Instructions: Please read the following instructions thoroughly

- For the entire assignment, use `Python` for your analysis. Write your code in a Jupyter Notebook named as `[your-student-ID]_hw3.ipynb` (e.g., `2025-20000_hw3.ipynb`). The use of `R` is not allowed. You are allowed to use any libraries in `Python`.
- Type up your report and save as PDF named as `[your-student-ID]_hw3.pdf`. We do not allow the submission of a photo or a scanned copy of hand-written reports.
- Please upload the two files on eTL **without zipping**. Submissions via email are not allowed. The violation of the filename or submission instruction will result in the penalty of 5 points.
- You can discuss the assignment with your classmates but each student must write up his or her own solution and write their own code. Explicitly mention your classmate(s) you discussed with or reference you used (e.g., website, Github repo) if there is any. If we detect a copied code without reference, it will be treated as a serious violation of the student code of conduct.
- We will apply a grace period of late submissions with a delay of each hour increment being discounted by 5% after the deadline (i.e., 1-minute to 1-hour delay: 95% of the graded score, 1 to 2-hour delay: 90% of the graded score, 2 to 3-hour delay: 85%, so on). Hence, if you submit after 20 hours post-deadline, you will receive 0 points. No excuses for this policy, so please make sure to submit in time.

1. [**50 pts**] In this part, you will use the `CrabAge.csv` dataset and the target variable will be the age. You will implement and train fully connected networks using only `numpy`—no deep-learning frameworks (i.e., you cannot use `PyTorch`, `TensorFlow`, or any other similar framework for this part). You must implement both the forward and backward passes manually.

   (a) **[15 pts] Shallow Network (One Hidden Layer).**
   Consider a network with one hidden layer with 3 hidden units:

   $$\mathbf{h}_1 = \mathbf{a}(\boldsymbol{\Omega}_0\mathbf{x} + \boldsymbol{\beta}_0), \qquad \hat{\mathbf{y}} = \boldsymbol{\Omega}_1\mathbf{h}_1 + \boldsymbol{\beta}_1,$$

   where $\mathbf{a}(\cdot)$ is the ReLU activation.

   - Implement the forward pass using vectorized `numpy` operations.
   - Define the mean-squared-error (MSE) loss $L = \frac{1}{N}\sum_i \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|_2^2$.
   - Derive and implement backpropagation manually, computing all partial derivatives: $\frac{\partial L}{\partial \boldsymbol{\Omega}_0}, \frac{\partial L}{\partial \boldsymbol{\beta}_0}, \frac{\partial L}{\partial \boldsymbol{\Omega}_1}, \frac{\partial L}{\partial \boldsymbol{\beta}_1}$.
   - Train the model using stochastic gradient descent (SGD) on the training data.
   - Plot training loss vs. epochs.

   (b) **[25 pts] Deep Network (Two Hidden Layers).**
   Now consider a two-hidden-layer network with 3 hidden units in each layer:

   $$\mathbf{h}_1 = \mathbf{a}(\boldsymbol{\Omega}_0\mathbf{x} + \boldsymbol{\beta}_0), \qquad \mathbf{h}_2 = \mathbf{a}(\boldsymbol{\Omega}_1\mathbf{h}_1 + \boldsymbol{\beta}_1), \qquad \hat{\mathbf{y}} = \boldsymbol{\Omega}_2\mathbf{h}_2 + \boldsymbol{\beta}_2.$$

- Write down the explicit forward and backward equations, showing how the chain rule propagates through layers.
- Implement both forward and backward passes in `numpy`.
- Train your network using your own backpropagation and SGD implementation.
- Plot training loss vs. epochs.

(c) **[10 pts]** Using the two models, shallow and deep networks, compare results.

- On a separate validation dataset, compute validation losses for the shallow and deep networks trained in (a) and (b)
- Discuss the impact of depth on convergence behavior on training and test prediction error.

2. [**50 pts**] In this part, you will use the `HousePrice.csv` dataset and the target variable will be the house price. You may use `PyTorch` to train larger and deeper networks. Compare the optimization performance of gradient descent (GD), SGD, and Adam on the same architecture.

(a) **[15 pts] Building a Deep Network.**
Construct a network with three hidden layers, each of width 64 (i.e., 64 hidden units in each layer), using ReLU activation:

$$\mathbf{h}_1 = \mathbf{a}(\boldsymbol{\Omega}_0\mathbf{x} + \boldsymbol{\beta}_0), \quad \mathbf{h}_2 = \mathbf{a}(\boldsymbol{\Omega}_1\mathbf{h}_1 + \boldsymbol{\beta}_1), \quad \mathbf{h}_3 = \mathbf{a}(\boldsymbol{\Omega}_2\mathbf{h}_2 + \boldsymbol{\beta}_2), \quad \hat{\mathbf{y}} = \boldsymbol{\Omega}_3\mathbf{h}_3 + \boldsymbol{\beta}_3.$$

Train with vanilla SGD (batch size 1). Report training and validation losses vs. epochs.

(b) **[25 pts] Comparison of Optimizers.**
Train the same network under three optimizers:

(i) Full-batch GD,
(ii) Mini-batch SGD (batch size 32),
(iii) Adam (with suitable tuned β_1, β_2 parameters).

- Keep initialization and learning rates consistent across optimizers.
- Plot training and validation loss curves for all three optimizers.
- Compare convergence rate (over epoch) and final performance.

(c) **[10 pts] Training with Regularization.**
Incorporate *explicit regularization* into training (Lecture 19). Add an L2 penalty on the network parameters to the loss function with $\lambda > 0$:

$$\tilde{L} = L[\{\mathbf{x}_i, \mathbf{y}_i\}] + \lambda \sum_k \|\boldsymbol{\Omega}_k\|_F^2,$$

- Train the same network with $\lambda \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$, using Adam as the optimizer.
- Plot training and validation losses for each λ.
- Discuss how increasing λ affects overfitting and weight magnitudes (connect to the "weight decay" interpretation of L2 regularization).