

# Machine Learning I

Bjoern Andres

Machine Learning for Computer Vision  
TU Dresden



<https://mlcv.cs.tu-dresden.de/courses/26-winter/ml1/>

Winter Term 2026/2027

**Contents.** This part of the course is about the supervised learning of composite functions, i.e., **supervised deep learning**.

- ▶ It introduces **(artificial neural) networks**, the family of composite functions defined by a network, and the logistic regression problem for such a family.
- ▶ It introduces two algorithms for computing partial derivatives of a function defined by a network, **forward propagation** and **backward propagation**.
- ▶ It compares these algorithms (exercises).

**Notation.** Let  $G = (V, E)$  a digraph.

- For any  $v \in V$ , let

$$P_v = \{u \in V \mid uv \in E\} \quad \text{the set of **parents** of } v \quad (1)$$

$$C_v = \{w \in V \mid vw \in E\} \quad \text{the set of **children** of } v. \quad (2)$$

- For any  $u, v \in V$ , let  $\mathcal{P}(u, v)$  denote the set of all  $uv$ -paths of  $G$ . (Any path is a subgraph. For any node  $u$ , the  $uu$ -path  $(\{u\}, \emptyset)$  exists.)

Let  $G$  be **acyclic**.

- For any  $v \in V$ , let

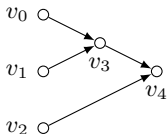
$$A_v = \{u \in V \mid \mathcal{P}(u, v) \neq \emptyset\} \setminus \{v\} \quad \text{the set of **ancestors** of } v \quad (3)$$

$$D_v = \{w \in V \mid \mathcal{P}(v, w) \neq \emptyset\} \setminus \{v\} \quad \text{the set of **descendants** of } v. \quad (4)$$

**Definition.** An **(artificial neural) network** is a tuple  $(V, V_h, V_o, E, \Theta, \{g_{v\theta}: \mathbb{R}^{P_v} \rightarrow \mathbb{R}\}_{v \in (V_h \cup V_o) \setminus V, \theta \in \Theta})$  with the following properties:

- ▶  $G = (V \cup V_h \cup V_o, E)$  is an acyclic digraph.
- ▶ For any  $v \in V$ , called an **input node**,  $P_v = \emptyset$ .
- ▶ For any  $v \in V_o$ , called an **output node**,  $C_v = \emptyset$ .
- ▶ For any  $v \in V_h$ , called a **hidden node**,  $P_v \neq \emptyset$  and  $C_v \neq \emptyset$ .

**Example.** Consider  $V = \{v_0, v_1, v_2\}$ ,  $V_h = \{v_3\}$ ,  $V_o = \{v_4\}$  and the digraph:



Consider, in addition,  $\Theta = \{\theta_0, \theta_1\}$  and

$$g_{v_3\theta}: \mathbb{R}^{\{v_0, v_1\}} \rightarrow \mathbb{R}: \quad x \mapsto x_{v_0} + \theta_0 x_{v_1} \quad (5)$$

$$g_{v_4\theta}: \mathbb{R}^{\{v_2, v_3\}} \rightarrow \mathbb{R}: \quad x \mapsto x_{v_2} + x_{v_3}^{\theta_1} . \quad (6)$$

**Definition.** For any network  $(V, V_h, V_o, E, \Theta, \{g_{v\theta}: \mathbb{R}^{P_v} \rightarrow \mathbb{R}\}_{v \in (V_h \cup V_o) \setminus V, \theta \in \Theta})$ , any  $v \in V \cup V_h \cup V_o$  and any  $\theta \in \Theta$ , let

$$\alpha_{v\theta}: \mathbb{R}^V \rightarrow \mathbb{R}: \quad x \mapsto \begin{cases} x_v & \text{if } v \in V \\ g_{v\theta}(\alpha_{P_v\theta}(x)) & \text{otherwise} \end{cases} . \quad (7)$$

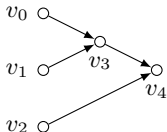
For any  $\theta \in \Theta$ , the **function defined by the network** with parameters  $\theta$  is

$$f_\theta: \mathbb{R}^V \rightarrow \mathbb{R}^{V_o}: \quad x \mapsto \alpha_{V_o\theta}(x) . \quad (8)$$

For any  $x \in \mathbb{R}^V$ :

- ▶  $\alpha_{v\theta}(x)$  is called the **activation** of  $v$  for **input**  $x$  and **parameters**  $\theta$ ,
- ▶  $f_\theta(x)$  is called the **output** of the network for input  $x$  and parameters  $\theta$ .

**Example.** Consider  $V = \{v_0, v_1, v_2\}$ ,  $V_h = \{v_3\}$ ,  $V_o = \{v_4\}$  and the digraph:



Consider, in addition,  $\Theta = \{\theta_0, \theta_1\}$  and

$$g_{v_3\theta}: \mathbb{R}^{\{v_0, v_1\}} \rightarrow \mathbb{R}: \quad x \mapsto x_{v_0} + \theta_0 x_{v_1} \quad (9)$$

$$g_{v_4\theta}: \mathbb{R}^{\{v_2, v_3\}} \rightarrow \mathbb{R}: \quad x \mapsto x_{v_2} + x_{v_3}^{\theta_1} . \quad (10)$$

The network  $(V, V_h, V_o, E, \Theta, \{g_{v_3\theta}, g_{v_4\theta}\})$  defines the function

$$f_\theta: \mathbb{R}^V \rightarrow \mathbb{R}^{V_o}: \quad x \mapsto x_{v_2} + (x_{v_0} + \theta_0 x_{v_1})^{\theta_1} . \quad (11)$$

Let  $(V, V_h, V_o, E, \Theta, g)$  be a network with  $|V_o| = 1$  and  $\Theta = \mathbb{R}^J$  for some finite set  $J \neq \emptyset$ . Let  $f$  be the family of functions defined by this network.

The instance of the  **$l_2$ -regularized logistic regression problem** wrt.  $f$ , labeled data  $T = (S, \mathbb{R}^V, x, y)$  and  $\sigma \in \mathbb{R}^+$  has the objective function  $\varphi: \Theta \rightarrow \mathbb{R}_0^+$  such that for all  $\theta \in \Theta$ :

$$\varphi(\theta) = \sum_{s \in S} \left( -y_s f_\theta(x_s) + \ln \left( 1 + e^{f_\theta(x)} \right) \right) + \frac{1}{2\sigma^2} \|\theta\|_2^2 . \quad (12)$$

Its solutions are those  $\hat{\theta} \in \Theta$  for which

$$\varphi(\hat{\theta}) = \inf \{ \varphi(\theta) \mid \theta \in \Theta \} . \quad (13)$$

### Remark.

- ▶ (12) generalizes  $l_2$ -regularized linear logistic regression
- ▶ If  $f$  is non-linear in  $\theta$ , (12) can be non-convex.

If  $f$  is differentiable, we can apply, e.g., the **steepest descent algorithm** with fixed  $\epsilon, \eta \in \mathbb{R}^+$ :

---

```
 $\theta := 0$ 
repeat
   $d := \nabla \varphi(\theta)$ 
   $\theta := \theta - \eta d$ 
  if  $\|d\| < \epsilon$ 
    return  $\theta$ 
```

---

Next, we will describe two techniques for computing  $\nabla_{\theta} f$ , **forward propagation** and **backward propagation**.

## Learning of Composite Functions – Deep Learning

**Lemma.** Let  $j \in J$ . For any  $v \in V$ :  $\frac{\partial \alpha_{v\theta}}{\partial \theta_j} = 0$ . For any  $v \in (V_h \cup V_o) \setminus V$ :

$$\frac{\partial \alpha_{v\theta}}{\partial \theta_j} = \sum_{u \in (A_v \cup \{v\}) \setminus V} \frac{\partial g_{u\theta}}{\partial \theta_j} \Delta_{uv} \quad (14)$$

with

$$\Delta_{uv} := \sum_{(V', E') \in \mathcal{P}(u, v)} \prod_{(u', v') \in E'} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}}. \quad (15)$$

**Remark.** For any node  $u$ :  $\Delta_{uu} = 1$ . For any  $u, v$  with  $\mathcal{P}(u, v) = \emptyset$ :  $\Delta_{uv} = 0$ .

Proof (idea).

$$\begin{aligned} \frac{\partial \alpha_{v\theta}}{\partial \theta_j} &= \frac{\partial g_{v\theta}}{\partial \theta_j} + \sum_{u \in P_v} \frac{\partial g_{v\theta}}{\partial \alpha_{u\theta}} \frac{\partial \alpha_{u\theta}}{\partial \theta_j} \quad (16) \\ &= \frac{\partial g_{v\theta}}{\partial \theta_j} + \sum_{u \in P_v} \frac{\partial g_{v\theta}}{\partial \alpha_{u\theta}} \frac{\partial g_{u\theta}}{\partial \theta_j} + \sum_{u \in P_v} \sum_{u' \in P_u} \frac{\partial g_{v\theta}}{\partial \alpha_{u\theta}} \frac{\partial g_{u\theta}}{\partial \alpha_{u'\theta}} \frac{\partial \alpha_{u'\theta}}{\partial \theta_j} \\ &= \text{repeated application (16)} \\ &= \sum_{u \in (A_v \cup \{v\}) \setminus V} \frac{\partial g_{u\theta}}{\partial \theta_j} \sum_{(V', E') \in \mathcal{P}(u, v)} \prod_{(u', v') \in E'} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}} \end{aligned}$$

**Lemma (forward propagation).** For all nodes  $u \neq w$  such that  $\mathcal{P}(u, w) \neq \emptyset$ :

$$\Delta_{uw} = \sum_{v \in P_w} \frac{\partial g_{w\theta}}{\partial \alpha_{v\theta}} \Delta_{uv} \quad (17)$$

Proof.

$$\begin{aligned} \Delta_{uw} &= \sum_{(V', E') \in \mathcal{P}(u, w)} \prod_{(u', v') \in E'} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}} \\ &= \sum_{v \in P_w} \sum_{(V'', E'') \in \mathcal{P}(u, v)} \prod_{(u', v') \in E'' \cup \{v, w\}} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}} \\ &= \sum_{v \in P_w} \frac{\partial g_{w\theta}}{\partial \alpha_{v\theta}} \sum_{(V'', E'') \in \mathcal{P}(u, v)} \prod_{(u', v') \in E''} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}} \\ &= \sum_{v \in P_w} \frac{\partial g_{w\theta}}{\partial \alpha_{v\theta}} \Delta_{uv} \end{aligned}$$

□

The **forward propagation algorithm** computes  $\Delta_{uw}$  for one node  $u$  and all nodes  $w$ . It is defined wrt. an arbitrary partial order  $<_P$  of the nodes such that

$$\forall w \in V_h \cup V_o \quad \forall w' \in P_w: \quad w' <_P w . \quad (18)$$

---

**Input:**

Network  $(V, V_h, V_o, E, \Theta, \{g_{v\theta}: \mathbb{R}^{P_v} \rightarrow \mathbb{R}\}_{v \in (V_h \cup V_o) \setminus V, \theta \in \Theta})$

Node  $u \in V \cup V_h \cup V_o$

---

**for**  $w$  ordered by  $<_P$  (18)

**if**  $w = u$

$$\Delta_{uw} := 1$$

**else if**  $\mathcal{P}(u, w) = \emptyset$

$$\Delta_{uw} := 0$$

**else**

$$\Delta_{uw} := \sum_{v \in P_w} \frac{\partial g_{w\theta}}{\partial \alpha_{v\theta}} \Delta_{uv} \quad (17)$$


---

**Lemma (backward propagation).** For all nodes  $u \neq w$  such that  $\mathcal{P}(u, w) \neq \emptyset$ :

$$\Delta_{uw} = \sum_{v \in C_u} \frac{\partial g_{v\theta}}{\partial \alpha_{u\theta}} \Delta_{vw} \quad (19)$$

Proof.

$$\begin{aligned} \Delta_{uw} &= \sum_{(V', E') \in \mathcal{P}(u, w)} \prod_{(u', v') \in E'} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}} \\ &= \sum_{v \in C_u} \sum_{(V'', E'') \in \mathcal{P}(v, w)} \prod_{(u', v') \in E'' \cup \{(u, v)\}} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}} \\ &= \sum_{v \in C_u} \frac{\partial g_{v\theta}}{\partial \alpha_{u\theta}} \sum_{(V'', E'') \in \mathcal{P}(v, w)} \prod_{(u', v') \in E''} \frac{\partial g_{v'\theta}}{\partial \alpha_{u'\theta}} \\ &= \sum_{v \in C_u} \frac{\partial g_{v\theta}}{\partial \alpha_{u\theta}} \Delta_{vw} \end{aligned}$$

□

The **backward propagation algorithm** computes  $\Delta_{uw}$  for one node  $w$  and all nodes  $u$ . It is defined wrt. an arbitrary partial order  $<_C$  of the nodes such that

$$\forall u \in V \cup V_h \quad \forall v \in C_u: \quad v <_C u . \quad (20)$$

---

**Input:**

Network  $(V, V_h, V_o, E, \Theta, \{g_{v\theta}: \mathbb{R}^{P_v} \rightarrow \mathbb{R}\}_{v \in (V_h \cup V_o) \setminus V, \theta \in \Theta})$

Node  $w \in V \cup V_h \cup V_o$

---

**for**  $u$  ordered by  $<_C$  (20)

**if**  $u = w$

$$\Delta_{uw} := 1$$

**else if**  $\mathcal{P}(u, w) = \emptyset$

$$\Delta_{uw} := 0$$

**else**

$$\Delta_{uw} := \sum_{v \in C_u} \frac{\partial g_{v\theta}}{\partial \alpha_{u\theta}} \Delta_{vw} \quad (19)$$


---

### Summary.

- ▶ An **(artificial neural) network** defines a family of composite functions.
- ▶ The  $l_2$ -regularized logistic regression problem for such a family is potentially non-convex.
- ▶ If the family is differentiable, we can apply the steepest descent algorithm. Two algorithms for computing partial derivatives of a function defined by a network are **forward propagation** and **backward propagation**.