Deep Learning

Problem Set 1

Instructor: Dr. Lynn Kaack

Submitted by:

Xiaohan Wu - 237867
Farhan Yusuf Shaikh - 249899
Fanus Ghorjani - 248835

Date: October 3, 2025

October 3, 2025

1 Theoretical Part

1.1 Optimization

Function

$$f(x, y, z) = x^2 + y^2 + z^2 - xyz$$

The function f takes three real numbers as input: x, y, and z. These represent a vector

$$\mathbf{v} = (x, y, z) \in \mathbb{R}^3$$

from three-dimensional space. The output of the function is a real number:

$$f : \mathbb{R}^3 \to \mathbb{R}.$$

In other words, f is a function that maps three numbers to a single number. One can think of it as a mini-model of the type of optimization that occurs in deep learning with many parameters.

1. Step: Gradient

The gradient of a function $f(x, y, z)$ is the vector of its first partial derivatives. For

$$f(x, y, z) = x^2 + y^2 + z^2 - xyz,$$

we obtain

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right).$$

Computing the partial derivatives gives

$$\nabla f = (2x - yz,\ 2y - xz,\ 2z - xy).$$

The gradient points in the direction of the steepest ascent of the function. To determine the critical points, we set $\nabla f = 0$, which yields

$$2x - yz = 0, \qquad 2y - xz = 0, \qquad 2z - xy = 0.$$

2. Step: Critical points (Results from $\nabla f = 0$)

From the condition $\nabla f = 0$ we get the system of equations

$$\begin{cases} 2x - yz = 0 & \Rightarrow \quad 2x = yz, \\ 2y - xz = 0 & \Rightarrow \quad 2y = xz, \\ 2z - xy = 0 & \Rightarrow \quad 2z = xy. \end{cases}$$

This system tells us how the variables x, y, z must relate to each other at critical points. In other words, at these points the slope of f in every direction is zero, so the function is "flat" there. Such points are candidates for minima, maxima, or saddle points.

Solutions:

- The *trivial solution*: $(0, 0, 0)$. Here all variables are zero, so the equations are satisfied directly.
- *Symmetric solutions:* If all variables are equal, $x = y = z$, then the equations simplify to $2x = x \cdot x$. This gives $x = 0$ or $x = 2$, so we get the points $(0, 0, 0)$ (already known) and $(2, 2, 2)$.
- *Other solutions:* If two variables are the same and the third is the negative of them, for example $x = y = -z$, we also get valid solutions. One example is $(-2, -2, 2)$. By permuting the roles of x, y, z we get the points $(-2, 2, -2)$ and $(2, -2, -2)$.

The critical points are: (0,0,0), (2,2,2), (-2,-2,2), (-2,2,-2), (2,-2,-2).

These are all possible points where the gradient vanishes (=0). In the next step, we will use the Hessian matrix to classify them as minima, maxima, or saddle points.

3. Step: Hessian Matrix

The Hessian matrix collects all second derivatives of $f(x, y, z)$:

$$H = \begin{bmatrix} 2 & -z & -y \\ -z & 2 & -x \\ -y & -x & 2 \end{bmatrix}.$$

Hessian at Critical Points

Point $(0,0,0)$

$$H = 2I_3$$

All eigenvalues are $2 > 0 \;\Rightarrow\;$ **positive definite** $\;\Rightarrow\;$ **local minimum**.

Point $(2,2,2)$

$$H = \begin{bmatrix} 2 & -2 & -2 \\ -2 & 2 & -2 \\ -2 & -2 & 2 \end{bmatrix}$$

Eigenvalues: $\{-2,4,4\} \;\Rightarrow\;$ **indefinite** $\;\Rightarrow\;$ **saddle point**.

Point $(-2,-2,2)$ **(and permutations)**

$$H = \begin{bmatrix} 2 & -2 & 2 \\ -2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

Mixed eigenvalues $\;\Rightarrow\;$ **indefinite** $\;\Rightarrow\;$ **saddle point**.

Point $(-2,-2,2)$ (and permutations)

$$H = \begin{bmatrix} 2 & -2 & 2 \\ -2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}.$$

To classify this point we compute the eigenvalues explicitly. The characteristic polynomial is

$$\det(H-\lambda I) = (2-\lambda)\big((2-\lambda)^2-4\big)-(-2)(-2(2-\lambda)-4)+2((-2)(2)-(2-\lambda)(-2)),$$

which simplifies to

$$-\lambda^3 + 6\lambda^2 - 4\lambda - 16.$$

Solving this cubic gives the eigenvalues

$$\lambda_1 = -2, \quad \lambda_2 = 2, \quad \lambda_3 = 6.$$

Thus we have mixed signs, so the Hessian is *indefinite*, and the point $(-2,-2,2)$ (as well as its permutations) is a **saddle point**.

Why do we look at eigenvalues?

Eigenvalues are numbers that describe how a matrix stretches or compresses space in certain directions. In this context, they tell us about the *curvature* of f at a critical point: If all eigenvalues are positive, the surface is curved upwards in all directions, so the point is a *local minimum.*

If all eigenvalues are negative, the surface is curved downwards in all directions, so the point is a *local maximum.* If there is a mix of positive and negative eigenvalues, the surface goes up in some directions and down in others. The point is a *saddle point.*
The eigenvalues are the tool we use to classify each critical point as a minimum, maximum, or saddle.

1.2 Activation functions

ReLU Explanation

The Rectified Linear Unit (ReLU) is one of the most widely used activation functions in deep learning. It is defined piecewise as

$$\mathrm{ReLU}(s) = \max(0, s) = \begin{cases} 0, & s < 0, \\ s, & s \geq 0. \end{cases}$$

It outputs zero for negative inputs and keeps positive inputs unchanged. This makes the function simple and efficient, while introducing non-linearity into the network. It looks like a line with a "kink" at the origin: flat for negative values and linear for positive values.

We consider the function with a fixed input $x \in \mathbb{R}$:

$$f(b, w) = \mathrm{ReLU}(b + xw).$$

1. Step: Case differentiation for s = b+xw

Since ReLU is defined piecewise, we must separate two cases depending on the sign of s = b+xw

- **Case A:** $s < 0$
  In this case, $f(b, w) = 0$, which is constant. so the derivatives are
  $$\frac{\partial f}{\partial b} = 0, \qquad \frac{\partial f}{\partial w} = 0.$$
- **Case B:** $s \geq 0$
  Here, $f(b, w) = b + xw$. The derivatives are
  $$\frac{\partial f}{\partial b} = 1, \qquad \frac{\partial f}{\partial w} = x.$$

We can see that the gradient depends on the sign of $b + xw$ because the ReLU function has a "kink" at zero.

Boundary case $s = 0$
At $s = 0$, the ReLU function is not differentiable. We use a subgradient which can take any value between 0 and 1. Deep learning libraries usually set the derivative to 0 at this point.

Summary

The gradient of $f(b, w) = \text{ReLU}(b + xw)$ with respect to b and w is

$$\nabla f(b, w) = \begin{cases} (0,\, 0), & b + xw < 0, \\ (1,\, x), & b + xw > 0, \\ \text{not differentiable (subgradient)}, & b + xw = 0. \end{cases}$$

1.3 Overfitting

Overfitting describes the situation where a model memorizes the training data too closely, including random noise of the data, instead of capturing the underlying structure. This leads to a very low training error, while the test error remains high or even increases. In the cases considered here, overfitting appears in two dimensions: with respect to the training set size and the model complexity.

In this assignment, we assume a continuous input domain with a smooth distribution, so that no training or test cases are ever exactly duplicated. Models that exhibit overfitting in this setting have therefore learned patterns that do not generalize.

1.3.1 Error Rate vs. Training Set Size

We begin with the first graph, which illustrates the typical behavior of the training error rate as a function of the training set size. The graph shows two curves: the training error (in orange) and the test error (in green) as functions of the size of the training dataset. The y-axis shows the error rate with 0 marked at the bottom, error increasing upwards. The x-axis increases to the right, indicating larger training set size.

For very small datasets, the model can almost perfectly memorize the few available examples. This results in a very low training error, while the test error remains high, since the model does not generalize. It shows a typical case of overfitting: as the training set size increases, the task of perfectly fitting every data point becomes more difficult. The training error therefore increases slightly. At the same time, the test error decreases, since the model is able to capture more representative patterns of the underlying distribution and generalizes better.

Both curves converge towards the *Bayes Error*: the irreducible error that even a perfect model cannot go below due to inherent noise in the data. The two curves never intersect; instead, both converge towards the Bayes error as the training set size grows. The difference between the test and training error represents the

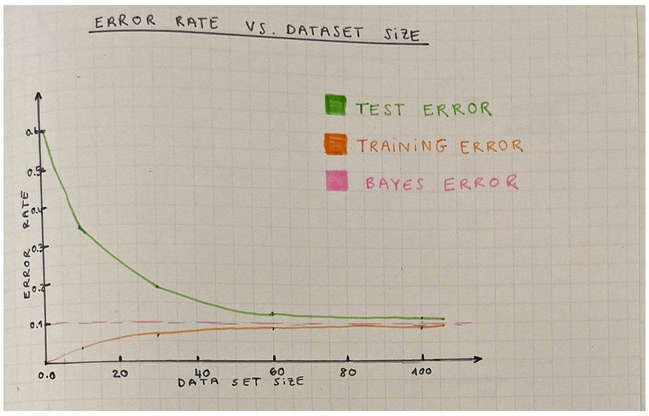

Figure 1: Error Rate vs. Training Set Size

generalization gap. This gap is large for small training sets (strong overfitting) and decreases as the dataset size grows.

1.3.2 Error Rate vs. Model Complexity

The second graph shows the relationship between error rate and model complexity for a fixed training set size. Two curves are shown: the training error (orange) and the test error (green). Again, the y-axis shows the error rate with 0 marked at the bottom, error increasing upwards and the x-axis increases to the right, indicating larger model complexity.

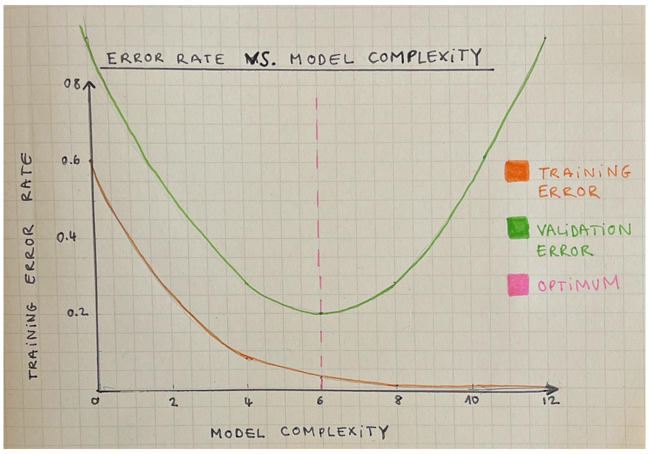

Figure 2: Error Rate vs. Model Complexity

For simple models, both the training and test errors are high, so the model has

no capacity to understand the underlying patterns, which correspond to the underfitting phase.
As the model complexity increases, the training error decreases monotonically. The test error also decreases, since the model becomes more expressive and can better approximate the true distribution. After a certain point, the test error reaches its minimum. This is the phase of optimal complexity, where the model balances bias and variance.

Beyond this point, the training error continues to decrease towards zero, but the test error starts to increase again. The model becomes overly complex and begins to memorize noise in the training data rather than generalizing well. This is the overfitting phase.

After a certain point, the test error reaches its minimum. This point corresponds to the *optimal complexity*, where the model achieves the best trade-off between bias and variance. At this complexity, the model is expressive enough to capture the true structure in the data without yet overfitting to noise or specific details in the training set.

To the right of this line, the model is too complex for the data available.
This graph visualizes the bias-variance tradeoff: *underfitting* on the left, a region of good generalization in the middle, and *overfitting* on the right. The optimal model complexity lies at the minimum of the test error curve and is shown by a pink dotted line.

1.3.3 Error Rate vs. Training Epochs

The third graph illustrates the error rate as a function of the number of training epochs for a neural network of fixed complexity. Two curves are shown: the training error and the validation (or test) error.
The training error decreases monotonically with the number of epochs, as the network continuously adapts its weights to better fit the training data. With enough epochs, the training error can become arbitrarily small.

The validation error initially decreases as well, since the model is learning meaningful structure from the data. However, after a certain number of epochs, the validation error reaches a minimum and then begins to increase again. This increase indicates that the model starts to overfit: it memorizes details and noise from the training set that do not generalize.

A vertical dashed line in the graph marks the phase where the validation error is minimal. At this point, the model achieves its best generalization performance. Early stopping is a form of regularization where training is halted at this phase instead of continuing until the training error converges. It effectively limits the model's capacity to overfit by preventing it from fully minimizing the training error at the expense of the validation error. This makes it a practical and widely

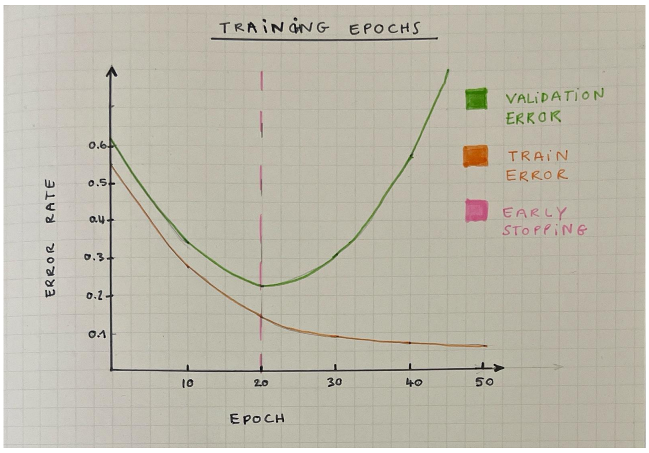

Figure 3: Error Rate vs. Training Epochs

used regularization method in neural networks.
This graph demonstrates that while the training error decreases continuously with epochs, the validation error has a U-shape, and early stopping chooses the point of minimum validation error as the optimal stopping point.

1.4 Capacity of a neural network

We need a single hidden layer network with exactly two hidden units that realizes a *logistic* model whose logit is

$$\beta_0 + \beta_1 x_1 x_2,$$

i.e.

$$\hat{y} = \sigma\big(\beta_0 + \beta_1 x_1 x_2\big), \qquad \sigma(z) = \frac{1}{1+e^{-z}}.$$

1. Logistic model" models the probability of a bianry outcome which means the final output must be a sigmoid of a linear predictor that maps any real number into [0,1]. Therefore, the network's pre-activation at the output must equal $\beta_0 + \beta_1 x_1 x_2$. The only nontrivial part is manufacturing the interaction $x_1 x_2$ from (x_1, x_2) using one hidden layer.

2. **Reduce to squared sums/differences.** Use the identity

$$x_1 x_2 = \frac{(x_1+x_2)^2 - (x_1-x_2)^2}{4}.$$

So it suffices to make the two hidden units output $(x_1+x_2)^2$ and $(x_1-x_2)^2$; then a suitable linear combination at the output will recover $x_1 x_2$ exactly.

3. **Choose hidden activation and weights to realize these squares.** Let the hidden activation be $g(z) = z^2$. For input $x = (x_1, x_2)^\top$, choose

$$W^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

   Then the hidden pre-activations are

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = W^{[1]}x + b^{[1]} = \begin{bmatrix} x_1 + x_2 \\ x_1 - x_2 \end{bmatrix},$$

   and applying g gives the hidden outputs

$$h_1 = g(a_1) = (x_1 + x_2)^2, \qquad h_2 = g(a_2) = (x_1 - x_2)^2.$$

4. **Pick output weights/bias to match the logit.** Using the identity,

$$x_1 x_2 = \frac{h_1 - h_2}{4}.$$

   Therefore choose the output pre-activation

$$a_{\text{out}} = \beta_0 + \frac{\beta_1}{4} h_1 - \frac{\beta_1}{4} h_2 = \beta_0 + \beta_1 x_1 x_2,$$

   which fixes

$$W^{[2]} = \begin{bmatrix} \frac{\beta_1}{4} & -\frac{\beta_1}{4} \end{bmatrix}, \qquad b^{[2]} = \beta_0.$$

5. **Produce the logistic output and verify.** Apply the sigmoid at the output:

$$\hat{y} = \sigma(a_{\text{out}}) = \sigma\big(\beta_0 + \beta_1 x_1 x_2\big),$$

   which is exactly the required logistic model.

Compact matrix summary (dimensions):

$$W^{[1]} \in \mathbb{R}^{2\times 2},\ b^{[1]} \in \mathbb{R}^2, \qquad W^{[2]} \in \mathbb{R}^{1\times 2},\ b^{[2]} \in \mathbb{R}.$$

1.5 Neural network theory (backprop for $w_{ij}^{[1]}$)

Setup (one training example). Input $x \in \mathbb{R}^d$ (column), one–hot label $y \in \mathbb{R}^K$. Two hidden layers with ReLU, output layer with softmax. Let $H^{[1]}$ and $H^{[2]}$ be the numbers of hidden units.

$$\begin{aligned}
a_i^{[1]} &= b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]}\, x_j, \qquad h_i^{[1]} = \sigma\big(a_i^{[1]}\big) \quad (\sigma = \text{ReLU}),\\
a_m^{[2]} &= b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]}\, h_i^{[1]}, \quad h_m^{[2]} = \sigma\big(a_m^{[2]}\big),\\
a_k^{[3]} &= b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]}\, h_m^{[2]}, \quad f_k = \text{softmax}_k(a^{[3]}) = \frac{e^{a_k^{[3]}}}{\sum_{\ell=1}^{K} e^{a_\ell^{[3]}}}.
\end{aligned}$$

Squared-error (quadratic) loss:

$$L \;=\; \sum_{k=1}^{K} \big(f_k - y_k\big)^2.$$

Goal. Derive the gradient-descent update for the first-layer weights $w_{ij}^{[1]}$.

1. Gradient-descent update (iterate $r \to r{+}1$)

$$\boxed{w_{ij}^{[1]}(r{+}1) \;=\; w_{ij}^{[1]}(r) \;-\; \eta\,\frac{\partial L}{\partial w_{ij}^{[1]}}}$$

for learning rate $\eta > 0$. It remains to compute $\dfrac{\partial L}{\partial w_{ij}^{[1]}}$.

2. Chain rule factorization

Backpropagate from the loss to the parameter:

$$\frac{\partial L}{\partial w_{ij}^{[1]}} = \underbrace{\frac{\partial L}{\partial a^{[3]}}}_{\delta^{[3]}}\,\frac{\partial a^{[3]}}{\partial h^{[2]}}\,\underbrace{\frac{\partial h^{[2]}}{\partial a^{[2]}}}_{\sigma'(a^{[2]})}\,\frac{\partial a^{[2]}}{\partial h^{[1]}}\,\underbrace{\frac{\partial h^{[1]}}{\partial a^{[1]}}}_{\sigma'(a^{[1]})}\,\frac{\partial a^{[1]}}{\partial w_{ij}^{[1]}}.$$

We compute each factor explicitly.

3. Output layer delta $\delta^{[3]} = \partial L/\partial a^{[3]}$ (softmax + SSE)

First, $\dfrac{\partial L}{\partial f} = 2(f - y)$. The softmax derivative is

$$\frac{\partial f_m}{\partial a_k^{[3]}} \;=\; f_m(\delta_{mk} - f_k).$$

Therefore, by the chain rule

$$\boxed{\delta_k^{[3]} = \frac{\partial L}{\partial a_k^{[3]}} = \sum_{m=1}^{K} 2(f_m - y_m)\, f_m\,(\delta_{mk} - f_k).}$$

4. Backprop to layer 2 (linear map then ReLU)

Remember the forward path $a^{[3]} = W^{[3]}h^{[2]} + b^{[3]}$
The linear map gives $(\partial a^{[3]}/\partial h^{[2]}) = (W^{[3]})$, so

$$\tilde{\delta}^{[2]} \;=\; (W^{[3]})^{\top}\delta^{[3]}.$$

ReLU derivative: $\sigma'(a) = \mathbf{1}\{a > 0\}$ (choose 0 at $a = 0$). Hence

$$\boxed{\delta^{[2]} \;=\; \tilde{\delta}^{[2]} \odot \mathbf{1}\{a^{[2]} > 0\} \;=\; \big((W^{[3]})^{\top}\delta^{[3]}\big) \odot \mathbf{1}\{a^{[2]} > 0\}.}$$

5. Backprop to layer 1 (linear map then ReLU)

Similarly,

$$\tilde{\delta}^{[1]} \;=\; (W^{[2]})^\top \delta^{[2]}, \qquad \boxed{\delta^{[1]} \;=\; \tilde{\delta}^{[1]} \odot \mathbf{1}\{a^{[1]} > 0\} \;=\; \big((W^{[2]})^\top \delta^{[2]}\big) \odot \mathbf{1}\{a^{[1]} > 0\}.}$$

6. Gradient w.r.t. first-layer weights

Since $a_i^{[1]} = b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j$, we have

$$\frac{\partial a_i^{[1]}}{\partial w_{ij}^{[1]}} = x_j.$$

Therefore,

$$\boxed{\frac{\partial L}{\partial w_{ij}^{[1]}} \;=\; \delta_i^{[1]}\, x_j.}$$

In matrix form (for one example):

$$\boxed{\nabla_{W^{[1]}} L \;=\; \delta^{[1]}\, x^\top, \qquad \nabla_{b^{[1]}} L \;=\; \delta^{[1]}.}$$

7. Final update

Combining Step 1 and Step 6,

$$\boxed{W^{[1]} \;\leftarrow\; W^{[1]} - \eta\,(\delta^{[1]} x^\top), \qquad b^{[1]} \;\leftarrow\; b^{[1]} - \eta\,\delta^{[1]}.}$$

Mini-batch (size B). Average the per-example gradients:

$$W^{[1]} \leftarrow W^{[1]} - \eta\,\frac{1}{B}\sum_{t=1}^{B} \delta^{[1](t)} (x^{(t)})^\top, \qquad b^{[1]} \leftarrow b^{[1]} - \eta\,\frac{1}{B}\sum_{t=1}^{B} \delta^{[1](t)}.$$

Shapes (sanity check). $W^{[1]} \in \mathbb{R}^{H^{[1]}\times d}$, $b^{[1]} \in \mathbb{R}^{H^{[1]}}$, $W^{[2]} \in \mathbb{R}^{H^{[2]}\times H^{[1]}}$, $W^{[3]} \in \mathbb{R}^{K\times H^{[2]}}$, $x \in \mathbb{R}^d$, $\delta^{[1]} \in \mathbb{R}^{H^{[1]}}$, $\delta^{[2]} \in \mathbb{R}^{H^{[2]}}$, $\delta^{[3]} \in \mathbb{R}^{K}$. All products above are dimensionally consistent.

References and Acknowledgements

In this assignment we used the help of following course materials and external resources:

- Course lecture notes and session slides 1-3 (Deep Learning, Fall 2025).
- Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning.* MIT Press.

- Wikipedia articles on Hessian matrix, eigenvalues, ReLU, and overfitting.
- Holbrook, R. (Kaggle). *Overfitting and Underfitting.*
  https://www.kaggle.com/code/ryanholbrook/overfitting-and-underfitting
- Vitalflux Blog. *Overfitting vs Underfitting Concepts & Interview Questions.*
  https://vitalflux.com/overfitting-underfitting-concepts-interview-questions/
- StatQuest with Josh Starmer (YouTube playlists).
  https://www.youtube.com/@statquest/playlists

Use of Generative AI. OpenAI's `ChatGPT (GPT-5)` was used to support this submission. Specifically, it was applied for:

- Translation of text passages from German into English,
- Formatting content in LaTeX (equations, tables, captions),
- Providing additional explanatory to clarify mathematical questions.
- Fixing and improving coding syntax and style in Part 2 (practical coding tasks),
- Suggesting best practices for structuring and commenting the Python code.

All mathematical derivations and final results were checked and validated between the group members, and the coding solutions were executed and verified independently to ensure correctness.