Assigned: 16 September 2020

Deep Learning – Homework #2

EE 599: Fall 2020

Due: Thursday, 01 October 2020 at 23:59. Submission instructions will follow separately on Canvas.

1 Character recognition using a trained MLP

In this problem you will use numpy to program the feed-forward phase for a deep multilayer perceptron (MLP) neural network. The "Introduction" lecture slides describe the process starting on slide 33. Feedback on this assignment will be valuable for a future homework in which you extend your network to include back-propagation.

The MNIST dataset of handwritten digits is one of the earliest and most commonly used datasets to benchmark machine learning classifiers. Each datapoint contains 784 input features – the pixel values from a 28x28 image – and it belongs to one of 10 output classes – representing the numbers 0–9. 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 arrays $\mathbf{W}_1$, $\mathbf{b}_1$, $\mathbf{W}_2$, $\mathbf{b}_2$, $\mathbf{W}_3$, $\mathbf{b}_3$. 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:

- $\text{ReLU}(x) = \max(0, x)$.
- $\text{Softmax}(\mathbf{x}) = \left[\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \quad \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \quad \cdots, \quad \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_n}}\right]$.

  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: $\text{Softmax}([0, 1, 2]) = [0.09, 0.24, 0.67]$ so you can conclude that the the 3rd element is the most likely outcome. $n = 10$ for the MNIST case.

(d) Use numpy to create an MLP to classify each 784-dimensional image into the target 10-dimensional output. Use ReLU activation for the two hidden layers and softmax activation in 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) Investigate a 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 for visualization:

```
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
```

2 Logistical Regression SGD

Recall the logistic regressor from lecture:

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

which gives output p that you can interpret as the probability that $y = 1$. Derive the SGD update equations for trainable parameters $\mathbf{w}$ and b if label $y \in \{0, 1\}$. Assume a binary cross-entropy loss

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

3 LMS algorithm for channel modeling/tracking

Consider the the system in Fig. 1. It shows a model that tracks the input to predict future outputs. Fig. 1(a) shows how z_n is observed after x_n passes through a linear system followed by the addition of Gaussian noise. 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. The ideal Wiener filter $\mathbf{w}_{\mathrm{llms}}$ predicts future outputs as Fig. 1(b) shows. The LMS algorithm is a way to approximate this optimal filter Fig. 1(c). The LMS algorithm also works "online". In this mode it can track and update the taps of h_n based on an ("un-matched") time-varying signal where data comes from a complex (and unknown to you) process.

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

1. This question will have you compute the 3-tap Wiener filter coefficients assuming a "matched" condition. The model is

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

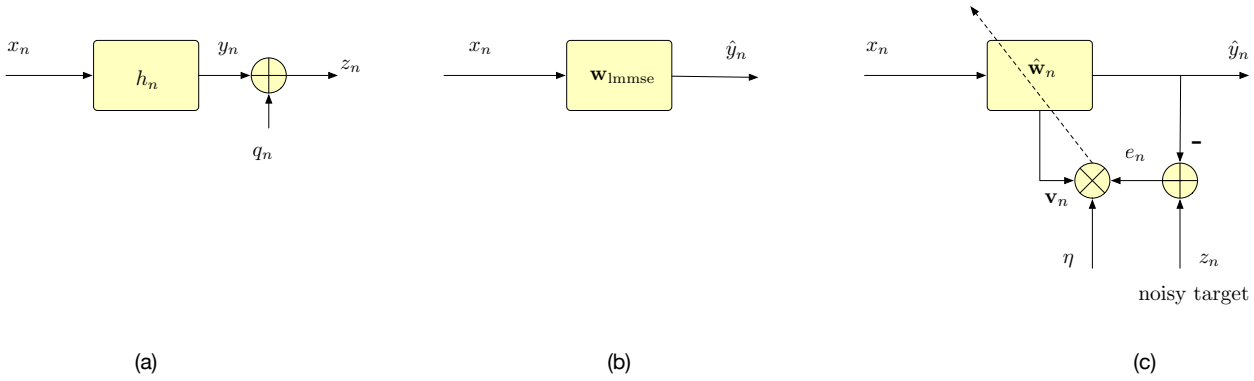

Figure 1: (a) MMSE estimation model. (b) System using LMMSE estimator. (c) Applying LMS with noisy targets, z_n.

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

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

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

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

where the correlation matrices do not depend on the time n (you will verify). Follow these steps to derive the Wiener filter.

(a) Show that $\mathbb{E}\{\mathbf{v}_n(u) z_n(u)\} = \mathbb{E}\{\mathbf{v}_n(u) y_n(u)\}$. Thus the Wiener filter to estimate $y_n(u)$ 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) Find the (3×3) matrix $\mathbf{R}_{\mathbf{v}_n}$. Show that it is not a function of n.

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

(e) Derive the LMMSE Wiener filter taps $\mathbf{w}_{\text{lmmse}}$. Compute the filter taps numerically for SNR=3 dB and SNR=10 dB.

(f) Find the LMMSE, *i.e.,* the residual mean squared error, for the Wiener filter. Compute the LMMSE numerically for SNR=10 dB and SNR=3 dB. Do this for the two cases: estimate (1) $z_n(u)$ and (2) $y_n(u)$ using the Wiener filter. What is the difference?

2. In this part you will use a datafile that contains sequences with SNR=10 dB and SNR=3 dB. There are 600 sequences of length 501 samples each. Each case (3 dB and 10 dB) has an x

[1] $\text{SNR} = \frac{\sigma_x^2 (h_0^2 + h_1^2 + h_2^2)}{\sigma_q^2}$ may make more sense but this is the definition I used in the simulations

and z array: `matched_10_x`, `matched_10_z` and `matched_3_x`, `matched_3_z`. You will build filter and then use the data to produce curves as in the lecture slides. Note: you may also need to use provided data for y_n and $\mathbf{v}_n$. You *can* reproduce $\mathbf{v}_n$ from x_n but we included the value to simplify your task.

(a) Program the LMS algorithm using input (regressor) $\mathbf{v}_n$ 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 (average of $(z_n - \hat{z}_n)^2$) averaged over all sequences.

(c) How does the MSE for these learning curves compare to the LMMSE in the analytical solution above? Note: you may also try to use y_n in place of z_n if you would like to explore (optional).

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

3. This part uses a single realization for the input sequence x ($\mathbf{v}$ 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 in (2) 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 you coefficient estimates against the true coefficients for this case.

4. 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 $\hat{\mathbf{R}}_{\mathbf{v}_n}$ and $\hat{\mathbf{r}}_n$ and the corresponding LLSE using the entire set. Is this value lower than the LMS learning curve after convergence?

4 Human vs Computer random sequences

Note: We will work portions of this problem in class.

1. Use the file `binary_random_20fa.hdf5`. This file contains student generated sequences from HW1 as well as an equal number of randomly generated numpy sequences. Use the data to construct the 20×20 sample correlation matrix matrix $\hat{\mathbf{R}}$. Note: we converted the data from $\{0, 1\}$ to ± 1.

2. Compute the eigenvectors and eigenvalues for $\hat{\mathbf{R}}$. What is the variance of the most significant two components? What percentage of the total variance do these two components capture? What is the significance of the eigenvectors $\mathbf{e}_0$ and $\mathbf{e}_1$ and why do these eigenvectors 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 then merge the two data sets. Compute the linear classifier weight vector $\mathbf{w}$ to optimally estimate the label y given the 20×1 sequence vector. 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 number of samples from each class (human vs. computer) – *e.g.,* 100 each. Project these onto eigenvectors $\mathbf{e}_0$ and $\mathbf{e}_1$ and overlay a scatter plot of the samples. Add the decision boundary in 2-dimensional space to the plot.

5. Compute the logistic regression weight vector $\mathbf{w}$ that estimates the probability of the $+1$ label using the (20×1) vector 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

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

and that the multiclass cross-entropy cost is given by

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

where $\mathbf{y}$ is a vector of *ground truth* labels. We define the error (vector) of the output layer 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]

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

Show that $\boldsymbol{\delta} = \mathbf{a} - \mathbf{y}$ assuming $\mathbf{y}$ is one-hot.

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

6 Backprop by Hand

Consider an MLP with three input nodes, two hidden layers, and three outputs. The hidden layers use the ReLU activation function and the output layer users 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:** Perform the feedforward calculation for the input vector $\mathbf{x} = [\ +1\ \ -1\ \ +1\ ]^{\mathrm{t}}$. Assume the standard notation used in the slides and handouts, *i.e.,* $\mathbf{s}^{(l)}$ is the linear activation with $\mathbf{a}^{(l)} = \underline{h}(\mathbf{s}^{(l)})$ and $\dot{\mathbf{a}}^{(l)} = \underline{\dot{h}}(\mathbf{s}^{(l)})$. Fill in the following table:

l :	1	2	3

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

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

l :	1	2	3

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

7 Avoiding Softmax Computations in Multiclass Classification

Computing the softmax activation over a large number of classes is computationally intensive. We will later see how the `word2vec` word embedding algorithm uses a trick to avoid computing the softmax. In this problem you will explore an alternate method to avoid softmax in multiclass classification problems. Consider an output layer with M neurons for an M-ary classification problem. The activation for the output layer is sigmoid:

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

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

1. Explain why the alternate activation in (9) in the context of 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. The binary cross-entropy loss is often used in binary classification problems,

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

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

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

   assuming one-hot labels. Find the back-propagation 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 in some way?