CMC MSU Department of Algorithmic Languages
Samsung Moscow Research Center

Neural Networks for Natural Language Processing

Нейронные сети в задачах автоматической обработки текстов

Lecture 6. Hyperparameter selection. Bias / variance trade-off.

Arefyev Nikolay
CMC MSU Department of Algorithmic Languages &
Samsung Moscow Research Center

A machine learning algorithm usually corresponds to a combination of the following 3 elements:

(either explicitly specified or implicit)

- ✓ the choice of a specific function family: F
  (often a parameterized family)
- ✓ a way to evaluate the quality of a function $f \in F$
  (typically using a cost (or loss) function L
  mesuring how wrongly f prédicts)
- ✓ a way to search for the «best» function $f \in F$
  (typically an optimization of function parameters to minimize the overall loss over the training set).

Evaluating the quality of f(x)

- Real quantities we are interested in:
  - lifes saved, time saved, $ earned
- Standard evaluation metrics for the task:
  - Classification:
    - accuracy / misclassification error rate
    - precision / recall / f-measure
  - MT: BLEU (Bilingual Evaluation Understudy)
- Actual (surrogate) loss minimized

Surrogate loss

- For a classification task: $f : \mathbb{R}^d \to \{0, \ldots, m-1\}$
  misclassification error loss: $L(f(x), y) = I_{\{f(x) \neq y\}}$

Problem: it is hard to *optimize* the misclassification loss directly
(gradient is 0 everywhere. NP-hard with a linear classifier) Must use a surrogate loss:

	Binary classifier	Multiclass classifier
Probabilistic classifier	Outputs probability of class 1 $g(x) \approx P(y=1 \mid x)$ Probability for class 0 is $1-g(x)$ Binary cross-entropy loss: $L(g(x),y) = -(y \log(g(x)) + (1-y) \log(1-g(x)))$ Decision function: $f(x) = I_{g(x)>0.5}$	Outputs a vector of probabilities: $g(x) \approx (P(y=0\|x), \ldots, P(y=m-1\|x))$ Negated conditional log likelihood loss $L(g(x),y) = -\log g(x)_y$ Decision function: $f(x) = \text{argmax}(g(x))$
Non-probabilistic classifier	Outputs a «score» $g(x)$ for class 1. score for the other class is $-g(x)$ Hinge loss: $L(g(x),t) = \max(0, 1-tg(x))$ where $t=2y-1$ Decision function: $f(x) = I_{g(x)>0}$	Outputs a vector $g(x)$ of real-valued scores for the m classes. Multiclass margin loss $L(g(x),y) = \max(0, 1+\max_{k \neq y}(g(x)_k) - g(x)_y)$ Decision function: $f(x) = \text{argmax}(g(x))$

Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

Expected / Empirical risk

Examples (x,y) are supposed drawn i.i.d. from an unknown true distribution $p(\mathrm{x},\mathrm{y})$ (from nature or industrial process)

- Generalization error = Expected risk (or just «Risk»)
  «how poorly we will do on average on the infinity of future examples from that unknown distribution»

$$R(f) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})}[L(f(\mathbf{x}), \mathbf{y})]$$

- Empirical risk = average loss on a finite dataset
  «how poorly we're doing on average on this finite dataset»

$$\hat{R}(f, D) = \frac{1}{|D|} \sum_{(\mathbf{x},\mathbf{y}) \in D} L(f(\mathbf{x}), \mathbf{y})$$

where |D| is the number of examples in D

Empirical risk minimization principle

Examples (x,y) are supposed drawn i.i.d. from an unknown true distribution $p(\mathrm{x},\mathrm{y})$ (nature or industrial process)

- We'd love to find a predictor that minimizes the generalization error (the expected risk)
- But can't even compute it! (expectation over unknown distribution)
- Instead: **Empirical risk minimization principle**

«Find predictor that minimizes average loss over a trainset»

$$\hat{f}(D_{\text{train}}) = \underset{f \in F}{\operatorname{argmin}}\, \hat{R}(f, D_{\text{train}})$$

This is the training phase in ML

Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

Eval expected risk = generalization error

- We can't compute expected risk $R(f)$
- But $\hat{R}(f, D)$ is a good estimate of $R(f)$ provided:
  - D was not used to find/choose f
    otherwise estimate is biased ⇨ can't be the training set!
  - D is large enough (otherwise estimate is too noisy); drawn from p

➡ Must keep a separate test-set $D_{\text{test}} \neq D_{\text{train}}$ to properly estimate generalization error of $\hat{f}(D_{\text{train}})$:

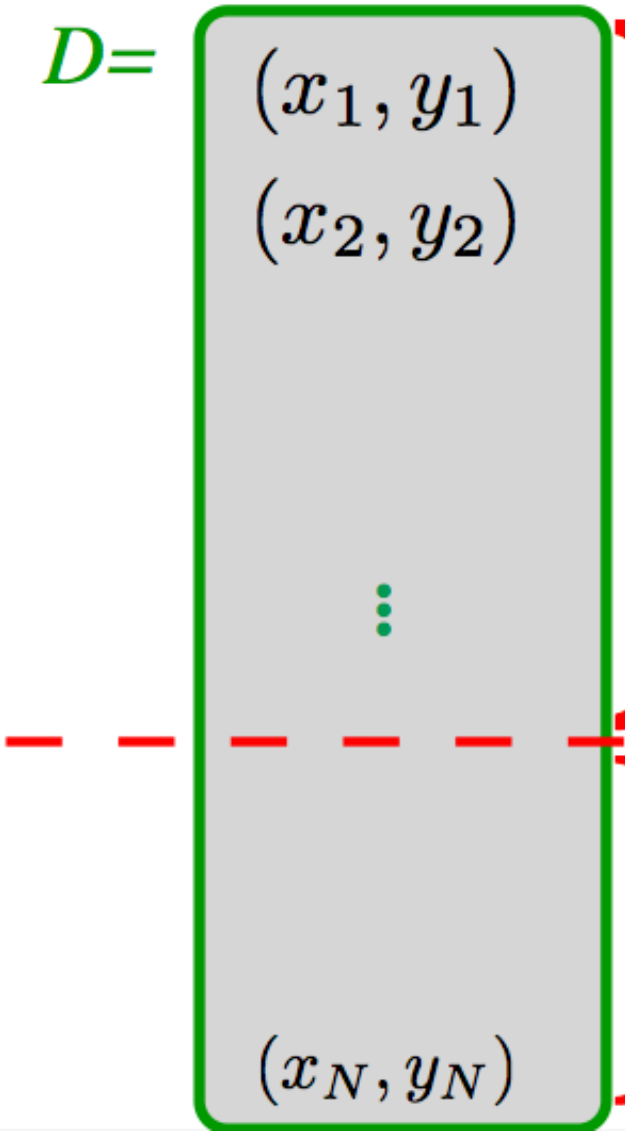
$$R(\hat{f}(D_{\text{train}})) \approx \hat{R}(\hat{f}(D_{\text{train}}), D_{\text{test}})$$

generalization error

average error on **test**-set (never used for training)

This is the test phase in ML

Train / test split



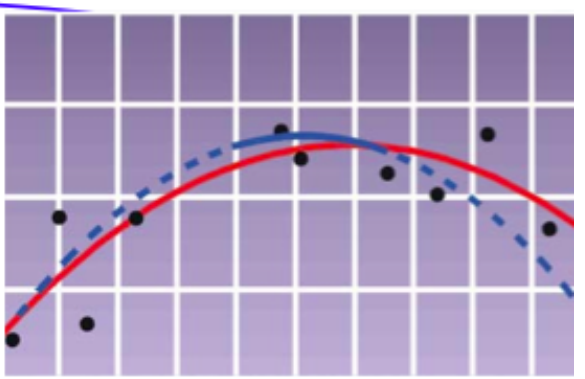
- Provided large enough dataset D drawn from $p(x,y)$
- Make sure examples are in random order.
- Split dataset in **two:** D_{train} and D_{test}
- Use D_{train} to choose/ optimize/find best predictor $f = \hat{f}(D_{\text{train}})$
- Use D_{test} to evaluate generalization performance of predictor f.

Function families

$F_{polynomial\ p}$

Polynomial predictor (of degree p):

$$f(x) = b + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_p x^p$$



F_{linear}

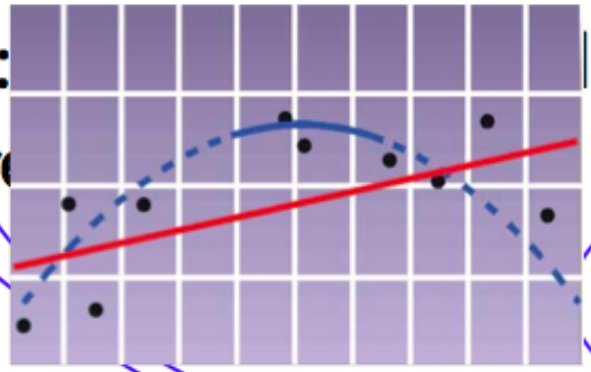
Linear (affine) predictor: $f_\theta(x) = wx + b$ (in 1 dimension)

(«linear regression») $f_\theta(x) = w^T x + b$ (in d dimensions)

$$\theta = \{w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

Q: [illegible]est

pr[illegible]



F_{const}

Constant predictor: $f_\theta(x)=b$

where $\theta=\{b\}$

(always predict the same value or class!)

Capacity of ML algorithm

- Choosing a specific Machine Learning algorithm means choosing a specific function family F.
- How «big, rich, flexible, expressive, complex» that family is, defines what is informally called the «capacity» of the ML algorithm.

  Ex: capacity($F_{polynomial\ 3}$) > capacity(F_{linear})
- One can come up with several formal measures of «capacity» for a function family / learning algorithm (e.g. VC-dimension Vapnik–Chervonenkis)
- One rule-of-thumb estimate, is the number of adaptable parameters: i.e. how many *scalar* values are contained in θ.

  Notable exception: chaining many linear mappings is still a linear mapping!

Example: regression

$t = \sin(2\pi x) + \epsilon$ In practice we don't know it!

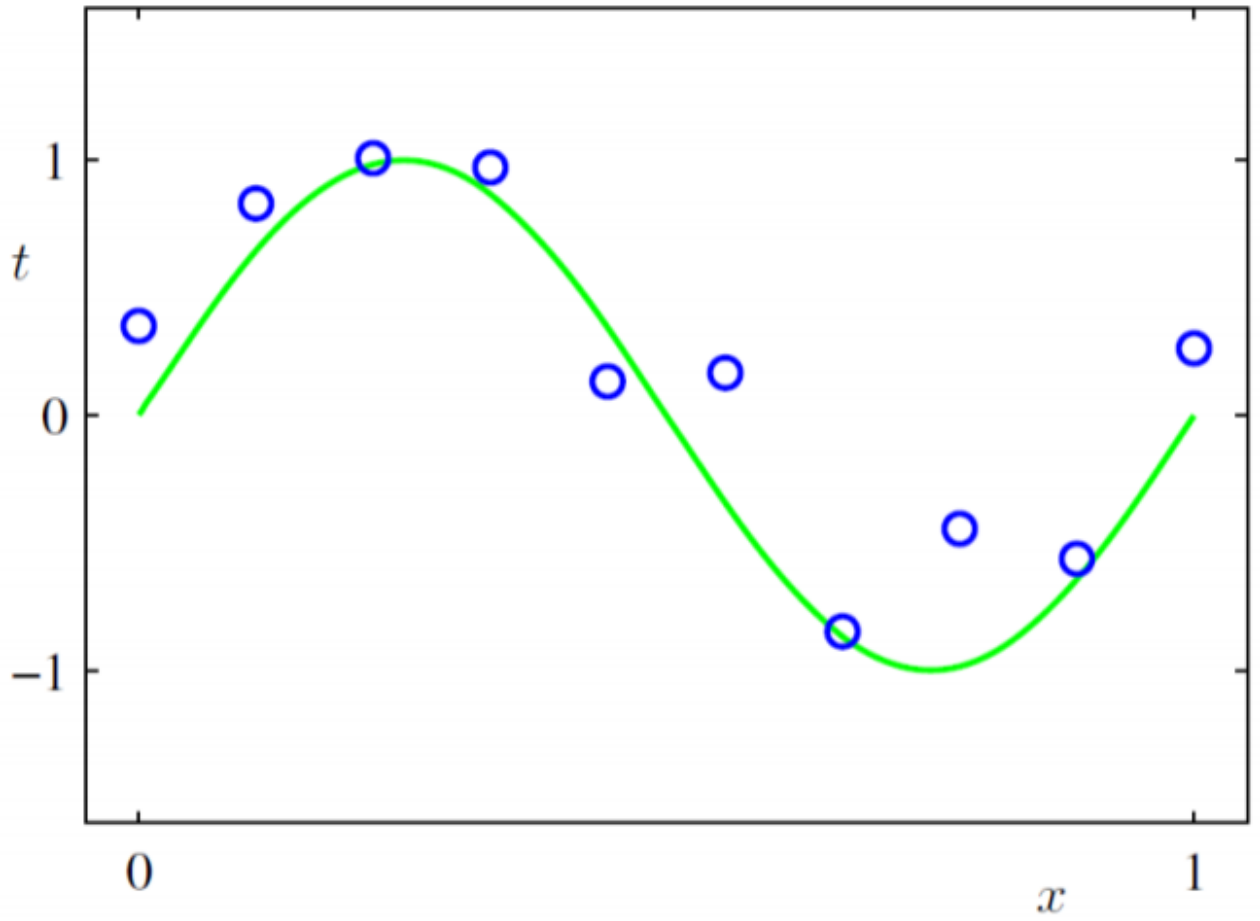
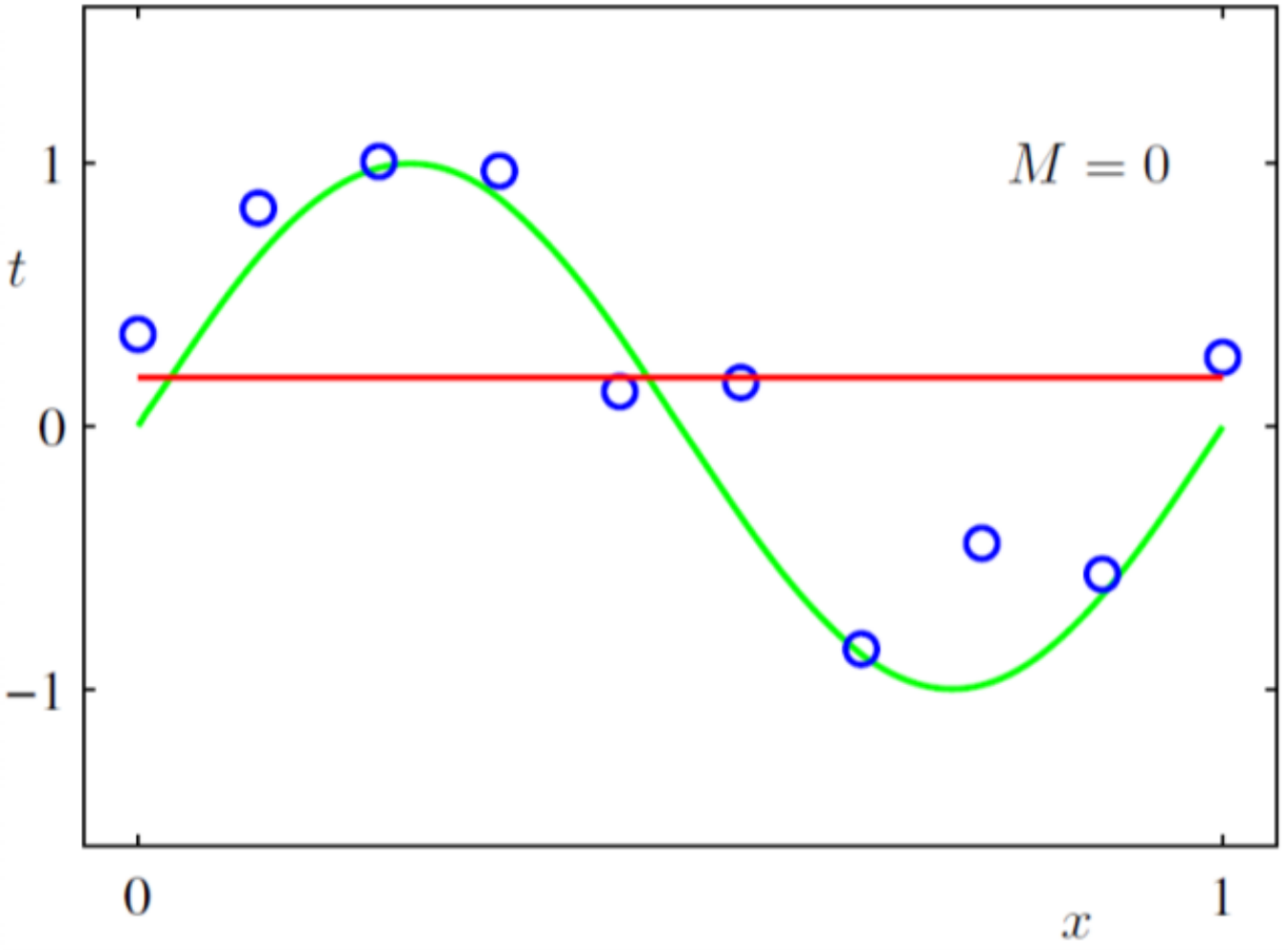


Figure from *Machine Learning and Pattern Recognition*, Bishop

Example: regression

$t = \sin(2\pi x) + \epsilon$



Regression using polynomial of degree *M*

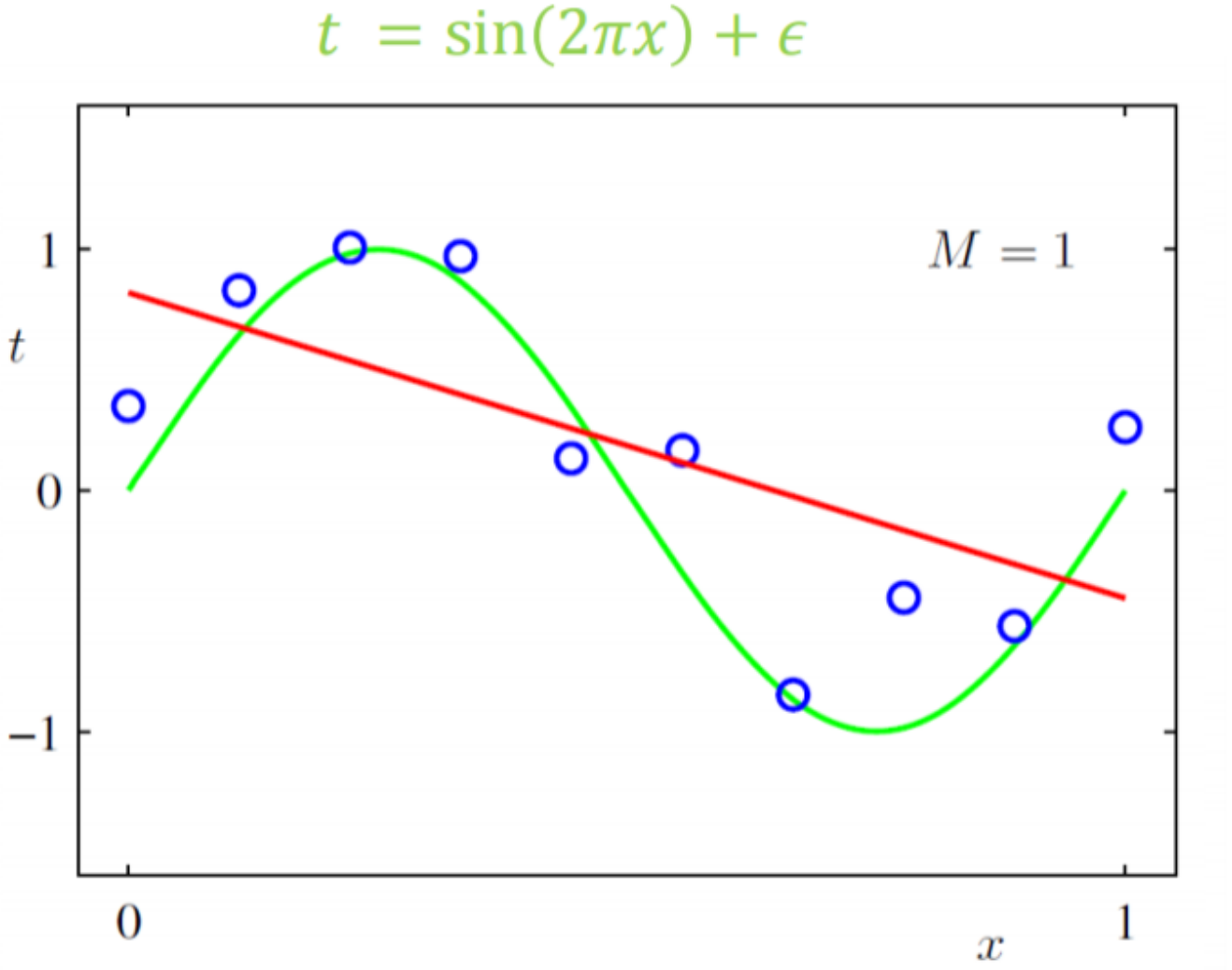
Capacity too low

Underfitting / High bias:

bad on train & test

Figure from *Machine Learning and Pattern Recognition*, Bishop

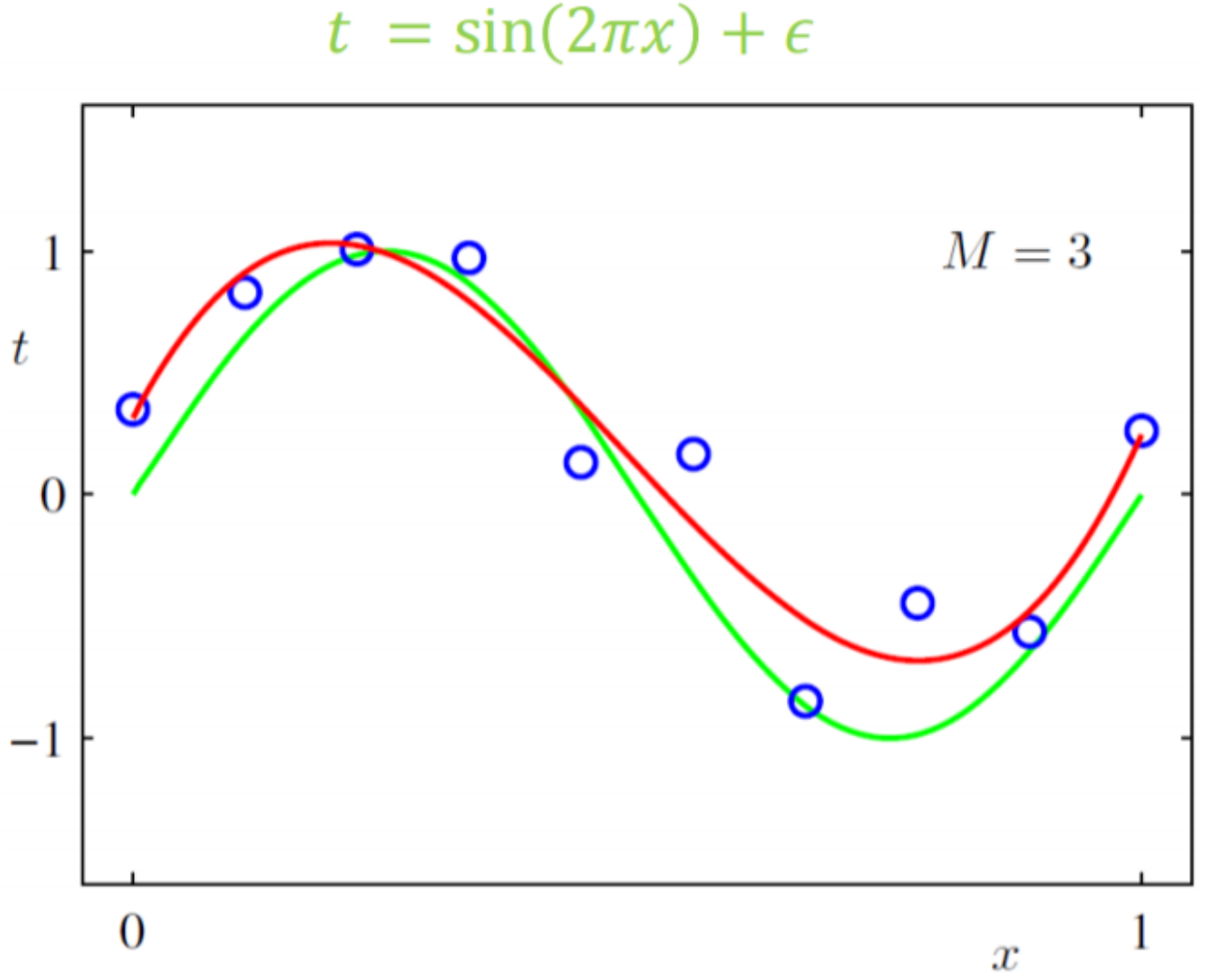
Example: regression



Capacity too low
Underfitting /
High bias:
bad on train & test

Figure from *Machine Learning and Pattern Recognition*, Bishop

Example: regression

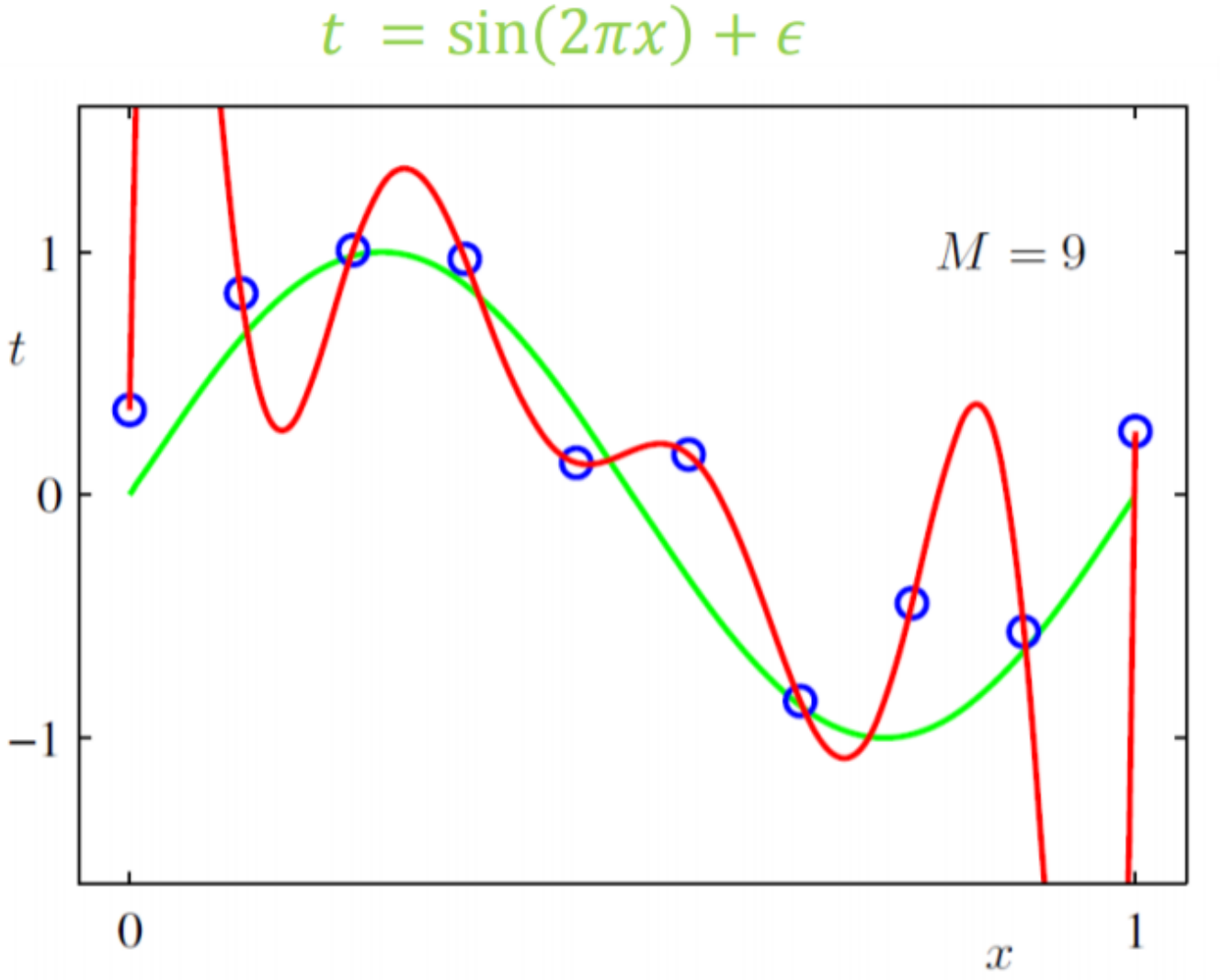


Near optimal capacity

good on train & test

Figure from *Machine Learning and Pattern Recognition*, Bishop

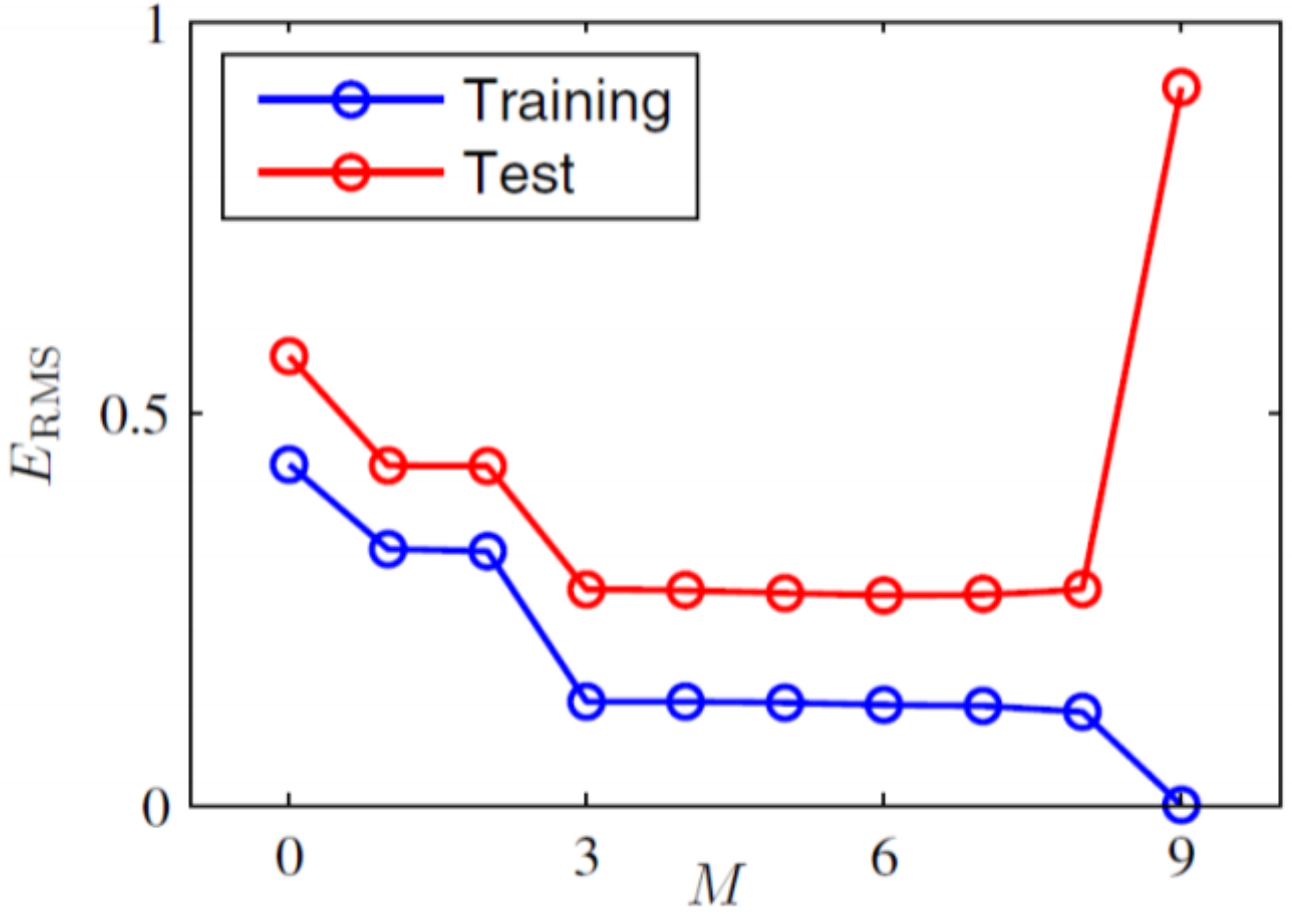
Example: regression



Capacity too high
Overfitting /
High variance:
good on train
bad on test

Figure from *Machine Learning and Pattern Recognition*, Bishop

Example: regression

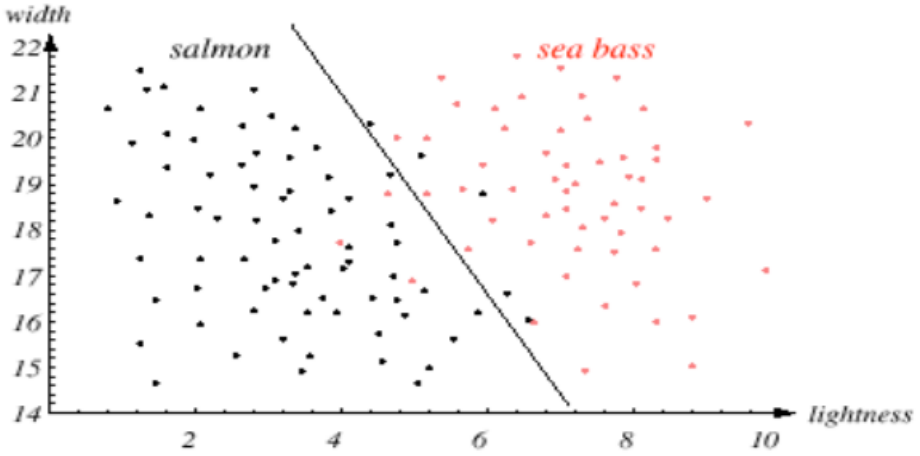


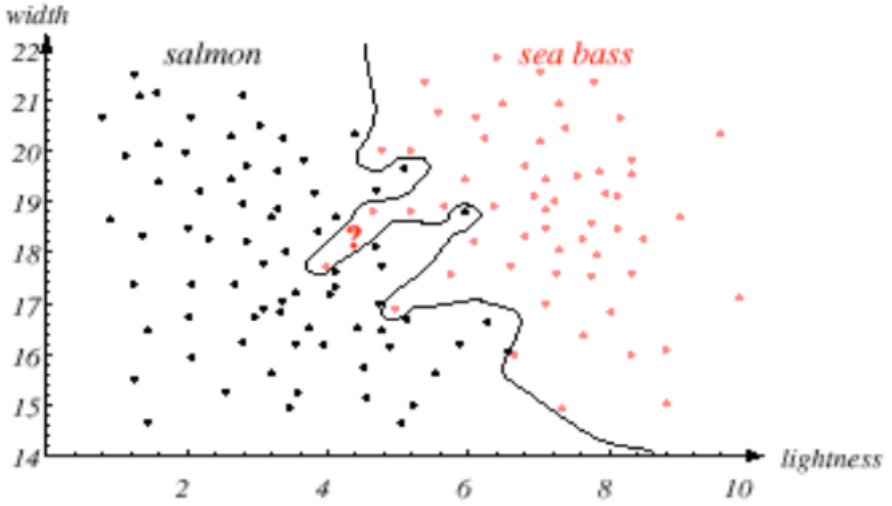
Don't choose model on:
Train set
Test set

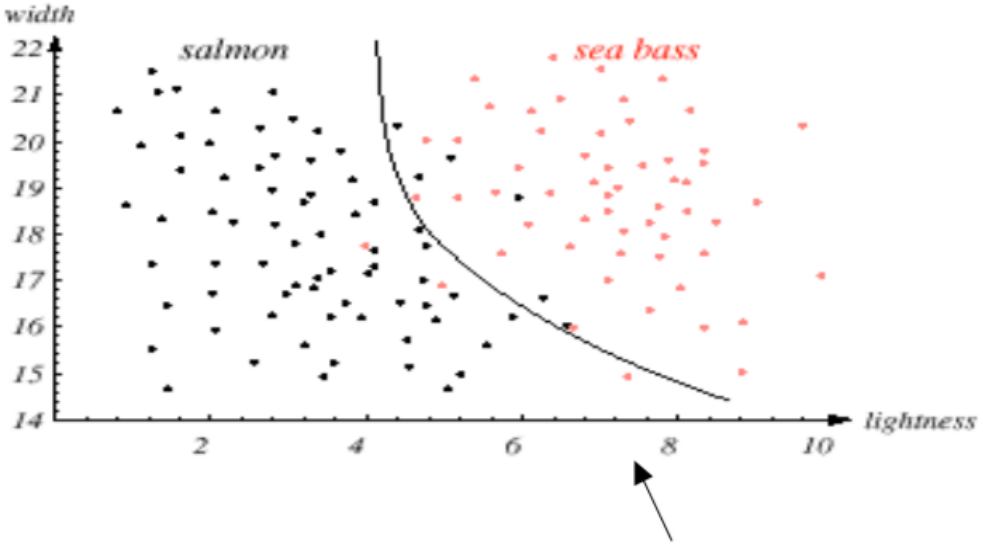
Use separate
Validation/development set

Figure from *Machine Learning and Pattern Recognition*, Bishop

Example: classification







People like this one. Why?

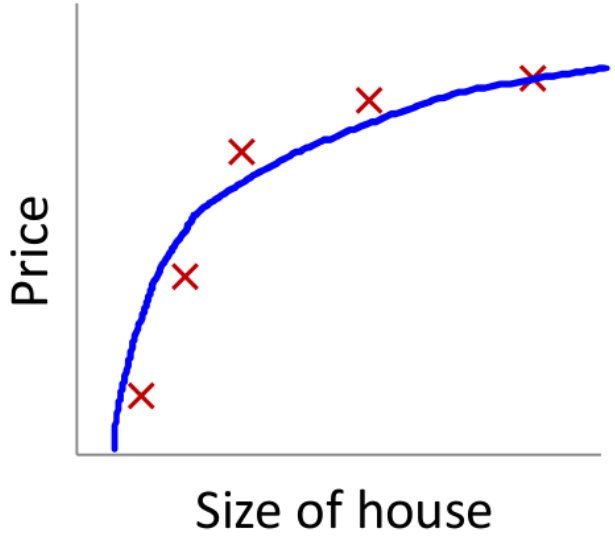
Andrew W. Moore. Cross-validation for detecting and preventing overfitting.

But we want large function family (NNs)!

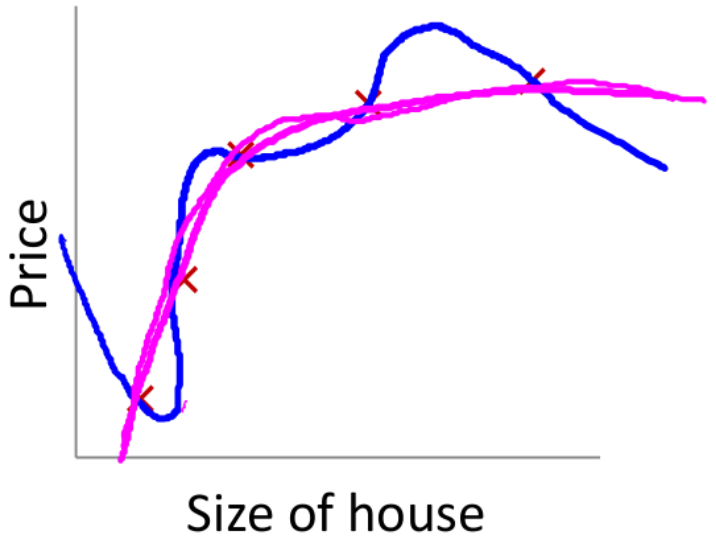
- L2 regularization for regression

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

What if lambda=0? Very large?



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

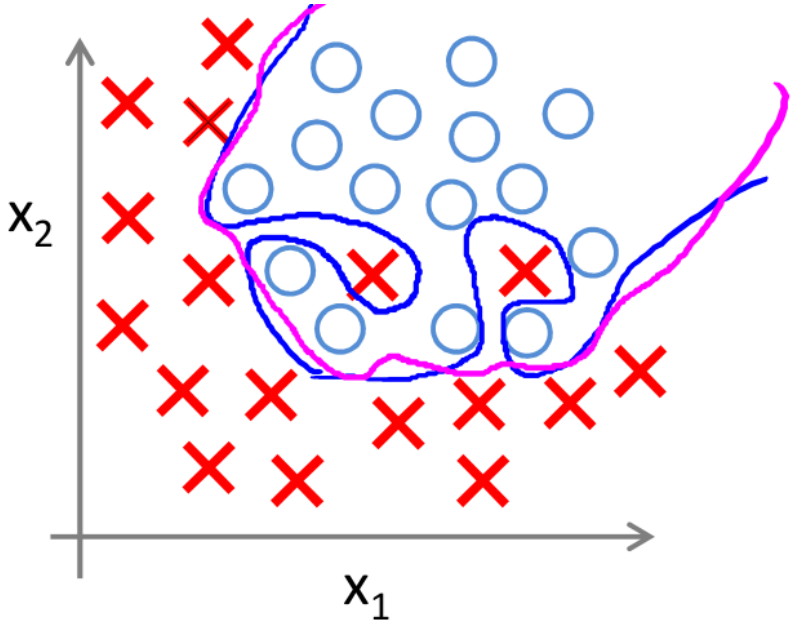
But we want large function family (NNs)!

- L2 regularization for classification

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

$$+ \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

$\theta_1, \theta_2, \ldots, \theta_n$



Andrew Ng, Machine Learning course @ coursera.org

Early stopping

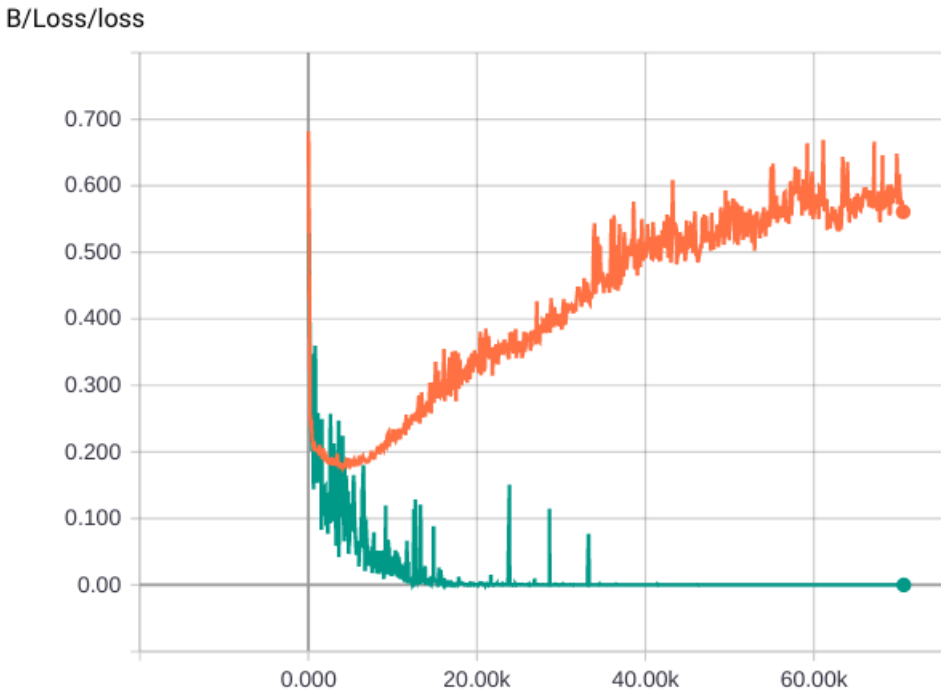
- We start with small (random) weights

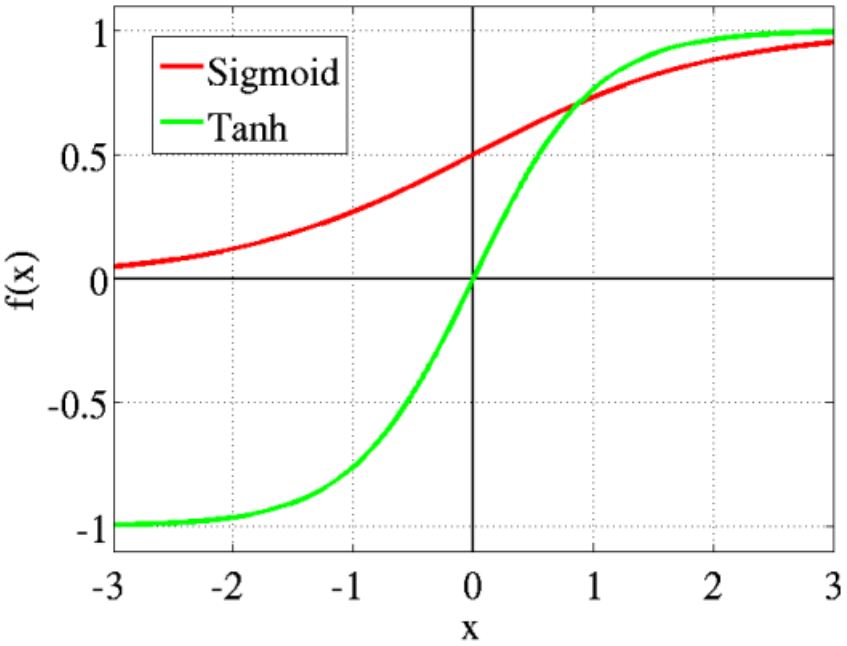
  => low capacity, approx. linear model

- During training the weights grow

  => capacity increase, non-linear range of activations

Stop training before the capacity is too high!

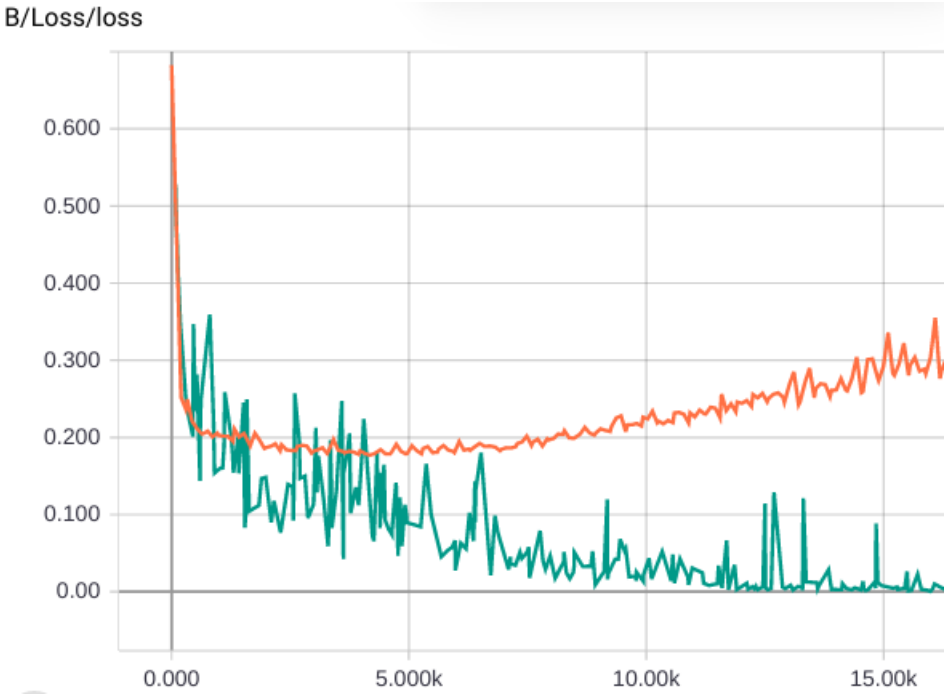


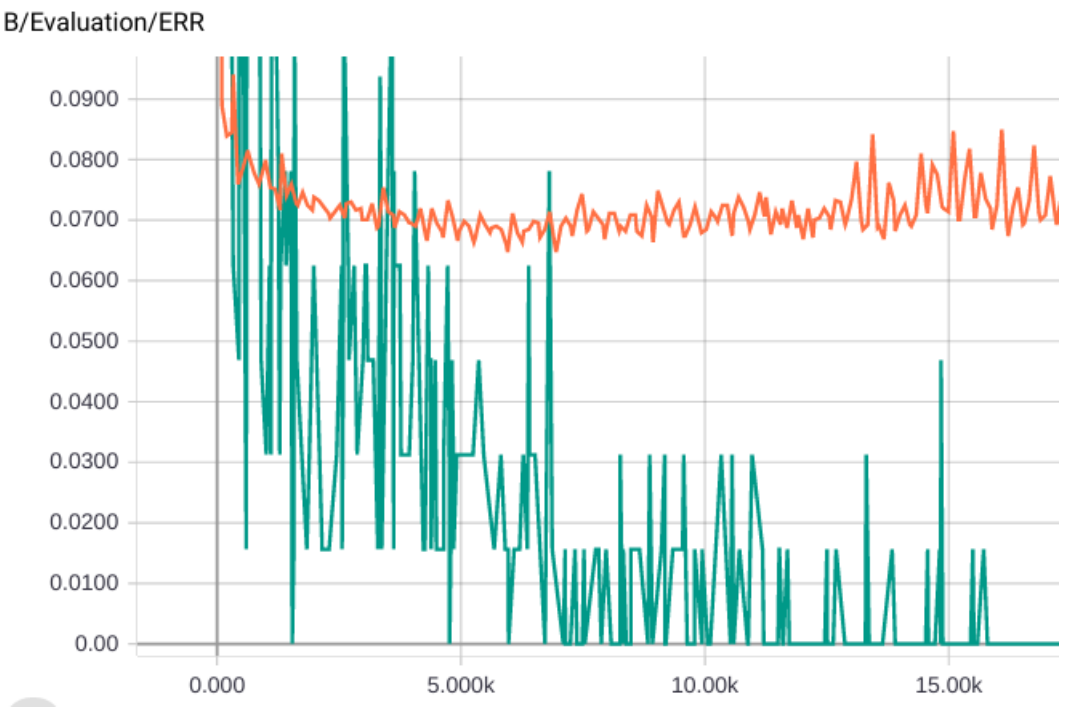


Early stopping

Stop training before the capacity is too high!

- Use weights giving highest quality on dev/val set





Effective Capacity of ML algorithm

The «effective» capacity of a ML algo is controlled by:

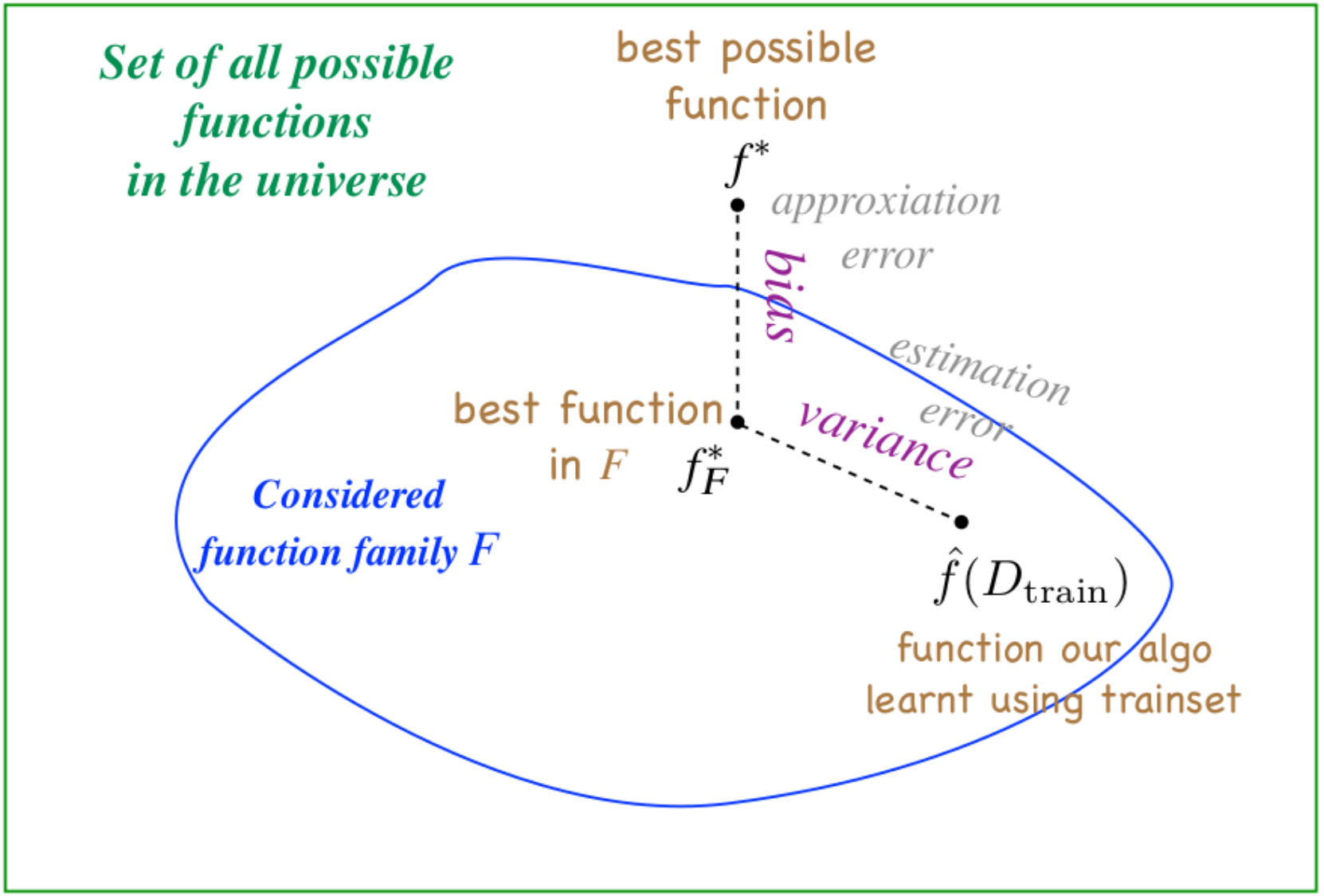
- Choice of ML algo, which determines big family F
- Hyper-parameters that further specify F
  e.g.: degree p of a polynomial predictor; Kernel choice in SVMs;
  #of layers and neurons in a neural network
- Hyper-parameters of «regularization» schemes
  e.g. constraint on the norm of the weights w
  (⇨ ridge-regression; L_2 weight decay in neural nets);
  Bayesian prior on parameters; noise injection (dropout); ...
- Hyper-parameters that control early-stopping of the iterative search/optimization procedure.
  (⇨ won't explore as far from the initial starting point)

Parameters vs. Hyperparameters

Algo	Capacity-control hyperparameters	Learned parameters
logistic regression (L_2 regularized)	strength of L_2 regularizer	w,b
linear SVM	C	w,b
kernel SVM	C; kernel choice & params (σ for RBF; degree for polynomal)	support vector weights: α
neural network	layer sizes; early stop; ...	layer weight matrices
decision tree	depth	the tree (with index and threshold of variables)
k-nearest neighbors	k; choice of metric	memorizes trainset

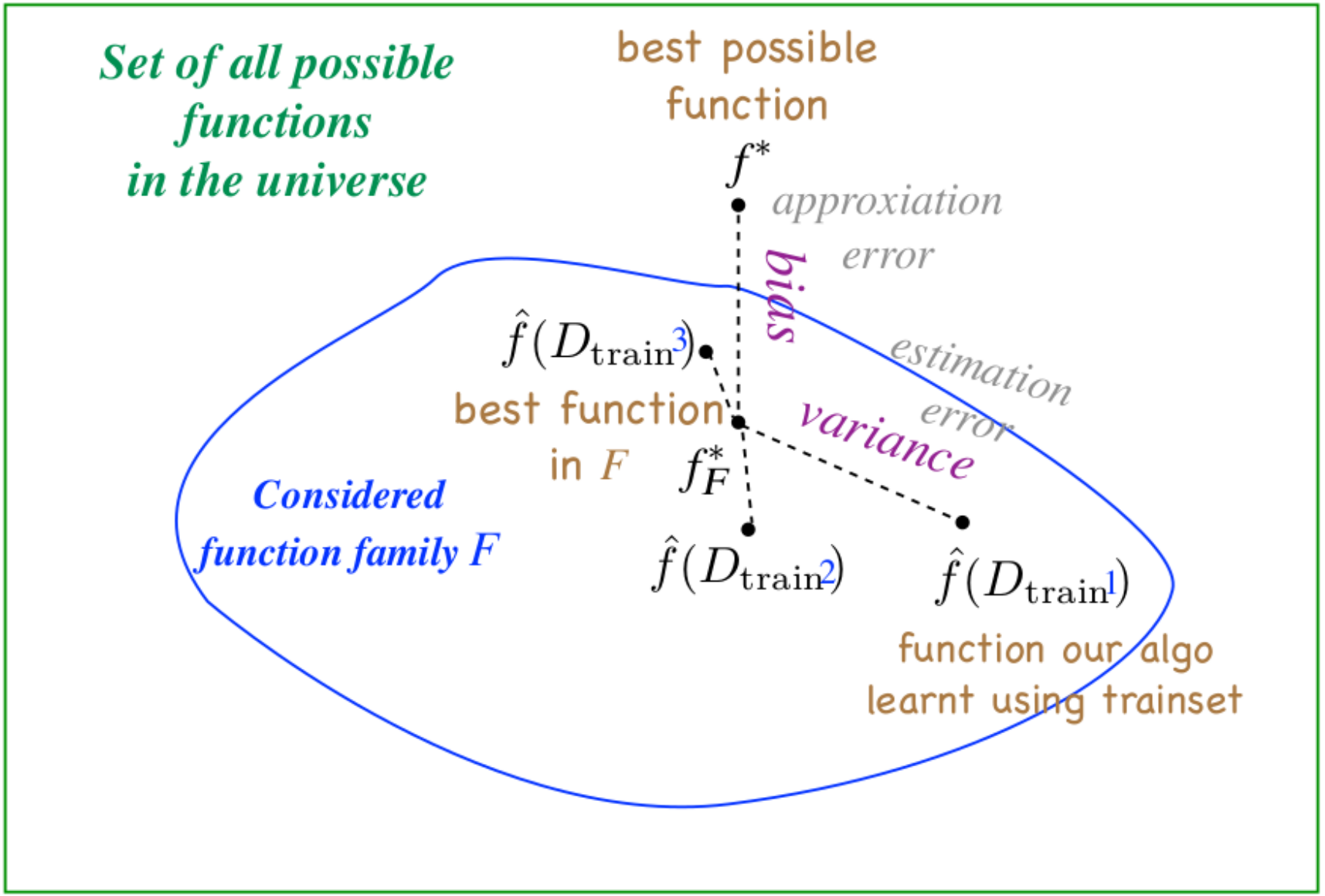
Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

Generalization error decomposition



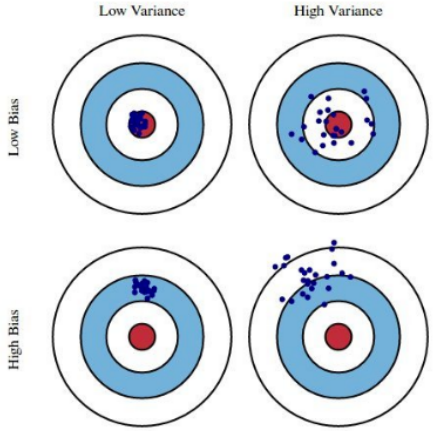
Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

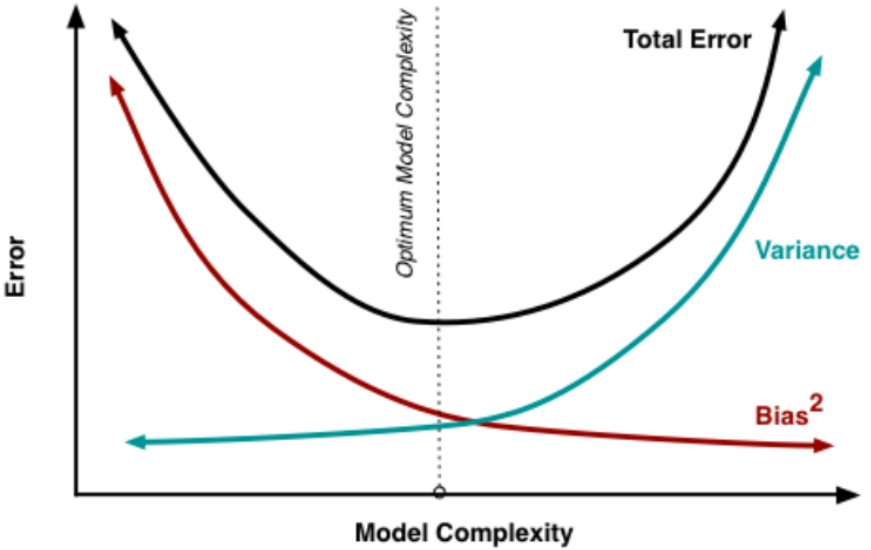
Generalization error decomposition



Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

Bias Variance Tradeoff





$$Y = f(X) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon).$$

$$Err(x) = E\left[(Y - \hat{f}(x))^2\right]$$

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

http://scott.fortmann-roe.com/docs/BiasVariance.html

Max Welling. Marrying Graphical Models & Deep Learning @ Deep learning summer school 2017,

Bias Variance Tradeoff

- Choosing richer F: capacity ↑
  ➪ bias ↓ *but* variance ↑.
- Choosing smaller F : capacity ↓
  ➪ variance ↓ *but* bias ↑.
- Optimal compromise... will depend on number of examples n
- Bigger n ➪ variance ↓
  So we can afford to increase capacity (to lower the bias)
  ➪ can use more expressive models
- The best regularizer is more data!

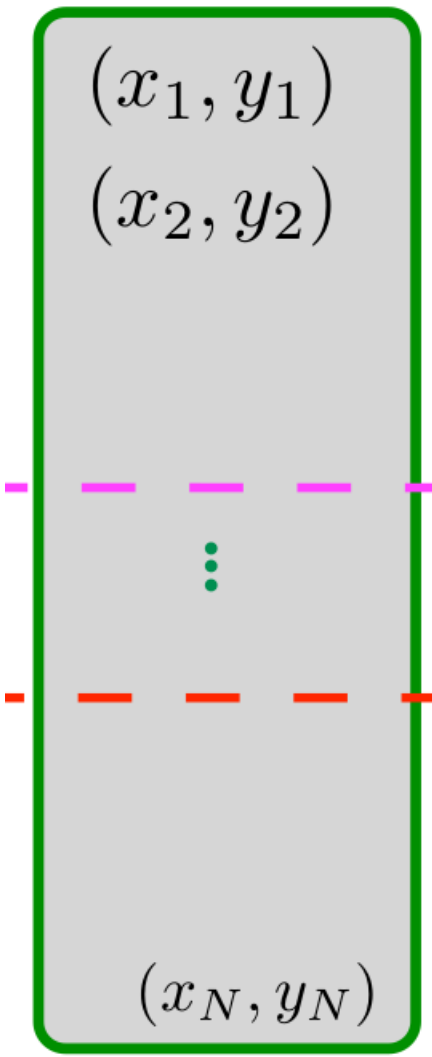
Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

Model selection how to

Make sure examples are in random order

Split data D in 3: D_{train} D_{valid} D_{test}

$D=$



Training set D_{train}

Validation set D_{valid}

Test set D_{test}

Model selection meta-algorithm:

For each considered model (ML algo) $\mathbf{A}$:

For each considered hyper-parameter config λ:

- train model $\mathbf{A}$ with hyperparams λ on D_{train}

$$\hat{f}_{\mathbf{A}_\lambda} = \mathbf{A}_\lambda(D_{\text{train}})$$

- evaluate resulting predictor on D_{valid} (with preferred evaluation metric)

$$e_{\mathbf{A}_\lambda} = \hat{R}(\hat{f}_{\mathbf{A}_\lambda}, D_{\text{valid}})$$

Locate $\mathbf{A}^*, \lambda^*$ that yielded best $e_{\mathbf{A}_\lambda}$

Either return $f^* = f_{\mathbf{A}^*_{\lambda^*}}$

Or retrain and return

$$f^* = \mathbf{A}^*_{\lambda^*}(D_{\mathbf{train}} \cup D_{\mathbf{valid}})$$

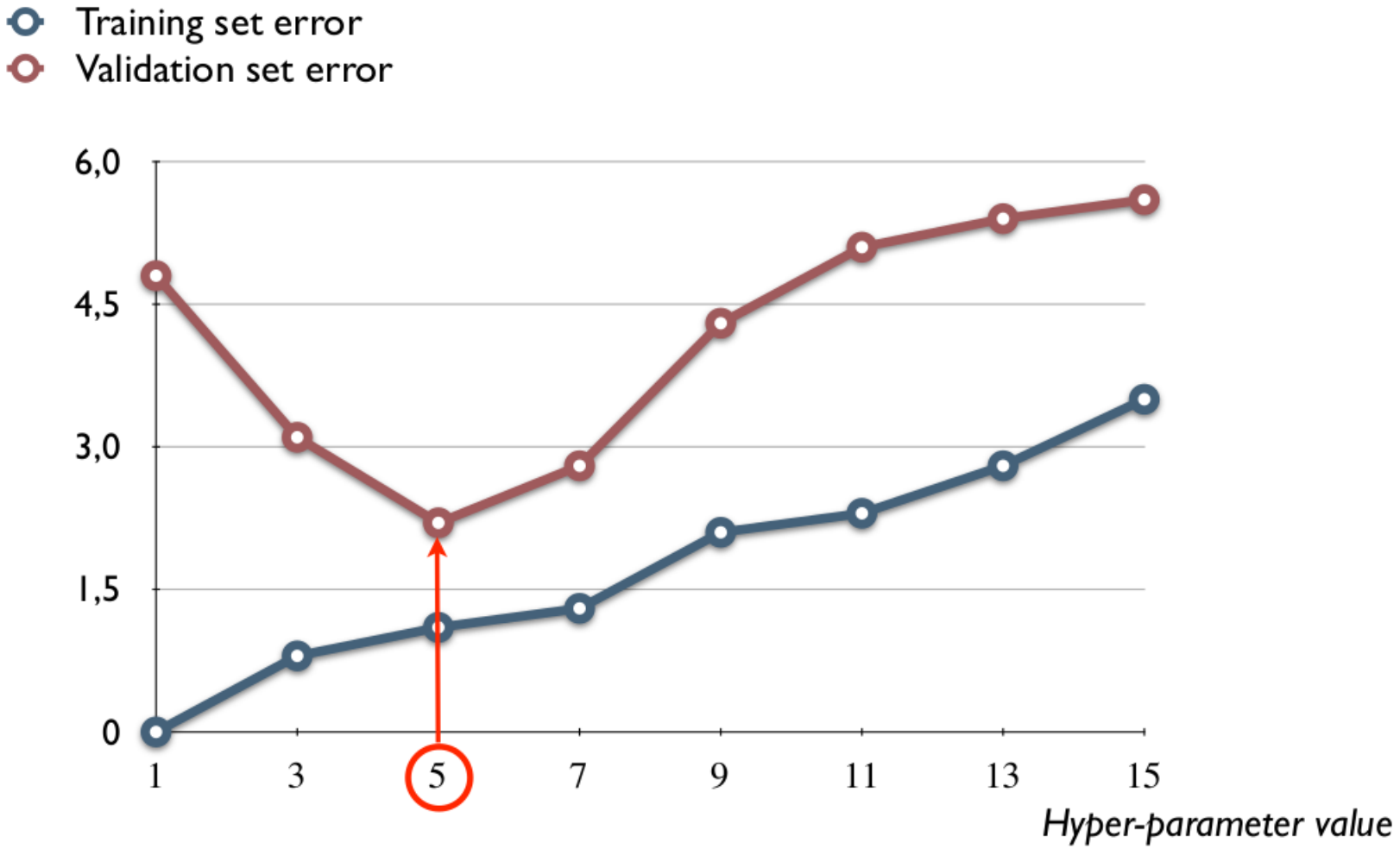
Finally: compute unbiased estimate of generalization performance of f^* using D_{test}

$$\hat{R}(f^*, D_{\mathbf{test}})$$

D_{test} must never have been used during training or model selection to select, learn, or tune anything.

Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

Hyperparameter selection

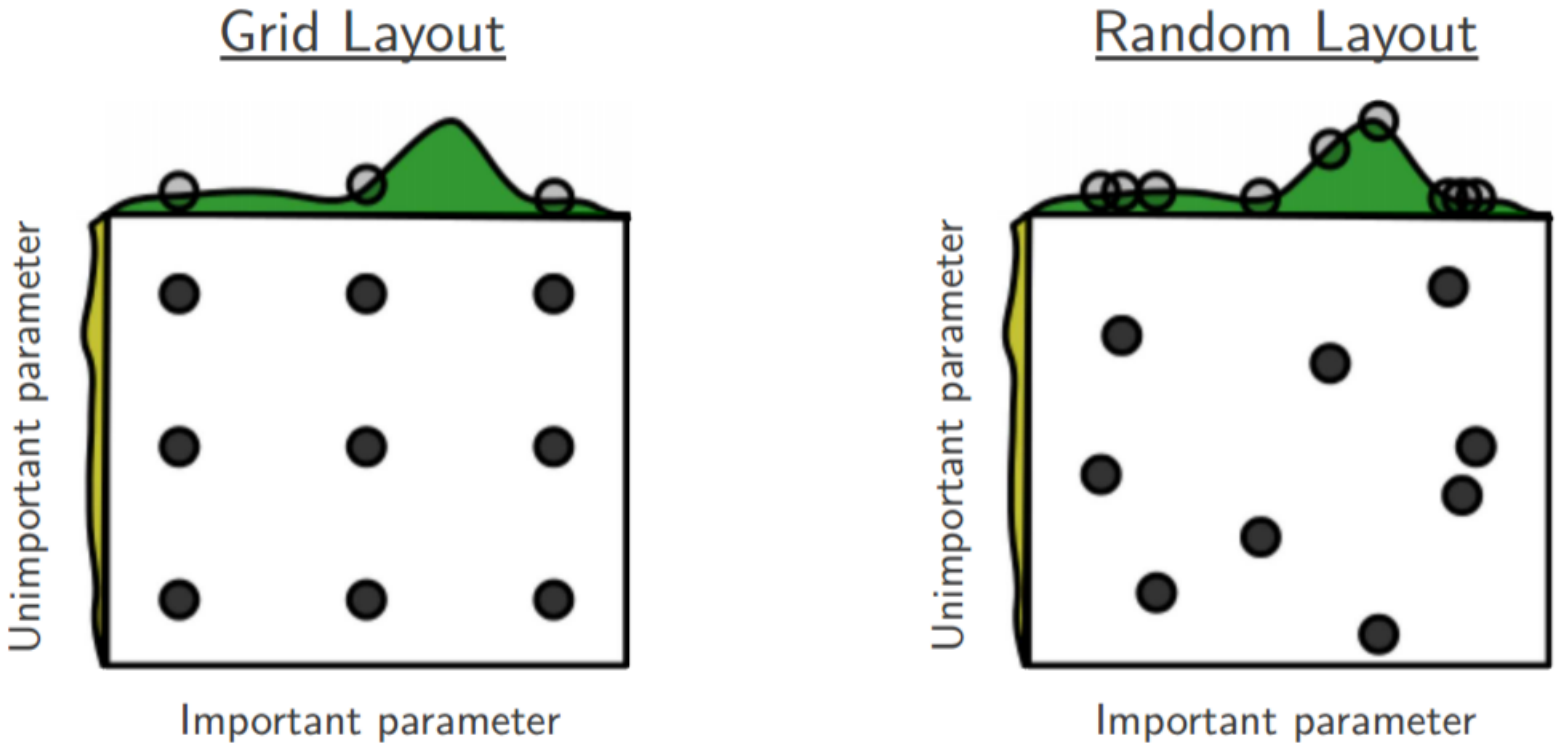


Hyper-parameter value which yields smallest error on validaiton set is 5
(it was 1 for the training set)

Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

Use random search

- Usually only some hyperparameters are important
  - Want to try many values for them



Bergstra and Bengio. Random Search for Hyper-Parameter Optimization, 2012.