# Problem Set 1.

Giorgio Coppola (224545)
Minho Kang (239742)
Sofiya Berdiyeva (246934)
Deep Learning E1394

October 3, 2025

## 1 Theoretical part

### 1.1 Optimization

**Problem 1.1.** 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.

*Solution.* We first compute the gradient of

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

which is

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

Setting $\nabla f = 0$ gives the system

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

—

**Step 1: Solving for critical points.** Multiplying the three equations together yields

$$(2x)(2y)(2z) = (xyz)(xyz) \quad \Longrightarrow \quad (xyz)^2 = 8xyz.$$

Rearranging,

$$xyz(xyz - 8) = 0.$$

- If $xyz = 0$, then at least one coordinate is zero. Substituting into the equations shows that all must be zero. Hence $(0, 0, 0)$ is a critical point.
- If $xyz = 8$, then none of $x, y, z$ are zero. From $2x = yz$, $2y = xz$, and $2z = xy$, we see that

$$xyz = 2x^2 = 2y^2 = 2z^2,$$

so $x^2 = y^2 = z^2$. Thus $y = \pm x$, $z = \pm x$, with an even number of minus signs to ensure $xyz = 8$. This gives four additional critical points:

$$(2, 2, 2),\ (2, -2, -2),\ (-2, 2, -2),\ (-2, -2, 2).$$

—

**Step 2: Hessian matrix.** The Hessian is

$$H_f(x, y, z) = \begin{pmatrix} 2 & -z & -y \\ -z & 2 & -x \\ -y & -x & 2 \end{pmatrix}.$$

At the critical points:

$$H_f(0,0,0) = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \qquad H_f(2,2,2) = \begin{pmatrix} 2 & -2 & -2 \\ -2 & 2 & -2 \\ -2 & -2 & 2 \end{pmatrix}.$$

The other Hessians (at $(2,-2,-2)$, $(-2,2,-2)$, $(-2,-2,2)$) are obtained from $H_f(2,2,2)$ by permuting coordinates and flipping signs. Algebraically, each is related by

$$H' = Q^\top H_f(2,2,2) Q$$

for some orthogonal matrix $Q$ (a product of a permutation and a diagonal $\pm 1$ matrix). Thus they are all orthogonally similar, and therefore share the same eigenvalues.

—

**Step 3: Computing Eigenvalues.**

- At $(0,0,0)$: $H_f(0,0,0) = 2I_3$ has eigenvalues $(2,2,2)$, positive definite. Hence $(0,0,0)$ is a *local minimum*.
- At $(2,2,2)$: the eigenvalues are $(-2,4,4)$, so it is a *saddle point*.
- By similarity, the other three nonzero critical points also have eigenvalues $(-2,4,4)$ and are likewise saddles.

—

**Conclusion.** The function $f$ has five critical points:

$$(0,0,0) \text{ (local minimum)}, \quad (2,2,2),\ (2,-2,-2),\ (-2,2,-2),\ (-2,-2,2) \text{ (saddle points)}.$$

□

## 1.2 Activation functions

**Problem 1.2.** 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 $x, b, w \in R$.

*Solution.* The gradient of $f$ w.r.t. $b$ and $w$ is:

$$\nabla f(b,w) = \left( \frac{\partial f}{\partial b}, \frac{\partial f}{\partial w} \right).$$

We have to solve when $b + xw < 0$ and when $b + xw \geq 0$. However, at $b + xw = 0$, ReLU is not differentiable. Therefore:

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

□

## 1.3 Overfitting

**Problem 1.3.1 Error rate versus dataset size.** Sketch a graph of the typical behavior of training error rate ($y$-axis) as a function of training set size ($x$-axis). Add to this graph a curve showing the typical behavior of test error rate versus training set size, on the same axes. (Assume that we have an infinite test set drawn independently from the same joint distribution as the training set). Indicate on your $y$-axis where zero error is and draw your graphs with increasing error upwards and increasing training set size rightwards.

*Solution.* **Training Error Behavior:** When the training set size is small, the model has fewer data points, which allows it to fit the training data easily, resulting in a very low training error. However, this often leads to overfitting, as the model may become too complex for the small dataset. As the training set size increases, the training error starts to rise slightly as the model becomes unable to perfectly fit the larger set. Eventually, the training error reaches a plateau, limited by the model's approximation error and irreducible noise in the data.

**Test Error Behavior:** When the training set is small, the model overfits the training data, leading to a high test error. As the training set size increases, the test error decreases as the model generalizes better, reducing overfitting. The test error eventually levels off at a floor set by the model's approximation error and irreducible noise (often referred to as the Bayes error). The test error remains above the training error but narrows as the sample size increases.

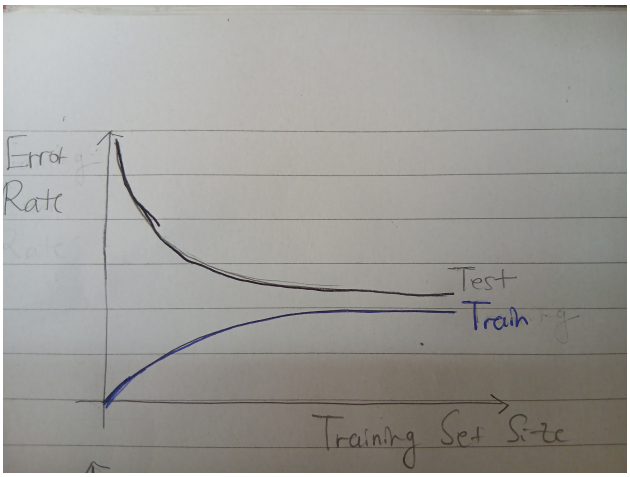


Figure 1: Typical training (blue) vs. test (black) error as training size increases.

□

**Problem 1.3.2 Error rate versus model complexity.** For a fixed training set size, sketch a graph of the typical behavior of training error rate ($y$-axis) versus model complexity ($x$-axis). Add to this graph a curve showing the typical behavior of the corresponding test error rate versus model complexity, on the same axes (again on an IID infinite test set). Show where on the $x$-axis you think is the most complex model that your data supports (mark this with a vertical line). Choose the $x$-range so that this line is neither on the extreme left nor on the extreme right. Indicate on your vertical axis where zero error is and draw your graphs with increasing error upwards and increasing complexity rightwards.

*Solution.* As model complexity increases, the training error typically decreases and approaches zero, as the model better fits the training data. For test error, a U-shaped pattern emerges: it decreases initially (due to reduced bias as the model captures the signal), but starts to rise once overfitting occurs. The vertical line marks the point of maximum complexity that the data can support. Increasing the sample size allows for higher model complexity, shifting the "max complexity" line to the right. This also moves the U-shape of the test error downward and to the right, while reducing the gap between the test and training error curves. □

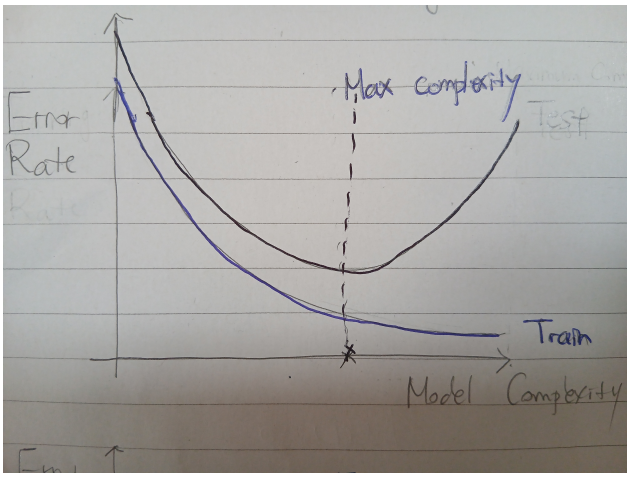


Figure 2: Training (blue) vs. test (black) error vs. model complexity (fixed training size).

**Problem 1.3.3 Training epochs.** Use a similar graph to illustrate the error rate as a function of training epochs in neural networks. One of the commonly used regularization methods in neural networks is early stopping. Describe (also using the graph) how early stopping is applied, and argue qualitatively why (or why not) early stopping is a reasonable regularization metric.

*Solution.* As the number of epochs increases, training error generally decreases and flattens as the optimizer fits the data. Validation error initially decreases (indicating a better fit to the signal) but then rises as the model starts to overfit to the noise in the training data. Early stopping halts training when performance on the validation set stops improving, selecting the epoch near the minimum of the validation curve. This acts as regularization by limiting training time, reducing model complexity, and improving generalization. □

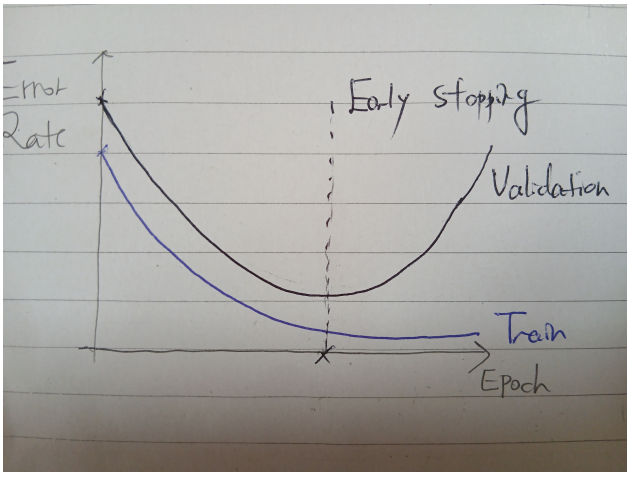


Figure 3: Training (blue) vs. validation (black) error vs. epoch.

## 1.4 Capacity of a neural network

**Problem 1.4.** 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.* To set up the problem, let $a^{[1]} = W^{[1]}X + b^{[1]}$ denote the hidden pre-activations, with components $a_1^{[1]}$ and $a_2^{[1]}$, and let $h_i = g(a_i^{[1]})$ be the hidden activations. We have to design $g$, $o$, and the weights so that the output pre-activation is $a^{[2]} = \beta_0 + \beta_1 x_1 x_2$ and $o(a^{[2]}) = \sigma(a^{[2]}) = \frac{1}{1+e^{-a^{[2]}}}$.

To solve this problem, we choose a hidden activation $g$ and output weights so that the output layer produces the desired pre-activation $a^{[2]}$ (here $g$ maps $a^{[1]}$ to $h$, and the output layer combines $h$ to form $a^{[2]}$). To isolate the interaction $x_1x_2$, we have to design the correct affine pre-activation functions of the hidden units $a_i^{[1]}$ and the correct hidden activation function $g(\cdot)$. The interaction term is

$$x_1x_2 = \tfrac{1}{4}\big[(x_1+x_2)^2 - (x_1-x_2)^2\big],$$

since

$$(x_1+x_2)^2 - (x_1-x_2)^2 = (x_1^2 + 2x_1x_2 + x_2^2) - (x_1^2 - 2x_1x_2 + x_2^2) = 4x_1x_2.$$

It follows that we should design the affine pre-activation functions of the hidden units to compute the squares of sums and differences. We set the hidden pre-activations to the sum and difference of the inputs and choose the hidden activation $g(z) = z^2$ ($z = a^{[1]}$ in this case). Therefore,

$$a_1^{[1]} = x_1 + x_2, \quad a_2^{[1]} = x_1 - x_2, \quad \text{and} \quad h_1 = (x_1+x_2)^2, \;\; h_2 = (x_1-x_2)^2,$$

which corresponds to weights $w_1 = (1,1)$, $w_2 = (1,-1)$ and biases $b_1 = 0$, $b_2 = 0$.

So, the output weights and bias are

$$v = \begin{pmatrix} \frac{\beta_1}{4} \\ -\frac{\beta_1}{4} \end{pmatrix}, \qquad b^{[2]} = \beta_0,$$

so that the output pre-activation is

$$a^{[2]} = b^{[2]} + v^\top h = \beta_0 + \tfrac{\beta_1}{4}h_1 - \tfrac{\beta_1}{4}h_2 = \beta_0 + \tfrac{\beta_1}{4}\big[(x_1+x_2)^2 - (x_1-x_2)^2\big] = \beta_0 + \beta_1 x_1 x_2.$$

Finally, we apply the output activation $o(a^{[2]}) = \sigma(a^{[2]}) = \frac{1}{1+e^{-a^{[2]}}}$ to obtain logistic regression with only the interaction term. □

## 1.5 Neural network theory

**Problem 1.5.** 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 $k = 1, 2, ..., K$ and one data point $x \in R^d$ can be written as such:

$$f_k(\theta; X) = o(a_k^{[3]}) = o\left(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]}\right) \tag{2}$$

$$h_m^{[2]} = \sigma(a_m^{[2]}) = \sigma\left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}\right) \tag{3}$$

$$h_i^{[1]} = \sigma(a_i^{[1]}) = \sigma\left(b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j\right). \tag{4}$$

where $\theta$ indicates the vector of all weights and biases.

While this is typically not recommended, in this exercise we use a 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 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.* Gradient descent weight update for the $(r+1)^{th}$ iteration:

$$w_{ij}^{(r+1)[1]} = w_{ij}^{(r)[1]} + \eta^{(r)} \Delta w_{ij}^{(r)[1]} = w_{ij}^{(r)[1]} + \eta^{(r)} \frac{\partial L(f(x), y)^{(r)}}{\partial w_{ij}^{(r)[1]}} \tag{5}$$

Given that we know the values of $w_{ij}^{[1]}$ at the $(r)^{th}$ iteration, as well as the value of learning rate $\eta^{(r)}$, the only thing to compute is the partial derivative of the loss function with respect to the weights of the first hidden layer.

$$\frac{\partial L(f(x), y)}{\partial w_{ij}^{[1]}} = \sum_{k=1}^{K} \frac{\partial L}{\partial f_k} \cdot \frac{\partial f_k}{\partial w_{ij}^{[1]}} = \sum_{k=1}^{K} -2(y_k - f_k) \frac{\partial f_k}{\partial w_{ij}^{[1]}} \tag{6}$$

Again, we have information on the observed values $y$ and predicted classes $f_k$. What we need to derive is the last term, namely $\frac{\partial f_k}{\partial w_{ij}^{[1]}}$. To do this, we will employ the chain rule.

$$\frac{\partial f_k}{\partial w_{ij}^{[1]}} = \overset{(1)}{\frac{\partial f_k}{\partial a_k^{[3]}}} \cdot \sum_{m=1}^{H^{[2]}} \overset{(2)}{\frac{\partial a_k^{[3]}}{\partial h_m^{[2]}}} \cdot \overset{(3)}{\frac{\partial h_m^{[2]}}{\partial a_m^{[2]}}} \cdot \overset{(4)}{\frac{\partial a_m^{[2]}}{\partial h_i^{[1]}}} \cdot \overset{(5)}{\frac{\partial h_i^{[1]}}{\partial a_i^{[1]}}} \cdot \overset{(6)}{\frac{\partial a_i^{[1]}}{\partial w_{ij}}}. \tag{7}$$

Terms #3 and #5 above are referring to the derivatives of ReLU activation functions, which are equal to 1 if $a_m^{[2]} > 0$ and $a_i^{[1]} > 0$, respectively, and equal to 0 in the opposite cases (i.e $a_m^{[2]} < 0$ and $a_i^{[1]} < 0$, in case of equality to zero they are not differentiable).

Breaking down the rest of the terms:

$$\frac{\partial f_k}{\partial a_k^{[3]}} = \frac{e^{a_k^{[3]}}}{\sum_{z=1}^{K} e^{a_z^{[3]}}} \cdot (1 - \frac{e^{a_k^{[3]}}}{\sum_{z=1}^{K} e^{a_z^{[3]}}}) \tag{(1)}$$

$$\frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} = (b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]})' = w_{km}^{[3]} \tag{(2)}$$

$$\frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} = (b_k^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]})' = w_{mi}^{[2]} \tag{(4)}$$

$$\frac{\partial a_i^{[1]}}{\partial w_{ij}} = (b_i^{[1]} + \sum_{j=1}^{d} w_{ij} x_j)' = x_j \tag{(6)}$$

Bringing it all back together:

$$\frac{\partial L(f(x), y)}{\partial w_{ij}^{[1]}} = \sum_{k=1}^{K} -2(y_k - f_k) \cdot \frac{e^{a_k^{[3]}}}{\sum_{z=1}^{K} e^{a_z^{[3]}}} \cdot (1 - \frac{e^{a_k^{[3]}}}{\sum_{z=1}^{K} e^{a_z^{[3]}}}) \cdot \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} \cdot 1_{\{a_m^{[2]} > 0\}} \cdot w_{mi}^{[2]} \cdot 1_{\{a_i^{[1]} > 0\}} \cdot x_j \tag{8}$$

Thus, we can use this equation to compute partial derivative of loss function with respect to the weights of the first hidden layer at each $r^{th}$ iteration.

□