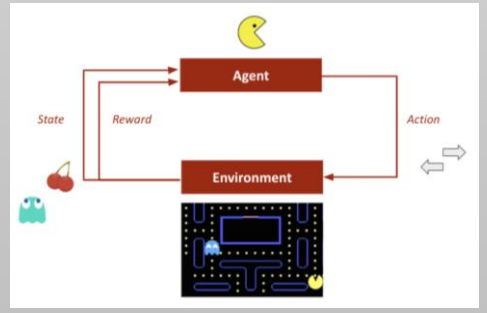
Linear Models for Regression

Sadegh Eskandari
Department of Computer Science, University of Guilan

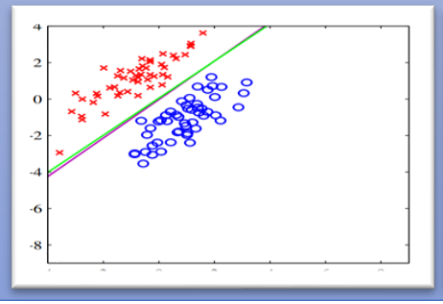
Remember: Algorithms that Can Learn

(Reinforcement)

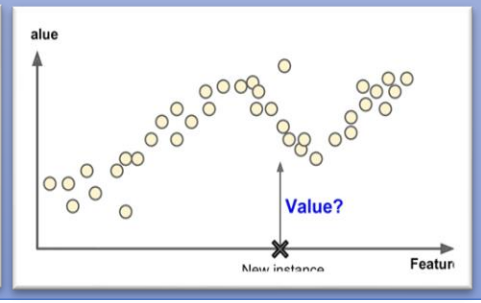


(Supervised)

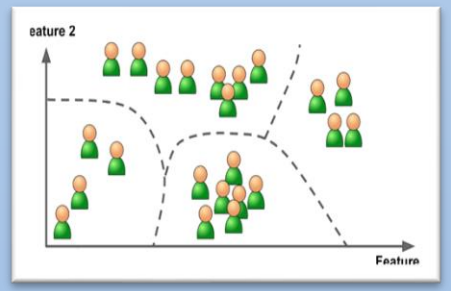
(Classification)



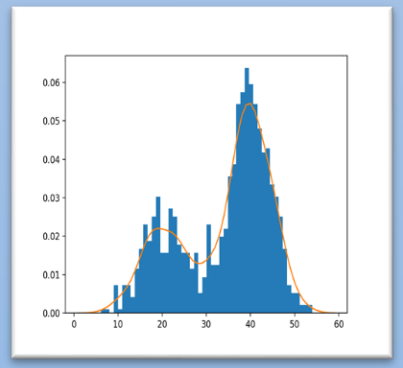
(Regression)



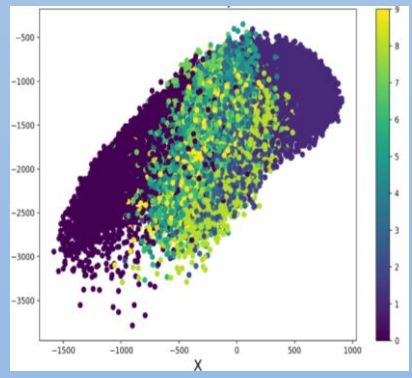
(Unsupervised)



(Clustering)



(Density Estimation)



(Visualization)

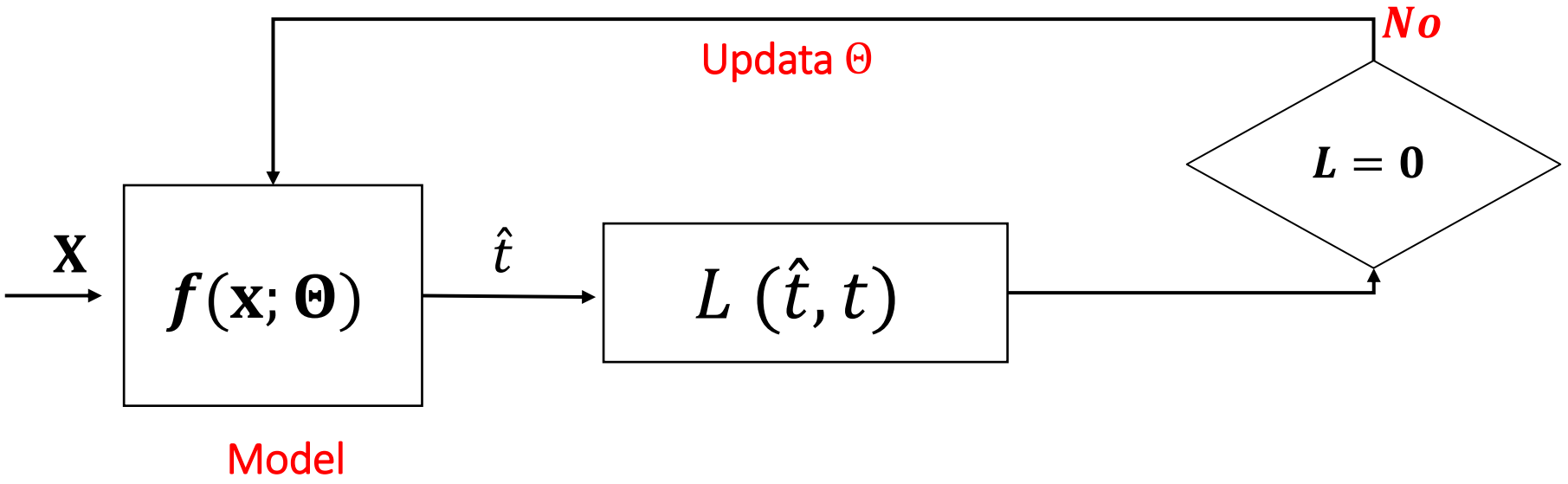
Remember: Supervised Learning

- Suppose that we are given a training set comprising N observations of random variable x (training set) :

$$\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_N})^{\boldsymbol{T}}$$

- Moreover, for each observation $\boldsymbol{x_i}$ we are given a target value t_i (training target):

$$\mathbf{t} = (t_1, t_2, \dots, t_N)^{\boldsymbol{T}}$$



Remember: Polynomial Curve Fitting

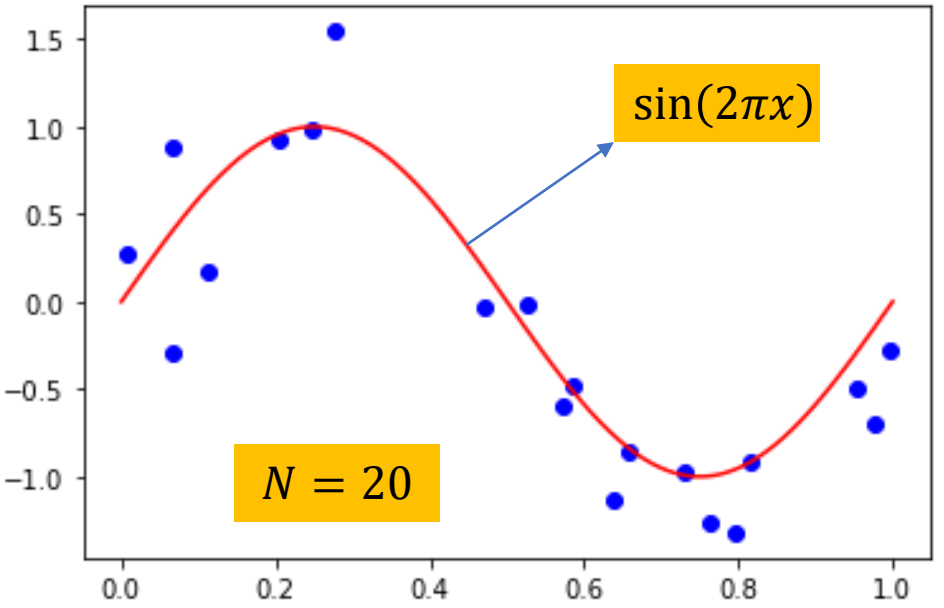
- $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$ is generated uniformaly in [0,1].
- $\boldsymbol{t} = \{t_i | \, t_i = \sin(2\pi x) + \mathcal{N}(0,0.3)\,, i = 1, 2, \dots, N\}$

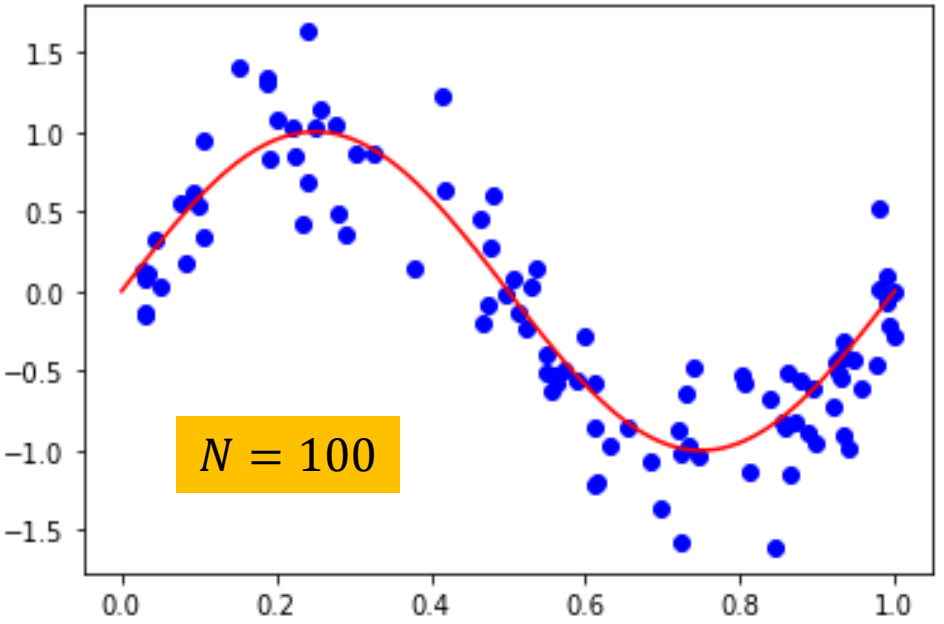
- The generating function in not known and the aim is to estimate it such that:
  - The estimated function should describe the training data
  - The estimated function should generalize to new data

- In particular, we shall fit the data using a polynomial function of the form

  $$y(\mathbf{x}; \boldsymbol{w}) = w_0 + w_1\mathbf{x} + w_2\mathbf{x}^2 + \cdots + w_M\mathbf{x}^M$$

  - M: the order of polynomial
  - $w \equiv [w_0, w_1, \dots, w_M]$: The model parameters (unknown in advance)
- $y(\mathbf{x}, \boldsymbol{w})$ is a linear function of the coefficients $\boldsymbol{w}$. Such functions are called linear models.

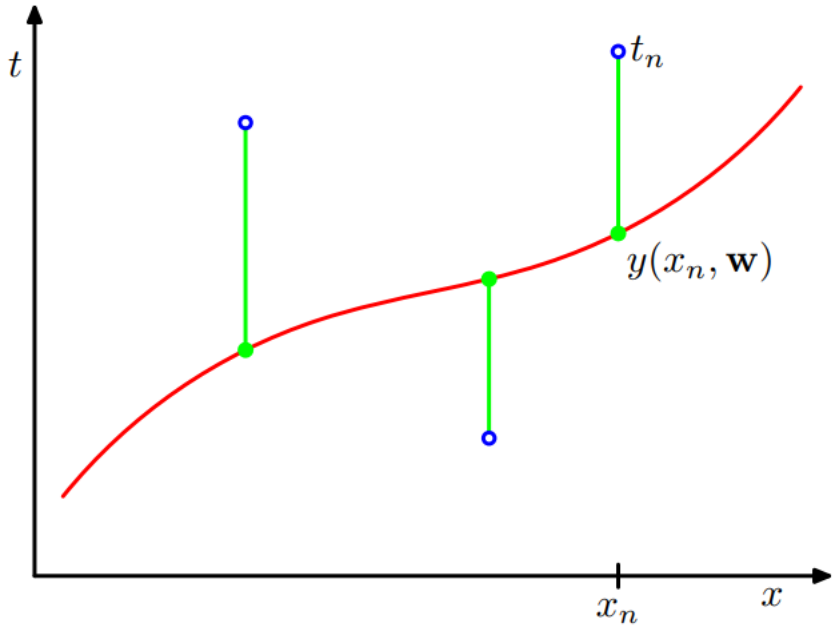




Remember: Polynomial Curve Fitting

- An error function (loss function) is required to measure the misfit between the function $y(\mathbf{x}, \boldsymbol{w})$, for any given $\boldsymbol{w}$, and the training data points.

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \boldsymbol{w}) - t_n\}^2$$



- $E(\boldsymbol{w})$ is a quadratic function of $\boldsymbol{w}$,
- Therefore $\frac{\partial E}{\partial \boldsymbol{w}}$ is linear in the elements of $\boldsymbol{w}$, and so the minimization of the error function has a unique solution, which can be found in closed form.

Linear Basis Function Models

Linear Basis Function Models

- Linear Regression

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

- Functions $\phi_j(\mathbf{x})$ are known as basis functions
- The parameter w_0 allows for any fixed offset in the data and is sometimes called a bias parameter
- It is often convenient to define an additional dummy basis function $\phi_0(\mathbf{x}) = 1$ so that

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

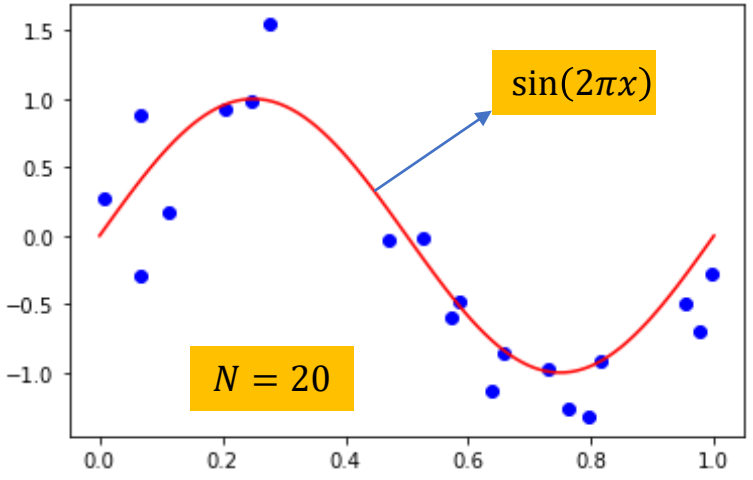
  - $\mathbf{w} = (w_0, w_1, \ldots, w_{M-1})^{\mathrm{T}}$
  - $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{M-1})^{\mathrm{T}}$

Linear Basis Function Models

- Variant of basis functions:
  - $\phi_j(\mathbf{x}) = \mathbf{x}$ (also linear with respect to **x**)
    - **Limitation:** unable to model non-linear data
  - $\phi_j(\mathbf{x}) = \mathbf{x}^j$ (polynomial)
    - **Limitation:** polynomial basis functions are global functions of the input variable, so that changes in one region of input space affect all other regions.
    - This can be resolved by dividing the input space up into regions and fit a different polynomial in each region, leading to spline functions
  - $\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$ (Gaussian basis function)
  - $\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$ where $\sigma(a) = \frac{1}{1+\exp(-a)}$ (Sigmoidal basis function)
- The analysis here is independent of the particular choice of basis function set 😐

Maximum likelihood and least squares

- **Remember:**
  - $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$ is generated uniformly in [0,1].
  - $\boldsymbol{t} = \{t_i | \, t_i = \sin(2\pi x) + \mathcal{N}(0,0.3)\,, i = 1, 2, \dots, N\}$

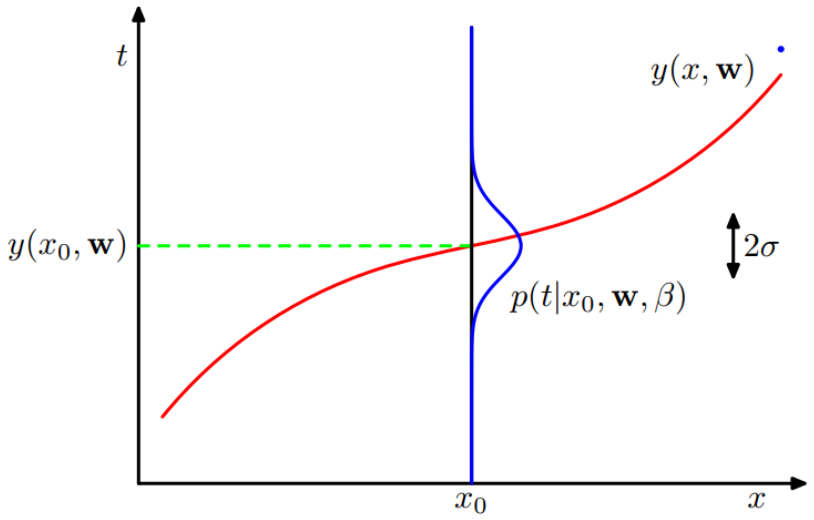


- Assume that the target variable t is given by a deterministic function $y(\mathbf{x}, \mathbf{w})$ with additive Gaussian noise so that

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon$$

  - ϵ is a zero mean Gaussian random variable with precision β.
  - Thus

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$



Maximum likelihood and least squares

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \qquad y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

- Consider a data set of inputs $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with corresponding target values $\mathbf{t} = \{t_1, \ldots, t_N\}$.
- Assuming that the data points are iid, the likelihood function is expressed as:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1})$$

- Note: in supervised learning problems (such as regression and classification), we are not seeking to model the distribution of the input variables.
  - **x** will always appear in the set of conditioning variables,
  - we will drop the **x** from expressions such as $p(t|\mathbf{x}, \boldsymbol{w}, \beta)$ in order to keep the notation uncluttered.

Maximum likelihood and least squares

$$p(\mathbf{t}|\mathbf{w},\beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n),\beta^{-1})$$

- Taking the logarithm of the likelihood function, and making use of the standard form for the univariate Gaussian, we have

$$\ln p(\mathbf{t}|\mathbf{w},\beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n),\beta^{-1}) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta\left(\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2\right)$$

Remember: Sum of Squares Error ($E_D(\boldsymbol{w})$)

- Therefore

$$\boldsymbol{w}_{ML} = arg \max_{\boldsymbol{w}} \ln p(\mathbf{t}|\mathbf{w},\beta)$$

$$\frac{\partial \ln p(\mathbf{t}|\mathbf{w},\beta)}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w}_{ML} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}$$

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

- $\boldsymbol{\Phi}$: The Design Matrix
- $\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}$: The Normal Equation for the least square problems
- $\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}$: The Moore-Penrose pseudo-inverse (a generalization of the of matrix inverse to nonsquare matrices)

Sequential Learning

$$\boldsymbol{w}_{ML} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}$$

- Maximum likelihood method is a batch technique, which involves processing the entire training set in one go.
- Then, it can be computationally costly for large data sets.
- If the data set is sufficiently large, it may be worthwhile to use sequential algorithms, also known as on-line algorithms
- The most well-known sequential learning technique is stochastic gradient descent (also known as sequential gradient descent)
- If the error function comprises a sum over data points $E = \sum_n E_n$, then after presentation of pattern n, the stochastic gradient descent algorithm updates the parameter vector $\boldsymbol{w}$ using

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n$$

- τ: The iteration number
- η: The learning rate parameter
- The value of $\boldsymbol{w}$ is initialized to some starting vector $\boldsymbol{w}^{(0)}$

Sequential Learning

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n \qquad \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

- For the case of the sum-of-squares error function, this gives

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\left(t_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\right)\boldsymbol{\phi}(\mathbf{x}_n)$$

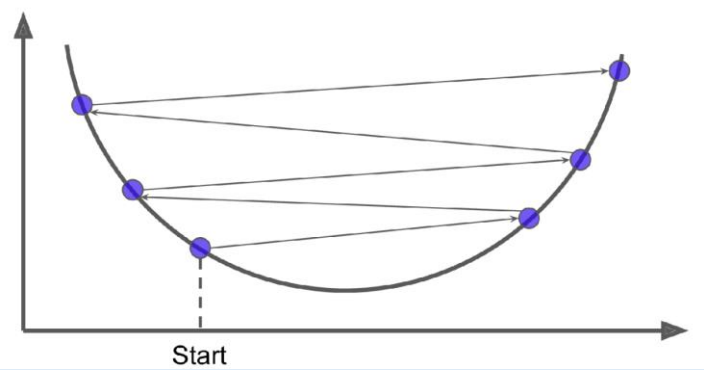
  - This is known as least-mean-squares or the LMS algorithm.

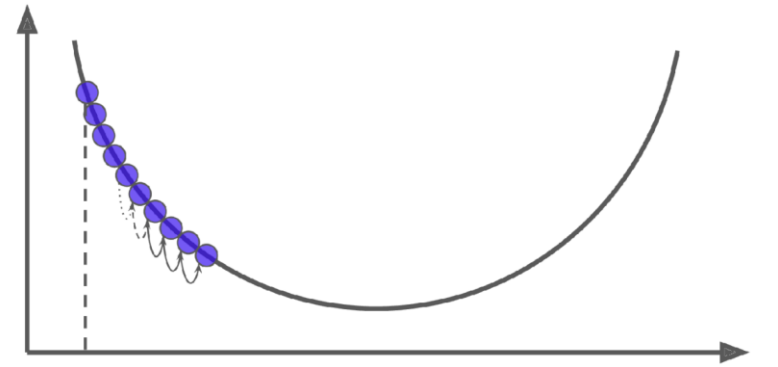
Sequential Learning

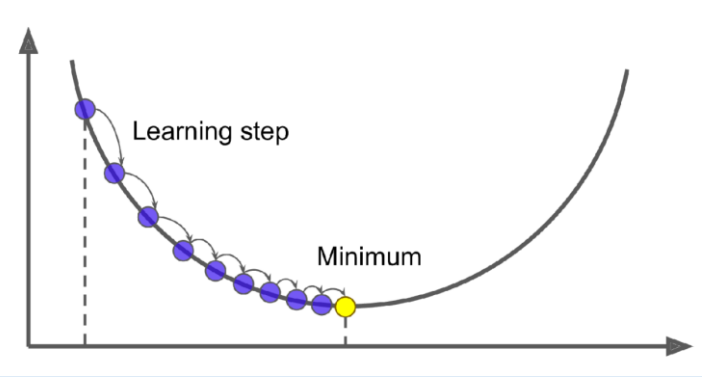
$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n$$

- The value of η needs to be chosen with care to ensure that the algorithm converges

- ❑ Very large learning rate: the algorithm diverges (left figure)
- ❑ Very small learning rate: the algorithm takes long time to converge (middle figure)
- ❑ Best: the learning step size is proportional to the slope of the cost function, so the steps gradually get smaller as the parameters approach the minimum (right figure)







Figures from hands-on machine learning (Aurélien Géron)

Regularized Least Squares

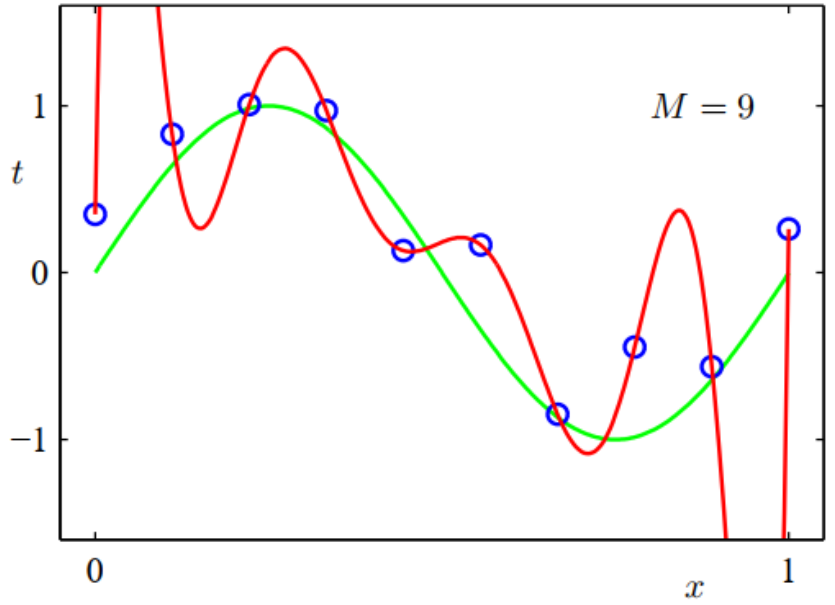
- **Remember**: We introduced the idea of regularization to control overfitting.
- In regularization technique, the total error function to be minimized takes the form

$$E_D(\boldsymbol{w}) + \lambda E_w(\boldsymbol{w})$$

- The regularization term takes the following general form:

$$E_w(\boldsymbol{w}) = \frac{1}{2}\sum_{j=1}^{M} \|w_j\|^q$$

  - $q = 1$(lasso): generally results sparse models
  - $q = 2$: (weight decay in machine learning literature and parameter shrinkage in statistics): encourages weight values to decay towards zero generally results sparse models



	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Multiple Outputs

- We have considered the case of a single target variable t.
- In some applications, we may wish to predict $K > 1$ target variables, which we denote collectively by the target vector $\mathbf{t}$.
- Two approaches can be used for this problem:
  - Introducing a different set of basis functions for each component of t, leading to multiple, independent regression problems.
  - Using the same set of basis functions to model all of the components of the target vector (this is a more interesting approach)
- For the case of second approach:

$$\boldsymbol{y}(\mathbf{x}, \boldsymbol{w}) = \mathbf{W}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$$

  - $\boldsymbol{y}$: a K-dimensional column vector
  - $\mathbf{W}$: an $M \times K$ matrix of parameters
  - $\boldsymbol{\phi}(\mathbf{x})$: an M-dimensional column vector with elements $\phi_j(\mathbf{x})$, with $\phi_0(\mathbf{x}) = 1$

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix}}_{\boldsymbol{y}} = \underbrace{\begin{bmatrix} w_{01} & w_{11} & \cdots & w_{M1} \\ w_{02} & w_{12} & \cdots & w_{M2} \\ \vdots & \vdots & & \vdots \\ w_{0K} & w_{1K} & \cdots & w_{MK} \end{bmatrix}}_{\mathbf{W}^{\mathrm{T}}} \underbrace{\begin{bmatrix} 1 \\ \phi_1 \\ \\ \vdots \\ \\ \phi_M \end{bmatrix}}_{\boldsymbol{\phi}}$$

Multiple Outputs

- Suppose we take the conditional distribution of the target vector to be an isotropic Gaussian of the form

$$p(\mathbf{t}|\mathbf{x}, \mathbf{W}, \beta) = \mathcal{N}(\mathbf{t}|\mathbf{W}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}), \beta^{-1}\mathbf{I})$$

- If we have a set of observations $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N$, we can combine these into a matrix $\mathbf{T}$ of size $N \times K$ such that the n^{th} row is given by $\boldsymbol{t}_n^T$. Similarly, we can combine the input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ into a matrix $\mathbf{X}$.

$$\mathbf{X} = \begin{matrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \\ \mathbf{x}_N \end{matrix} \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1D} \\ x_{21} & x_{22} & \ldots & x_{2D} \\ \vdots & \vdots & & \vdots \\ x_{N1} & x_{N2} & \ldots & x_{ND} \end{bmatrix}$$

$$\mathbf{T} = \begin{matrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \\ \mathbf{t}_N \end{matrix} \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1K} \\ t_{21} & t_{22} & \ldots & t_{2K} \\ \vdots & \vdots & & \vdots \\ t_{N1} & t_{N2} & \ldots & t_{NK} \end{bmatrix}$$

- The log likelihood function is then given by

$$\ln p(\mathbf{T}|\mathbf{X}, \mathbf{W}, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}(\mathbf{t}_n|\mathbf{W}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}\mathbf{I})$$

$$= \frac{NK}{2}\ln\left(\frac{\beta}{2\pi}\right) - \frac{\beta}{2}\sum_{n=1}^{N}\left\|\mathbf{t}_n - \mathbf{W}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\right\|^2$$

$$\frac{\partial \ln p(\mathbf{T}|\mathbf{X}, \mathbf{W}, \beta)}{\partial \mathbf{W}} = 0$$

$$\boldsymbol{W}_{ML} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{T}$$

The Bias-Variance Decomposition

The Bias-Variance Decomposition

- Suppose we model the function $h(\mathbf{x})$ using a parametric function $y(\mathbf{x}, \boldsymbol{w})$ governed by a parameter vector $\boldsymbol{w}$
- For any given data set $\mathcal{D}$, we can run our learning algorithm and obtain a prediction function $y(\mathbf{x}; \mathcal{D})$
- The squared loss of the prediction takes the form

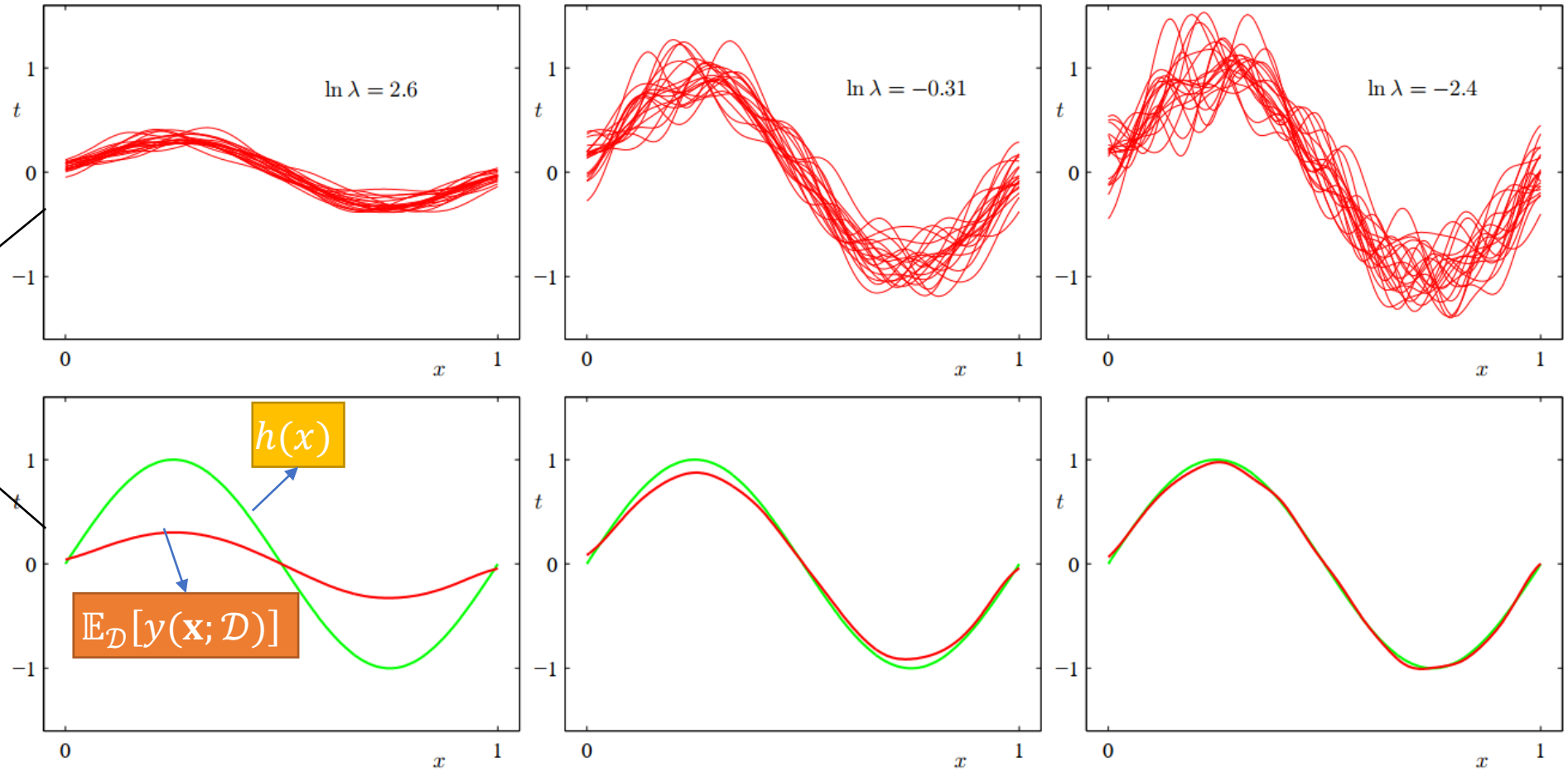
$$\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2$$

- Suppose we have a large number of iid data sets each of size N. Then $\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]$ represents the average prediction function over the ensemble of data sets
- It can be show that the average squared loss of the prediction over the ensemble of data sets takes the form

$$\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2] = \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2}_{\text{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2]}_{\text{Variance}}$$

The Bias-Variance Decomposition

$$\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2] = \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2}_{\text{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2]}_{\text{Variance}}$$



- Low Variance
- High Bias

- High Variance
- Low Bias

The Bias-Variance Decomposition

$$\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2] = \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2}_{\text{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2]}_{\text{Variance}}$$

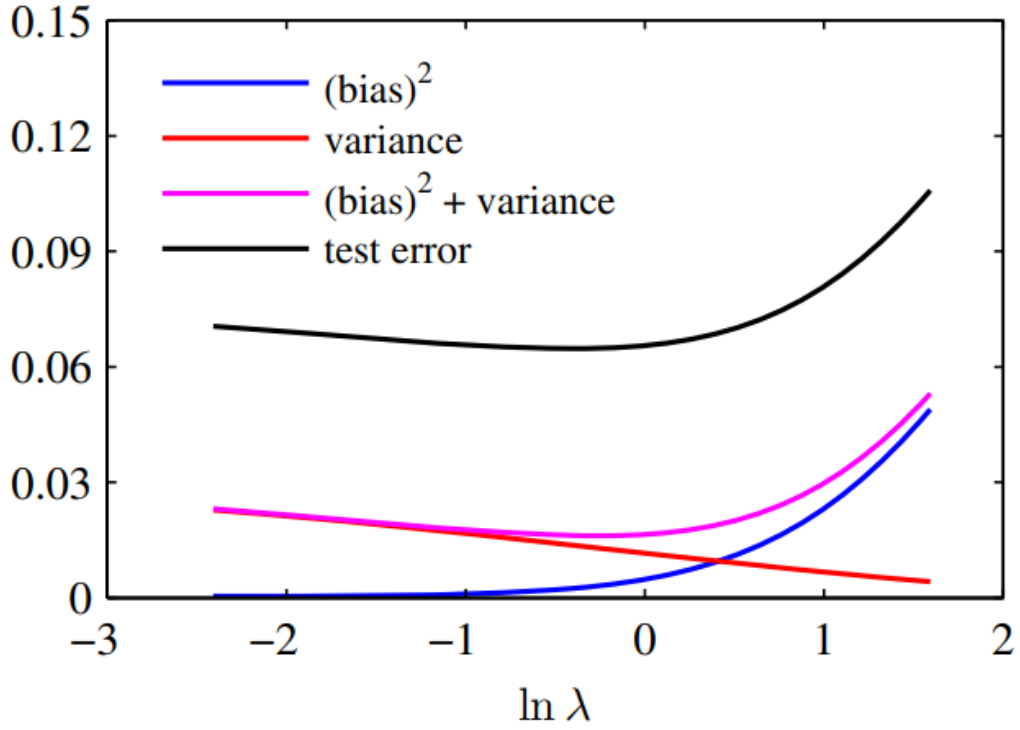


Figure: Plot of Squared bias and variance together with their sum, corresponding to the results in the previous figure. Also shown is the average test set error for a test data set size of 1000 points.

- The minimum value of $(bias)^2 + variance$ o ccurs around $\ln \lambda = -0.31$
- It is close to the value that gives the minimum error on the test data

Model Selection

Model Selection

Remember

- Polynomial curve fitting using regularized least squares

model $y(x; \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M$ Error $E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \boldsymbol{w}) - t_n\}^2 + \frac{\lambda}{2}\|w\|^2$

- **Question**: How to select an appropriate model for data (linear, non-linear, neural network, …)?
- **Question**: For a selected model, how to select its hyper-parameters (M and λ for linear model, num of layers and neurons for a neural network, …)
- **Idea 1:** Choose hyperparameters that work best on the training data

BAD: remember the over-fitting problem in polynomial curve fitting with $M = 9$

- **Idea 2:** Choose hyperparameters that work best on test data

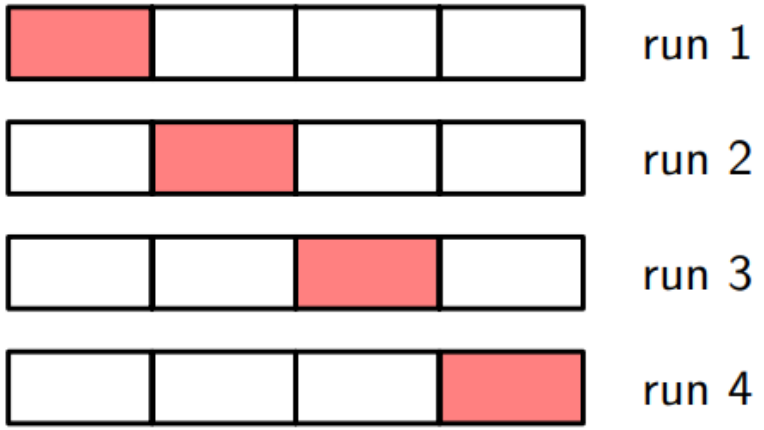
BAD: No idea how algorithm will perform on new data (never do this)

Model Selection

- **Idea 3:** Split data into train, validation; choose hyperparameters on validation and evaluate on test

Better: but not useful for small data sets

- **Idea 4:** Cross-Validation: Split data into folds, try each fold as validation and average the results



Very useful for small data sets, but bad for complex models (models with many hyper-parameters such as neural networks)

Better for small data sets