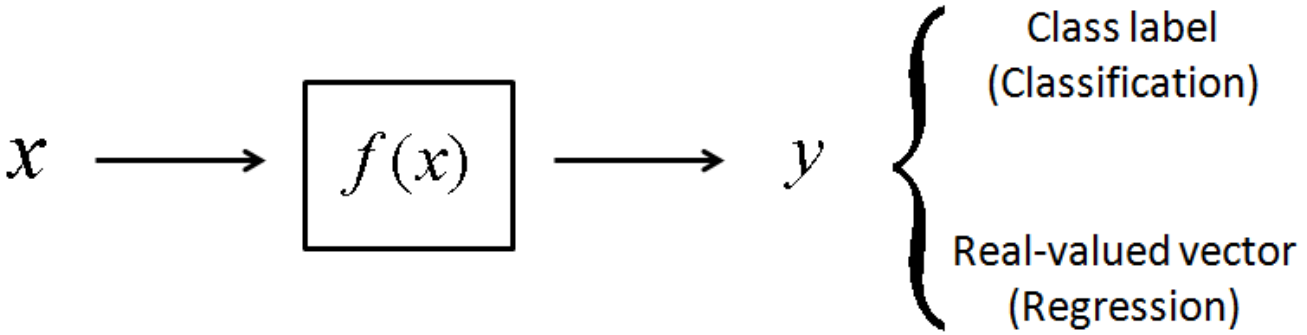
Machine Learning Basics

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

January 15, 2019

Machine Learning



Object recognition

{dog, cat, horse, flower, ...}

Low-resolution image

Super resolution

High-resolution image

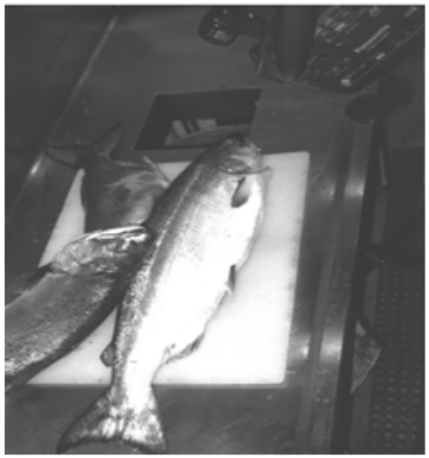
Classification

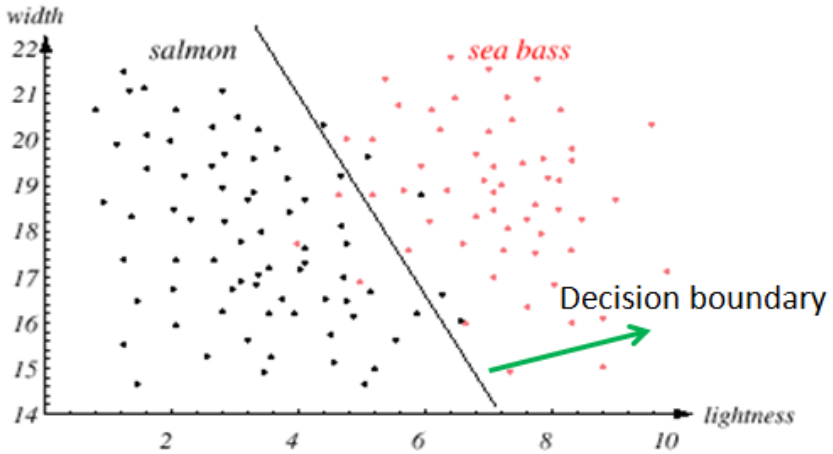
- $f(\mathbf{x})$ predicts the category that $\mathbf{x}$ belongs to

$$f : \mathcal{R}^D \rightarrow \{1, \ldots, K\}$$

- $f(\mathbf{x})$ is decided by the decision boundary
- As an variant, f can also predict the probability distribution over classes given $\mathbf{x}$, $f(\mathbf{x}) = P(y|\mathbf{x})$. The category is predicted as

$$y^* = \arg\max_k P(y = k|\mathbf{x})$$





(Duda et al. Pattern Classification 2000)

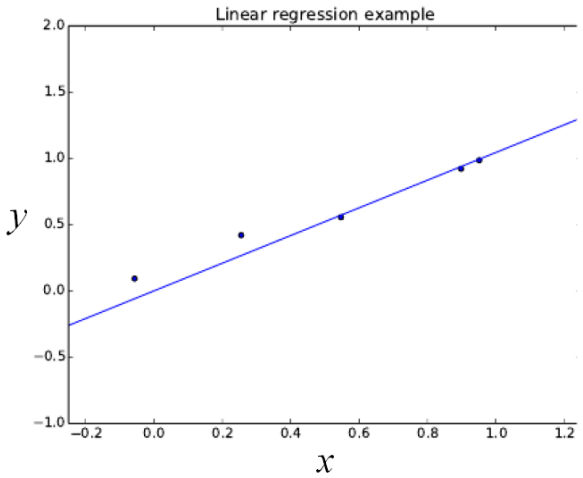
Regression

- Predict real-valued output

$$f : \mathcal{R}^D \rightarrow \mathcal{R}^M$$

- Example: linear regression

$$y = \mathbf{w}^t \mathbf{x} = \sum_{d=1}^{D} w_d x_d + w_0$$



(Bengio et al. Deep Learning 2014)

Training

- Training: estimate the parameters of f from $\{(\mathbf{x}_i^{(\text{train})}, y_i^{(\text{train})})\}$
  - Decision boundary, parameters of $P(y|\mathbf{x})$, and $\mathbf{w}$ in linear regression
- Optimize an objective function on the training set. It is a performance measure on the training set and could be different from that on the test set.
  - Mean squared error (MSE) for linear regression

$$\text{MSE}_{\text{train}} = \frac{1}{N} \sum_i ||\mathbf{w}^t \mathbf{x}_i^{(\text{train})} - y_i^{(\text{train})}||_2^2$$

  - Cross entropy (CE) for classification

$$\text{CE}_{\text{train}} = \frac{1}{N} \sum_i \log P(y = y_i^{(\text{train})} | \mathbf{x}_i^{(\text{train})})$$

Why not use classification errors $\#\{f(\mathbf{x}_i^{(\text{train})}) \neq y_i^{(\text{train})}\}$?

Optimization

- The choice of the objective function should be good for optimization
- Take linear regression as an example

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0$$

$$\Rightarrow \nabla_{\mathbf{w}} ||\mathbf{X}^{(\text{train})}\mathbf{w} - \mathbf{y}^{(\text{train})}||_2^2 = 0$$

$$\mathbf{w} = (\mathbf{X}^{(\text{train})t}\mathbf{X}^{(\text{train})})^{-1}\mathbf{X}^{(\text{train})t}\mathbf{y}^{(\text{train})}$$

where $\mathbf{X}^{(\text{train})} = [\mathbf{x}_1^{(\text{train})}, \ldots, \mathbf{x}_N^{(\text{train})}]$ and $\mathbf{y}^{(\text{train})} = [y_1^{(\text{train})}, \ldots, y_N^{(\text{train})}]$.

Generalization

- We care more about the performance of the model on new, previously unseen examples
- The training examples usually cannot cover all the possible input configurations, so the learner has to generalize from the training examples to new cases
- Generalization error: the expected error over **ALL** examples
- To obtain theoretical guarantees about generalization of a machine learning algorithm, we assume all the samples are drawn from a distribution $p(\mathbf{x}, y)$, and calculate generalization error (GE) of a prediction function f by taking expectation over $p(\mathbf{x}, y)$

$$GE_f = \int_{\mathbf{x},y} p(\mathbf{x}, y)\mathbf{Error}(f(\mathbf{x}), y)$$

Generalization

- However, in practice, $p(\mathbf{x}, y)$ is unknow. We assess the generalization performance with a test set $\{\mathbf{x}_i^{(\text{test})}, y_i^{(\text{test})}\}$

$$\text{Performance}_{\text{test}} = \frac{1}{M} \sum_{i=1}^{M} \text{Error}(f(\mathbf{x}_i^{(\text{test})}), y_i^{(\text{test})})$$

- We hope that both test examples and training examples are drawn from $p(\mathbf{x}, y)$ of interest, although it is unknown

Capacity

- The ability of the learner (or called model) to discover a function taken from a family of functions. Examples:
  - Linear predictor
    $$y = wx + b$$
  - Quadratic predictor
    $$y = w_2 x^2 + w_1 x + b$$
  - Degree-10 polynomial predictor
    $$y = b + \sum_{i=1}^{10} w_i x^i$$
  - The latter family is richer, allowing to capture more complex functions
- Capacity can be measured by the number of training examples $\{\mathbf{x}_i^{(\text{train})}, y_i^{(\text{train})}\}$ that the learner **could always fit**, no matter how to change the values of $\mathbf{x}_i^{(\text{train})}$ and $y_i^{(\text{train})}$

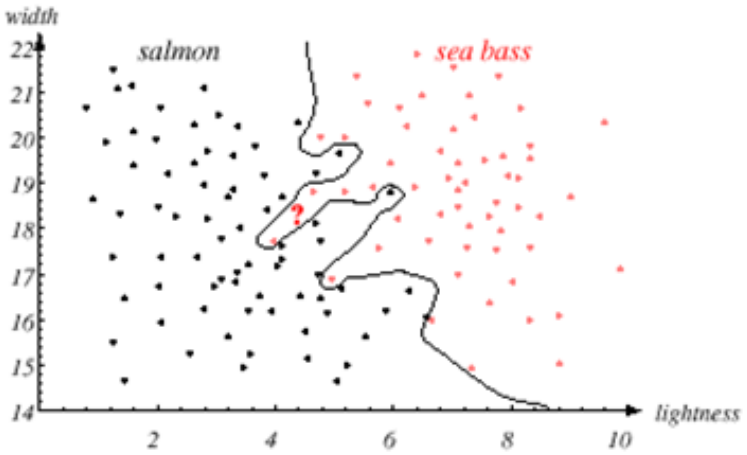
Underfitting

- The learner cannot find a solution that fits training examples well
  - For example, use linear regression to fit training examples $\{\mathbf{x}_i^{(\text{train})}, y_i^{(\text{train})}\}$ where $y_i^{(\text{train})}$ is an quadratic function of $\mathbf{x}_i^{(\text{train})}$
- Underfitting means the learner cannot capture some important aspects of the data
- Reasons for underfitting happening
  - Model is not rich enough
  - Difficult to find the global optimum of the objective function on the training set or easy to get stuck at local minimum
  - Limitation on the computation resources (not enough training iterations of an iterative optimization procedure)
- Underfitting commonly happens in deep learning with large scale training data and could be even a more serious problem than overfitting in some cases

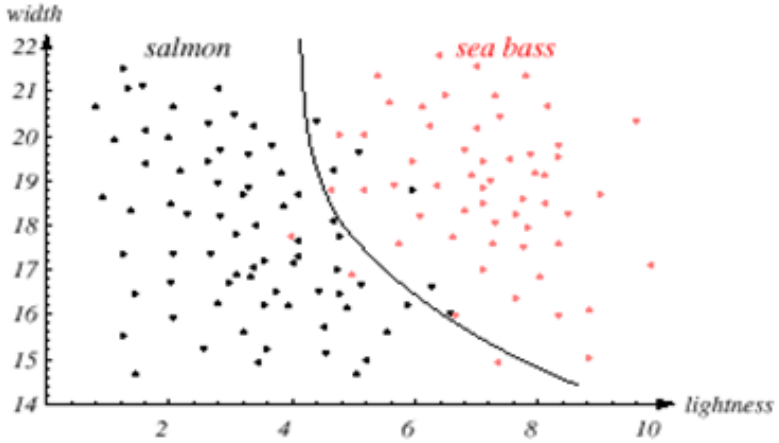
Overfitting

- The learner fits the training data well, but loses the ability to generalize well, i.e. it has small training error but larger generalization error
- A learner with large capacity tends to overfit
  - The family of functions is too large (compared with the size of the training data) and it contains many functions which all fit the training data well.
  - Without sufficient data, the learner cannot distinguish which one is most appropriate and would make an arbitrary choice among these apparently good solutions
  - A separate validation set helps to choose a more appropriate one
  - In most cases, data is contaminated by noise. The learner with large capacity tends to describe random errors or noise instead of the underlying models of data (classes)

Overfitting



Overly complex models lead to complicated decision boundaries. It leads to perfect classification on the training examples, but would lead to poor performance on new examples.



The decision boundary might represent the optimal tradeoff between performance on the training set and simplicity of classifier, therefore giving highest accuracy on new examples.

(Duda et al. Pattern Classification 2000)

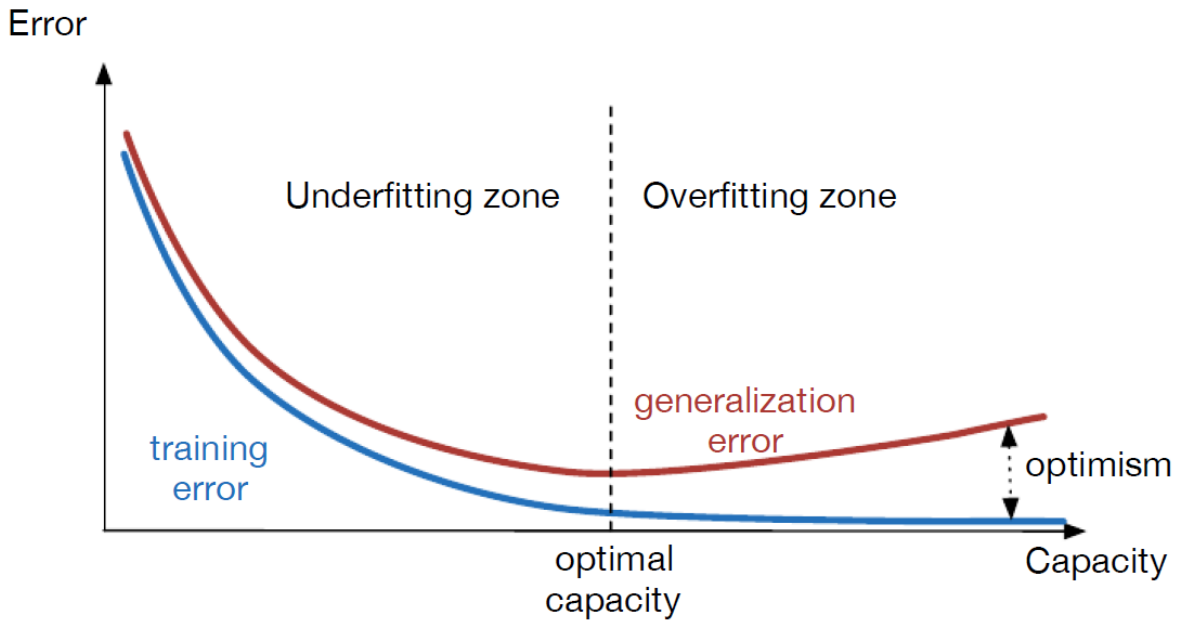
Occam's Razor

- **The fundamental element of machine learning is the trade-off between capacity and generalization**
- Occam's Razor states that among competing functions that could explains the training data, one should choose the "simpler" one. Simplicity is the opposite of capacity.
- Occam's Razor suggests us pick the family of functions just enough large enough to leave only one choice that fits well the data.

Optimal capacity

- Difference between training error and generalization error increases with the capacity of the learner
- Generalization error is a U-shaped function of capacity
- Optimal capacity capacity is associated with the transition from underfitting to overfitting
  - One can use a validation set to monitor generalization error empirically
- Optimal capacity should increase with the number of training examples

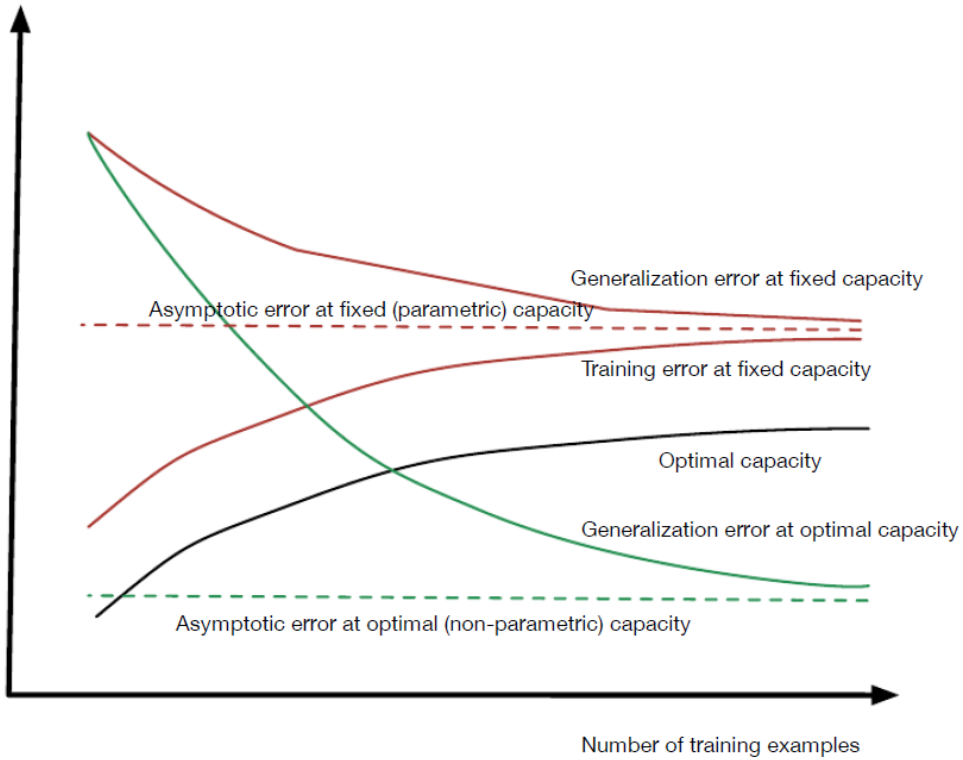
Optimal capacity



Typical relationship between capacity and both training and generalization (or test) error. As capacity increases, training error can be reduced, but the optimism (difference between training and generalization error) increases. At some point, the increase in optimism is larger than the decrease in training error (typically when the training error is low and cannot go much lower), and we enter the overfitting regime, where capacity is too large, above the optimal capacity. Before reaching optimal capacity, we are in the underfitting regime.

(Bengio et al. Deep Learning 2014)

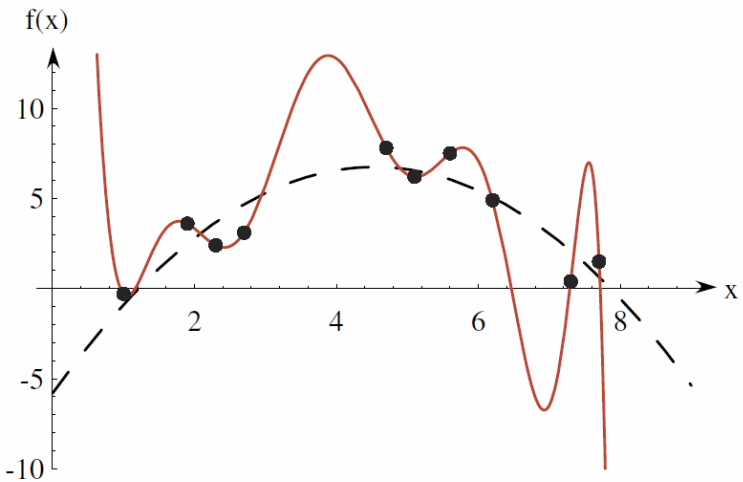
Optimal capacity



As the number of training examples increases, optimal capacity (bold black) increases (we can afford a bigger and more flexible model), and the associated generalization error (green bold) would decrease, eventually reaching the (non-parametric) asymptotic error (green dashed line). If capacity was fixed (parametric setting), increasing the number of training examples would also decrease generalization error (top red curve), but not as fast, and training error would slowly increase (bottom red curve), so that both would meet at an asymptotic value (dashed red line) corresponding to the best achievable solution in some class of learned functions.

(Bengio et al. Deep Learning 2014)

Exercise question



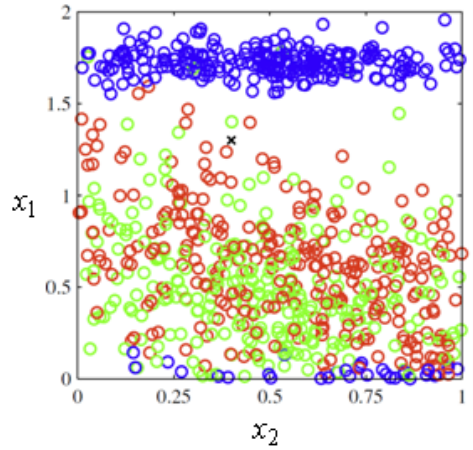
- In the figure above, the training data (10 black dots) were selected from a quadratic function plus Gaussian noise, i.e., $f(x) = w_2 x^2 + w_1 x^2 + b + \epsilon$ where $p(\epsilon) = N(0, \sigma^2)$. The degree-10 polynomial fits the data perfectly. Which learner should be chosen in order to better predict new examples? The second-order function or the 10th degree function?
- If the ten training examples were generated from a 10th degree polynomial plus Gaussian noise, which learned should be chosen?
- If the one million training examples were generated from a quadratic function plus Gaussian noise, which learned should be chosen?

How to reduce capacity?

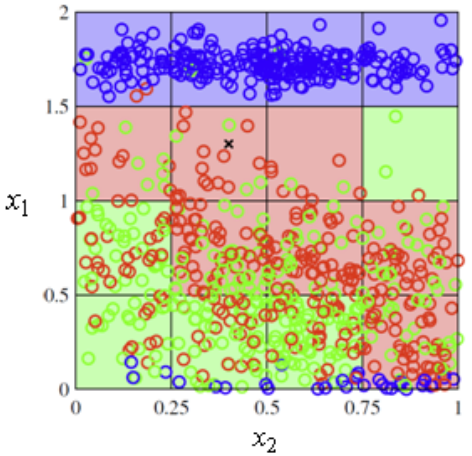
- Reduce the number of features
- Reduce the number of **independent** parameters
- Reduce the network size of deep models
- Reduce the number of training iterations
- Add regularization to the learner
- . . .

Curse of dimensionality

- Why do we need to reduce the dimensionality of the feature space?



Scatter plot of the training data of three classes. Two features are used. The goal is to classify the new testing point denoted by '×'.

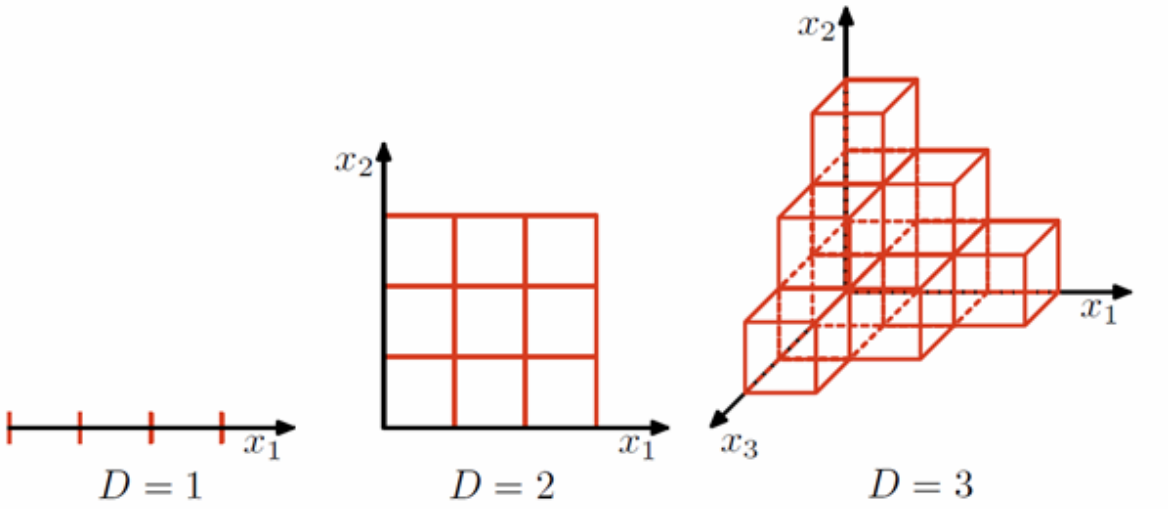


The feature space is uniformly divided into cells. A cell is labeled as a class, if the majority of training examples in that cell are from that class. The testing point is classified according to the label of the cell where it falls in.

(Duda et al. Pattern Classification 2000)

Curse of dimensionality

- The more training samples in each cell, the more robust the classifier
- The number of cells grows exponentially with the dimensionality of the feature space. If each dimension is divided into three intervals, the number of cells is $N = 3^D$
- Some cells are empty when the number of cells is very large!



(Duda et al. Pattern Classification 2000)

Regularization

- Equivalent to imposing a preference over the set of functions that a learner can obtain as a solution
- In Bayesian learning, it is reflected as a prior probability distribution over the space of functions (or equivalently their parameters) that the learn can assess
- Regularization prevents overfitting by adding penalty for complexity
- Training a classifier/regressor is to minimize

  Prediction error on the training set + **regularization**
- Examples
  - The objective function for linear regression becomes

$$\text{MSE}_{\text{train}} + \text{regularization} = \frac{1}{N}\sum_i (\mathbf{w}^t \mathbf{x}_i^{(\text{train})} - y_i^{(\text{train})})^2 + \lambda ||\mathbf{w}||_2^2$$

  - Multi-task learning, transfer learning, dropout, sparsity, pre-training

Function estimation

- We are interested in predicting y from input $\mathbf{x}$ and assume there exists a function that describes the relationship between y and $\mathbf{x}$, e.g. $y = f(\mathbf{x}) + \epsilon$, where ϵ is random noise following certain distribution.
- Prediction function f can be parametrized by a parameter vector θ.
- Estimating $\hat{f}_n$ from a training set $\mathcal{D}_n = \{(\mathbf{x}_1^{(\text{train})}, y_1^{(\text{train})}), \ldots, (\mathbf{x}_n^{(\text{train})}, y_n^{(\text{train})})\}$ is equivalent to estimating $\hat{\theta}_n$ from $\mathcal{D}_n$.
- Since $\mathcal{D}_n$ is randomly generated from a underlying distribution, both $\hat{\theta}$ and $\hat{f}$ are random variables (or vectors, or functions) distributed according to some probability distributions.
- The quality of estimation can be measured by bias and variance compared with the "true" parameter vector θ or function $\hat{f}$
- With a better design of the parametric form of the function, the learner could achieve low generalization error even with small capacity
- This design process typical involves domain knowledge

Bias

$$\text{bias}(\hat{\theta}) = E(\hat{\theta}) - \theta$$

where expectation is over all the train sets of size n sampled from the underlying distribution

- An estimator is called unbiased if $E(\hat{\theta}) = \theta$
- Example: Gaussian distribution. $p(\mathbf{x}_i; \theta) = \mathcal{N}(\theta, \Sigma)$ and the estimator is $\hat{\theta} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^{(\text{train})}$

$$E(\hat{\theta}) = E\left[\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^{(\text{train})}\right] = \frac{1}{n} \sum_{i=1}^{n} E\left[\mathbf{x}_i^{(\text{train})}\right] = \frac{1}{n} \sum_{i=1}^{n} \theta = \theta$$

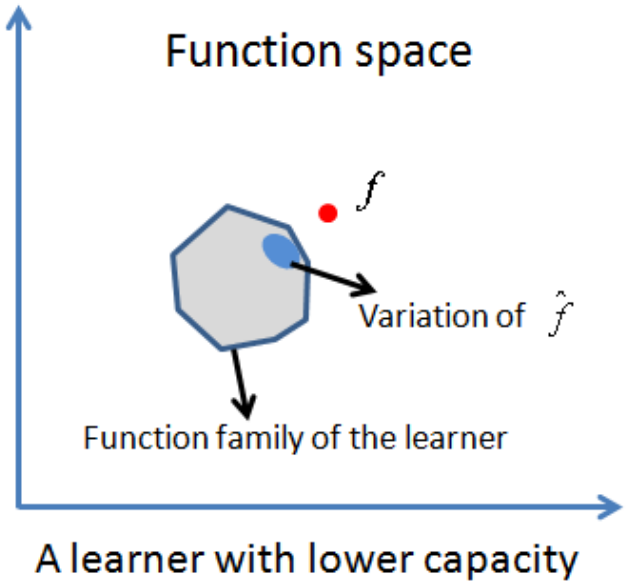
Variance

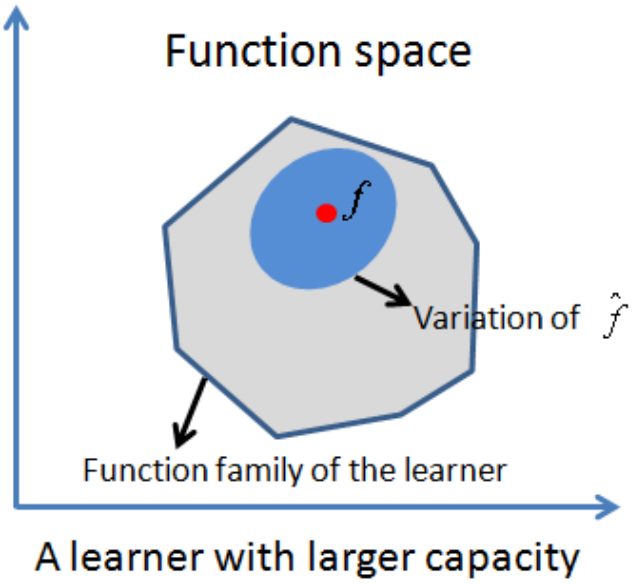
$$\text{Var}[\hat{\theta}] = E[(\hat{\theta} - E[\hat{\theta}])^2] = E[\hat{\theta}^2] - E[\hat{\theta}]^2$$

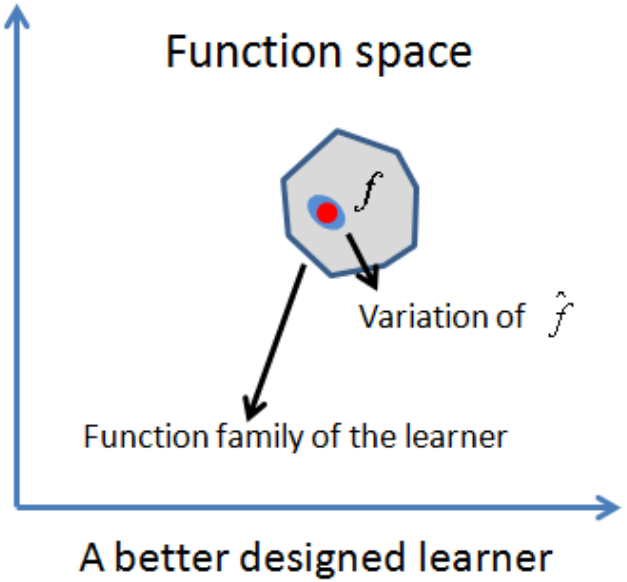
- Variance typically decreases as the size of the train set increases
- Both bias and variance are the sources of estimation errors

$$\text{MSE} = E[(\hat{\theta} - \theta)^2] = \text{Bias}(\hat{\theta})^2 + \text{Var}[\hat{\theta}]$$

- Increasing the capacity of a learner may also increase variance, although it has better chance to cover the true function


Function space
f
Variation of $\hat{f}$
Function family of the learner
A learner with lower capacity


Function space
f
Variation of $\hat{f}$
Function family of the learner
A learner with larger capacity

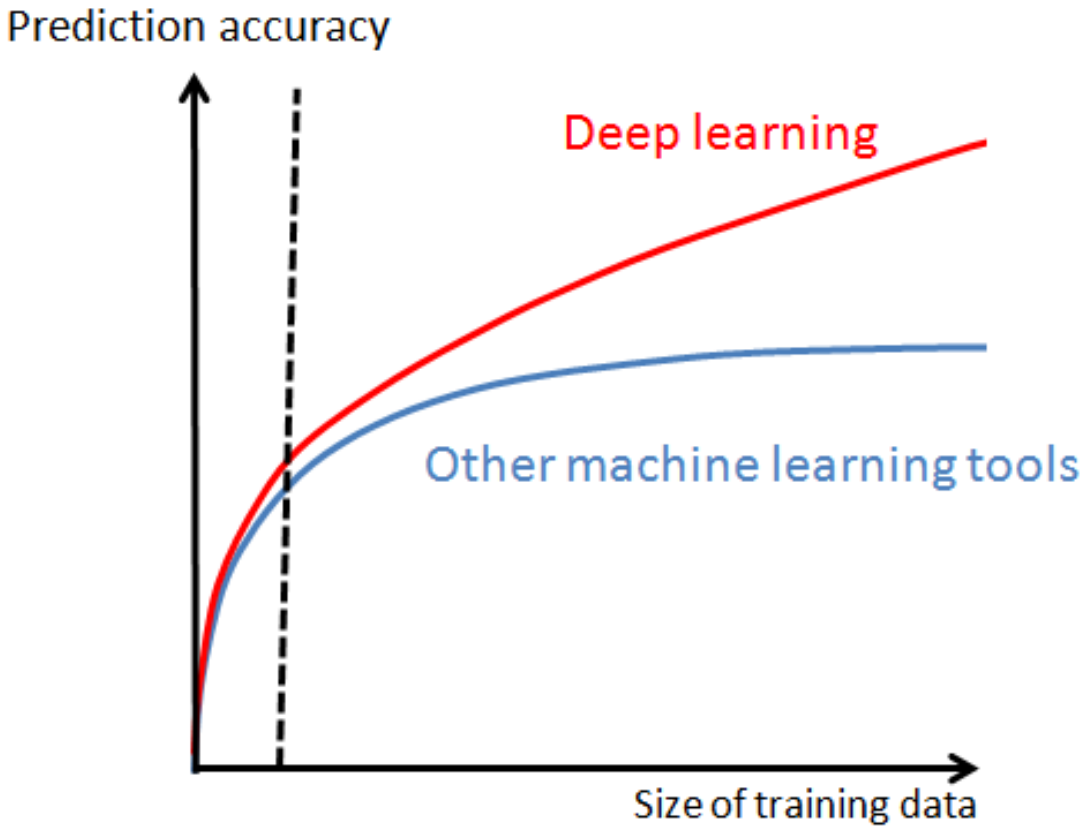

Function space
f
Variation of $\hat{f}$
Function family of the learner
A better designed learner

Summary: issues to be concerned in machine learning

- Effective optimization methods and models to address the underfitting problem
- How to balance the trade-off between capacity and generalization?
- How to effectively reduce capacity (which means also reducing estimation variance) without increasing the bias much?
- For machine learning with big training data, how to effectively increase capacity to cover or get closer to the true function to be estimated?

Open discussion

- Why does deep learning have different behavior than other machine learning methods for large scale training?



Discriminative model

- Directly model $P(y|\mathbf{x})$ and decision boundaries
- Learn the discriminative functions $g_k(\mathbf{x})$

$$y = \arg\max_k g_k(\mathbf{x})$$

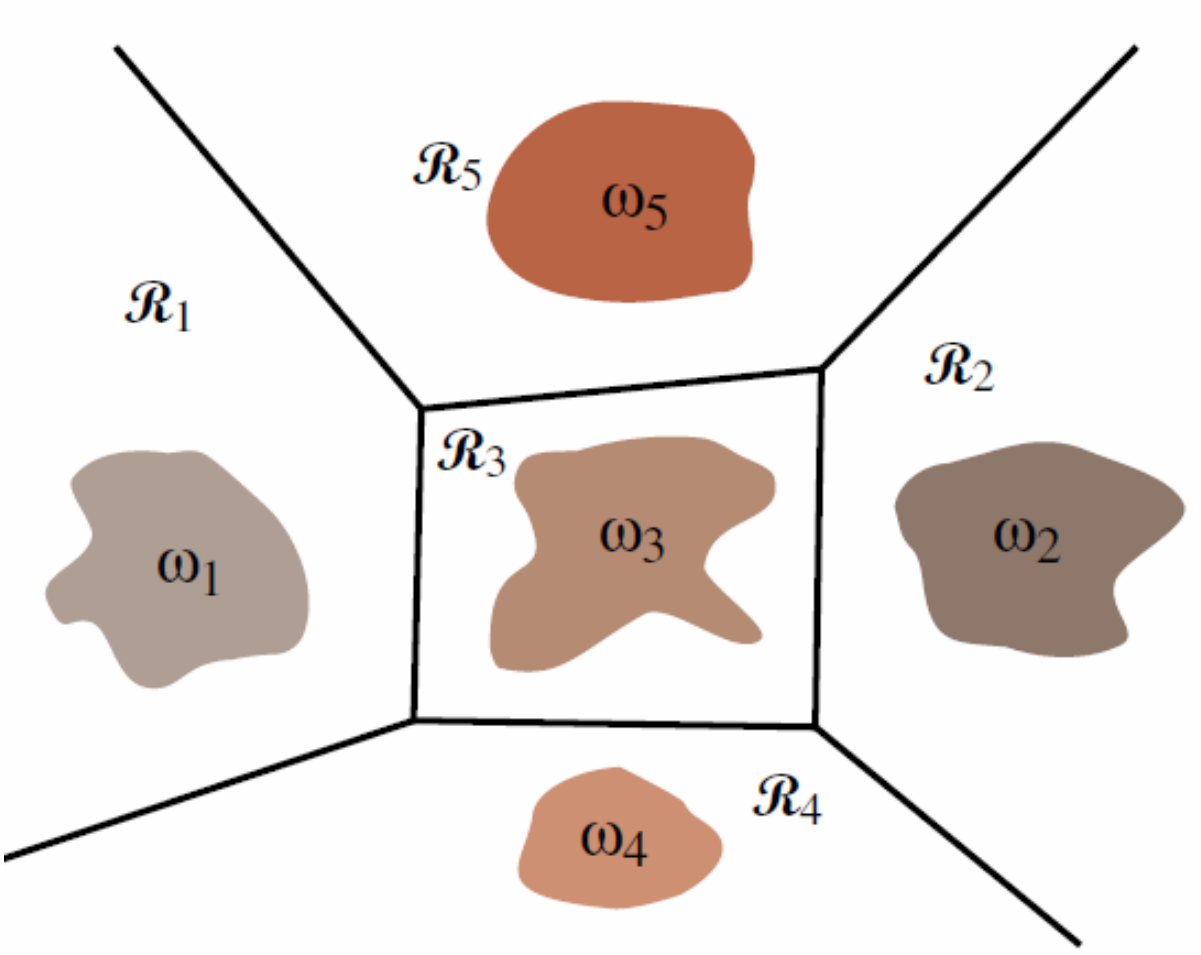
- In the linear case, $g_k(\mathbf{x}) = \mathbf{w}_k^t \mathbf{x}$
- $P(y|\mathbf{x})$ can be estimated from the linear discriminant functions

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{w}_j^t \mathbf{x}}}{\sum_{k=1}^{K} e^{\mathbf{w}_k^t \mathbf{x}}}$$

  It is also called softmax function
- Examples: SVM, boosting, K-nearest-neighbor

Discriminative model

- It is easier for discriminative models to fit data



Discriminative model

- Parameter $\theta = \{\mathbf{w}_k\}$ can be estimated from maximizing the data likelihood

$$\hat{\theta} = \arg\max_{\theta} P(\mathcal{D}_n|\theta) = \arg\max_{\theta} \prod_{i=1}^{n} P(y_n^{(\text{train})}|\mathbf{x}_n^{(train)}, \theta)$$

- *Maximum a posteriori* (MAP) estimation

$$\theta = \arg\max_{\theta} p(\theta|\mathcal{D}_n) = \arg\max_{\theta} \log P(\mathcal{D}_n|\theta) + \log p(\theta)$$

- According to the Bayes' rule, i.e., $p(\theta|\mathcal{D}_n) = P(\mathcal{D}_n|\theta)p(\theta)/P(\mathcal{D}_n)$,

$$\theta = \arg\max_{\theta} \log P(\mathcal{D}_n|\theta) + \log p(\theta)$$

$$\theta = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y_n^{(\text{train})}|\mathbf{x}_n^{(\text{train})}, \theta) + \log p(\theta)$$

- prior $p(\theta)$ corresponds to a regularizer, e.g.

$$p(\theta) = \mathrm{e}^{-\lambda||\theta||^2}$$

Generative model

- Estimate the underlying class conditional probability densities $p(\mathbf{x}|y=k)$ and then construct the classifier using the Bayesian decision theory

$$P(y=k|\mathbf{x}) = \frac{p(\mathbf{x}|y=k)P(y=k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y=k)P(y=k)}{\sum_{k'=1}^{K} p(\mathbf{x}|y=k')P(y=k')}$$

- $p(\mathbf{x}|y)$ and $P(y)$ are parameterized by θ
- Prior $P(y)$ can be used to model the dependency among predictions, such as the segmentation labels of pixels or predictions of speech sequences.
- It is more difficult to model class conditional probability densities. However, it also adds stronger regularization to model fitting, since the learned model not only needs to predict class labels but also generate the input data.
- It is easier to add domain knowledge when desgining the models of $p(\mathbf{x}|y)$

Supervised and unsupervised learning

- Supervised learning: the goal is to use input-label pairs, $(\mathbf{x}, y)$ to learn a function f that predicts a label (or a distribution over labels) given the input, $\hat{y} = f(\mathbf{x})$
- Unsupervised learning: no label or other target is provided. The data consists of a set of examples $\mathbf{x}$ and the objective is to learn about the statistical structure of $\mathbf{x}$ itself.
- Weakly supervised learning: the training data contains $(\mathbf{x}, y)$ pairs as in supervised learning, but the labels y are either unreliably present (i.e. with missing values) or noisy (i.e. where the label given is not the true label)

Unsupervised learning

- Find the "best" representation of data that reserves as much information about **x** as possible while being "simpler" than **x**

  Taking linear case as an example

$$\tilde{\mathbf{x}} = a_0 + \sum_{i=1}^{d'} a_i \mathbf{e}_i$$

  - Lower dimensional representation: $d' < d$
  - Sparse representation: the number of non-zero a_i is small
  - Independent representation: disentangle the sources of variations underlying the data distributions such that the dimensions of the representation are statistically independent, i.e. a_i and a_j are statistically independent

- Deep learning is to learn data representation, but in a nonlinear and hierarchical way

Smoothness Prior

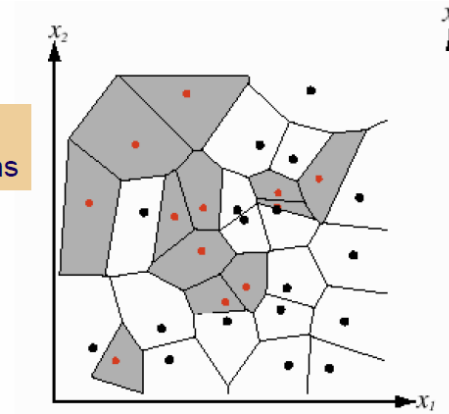
- Shallow models assume smoothness prior on the prediction function to be learned, i.e.

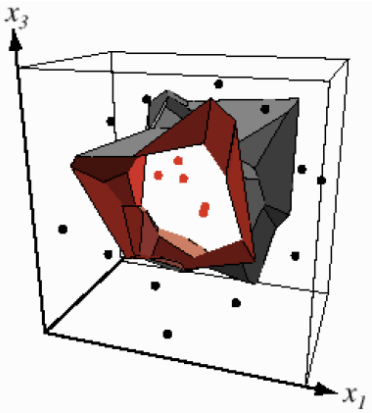
$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \epsilon)$$

  where ϵ is a small change.

- K-nearest neighbor predictors assume piecewise constant
  - For classification and $K = 1$, $f(\mathbf{x})$ is the output class associated with the nearest neighbor of $\mathbf{x}$ in the training set
  - For regression, $f(\mathbf{x})$ is the average of the outputs associated with the K nearest neighbors of $\mathbf{x}$
  - The number of distinguishable regions cannot be more than the number of training examples

2-dimensions





3-dimensions

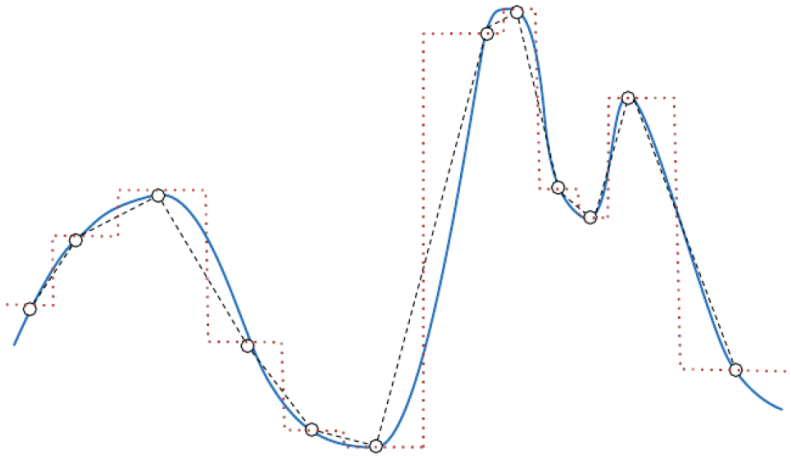
Interpolation with Kernel

$$f(\mathbf{x}) = b + \sum_{i=1}^{n} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

K is a kernel function, e.g., the Gaussian kernel

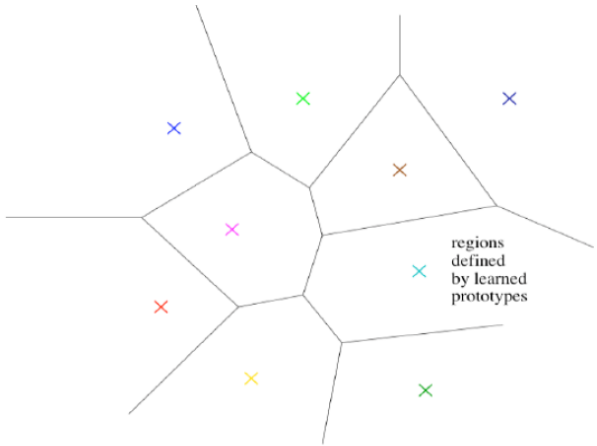
$$K(\mathbf{u}, \mathbf{v}) = N(\mathbf{u} - \mathbf{v}; 0, \sigma^2 I)$$

- b and α_i can be learned by SVM
- Treat each $\mathbf{x}_i$ is a template and the kernel function as a similarity function that matches a template and a test example



Local Representation

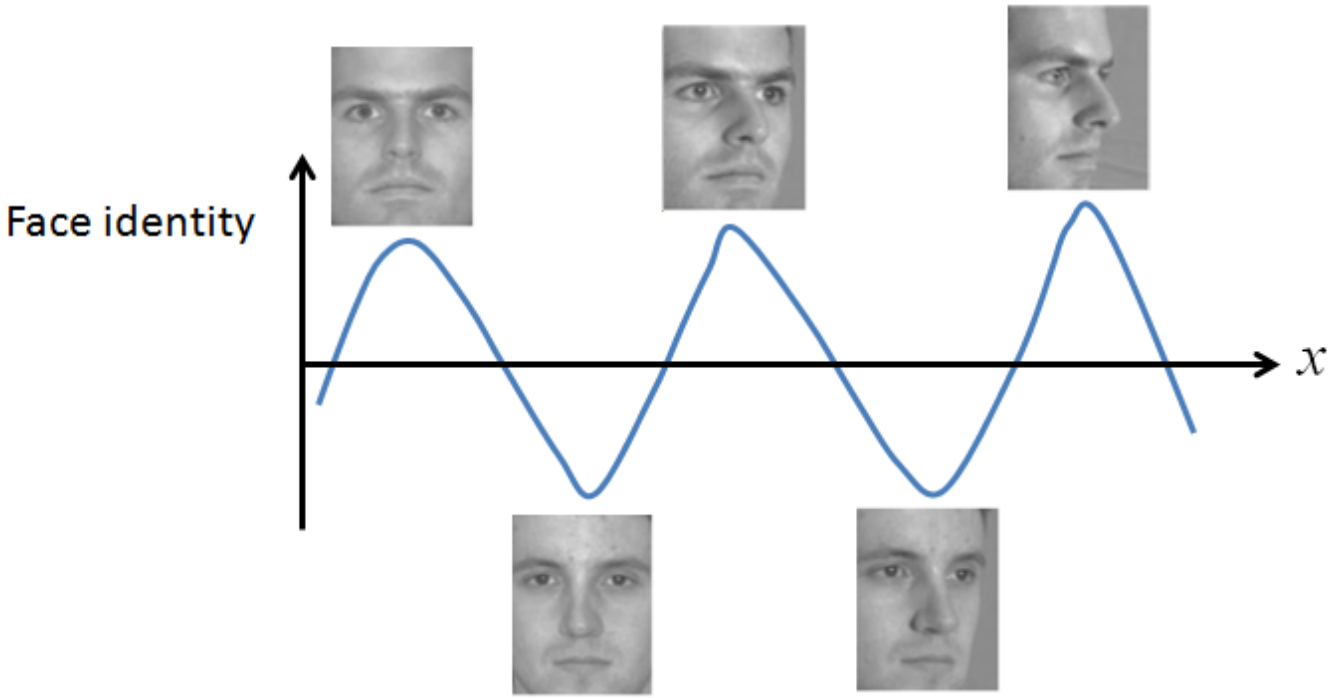
- One can think of the training samples as control knots which locally specify the shape of the prediction function
- The smoothness prior only allows the learner to generalize locally. If $(\mathbf{x}_i, y_i)$ is a supervised training example and $\mathbf{x}_i$ is a near neighbor of $\mathbf{x}$, we expect that $f(\mathbf{x}) \approx y_i$. Better generalization can be obtained with more neighbors.
- To distinguish $O(N)$ regions in the input space, shallow models require $O(N)$ examples (and typically there are $O(N)$ parameters associated with the $O(N)$ regions).



LOCAL PARTITION

Local Representation

- If the function is complex, more regions and more training samples are required.
- The representation learned by deep models can be generalized non-locally



Reference

Yoshua Bengio, Ian Goodfellow and Aaron Courville, Chapter 1 “Machine Learning Basics” in “Deep Learning,” Book in preparation for MIT Press, 2014.

http://www.iro.umontreal.ca/ bengioy/dlbook