Institute of Technological Studies of Bizerte

AY: 2025-2026 | **M1-S1: Dept. of Electrical Engineering**
Exam | **AI-ECUE122** | **Teacher: A. Mhamdi**
Jan. 2026 | **Time Limit: 1½ h**

This document contains **9** pages numbered from **1** to **9**. Upon receiving it, verify completeness. The **3** tasks are independent and can be solved in any order you prefer. The following rules apply:

❶ **A handwritten double-sided A4** sheet is permitted.

❷ **Any electronic material**, except basic calculator, is prohibited.

❸ **Mysterious or unsupported answers** will not receive full credit.

❹ **Round results** to the nearest thousandth *(i.e., the third digit after the decimal point)*.

❺ **Task №3:** Correct answers earn points as indicated. There is no negative scoring.

Task №1

⌛ 20mn | (4 points)

You have a fuzzy logic system that controls motor speed Ω based on two inputs:

Position error (ε): How far the motor is from target position;

Velocity error ($\delta\varepsilon$): How fast the motor is currently moving.

Position error (ε) membership functions:

$$\mu_{\text{small}}(\varepsilon) = 1 - \frac{\varepsilon}{5} \quad \text{for } 0 \leq \varepsilon \leq 5;\ \text{otherwise } 0$$

$$\mu_{\text{large}}(\varepsilon) = \frac{\varepsilon - 2}{8} \quad \text{for } \varepsilon \geq 2;\ \text{otherwise } 0$$

Velocity error ($\delta\varepsilon$) membership functions:

$$\mu_{\text{slow}}(\delta\varepsilon) = 1 - \frac{\delta\varepsilon}{3} \quad \text{for } 0 \leq \delta\varepsilon \leq 3;\ \text{otherwise } 0$$

$$\mu_{\text{fast}}(\delta\varepsilon) = \frac{\delta\varepsilon}{4} \quad \text{for } \delta\varepsilon \geq 0;\ \text{otherwise } 0$$

You have three rules:

$\Re_1$: If (ε is Small) and ($\delta\varepsilon$ is Slow) then $\Omega = 20$ RPM

$\Re_2$: If (ε is Small) and ($\delta\varepsilon$ is Fast) then $\Omega = 50$ RPM

$\Re_3$: If (ε is Large) and ($\delta\varepsilon$ is Slow) then $\Omega = 80$ RPM

Compute the output of a **Tsukamoto** inference system if the inputs are $\varepsilon = 3$ and $\delta\varepsilon = 1$.

Fuzzification *(Determine membership values)* 0.25×4

For $\varepsilon = 3$ and $\delta\varepsilon = 1$:

$$\mu_{\text{small}}(3) = 1 - \frac{3}{5} = \mathbf{0.4} \qquad \mu_{\text{slow}}(1) = 1 - \frac{1}{3} = \mathbf{0.667} \quad (\text{or } \tfrac{2}{3})$$

$$\mu_{\text{large}}(3) = \frac{3-2}{8} = \mathbf{0.125} \qquad \mu_{\text{fast}}(1) = \frac{1}{4} = \mathbf{0.25}$$

Apply Fuzzy Rules 0.5×3

Let's calculate the activation strength *(minimum of input memberships)* for each rule:

$$\begin{array}{llll}
\Re_1 & \Longrightarrow & \alpha_1 = \min(0.4, 0.667) = \mathbf{0.4} & \text{and } \Omega_1 = 20 \\
\Re_2 & \Longrightarrow & \alpha_2 = \min(0.4, 0.25) = \mathbf{0.25} & \text{and } \Omega_2 = 50 \\
\Re_3 & \Longrightarrow & \alpha_3 = \min(0.125, 0.667) = \mathbf{0.125} & \text{and } \Omega_3 = 80
\end{array}$$

Final output *(weighted average)* 1.5

$$\begin{aligned}
\Omega^* &= \frac{(\alpha_1 \times \Omega_1) + (\alpha_2 \times \Omega_2) + (\alpha_3 \times \Omega_3)}{\alpha_1 + \alpha_2 + \alpha_3} \\
&= \frac{(0.4 \times 20) + (0.25 \times 50) + (0.125 \times 80)}{0.4 + 0.25 + 0.125} \\
&\approx \mathbf{39.355} \text{ RPM}
\end{aligned}$$

The motor speed should be set to approximately **39.355 RPM**.

Task N°2

⌛ 50mn | (10 points)

In this exercise, you will manually compute how gradient information flows backward through a neural network during the backpropagation algorithm.

```julia
using Flux
model = Chain(
    Dense(3, 4, relu),
    Dense(4, 1, sigmoid)
)
loss(m, x, y) = Flux.binarycrossentropy(m(x), y)
# Training Data
x = [1.0; 0.5; 2.0]          # Single input with 3 features
y = [1.0]                    # Binary target (class 1)
```

The weights and biases are initialized as follows:

$$\mathrm{W}^{[1]} = \begin{pmatrix} 0.5 & -0.3 & 0.2 \\ -0.4 & 0.6 & -0.1 \\ 0.3 & -0.2 & 0.4 \\ -0.1 & 0.5 & -0.3 \end{pmatrix}, \quad \mathbf{b}^{[1]} = \begin{pmatrix} 0.1 \\ -0.05 \\ 0.02 \\ -0.08 \end{pmatrix}$$

$$\mathbf{w}^{[2]} = \begin{pmatrix} -0.2 & 0.4 & -0.3 & 0.5 \end{pmatrix}, \quad \mathrm{b}^{[2]} = 0.1$$

(a) (1 point) Determine the structure of the neural network, including the number of layers, dimensions, and activation functions.

Input layer: 3 inputs

Hidden layer: 4 neurons with `ReLU` activation

Output layer: 1 neuron with `sigmoid` activation

(b) (1 point) Name the loss function and explain why it is appropriate for this problem.

As indicated in the code, the loss function is the **Binary Cross-Entropy** (**BCE**) function

$$\mathcal{L} = -\mathrm{y}\log(\hat{\mathrm{y}}) - (1-\mathrm{y})\log(1-\hat{\mathrm{y}})$$

BCE is appropriate for binary classification problems where the target is binary (0 or 1) and the output is a probability.

(c) (2 points) Using the given input $\mathbf{x} = [1.0, 0.5, 2.0]^\mathsf{T}$ and the target $\mathrm{y} = 1.0$, compute all intermediate activations through the network.

Forward pass:

$$\begin{aligned} \mathbf{z}^{[1]} &= \mathrm{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}, \qquad \mathrm{W}^{[1]} \in \mathbb{R}^{4\times 3},\ \mathbf{b}^{[1]} \in \mathbb{R}^4 \\ \mathbf{a}^{[1]} &= \mathrm{ReLU}\left(\mathbf{z}^{[1]}\right) \\ \mathrm{z}^{[2]} &= \mathbf{w}^{[2]}\mathbf{a}^{[1]} + \mathrm{b}^{[2]}, \qquad \mathbf{w}^{[2]} \in \mathbb{R}^{1\times 4},\ \mathrm{b}^{[2]} \in \mathbb{R} \\ \hat{\mathrm{y}} = \mathrm{a}^{[2]} &= \sigma\left(\mathrm{z}^{[2]}\right) = \frac{1}{1+e^{-\mathrm{z}^{[2]}}} \end{aligned}$$

Compute $\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$

$$\mathbf{z}^{[1]} = \begin{pmatrix} 0.5 & -0.3 & 0.2 \\ -0.4 & 0.6 & -0.1 \\ 0.3 & -0.2 & 0.4 \\ -0.1 & 0.5 & -0.3 \end{pmatrix} \begin{pmatrix} 1.0 \\ 0.5 \\ 2.0 \end{pmatrix} + \begin{pmatrix} 0.1 \\ -0.05 \\ 0.02 \\ -0.08 \end{pmatrix}$$
$$= \begin{pmatrix} 0.85 \\ -0.35 \\ 1.02 \\ -0.53 \end{pmatrix}$$

Apply ReLU $\Longrightarrow \mathbf{a}^{[1]} = \mathrm{ReLU}\left(\mathbf{z}^{[1]}\right)$

$$\mathbf{a}^{[1]} = \begin{pmatrix} \max(0, 0.85) \\ \max(0, -0.35) \\ \max(0, 1.02) \\ \max(0, -0.53) \end{pmatrix} = \begin{pmatrix} 0.85 \\ 0 \\ 1.02 \\ 0 \end{pmatrix}$$

Compute $z^{[2]} = \mathbf{w}^{[2]}\mathbf{a}^{[1]} + b^{[2]}$ and $\hat{y} = \sigma\left(z^{[2]}\right)$

$$z^{[2]} = \begin{pmatrix} -0.2 & 0.4 & -0.3 & 0.5 \end{pmatrix} \begin{pmatrix} 0.85 \\ 0 \\ 1.02 \\ 0 \end{pmatrix} + 0.1$$
$$= -0.376$$

The final output $\hat{y}$ is:

$$\hat{y} = \sigma\left(z^{[2]}\right) = \frac{1}{1 + e^{-(-0.376)}} \approx 0.407$$

(d) (4 points) Compute the gradients $\frac{\partial \mathcal{L}}{\partial W}$ and $\frac{\partial \mathcal{L}}{\partial b}$ for all weight matrices and bias vectors using the backpropagation algorithm.

$\nabla_{a^{[2]}}\mathcal{L} = \nabla_{\hat{y}}\mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{1}{\hat{y}} \implies \nabla_{a^{[2]}}\mathcal{L} = -2.457$$

$\nabla_{z^{[2]}}\mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z^{[2]}} \implies \frac{\partial \mathcal{L}}{\partial z^{[2]}} = \hat{y} - 1 = -0.593$$

$\nabla_{w^{[2]}}\mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} \cdot \underbrace{\frac{\partial z^{[2]}}{\partial w^{[2]}}}_{a^{[1]}}$$

$$\implies \frac{\partial \mathcal{L}}{\partial w^{[2]}} = \begin{pmatrix} -0.504 & 0 & -0.605 & 0 \end{pmatrix}^T$$

$\nabla_{b^{[2]}}\mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} \implies \frac{\partial \mathcal{L}}{\partial b^{[2]}} = -0.593$$

$\nabla_{a^{[1]}}\mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial a^{[1]}} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} w^{[2]}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a^{[1]}} = \begin{pmatrix} 0.119 & -0.237 & 0.178 & -0.296 \end{pmatrix}^T$$

$\nabla_{z^{[1]}}\mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[1]}} \odot \mathbb{1}_{z^{[1]}>0}$$

where $\odot$ denotes element-wise multiplication. The ReLU derivative is:

$$\frac{\partial \text{ReLU}}{\partial z_i^{[1]}} = \begin{cases} 1 & \text{if } z_i^{[1]} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\implies \frac{\partial \mathcal{L}}{\partial z^{[1]}} = \begin{pmatrix} 0.119 & 0 & 0.178 & 0 \end{pmatrix}^T$$

$\nabla_{\mathbf{W}^{[1]}}\mathcal{L}$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \cdot \underbrace{\frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}}_{\mathbf{x}^{\mathsf{T}}} \\
&= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \cdot \mathbf{x}^{\mathsf{T}} \\
\implies \quad \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} &= \begin{pmatrix} 0.119 \\ 0 \\ 0.178 \\ 0 \end{pmatrix} \begin{pmatrix} 1.0 & 0.5 & 2.0 \end{pmatrix} \\
&= \begin{pmatrix} 0.119 & 0.0595 & 0.238 \\ 0 & 0 & 0 \\ 0.178 & 0.089 & 0.356 \\ 0 & 0 & 0 \end{pmatrix}
\end{aligned}
$$

$\nabla_{\mathbf{b}^{[1]}}\mathcal{L}$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1]}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \\
\implies \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1]}} &= \begin{pmatrix} 0.119 \\ 0 \\ 0.178 \\ 0 \end{pmatrix}
\end{aligned}
$$

Verifying with `Flux`'s automatic differentiation:

```julia
# Fix model's weights and biases
model.layers[1].weight .= [0.5 -0.3 0.2; -0.4 0.6 -0.1; 0.3 -0.2
↪ 0.4; -0.1 0.5 -0.3]
# model.layers[1].bias .= ...
loss(m, x, y) = Flux.binarycrossentropy(m(x), y)
grads, = Flux.gradient( m -> loss(m, x, y), model)
grads.layers[1]
# (weight = Float32[0.11858157 0.059290785 0.23716314; -0.0 -0.0
↪ -0.0; 0.177787236 0.08893618 0.35574472; -0.0 -0.0 -0.0], bias
↪ = Float32[0.11858157, 0.0, 0.17787236, 0.0], = nothing)
```

```julia
grads.layers[2]
# (weight = Float32[-0.5039717 -0.0 -0.604766 -0.0], bias =
↪  Float32[-0.59290785],  = nothing)
```

(e) (2 points) Using a learning rate $\eta = 0.1$, compute the updated value of the synaptic weight that connects the second input to the third neuron in the hidden layer[1].

The weight connecting the second input to the third hidden neuron is $W^{[1]}_{3,2}$.

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}_{3,2}} = \frac{\partial \mathcal{L}}{\partial z^{[1]}_{3}} \cdot x_2$$

Weight update with learning rate $\eta = 0.1$

$$W^{[1]}_{3,2} \leftarrow W^{[1]}_{3,2} - 0.1 \cdot \frac{\partial \mathcal{L}}{\partial W^{[1]}_{3,2}}$$

The weight connecting the second input to the third hidden neuron is $W^{[1]}_{3,2} = -0.2$ *(from row 3, column 2 of the weight matrix* $W^{[1]}$*)*. From the gradient matrix computed previously:

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}_{3,2}} = 0.089$$

Apply gradient descent update

$$W^{[1]}_{3,2} \leftarrow W^{[1]}_{3,2} - \eta \cdot \frac{\partial \mathcal{L}}{\partial W^{[1]}_{3,2}}$$

$$W^{[1]}_{3,2} = -0.2 - 0.1 \times 0.089 = -0.2 - 0.0089 = -0.2089$$

[1] Use the standard gradient descent update rule: $w \leftarrow w - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$.

AY: 2025-2026	Full Name: ....
M1-S1: Dept. of Electrical Engineering	ID: ....
EXAM | AI-ECUE122	Class: RAIA1 ....
Jan. 2026	Room: ....
Teacher: A. Mhamdi	Time Limit: $1\frac{1}{2}$ h

ANSWER SHEET

Task N°3 — 20mn | (6 points)

(a) (½ point) Which operator is used for exponentiation in Julia?
○ `**` √ `^` ○ `pow` ○ `exp`

(b) (½ point) How do you create a tuple in Julia?
√ `(1, 2, 3)` ○ `[1, 2, 3]` ○ `{1, 2, 3}` ○ `tuple(1, 2, 3)`

(c) (½ point) Which syntax is used for string interpolation in Julia?
○ `"Hello " + name` √ `"Hello $(name)"` ○ `f"Hello {name}"`

(d) (½ point) What is the default file extension for Julia source files?
○ `.julia` ○ `.jul` √ `.jl` ○ `.j`

(e) (½ point) Which Julia construct is used to handle exceptions and errors?
- ○ `if-else` statements for error checking
- ○ `@assert` macros only
- √ `try-catch-finally` blocks to catch exceptions and handle errors gracefully
- ○ Type annotations to prevent all errors

(f) (½ point) What is the correct syntax for creating a 1D array in Julia?
- ○ `arr = array(1, 2, 3)`
- √ `arr = [1, 2, 3]`
- ○ `arr = {1, 2, 3}`
- ○ `arr = (1, 2, 3)`

(g) (½ point) If you have a fuzzy set with membership function $\mu(x)$, how would you typically evaluate the membership degree at a specific input value x_0?
- ○ By integrating the membership function over a range
- ○ By computing the centroid of the fuzzy set
- √ By calling the membership function directly: $\mu(x_0)$

○ By finding the maximum value of the membership function

(h) (½ point) What is the purpose of defuzzification in a fuzzy control system?

○ To increase the granularity of fuzzy membership functions

√ To convert fuzzy output sets into a single crisp control action

○ To combine multiple fuzzy rules into one rule

○ To measure uncertainty in the fuzzy system

(i) (½ point) Which method computes the center of mass of the output fuzzy set?

√ Centroid (Center of Gravity)

○ Maximum membership

○ First of maxima

○ Mean of maxima

(j) (½ point) When implementing a **Mamdani** fuzzy inference system in `Julia`, what is the role of the aggregation step?

○ To scale membership values by the confidence of each rule

○ To convert crisp inputs into fuzzy values

○ To eliminate rules that have low activation levels

√ To combine the outputs of all activated fuzzy rules into a single output fuzzy set

(k) (½ point) Which loss function is commonly used for classification in `Flux.jl`?

○ `mse` √ `crossentropy` ○ `mae` ○ `huber_loss`

(l) (½ point) How do you calculate gradients in `Flux.jl`?

○ `Fuzzy.gradient(loss, params)`

√ `Flux.gradient(loss, params)`

○ `∇(() -> loss(), params)`

○ `Flux.∇(() -> loss(), params)`