1 Problem 1: Description of the Problem, Subscribe a ML/NN Based Solution

Process:

Definition:

Example:

2 Problem 2: Prescribe a strategy to optimize the NN.

Process:

Definition:

Example:

3 Problem 3: Explain step by step inference for basic algorithms (MLP, CNN, GNN, attention mechanism) in terms of numpy or basic tensor operations.

Process:

Definition:

Example:

4 Problem 4: Be able to explain why such a solution might work or fail.

Process:

Definition:

Example:

5 Neural Network Engineering

5.1 Data

Summary:

5.2 Evaluation

5.3 Optimization/Training

5.4 Regularization & Modelling

5.5 Experiments

6 Loss Functions

Summary:

7 Algorithms

Summary:

<table>
<tr><th>Algorithm</th><th>Inputs</th><th>Outputs</th><th>Equations</th></tr>
<tr><td>RNN</td><td>x_t, h_{t-1}</td><td>y_t, h_t</td><td>$h_t = \tanh(\text{Linear}(h_{t-1}) + \text{Linear}(x_t))$
$y_t = \text{MLP}(h_t)$</td></tr>
<tr><td colspan="4">• x_t: Input, h_t: Hidden state, y_t: Output</td></tr>
<tr><td>GRU</td><td>x_t, h_{t-1}</td><td>y_t, h_t</td><td>$z_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$
$r_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$
$\tilde{h}(t) = \tanh(\text{Linear}(x_t) + \text{Linear}(r_t \odot h_{t-1}))$
$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$</td></tr>
<tr><td colspan="4">• z_t: Update gate, r_t: Reset gate
• h_t: Hidden state</td></tr>
<tr><td>LSTM</td><td>x_t, h_{t-1}</td><td>h_t, c_t</td><td>$f_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$
$i_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$
$o_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$
$\tilde{c}_t = \tanh(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$
$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
$h_t = o_t \odot \tanh(c_t)$</td></tr>
<tr><td colspan="4">• f_t: Forget gate, i_t: Input/Update gate, o_t: Output gate
• h_t: Hidden state, c_t: Cell state</td></tr>
</table>

7.1 Geometric DL Blueprint for NNs

Summary: Unify various networks around symmetry. ADD IMAGE

8 Code to Paper

8.1 Ideas

Summary:

Exercise
Write research ideas. Get a mentor to rate them
Ask other researchers about their taste
Read about history of research ("The Structure of Scientific Revolutions")
De-risk your ideas: Proactive idea evaluation mitigates research risks (kill fast, learn fast) 1. Identify potential bottlenecks 2. Prioritize and commit X amt. of time to exploring them 3. Decide if you should continue or pivot

- Research Taste

Warning:
- Getting attached to one direction.
- Lack of research knowledge / intimacy.
- Environment is not supportive of your interests.

8.2 Code / Experiments

Summary:

Tools	Links
Artifacts to create/track w/ experiments • Data, code (scripts / modules), models (weights, configurations), results, predictions, plots, meeting notes, papers, documentation, etc.	
Git (Version Control): Enables effective tracking and collaborative changes • Tracking changes, collaboration, backup, revert. • Add, commit -m "message", push, pull, merge, diff, revert, branch, checkout, log, status, etc. • .gitignore: Ignore files, directories, or patterns	Git Guide
GitHub (Collaborative Code Hosting): Facilitates sharing and collaboration on code • Collaboration, sharing, open science, project hosting.	GitHub
Cookiecutter (Project Template): Standardizes project structure	Cookiecutter Repo

- Logical, flexible, and reasonably standardized project structure for doing and sharing data science work.
- File Structure
  - data/:
    - external/: Data from third party sources.
    - interim/: Intermediate data that has been transformed.
    - processed/: The final, canonical data sets for modeling.
    - raw/: The original, immutable data dump.
  - src/: Source code for use in this project.
    - __init__.py: Makes src a Python module.
    - config.py: Configuration settings.
    - dataset.py: Code to load data.
    - features.py: Code to build features.
    - modeling/: Code to train models.
      - __init__.py: Makes modeling a Python module.
      - predict.py: Code to make predictions.
      - train.py: Code to train models.
    - plots.py: Code to create plots.
  - docs/: Documentation for this project.
  - models/: Trained and serialized models, model predictions, or model summaries.
  - notebooks/: Jupyter notebooks.
  - references/: Data dictionaries, manuals, and all other explanatory materials.
  - reports/: Generated analysis as HTML, PDF, LaTeX, etc.
    - figures/: Generated graphics and figures to be used in reporting.
  - pyproject.toml: Project information and dependencies.
  - requirements.txt: The requirements file for reproducing the analysis environment.
  - setup.cfg: Configuration file for setting up the project.
  - LICENSE:
  - Makefile:
  - README.md:

Summary:

Tools	**Links**
Cookiecutter (Project Template): Standardizes project structure	Opinions

- **Design Philosophy:** Prioritizes conventions and reasonable defaults to streamline project setup. Opinions:
  - Data analysis is a DAG:
    1. Raw data
    2. Compute features
    3. Plot analysis on raw data
    4. Train model
    5. Compute statistics on features
  - Raw data is immutable (i.e. never change raw data)
    - **Dos:**
      - Pipeline code: Process raw data → final analysis.
      - Cache outputs: Serialize or cache intermediate steps.
      - Reproducible results: Enable full reproduction from code and raw data only.
    - **Don'ts:**
      - Never edit raw data: Avoid manual edits or format changes
      - Never overwrite raw data: Do not replace raw data with processed data.
      - Single raw data version: Maintain only one version of raw data.
  - Data should (mostly) not be kept in source control
    - GitHub warns for files over 50MB and rejects files over 100MB.
    - Use s3, azcopy, gcloud, drive to store data (i.e. cloud services)
    - Use cloudpathlib to access cloud data in the same way as pathlib to access local data.
  - Notebooks are for exploration and communication, source files are for repetitions.
  - Refactor the good parts into source code (Refactor Example).
    - Don't write code to do the same task in multiple notebooks.
  - Keep your modelling organized (PyTorch Example)
    - Predictions (csv), training log (csv), stats (txt), model config / hyperparameters (json)
  - Build from the environment up (Mamba)
    - Use mamba rather than conda for faster environment management.
    - Create a environment.yml file to manage dependencies.

8.3 Writing / Analyzing

Summary:

Exercise	Links
Writing Skeleton	Step-by-Step Guide to Undergraduate Writing How to write a first-class paper (Nature) Preparing Manuscript: Scientific Writing for Publication

1. Start w/ Figures (How would you want to telll the story?)
2. Write the structure (Introduction, Methods, Experiments and Results, Discussion)
3. 2-3 sentence pitch for your idea
4. Bullet points inside of each section (What are you expecting to cover?)
5. Fill in text, repeat.

Figures: Move quick, perfect later

- Figure #1: Tells problem in simple way (30s elevator pitch)
- Figure #2-3: Conceptual or data centric
  - How are you solving the problem?
  - What does the data look like?
- Figure #4-8: Quantitative evidence
- **Recommendations:**
  - Napkin/whiteboard figures first
  - Make good enough version w/ code (svg, png) using matplotlib, seaborn, etc.
  - Finetune w/ InkScape, Illustrator, GIMP, etc
- **Anatomy of a Figure Examples (L8):**
  - Slide 44: Task, Slide 45: Model + EDA
  - Slide 46-47: Quantitative evidence, Slide 48: Different ways of telling same story (e.g. tables or plots)

Pick good, consistent colors	ColorBrewer, Matplotlib Colormaps Seaborn Palettes, NeurIPS 18 Visualization for ML tutorial

- Be mindful of how colours can help tell a story, accessibility is also important.

Pair-writing (Como Pair-Coding)	Rubber Duck Debugging, Pair Programming Pair Writing, Pair Writing in Government

- Working together → Help communicate thoughts adn put you in a diff. attitude.

Communal writing (Social pressure → accountable)	Harvard Writing Center

1. Setup an objective, measurable goal.
2. Set a time for writing period, take breaks
3. Share progress at the end of each session, share writing stuggles if needed.
4. Reflect if there are some reasons why it is hard to write.

Writing: Focus on quick iterations

- Google docs and Paperpile (Copy DOI, paste, click, done) $\xRightarrow{\text{Export w/ bibtex}}$ LaTex (Overleaf)

Interactive Apps	Streamlit, Gradio Hugging Face, Hugging Face Spaces Academic Project Page Template

9 Symmetries, Tabular Data, Sets

9.1 Symmetries in Data

Summary: Transformations that preserve data characteristics.

Transformation Type	Description
Invariance ($f(g(x)) = f(x)$)	Invariant to a trans. if output is unchanged when the input does that trans.

- e.g. $f =$ label, $g =$ translation, scale, rotation. Regardless of transformation, label remains the same.
- Useful for classification tasks where transformations should not change the label.

Transformation Type	Description
Equivariance ($f(g(x)) = g(f(x))$)	Equivariant to a trans. if output changes in the same way as input.

- e.g. $f =$ position, $g =$ translation. If f is applied first and then g, the output changes in the same way as applying g first and then f.
- Useful in tasks where spatial relationships need to be preserved, such as object localization.

Data Type	Symmetry
Tabular Data	Row permutation invariance

- Ordering of rows does not affect the output.

Data Type	Symmetry
Sets	Element permutation invariance

- Elements have no inherent order, so the output should not change if elements are swapped.

Data Type	Symmetry
Images	Translation, rotation, and scaling invariance

- Image recognition should not be affected by the translation, rotation, or scaling of the image.

Data Type	Symmetry
Time-series	Time shift invariance

- Patterns should be the same regardless of when they occur.
- **Properties:**
  - Causal: Future data should not affect past data.
  - Non-Stationarity: Data distribution changes over time.
  - Trend and Seasonality: Data has a trend and seasonal patterns.
- **Evaluation performance:** Highly context dependent
  - Fixed future test set: Fixed time horizon.
  - Rolling test set: Sliding window.

Data Type	Symmetry
Graphs	Node permutation invariance

- Nodes can be rearranged without changing the graph's structure since the edges remain the same.

Data Type	Symmetry
Text	Sentence structure and paraphrasing invariance

- Rewording a sentence or changing its structure should not change its meaning.

9.2 Learning on Tabular Data

Notes:
- **Problem:** DL struggles with tabular data because it lacks the spatial and sequential structures found in images and time-series, making it difficult for NNs to extract meaningful patterns.
- **Soln:** XGBoost (tree-ensemble method):
  - Automatic feature selection.
  - Mixed data types.
  - Robust to outliers.
  - Capture nonlinear relationships.
  - Computationally efficient and fast.
  - Easy to set up.

9.3 Learning on Sets

Summary: Unordered collections of distinct/unique elements

Concepts	Description
Data Sets	Each data point is i.i.d. R.V. with no inherent order. • Points are indep. • Summing over loss fn is invariant to ordering of elements. • Unbiased estimate of the loss via stochastic subsampling.
Permutation Invariance	Output should not change if elements are permuted. • L9 Slide 23,26: NNs. • Deep Sets: NNs that are permutation invariant to the input set.
Pooling	Core operation for sets to summarie info across elements. • e.g. sum, mean, var/std, max, min, count, distribution statistics
Inductive Biases	Prior knowledge that can bias the learning process. • **Examples:** – Max pooling ignores all values except the maximum, which may lose important information. – Mean pooling blurs distinctions between large and small values, leading to loss of contrast. – Sum pooling can be sensitive to the number of elements, making it less robust to input size variations. • **Soln:** Principal Aggregation – Take many different types of aggregations and concatenate them.
Learning On Sets	DeepSets (learning/element) $\sum_{i}^{\text{Set}} f(e_i)$ Self-Attention (pairwise interaction) $\sum_{i,j}^{\text{Set}} f(e_i, e_j)$

10 CNN

Summary:

Concept	Description
2D Convolution Operation	Equivariant Linear Layer that applies a sliding, weighted sum across input $$F(x)_{i,j} = \sum_{a}^{K_h} \sum_{b}^{K_w} w_{a,b} x_{i+a,j+b}$$

- $F(x)_{i,j}$: Output at position (i, j), $w_{a,b}$: Weights
- $x_{i+a,j+b}$: Input at position $(i + a, j + b)$, K_h, K_w: Height and Width of Kernel

$$F(x)_{i,j} = \sum_{a}^{K_h} \sum_{b}^{K_w} \sum_{\text{out}}^{F} w_{\text{in},a,b,\text{out}} x_{i+a,j+b,c} \quad \text{in} \in [\text{Input}]$$

- F: Number of filters, c: Number of channels
- $w_{\text{in},a,b,\text{out}}$: Weights for input channel *in* and output filter *out*
- $x_{i+a,j+b,c}$: Input at position $(i + a, j + b)$ in channel c

Receptive Field	Defines the region of input visible to a neuron

- Determined by kernel size, stride, and network depth.
- Larger receptive fields capture more context.

Padding (Managing Boundary Effects)	Adds extra values around the inpjut to control output size

- Valid Padding: No padding, output smaller than input.
- Same Padding: Output size same as input.
- Full Padding: Output size larger than input.
- Zero Padding: Add zeros around input.

Stride and Downsampling	Larger strides result in greater downsampling of the input.

- Larger strides increase downsampling.
- **Pros:** Reduces computational cost.
- **Cons:** Can lead to info loss.

Dilated convolutions	Increase receptive field without increasing parameters

- Dilation rate: Spacing b/w kernel elements.
- **Pros:**
  - Increases receptive field. Without increasing parameters.
  - Useful for capturing long-range dependencies.

Pooling	Reduce spatial dimensions while preserving key features

- **Pros:**
  - Provides translation invariance.
  - Summarizes features in a local region.
  - Less sensitive to feature location.

Summary:

1D Convolution Operation	Leveraging spatial feature extraction for **sequential data** $F(x_i) = \sum_a^K w_a x_{i+a}$ $(x \star w)(i) = \sum_a^K k_a x_{i+a}$
Causal Convolutions + Causal Padding	Prevents future data influencing in convolutions

- Causal Convolutions: Only use past data in convolutions.
- Causal Padding: Pad input with zeros to left of the sequence to prevent future data influencing output.
- e.g. WaveNet: Dilated Causal Convolutions
  - Captures long-range dependencies.

Learning Optimal Filters	Learn the kernel weights automatically from data.
Convolution as Cross-Correlation	Convolution is a cross-correlation with flipped kernel. $(f \star g)_{i,j} = \sum_a^{K_h} \sum_b^{K_w} f_{i+a,j+b} g_{a,b}$

- $f_{i,j}$: Pixel at (i, j), $g_{a,b}$: Kernel at (a, b)

11 RNN

Summary:

Concept	**Description**
Gating Mechanisms: Controlling Information Flow	Enabling selective filtering and modulation of activations $\text{out} = \text{sigmoid}(\text{Linear})(z) \odot x$ or $\tanh(\text{Linear})(z) \odot x$

- $\text{sigmoid}(\text{Linear})(z)$: Acts as a filter, determining how much information should be retained or forgotten.
  - 0: Forget, 1: Retain
- $\tanh(\text{Linear})(z)$: Scales values between -1 and 1, maintaining zero-centered activations and helping control the magnitude of updates.

Gating: Feature-wise Modulation	Feature-wise input Linear Modulation relies heavily on gating $\text{FiLM}(x, z) = \gamma(z) \odot x + \beta(z) \quad \gamma, \beta = \text{MLPs}(z)$

- **Applications:** Style transfer, context conditioning, out-of domain adaptation.

Unrolled in Time	Visualizing the iterative processing of sequential input.

- Notice that when optimizing a RNN, we are backpropagating through time.

Challenges	
Vanishing/Exploding Gradients	Challenges in RNN training due to gradient flow

- Since we only have a single learnable, parametrized operation that we apply iteratively across time, gradients can either vanish or explode.
  - e.g. $w * w * w \ldots w$ can either vanish or explode.
- **Solutions:** Gradient clipping, gating mechanisms, LSTM/GRU cells, normalization.

Long-Term Dependencies	RNNs struggle to retain information over extended sequences

- RNNs tend to favour recent information.
- For long-range information to propagate, it needs to survive multiple iterations.

11.1 Approximations for Sequence Data

Summary:

Approximations	**Equation**
Markov Property (Conditional Independence):	$P(X_t \mid X_{t-1}, X_{t-2}, \ldots, X_1) = P(X_t \mid X_{t-1})$

- Simplifies the joint distribution of a sequence to a product of conditional probabilities.

Weak Stationary (Mean):	$E[X_t] = \mu \; \forall t$

- The mean of the sequence is constant over time.

Weak Stationary (Covariance):	$\text{Cov}(X_t, X_{t+\tau}) = \gamma(\tau) \; \forall t$

- The covariance between two points in the sequence is constant over time.

12 GNN