EE599 Deep Learning – Homework 2

©K.M. Chugg

February 8, 2020

assigned: Feb. 6, 2020
due: Feb. 19, 2020

1 Character recognition using a trained MLP

The purpose of this problem is to have you program, using numpy, the feedforward processing for a multilayer perceptron (MLP) deep neural network. This processing is defined on slide 33 of the "introduction" lecture slides. The grading of this homework will give you feedback on your FF program which is valuable since in a future homework, you will be building on this to program the back-propagation as well.

The MNIST dataset of handwritten digits is widely used as a beginner dataset for benchmarking machine learning classifiers. It has 784 input features (pixel values in each image) and 10 output classes representing numbers 0–9. We have trained a MLP on MNIST, with a 784-neuron input layer, 2 hidden layers of 200 and 100 neurons, and a 10-neuron output layer. The activation functions used are ReLU for the hidden layers and softmax for the output layer.

(a) Extract the weights and biases of the trained network from `mnist_network_params.hdf5`. The file has 6 keys corresponding to numpy arrays W_1, $\mathbf{b}_1$, $\mathbf{W}_2$, $\mathbf{b}_2$, $\mathbf{W}_3$, $\mathbf{b}_3$. You may want to check their dimensions by using the 'shape' attribute of a numpy array.

(b) The file `mnist_testdata.hdf5` contains 10,000 images in the key 'xdata' and their corresponding labels in the key 'ydata'. Extract these. Note that each image is 784-dimensional and each label is one-hot 10-dimensional. So if the label for an image is $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$, it means the image depicts a 3.

(c) Write functions for calculating ReLU and softmax. These are given as:

- $\text{ReLU}(x) = \max(0, x)$
- $\text{Softmax}(\mathbf{x}) = \left[\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \quad \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \quad \cdots, \quad \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_n}}\right]$. Note that the softmax function operates on a vector of size n and returns another vector of size n which is a probability distribution. For example, $\text{Softmax}([0, 1, 2]) = [0.09, 0.24, 0.67]$. This indicates that the 3rd element is the most likely outcome. For the MNIST case, $n = 10$.

(d) Using numpy, program a MLP that takes a 784-dimensional input image and calculates its 10-dimensional output. Use ReLU activation for the 2 hidden layers and softmax for the output layer.

(e) Compare the output with the true label from 'ydata'. The input is correctly classified if the position of the maximum element in the MLP output matches with the position of the 1 in ydata. Find the total number of correctly classified images in the whole set of 10,000. You should get 9790 correct.

(f) Sample some cases with correct classification and some with incorrect classification. Visually inspect them. For those that are incorrect, is the correct class obvious to you? You can use matplotlib for visualization:

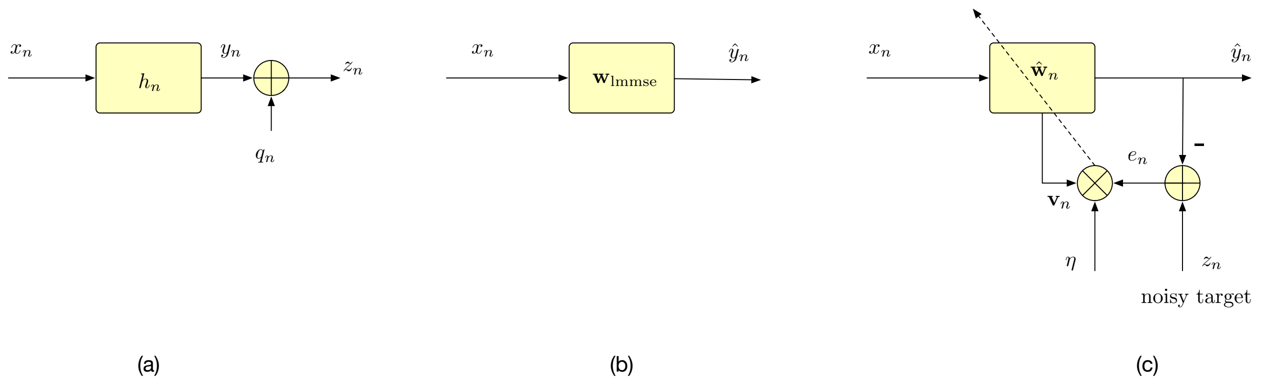

Figure 1: (a) a model for MMSE estimation. (b) a system implementing the LMMSE estimator. (c) the LMS algorithm with noisy targets, z_n.

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

Here, `i` is the index of the image you want to visualize, i.e. `xdata[i]` is a 784-dimensional numpy array.

2 Logistical Regression SGD

From lecture, we have that the output of a logistical regressor is

$$p = \sigma(\mathbf{w}^{\mathrm{t}}\mathbf{x} + b)$$

Derive the equations for using SGD to update the trainable parameters $\mathbf{w}$ and b when the label is $y \in \{0, 1\}$. Recall that p is the output of the regressor and is interpreted as the probability that $y = 1$. Also recall that the binary cross-entropy loss is used in this case

$$C = -y \log p - (1 - y) \log(1 - p)$$

3 LMS algorithm for channel modeling/tracking

In this problem, you will consider the the system of Fig. 1. This can be viewed as estimating a channel or system so that future outputs can be predicted. Fig. 1(a), shows a model where x_n goes through a linear system and is observed in additive Gaussian noise. We will consider the case where h_n is a 3-tap filter. In this case, a Wiener filter with length $L = 3$ is "matched" to this signal generation model. After computing the ideal Wiener filter of length 3, $\mathbf{w}_{\text{llms}}$, this can be used as model to predict future outputs – this is shown in Fig. 1(b). The LMS algorithm may be used to approximate this as shown in Fig. 1(c).

In addition to the Wiener Filter and LMS algorithm for this matched model, we will consider a time-varying model to show that the LMS algorithm can track the taps of h_n and also a case where the data is generated using a more complex (unknown to you) model in place of Fig. 1(a).

This problem is based on the data file `lms_fun.hdf5`. This file has several data arrays with keys described below.

1. In this part, you will find the 3-tap Wiener filter for the "matched" condition. Here the model is

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

where $x_n(u)$ is a sequence of iid, standard Gaussians, $h_0 = 1$, $h_1 = 0.5$, and $h_2 = 0.25$. The noise added, $q_n(u)$ is also iid, Gaussian with zero mean and variance σ_q^2. The SNR is defined as[1]

$$\text{SNR} = \frac{\sigma_x^2}{\sigma_q^2} = \frac{1}{\sigma_q^2} \tag{3}$$

Consider the Wiener filter LMMSE estimator that estimates $z_n(u)$ from $\mathbf{v}_n(u) = [x_n(u), x_{n-1}(u), x_{n-2}(u)]^{\mathrm{t}}$:

$$\hat{z}_n = \mathbf{w}_{\text{lmmse}}^{\mathrm{t}} \mathbf{v}_n \qquad \mathbf{w}_{\text{lmmse}} = \mathbf{R}_{\mathbf{v}}^{-1} \mathbf{r}_{\mathbf{v}z} \tag{4}$$

where we have assumed that the correlation matrices are not a function of the time n (you will verify). This problem will walk you through the derivation of the Wiener filter.

   (a) Show that $\mathbb{E}\{\mathbf{v}_n(u) z_n(u)\} = \mathbb{E}\{\mathbf{v}_n(u) y_n(u)\}$ so that the Wiener filter for estimating $y_n(u)$ and $z_n(u)$ is the same and $\hat{z}_n = \hat{y}_n$.

   (b) Show that $R_{xz}[m] = R_{xy}[m] = \mathbb{E}\{x_n(u) z_{n+m}(u)\} = h_m$.

   (c) Fnd the (3×3) matrix $\mathbf{R}_{\mathbf{v}_n}$ and show that is is not a function of n.

   (d) Use the above results to find $\mathbf{r}_n = \mathbb{E}\{\mathbf{v}_n(u) y_n(u)\} = \mathbb{E}\{\mathbf{v}_n(u) z_n(u)\}$ and show it is not a function of n.

   (e) Find the Wiener filter (LMMSE) filter taps $\mathbf{w}_{\text{lmmse}}$. Compute this numerically for SNRs of 3 and 10 dB.

   (f) What is the LMMSE – *i.e.,* the residual mean squared error for the Wiener filter? Compute this numerically for SNRs of 10 and 3 dB. Do this for both the case where the Wiener filter is used to estimate $z_n(u)$ and $y_n(u)$ – what is the difference?

2. There is data corresponding to 10 dB and 3 dB of SNR cases shown in lecture. Specifically, there are 600 sequences of length 501 samples each. There is an x and z array for each: – *i.e.,* `matched_10_x`, `matched_10_z` and `matched_3_x`, `matched_3_z` for 10 dB and 3 dB SNR, respectively. Use these to develop your code and run curves similar to those shown in the slides. To aid in this process, there is also data for y_n and $\mathbf{v}_n$. Note that $\mathbf{v}_n$ can be reproduced from x_n, but we have done this for you to help out.

   (a) Program the LMS algorithm with input (regressor) $\mathbf{v}_n$ and "noisy target" z_n. This corresponds to the example given in lecture.

   (b) Plot learning curves for $\eta = 0.05$ and $\eta = 0.15$ for each SNR. Specifically, plot the MSE (average of $(z_n - \hat{z}_n)^2$), averaged over the 600 sequences.

   (c) How does the MSE for these learning curves compare to the LMMSE found in the analytical part above? Note that you can also try using y_n in place of z_n if you would like to explore (optional).

   (d) What is the largest value of η that you can use without having divergent MSE?

3. In this part, there is a single realization an x ($\mathbf{v}$ and y sequence, each of length 501 - *e.g.,* `timevarying_v`, `timevarying_z`. In this case the data were generated using a time-varying linear filter – coefficients

[1] It may make more sense to use $\text{SNR} = \frac{\sigma_x^2(h_0^2+h_1^2+h_2^2)}{\sigma_q^2}$, but I used the definition specified here for my simulations.

in (2) actually vary with n. There is a dataset called `timevarying_coefficents` that has the 3 coefficients as they change with time. Plot these coefficients vs. time (n). Run the LMS algorithm using the x and z datasets and and vary the learning rate to find a good value of η where the LMS algorithm tracks the coefficient variations well. Plot the estimated coefficients along with the true coefficients for this case.

4. In this part, use the dataset with key `mismatched_x` and `mismatched_y`. This is also a set of 600 sequences of length 501 samples each. This data was generated with a model that is not linear and is unknown to you.

   (a) Run the LMS algorithm over all 600 sequences and plot the average learning curve for a goods choice of η. Note: in this case, you only have access to y_n, which may contain noise.

   (b) Using the entire data set, compute $\hat{\mathbf{R}}_{\mathbf{v}_n}$ and $\hat{\mathbf{r}}_n$ and the corresponding LLSE. Is this lower than the LMS learning curve after convergence?

4 Human vs Computer random sequences

Note: Portions of this problem will be worked in class.

1. Use the file `binary_random_sp2030.hdf5`. This file has all of the data generated by students in HW1 and also has an equal number of sequences generated using numpy. Using this data, construct the (20×20) matrix $\hat{\mathbf{R}}$ – *i.e.,* the sample correlation matrix for the data. Note that the data has been converted from 0 and 1 to ± 1.
2. Find the eigen-vectors and eigen-values for $\hat{\mathbf{R}}$. What is the variance of the most significant two components of the data? What percentage of the total variance is captured by these two components? Can you see any significance in the eigen-vectors $\mathbf{e}_0$ and $\mathbf{e}_1$ that would suggest why they capture much of the variation? Plot λ_k vs. k on a stem plot.
3. Create label data to indicate human or computer – *i.e.,* computer: $y = +1$, human: -1 and merge the two data sets. Compute the linear classifier weight vector $\mathbf{w}$ that maps the (20×1) vector to an estimate of the label. What is the error rate when you threshold $\hat{y}$ to a hard decision?
4. Visualize the data and classifier in two dimensions. Take a reasonable number of samples from each the computer and human data – *e.g.,* 100 each. Project these onto $\mathbf{e}_0$ and $\mathbf{e}_1$ and plot a scatter plot of the data. Add to this plot the decision boundary projection in this 2D space.
5. Compute the logistical regression weight vector $\mathbf{w}$ that maps the (20×1) vector to an estimate of the probability of the label. What is the error rate when you threshold $\hat{y}$ to a hard decision?

5 Backprop Initialization for Multiclass Classification

Recall that the softmax function $\mathbf{h}(\cdot)$ takes an M-dimensional input vector $\mathbf{s}$ and outputs an M-dimensional output vector $\mathbf{a}$ given as

$$\mathbf{a} = \mathbf{h}(\mathbf{s}) = \frac{1}{\sum_{m=0}^{M-1} e^{s_m}} \begin{bmatrix} e^{s_0} \\ e^{s_1} \\ \vdots \\ e^{s_{M-1}} \end{bmatrix} \tag{5}$$

Recall that multiclass cross-entropy cost is given as

$$C = -\sum_{i=1}^{n} y_i \ln a_i \tag{6}$$

where $\mathbf{y}$ is a given vector of ground truth labels. Finally, recall that the error vector of the output layer is given as:

$$\boldsymbol{\delta} = \nabla_{\mathbf{s}} C = \dot{\mathbf{A}} \nabla_{\mathbf{a}} C \tag{7}$$

where $\dot{\mathbf{A}}$ is the matrix of derivatives of softmax, given as[2]

$$\dot{\mathbf{A}} = \frac{d\mathbf{h}(\mathbf{s})}{d\mathbf{s}} = \begin{bmatrix} \frac{\partial p_0}{\partial s_0} & \cdots & \frac{\partial p_{M-1}}{\partial s_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_0}{\partial s_{M-1}} & \cdots & \frac{\partial p_{m-1}}{\partial s_{M-1}} \end{bmatrix} \tag{8}$$

Assuming $\mathbf{y}$ is one-hot, show that $\boldsymbol{\delta} = \mathbf{a} - \mathbf{y}$

6 Backprop by Hand

An MLP has three input nodes, two hidden layers, and three outputs. The activation for the hidden layers is ReLu. The output layer is softmax. The weights and biases for this MLP are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

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

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

1. **Feedforward Computation:** In this part you will perform the feedforward computation for the input vector $\mathbf{x} = [\ +1\ -1\ +1\ ]^{\mathrm{t}}$. The standard notation used in the slides and handouts is used. Specifically, $\mathbf{s}^{(l)}$ is the linear activation – *i.e.,* $\mathbf{a}^{(l)} = \underline{h}(\mathbf{s}^{(l)})$ – and $\dot{\mathbf{a}}^{(l)} = \dot{\underline{h}}(\mathbf{s}^{(l)})$. Fill in the following table:

l :	1	2	3
$\mathbf{s}^{(l)}$:			
$\mathbf{a}^{(l)}$:			
$\dot{\mathbf{a}}^{(l)}$:			(not needed)

2. **Backpropagation Computation:** Run the standard SGD backpropagation for this input assuming a multi-category cross-entropy loss function and the one-hot labeled target: $\mathbf{y} = [\ 0\ \ 0\ \ 1\ ]^{\mathrm{t}}$. Again, our standard notation is used so that $\boldsymbol{\delta}^{(l)} = \nabla_{\mathbf{s}^{(l)}} C$. Enter these delta values in the table below and also provide the updated weights and biases assuming a learning rate of $\eta = 0.5$.

l :	1	2	3

[2]This is the denominator convention with the left-handed chain rule.

$\boldsymbol{\delta}^{(l)}$:			
$\mathbf{W}^{(l)}$:			
$\mathbf{b}^{(l)}$:			

7 Avoiding Softmax Computations in Multiclass Classification

Computing the softmax activation for a large number of classes can be computationally intensive. For example, we will see later that the word2vec word embedding method uses a trick to avoid computing the softmax for this reason. In this problem, we explore a different method for avoiding softmax computation in multiclass classification problems. In the following, we consider an output layer with M neurons for an M-ary classification problem. The activation for this final layer is a sigmoid so that the final activation is

$$a_i = \sigma(s_i) = \frac{1}{1 + e^{-s_i}} \tag{9}$$

where s_i is the linear or pre-activation for the final layer. In other words, the softmax, which computes an M-ary probability mass function (pmf) while this approach computes M 2-ary pmfs – *i.e.,* one 2-ary pmf for each output class. The labels $\mathbf{y}$ are one-hot encoded for the M-ary classification problem. Specifically, only one component of $\mathbf{y}$ is 1 and the rest are 0.

1. Explain why using the activations in (9) with the standard multi-class cross-entropy function does not make sense. Hint: recall that the milti-class cross-entropy function may be viewed as a constant offset from the KL Divergence between two pmfs.

2. Recall that for a binary classification problem, the binary cross-entropy loss is used

$$\text{BCE}(a) = -y \log a - (1 - y) \log(1 - y) \tag{10}$$

Consider the multi-class loss function that is the sum of the BCE for each class

$$C_{M-BCE} = -\left[\sum_{m=0}^{M-1} y_m \log a_m + (1 - y_m) \log(1 - y_m)\right] \tag{11}$$

where here the label is still one-hot. Find the back-propogation gradient initialization for this case – *i.e.,* find $\nabla_{\mathbf{s}} C_{M-BCE}$.

3. How does your result for part 2 of this problem compare to the standard multi-class gradient initialization from problem 5? In what ways is it similar and in what ways does it differ. Can the C_{M-BCE} be viewed as resulting from the KL divergence?