Pattern Information Processing: Neural Networks

Masashi Sugiyama
(Department of Computer Science)

Contact: W8E-505
sugi@cs.titech.ac.jp
http://sugiyama-www.cs.titech.ac.jp/~sugi/

Linear/Non-Linear Models

- Linear model: $\hat{f}(\boldsymbol{x})$ is linear with respect to $\boldsymbol{\alpha}$

  (Note: not necessarily linear with respect to $\boldsymbol{x}$)

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{b} \alpha_i \varphi_i(\boldsymbol{x})$$

- Non-linear model: Otherwise

Today’s Plan

- Neural networks
- Least-squares in neural networks:
  Error back-propagation algorithm

Non-Linear Models

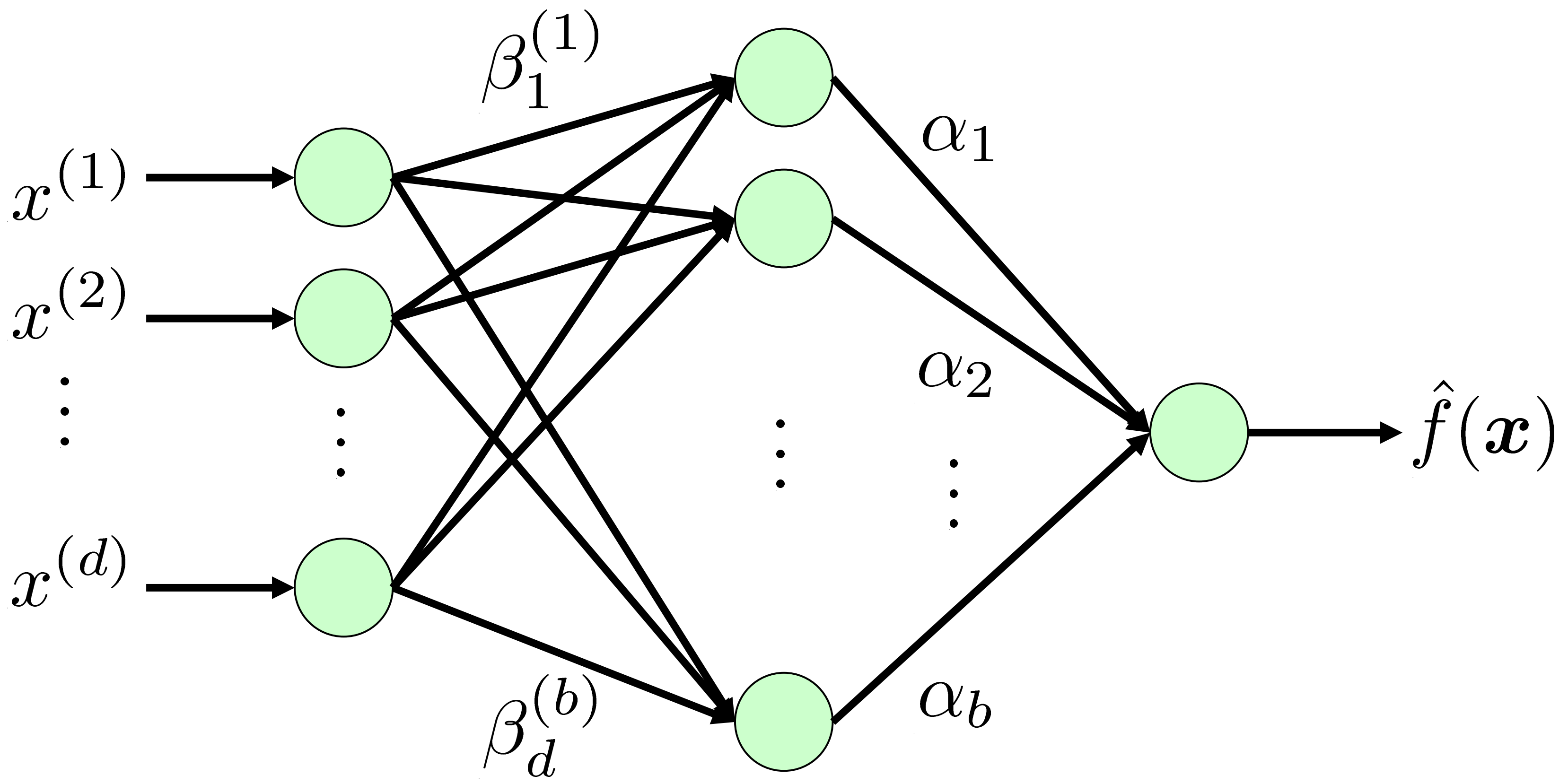
- A popular choice: Hierarchical models

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{b} \alpha_i \varphi(\boldsymbol{x}; \boldsymbol{\beta}_i)$$

- Basis function is parameterized by $\boldsymbol{\beta}_i$

Three-Layer Networks

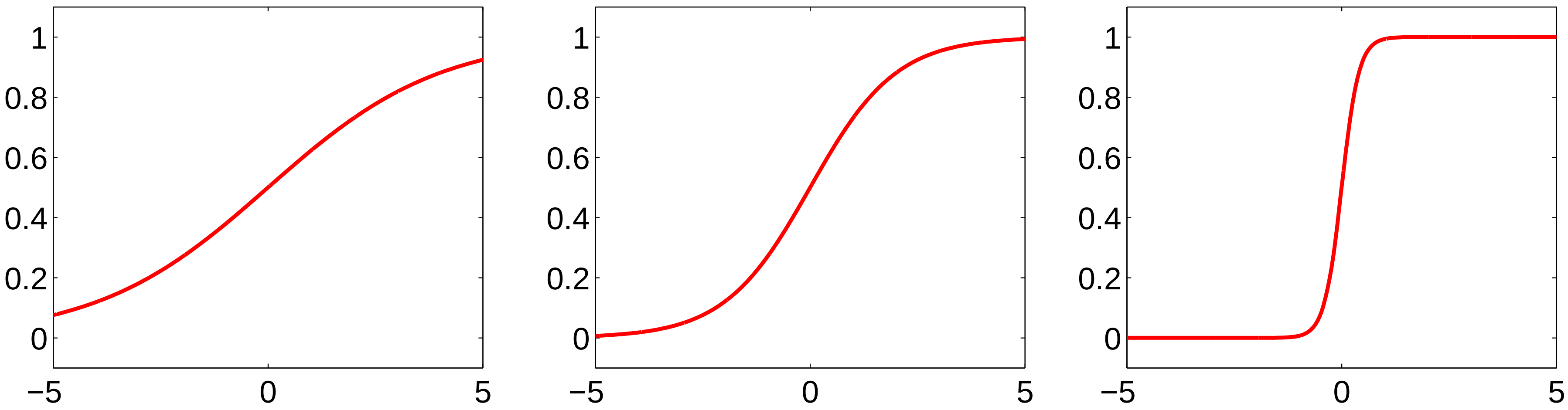
- Such a hierarchical model can be represented as a 3-layer network.



Sigmoidal Function

- A typical basis function: Sigmoidal function

$$\varphi(\boldsymbol{x};\boldsymbol{\beta},h) = \frac{1}{1+\exp\left(-\langle \boldsymbol{x},\boldsymbol{\beta}\rangle - h\right)}$$



β :Small β :Medium β :Large

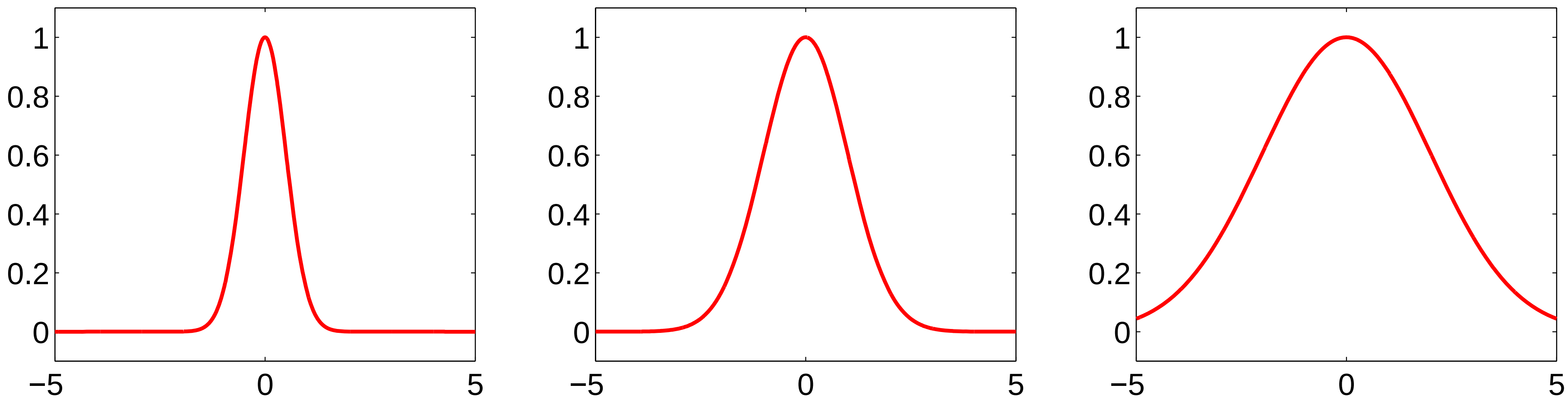
Perceptrons

- The behavior of the sigmoidal functions is similar to the neurons in the brain.
- For this reason, hierarchical models with sigmoidal functions are called artificial neural networks or perceptrons.
- Mathematically, 3-layer neural networks can approximate any continuous functions with arbitrary small error ("universal approximator").

Gaussian Radial Basis Function

- Another popular basis function: Gaussian radial basis function

$$\varphi(\boldsymbol{x};\boldsymbol{\beta},c)=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{\beta}\|^2}{2c^2}\right)$$



c :Small c :Medium c :Large

Least-Squares Learning

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{b} \alpha_i \varphi(\boldsymbol{x}; \boldsymbol{\beta}_i)$$

$$\boldsymbol{w} = (\boldsymbol{\alpha}^\top, \boldsymbol{\beta}_1^\top, \boldsymbol{\beta}_2^\top, \ldots, \boldsymbol{\beta}_b^\top)^\top.$$

- Least-squares learning is often used for training hierarchical models.

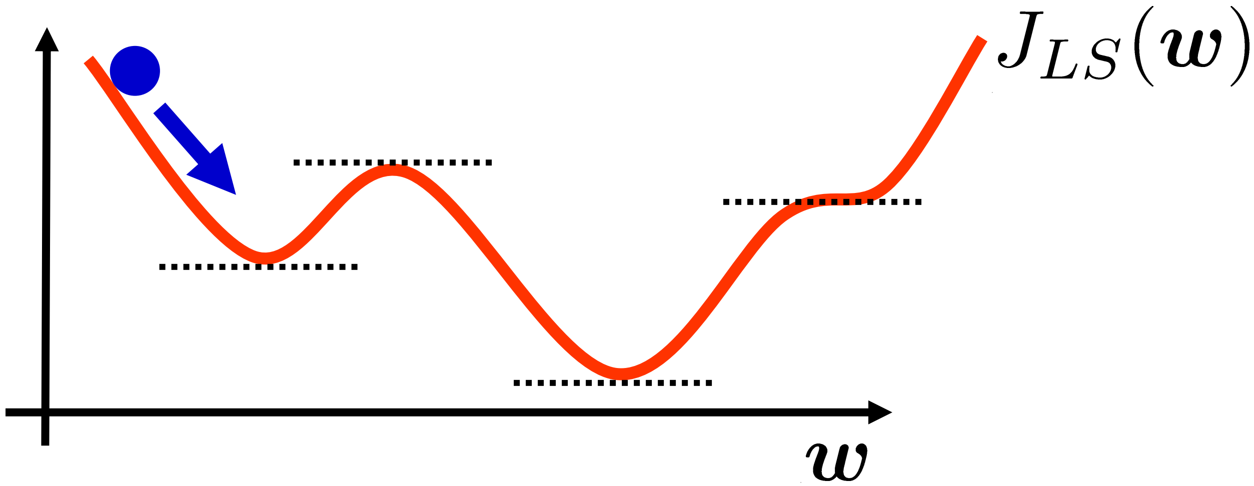
$$\min_{\boldsymbol{w}} J_{LS}(\boldsymbol{w})$$

$$J_{LS}(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\hat{f}(\boldsymbol{x}_i) - y_i \right)^2$$

How to Obtain Solutions

- No analytic solution is known.
- Simple gradient search is usually used.

$$\widehat{\boldsymbol{w}}^{new} \longleftarrow \widehat{\boldsymbol{w}}^{old} - \varepsilon \nabla J_{LS}(\widehat{\boldsymbol{w}}^{old})$$



- It converges to one of the local minima.

Error Back-Propagation

- Efficient calculation of gradient for Sigmoidal basis functions

$$\frac{\partial J_{LS}}{\partial \alpha_j} = 2\sum_{i=1}^{n} o_{i,j}\left(\hat{f}(\boldsymbol{x}_i) - y_i\right)$$

$$\frac{\partial J_{LS}}{\partial \beta_j^{(k)}} = 2\alpha_j \sum_{i=1}^{n} o_{i,j}\left(1 - o_{i,j}\right) x_i^{(k)} \left(\hat{f}(\boldsymbol{x}_i) - y_i\right)$$

$$\frac{\partial J_{LS}}{\partial h_j} = 2\alpha_j \sum_{i=1}^{n} o_{i,j}\left(1 - o_{i,j}\right)\left(\hat{f}(\boldsymbol{x}_i) - y_i\right)$$

$$o_{i,j} = \varphi(\boldsymbol{x}_i; \boldsymbol{\beta}_j, h_j)$$

Error Back-Propagation (cont.)

- When the output values of the network are calculated, the input points are propagated following the forward path.
- On the other hand, when the gradients are calculated, the output error $(\hat{f}(\boldsymbol{x}_i) - y_i)$ is propagated backward.
- For this reason, this algorithm is called the error back-propagation.
- However, it is gradient descent so global convergence is not guaranteed.

Stochastic Gradient Descent

- In the usual gradient method, all training examples are used at the same time.
- In practice, the following stochastic method would be computationally advantageous.
  - Randomly choose one of the training examples (say, $(\boldsymbol{x}_i, y_i)$)
  - Update the parameter vector by

$$\widehat{\boldsymbol{w}}^{new} \longleftarrow \widehat{\boldsymbol{w}}^{old} - \varepsilon \nabla J_i(\widehat{\boldsymbol{w}}^{old})$$

$$J_i(\boldsymbol{w}) = (\hat{f}(\boldsymbol{x}_i) - y_i)^2$$

  - Repeat this procedure until convergence.

Avoiding Overfitting

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{b} \alpha_i \varphi(\boldsymbol{x}; \boldsymbol{\beta}_i)$$

$$\boldsymbol{w} = (\boldsymbol{\alpha}^\top, \boldsymbol{\beta}_1^\top, \boldsymbol{\beta}_2^\top, \ldots, \boldsymbol{\beta}_b^\top)^\top.$$

- LS overfits to noisy samples.
  - Regularization:

$$\min_{\boldsymbol{w}} [J_{LS}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|^2]$$

$$J_{LS}(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\hat{f}(\boldsymbol{x}_i) - y_i \right)^2 \qquad \lambda > 0$$

  - Early stopping: Stop gradient descent before it converges

Homework

1. Prove that the gradients for Sigmoidal basis functions are given as

$$\frac{\partial J_{LS}}{\partial \alpha_j} = 2\sum_{i=1}^{n} o_{i,j}\left(\hat{f}(\boldsymbol{x}_i) - y_i\right)$$

$$o_{i,j} = \varphi(\boldsymbol{x}_i; \boldsymbol{\beta}_j, h_j)$$

$$\frac{\partial J_{LS}}{\partial \beta_j^{(k)}} = 2\alpha_j \sum_{i=1}^{n} o_{i,j}\left(1 - o_{i,j}\right) x_i^{(k)} \left(\hat{f}(\boldsymbol{x}_i) - y_i\right)$$

$$\frac{\partial J_{LS}}{\partial h_j} = 2\alpha_j \sum_{i=1}^{n} o_{i,j}\left(1 - o_{i,j}\right)\left(\hat{f}(\boldsymbol{x}_i) - y_i\right)$$

Homework (cont.)

2. For your own toy 1-dimensional data, perform simulations using
    - 3-layer neural networks with sigmoid functions
    - Error back-propagation

   and analyze the results, e.g., by changing
    - Target functions
    - Number of samples
    - Noise level
    - Number of hidden neurons