Lecture 2: Non-linear methods for statistical learning

Minwoo Chae

Department of Industrial and Management Engineering
Pohang University of Science and Technology

KMS-NIMS Summer School on AI, 2025

Introduction

- In both regression and classification, we have modeled the unknown function using linear functions.
- In many applications, however, linear models are not sufficient.
- From a decision-theoretic perspective, linear models can incur large bias (approximation error).
- Nonlinear models often result in higher variance (estimation error), but can substantially reduce bias.
- In this lecture, we study several nonlinear methods based on empirical risk minimization (ERM).

Outline

Basis expansions

- Let $\phi_j : \mathbb{R}^p \to \mathbb{R}$, for $j = 1, \ldots, M$, be the jth transformation (or basis function) of $\mathbf{X}$.
- We then model the function as

$$f(\mathbf{X}) = \sum_{j=1}^{M} \beta_j \phi_j(\mathbf{X}),$$

  which is a linear expansion in the transformed features.
- The model is linear in the parameters, but nonlinear in the original input $\mathbf{X}$.

Examples of basis functions

- $\phi_j(\mathbf{X}) = X_j$ for $j = 1, \ldots, p$
- $\phi_j(\mathbf{X}) = X_k^2$ or $h_j(\mathbf{X}) = X_k X_l$
- $\phi_j(\mathbf{X}) = \log(X_k), \sqrt{X_k}, \ldots$
- $\phi_j(\mathbf{X}) = I(L \leq X_k < U)$
- General polynomials

Choice of basis functions

- Compared to linear models, the main purpose of using nonlinear models is to reduce approximation error.
- However, using too many basis functions can lead to large estimation error.
- Therefore, it is important to choose basis functions that balance approximation and estimation errors.
- This is a difficult task because the target function is unknown.
- There are many types of basis functions that can approximate general functions.

Polynomial regression

- For p-dimensional $\mathbf{X}$, an mth-order polynomial model is given by

$$f(\mathbf{X}) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j + \sum_{j,k} \beta_{jk} X_j X_k + \cdots + \sum_{j_1,\ldots,j_m} \beta_{j_1,\ldots,j_m} X_{j_1} \cdots X_{j_m}.$$

- Every smooth function can be approximated by a polynomial. (Weierstrass approximation theorem)
- As p increases, the number of parameters grows exponentially.

Some popular basis functions

- Commonly used basis functions include:
  - Splines
  - Fourier bases
  - Wavelet bases
- For all types of basis functions, it is important to choose an appropriate number of basis terms.

Example: Smooth function estimation

- To gain intuition, consider a series expansion of f_0:

$$f_0(\mathbf{x}) = \sum_{j=1}^{\infty} c_j \phi_j(\mathbf{x}) = \underbrace{\sum_{j=1}^{J} c_j \phi_j(\mathbf{x})}_{\text{low frequency}} + \underbrace{\sum_{j>J} c_j \phi_j(\mathbf{x})}_{\text{high frequency}},$$

  where $(\phi_j)_{j\geq 1}$ is a suitable basis.
  - *e.g.* Fourier basis, wavelets
  - Such expansions are valid for sufficiently smooth functions.
- If f_0 is sufficiently regular, it can be well-approximated using only the low-frequency components, i.e., the coefficients c_j decay as j increases.

Example: Smooth function estimation (cont.)

- Consider a model using the first J basis functions:

$$\mathcal{F} = \left\{ f : f(\cdot) = \sum_{j=1}^{J} \beta_j \phi_j(\cdot),\ \beta_j \in \mathbb{R} \right\}.$$

- The approximation error for estimating f_0 using this model is

$$\left\| f_0(\cdot) - \sum_{j=1}^{J} c_j \phi_j(\cdot) \right\| = \left\| \sum_{j>J} c_j \phi_j(\cdot) \right\|.$$

- The complexity of the model is primarily determined by J.
  - In some cases, additional constraints may be imposed on the coefficients β_j.

Remarks

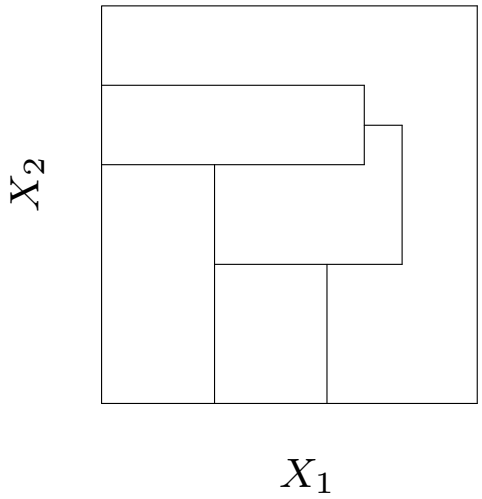
- The choice of basis functions and the number of basis terms are key components in basis expansion approaches.
- Various strategies exist, but there are no universally optimal or automatic rules.
- In practice, cross-validation is a simple and effective method for model selection.

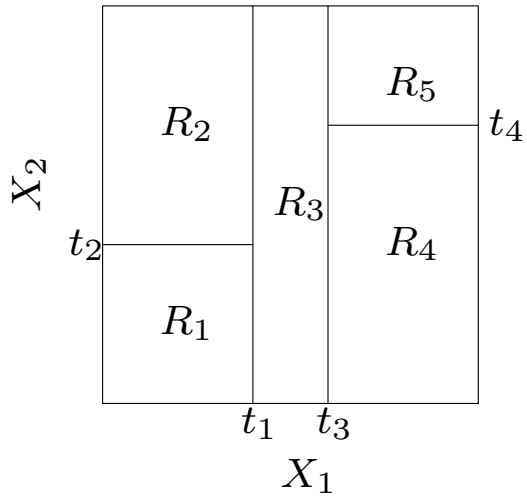
Outline

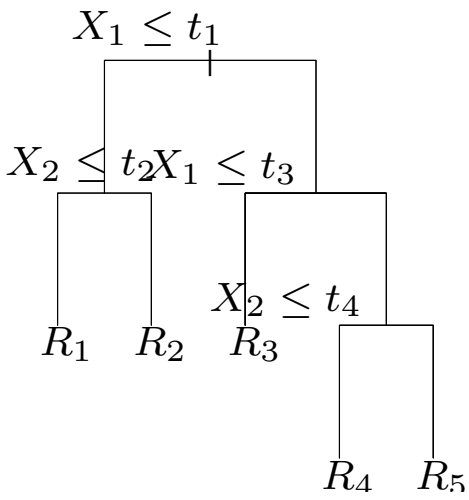
Introduction

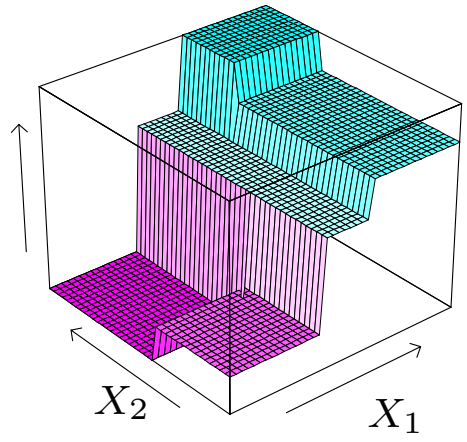
- Tree ensemble methods, such as boosting and random forests, are among the most successful learning algorithms.
- Tree ensemble methods combine many trees in an additive manner.
- Despite the rise of deep learning, they remain some of the most popular methods for analyzing tabular data.
- We begin with trees.

Tree


X_2
X_1


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_1
X_2


$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5


X_2
X_1

tree (cont.)

- The terminal nodes (or leaves) of the tree correspond to regions $\mathcal{R}_1, \ldots, \mathcal{R}_M$.
- Given these regions, we can estimate f by

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \hat{c}_m I(\mathbf{x} \in \mathcal{R}_m),$$

where

$$\hat{c}_m = \underset{c \in \widetilde{\mathcal{Y}}}{\operatorname{argmin}} \sum_{i:\mathbf{X}_i \in \mathcal{R}_m} L(Y_i, c)$$

is the empirical risk minimizer within region $\mathcal{R}_m$.

tree (cont.)

- When the loss function is the squared error loss, empirical risk minimization at each region reduces to computing the mean.
- For other loss functions, a closed-form solution is rarely available.
- In such cases, one can apply the Newton–Raphson method for optimization.

tree (cont.)

- The key component in constructing a decision tree is recursively splitting the input space using input variables.
- Given the data and the number of terminal nodes, constructing the optimal regions (e.g., minimizing training error) is computationally infeasible.
- Hence, tree construction is usually based on stepwise methods, including growing and pruning.
- We focus on the growing step.

Splitting rule

- At each step, we determine a splitting variable X_j and a corresponding splitting criterion.
  - For a continuous variable X_j, the splitting criterion is a threshold s, that is, $X_j \leq s$ (left node) and $X_j > s$ (right node).
  - For a categorical variable X_j with values in $\{1, \ldots, K\}$, the splitting criterion is any binary partition of $\{1, \ldots, K\}$.
- The splitting variable X_j and the criterion are chosen to minimize the training error.

Splitting rule (cont.)

- Assume all input variables are continuous, and define

$$\mathcal{R}_1(j,s) = \{x : X_j \leq s\} \quad \text{and} \quad \mathcal{R}_2(j,s) = \{x : X_j > s\}.$$

- Then we seek the splitting variable j and split point s that solve

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in \mathcal{R}_1(j,s)} L(Y_i, c_1) + \min_{c_2} \sum_{x_i \in \mathcal{R}_2(j,s)} L(Y_i, c_2) \right].$$

Stopping rule

- A stopping rule determines when to terminate further splitting.
- Common criteria include:
  - All observations in a node belong to the same class.
  - The number of observations in a node is too small.
  - The decrease in impurity is negligible.
  - The depth of the node exceeds a predefined threshold.
- Many tree ensemble methods simply limit the maximum depth.

Remarks

- A single decision tree often performs poorly in terms of predictive accuracy.
- Tree ensemble methods can substantially improve the predictive performance over a single tree.
- We focus on gradient boosting.

History of boosting

- An off-the-shelf procedure for data analysis.
- Originally developed by Schapire (1990) and Freund (1995).
- AdaBoost was introduced by Freund and Schapire (1996).
- Friedman et al. (2000) provided a statistical perspective.
- Friedman (2001) proposed a learning algorithm called gradient boosting.
- Scalable implementations:
  - XGBoost
  - LightGBM

Schapire, R. E. "The strength of weak learnability". *Mach. Learn.* 1990

Freund, Y. "Boosting a weak learning algorithm by majority". *Inform. and Comput.* 1995

Freund, Y. & Schapire, R. E. "Experiments with a new boosting algorithm". *Proc. ICML*. 1996

Friedman, J., Hastie, T. & Tibshirani, R. "Additive logistic regression: A statistical view of boosting (with discussion)". *Ann. Statist.* 2000

Friedman, J. H. "Greedy function approximation: A gradient boosting machine". *Ann. Statist.* 2001

Hastie, T., Tibshirani, R. & Friedman, J. H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* (Springer, New York, 2009)

Boosting as an additive expansion

- Boosting fits an additive expansion of trees:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m h(\mathbf{x}; \theta_m)$$

- The model can be fit via empirical risk minimization (ERM):

$$\underset{\{\beta_m, \theta_m\}_{m=1}^{M}}{\text{minimize}} \sum_{i=1}^{n} L\left(Y_i, \sum_{m=1}^{M} \beta_m h(\mathbf{X}_i; \theta_m)\right)$$

- This joint optimization is typically computationally infeasible.
- A simple alternative:
  - Update the expansion sequentially.
  - Consider only one basis function at each step.
  - This approach is generally referred to as forward stagewise updates.

Forward stagewise updates

- Suppose the current model is

$$f_{m-1}(\mathbf{x}) = \sum_{j=1}^{m-1} \beta_j h(\mathbf{x}; \theta_j)$$

- The next update is obtained by solving

$$(\beta_m, \theta_m) = \underset{(\beta,\theta)}{\operatorname{argmin}} \sum_{i=1}^{n} L\Big(Y_i, f_{m-1}(\mathbf{X}_i) + \beta h(\mathbf{X}_i; \theta)\Big)$$

- That is, forward stagewise updates add a new basis function to the expansion without modifying the existing components.

Example: Forward stagewise regression

- Consider the squared error loss:

$$L(Y, f(\mathbf{X})) = \big(Y - f(\mathbf{X})\big)^2$$

- Let $R_{im} = Y_i - f_{m-1}(\mathbf{X}_i)$ be the residual at step m. Then,

$$L\big(Y_i, f_{m-1}(\mathbf{X}_i) + \beta h(\mathbf{X}_i; \theta)\big) = \big(R_{im} - \beta h(\mathbf{X}_i; \theta)\big)^2$$

- At each step, the term $\beta_m h(\cdot; \theta_m)$ that best fits the current residuals is added to the model.

Remarks

- In summary, boosting performs forward stagewise updates of additive function expansions:
  1. Set an initial estimator f_0.
  2. For $m = 1, 2, \ldots$

$$(\beta_m, \theta_m) = \underset{(\beta,\theta)}{\operatorname{argmin}} \sum_{i=1}^{n} L\big(y_i, f_{m-1}(\mathbf{X}_i) + \beta h(\mathbf{X}_i; \theta)\big)$$

  and set $f_m(\cdot) = f_{m-1}(\cdot) + \beta_m h(\cdot; \theta_m)$.

- The method can be applied to both regression and classification.
- Computation under the squared loss is much simpler than with other loss functions.
- For other loss functions, gradient boosting—which mimics least squares boosting—is preferred.

Gradient boosting

- Recall the empirical risk minimization (ERM) principle:

$$\text{minimize } L(\mathbf{f}) = \sum_{i=1}^{n} L\big(Y_i, f(\mathbf{X}_i)\big),$$

  where $\mathbf{f} = (f(\mathbf{X}_1), \ldots, f(\mathbf{X}_n))^T$.
- Note that f is constrained to be a sum of basis functions (e.g., trees).
- Ignoring this constraint, consider the unconstrained optimization problem:

$$\hat{\mathbf{f}} = \underset{\mathbf{f} \in \mathbb{R}^n}{\operatorname{argmin}}\, L(\mathbf{f})$$

Gradient boosting (cont.)

- Given the current solution $\mathbf{f}_{m-1}$, we can update it via gradient descent:

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m,$$

where

$$\mathbf{g}_m = \left.\frac{\partial L}{\partial \mathbf{f}}\right|_{\mathbf{f}=\mathbf{f}_{m-1}}$$

and ρ_m is the learning rate.

Gradient boosting (cont.)

- In the m-th update step of boosting, we perform

$$f_m(\cdot) = f_{m-1}(\cdot) + h(\cdot; \theta_m),$$

  where we aim to find θ_m such that the vector $(h(\mathbf{X}_1; \theta_m), \ldots, h(\mathbf{X}_n; \theta_m))^T$ is as close as possible to $-\rho_m \mathbf{g}_m$.
- This can be achieved by fitting a regression tree to the pseudo-response vector $-\rho_m \mathbf{g}_m$, that is, by solving

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{n} \big[-\rho_m g_{mi} - h(\mathbf{X}_i; \theta) \big]^2.$$

Gradient boosting (cont.)

- The resulting update has the form

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{mj} I(\mathbf{x} \in \mathcal{R}_{mj}),$$

  where $\{\mathcal{R}_{mj}\}_{j=1}^{J}$ are the terminal regions (leaves) of the regression tree.

- The fit can be improved by replacing each γ_{mj} with

$$\widetilde{\gamma}_{mj} = \underset{\gamma}{\operatorname{argmin}} \sum_{\mathbf{x}_i \in \mathcal{R}_{mj}} L\big(Y_i, f_{m-1}(\mathbf{X}_i) + \gamma\big),$$

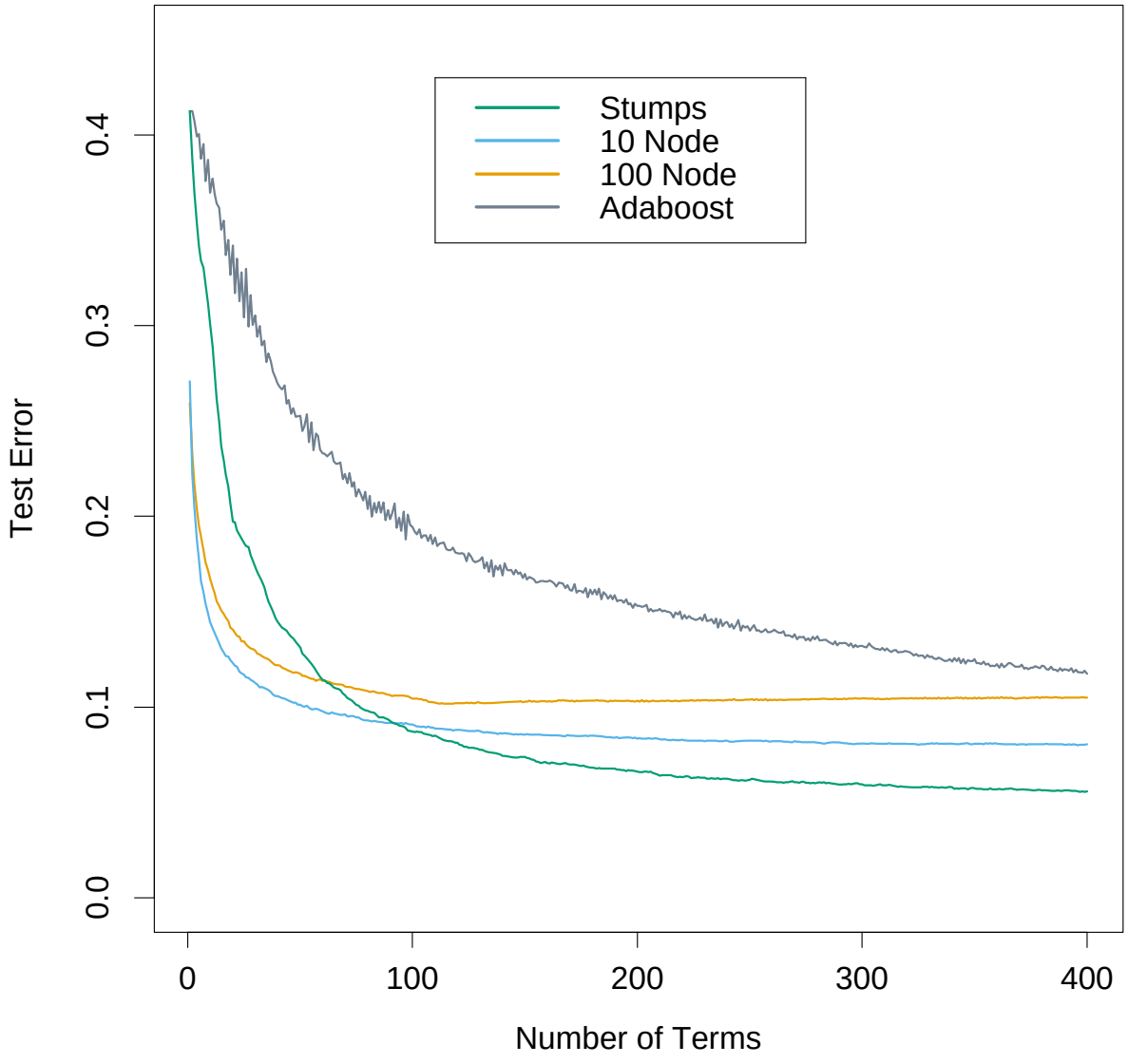
  which minimizes the loss over each leaf region.

- With this adjustment step, it is no longer necessary to consider ρ_m explicitly.

Gradient tree boosting algorithm

1. Initialize
$$f_0(\mathbf{x}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(Y_i, \gamma).$$
2. For $m = 1, 2, \ldots$
   1. Let $\mathbf{f}_{m-1} = (f_{m-1}(\mathbf{X}_1), \ldots, f_{m-1}(\mathbf{X}_n))^T$, and compute the negative gradient:
$$\mathbf{g}_m = \left.\frac{\partial L}{\partial \mathbf{f}}\right|_{\mathbf{f}=\mathbf{f}_{m-1}}.$$
   2. Fit a regression tree to the pseudo-response vector $-\mathbf{g}_m$, yielding terminal regions $\mathcal{R}_{mj}$, for $j = 1, \ldots, J$.
   3. For each region $\mathcal{R}_{mj}$, compute the optimal leaf value:
$$\gamma_{mj} = \underset{\gamma}{\operatorname{argmin}} \sum_{\mathbf{X}_i \in \mathcal{R}_{mj}} L\big(Y_i, f_{m-1}(\mathbf{X}_i) + \gamma\big).$$
   4. Update the model:
$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{mj} I(\mathbf{x} \in \mathcal{R}_{mj}).$$

Example: Synthetic data



Remarks

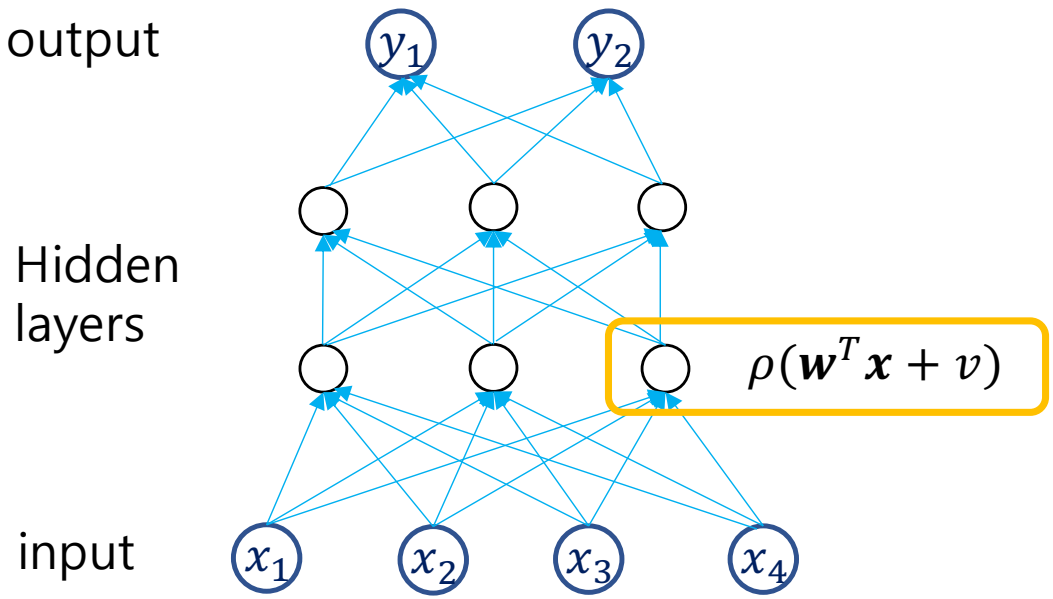
- Although boosting may overfit the data when the number of iterations M is very large, the overfitting tends to occur slowly.
- In practical implementations, various heuristic regularization techniques and scalable approaches are employed.

Outline

Introduction

- Deep learning is a broad term for methods that model unknown functions using deep neural networks (DNNs).
- A vanilla feedforward DNN has the following architecture:

$$\mathbf{f}(\mathbf{x}) = (W_L \circ \rho_{\mathbf{v}_L} \circ W_{L-1} \circ \rho_{\mathbf{v}_{L-1}} \circ \cdots \circ W_1 \circ \rho_{\mathbf{v}_1} \circ W_0)(\mathbf{x})$$



Activation function

- In neural networks, the function $\rho : \mathbb{R} \to \mathbb{R}$ is called the activation function.
- Common choices include:
  - Sigmoid (logistic): $\rho(x) = \frac{1}{1+e^{-x}}$
  - ReLU (Rectified Linear Unit): $\rho(x) = \max\{x, 0\}$
  - RePU (Rectified Power Unit): $\rho(x) = \max\{x^p, 0\}$
  - Leaky ReLU: $\rho(x) = \max\{x, ax\}$, where $a \in (0, 1)$
  - Hyperbolic tangent (tanh): $\rho(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
  - SoftPlus: $\rho(x) = \log(1 + e^x)$
  - ...

Shallow networks

- A shallow network is a neural network with a single hidden layer.
- When the output dimension is 1, a shallow network can be represented as

$$f(\mathbf{x}) = \sum_{j=1}^{m} c_j \rho(\mathbf{w}_j^T \mathbf{x} + v_j), \quad \mathbf{w}_j \in \mathbb{R}^d, \; v_j, c_j \in \mathbb{R}.$$

- Typically, the activation function ρ and the number of hidden units m are determined as part of the model selection process.

Parameters in a shallow network

$$f(\mathbf{x}) = \sum_{j=1}^{m} c_j \rho(\mathbf{w}_j^T \mathbf{x} + v_j), \quad \mathbf{w}_j \in \mathbb{R}^d, \; v_j, c_j \in \mathbb{R}.$$

- Parameters:
  - $\mathbf{w}_j$: weight vectors
  - v_j: bias terms
  - c_j: output layer coefficients

Deep neural networks

- For $\mathbf{v} = (v_1, \ldots, v_r)^T$ and $\mathbf{y} = (y_1, \ldots, y_r)^T$, define the activation operator $\rho_{\mathbf{v}} : \mathbb{R}^r \to \mathbb{R}^r$ by

$$\rho_{\mathbf{v}}(\mathbf{y}) = \big(\rho(y_1 - v_1), \ldots, \rho(y_r - v_r)\big)^T.$$

- Network architecture $(L, \mathbf{p})$:
  - L: number of hidden layers
  - $\mathbf{p} = (p_0, \ldots, p_{L+1})$: width vector, where p_ℓ denotes the number of units in layer ℓ

Deep neural networks (cont.)

- A DNN with architecture $(L, \mathbf{p})$ is defined by

$$\mathbf{f}(\mathbf{x}) = \big(\rho_{\text{out}} \circ W_L \circ \rho_{\mathbf{v}_L} \circ W_{L-1} \circ \rho_{\mathbf{v}_{L-1}} \circ \cdots \circ W_1 \circ \rho_{\mathbf{v}_1} \circ W_0\big)(\mathbf{x}),$$

  where ρ_{out} is the output activation function.
- Parameters:
  - $W_i \in \mathbb{R}^{p_i \times p_{i+1}}$: weight matrix for layer i
  - $\mathbf{v}_i \in \mathbb{R}^{p_i}$: bias vector for layer i
- As before, the choice of network architecture and activation function is considered part of the model selection process.

Output activation function

- The choice of output activation function depends on the task at hand.
- For regression, ρ_{out} is typically the identity function.
- For classification, the softmax function is a standard choice. It is defined as

$$\rho_{\text{out}}(\mathbf{z}) = \left(\frac{e^{z_1}}{\sum_{j=1}^{r} e^{z_j}}, \ldots, \frac{e^{z_r}}{\sum_{j=1}^{r} e^{z_j}} \right)^T,$$

  which maps a vector $\mathbf{z} \in \mathbb{R}^r$ to a probability vector in $[0, 1]^r$.

Structured networks

- State-of-the-art neural networks often employ specific architectural structures.
- Examples of important structured networks:
  - Sparse networks
  - Residual networks
  - Convolutional neural networks
  - Recurrent neural networks
  - Transformers
  - ...
- These structured architectures are designed for specific purposes.
- In this lecture, we focus on vanilla feedforward DNNs with sparsity.

Estimation of parameters in supervised learning

- Let $(\mathbf{X}_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \ldots, n$, be given data.
- For a given network architecture $(L, \mathbf{p})$, let f_θ be an unknown function of interest parametrized by

$$\theta = (W_0, \ldots, W_L, \mathbf{v}_1, \ldots, \mathbf{v}_L).$$

- Let $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty]$ be a loss function.
- Then, one can estimate θ by minimizing

$$R_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\theta(\mathbf{X}_i)).$$

- This is known as the empirical risk minimization.

Loss functions

- Squared error loss for regression:

$$R_n(\theta) = R_n(f_\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - f_\theta(\mathbf{X}_i)\right)^2$$

- Cross-entropy for K-class classification:

$$R_n(\theta) = R_n(\mathbf{f}_\theta) = -\frac{1}{n}\sum_{i=1}^{n}\mathbf{Y}_i^T \log\left(\mathbf{f}_\theta(\mathbf{X}_i)\right)$$

  - $\mathbf{Y}_i \in \{0,1\}^K$: One-hot encoded response
  - $\mathbf{f}_\theta(\mathbf{X}_i)$: A probability vector

Optimization

- The empirical risk $R_n(\theta)$ can be minimized via gradient descent:

$$\theta_{t+1} = \theta_t - \eta_t \nabla R_n(\theta_t)$$

  - $\eta_t \in (0, 1]$ is the learning rate.
  - In practice, stochastic gradient methods are typically used.
- Since the objective is non-convex and high-dimensional, many practical challenges arise—an area with extensive literature.
  - These issues are beyond the scope of this lecture.

Empirical risk minimizer

- We study the behavior of the minimizer

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} R_n(\theta),$$

  where Θ is the parameter space.
- Equivalently, we may consider

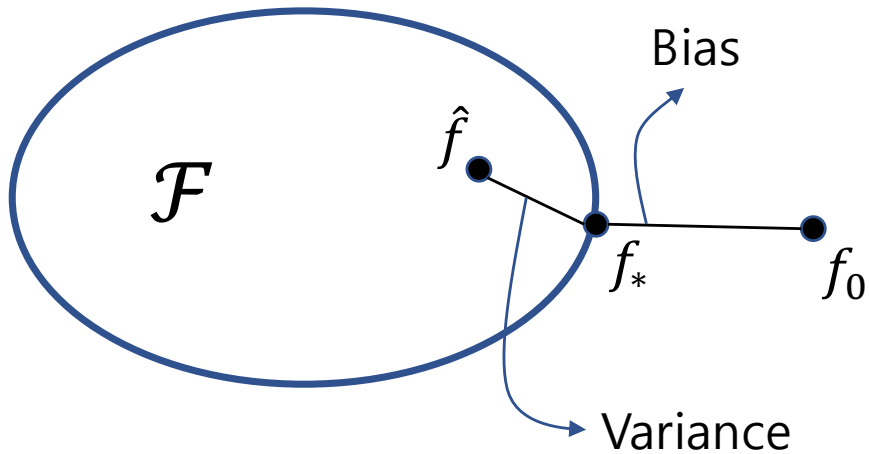
$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} R_n(f),$$

  where $\mathcal{F}$ is a function class parametrized by θ.
- Our goal is for the risk $R(\hat{f})$ to be sufficiently small.

Bias-variance tradeoff

- Recall the bias–variance tradeoff:

$$\|\hat{f} - f_0\| \leq \underbrace{\|f_* - f_0\|}_{\substack{\text{Bias} \\ \text{(approximation error)}}} + \underbrace{\|\hat{f} - f_*\|}_{\substack{\text{Variance} \\ \text{(estimation error)}}} .$$



- In the next few slides, we study the approximation properties of DNNs.

Universal approximation theorem

THEOREM (Universal approximation theorem) A shallow neural network can approximate any continuous function f arbitrarily well, provided it has sufficiently many hidden units.

- This result was established by several researchers in the 1980s; see Section 6 of Schmidt-Hieber (2020) for detailed references.

Schmidt-Hieber, J. "Nonparametric regression using deep neural networks with ReLU activation function". *Ann. Statist.* 2020

Universal approximation theorem (cont.)

- However, the universal approximation theorem does not guarantee efficient estimation using shallow networks.
- Achieving a sufficiently small approximation error may require an excessively large model, which can result in high estimation error.
- For approximating d-dimensional β-smooth functions, Mhaskar showed that the approximation error of shallow networks with $O(s)$ units is bounded by $O(s^{-\beta/d})$.

Mhaskar, H. N. "Neural networks for optimal approximation of smooth and analytic functions". *Neural Comput.* 1996

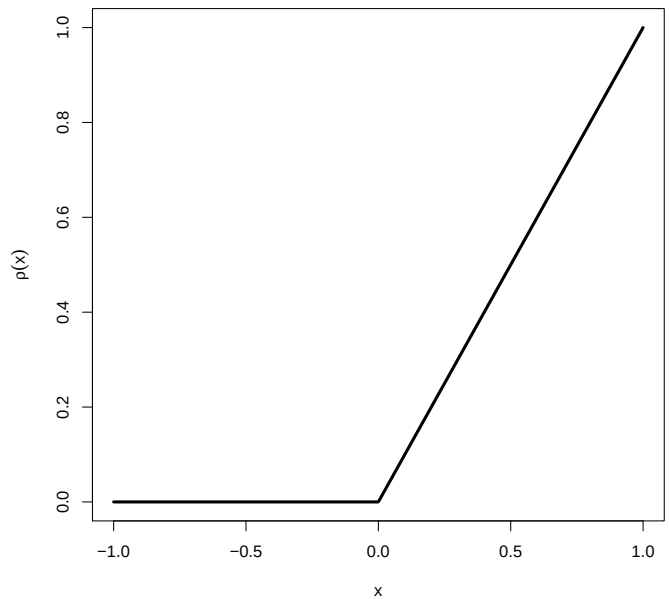
DNN with ReLU activation function

- Many structured functions can be approximated much more efficiently by DNNs than by shallow networks.
- Since ReLU is one of the most widely used activation functions, this lecture focuses on deep ReLU networks.
- Approximation properties of DNNs with general activation functions have also been studied in the literature.
  (*e.g.* Ohn and Kim, 2019)

Ohn, I. & Kim, Y. “Smooth function approximation by deep neural networks with general activation functions”. *Entropy*. 2019

DNN with ReLU activation function (cont.)

- Recall the ReLU activation function: $\rho(x) = \max\{0, x\}$.
- Although ReLU networks are piecewise linear, they can efficiently approximate arbitrarily smooth functions.



Basic properties of ReLU networks

$$\rho(x) = \max\{0, x\}$$

- Projection property:

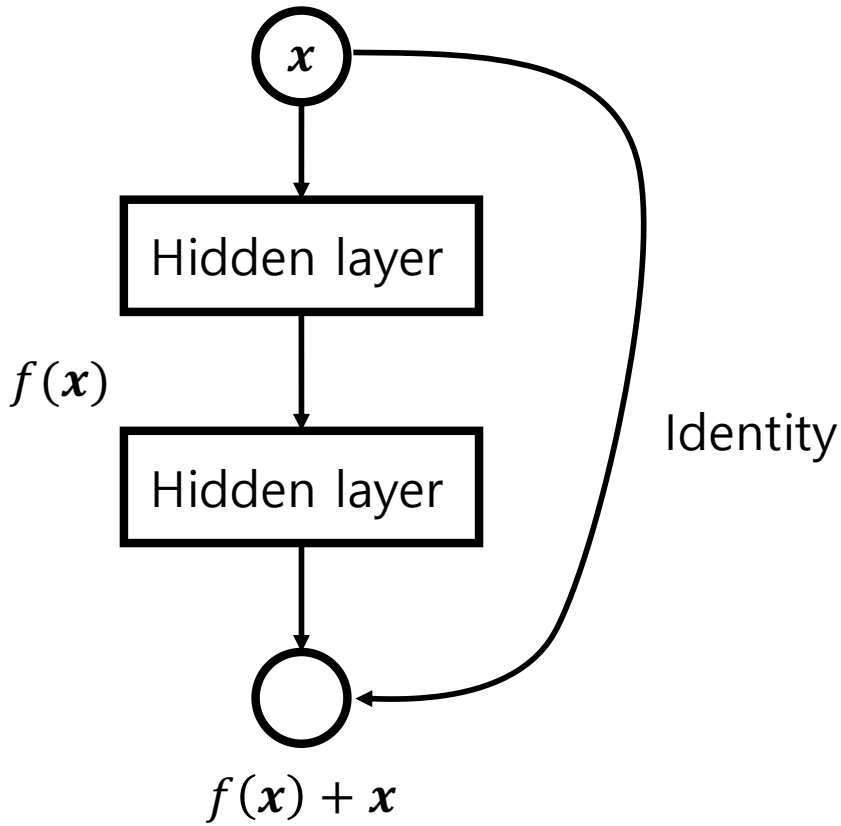
$$\rho \circ \rho = \rho$$

- Efficient learning of identity function:

$$x = \rho(x) - \rho(-x)$$

  - Skip connections can be learned efficiently.

Skip connections



Approximation of square function

- Function approximation with ReLU networks is often based on Taylor expansion.
- Approximating arithmetic operations using ReLU networks is essential.
- The addition operation $(x, y) \mapsto x + y$ can be easily implemented using ReLU networks.
- A key challenge is to approximate the multiplication operation $(x, y) \mapsto xy$.
- This can be reduced to the approximation of the square function $x \mapsto x^2$, since

$$xy = \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2.$$

Approximation of square function (cont.)

- Deep ReLU networks can approximate the square function very efficiently.
- Define $T_k : [0, 2^{-2(k-1)}] \to [0, 2^{-2k}]$ as

$$T_k(x) = \rho(x/2) - \rho(x - 2^{1-2k}).$$

  - This is a ReLU network with no hidden layers and coefficients bounded by 1.
- Define $R_k : [0, 1] \to [0, 2^{-2k}]$ as

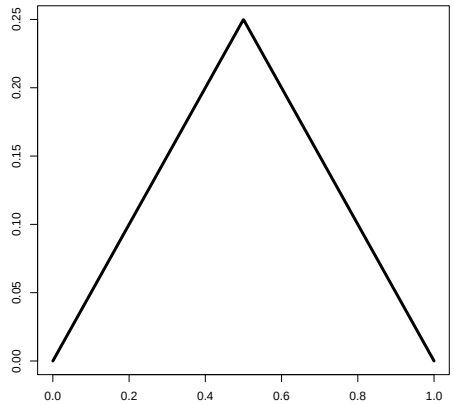
$$R_k = T_k \circ T_{k-1} \circ \cdots \circ T_1.$$

Yarotsky, D. "Error bounds for approximations with deep ReLU networks". *Neural Networks*. 2017

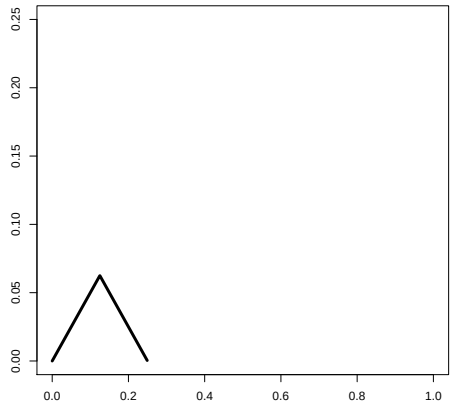
Telgarsky, M. "Benefits of depth in neural networks". *Proc. COLT*. 2016

Schmidt-Hieber, J. "Nonparametric regression using deep neural networks with ReLU activation function". *Ann. Statist.* 2020

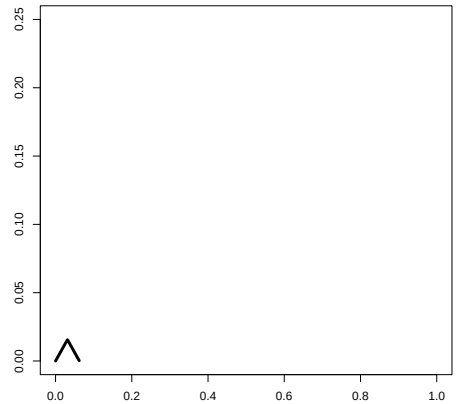
Approximation of square function (cont.)



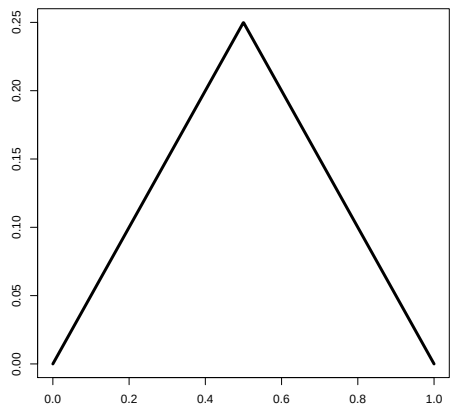
(a) T_1



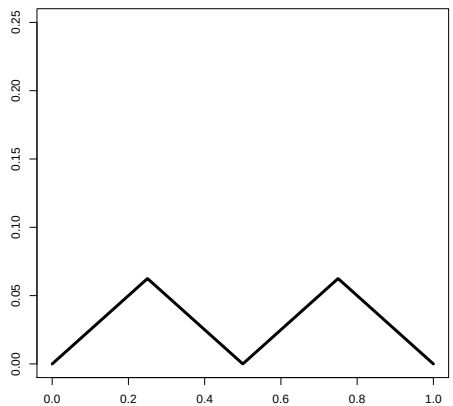
(b) T_2



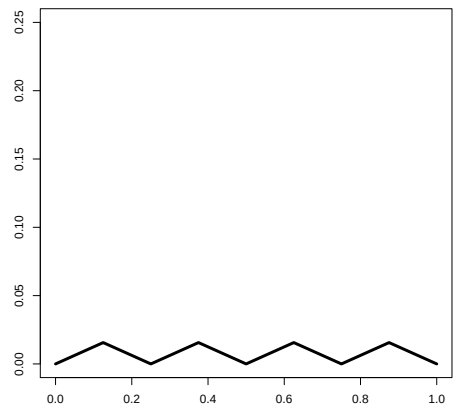
(c) T_3



(d) R_1



(e) R_2



(f) R_3

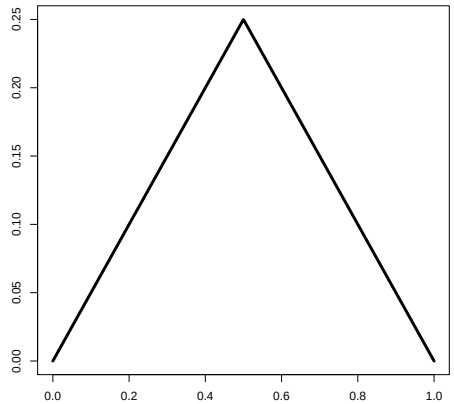
Approximation of square function (cont.)

- Let $F_k(x) = R_1(x) + \cdots + R_k(x)$.
- Then, it can be shown that

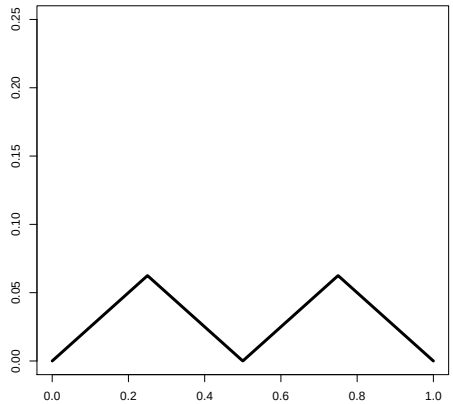
$$|x(1-x) - F_k(x)| \leq 2^{-k}.$$

- Only $O(k)$ network parameters are needed to achieve approximation error 2^{-k}.
  - In contrast, a shallow ReLU network requires at least $O(2^{k/2})$ parameters.
- Hence, deep ReLU networks approximate the square function efficiently.

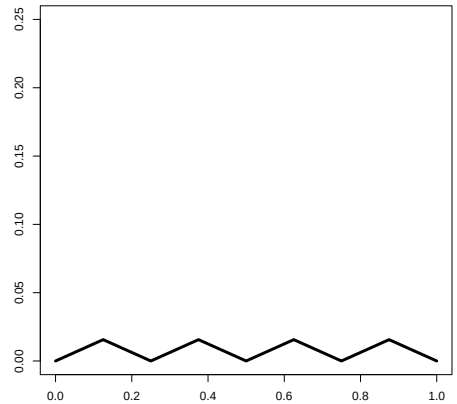
Approximation of square function (cont.)



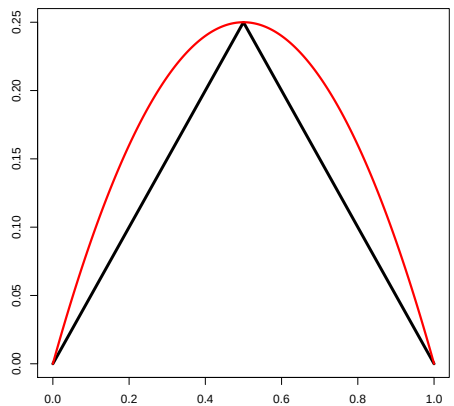
(g) R_1



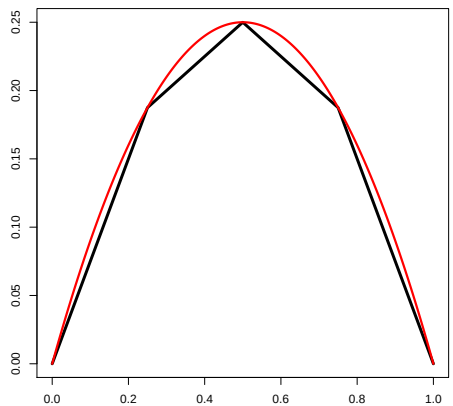
(h) R_2



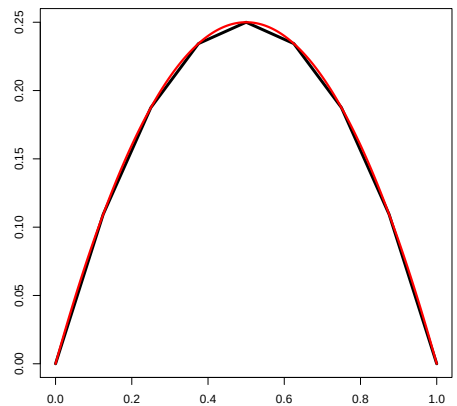
(i) R_3



(j) F_1



(k) F_2



(l) F_3

Approximation of polynomials

- A polynomial consists of finitely many additions and multiplications.
- Since deep ReLU networks can approximate addition and multiplication, it is straightforward to construct networks that approximate arbitrary polynomials.
- Such networks can approximate any polynomial with approximation error decreasing geometrically as the number of (nonzero) network parameters increases.
  - We omit the technical details.

Localization

- Taylor expansion of $f \in \mathcal{C}^\beta[0,1]^d$ at $\mathbf{a}$:

$$f(\mathbf{x}) \approx P_{\mathbf{a}}^{\beta} f(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|<\beta} (\partial^{\boldsymbol{\alpha}} f)(\mathbf{a}) \frac{(\mathbf{x}-\mathbf{a})^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!}$$

- Consider Taylor expansions centered at grid points:

$$\mathbf{D}(M) = \left\{ \left(\frac{\ell_1}{M}, \ldots, \frac{\ell_d}{M} \right) : \ell_j \in \{0, 1, \ldots, M\} \right\}.$$

- Partition of unity on $[0,1]^d$:

$$1 = \sum_{\mathbf{a} \in \mathbf{D}(M)} \prod_{j=1}^{d} \rho\big(1 - M|x_j - a_j|\big)$$

Localization (cont.)

- Therefore,

$$f(\mathbf{x}) \approx \sum_{\mathbf{a}\in\mathbf{D}(M)} P_{\mathbf{a}}^{\beta} f(\mathbf{x}) \prod_{j=1}^{d} \rho\big(1 - M|x_j - a_j|\big).$$

- Each $P_{\mathbf{a}}^{\beta}$ is a polynomial and can thus be efficiently approximated by deep ReLU networks.
- The localization functions $\prod_{j=1}^{d} \rho\big(1 - M|x_j - a_j|\big)$ can also be approximated by deep ReLU networks with coefficients bounded by 1.
- We omit technical details.

Approximation of smooth functions by DNN

- In summary, every function in $\mathcal{C}^\beta$ can be efficiently approximated by deep ReLU networks.
- Specifically, for any $f \in \mathcal{C}^\beta$ (with bounded norm) and $\epsilon > 0$, one can construct a sparse neural network function f_{NN} with $O(\epsilon^{-d/\beta})$ nonzero coefficients such that

$$\|f - f_{\text{NN}}\|_\infty \leq \epsilon.$$

- In Schmidt-Hieber's construction, all network coefficients are bounded by 1, and the number of hidden layers satisfies $L \asymp \log \epsilon^{-1}$.

Approximation of smooth functions by DNN (cont.)

- There are various approaches for studying the approximation properties of DNNs.
  - Bounded number of hidden layers, i.e., $L \asymp 1$ (Petersen and Voigtlaender, 2018)
  - Very deep and thin networks (Park et al., 2021)
  - General activation functions (Ohn and Kim, 2019)
  - Functions on manifolds (Chen et al., 2019)
  - Wavelet-based constructions (Daubechies et al., 2022)
  - ...

Petersen, P. & Voigtlaender, F. "Optimal approximation of piecewise smooth functions using deep ReLU neural networks". *Neural Networks*. 2018

Park, S., Yun, C., Lee, J. & Shin, J. "Minimum width for universal approximation". *Proc. ICLR*. 2021

Ohn, I. & Kim, Y. "Smooth function approximation by deep neural networks with general activation functions". *Entropy*. 2019

Chen, M., Jiang, H., Liao, W. & Zhao, T. "Efficient approximation of deep ReLU networks for functions on low dimensional manifolds". *Proc. NeurIPS*. 2019

Daubechies, I. *et al.* "Neural network approximation of refinable functions". *IEEE Trans. Inform. Theory*. 2022

Remarks

- The approximation theory of DNNs plays an important role in understanding deep learning.
- However, approximation theory alone cannot fully explain the remarkable performance of deep learning in practice.
- A comprehensive theory of deep learning should also account for estimation and optimization errors.

Thank you for attention!