

Neural Network

Last update: November 23, 2024



fit@hcmus

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Neural Network — idea

If our linear model is **underfitting** training data, what should we do?

- Approach (1) – feature engineering:
 $\mathbf{x} \rightarrow \text{hand-crafted } \Phi \rightarrow \mathbf{z} \rightarrow \text{linear model} \rightarrow y$
- Approach (2) – Neural Network:
 $\mathbf{x} \rightarrow \text{learned } \Phi \rightarrow \mathbf{z} \rightarrow \text{linear model} \rightarrow y$

Q: Which approach requires **less human brain work**?

A: (2)

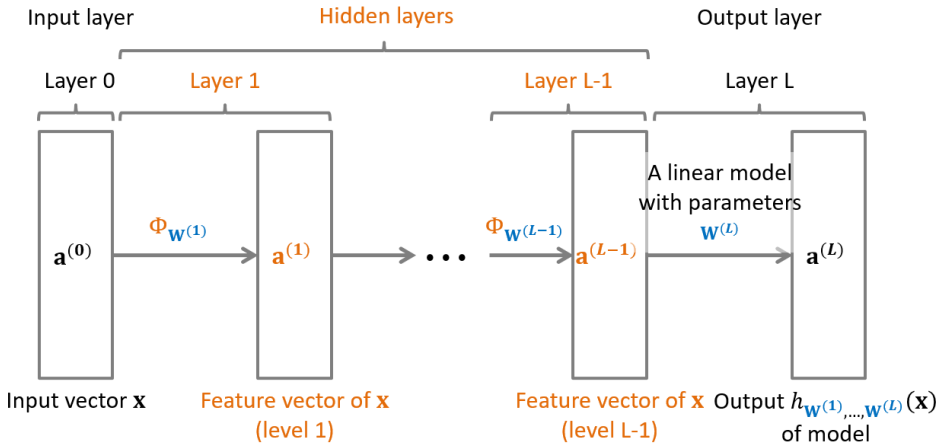
Q: Which approach produces **z less interpretable**?

A: (2)

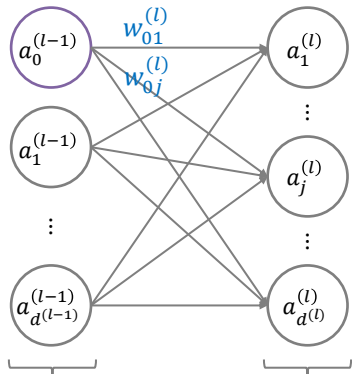
Q: Which approach requires **more training data**?

A: (2)

Neural Network — model form

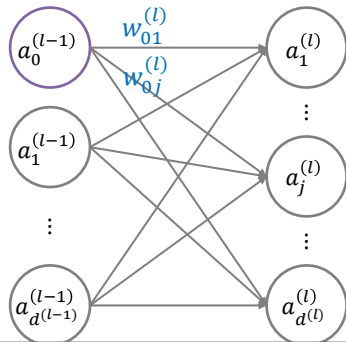


How is $a^{(l)}$ computed from $a^{(l-1)}$?



- Each node is a neuron
- Value inside a node is the output of this neuron
- Each edge is a **weight (parameter)**
- **The output of a neuron is computed as follows:** (1) compute the **weighted** sum of neurons' outputs in the previous layer, (2) pass the result through a nonlinear function – called **activation function** (e.g. logistic)
 - $w_{ij}^{(l)}$: the weight corresponding to the edge from neuron i in layer $l-1$ to neuron j in layer l
($1 \leq l \leq L$, $0 \leq i \leq d^{(l-1)}$, $1 \leq j \leq d^{(l)}$)
 - $a_j^{(l)} = \theta(s_j^{(l)})$ with $s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)}$ and θ is some activation function

How is $a^{(l)}$ computed from $a^{(l-1)}$?



- Each node is a neuron
- Value inside a node is the output of this neuron
- Each edge is a **weight (parameter)**
- **The output of a neuron is computed as follows:** (1) compute the **weighted** sum of neurons' outputs in the previous layer, (2) pass the result through a nonlinear function – called **activation function** (e.g. logistic)
 - $w_{ij}^{(l)}$: the weight corresponding to the edge from neuron i in layer $l-1$ to neuron j in layer l
($1 \leq l \leq L$, $0 \leq i \leq d^{(l-1)}$, $1 \leq j \leq d^{(l)}$)
 - $a_j^{(l)} = \theta(s_j^{(l)})$ with $s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)}$ and θ is some activation function

Common **activation functions**:

- Logistic: $\theta(s) = \frac{1}{1+e^{-s}} \in [0, 1]$
- Tanh: $\theta(s) = 2\text{logistic}(2s) - 1 \in [-1, 1]$
- Relu: $\theta(s) = \max(0, s) \in [0, \infty)$
- ...

Q: Does layer L (last layer) need neuron 0 (whose output value is always 1)?

A: No

hidden layers, # neurons / hidden layer, and activation function are **hyper-parameters** (we must choose them before training)

Q: If we \uparrow # hidden layers as well as # neurons / hidden layer, training error \uparrow/\downarrow ?

A: \downarrow (this is good or bad?)

But it can \uparrow if **difficulties of minimizing E_{train}** kick in:

The deeper (having more hidden layer) the network, the more complex the surface of E_{train}

Neural Network — training

Given training data: $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$
where:

- $\mathbf{x}^{(n)} \in \mathbb{R}^{d+1}$
- $y^{(n)} \in \mathbb{R}$ (regression) or $\{1, 2, \dots, K\}$ (classification)

We should choose model $h_{\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)}}(\mathbf{x})$
with $\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)} = ?$

Steps to find \mathbf{W} 's

Step 1. Define error function on training data

$$E_{\text{train}}(\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)}) = \frac{1}{N} \sum_{n=1}^N e \left(h_{\mathbf{w}^{(1)}, \dots}(\mathbf{x}^{(n)}), y^{(n)} \right)$$

Step 2. $\min_{\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)}} E_{\text{train}}(\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)})$

Can use iterative gradient-based algorithms such as SGD

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \alpha \frac{1}{\text{mini-batch size}} \sum_{(\mathbf{x}, y) \in \text{mini-batch}} \frac{\partial e \left(h_{\mathbf{w}^{(1)}, \dots}(\mathbf{x}), y \right)}{\partial w_{ij}^{(l)}}$$

Inside SGD, can use **back-propagation** algorithm to compute $\frac{\partial e}{\partial w_{ij}^{(l)}} \forall l, i, j$ **efficiently**

Review of chain rule

1 path

$$x \rightarrow y = f_1(x) \rightarrow z = f_2(y)$$

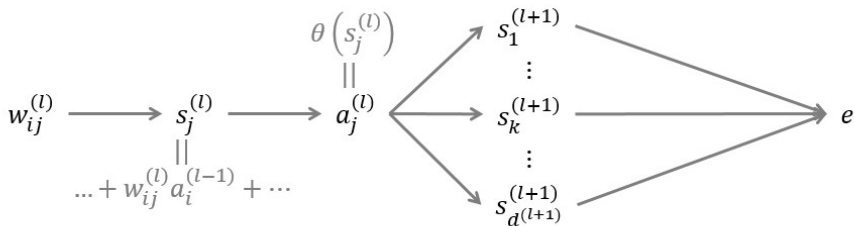
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

More than 1 path

$$x \rightarrow y_1 = f_1(x) \rightarrow z = f_3(y_1, y_2)$$

$$\searrow y_2 = f_2(x) \nearrow$$

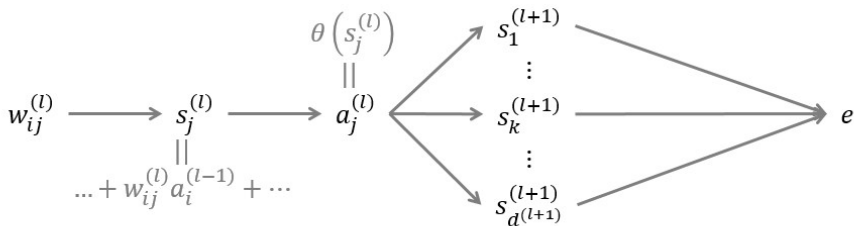
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$



$$\frac{\partial e}{\partial w_{ij}^{(l)}} = \underbrace{\frac{\partial e}{\partial s_j^{(l)}}}_{\delta_j^{(l)}} \underbrace{\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}}_{a_i^{(l-1)}}$$

$$\quad \quad \quad \underbrace{\frac{\partial e}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial s_j^{(l)}}}_{\theta'(s_j^{(l)})}$$

$$\quad \quad \quad \sum_{k=1}^{d^{(l+1)}} \underbrace{\frac{\partial e}{\partial s_k^{(l+1)}}}_{\delta_k^{(l+1)}} \underbrace{\frac{\partial s_k^{(l+1)}}{\partial a_j^{(l)}}}_{w_{jk}^{(l+1)}}$$



$$\frac{\partial e}{\partial w_{ij}^{(l)}} = \underbrace{\frac{\partial e}{\partial s_j^{(l)}}}_{\delta_j^{(l)}} \underbrace{\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}}_{a_i^{(l-1)}}$$

$$\delta_j^{(l)} = \theta'(s_j^{(l)}) \sum_{k=1}^{d^{(l+1)}} w_{jk}^{(l+1)} \delta_k^{(l+1)}$$

To compute $\frac{\partial e}{\partial w_{ij}^{(l)}} \forall l, i, j$, we need to compute:

1. $a_i^{(l-1)} \forall l, i$: $\mathbf{x} = \mathbf{a}^{(0)} \rightarrow \mathbf{a}^{(1)} \rightarrow \dots \rightarrow \mathbf{a}^{(L)}$
2. $\delta_j^{(l)} \forall l, j$: $\underbrace{\delta^{(1)} \leftarrow \dots \leftarrow \delta^{(L-1)} \leftarrow \delta^{(L)}}_{\text{Back-prop}}$

Q: Do we need to compute $\delta_0^{(l)}$ (delta of neuron 0 whose output value is always 1)?

A: No

Q: If layer L (last layer) is Softmax Regression (and the error function is cross-entropy), what is the formula to compute $\delta_j^{(L)}$?

Hint: In "HW2-Slide.pdf", we have this formula for Softmax Regression:

$$\frac{\partial E_{train}}{\partial w_{ij}} = \frac{1}{N} \sum_{n=1}^N x_i^{(n)} (h_{\mathbf{w}}(\mathbf{x}^{(n)})_j - \text{onehot}(y^{(n)})_j)$$

A: $\delta_j^{(L)} = h_{\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)}}(\mathbf{x})_j - \text{onehot}(y)_j$
with $h_{\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)}}(\mathbf{x})_j \equiv a_j^{(L)}$

Sketch of SGD implementation for Neural Network

1. Initialize $\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)}$
(it's important to initialize "properly")
2. Repeat until termination criteria are satisfied:
 - A. Shuffle the order of training examples
 - B. For mini-batch $b = 1, \dots, \frac{N}{B}$:
Do "SGD mini-batch steps" to update $\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(L)}$

SGD mini-batch steps

1. Compute output matrices of all layers:

$$\mathbf{A}^{(0)} = mb_X$$

$$\mathbf{A}^{(l)} = some_funct \left(\mathbf{A}^{(l-1)}, \mathbf{W}^{(l)} \right) \quad l = 1, \dots, L-1$$

$$\mathbf{A}^{(L)} = some_funct \left(\mathbf{A}^{(L-1)}, \mathbf{W}^{(L)} \right)$$

2. Compute delta matrix of layer L :

$$\mathbf{D}^{(L)} = some_funct \left(\mathbf{A}^{(L)}, mb_Y \right)$$

3. Compute gradient of layer L :

$$\mathbf{G}^{(L)} = some_funct \left(\mathbf{A}^{(L-1)}, \mathbf{D}^{(L)} \right)$$

4. Update weight matrix of layer L :

$$\mathbf{W}^{(L)} \leftarrow \mathbf{W}^{(L)} - \alpha \mathbf{G}^{(L)}$$

5. For layer $l = L - 1, \dots, 1$:

- A. Compute delta matrix of layer l :

$$\mathbf{D}^{(l)} = some_funct \left(\mathbf{D}^{(l+1)}, \mathbf{W}^{(l+1)}, \mathbf{A}^{(l)} \right)$$

- B. Compute gradient of layer l : similar to step 3

- C. Update weight matrix of layer l : similar to step 4

Neural Network — dealing with overfitting

- Weight decay (L2 regularization)
- Max norm constraint
- Drop-out
- Data augmentation
- ...