데이터분석기법

12주차: Deep Learning, Multi-Layer Perceptron, and Backpropagation

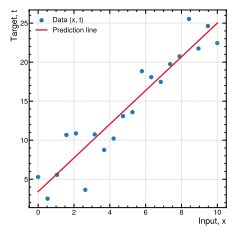
양승진
slowmoyang@gmail.com

세종대학교

2025년 5월 20일

Linear Model vs. XOR Problem

Linear Regression Model

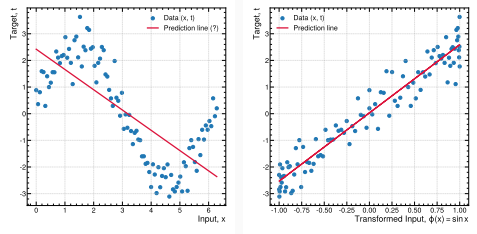


- In the 1D case, linear regression is a ML model of the form:

$$y = f(x) = wx + b, \tag{1}$$

  - where w is **weight** and b is the **bias** – both are trainable **parameters**
- The output y has a linear relationship with the input x
- Note that the linear model is also linear w.r.t. the parameters w

Basis Function



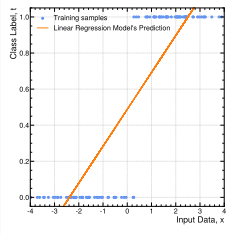
Comparison of linear regression fits using raw input x (left) versus a sine-transformed basis $\phi(x) = \sin x$ (right).

- A **basis function** transforms $\mathbf{x}$ to enable a linear regression model to capture nonlinearity between t and $\mathbf{x}$

$$y = f(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^{M} w_j \phi_j(\mathbf{x}), \tag{2}$$

  - where $\phi_0(\mathbf{x}) = 1$

Classification Problem



- In classification, the target t is a discrete value indicating the class to which each input belongs (e.g. cat: 0, dog: 1)
- In binary classification,
  - $t = 0$: negative
  - $t = 1$ positive
- Linear regression models do not work well on classification problems because of the nonlinearity between the input x and the discrete class labels t

Logistic Regression Model

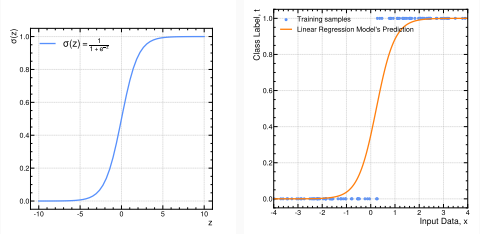
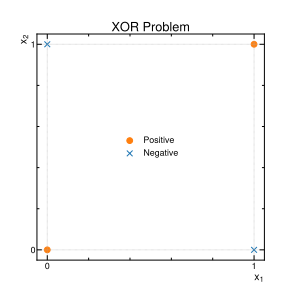


Illustration of the sigmoid function (left) and the logistic regression model (right)

- **Logistic regression model**: $y = \sigma(\mathbf{w}^T\mathbf{x} + b)$
- $z = \mathbf{w}^T\mathbf{x} + b$: logit
- $\sigma(z) = \frac{1}{1+e^{-z}}$: the logistic or sigmod function

XOR problem



- The exclusive OR (XOR) problem is a binary classification task whose output is true only when exactly one of its two inputs is true, making it non-linearly separable and unsolvable by a simple linear model
- Unless we hand-craft highly informative basis functions, even a logistic regression model cannot solve the XOR problem
- Now it's time to go deep

Deep Learning

Directed Acyclic Graphs

- A directed graph without cycles, commonly used to describe computation
- Edges: data
- Nodes: operations
- A node can be a DAG itself, providing high level abstractions
- The main idea behind many DL frameworks like PyTorch and TensorFlow

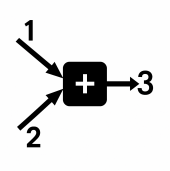
■ Math

$$1 + 2 = 3$$

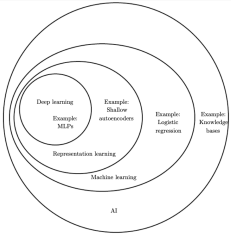
■ Code

```
>>> import torch
>>> x = torch.tensor(1)
>>> y = torch.tensor(2)
>>> torch.add(x, y)
torch.tensor(3)
```

■ DAG



Deep Learning

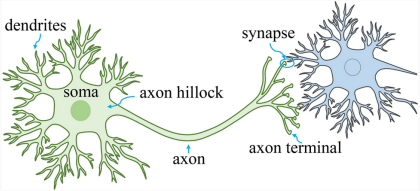


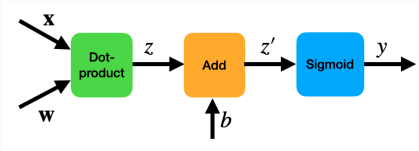
Source: [1]

- "Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned." [2]

Perceptron

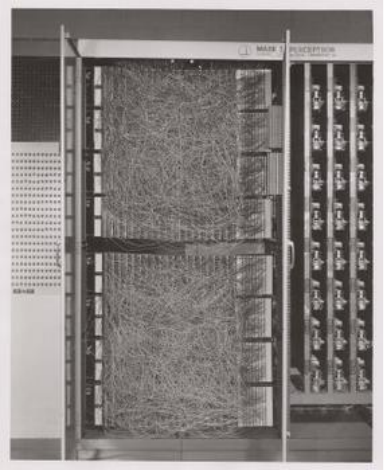
- In 1958, Frank Rosenblatt invented the perceptron, which was originally developed as a binary classifier
- Perceptron: $y = f(\mathbf{x}; \theta) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$
  - $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n$ and $y, b \in \mathbb{R}$
  - $\theta = \{\mathbf{w}, b\}$ adjustable parameters
- σ is called a non-linear function called activation
  - e.g. sigmoid, $\sigma(z) = \frac{1}{1+e^{-z}}$





Perceptron (cont.)

- In 1958, Frank Rosenblatt invented the perceptron, which was originally developed as a binary classifier
- Perceptron: $y = f(\mathbf{x}; \theta) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$
  - $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n$ and $y, b \in \mathbb{R}$
  - $\theta = \{\mathbf{w}, b\}$ adjustable parameters
- σ is called a non-linear function called activation
  - e.g. sigmoid, $\sigma(z) = \frac{1}{1+e^{-z}}$



Mark I Perceptron machine

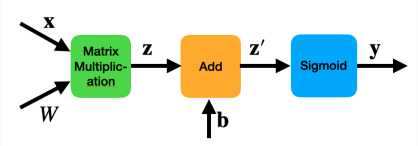
Perceptron: Many-to-Many

- A perceptron can be extended to predict a vector instead of a scalar

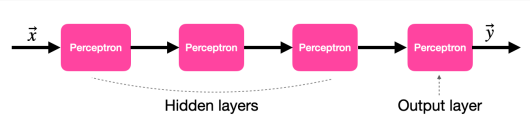
$$\mathbf{y} = \sigma\left(W\mathbf{x} + \mathbf{b}\right) = \left(\sigma(z_1), \ldots, \sigma(z_M)\right) \tag{3}$$

  - $\mathbf{x} \in \mathbb{R}^n$, $W \in \mathbb{R}^{m \times n}$, and $\vec{y}, \vec{b} \in \mathbb{R}^m$

- Activation function is applied into each element of an input vector

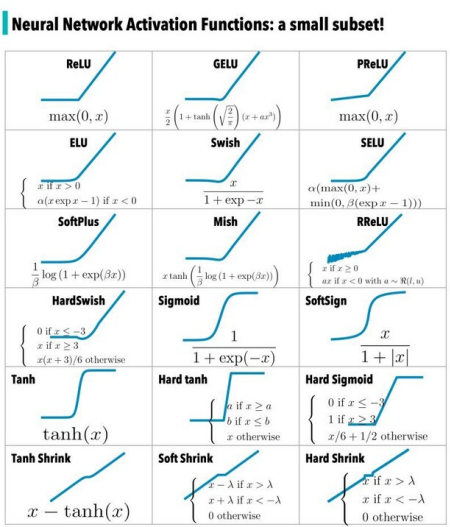


Multi-Layer Perceptron



- A **feedforward neural network (FFNN)** or **multilayer perceptron (MLP)** is just a function composed of multiple Perceptrons
- **The universal approximation theorem**: "**an MLP with a linear output layer and at least one hidden layer with any squashing activation function** (such as the logistic sigmoid activation function) **can approximate any** Borel measurable **function** from one finite-dimensional space to another with any desired non-zero amount of error, **provided that the network is given enough hidden units.**"
- MLP can solve the XOR problem! $\longrightarrow$https://playground.tensorflow.org

Activation Functions



Backpropagation

Gradient Descent

- Gradient descent (GD) is an iterative method—it updates a model step by step until a stopping condition is satisfied
- Starting from an initial guess (like setting all weights to zero), we repeatedly update the weights in the direction that most reduces the cost

$$w_j \leftarrow w_j - \eta \frac{\partial L}{\partial w_j}, \tag{4}$$

  where η is the learning rate
- In modern ML softwares such as SCIKIT-LEARN, PYTORCH and TENSORFLOW, GD is fully automated

Backpropagation

- The backpropagation is a gradient descent algorithm with the chain-rule

$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

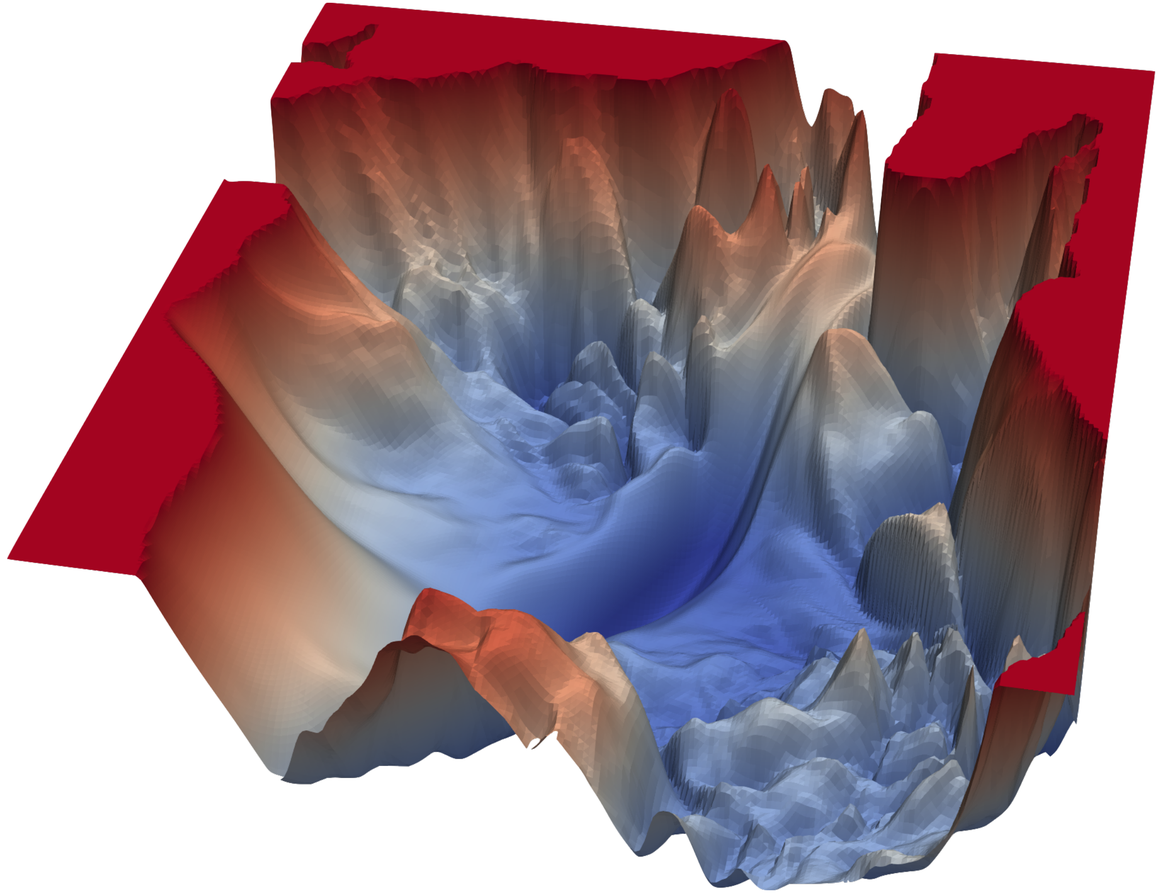
$$\Delta y = \frac{\partial y}{\partial x} \Delta x$$

$$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

Gradient Descent

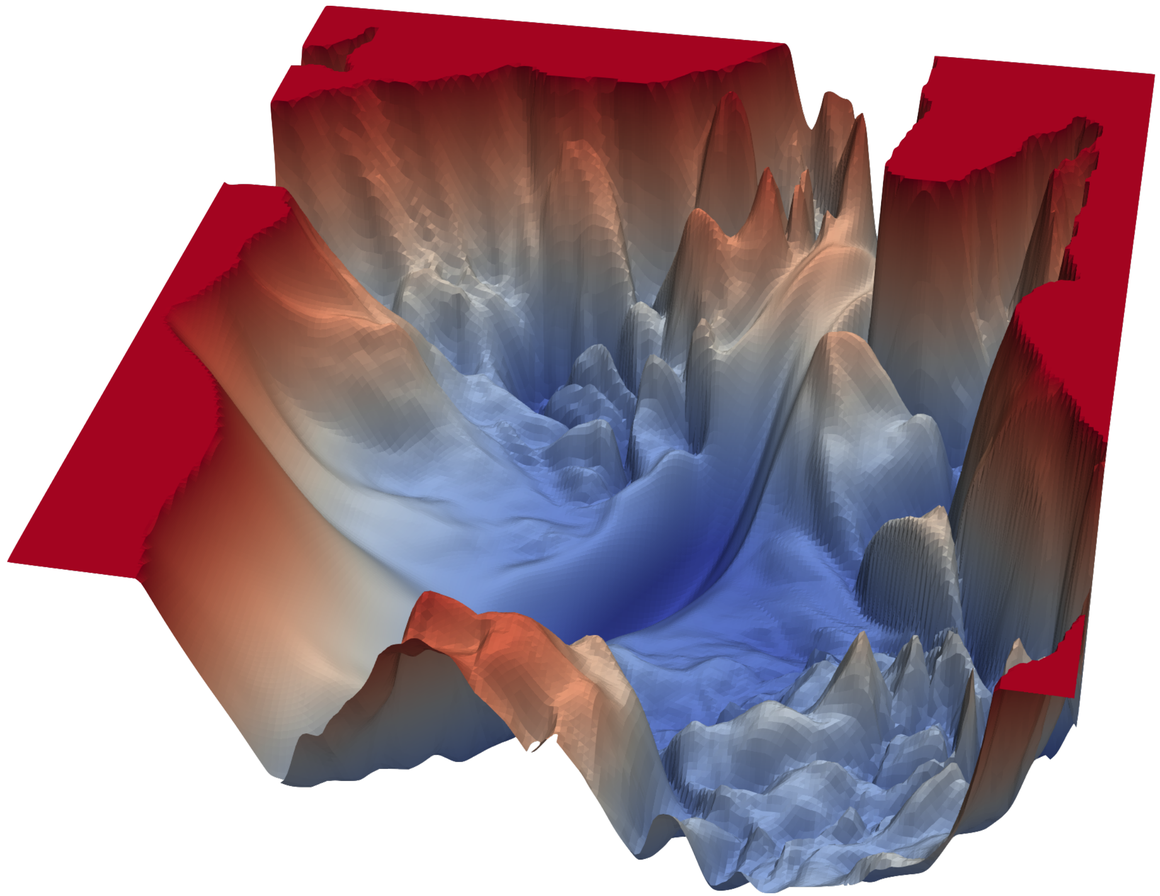
- Require:
  - $\mathbb{D}, f(\mathbf{x}; \theta), J$
  - η, learning rate
- Loop until convergence:
  - Compute a loss function,
    $$J(\theta) = \frac{1}{|\mathbb{D}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathbb{D}} J(f(\mathbf{x}; \theta), \mathbf{y})$$
  - Compute gradients: $\frac{\partial J}{\partial \theta}$
  - Update parameters: $\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}$



Source: arXiv:1712.09913v3

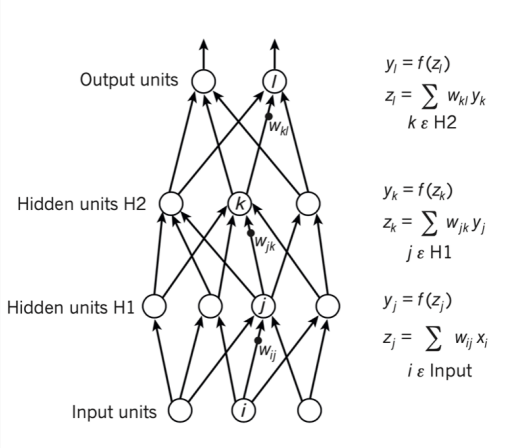
Stochastic Gradient Descent

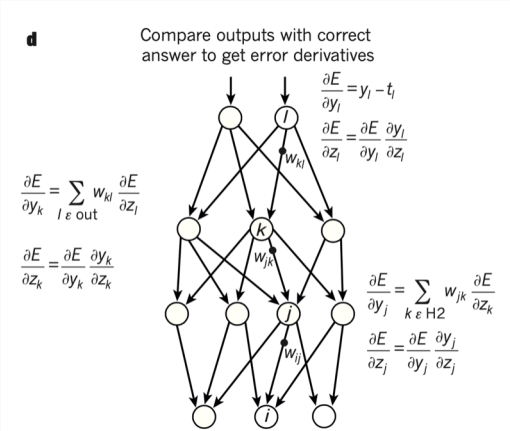
- Require:
  - $\mathbb{D}, f(\mathbf{x}; \theta)$, and J
  - η, learning rate
  - B, batch size, $(B \leq |\mathbb{D}|)$
- Loop until convergence
  - Sample a minibatch of examples drawn from $\mathbb{D}$:
    $\mathbb{B} = \left\{ \left(\mathbf{x}^{(m)}, \mathbf{y}^{(m)} \right) \mid 1 \leq m \leq B \right)$, where $(\mathbf{x}_m, \mathbf{y}_m) \sim \mathbb{D}$
  - Compute a loss function: $J(\theta) = \dfrac{1}{|\mathbb{B}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathbb{B}} J(f(\mathbf{x}; \theta), \mathbf{y})$
  - Compute gradients: $\dfrac{\partial J}{\partial \theta}$
  - Update parameters: $\theta \leftarrow \theta - \eta \dfrac{\partial J}{\partial \theta}$



Source: arXiv:1712.09913v3

Backpropagation (cont.)



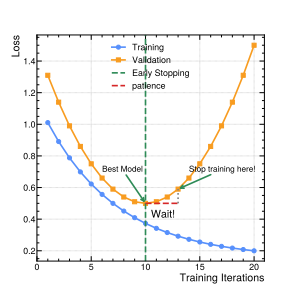


Prediction (left) and gradient calculation (right)

Stochastic Gradient Descent

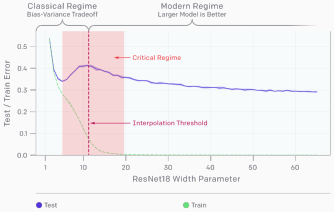
- fixme

Early Stopping



- Stop training before the model starts to overfit

Early Stopping: Deep Double Descent



Source: [3]

- Deep double descent refers to the phenomenon where a model's test error initially decreases, then rises again around the point where it just fits the training data, and finally decreases a second time as the model becomes increasingly overparameterized
- When training a DL model on a large dataset, the early stopping patience might be required to be a sufficiently large value

Questions?

References i

I. Goodfellow, Y. Bengio, and A. Courville.
***Deep Learning*.**
MIT Press, 2016.
http://www.deeplearningbook.org.

Y. LeCun, Y. Bengio, and G. Hinton.
Deep learning.
nature, 521(7553):436–444, 2015.

References ii

P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever.
Deep double descent: Where bigger models and more data hurt.
Journal of Statistical Mechanics: Theory and Experiment, 2021(12):124003, 2021.