Assigned: 03 September 2020

Deep Learning – Homework #1

EE 599: Fall 2020

Due: Tuesday, 15 September 2020 at 23:59. Submission instructions will follow separately on Canvas.

Problem 1: Working with hd5 format data files

In this problem you will create an hd5 file containing a numpy array of binary random sequences that you generate yourself. The hd5 format is useful because it can store multiple data objects in a single file keyed by object name – *e.g.*, you can store a numpy array of floats called `regressor` and a numpy integer array called `labels` in the same file. Hd5 also allows for fast non-sequential access to objects in the file without scanning the entire file. This means you can efficiently access objects and data such as `x[idxs]` with non-consecutive indexes *e.g.*, `idxs = [2, 234, 512]`. This random-access property is useful when extracting a random subset from a larger training database.

Follow these steps:

1. Run the provided template python file (`random_binary_collection.py`). The script is set to `DEBUG` mode by default.
2. Experiment with the assert statements to trap errors and understand what they are doing by using the shape method on numpy arrays, etc.
3. Set the `DEBUG` flag to `False`. Then **manually** create 25 binary sequences each with length 20. **It is important that you do this by hand, *i.e.,* without the aid of a compute random number generator or, for example, a coin.**
4. Verify that your hd5 file was written properly by checking that it can be read-back.
5. Name your hd5 file using your full name and submit it as directed for the assignment.

Problem 2: A simple feedforward (MLP) neural network

A particular MLP has two input nodes, one hidden layer, and two outputs. The two sets of weights and biases are given by

$$\mathbf{W}_1 = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{W}_2 = \begin{bmatrix} 2 & 2 \\ 3 & -3 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 0 \\ -4 \end{bmatrix}.$$

The non-linear activation for the hidden layer is a ReLu (*rectified-linear unit*) – that is, $h(x) = \max(x, 0)$. The output layer is linear (*i.e.,* identity activation). What is the output activation for input $\mathtt{x} = [\mathtt{+1} \ \ \mathtt{-1}]^{\mathrm{t}}$?

Problem 3: Convolutions and correlations

Typical undergraduate signals and systems classes covers one- and two-dimensional convolutions and correlations. Convolutional Neural Networks (CNNs) use both operations. This is a review problem. The convolution of a one-dimensional, discrete time (*i.e.*, integer index) signal $x[n]$ with $h[n]$ is

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

and the correlation is

$$r[n] = x[n] \star h[n] = \sum_{k=-\infty}^{\infty} x[k]h[k+n].$$

For two-dimensional signals (*e.g.*, images and image filters) the convolution is

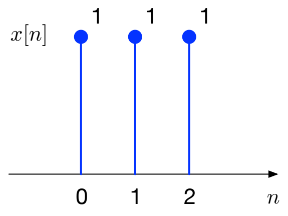
$$y[i][j] = x[i][j] * h[i][j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[m][n]h[i-m][j-n]$$

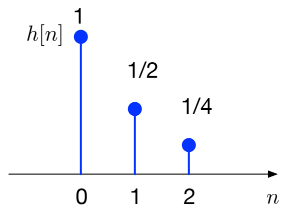
and correlation is

$$r[i][j] = x[i][j] \star h[i][j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[m][n]h[m+i][n+j].$$

1. Show that $y[n] = x[n] * h[n] = x[n] \star h[-n]$ in one-dimension — *i.e.*, that the correlation is convolution but using the time-reflected (time-reversed) signal. State and show an analogous relationship between correlation and convolution in two-dimensions.

2. Compute $y[n] = x[n] * h[n]$ for the signals shown below. Use Python to plot the result as a stem-plot.





3. Compute the two-dimensional correlation $r[i][j] = x[i][j] \star h[i][j]$ for the signals below. Assume values at indices not shown are zero and that you only need to show the output over the non-zero regions. Then plot a heat-map of the result using matplotlib.pyplot.matshow.

$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$
$\ldots$	0	0	0	0	0	0	0	$\ldots$
$\ldots$	0	$x[-2][1] = 1$	$x[-1][1] = 1$	$x[0][1] = 1$	$x[1][1] = 1$	$x[2][1] = 1$	0	$\ldots$
$\ldots$	0	$x[-2][0] = 1$	$x[-1][0] = 1$	$x[0][0] = 1$	$x[1][0] = 1$	$x[2][0] = 1$	0	$\ldots$
$\ldots$	0	$x[-2][-1] = 1$	$x[-1][-1] = 1$	$x[0][-1] = 1$	$x[1][-1] = 1$	$x[2][-1] = 1$	0	$\ldots$
$\ldots$	0	0	0	0	0	0	0	$\ldots$
$\iddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

and

$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$
$\ldots$	0	0	0	0	0	$\ldots$
$\ldots$	0	$h[-1][1] = 1/4$	$h[0][1] = 1/2$	$h[1][1] = 1/4$	0	$\ldots$
$\ldots$	0	$h[-1][0] = 1/2$	$h[0][0] = 1$	$h[1][0] = 1/2$	0	$\ldots$
$\ldots$	0	$h[-1][-1] = 1/4$	$h[0][-1] = 1/2$	$h[1][-1] = 1/4$	0	$\ldots$
$\ldots$	0	0	0	0	0	$\ldots$
$\iddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Problem 4: Building Python proficiency – Filter design

Python includes many filter design and analysis tools similar to those in Matlab. In this problem you will design and plot the frequency response for several ARMA filters. See Figure 1 for review and reference.

Many deep learning algorithms use first order ($L = 1$) AR filters with unit gain at DC – *i.e.,* the gain of the response at zero frequency is 1. Note that by "AR" you can assume that $b[i] = 0$ for all $i > 0$. This type of filter has a single parameter α which sets the pole location in the z-plane.

1. Specify the difference equation and the Z-transform for this first order AR filter – *i.e.,* specify $b[0]$, and $a[1]$ in terms of α. Note that in the standard form (Figure 1) $a[0]$ is set to 1 by default. But you must explicitly state $a[0]$ in the Python routine just as in MatLab.

   Note: *This is enough information to fully specify this filter. Contact your TA for a brief review of signals and systems and additional help if you do not see that.*

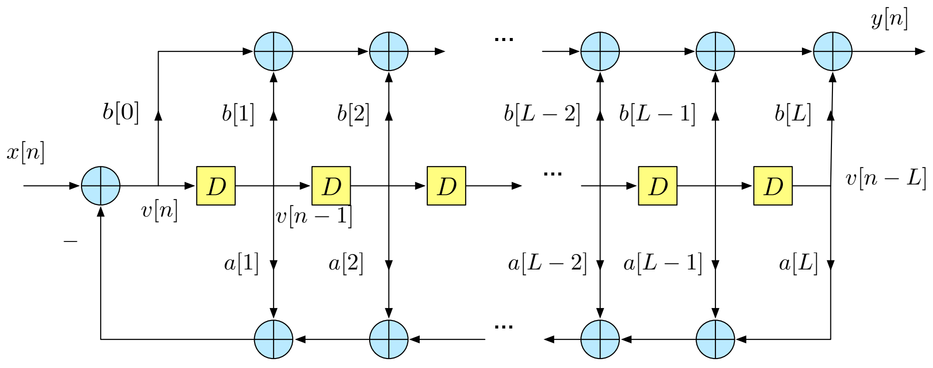
2. Plot the magnitude of the frequency response for $\alpha = 0.9$, $\alpha = 0.5$, $\alpha = 0.1$, $\alpha = -0.5$. Use the linear normalized frequency ν, which is unique on $[-1/2, +1/2]$ and has units of cycles per sample. **Hint:** `use scipy.signal.freqz`.

3. Give the time constant for $\alpha = 0.9$, $\alpha = 0.5$, $\alpha = 0.1$. Here we define the time-constant as the number of samples required for the impulse response to decay to 20% of its value at $n = 0$.

Python has equivalent commands for most of MatLab's filter design tools. This portion is intended to give you experience with these commands.

1. Design an $L = 4$ Butterworth filter with bandwidth of $\nu_0 = 0.25$ – here normalized linear frequency is denoted by ν in cycles/sample – *i.e.,* the frequency response is periodic in ν with period one. Provide the AR and MA coefficients. Then plot the magnitude of the frequency response.

2. Create a numpy array with length 300 comprising of i.i.d. realizations from the standard normal distribution. Filter the sequence using your $L = 4$ Butterworth filter. Then plot the input and output signals on the same plot. **Hint:** Use `scipy.signal.lfilter`.



$$v[n] = x[n] - (a[1]v[n-1] + a[2]v[n-2] + \cdots a[L]v[n-L])$$

$$y[n] = b[0]v[n] + b[1]v[n-1] + b[2]v[n-2] + \cdots + b[L]v[n-L]$$

$$\text{state}[n] = (v[n-1], v[n-1], \ldots v[n-L])$$

implements this difference equation:

$$y[n] = \sum_{i=0}^{L} b[i]x[n-i] - \sum_{i=1}^{L} a[i]y[n-i]$$

Frequency response:

$$H(z) = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2} \cdots + b[L]z^{-L}}{1 + a[1]z^{-1} + a[2]z^{-2} \cdots + a[L]z^{-L}} \qquad z = e^{j2\pi\nu}$$

Figure 1: The general format and notation for an ARMA filter. The arrays of AR coefficients $a[n]$ and MA coefficients $b[n]$ define the filter. The order of teh filter is the number of delay elements in thsi diagram, L.