Assigned: 30 September 2021

Homework #3 – Estimation and LMS

EE 541: Fall 2021

Due: Friday, 08 October 2021 at 23:59. Late penalty: 10% per 24-hours before 10 October at 23:59. Submission instructions will follow separately on canvas.

1. Recall that the output for hidden layer l is given by $a^{(l)} = h_l\left(W_l\, a^{(l-1)} + b_l\right)$ Say an MLP has two input nodes, one hidden layer, and two outputs. The two sets of weights and biases are given by:

$$\boldsymbol{W_1} = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix}, \qquad b_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\boldsymbol{W_2} = \begin{bmatrix} 2 & 2 \\ 3 & -3 \end{bmatrix}, \qquad b_2 = \begin{bmatrix} 0 \\ -4 \end{bmatrix}.$$

The non-linear activation for the hidden layer is ReLU (rectified linear unit) – that is $h(x) = \max(x, 0)$. The output layer is linear (*i.e.*, identify activation function). What is the output activation for input $x = [+1;\ -1]^T$?

2. The hd5 format can store multiple data objects in a single fille each keyed by object name – *e.g.*, you can store a numpy float array called regressor and a numpy integer array called labels in the same file. Hd5 also allows fast non-sequential access to objects 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.

In this problem you will create an hd5 file containing a numpy array of binary random sequences that you generate yourself. 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`. Manually create 25 binary sequences each with length 20. It is important that you do this by hand, *i.e.*, **do not use a coin, computer, or random number generator**.

(4) Verify that your hd5 file was written properly by checking that it can be read-back.

(5) Submit your hd5 file as directed.

3. The MNIST dataset of handwritten digits is one of the earliest and most used datasets to benchmark machine learning classifiers. Each datapoint contains 784 input features – the pixel values from a 28x28 image – and belongs to one of 10 output classes – representing the numbers 0-9.

   In this problem you will use numpy to program the feed-forward phase for a deep multilayer perceptron (MLP) neural network. We provide you with a pre-trained MLP using the MNIST dataset. The MLP has an input layer with 784-neurons, 2 hidden layers of 200 and 100 neurons, and a 10-neuron output layer. The model assumes the ReLU activation for the hidden layers and the softmax function in the output layer.

   (a) Extract the weights and biases of the pre-trained network from `mnist_network_params.hdf5`. The file has 6 keys corresponding to: W_1 , b_1, W_2, b_2, W_3, b_3. Verify the dimension of each numpy array with the shape property.

   (b) The file `mnist_testdata.hdf5` contains 10,000 images. `xdata` holds pixel intensities and `ydata` contains the corresponding class labels. Extract these. Note: each image vector is 784- dimensions and the label is 10-dimensional with one-hot encoded, *i.e.*, label `[0,0,0,1,0,0,0,0,0,0]` means the image is number "3".

   (c) Write functions to calculate ReLU and softmax:

   - $\mathrm{ReLU}(x) = \max(0, x)$.
   - $\mathrm{Softmax}(x) = \left[\frac{e^{x_1}}{\sum_{i=1}^{n} e_i^x}, \frac{e^{x_2}}{\sum_{i=1}^{n} e_i^x}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^{n} e_i^x}\right]$.

     The softmax function takes a vector of size n and returns another vector of size n that you can interpret as a probability distribution. For example: $\mathrm{Softmax}([0;\ 1;\ 2]) = [0{:}\,09;\ 0{:}\,24;\ 0{:}\,67]$ so you can conclude that the 3rd element is the most likely outcome.

   (d) Use numpy to create an MLP to classify 784-dimensional images into the target 10-dimensional output. Use ReLU activation for the two hidden layers and softmax the output layer.

   (e) Compare your prediction with the (true) `ydata` label. Count the classification as correct if the position of the maximum element in your prediction matches with the position of the 1 in `ydata`. Tally the number of correctly classified images from the whole set of 10,000. [hint: 9790 correct].

   (f) Identify and investigate several inputs that your MLP classified correctly and several it classified incorrectly. Inspect them visually. Is the correct class obvious to you in the incorrect cases? Use matplotlib to visualize:

```
import matplotlib.pyplot as plt
plt.imshow(xdata[i].reshape(28,28))
```

```
    plt.show()
    # the index i selects which image to visualize
    # xdata[i] is a 784x1 numpy array
```

4. Consider the system in Figure 1. It shows a model that predicts future outputs y_n from current and past inputs x_n. Figure 1(a) shows the observed signal z_n which is y_n – generated as x_n passes through a linear system h_n – corrupted by additive Gaussian noise q_n. Consider the case where h_n is a "matched" Wiener filter with length $L = 3$ (*i.e.*, 3-tap filter). "Matched" means that the filter coefficients are matched to the x_n signal generator. Figure 1(b) shows how to apply the ideal Wiener filter w_{LMMSE} to the input $\boldsymbol{v_n}(u) = [\,x_n(u),\ x_{n-1}(u),\ x_{n-2}(u)]^T$ to predict future outputs $\hat{y}_n$

$$\hat{y}_n = \boldsymbol{w}_{\text{LMMSE}}^T\,\boldsymbol{v}_n$$

$$\boldsymbol{w}_{\text{LMMSE}} = \boldsymbol{R}_v^{-1}\,\boldsymbol{r}_{v_y}$$

where the correlation matrices do not depend on the time n. Figure 1(c) shows the LMS filter which can *learn* to approximate the optimal filter. The LMS algorithm can also work "online". In this mode it tracks and adaptively update the taps of h_n based on an ("**un**matched") time-varying signal even when the data comes from a complex (or unknown) process.

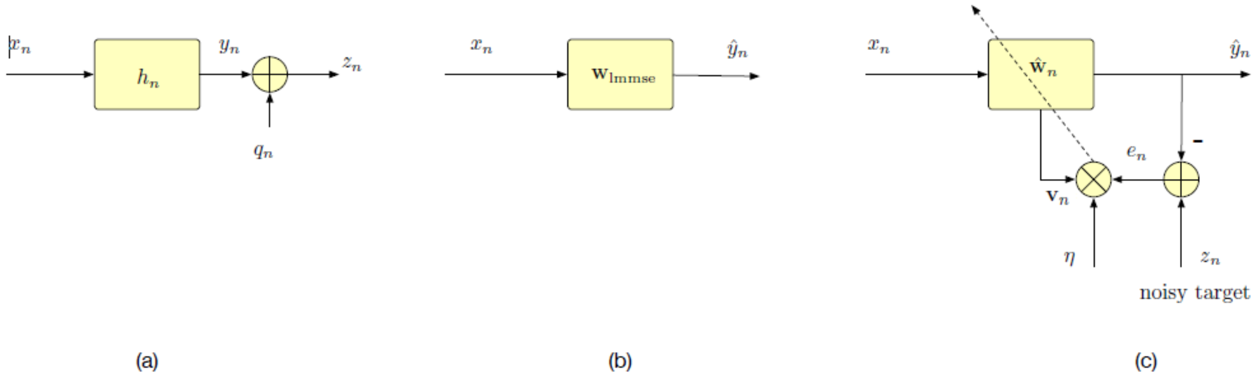

Figure 1 (a) MMSE estimation model. (b) System using LMMSE estimator. (c) Applying LMS with noisy targets z_n.

Use the datafile `lms_fun.hdf5`. The file includes several data arrays with keys described below. You will first compute the 3-tap Wiener filter coefficients assuming a "matched" condition. The model is

$$z_n(u) = y_n(u) + q_n(u)$$

$$z_n(u) = h_0\,x_n(u) + h_1\,x_{n-1}(u) + h_2\,x_{n-2}(u) + q_n(u)$$

where $x_n(u)$ is a sequence of i.i.d. standard normal random variables with $h_0 = 1$, $h_1 = 0.5$, and $h_2 = 0.25$. The noise $q_n(u)$ is also i.i.d. normal with zero mean and variance σ_q^2. Define the signal-to-noise ratio (SNR) as

$$\text{SNR} = \frac{\sigma_x^2}{\sigma_q^2} = \frac{1}{\sigma_q^2}$$

1. The hd5 datafile contains 600 sequences of length 501 samples for each of SNR=3 dB and SNR=10 dB. It includes an x and z array for each noise intensity: `matched_10_x`, `matched_10_z` and `matched_3_x`, `matched_3_z`. Build a filter and use the data to produce curves as in the lecture slides. Note: you may also need to use the included data for y_n and v_n. You *could* generate v_n from x_n but the datafile includes the value to simplify your task.

   (a) Program the LMS algorithm using input (regressor) v_n and "noisy target" z_n. This corresponds to the example given in lecture.

   (b) Plot learning curves for each SNR with $\eta = 0.05$ and $\eta = 0.15$. Learning curves are plots of the MSE – $[z_n - \hat{z}_n]^2$ – w.r.t. sample number and averaged over all sequences.

   (c) What is the largest value of η that does not lead to divergence of the MSE?

2. This part uses a single realization for the input sequence x (v and output sequence y each of length 501 – *e.g.*, `timevarying_v`, `timevarying_z`. But these data came from a time-varying (but linear) filter – *i.e.*, the coefficients above vary with n. The dataset `timevarying_coefficents` contains the sequences of the 3 coefficients with respect to time.

   Plot the coefficients vs. time n. Run the LMS algorithm using the x and z data. Find a learning rate η so that your LMS algorithm tracks the coefficient variations. Plot your coefficient estimates against the true coefficients for this case.

3. This part uses the dataset with keys `mismatched_x` and `mismatched_y`. This is a set of 600 sequences of length 501 samples each. But this data comes from a non-linear (and unknown to you) process.

   (a) Run the LMS algorithm over all 600 sequences. Find several good values for η and plot the average learning curves. Note: you only have access to the noisy y_n in this part.

   (b) Compute $\hat{R}_{v_n}$ and $\hat{r}_n$ and the corresponding LLSE using the entire set. Is this value lower than the LMS learning curve after convergence?