Feature Learning for Image Data: from Dictionary Learning to Deep Learning

Jeremy Watt
jermwatt@gmail.com

Reza Borhani
borhani@u.northwestern.edu

Today's talk overview

Part I. An overview of features / unsupervised feature learning- principles of feature engineering, unsupervised feature learning, dictionary learning / sparse coding

Part II. Supervised feature learning – elements of function approximation, neural networks and fixed basis kernels, learning features for regression and classification

Part III. Deep learning and nonlinear optimization – a host of reasons, convolutional networks, a primer on the backpropogation algorithm

Part II

Regression as continuous function approximation

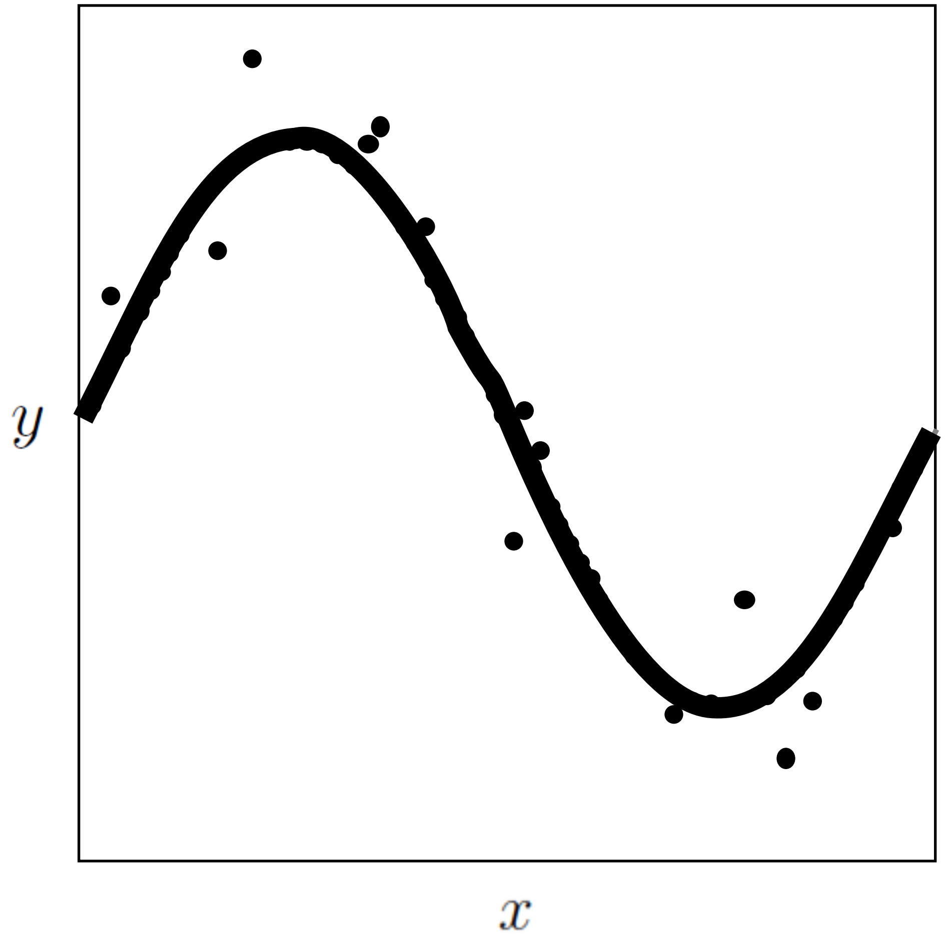
- A typical/realistic regression dataset

- The underlying continuous data-generating function is a sinusoidal in this case. Our goal is to approximate this function

- Realistic data is always noisy but lets momentarily assume that data is clean (i.e., noiseless)

- Still infinitely-many potential continuous functions that go through all these samples. Lets increase the samples!

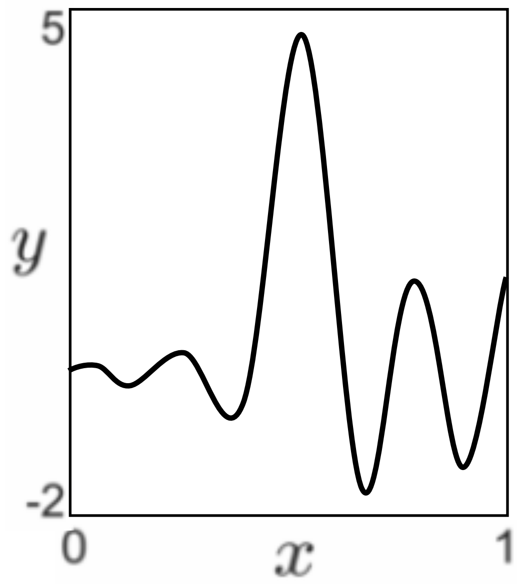
- As the number of samples goes to infinity, the problem of regression reduces to the classic problem of *continuous function approximation*



Continuous function approximation

what we have: $\{(x,\, y(x))\}$ for all x

what we want: An approximate functional form for $y\,(x)$



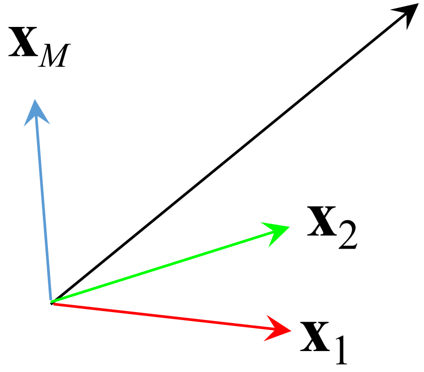
The functional form is found via decomposing $y\,(x)$ over a given basis as

$$y\,(x) \approx \sum_{m=0}^{M} f_m\,(x)\,w_m$$

This idea is closely related to the concept of representing a given vector (i.e. a discrete function) over a set of basis vectors

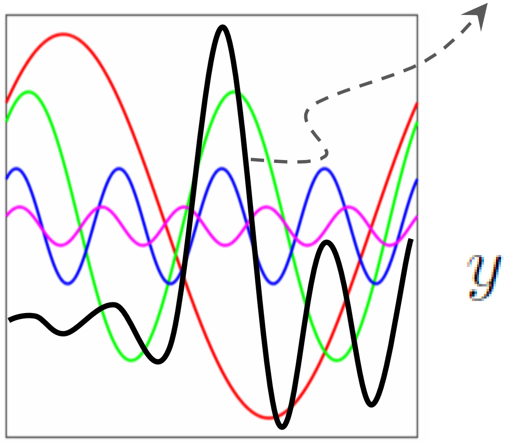
Vector vs. continuous function approximation

$\mathbf{y}$ (P dimensional)



$$\mathbf{y} \approx \sum_{m=1}^{M} \mathbf{x}_m w_m = \overset{\mathbf{x}_1}{\left[\;\cdot\ \mathrm{w}_1\right]} + \overset{\mathbf{x}_2}{\left[\;\cdot\ \mathrm{w}_2\right]} + \dots + \overset{\mathbf{x}_M}{\left[\;\cdot\ \mathrm{w}_M\right]}$$

$y\left(x\right)$ (∞ dimensional!)



$$y\left(x\right) \approx \sum_{m=0}^{M} f_m\left(x\right) w_m = \overset{f_0\left(x\right)}{\left[\;\cdot\ \mathrm{w}_0\right]} + \overset{f_1\left(x\right)}{\left[\;\cdot\ \mathrm{w}_1\right]} + \dots + \overset{f_M\left(x\right)}{\left[\;\cdot\ \mathrm{w}_M\right]}$$

Common bases for continuous function approximation

1. Fixed bases

- Polynomial basis
- Fourier basis

- Consists of monomials of varying degree
- Taylor expansion of a function also uses these basis functions

- Consists of sine and cosine waves of varying frequency
- Named after its inventor Joseph Fourier who used these basis functions in the early 1800s to study heat diffusion

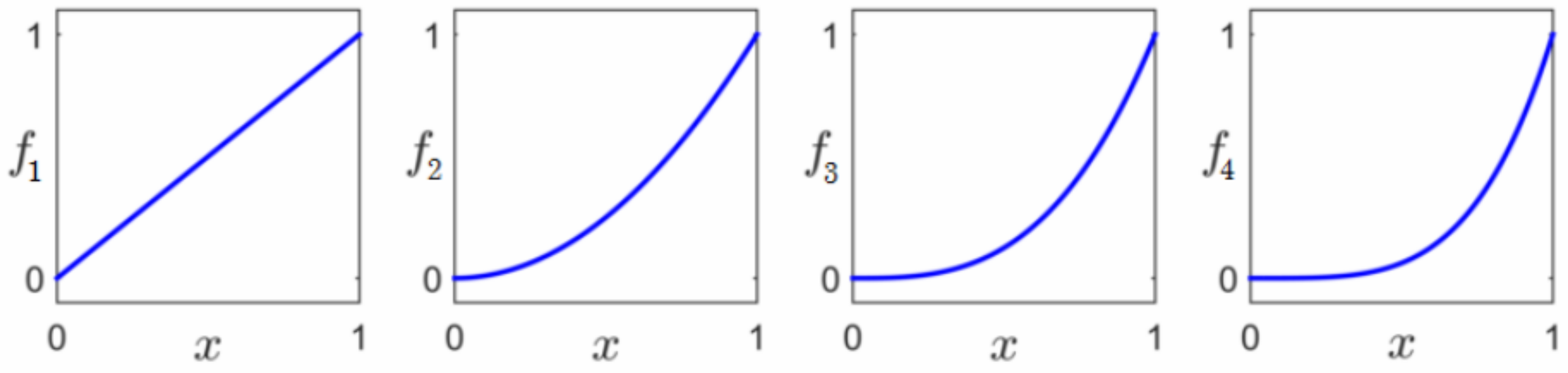
2. Adjustable bases

- Feed-forward neural network basis

- The name is coined by neuroscientists in the late 1940s who used this sort of basis to roughly model how the human brain processes information

Common bases for continuous function approximation

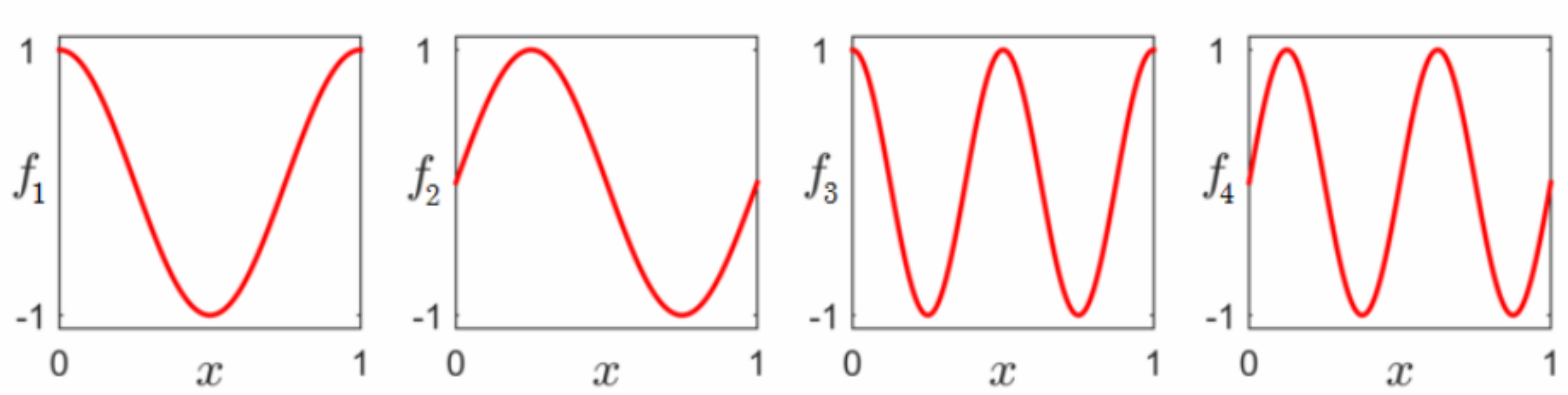
1. Fixed bases
   - Polynomial basis
   - Fourier basis
2. Adjustable bases
   - Feed-forward neural network basis



$$f_0(x) = 1$$
$$f_m(x) = x^m \quad \text{for all } m \geq 1$$

Common bases for continuous function approximation

1. Fixed bases
   - Polynomial basis
   - Fourier basis
2. Adjustable bases
   - Feed-forward neural network basis

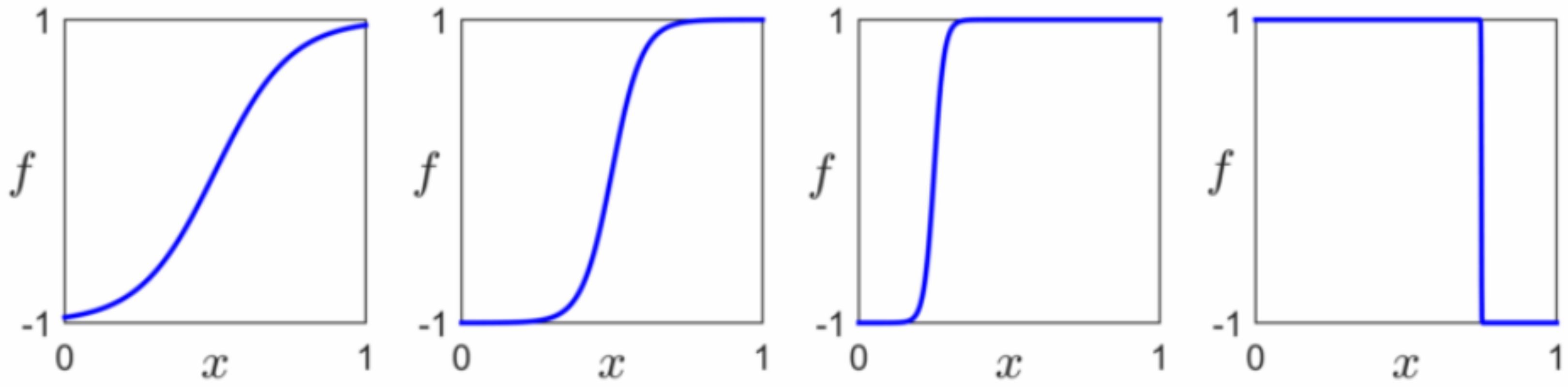


$$
\begin{aligned}
&f_0(x) = 1 \\
&f_{2m-1}(x) = \cos(2\pi m x) \qquad \text{for all } m \geq 1 \\
&f_{2m}(x) = \sin(2\pi m x) \qquad \text{for all } m \geq 1
\end{aligned}
$$

Common bases for continuous function approximation

1. Fixed bases
   - Polynomial basis
   - Fourier basis

2. Adjustable bases
   - Feed-forward neural network basis
     - Single hidden-layer network



$$f_0(x) = 1$$
$$f_m(x) = a(c_m + xv_m) \quad \text{for all } m \geq 1$$

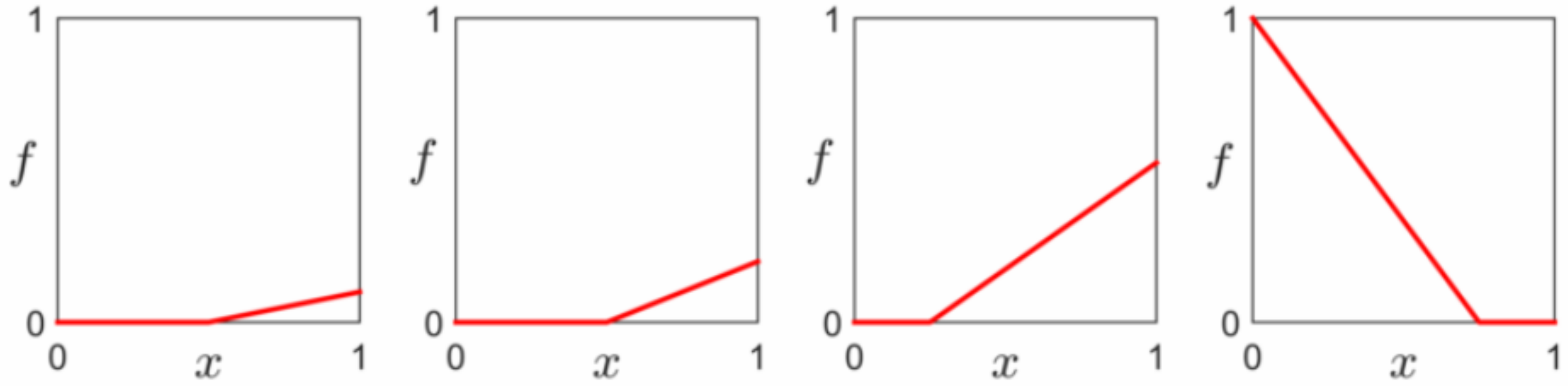
'tanh' activation

$$a(\cdot) = \tanh(\cdot)$$

Common bases for continuous function approximation

1. Fixed bases
   - Polynomial basis
   - Fourier basis

2. Adjustable bases
   - Feed-forward neural network basis
     - Single hidden-layer network



$$\boxed{\begin{aligned} f_0(x) &= 1 \\ f_m(x) &= a(c_m + x v_m) \quad \text{for all } m \geq 1 \end{aligned}}$$

'max' or 'rectified linear unit' activation

$$a(\cdot) = \max(0, \cdot)$$

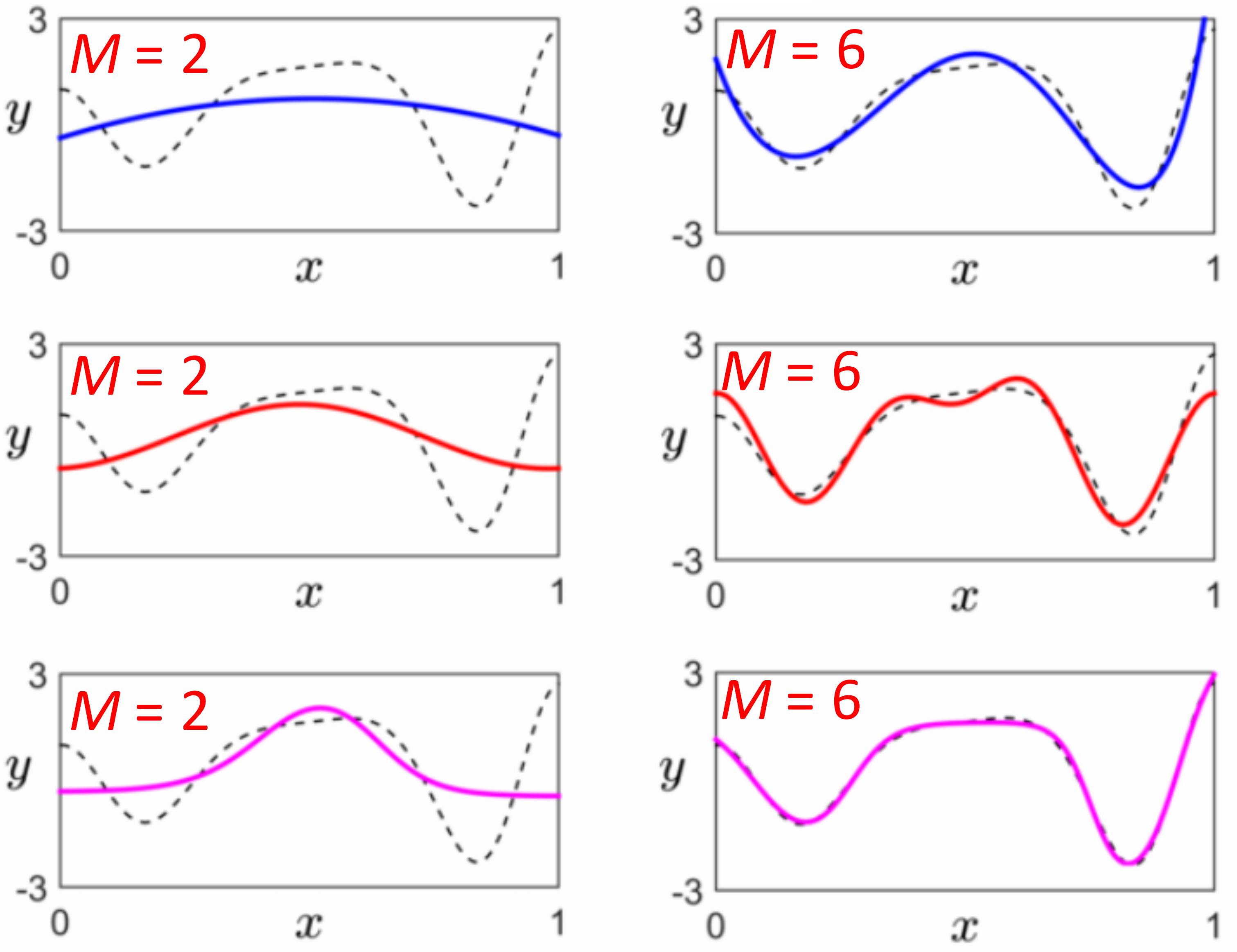
Common bases for continuous function approximation

$$y(x) \approx \sum_{m=0}^{M} f_m(x)\, w_m$$

Polynomial basis

Fourier basis

Single hidden-layer basis
(with 'tanh' activation)



What about vector-valued inputs?

	Scalar x	Vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$
Approximation	$y(x) \approx \sum_{m=0}^{M} f_m(x)\, w_m$	$y(\mathbf{x}) \approx \sum_{m=0}^{M} f_m(\mathbf{x})\, w_m$
Polynomial	$f_m(x) = x^m$	$f_m(\mathbf{x}) = x_1^{m_1} x_2^{m_2} \cdots x_N^{m_N}$
Single hidden-layer NN	$f_m(x) = a(c_m + x v_m)$	$f_m(\mathbf{x}) = a\left(c_m + \mathbf{x}^T \mathbf{v}_m\right)$

From single hidden-layer to multi hidden-layer bases

- Flexibility of the single hidden-layer basis functions is gained by introduction of adjustable internal parameters c_m and $\mathbf{v}_m$

$$f_m(\mathbf{x}) = a\left(c_m + \mathbf{x}^T\mathbf{v}_m\right)$$

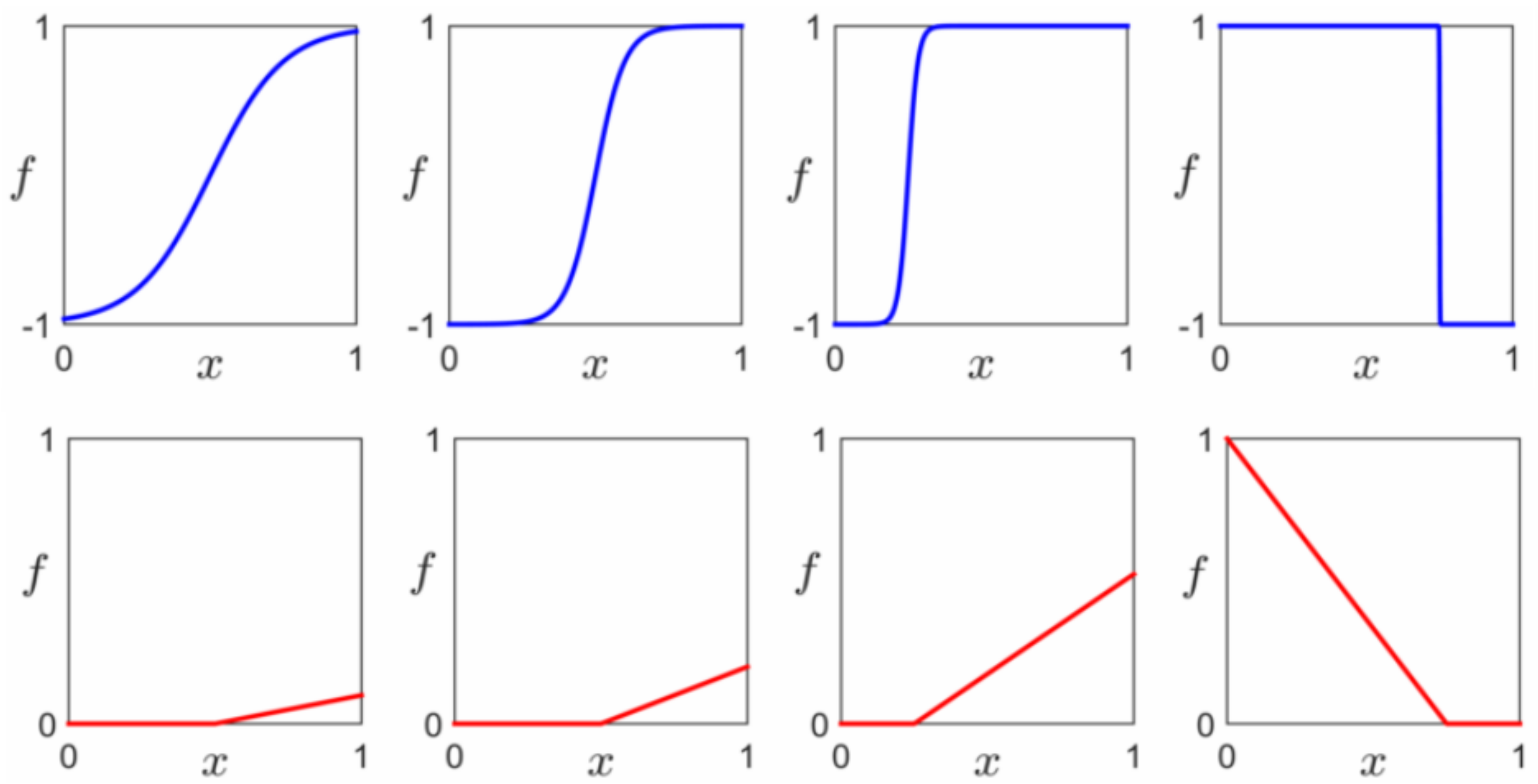
- To create even more flexible basis functions we can compose the activation function with itself, giving *two-hidden layer basis* functions of the form

$$f_m(\mathbf{x}) = a\left(c_m^{(1)} + \sum_{m_2=1}^{M_2} a\left(c_{m_2}^{(2)} + \mathbf{x}^T\mathbf{v}_{m_2}^{(2)}\right) v_{m_2,m}^{(1)}\right)$$

Two hidden-layer basis function: even more flexible

‘tanh’ activation

‘max’ activation



Want more flexibility? Keep adding layers!

- The more layers we add, the more flexible each basis element becomes, e.g., a 3-hidden layer basis function with ‘max’ activation takes the form

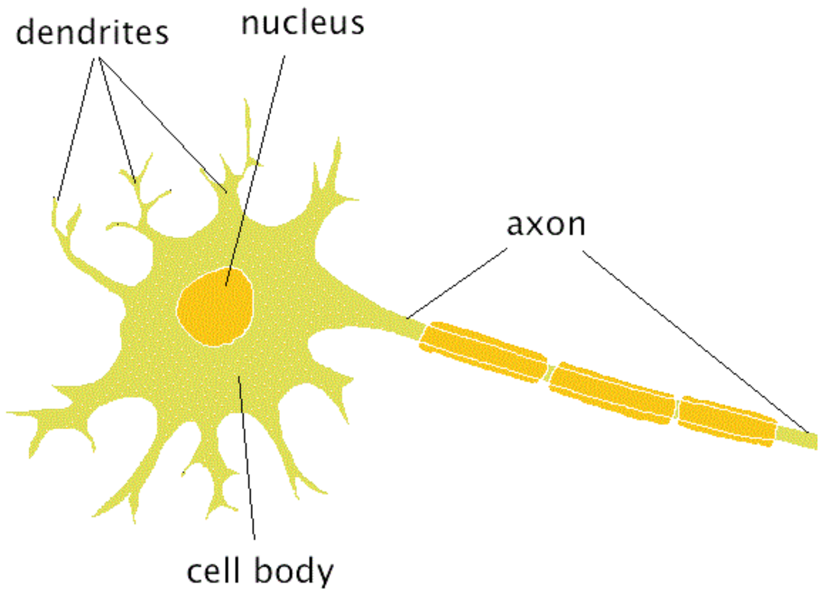
$$f_m\left(\mathbf{x}\right) = \max\left(0,\ c_m^{(1)} + \sum_{m_2=1}^{M_2} \max\left(0,\ c_{m_2}^{(2)} + \sum_{m_3=1}^{M_3} \max\left(0,\ c_{m_3}^{(3)} + \mathbf{x}^T\mathbf{v}_{m_3}^{(3)}\right) v_{m_3,m_2}^{(2)}\right) v_{m_2,m}^{(1)}\right)$$

- This composition procedure can be repeated to arrive at a general *L-hidden layer basis*

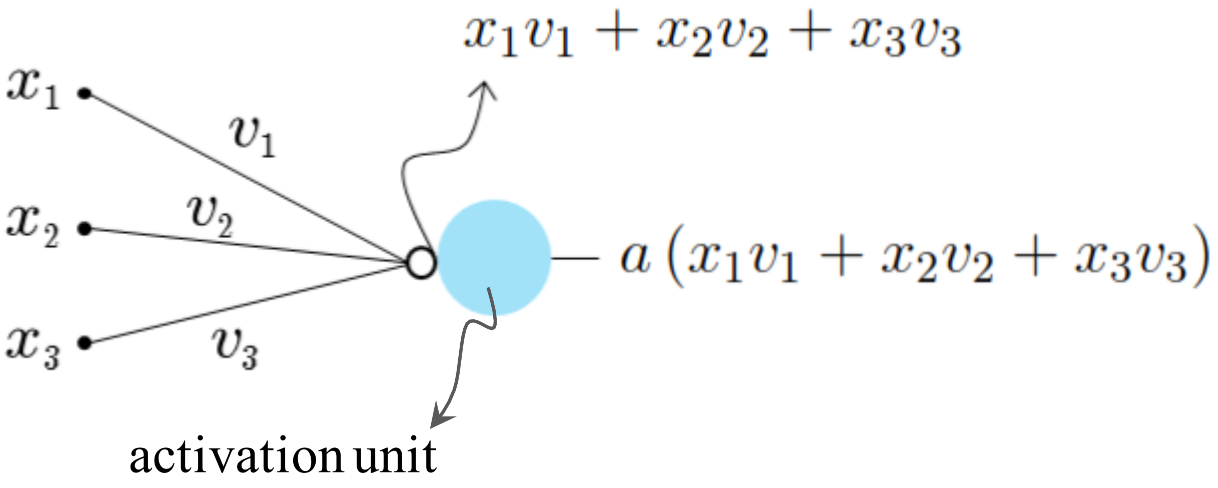
- A basis with *L*>2 or 3 hidden-layers is usually called a **deep network**

Graphical representation of a neural network

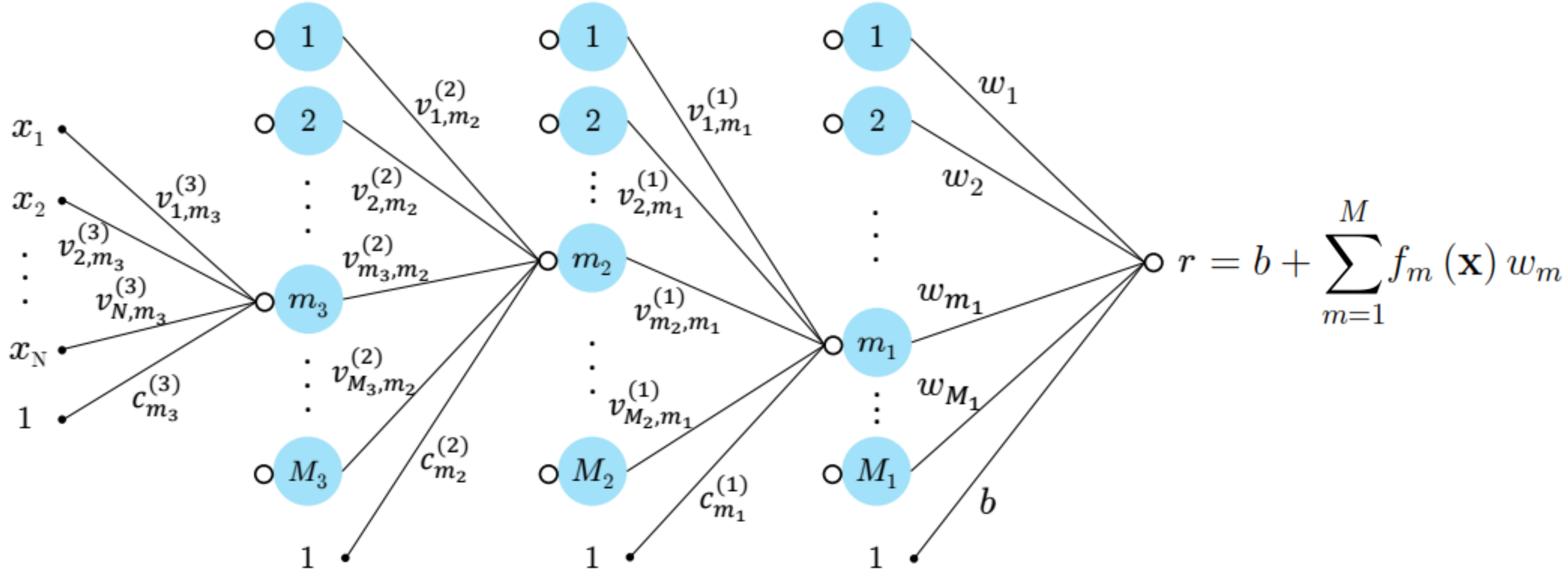
A biological neuron:



An artificial neuron:



Graphical representation of a neural network



$$f_m(\mathbf{x}) = \max\left(0,\ c_m^{(1)} + \sum_{m_2=1}^{M_2} \max\left(0,\ c_{m_2}^{(2)} + \sum_{m_3=1}^{M_3} \max\left(0,\ c_{m_3}^{(3)} + \mathbf{x}^T \mathbf{v}_{m_3}^{(3)}\right) v_{m_3,m_2}^{(2)}\right) v_{m_2,m}^{(1)}\right)$$

How do we tune the weights?

$$y(\mathbf{x}) \approx \sum_{m=0}^{M} f_m(\mathbf{x})\, w_m$$

?

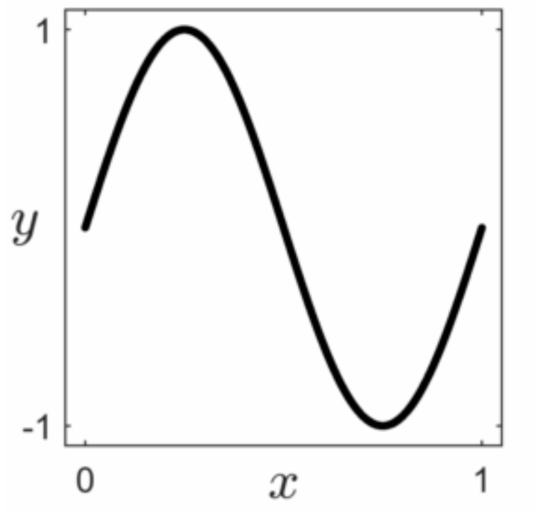
Recovering the weights

$$y(\mathbf{x}) \approx \sum_{m=0}^{M} f_m(\mathbf{x})\, w_m$$

- There is a continuous Least Squares integral problem one can formulate to at least, in theory, recover weights (see [29])
- However this problem is highly intractable esp. when using neural net bases
- One can approximate this integral by discretizing all the continuous functions involved

Recovering the weights

Pure function approximation

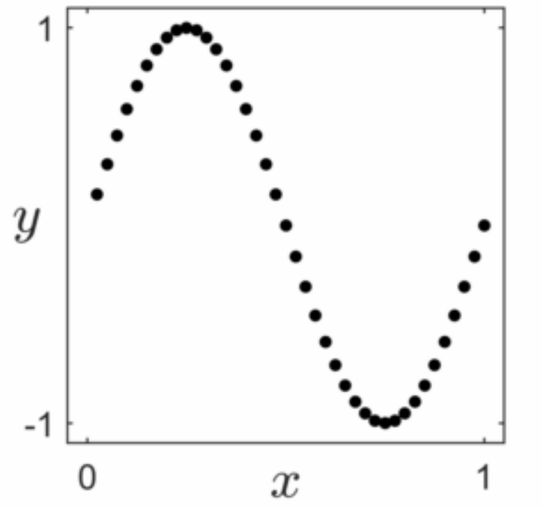


- *infinite* access to data
- data is clean

$$\sum_{m=0}^{M} f_m(\mathbf{x})\, w_m \approx y(\mathbf{x})$$

for all $\mathbf{x}$ in domain

Ideal regression dataset

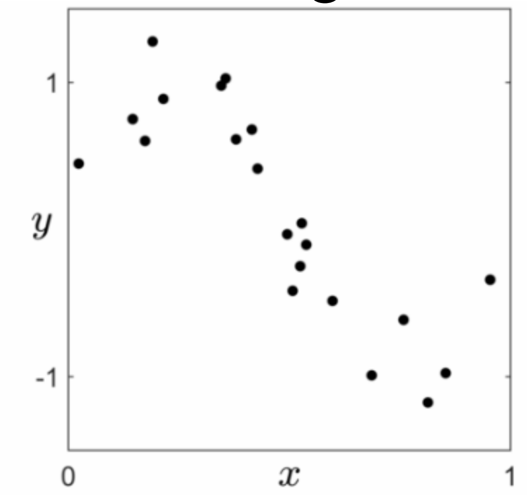


- P is relatively large
- samples are evenly distributed
- data is clean

$$\sum_{m=0}^{M} f_m(\mathbf{x}_p)\, w_m \approx y(\mathbf{x}_p)$$

for $p = 1 \ldots P$

Realistic regression dataset



- P is relatively small
- samples are ***not*** evenly distributed
- data is ***noisy*** $y_p = y(\mathbf{x}_p) + \epsilon_p$

$$\sum_{m=0}^{M} f_m(\mathbf{x}_p)\, w_m \approx y_p$$

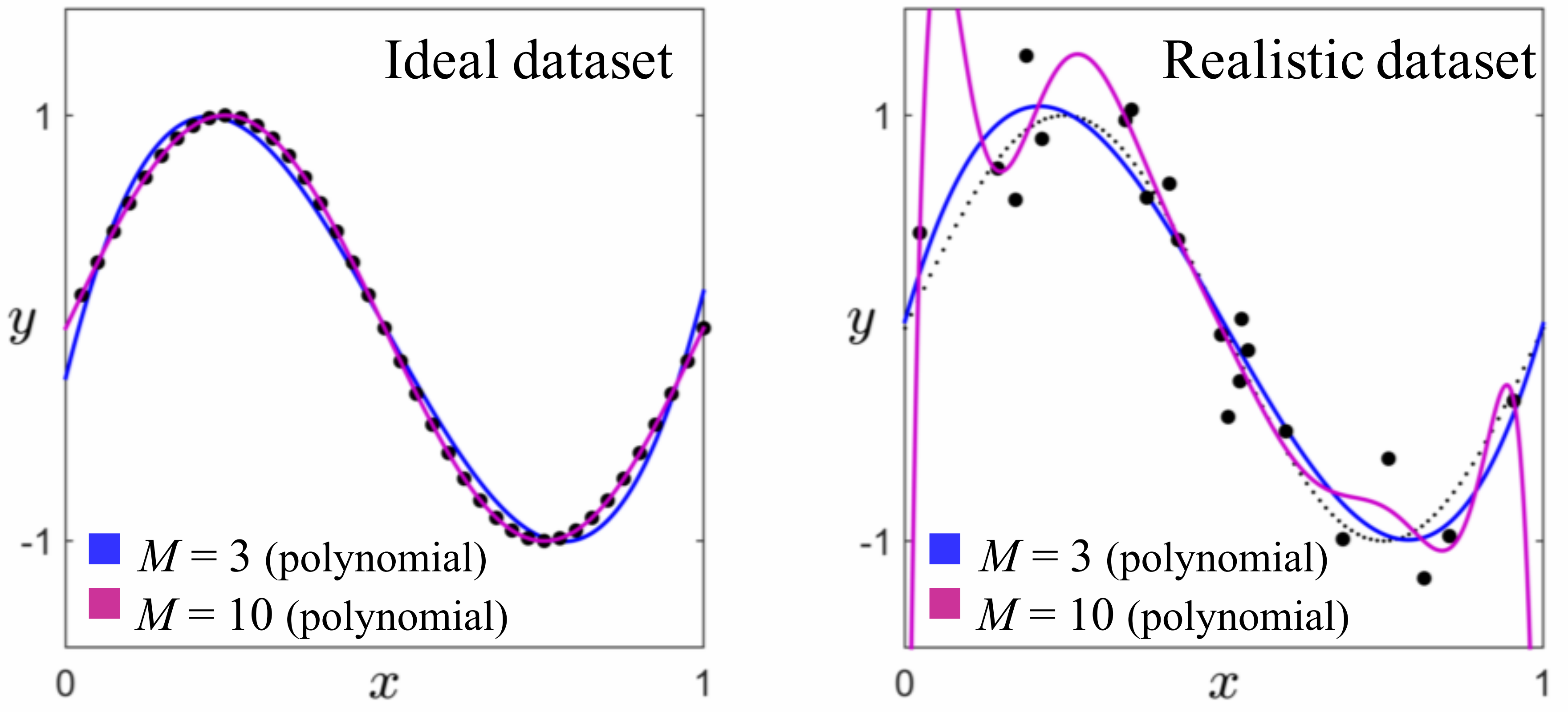
for $p = 1 \ldots P$

How to choose M?

?

$$y(\mathbf{x}) \approx \sum_{m=0}^{M} f_m(\mathbf{x})\, w_m$$

Choice of *M*



- With realistic data, increasing M could *worsen* our approximation of the underlying function!

In search of a sweet spot for *M*

- When M is too small, the corresponding model will be too rigid and inflexible to effectively approximate the underlying phenomenon → *underfitting*

- When M is too large, the corresponding will be needlessly complicated resulting in a very close fit to our noisy data → *overfitting*

- When M is chosen appropriately, the resulting model will be simple yet flexible enough to explain the underlying phenomenon → *Occam's Razor*

What basis to choose?

$$y(\mathbf{x}) \approx \sum_{m=0}^{M} f_m(\mathbf{x})\, w_m$$

?

Which basis to choose?

Depends entirely on the underlying phenomenon. Even though one could not have a full understanding of the data-generating function, any cues can lead to a particular choice of basis or eliminate potential candidates, e.g. :

- The gravitational phenomenon underlying Galileo's ramp dataset is quadratic in nature, inferring the appropriateness of a *Polynomial basis*

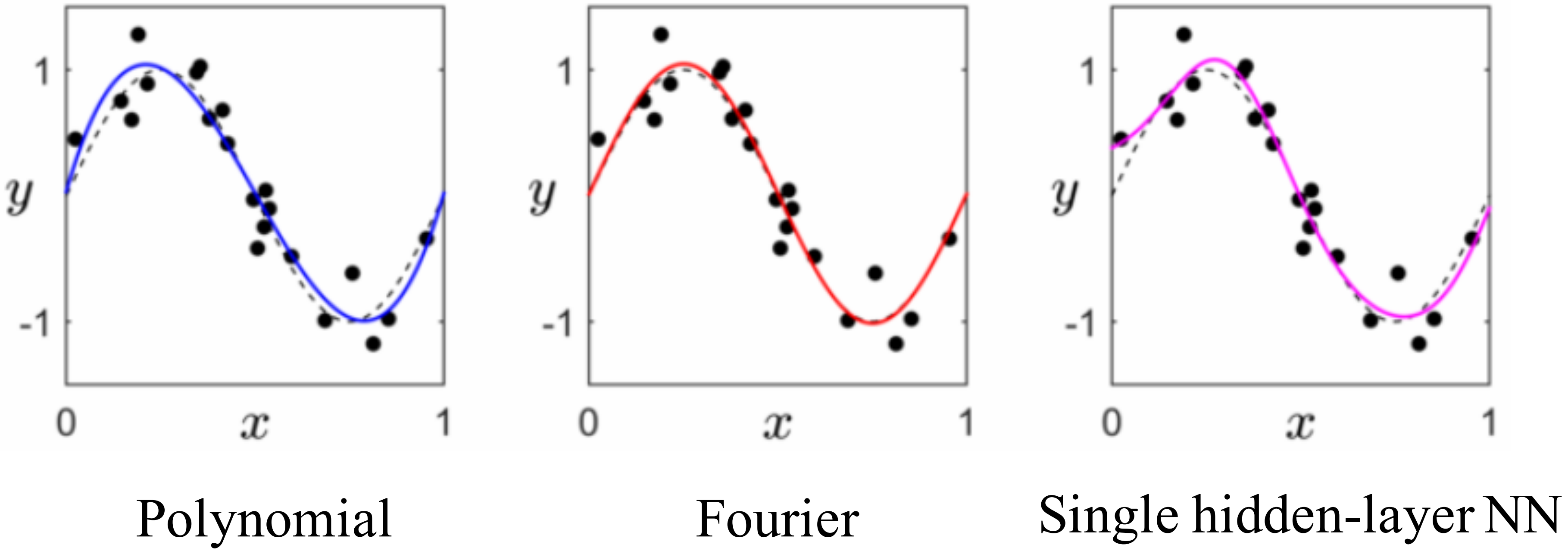
- *Fourier basis* is intuitively appropriate when dealing with periodic phenomena arising in a variety of disciplines including speech processing and financial modeling

- *Neural network bases* are often employed with image and audio data due to their compositional structure as well as a belief in the correspondence of these bases and the way such data is processed by the brain

When the choice of basis is arbitrary

- P is large
- evenly distributed samples
- no noise

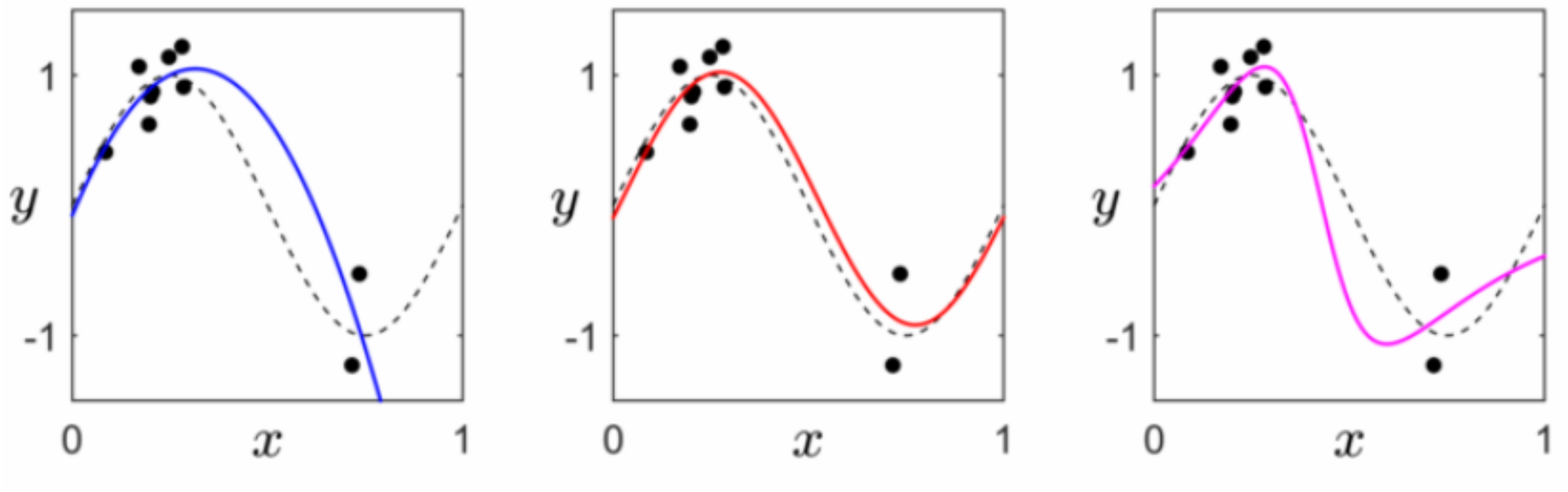
- P is small
- samples distributed very unevenly
- extreme levels of noise



When the choice of basis is arbitrary

- P is large
- evenly distributed samples
- no noise

- P is small
- samples distributed very unevenly
- extreme levels of noise



Polynomial

Fourier

Single hidden-layer NN

Learning features for classification

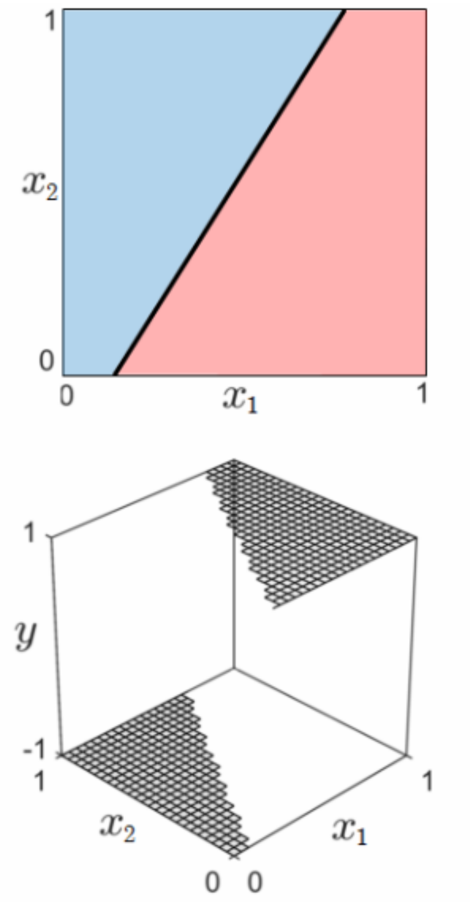
Classification as function approximation

- From the perspective of logistic regression, classification is a binary-output regression problem where the function approximated $y\left(\mathbf{x}\right) \in \{-1, +1\}$

- Because of this, the story with classification is very similar to the one for regression: i.e., the general classification problem is one of function approximation

- The only real difference is that now we approximate piecewise continuous functions using continuous bases like polynomials and neural networks

Classification as function approximation

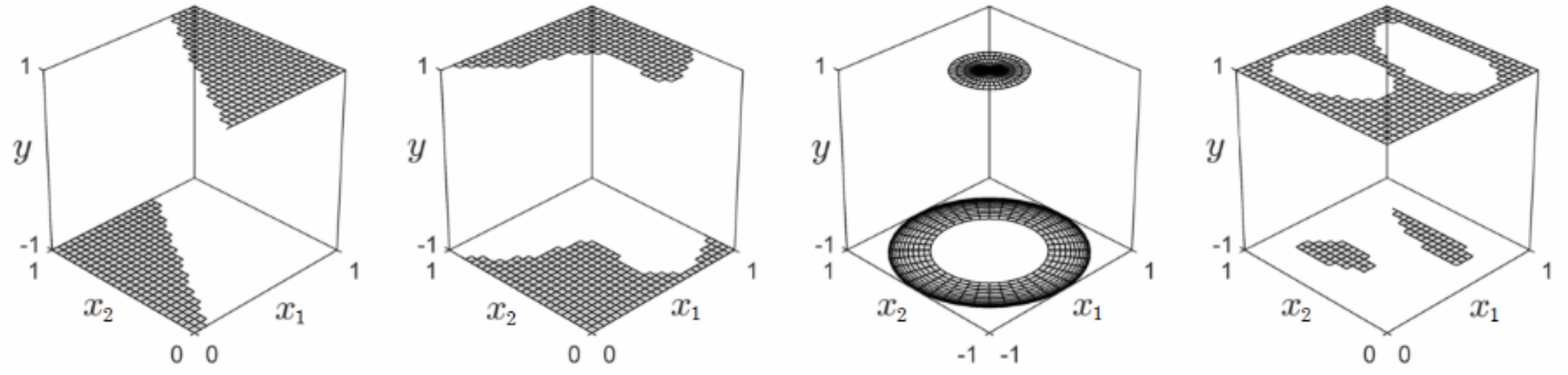


a step function

a generalized step function (an “indicator” function)

Logistic approximation in action

original indicator function $y(\mathbf{x})$

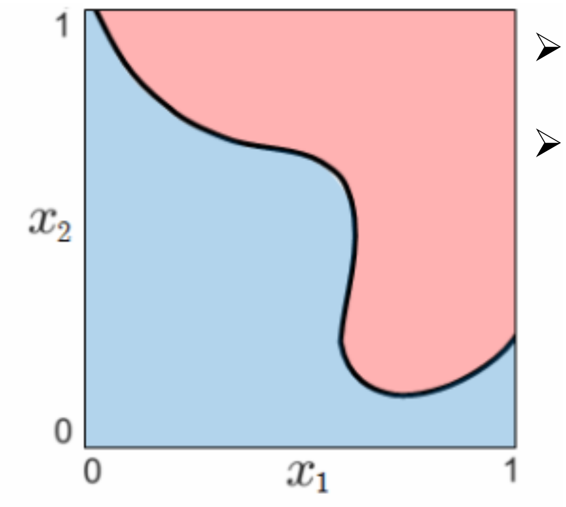


$$\sum_{m=0}^{M} f_m(\mathbf{x})\, w_m \approx y(\mathbf{x})$$

$$\tanh\left(\sum_{m=0}^{M} f_m(\mathbf{x})\, w_m\right) \approx y(\mathbf{x})$$

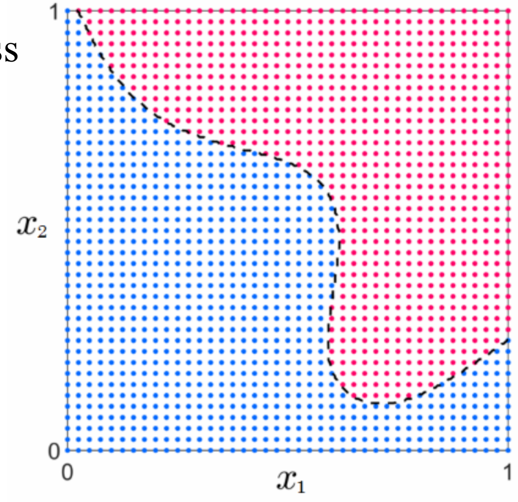
Ideal and realistic classification datasets

Pure function approximation



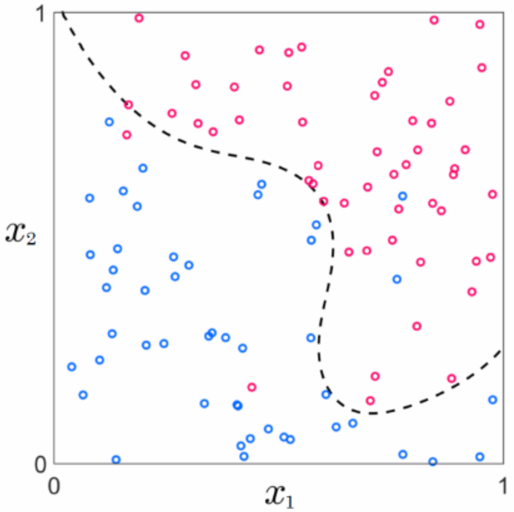
- *infinite* access to data
- data is clean

Ideal classification dataset



- P is relatively large
- samples distributed evenly
- data is clean

Realistic classification dataset



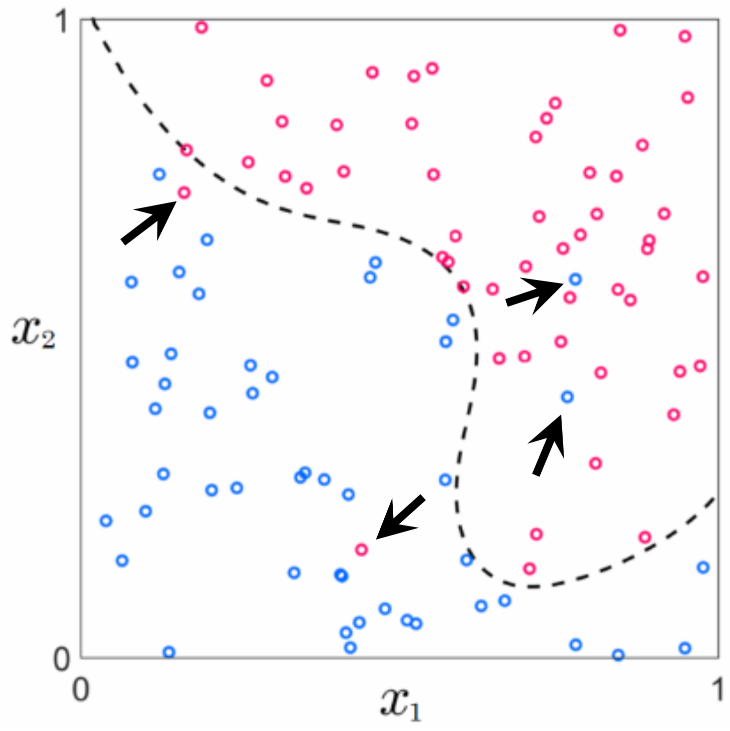
- P is relatively small
- samples are ***not*** evenly distributed
- data is ***noisy***

The general case of two class classification is an (indicator) function approximation problem based on noisy samples of the underlying function.

Noise in classification

Noise in regression: $y_p = y\left(\mathbf{x}_p\right) + \epsilon_p$

Noise in classification: Some data points have been assigned the wrong labels



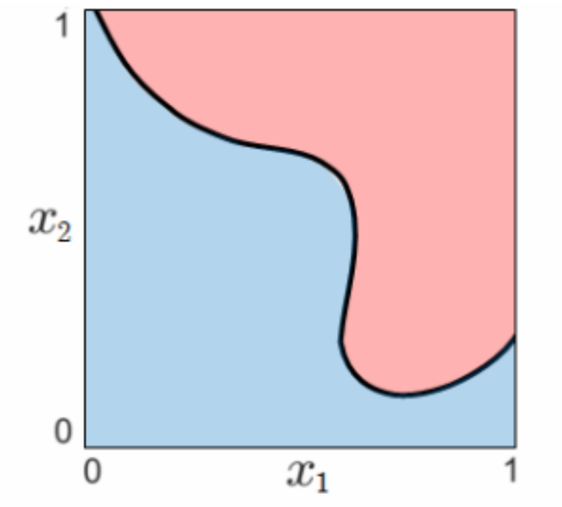
How do we tune the weights?

$$y\left(\mathbf{x}\right) \approx \tanh\left(\sum_{m=0}^{M} f_m\left(\mathbf{x}\right) w_m\right)$$

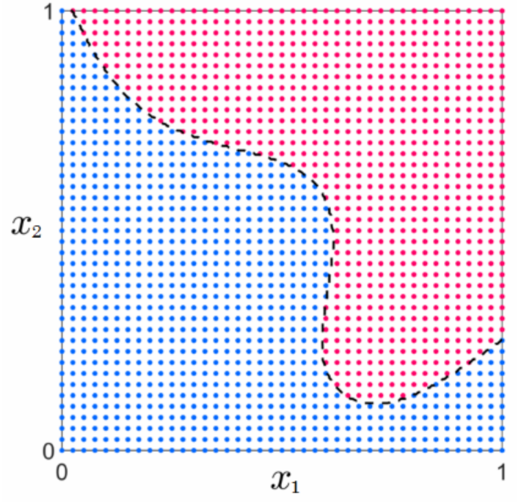
?

Recovering the weights

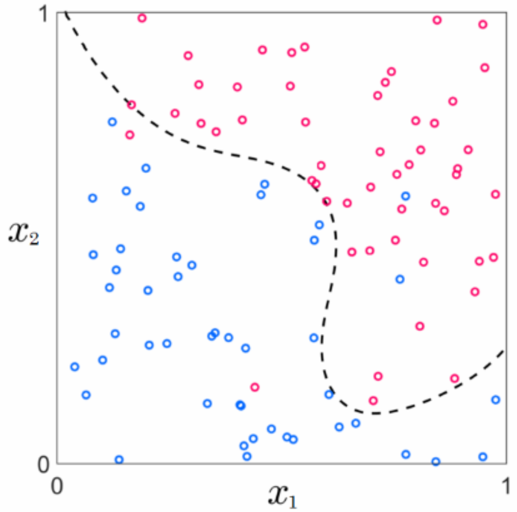
Pure function approximation



Ideal classification dataset



Realistic classification dataset

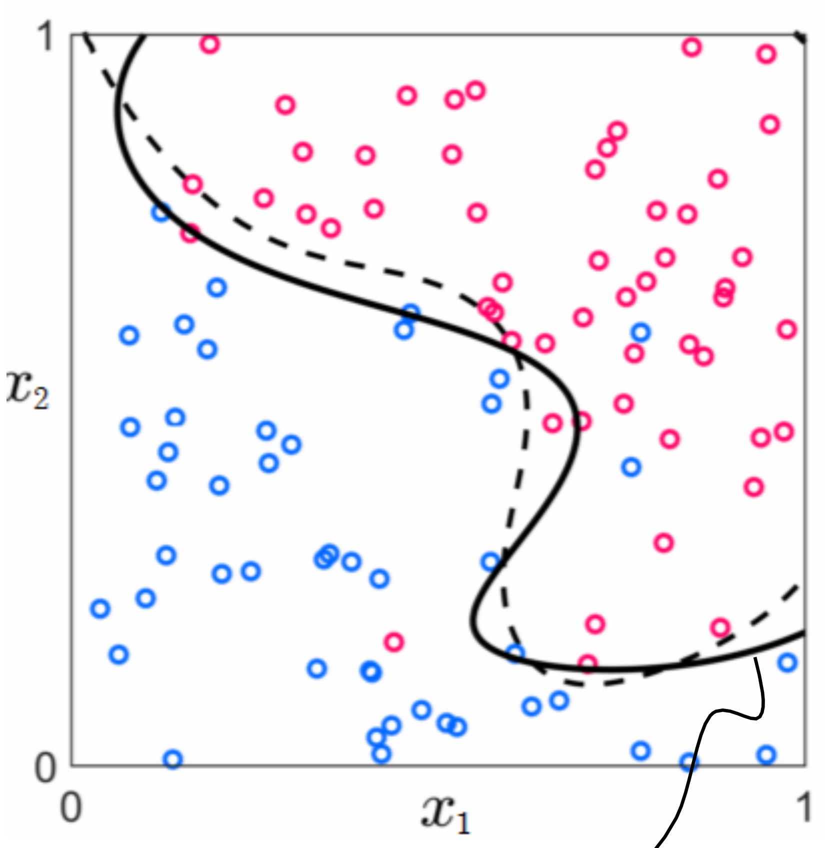


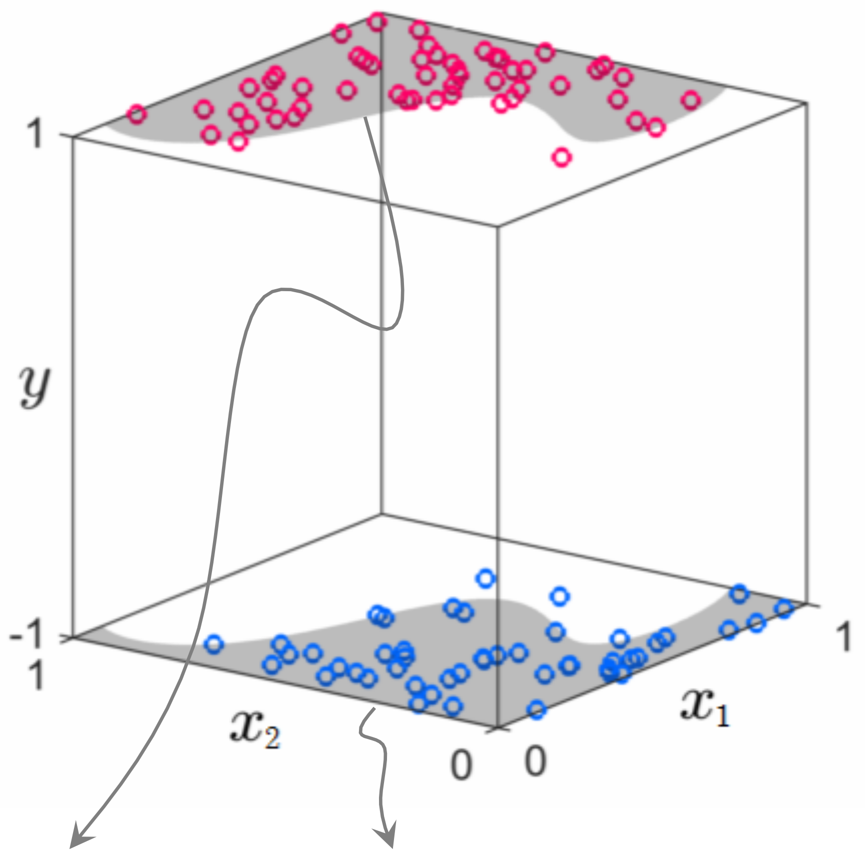
$$\tanh\left(\sum_{m=0}^{M} f_m\left(\mathbf{x}_p\right) w_m\right) \approx y_p$$

A tractable recovery problem can be formed giving a way of finding such parameters (see [29])

Example toy dataset

Feature map: Polynomial



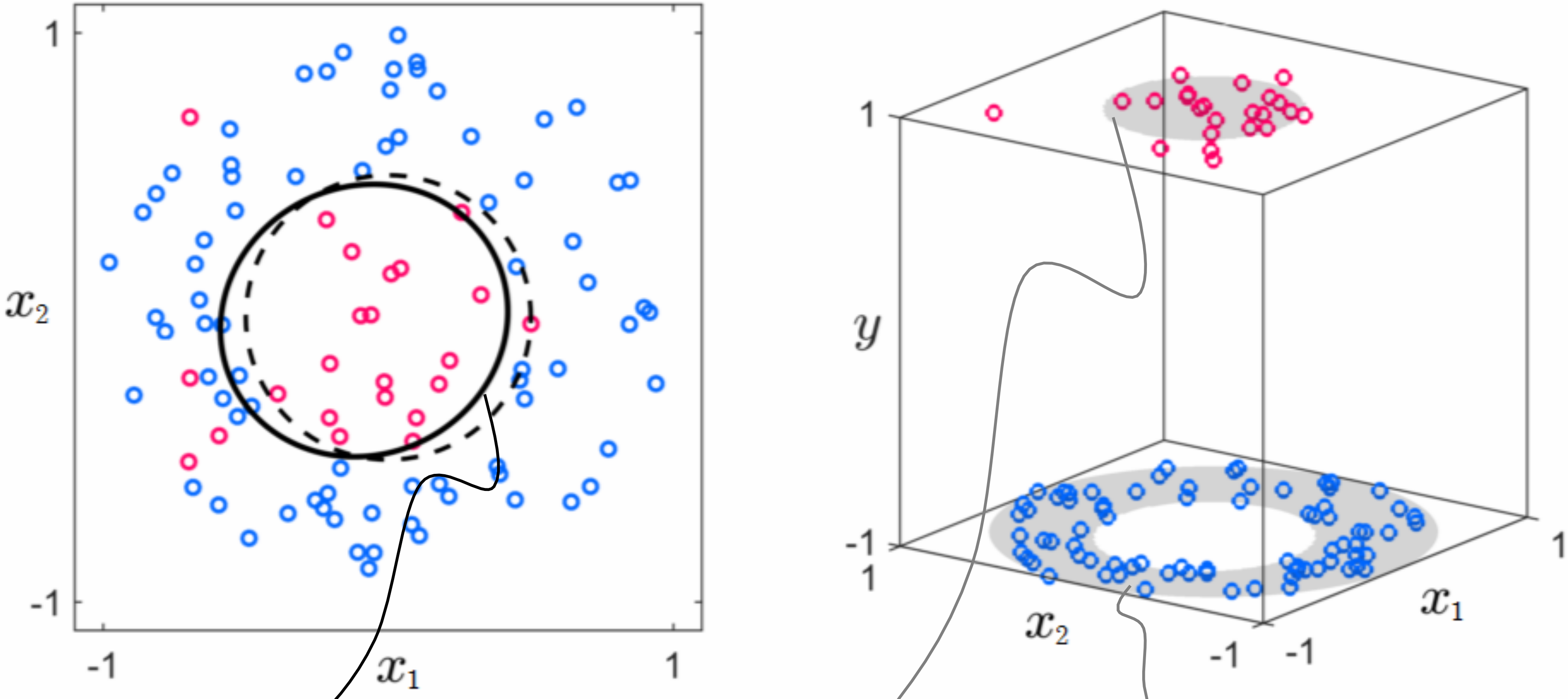


$$\sum_{m=0}^{M} f_m(x)\, w_m = 0$$

$$y(\mathbf{x}) = \operatorname{sign}\left(\sum_{m=0}^{M} f_m(x)\, w_m\right)$$

Example toy dataset

Feature map: Polynomial

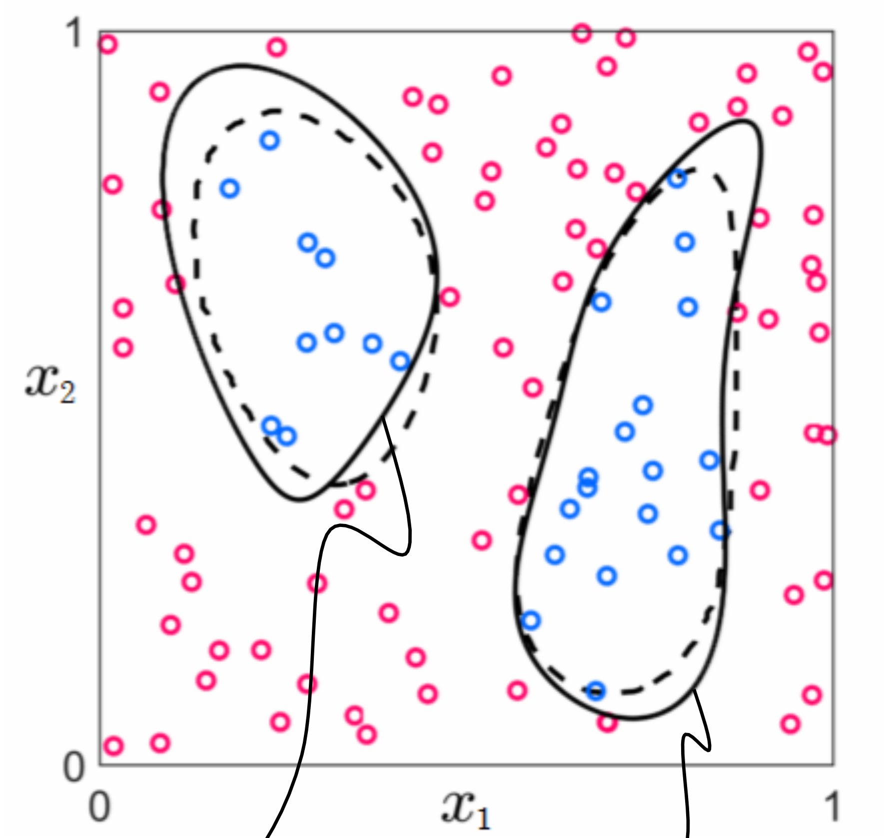


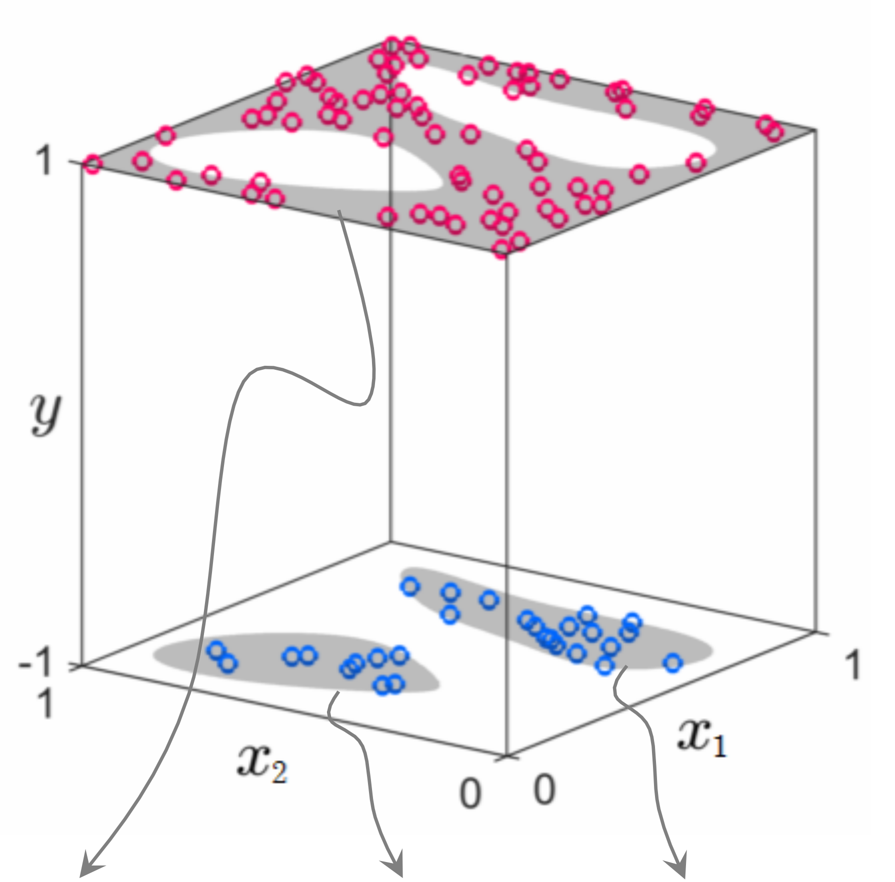
$$\sum_{m=0}^{M} f_m(x)\, w_m = 0$$

$$y(\mathbf{x}) = \operatorname{sign}\left(\sum_{m=0}^{M} f_m(x)\, w_m\right)$$

Example toy dataset

Feature map: Polynomial



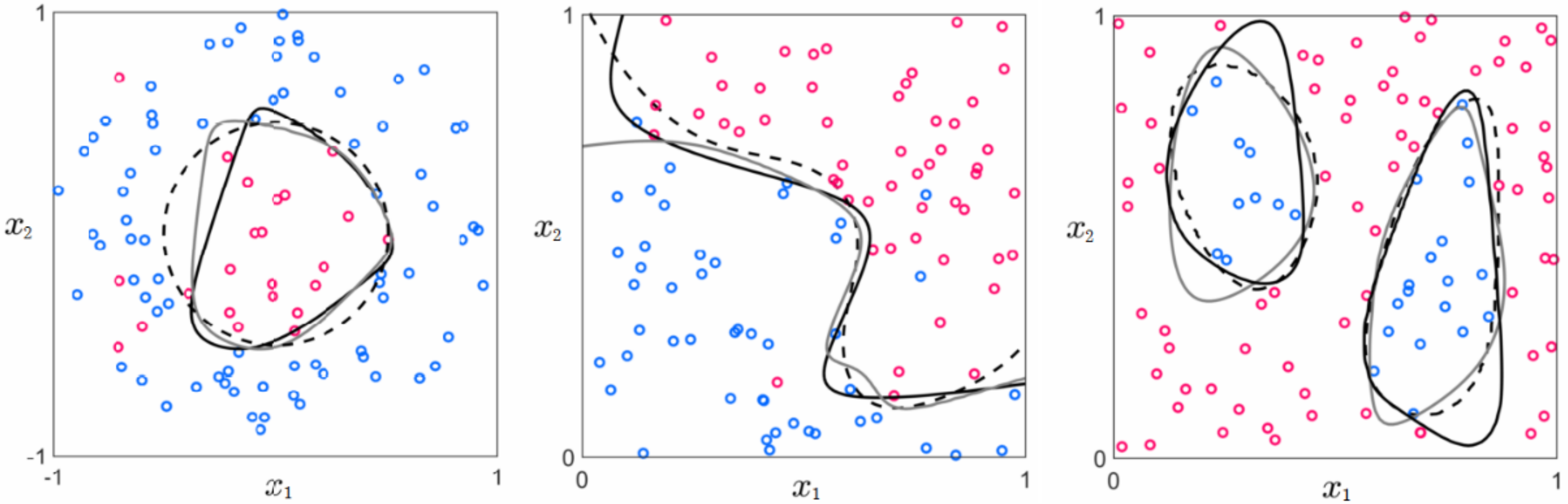


$$\sum_{m=0}^{M} f_m(x)\, w_m = 0$$

$$y(\mathbf{x}) = \mathrm{sign}\left(\sum_{m=0}^{M} f_m(x)\, w_m\right)$$

Example toy dataset

Feature map: single hidden-layer NN

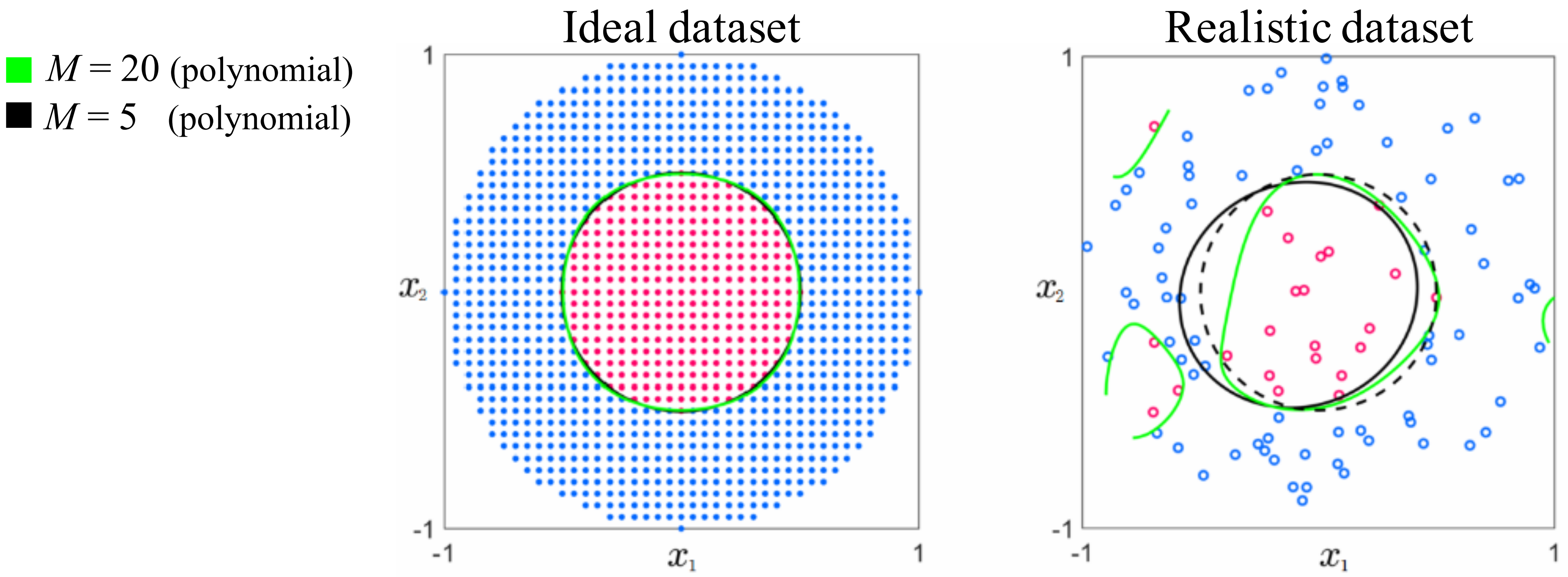


How to choose M?

$$y(\mathbf{x}) \approx \tanh\left(\sum_{m=0}^{M} f_m(\mathbf{x})\, w_m\right)$$

?

Choice of *M*



- With realistic data, increasing M could *worsen* our approximation of the underlying function!
- So again cross-validation is used to determine a proper M and avoid ***overfitting***

What basis to choose?

$$y(\mathbf{x}) \approx \tanh\left(\sum_{m=0}^{M} \overset{?}{f_m}(\mathbf{x})\, w_m\right)$$

Which basis to choose?

- Again, any cues about the underlying data-generating function can lead to a particular choice of basis or eliminate potential candidates

- As with regression, the choice of basis becomes arbitrary if our data has the desired properties of an ideal dataset for classification

  - P is large
  - evenly distributed samples
  - no noise

  - P is small
  - samples distributed very unevenly
  - extreme levels of noise

‘Quality of data’ spectrum

Fixed basis kernels

Combinatorial explosion with fixed basis

A degree D polynomial of dimension N input includes all monomials of the form

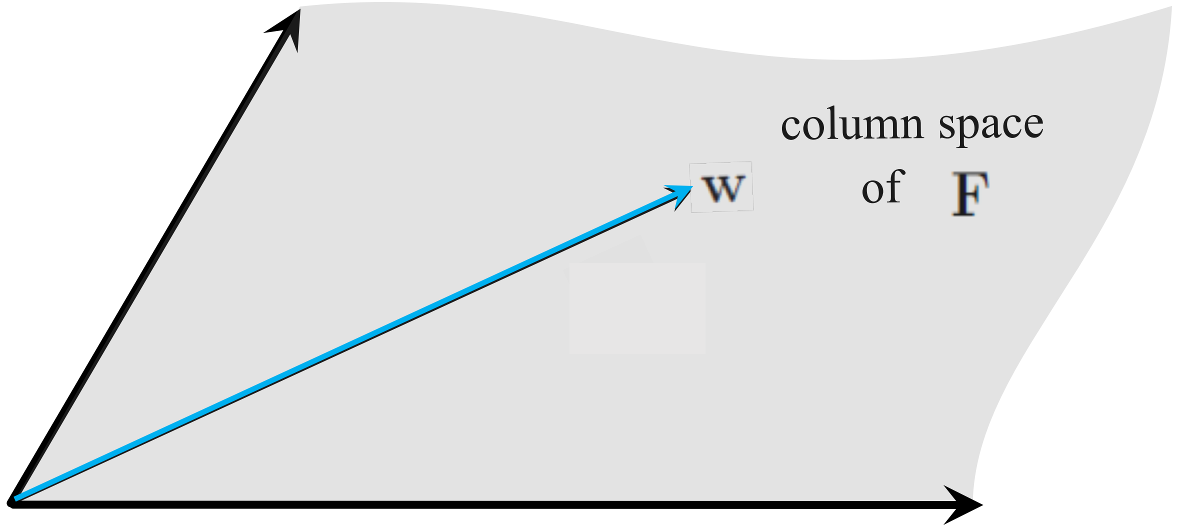
$$f_m(\mathbf{x}) = x_1^{m_1} x_2^{m_2} \cdots x_N^{m_N}$$

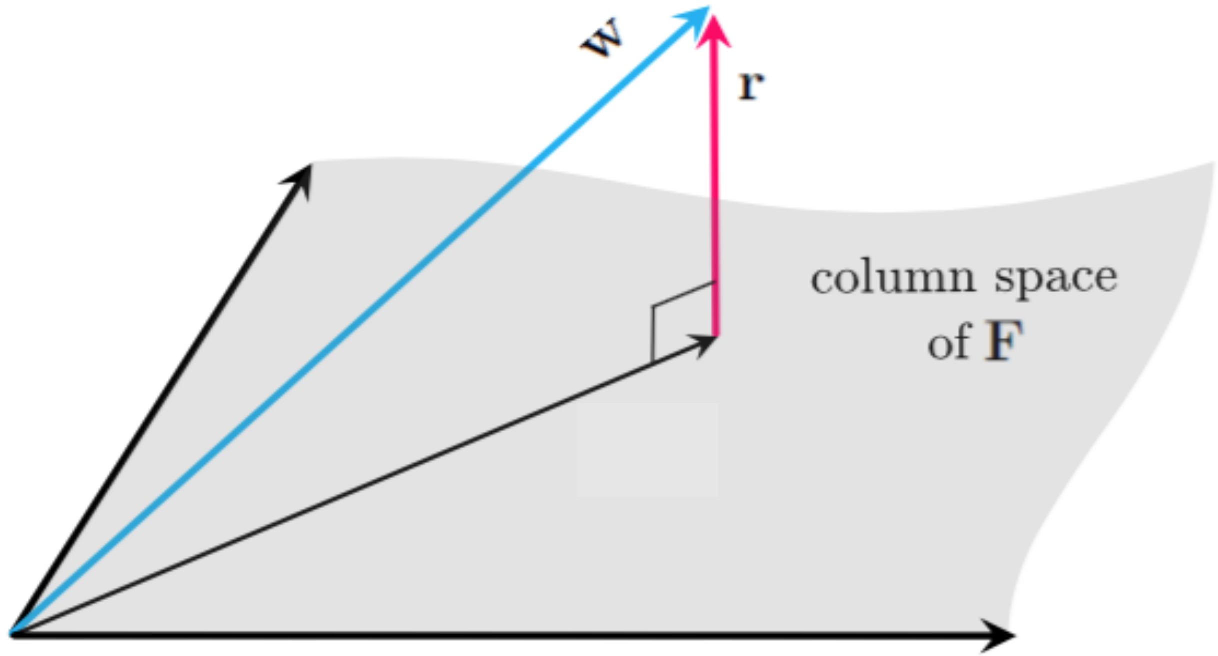
where $0 \leq m_1 + m_2 + \ldots + m_N \leq D$

There are $\binom{N+D}{D}$ such monomial terms!

For $D = 5$ and $N = 100$ → $M = 96, 560, 646$
For $D = 5$ and $N = 500$ → $M = 268, 318, 178, 226$

A universal method for kernelization





$$\mathbf{F} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{f}_1 & \mathbf{f}_2 & & \mathbf{f}_P \\ \vdots & \vdots & & \vdots \end{bmatrix} \quad \mathbf{w} = \sum_{p=1}^{P} \mathbf{f}_p z_p = \mathbf{F}\mathbf{z}$$

$$\mathbf{w} = \mathbf{F}\mathbf{z} + \mathbf{r}$$

where $\quad \mathbf{f}_p^T \mathbf{r} = 0 \quad$ for all p

Kernelizing Least Squares regression

$$\mathbf{f}_p = \begin{bmatrix} f_1(\mathbf{x}_p) & f_2(\mathbf{x}_p) & \cdots & f_M(\mathbf{x}_p) \end{bmatrix}^T$$

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_M \end{bmatrix}^T$$

$$g(w_0, w_1, \ldots\; g(b, \mathbf{w}) = \sum_{p=1}^{P} \left(\sum_{m=0}^{M} f_m(\mathbf{x}_p)\, w_m - y_p \right)^2$$

$$b = w_0$$

Form the matrix $\mathbf{F} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{f}_1 & \mathbf{f}_2 & & \mathbf{f}_P \\ \vdots & \vdots & & \vdots \end{bmatrix}_{M\times P}$ and decompose $\mathbf{w}$ over its columns as

$$\underset{M\times 1}{\mathbf{w}} = \mathbf{F}\underset{P\times 1}{\mathbf{z}} + \underset{M\times 1}{\mathbf{r}}$$

Kernelizing Least Squares regression

Plugging in $\mathbf{w} = \mathbf{F}\mathbf{z} + \mathbf{r}$

$$g\left(b, \mathbf{w}\right) = \sum_{p=1}^{P} \left(b + \mathbf{f}_p^T \left(\mathbf{F}\mathbf{z} + \mathbf{r}\right) - y_p\right)^2 = \sum_{p=1}^{P} \left(b + \mathbf{f}_p^T \mathbf{F}\mathbf{z} - y_p\right)^2 = g\left(b, \mathbf{z}\right)$$

We got rid of $\mathbf{w}$!

All that's left to do is form the $P \times P$ kernel matrix $\mathbf{H} = \mathbf{F}^T\mathbf{F}$

For polynomial and Fourier bases, we can form **H** directly without explicitly forming **F**

Cost function	Original version	Kernelized version
Least Squares regression	$\sum_{p=1}^{P}\left(b+\mathbf{f}_p^T\mathbf{w}-y_p\right)^2$	$\sum_{p=1}^{P}\left(b+\mathbf{h}_p^T\mathbf{z}-y_p\right)^2$

"Kernelizing" ML problems

- ✓"kernelizing" allows us to rewrite machine learning models employing fixed basis (like polynomials) in a different but equivalent way and avoid this problem
- ✓Virtually all machine learning models can be kernelized using the same simple argument (see [29]), this includes linear and nonlinear:
  - ✓ support vector machines, logistic regression, Least Squares regression, K-means, principal component analysis (PCA), …
- ✓New fixed bases can be defined directly via their kernels (e.g., Radial Basis Functions)
- Ⓧ while scaling gracefully in N the dimension of the input every 'kernelized' form scales extremely poorly with P the size of the dataset
- ✓ Classic methods exist [49] and promising research currently underway to ameliorate this issue [47-48]

Key points on neural nets

1. Regression / classification are noisy sampled function approximation problems
2. Fixed / neural network bases both often used for regression / classification tasks
3. In the language of machine learning: basis functions = features
4. “Feature learning” means to determine a good set of such basis functions
5. An alternative to classical fixed bases (like e.g., polynomials) a neural net basis function is a *composition* of functions, typically of a fixed type like e.g., tanh or max
6. This compositional structure gives each neural network basis function a number of parameters that provide significant flexibility
7. The more compositions or ‘layers’ the more flexible a neural net basis function becomes

At a glance: fixed vs. neural net bases

Some strengths and weaknesses of each type of bases

- fixed basis kernels (e.g., polynomials):
  - ✓ induce convex costs for regression and classification
  - Ⓧ but scale very poorly with size of dataset
- adjustable neural network bases:
  - Ⓧ induce *nonconvex* costs for regression and classification
  - ✓ recently this *non-convexity* has been overcome in important instances (more on this in part III)
  - ✓ scale more gracefully with both the dimension of input and size of dataset
  - ✓ allow inclusion of knowledge (more on this in part III)

end of Part II