On a simple matrix based neural network model

NAME

Summary

This short paper will be examining a small scale feed-forward neural network, comprising of 2 layers of neurons, comprising of a layer with 800 and one of 10.

Contents

Introduction

Neural networks (NNs) are an extremely powerful tool. The standard model features a number of layers, with a feed-forward model and a back-propogation algorithm in order to "train" each layer in the model. The NN being discussed will be a classification NN, with a purpose of recognizing a handwritten digit. The (nth) section wil be examining the behaviour of a NN, the (nth) section will be examining the feed-forward aspect of the NN, and the (nth) sections will be examining the back-propogation algorithm. Finally, in the (nth) section, we will be discussing the application of batches to NN's, as well as how the gradient changes when dealing with many images.

1 The behaviour of a neural network

This section will be discussing the behaviour of a neural at a high level.

1.1 The input and output of a neural network

A NN, in principle, is a function. The NN takes in some form of an input, and returns the NN's certainty that it was a certain digit. A network of this form could thus be represented as a map:

$$f : \mathbb{R}^{1\times m} \to \mathbb{R}^{1\times n} \tag{1}$$

where m represents the dimension. The output, $\mathbf{Z}$, represents the confidence that the number is the index position. That is,

$$\% certainty_j = \mathbf{Z}_j$$

1.2 Traits of the inner layers of the NN

Each layer has a number of neurons. Each neuron has a number of weights, as well as a bias term. Intuitively, the weights and bias modify the input such that the output can be normalized to find the certainty of each digit. The NN being examined will have 2 layers, with 800 neurons in the first layer, while having 10 neureons in the second layer.

1.3 The shape of the input vector

As seen before in equation (1), the function f is defined to take $\mathbb{R}^{1\times m}$ to $\mathbb{R}^{1\times n}$. This choice of variables was intentional. Our input variable, $\mathbf{x}$, will be of form $\mathbb{R}^{1\times m}$, where $m = 768$, in order to fit the input data.

1.4 The shape of the weight and bias terms in a single neuron

For now, let's define the weight term to be of form $\mathbb{R}^{768\times 1}$, and the bias term to be of form $\mathbb{R}$. The reasoning for these dimension choices will be justified in the next section.

1.5 The shape of the weight and bias terms in a layer

Since a layer comprises multiple neurons, the weight and bias term can be expressed as a vector or vectors, or a matrix. We wil therefore choose the weight and bias of an arbitrary layer j to be $\mathbb{R}^{n_{j-1}\times n_j}$ and $\mathbb{R}^{1\times n_j}$ respectively, where n_j denotes the number of neurons in layer j

2 The feedforward algorithm

This section will be demonstrating the feedforward algorithm.
Matrix and vector addition in this section will be defined according to appendix (B)

2.1 The variables in each layer of a NN

Each layer in a NN will have 5 variables. The 5 variables are the following: the input, the net, the output, the weight, and the bias. The input, weight and bias combine to form the net, which is then run through an activation function to obtain out. From the Formula and notation section, r will be the activation function for layer 1, with s for layer 2.

2.2 The feedforward function

Given a layer, define net to be

$$\mathbf{net} = \mathbf{input} \cdot \boldsymbol{weights} + bias \tag{2}$$

Let us denote the activation function of layer j to be $a_j(\mathbf{z})$. Then, in accordance with section (2.1), the out will be defined to be

$$\mathbf{out} = a(\mathbf{net}) \tag{3}$$

Combining equations (2) and (3) yields

$$\mathbf{out} = a(\mathbf{input} \cdot \boldsymbol{weights} + bias) \tag{4}$$

Using equation (4), we can the find the output of the second layer to be

$$\mathbf{out_2} = s(a(\mathbf{input}_1 \cdot \boldsymbol{weights}_1 + bias_1) \cdot \boldsymbol{weights}_2 + bias_2) \tag{5}$$
$$\implies \mathbf{out_2} = s(\mathbf{net}_2) \tag{6}$$

2.3 What is loss?

Loss is defined to be a value that represents how far off $\mathbf{out}_2$ is from the correct input. This function is denoted $L_{\mathbf{y}}(\mathbf{out}_2)$ where $\mathbf{y}$ is an one hot encoded vector (see appendix A) and the definition can be found in the Formula and notation section.

2.4 The shape of the weight and bias terms in a layer(2)

In section (1.4), we defined the shape of the weight term to be $\mathbb{R}^{n_{j-1} \times n_j}$ Now that we have introduced the loss calculation, we can justify this shape. Let the weight matrix in layer 1 be $\mathbb{R}^{a \times b}$ Substituting the dimensions of the variables in layer 1 into equation (2) yields

$$\mathbf{out} = \mathbb{R}^{1 \times 768} \cdot \mathbb{R}^{a \times b} + \mathbb{R} \tag{7}$$

by the definition of matrix multiplication, we see that a must be 768.
out in this case, is of shape 1xb, meaning that the weight matrix in layer 2 must have shape bxc, where c is some arbitrary constant. In that case, $\mathbf{out}_2$ is of shape 1xc. For convenience purposes, we will consider c to be 10.

3 The back propogation algorithm

In this section, we will be examining the back propogation algorithm

3.1 Demonstration of the multivariate chain rule

Given some function $f(g(\mathbf{t}))$, the derivative with respect to $\mathbf{t}$ is

$$\frac{\partial f}{\partial \mathbf{t}} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial \mathbf{t}}$$

This is very important for us as it allows us to find the derivative with respect to each paramater relatively easily.

3.2 Minimizing loss

As mentioned in section (2.3), loss is the measure of how different the output was compared to the correct output. Informally, this can be thought of as how incorrect the output is. Naturally, the goal would be to reduce the loss, in order to make the order more correct. This can be done by finding the so called gradient, and subtracting that from every matrix. This is represented with the general formula:

let $\mathbf{a}^n$ denote the nth iteration of $\mathbf{a}$.
given some $f(\mathbf{a})$, the **local** minimum can be found with the following iterative process:
$\mathbf{a}^{n+1} = \mathbf{a}^n - \eta\nabla_{\mathbf{a}^n} f$
where $\eta \in (0, 1)$ is the learning rate of the network and $\nabla_{\mathbf{a}^n}$ is the gradient with respect to $\mathbf{a}^n$(see appendix D)

3.3 Applying the chain rule to the loss function

In this subsection, I will be demonstrating how to use the chain rule to find the partial derivative with respect to each non-input variable, i.e weights and biases.

$$\frac{\partial L}{\partial \boldsymbol{w}_2} = \frac{\partial L}{\partial \mathbf{out}_2}\frac{\partial \mathbf{out}_2}{\partial \mathbf{net}_2}\frac{\partial \mathbf{net}_2}{\partial \boldsymbol{w}_2}$$
$$\frac{\partial L}{\partial \mathbf{b}_2} = \frac{\partial L}{\partial \mathbf{out}_2}\frac{\partial \mathbf{out}_2}{\partial \mathbf{net}_2}\frac{\partial \mathbf{net}_2}{\partial \mathbf{b}_2}$$

As it is clear, $\frac{\partial L}{\partial \mathbf{out}_2}\frac{\partial \mathbf{out}_2}{\partial \mathbf{net}_2}$ appears quite often. Denote this quantity δ_2, and the equations simplify to

$$\frac{\partial L}{\partial \boldsymbol{w}_2} = \delta_2 \frac{\partial \mathbf{net}_2}{\partial \boldsymbol{w}_2}$$
$$\frac{\partial L}{\partial \mathbf{b}_2} = \delta_2 \frac{\partial \mathbf{net}_2}{\partial \mathbf{b}_2}$$

Now, let's consider the derivatives with respect to the values in the first layer

$$\frac{\partial L}{\partial \boldsymbol{w}_1} = \frac{\partial L}{\partial \mathbf{out}_2}\frac{\partial \mathbf{out}_2}{\partial \mathbf{net}_2}\frac{\partial \mathbf{net}_2}{\partial \mathbf{out}_1}\frac{\partial \mathbf{out}_1}{\partial \mathbf{net}_1}\frac{\partial \mathbf{net}_1}{\partial \boldsymbol{w}_1}$$
$$\frac{\partial L}{\partial \mathbf{b}_1} = \frac{\partial L}{\partial \mathbf{out}_2}\frac{\partial \mathbf{out}_2}{\partial \mathbf{net}_2}\frac{\partial \mathbf{net}_2}{\partial \mathbf{out}_1}\frac{\partial \mathbf{out}_1}{\partial \mathbf{net}_1}\frac{\partial \mathbf{net}_1}{\partial \mathbf{b}_1}$$

Again, let us simplify this expression.

$$\delta_1 \equiv \frac{\partial L}{\partial \mathbf{out}_2}\frac{\partial \mathbf{out}_2}{\partial \mathbf{net}_2}\frac{\partial \mathbf{net}_2}{\partial \mathbf{out}_1}\frac{\partial \mathbf{out}_1}{\partial \mathbf{net}_1}$$

As is clear, this can be defined as the following.

$$\delta_1 = \delta_2 \frac{\partial \mathbf{net}_2}{\partial \mathbf{out}_1}\frac{\partial \mathbf{out}_1}{\partial \mathbf{net}_1}$$

In our network, this is the extent of what we need. However, a general form is provided in appendix E.

3.4 Calculating the partial derivatives

Due to the nature of the L function and the s function, it is difficult to obtain each derivative in the $delta_2$ term. Instead of each step of the partial differential, we will skip directly to $\frac{\partial L}{\partial \mathbf{net}_2}$. A derivation of this can be found in appending A.F. This value is equal to $(\mathbf{out}_2 - \mathbf{y})$. This implies that

$$\delta_2 = (\mathbf{out}_2 - \mathbf{y})$$

Now that we have this information, let us calculate δ_1.

$$\delta_1 = \delta_2 \frac{\partial \mathbf{net}_2}{\partial \mathbf{out}_1}\frac{\partial \mathbf{out}_1}{\partial \mathbf{net}_1}$$
$$\frac{\partial \mathbf{net}_2}{\partial \mathbf{out}_1} = \boldsymbol{w}_2^T$$
$$\implies \delta_1 = ((\mathbf{out}_2 - y) \cdot \boldsymbol{w}_2^T) \otimes \frac{\partial \mathbf{out}_1}{\mathbf{net}_1}$$

with this information, it the rest of the required derivatives are easy to calculate;

$$\frac{\partial L}{\partial \boldsymbol{w}_2} = \delta_2 \frac{\partial \mathbf{net}_2}{\partial \boldsymbol{w}_2}$$
$$\frac{\partial L}{\partial \mathbf{b}_2} = \delta_2 \frac{\partial \mathbf{net}_2}{\partial \mathbf{b}_2}$$
$$\frac{\partial L}{\partial \boldsymbol{w}_1} = \delta_1 \frac{\partial \mathbf{net}_1}{\partial \boldsymbol{w}_1}$$
$$\frac{\partial L}{\partial \mathbf{b}_1} = \delta_1 \frac{\partial \mathbf{net}_1}{\partial \mathbf{b}_1}$$

4 Formulae and notation

$\mathbb{R}^{a\times b}$ denotes matricies of dimension $a \times b$ over the field $\mathbb{R}$
All figures marked in **bold** denote row vectors
All figured marked in ***bold italics*** denote matricies;
Given some function f:$\mathbb{R}^{1\times n} \to \mathbb{R}^{1\times m}$, $\frac{\partial f}{\partial \mathbf{z}} = \nabla_{\mathbf{z}} f$
$\mathbf{Z}_j$ represents the value at the j-th index of $\mathbf{Z}$
Activation function of layer 1: $r(\mathbf{z}) : \mathbb{R}^a \to \mathbb{R}^a = [max(0, \mathbf{z}_j)]_j$
Activation function of layer 2: $s(\mathbf{z}) : \mathbb{R}^a \to \mathbb{R}^a = [\frac{e^{\mathbf{z}_j}}{\sum_{n=0}^{a} e^{\mathbf{z}_n}}]_j$
Loss function: $L_{\mathbf{y}}(\mathbf{z}) : \mathbb{R}^a \to \mathbb{R} = -\sum_{n=0}^{a} \mathbf{y}_n ln(s(\mathbf{z})_n)$
Kronecker delta: $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

5 Appendix

A

An one hot encoded vector is a variable $\mathbf{y} \in \mathbb{Z}^{n+}$, where $\mathbb{Z}^{n+}$ denotes the integers greater than, or equal to 0, such that the sum of the elements of $\mathbf{y} = 1$

B

Initially, an addition between $\mathbb{R}^{m\times 1}$ and $\mathbb{R}$ seems ill defined. we will then define it as:
let $a \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^{1\times n}$

define $a+b$ to be $\begin{pmatrix} a_1+b \\ a_2+b \\ \vdots \\ a_m+b \end{pmatrix}$, where a_j denotes the j'th row of a and a_j+b is standard vector addition

C

Since differentiation with respect to a matrix is ill-defined under many formalisms of calculus, we will set some ground rules:

$$\frac{\partial \mathbf{x}\,\boldsymbol{W}}{\partial \mathbf{x}} = \boldsymbol{W}^T$$
$$\frac{\partial \mathbf{x}\,\boldsymbol{W}}{\partial \boldsymbol{W}} = \mathbf{x}^T$$

Another rule we will set:

$$\begin{aligned} &\text{let } f : \mathbb{R}^j \to \mathbb{R} \\ &\text{let } \boldsymbol{W} \in \mathbb{R}^{m\times n} \\ \implies &\tfrac{\partial f}{\partial \boldsymbol{W}} \in \mathbb{R}^{m\times n} \end{aligned}$$

Since the chain rule raises many vector valued arguments, we will need to do products thereof elementwise, in order to maintain the shape of the arguments That is, an operation $\otimes$ will be used when multiplying differentials. An example being some function $f(g(x))$:

$$\frac{\partial f}{\partial \mathbf{t}} = \frac{\partial f}{\partial g} \otimes \frac{\partial g}{\partial \mathbf{t}}$$

This will be shortened to

$$\frac{\partial f}{\partial g}\frac{\partial g}{\partial \mathbf{t}}$$

for the sake of brevity.
An important thing to note is that the hadamard product behaves differently under certain conditions.
Let us consider $\frac{\partial L}{\partial a}\frac{\partial a}{\partial \boldsymbol{W}}$. Contrary to the claim made above, this is equivalent to $\boldsymbol{W}^T \cdot \frac{\partial L}{\partial a}$

$$J_{\mathbf{z}}(s) = \begin{bmatrix} \frac{\partial s_1}{\partial \mathbf{z}_1} & \cdots & \frac{\partial s_1}{\partial \mathbf{z}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial s_n}{\partial \mathbf{z}_1} & \cdots & \frac{\partial s_n}{\partial \mathbf{z}_n} \end{bmatrix}$$

D

The gradient of a function $f(\mathbf{z}) : \mathbb{R}^n \to \mathbb{R}$ with respect to some vector $\mathbf{a}$ is notated $\nabla_{\mathbf{a}}$ and is defined $\begin{bmatrix} \frac{\partial f}{\partial \mathbf{z}_1}(\mathbf{a}_1) \\ \frac{\partial f}{\partial \mathbf{z}_2}(\mathbf{a}_2) \\ \vdots \\ \frac{\partial f}{\partial \mathbf{z}_n}(\mathbf{a}_n) \end{bmatrix}$

E

Assuming a m layer network, δ_n has a recursive formula: $\delta_n = \begin{cases} \frac{\partial L}{\partial \mathbf{out}_n} \frac{\partial \mathbf{out}_n}{\partial \mathbf{net}_n} & n = m \\ \delta_{n+1} \frac{\partial \mathbf{net}_{n+1}}{\partial \mathbf{out}_n} \frac{\partial \mathbf{out}_n}{\partial \mathbf{net}_n} & n \neq m \end{cases}$

F

Denote the activation function of the second layer $s(\mathbf{z})$. Since this is a vector valued function, returning a vector, we expect the derivative to be the Jacobian matrix, a matrix denoted $J_{\mathbf{z}}(s)$. The following derivation will show that. Denote s_k to be the kth element of the output of s.

Note that since $s_i = \frac{e^{\mathbf{z}_i}}{\sum_{n=0}^{dim(z)} e^{\mathbf{z}_n}}$, it is easier to use logarithmic differentiation

$$\begin{aligned}
\frac{\partial ln(s_i)}{\partial \mathbf{z}_j} &= \frac{1}{s_i}\frac{\partial s_i}{\partial \mathbf{z}_j} \\
\implies \frac{\partial s_i}{\partial \mathbf{z}_j} &= \frac{\partial ln(s_i)}{\partial \mathbf{z}_j} s_i \\
&= s_i \frac{\partial}{\partial \mathbf{z}_j} ln\left(\frac{e^{\mathbf{z}_i}}{\sum_{n=0}^{dim(z)} e^{\mathbf{z}_n}}\right) \\
&= s_i \frac{\partial}{\partial \mathbf{z}_j}(ln(e^{\mathbf{z}_i}) - ln(\sum_{n=0}^{dim(z)} e^{\mathbf{z}_n})) \\
&= s_i \frac{\partial}{\partial \mathbf{z}_j}(\mathbf{z}_i - ln(\sum_{n=0}^{dim(z)} e^{\mathbf{z}_n})) \\
&= s_i(\delta_{ij} - \frac{\partial}{\partial \mathbf{z}_j} ln(\sum_{n=0}^{dim(z)} e^{\mathbf{z}_n})) \\
&= s_i(\delta_{ij} - \frac{1}{\sum_{n=0}^{dim(z)} e^{\mathbf{z}_n}} e^{\mathbf{z}_j}) \\
&= s_i(\delta_{ij} - s_j)
\end{aligned}$$

Denote the loss function with respect to some one hot encoded vector $\mathbf{y}$ $L_{\mathbf{y}}(\mathbf{z})$
In the following derivation, we will be differentiating L with respect to each

component of $\mathbf{z}$ and building it back into a vector at the end

$$\begin{aligned}
\frac{\partial L_{\mathbf{y}}}{\partial \mathbf{z}_j} &= -\frac{\partial}{\partial \mathbf{z}_j} \sum_{n=0}^{a} \mathbf{y}_n ln(s(\mathbf{z})_n) \\
&= -\sum_{n=0}^{a} \mathbf{y}_n \frac{\partial}{\partial \mathbf{z}_j} ln(s(\mathbf{z})_n) \\
&= -\sum_{n=0}^{a} \mathbf{y}_n \frac{\partial}{\partial \mathbf{z}_j} ln(s(\mathbf{z})_n) \\
&= -\sum_{n=0}^{a} \mathbf{y}_n \frac{\partial}{\partial \mathbf{z}_j} ln(s_n) \\
&= -\sum_{n=0}^{a} \mathbf{y}_n \frac{1}{s_n} \frac{\partial s_n}{\partial \mathbf{z}_j} \\
&= -\sum_{n=0}^{a} \mathbf{y}_n \frac{1}{s_n} s_n(\delta_{nj} - s_j) \\
&= -\sum_{n=0}^{a} \mathbf{y}_n(\delta_{nj} - s_j) \\
&= -\mathbf{y}_j - \sum_{n=0}^{a} -s_j \mathbf{y}_n \\
&= -\mathbf{y}_j + s_j \sum_{n=0}^{a} \mathbf{y}_n \\
&= s_j - \mathbf{y}_j
\end{aligned}$$