NOTE: REPORT [5 pages max] only on part 2 [mainly on labTS, little weight on report]

Requirements:

* Create NN mini-library (**use numpy**) [data pre-processing, training and evaluation and backprop] & create and train NN for regression [PREDICT HOUSE PRICES]. **Pytorch** or **mini library**
* Mini lib: Linear layer class, activation functions, multi-layer network class, trainer class data preprocessing [os, sys and math for differentiation]
* Second part for report [model, evaluation setup, hyperparameter tuning and final results]

Part 1:

* Linear layer: **\_W [standard normal] and \_b [0]** are learnable parameters [initialise them].
  + FP method: output of input numpy array [batch of inputs]. Store data for grad computation [backprop] 🡪 use \_**cache\_current attribute {store grads]**
  + BP: gradient of function wrt output of current layer [grad\_w\_current and grad\_b\_current store gradients]. From wrt outputs to wrt params and give wrt inputs
  + Parameter update: 1 step of grad descent on params of layer [stored grads and learning rate args]
  + **Text, letter

    Description automatically generated**
* Activation fn classes: Sigmoid and ReluLayer
  + FP: elementwise transformation of inputs, store data for grad computation
* Multilayer:
  + Stack linear layers and activation functions
  + Constructor: instances of linearlayer and activation classes [input\_dim [number in first linear layer, list of output neurons in each linear layer, activation functions to apply to output of each linear layer, store instances in layers attribute
  + FP: LinearLayer instances defined in constructor will handle storage of gradient data
  + BP: Storage of computed gradients again automatically handled by LinearClass list attribute
  + Param update:
* Trainer:
  + Data shuffling, training using minibatch grad descent, and compute loss on validation
  + Loss\_fun argument [mse or cross entropy]
  + Data shuffling [return random re-ordering]
  + Training loop: shuffle\_flag, split dataset into minibatches of size batch\_size
  + Evaluation loss: eval\_loss method [compute and return loss on provided evaluation dataset]
* Preprocessor:
  + Normalise data [min\_max scaling, and lie in interval [0, 1]]
  + Constructor: Compute and store data norm params
  + Apply: return pre processed version
  + prep.revert(prep.apply(A)) returns A

Part 2:

* Data specific: numerical and textual info + missing values [handle such data in both housing and unseen datasets]. Pandas and scikit-learn for preprocessor method