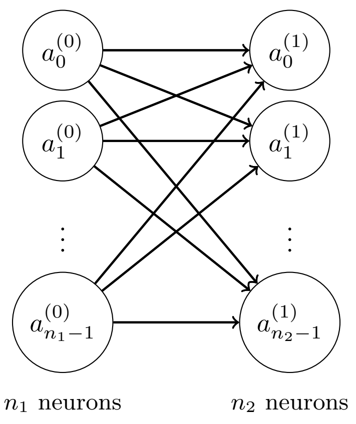
Neural Network – Layer by Layer Based System

Notation and Structure:

We model the neural network as a composition of layers where each layer receives the activation values from the previous layer, applies a linear transformation using weights and biases, and then passes the result through an activation function.

Let:

- The 0th layer (input layer) have n_0 neurons
- The 1st layer (next layer) have n_1 neurons



The activation of the 0th neuron in the 1st layer is:

$$a_0^{(1)} = \sigma\left(w_{0,0}^{(1)}a_0^{(0)} + w_{1,0}^{(1)}a_1^{(0)} + \cdots + w_{n_1,0}^{(1)}a_{n_1}^{(0)} + b_0^{(1)}\right)$$

This can be written in vector-matrix form as:

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{n_2}^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{0,0}^{(1)} & w_{1,0}^{(1)} & \cdots & w_{n_1,0}^{(1)} \\ w_{0,1}^{(1)} & w_{1,1}^{(1)} & \cdots & w_{n_1,1}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0,n_2}^{(1)} & w_{1,n_2}^{(1)} & \cdots & w_{n_1,n_2}^{(1)} \end{bmatrix}\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_{n_1}^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_{n_2}^{(1)} \end{bmatrix}\right)$$

Dimensions:

- $\vec{a}^{(0)}$ is $(n_1+1)\times 1$
- $W^{(1)}$ is $(n_2+1)\times(n_1+1)$
- $\vec{b}^{(1)}$ is $(n_2+1)\times 1$
- Resulting $\vec{a}^{(1)}$ is $(n_2+1)\times 1$

Generalizing:

If we are going from the k^{th} layer to the $(k+1)^{\text{th}}$ layer:

- n_k+1 neurons in layer k
- $n_{k+1}+1$ neurons in layer $k+1$

Then the activation of the j^{th} neuron in the $(k+1)^{\text{th}}$ layer is:

$$a_j^{(k+1)} = \sigma\left(w_{j,0}^{(k+1)} a_0^{(k)} + w_{j,1}^{(k+1)} a_1^{(k)} + \cdots + w_{j,n_k}^{(k+1)} a_{n_k}^{(k)} + b_j^{(k+1)}\right)$$

This can be written in summation form:

$$a_j^{(k+1)} = \sigma\left(\sum_{i=0}^{n_k} w_{j,i}^{(k+1)} \cdot a_i^{(k)} + b_j^{(k+1)}\right)$$

And in vectorized matrix notation:

$$\vec{a}^{(k+1)} = \sigma\left(W^{(k+1)}\vec{a}^{(k)} + \vec{b}^{(k+1)}\right)$$

Dimensions:

- $\vec{a}^{(k)}$: $(n_k + 1) \times 1$
- $W^{(k+1)}$: $(n_{k+1} + 1) \times (n_k + 1)$
- $\vec{b}^{(k+1)}$: $(n_{k+1} + 1) \times 1$
- Result: $\vec{a}^{(k+1)}$ is $(n_{k+1} + 1) \times 1$

Backpropagation: Single Neuron per Layer (Index-Based Derivation)

We consider a neural network where each layer contains a single neuron. Our goal is to derive the gradients required for backpropagation, using index notation and recursive application of the chain rule.

Let the activation in the L^{th} layer be defined as:

$$a^{(L)} = \sigma(z^{(L)}), \quad \text{where } z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

This defines a feedforward network of single neurons across layers.

The cost function is defined as:

$$C_0 = (a^{(L)} - y)^2 \quad \text{where } y \text{ is the desired output}$$

Gradient of Cost w.r.t. Final Layer Parameters (Using Chain Rule)

Using the chain rule:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial C_0}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}}, \qquad \frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial C_0}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial b^{(L)}}$$

Averaging Gradients Across Training Examples

If we have n training examples, then:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n}\sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}, \quad \text{where } C_k \text{ is the loss for the } k^{\text{th}} \text{ training example}$$

Recursive Derivation for Earlier Layers (Using Chain Rule)

To compute gradient w.r.t. the activation in the previous layer:

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial C_0}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}}$$

Recursive Generalization (Using Chain Rule)

To compute gradient with respect to any earlier weight:

$$\frac{\partial C_0}{\partial w^{(k)}} = \frac{\partial C_0}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(k+1)}}{\partial a^{(k)}} \cdot \frac{\partial a^{(k)}}{\partial z^{(k)}} \cdot \frac{\partial z^{(k)}}{\partial w^{(k)}}$$

And for any earlier bias:

$$\frac{\partial C_0}{\partial b^{(k)}} = \frac{\partial C_0}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(k+1)}}{\partial a^{(k)}} \cdot \frac{\partial a^{(k)}}{\partial z^{(k)}} \cdot \frac{\partial z^{(k)}}{\partial b^{(k)}}$$

This recursive formulation allows us to compute gradients for any parameter in the network using the chain rule applied layer-by-layer from output to input.

Backpropagation Generalized: Multiple Neuron in Each Layer

In this section, we extend the backpropagation formulation to the case where each layer has multiple neurons.

Let the k^{th} layer have n_{k-1} neurons, and let y_j denote the desired output at the j^{th} neuron in the final layer L. Then, the cost function is given by:

$$C_0 = \frac{1}{n_L} \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$

The forward propagation relation for a single neuron j in layer L is:

$$a_j^{(L)} = \sigma\left(\sum_{i=0}^{n_{L-1}-1} w_{ji}^{(L)} a_i^{(L-1)} + b_j^{(L)} \right) = \sigma(z_j^{(L)})$$

Using the chain rule, the gradient of the cost with respect to a weight $w_{ji}^{(L)}$ and bias $b_j^{(L)}$ is:

$$\frac{\partial C_0}{\partial w_{ji}^{(L)}} = \frac{\partial C_0}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial w_{ji}^{(L)}}$$

$$\frac{\partial C_0}{\partial b_j^{(L)}} = \frac{\partial C_0}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}$$

This pattern continues across layers using recursive application of the chain rule. For instance, the activation at earlier layers such as $L-1$, $L-2$, etc. is given by:

$$a_k^{(L-1)} = \sigma\left(\sum_{i=0}^{n_{L-2}-1} w_{ki}^{(L-1)} a_i^{(L-2)} + b_k^{(L-1)} \right) = \sigma(z_k^{(L-1)})$$

$$a_k^{(L-2)} = \sigma\left(\sum_{i=0}^{n_{L-3}-1} w_{ki}^{(L-2)} a_i^{(L-3)} + b_k^{(L-2)} \right) = \sigma(z_k^{(L-2)})$$

Again, at any arbitrary layer k:

$$a_j^{(k)} = \sigma(z_j^{(k)}) = \sigma\left(\sum_{i=0}^{n_{k-1}-1} w_{ji}^{(k)} a_i^{(k-1)} + b_j^{(k)}\right)$$

The generalized gradient with respect to an earlier weight $w_{ab}^{(L-1)}$ involves summing over the entire layer:

$$\frac{\partial C_0}{\partial w_{ab}^{(L-1)}} = \sum_{j=0}^{n_L-1} \sum_{i=0}^{n_{L-1}-1} \frac{\partial C_0}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} \cdot \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \cdot \frac{\partial z_i^{(L-1)}}{\partial w_{ab}^{(L-1)}}$$

Similarly, for the biases:

$$\frac{\partial C_0}{\partial b_i^{(L-1)}} = \sum_{j=0}^{n_L-1} \sum_{i=0}^{n_{L-1}-1} \frac{\partial C_0}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} \cdot \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \cdot \frac{\partial z_i^{(L-1)}}{\partial b_i^{(L-1)}}$$

These expressions show how backpropagation systematically applies the chain rule through all neurons and layers, eventually reaching the desired parameter to compute gradients for updating during training.

Recursive Gradient Generalization (Layer-wise)

We now write the recursive formulation to generalize the gradient of the cost function C_0 with respect to any weight $w_{ab}^{(k)}$ or bias $b_i^{(k)}$ using layer-wise nested summations and the chain rule.

$$\frac{\partial C_0}{\partial w_{ab}^{(k)}} = \sum_{k_L=0}^{n_L-1} \sum_{k_{L-1}=0}^{n_{L-1}-1} \cdots \sum_{k_{(k+1)}=0}^{n_{(k+1)}-1} \left[\frac{\partial C_0}{\partial a_{k_L}^{(L)}} \cdot \frac{\partial a_{k_L}^{(L)}}{\partial z_{k_L}^{(L)}} \cdot \frac{\partial z_{k_L}^{(L)}}{\partial a_{k_{L-1}}^{(L-1)}} \cdot \cdots \cdot \frac{\partial z_{k_{(k+1)}}^{(k+1)}}{\partial a_a^{(k)}} \cdot \frac{\partial a_a^{(k)}}{\partial z_a^{(k)}} \cdot \frac{\partial z_a^{(k)}}{\partial w_{ab}^{(k)}} \right]$$

Similarly, for the bias $b_i^{(k)}$:

$$\frac{\partial C_0}{\partial b_i^{(k)}} = \sum_{k_L=0}^{n_L-1} \sum_{k_{L-1}=0}^{n_{L-1}-1} \cdots \sum_{k_{(k+1)}=0}^{n_{(k+1)}-1} \left[\frac{\partial C_0}{\partial a_{k_L}^{(L)}} \cdot \frac{\partial a_{k_L}^{(L)}}{\partial z_{k_L}^{(L)}} \cdot \frac{\partial z_{k_L}^{(L)}}{\partial a_{k_{L-1}}^{(L-1)}} \cdot \cdots \cdot \frac{\partial z_{k_{(k+1)}}^{(k+1)}}{\partial a_i^{(k)}} \cdot \frac{\partial a_i^{(k)}}{\partial z_i^{(k)}} \cdot \frac{\partial z_i^{(k)}}{\partial b_i^{(k)}} \right]$$

Vectorizing Backpropagation Equations

We now shift from index-based expressions to their compact vectorized form using matrix operations. This enables more efficient computation and aligns naturally with frameworks like NumPy.

We begin with the chain rule expansion for weight and bias gradients in the ℓ^{th} layer:

$$\frac{\partial C_0}{\partial w_{ab}^{(\ell)}} = \frac{\partial C_0}{\partial z_a^{(\ell)}} \cdot \frac{\partial z_a^{(\ell)}}{\partial w_{ab}^{(\ell)}} \quad \text{and} \quad \frac{\partial C_0}{\partial b_a^{(\ell)}} = \frac{\partial C_0}{\partial z_a^{(\ell)}} \cdot \frac{\partial z_a^{(\ell)}}{\partial b_a^{(\ell)}}$$

From the linear equation in forward propagation:

$$z_a^{(\ell)} = \sum_{i=0}^{n_{\ell-1}-1} w_{ai}^{(\ell)} a_i^{(\ell-1)} + b_a^{(\ell)}$$

We have:

$$\frac{\partial z_a^{(\ell)}}{\partial w_{ab}^{(\ell)}} = a_b^{(\ell-1)} \quad \text{and} \quad \frac{\partial z_a^{(\ell)}}{\partial b_a^{(\ell)}} = 1$$

Substituting these into the gradients:

$$\frac{\partial C_0}{\partial w_{ab}^{(\ell)}} = \frac{\partial C_0}{\partial z_a^{(\ell)}} \cdot a_b^{(\ell-1)} \quad \text{and} \quad \frac{\partial C_0}{\partial b_a^{(\ell)}} = \frac{\partial C_0}{\partial z_a^{(\ell)}}$$

Now, define the error vector:

$$\delta_a^{(\ell)} = \frac{\partial C_0}{\partial z_a^{(\ell)}}$$

This gives us:

$$\frac{\partial C_0}{\partial w_{ab}^{(\ell)}} = \delta_a^{(\ell)} \cdot a_b^{(\ell-1)} \quad \text{and} \quad \frac{\partial C_0}{\partial b_a^{(\ell)}} = \delta_a^{(\ell)}$$

Vectorized Form

Let $\delta^{(\ell)}$ and $a^{[\ell-1]}$ be column vectors of shape $(n_\ell \times 1)$ and $(n_{\ell-1} \times 1)$ respectively:

$$\delta^{(\ell)} = \begin{bmatrix} \delta_0^{(\ell)} \\ \delta_1^{(\ell)} \\ \vdots \\ \delta_{n_\ell-1}^{(\ell)} \end{bmatrix}, \qquad \mathbf{a}^{[\ell-1]} = \begin{bmatrix} a_0^{(\ell-1)} \\ a_1^{(\ell-1)} \\ \vdots \\ a_{n_{\ell-1}-1}^{(\ell-1)} \end{bmatrix}$$

Then the gradients with respect to the entire weight matrix and bias vector can be expressed as:

$$\frac{\partial C_0}{\partial \mathbf{W}^{(\ell)}} = \delta^{(\ell)} (\mathbf{a}^{[\ell-1]})^T \quad \text{and} \quad \frac{\partial C_0}{\partial \mathbf{b}^{(\ell)}} = \delta^{(\ell)}$$

These vectorized results significantly simplify and speed up training computations. They also generalize well to batch processing, GPU acceleration, and matrix algebra frameworks.

Let us now examine the meaning of the matrix multiplication $\delta^{(\ell)}(a^{[\ell-1]})^T$ in detail.

This produces an $n_\ell \times n_{\ell-1}$ matrix as follows:

$$\delta^{(\ell)}(a^{[\ell-1]})^T = \begin{bmatrix} \delta_0^{(\ell)} \\ \delta_1^{(\ell)} \\ \vdots \\ \delta_{n_\ell-1}^{(\ell)} \end{bmatrix} \Big[(a_0^{(\ell-1)} \quad a_1^{(\ell-1)} \quad \cdots \quad a_{n_{\ell-1}-1}^{(\ell-1)}) \Big] = \begin{bmatrix} \delta_0^{(\ell)} a_0^{(\ell-1)} & \delta_0^{(\ell)} a_1^{(\ell-1)} & \cdots & \delta_0^{(\ell)} a_{n_{\ell-1}-1}^{(\ell-1)} \\ \delta_1^{(\ell)} a_0^{(\ell-1)} & \delta_1^{(\ell)} a_1^{(\ell-1)} & \cdots & \delta_1^{(\ell)} a_{n_{\ell-1}-1}^{(\ell-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n_\ell-1}^{(\ell)} a_0^{(\ell-1)} & \delta_{n_\ell-1}^{(\ell)} a_1^{(\ell-1)} & \cdots & \delta_{n_\ell-1}^{(\ell)} a_{n_{\ell-1}-1}^{(\ell-1)} \end{bmatrix}$$

This matrix is equal to:

$$\frac{\partial C_0}{\partial \mathbf{W}^{(\ell)}}$$

Recursive Formula for δ (Backward Pass)

To propagate δ backwards to earlier layers, we apply the chain rule again:

$$S_a^{[\ell]} = \frac{\partial C_0}{\partial z_a^{(\ell)}} = \sum_{k=0}^{n_{\ell+1}-1} \left(\frac{\partial C_0}{\partial z_k^{(\ell+1)}} \cdot \frac{\partial z_k^{(\ell+1)}}{\partial a_a^{(\ell)}} \cdot \frac{\partial a_a^{(\ell)}}{\partial z_a^{(\ell)}} \right)$$

From the linear equation in forward propagation:

$$z_k^{(\ell+1)} = \sum_{i=0}^{n_\ell-1} w_{ki}^{(\ell+1)} a_i^{(\ell)} + b_k^{(\ell+1)} \Rightarrow \frac{\partial z_k^{(\ell+1)}}{\partial a_a^{(\ell)}} = w_{ka}^{(\ell+1)}$$

Also,

$$a_a^{(\ell)} = \sigma(z_a^{(\ell)}) \Rightarrow \frac{\partial a_a^{(\ell)}}{\partial z_a^{(\ell)}} = \sigma'(z_a^{(\ell)})$$

Putting this together:

$$\delta_a^{(\ell)} = \sum_{k=0}^{n_{\ell+1}-1} \delta_k^{(\ell+1)} w_{ka}^{(\ell+1)} \cdot \sigma'(z_a^{(\ell)})$$

Matrix Form of δ (Recursive)

Let's extend this result for all neurons in a layer using vectorized form. Consider:

$$\delta_1^{(\ell)} = \delta_1^{(\ell+1)} w_{1,1}^{(\ell+1)} + \delta_2^{(\ell+1)} w_{2,1}^{(\ell+1)} + \cdots + \delta_{n_{\ell+1}-1}^{(\ell+1)} w_{(n_{\ell+1}-1),1}^{(\ell+1)}$$

$$\delta_2^{(\ell)} = \delta_1^{(\ell+1)} w_{1,2}^{(\ell+1)} + \delta_2^{(\ell+1)} w_{2,2}^{(\ell+1)} + \cdots + \delta_{n_{\ell+1}-1}^{(\ell+1)} w_{(n_{\ell+1}-1),2}^{(\ell+1)}$$

Hence,

$$\delta^{(\ell)} = \left(W^{[\ell+1]}\right)^T \delta^{(\ell+1)} \odot \sigma'(z^{(\ell)})$$

Final 3 Vectorized Results

We arrive at the following vectorized results:

$$\delta^{(\ell)} = \left(W^{[\ell+1]}\right)^T \delta^{(\ell+1)} \odot \sigma'(z^{(\ell)})$$

$$\frac{\partial C_0}{\partial b^{(\ell)}} = \delta^{(\ell)}$$

$$\frac{\partial C_0}{\partial W^{(\ell)}} = \delta^{(\ell)} (a^{[\ell-1]})^T$$

Neural Network Configuration

We construct a fully connected neural network with 4 layers:

- Input layer: 784 neurons (flattened 28×28 image)
- First hidden layer: 32 neurons
- Second hidden layer: 32 neurons
- Output layer: 10 neurons (digit classification via softmax)

Backpropagation Derivations Used in Code

For a batch of n training examples, the derivations applied in the Python code are as follows:

Forward Propagation Let $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$ and $A^{[l]} = \sigma(Z^{[l]})$ for hidden layers, and $A^{[3]} = \text{softmax}(Z^{[3]})$ in the output layer.

Output Layer Error (Cross-Entropy Loss + Softmax) If Y is the one-hot true label matrix and $A^{[3]}$ is the prediction:

$$\delta^{[3]} = A^{[3]} - Y \quad (10 \times n)$$

Softmax + Cross Entropy Derivative (Full Expression)

Recall that when combining softmax activation with the cross-entropy loss, the gradient simplifies nicely. But in full generality, using the chain rule, we obtain:

$$\frac{\partial C}{\partial z_i} = \sum_{j=1}^{10} \frac{\partial C}{\partial A_j} \cdot \frac{\partial A_j}{\partial z_i}$$

From the softmax function:

$$A_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}} \Rightarrow \frac{\partial A_j}{\partial z_i} = \begin{cases} A_j(1 - A_j) & \text{if } i = j \\ -A_j A_i & \text{if } i \neq j \end{cases}$$

And from the cross-entropy loss:

$$\frac{\partial C}{\partial A_j} = -\frac{y_j}{A_j}$$

Substituting in, the full expression simplifies to:

$$\frac{\partial C}{\partial z_i} = A_i - y_i$$

Conclusion: This simplification is one of the key reasons why softmax is almost always paired with cross-entropy — the gradient becomes extremely efficient to compute.

Gradient w.r.t. Output Layer Parameters

$$\frac{\partial C}{\partial W^{[3]}} = \frac{1}{n}\delta^{[3]}(A^{[2]})^T \quad (10 \times 32)$$

$$\frac{\partial C}{\partial b^{[3]}} = \frac{1}{n}\sum \delta^{[3]} \quad (10 \times 1)$$

Hidden Layer Errors Using ReLU activation, where $\sigma'(Z) = \nVdash(Z > 0)$:

$$\delta^{[2]} = (W^{[3]})^T \delta^{[3]} \circ \sigma'(Z^{[2]}) \quad (32 \times n)$$

$$\delta^{[1]} = (W^{[2]})^T \delta^{[2]} \circ \sigma'(Z^{[1]}) \quad (32 \times n)$$

Gradients for Hidden Layers

$$\frac{\partial C}{\partial W^{[l]}} = \frac{1}{n}\delta^{[l]}(A^{[l-1]})^T \quad \text{for } l = 1, 2$$

$$\frac{\partial C}{\partial b^{[l]}} = \frac{1}{n}\sum \delta^{[l]} \quad \text{for } l = 1, 2$$

These derivatives allow efficient parameter updates using gradient descent:

$$W^{[l]} \leftarrow W^{[l]} - \eta\frac{\partial C}{\partial W^{[l]}}, \quad b^{[l]} \leftarrow b^{[l]} - \eta\frac{\partial C}{\partial b^{[l]}}$$