Homework 6

1. **Fitting quadratic losses with non-quadratic regularizers to data.** In `non_quad_regs.json`, you will find a $50 \times 30$ matrix $U_{\text{train}}$ and a 50-vector $v_{\text{train}}$ consisting of raw training input and output data, and a $50 \times 30$ matrix $U_{\text{test}}$ and a 50-vector $v_{\text{test}}$ consisting of raw test input and output data, respectively. We will work with input and output embeddings $x = \phi(u) = u$ and $y = \psi(v) = v$. Our performance metric is the RMS error on the test data set.

In `regression_fit.jl` we have also provided you with a function

\[
\text{regression_fit}(X, Y, l, r, \lambda) \n\]

This function takes in input/output data $X$ and $Y$, a loss function $l(\hat{y}, y)$, a local regularizer function $r(\theta)$, and a local regularization hyper-parameter $\lambda$. It outputs the model parameters $\theta$ for the RERM linear predictor. You must include the `Flux` and `LinearAlgebra` Julia packages in your code in order to utilize this function. You will use this function to fit a linear predictor to the given data using a quadratic loss function, and regularizers corresponding to the penalty functions $q : \mathbb{R} \to \mathbb{R}$ below.

- No regularization: $q(a) = 0$.
- Quadratic: $q(a) = a^2$.
- Absolute value: $q(a) = |a|$.
- $q(a) = |a|^{1/2}$.

For each predictor, plot the training and test RMS errors on the same plot, for regularization parameters taking 20 values logarithmically spaced between $10^{-3}$ and $10^1$. (Thus, there are four figures in total, one for each predictor, with two plots on each figure.) Which model performs best? Create a one-sentence conjecture or story about why the particular model was the best one.

*Note.* The last penalty corresponds to a regularizer that some call the $\ell_{0.5}$ regularizer, but this is not correct; the corresponding regularizer is not a norm.

*Hint.* The training and test RMS error plots corresponding to $q(a) = 0$ will be constant, as $\lambda q(a) = q(a) = 0$ for all $\lambda$.

*Julia hint.* You will need to define the regularizer functions in Julia. You can do this in a compact (but readable) form by defining the function inline, for example, for the quadratic regularizer, $r(\theta) = \text{sum}(\theta.^2)$. For the quadratic loss function, use $l(\hat{y}, y) = (\hat{y} - y).^2$.

2. **Fitting neural networks with Julia.** In `nn_regression.json`, you will find a $4000 \times 30$ matrix $U_{\text{train}}$ and a 4000-vector $v_{\text{train}}$ consisting of raw training input and output
data, and a 1000 × 30 matrix \( U_{\text{test}} \) and a 1000-vector \( v_{\text{test}} \) consisting of raw test input and output data, respectively. We will work with input and output embeddings \( x = \phi(u) = u \) and \( y = \psi(v) = v \).

In `nn_regression.jl` we have also provided you with a function

\[
\text{nn\_regression}(X, Y, \lambda).
\]

This function takes in input/output data \( X \) and \( Y \), and a local regularization hyper-parameter \( \lambda \). It outputs the model parameters \( \theta \) for a neural network with parameters and activation functions defined in the code. You must include the Flux Julia package in your code in order to utilize this function.

(a) Use linear regression (without regularization) to fit a linear predictor to the training data. Report the training and test RMS errors.

(b) Inspect the `nn\_regression` function in `nn\_regression.jl`. What is the form of the neural network model in the code (i.e., the form of \( \hat{y} \))? Specify the number of layers; for all layers, specify the activation function and model parameters (including dimensions). In total, many scalar entries of the model parameters are there? (For example, \( A \in \mathbb{R}^{13 \times 10} \) has \( 13 \times 10 = 130 \) scalar entries.)

(c) Using `nn\_regression.jl`, fit a neural network to the training data for regularization parameters taking 10 values logarithmically spaced between \( 10^{-3} \) and \( 10^0 \). Report the best test RMS error and the corresponding regularization parameter.

Remark. It is normal for training this neural network to take several minutes.

Hint. The code in `nn\_regression.jl` randomly initializes the model parameters; in order to generate reproducible results on your end, we suggest you use the `seed` function from the `Random` package with a seed of your choice, e.g., `Random.seed(0)`.