**Lab 10: Neural networks**

# Introduction

Neural networks are the new hot thing in machine learning. In this lab, we’ll look at two flavors of neural networks: echo-state networks and deep networks. Since neural networks involve some pretty sophisticated tools to implement, this lab will heavily utilize pre-existing code to demonstrate the tunable parameters and capabilities of neural network decoders.

# Software

This lab must be completed using MATLAB.

# Part 1) Chaos networks & FORCE learning

Echo-state networks are sparsely connected networks with normally distributed connection weights. The behavior of these networks depends largely on a parameter , that relates the variance of connection weights to the number of neurons in the network. FORCE learning can be used to train a random echo-state network to produce specific outputs. Here we examine an example of FORCE learning used to train a neural network to generate a sum of four sine waves.

1. Open force\_external\_feedback\_loop.m. It is modified version script from “Generating Coherent Patterns of Activity from Chaotic Neural Networks”, by David Sussillo and Larry Abbott (2009).
2. The script runs two ways, depending on the value of ‘trainTestdat’. When ‘trainTestdat’ is set to 1, the echo-state network is trained and tested on summed-sinusoidal data. When ‘trainTestdat’ is set to 0, the echo-state network runs on its own without training data, which can be useful to see if the network is chaotic or not (if it is not chaotic, the weight values will quickly converge to 0).
3. Run the script for values of between 0.9 and 1.5. Look at the training and testing error for each value of . These values are automatically reported by the script (look for MAE).
4. Describe the qualitative change that occurs at and provide plots of your training and testing results for one above and below 1 (2 plots with 2 subplots each). Also look at how the weights change for each of the 2 values without training (setting ‘trainTestdat’ to 0) and provide plots. Why is there a change at ?
5. With , run the code with 2000 neurons and 100 neurons. Report the training and test error for each (no plots). Note that since certain parameters are initialized pseudo-randomly, there will be some variance in performance across different iterations of the same code.
6. How small can the neural network get before getting more than ten times the test error of a 2000-neuron network? That is, what is the minimum number of neurons necessary to achieve this level of performance? Provide values for your neuron count and mean error. (no plots), remember that these scripts are not deterministic, so there will be some variance in performance for the same code.

# Part 2) Deep learning forward propagation

While echo-state networks are pseudo-randomly organized, deep networks can be represented as well-organized hierarchical networks. This allows each layer of the network to be trained in sequence, creating a hierarchy of data representations within the network. This type of organization resembles biological neural networks, such as the visual cortex. In this part, you will implement the forward propagation part of a deep learning neural network to classify neural data from a reach task.

Download run\_script.m, GetAllData.m, TrainAndTest.m, NeuroNetClassifier.m, and firingrate.mat to your working directory.

* run\_script.m executes the other scripts to train and test a neural network algorithm script that decodes firing rates into reach directions.
* GetAllData.m reads in data and formats it for use in the neural network algorithm.
* TrainAndTest.m creates, trains, and tests a neural network using 5-fold validation. The neural network created for this lab has two hidden layers with 95 inputs, 40 neurons in the first hidden layer, 20 layers in the second hidden layer, and 8 neurons at the output layer.
* NeuroNetClassifier.m contains a suite of functions for implementing a neural network.
* firingrate.mat contains the same reach data used in Lab 7. The 3D matrix contains firing rates from 95 units during an 8-direction reach task. Each reach direction has 182 trials.

1. Since training a neural network involves a lot of gradient descent, which is beyond the scope of this course, we will focus on implementation of forward propagation. At line 64 in NeuroNetClassifier.m, there is a skeleton for a forward propagation function, missing the components needed to perform forward propagation of the hidden layers. Your job is to complete this function, which is used throughout the script to train and test the neural network.

Some important hints:

* The flow of values from one layer to another follows this logic:

apply weights

and sum

output of layer k

OR

input of layer k+1

apply nonlinear

transform

z

Inputs to layer k

* If you’re not familiar with MATLAB structs and cell arrays, it might be helpful to read a bit about them. They are utilized heavily in this code.
* The input obj is a MATLAB struct variable that contains the neural network. You can access different properties of the neural network by accessing different parts of the obj struct:
  1. obj.num\_layers gives the number of layers in the neural network, not including the input layer.
  2. obj.dimensions\_each\_layer gives a vector containing the number of neurons in each layer of the neural network, including the input layer.
  3. obj.coefficients{k} gives a matrix of weights for connections between layers and . That is, it contains the values of for those two layers.
* The input input is a matrix, where is the number of inputs (the number of units in this example) and is the number of samples. Note that this means the function must be to handle multiple samples at once. may vary, so write your code in a way that’s robust to different sample sizes.
* The output result is a cell array containing the output values of each layer, where is equal to the value of obj.num\_layers. The first two cells should contain matrices that are and Don’t worry about the last cell, as its values are computed in a different way.
* In each iteration of the for loop, you need to do three things:
  1. Calculate the output of the hidden layer, given its inputs and connection weights. Use the sigmoid function described in lecture, so that the output to neuron is .
  2. Save the outputs to result.
  3. Pass the output on as the next layer’s input.

Note: After calculating the output from a given layer, add a row of ones to the bottom of the matrix before saving it to results and passing it on as the input to the next layer. This is necessary for the gradient descent algorithm.

Note: The last part of the function, provided in the skeleton, takes an input (called ‘input‘) that should come from the last hidden layer.

1. Once complete, run the algorithm and report its performance (training and testing accuracy).
2. Go into TrainAndTest.m and switch the number of neurons in the two hidden layers. What impact does this have on training and test accuracy? What happens if you remove either of the hidden layers? Explain your results.

# Part 3) Deep learning using Matlab’s Neural Network Toolbox

Part 2 is great way to understand the underlying mechanisms of neural networks. However, researchers do not usually write code for the entire neural network. In this part, you will play with a deep neural network using Matlab’s neural network toolbox. Because the neural network toolbox is new, it likely requires Matlab 2017a or later.

Side-note: If you are really interested in using neural networks or high-level machine learning algorithms, R and Python are the two programs that most data scientists use (some basic examples found here: <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>). For deep learning, Keras (<https://keras.io/>) seems to be a popular choice because it can run the popular Theano and Tensorflow packages and works in R and Python. In addition, Kaggle (<https://www.kaggle.com/>) has several datasets to practice on, along with many explanations of high-level machine learning topics applied to real data from some of the top data scientists in the world.

1. Open ‘DeepLearning\_autoenc.m’ and look through the code (you do not need to add any extra code to this). Briefly describe the overall structure of the deep neural network you are using here. What is this network doing?
2. Run the code once (shortcut key: F5). As the layers of the network are trained, a visualization should pop up with bars that move around. What do the bars labelled ‘epoch’ and ‘gradient’ represent? What determines when training is completed for each layer?
3. After the network finishes training, two other figures pop up. One is the confusion matrix, showing testing performance. The other one shows the autoencoder weights. Explain what these autoencoder weights mean for translating the original data to the final features. Draw some conclusion from the autoencoder weights. (Hint: Is there anything unusual with the second autoencoder weights?)
4. The code randomly selects a certain percentage of the spike data to be training data and a certain percentage to be test data. It should currently be set at 30%. Vary this number between 1% and 90% at several intervals. Plot the percentage of data used for training vs. the overall classification accuracy from the confusion matrix. When does performance deteriorate? What does this tell you about this deep net?
5. Now try running the deepnet with more layers (still using 30% of the data). First, try 2 autoencoders of size [20 10]. Then try 4 autoencoders of size [25 20 15 10]. What’s going on? Provide a plot similar to step 3 with the classification accuracy for percentages of training data between 1% to 90% (just pick 4 spread out points in this range for this plot to save time). Plot the results from both deepnets on the same graph.

# Part 4) Modifying a pre-defined convolutional neural network

Convolutional neural networks were inspired by the structure of visual cortex in animals, in that each neuron has a small receptive field that combines with other neurons to encompass the entire picture. Convolutional neural networks are very popular for image recognition.

If you have little training data, it can be difficult to construct a new neural network. Instead, it is common to fine-tune a pre-existing neural network that has already been trained, which is known as transfer learning. Transfer learning allows for faster training because we only have to retrain the layers at the end of the network. For this problem, you will be using a simple convolutional neural network that was trained on images of letters, and re-training it to classify digits.

1. Open ‘lab9convolutionalNeuralNet.m’. Load in the pre-trained network, and examine the network layers (use ‘net.Layers’). Briefly describe the network structure, starting from the image input (layer 1) to classification output (layer 7).
2. Layers 5-7 are used to classify the features into different classes for the letter data. Remove these 3 layers and replace them with a fully connected layer (set output size appropriately, and set the 'WeightLearnRateFactor' and 'BiasLearnRateFactor' to 20), a softmax layer, and a classification layer (in that order). Store these new layers in a separate variable from the original net and train the new network on the digit data using the transfer options (using the ‘trainNetwork’ function).
3. Now that you have trained your new network, let’s see how it does. Use the ‘classify’ function on your trained network and test set. Calculate the classification accuracy using the total correct test classifications divided by the total number of test classifications.
4. Find the line with the splitEachLabel function. This splits the data into training and test data, using the number in the second argument. It should be set to 0.5, which splits half the data into training and testing. Re-train and test your network using different percentage of the total data for training. Make a plot showing the classification accuracy (y-axis) for the percent of data used for training (x-axis). Include 5% and 90%, along with several steps in between (because the order is randomized, classification accuracy will fluctuate).
5. Matlab provides a way to visualize strongly active network layers, using the ‘deepDreamImage’ function. This can highlight the image features learned by the network. Run the code for deepDreamImage. (Set percent of data for training to 50% and make sure the neural network variable in deepDreamImage matches your trained network variable.) It will take a few minutes to run because it is looking at all 10 digits. Which features of the digits does the network seem to capture best? Which digits are easiest to see from the deepDreamImage results?

# Guidelines for Lab Report (on Labs 10 and 11 together)

*Introduction:* The introduction should be one paragraph long summarizing the motivation for developing the tools used in this lab and what they can be used for, along with a brief summary of everything you will show in this lab report.

*Methods:* From Lab 10, there should be methods paragraphs (and diagrams where necessary) on:

1. Descriptions of the algorithms used
2. Rationale for the development of each algorithm
3. How the algorithms were implemented

Include the code as an Appendix to your report. Cite sources for any values used in your models.

*Results:* You should include the following in your Results:

1. The behavior/structure of each algorithm
2. How different parameters effect the performance of each algorithm (especially for parts 1-3)
3. Plots of training/testing for one above and below 1 for part 2. Also include plot of weights (no training) over time for above and below 1 (4 plots total)
4. Plots of the percent training data v. classification accuracy for the original deepnet and a similar plot for the deepnets with 2 and 4 layers. Also include a plot of the original deepnet autoencoder weights. (3 plots total)
5. Plots of your CNN percent training data v. classification accuracy and a plot of the deep dream results. (2 plots total)

Note that there are several plots, so be sure to put most of them outside the 4 page limit.

*Discussion:* Should be 2-3 paragraphs long describing what you could use these tools for in the future.

This report will be combined with Lab 11, to create one cohesive report. The report (not including Appendix) should be no longer than 4 pages. Use 12 pt. font and 1.15-1.5 line spacing. If your text is over the 4-page limit with figures, you can move your figures to an appendix section that goes beyond the 4-page limit. However, any text that goes beyond this limit will not be graded, except for figures, figure titles (no captions), and your code.

Please upload your report to Canvas and leave a hard-copy with your GSI in lab. The hard-copy will be graded, so be sure different lines on your plots are distinguishable (using color or different line styles).