Homework 6

1. **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 $1000 \times 30$ 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

$$
nn\_\text{regression}(X, Y, \lambda).$$

This function takes in input/output data $X$ and $Y$, and a local regularization hyperparameter $\lambda$. 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, how many scalar 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)`.

**Hint:** Use `lam` $\text{d} = 10 \cdot \text{range}(-3, 0, \text{length}=10)$ to compute the $\lambda$ values.

**Hint:** Call `predictall(model, U)` to compute a vector of $\hat{y}$.

2. **Multi-class animal classification.** Our task is to create a predictor which identifies the class type of the animal based on 16 traits: HAIR, FEATHERS, EGGS, MILK, AIRBORNE, AQUATIC, PREDATOR, TOOTHED, BACKBONE, BREATHES, VENOMOUS, FINS, LEGS, TAIL, DOMESTIC, CATSIZE. All traits except LEGS (Numeric) are Boolean.

There are 7 classes of animals. (Unfortunately the names of the animals are not included in the data file, only their numbers.)

In `zoo.json`, you will find an $n \times d$ matrix $U$ of data, with rows $(u^i)^T$, with $u^i \in \mathbb{R}^{16}$, and $n$-vector $v$ with the class type of the animal.
We have provided a function

```julia
nn_multiclass_classification(X, Y, n_classes)
```

You can find this in `nn_multiclass_classification.jl`. Be aware that instead of passing a lambda value as you did in `nn_regression`, in this function you must pass the number of classes.

(a) Inspect `nn_multiclass_classification.jl`. What is the form of the neural network model? Specify the number of layers: for all layers, specify the activation function and model parameters (including dimensions). In total, how many scalar model parameters are three?

(b) Convert the `v` vector provided in `zoo.json` to one-hot format. You do not need to do any processing on the `U` matrix (although you are welcome to experiment). Using the provided function, train a classifier on a randomly selected 90% of the data. Report the accuracy for the training set and test set. Compute and report the confusion matrix for the training and test set.

*Hint:* To convert a scalar `vi` with 7 possible values to one-hot format, use `onehot(vi, 1:7)`. To convert a vector `v` to one-hot format, use `onehotbatch(y, 1:7)`. Run this in the Julia REPL and make sure you understand what dimensions the result takes.

*Hint:* There isn’t that much data to train with, so don’t expect perfect accuracy. 75% is fine. You may see varied results due to the random train-test split: if this worries you, use `Random.seed!(0)` to get reproducible results.

*Hint:* You can display a matrix `X` in a Jupyter notebook by calling `display(X)`

(c) In this problem, we provided your neural network architecture. However, when solving a problem on your own you will need to decide how many layers to use, and how many parameters to have in each layer. If you have too many parameters and not enough data, it will not be possible to train the network with your dataset. However, if you have too few layers or parameters, it will not be possible to learn complex features of your dataset. We’ll investigate two changes to this network. First, try adding another Dense layer. (You can just insert one in the middle of the `Chain`: the size and activation function is up to you.) Describe the change(s) you made. Retrain your network and report the train and test accuracy. How would you explain this performance?

*Hint:* If you use Jupyter notebooks, don’t forget to reload your notebook after modifying `nn_multiclass_classification.jl`.

(d) Remove the layer you added in Part (c). Now we will try making the network smaller. You can change the number of parameters, delete a layer, or both. Describe the change(s) you made. Retrain your network and report the train and test accuracy. Provide a (brief) comment on the results.