Build and Train a Neural Network with Spreadsheet/Excel Native Functions

Yalong Pi, Snehashis Chakraborty

March 7, 2024

1 Introduction

This document shows how to build a neural network using Microsoft Excel or Google spread sheet without any other plugins. The purpose is to illustrate the internal calculation steps to enhance the general understanding of machine learning techniques that are based on neural networks. This method is for education purpose and cannot replace the established package such as PyTorch or TensorFlow.

The toy dataset in Table 1 represents three input (x) output (y) data points. Instead of linear regression with the least square method, we want to try a neural network to approximate the relationship between x and y, such that given another x we can predict the y value with some accuracy.

Consider a simple neural network below in Figure 1 that has one input (x), two neurons, and one output (y) to perform gradient descent on the dataset. We assume no bias, and name the weights w_{11}, w_{12}, w_{21}, w_{22}, and the intermediate output o_1 and o_2.

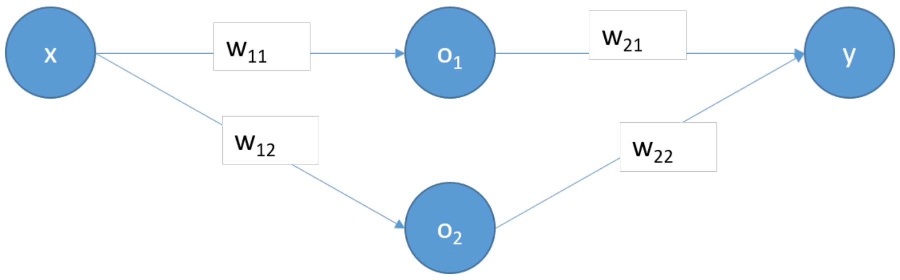

Figure 1: Neural network structure with 2 neurons in the hidden layer.

The neural network forward pass is straight forward to calculate using the following equations 1-3. This process is easy to set up in spreadsheets. Using the intermediate values in the forward pass, the gradient can be expressed in a way that reuses the values already computed.

$$o_1 = w_{11} \cdot x \tag{1}$$

$$o_2 = w_{12} \cdot x \tag{2}$$

$$y = w_{21} \cdot o_1 + w_{22} \cdot o_2 \tag{3}$$

x	y
1	10
2	20
5	30

Table 1: Data

2 Loss Function and Gradient

With randomly initialized weights we can get the forward pass results and save the w_{11}, w_{12}, w_{21}, w_{22}, and the intermediate output o_1 and o_2. Next, we want to update each weight using gradient descent with a Learning Rate (LR), e.g., 0.001.

Here, we use a loss function of square error, Equation 4, for simpler expressions. It is worth to note that this loss function work with individual data pairs and batches. For a batch with n samples, the batch loss is the value of averaging all sample losses.

$$loss = (y - y_{gt})^2 \tag{4}$$

Here, we take w_{22} updating with one training sample as an example. Equations 5 expresses it gradient descent process in one step, where $\cdot'$ means the updated weight.

$$w'_{22} = w_{22} - LR \cdot \frac{\partial loss}{\partial w_{22}} \tag{5}$$

We can write out the loss with all known values in the forward pass that we calculated: w_{11}, w_{12}, w_{21}, w_{22}, and the intermediate output o_1 and o_2, plus the ground truth (gt) output values y_{gt}.

$$\begin{aligned} loss &= (y - y_{gt})^2 \\ &= y^2 - 2y \cdot y_{gt} + y_{gt}^2 \\ &= (w_{21} \cdot o_1 + w_{22} \cdot o_2)^2 - 2w_{21} \cdot o_1 \cdot y_{gt} - 2w_{22} \cdot o_2 \cdot y_{gt} + y_{gt}^2 \\ &= w_{21}^2 \cdot o_1^2 + 2w_{21} \cdot o_1 \cdot w_{22} \cdot o_2 + w_{22}^2 \cdot o_2^2 - 2w_{21} \cdot o_1 \cdot y_{gt} - 2w_{22} \cdot o_2 \cdot y_{gt} + y_{gt}^2 \end{aligned} \tag{6}$$

We treat w_{22} as the variable and everything else constant to get the partial derivatives of the loss with respect to w_{22}: $\frac{\partial loss}{\partial w_{22}}$

$$\begin{aligned} \frac{\partial loss}{\partial w_{22}} &= 2w_{21} \cdot o_1 \cdot o_2 + 2w_{22} \cdot o_2^2 - 2o_2 \cdot y_{gt}) \\ &= 2o_2(w_{21} \cdot o_1 + w_{22} \cdot o_2 - y_{gt})) \\ &= 2 \cdot o_2 \cdot (y - y_{gt}) \end{aligned} \tag{7}$$

We can also use chain rule to get the same results. Notice that y, y_{gt}, and o_2 is all we need to compute w'_{22}.

$$w'_{22} = w_{22} - LR \cdot \frac{\partial loss}{\partial w_{22}} = w_{22} - LR \cdot \frac{\partial loss}{\partial y} \cdot \frac{\partial y}{\partial w_{22}} = w_{22} - LR \cdot 2 \cdot (y - y_{gt}) \cdot o_2 \tag{8}$$

Now we do the same for all wights using the existing values in the forward pass. Note that in the forward pass we used w_{21} and w_{22}.

$$w'_{21} = w_{21} - LR \cdot \frac{\partial loss}{\partial w_{21}} = w_{21} - LR \cdot 2 \cdot o_1 \cdot (y - y_{gt}) \tag{9}$$

$$w'_{11} = w_{11} - LR \cdot \frac{\partial loss}{\partial y} \cdot \frac{\partial y}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_{11}} = w_{11} - LR \cdot 2 \cdot (y - y_{gt}) \cdot w_{21} \cdot x \tag{10}$$

$$w'_{12} = w_{12} - LR \cdot \frac{\partial loss}{\partial y} \cdot \frac{\partial y}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_{12}} = w_{12} - LR \cdot 2 \cdot (y - y_{gt}) \cdot w_{22} \cdot x \tag{11}$$

3 Implementation

Using the reasoning above, we can construct a table like Table 2 where each column is one training epoch starting with randomly initialized weights. Row ”x” and ”y” repeat every epoch serving as the training data. These data and the updating mechanisms will repeat for many following epochs by using the ”Copy a formula” function in spreadsheet or Excel.

The key is actually the second epoch where we set up the weight updating rules based on the first epoch results (forward pass). The cells in Table 2 that have Eq. indicate such rules need to be set up using the corresponding equations. The actual numbers are the actual computing results.

Equation 12-15 are spelled out again for each data point so we can use them to set up the spreadsheet math.

$$g_{11} = 2 \cdot (y - y_{gt}) \cdot w_{21} \cdot x \tag{12}$$

$$g_{12} = 2 \cdot (y - y_{gt}) \cdot w_{22} \cdot x \tag{13}$$

$$g_{21} = 2 \cdot o_1 \cdot (y - y_{gt}) \tag{14}$$

$$g_{22} = 2 \cdot (y - y_{gt}) \cdot o_2 \tag{15}$$

In batch, the batch gradient (bg) is calculatedly by averaging the gradients over all n examples, using the gradient (g) calculated previously, i.e., g_{11}, g_{12} , g_{21}, and g_{22}. Use the following for the spreadsheet.

$$bg_{11} = \frac{1}{n} \sum_{i=1}^{n} g_{11}^{n} \tag{16}$$

$$bg_{12} = \frac{1}{n} \sum_{i=1}^{n} g_{12}^{n} \tag{17}$$

$$bg_{21} = \frac{1}{n} \sum_{i=1}^{n} g_{21}^{n} \tag{18}$$

$$bg_{22} = \frac{1}{n} \sum_{i=1}^{n} g_{22}^{n} \tag{19}$$

This is the setup for Epoch one. The following epoch will use the previous batch gradients and Equation 8-11 to update the weights w_{11}, w_{12}, w_{21}, w_{22}. Notice all the weights are shared by the neural network so it takes three cells in a row. To simplify the process, activation functions are not used, i.e., this is still a linear approximation.

epoch	1			2			...
x	1	2	5	1	2	5	...
y	10	20	30	10	20	30	...
w11	0.3			Eq.10			...
w12	0.1			Eq. 11			...
o1	0.3	0.6	1.5	0.41388	0.82776	2.0694	...
o2	0.1	0.2	0.5	0.188573	0.377147	0.942867	...
w21	0.9			Eq. 9			...
w22	0.7			Eq. 8			...
y_pred	0.34	0.68	1.7	0.52259	1.045181	2.612951	...
loss	93.3156	373.2624	800.89	89.82129	359.2852	750.0504	...
g11	Eq. 12	Eq. 12	Eq. 12	-17.7789	-71.1154	-256.88	...
g12	Eq. 13	Eq. 13	Eq. 13	-13.5082	-54.0329	-195.175	...
g21	Eq. 14	Eq. 14	Eq. 14	-7.84502	-31.3801	-113.35	...
g22	Eq. 15	Eq. 15	Eq. 15	-3.57437	-14.2975	-51.6447	...
bg11	Eq. 16			-115.2579574			...
bg12	Eq. 17			-87.57193007			...
bg21	Eq. 18			-50.85820654			...
bg22	Eq. 19			-23.17217922			...
batch loss	422.4893333			399.7189663			...

Table 2: Excel example with the first two epochs. The key is the second epoch which set the update mechanism with the gradients from previous step. This will repeat in the following epochs.

Altogether, we can see the training progress with batch losses in Figure 2 with 20 epochs or so. We can deploy the model by using the latest weights and Equation 1-3 to predict any input value we are interested in.

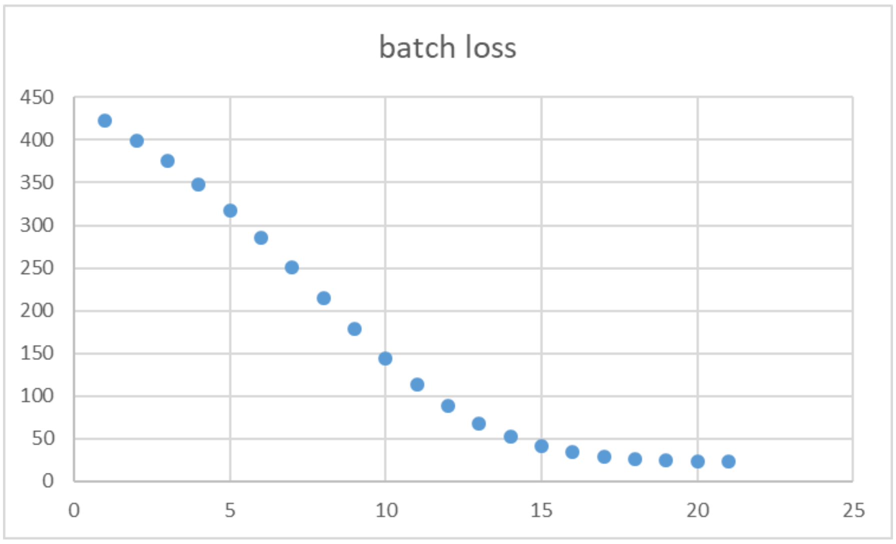

Figure 2: Expected training batch loss over epoch plot.