Backpropagation and Gradient-Based Optimization in Multilayer Perceptrons (MLPs)

Parth Agrawal

December 2025

Backpropagation and Gradient-Based Optimisation in Multilayer Perceptrons (MLPs)

In this report, we will cover two of the most important aspects of Multi-Layered Perceptrons, Backpropagation, and Gradient-Based Optimisation.

the-objective-function-loss-in-neural-networks

1 The Objective Function – Loss in Neural Networks

A loss function is a way to measure the performance and accuracy of a Machine Learning Model in numerical terms. The role of the loss function is to

- Measure the performance
- Direction for Improvement
- Balancing Bias and Variance
- Influencing model Behaviour

The choice of loss function depends on the problem type, specifically whether it is classification or regression. When performing a classification problem, we generally use functions like cross-entropy loss, whereas functions such as mean squared error are also used. This is because regression problems have continuous loss, whereas classification problems have discrete loss.

The smoothness and convexity of the loss function affect ease of training and speed.

mean-squared-error-loss-regression

1.1 Mean Squared Error Loss (Regression)

An example of a loss function for regression problems is the Mean Squared Error function, also known as the L2 function. The mathematical representation is -

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MSE is recommended in problems where the presence of outliers must be penalised seriously.

binary-cross-entropy-loss

1.2 Binary Cross-Entropy Loss

This is the loss function typically chosen for binary classification tasks.

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

BCE encourages the model to refine its predictions, which are probabilities for the appropriate class during the training.

categorical-cross-entropy-loss

1.3 Categorical Cross-Entropy Loss

The Categorical Cross-Entropy loss function is used for multiclass classification problems. The softmax function is usually used in such problems.

the-engine-of-learning—backpropagation

2 The Engine of Learning - Backpropagation

When calculating the derivative of the loss function to change the weights and biases, we go from the output layer to the input layer. This is called Backpropagation.

At the core of Backpropagation is the idea to think of the loss function as a graph and using gradient descent to approach a local minimum of the function. This is achieved by calculating the derivative using the Backpropagation algorithm.

The computational cost of the backpropagation function is equivalent to two forward passes, since it requires one forward pass computation and a backward pass for derivatives. This makes it extremely powerful.

We use the **Chain Rule** of calculus to find the derivatives of the subsequent layers. According to the rule, the derivative can be written as a product of derivatives with respect to different variables, allowing one to stack the functions to find the derivative of each layer.

In Backpropagation, we first compute the error term (which is quantitatively the difference between the present prediction and the correct prediction) to find the derivative of the cost function with respect to it.

example—derivation-of-the-gradient-of-the-weight-matrix-w

2.1 Example - Derivation of the gradient of the weight matrix WExample - Derivation of the gradient of the weight matrix W

Let the input be x of dimensions (n x 1), weights W of dimensions (m x n),

and output pre-activation be $z = Wx + b$ of dimensions (m x 1). Let activation be $\mathbf{y} = \sigma(\mathbf{z})$ of dimension (m x 1), and loss be L.

The general chain rule application in this case is -

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

First, we calculate the error term δ (delta),

$$\delta = \frac{\partial L}{\partial \mathbf{z}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{z}}\right)^T \frac{\partial L}{\partial \mathbf{y}}$$

Let's look at the Jacobian $\frac{\partial \mathbf{y}}{\partial \mathbf{z}}$:

Since $\mathbf{y} = \sigma(\mathbf{z})$ acts element-wise, the output y_i depends only on z_i. Therefore, the Jacobian $\mathbf{J}_\sigma = \frac{\partial \mathbf{y}}{\partial \mathbf{z}}$ is a diagonal matrix:

$$\mathbf{J}_\sigma = (\,\sigma\,)'\,(z_1) 0 \ldots 0 \sigma'(z_2) \ldots \vdots \ddots .$$

When we multiply this Jacobian by the gradient vector $\frac{\partial L}{\partial \mathbf{y}}$, it is mathematically equivalent to an element-wise multiplication (Hadamard product, denoted by $\odot$):

$$\delta = \frac{\partial L}{\partial \mathbf{y}} \odot \sigma'(\mathbf{z})$$

Now we move from $\mathbf{z}$ to $\mathbf{W}$. We know that $\mathbf{z} = \mathbf{W}\mathbf{x}$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

The k-th element of $\mathbf{z}$ is defined as:

$$z_k = \sum_{j=1}^{n} W_{kj} x_j$$

$$\frac{\partial z_k}{\partial W_{ij}} = \{\, x_{\,j}\; if k = i 0 if k \neq i$$

Therefore,

$$\frac{\partial L}{\partial W_{ij}} = \sum_k \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial W_{ij}}$$

Finally,

$$\frac{\partial L}{\partial W_{ij}} = \delta_i x_j$$

The derivation satisfies dimensional consistency: $(m \times 1) \times (1 \times n) = (m \times n)$.

gradient-descent-and-its-variants

3 Gradient Descent and its Variants

After calculating the derivative from Backpropagation, changing the weights and biases is known as Gradient Descent. It is moving towards the local minimum.

Batch Gradient Descent is when we compute loss and gradient over the whole dataset. This will give the most accurate gradient, but is hugely wasteful of computational resources in a dataset of millions of examples.

Stochastic Gradient Descent is a method that computes the gradient only over a single example. This is rarely used since, although it is swift, it is highly noisy and inefficient.

Mini-Batch Gradient Descent is the most widely used technique, which involves computing the gradient over a small batch of data (such as 256 samples) rather than the entire set. This provides an average of the entire dataset, striking a correct balance between accuracy and computational speed.

Momentum is used to accelerate the gradient descent process by incorporating an exponentially weighted moving average of past gradients. This helps smooth out the trajectory of the optimisation, allowing the algorithm to converge more quickly by reducing oscillations.

Standard Weight Update Rule:

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

SGD with Momentum update rule:

$$v_{t+1} = \gamma v_t + \eta \nabla L(w_t)$$

$$w_{t+1} = w_t - v_{t+1}$$

advanced-optimisers—adam-and-rmsprop

4 Advanced Optimisers - Adam and RMSProp

Advanced Optimizers are used to adjust learning rates for different parameters.

When we have the same learning rate for all the parameters in our model, this causes the model to only approach the gradient with a fixed change. This is not optimal since some parameters may have different sensitivities and can cause the gradient descent process to oscillate back and forth. Thus, having different learning rates for different parameters actually helps greatly.

RMSprop is an adaptive learning rate method that utilises an exponentially weighted moving average of squared gradients, which helps mitigate the issue of diminishing learning rates.

Adam (Adaptive Moment Estimation) optimizer combines the advantages of Momentum and RMSprop techniques to adjust learning rates during

training. It works well with large datasets and complex models because it utilises memory efficiently and automatically adjusts the learning rate for each parameter.

RMSprop Update:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) g_t^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Adam Update (Bias-corrected moments):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

activation-functions-and-derivatives

5 Activation Functions And Derivatives

We need to introduce non-linearity in our multilayer perceptrons, which is achieved through the use of activation functions.

Nonlinear activation functions help introduce additional complexity into neural networks and enable them to "learn" to approximate a much higher range of functions.

The **sigmoid activation function** or **logistic activation function** is a very popular nonlinear activation function that maps input data to the output range of (0, 1). The **tanh activation** function is somewhat similar to the sigmoid in the sense that it also maps the input values to an s-shaped curve, but in this case, the output range is (-1, 1) and is centred at 0, which solves one of the problems with the sigmoid function.

The Rectified Linear Unit (ReLU) function is the most widely used activation function today. Its beauty lies in its simplicity. It simply maps all negative values to zero, leaving positive values unchanged. This conserves the gradients while also being significantly faster than other functions, such as the sigmoid function.

The **Leaky ReLU** is a special case of the ReLU function, where negative values are multiplied by a small value, typically 0.01. This allows it to retain some information from negative gradients. The disadvantage is that updating weights can be very small.

Sigmoid derivative: $f'(x) = f(x)(1 - f(x))$

ReLU function: $f(x) = \max(0, x)$.

ReLU derivative: 1 if $x > 0$, else 0.

optimisation-challenges

6 Optimisation Challenges

Vanishing Gradients occur when gradients become extremely small as they propagate through various layers. When using the sigmoid activation functions, the derivative is always less than 0.25, and due to repeated multiplications, the gradients diminish in size.

Exploding Gradients occur when gradients grow exponentially during Backpropagation. When the product of gradients exceeds 1, it causes them to grow exponentially.

To solve this, we can employ methods like gradient clipping, utilise alternative activation functions such as ReLU, or employ weight initialisation techniques like Xavier initialisation and He initialisation. We can also do Batch Normalization.

hyperparameters—learning-rate

7 Hyperparameters - Learning Rate

The learning rate (**LR**) is the most critical hyperparameter in training.

If the learning rate is too high, the model is unable to settle at a local minimum (overshooting). When it is too low, the training process is extremely slow and inefficient (slow convergence).

LR Schedulers are algorithms that automatically adjust the learning rate of models during the training process. For example, StepLR, which reduces the learning rate by a fixed factor at regular intervals, and ExponentialLR, which reduces the learning rate exponentially by multiplying it by a factor like 0.01 at each step.

Cylindrical LR oscillates between minimum and maximum learning rates during the training process multiple times.

Step Decay Formula:

$$LR = LR_0 \cdot drop\_factor^{\lfloor \frac{epoch}{epochs\_drop} \rfloor}$$

sources

8 Sources

- DataCamp. (2024, December 4). Loss Functions in Machine Learning Explained. Retrieved from https://www.datacamp.com/tutorial/loss-function-in-machine-learning
- Nielsen, M. (n.d.). How the Backpropagation Algorithm Works (Chapter 2). In Neural Networks and Deep Learning. Retrieved from http://neuralnetworksanddeeplearning.com
- Stanford University. (n.d.). Optimization: Stochastic Gradient Descent (CS231n: Deep Learning for Computer Vision). Retrieved from https://cs231n.github.io/optimization-1/
- GeeksforGeeks. (2025, October 4). What is Adam Optimiser? Retrieved from https://www.geeksforgeeks.org/deep-learning/adam-optimizer/

- SuperAnnotate. (2023, October 12). Activation functions in neural networks (Updated 2024). Retrieved from https://www.superannotate.com/blog/activation-functions-in-neural-networks/
- Sharma, K. (2023, July 8). Vanishing/Exploding gradients problem. Medium. Retrieved from https://medium.com/@kushansharma1/vanishing-exploding-gradients-problem-1901bb2db2b2
- Chugani, V. (2025, May 16). A gentle introduction to learning rate schedulers. MachineLearningMastery. Retrieved from https://machinelearningmastery.com/a-gentle-introduction-to-learning-rate-schedulers/