AI Applications Lecture 3

Functions Represented by Neural Networks

SUZUKI, Atsushi
Jing WANG

Outline

Introduction

1.1 Review of the Previous Lecture

In the last lecture, we saw that many machine learning models, including neural networks, can be understood within a unified framework.

- Many machine learning models can be formulated as a **parametric function** $\{f_{\boldsymbol{\theta}}\}$, where the specific computation is determined by the values of parameters $\boldsymbol{\theta}$.

1.1 Review of the Previous Lecture

In the last lecture, we saw that many machine learning models, including neural networks, can be understood within a unified framework.

- Many machine learning models can be formulated as a **parametric function** $\{f_{\boldsymbol{\theta}}\}$, where the specific computation is determined by the values of parameters $\boldsymbol{\theta}$.
- The process of solving a task is divided into two phases.
    - **Training:** Using data to find the parameter $\boldsymbol{\theta}^*$ that best solves the task.
    - **Inference:** Using the trained parameter $\boldsymbol{\theta}^*$ to provide a new input to the function $f_{\boldsymbol{\theta}^*}$ and obtain an output.

1.2 Learning Outcomes of This Lecture

Through this lecture, students will aim to be able to perform the following tasks:

- Describe in mathematical terms the function represented by a neural network defined by a general **Directed Acyclic Graph (DAG)**.

1.2 Learning Outcomes of This Lecture

Through this lecture, students will aim to be able to perform the following tasks:

- Describe in mathematical terms the function represented by a neural network defined by a general **Directed Acyclic Graph (DAG)**.
- State the definitions of frequently used terms in modern neural network applications, such as **architecture**, **checkpoint**, and **model**.

1.2 Learning Outcomes of This Lecture

Through this lecture, students will aim to be able to perform the following tasks:

- Describe in mathematical terms the function represented by a neural network defined by a general **Directed Acyclic Graph (DAG)**.
- State the definitions of frequently used terms in modern neural network applications, such as **architecture**, **checkpoint**, and **model**.
- Explain why the distinction between architecture and checkpoint is important for utilizing AI in real-world applications.

1.3 Policy of This Lecture

Although many students may have already studied neural networks, this lecture will deliberately start from their mathematical definition. The goal is to **define neural networks in the most general form possible**.

1.3 Policy of This Lecture

Although many students may have already studied neural networks, this lecture will deliberately start from their mathematical definition. The goal is to **define neural networks in the most general form possible**.

The field of AI and machine learning is developing very rapidly, and the specific models widely known today may well be outdated by the time you graduate.

1.3 Policy of This Lecture

Although many students may have already studied neural networks, this lecture will deliberately start from their mathematical definition. The goal is to **define neural networks in the most general form possible**.

The field of AI and machine learning is developing very rapidly, and the specific models widely known today may well be outdated by the time you graduate.

Therefore, this lecture emphasizes understanding more general and universal concepts that will be applicable in the future and will not become obsolete.

1.3 Policy of This Lecture

One might ask, "Is a fundamental understanding of neural networks necessary if we are to use them as black boxes?" The answer to this question is "Yes."

1.3 Policy of This Lecture

One might ask, "Is a fundamental understanding of neural networks necessary if we are to use them as black boxes?" The answer to this question is "Yes."

In particular, the **distinction between architecture and checkpoint** that we will learn in this lecture is extremely important for using AI in the real world.

1.3 Policy of This Lecture

One might ask, "Is a fundamental understanding of neural networks necessary if we are to use them as black boxes?" The answer to this question is "Yes."

In particular, the **distinction between architecture and checkpoint** that we will learn in this lecture is extremely important for using AI in the real world.

Without understanding this distinction, you will not be able to correctly understand the issue of AI **licenses**, which we will cover in the next lecture. Using AI without regard for its license can lead to legal risks such as copyright infringement.

Preparation: Mathematical Notations

2. Preparation: Mathematical Notations

Here is a review of the basic mathematical notations used in this lecture.

- **Definition:**
  - $(\mathrm{LHS}) := (\mathrm{RHS})$: The left-hand side is defined by the right-hand side.

2. Preparation: Mathematical Notations

Here is a review of the basic mathematical notations used in this lecture.

- **Definition:**
    - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: The left-hand side is defined by the right-hand side.
- **Set:** Denoted by uppercase calligraphic letters (e.g., $\mathcal{A}$).
    - $x \in \mathcal{A}$: element x belongs to set $\mathcal{A}$.
    - $\{\}$: The empty set.
    - $\{a, b, c\}$: The set consisting of elements a, b, c.
    - $\{x \in \mathcal{A} | P(x)\}$: The set of elements in $\mathcal{A}$ for which $P(x)$ is true.
    - $\mathbb{R}$ (reals), $\mathbb{Z}$ (integers), with subscripts like $_{>0}$ (positive) or $_{\geq 0}$ (non-negative).
    - $[1, k]_{\mathbb{Z}} \coloneqq \{1, 2, \ldots, k\}$.

2. Preparation: Mathematical Notations

- **Function:**
  - $f : X \to Y$: f is a map from set X to set Y.
  - $y = f(x)$: The output of f for input $x \in X$ is $y \in Y$.

2. Preparation: Mathematical Notations

- **Function:**
  - $f : X \to Y$: f is a map from set X to set Y.
  - $y = f(x)$: The output of f for input $x \in X$ is $y \in Y$.
- **Vector:** A column of numbers, denoted by bold italic lowercase letters (e.g., $\boldsymbol{v} \in \mathbb{R}^n$). The i-th element is v_i.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \quad \boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

2. Preparation: Mathematical Notations

- **Function:**
  - $f : X \to Y$: f is a map from set X to set Y.
  - $y = f(x)$: The output of f for input $x \in X$ is $y \in Y$.
- **Vector:** A column of numbers, denoted by bold italic lowercase letters (e.g., $\boldsymbol{v} \in \mathbb{R}^n$). The i-th element is v_i.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \quad \boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

- **Matrix:** Denoted by bold italic uppercase letters (e.g., $\boldsymbol{A} \in \mathbb{R}^{m,n}$). The element in the i-th row, j-th column is $a_{i,j}$.

2. Preparation: Mathematical Notations

- **Function:**
    - $f : X \to Y$: f is a map from set X to set Y.
    - $y = f(x)$: The output of f for input $x \in X$ is $y \in Y$.
- **Vector:** A column of numbers, denoted by bold italic lowercase letters (e.g., $\boldsymbol{v} \in \mathbb{R}^n$). The i-th element is v_i.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \quad \boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

- **Matrix:** Denoted by bold italic uppercase letters (e.g., $\boldsymbol{A} \in \mathbb{R}^{m,n}$). The element in the i-th row, j-th column is $a_{i,j}$.
- **Tensor:** In this lecture, simply a multi-dimensional array.
    - Vector = 1st-order tensor, Matrix = 2nd-order tensor.
    - Higher-order tensors are denoted by $\underline{\boldsymbol{A}}$.

What is a Neural Network?

3. What is a Neural Network?

An **Artificial Neural Network (ANN)** is a type of parametric function.

3. What is a Neural Network?

An **Artificial Neural Network (ANN)** is a type of parametric function.

Remark (On Terminology)

"Neural network" originally refers to biological neural circuitry. Engineering models are "artificial neural networks." However, in modern AI, the two are best understood as independent concepts. In this lecture, "neural network" will mean an artificial one.

3. What is a Neural Network?

An **Artificial Neural Network (ANN)** is a type of parametric function.

Remark (On Terminology)

"Neural network" originally refers to biological neural circuitry. Engineering models are "artificial neural networks." However, in modern AI, the two are best understood as independent concepts. In this lecture, "neural network" will mean an artificial one.

Remark (Scope of this Lecture)

All networks in this lecture are **feedforward neural networks** (e.g., MLP, CNN, Transformer). We will not cover non-feedforward networks like Boltzmann machines.

3. What is a Neural Network?

An **Artificial Neural Network (ANN)** is a type of parametric function.

Remark (On Terminology)

"Neural network" originally refers to biological neural circuitry. Engineering models are "artificial neural networks." However, in modern AI, the two are best understood as independent concepts. In this lecture, "neural network" will mean an artificial one.

Remark (Scope of this Lecture)

All networks in this lecture are **feedforward neural networks** (e.g., MLP, CNN, Transformer). We will not cover non-feedforward networks like Boltzmann machines.

In a nutshell, a feedforward neural network is a **parametric function defined by a directed acyclic graph (DAG) where each node has a fixed function, and many edges are associated with parameters (real values).**

Directed Acyclic Graph (DAG)

4. Directed Acyclic Graph (DAG)

To define a neural network's structure, we start with its backbone, a directed graph.

4. Directed Acyclic Graph (DAG)

To define a neural network's structure, we start with its backbone, a directed graph.

Definition (Directed Multigraph)

A **directed multigraph** G is a tuple $G = (\mathcal{V}, \mathcal{E}, \mathrm{Tail}, \mathrm{Head})$, consisting of:

- a set of **nodes** or **vertices** $\mathcal{V}$,
- a set of **edges** or **directed arcs** $\mathcal{E}$,
- and two maps, $\mathrm{Tail} : \mathcal{E} \to \mathcal{V}$ and $\mathrm{Head} : \mathcal{E} \to \mathcal{V}$, which assign a tail and head node to each edge.

4. Directed Acyclic Graph (DAG)

We can define relationships between nodes:

- **Incoming edges** of a node v: $\mathcal{E}_{\text{in}}^{(v)} = \{e \in \mathcal{E} \mid \text{Head}(e) = v\}$.
- **Outgoing edges** of a node v: $\mathcal{E}_{\text{out}}^{(v)} = \{e \in \mathcal{E} \mid \text{Tail}(e) = v\}$.
- A **parent node** of v is the tail of an incoming edge.
- A **child node** of v is the head of an outgoing edge.

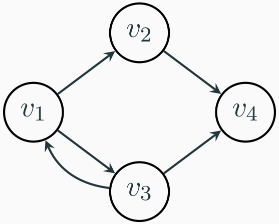


Figure 1: Example of a directed graph. Here, v_1 is a parent of v_2, and v_4 is a child of v_3.

4. Directed Acyclic Graph (DAG)

Definition (Path and Cycle)

In a directed multigraph G:

- A **path** is a sequence of nodes and edges, where each edge connects a node to the next one in the sequence.
- A **cycle** is a path that starts and ends at the same node.

4. Directed Acyclic Graph (DAG)

Definition (Path and Cycle)

In a directed multigraph G:

- A **path** is a sequence of nodes and edges, where each edge connects a node to the next one in the sequence.
- A **cycle** is a path that starts and ends at the same node.

Definition (Directed Acyclic Graph)

When a directed multigraph G has **no cycles**, it is called a **Directed Acyclic Graph (DAG)**.

4. Directed Acyclic Graph (DAG)

Definition (Path and Cycle)

In a directed multigraph G:

- A **path** is a sequence of nodes and edges, where each edge connects a node to the next one in the sequence.
- A **cycle** is a path that starts and ends at the same node.

Definition (Directed Acyclic Graph)

When a directed multigraph G has **no cycles**, it is called a **Directed Acyclic Graph (DAG)**.

Intuitively, a DAG allows only "one-way" flows. This means the nodes can be **topologically sorted**.

4. Directed Acyclic Graph (DAG)

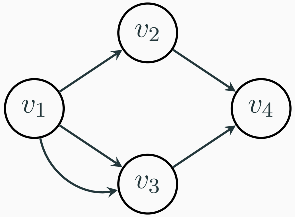


Figure 2: Example of a DAG (including multi-edges)

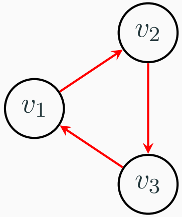


Figure 3: Example that is not a DAG (a cycle $v_1 \to v_2 \to v_3 \to v_1$ exists)

4. Directed Acyclic Graph (DAG)

In a DAG, there is always at least one node with no incoming edges.

4. Directed Acyclic Graph (DAG)

In a DAG, there is always at least one node with no incoming edges.

These nodes become the **input nodes** that receive the input when the neural network is viewed as a function.

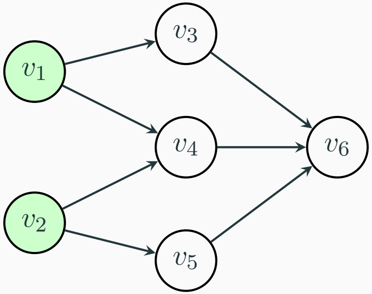

$$\begin{tikzcd}
 & v_3 \arrow[dr] & \\
v_1 \arrow[ur] \arrow[r] & v_4 \arrow[r] & v_6 \\
v_2 \arrow[ur] \arrow[r] & v_5 \arrow[ur] &
\end{tikzcd}$$

Figure 4: Example of a DAG. Nodes v_1, v_2 (green) are the input nodes.

The Function Represented by a Neural Network

5. The Function Represented by a Neural Network

A neural network is a collection of simple computational units (nodes and edges) connected in a DAG structure. The entire network represents a massive composite function.

5. The Function Represented by a Neural Network

A neural network is a collection of simple computational units (nodes and edges) connected in a DAG structure. The entire network represents a massive composite function.

- **Nodes:** Receive values from parent nodes or network inputs, apply a fixed **activation function**, and compute a new value.

5. The Function Represented by a Neural Network

A neural network is a collection of simple computational units (nodes and edges) connected in a DAG structure. The entire network represents a massive composite function.

- **Nodes:** Receive values from parent nodes or network inputs, apply a fixed **activation function**, and compute a new value.
- **Edges:** Receive a value from their tail node, multiply it by a parameter value (a **weight**), and pass it to their head node.

5. The Function Represented by a Neural Network

First, we define the parameter-independent part, the **architecture**.

Definition (Neural Network Architecture)

A neural network architecture is rigorously defined by a tuple including:

1. **Computation Graph:** A DAG $G = (\mathcal{V}, \mathcal{E}, \mathsf{Tail}, \mathsf{Head})$, where nodes are topologically sorted integers.
2. **Input/Output Dimensions:** The overall network's input dimension d_{input} and output dimension d_{output}.
3. **Node Specifications:** For each node $v \in \mathcal{V}$:
   - An **activation function** $\mathrm{g}^{(v)} : \mathbb{R}^{d_{\text{arg}}^{(v)}} \to \mathbb{R}^{d_{\text{ret}}^{(v)}}$.
   - An **argument source tuple** $\mathsf{NodeSrc}^{(v)}$, specifying where each argument for $\mathrm{g}^{(v)}$ comes from (either a network input or an incoming edge).

5. The Function Represented by a Neural Network

Definition (Neural Network Architecture (cont.))

4. **Edge Specifications:** For each edge $e \in \mathcal{E}$, an **edge source index** $\mathrm{EdgeSrc}(e)$ indicating which component of its tail node's return vector to use.
5. **Parameter Specifications:**
   - The total number of parameters d_{param}.
   - A **weight map** $\mathrm{Weight} : \mathcal{E} \to [1, d_{\mathrm{param}}]_{\mathbb{Z}}$ that assigns a parameter to each edge (allowing for **parameter sharing**).
   - Sets of trainable (**variable**) and non-trainable (**fixed**) parameters.
6. **Output Specifications:** An **output source map** OutSrc specifying which component of which node's return vector corresponds to each component of the overall network output.

5. The Function Represented by a Neural Network

Definition (The Function Represented by a Neural Network)

Given an architecture and specific parameter values (a **checkpoint**) $\boldsymbol{\theta} \in \mathbb{R}^{d_{\text{param}}}$, the function $f_{\boldsymbol{\theta}} : \mathbb{R}^{d_{\text{input}}} \to \mathbb{R}^{d_{\text{output}}}$ is computed as follows for an input $\boldsymbol{x}$:

1. **Per-Node Computation:** In topological order ($v \in \mathcal{V}$):
   1.1 Construct the argument vector $\boldsymbol{a}^{(v)}$ for node v's activation function $\mathrm{g}^{(v)}$. Each argument is taken from either a network input x_j or a weighted value from a parent node, $\theta_p \times r_k^{(u)}$.
   1.2 Compute the return value vector for node v: $\boldsymbol{r}^{(v)} := \mathrm{g}^{(v)}(\boldsymbol{a}^{(v)})$.
2. **Overall Network Output:** After all nodes are computed, construct the output vector $\boldsymbol{y}$ by picking the specified return values from the nodes.

5. The Function Represented by a Neural Network

Remark (On Generality)

This definition is very general.

- It can represent common operations like the weighted sum in MLPs by defining the activation function appropriately (e.g., $\mathrm{g}^{(v)}(\boldsymbol{a}) =$ ReLU$(\sum_i a_i)$).
- It can also represent complex layers like convolutions in CNNs and attention in Transformers.
- With mild constraints on the activation functions (e.g., local Lipschitz continuity), the parameters for this entire general class of functions can be trained using a unified method (backpropagation + gradient descent).

This means we don't need a separate training theory for each new architecture.

5.1 Specific Computation Examples

Let’s look at a few examples to see how this formal definition can represent various functions.

5.1 Specific Computation Examples

Let's look at a few examples to see how this formal definition can represent various functions.

The key takeaways from this section are:

- Simple models like linear and logistic regression are just special cases of neural networks. A neural network is a broad concept that includes these.

5.1 Specific Computation Examples

Let's look at a few examples to see how this formal definition can represent various functions.

The key takeaways from this section are:

- Simple models like linear and logistic regression are just special cases of neural networks. A neural network is a broad concept that includes these.
- The architecture of a neural network can be very flexible and irregular. The design freedom is immense, opening the door for future innovations.

5.1 Specific Computation Examples

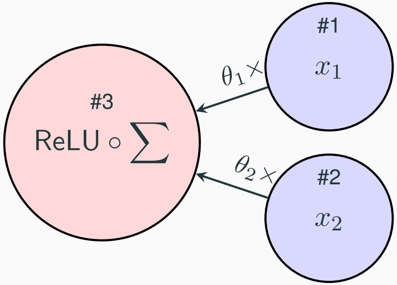


Figure 5: Legend for DAG representation of a neural network.

- **Nodes:** Circles are computational units. Blue for input, red for output. Inside is the node number and activation function.
- **Activation Function Shorthand:** A function name like 'ReLU' on a node with multiple inputs is shorthand for "sum the inputs, then apply the function" (e.g.,

5.1 Example: Linear Regression

Consider the linear regression model $f(\boldsymbol{x}) = w_1x_1 + w_2x_2 + b$. This can be represented by the following architecture.

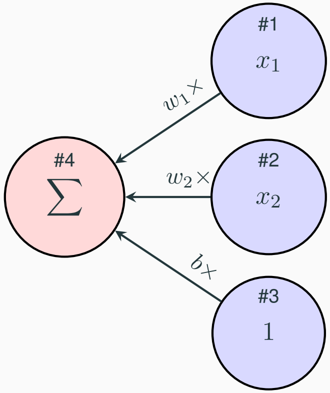


Figure 6: DAG representation of a linear regression

Architecture:

- **Nodes:** 1, 2, 3 for inputs (x_1, x_2, and a constant 1 for the bias).
- **Node 4:** Output node. Activation function is summation ($\sum$).
- **Edges:** Weights are parameters w_1, w_2, b.

5.1 Example: Linear Regression

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [w_1, w_2, b]^\top = [1.0, -2.0, 0.5]^\top$.
- Input: $\boldsymbol{x} = [x_1, x_2]^\top = [3.0, 4.0]^\top$.

5.1 Example: Linear Regression

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [w_1, w_2, b]^\top = [1.0, -2.0, 0.5]^\top$.
- Input: $\boldsymbol{x} = [x_1, x_2]^\top = [3.0, 4.0]^\top$.

1. **Nodes 1, 2, 3**: These are input nodes.
   - Return value of node 1: $\boldsymbol{r}^{(1)} = [x_1] = [3.0]$.
   - Return value of node 2: $\boldsymbol{r}^{(2)} = [x_2] = [4.0]$.
   - Return value of node 3: $\boldsymbol{r}^{(3)} = [1.0]$ (constant bias input).

5.1 Example: Linear Regression

2. **Node 4**: This node sums its arguments. First, we compute the arguments.
    - Arg 1 (from node 1): $a_1^{(4)} = w_1 \times r_1^{(1)} = 1.0 \times 3.0 = 3.0$.
    - Arg 2 (from node 2): $a_2^{(4)} = w_2 \times r_1^{(2)} = -2.0 \times 4.0 = -8.0$.
    - Arg 3 (from node 3): $a_3^{(4)} = b \times r_1^{(3)} = 0.5 \times 1.0 = 0.5$.

5.1 Example: Linear Regression

2. **Node 4**: This node sums its arguments. First, we compute the arguments.
   - Arg 1 (from node 1): $a_1^{(4)} = w_1 \times r_1^{(1)} = 1.0 \times 3.0 = 3.0$.
   - Arg 2 (from node 2): $a_2^{(4)} = w_2 \times r_1^{(2)} = -2.0 \times 4.0 = -8.0$.
   - Arg 3 (from node 3): $a_3^{(4)} = b \times r_1^{(3)} = 0.5 \times 1.0 = 0.5$.

The return value of node 4 is the sum of these arguments:

$$\boldsymbol{r}^{(4)} = \mathrm{g}^{(4)}(3.0, -8.0, 0.5) = [3.0 - 8.0 + 0.5] = [-4.5]$$

5.1 Example: Linear Regression

2. **Node 4**: This node sums its arguments. First, we compute the arguments.
   - Arg 1 (from node 1): $a_1^{(4)} = w_1 \times r_1^{(1)} = 1.0 \times 3.0 = 3.0$.
   - Arg 2 (from node 2): $a_2^{(4)} = w_2 \times r_1^{(2)} = -2.0 \times 4.0 = -8.0$.
   - Arg 3 (from node 3): $a_3^{(4)} = b \times r_1^{(3)} = 0.5 \times 1.0 = 0.5$.

The return value of node 4 is the sum of these arguments:

$$\boldsymbol{r}^{(4)} = \mathrm{g}^{(4)}(3.0, -8.0, 0.5) = [3.0 - 8.0 + 0.5] = [-4.5]$$

Final Output: The network's output is taken from node 4, so $\boldsymbol{y} = [-4.5]$.

5.1 Example: Linear Regression

Exercise

For parameters $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$, find the output for the same input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$.

5.1 Example: Linear Regression

Exercise

For parameters $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$, find the output for the same input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$.

Solution:

- The arguments to node 4 are:
  - $a_1^{(4)} = -0.5 \times 3.0 = -1.5$.
  - $a_2^{(4)} = 3.0 \times 4.0 = 12.0$.
  - $a_3^{(4)} = -1.0 \times 1.0 = -1.0$.
- The sum is $\boldsymbol{r}^{(4)} = [-1.5 + 12.0 - 1.0] = [9.5]$.
- Final Output: $\boldsymbol{y} = [9.5]$.

5.1 Example: Logistic Regression

Consider the logistic regression model $f(\boldsymbol{x}) = \sigma(w_1 x_1 + w_2 x_2 + b)$, where σ is the sigmoid function.

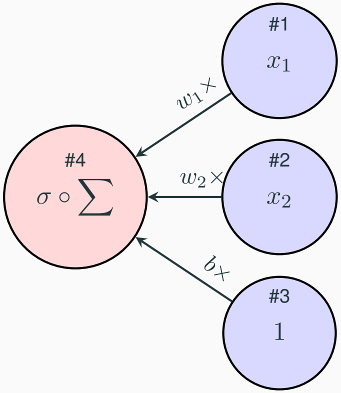


Figure 7: DAG representation of a logistic regression

Architecture: Almost the same as linear regression. The only change is the activation function of node 4.

- **Node 4 Activation:** $\mathrm{g}^{(4)}(\boldsymbol{a}) = \sigma(\sum_i a_i)$.

5.1 Example: Logistic Regression

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [1.0, -2.0, 0.5]^\top$.
- Input: $\boldsymbol{x} = [3.0, 4.0]^\top$.

5.1 Example: Logistic Regression

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [1.0, -2.0, 0.5]^\top$.
- Input: $\boldsymbol{x} = [3.0, 4.0]^\top$.

The computation of the arguments for node 4 is identical to the linear regression case.

$$\boldsymbol{a}^{(4)} = [3.0, -8.0, 0.5]^\top$$

The sum is $3.0 - 8.0 + 0.5 = -4.5$.

5.1 Example: Logistic Regression

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [1.0, -2.0, 0.5]^\top$.
- Input: $\boldsymbol{x} = [3.0, 4.0]^\top$.

The computation of the arguments for node 4 is identical to the linear regression case.

$$\boldsymbol{a}^{(4)} = [3.0, -8.0, 0.5]^\top$$

The sum is $3.0 - 8.0 + 0.5 = -4.5$.

The return value of node 4 applies the sigmoid function:

$$\boldsymbol{r}^{(4)} = \mathrm{g}^{(4)}(\boldsymbol{a}^{(4)}) = [\sigma(-4.5)]$$

$$\sigma(-4.5) = \frac{1}{1+e^{4.5}} \approx \frac{1}{1+90.017} \approx 0.01098$$

5.1 Example: Logistic Regression

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [1.0, -2.0, 0.5]^\top$.
- Input: $\boldsymbol{x} = [3.0, 4.0]^\top$.

The computation of the arguments for node 4 is identical to the linear regression case.

$$\boldsymbol{a}^{(4)} = [3.0, -8.0, 0.5]^\top$$

The sum is $3.0 - 8.0 + 0.5 = -4.5$.

The return value of node 4 applies the sigmoid function:

$$\boldsymbol{r}^{(4)} = \mathrm{g}^{(4)}(\boldsymbol{a}^{(4)}) = [\sigma(-4.5)]$$

$$\sigma(-4.5) = \frac{1}{1+e^{4.5}} \approx \frac{1}{1+90.017} \approx 0.01098$$

Final Output: $\boldsymbol{y} \approx [0.01098]$.

5.1 Example: Logistic Regression

Exercise

For parameters $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$, find the output of the logistic regression model for the input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$.

5.1 Example: Logistic Regression

Exercise

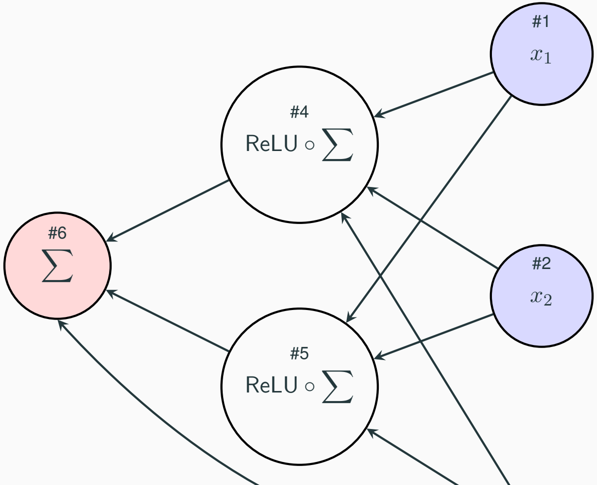
For parameters $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$, find the output of the logistic regression model for the input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$.

Solution:

- The sum of arguments for node 4 is 9.5 (from the linear regression exercise).
- The return value is $\boldsymbol{r}^{(4)} = [\sigma(9.5)]$.
- $\sigma(9.5) = \frac{1}{1+e^{-9.5}} \approx \frac{1}{1+0.0000748} \approx 0.999925$.
- Final Output: $\boldsymbol{y} \approx [0.999925]$.

5.1 Example: Multilayer Perceptron (MLP)

Consider an MLP with a 2D input, a 2D hidden layer (with ReLU), and a 1D output.



5.1 Example: Multilayer Perceptron (MLP)

Let's execute the computation.

- Checkpoint:
  $\boldsymbol{\theta} = [w_{11}, w_{12}, w_{21}, w_{22}, b_1, b_2, v_1, v_2, c]^\top = [1, 1, -1, -1, 0, 1, 2, 3, -1]^\top$.
- Input: $\boldsymbol{x} = [3.0, 4.0]^\top$.

5.1 Example: Multilayer Perceptron (MLP)

Let's execute the computation.

- Checkpoint:
  $\boldsymbol{\theta} = [w_{11}, w_{12}, w_{21}, w_{22}, b_1, b_2, v_1, v_2, c]^\top = [1, 1, -1, -1, 0, 1, 2, 3, -1]^\top$.
- Input: $\boldsymbol{x} = [3.0, 4.0]^\top$.

1. **Nodes 1, 2, 3**: Return input values $[3.0], [4.0], [1.0]$ respectively.

5.1 Example: Multilayer Perceptron (MLP)

Let's execute the computation.

- Checkpoint:
  $\boldsymbol{\theta} = [w_{11}, w_{12}, w_{21}, w_{22}, b_1, b_2, v_1, v_2, c]^\top = [1, 1, -1, -1, 0, 1, 2, 3, -1]^\top$.
- Input: $\boldsymbol{x} = [3.0, 4.0]^\top$.

1. **Nodes 1, 2, 3**: Return input values $[3.0], [4.0], [1.0]$ respectively.
2. **Node 4 (Hidden 1)**:
   - Sum of arguments: $(1 \times 3.0) + (1 \times 4.0) + (0 \times 1.0) = 7.0$.
   - Return value: $\boldsymbol{r}^{(4)} = [\text{ReLU}(7.0)] = [7.0]$.

5.1 Example: Multilayer Perceptron (MLP)

3. **Node 5 (Hidden 2)**:
    - Sum of arguments: $(-1 \times 3.0) + (-1 \times 4.0) + (1 \times 1.0) = -3.0 - 4.0 + 1.0 = -6.0$.
    - Return value: $\boldsymbol{r}^{(5)} = [\text{ReLU}(-6.0)] = [0.0]$.

5.1 Example: Multilayer Perceptron (MLP)

3. **Node 5 (Hidden 2)**:
    - Sum of arguments: $(-1 \times 3.0) + (-1 \times 4.0) + (1 \times 1.0) = -3.0 - 4.0 + 1.0 = -6.0$.
    - Return value: $\boldsymbol{r}^{(5)} = [\text{ReLU}(-6.0)] = [0.0]$.
4. **Node 6 (Output)**:
    - Arguments come from nodes 4, 5, and the bias node 3.
    - Sum of arguments:
      $(v_1 \times r_1^{(4)}) + (v_2 \times r_1^{(5)}) + (c \times r_1^{(3)}) = (2 \times 7.0) + (3 \times 0.0) + (-1 \times 1.0)$.
    - $= 14.0 + 0.0 - 1.0 = 13.0$.
    - Return value: $\boldsymbol{r}^{(6)} = [13.0]$.

5.1 Example: Multilayer Perceptron (MLP)

3. **Node 5 (Hidden 2)**:
    - Sum of arguments: $(-1 \times 3.0) + (-1 \times 4.0) + (1 \times 1.0) = -3.0 - 4.0 + 1.0 = -6.0$.
    - Return value: $\boldsymbol{r}^{(5)} = [\text{ReLU}(-6.0)] = [0.0]$.
4. **Node 6 (Output)**:
    - Arguments come from nodes 4, 5, and the bias node 3.
    - Sum of arguments: $(v_1 \times r_1^{(4)}) + (v_2 \times r_1^{(5)}) + (c \times r_1^{(3)}) = (2 \times 7.0) + (3 \times 0.0) + (-1 \times 1.0)$.
    - $= 14.0 + 0.0 - 1.0 = 13.0$.
    - Return value: $\boldsymbol{r}^{(6)} = [13.0]$.

Final Output: $\boldsymbol{y} = [13.0]$.

5.1 Example: Multilayer Perceptron (MLP)

Remark (Vector Notation for MLP)

The MLP's computation can be expressed concisely using vectors and matrices. Let the input be $\boldsymbol{x}$, hidden weights be $\boldsymbol{W}$, hidden bias be $\boldsymbol{b}$, output weights be $\boldsymbol{v}$, and output bias be c.

$$y = c + \boldsymbol{v}^{\top} \mathrm{ReLU}\left(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}\right) \tag{1}$$

Here, ReLU applied to a vector means it's applied to each element (**element-wise application**).

5.1 Example: Multilayer Perceptron (MLP)

Exercise

In the MLP example, find the output for the input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$ with parameters

$$\boldsymbol{\theta} = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \\ 1 \\ -1 \\ -2 \\ 1 \\ 0.5 \end{bmatrix}.$$

5.1 Example: Multilayer Perceptron (MLP)

Exercise

In the MLP example, find the output for the input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$ with parameters

$$\boldsymbol{\theta} = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \\ 1 \\ -1 \\ -2 \\ 1 \\ 0.5 \end{bmatrix}.$$

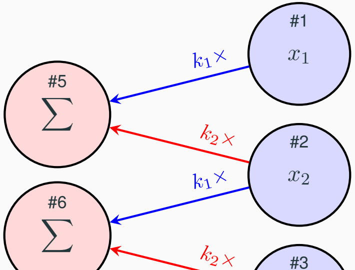
Solution:

5.1 Example: 1D Convolution

As an example of **parameter sharing**, let's consider a 1D convolutional layer with a kernel of size 2, $\boldsymbol{k} = [k_1, k_2]^\top$.

$$y_1 = k_1 x_1 + k_2 x_2$$
$$y_2 = k_1 x_2 + k_2 x_3$$
$$y_3 = k_1 x_3 + k_2 x_4$$

The same parameters k_1, k_2 are reused.



5.1 Example: 1D Convolution

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [k_1, k_2]^\top = [0.5, -1.0]^\top$.
- Input: $\boldsymbol{x} = [2.0, -1.0, 3.0, 0.0]^\top$.

5.1 Example: 1D Convolution

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [k_1, k_2]^\top = [0.5, -1.0]^\top$.
- Input: $\boldsymbol{x} = [2.0, -1.0, 3.0, 0.0]^\top$.

1. **Nodes 1-4**: Return input values $[2.0], [-1.0], [3.0], [0.0]$.
2. **Node 5 (Output 1)**:
   - Arguments: $(\theta_1 \times r_1^{(1)}), (\theta_2 \times r_1^{(2)}) = (0.5 \times 2.0), (-1.0 \times -1.0)$.
   - Sum: $1.0 + 1.0 = 2.0$. Return value: $\boldsymbol{r}^{(5)} = [2.0]$.

5.1 Example: 1D Convolution

Let's execute the computation.

- Checkpoint: $\boldsymbol{\theta} = [k_1, k_2]^\top = [0.5, -1.0]^\top$.
- Input: $\boldsymbol{x} = [2.0, -1.0, 3.0, 0.0]^\top$.

1. **Nodes 1-4**: Return input values $[2.0], [-1.0], [3.0], [0.0]$.
2. **Node 5 (Output 1)**:
    - Arguments: $(\theta_1 \times r_1^{(1)}), (\theta_2 \times r_1^{(2)}) = (0.5 \times 2.0), (-1.0 \times -1.0)$.
    - Sum: $1.0 + 1.0 = 2.0$. Return value: $\boldsymbol{r}^{(5)} = [2.0]$.
3. **Node 6 (Output 2)**:
    - Arguments: $(\theta_1 \times r_1^{(2)}), (\theta_2 \times r_1^{(3)}) = (0.5 \times -1.0), (-1.0 \times 3.0)$.
    - Sum: $-0.5 - 3.0 = -3.5$. Return value: $\boldsymbol{r}^{(6)} = [-3.5]$.

5.1 Example: 1D Convolution

4. **Node 7 (Output 3)**:
    - Arguments: $(\theta_1 \times r_1^{(3)}), (\theta_2 \times r_1^{(4)}) = (0.5 \times 3.0), (-1.0 \times 0.0)$.
    - Sum: $1.5 + 0.0 = 1.5$. Return value: $\boldsymbol{r}^{(7)} = [1.5]$.

5.1 Example: 1D Convolution

4. **Node 7 (Output 3)**:
    - Arguments: $(\theta_1 \times r_1^{(3)}), (\theta_2 \times r_1^{(4)}) = (0.5 \times 3.0), (-1.0 \times 0.0)$.
    - Sum: $1.5 + 0.0 = 1.5$. Return value: $\boldsymbol{r}^{(7)} = [1.5]$.
5. **Network Output**: The output is the concatenation of the return values from nodes 5, 6, and 7.

$$\boldsymbol{y} = \begin{bmatrix} r_1^{(5)} \\ r_1^{(6)} \\ r_1^{(7)} \end{bmatrix} = \begin{bmatrix} 2.0 \\ -3.5 \\ 1.5 \end{bmatrix}$$

5.1 Example: 1D Convolution

Remark (Definition of Convolution)

The convolution operation, denoted by ‘*‘, is more generally defined by considering **padding** (adding zeros to the input ends) and **stride** (the step size when sliding the kernel).

The output size d_{out} depends on the input size d_{in}, kernel size d_{k}, padding p, and stride s:

$$d_{\text{out}} = \lfloor \frac{d_{\text{in}} + 2p - d_{\text{k}}}{s} \rfloor + 1$$

Our example was a case of $p = 0$ (no padding) and $s = 1$ (stride 1).

5.1 Example: 1D Convolution

Exercise

In the 1D convolution example, find the output for the input $\boldsymbol{x} = \begin{bmatrix} 2.0 \\ -1.0 \\ 3.0 \\ 0.0 \end{bmatrix}$ with parameters $\boldsymbol{\theta} = [k_1, k_2]^\top = \begin{bmatrix} 1.5 \\ 1.0 \end{bmatrix}$.

5.1 Example: 1D Convolution

Exercise

In the 1D convolution example, find the output for the input $\boldsymbol{x} = \begin{bmatrix} 2.0 \\ -1.0 \\ 3.0 \\ 0.0 \end{bmatrix}$ with parameters $\boldsymbol{\theta} = [k_1, k_2]^\top = \begin{bmatrix} 1.5 \\ 1.0 \end{bmatrix}$.

Solution:

- Output 1 (Node 5): $(1.5 \times 2.0) + (1.0 \times -1.0) = 3.0 - 1.0 = 2.0$.
- Output 2 (Node 6): $(1.5 \times -1.0) + (1.0 \times 3.0) = -1.5 + 3.0 = 1.5$.
- Output 3 (Node 7): $(1.5 \times 3.0) + (1.0 \times 0.0) = 4.5 + 0.0 = 4.5$.

Final Output: $\boldsymbol{y} = [2.0, 1.5, 4.5]^\top$.

5.1 Example: A Complex Network

Our framework allows for irregular structures, multi-dimensional outputs from nodes, and multiple edges between nodes.

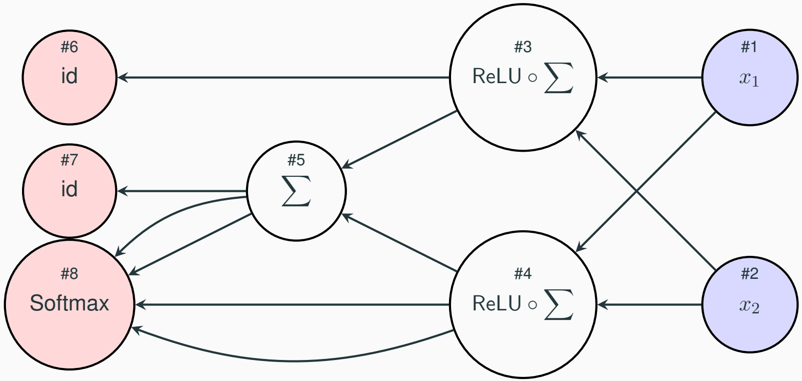


Figure 10: Example of a network with a complex structure

5.1 Example: A Complex Network

Input: $\boldsymbol{x} = [1.0, -2.0]^\top$. Weights are given.

1. **Nodes 1, 2**: $\boldsymbol{r}^{(1)} = [1.0]$, $\boldsymbol{r}^{(2)} = [-2.0]$.

5.1 Example: A Complex Network

Input: $\boldsymbol{x} = [1.0, -2.0]^\top$. Weights are given.

1. **Nodes 1, 2**: $\boldsymbol{r}^{(1)} = [1.0]$, $\boldsymbol{r}^{(2)} = [-2.0]$.
2. **Node 3**: $(\theta_{(1,3)} = 1, \theta_{(2,3)} = 1)$
    - Sum of args: $(1 \times 1.0) + (1 \times -2.0) = -1.0$.
    - Return value: $\boldsymbol{r}^{(3)} = [\text{ReLU}(-1.0)] = [0.0]$.

5.1 Example: A Complex Network

Input: $\boldsymbol{x} = [1.0, -2.0]^\top$. Weights are given.

1. **Nodes 1, 2**: $\boldsymbol{r}^{(1)} = [1.0]$, $\boldsymbol{r}^{(2)} = [-2.0]$.
2. **Node 3**: ($\theta_{(1,3)} = 1, \theta_{(2,3)} = 1$)
   - Sum of args: $(1 \times 1.0) + (1 \times -2.0) = -1.0$.
   - Return value: $\boldsymbol{r}^{(3)} = [\text{ReLU}(-1.0)] = [0.0]$.
3. **Node 4**: ($\theta_{(1,4)} = 1, \theta_{(2,4)} = -1$)
   - Sum of args: $(1 \times 1.0) + (-1 \times -2.0) = 3.0$.
   - Return value: $\boldsymbol{r}^{(4)} = [\text{ReLU}(3.0)] = [3.0]$.

5.1 Example: A Complex Network

4. **Node 5**: ($\theta_{(3,5)} = 0.5, \theta_{(4,5)} = -0.5$)
    - Sum of args: $(0.5 \times r_1^{(3)}) + (-0.5 \times r_1^{(4)}) = (0.5 \times 0.0) + (-0.5 \times 3.0) = -1.5$.
    - Return value: $\boldsymbol{r}^{(5)} = [-1.5]$.

5.1 Example: A Complex Network

4. **Node 5**: ($\theta_{(3,5)} = 0.5, \theta_{(4,5)} = -0.5$)
    - Sum of args: $(0.5 \times r_1^{(3)}) + (-0.5 \times r_1^{(4)}) = (0.5 \times 0.0) + (-0.5 \times 3.0) = -1.5$.
    - Return value: $\boldsymbol{r}^{(5)} = [-1.5]$.
5. **Node 6**: ($\theta_{(3,6)} = 1$)
    - Argument: $(1 \times r_1^{(3)}) = 0.0$. Return value: $\boldsymbol{r}^{(6)} = [0.0]$.

5.1 Example: A Complex Network

4. **Node 5**: ($\theta_{(3,5)} = 0.5, \theta_{(4,5)} = -0.5$)
   - Sum of args: $(0.5 \times r_1^{(3)}) + (-0.5 \times r_1^{(4)}) = (0.5 \times 0.0) + (-0.5 \times 3.0) = -1.5$.
   - Return value: $\boldsymbol{r}^{(5)} = [-1.5]$.
5. **Node 6**: ($\theta_{(3,6)} = 1$)
   - Argument: $(1 \times r_1^{(3)}) = 0.0$. Return value: $\boldsymbol{r}^{(6)} = [0.0]$.
6. **Node 7**: ($\theta_{(5,7)} = 2$)
   - Argument: $(2 \times r_1^{(5)}) = -3.0$. Return value: $\boldsymbol{r}^{(7)} = [-3.0]$.

5.1 Example: A Complex Network

8. **Node 8**: This node has 4 arguments coming from nodes 4 and 5.
    - $a_1^{(8)} = \theta_{(4,8),1} r_1^{(4)} = 1 \times 3.0 = 3.0$
    - $a_2^{(8)} = \theta_{(4,8),2} r_1^{(4)} = -1 \times 3.0 = -3.0$
    - $a_3^{(8)} = \theta_{(5,8),1} r_1^{(5)} = 1 \times -1.5 = -1.5$
    - $a_4^{(8)} = \theta_{(5,8),2} r_1^{(5)} = 1 \times -1.5 = -1.5$

5.1 Example: A Complex Network

8. **Node 8**: This node has 4 arguments coming from nodes 4 and 5.
   - $a_1^{(8)} = \theta_{(4,8),1} r_1^{(4)} = 1 \times 3.0 = 3.0$
   - $a_2^{(8)} = \theta_{(4,8),2} r_1^{(4)} = -1 \times 3.0 = -3.0$
   - $a_3^{(8)} = \theta_{(5,8),1} r_1^{(5)} = 1 \times -1.5 = -1.5$
   - $a_4^{(8)} = \theta_{(5,8),2} r_1^{(5)} = 1 \times -1.5 = -1.5$

   The activation function computes Softmax $\left(\left[\begin{smallmatrix} a_1+a_3 \\ a_2+a_4 \end{smallmatrix}\right]\right)$.

$$\boldsymbol{z} = \begin{bmatrix} 3.0 - 1.5 \\ -3.0 - 1.5 \end{bmatrix} = \begin{bmatrix} 1.5 \\ -4.5 \end{bmatrix}$$

$$\boldsymbol{r}^{(8)} = \text{Softmax}(\boldsymbol{z}) = \frac{1}{e^{1.5} + e^{-4.5}} \begin{bmatrix} e^{1.5} \\ e^{-4.5} \end{bmatrix} \approx \begin{bmatrix} 0.9975 \\ 0.0025 \end{bmatrix}$$

5.1 Example: A Complex Network

8. **Node 8**: This node has 4 arguments coming from nodes 4 and 5.
    - $a_1^{(8)} = \theta_{(4,8),1} r_1^{(4)} = 1 \times 3.0 = 3.0$
    - $a_2^{(8)} = \theta_{(4,8),2} r_1^{(4)} = -1 \times 3.0 = -3.0$
    - $a_3^{(8)} = \theta_{(5,8),1} r_1^{(5)} = 1 \times -1.5 = -1.5$
    - $a_4^{(8)} = \theta_{(5,8),2} r_1^{(5)} = 1 \times -1.5 = -1.5$

    The activation function computes Softmax $\left(\begin{bmatrix} a_1+a_3 \\ a_2+a_4 \end{bmatrix}\right)$.

$$\boldsymbol{z} = \begin{bmatrix} 3.0 - 1.5 \\ -3.0 - 1.5 \end{bmatrix} = \begin{bmatrix} 1.5 \\ -4.5 \end{bmatrix}$$

$$\boldsymbol{r}^{(8)} = \text{Softmax}(\boldsymbol{z}) = \frac{1}{e^{1.5} + e^{-4.5}} \begin{bmatrix} e^{1.5} \\ e^{-4.5} \end{bmatrix} \approx \begin{bmatrix} 0.9975 \\ 0.0025 \end{bmatrix}$$

Final Output: Concatenating outputs from nodes 6, 7, and 8 gives $\boldsymbol{y} \approx [0.0, -3.0, 0.9975, 0.0025]^\top$.

5.1 Example: A Complex Network

Exercise

For the same complex architecture and input $x = \begin{bmatrix} 1.0 \\ -2.0 \end{bmatrix}$, calculate the output with a different set of parameters (provided in the lecture notes).

5.1 Example: A Complex Network

Exercise

For the same complex architecture and input $\boldsymbol{x} = \begin{bmatrix} 1.0 \\ -2.0 \end{bmatrix}$, calculate the output with a different set of parameters (provided in the lecture notes).

Solution:

- Node 3 output $\boldsymbol{r}^{(3)} = [\text{ReLU}(1.5)] = [1.5]$.
- Node 4 output $\boldsymbol{r}^{(4)} = [\text{ReLU}(1.0)] = [1.0]$.
- Node 5 output $\boldsymbol{r}^{(5)} = [1.5 + 1.0] = [2.5]$.
- Node 6 output $\boldsymbol{r}^{(6)} = [-2 \times 1.5] = [-3.0]$.
- Node 7 output $\boldsymbol{r}^{(7)} = [1 \times 2.5] = [2.5]$.
- Node 8 input vector $\boldsymbol{z} = \begin{bmatrix} (0.5\times1.0)+(-1\times2.5) \\ (0.5\times1.0)+(1\times2.5) \end{bmatrix} = \begin{bmatrix} -2.0 \\ 3.0 \end{bmatrix}$.
- Node 8 output $\boldsymbol{r}^{(8)} = \text{Softmax}(\begin{bmatrix} -2.0 \\ 3.0 \end{bmatrix}) \approx [0.0067, 0.9933]^\top$.

Architecture and Checkpoints

6. Architecture and Checkpoints

Let’s summarize the key components of a neural network based on our discussion.

1. A human first designs the computational blueprint: the DAG, activation functions, and parameter assignments.
2. Then, a learning process (**training**) finds the specific numerical values for the parameters suitable for the task.

6. Architecture and Checkpoints

Let's summarize the key components of a neural network based on our discussion.

1. A human first designs the computational blueprint: the DAG, activation functions, and parameter assignments.
2. Then, a learning process (**training**) finds the specific numerical values for the parameters suitable for the task.

Definition (Architecture and Checkpoint)

- **Architecture:** The design information of a neural network, i.e., **everything other than the specific values of the parameters.** (This is what we formally defined earlier).
- **Checkpoint:** A set of **specific parameter values** (a list of numbers) that have been learned for a specific architecture.

6. Architecture and Checkpoints

Remark (The Ambiguity of the Word "Model")

The term "**model** of a neural network" can be ambiguous, so caution is required.

- In theoretical contexts or research papers, "model" often refers **only to the architecture.** (e.g., "The Transformer model").

6. Architecture and Checkpoints

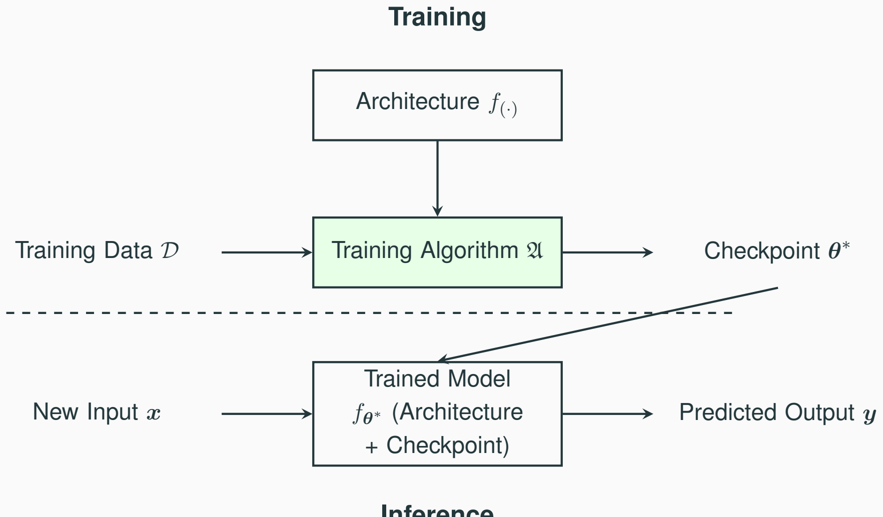
Remark (The Ambiguity of the Word "Model")

The term "**model** of a neural network" can be ambiguous, so caution is required.

- In theoretical contexts or research papers, "model" often refers **only to the architecture.** (e.g., "The Transformer model").
- In applied contexts or software libraries, "model" often refers to the **pair of an architecture and a checkpoint.** (e.g., "a pre-trained model").

6. Architecture and Checkpoints

This distinction maps directly to the training and inference scheme.


Training
Architecture $f_{(\cdot)}$
Training Data $\mathcal{D}$
Training Algorithm $\mathfrak{A}$
Checkpoint $\boldsymbol{\theta}^*$
New Input $\boldsymbol{x}$
Trained Model
$f_{\boldsymbol{\theta}^*}$ (Architecture
+ Checkpoint)
Predicted Output $\boldsymbol{y}$
Inference

Summary and Future Outlook

7.1 Today's Summary

- A neural network is a type of **parametric function** whose specific computation is determined by the values of its parameters.

7.1 Today’s Summary

- A neural network is a type of **parametric function** whose specific computation is determined by the values of its parameters.
- Its computational structure is rigorously defined by a **Directed Acyclic Graph (DAG)**.

7.1 Today's Summary

- A neural network is a type of **parametric function** whose specific computation is determined by the values of its parameters.
- Its computational structure is rigorously defined by a **Directed Acyclic Graph (DAG)**.
- The specification of a neural network is clearly separated into two components: the **architecture** (the "blueprint") and the **checkpoint** (the specific parameter values).

7.1 Today’s Summary

- A neural network is a type of **parametric function** whose specific computation is determined by the values of its parameters.
- Its computational structure is rigorously defined by a **Directed Acyclic Graph (DAG)**.
- The specification of a neural network is clearly separated into two components: the **architecture** (the "blueprint") and the **checkpoint** (the specific parameter values).
- A specific, computable "trained model" is completed only when an architecture and a checkpoint are combined.

7.2 Preview of the Next Lecture

This time, we have seen that a neural network is a parametric function composed of two elements: the architecture and the checkpoint. With an understanding of this distinction, you are ready to use neural networks as black boxes.

7.2 Preview of the Next Lecture

This time, we have seen that a neural network is a parametric function composed of two elements: the architecture and the checkpoint. With an understanding of this distinction, you are ready to use neural networks as black boxes.

Next time, we will discuss the inescapable issue of **licenses** when using publicly available neural network models.

7.2 Preview of the Next Lecture

This time, we have seen that a neural network is a parametric function composed of two elements: the architecture and the checkpoint. With an understanding of this distinction, you are ready to use neural networks as black boxes.

Next time, we will discuss the inescapable issue of **licenses** when using publicly available neural network models.

In fact, the distinction between architecture and checkpoint, and between "training" and "inference," which we learned today, is critically important for understanding the scope of license application.

Appendix: Constraints on Activation Functions i

In theory, any function can be an activation function. In practice, we choose functions with specific properties for two main reasons:

1. **Possibility of Training:** We need to use **gradient methods** for efficient training. This requires the gradient (how a small parameter change affects the output) to be well-defined.
2. **Reusability of Parameters:** Techniques like **fine-tuning** (reusing a trained checkpoint for a new task) require stability. A small change in parameters should only cause a small change in the output.

Appendix: Constraints on Activation Functions ii

Appendix: Constraints on Activation Functions iii

- It mathematically justifies the use of gradient methods.
- It ensures the entire neural network is also locally Lipschitz continuous with respect to its parameters θ, enabling stable training and fine-tuning.