CS 6316 Machine Learning

Neural Networks

Yangfeng Ji

Department of Computer Science
University of Virginia

UNIVERSITY *of* VIRGINIA | ENGINEERING

Overview

From Logistic Regression to Neural Networks

Logistic Regression

▶ An unified form for $y \in \{-1, +1\}$

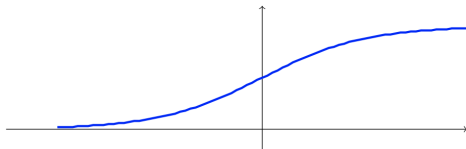$$p(Y = +1 \mid x) = \frac{1}{1 + \exp(-\langle w, x \rangle)} \qquad (1)$$

Logistic Regression

▶ An unified form for $y \in \{-1, +1\}$

$$p(Y = +1 \mid x) = \frac{1}{1 + \exp(-\langle w, x \rangle)} \qquad (1)$$

▶ The sigmoid function $\sigma(a)$ with $a \in \mathbb{R}$

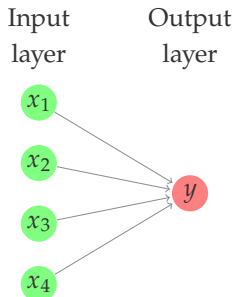$$\sigma(a) = \frac{1}{1 + \exp(-a)} \qquad (2)$$

Graphical Representation

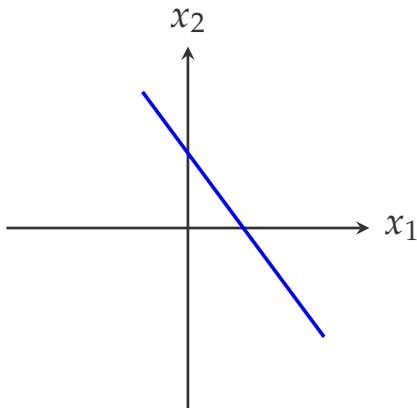▶ A specific example of LR

$$p(Y = 1 \mid \boldsymbol{x}) = \sigma(\sum_{j=1}^{4} w_j \boldsymbol{x}_{\cdot,j}) \qquad (3)$$

▶ The graphical representation of this LR model is

4

Capacity of a LR

Logistic regression gives a linear decision boundary

From LR to Neural Networks

Build upon logistic regression, a simple neural network can be constructed as

$$z_k = \sigma(\sum_{j=1}^{d} w_{k,j}^{(1)} x_{\cdot,j}) \quad k \in [K] \tag{4}$$

$$P(y = 1 \mid x) = \sigma(\sum_{k=1}^{K} w_k^{(o)} z_k) \tag{5}$$

- $x \in \mathbb{R}^d$: d-dimensional input
- $y \in \{-1, +1\}$ (binary classification problem)

From LR to Neural Networks

Build upon logistic regression, a simple neural network can be constructed as

$$z_k = \sigma(\sum_{j=1}^{d} w_{k,j}^{(1)} x_{\cdot,j}) \quad k \in [K] \qquad (4)$$

$$P(y = 1 \mid \boldsymbol{x}) = \sigma(\sum_{k=1}^{K} w_k^{(o)} z_k) \qquad (5)$$

- ▶ $\boldsymbol{x} \in \mathbb{R}^d$: d-dimensional input
- ▶ $y \in \{-1, +1\}$ (binary classification problem)
- ▶ $\{w_{k,i}^{(1)}\}$ and $\{w_k^{(o)}\}$ are two sets of the parameters, and
- ▶ K is the number of hidden units, each of them has the same form as a LR.

Mathematical Formulation

▶ Element-wise formulation

$$z_k = \sigma(\sum_{j=1}^{d} w_{k,j}^{(1)} x_{\cdot,j}) \quad k \in [K] \qquad (6)$$

$$P(y = +1 \mid x) = \sigma(\sum_{k=1}^{K} w_k^{(o)} z_k) \qquad (7)$$

Mathematical Formulation

▶ Element-wise formulation

$$z_k = \sigma\left(\sum_{j=1}^{d} w_{k,j}^{(1)} x_{\cdot,j}\right) \quad k \in [K] \quad (6)$$

$$P(y = +1 \mid x) = \sigma\left(\sum_{k=1}^{K} w_k^{(o)} z_k\right) \quad (7)$$

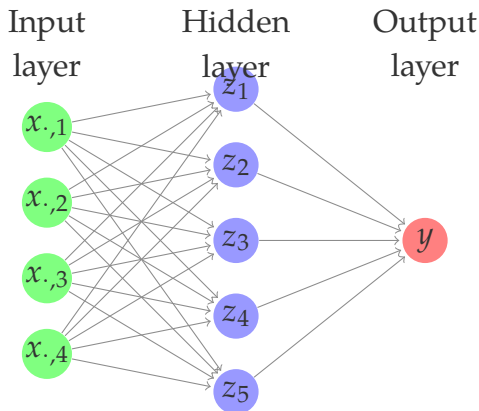▶ Matrix-vector formulation

$$z = \sigma(\mathbf{W}^{(1)} x) \quad (8)$$

$$P(y = +1 \mid x) = \sigma((\mathbf{w}^{(o)})^{\mathsf{T}} z) \quad (9)$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{K \times d}$ and $\mathbf{w}^{(o)} \in \mathbb{R}^{K}$

Graphical Representation
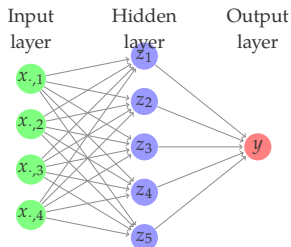

- Depth: 2 (two-layer neural network)
- Width: 5 (the maximal number of units in each layer)

Hypothesis Space

The hypothesis space of neural networks is usually
defined by the **architecture** of the network, which includes

- the nodes in the network,
- the connections in the network, and
- the **activation function** (e.g., σ)

Other Activation Functions
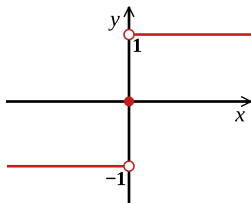
(a) Sign function

Other Activation Functions
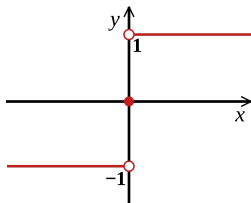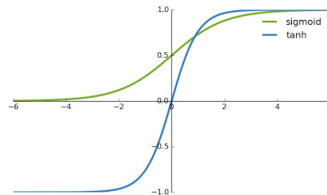
(a) Sign function

(b) Tanh function

Other Activation Functions
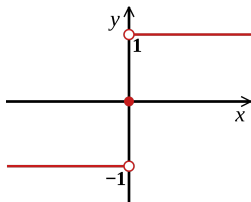
(a) Sign function
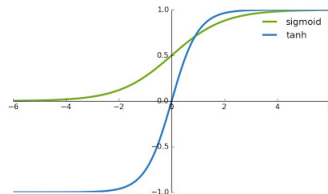
(b) Tanh function
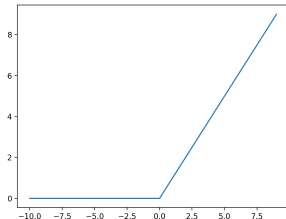
(c) ReLU function
[Jarrett et al., 2009]

Another Network/Hypothesis Space

Simply increasing the number of layers or increase the number of hidden units, we can create another hypothesis space
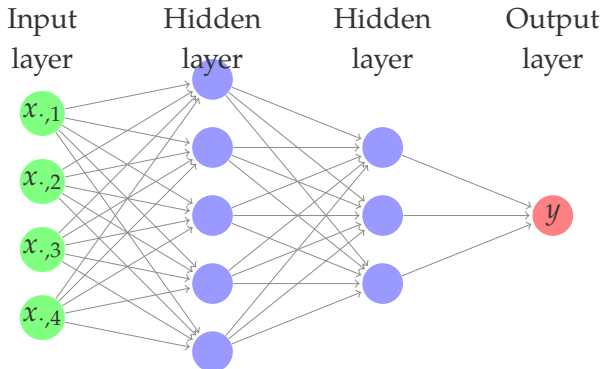
Expressive Power of Neural Networks

Consider a neural network defined by the following functions

$$z_k = \text{sign}(\sum_{j=1}^{d} w_{k,j}^{(1)} x_{\cdot,j}) \quad k \in [K] \qquad (10)$$

$$h(\boldsymbol{x}) = \text{sign}(\sum_{k=1}^{K} w_k^{(o)} z_k) \qquad (11)$$

where $\text{sign}(a)$ is the sign function.

Two-layer NNs with Sign Function

Consider a neural network defined by the following functions

$$z_k = \text{sign}(\sum_{j=1}^{d} w_{k,j}^{(1)} x_{\cdot,j}) \quad k \in [K] \tag{10}$$

$$h(\boldsymbol{x}) = \text{sign}(\sum_{k=1}^{K} w_k^{(o)} z_k) \tag{11}$$

where $\text{sign}(a)$ is the sign function.

$h(\boldsymbol{x})$ can be rewritten as

$$h(\boldsymbol{x}) = \text{sign}\left(\sum_{k=1}^{K} w_k^{(o)} \cdot \text{sign}(\sum_{j=1}^{d} w_{k,i}^{(1)} x_{\cdot,j})\right) \tag{12}$$

Decision Boundary

$h(x)$ is defined by a combination of K linear predictors
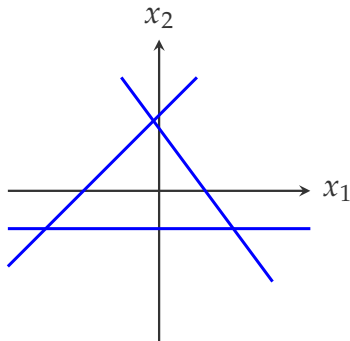
Similar conclusion applies to other activation functions.

[Shalev-Shwartz and Ben-David, 2014, Page 274]

Universal Approximation Theorem

Restrict the inputs $x_{\cdot,j} \in \{-1, +1\} \forall j \in [d]$ as binary

Universal Approximation Theorem

For every d, there exists a two-layer neural network
(Equations 10 − 11), such that this hypothesis space
contains all functions from $\{-1, +1\}^d$ to $\{-1, +1\}$

[Shalev-Shwartz and Ben-David, 2014, Section 20.3]

Universal Approximation Theorem

Restrict the inputs $x_{\cdot,j} \in \{-1, +1\} \forall j \in [d]$ as binary

Universal Approximation Theorem

For every d, there exists a two-layer neural network (Equations 10 – 11), such that this hypothesis space contains all functions from $\{-1, +1\}^d$ to $\{-1, +1\}$

▶ The minimal size of network that satisfies the theorem is exponential in d

▶ Similar results hold for σ as the activation function

[Shalev-Shwartz and Ben-David, 2014, Section 20.3]

Learning Neural Networks

Neural Network Predictions

Consider a binary classification problem with $\mathcal{Y} = \{-1, +1\}$,

- A two-layer neural network gives the following prediction as

$$P(Y = +1 \mid x) = \sigma \left((w^{(o)})^{\mathsf{T}} \sigma(W^{(1)} x) \right) \qquad (13)$$

where $\{w^{(o)}, W^{(1)}\}$ are the parameters

Neural Network Predictions

Consider a binary classification problem with $\mathcal{Y} = \{-1, +1\}$,

- A two-layer neural network gives the following prediction as

$$P(Y = +1 \mid x) = \sigma\left((w^{(o)})^{\mathsf{T}}\sigma(\mathbf{W}^{(1)}x)\right) \qquad (13)$$

  where $\{w^{(o)}, \mathbf{W}^{(1)}\}$ are the parameters

- Assume the ground-truth label is y, let's introduce an empirical distribution

$$q(Y = y' \mid x) = \delta(y', y) = \begin{cases} 1 & y' = y \\ 0 & y' \neq y \end{cases} \qquad (14)$$

Cross Entropy

Given one data point, The loss function of a neural network is usually defined as the cross entropy of the prediction distribution p and the empirical distribution p

$$
\begin{aligned}
H(q, p) \;=\; & -q(Y = +1 \mid \boldsymbol{x}) \log p(Y = +1 \mid \boldsymbol{x}) \\
& -q(Y = -1 \mid \boldsymbol{x}) \log p(Y = -1 \mid \boldsymbol{x}) \quad (15)
\end{aligned}
$$

Cross Entropy

Given one data point, The loss function of a neural network is usually defined as the cross entropy of the prediction distribution p and the empirical distribution p

$$
\begin{aligned}
H(q, p) \;=\; & -q(Y = +1 \mid x) \log p(Y = +1 \mid x) \\
& -q(Y = -1 \mid x) \log p(Y = -1 \mid x) \quad (15)
\end{aligned}
$$

Since q is defined with a Delta function, Depending on y, we have

$$
H(q, p) = \begin{cases} -\log p(Y = +1 \mid x) & Y = +1 \\ -\log p(Y = -1 \mid x) & Y = -1 \end{cases} \quad (16)
$$

Cross Entropy

Given one data point, The loss function of a neural network is usually defined as the cross entropy of the prediction distribution p and the empirical distribution p

$$H(q, p) = -q(Y = +1 \mid x) \log p(Y = +1 \mid x)$$
$$-q(Y = -1 \mid x) \log p(Y = -1 \mid x) \quad (15)$$

Since q is defined with a Delta function, Depending on y, we have

$$H(q, p) = \begin{cases} -\log p(Y = +1 \mid x) & Y = +1 \\ -\log p(Y = -1 \mid x) & Y = -1 \end{cases} \quad (16)$$

It is equivalent to the negative log-likelihood (NLL) function used in learning LR.

▶ Given a set of training example $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$, the loss function is defined as

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \log p(y_i \mid \boldsymbol{x}_i) \tag{17}$$

where $\boldsymbol{\theta}$ indicates all the parameters in a network.

- Given a set of training example $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, the loss function is defined as

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^m \log p(y_i \mid \boldsymbol{x}_i) \tag{17}$$

where $\boldsymbol{\theta}$ indicates all the parameters in a network.

- For example, $\boldsymbol{\theta} = \{\boldsymbol{w}^{(o)}, \mathbf{W}^{(1)}\}$, for the previously defined two-layer neural network

ERM

- Given a set of training example $S = \{(x_i, y_i)\}_{i=1}^m$, the loss function is defined as

$$L(\theta) = -\sum_{i=1}^m \log p(y_i \mid x_i) \tag{17}$$

where θ indicates all the parameters in a network.

- For example, $\theta = \{w^{(o)}, W^{(1)}\}$, for the previously defined two-layer neural network

- Just like learning a LR, we can use gradient-based learning algorithm

Gradient-based Learning

A simple scratch of gradient-based learning[1]

   1. Compute the gradient of $\boldsymbol{\theta}$, $\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

---

[1]More detail will be discussed in the next lecture

A simple scratch of gradient-based learning[1]

1. Compute the gradient of $\boldsymbol{\theta}$, $\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

2. Update the parameter with the gradient

$$\boldsymbol{\theta}^{(\text{new})} \leftarrow \boldsymbol{\theta}^{(\text{old})} - \eta \cdot \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(\text{old})}} \tag{18}$$

where η is the learning rate

---

[1]More detail will be discussed in the next lecture

Gradient-based Learning

A simple scratch of gradient-based learning[1]

1. Compute the gradient of $\boldsymbol{\theta}$, $\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

2. Update the parameter with the gradient

$$\boldsymbol{\theta}^{(\text{new})} \leftarrow \boldsymbol{\theta}^{(\text{old})} - \eta \cdot \left.\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(\text{old})}} \qquad (18)$$

   where η is the learning rate

3. Go back step 1 until it converges

---

[1]More detail will be discussed in the next lecture

Gradient Computation

Consider the two-layer neural network with one training example (x, y), to further simplify the computation, we assume $y = +1$

$$\log p(y \mid x) = \log \sigma \left((w^{(o)})^\mathsf{T} \sigma(W^{(1)}x) \right) \qquad (19)$$

Gradient Computation

Consider the two-layer neural network with one training example (x, y), to further simplify the computation, we assume $y = +1$

$$\log p(y \mid x) = \log \sigma \left((w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x) \right) \qquad (19)$$

The gradient with respect to $w^{(o)}$ is

$$\frac{\partial L(\boldsymbol{\theta})}{\partial w^{(o)}} = -\frac{\partial \log \sigma\left(\cdot \right)}{\partial \sigma\left(\cdot \right)} \cdot \frac{\partial \sigma\left((w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x) \right)}{\partial (w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x)} \cdot \frac{\partial (w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x)}{\partial w^{(o)}}$$

$$(20)$$

Gradient Computation

Consider the two-layer neural network with one training example (x, y), to further simplify the computation, we assume $y = +1$

$$\log p(y \mid x) = \log \sigma \left((w^{(o)})^\mathsf{T} \sigma(W^{(1)} x) \right) \qquad (19)$$

The gradient with respect to $w^{(o)}$ is

$$\frac{\partial L(\theta)}{\partial w^{(o)}} = -\frac{\partial \log \sigma(\cdot)}{\partial \sigma(\cdot)} \cdot \frac{\partial \sigma\left((w^{(o)})^\mathsf{T} \sigma(W^{(1)} x) \right)}{\partial (w^{(o)})^\mathsf{T} \sigma(W^{(1)} x)} \cdot \frac{\partial (w^{(o)})^\mathsf{T} \sigma(W^{(1)} x)}{\partial w^{(o)}}$$

$$= -\left\{ 1 - \sigma\left((w^{(o)})^\mathsf{T} \sigma(W^{(1)} x) \right) \right\} \cdot \sigma(W^{(1)} x) \qquad (20)$$

which is in the similar form as the LR updating equation.

Gradient Computation (II)

The gradient with respect to $W^{(1)}$ is

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{w}^{(o)}} = -\frac{\partial \log \sigma\left(\cdot\right)}{\partial \sigma\left(\cdot\right)} \cdot \frac{\partial \sigma\left((\boldsymbol{w}^{(o)})^{\mathsf{T}}\sigma(\mathbf{W}^{(1)}\boldsymbol{x})\right)}{\partial (\boldsymbol{w}^{(o)})^{\mathsf{T}}\sigma(\mathbf{W}^{(1)}\boldsymbol{x})}$$

$$\cdot \frac{\partial (\boldsymbol{w}^{(o)})^{\mathsf{T}}\sigma(\mathbf{W}^{(1)}\boldsymbol{x})}{\partial \sigma(\mathbf{W}^{(1)}\boldsymbol{x})} \cdot \frac{\partial \mathbf{W}^{(1)}\boldsymbol{x}}{\partial \mathbf{W}^{(1)}} \tag{21}$$

Gradient Computation (II)

The gradient with respect to $W^{(1)}$ is

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{w}^{(o)}} = -\frac{\partial \log \sigma\left(\cdot\right)}{\partial \sigma\left(\cdot\right)} \cdot \frac{\partial \sigma\left((\boldsymbol{w}^{(o)})^{\mathsf{T}}\sigma(\mathbf{W}^{(1)}\boldsymbol{x})\right)}{\partial (\boldsymbol{w}^{(o)})^{\mathsf{T}}\sigma(\mathbf{W}^{(1)}\boldsymbol{x})}$$

$$\cdot\frac{\partial (\boldsymbol{w}^{(o)})^{\mathsf{T}}\sigma(\mathbf{W}^{(1)}\boldsymbol{x})}{\partial \sigma(\mathbf{W}^{(1)}\boldsymbol{x})} \cdot \frac{\partial \mathbf{W}^{(1)}\boldsymbol{x}}{\partial \mathbf{W}^{(1)}} \qquad (21)$$

- ▶ Both of them are the applications of the chain rule in calculus plus some derivatives of basic functions
- ▶ In the literature of neural networks, it is called the back-propagation algorithm [Rumelhart et al., 1986]

Computation Graph

Forward Operations

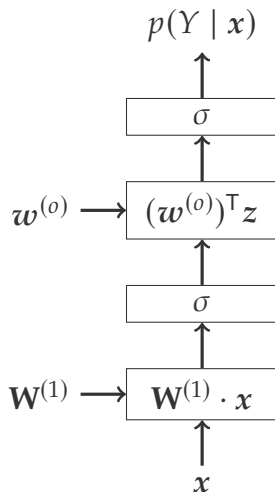Consider the example of a two-layer neural network

$$P(Y = +1 \mid \boldsymbol{x}) = \sigma \left((\boldsymbol{w}^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)}\boldsymbol{x}) \right) \qquad (22)$$

A neural network is a composition of some basic functions and operations. For example

- $\sigma(\cdot)$
- matrix transpose $(\boldsymbol{w}^{(o)})^{\mathsf{T}}$
- matrix-vector multiplication $\mathbf{W}^{(1)}\boldsymbol{x}$

Forward Graph

The computation graph of the two-layer neural network[2]

$$p(Y \mid \boldsymbol{x})$$

$$\sigma$$

$$\boldsymbol{w}^{(o)} \longrightarrow (\boldsymbol{w}^{(o)})^{\mathsf{T}} \boldsymbol{z}$$

$$\sigma$$

$$\mathbf{W}^{(1)} \longrightarrow \mathbf{W}^{(1)} \cdot \boldsymbol{x}$$

$$\boldsymbol{x}$$

---

[2]For simplicity, the transpose operation is ignored from the graph

Backward Operations

Similarly, the gradient of neural network parameters are computed with a series of backward operations associated with the derivative of some basic function. For example

- $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$
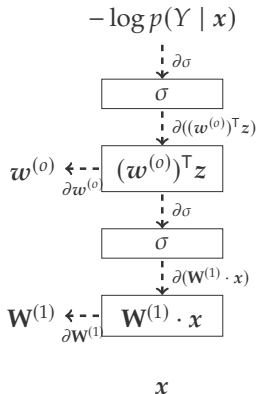- $\frac{\partial \mathbf{a}^\top x}{\partial x} = \mathbf{a}$
- $\frac{\partial \log(x)}{\partial x} = \frac{1}{x}$
- $\frac{\partial \mathbf{W} x}{\partial x} = \begin{bmatrix} x^\top \\ \vdots \\ x^\top \end{bmatrix}$
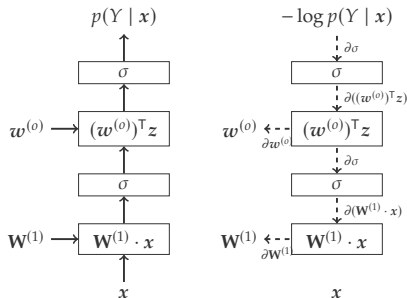
Backward Graph

With the chain rule, gradient of the loss function with respect to any parameter can be computed backward step-by-step along the path
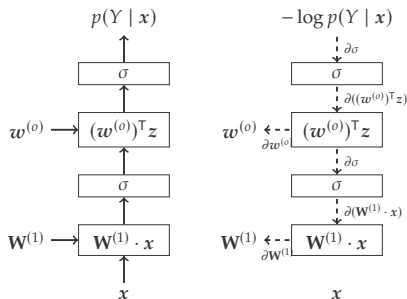
Computation Graph

Perform the forward/backward step with a graph of basic operations (e.g., PyTorch, Tensorflow)

Computation Graph

Perform the forward/backward step with a graph of basic operations (e.g., PyTorch, Tensorflow)

- ▶ Modular implementation: implement each module with its forward/backward operations together
- ▶ Automatic differentiation: automatically run with the backward step

What is Deep Learning?

> **Definition**
>
> **Deep Learning** is building a system by assembling parameterized modules into a (possibly dynamic) computation graph, and training it to perform a task by optimizing the parameters using a gradient-based method.

[LeCun, 2020, AAAI 2020 Keynote]

Reference

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009).
What is the best multi-stage architecture for object recognition?
In *Proceedings of the 12th International Conference on Computer Vision*, pages 2146–2153. IEEE.

LeCun, Y. (2020).
Self-supervised learning.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986).
Learning representations by back-propagating errors.
Nature, 323(6088):533–536.

Shalev-Shwartz, S. and Ben-David, S. (2014).
Understanding machine learning: From theory to algorithms.
Cambridge university press.