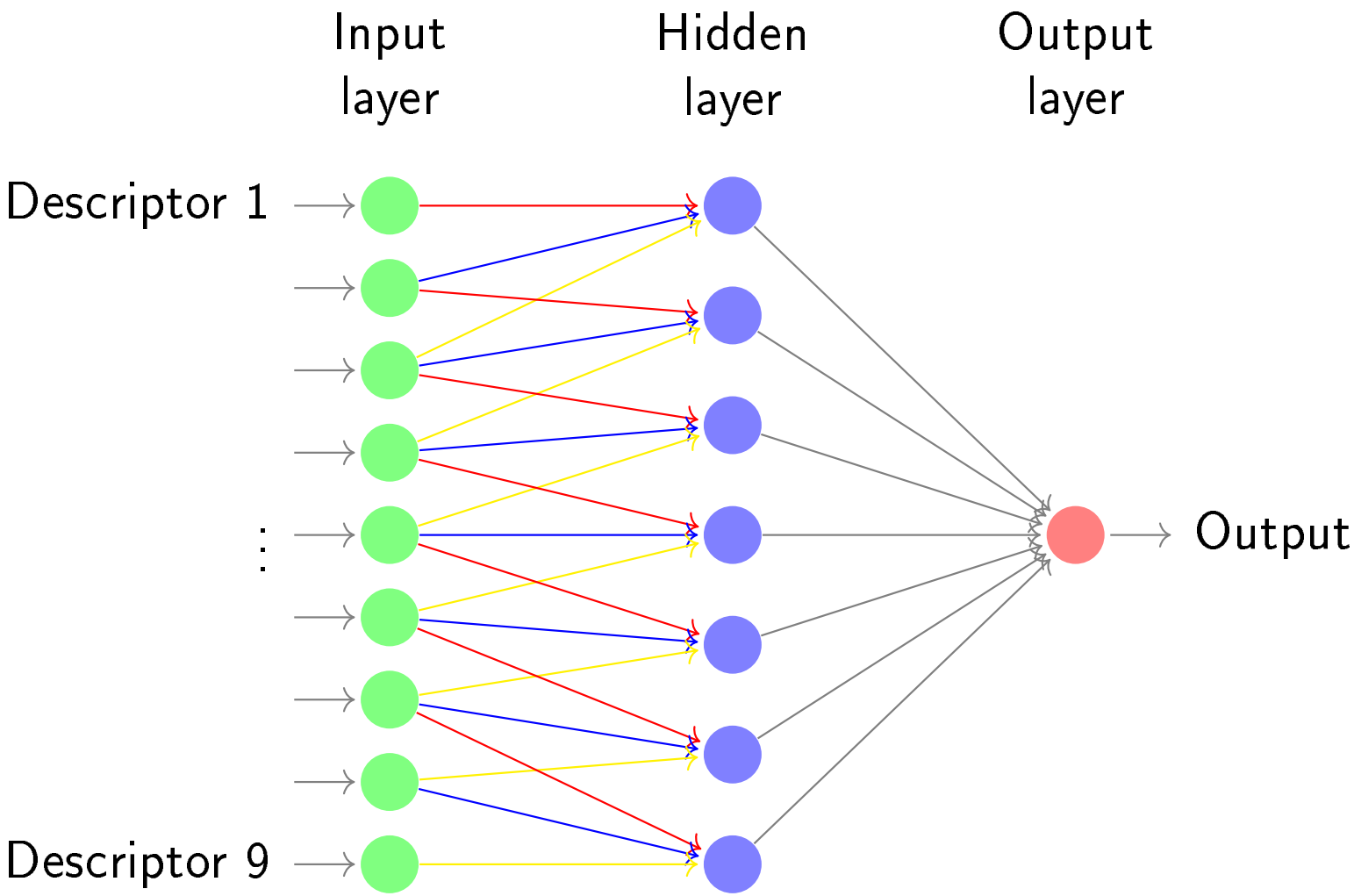
This can also be thought of as sliding a single hidden node along the input, hence the convolution interpretation.

Convolutional neural networks



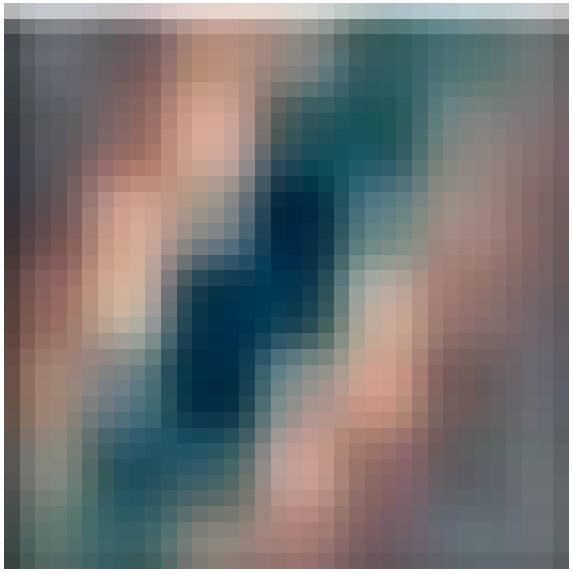
- A set of shared weights used in convolution is called a **filter**.
- For images, neighborhood is typically 2D (descriptors are an array of pixels).

Convolutional neural networks



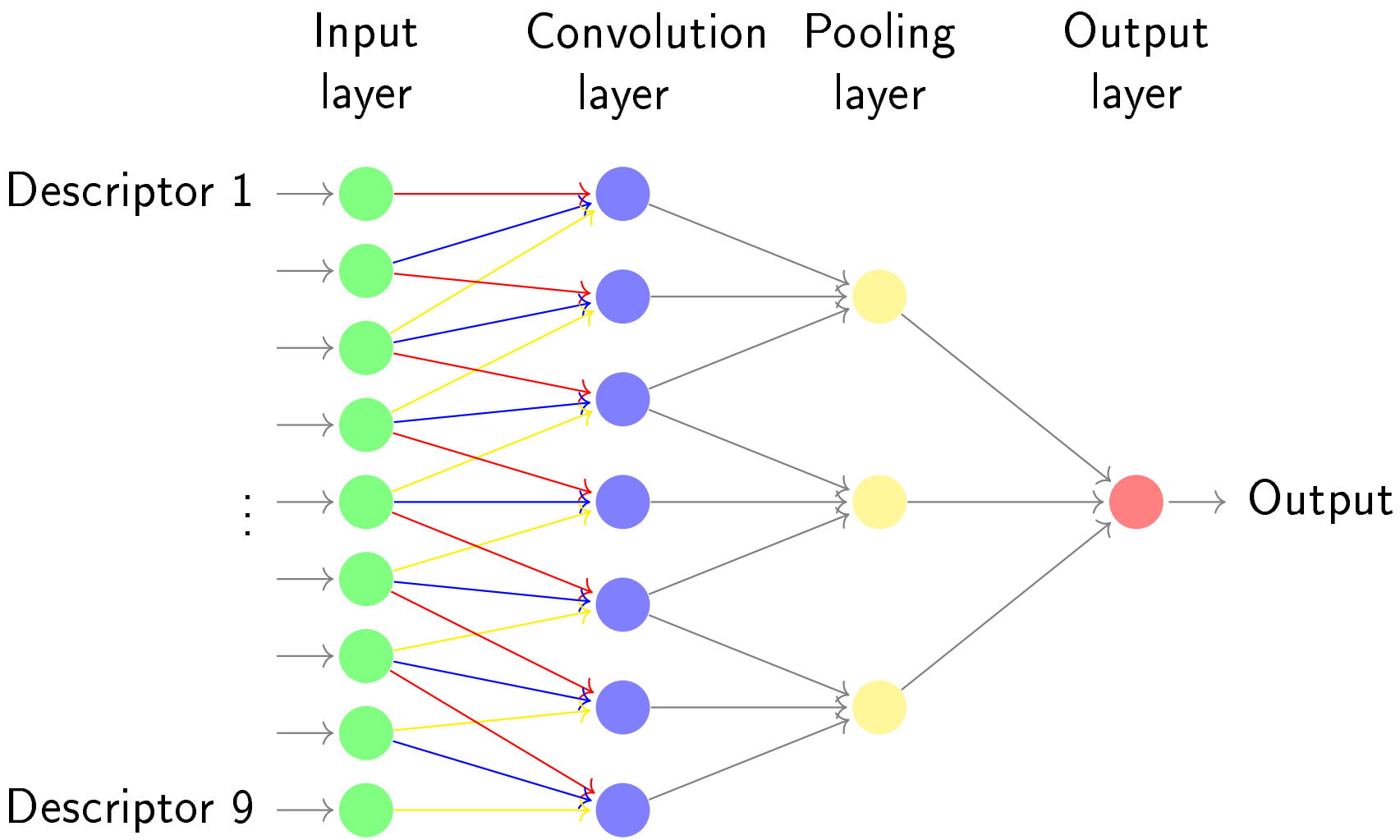
If we use this architecture with a fixed filter, the hidden layer will represent how similar each local patch is to the filter.

Convolutional neural networks



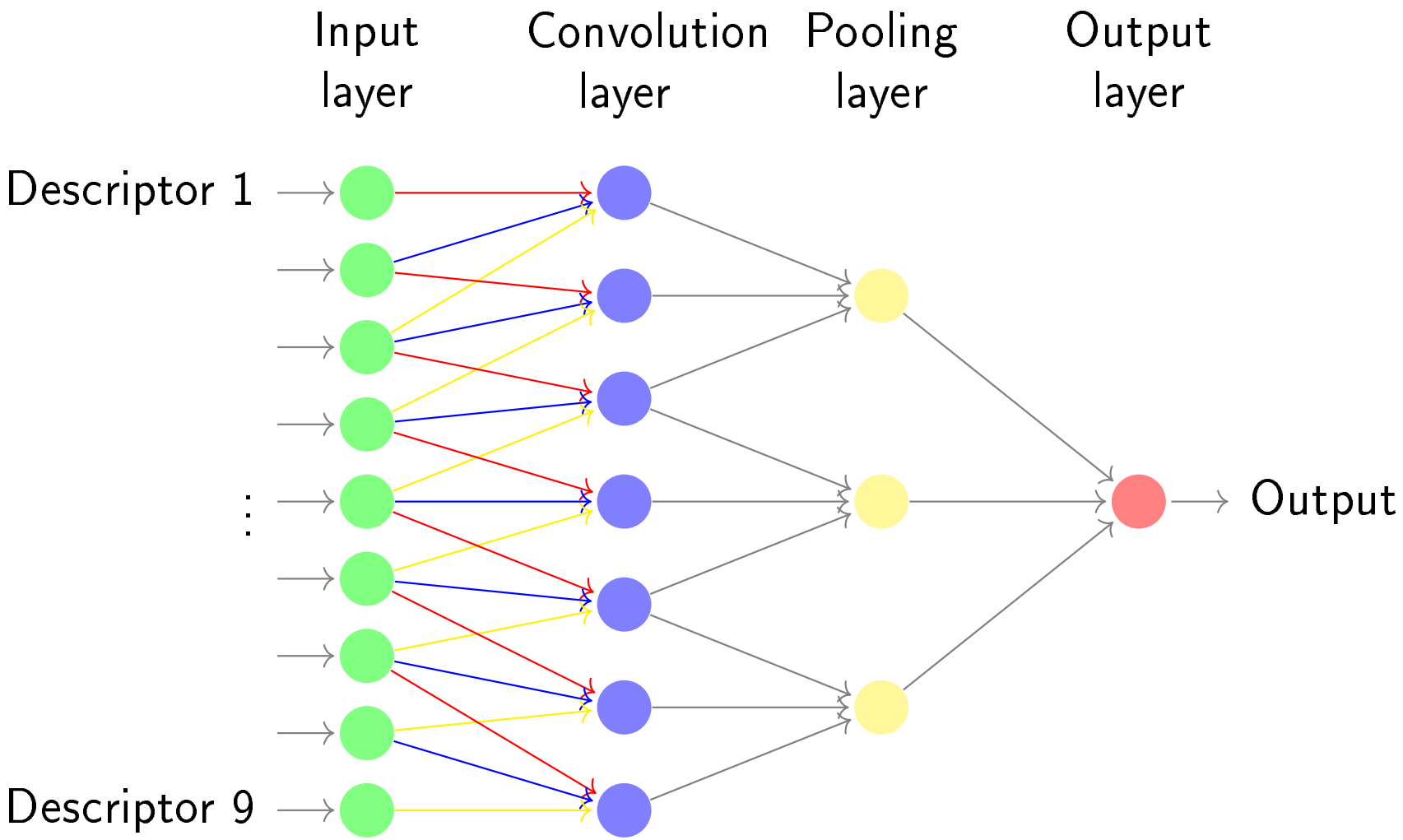
For example, this filter (organized as a 2D, color-coded matrix) would output large values in regions containing a diagonal edge.

Convolutional neural networks



Typical CNNs add a **pooling** layer after the convolution (max, mean...).

Convolutional neural networks



Output of the pooling layer indicates the presence of something resembling the filter in a larger region of the image.

Convolutional neural networks

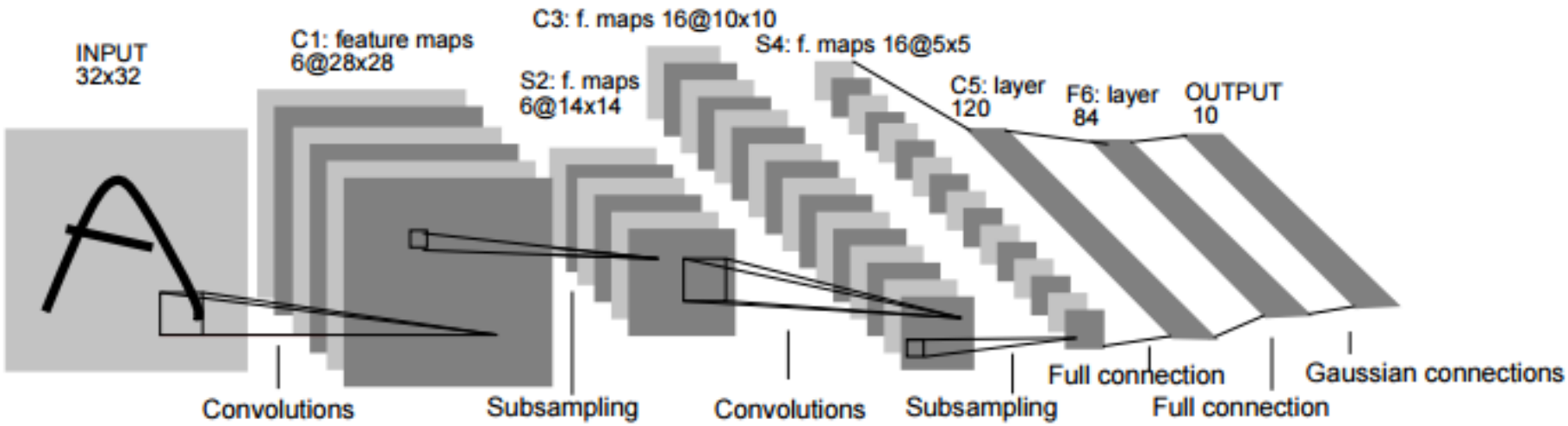


Figure: Picture from Le Cun *et al.*, 1998

In practice:

- Several filters are used.
- Several convolutional layers are stacked. Each layer takes as input the pooled filters of the previous one.

Convolutional neural networks

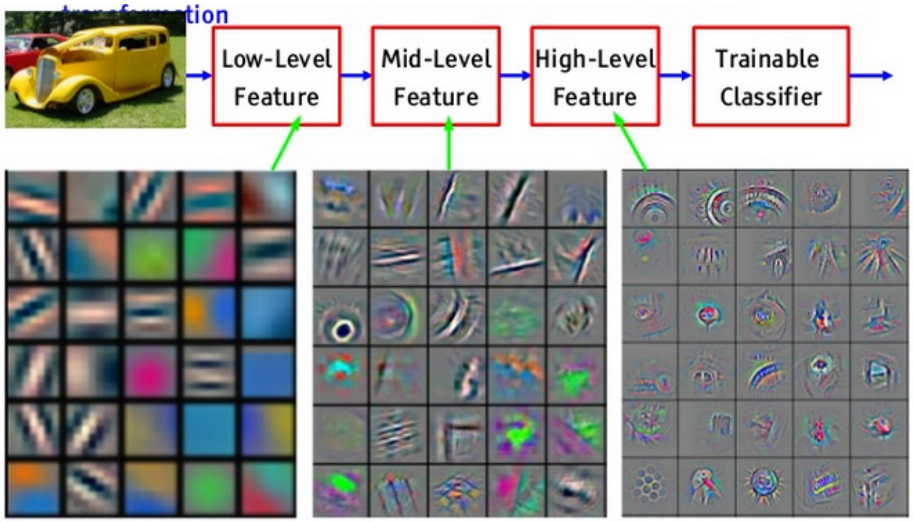


Figure: Picture from Yann LeCun's tutorial, based on Zeiler and Fergus, 2014.

- Importantly: filters are **learned**, not fixed. Interpretation: we use the data to learn descriptors which are optimal for the prediction task.
- First layers learn simple features (edges).
- Upper convolution layers learn more complex features by composition.

Convolutional neural networks

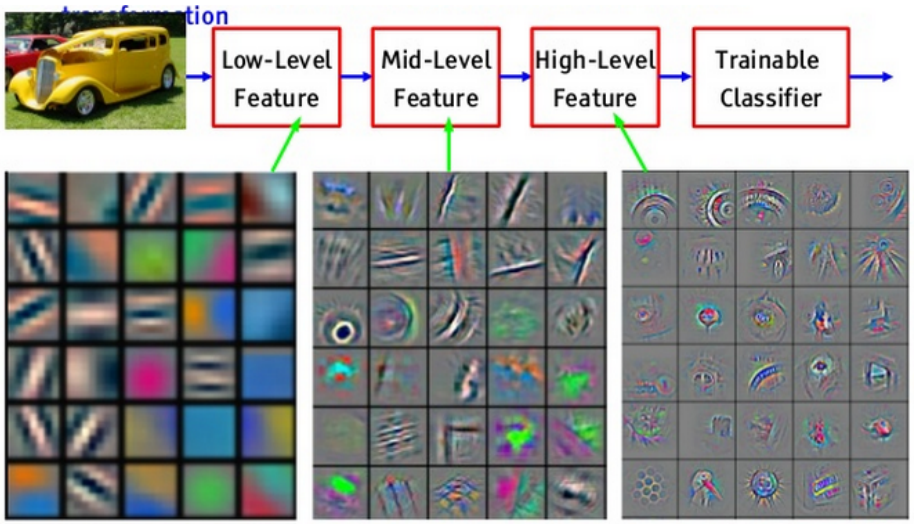


Figure: Picture from Yann LeCun's tutorial, based on Zeiler and Fergus, 2014.

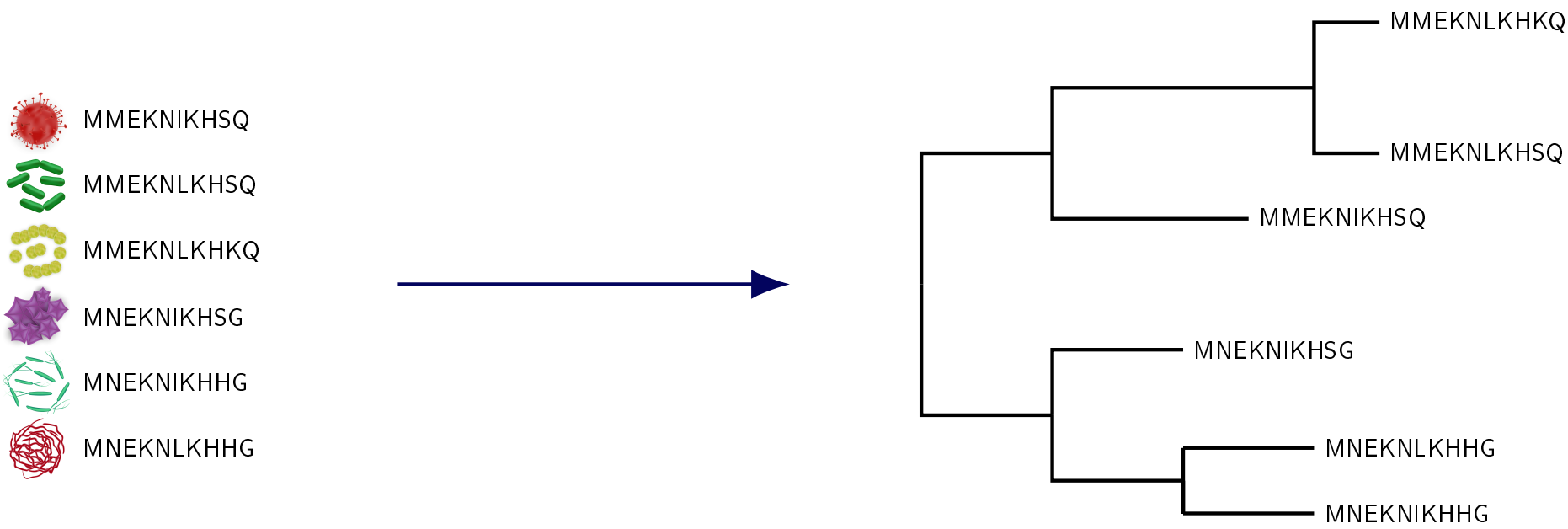
Two interpretations for the same architecture:

- Learning weights of complex parameterized functions, with some regularization.
- Learning predictive features and classifier over these features.

Part V

Machine learning for phylogenetic reconstruction

Back to neural inference

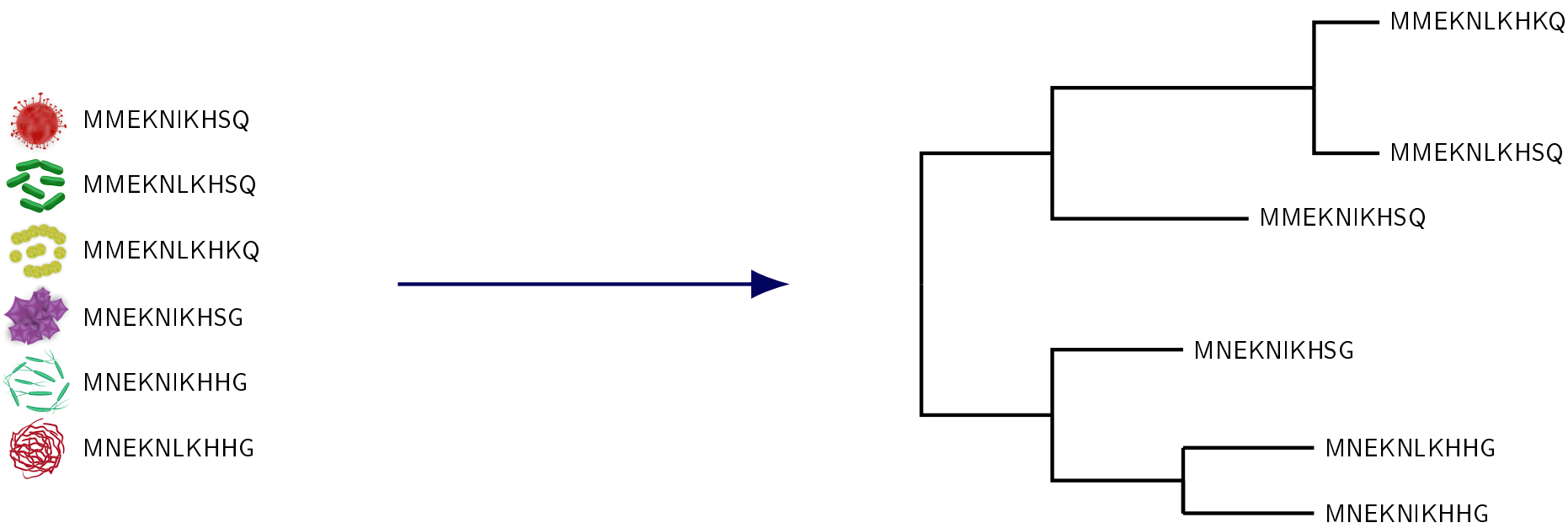


We need a learnable function that:

- outputs a phylogenetic tree.

- takes as input a set of homologous sequences,

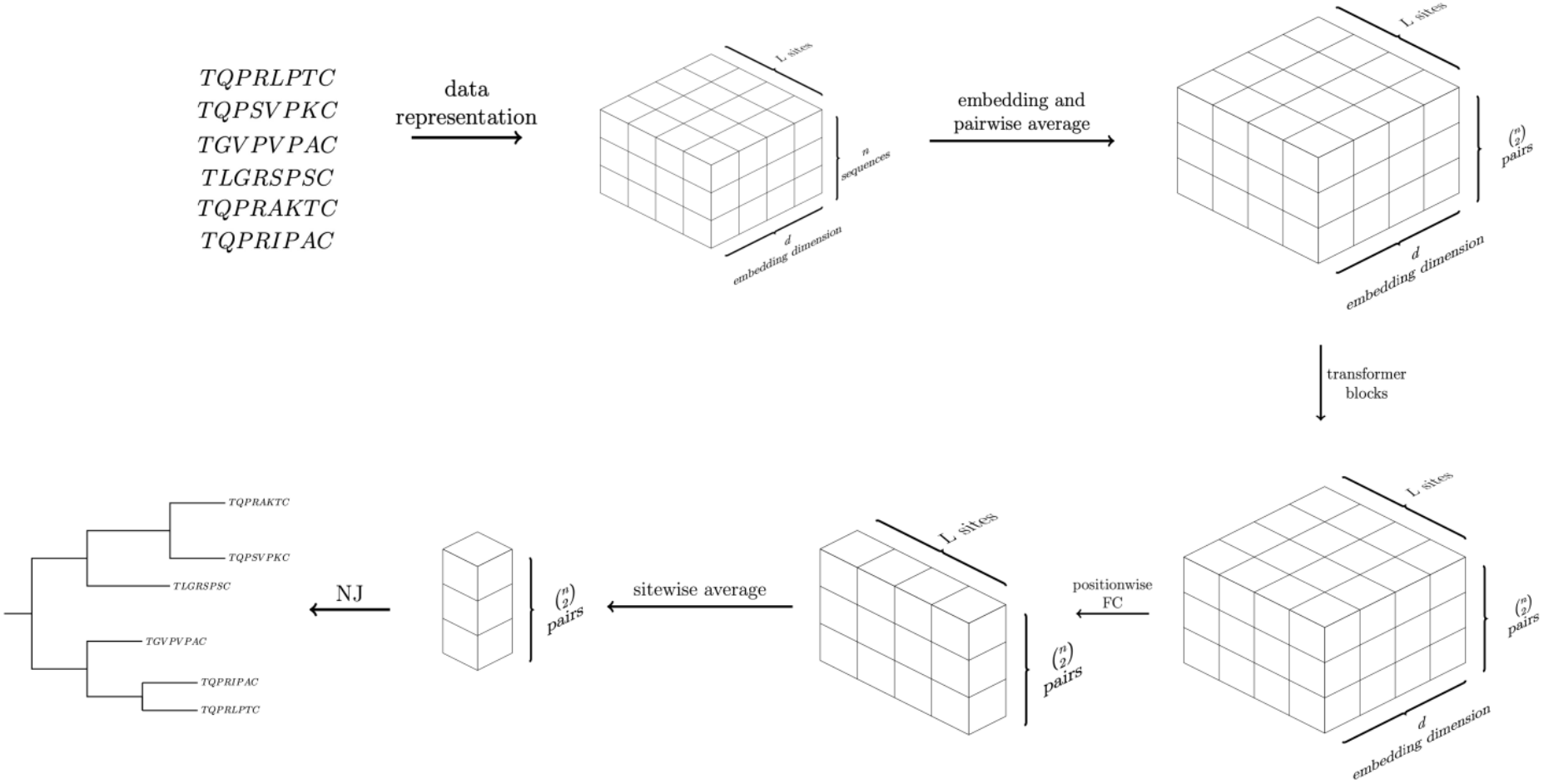
Back to neural inference



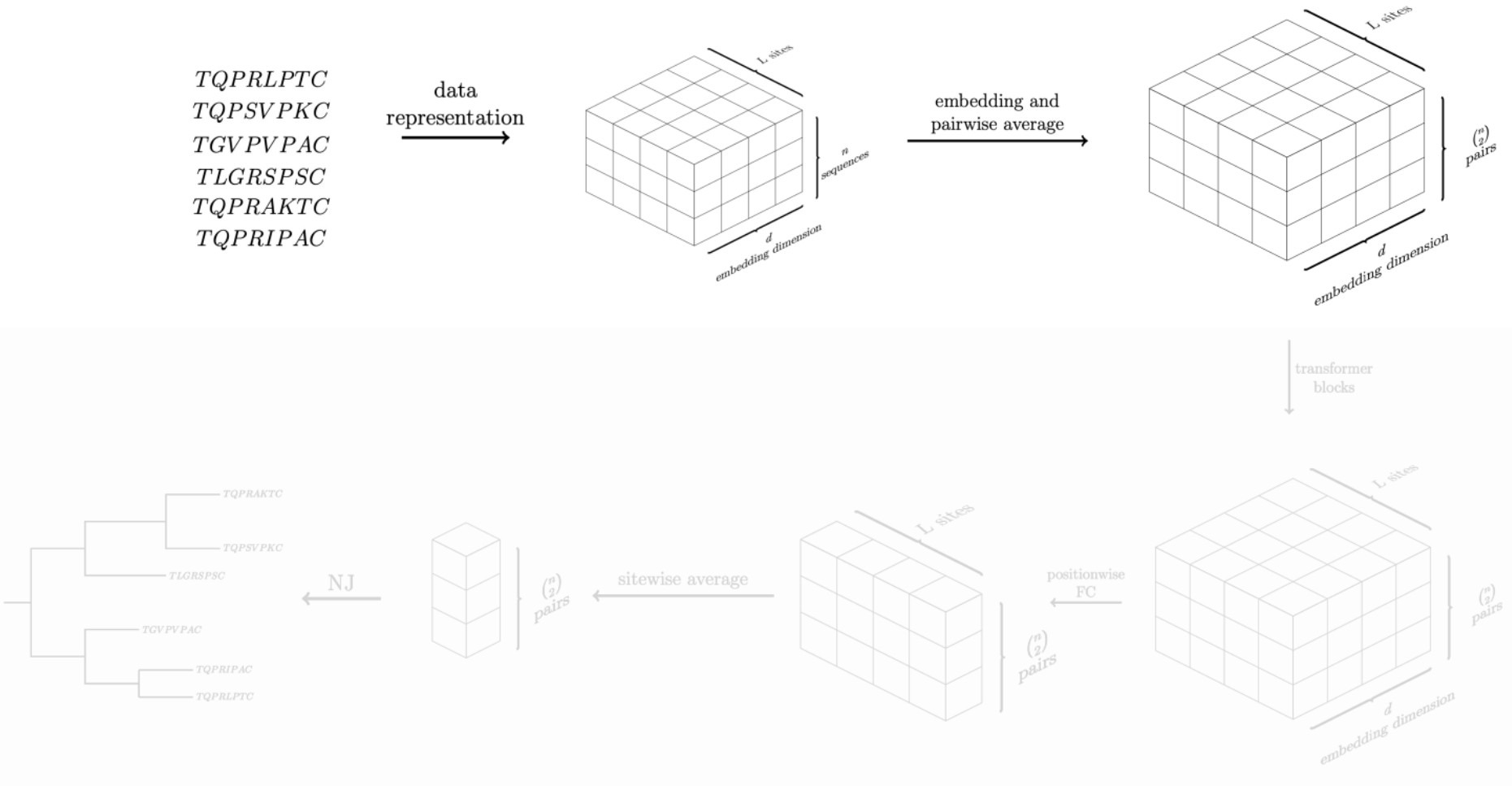
We need a learnable function that:

- outputs a phylogenetic tree.
  $\rightarrow$ **use evolutionary distances as a proxy.**
  Existing (non-scalable) alternative: classify topologies
- takes as input a set of homologous sequences,
  $\rightarrow$ **use self-attention**.

Phyloformer overview



Phyloformer overview

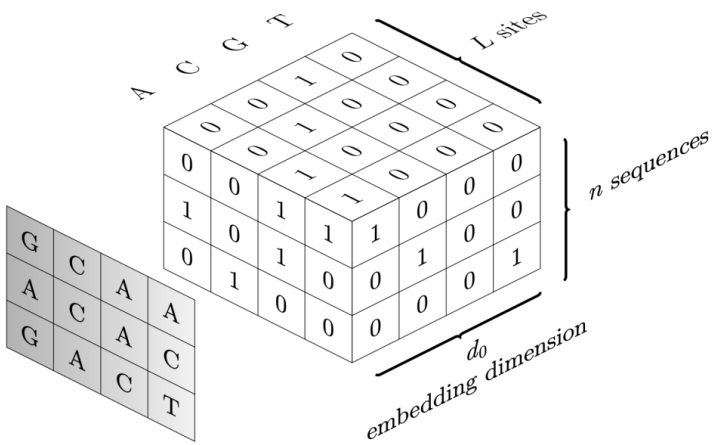


One-hot encoding for aligned sequences

A single sequence:

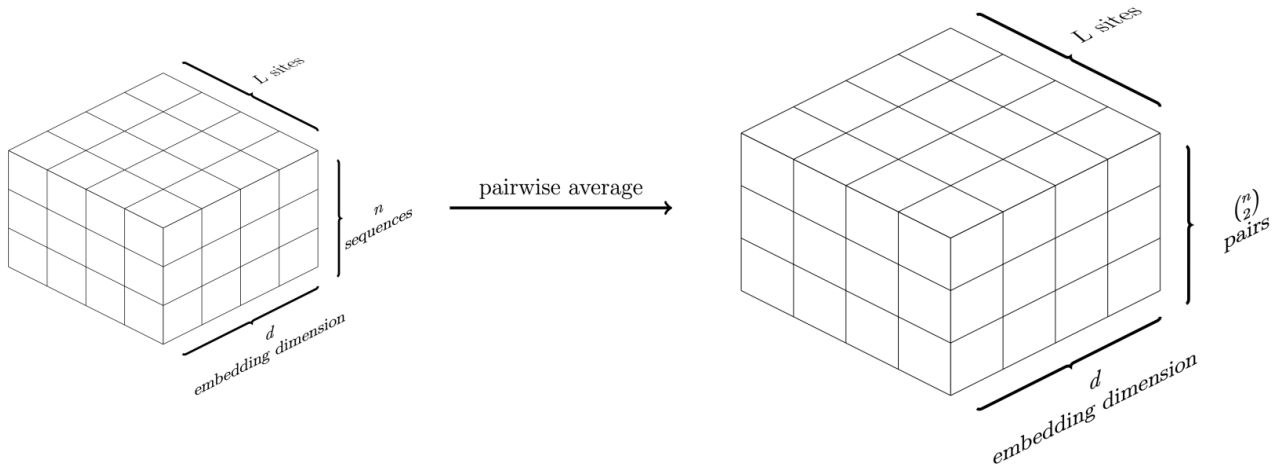
	A	A	C	G	T	$\ldots$
A	1	1	0	0	0	$\ldots$
C	0	0	1	0	0	$\ldots$
T	0	0	0	0	1	$\ldots$
G	0	0	0	1	0	$\ldots$

A set of aligned sequences:



Our alphabet is actually {A, R, N, D, ..., Y, V, X, -} so $d_0 = 22$.

Encoding **pairs** of aligned sequences

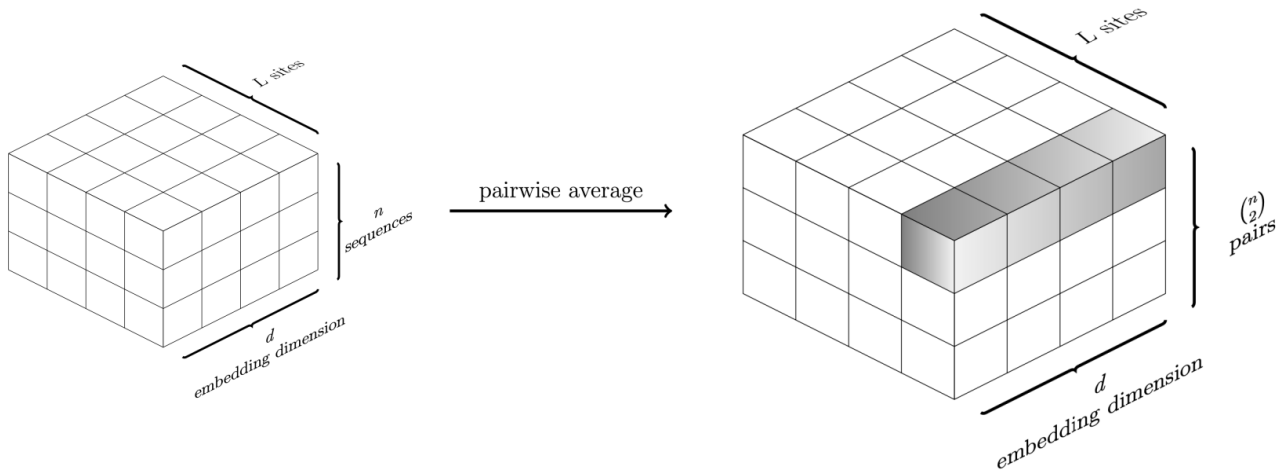


- We choose to work on pairs of sequences (predict distance for each).
- We represent each pair by simply averaging over sequences.

	A	A	C	G	T	...
	A	T	C	C	T	...
A	1	0.5	0	0	0	...
C	0	0	1	0.5	0	...
T	0	0.5	0	0	1	...
G	0	0	0	0.5	0	...

- We now have a set of $\binom{n}{2} \times L$ amino acids encoded as $\mathbb{R}^{d=22}$ vectors.

Encoding **pairs** of aligned sequences

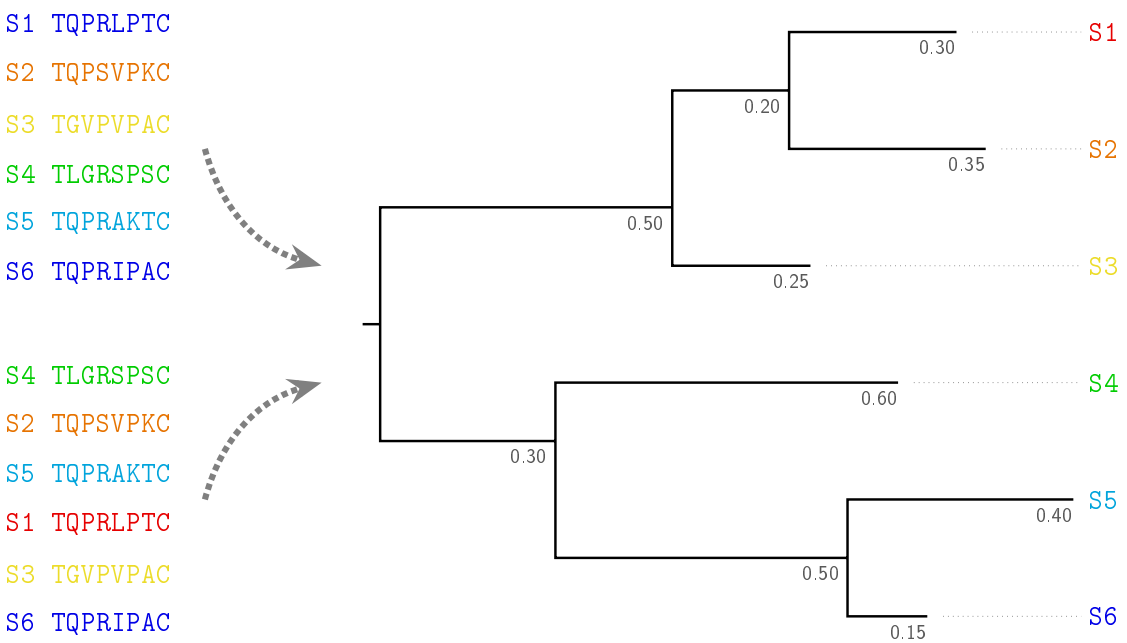


- We choose to work on pairs of sequences (predict distance for each).
- We represent each pair by simply averaging over sequences.

	A	A	C	G	T	...
	A	T	C	C	T	...
A	1	0.5	0	0	0	...
C	0	0	1	0.5	0	...
T	0	0.5	0	0	1	...
G	0	0	0	0.5	0	...

- We now have a set of $\binom{n}{2} \times L$ amino acids encoded as $\mathbb{R}^{d=22}$ vectors.

Accounting for permutation invariance with self-attention



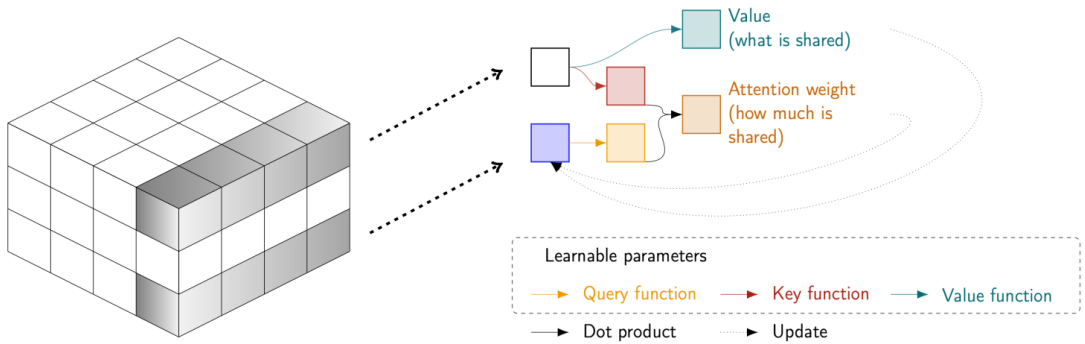
This has **no reason to be true in general** (*e.g.* linear function)!

Need to retain some expressivity.
E.g. average provides invariance but discards a lot of information.

Self-attention in a nutshell

Functions acting on unordered sets

- Updates each element as a linear combination of all of them.
- Output is a new representation of the same set. Iterate.

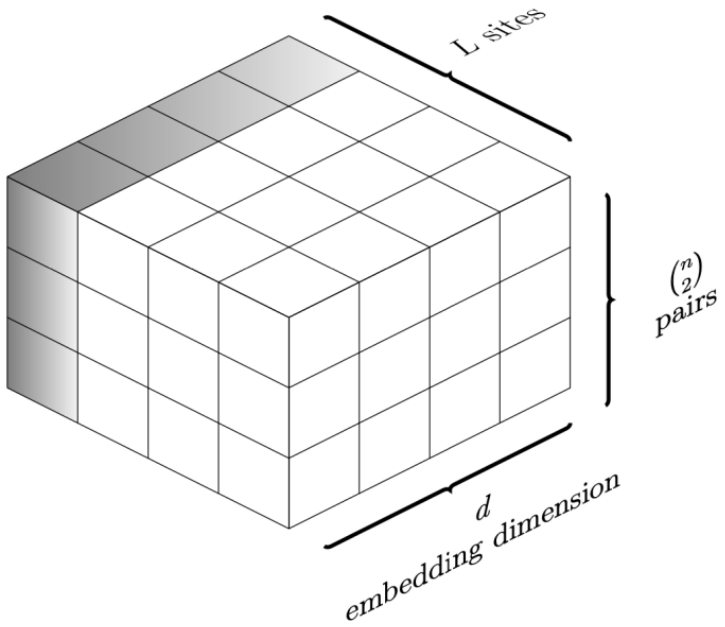


Updates

- Learnable part: function of two elements, giving weight of one in the update of the other.
- Provides equivariance, modularity to any cardinal.
- Iteratively builds a set-aware representation for each pair.

Axial attention

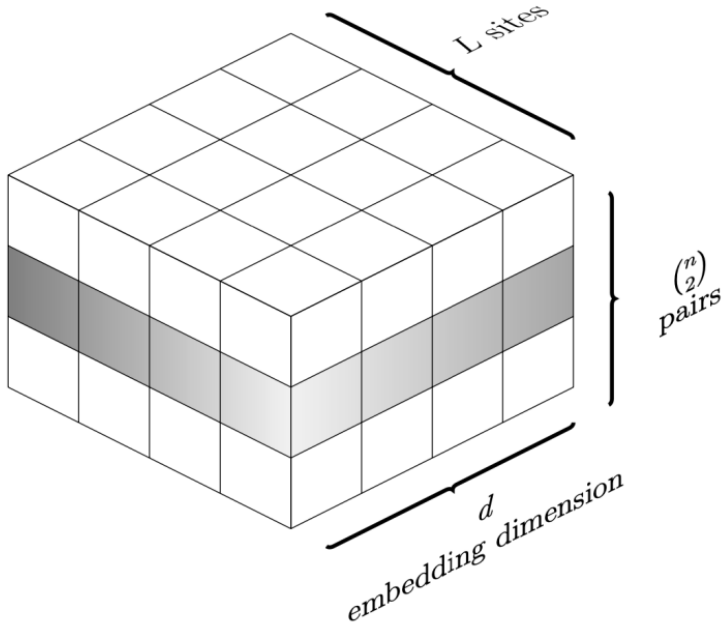
- We need equivariance both across pairs and sites.
- Alternate between column- and row-wise attention.



For each site, update each pair using all others.

Axial attention

- We need equivariance both across pairs and sites.
- Alternate between column- and row-wise attention.



For each pair, update each site using all others.

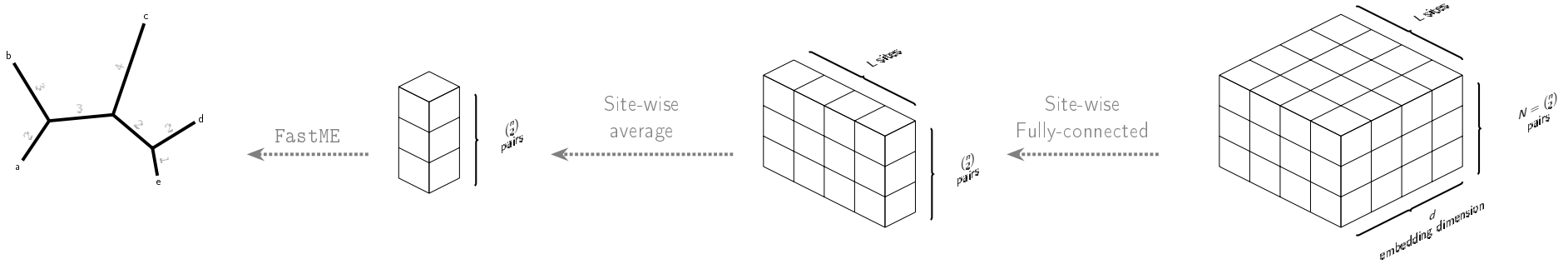
Builds a **set-aware representation** of residuals

"I'm an Alanine" $\rightarrow$

- "I'm an Alanine,
- some homologous sequences have Serines,
- many residues in the sequence are hydrophobic,
- this site is conserved,
- ..."

This representation is optimized with respect to the prediction objective.

Phyloformer overview

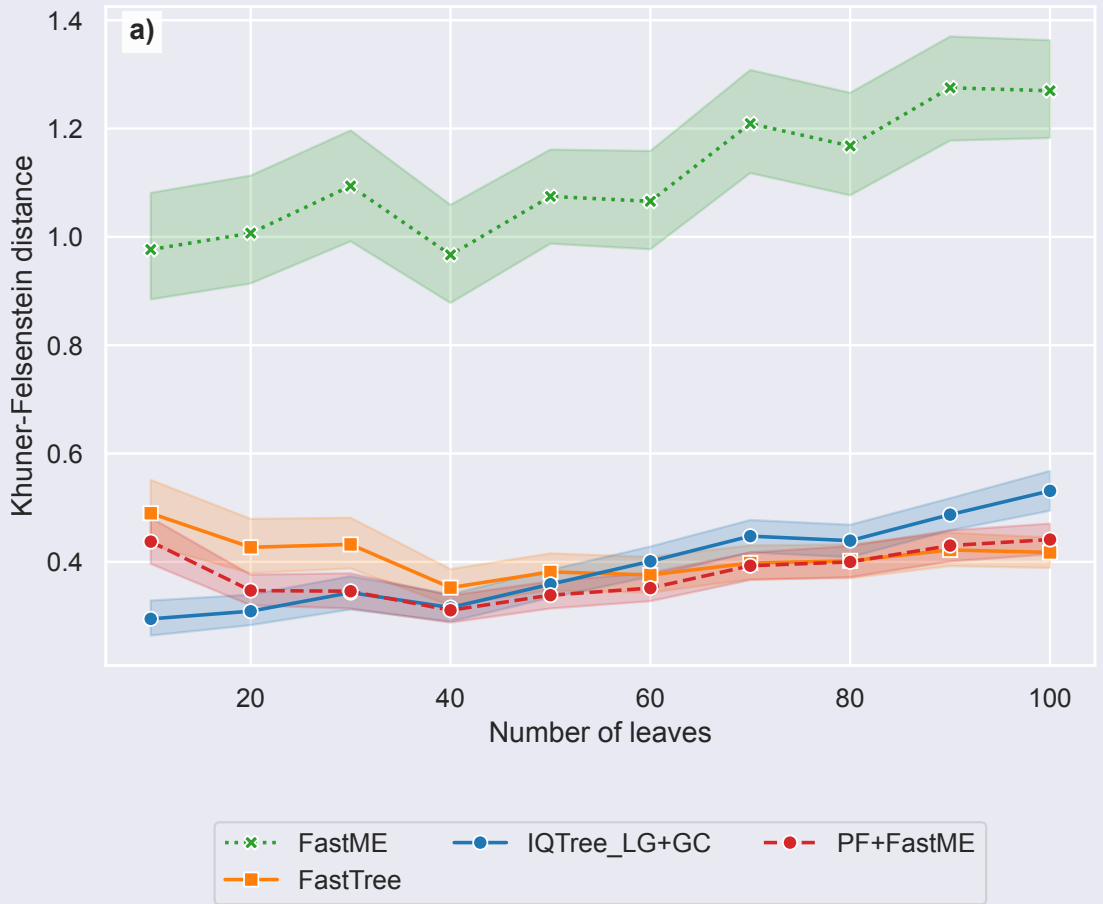


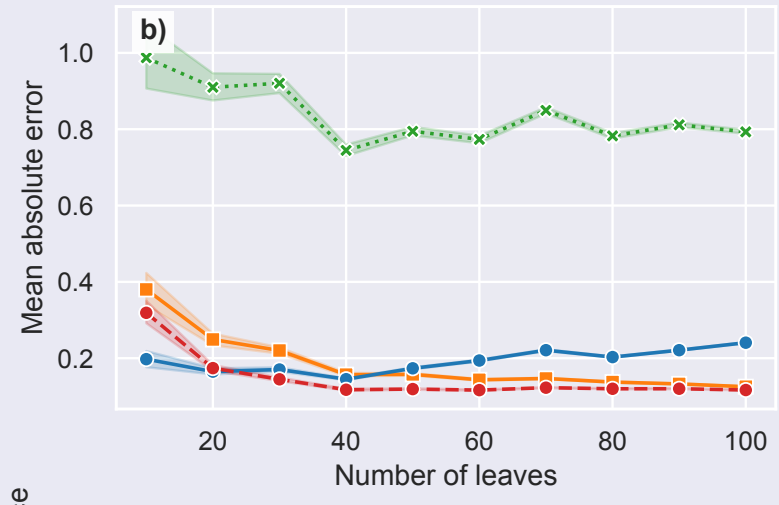
Final step: predict pairwise distances

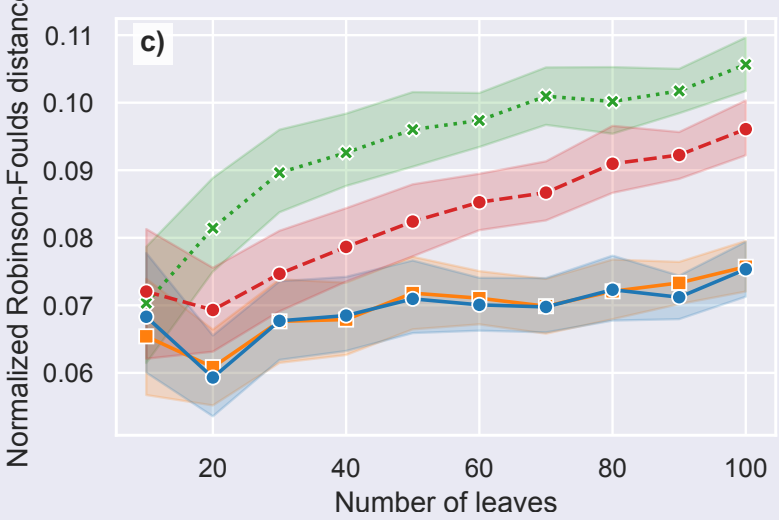
- Predict one number for each residual.
- Pool across sites to obtain a single value per pair.
- Loss function happens at this level:
  compare to true distance on simulated data.

We then use a distance method to build the tree (not end-to-end).

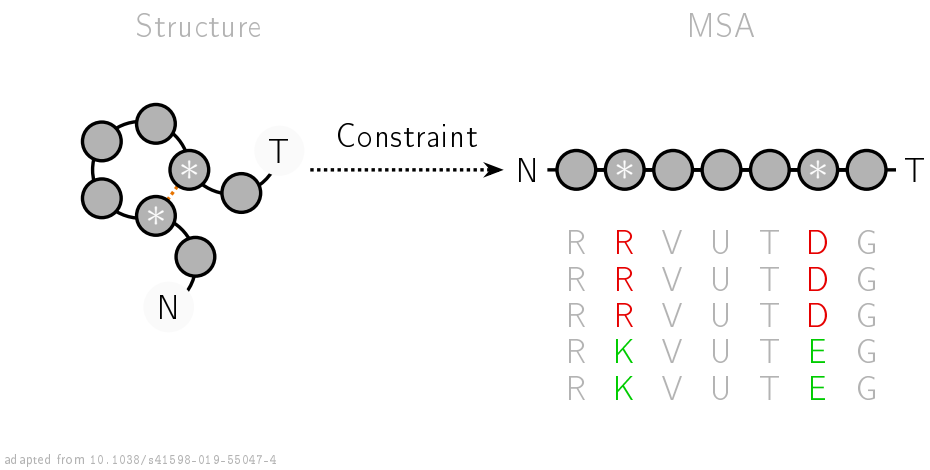
Results - Under LG+GC model, PF performs on par with ML





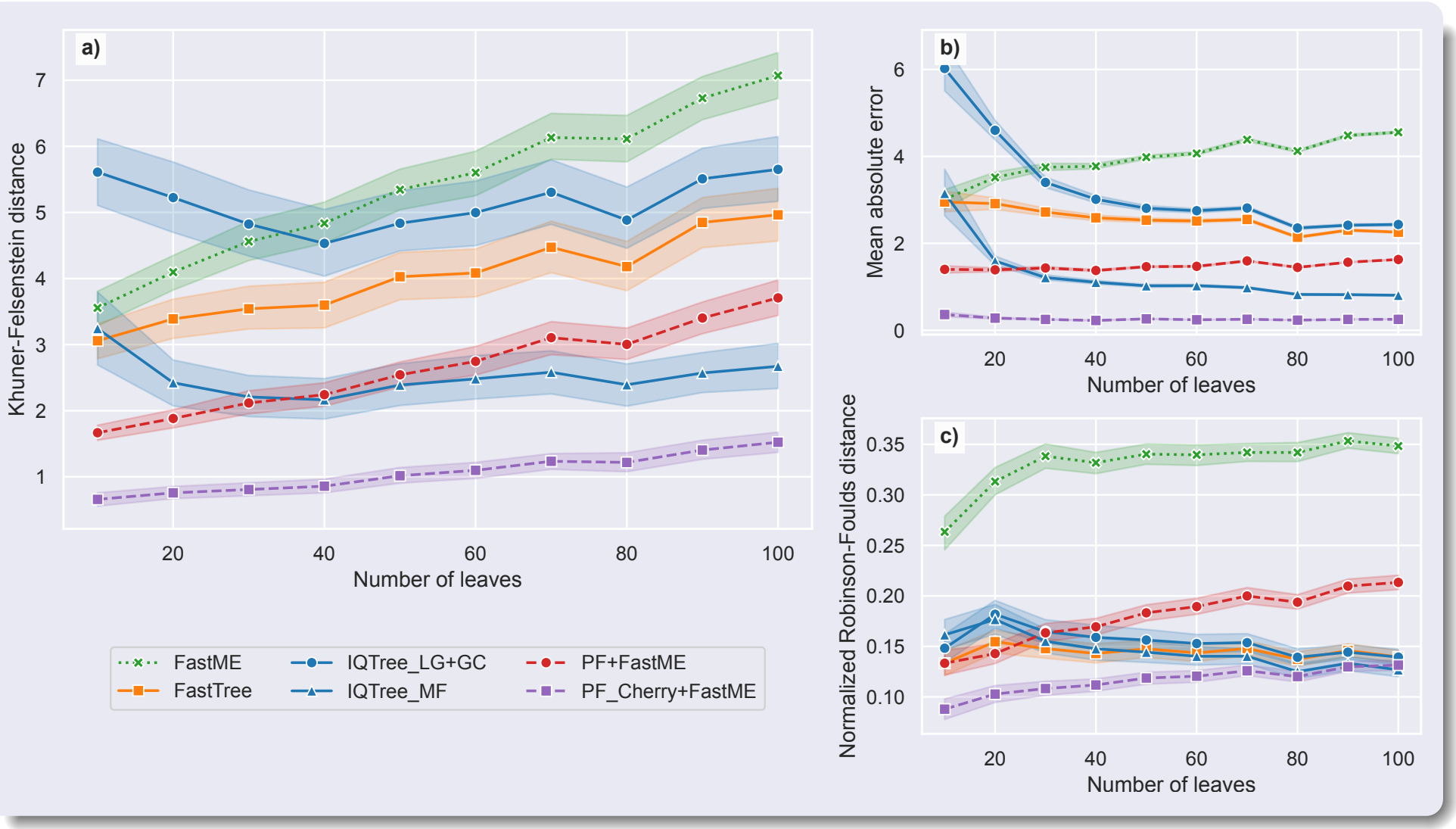


Results - What about a more complex model ?

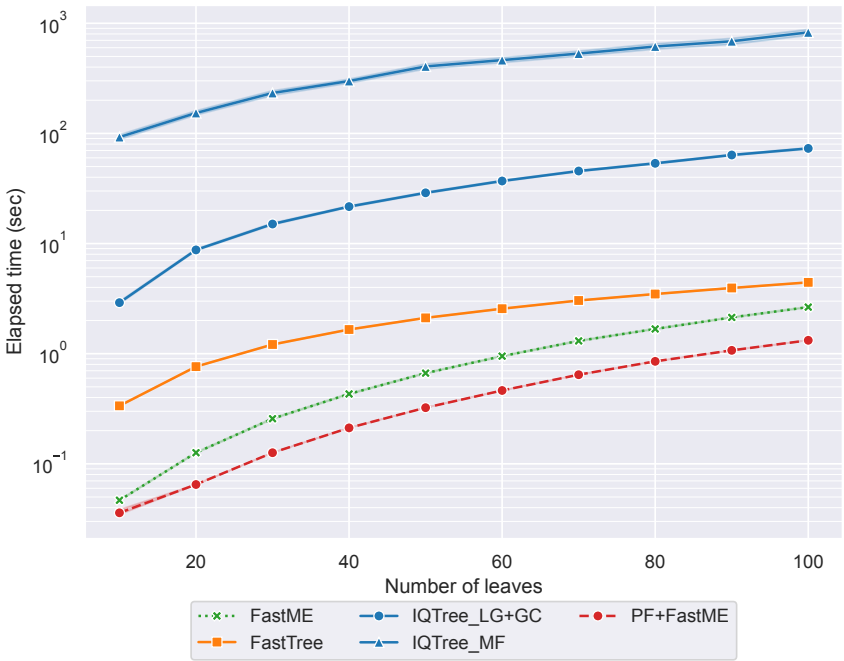


- We **simulate** 250 pairs of **adjacent co-evolving sites**
- We use a 400×400 substitution **matrix** to describe residue **co-evolution**, from **CherryML**
- Most **ML** methods would consider **sites independent**

Results - Under a co-evolution model, PF performs the best



Results - Inference speed

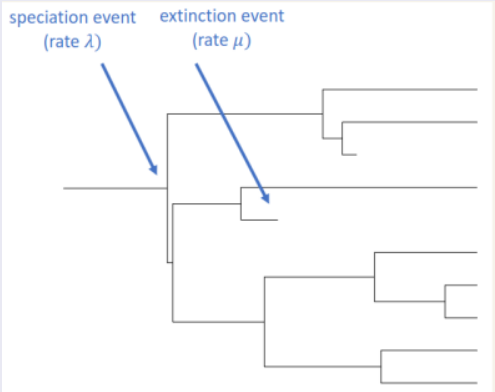


- **Phyloformer** is the **fastest** method
- Phyloformer is even **faster than FastME** on its own
- Inference **speed** is **independent** from model **complexity**

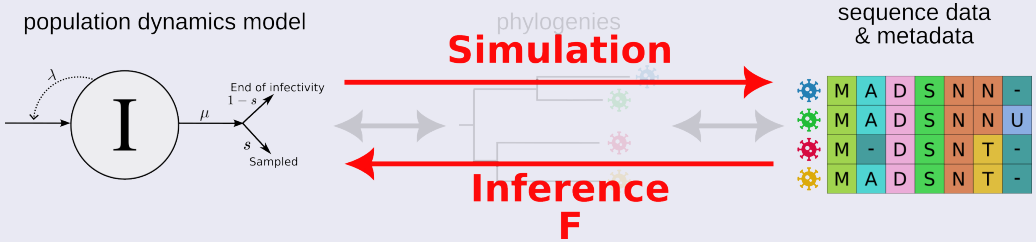
Phylodynamics: evolutionary parameter inference

Phylodynamics vs Phylogenetics

- So far we have sampled trees from a **parameterized distribution**.
- These parameters themselves have a meaning in
  - epidemiology (R_0, duration),
  - ecology (biodiversification).



Phylodynamics from sequences (skip the tree)



- Existing likelihood-free phylodynamics methods start from phylogenies.
- Skipping the tree: faster, handles phylogenetic uncertainty and cases where there is no tree (e.g. recombination).

Differences with Phyloformer

Posterior inference on $(R_0, \text{duration})$ with quantile regression

- Reminder: $\arg\min_m \sum_i |m - R_0^i|$ estimates the median of $p(R_0)$.
- We are interested in the *conditional* median of $p(R_0|\text{sequence})$.
- Our network m_θ minimizes $\arg\min_\theta \sum_i |m_\theta(\text{sequence}_i) - R_0^i|$.
- Generalizes to other quantiles with the pinball loss (asymetric).

Accounting for dates

- In epidemiology, we have (and need) dated sequences.
- We incorporate this information through positional encodings.

Permutation invariance vs equivariance

- We want a single prediction per MSA, not per pair.
- We don't form pairs (better scaling).
- We use special CLS tokens for global pooling.

Transformers for EpiDemiological DYnamics (TEDDY)

Setting

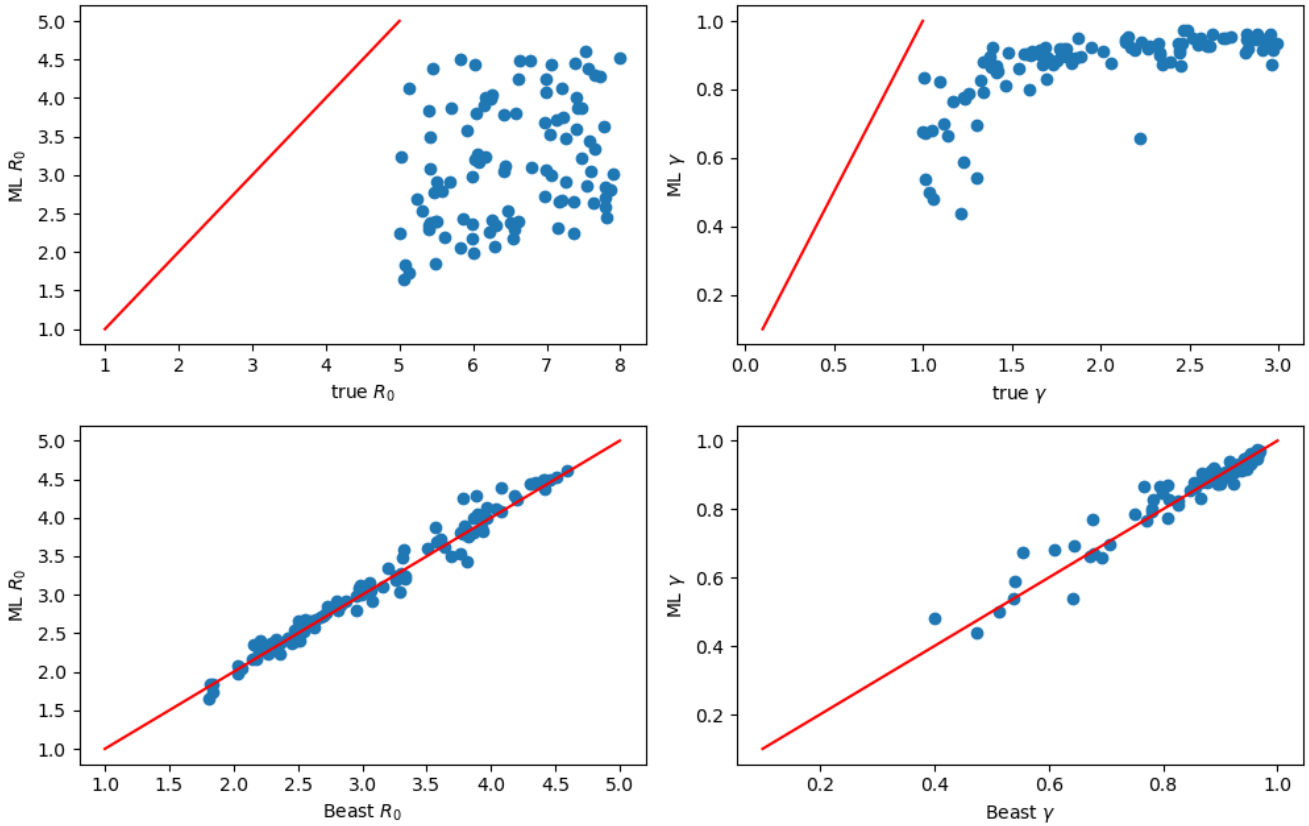
- Sample $R_0 \sim \mathcal{U}(1, 5)$ and duration $\sim \mathcal{U}(0.1, 1)$.
- Then 50-leave trees from birth-death(R_0, duration)
- Then 1000-long sequences from these trees.

Parameter	BEAST2	Teddy (ours)
R_0	0.18	0.18
duration	0.25	0.26
Time for 1000 runs	17 days	50s

- Same relative errors as BEAST2 (SOTA), 1e5 x faster.
- 95% credible intervals correctly estimated in both cases.

(Non-)robustness to strong prior misspecification

- Network trained on $R_0 \in [1, 5] \times \gamma \in [0.1, 1]$.



- Performs poorly on data where $R_0 \in [5, 8] \times \gamma \in [1, 3]$.
- But behaves exactly like BEAST2.

Perspective

End-to-end from sequences to the tree

- Include the alignment step in the network.
- Output a tree.

Both extensions would require some work on differentiability of the corresponding steps.

Use neural inferance to build sampling distributions

- Similar idea as GANs, VAEs.
- Amounts to constraining the marginal $p(\text{MSA})$.
- Also requires differentiability.