

## 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:**

Algorithm	Inputs	Outputs	Equations
RNN	$x_t, h_{t-1}$	$y_t, h_t$	$h_t = \tanh(\text{Linear}(h_{t-1}) + \text{Linear}(x_t))$ $y_t = \text{MLP}(h_t)$ <ul style="list-style-type: none"> <li><math>x_t</math>: Input, <math>h_t</math>: Hidden state, <math>y_t</math>: Output</li> </ul>
GRU	$x_t, h_{t-1}$	$y_t, h_t$	$z_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$ $r_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$ $\tilde{h}(t) = \tanh(\text{Linear}(x_t) + \text{Linear}(r_t \odot h_{t-1}))$ $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$ <ul style="list-style-type: none"> <li><math>z_t</math>: Update gate, <math>r_t</math>: Reset gate</li> <li><math>h_t</math>: Hidden state</li> </ul>
LSTM	$x_t, h_{t-1}$	$h_t, c_t$	$f_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$ $i_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$ $o_t = \text{sigmoid}(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$ $\tilde{c}_t = \tanh(\text{Linear}(x_t) + \text{Linear}(h_{t-1}))$ $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ $h_t = o_t \odot \tanh(c_t)$ <ul style="list-style-type: none"> <li><math>f_t</math>: Forget gate, <math>i_t</math>: Input/Update gate, <math>o_t</math>: Output gate</li> <li><math>h_t</math>: Hidden state, <math>c_t</math>: Cell state</li> </ul>

### 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	
<ul style="list-style-type: none"> <li>Data, code (scripts / modules), models (weights, configurations), results, predictions, plots, meeting notes, papers, documentation, etc.</li> </ul>	
Git (Version Control): Enables effective tracking and collaborative changes <ul style="list-style-type: none"> <li>Tracking changes, collaboration, backup, revert.</li> <li>Add, commit -m "message", push, pull, merge, diff, revert, branch, checkout, log, status, etc.</li> <li>.gitignore: Ignore files, directories, or patterns</li> </ul>	<a href="#">Git Guide</a>
GitHub (Collaborative Code Hosting): Facilitates sharing and collaboration on code <ul style="list-style-type: none"> <li>Collaboration, sharing, open science, project hosting.</li> </ul>	<a href="#">GitHub</a>
Cookiecutter (Project Template): Standardizes project structure <ul style="list-style-type: none"> <li>Logical, flexible, and reasonably standardized project structure for doing and sharing data science work.</li> <li>File Structure               <ul style="list-style-type: none"> <li>data/:                   <ul style="list-style-type: none"> <li>external/: Data from third party sources.</li> <li>interim/: Intermediate data that has been transformed.</li> <li>processed/: The final, canonical data sets for modeling.</li> <li>raw/: The original, immutable data dump.</li> </ul> </li> <li>src/: Source code for use in this project.                   <ul style="list-style-type: none"> <li>__init__.py: Makes src a Python module.</li> <li>config.py: Configuration settings.</li> <li>dataset.py: Code to load data.</li> <li>features.py: Code to build features.</li> <li>modeling/: Code to train models.                       <ul style="list-style-type: none"> <li>__init__.py: Makes modeling a Python module.</li> <li>predict.py: Code to make predictions.</li> <li>train.py: Code to train models.</li> </ul> </li> <li>plots.py: Code to create plots.</li> </ul> </li> <li>docs/: Documentation for this project.</li> <li>models/: Trained and serialized models, model predictions, or model summaries.</li> <li>notebooks/: Jupyter notebooks.</li> <li>references/: Data dictionaries, manuals, and all other explanatory materials.</li> <li>reports/: Generated analysis as HTML, PDF, LaTeX, etc.                   <ul style="list-style-type: none"> <li>figures/: Generated graphics and figures to be used in reporting.</li> </ul> </li> <li>pyproject.toml: Project information and dependencies.</li> <li>requirements.txt: The requirements file for reproducing the analysis environment.</li> <li>setup.cfg: Configuration file for setting up the project.</li> <li>LICENSE:</li> <li>Makefile:</li> <li>README.md:</li> </ul> </li> </ul>	<a href="#">Cookiecutter Repo</a>

**Summary:**

Tools	Links
Cookiecutter (Project Template): Standardizes project structure	
<ul style="list-style-type: none"> <li>• <b>Design Philosophy:</b> Prioritizes conventions and reasonable defaults to streamline project setup. Opinions: <ul style="list-style-type: none"> <li>– Data analysis is a DAG: <ol style="list-style-type: none"> <li>1. Raw data</li> <li>2. Compute features</li> <li>3. Plot analysis on raw data</li> <li>4. Train model</li> <li>5. Compute statistics on features</li> </ol> </li> <li>– Raw data is immutable (i.e. never change raw data) <ul style="list-style-type: none"> <li>* <b>Dos:</b> <ul style="list-style-type: none"> <li>· Pipeline code: Process raw data → final analysis.</li> <li>· Cache outputs: Serialize or cache intermediate steps.</li> <li>· Reproducible results: Enable full reproduction from code and raw data only.</li> </ul> </li> <li>* <b>Don'ts:</b> <ul style="list-style-type: none"> <li>· Never edit raw data: Avoid manual edits or format changes</li> <li>· Never overwrite raw data: Do not replace raw data with processed data.</li> <li>· Single raw data version: Maintain only one version of raw data.</li> </ul> </li> </ul> </li> <li>– Data should (mostly) not be kept in source control <ul style="list-style-type: none"> <li>* GitHub warns for files over 50MB and rejects files over 100MB.</li> <li>* Use s3, azcopy, gcloud, drive to store data (i.e. cloud services)</li> <li>* Use cloudpathlib to access cloud data in the same way as pathlib to access local data.</li> </ul> </li> <li>– Notebooks are for exploration and communication, source files are for repetitions.</li> <li>– Refactor the good parts into source code (<b>Refactor Example</b>). <ul style="list-style-type: none"> <li>* Don't write code to do the same task in multiple notebooks.</li> </ul> </li> <li>– Keep your modelling organized (<b>PyTorch Example</b>) <ul style="list-style-type: none"> <li>* Predictions (csv), training log (csv), stats (txt), model config / hyperparameters (json)</li> </ul> </li> <li>– Build from the environment up (<b>Mamba</b>) <ul style="list-style-type: none"> <li>* Use mamba rather than conda for faster environment management.</li> <li>* Create a environment.yml file to manage dependencies.</li> </ul> </li> </ul> </li> </ul>	

## 8.3 Writing / Analyzing

### Summary:

Exercise	Links
Writing Skeleton	<a href="#">Step-by-Step Guide to Undergraduate Writing</a> <a href="#">How to write a first-class paper (Nature)</a> <a href="#">Preparing Manuscript: Scientific Writing for Publication</a>
<ol style="list-style-type: none"> <li>1. Start w/ Figures (How would you want to tell the story?)</li> <li>2. Write the structure (Introduction, Methods, Experiments and Results, Discussion)</li> <li>3. 2-3 sentence pitch for your idea</li> <li>4. Bullet points inside of each section (What are you expecting to cover?)</li> <li>5. Fill in text, repeat.</li> </ol>	
Figures: Move quick, perfect later <ul style="list-style-type: none"> <li>• Figure #1: Tells problem in simple way (30s elevator pitch)</li> <li>• Figure #2-3: Conceptual or data centric               <ul style="list-style-type: none"> <li>– How are you solving the problem?</li> <li>– What does the data look like?</li> </ul> </li> <li>• Figure #4-8: Quantitative evidence</li> <li>• <b>Recommendations:</b> <ul style="list-style-type: none"> <li>– Napkin/whiteboard figures first</li> <li>– Make good enough version w/ code (svg, png) using matplotlib, seaborn, etc.</li> <li>– Finetune w/ Inkscape, Illustrator, GIMP, etc</li> </ul> </li> <li>• <b>Anatomy of a Figure Examples (L8):</b> <ul style="list-style-type: none"> <li>– Slide 44: Task, Slide 45: Model + EDA</li> <li>– Slide 46-47: Quantitative evidence, Slide 48: Different ways of telling same story (e.g. tables or plots)</li> </ul> </li> </ul>	
Pick good, consistent colors	<a href="#">ColorBrewer</a> , <a href="#">Matplotlib Colormaps</a> <a href="#">Seaborn Palettes</a> , <a href="#">NeurIPS 18 Visualization for ML tutorial</a>
<ul style="list-style-type: none"> <li>• Be mindful of how colours can help tell a story, accessibility is also important.</li> </ul>	
Pair-writing (Como Pair-Coding)	<a href="#">Rubber Duck Debugging</a> , <a href="#">Pair Programming</a> <a href="#">Pair Writing</a> , <a href="#">Pair Writing in Government</a>
<ul style="list-style-type: none"> <li>• Working together → Help communicate thoughts and put you in a diff. attitude.</li> </ul>	
Communal writing (Social pressure → accountable)	<a href="#">Harvard Writing Center</a>
<ol style="list-style-type: none"> <li>1. Setup an objective, measurable goal.</li> <li>2. Set a time for writing period, take breaks</li> <li>3. Share progress at the end of each session, share writing struggles if needed.</li> <li>4. Reflect if there are some reasons why it is hard to write.</li> </ol>	
Writing: Focus on quick iterations	
<ul style="list-style-type: none"> <li>• Google docs and Paperpile (Copy DOI, paste, click, done) <math>\xRightarrow{\text{Export w/ bibtex}}</math> LaTeX (Overleaf)</li> </ul>	
Interactive Apps	<a href="#">Streamlit</a> , <a href="#">Gradio</a> <a href="#">Hugging Face</a> , <a href="#">Hugging Face Spaces</a> <a href="#">Academic Project Page Template</a>

## 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. <ul style="list-style-type: none"> <li>e.g. <math>f</math> = label, <math>g</math> = translation, scale, rotation. Regardless of transformation, label remains the same.</li> <li>Useful for classification tasks where transformations should not change the label.</li> </ul>
Equivariance ( $f(g(x)) = g(f(x))$ )	Equivariant to a trans. if output changes in the same way as input. <ul style="list-style-type: none"> <li>e.g. <math>f</math> = position, <math>g</math> = translation. If <math>f</math> is applied first and then <math>g</math>, the output changes in the same way as applying <math>g</math> first and then <math>f</math>.</li> <li>Useful in tasks where spatial relationships need to be preserved, such as object localization.</li> </ul>
Data Type	Symmetry
Tabular Data	Row permutation invariance <ul style="list-style-type: none"> <li>Ordering of rows does not affect the output.</li> </ul>
Sets	Element permutation invariance <ul style="list-style-type: none"> <li>Elements have no inherent order, so the output should not change if elements are swapped.</li> </ul>
Images	Translation, rotation, and scaling invariance <ul style="list-style-type: none"> <li>Image recognition should not be affected by the translation, rotation, or scaling of the image.</li> </ul>
Time-series	Time shift invariance <ul style="list-style-type: none"> <li>Patterns should be the same regardless of when they occur.</li> </ul>
Graphs	Node permutation invariance <ul style="list-style-type: none"> <li>Nodes can be rearranged without changing the graph's structure since the edges remain the same.</li> </ul>
Text	Sentence structure and paraphrasing invariance <ul style="list-style-type: none"> <li>Rewording a sentence or changing its structure should not change its meaning.</li> </ul>

## 9.2 Learning on Tabular Data

## 9.3 Learning on Sets