A Feed-Forward Neural Network for Handwritten Digit Recognition

Wang Daming[†, a]

[a]University of Victoria
[b]ECE 503 Optimization for Machine Learning

Abstract This paper shows a small feed-forward neural network for Handwritten Digit Recognition on the MNIST dataset. The code is written in Matlab. I began with the process of loading and preprocessing the data, then convert the digit images into normalized vectors and perform transform class labels into one-hot encodings. The neural network consists of an input layer, a hidden layer, and an output layer. The input layer has 784 input nodes. The hidden layer uses the ReLU activation function, and the output layer uses the softmax function. Together, the model can accuractely predict the digit of the handwritten images from 0 to 9. In order to optimize the model, I use forward and backward passes to compute the loss and gradients to apply mini-batch gradient descent for parameters. The model can achieve a high accuracy on both validation and test sets, which shows the effectiveness of the neural network and the optimization algorithm of our model. The result is shown in the format of a confusion matrix. The complete code is available in the appendix.

Keywords: MNIST; Neural Network; Handwritten Digit Recognition.

1. Introduction

Over the past few years, the field of machine learning met a significant growth especially in the area of deep learning. Moreover, most of the deep learning models are based on neural networks. Since professor metioned how the neural network works in the last few lectures, it made me wonder how we can do with a simple neural networking model and tried to play with it. As for the database, I chose the MNIST since most of the model used to process MNIST dataset is using logistic regression with a fundamental linear model used as a baseline for classification tasks (**?**).

1.1 Useful tools for reference managing

There are several useful tools to help organize your references. Here are some of them:

Submitted on December 11, 2024.
[†]wdm1732418365@gmail.com

(i) https://www.mendeley.com
(ii) https://www.jabref.org
(iii) https://www.zotero.org/
(iv) https://truben.no/latex/bibtex/

2. Methodology

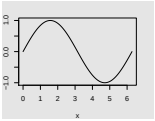
In this section you should discuss the methodology.[1]

2.1 Figures

Figure 1 was generated in R through the following code:

```
x = seq(from=0, to=2*pi, length=100)
cm = 1/2.54 # this is just for defining units of
    measurement
pdf(file='plot.pdf', width=9*cm, height=7*cm, bg=rgb
    (0,0,0,.1))
par(mai = c(2*cm,1*cm,.5*cm,1*cm))
plot(x, sin(x), type ='l')
dev.off()
```

Ideally, the dimensions of the figure should be controlled *outside* of LaTeX, as the preceding R code illustrates. If you cannot generate or obtain the figure with appropriate sizing, then adding the optional argument `[width=9cm]` to the `\includegraphics` command will do the job. Below we illustrate usage of the `\includegraphics[width=2cm]{media/plot.pdf}` command.



2.2 The model

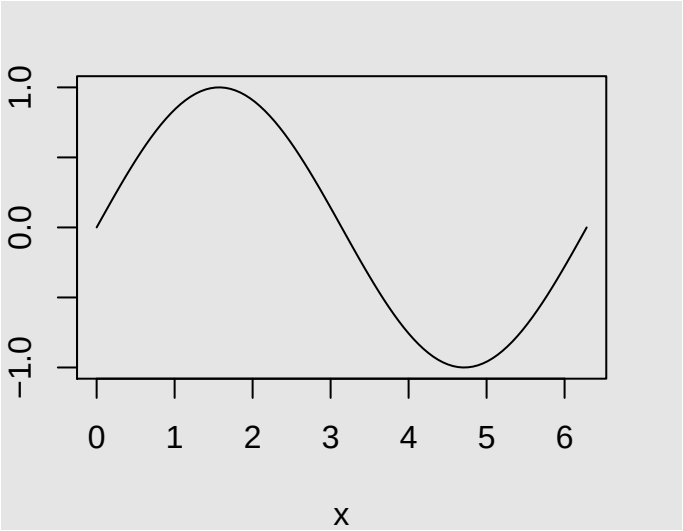
We write inline equations as $x = x$ or displayed equations as

$$\mathrm{d}X_t = \mu \mathrm{d}t + \sigma \mathrm{d}B_t \tag{1}$$

and reference equations using `\eqref{eq1}` to display as equation (1). Maybe we should have added this in section 1. You can also reference theorems, for example `\autoref{thm:1}` will produce Theorem 1.

[1] Footnote links should come after punctuation.

Figure 1
A figure



Definition 1. We say that x **is equal to** x whenever $x = x$.

Lemma 1. $x \geqslant y$ if and only if $y \leqslant x$.

Proof. This is left as an exercise. ■

Proposition 1. $x = x$ if and only if $x = x$.

Theorem 1. If $x = x$ and $y = y$, then $x > y$ implies $x > y$.

Proof. A proof with default title. ■

A proof with custom title. This is trivial. ■

Corollary 1. $x > y$ of and only if $y < x$.

Remark 1. This is a remark.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque

penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

3. Results

You can add tables easily: see Table 1. There are three custom column types that accept width specification: `L`, `C` and `R`, which work similarly to the standard `p` column type; for instance, use `C{4cm}` for a (horizontally) centered column 4cm wide. Notice, however, that LaTeX has some inconsistencies regarding lengths, as Example 1 illustrates. Thus, some manual fine-tuning may be necessary to obtain tables with the desired width.

Table 1
A simple table

variable	value	*p*-value
X	1	0.0
Y	−1	0.8
You can write long texts inside table cells, with custom linebreaks	A	B

Table descriptions go here. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Example 1. This example illustrates length inconsistencies in LaTeX.

a	b	c
x	y	z

The source code yielding the rule and table above is as follows:

```
\noindent\rule{.6\textwidth}{1cm}
\noindent\begin{tabular}{L{.2\textwidth}C{.2\textwidth}R
    {.2\textwidth}}
\toprule
x & y & z\tabularnewline\bottomrule
\end{tabular}
```

3.1 Some additional features

Table 2 illustrates how to align numbers by the decimal place marker. It also shows how to implement the `\multirow` command.

Table 2
Another simple table

variable		value	p-value
X		1.001	0.0
Y		−10.00	0.8
Z	Z_1	1.1	0
	Z_2	2.2	0

Here are two useful tools to help formatting LaTeX tables:

(i) https://www.tablesgenerator.com

(ii) https://truben.no/table/

Acknowledgments Author One would like to thank Institution One for financial support.

Conflict of interest The authors declare no conflict of interest.

Artificial Intelligence This research utilized AI tools to assist in data analysis, manuscript drafting, and figure generation. All AI-generated content was critically reviewed and validated by the authors to ensure accuracy and alignment with the scientific integrity of the study. The use of AI adhered to ethical guidelines, ensuring transparency and compliance with academic standards. Any biases or limitations inherent to the AI tools were carefully considered in the interpretation of results. The authors affirm that the AI tools did not compromise the originality or integrity of the work.

References

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11): 2278–2324.

A. Matlab Code

The complete Matlab Code used in this report is listed below:

loadMNISTImages.m

```
% Load MNIST images from a IDX3-ubyte file
function images = loadMNISTImages(filename)
    % open file with big-endian format
    file = fopen(filename,'r','b');
    % use first 4 bytes to chekc the file type
    if fread(file,1,'int32') ~= 2051
        error('Invalid magic number in %s', filename);
    end
    numImages = fread(file,1,'int32');
    numRows = fread(file,1,'int32');
    numCols = fread(file,1,'int32');
    fprintf('Loading %d images of size %d x %d from %s\n
    ', numImages, numRows, numCols, filename);
    % each pixel is stored as an unsigned byte
    images = fread(file,inf,'unsigned char');
    images = reshape(images, numRows*numCols, numImages)
    ;
    images = double(images)/255;
    fclose(file);
    fprintf('Successfully loaded %d images from %s.\n',
    numImages, filename);
end
```

loadMNISTLabels.m

```
% Load MNIST labels from a IDX1-ubyte file
function labels = loadMNISTLabels(filename)
    fid = fopen(filename,'r','b');
    if fread(fid,1,'int32') ~= 2049
        error('Invalid magic number in %s', filename);
    end
    % read the next 4 bytes
    numLabels = fread(fid,1,'int32');
    labels = fread(fid,inf,'unsigned char');
    fclose(fid);
    fprintf('Loading %d labels from %s\n', numLabels,
    filename);
end
```

foward_pass.m

```
function [Y_hat, cache] = forward_pass(X, W1, b1, W2, b2
    )
    % X: input
    % 784 * N

    % linear combination for hidden layer neurons
    % 100 * N
    Z1 = W1 * X + b1;

    % ReLU activation for hidden layer
    A1 = max(Z1, 0);

    % linear combination for output layer neurons
    % 10 * N
    Z2 = W2 * A1 + b2;

    % softmax function that convert Z2 to probabilities
    exps = exp(Z2 - max(Z2, [], 1));
    Y_hat = exps./sum(exps, 1);

    % store Z1, A1, Z2 for backward_pass function
    cache.Z1 = Z1;
    cache.A1 = A1;
    cache.Z2 = Z2;
end
```

backward_pass.m

```
function [loss, dW1, db1, dW2, db2] = backward_pass(X, Y
    , W1, b1, W2, b2)
    % get the predicted probabilities and Z1, A1, Z2
    from forward_pass
    [Y_hat, cache] = forward_pass(X, W1, b1, W2, b2);

    % average the loss and gradients over all samples
    N = size(X,2);

    % compute cross-entropy loss
    loss = -sum(sum(Y .* log(Y_hat+1e-12)))/N;

    % gradient of loss of Z2 for output layer
    % 10 * N
    dZ2 = (Y_hat - Y);
    % gradient of W2
    % 10 * 100
```

```
    dW2 = (dZ2 * cache.A1')/size(X,2);
    % 10 * 1
    db2 = mean(dZ2,2);
    % 100 * N
    dA1 = W2' * dZ2;
    % ReLU derivative
    dZ1 = dA1 .* (cache.A1>0);
    % 100 * 784
    dW1 = (dZ1 * X')/size(X,2);
    % 100 * 1
    db1 = mean(dZ1,2);
end
```

compute_loss.m

```
function [loss, acc] = compute_loss(X, Y, W1, b1, W2, b2
    )
    [Y_hat, ~] = forward_pass(X, W1, b1, W2, b2);
    N = size(X,2);
    loss = -sum(sum(Y.*log(Y_hat+1e-12)))/N;
    [~, pred] = max(Y_hat, [], 1);
    [~, labels] = max(Y, [], 1);
    acc = mean(pred == labels);
end
```

project.m

```
    % load training and test data
    train_images = loadMNISTImages('train-images.idx3-
    ubyte');
    train_labels = loadMNISTLabels('train-labels.idx1-
    ubyte');
    test_images = loadMNISTImages('t10k-images.idx3-
    ubyte');
    test_labels = loadMNISTLabels('t10k-labels.idx1-
    ubyte');

    % convert labels to one-hot encoding:
    numClasses = 10;
    Y_train = zeros(numClasses, length(train_labels));
    for i = 1:length(train_labels)
        Y_train(train_labels(i)+1, i) = 1;
    end
    Y_test = zeros(numClasses, length(test_labels));
    for i = 1:length(test_labels)
```

```
        Y_test(test_labels(i)+1, i) = 1;
    end

    X_train = train_images; % 784 * 60000
    X_test = test_images; % 784 * 10000

    % create a validation set from training data to verify the correctness on
    % training sets to have a basic idea of accuracy.
    val_size = 5000;
    X_val = X_train(:, end-val_size+1:end);
    Y_val = Y_train(:, end-val_size+1:end);
    X_train = X_train(:, 1:end-val_size);
    Y_train = Y_train(:, 1:end-val_size);

    % initialize Neural Network Parameters
    % We have 28x28 = 784
    inputSize = 784;
    hiddenSize = 100;
    outputSize = 10;

    % make sure the random numbers generated are the same every time
    rng('default');
    % W1 is a matrix of size hiddenSize * inputSize
    % initialize some small random values and scaled by 0.01
    W1 = 0.01*randn(hiddenSize, inputSize);
    b1 = zeros(hiddenSize, 1);
    W2 = 0.01*randn(outputSize, hiddenSize);
    b2 = zeros(outputSize, 1);

    % Mini-Batch Gradient Descent Training
    numEpochs = 5;
    batchSize = 128;
    % change the learning_rate
    learning_rate = 0.87;
    fprintf('Learning Rate is %f \n', learning_rate);
    numTrain = size(X_train, 2);

    for epoch = 1:numEpochs
        % shuffle training data
        idx = randperm(numTrain);
        X_train = X_train(:, idx);
```

```
        Y_train = Y_train(:, idx);

        % loop over mini-batches
        for start_i = 1:batchSize:numTrain
            end_i = min(start_i+batchSize-1, numTrain);
            X_batch = X_train(:, start_i:end_i);
            Y_batch = Y_train(:, start_i:end_i);
            [loss, dW1, db1, dW2, db2] = backward_pass(X_batch, Y_batch, W1, b1, W2, b2);
            W1 = W1 - learning_rate * dW1;
            b1 = b1 - learning_rate * db1;
            W2 = W2 - learning_rate * dW2;
            b2 = b2 - learning_rate * db2;
        end

        % compute loss on training and validation set
        [train_loss, ~] = compute_loss(X_train, Y_train, W1, b1, W2, b2);
        [val_loss, val_acc] = compute_loss(X_val, Y_val, W1, b1, W2, b2);
        fprintf('Epoch %d: Train Loss = %.4f | Val Loss = %.4f | Val Acc = %.2f%%\n', ...
            epoch, train_loss, val_loss, val_acc*100);
    end

    % evaluate on test sets
    [test_loss, test_acc] = compute_loss(X_test, Y_test, W1, b1, W2, b2);
    fprintf('Test Loss = %.4f | Test Accuracy = %.2f%%\n', test_loss, test_acc*100);

    % test set confusion matrix
    [Y_hat_test, ~] = forward_pass(X_test, W1, b1, W2, b2);
    [~, pred_test] = max(Y_hat_test, [], 1);
    [~, true_test] = max(Y_test, [], 1);
    confMat = confusionmat(true_test, pred_test);
    disp('Confusion Matrix:');
    disp(confMat);

    % normalize the confusion matrix
    confMatNorm = bsxfun(@rdivide, confMat, sum(confMat,2));
    imagesc(confMatNorm);
```

```
    colorbar;
    title('Normalized Confusion Matrix');
    xlabel('Predicted Digit');
    ylabel('True Digit');
    axis square;
```

Output:

```
Loading 60000 images of size 28 x 28 from train-images.idx3-ubyte
Successfully loaded 60000 images from train-images.idx3-ubyte.
Loading 60000 labels from train-labels.idx1-ubyte
Loading 10000 images of size 28 x 28 from t10k-images.idx3-ubyte
Successfully loaded 10000 images from t10k-images.idx3-ubyte.
Loading 10000 labels from t10k-labels.idx1-ubyte
Learning Rate is 0.870000
Epoch 1: Train Loss = 0.1412 | Val Loss = 0.1168 | Val Acc = 96.40%
Epoch 2: Train Loss = 0.1053 | Val Loss = 0.0995 | Val Acc = 97.18%
Epoch 3: Train Loss = 0.0941 | Val Loss = 0.0977 | Val Acc = 97.06%
Epoch 4: Train Loss = 0.0567 | Val Loss = 0.0797 | Val Acc = 97.42%
Epoch 5: Train Loss = 0.0577 | Val Loss = 0.0808 | Val Acc = 97.60%
Test Loss = 0.0945 | Test Accuracy = 97.26%
```

R