Understanding the Mathematics Behind the MLP Implementation

This document explains the mathematical details behind a Multi-Layer Perceptron (MLP) implementation specific to my code at: https://github.com/CooleMuetze/mlp. We will go through the following components step by step:

1. Initialization
2. Forward Pass
3. Loss Function (MSE)
4. Backpropagation

1. Initialization

The MLP class is initialized with the following parameters:

- `layers` : A list of integers specifying the number of neurons in each layer.
- `activation` : The activation function applied to each layer. In this implementation, a single activation function is used for all layers.
- `biasInitialization` : A small value (default is `0.05`) used to initialize the biases.
- `weightInitializationMultiplier` : A multiplier (default is `0.1`) used to scale the randomly initialized weights.

Weight and Bias Initialization

For each layer l, we initialize:

- **Weights** W^l as a matrix with dimensions $(\text{neurons in layer } l, \text{neurons in layer } l-1)$. The entries are sampled with random numbers and scaled by `weightInitializationMultiplier` .
- **Biases** b^l as a column vector with dimensions $(\text{neurons in layer } l, 1)$. All entries are initialized to the small value `biasInitialization` .

The goal of initializing the weights randomly is to break symmetry, ensuring that each neuron learns a unique function. Initializing the biases to small positive values helps prevent ReLU activation from outputting zero during the early stages of training.

2. Forward Pass

The forward pass computes the output of the MLP by passing the input through each layer sequentially. The calculation at each layer involves:

Mathematical Formulation

For each layer l:

1. Compute the pre-activation value (linear transformation):

$$z^l = W^l \cdot h^{l-1} + b^l$$

   - z^l is the weighted sum plus bias for layer l.
   - W^l is the weight matrix.
   - h^{l-1} is the output from the previous layer (or the input for the first layer).
   - b^l is the bias vector for layer l.
2. Apply the activation function $\sigma(z^l)$:

$$h^l = \sigma(z^l)$$

   - h^l is the output of layer l, which becomes the input for the next layer.

Summary of Forward Pass

The forward pass in a Multi-Layer Perceptron (MLP) refers to the process of passing input data through the network to obtain the output. This involves computing the weighted sum of inputs and biases at each layer, followed by applying an activation function to produce the output for that layer. The output of one layer becomes the input for the next, and this process continues until the final output layer is reached. The forward pass is essential for making predictions.

3. Loss Function (MSE)

The Mean Squared Error (MSE) is used as the cost function. It measures the squared differences between predicted and true values (there are other cost functions)

Mathematical Definition

$$\text{MSE} = \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_{\text{pred},i} - y_{\text{true},i})^2$$

- y_{pred} is the predicted output.
- y_{true} is the true output (target).

The derivative of MSE with respect to the predicted output y_{pred} is:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{pred}}} = \frac{2}{N}(y_{\text{pred}} - y_{\text{true}})$$

This derivative is essential for computing the error term during backpropagation.

4. Backpropagation

Backpropagation is the process of computing the gradients of the weights and biases to minimize the loss function using the chain rule. The goal is to adjust the weights and biases in such a way that the loss function is minimized, leading to more accurate predictions. To achieve this, we need to understand how changes in the weights and biases affect the loss function. This involves calculating the error at each layer and propagating it backward through the network.

Step 1: Computing the Error at the Output Layer

Step-by-Step Derivation

The error term for the last layer, denoted as δ^L, is a crucial component in the backpropagation process. It represents the gradient of the loss function with respect to the pre-activation values of the last layer. In simpler terms, it measures how much the loss function would change if the pre-activation values of the last layer were adjusted.

Why is δ^L Important?

1. **Direct Influence on Loss**: Since δ^L is derived from the loss function, it directly indicates how the output of the network (predicted values) affects the overall error. A higher δ^L means that a small change in the pre-activation value z^L will significantly impact the loss.

2. **Guiding Weight Updates**: The value of δ^L is used to compute the gradients for the weights and biases in the last layer. These gradients are essential for updating the weights and biases during the training process.
3. **Propagation to Previous Layers**: The error term δ^L serves as the starting point for backpropagating the error through the network. The error terms for previous layers ($\delta^{L-1}, \delta^{L-2}, \ldots$) are computed based on δ^L. Therefore, accurately calculating δ^L is critical for the entire backpropagation process.

In summary, δ^L quantifies the sensitivity of the loss function to the pre-activation values of the last layer. It is a key factor in adjusting the network's parameters to minimize the loss and improve the model's predictions.

How to calculate δ^L

1. **Derivative of the Cost Function with Respect to the Output**:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{pred}}} = \frac{2}{N}(y_{\text{pred}} - y_{\text{true}})$$

   This derivative tells us how much the predicted output y_{pred} needs to change to reduce the error.
2. **Derivative of the Output with Respect to the Pre-activation Value**:
   To understand the derivative of the output with respect to the pre-activation value, let's first express y_{pred} as a function of z^L:

$$y_{\text{pred}} = \sigma(z^L)$$

   Here, σ is the activation function applied to the pre-activation value z^L.
   Now, we can compute the derivative of y_{pred} with respect to z^L:

$$\frac{\partial y_{\text{pred}}}{\partial z^L} = \frac{\partial \sigma(z^L)}{\partial z^L} = \sigma'(z^L)$$

   This tells us how sensitive the output y_{pred} is to changes in the pre-activation value z^L.
3. **Applying the Chain Rule**:
   Using the chain rule, we combine these derivatives to find the derivative of the cost function with respect to z^L:

$$\delta^L = \frac{\partial \mathcal{L}}{\partial z^L} = \frac{\partial \mathcal{L}}{\partial y_{\text{pred}}} \cdot \frac{\partial y_{\text{pred}}}{\partial z^L}$$

Substituting the derivatives from steps 1 and 2, we get:

$$\delta^L = \frac{2}{N}(y_{\text{pred}} - y_{\text{true}}) \cdot \sigma'(z^L)$$

Explanation

By multiplying these two gradients, we obtain δ^L, which represents the sensitivity of the loss with respect to the pre-activation value z^L. This value is crucial for backpropagation as it helps in updating the weights and biases to minimize the loss.

This detailed step-by-step derivation ensures that we correctly compute the error term for the last layer, which is essential for the subsequent steps in backpropagation.

Step 2: Propagating the Error Backward

To minimize the loss function, we need to adjust the weights and biases in the network. This adjustment is based on the error term, which tells us how much each weight and bias contributes to the overall error. By propagating this error backward through the network, we can determine the necessary adjustments for each layer.

Derivation of the Error Term for Layer $l - 1$

For layers $l = L - 1, L - 2, \ldots, 1$, the error term δ^{l-1} is computed as follows:

1. **Error Term for the Current Layer**:
   First, we need to understand the error term for the current layer l:

   $$\delta^l = \frac{\partial \mathcal{L}}{\partial z^l}$$

   This is the error term for the current layer l. It represents how much the loss changes with respect to the pre-activation value z^l.
2. **Gradient of the Pre-activation Value with Respect to the Previous Layer's Output**:
   Next, we need to determine how the pre-activation value z^l changes with respect to the output of the previous layer h^{l-1}:

   $$z^l = W^l \cdot h^{l-1} + b^l$$

   Taking the derivative of z^l with respect to h^{l-1}:

$$\frac{\partial z^l}{\partial h^{l-1}} = W^l$$

This tells us how the pre-activation value z^l changes with respect to the output of the previous layer h^{l-1}.

3. **Applying the Chain Rule**:
   Using the chain rule, we combine these derivatives to find the derivative of the cost function with respect to z^{l-1}:

$$\delta^{l-1} = \frac{\partial \mathcal{L}}{\partial z^{l-1}} = \frac{\partial \mathcal{L}}{\partial z^l} \cdot \frac{\partial z^l}{\partial h^{l-1}} \cdot \frac{\partial h^{l-1}}{\partial z^{l-1}}$$

   Substituting the known derivatives:

$$\delta^{l-1} = \delta^l \cdot W^l \cdot \sigma'(z^{l-1})$$

4. **Transposing the Weight Matrix**:
   To match the dimensions correctly, we take the transpose of the weight matrix W^l:

$$\delta^{l-1} = (W^l)^T \cdot \delta^l \cdot \sigma'(z^{l-1})$$

By understanding and computing these error terms, we can effectively propagate the error backward through the network, allowing us to update the weights and biases to minimize the loss function.

Explanation

By combining these gradients, we effectively propagate the error backward through the network, layer by layer. This backward propagation of error is crucial for updating the weights and biases to minimize the loss function.

This detailed step-by-step derivation ensures that we correctly compute the error term for each layer, which is essential for the subsequent steps in backpropagation.

Step 3: Updating Weights and Biases

In this step, we focus on updating the weights and biases of the MLP to minimize the loss function. After computing the error terms for each layer during backpropagation, we use these error terms to adjust the weights and biases. The goal is to iteratively refine these parameters so that the MLP makes more accurate predictions. This process involves calculating the gradients of the loss function with respect to the weights and biases and then updating them in the direction that reduces the loss.

By doing so, we ensure that the MLP learns from the training data and improves its performance over time.

Mathematical Derivation

1. **Gradient of the Loss Function with Respect to Weights**:
   To derive the gradient of the loss function $\mathcal{L}$ with respect to the weights W^l in layer l using the chain rule, we follow these steps:

$$\frac{\partial \mathcal{L}}{\partial W^l} = \frac{\partial \mathcal{L}}{\partial z^l} \cdot \frac{\partial z^l}{\partial W^l}$$

   - $\frac{\partial \mathcal{L}}{\partial z^l} = \delta^l$ is the error term for layer l, representing the gradient of the loss function with respect to the pre-activation values z^l.
   - $z^l = W^l \cdot h^{l-1} + b^l$, so $\frac{\partial z^l}{\partial W^l} = h^{l-1}$.

   Combining these, we get:

$$\frac{\partial \mathcal{L}}{\partial W^l} = \delta^l \cdot (h^{l-1})^T$$

   This gradient tells us how much the loss function would change if we made a small change to the weights W^l. By moving in the direction opposite to this gradient (i.e., subtracting it), we reduce the loss.

2. **Gradient of the Loss Function with Respect to Biases**:
   To derive the gradient of the loss function $\mathcal{L}$ with respect to the biases b^l in layer l using the chain rule, we follow these steps:

$$\frac{\partial \mathcal{L}}{\partial b^l} = \frac{\partial \mathcal{L}}{\partial z^l} \cdot \frac{\partial z^l}{\partial b^l}$$

   - $\frac{\partial \mathcal{L}}{\partial z^l} = \delta^l$ is the error term for layer l.
   - $z^l = W^l \cdot h^{l-1} + b^l$, so $\frac{\partial z^l}{\partial b^l} = 1$.

   Combining these, we get:

$$\frac{\partial \mathcal{L}}{\partial b^l} = \delta^l$$

   This gradient tells us how much the loss function would change if we made a small change to the biases b^l. Again, by moving in the direction opposite to this gradient, we reduce the loss.

Why Moving in the Negative Gradient Direction Minimizes the Loss Function

The gradient of the loss function with respect to the weights and biases indicates the direction of the steepest ascent, meaning it shows how to increase the loss function. To minimize the loss function, we need to move in the opposite direction, which is the direction of the steepest descent. This is why we subtract the gradient during the update step.

By following this approach, we ensure that each update step moves the weights and biases closer to the values that minimize the loss, leading to better model performance.

How to update weights and biases

Based on the previous calculations, we know the direction in which the weights and biases need to be updated. Therefore, we adjust them in this direction, scaled by the learning rate.

With all δ^l values computed, the weights and biases are updated as follows:

1. **Weight Update**:

$$W^l \leftarrow W^l - \eta \cdot \delta^l \cdot (h^{l-1})^T$$

   - η is the learning rate.
   - δ^l is the error for the current layer.
   - $(h^{l-1})^T$ is the transposed output from the previous layer.
2. **Bias Update**:

$$b^l \leftarrow b^l - \eta \cdot \delta^l$$

   - η is the learning rate.
   - δ^l is the error for the current layer.

The learning rate η controls the step size of the updates to prevent overshooting the minimum loss.

Explanation

- **Direction of Update**: The gradients $\frac{\partial \mathcal{L}}{\partial W^l}$ and $\frac{\partial \mathcal{L}}{\partial b^l}$ indicate the direction in which the weights and biases should be adjusted to decrease the loss. By subtracting these gradients, we move in the direction that minimizes the loss.
- **Learning Rate** η: The learning rate η controls the size of the update steps. A smaller η results in smaller steps, which can lead to more precise convergence but may take longer. A larger η results in larger steps, which can speed up convergence but risks overshooting the minimum loss.

By iteratively updating the weights and biases using these gradients, the MLP learns to minimize the loss function, leading to better performance on the given task.

Conclusion

This MLP implementation involves careful initialization of weights and biases, a forward pass to compute layer outputs, and a detailed backpropagation algorithm to update weights and biases based on the error gradient. Each mathematical step ensures that the MLP learns the optimal weights for minimizing the loss function.

> Author: Leonard Grün