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

$$\text{nn_regression}(X, Y, \lambda).$$

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, 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 lambdas = 10 .^ range(-3,0,length=10) to compute the $\lambda$ values.

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

**Solution.**

(a) The test RMS error is 5.82.

(b) The form of the model is

$$\hat{y} = g^4(g^3(g^2(g^1(x)))),$$

where $g^1 : \mathbb{R}^{30} \to \mathbb{R}^{10}$, $g^2 : \mathbb{R}^{10} \to \mathbb{R}^{10}$, $g^3 : \mathbb{R}^{10} \to \mathbb{R}^{10}$, and $g^4 : \mathbb{R}^{10} \to \mathbb{R}$. $g^1$, $g^2$, and $g^3$ all have ReLu activation functions, while $g^4$ has an identity activation function. There are thus $(31 \times 10) + (11 \times 10) + (11 \times 10) + (11 \times 1) = 541$ scalar parameters.
(c) The best test RMS error was 1.05 for $\lambda = 0.004$.

```plaintext
println("Importing packages....")

using LinearAlgebra
# using Statistics
using Random
using Flux

include("readclassjson.jl")
include("writeclassjson.jl")
# include("regression_fit.jl")
include("nn_regression.jl")

import PyCall
import PyPlot; const plt = PyPlot; plt.plt.style.use("seaborn")

# Random.seed!(0)

# println("Making data....")

# n = 5000
# ntrain=4000
# d = 30

# X = randn(n,d)
# y = [ 0.2 * sum(X[i, 10:15].^2) + log(sum(abs.(X[i,1:30]))) + X[i,2]*X[i,1].^2 + 1.5*sin(X[i,4]) + 0.2*sqrt(abs(X[i,3])) + 0.3*X[i,8] + 0.42*X[i,7] + 2*rand() for i=1:n]

# global X_train = X[1:ntrain, :]
# X_test = X[ntrain+1:n, :]

# global y_train = y[1:ntrain]
# y_test = y[ntrain+1:n]

# # outlier_idxs_train = shuffle(1:500)[1:50]
# # y_train[outlier_idxs_train] = -y_train[outlier_idxs_train]

# # outlier_idxs_test = shuffle(501:1000)[1:100]
# # y_test[outlier_idxs_test] = -y_test[outlier_idxs_test]

# Data = Dict()
# Data["U_train"] = X_train
# Data["v_train"] = y_train
# Data["U_test"] = X_test
```
# Data
[v_test] = y_test
# writeclassjson(Data, "neural_net.json")

println("Loading data....")
Data = readclassjson("nn_regression.json")
X_train = Data["U_train"]
X_test = Data["U_test"]
y_train = Data["v_train"]
y_test = Data["v_test"]

################################

function RMSE(yhat, y)
    mse = sum((yhat .- y).^2)/length(y)
    return sqrt(mse)
end

nnz(a, tol) = sum(abs.(a) .<= tol)

################################

# predictall(model,X) = vcat([model(x) for x in eachrow(X)]...)
# train_error = rmse(predictall(model,Xtrain), Ytrain)

################################

function ridgeregression(X,Y,lambda)
    n,d = size(X)
    m = size(Y,2)
    A = [X; sqrt(lambda*n)*I(d)]
    B = [Y; zeros(d,m)]
    theta = A\B
end

println("linear regression...")
theta = X_train \ y_train
predictall(theta,X) = vcat([theta' * x for x in eachrow(X)]...)
println(RMSE(predictall(theta,X_test), y_test))

println("nn regression....")

lambdas = 10 .^ range(-3,0,length=10)
train_errors = []
test_errors = []
for in lambdas
    model = nn_regression(X_train, y_train, )
    predictall(model, X) = vcat([model(x) for x in eachrow(X)]...)
    train_error = RMSE(predictall(model, X_train), y_train)
    test_error = RMSE(predictall(model, X_test), y_test)
    push!(train_errors, train_error)
    push!(test_errors, test_error)
end
println(test_errors)
println("Min RMSE, lambda: ", minimum(test_errors), lambdas[argmin(test_errors)])

# plt.figure()
# plt.plot(lambdas, train_errors, color="blue", label="train")
# plt.legend()
# plt.tight_layout()
# plt.savefig("train.pdf", bbox_inches="tight")

# plt.figure()
# plt.plot(lambdas, test_errors, color="red", label="test")
# plt.legend()
# plt.tight_layout()
# plt.savefig("test.pdf", bbox_inches="tight")

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

```
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 $v_i$ with 7 possible values to one-hot format, use `onelhot(v, 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.

**Solution.**

(a) The form of the model is

$$\hat{y} = g^2(g^1(x)), $$

where $g^1 : \mathbb{R}^{16} \to \mathbb{R}^{32}$, and $g^3 : \mathbb{R}^{32} \to \mathbb{R}^{7}$. $g^1$ has a ReLu activation function, while $g^2$ has a sigmoid activation function. There are thus $(17 \times 32) + (33 \times 7) = 775$ scalar parameters.

(b) The accuracy we saw with different random seeds was between 75% and 91%. Anything around here is reasonable. Unfortunately due to the small dataset, this depends on the random train-test split.

(c) With one extra layer `Dense(2*d, 2*d)` added, the test accuracy we saw was 50%. The train accuracy was 64%. A reasonable explanation is that there are now too
many parameters to train on our small dataset. Another possible result is that it performs much worse on the test set than on the train set, indicating overfitting.

(d) With the intermediate layer size decreased to $d = 16$, the performance is comparable to Part (b). (Results may vary if the student did something else.) A reasonable comment might be that we don’t actually need the larger layer size, indicating the relationship between $U$ and $V$ is not that complex.
In [7]:
using Flux
using Flux: logitcrossentropy, normalise, onecold, onehotbatch
using Statistics: mean
using LinearAlgebra
using Random
Random.seed!(0)

include("readclassjson.jl")
include("nn_multiclass_classification.jl")

Out[7]: nn_multiclass_classification (generic function with 1 method)

---

Import, split, process data

In [ ]:

data = readclassjson("zoo.json")
X = data["X"]
y = data["y"]
n_animals = 7

N = size(X)[1]
indices = shuffle(1:N)
N_split = div(N,10)*9  # 90% split
println("Out of $(N) samples, $(N_split) used for training. $(N-N_split) used for test.")

X_train = X[indices[1:N_split],:]
X_test = X[indices[N_split:end],:]

y_onehot = onehotbatch(y, 1:n_animals)
y_onehot = y_onehot'  # make dimensions work out better

y_train = y_onehot[indices[1:N_split],:]
y_test = y_onehot[indices[N_split:end],:]

---

Define testing functions

In [107]:
# Translate output to labels (not one-hot)
get_labels(model, X) = [argmax(model(X[i,:])) for i=1:size(X)[1]]

# Accuracy Function
accuracy(X, y, model) = mean(get_labels(model, X) .== onecold(y'))

# Function to build confusion matrix
function confusion_matrix(X, y, model)
    ŷ = onehotbatch(get_labels(model, X_test), 1:7)
    ŷ * y_test
end

function test(model, X_test, y_test)
    # Testing model performance on test data
    accuracy_score = accuracy(X_test, y_test, model)
    println("Accuracy: $accuracy_score")

    # To avoid confusion, here is the definition of a Confusion Matrix: https://en.wikipedia.org/wiki/Confusion_matrix
    println("Confusion Matrix:")
    display(confusion_matrix(X_test, y_test, model))
end

Out[107]: test (generic function with 1 method)

Train and test the model

In [ ]:
model = nn_multiclass_classification(X_train, y_train, n_animals, numiter s=50)

In [109]:
test(model, X_test, y_test)

7×7 Matrix{Int64}:
  4  0  0  0  1  0  0
  0  4  0  0  0  0  0
  0  0  0  0  0  0  0
  0  0  1  0  0  0  0
  0  0  0  0  0  0  0
  0  0  0  0  1  0  0
  0  0  0  0  0  0  1

Accuracy: 0.9166666666666666

Confusion Matrix: