Saaif Ahmed

661925946

ahmeds7

3/14/2022

Introduction to Deep Learning: Programming Assignment #2 Report

1: Multi-Class Logistic Regression Learning Problem Statement

For a scenario where $y \in \{1, \dots, K\}$ we can use a multi-classifier logistic regression learning model to build a reasonably accurate system that can predict y by using a series of weights $\vec{w}_1, \dots, \vec{w}_k$ in a matrix Θ such that if the data set is $N \times M$ then $W^{M \times K}$ with each $\overrightarrow{w_i}^{M \times 1}$ and passing them into a soft max function for logistic regression.

For this model we want to learn the parameters to identify images of digits from 0-9. We advance this model by constructing a neural network of 1 input layer, 2 hidden layers, and 1 output layer. We want to learn the parameters $\Theta = [W_1, \vec{w}_1, W_2, \overrightarrow{w_2}, W_3, \overrightarrow{w_3}]$. Where $W_1 \rightarrow W_3$ are weight matrices $W_i = [\overrightarrow{w_1}, \dots \overrightarrow{w_k}]$ for each of the classes k. W_1 is for the input layer to 1st hidden layer, W_2 is for the 1st hidden layer to the 2nd hidden layer, and W_3 is for the 2nd hidden layer to the output layer. $\overrightarrow{w_i}$ is the bias vector for the requisite layer weight matrix.

2: Model Description

The data is a series of images of 28x28 pixel size. We have both hidden layers use the ReLu activation function and the output layer uses the softmax activation function.

ReLu Function is as follows: $ReLu(x) = \max\{0, x\}$

Softmax Function is as follows: $\sigma_M(\vec{x}^T[m]\Theta_k) = \frac{e^{\vec{x}^T[m]\Theta_k}}{\sum_{k=1}^{K} e^{\vec{x}^T[m]\Theta_k}}$

The hidden layers have 100 nodes each and the output layer has 10 nodes, one for each digit. For each class k the function determines the value of the node through the functions computation for that class k. The output is then determined by the node with the largest value in that output layer.

To optimize this model we minimize the cross-entropy loss function. We define the negative log likelihood as

$$L(\vec{y}[m], \hat{y}[m]) = -\log\big(p(\vec{y}[m]|\vec{x}[m])\big) = -\sum_{\{k=1\}}^{K} \vec{y}[m][k] * \log(\hat{y}[m][k])$$

To find the optimized parameters we minimize the loss function. To do so we define the forward computation of the neural network. With input layer X and hidden layers H_1 , H_2 and output layer $\hat{y}$ we have that:

$$H_1 = ReLu((W_1)^T X + \overrightarrow{w_1})$$
$$H_2 = ReLu((W_2)^T H_1 + \vec{w}_2)$$

$$\hat{y} = \sigma_M((W_3)^T H_2 + \overrightarrow{w_3})$$

Now we back propagate through the layers to find the gradients $\nabla W_{1,2,3}$, $\nabla \vec{w}_{1,2,3}$ to learn the parameters.

We have that $\hat{y}[m] = \sigma((W_3)^T H_2 + \overrightarrow{w_3})$.

We call $\vec{z}[m] = (W_3)^T H_2 + \overrightarrow{w_3}$

We have that $\nabla W_3[m] = \frac{\partial \vec{z}[m]}{\partial W_3} \frac{\partial \sigma(\vec{z}[m])}{\partial \vec{z}[m]} \nabla \hat{y}[m]$

Firstly $\nabla \hat{y}[m] = -\frac{\partial L(\vec{y}[m], \hat{y}[m])}{\partial \hat{y}} = \begin{bmatrix} \frac{\vec{y}[m][1]}{\hat{y}[m][1]} \\ \cdots \\ \frac{\vec{y}[m][k]}{\hat{y}[m][k]} \end{bmatrix}$

Next $\frac{\partial \sigma(\vec{z}[m])}{\partial \vec{z}[m]}$ is a $K \times K$ matrix.

$$\frac{\partial \sigma(\vec{z}[m])}{\partial \vec{z}[m]} = \begin{bmatrix} \frac{\partial \sigma(\vec{z}[m][1])}{\partial \vec{z}[m][1]} & \cdots & \frac{\partial \sigma(\vec{z}[m][k])}{\partial \vec{z}[m][1]} \\ \vdots & \ddots & \vdots \\ \frac{\partial \sigma(\vec{z}[m][1])}{\partial \vec{z}[m][k]} & \cdots & \frac{\partial \sigma(\vec{z}[m][k])}{\partial \vec{z}[m][k]} \end{bmatrix}$$

Such that $\frac{\partial \sigma(\vec{z}[m][k])}{\partial \vec{z}[m][i]} = \begin{Bmatrix} \sigma_M(\vec{z}[m][i])\left(1 - \sigma_M(\vec{z}[m][k])\right) \; if \; k = 1 \\ -\sigma_M(\vec{z}[m][k])\sigma_M(\vec{z}[m][i]) \; else \end{Bmatrix}$

Finally $\frac{\partial \vec{z}[m]}{\partial W_3}$ is a $N \times K \times K$ tensor.

$$\frac{\partial \vec{z}[m]}{\partial W_3} = \begin{bmatrix} \frac{\partial \vec{z}[m][1]}{\partial W_3} & \cdots & \frac{\partial \vec{z}[m][k]}{\partial W_3} \end{bmatrix}$$

$$\frac{\partial \vec{z}[m][k]}{\partial W_3[\;\;][i]} = \{ \begin{matrix} H_2 \; if \; k = 1 \\ 0 \; else \end{matrix} \}$$

Thus we have that $\nabla W_3 = \frac{\partial \vec{z}[m]}{\partial W_3} \frac{\partial \sigma(\vec{z}[m])}{\partial \vec{z}[m]} \nabla \hat{y}[m]$ with each term defined above.

Or in the case that conditions are matched $\nabla W_3 = (H_2)(\sigma(z[m])(1 - \sigma(z[m]))(\nabla \hat{y}[m])$

For the bias $\vec{w}_3$: $\nabla \vec{w}_3 = \frac{\partial \sigma(\vec{z}[m])}{\partial \vec{z}[m]} \nabla \hat{y}[m]$

Now we find the gradient ∇W_2

We call $\overrightarrow{z_2}[m] = W_2^T H_1 + \overrightarrow{w_2}$

$$\nabla H_2[m] = W_3 \left(\frac{\partial \sigma(\vec{z}[m])}{\partial \vec{z}[m]}\right) \nabla \hat{y}[m][i]$$

$$\nabla W_2[m] = \frac{\partial \overrightarrow{z_2}[m]}{\partial W_2} \frac{\partial ReLu(\overrightarrow{z_2}[m])}{\partial \overrightarrow{z_2}[m]} \nabla H_2[m]$$

Such that

$$\frac{\partial \overrightarrow{z_2}[m][i]}{\partial W_2[\ \ ][j]} = \{H_1 \ if \ i = j \ ; 0 \ else\}$$

$$\frac{\partial ReLu(\overrightarrow{z_2}[m][i])}{\overrightarrow{z_2}[m][j]} = \begin{Bmatrix} 1 \ if \ i = j \\ 0 \ else \ if \ i \neq j \\ c \ else \ if \ \overrightarrow{z_2}[m][j] = 0 \end{Bmatrix}$$

Thus if conditions are met $\nabla W_2 = \ H_1 * W_3 \left(\frac{\partial \sigma(\vec{z}[m])}{\partial \vec{z}[m]}\right) \nabla \hat{y}[m][i]$

And for the bias

$$\nabla \overrightarrow{w_2} = \frac{\partial ReLu(\overrightarrow{z_2}[m])}{\partial \overrightarrow{z_2}[m]} \nabla H_2[m]$$

Where the terms are defined above.

For the last weight matrix W_1 we have

$$\nabla H_1[m] = W_2 \frac{\partial ReLu(\overrightarrow{z_2}[m])}{\partial \overrightarrow{z_2}[m]} \nabla H_2[m]$$

$$\nabla W_1 = \frac{\partial \overrightarrow{z_3}[m]}{\partial W_1} * \frac{\partial ReLu(\overrightarrow{z_3}[m])}{\partial \overrightarrow{z_3}[m]} \nabla H_1[m]$$

Where $\overrightarrow{z_3} = W_1^T X + W_1$

The term $\frac{\partial ReLu(\overrightarrow{z_3}[m])}{\partial \overrightarrow{z_3}[m]}$ can be found the same way as the previous term $\frac{\partial ReLu(\overrightarrow{z_2}[m])}{\partial \overrightarrow{z_2}[m]}$ instead using the $\overrightarrow{z_3}$ vector. The same for $\frac{\partial \overrightarrow{z_3}[m]}{\partial W_1}$ but using $X[m]$ instead of H_1

So if conditions are met we can have $\nabla W_1 = X[m] * W_2 \ \frac{\partial ReLu(\overrightarrow{z_2}[m])}{\partial \overrightarrow{z_2}[m]} \ \nabla H_2[m]$

And for the bias we have $\nabla \overrightarrow{w_1} = \ \frac{\partial ReLu(\overrightarrow{z_3}[m])}{\partial \overrightarrow{z_3}[m]} \ \nabla H_1[m]$

Where the terms are defined above.

Gradients are updated along the following rule at iteration t

$$W_i^t = W_i^{t-1} - \eta \nabla \mathrm{W_i^{t-1}}$$

And the same holds for the bias vectors $\overrightarrow{w_i}$.

We do not use all the data in D. Instead we randomly choose 50 entries from the dataset to both forward compute and back propagate for each iteration.

3: Hyper Parameters

Hyper Parameters: For our experiment we have a few hyper parameters that directly contribute to the learning of the parameters Θ.

Learning rate η is a hyper parameter that multiplies the gradient for the theta updates. This model chose $\eta = 0.0001$. With too large of a learning rate the loss function will not converge. Therefore we choose small enough η such that the function begins to converge.

Iterations: another hyper parameter that contributes to how many times we update Θ. To generate the date, 100 iterations was chosen to be a suitable amount to achieve an accurate Θ.

Dimensions of dataset D: a hyper parameter that determined our gradient descent method. This model used full batch gradient descent, using subsets of size 50 in D. As $D = \{\vec{x}[m], \vec{y}[m]\}, m = 1,2,\ldots,M$ and input data $X^{50\times784}$ with output labels $\vec{y}^{10\times1}$, the weight matrix dimensions differed due to the structure of the neural network.

- $W_1^{784\times10}, W_2^{100\times10}, W_3^{100\times10}$
- $\overrightarrow{w_1}^{100\times1}, \overrightarrow{w_2}^{100\times1}, \overrightarrow{w_3}^{10\times1}$

While the learned parameters were saved in the format $\Theta = [W_1, \vec{w}_1, W_2, \overrightarrow{w_2}, W_3, \overrightarrow{w_3}]$. The biases and weights were split up for computation purposes.

This led to computationally expensive gradients, but also allowed conversion to happen earlier.

ReLu Constant "c": This was a catch case for if the derivative was trying to be calculated when the ReLu function was 0. To avoid this error we added a Laplace smoothing term given by $c = \frac{1}{M}$ where M is the size of the dataset.

Other Model Parameters and Information: Θ was initialized to small values $\in \{0,1\}$ in the requisite dimension. The learning rate and number of iterations was determined via trial and error. Furthermore while the Θ matrix was updated at all iterations, the errors, accuracy, and overall loss was only taken at every 10^{th} interval to improve runtime execution.

4: Results

5: Discussion

In theory the neural network would run slower than any other model. To complete one epoch it must computer 6 separate gradients. While we did use mini batches of size 50, due to the complexity of the gradients it still took a very long time to run. The model, does not suffer in performance. It is able to reach conversion very rapidly in terms of epochs. Between iterations we expect to see a bit of noise when it comes to accuracy and total loss. The loss may not

objectively decrease between given iterations. This is due to the non-linear function of the neural network. Activation functions in particular are not set in stone in what they are capable of. They too have limitations and approximations. But with the ReLu function we see that the function, over time, can minimize the loss function well. To solve the problem of overfitting we added our biases but we could also implement the random dropping of neurons for a more advanced neural network.