Backpropagation and Gradient-Based Optimization in Multilayer Perceptrons

Vaibhav Chourasia

1 Introduction

A multilayer perceptron (MLP) is a function built by composing many simple units called neurons. Each neuron takes some numbers as input, forms a weighted sum, passes it through a non-linear function, and outputs a number. By arranging many neurons into layers, and stacking several layers, an MLP can approximate very complex relationships between inputs and outputs.

Training an MLP means adjusting its parameters (weights and biases) so that its outputs are close to target values on a dataset. This requires three main ideas:

- a loss function to quantify how wrong the network is,
- backpropagation to compute how each parameter affects the loss,
- gradient-based optimization methods (gradient descent, Adam, etc.) to update parameters.

2 From a single neuron to a layer

2.1 Single neuron in scalar form

Consider a neuron with d inputs $x_1, x_2, \ldots, x_d$. The neuron has parameters: weights $w_1, \ldots, w_d$ and bias b. It computes a weighted sum

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

Then it applies a non-linear activation function σ to produce its output

$$a = \sigma(z)$$

This is the basic scalar definition.

2.2 Writing the neuron in vector form

We now collect the inputs and weights into vectors. Define

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

The scalar weighted sum can be written as a dot product:

$$z = w_1 x_1 + \cdots + w_d x_d = w^T x$$

Then

$$z = w^T x + b, \quad a = \sigma(z)$$

We have now derived the vector form of a single neuron from the scalar expression.

2.3 Deriving a full layer from many neurons

Suppose there are n neurons in the same layer. Each neuron has its own weights and bias:

- neuron 1 has weights $w_1^{(1)}, \ldots, w_d^{(1)}$, bias b_1,
- neuron 2 has weights $w_1^{(2)}, \ldots, w_d^{(2)}$, bias b_2,
- and so on up to neuron n.

The pre-activation of neuron j is

$$z_j = \sum_{k=1}^{d} w_k^{(j)} x_k + b_j$$

and its activation is

$$a_j = \sigma(z_j)$$

We now combine all neurons into matrix form. Let each neuron's weight vector be

$$w_j = \begin{bmatrix} w_1^{(j)} \\ w_2^{(j)} \\ \vdots \\ w_d^{(j)} \end{bmatrix}$$

We stack these as rows into a matrix

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_n^T \end{bmatrix}$$

so W has shape $n \times d$. We also define the bias vector

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Now compute $Wx + b$:

$$Wx = \begin{bmatrix} w_1^T x \\ w_2^T x \\ \vdots \\ w_n^T x \end{bmatrix} = \begin{bmatrix} \sum_k w_k^{(1)} x_k \\ \sum_k w_k^{(2)} x_k \\ \vdots \\ \sum_k w_k^{(n)} x_k \end{bmatrix}$$

Adding b:

$$Wx + b = \begin{bmatrix} \sum_k w_k^{(1)} x_k + b_1 \\ \sum_k w_k^{(2)} x_k + b_2 \\ \vdots \\ \sum_k w_k^{(n)} x_k + b_n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

Call this vector z. The activation for the whole layer is defined by applying σ elementwise:

$$a = \sigma(z)$$

So we have derived the layer equations:

$$z = Wx + b, \quad a = \sigma(z)$$

3 Multilayer perceptron forward pass

3.1 Stacking layers

To build a deep network, we feed the output of one layer into the next.

Let the input to the network be $a^{(0)} = x$. For layer ℓ we have parameters $W^{(\ell)}$, $b^{(\ell)}$, pre-activation $z^{(\ell)}$, and activation $a^{(\ell)}$. From the single-layer derivation, we get

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$
$$a^{(\ell)} = \sigma^{(\ell)}(z^{(\ell)})$$

This simply repeats the layer logic multiple times. The final output is

$$\hat{y} = a^{(L)}$$

for a network with L layers.

4 Loss functions and their derivatives

Training aims to make the network output close to the target y. For a single training example, we define a scalar loss $\ell(\hat{y}, y)$ that measures the prediction error.

4.1 Mean squared error

For regression with scalar output, a common loss is

$$\ell = \frac{1}{2}(\hat{y} - y)^2$$

We differentiate with respect to $\hat{y}$. Using the rule $d(u^2)/du = 2u$:

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{1}{2} \cdot 2(\hat{y} - y) = \hat{y} - y$$

This simple expression appears in backprop as the starting gradient at the output.

4.2 Binary cross entropy with sigmoid

For binary classification with target $y \in \{0, 1\}$ and prediction $\hat{y} \in (0, 1)$, the binary cross entropy is

$$\ell = -\left[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right]$$

We differentiate with respect to $\hat{y}$:

$$\frac{\partial \ell}{\partial \hat{y}} = -\left[y \cdot \frac{1}{\hat{y}} + (1 - y) \cdot \frac{-1}{1 - \hat{y}}\right] = -\left[\frac{y}{\hat{y}} - \frac{1 - y}{1 - \hat{y}}\right] = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

Often we do not work with $\hat{y}$ directly, but with a logit z and $\hat{y} = \sigma(z)$ where

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The derivative of σ is

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) = \hat{y}(1 - \hat{y})$$

Now use the chain rule:

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \cdot \hat{y}(1 - \hat{y}) = \hat{y} - y$$

So for sigmoid output with binary cross entropy, the derivative with respect to z is simply $\hat{y} - y$, just like MSE with linear output.

5 Gradient descent from Taylor expansion

Let $\mathcal{L}(\theta)$ be the loss as a function of parameters θ. Consider a small change $\Delta\theta$. Using a first-order Taylor expansion:

$$\mathcal{L}(\theta + \Delta\theta) \approx \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T \Delta\theta$$

We want to choose $\Delta\theta$ so that $\mathcal{L}$ decreases. If we choose

$$\Delta\theta = -\eta \nabla\mathcal{L}(\theta)$$

for a small positive η, we get

$$\mathcal{L}(\theta + \Delta\theta) \approx \mathcal{L}(\theta) - \eta \|\nabla\mathcal{L}(\theta)\|^2$$

Since the second term is non-positive, the loss decreases for small enough η.
Therefore gradient descent uses the update

$$\theta_{t+1} = \theta_t - \eta \nabla\mathcal{L}(\theta_t)$$

In practice, we often approximate $\nabla\mathcal{L}$ using a mini-batch of training examples.

6 Backpropagation: detailed derivation for a two-layer MLP

We now derive backprop fully for a small network with one hidden layer. This shows clearly how derivatives connect through the network.

Consider:

$$z_j^{(1)} = \sum_k W_{jk}^{(1)} x_k + b_j^{(1)}, \quad a_j^{(1)} = \sigma(z_j^{(1)})$$

$$z^{(2)} = \sum_j W_j^{(2)} a_j^{(1)} + b^{(2)}, \quad \hat{y} = z^{(2)}$$

Loss:

$$\ell = \frac{1}{2}(\hat{y} - y)^2$$

6.1 Output layer gradients

We first compute the derivative of the loss with respect to the output $\hat{y}$:

$$\frac{\partial \ell}{\partial \hat{y}} = \hat{y} - y$$

Since $\hat{y} = z^{(2)}$, we also have

$$\frac{\partial \ell}{\partial z^{(2)}} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{(2)}} = (\hat{y} - y) \cdot 1 = \hat{y} - y$$

We define

$$\delta^{(2)} = \frac{\partial \ell}{\partial z^{(2)}} = \hat{y} - y$$

Now compute gradients with respect to $W_j^{(2)}$ and $b^{(2)}$.
From

$$z^{(2)} = \sum_j W_j^{(2)} a_j^{(1)} + b^{(2)}$$

we get

$$\frac{\partial z^{(2)}}{\partial W_j^{(2)}} = a_j^{(1)}, \quad \frac{\partial z^{(2)}}{\partial b^{(2)}} = 1$$

By chain rule:

$$\frac{\partial \ell}{\partial W_j^{(2)}} = \frac{\partial \ell}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W_j^{(2)}} = \delta^{(2)} a_j^{(1)}$$

$$\frac{\partial \ell}{\partial b^{(2)}} = \frac{\partial \ell}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(2)}} = \delta^{(2)} \cdot 1 = \delta^{(2)}$$

6.2 Hidden layer error signal

We now compute $\delta_j^{(1)} = \frac{\partial \ell}{\partial z_j^{(1)}}$.

Using the chain rule:

$$\frac{\partial \ell}{\partial z_j^{(1)}} = \frac{\partial \ell}{\partial a_j^{(1)}} \cdot \frac{\partial a_j^{(1)}}{\partial z_j^{(1)}}$$

First find $\frac{\partial \ell}{\partial a_j^{(1)}}$. We know

$$z^{(2)} = \sum_m W_m^{(2)} a_m^{(1)} + b^{(2)}$$

so

$$\frac{\partial z^{(2)}}{\partial a_j^{(1)}} = W_j^{(2)}$$

Chain rule:

$$\frac{\partial \ell}{\partial a_j^{(1)}} = \frac{\partial \ell}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial a_j^{(1)}} = \delta^{(2)} W_j^{(2)}$$

Now, $a_j^{(1)} = \sigma(z_j^{(1)})$, so

$$\frac{\partial a_j^{(1)}}{\partial z_j^{(1)}} = \sigma'(z_j^{(1)})$$

Therefore

$$\delta_j^{(1)} = \frac{\partial \ell}{\partial z_j^{(1)}} = \delta^{(2)} W_j^{(2)} \sigma'(z_j^{(1)})$$

6.3 Hidden layer parameter gradients

Finally, we need $\frac{\partial \ell}{\partial W_{jk}^{(1)}}$ and $\frac{\partial \ell}{\partial b_j^{(1)}}$.

From

$$z_j^{(1)} = \sum_k W_{jk}^{(1)} x_k + b_j^{(1)}$$

we get

$$\frac{\partial z_j^{(1)}}{\partial W_{jk}^{(1)}} = x_k, \quad \frac{\partial z_j^{(1)}}{\partial b_j^{(1)}} = 1$$

Chain rule:

$$\frac{\partial \ell}{\partial W_{jk}^{(1)}} = \frac{\partial \ell}{\partial z_j^{(1)}} \cdot \frac{\partial z_j^{(1)}}{\partial W_{jk}^{(1)}} = \delta_j^{(1)} x_k$$

$$\frac{\partial \ell}{\partial b_j^{(1)}} = \frac{\partial \ell}{\partial z_j^{(1)}} \cdot \frac{\partial z_j^{(1)}}{\partial b_j^{(1)}} = \delta_j^{(1)}$$

We have now derived all gradients for the two-layer MLP.

7 General backpropagation in vector form

We now generalize the previous derivation to multiple layers.

Define for each layer ℓ the vector

$$\delta^{(\ell)} = \frac{\partial \mathcal{L}}{\partial z^{(\ell)}}$$

At the output layer L, for a loss $\ell(a^{(L)}, y)$ and activation $a^{(L)} = \sigma^{(L)}(z^{(L)})$:

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial a^{(L)}} \odot \sigma^{(L)\prime}(z^{(L)})$$

where $\odot$ denotes elementwise multiplication.

For a hidden layer ℓ we use the same pattern as the two-layer derivation. First we get

$$\frac{\partial \mathcal{L}}{\partial a^{(\ell)}} = (W^{(\ell+1)})^T \delta^{(\ell+1)}$$

Then, since $a^{(\ell)} = \sigma^{(\ell)}(z^{(\ell)})$,

$$\delta^{(\ell)} = \frac{\partial \mathcal{L}}{\partial z^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial a^{(\ell)}} \odot \sigma^{(\ell)\prime}(z^{(\ell)}) = \left((W^{(\ell+1)})^T \delta^{(\ell+1)}\right) \odot \sigma^{(\ell)\prime}(z^{(\ell)})$$

Gradients with respect to parameters are:

$$\frac{\partial \mathcal{L}}{\partial W^{(\ell)}} = \delta^{(\ell)} (a^{(\ell-1)})^T$$

$$\frac{\partial \mathcal{L}}{\partial b^{(\ell)}} = \delta^{(\ell)}$$

8 Activation functions and their derivatives

8.1 Sigmoid

Definition:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

We derive its derivative. Consider:

$$\sigma(z) = (1 + e^{-z})^{-1}$$

Differentiate using the chain rule and power rule:

$$\sigma'(z) = -1 \cdot (1 + e^{-z})^{-2} \cdot \frac{d}{dz}(1 + e^{-z}) = -(1 + e^{-z})^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

Multiply numerator and denominator by e^z:

$$\sigma'(z) = \frac{1}{(e^z + 1)^2}$$

Notice that

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad 1 - \sigma(z) = \frac{e^{-z}}{1 + e^{-z}}$$

Then

$$\sigma(z)(1-\sigma(z)) = \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} = \frac{e^{-z}}{(1+e^{-z})^2}$$

which matches the earlier expression. So

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

8.2 Hyperbolic tangent

Definition:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

One can derive that

$$\tanh'(z) = 1 - \tanh(z)^2$$

This is often used in backprop when hidden layers use tanh.

8.3 ReLU

Definition:

$$\mathrm{ReLU}(z) = \max(0, z)$$

Derivative:

$$\mathrm{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

In implementation, this derivative determines which neurons pass gradients backward and which do not.

9 Exploding and vanishing gradients

Backpropagation computes error signals layer by layer:

$$\delta^{(\ell)} = \left(W^{(\ell+1)}\right)^T \delta^{(\ell+1)} \odot \sigma^{(\ell)\prime}(z^{(\ell)})$$

If we focus on the magnitude only and replace $\odot$ with a rough multiplication by some factor, we can approximate

$$\|\delta^{(\ell)}\| \approx \|W^{(\ell+1)}\| \cdot c_\ell \cdot \|\delta^{(\ell+1)}\|$$

for some factor c_ℓ related to $\sigma^{(\ell)\prime}$. Repeating this many times gives

$$\|\delta^{(1)}\| \approx \left(\prod_{\ell=1}^{L-1} \|W^{(\ell+1)}\| c_\ell\right) \|\delta^{(L)}\|$$

If most factors $\|W^{(\ell)}\|c_\ell$ are less than 1, the product tends to zero for large L, and gradients vanish for layers near the input. If they are larger than 1, the product can grow, causing exploding gradients.

This explains why activation choice and initialization (for example He or Xavier initialization) are important in deep networks.

10 Optimization algorithms

10.1 Stochastic gradient descent

In practice, we estimate gradients using minibatches. For parameters θ and gradient estimate g_t at step t:

$$\theta_{t+1} = \theta_t - \eta g_t$$

This is stochastic gradient descent (SGD).

10.2 Momentum

SGD can be noisy and slow. Momentum introduces a velocity variable v_t:

$$v_{t+1} = \mu v_t - \eta g_t$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

To see what v_t represents, expand a few steps:

$$v_1 = \mu v_0 - \eta g_0 = -\eta g_0$$
$$v_2 = \mu v_1 - \eta g_1 = -\eta(\mu g_0 + g_1)$$

Continuing, one can see that v_t is a weighted sum of past gradients:

$$v_t \approx -\eta(g_{t-1} + \mu g_{t-2} + \mu^2 g_{t-3} + \dots)$$

Thus momentum effectively averages gradients over time, smoothing noise and accelerating motion in directions where the gradient keeps pointing the same way.

10.3 RMSProp

RMSProp adapts the learning rate for each parameter based on the recent magnitude of its gradient. It maintains a running average of squared gradients:

$$s_{t+1} = \rho s_t + (1-\rho) g_t^2$$

where the square is elementwise. Then the update is

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{s_{t+1}} + \epsilon} g_t$$

If a parameter has large gradients, s_t becomes large and the denominator increases, reducing the effective step size for that parameter.

10.4 Adam

Adam combines the ideas of momentum and RMSProp. It keeps both a moving average of gradients and of squared gradients:

$$m_{t+1} = \beta_1 m_t + (1-\beta_1) g_t$$
$$v_{t+1} = \beta_2 v_t + (1-\beta_2) g_t^2$$

Here m_t behaves like momentum and v_t like RMSProp's s_t.

Because both m_0 and v_0 start at zero, their early values are biased towards zero. To correct this, Adam uses bias-corrected versions:

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}}$$
$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}}$$

The final update is

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon}$$

This gives direction from $\hat{m}_{t+1}$ and scale from $\hat{v}_{t+1}$.

11 Learning rate and convergence

Consider a simple one-dimensional quadratic loss

$$J(\theta) = \frac{1}{2} a \theta^2$$

with $a > 0$. The derivative is

$$\frac{dJ}{d\theta} = a\theta$$

Gradient descent with learning rate η gives

$$\theta_{t+1} = \theta_t - \eta a \theta_t = (1 - \eta a)\theta_t$$

We can solve this recurrence:

$$\theta_t = (1 - \eta a)^t \theta_0$$

For convergence to zero, we need $|1 - \eta a| < 1$. This inequality becomes

$$-1 < 1 - \eta a < 1$$

which simplifies to

$$0 < \eta a < 2 \quad \Rightarrow \quad 0 < \eta < \frac{2}{a}$$

So η must lie in a certain range: too small means very slow convergence, and too large leads to divergence. In higher dimensions, a similar condition involves the largest eigenvalue of the Hessian matrix.

12 Conclusion

We started from the definition of a single neuron and derived the vector and matrix forms for a layer. By stacking layers, we obtained the forward pass of an MLP. We defined loss functions such as mean squared error and binary cross entropy and derived their gradients.

Using Taylor expansion, we derived gradient descent and then, using scalar chain rule, we built backpropagation for a two-layer MLP step by step. Generalizing this gave the standard vector form of backpropagation for deep networks. We derived the derivatives of common activation functions and studied how repeated multiplication by weights and activation derivatives leads to exploding and vanishing gradients.

Finally, we derived and explained optimization methods such as SGD with momentum, RMSProp, and Adam, and analyzed the effect of the learning rate using a simple quadratic example.