KAIST AI Mini-Course Review: The Core Challenges

Lecture 1 Challenges (Ch. 1-2: Linear Algebra & Calculus)

Challenge 1: The Tensor Contraction Speedup (Einstein Notation)

Preamble: Modern deep learning frameworks rely on efficient GPU computation. This efficiency is achieved by eliminating slow loops and using compact tensor operations. Can you correctly translate the language of speed?

Reference: Exercise 1.1.3[1] (Tensor Contraction)

Exercise: Express the following two operations using Einstein Summation Notation: (1) The matrix-vector product $y = Ax$. (2) The Frobenius norm of a matrix A: $\|A\|_F^2 = \sum_{i,j} A_{ij}^2$.

Question 1: Given the contraction $C_{ikl} = A_{ij}B_{jkl}$, what is the final dimensionality of C?

Question 2: In GPU programming, why is **memory contiguity** (how elements are stored) often more important than the theoretical floating-point operation count (FLOPs)?

Challenge 2: The Dimensionality Curse (SVD/PCA)

Preamble: We live in a world of high-dimensional data. Using raw data leads to unmanageably large matrices, overfitting, and slow training. The SVD is our tool for disciplined reduction.

Reference: Exercise 1.2.2: SVD for Matrix Completion: Low-Rank Approximation and Prediction

Exercise: Given a matrix A representing a dataset, calculate the minimum number of singular values k required to retain 90% of the total energy (Frobenius norm squared) of A.

Question 1: How does the choice of k directly reflect the **trade-off between memory and information loss**?

Question 2: Why is SVD on the covariance matrix **computationally superior** to finding the eigenvectors of $A^T A$ for large N but small D?

Challenge 3: Chain Rule and the Backpropagation Mechanism

Preamble: The core of training any neural network relies on the Chain Rule of calculus, which translates into the Backpropagation algorithm. This challenge requires you to rigorously apply the chain rule to derive the gradient of a simple two-layer network.

Reference: Exercise 2.1.1 AD and Jacobian-Vector Products

[1] Exercise numbering corresponds to the current draft of the "Mathematics of Generative AI" living book, which is available at `https://github.com/mchertkov/Mathematics-of-Generative-AI-Book`. Here and below, a shorter version of the challenge is presented. See the current draft of the book for complete formulation.

Exercise: Given a simple two-layer network $f(x) = W_2\sigma(W_1 x)$, where $W_1 \in \mathbb{R}^{h\times d}$, $W_2 \in \mathbb{R}^{m\times h}$, and σ is an element-wise activation function:

(a) Write the full Jacobian $J = \frac{\partial f}{\partial x}$ as a product of matrix and diagonal matrix Jacobians.

(b) If L is a scalar loss function, use the chain rule to derive the gradient $\frac{\partial L}{\partial W_1}$ in terms of W_2, σ', and the loss gradient $\frac{\partial L}{\partial f}$.

Question 1: Explain how the *chain rule* naturally leads to the *forward pass* (for the function value $f(x)$) being separate from the *backward pass* (for the gradient ∇L) in terms of data flow.

Question 2: Consider a very deep network with T layers. If the weights are initialized with large variance, the product of many Jacobians can lead to the *exploding gradient* problem. How do common techniques like *gradient clipping* address this issue?

Challenge 4: Learning Dynamics from Data (ODE-Based Parameter Estimation)

Preamble: In dynamics-aware AI, the goal shifts from merely fitting data points $x(t)$ to accurately modeling the underlying rate of change, $\dot{x}(t)$, with a model $f(t; x(t); \theta)$ (possibly physics-informed). This challenge explores the power of fitting vector fields over fitting trajectories.

Reference: Exercise 2.2.3: Regression and Parameter Estimation for an ODE

Exercise: Given noisy data for a system which we believe is described by $\dot{x} = -\gamma x + \cos(2t)$, use ODE-based regression to fit the dynamic model and estimate the parameter γ.

Question 1: Explain the fundamental advantage of *ODE-based regression* (fitting $\dot{x}$ directly) over *Direct Regression* (fitting $x(t)$ with a standard polynomial) when the data is corrupted by noise.

Question 2: In the context of estimating the parameter γ, what key property does the ODE formulation leverage that helps stabilize the estimation, even when the trajectory data $x(t)$ is highly variable?

Lecture 2 Challenges (Ch. 3–4: Optimization & Deep Learning)

Challenge 5: Following the Flow — Optimization Landscapes as Vector Fields

Preamble: Optimization algorithms in AI are best understood geometrically: as trajectories of a vector field on a loss landscape. This challenge contrasts the predictability of convex flows with the fragility and multiplicity of paths in non-convex settings.

Reference: Exercise 3.2.1 (Gradient Descent Trajectories in Convex and Non-Convex Landscapes)

Exercise: Using the companion notebook `Convex_vs_NonConvex_Landscapes.ipynb`, implement gradient descent and visualize parameter trajectories over:

(a) a convex quadratic loss,

(b) a non-convex neural-network loss.

Overlay trajectories corresponding to multiple initializations on the provided contour plots.

Question 1: In the convex case, why do all trajectories converge to the same minimizer despite different initial conditions? How does the learning rate affect the geometry of the path?

Question 2: In the non-convex case, do different initializations lead to different basins of attraction? What qualitative role does noise (SGD-like perturbations) play in navigating saddle points and shallow minima?

Challenge 6: Optimization Meets Data Geometry — A Tiny Transformer Case Study

Preamble: Even the smallest transformers expose the sensitivity of adaptive optimizers to data geometry, sequence structure, and hyperparameters. This challenge uses a minimal transformer to make these effects visible and interpretable.

Reference: Exercise 3.5.1 (Exploring Optimizers and Data Geometry in Tiny Transformers)

Exercise: Starting from `tiny-transformer.ipynb`, retrain the model under controlled modifications:

(a) change the training text (natural or synthetic),

(b) vary the context window length,

(c) compare SGD, Adam, and RMSProp under matched settings.

Question 1: How does sequence structure (e.g., repetition, punctuation, spacing) affect optimizer performance and convergence speed?

Question 2: RMSProp is known to stagnate after large early gradients. Explain this phenomenon geometrically and contrast it with Adam's two-moment adaptation.

Question 3: What do confusion matrices reveal about how optimization and data geometry jointly shape learned representations?

Challenge 7: Capacity, Scale, and Inductive Bias in CNNs

Preamble: Convolutional neural networks balance expressiveness, generalization, and computational cost through architectural choices. This challenge probes how filter count and size shape that balance.

Reference: Exercise 4.1.3 (Exploring the Impact of Filter Count and Size in a CNN)

Exercise: Using `cnn-simple-MNIST.ipynb`, systematically vary:

(a) the number of filters per convolutional layer,

(b) the convolutional kernel size.

Record training/test accuracy and convergence curves for each configuration.

Question 1: How does increasing channel count affect expressiveness and overfitting on MNIST?

Question 2: How do larger kernels alter the effective receptive field, and why does this not always translate into better performance?

Question 3: How would these conclusions change for deeper architectures or more complex datasets?

Challenge 8: Compression, Geometry, and Abstraction in Deep Networks

Preamble: Deep networks do not merely fit data — they progressively compress and reshape it onto low-dimensional manifolds. This challenge makes the geometry of representation learning explicit.

Reference: Exercise 4.3.2 (Exploring Learned Low-Dimensional Manifolds)

Exercise: Using `CNN-MNIST-PCA.ipynb`, perform PCA on activations across multiple layers of a trained CNN.

Question 1: How does intrinsic dimensionality evolve with depth, and what does this reveal about hierarchical abstraction?

Question 2: Compare PCA spectra before and after nonlinearities. How do activations reshape the data manifold?

Question 3: When training on reduced data, does the representation compress or memorize? How is this reflected in the PCA spectrum?