Assignment (3)

Artificial Intelligence and Machine Learning (24-787 Fall 2022)

Extended Due Date: 10/20/2022(Thu) @ 11:59 pm EST

In case a problem requires programming, it should be programmed in Python. In Programming, you should use plain Python language, unless otherwise stated. For example, if the intention of a Problem is familiarity with numpy library, it will be clearly noted in that problem to use numpy. Please submit your homework through Gradescope.

Submissions: There are two steps to submitting your assignment on Gradescope:

1. HW03 Writeup: Submit a combined pdf file containing the **answers to theoretical questions** as well as the **pdf form of the FILE.ipynb notebooks**.

- Convert the jupyter notebook to a pdf file. Ensure that the submitted notebooks have been run and the cell outputs are visible - **Hint:** Restart and Run All option in the Kernel menu. **Make sure all plots are visible in the pdf.**
- If an assignment has theoretical and mathematical derivation, scan your handwritten solution and make a PDF file.
- Then concatenate them all together in your favorite PDF viewer/editor. The file name (FILE)should be saved as **HW-assignmentnumber-andrew-ID.pdf**. For example for assignment 1, my FILE = HW-1-andrewid.pdf
- Submit this final PDF on Gradescope. During submission, it will prompt you to select the pages of the PDF that correspond to each question, **make sure to tag the questions correctly!**

2. HW03 Code: Submit a ZIP folder containing the FILE.ipynb notebooks for each of the programming questions. The ZIP folder containing your iPython notebook solutions should be named as HW-assignmentnumber-andrew-ID.zip

You can refer to Numpy documentation while working on this assignment. **ANY** deviations from the submission structure shown below would attract penalty to the assignment score. Please use Piazza for any questions on the assignment.

PROBLEM 1

Feature Engineering **[30 points]**

In this problem, you'll get some hands-on experience on a real-world dataset. Assume you already have a model that could predict the house price, you are going to predict the residual error of this model. Specifically, your target is the error in logarithmic scale (**logerror**), and your inputs are some features that may contribute to the house price, like the number of bathrooms, location, etc.

Data Description

You are given three .csv files, namely **train.csv**, **properties.csv** and **data_dictionary.csv**. The **train.csv** includes transaction id, log error, and transaction date. The **properties.csv** includes the transaction id and a list of features. The **data_dictionary.csv** gives some explanation of what each feature means.

a) (Programming problem) Firstly, let's load and visualize data.

1. Use pandas to load **train.csv** and **properties.csv** into two DataFrames. Merge them into a new DataFrame based on the transaction id.

2. Since there are some outliers at both ends of the logerror, replace these outliers with proper maximum/minimum value. Specifically, replace the log error for the smallest 1% of data (the first percentile and below) with the value at the first percentile, and replace the largest 1% of the data (99th percentile and above) with the value at the 99th percentile. (HINT: You can use np.percentile to implement this.)

3. Make a scatter plot by plotting each value of the logerror vector against its index in the vector. Also, make a histogram of **logerror**. (HINT: You should find logerror follows a nice normal distribution. To see this more clearly in the scatter plot, you can decrease the marker size in the plt.scatter command.)

b) (Programming problem) Usually, a dataset will have a lot of missing values (NaN), so we'll first do data cleaning before moving to the next part.

1. Build a new DataFrame that has two columns: **"column_name"** and **"missing_count"**. The **"column_name"** contains every column in merged DataFrame. The **"missing_count"** counts how many missing data that certain column has.
2. Adding a new column called **"missing_ratio"** to this new DataFrame. This new column stores the ratio of number of missing data to the number of total data.
3. Fill the missing data of each feature in merged DataFrame (the df you got from **a**) by its mean value. You can ignore the non-numeric features in this process.

c) (Programming problem) At this point, we can do some univariate analysis. For this problem, we will look at the correlation coefficient of each of these variables.

1. For each variable, compute the correlation coefficient with logerror. After that, sort and make a bar chart of these coefficients.
2. If your bar chart is right, there are few variables at the top of this chart without any correlation values. Explain why.

d) (Programming problem) Now it's time to apply a non-linear regression model.

1. To simplify, in this problem we only use float value features. Therefore, drop the categorical features, "id" and "transactiondate" in your merged dataset.
2. Split your data into train and test following the 70/30 ratio. Calculate the mean, μ and standard deviation, σ of each feature in your training data, X_{train}. Normalize the input data for your model using these values, with the operation $X_{norm} = \frac{X-\mu}{\sigma}$. Train a fully-connected neural network, using **sklearn.neural_network.MLPRegressor** model to predict the logerror based on these normalized input features.

3. Report the Mean Square Error (MSE) of the test set.

PROBLEM 2

Topic: Neural Net **[50 points]**

In this problem, you will be implementing a MultiLayer Perceptron for image classification. Please follow the provided template to build your code, no third-party deep learning packages are allowed in this section (e.g. PyTorch). You are only allowed to use the python libraries **os**, **numpy**, **pandas**, **seaborn** and **matplotlib**.

Fig. 1: Random Images of each of the ten classes in Fashion-MNIST.

Here, you will be classifying images in the Fashion-MNIST dataset, where given a 28 by 28 image of an article of clothing, your model will aim to correctly classify it. As you proceed through this problem, it is helpful to derive the backpropagation steps by hand before implementing the relevant function. The images belong to ten categories [T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot], represented by the value [0, 1, ..., 9] respectively. Here, we'll work with a smaller subset of the Fashion-MNIST dataset, which contains 4000 images with pixel values between 0 and 1. You are provided with the training data (train.csv) and test data (test.csv) that contains the images and labels. Each row contains 785 columns, the first 784 columns are the pixel values, and the last column is the corresponding label.

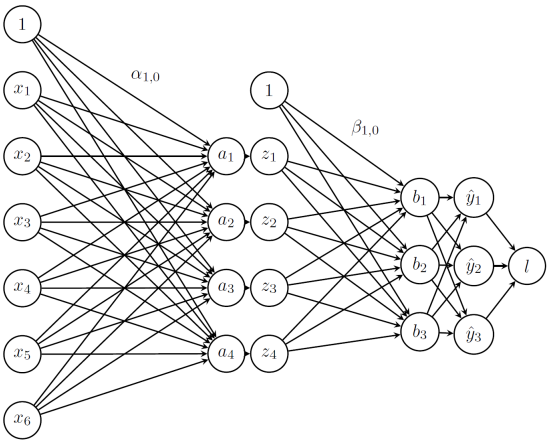


Fig. 2: A smaller version of the neural network you will implement in this question. The number of neurons in each layer, number of input features, and number of predicted classes are smaller than what you will implement in the programming problem, but the overall structure and notation is the same.

We'll be implementing a neural network with one hidden layer, with a sigmoid activation function for the hidden layer, and a softmax operation on the output layer. The input vectors are of length 784, the hidden layer will consist of 256 hidden units, and the output is a probability distribution over the 10 classes. The network can be defined by a parameter matrix α, representing the weights of the first layer, and β, representing the weights of the second layer. Each element, y_k of the output vector, y, represents the probability that x belongs to class k. The following expressions define the forward pass of the network:

- For the output layer:

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}, \quad k \in \{1, \ldots, K\}, \quad \text{(softmax activation)}$$

$$b_k = \beta_{k,0} + \sum_{j=1}^{D} \beta_{kj} z_j, \quad k \in \{1, \ldots, K\}, \quad \text{(pre-activation)}$$

- For the hidden layer:

$$z_j = \frac{1}{1 + \exp(-a_j)}, \quad j \in \{1, \ldots, D\}, \quad (\sigma - \text{activation})$$

$$a_j = \alpha_{j,0} + \sum_{i=1}^{M} \alpha_{ji} x_i, \quad j \in \{1, \ldots, D\}, \quad \text{(pre-activation)}$$

It is possible to compactly express this model by assuming that $x_0 = 1$ is a bias feature on the input and that $z_0 = 1$ is also fixed. In this way, we have two parameter matrices α of shape $D \times (M+1)$ and β of shape $K \times (D+1)$. The extra 0-th column of each matrix (i.e. $\alpha_{\cdot,0}$ and $\beta_{\cdot,0}$) hold the bias parameters. In this question, $D = 256$, representing the 256 hidden units, $= 10$, representing the 10 different classes, and $M = 784$, representing the number of features (pixels) in the input data. With these considerations we have,

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}, \quad k \in \{1, \ldots, K\}$$

$$b_k = \sum_{j=0}^{D} \beta_{kj} z_j, \quad k \in \{1, \ldots, K\}$$

$$z_j = \frac{1}{1 + \exp(-a_j)}, \quad j \in \{1, \ldots, D\}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \quad j \in \{1, \ldots, D\}$$

Since the output corresponds to a probabilistic distribution over the K classes, the objective (cost) function we will use for training our neural network is the average cross entropy,

$$L(\alpha, \beta) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log(\hat{y}_k^{(n)}) \tag{1}$$

over the training dataset,

$$\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}, \quad \text{for } n \in \{1, \ldots N\}$$

You should optimize this objective function using stochastic gradient descent, computing the gradients for each training example with backpropagation. For reproducibility of your results, do not shuffle your data.

a) (Theoretical Problem) **[20 points]**

First, using the chain rule, we will analytically derive some of the values we will need to use to update the neural network weights. Figure 2 contains a diagram of a miniature network that you can use to help visualize your gradient calculations. Pay close attention to the shape of the matrices and vectors produced in this problem. You can assume the calculations in part a are for a single input vector $x^{(n)}$ and output value $y^{(n)}$.

i) Derive an expression for the derivative $\frac{\partial \mathcal{L}}{\partial b_k}$ in terms of $\hat{y}_k$ and y_k. (Hint: you can express the derivative of the softmax function in terms of the softmax function itself).

ii) Derive an expression for the derivative $\frac{\partial \mathcal{L}}{\partial \beta_{k,j}}$ in terms of $\frac{\partial \mathcal{L}}{\partial b_k}$ and z_j. Next, express $\frac{\partial \mathcal{L}}{\partial \beta}$ as a matrix multiplication, in terms of $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ and $\mathbf{z}$. Here, $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ is a K-length vector, containing the values $\frac{\partial \mathcal{L}}{\partial b_k}$. $\mathbf{z}$ is z_j in vector notation, a D-length vector which contains each z_j value.

iii) Derive an expression for the derivative $\frac{\partial \mathcal{L}}{\partial \mathbf{z}}$ in terms of the vector $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ and a modified weight matrix β'. β' is a matrix of shape $K \times D$ only containing the weights, it does not have the first column containing the bias parameters.

iv) Derive an expression for the derivative $\frac{\partial \mathcal{L}}{\partial \alpha_{j,i}}$ in terms of $\frac{\partial \mathcal{L}}{\partial z_j}$, z_j and an input node, $x_i^{(n)}$. Next, express this as a matrix multiplication, writing $\frac{\partial \mathcal{L}}{\partial \alpha}$ in terms of $\frac{\partial \mathcal{L}}{\partial \mathbf{z}}$, $\mathbf{z}$ and an input vector $\mathbf{x}^{(n)}$. $\mathbf{x}^{(n)}$ is a $M+1$ length vector, including a value of 1 to represent the bias term.

b) (Programming Problem) **[5 points]**

Following the instructions given in the notebook template, implement the forward pass of the neural network. To do so, you will be required to implement the functions sigmoid_forward(), linear_forward(), and softmax_xeloss_forward(). Initialize your weights by calling the load_params() function. Use a batch size of 1. The output of the first five nodes for **a**, **b**, and **z** are provided in the folder "check", you may use this to check your implementation.

c) (Programming Problem) **[10 points]**

Following the instructions given in the notebook template and using the default weights, implement the backward pass of the neural network. Use a batch size of 1. Initialize your weights by calling the load_params() function. To do so, you will be required to implement the functions sigmoid_backward(), linear_backward(), and softmax_xeloss_backward().

d) (Programming Problem) **[10 points]**

Implement the training loop of the network, following the instructions in the notebook template. Use a batch size of 1. Initialize your weights by calling the load_params() function. Plot the loss on the training set and the loss on the testing set over 20 epochs on the same figure. Also, show the confusion matrix of your test set predictions using the provided code snippet. For the category 0 (t-shirt), show a sample that was correctly predicted, and a sample that was incorrectly predicted as a different type of clothing. The updated weights for the alpha and beta matrices, after the first data point during the first epoch is seen, are provided in the folder "check". You may use this to check your implementation.

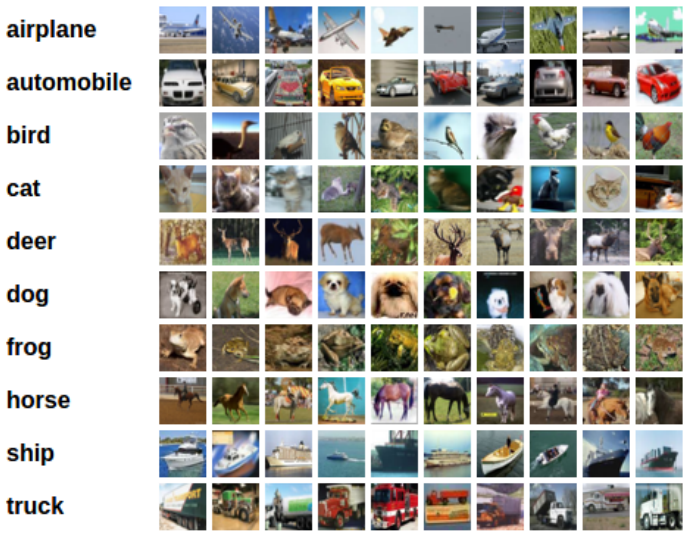
e) (Programming Problem) **[5 points]**

Using a batch size of 50, plot the loss on the testing set against the number of epochs for the learning rates [5e-1,1e-1, 1e-2, 1e-3] on the same figure. Using a batch size of 50, plot the loss on the testing set against the number of epochs, for the three other weight initializations that are mentioned in the template code (weights initialized to zero, random initial weights, weights initialized to one). For both of these plots, the model should be trained for 20 epochs.

PROBLEM 3

CIFAR-10 Classification using CNN **[30 points]**

In this question, you will apply Convolution Neural Networks (CNN) to perform image classification. Specifically, you will use Pytorch to build CNN models and evaluate the accuracy. Sample code for loading in the dataset is provided in the notebook p3.ipynb. Follow the instructions in the sample code and answer the questions below.



a) Training the CNN model with Cross Entropy Loss - 15 points: With the data loaded above, follow the instructions provided in the sample code to build your own CNN model. While you should be able to complete this problem while training on CPU, we recommend using **Google Colaboratory** in this question for faster training. Colab provides a free GPU for you to train a CNN model. The model should follow the following structure:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
              ReLU-2            [-1, 6, 28, 28]               0
            Conv2d-3           [-1, 16, 26, 26]             880
              ReLU-4           [-1, 16, 26, 26]               0
            Conv2d-5           [-1, 24, 24, 24]           3,480
              ReLU-6           [-1, 24, 24, 24]               0
            Linear-7                 [-1, 120]       1,659,000
              ReLU-8                 [-1, 120]               0
            Linear-9                  [-1, 84]          10,164
             ReLU-10                  [-1, 84]               0
           Linear-11                  [-1, 10]             850
================================================================
```

Fig. 3: The CNN structure to implement

.

You can check the configuration of your CNN model with the torchsummary package, using the code provided in the template. If implemented correctly, your accuracy in these questions should be greater than 50% after training. The estimated training time for a shallow CNN model (three convolution layers) is around 10 mins using a Google Colab GPU.

i.) Assuming an input consisting of a three-channel 32x32 image, calculate the kernel size required in each convolution layer to produce the output shapes listed in Figure 3. The output shape format is [-1, num_channels, height, width], where -1 refers to an arbitrary batch size. Assume the stride and padding are the default PyTorch Conv2D values (stride = 1, padding = 0).

ii.) Using the layers calculated in i), implement the CNN using the provided notebook template. Define a learning rate of 1E-3, use the Adam optimizer, and train the model for 20 epochs. Use the softmax cross-entropy loss to train the model.

iii.) Visualize your results by performing the following steps.

- Plot your accuracy versus iterations
- Plot your loss versus iterations
- Show the predictions on five random images from the test set.
- Using the provided code snippet, visualize the first channel of the feature map before the third convolution layer is applied, for the first sample of the first batch.

b) Adding additional components to the CNN - 15 points: In this section, you will perform an ablation study to examine how different components of the CNN influence performance.

i.) Starting with the CNN described in part a), modify the network to include a max pooling layer of factor 2 after the second convolution layer. Plot the accuracy against the number of iterations, along with the accuracy vs iterations for the baseline CNN that you developed in Part a). Visualize the first channel of the feature map before the third convolution layer is applied, for the first sample of the first batch using the provided code snippet. Also, report the memory required for the model. (Note: The memory of the model is displayed in the torchsummary output). Briefly describe what you observe.

ii.) Starting with the CNN described in part b.i.), modify the network to include five layers of convolution, matching the network summary shown below. Plot the accuracy against the number of iterations, along with the accuracy vs iterations for the CNN that you developed in Part b.i) and report the memory required for the model. In 1-2 sentences, describe what you have observed from these metrics, and explain why this behavior occurs.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 6, 28, 28]	456
ReLU-2	[-1, 6, 28, 28]	0
Conv2d-3	[-1, 16, 26, 26]	880
ReLU-4	[-1, 16, 26, 26]	0
MaxPool2d-5	[-1, 16, 13, 13]	0
Conv2d-6	[-1, 24, 11, 11]	3,480
ReLU-7	[-1, 24, 11, 11]	0
Conv2d-8	[-1, 24, 9, 9]	5,208
ReLU-9	[-1, 24, 9, 9]	0
Conv2d-10	[-1, 24, 7, 7]	5,208
ReLU-11	[-1, 24, 7, 7]	0
Linear-12	[-1, 120]	141,240
ReLU-13	[-1, 120]	0
Linear-14	[-1, 84]	10,164
ReLU-15	[-1, 84]	0
Linear-16	[-1, 10]	850

Fig. 4: The CNN structure to implement in b.ii).

.

iii.) Starting with the CNN described in part b.ii.), modify the network to include four times as many output channels for each convolutional layer. Plot the accuracy against the number of iterations, along with the accuracy vs iterations for the CNN that you developed in Part b.ii) and report the memory required for the model. In 1-2 sentences, describe what you have observed from these metrics, and explain why this behavior occurs.