GRAD-E1394: Deep Learning
Hertie School
Fall 2025
Prof. Lynn Kaack

Problem Set 1

Group C: Dominik A. (232487), Elena D. (232619), Benjamin A. (235194)[1]

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.

In general, to find critical points of a multivariate function we will calculate each of the partial derivatives, set them equal to 0, solve system of equations, then also identify where in function these derivatives dont exist- when they dont equal 0. Then to classify, we will use these critical points to compute the hessian matrix, and solve for eigenvalues to classify what each critical point is.

Step 1: Gradient Critical points occur at:

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

Compute the partial derivatives:

$$f_x = 2x - yz, \qquad f_y = 2y - xz, \qquad f_z = 2z - xy.$$

Setting these equal to zero gives the system:

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

Step 2: Critical points

- For the first point, we can simply solve for if $x = 0$, then from the equations it follows that $y = 0$ and $z = 0$. Thus, $(0, 0, 0)$ is the first critical point.
- If $x, y, z \neq 0$, then we can rearrange the formula to get each variable by itself, and then substitute the first formula into the second and solve. Substituting $x = yz/2$ and $y = xz/2$ gives $z^2 = 4$, and then the answer of $z = \pm 2$. From there we would substitute $z = \pm 2$ into the original function which gives us $x = y = \pm 2$ and solve to get each non zero critical point.
  - If $z = 2$: then $x = y$ and $xy = 4$, so $(2, 2, 2)$ and $(-2, -2, 2)$.
  - If $z = -2$: then $x = -y$ and $xy = -4$, so $(2, -2, -2)$ and $(-2, 2, -2)$.

So, the sets of different critical points are:

$$(0, 0, 0), \quad (2, 2, 2), \quad (-2, -2, 2), \quad (2, -2, -2), \quad (-2, 2, -2).$$

[1] **Note on GenAI use:** ChatGPT was used to verify selected results (which we initially derived on paper) and to generate some of the more complex LaTeX formulas e.g. for Problem 1.5. GitHub Copilot was used for type annotations/ function docstrings/ comments in the coding part of the assignment.

Step 3: Hessian The Hessian of the function f is

$$H(x,y,z) = \begin{bmatrix} 2 & -z & -y \\ -z & 2 & -x \\ -y & -x & 2 \end{bmatrix}$$

which is just the second derivative test from the original function.

Step 4: Classification Next, we can create and evaluate the hessian matrix at each critical point and then compute the eigenvalues to classify each critical point.

- At $(0,0,0)$ the Hessian is:
$$H(0,0,0) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$
- At $(2,2,2)$:
$$H(2,2,2) = \begin{bmatrix} 2 & -2 & -2 \\ -2 & 2 & -2 \\ -2 & -2 & 2 \end{bmatrix}$$
- At $(-2,-2,2)$:
$$H(-2,-2,2) = \begin{bmatrix} 2 & -2 & 2 \\ -2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$
- At $(2,-2,-2)$:
$$H(2,-2,-2) = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & -2 \\ 2 & -2 & 2 \end{bmatrix}$$
- At $(-2,2,-2)$:
$$H(-2,2,-2) = \begin{bmatrix} 2 & 2 & -2 \\ 2 & 2 & 2 \\ -2 & 2 & 2 \end{bmatrix}$$
- The first Hessian matrix of $(0,0,0)$ gives the eigenvalues of $(2,2,2)$ showing all values as positive which means it is classified as a local minimum. Each of the other critical points gives eigenvalues $(-2,4,4)$ which as we can see contain both positive and negative values, therefore are all classified as saddle points.[2]

  Final Classification:

  - Local minimum: (0,0,0)
  - Saddle points: (2,2,2), (-2,-2,2), (2,-2,-2), (-2,2,-2)
  - No local maxima.

[2]Eigenvalues were verified using Wolfram Alpha.

1.2 Activation functions

Compute the gradient of

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

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

Step 1: Rewrite with one variable to make the function easier. Let $u = b + xw$. Then $f(b,w) = \mathrm{ReLU}(u)$.

Step 2: Take derivative of ReLU.

$$\frac{d}{du}\,\mathrm{ReLU}(u) = \begin{cases} 0 & u < 0, \\ \text{undefined (subgradient in } [0,1]) & u = 0, \\ 1 & u > 0. \end{cases}$$

Step 3: Partial derivatives of u with respect to each sub variable.

$$\frac{\partial u}{\partial b} = \frac{\partial}{\partial b}(b+xw) = 1+0 = 1, \qquad \frac{\partial u}{\partial w} = \frac{\partial}{\partial w}(b+xw) = 0+x = x.$$

Step 4: Apply chain rule.

$$\frac{\partial f}{\partial b} = \mathrm{ReLU}'(u) \cdot \frac{\partial u}{\partial b} = \mathrm{ReLU}'(u) \cdot 1,$$

$$\frac{\partial f}{\partial w} = \mathrm{ReLU}'(u) \cdot \frac{\partial u}{\partial w} = \mathrm{ReLU}'(u) \cdot x.$$

Step 5: Result.

$$\frac{\partial f}{\partial b} = \begin{cases} 1 & \text{if } b+xw > 0, \\ 0 & \text{if } b+xw < 0, \end{cases} \qquad \frac{\partial f}{\partial w} = \begin{cases} x & \text{if } b+xw > 0, \\ 0 & \text{if } b+xw < 0. \end{cases}$$

Step 6: Compact form

$$\nabla_{(b,w)}\,\mathrm{ReLU}(b+xw) = \begin{cases} (1,x), & (b+xw > 0) \\ (0,0), & (b+xw < 0) \end{cases}$$

Interpretation. When the input to the ReLU ($b+xw$) is positive, the output is sensitive to both parameters: increasing b increases the output one-for-one, and increasing w changes the output in proportion to x. When the input is negative, the neuron is "off" and parameter changes have no effect on the output.

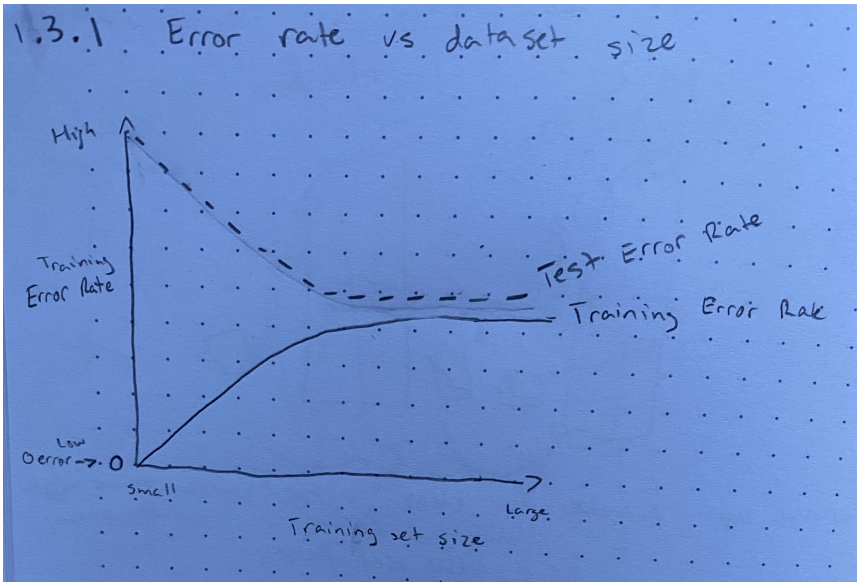
1.3 Overfitting

This question will test your general understanding of overfitting as it relates to model complexity and training set size. Consider a continuous domain and a smooth joint distribution over inputs and outputs, so that no test or training case is ever duplicated exactly. The graphs in your solutions should be drawn on paper and then included as a picture. In your answers describe how your graphs should be read.

1.3.1 Error rate versus dataset size

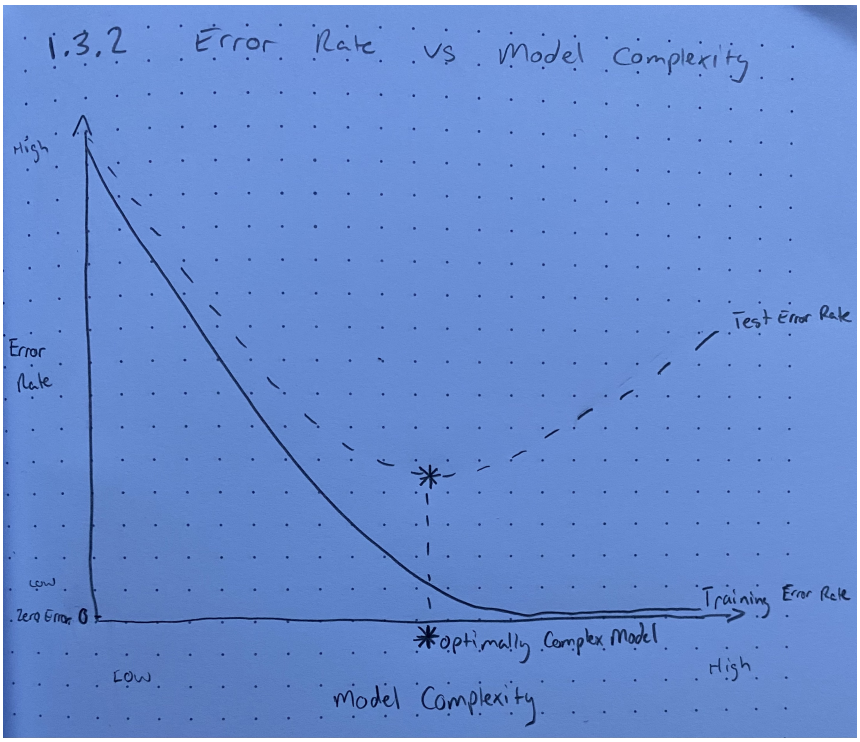
With a small training set, the model fits the data easily, resulting in very low training error but high risk of overfitting. As the dataset grows, training error rises slightly because the model cannot perfectly fit all points, eventually plateauing due to the model's approximation limits and irreducible noise.

For the test set, small training sizes cause overfitting, leading to high test error. As more data is added, the model generalizes better, reducing test error, which eventually levels off. Test error always stays above training error but the gap narrows with larger datasets.



1.3.2 Error rate versus model complexity

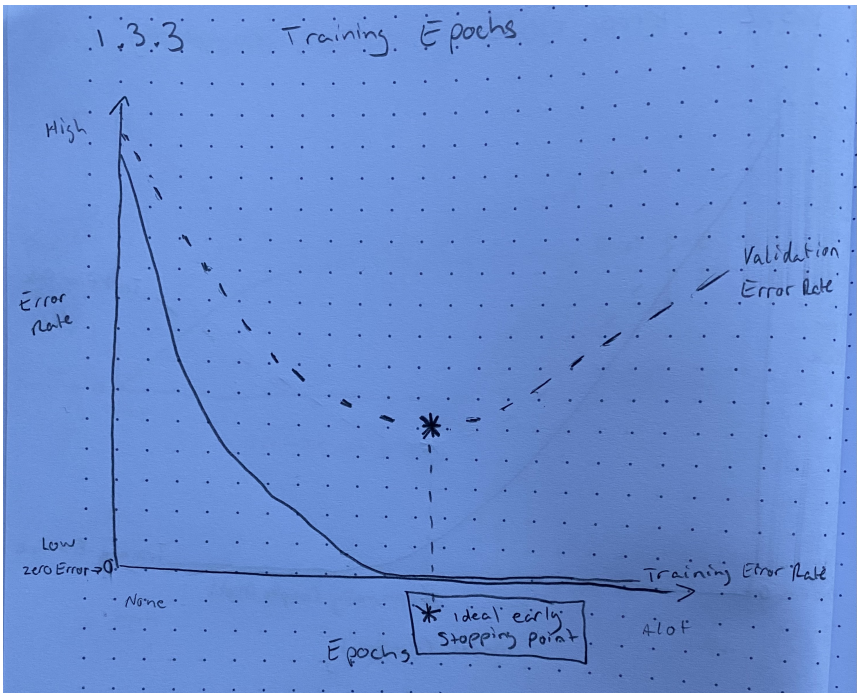
For a fixed dataset, training error decreases with model complexity, approaching zero as the model fits the data more closely. Test error follows a U shaped curve with it initially decreasing as bias reduces, then rises once overfitting occurs. The maximum supported complexity for the data is marked with a vertical line. Increasing the training size allows higher complexity, shifting this line and the U-shaped test error curve to the right, while lowering test error and reducing the gap with training error.



1.3.3 Training Epochs

When the number of epochs increases, training error generally decreases and flattens as the learning optimizer fits to the data. Validation error also initially decreases but then rises when the model starts to overfit to the noise in the training data.

Early stopping is a good regularization technique because it directly addresses the problem of overfitting in an intuitive and computationally efficient way. By monitoring the models performance on a validation set, it prevents the network from training for too long, making sure that the final model is the one that best generalizes to unseen data, not the one that just memorizes the training data's noise. This process limits the model's complexity, as the weights arent given enough epochs to get to specialized on the data. Additionally, early stopping provides efficiency for computational savings by ending the training process as soon as performance on the validation set gets worse. It is also easy to implement, not requiring modification of the model's architecture or the loss function. However, one potential issue of early stopping is the loss of potential improvement. It's possible that a temporary increase in validation error could be a part of the models learning process, and stopping the training too early might cause the model to miss a later period of improved generalization. This can happen if the model needs more time to converge to a better solution, and a premature early stopping rule could prevent it from ever reaching that point.



1.4 Capacity of a 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: $\beta_0 + \beta_1 x_1 x_2$.

We construct a single hidden layer network with two hidden units/nodes that implements a logistic regression whose pre-sigmoid input needs to be in the form of $\beta_0 + \beta_1 x_1 x_2$.

We choose the hidden activation $g(u) = u^2$ and the output activation $o(a) = \sigma(a) = 1/(1 + e^{-a})$ which is the logisitc function. We choose the hidden pre-activations to be

$$a_1 = x_1 + x_2, \qquad a_2 = x_1 - x_2,$$

which is obtained with hidden weights w and biases b:

$$w_{1\cdot}^{(1)} = (1, 1),\ b_1^{(1)} = 0, \qquad w_{2\cdot}^{(1)} = (1, -1),\ b_2^{(1)} = 0.$$

Difference of these two hidden outputs isolates the cross-term and rescales it:

$$h_1 - h_2 = (x_1 + x_2)^2 - (x_1 - x_2)^2 = 4x_1 x_2.$$

Since we want output of the form $\beta_0 + \beta_1 x_1 x_2$, we do it by taking a linear combination

$$\frac{\beta_1}{4} h_1 - \frac{\beta_1}{4} h_2 = \beta_1 x_1 x_2.$$

So to make the coefficient of $x_1 x_2$ exactly β_1, we choose the output weights

$$v_1 = \tfrac{\beta_1}{4}, \qquad v_2 = -\tfrac{\beta_1}{4},$$

together with an output bias

$$b_{\text{out}} = \beta_0.$$

so the pre-sigmoid output is

$$a_{\text{out}} = b_{\text{out}} + v_1 h_1 + v_2 h_2 = \beta_0 + \frac{\beta_1}{4}\big((x_1 + x_2)^2 - (x_1 - x_2)^2\big) = \beta_0 + \beta_1 x_1 x_2.$$

Applying the sigmoid gives the logistic model we wanted:

$$o(a_{\text{out}}) = \sigma(\beta_0 + \beta_1 x_1 x_2).$$

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.

1. General update rule

Gradient descent update at iteration r is

$$w_{ij}^{[1](r+1)} = w_{ij}^{[1](r)} - \eta \left.\frac{\partial L}{\partial w_{ij}^{[1]}}\right|_{\theta^{(r)}}.$$

So the problem reduces to computing $\partial L / \partial w_{ij}^{[1]}$.

2. Loss function (quadratic)

For one training example:

$$L = \tfrac{1}{2}\sum_{k=1}^{K}(f_k - y_k)^2, \qquad \text{so} \qquad \frac{\partial L}{\partial f_k} = f_k - y_k.$$

3. Chain-rule

By the chain rule, the gradient with respect to a first-layer weight is

$$\frac{\partial L}{\partial w_{ij}^{[1]}} = \sum_{k=1}^{K} \frac{\partial L}{\partial f_k} \cdot \frac{\partial f_k}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial h^{[2]}} \cdot \frac{\partial h^{[2]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial h^{[1]}} \cdot \frac{\partial h^{[1]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial w_{ij}^{[1]}}.$$

Next we expand each derivative.

4. Expanding each derivative piece

Step 1: Derivative of Loss w.r.t Output f_k

$$L = \frac{1}{2}\sum_{k=1}^{K}(f_k - y_k)^2$$
$$\frac{\partial L}{\partial f_k} = \frac{\partial}{\partial f_k}\frac{1}{2}(f_k - y_k)^2 = f_k - y_k$$

Quadratic loss derivative.

Step 2: Derivative of output f w.r.t. pre-activation $\mathbf{a}^{[3]}$ (softmax) The softmax activation for outputs is

$$f_k = \text{softmax}_k(\mathbf{a}^{[3]}) = \frac{e^{a_k^{[3]}}}{\sum_{t=1}^{K} e^{a_t^{[3]}}}.$$

Softmax jacobian:

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

Equivalently, as expression:

$$\frac{\partial f_l}{\partial a_k^{[3]}} = f_k(\delta_{lk} - f_l),$$

where δ_{lk} is the Kronecker delta.

*T*he Jacobian captures the coupling between output units produced by the softmax.

Step 3: Derivative of Output Pre-activation w.r.t Hidden Layer 2 Activation

$$a_k^{[3]} = b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]}$$

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

Linear combination derivative.

Step 4: Derivative of Hidden Layer 2 Activation w.r.t Pre-activation

$$h_m^{[2]} = \sigma(a_m^{[2]})$$

$$\frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} = \sigma'(a_m^{[2]})$$

ReLU derivative: $\sigma'(x) = 1$ if $x > 0$, else 0.

Step 5: Derivative of Hidden Layer 2 Pre-activation w.r.t Hidden Layer 1 Activation

$$a_m^{[2]} = b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}$$

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

Linear combination derivative.

Step 6: Derivative of Hidden Layer 1 Activation w.r.t Pre-activation

$$h_i^{[1]} = \sigma(a_i^{[1]})$$

$$\frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} = \sigma'(a_i^{[1]})$$

ReLU derivative.

Step 7: Derivative of Hidden Layer 1 Pre-activation w.r.t Weight $w_{ij}^{[1]}$

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

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

Linear dependence on input.

Step 8: Full Gradient (Chain Rule)

Combining all steps:

$$\frac{\partial L}{\partial w_{ij}^{[1]}} = \sum_{k=1}^{K} \left[\sum_{l=1}^{K} (f_l - y_l)\, f_k(\delta_{lk} - f_l) \right] \left[\sum_{m=1}^{H^{[2]}} w_{km}^{[3]}\, \sigma'(a_m^{[2]})\, w_{mi}^{[2]} \right] \sigma'(a_i^{[1]})\, x_j$$

Explanation: Multiply all derivatives along the path from $w_{ij}^{[1]} \to a_i^{[1]} \to h_i^{[1]} \to a_m^{[2]} \to h_m^{[2]} \to a_k^{[3]} \to f_k \to L$ and sum over all hidden units and outputs.

Which can be plugged back into the gradient-descent update rule of

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