EE599 Deep Learning – Homework 2

©K.M. Chugg, B. Franzke, S. Dey

January 19, 2019

assigned: Jan. 16, 2019
due: Jan. 30, 2019

Problem 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 $\boldsymbol{W}_1$, $\boldsymbol{b}_1$, $\boldsymbol{W}_2$, $\boldsymbol{b}_2$, $\boldsymbol{W}_3$, $\boldsymbol{b}_3$. 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:

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

```
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.

Problem 2: Thought problem on features for classifying human-random vs. computer-random binary sequences

The "random" binary data collected in homework 1 is to be used for classifying human-generated vs. computer-generated random sequences. It is well-known that humans have difficulty generating such sequences with statistics that match those of a truly random binary sequence. For example Claude Shannon built a mind-reading machine that exploited this property.

Suppose we use a computer to generate a set of random sequences of the same length as your HW1 and a total number of computer generated sequences equal to the total number of student sequences generated. We could feed these 20 dimensional binary vectors into a ML algorithm (*e.g.,* an MLP), but we may also consider first generating features.

What are some reasonable features that you think would simplify this classification problem? To answer this, consider what inherently makes a human-generated sequence different that a computer-generated sequence.

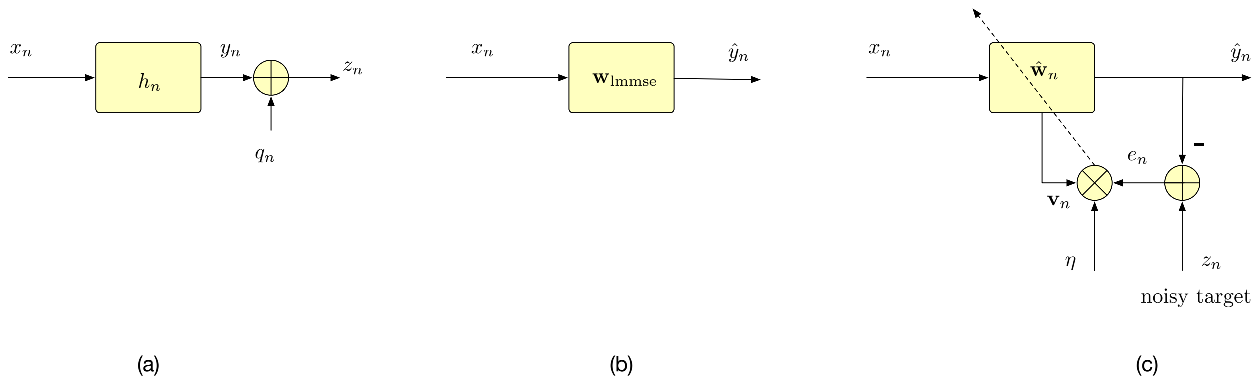

Figure 1: (a) a model for MMSE estimation. (b) a system implementing the LMMSE estimator. (c) the LMS algorithm with noisy targets, z_n.

Problem 3: Using the LMS algorithm for channel modeling/tracking on "matched" and "mismatched" data.

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

$$z_n(u) = y_n(u) + q_n(u) \tag{1}$$

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

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]

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

[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.

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

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

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 $\mathbf{w}_{\mathrm{lmmse}}$. 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 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?

Problem 4: MLP model exploration for the data from Problem 3

The choice of neural network architecture is a critical modelling decision. In this problem you will experiment with network geometry under the constraint of a fixed maximum number of neurons. You will design a network to predict the non-linear data from Problem 3.4.

For this problem use the `scikit-learn neural_network` toolkit to define, train, and test your network. You will see each aspect of this process in greater detail throughout the term. But you can treat neural network training as a "black-box" for this problem.

1. **Load data.** Load `mismatched_v` and `mismatched_y` data from Problem 3.4. You will train a neural network to learn $f([x_k, x_{k-1}, x_{k-2}]) = f(\mathbf{v}_k) = y_k$. Your network will consist of three input neurons, one output neuron, and at most 10 neurons in hidden layers. *Reshape* `mismatched_v[600][501][3]` to `\mismatched_v[300600][3]` and `\mismatched_y[600][501]` to `\mismatched_y[300600]`. It is critical to keep the input and output data matched by index so that `f(mismatched_v[k]) = mismatched_y[k]` for all k.

2. **Split data.** Use `sklearn.model_selection.train_test_split` to perform an 70/30 split of your data. This is very important and you will learn more about *overfitting* later in this course. You should now have 4 vectors: `train_X[210420][3]`, `train_y[210420][1]`, `test_X[90180][3]`, and `test_y[90180][1]`.

3. **Define network.** Use `sklearn.neural_network.MLPRegressor` to define your neural network layout. Use the ReLU activation function (`activation = 'relu'`). Use `hidden_layer_sizes` to define the layout. **Use a maximum of 10 neurons**. Experiment with different configurations. Vary the depth and the number of neurons at each layer to optimize the performance (below). You may also adjust training parameters. The code snippet below gives suggested values. Note: scikit-learn infers the input and output sizes based on the data so you only need to specify the *hidden layers*. The three input and one output neuron do not count against your 10 neuron *budget*.

4. **Train network.** Use `nn.fit` to train your network. Use **only** the training data (`train_X` and `train_y`). Do not cheat and use your testing data. It may seem to improve accuracy but it will greatly reduce performance against new data (due to *overfitting*).

5. **Evaluate performance.** Use `nn.predict` to produce `predict_y` using `test_X`. Use `test_y` only as a means to evaluate `predict_y`. The goal is to reduce the error between `predict_y` and `test_y`. You should also check the input vs output plot for subsets of the predictions to visually verify network is working. Experiment with different network configurations to improve performance.

6. **Submission.** We will release how to format and submit your work. We will evaluate your network with new data generated from the same stochastic process and then compute the overall performance using the L2 norm (Euclidean distance).

Tips:

1. Use `nn.get_params` and `nn.set_params` to save network state. `nn.train` produces different weights and biases based on the train/test split and the data order. This means

that re-training the same network can produce different results. `nn.get_params` returns a Python `dict` that you can save to a file for submitting. You can use `nn.set_params` to reset your network to the saved state.

2. Try using fewer than 10 neurons in your network.

3. Consider using a loop to *automate* some of your testing.

4. Consider training on only a subset of data if training takes a long time. It is better to increase the `test_size` fraction (try 0.5 or even higher) rather than ignore data.

Sample code:

```

from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split

## LOAD DATA from lms_fun.hdf5

## 1. RESHAPE mismatched_v and mismatched_y to
## V[300600][3] and y[300600]

## 2. 70/30 TESTING AND TRAINING SPLIT
## V_train, V_test, y_train, y_test = train_test_split(V, y, test_size=0.3)

## 3. DEFINE NETWORK
## nn = MLPRegressor(hidden_layer_sizes=(<...>, <...>,), solver='lbfgs',
    activation='relu', alpha=1e-4, random_state=1)

## 4. TRAIN NETWORK
## nn.fit(V_train, y_train)
##   -- OR LOAD SAVED --
## nn.set_params(...)

## 5. EVALUATE NETWORK
## nn.predict(V_test)
## see: sklearn.metrics.mean_squared_error

## * INSPECT PLOTS

## * SAVE STATE: nn.get_params
```

Problem 5: Lienar Classifier: Human vs Computer random sequences

Note: This problem will be worked in class and does not need to be submitted.

1. Use the file `binary_random_sp2019.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.