Lecture 2

❖ **General (Artificial) Neural Networks**

CENG 632- Computational Intelligence, 2024-2025, Spring
Assist. Prof. Dr. Osman GÖKALP

Adapted from lecture notes of Christian Borgelt (https://computational-intelligence.eu/Lecture+Material.html)

Summary of Key Takeaways from Last Week

- A **Threshold Logic Unit** (TLU), also known as a **perceptron**, is a simple binary classifier.
- A single TLU has a limitation: it can only compute **linearly separable** functions.
- **Networks of TLUs can compute arbitrary** boolean functions, even those that are not linearly separable.
- Using the **delta rule**, we can **train a single TLU**; however, a **network of TLUs cannot** be trained using this method.
- **Online training** updates parameters (weights) after each training sample has been processed.
- **Batch learning** updates parameters after all training samples have been processed.

General (Artificial) Neural Networks

General Neural Networks

Basic graph theoretic notions

A (directed) **graph** is a pair $G = (V, E)$ consisting of a (finite) set V of **vertices** or **nodes** and a (finite) set $E \subseteq V \times V$ of **edges**.

We call an edge $e = (u, v) \in E$ **directed** from vertex u to vertex v.

Let $G = (V, E)$ be a (directed) graph and $u \in V$ a vertex.
Then the vertices of the set

$$\mathrm{pred}(u) = \{v \in V \mid (v, u) \in E\}$$

are called the **predecessors** of the vertex u
and the vertices of the set

$$\mathrm{succ}(u) = \{v \in V \mid (u, v) \in E\}$$

are called the **successors** of the vertex u.

General Neural Networks

General definition of a neural network

An (artificial) **neural network** is a (directed) graph $G = (U, C)$,
whose vertices $u \in U$ are called **neurons** or **units** and
whose edges $c \in C$ are called **connections**.

The set U of vertices is partitioned into

- the set U_{in} of **input neurons**,
- the set U_{out} of **output neurons**, and
- the set U_{hidden} of **hidden neurons**.

It is

$$U = U_{\text{in}} \cup U_{\text{out}} \cup U_{\text{hidden}},$$

$$U_{\text{in}} \neq \emptyset, \qquad U_{\text{out}} \neq \emptyset, \qquad U_{\text{hidden}} \cap (U_{\text{in}} \cup U_{\text{out}}) = \emptyset.$$

General Neural Networks

Each connection $(v, u) \in C$ possesses a **weight** w_{uv} and
each neuron $u \in U$ possesses three (real-valued) state variables:

- the **network input** net_u,
- the **activation** act_u, and
- the **output** out_u.

Each input neuron $u \in U_{\mathrm{in}}$ also possesses a fourth (real-valued) state variable,

- the **external input** ext_u.

Furthermore, each neuron $u \in U$ possesses three functions:

- the **network input function** $f_{\mathrm{net}}^{(u)} : \mathbb{R}^{2|\mathrm{pred}(u)|+\kappa_1(u)} \to \mathbb{R}$,
- the **activation function** $f_{\mathrm{act}}^{(u)} : \mathbb{R}^{\kappa_2(u)} \to \mathbb{R}$, and
- the **output function** $f_{\mathrm{out}}^{(u)} : \mathbb{R} \to \mathbb{R}$,

which are used to compute the values of the state variables.

How to interpret directions?

Connection: $\rightarrow$
Weight: $\leftarrow$

General Neural Networks

Types of (artificial) neural networks:

- If the graph of a neural network is **acyclic**, it is called a **feed-forward network**.
- If the graph of a neural network contains **cycles** (backward connections), it is called a **recurrent network**.

Representation of the connection weights as a matrix:

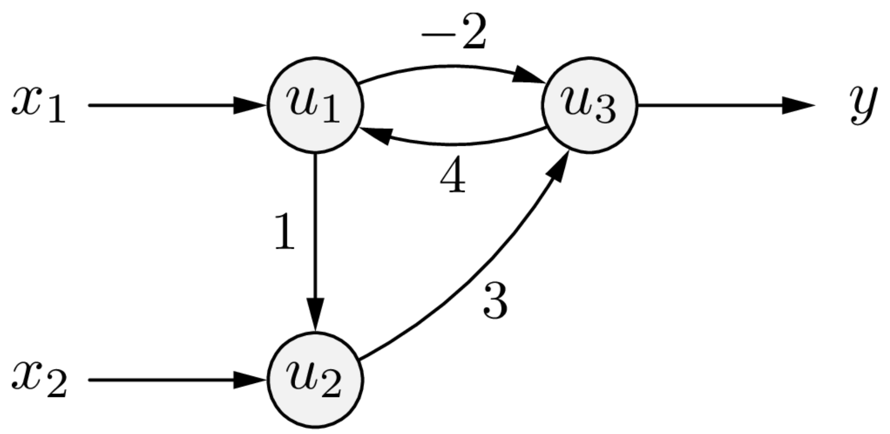
Remember that a weight $\boldsymbol{w_{uv}}$ is defined for edge $(\mathbf{v}, \mathbf{u})$.

$$\begin{array}{cccc|c} u_1 & u_2 & \dots & u_r & \\ \hline w_{u_1 u_1} & w_{u_1 u_2} & \dots & w_{u_1 u_r} & u_1 \\ w_{u_2 u_1} & w_{u_2 u_2} & & w_{u_2 u_r} & u_2 \\ \vdots & & & \vdots & \vdots \\ w_{u_r u_1} & w_{u_r u_2} & \dots & w_{u_r u_r} & u_r \end{array}$$

weights of the connections that lead to a specific neuron ($\boldsymbol{u_1}$)

$r = |U|$

A simple recurrent neural network



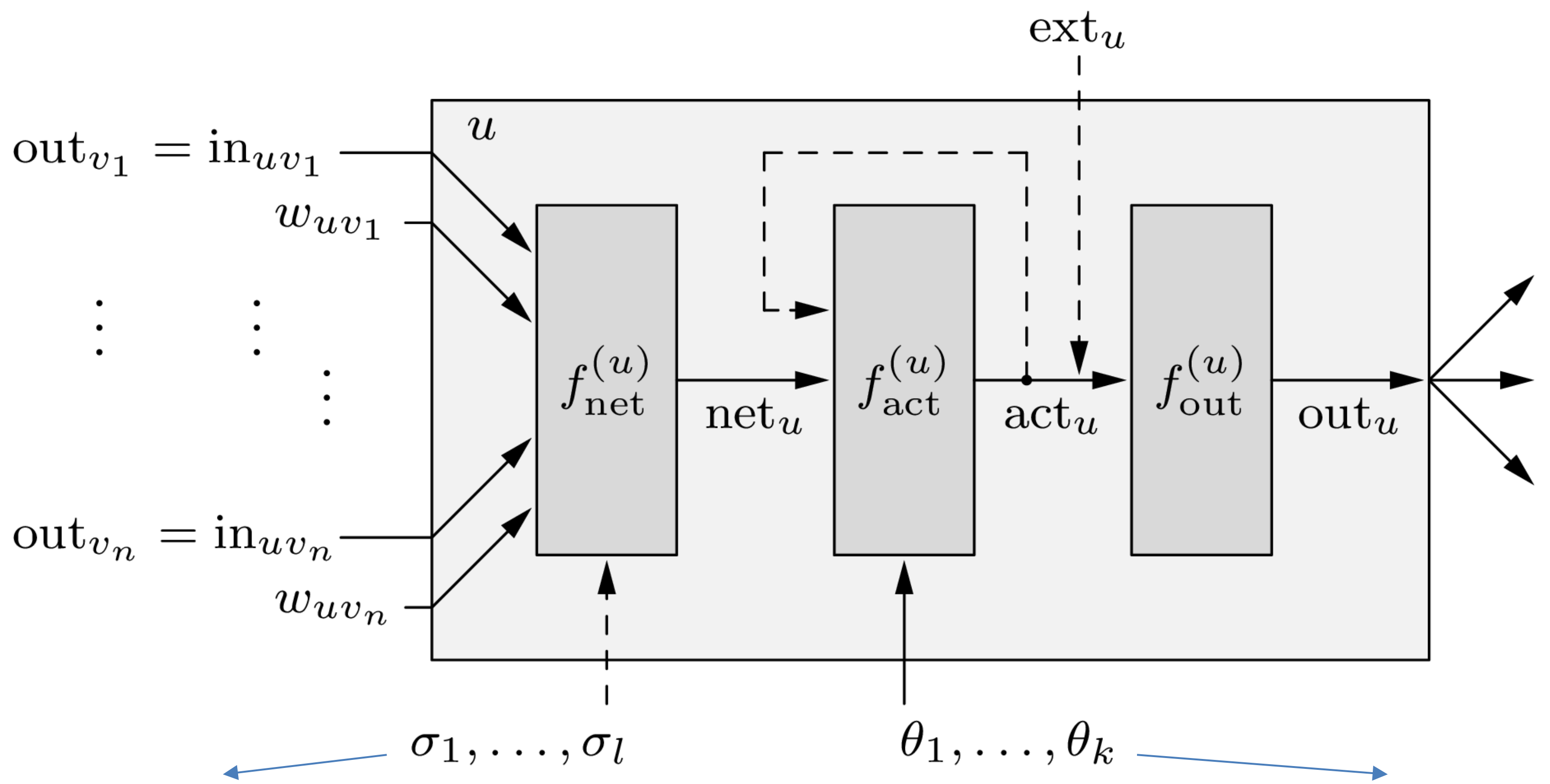
Weight matrix of this network

$$\begin{array}{c c} & \begin{array}{ccc} u_1 & u_2 & u_3 \end{array} \\ & \left(\begin{array}{ccc} 0 & 0 & 4 \\ 1 & 0 & 0 \\ -2 & 3 & 0 \end{array}\right) \begin{array}{c} u_1 \\ u_2 \\ u_3 \end{array} \end{array}$$

$$U_{in} = \{u_1, u_2\}$$
$$U_{out} = \{u_3\}$$
$$U_{hidden} = \emptyset$$

Structure of a Generalized Neuron

A generalized neuron is a simple numeric processor



These parameters are specific to a model. For example they are not used for Multi Layer Perceptrons. They are used for Radial Bais Function Networks.

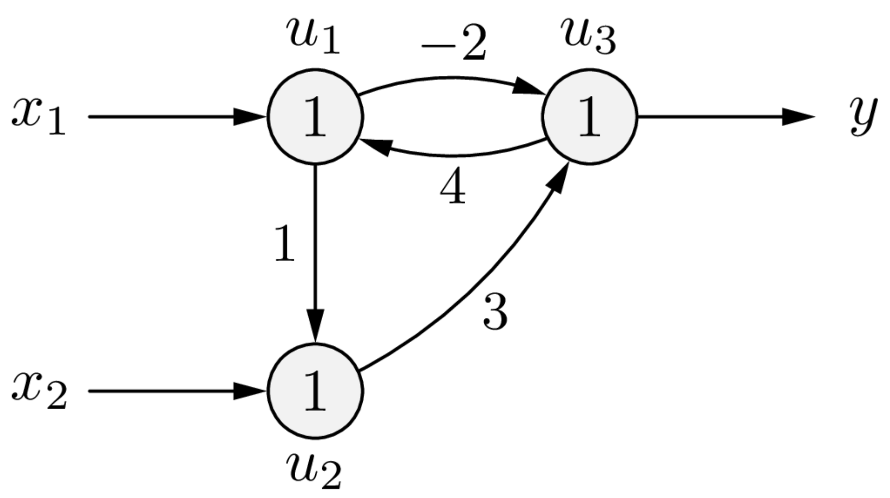
Parameters of the activation function employed.

Phases of computation

- **Input phase**: the external inputs are fed into the network
  - serves the purpose to initialize the network.
  - the activations of the **input neurons are set** to the values of the **corresponding external inputs**.
  - the activations of the remaining neurons are initialized arbitrarily, usually by simply setting them to 0. In addition, the output function is applied to the initialized activations, so that all neurons produce initial outputs

- **Work phase**: the output of the neural network is computed
  - the **external inputs are switched off** and the activations and outputs of the **neurons are recomputed (possibly multiple times)**.
  - If a neuron does not receive any network input, because it does not have any predecessors, we define that it simply maintains its activation (and thus also its output).
  - The recomputations are terminated either if the network reaches a **stable state**, that is, if further **recomputations do not change the outputs** of the neurons anymore, **or** if a **predetermined number of recomputations** have been carried out.

General Neural Networks: Example



$$f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = \textstyle\sum_{v \in \text{pred}(u)} w_{uv} \text{in}_{uv} = \sum_{v \in \text{pred}(u)} w_{uv} \, \text{out}_v$$

$$f_{\text{act}}^{(u)}(\text{net}_u, \theta) = \begin{cases} 1, & \text{if} \quad \text{net}_u \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

$$f_{\text{out}}^{(u)}(\text{act}_u) = \text{act}_u$$

General Neural Networks: Example

Updating the activations of the neurons

	u_1	u_2	u_3	
input phase	**1**	**0**	**0**	
work phase	1	0	**0**	$\text{net}_{u_3} = -2 \quad < 1$
	0	0	0	$\text{net}_{u_1} = 0 \quad < 1$
	0	**0**	0	$\text{net}_{u_2} = 0 \quad < 1$
	0	0	**0**	$\text{net}_{u_3} = 0 \quad < 1$
	0	0	0	$\text{net}_{u_1} = 0 \quad < 1$

(Note that the external input of u_1 is no longer available, but has been switched off.)

For recurrent networks: ordering is explicitly determined like this one.

- Order in which the neurons are updated:
  $u_3, u_1, u_2, u_3, u_1, u_2, u_3, \ldots$
- Input phase: activations/outputs in the initial state.
- Work phase: activations/outputs of the next neuron to update (bold) are computed from the outputs of the other neurons and the weights/threshold.
- A stable state with a unique output is reached.

However, for feed forward networks: ordering is topological. Ordering is automatically decided.

General Neural Networks: Example

Updating the activations of the neurons

	u_1	u_2	u_3	
input phase	**1**	**0**	**0**	
work phase	1	0	**0**	$\text{net}_{u_3} = -2 \quad < 1$
	1	**1**	0	$\text{net}_{u_2} = 1 \quad \geq 1$
	0	1	0	$\text{net}_{u_1} = 0 \quad < 1$
	0	1	**1**	$\text{net}_{u_3} = 3 \quad \geq 1$
	0	**0**	1	$\text{net}_{u_2} = 0 \quad < 1$
	1	0	1	$\text{net}_{u_1} = 4 \quad \geq 1$
	1	0	**0**	$\text{net}_{u_3} = -2 \quad < 1$

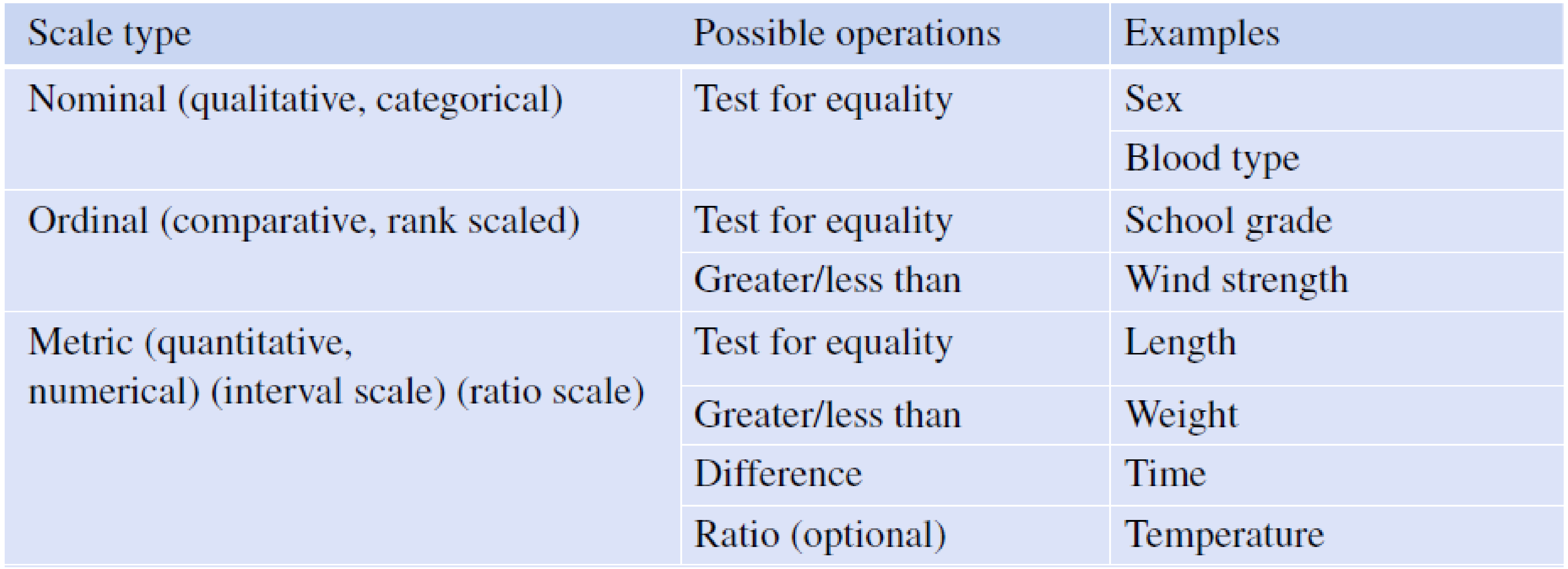
In this example we try different ordering then we tried for the previous one.

Order in which the neurons are updated:
$u_3, u_2, u_1, u_3, u_2, u_1, u_3, \ldots$

- No stable state is reached (oscillation of output).

Scale Types and Encoding

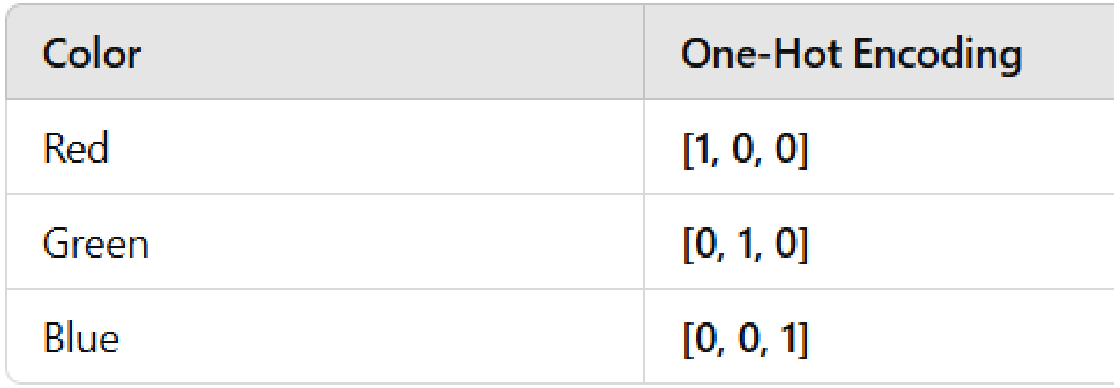
- In practice, we meet attributes having different so-called **scale types**, which may require preprocessing.



Scale type	Possible operations	Examples
Nominal (qualitative, categorical)	Test for equality	Sex
		Blood type
Ordinal (comparative, rank scaled)	Test for equality	School grade
	Greater/less than	Wind strength
Metric (quantitative, numerical) (interval scale) (ratio scale)	Test for equality	Length
	Greater/less than	Weight
	Difference	Time
	Ratio (optional)	Temperature

Encoding Nominal Attributes:1-hot encoding

- The typical approach to handle nominal attributes is so-called **1-in-n encod**ing (also called **1-hot encoding**)
- Each nominal attribute is represented by as many binary variables as it has values, each variable corresponds to one attribute value



Color	One-Hot Encoding
Red	[1, 0, 0]
Green	[0, 1, 0]
Blue	[0, 0, 1]

What About Ordinal Attributes?

- Ordinal attributes usually pose an unpleasant problem
- Of course, one may use 1-in-n encoding for them, treating them like nominal attributes, and this is actually what is most commonly done.
- However, this discards the ordinal structure of the domain of such an attribute.
- On the other hand, if we represent the values of an ordinal attribute by numbers that reflect the ordering, they will be treated as if they were metric attributes, implicitly assuming that at least differences can be computed meaningfully.
- Unfortunately, there seems to be no perfect solution for this problem. One has to choose for the application at hand which drawback is easier to tolerate

General Neural Networks: Preprocesing

- N**ormalize** the inputs of a neural network in order to **avoid** certain numerical problems, which can result from an **unequal scaling** of the different input variables.
- Most commonly, each input variable is scaled in such a way that it has arithmetic mean 0 and variance 1 (so-called **z-score normalization**).

General Neural Networks: Preprocessing

Normalization of the input vectors

- Compute expected value and (corrected) standard deviation for each input:

$$\mu_k = \frac{1}{|L|} \sum_{l \in L} \mathrm{ext}_{u_k}^{(l)} \qquad \text{and} \qquad \sigma_k = \sqrt{\frac{1}{|L| \mathbf{- 1}} \sum_{l \in L} \left(\mathrm{ext}_{u_k}^{(l)} - \mu_k \right)^2},$$

- Normalize the input vectors to expected value 0 and standard deviation 1:

$$\mathrm{ext}_{u_k}^{(l)(\mathrm{new})} = \frac{\mathrm{ext}_{u_k}^{(l)(\mathrm{old})} - \mu_k}{\sigma_k}$$

- Such a normalization avoids unit and scaling problems.
  It is also known as **z-scaling** or **z-score standardization**.

General Neural Networks: Training

Definition of learning tasks for a neural network

A **fixed learning task** L_{fixed} for a neural network with

- n input neurons $U_{\text{in}} = \{u_1, \ldots, u_n\}$ and
- m output neurons $U_{\text{out}} = \{v_1, \ldots, v_m\}$,

is a set of **training patterns** $l = (\vec{\imath}^{\,(l)}, \vec{o}^{\,(l)})$, each consisting of

- an **input vector** $\vec{\imath}^{\,(l)} = (\operatorname{ext}_{u_1}^{(l)}, \ldots, \operatorname{ext}_{u_n}^{(l)})$ and
- an **output vector** $\vec{o}^{\,(l)} = (o_{v_1}^{(l)}, \ldots, o_{v_m}^{(l)})$.

A fixed learning task is solved, if for all training patterns $l \in L_{\text{fixed}}$ the neural network computes from the external inputs contained in the input vector $\vec{\imath}^{\,(l)}$ of a training pattern l the outputs contained in the corresponding output vector $\vec{o}^{\,(l)}$.

General Neural Networks: Training

Solving a fixed learning task: Error definition

- Measure how well a neural network solves a given fixed learning task.
- Compute differences between desired and actual outputs.
- Do not sum differences directly in order to avoid errors canceling each other.
- Square has favorable properties for deriving the adaptation rules.

$$e = \sum_{l \in L_{\text{fixed}}} e^{(l)} = \sum_{v \in U_{\text{out}}} e_v = \sum_{l \in L_{\text{fixed}}} \sum_{v \in U_{\text{out}}} e_v^{(l)},$$

$$\text{where} \qquad e_v^{(l)} = \left(o_v^{(l)} - \text{out}_v^{(l)} \right)^2$$

Advantages of the «square of the deviations»

$$e_v^{(l)} = \left(o_v^{(l)} - \mathrm{out}_v^{(l)}\right)^2$$

- It is clear that we cannot simply sum the deviations directly, since then positive and **negative deviations could cancel**.
- The square of the deviation of the actual and the desired output is **continuously differentiable everywhere**, while the derivative of the absolute value does not exist/is discontinuous at 0.
- **Large deviations** from the desired output are **weighted more severely**, so that there is a tendency that during training, **individual** strong deviations (i.e., for individual training patterns) from the desired value are **avoided**.

Regression vs Classification

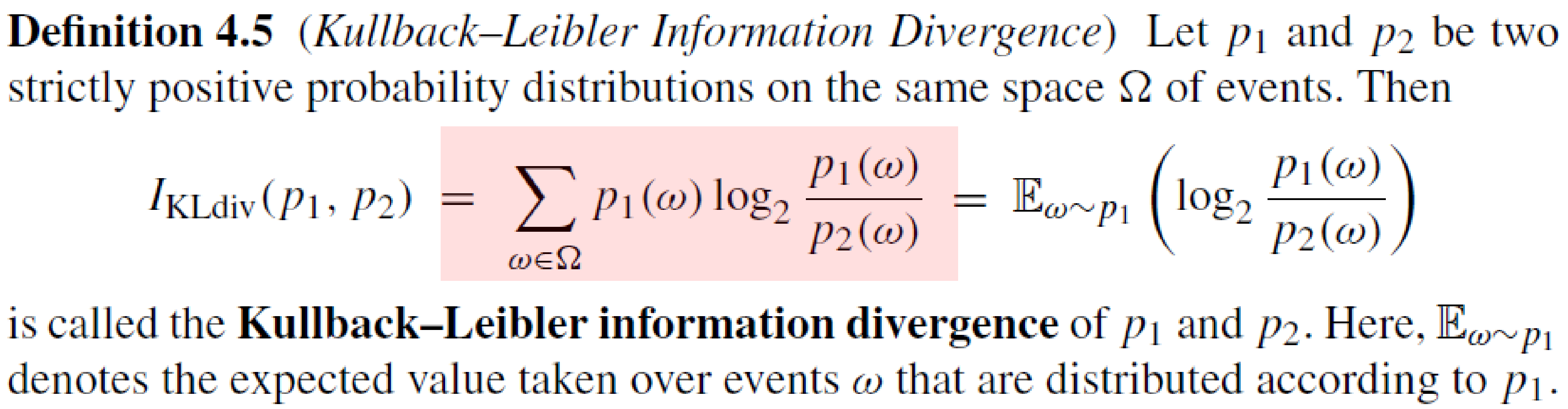
- **Regression**: the output (or target) attributes are metric.
  - The sum of squared deviations is certainly a reasonable error measure.

- **Classification**: : the output (or target) attributes are nominal.
  - We have to encode their values, for example, with 1-in-n encoding.
  - After encoding, we can still use the sum of squared deviations. **However, there are some problems with this approach:**
    - **Decode is needed:** the output of a neural network is not a clean 1-in-n encoding.
    - Solution: **softmax function (can handle any list (including negative values))**

$$y'_i = \frac{e^{y_i}}{\sum_{k=1}^{n} e^{y_k}} \longrightarrow \text{Probability distribution over n values.}$$

Measuring error with Sofmax function

- Output of a neural network with a nominal target can be **interpreted as a probability distribution** over classes.
- So we can **compare** the output (**softmax**) probability distribution with the **true target values**. One can use Kullback-Leiber (KL) Divergence to measure the error.



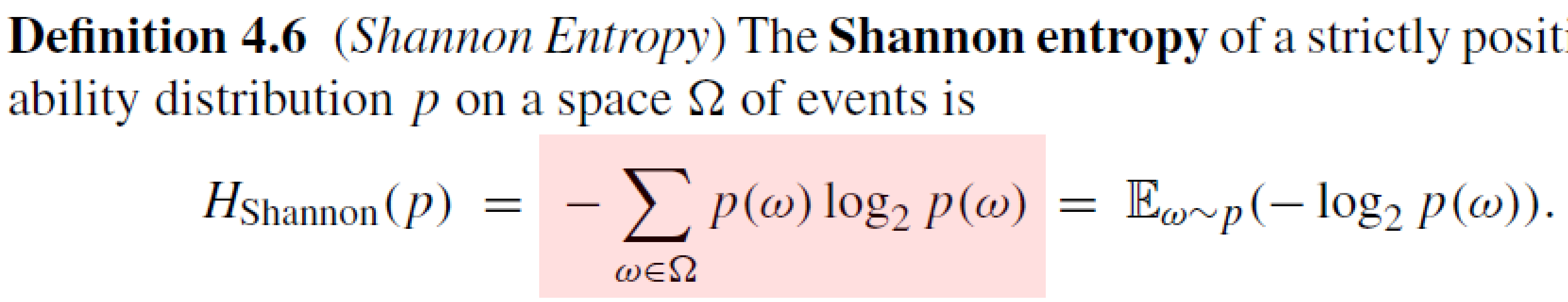
Definition 4.5 (*Kullback–Leibler Information Divergence*) Let p_1 and p_2 be two strictly positive probability distributions on the same space Ω of events. Then

$$I_{\mathrm{KLdiv}}(p_1, p_2) = \sum_{\omega \in \Omega} p_1(\omega) \log_2 \frac{p_1(\omega)}{p_2(\omega)} = \mathbb{E}_{\omega \sim p_1} \left(\log_2 \frac{p_1(\omega)}{p_2(\omega)} \right)$$

is called the **Kullback–Leibler information divergence** of p_1 and p_2. Here, $\mathbb{E}_{\omega \sim p_1}$ denotes the expected value taken over events ω that are distributed according to p_1.

Cross Entropy for Measuring Error - I

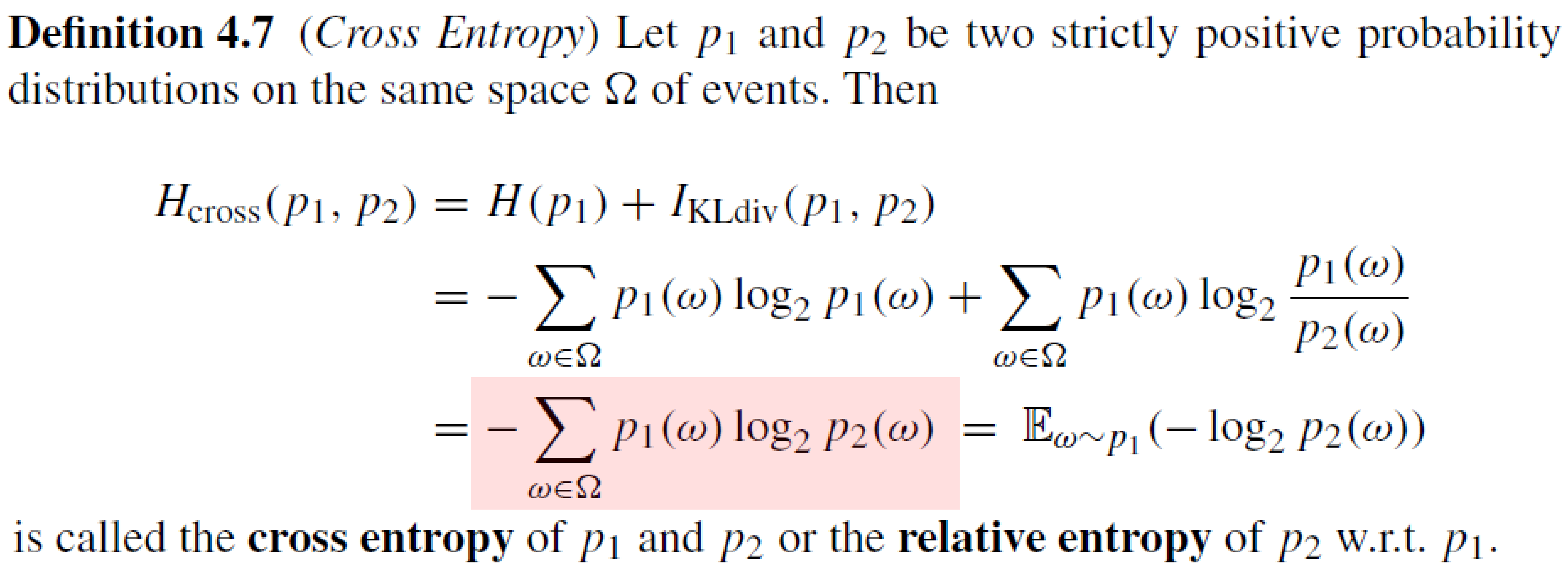
- Although KL divergence measures the error, **cross-entropy combines** KL divergence with **Shannon** entropy to quantify the number of extra bits needed to encode data when using the predicted distribution instead of the true one, making it more suitable for optimization in machine learning.



Definition 4.6 (*Shannon Entropy*) The **Shannon entropy** of a strictly positive probability distribution p on a space Ω of events is

$$H_{\text{Shannon}}(p) = -\sum_{\omega \in \Omega} p(\omega) \log_2 p(\omega) = \mathbb{E}_{\omega \sim p}(-\log_2 p(\omega)).$$

Cross Entropy for Measuring Error - II



Definition 4.7 (*Cross Entropy*) Let p_1 and p_2 be two strictly positive probability distributions on the same space Ω of events. Then

$$\begin{aligned} H_{\text{cross}}(p_1, p_2) &= H(p_1) + I_{\text{KLdiv}}(p_1, p_2) \\ &= -\sum_{\omega\in\Omega} p_1(\omega)\log_2 p_1(\omega) + \sum_{\omega\in\Omega} p_1(\omega)\log_2 \frac{p_1(\omega)}{p_2(\omega)} \\ &= -\sum_{\omega\in\Omega} p_1(\omega)\log_2 p_2(\omega) = \mathbb{E}_{\omega\sim p_1}(-\log_2 p_2(\omega)) \end{aligned}$$

is called the **cross entropy** of p_1 and p_2 or the **relative entropy** of p_2 w.r.t. p_1.

General Neural Networks: Training

Definition of learning tasks for a neural network

A **free learning task** L_{free} for a neural network with

- n input neurons $U_{\text{in}} = \{u_1, \ldots, u_n\}$,

is a set of **training patterns** $l = (\vec{\imath}^{\,(l)})$, each consisting of

- an **input vector** $\vec{\imath}^{\,(l)} = (\,\text{ext}_{u_1}^{(l)}, \ldots, \text{ext}_{u_n}^{(l)}\,)$.

Properties:

- There is no desired output for the training patterns.
- Outputs can be chosen freely by the training method.
- Solution idea: **Similar inputs should lead to similar outputs.**
  (clustering of input vectors)

Conclusion - I

- An artificial neural network can be represented by a **directed graph**.
- Neurons can be classified into ‘**input**’, ‘**hidden**’, and ‘**output**’.
- If there is **no directed cycles**, it is called **feed forward network**.
- If graph contains loops or **directed cycles**, it is called **recurrent network.**
- **Calculation** of a neural networks has two phases: **input phase** and **work phase**.
- **Feedforward networks** use **topological ordering** in their calculations, as information flows in one direction through the layers, **whereas** the final **output may depend on the order** in which the neurons recompute their outputs for **recurrent networks**.

Conclusion - II

- Attributes may have **different scale types**: **nominal**, **ordinal**, **metric**.
- Nominal and ordinal types can be handled using **1-hot encoding**.
- Attributes may have **unequal scaling**. One possible solution is **z-score normalization**.
- **Squared deviation** error can be used in **regression** tasks.
- However, **classification** task needs special error calculation: **cross entropy**.