Deep Learning PS1 Group A – Theorical Part

Giulia Maria Petrilli (236888)
Nicolas Reichardt (245611)
Elena Murray (239793)

September 2025

1 Theoretical Part

1.1 Optimization

Let $f : R^3 \to R$ be defined by $f(x, y, z) = x^2 + y^2 + z^2 - xyz$. Identify and classify all critical points of f. You may use tools like Wolfram Alpha to help you with eigenvalues.

Step 1: Compute the gradient

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

1) $\frac{\partial f}{\partial x} = x^2 + y^2 + z^2 - xyz$
$= 2x - yz$

2) $\frac{\partial f}{\partial y} = x^2 + y^2 + z^2 - xyz$
$= 2y - xz$

3) $\frac{\partial f}{\partial z} = x^2 + y^2 + z^2 - xyz$
$= 2z - xy$

Step 2: Set the gradient to 0 and rearrange to isolate each variable

1) $2x - yz = 0$
$x = \frac{1}{2}yz$

2) $2y - xz = 0$
$y = \frac{1}{2}xz$

3) $2z - xy = 0$
$z = \frac{1}{2}xy$

Step 3: Solve

If x = 0, then in equation 2):

$$\begin{aligned} 2y - xz &= 0 \\ 2y - (0)z &= 0 \\ 2y &= 0 \\ y &= 0 \end{aligned}$$

If x = 0 and y = 0, in equation 3):

$$\begin{aligned} 2z - xy &= 0 \\ 2z &= 0 \\ z &= 0 \end{aligned}$$

Check with equation 1) if x,y,z = 0:

$$\begin{aligned} 2x - yz &= 0 \\ 0 &= 0 \end{aligned}$$

First critical point: (0,0,0).

Assume $x, y, z \neq 0$:

$$\begin{aligned} \text{From equation (1):} \quad & x = \tfrac{1}{2}yz \\ \text{Plug this into (2):} \quad & 2y - \left(\tfrac{1}{2}yz\right) z = 0 \\ & 2y - \tfrac{1}{2}yz^2 = 0 \\ & 2y = \tfrac{1}{2}yz^2 \\ & 4y = yz^2 \\ & y = \tfrac{1}{4}yz^2 \\ & y - \tfrac{1}{4}yz^2 = 0 \\ & y\left(1 - \tfrac{1}{4}z^2\right) = 0 \\ \text{Case } y \neq 0: \quad & 1 - \tfrac{1}{4}z^2 = 0 \\ & 1 = \tfrac{1}{4}z^2 \\ & 4 = z^2 \\ & z = \pm\sqrt{4} \\ & z = \pm 2 \end{aligned}$$

If z = 2, then for equation 1):

$$x = \frac{1}{2}yz$$
$$x = \frac{1}{2}y \cdot 2$$
$$x = y$$

For equation 3:

$$2z - xy = 0$$
$$2z = xy$$
$$2 \cdot 2 = xy$$
$$4 = xy$$

We know that y = x

$$4 = x \cdot x$$
$$4 = x^2$$
$$x = \pm\sqrt{4}$$
$$x = \pm 2$$

Recover y, as y = x:
If x = 2, then y = 2
If x = -2, then y = -2

So when z = 2, the solutions are (2,2,2) and (-2, -2, 2).

Check with equation (2):

$$2y - xz = 0$$
$$2 \cdot 2 - 2 \cdot 2 = 0 \quad \text{(satisfied)}$$

$$2y - xz = 0$$
$$2 \cdot (-2) - (-2) \cdot 2 = 0 \quad \text{(satisfied)}$$

If z = -2, then for equation 1):

$$x = \frac{1}{2}yz$$
$$x = \frac{1}{2}y \cdot -2$$
$$x = -y$$

For equation 3):

$$2z - xy = 0$$
$$2z = xy$$
$$2 \cdot -2 = xy$$

We know that x = -y, or y = -x

$$-4 = x \cdot -x$$
$$-4 = -x^2$$
$$x^2 = 4$$
$$x = \pm\sqrt{4}$$
$$x = \pm 2$$

Recover y, as y = -x:
If x = 2, then y = -2
If x = -2, then y = 2

So when z = -2, the solutions are (2,-2,-2) and (-2, 2, -2).

Check with equation (2):

$$2y - xz = 0$$
$$2 \cdot (-2) - 2 \cdot (-2) = 0 \quad \text{(satisfied)}$$

$$2y - xz = 0$$
$$2 \cdot 2 - (-2) \cdot (-2) = 0 \quad \text{(satisfied)}$$

So all critical points are: (0,0,0), (-2,-2,2), (2,2,2), (2, -2, -2), (-2,2,-2).

Step 4: Take second derivatives to build the Hessian:

$$f_{xx} = \frac{\partial}{\partial x}(2x - yz) = 2$$

$$f_{yy} = \frac{\partial}{\partial y}(2y - xz) = 2$$

$$f_{zz} = \frac{\partial}{\partial z}(2z - xy) = 2$$

$$f_{xy} = \frac{\partial}{\partial y}(2x - yz) = -z$$

$$f_{yx} = \frac{\partial}{\partial x}(2y - xz) = -z$$

$$f_{xz} = \frac{\partial}{\partial z}(2x - yz) = -y$$

$$f_{zx} = \frac{\partial}{\partial x}(2z - xy) = -y$$

$$f_{yz} = \frac{\partial}{\partial z}(2y - xz) = -x$$

$$f_{zy} = \frac{\partial}{\partial y}(2z - xy) = -x$$

The Hessian matrix is therefore:

$$H = \begin{bmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{bmatrix} = \begin{bmatrix} 2 & -z & -y \\ -z & 2 & -x \\ -y & -x & 2 \end{bmatrix}.$$

Step 5: Evalute the Hessian

$$H(0,0,0) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

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

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

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

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

Step 6: Classify
Using Wolfram Alpha to find the eigenvalues:

H(0,0,0)= $\lambda = 2, \quad \lambda = 2, \quad \lambda = 2$
All positive values: minimum.

H(-2,-2,2)= $\lambda = 4, \quad \lambda = 4, \quad \lambda = -2$
Mixed values: saddle.

H(2,2,2)= $\lambda = 4, \quad \lambda = 4, \quad \lambda = -2$
Mixed values: saddle.

H(2,-2,-2)= $\lambda = 4, \quad \lambda = 4, \quad \lambda = -2$
Mixed values: saddle.

H(-2,2,-2)= $\lambda = 4, \quad \lambda = 4, \quad \lambda = -2$
Mixed values: saddle.

1.2 Activation functions

Compute the gradient of the function

$$f(b,w) = ReLU(b+xw) = \begin{cases} 0 & \text{if } b+xw < 0 \\ b+xw & \text{if } b+xw \geq 0 \end{cases} \tag{1}$$

with respect to the parameters b and w, with $x, b, w \in R$.

The Relu activation function and its derivatives:

$$g(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases}$$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \end{cases}$$

and undefined in $z = 0$.

Take the derivatives of the inner function $b + xw$:

If $b + xw < 0$, then $f(b, w) = 0$:

$$\frac{\partial}{\partial b}(0) = 0, \quad \frac{\partial}{\partial w}(0) = 0.$$

If $b + xw \geq 0$, then $f(b, w) = b + xw$:

$$\frac{\partial}{\partial b}(b + xw) = 1, \quad \frac{\partial}{\partial w}(b + xw) = x$$

So the gradient is:

$$\nabla f(b, w) = \begin{cases} (0, 0) & \text{if } b + xw < 0 \\ (1, x) & \text{if } b + xw > 0 \\ \text{undefined} & \text{if } b + xw = 0 \end{cases}$$

At the point $b + xw = 0$, the left-hand and right-hand derivatives are different, so the derivative is not defined.

1.3 Overfitting

1.3.1 This graph shows the behavior of the training error and the testing error as the size of the training set increases. The y-axis shows the error rate (increasing error upwards) and the x-axis displays the training set size (increasing to the right). As the training set size increases, the training error will increase (since the model has more data to fit and cannot perfectly memorize it). At the same time, the testing error will decrease (since the model generalizes better with more data). Both will then eventually converge toward a similar value as the dataset becomes sufficiently large. The training error will however stay lower than the testing error.

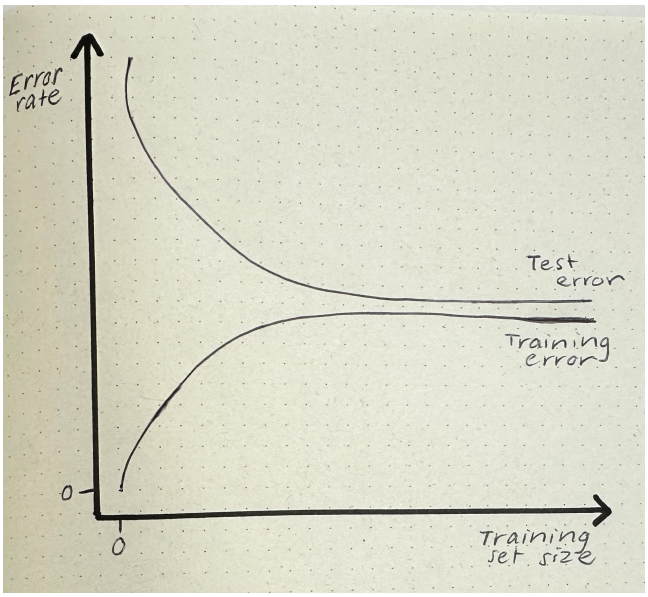

Figure 1: Error rate versus dataset size

1.3.2 The following graph illustrates the relationship between model complexity and error rate. When the model is at low complexity, the error rate is high, because the model is too simple to capture the underlying patterns in the data, leading to underfitting. As the model complexity progresses, the model achieves the best balance between bias and variance. At this sweet spot (drawn in figure 2 as a vertical dashed line), the generalization is at its prime. After that, the test error starts to increase again, because the model memorizes the training data too well, including noise. This results in high variance and overfitting.

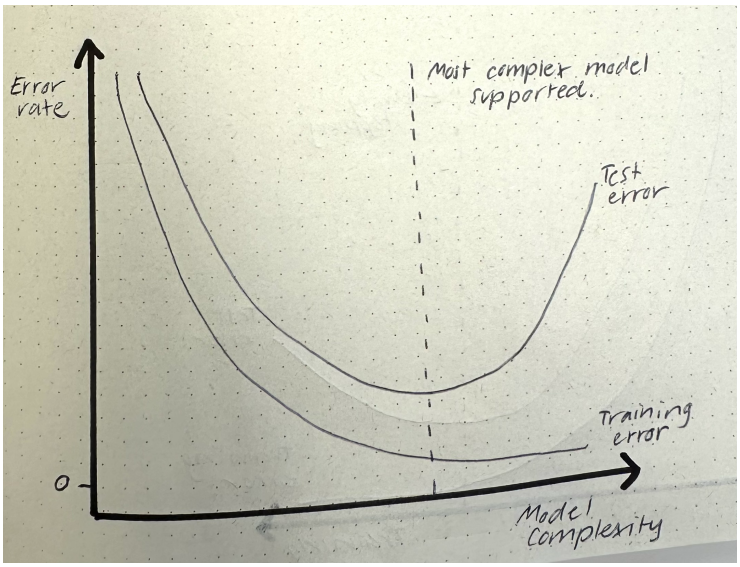

Figure 2: Error rate versus model complexity

1.3.3

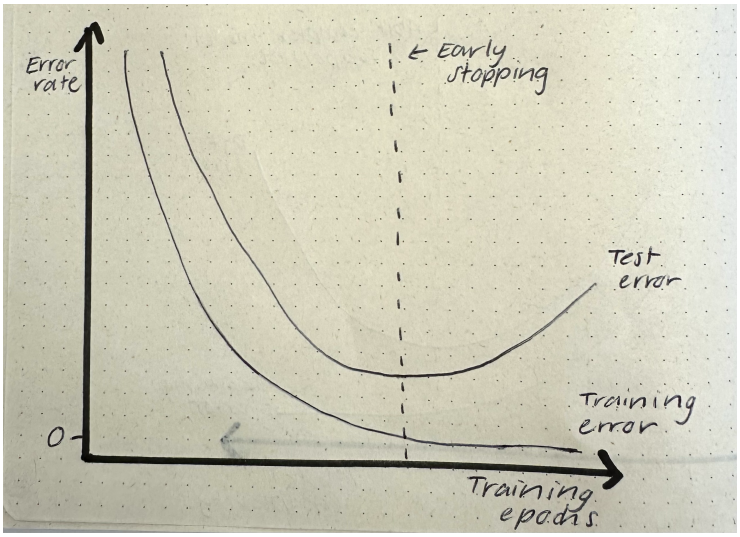

Figure 3: Error rate versus training epochs

This graph displays the relationship between increasing number of training epochs and error rate. At a low number of training epochs, the model has not had enough passes through the data to optimise the parameters and therefore the error starts higher for both test and training data (underfitting). As the

number of epochs increases, the model begins to fit the data better and the error reduces. The training error decreases as the model fine-tunes its parameters to the training set - however, as can be seen by the increase in the test error after a certain point, this is where the model starts to overfit the training data and it does not generalise well to the test data.

Early stopping can be used to prevent overfitting in deep learning neural networks, whereby the the model training is stopped when the test error either fails to improve or gets worse over subsequent training epochs. This is a reasonable regularization metric as it stops the model training when it seems to be at an optimal compromise between training accuracy and generalization performance. However, noisy validation metrics or very large models that need many epochs to converge may be at risk of applying early stopping prematurely. (Paul, 2025).

1.4 Capacity of neural network

We have a single hidden-layer neural network with two hidden units, and data with 2 features

$$X = (x_1, x_2).$$

Design an activation function $g(\cdot)$ applied in the hidden layer, and an output activation function $o(\cdot)$ applied in the output layer, as well as a set of values of the weights that will create a logistic regression model where the input to the logistic function is of the form:

$$\beta_0 + \beta_1 x_1 x_2.$$

Solution: Essentially, $\beta_0 + \beta_1 x_1 x_2$ instructs on the activation function choice, both for the hidden and the output layers. In the hidden layers, the presence of the $x_1 x_2$ term means that the activation function needs to compute a product. To achieve that, we need an hidden-layer activation function that squares its input. Since the function is logistic, the output will be a probability. Hence, the function to apply to the output layer is a sigmoid function.

Our goal is to design a neural network that outputs

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

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic (sigmoid) function. This means the output pre-activation must satisfy

$$z = \beta_0 + \beta_1 x_1 x_2.$$

Hidden Layer: Through an algebraic identity, we can engineer the network to compute the required interaction term $x_1 x_2$

$$(x_1 + x_2)^2 - (x_1 - x_2)^2 = 4x_1 x_2.$$

Thus, if we define two hidden units as

$$h_1 = (x_1 + x_2)^2, \qquad h_2 = (x_1 - x_2)^2,$$

then the product can be recovered as

$$x_1 x_2 = \frac{1}{4}(h_1 - h_2).$$

Let the hidden pre-activations be

$$a^{[1]} = W^{[1]} x + b^{[1]},$$

The first-layer pre-activations are: $\quad a_1^{[1]} = x_1 + x_2, \qquad a_2^{[1]} = x_1 - x_2.$

The way $a_1^{[1]}$ and $a_2^{[1]}$ are written down already encodes a choice of weights and biases. Each coefficient in front of x_1 and x_2 is an entry in the weight matrix. The constants added (here 0) are the bias vector

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad W^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

We then apply the hidden activation function

$$g(x) = x^2,$$

so that

$$h_i = g(a_i^{[1]}), \quad i = 1, 2.$$

Output Layer:
The output pre-activation is a linear combination of the hidden units:

$$z = \beta_0 + \frac{\beta_1}{4} h_1 - \frac{\beta_1}{4} h_2.$$

Plugging in h_1 and h_2 gives

$$z = \beta_0 + \frac{\beta_1}{4}\Big((x_1 + x_2)^2 - (x_1 - x_2)^2\Big) = \beta_0 + \beta_1 x_1 x_2.$$

Finally, applying the logistic function at the output yields

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

To Summarize:
Hidden layer

- Activation:

$$g(a) = a^2$$

- Weights:
$$W = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad b^{(h)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
- Outputs:
$$h_1 = (x_1 + x_2)^2, \quad h_2 = (x_1 - x_2)^2$$

Output layer

- Weights:
$$v = \begin{bmatrix} \frac{\beta_1}{4} \\ -\frac{\beta_1}{4} \end{bmatrix}, \quad b^{(o)} = \beta_0$$
- Activation:
$$o(z) = \sigma(z)$$

1.5 Neural network theory

Derive the weight updates in gradient descent for a neural network with 2 hidden layers (superscripts [1] and [2]) that each have $H^{[1]}$ and $H^{[2]}$ hidden units respectively. The output layer has superscript [3], and we want to classify the data into K classes.

The output of the network for classes $k = 1, 2, \ldots, K$ and one data point $x \in \mathbb{R}^d$ can be written as:

$$f_k(\theta; x) = o(a_k^{[3]}) = o\left(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]}\right),$$
$$h_m^{[2]} = \sigma(a_m^{[2]}) = \sigma\left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}\right),$$
$$h_i^{[1]} = \sigma(a_i^{[1]}) = \sigma\left(b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j\right),$$

where θ indicates the vector of all weights and biases.

While this is typically not recommended, in this exercise we use a quadratic loss function for classification. We use a softmax activation function at the output layer and a ReLU activation function at the hidden layers. Derive the weight update for the weights of the first hidden layer $w_{ij}^{[1]}$. Start by stating the gradient descent weight update for the $(r+1)$-th iteration as a function of the r-th iteration, and then compute the partial derivative needed in the update. *Show all the steps in your derivation. You may use the derivatives for activation functions introduced in class. There is no need to show the derivations of those.*

Solution: For any weight $w_{ij}^{[1]}$, the gradient descent update is:

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

where:

- η is the learning rate,
- L is the loss function (here quadratic loss),
- r indexes the iteration.

So we have to compute

$$\frac{\partial L}{\partial w_{ij}^{[1]}}.$$

Since $w_{ij}^{[1]}$ influences L through both hidden layers and the output layer, we need to use the chain rule. By the chain rule, the derivative of the loss with respect to $w_{ij}^{[1]}$ is:

$$\frac{\partial L}{\partial w_{ij}^{[1]}} = \sum_{k=1}^{K} \frac{\partial L}{\partial f_k} \frac{\partial f_k}{\partial a_k^{[3]}} \frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} \frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} \frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} \frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} \frac{\partial a_i^{[1]}}{\partial w_{ij}^{[1]}}$$

Computing each term:

- Quadratic loss derivative:

$$\frac{\partial L}{\partial f_k} = f_k - y_k$$

- Softmax derivative:

$$\frac{\partial f_k}{\partial a_l^{[3]}} = \begin{cases} f_k(1 - f_k), & k = l \\ -f_k f_l, & k \neq l \end{cases}$$

- Output layer to second hidden layer:

$$\frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} = w_{km}^{[3]}$$

- ReLU derivatives:

$$\frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} = \mathrm{ReLU}'(a_m^{[2]}), \quad \frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} = \mathrm{ReLU}'(a_i^{[1]})$$

- Second hidden to first hidden:

$$\frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} = w_{mi}^{[2]}$$

- First hidden to weight:

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

Define deltas:

$$\delta_k^{[3]} = \sum_{l=1}^{K} (f_l - y_l) \frac{\partial f_l}{\partial a_k^{[3]}}$$

$$\delta_m^{[2]} = \mathrm{ReLU}'(a_m^{[2]}) \sum_{k=1}^{K} w_{km}^{[3]} \delta_k^{[3]}$$

$$\delta_i^{[1]} = \mathrm{ReLU}'(a_i^{[1]}) \sum_{m=1}^{H^{[2]}} w_{mi}^{[2]} \delta_m^{[2]}$$

Gradient and Weight Update: The gradient with respect to $w_{ij}^{[1]}$ is:

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

The weight update is

$$\boxed{w_{ij}^{[1](r+1)} = w_{ij}^{[1](r)} - \eta \, \delta_i^{[1]} \, x_j}$$

GenAI Use and Sources Consulted:

Paul, R. (2025) ML Interview Q Series: How does Early Stopping function within deep learning, and why is it beneficial? Accessed at: https://www.rohan-paul.com/p/ml-interview-q-series-how-does-early

GPT-5 via Chat GPT was used to help understand concepts and guide the process of answering the questions (ie. using prompts such as: "what are the general steps when solving for critical points in a multivariate function?" and "explain in detail why the derivative of the Relu function is undefined when z = 0". In addition, it was used to assist in the typesetting of Latex.
Wolfram Alpha was used to find the Eigenvalues in question 1.1.