Robust and Sparse

- Combining Huber's loss with ℓ_1 constraint.

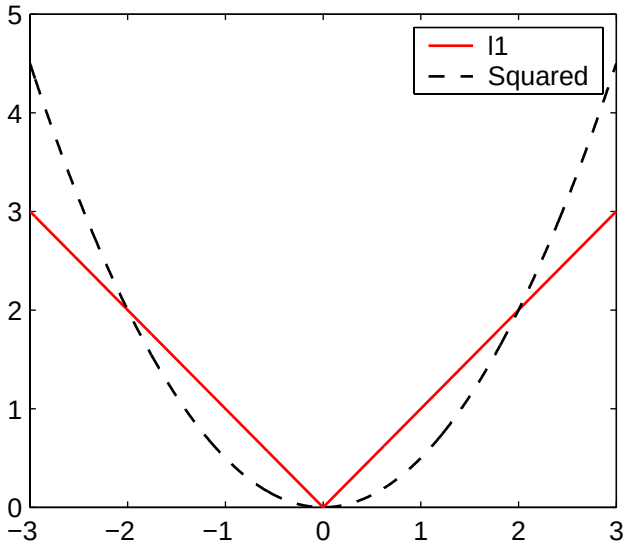
$$\hat{\boldsymbol{\alpha}}_{SparseHuber} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} J_{Huber}(\boldsymbol{\alpha}) \quad \text{subject to } \|\boldsymbol{\alpha}\|_1 \leq C$$

- Solving quadratic programming problem is computationally rather demanding.
- Is it possible to make it faster?

l1 Loss

- Quadratic term comes from Huber's loss.
- ℓ_1-loss is linear.

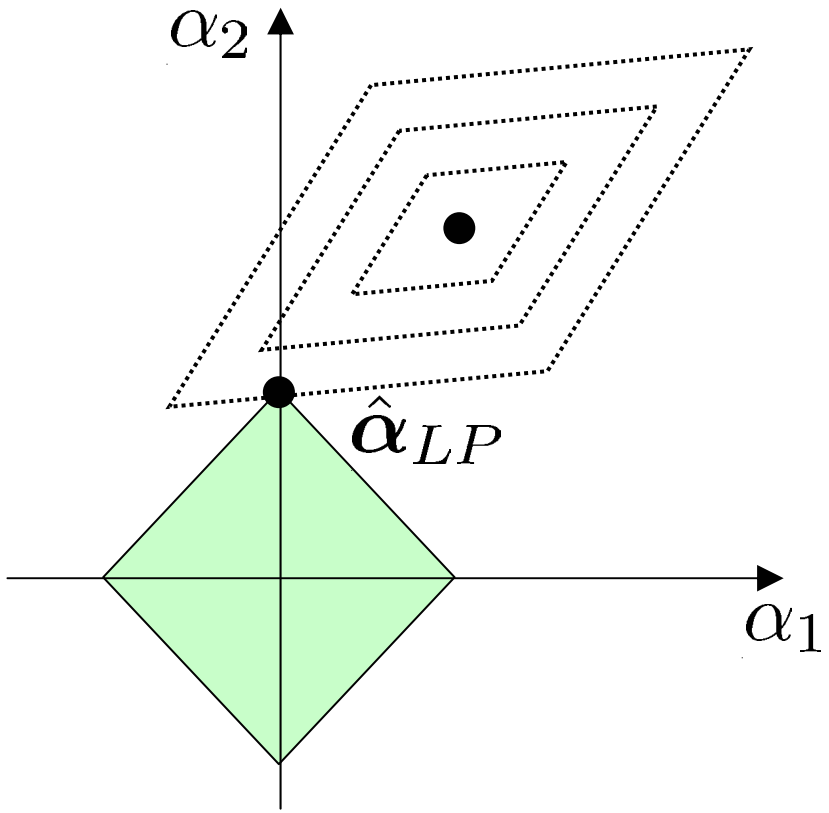
$$\sum_{i=1}^{n} \left| \hat{f}(\boldsymbol{x}_i) - y_i \right|$$



Linear Programming Learning

- Combine ℓ_1 loss and ℓ_1 regularizer.

$$J_{LP}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \left| \hat{f}(\boldsymbol{x}_i) - y_i \right| + \lambda \sum_{i=1}^{p} |\alpha_i|$$



How to Obtain Solutions

- Trick to avoid absolute value:

$$\|\boldsymbol{\alpha}\|_1 = \min_{\boldsymbol{u}\in\mathbb{R}^p} \sum_{i=1}^{p} u_i$$

$$\text{subject to } -\boldsymbol{u} \leq \boldsymbol{\alpha} \leq \boldsymbol{u},$$

- Solution is given by

$$\min_{\boldsymbol{\alpha},\boldsymbol{u}\in\mathbb{R}^p,\boldsymbol{h}\in\mathbb{R}^n} \left[\sum_{i=1}^{n} h_i + \lambda \sum_{i=1}^{p} u_i\right]$$

$$\text{subject to } -\boldsymbol{h} \leq \boldsymbol{X}\boldsymbol{\alpha} - \boldsymbol{y} \leq \boldsymbol{h}$$

$$-\boldsymbol{u} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}$$

Linearly Constrained Linear Programming Problem

- Standard optimization softwares can solve the following form of linearly constrained linear programming problems.

$$\min_{\boldsymbol{\beta}} \langle \boldsymbol{\beta}, \boldsymbol{q} \rangle \quad \text{subject to } \boldsymbol{V}\boldsymbol{\beta} \leq \boldsymbol{v}$$
$$\boldsymbol{G}\boldsymbol{\beta} = \boldsymbol{g}$$

Sparseness and Robustness

	Sparse-ness	Robust-ness	Optimizat ion
ℓ_1 constrained LS	Yes	No	Quadratic
Huber's method	No	Yes	Quadratic
ℓ_1 constrained Huber	Yes	Yes	Quadratic
Linear programming	Yes	Yes	Linear

Non-Linear Models

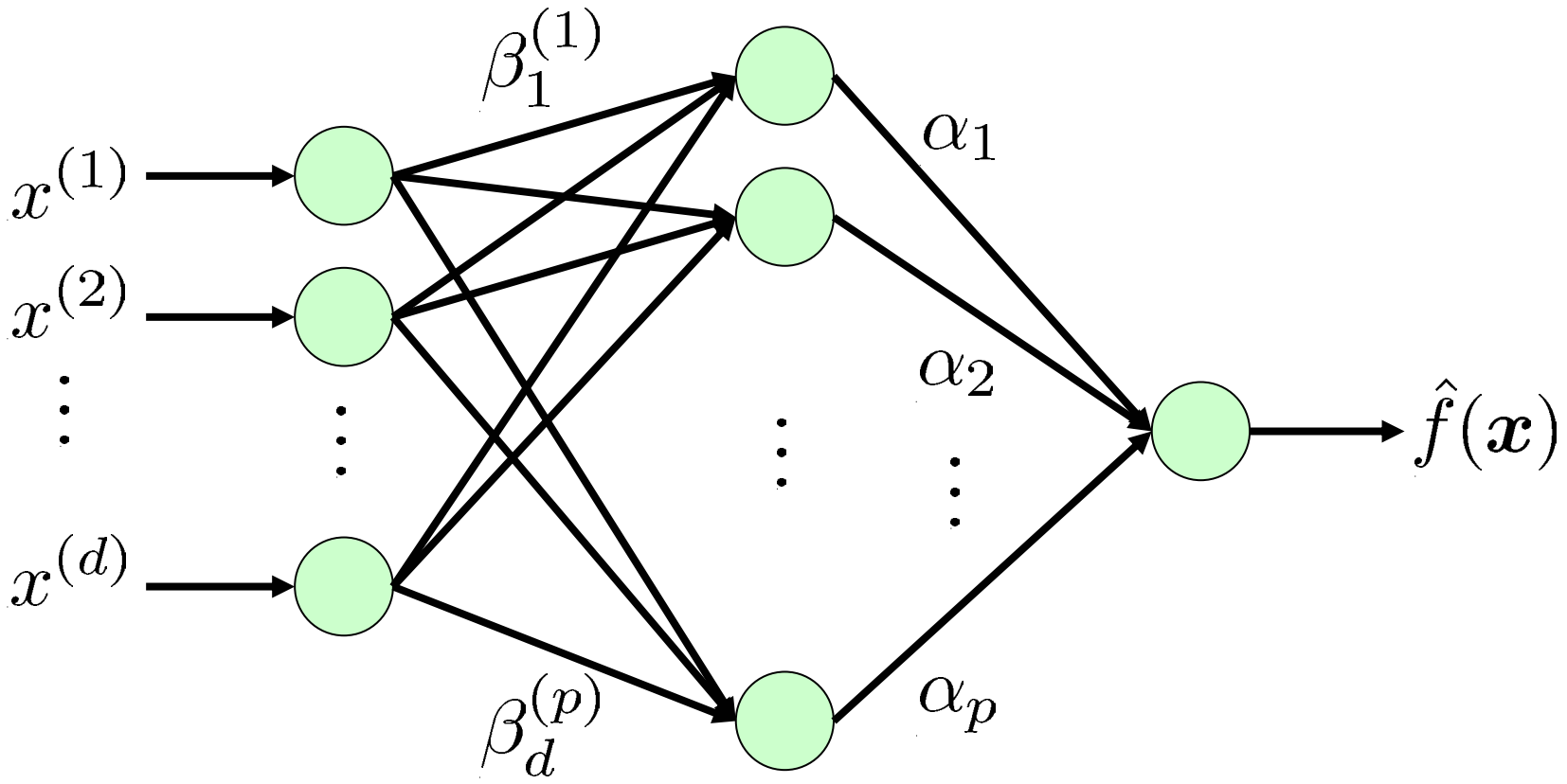
- A popular choice: Hierarchical models

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

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

Three-Layer Networks

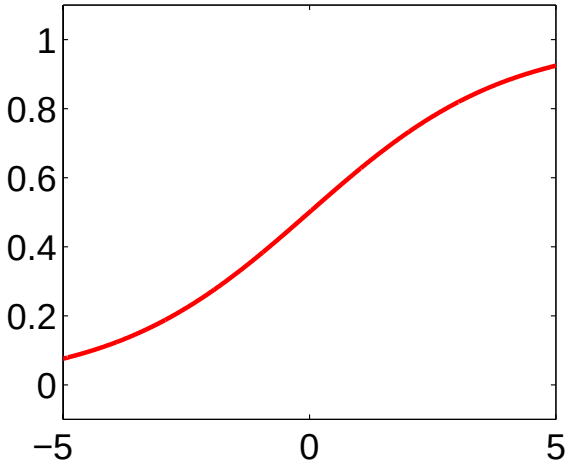
- Such a hierarchical model can be represented as a 3-layer network.



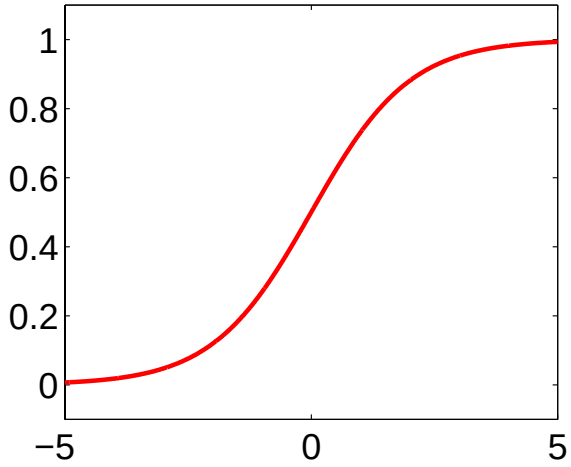
Sigmoidal Function

- A typical basis function: Sigmoidal function

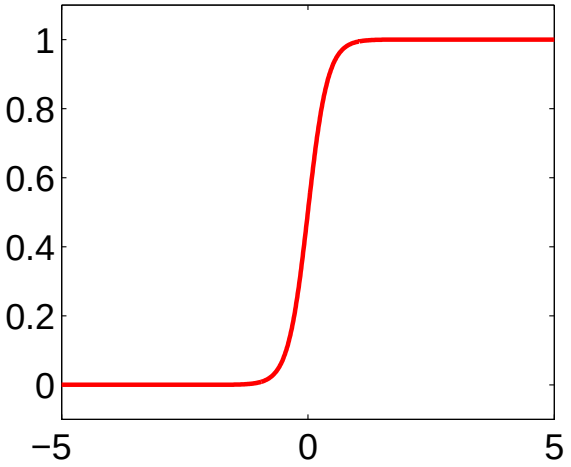
$$\varphi(\boldsymbol{x};\boldsymbol{\beta},b)=\frac{1}{1+\exp\left(-\langle\boldsymbol{x},\boldsymbol{\beta}\rangle-b\right)}$$



$\boldsymbol{\beta}$:Small



$\boldsymbol{\beta}$:Medium



$\boldsymbol{\beta}$:Large

Perceptrons

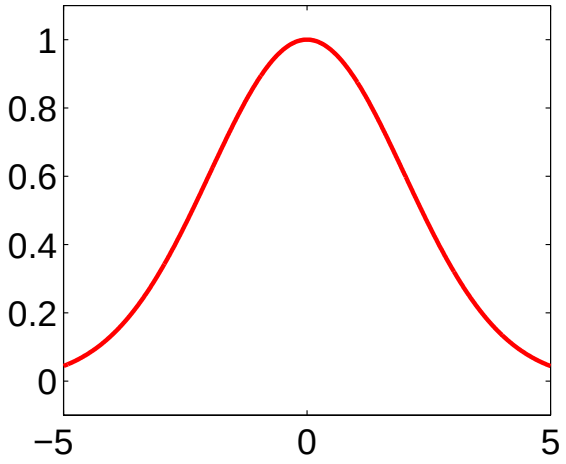
- The behavior of the sigmoidal functions is similar to the neurons in the brain.
- For this reason, hierarchical models with sigmoidal functions are called artificial neural networks or perceptrons.
- Mathematically, 3-layer neural networks can approximate any continuous functions with arbitrary small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

Gaussian Radial Basis Function

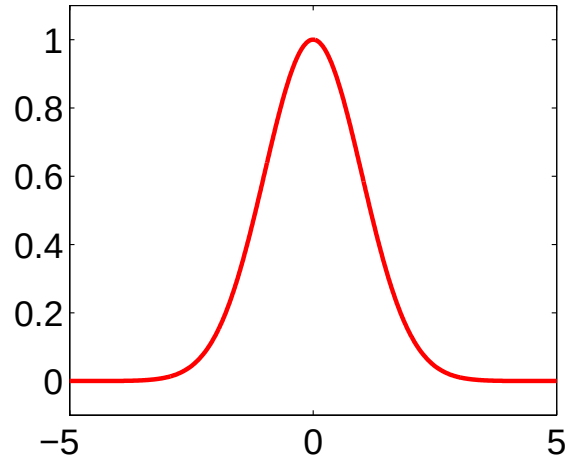
- Another popular basis function:

  Gaussian radial basis function

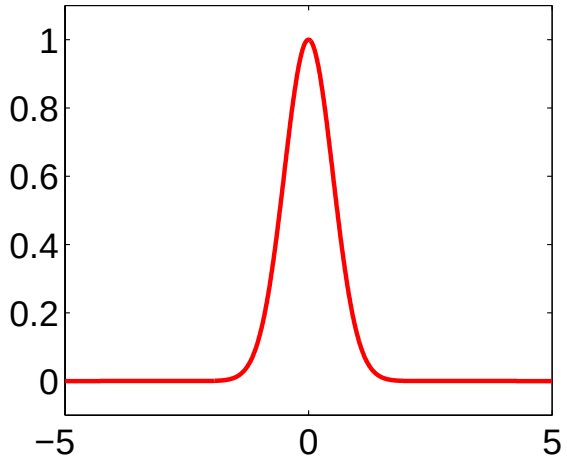
$$\varphi(\boldsymbol{x};\boldsymbol{\beta},c) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{\beta}\|^2}{2c^2}\right)$$



c :Small



c :Medium



c :Large

Non-Linear Learning for Non-Linear Models

- Sigmoidal neural networks
- Error back-propagation algorithm

Least-squares Learning

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

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

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

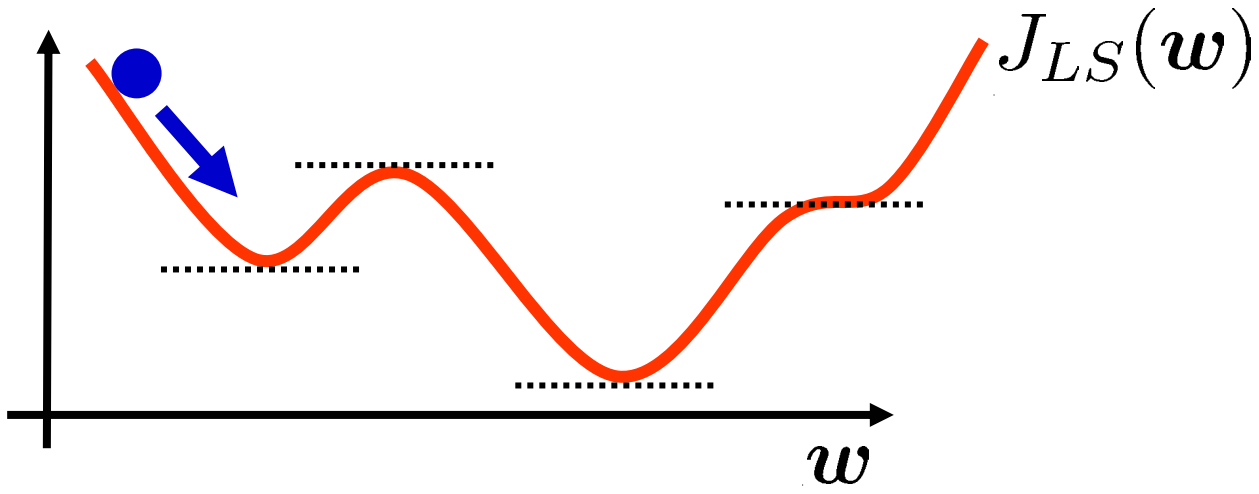
$$\min_{\boldsymbol{w}} J_{LS}(\boldsymbol{w})$$

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

How to Obtain Solutions

- No analytic solution is known.
- We usually use simple gradient search.

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



- It converges to one of the local minima.

Error Back-Propagation

- Efficient Calculation of Gradient for Sigmoidal Basis Functions

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

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

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

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

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 search 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.