Neural networks from scratch

Artur Ayrapetyan

1 Single-neuron neural network

We begin with the special setting: a feedforward network that has a single neuron per layer. First we look at 3 layers' network and then generalize it to J layers.

Dimensions of variables

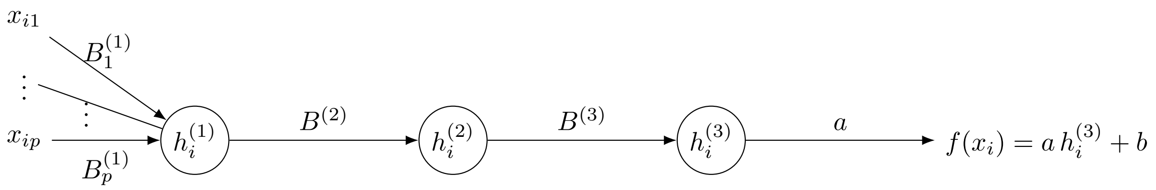
$$x_i^\top = \begin{bmatrix} x_{i1} & x_{i2} & \cdots & x_{ip} \end{bmatrix} \in \mathbb{R}^{1\times p},$$

$$B^{(1)} = \begin{bmatrix} B_1^{(1)} \\ B_2^{(1)} \\ \vdots \\ B_p^{(1)} \end{bmatrix} \in \mathbb{R}^{p\times 1},$$

$$B^{(2)}, B^{(3)}, c^{(1)}, c^{(2)}, c^{(3)}, a_0, b \in \mathbb{R}, \qquad z_i^{(l)},\, h_i^{(l)} \in \mathbb{R}, \quad l = 1, 2, 3.$$

Here, $u(\cdot)$ denotes the **activation function**, and $u'(\cdot)$ its derivative.
Let's look at 3 layers one neuron network:

Forward pass



$$z_i^{(1)} = \sum_{k=1}^{p} x_{ik}\, B_k^{(1)} + c^{(1)}, \quad \Rightarrow \quad h_i^{(1)} = u\Big(z_i^{(1)}\Big),$$

$$z_i^{(2)} = h_i^{(1)} B^{(2)} + c^{(2)}, \quad \Rightarrow \quad h_i^{(2)} = u\Big(z_i^{(2)}\Big),$$

$$z_i^{(3)} = h_i^{(2)} B^{(3)} + c^{(3)}, \quad \Rightarrow \quad h_i^{(3)} = u\Big(z_i^{(3)}\Big).$$

Output

$$f(x_i) = a\,h_i^{(3)} + b.$$

Gradients

$$\frac{\partial f(x_i)}{\partial h_i^{(3)}} = a.$$

$$\frac{\partial f(x_i)}{\partial b} = 1.$$

$$\frac{\partial f(x_i)}{\partial c^{(3)}} = a\, u'\Big(z_i^{(3)}\Big)\ .$$

$$\frac{\partial f(x_i)}{\partial B^{(3)}} = a\, u'\Big(z_i^{(3)}\Big)\ h_i^{(2)}.$$

$$\frac{\partial f(x_i)}{\partial B^{(2)}} = a\, u'\Big(z_i^{(3)}\Big)\, B^{(3)}\, u'\Big(z_i^{(2)}\Big)\ h_i^{(1)}.$$

$$\frac{\partial f(x_i)}{\partial c^{(2)}} = a\, u'\Big(z_i^{(3)}\Big)\, B^{(3)}\, u'\Big(z_i^{(2)}\Big)\ .$$

$$\frac{\partial f(x_i)}{\partial B^{(1)}} = a\, u'\Big(z_i^{(3)}\Big)\, B^{(3)}\, u'\Big(z_i^{(2)}\Big)\, B^{(2)}\, u'\Big(z_i^{(1)}\Big)\ x_{ik},.$$

$$\frac{\partial f(x_i)}{\partial c^{(1)}} = a\, u'\Big(z_i^{(3)}\Big)\, B^{(3)}\, u'\Big(z_i^{(2)}\Big)\, B^{(2)}\, u'\Big(z_i^{(1)}\Big)$$

.

Generalisation on $j = (1, \ldots, J)$ layers, one neuron

$$\frac{\partial f(x)}{\partial B^{(j)}} \;=\; a \cdot \prod_{\ell=j}^{J} u'\Big(z_i^{(\ell)}\Big) \;\cdot\; \prod_{\ell=j+1}^{J} B^{(\ell)} \;\cdot\; \begin{cases} x_{ik} & \text{if } j = 1, \\ h_i^{(j-1)} & \text{if } j > 1. \end{cases}$$

and for c, the gradient becomes evident :

$$\frac{\partial f(x)}{\partial c^{(j)}} \;=\; a \cdot \prod_{\ell=j}^{J} u'\Big(z_i^{(\ell)}\Big) \;\cdot\; \prod_{\ell=j+1}^{J} B^{(\ell)}\ .$$

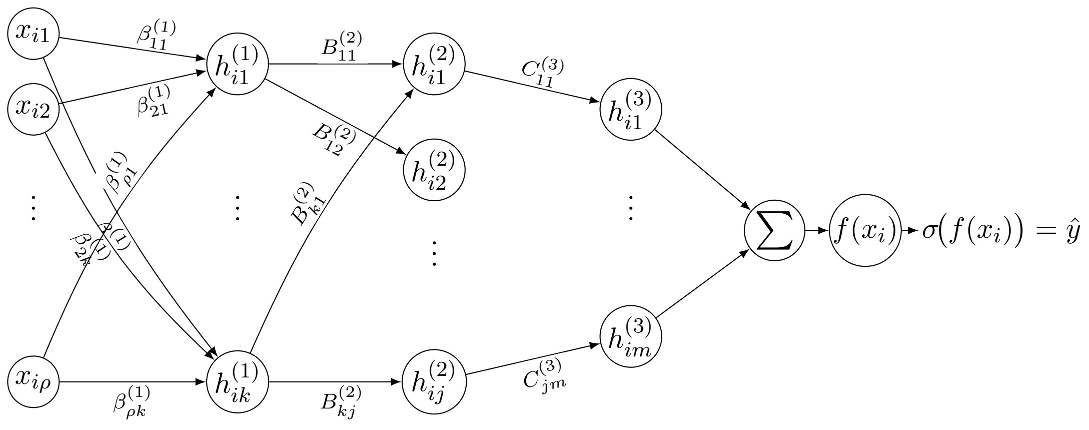
- If $j = J$, then $\prod_{\ell=j+1}^{J} B^{(\ell)}$ is an empty product, and by convention equals 1.

2 Multilayer Perceptron (MLP), binary case

We start with a 3-layer network and then generalize it to J layers:

$$\begin{aligned} \text{1st layer: } & K \text{ neurons,} \\ \text{2nd layer: } & J \text{ neurons,} \\ \text{3rd layer: } & M \text{ neurons.} \end{aligned}$$

Forward Pass



—

$$z_{ik}^{(1)} = \sum_{m=1}^{p} x_{im} B_{mk}^{(1)} + c_k^{(1)} \quad \Rightarrow \quad h_{ik}^{(1)} = u\Big(z_{ik}^{(1)}\Big), \quad \forall k \in \{1, \ldots, K\}$$

Matrix form:

$$(x_{i1}, \ldots, x_{ip}) \begin{bmatrix} B_{1k}^{(1)} \\ \vdots \\ B_{pk}^{(1)} \end{bmatrix} + c_k^{(1)}$$

Vector output:

$$h_i^{(1)} = \big(h_{i1}^{(1)}, \ldots, h_{iK}^{(1)}\big)$$

or in Matrix form :

$$\underset{n \times p}{\mathbf{X}} \;@\; \underset{p \times K}{\mathbf{B}^{(1)}} + \underset{n \times 1}{\mathbf{1}_n} \;@\; \underset{1 \times K}{\mathbf{c}^{(1)\top}} = \underset{n \times K}{\mathbf{Z}^{(1)}} \quad \Rightarrow \quad \underset{n \times K}{\mathbf{H}^{(1)}} = u\Big(\mathbf{Z}^{(1)}\Big)$$

—

$$z_{ij}^{(2)} = \sum_{k=1}^{K} h_{ik}^{(1)} B_{kj}^{(2)} + c_j^{(2)} \quad \Rightarrow \quad h_{ij}^{(2)} = u\Big(z_{ij}^{(2)}\Big), \quad \forall j \in \{1, \ldots, J\}$$

Matrix form:

$$(h_{i1}^{(1)}, \ldots, h_{iK}^{(1)}) \begin{bmatrix} B_{1j}^{(2)} \\ \vdots \\ B_{Kj}^{(2)} \end{bmatrix} + c_j^{(2)}$$

Vector output:

$$h_i^{(2)} = \big(h_{i1}^{(2)}, \ldots, h_{iJ}^{(2)}\big)$$

or in matrix format :

$$\underset{n \times K}{\mathbf{H}^{(1)}} \;@\; \underset{K \times J}{\mathbf{B}^{(2)}} + \underset{n \times 1}{\mathbf{1}_n} \;@\; \underset{1 \times J}{\mathbf{c}^{(2)\top}} = \underset{n \times J}{\mathbf{Z}^{(2)}} \quad \Rightarrow \quad \underset{n \times J}{\mathbf{H}^{(2)}} = u\Big(\mathbf{Z}^{(2)}\Big)$$

—

$$z_{im}^{(3)} = \sum_{j=1}^{J} h_{ij}^{(2)} B_{jm}^{(3)} + c_m^{(3)} \quad \Rightarrow \quad h_{im}^{(3)} = u\Big(z_{im}^{(3)}\Big), \quad \forall m \in \{1, \ldots, M\}$$

Matrix form:

$$(h^{(2)}_{i1}, \dots, h^{(2)}_{iJ}) \begin{bmatrix} B^{(3)}_{1m} \\ \vdots \\ B^{(3)}_{Jm} \end{bmatrix} + c^{(3)}_{m}$$

Vector output:

$$h^{(3)}_{i} = \big(h^{(3)}_{i1}, \dots, h^{(3)}_{iM}\big)$$

or in matrix format :

$$\underset{n\times J}{\mathbf{H}^{(2)}} \;@\; \underset{J\times M}{\mathbf{B}^{(3)}} \;+\; \underset{n\times 1}{\mathbf{1}_n} \;@\; \underset{1\times M}{\mathbf{c}^{(3)\top}} \;=\; \underset{n\times M}{\mathbf{Z}^{(3)}} \quad\Rightarrow\quad \underset{n\times M}{\mathbf{H}^{(3)}} = u\Big(\mathbf{Z}^{(3)}\Big)$$

—

Output Layer

$$f(x_i) = \sum_{o=1}^{M} h^{(3)}_{io} a_o + b$$

or in matrix format :

$$\underset{n\times 1}{\mathbf{f}} \;=\; \underset{n\times M}{\mathbf{H}^{(3)}} \;@\; \underset{M\times 1}{\mathbf{a}} \;+\; \underset{n\times 1}{b\,\mathbf{1}_n} \quad\Rightarrow\quad \underset{n\times 1}{\hat{\mathbf{y}}} = \sigma(\mathbf{f})$$

$$\hat{y} = \sigma(f(x_i)) = P(y = 1 \mid x)$$

In binary classification for the output layer we use the CDF of the logistic law (sigmoid) to return values between 0 and 1. Thus, as in logistic regression, we maximise our log-likelihood:

$$L = \sum_{i=1}^{N} \Big(y_i \log(\sigma(f(x_i))) + (1-y_i)\log(1-\sigma(f(x_i)))\Big), \quad \forall i \in \{1,\dots,N\}$$

where

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{is the sigmoid function,}$$

Gradients

Log-likelihood and output residual.

$$\ell_i = y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i), \qquad L = \sum_{i=1}^{N} \ell_i, \qquad \delta_i = \frac{\partial \ell_i}{\partial f(x_i)} = y_i - \hat{y}_i.$$

Because

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

Output layer gradients

$$\frac{\partial \ell_i}{\partial a_m} = \delta_i\, h^{(3)}_{im}, \qquad \frac{\partial \ell_i}{\partial b} = \delta_i, \qquad L = \sum_{i=1}^{N} \ell_i$$

$$\frac{\partial L}{\partial a_m} = \sum_{i=1}^{N} \delta_i\, h^{(3)}_{im}, \qquad \frac{\partial L}{\partial b} = \sum_{i=1}^{N} \delta_i.$$

Matrix form Let $H^{(3)} = \begin{bmatrix} h_{im}^{(3)} \end{bmatrix} \in \mathbb{R}^{N\times M}$, $\delta = \begin{bmatrix}\delta_1 & \cdots & \delta_N\end{bmatrix}^\top \in \mathbb{R}^N$, and $\mathbf{1} = \begin{bmatrix}1 & \cdots & 1\end{bmatrix}^\top \in \mathbb{R}^N$. Then

$$\nabla_a L \;=\; (H^{(3)})^\top \delta \;=\; \begin{bmatrix} h_{11}^{(3)} & \cdots & h_{1M}^{(3)} \\ \vdots & \ddots & \vdots \\ h_{N1}^{(3)} & \cdots & h_{NM}^{(3)} \end{bmatrix}^\top \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_N \end{bmatrix} \;=\; \begin{bmatrix} \sum_{i=1}^N h_{i1}^{(3)}\,\delta_i \\ \vdots \\ \sum_{i=1}^N h_{iM}^{(3)}\,\delta_i \end{bmatrix}.$$

$$\nabla_b L \;=\; \mathbf{1}^\top \delta \;=\; \begin{bmatrix}1 & \cdots & 1\end{bmatrix} \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_N \end{bmatrix} \;=\; \sum_{i=1}^N \delta_i.$$

Backprop error into layer 3. For each sample i and unit m,

$$e_{im}^{(3)} := \frac{\partial \ell_i}{\partial z_{im}^{(3)}} = \underbrace{\frac{\partial \ell_i}{\partial f(x_i)}}_{\delta_i}\,\underbrace{\frac{\partial f(x_i)}{\partial h_{im}^{(3)}}}_{a_m}\,\underbrace{\frac{\partial h_{im}^{(3)}}{\partial z_{im}^{(3)}}}_{u'(z_{im}^{(3)})} = \delta_i\, a_m\, u'\big(z_{im}^{(3)}\big).$$

Matrix form Let $\delta = \begin{bmatrix}\delta_1 & \cdots & \delta_N\end{bmatrix}^\top \in \mathbb{R}^N$, $a = \begin{bmatrix}a_1 & \cdots & a_M\end{bmatrix}^\top \in \mathbb{R}^M$, and $U' := u'(Z^{(3)}) = [\,u'(z_{im}^{(3)})\,] \in \mathbb{R}^{N\times M}$. Then

$$E^{(3)} \;=\; (\delta\, a^\top) \;\odot\; U'.$$

with

$$\delta\, a^\top \;=\; \begin{bmatrix} \delta_1 a_1 & \delta_1 a_2 & \cdots & \delta_1 a_M \\ \delta_2 a_1 & \delta_2 a_2 & \cdots & \delta_2 a_M \\ \vdots & \vdots & \ddots & \vdots \\ \delta_N a_1 & \delta_N a_2 & \cdots & \delta_N a_M \end{bmatrix}, \qquad U' \;=\; \begin{bmatrix} u'(z_{11}^{(3)}) & u'(z_{12}^{(3)}) & \cdots & u'(z_{1M}^{(3)}) \\ u'(z_{21}^{(3)}) & u'(z_{22}^{(3)}) & \cdots & u'(z_{2M}^{(3)}) \\ \vdots & \vdots & \ddots & \vdots \\ u'(z_{N1}^{(3)}) & u'(z_{N2}^{(3)}) & \cdots & u'(z_{NM}^{(3)}) \end{bmatrix}.$$

Hadamard-multiplying entrywise gives

$$E^{(3)} \;=\; (\delta\, a^\top) \odot U' \;=\; \begin{bmatrix} \delta_1 a_1 u'(z_{11}^{(3)}) & \delta_1 a_2 u'(z_{12}^{(3)}) & \cdots & \delta_1 a_M u'(z_{1M}^{(3)}) \\ \delta_2 a_1 u'(z_{21}^{(3)}) & \delta_2 a_2 u'(z_{22}^{(3)}) & \cdots & \delta_2 a_M u'(z_{2M}^{(3)}) \\ \vdots & \vdots & \ddots & \vdots \\ \delta_N a_1 u'(z_{N1}^{(3)}) & \delta_N a_2 u'(z_{N2}^{(3)}) & \cdots & \delta_N a_M u'(z_{NM}^{(3)}) \end{bmatrix},$$

Layer 3 parameter gradients (per sample i).

$$\frac{\partial \ell_i}{\partial B_{jm}^{(3)}} \;=\; e_{im}^{(3)}\, h_{ij}^{(2)}, \qquad \frac{\partial \ell_i}{\partial c_m^{(3)}} \;=\; e_{im}^{(3)}.$$

Summed over samples for $L = \sum_{i=1}^N \ell_i$.

$$\frac{\partial L}{\partial B_{jm}^{(3)}} \;=\; \sum_{i=1}^N h_{ij}^{(2)}\, e_{im}^{(3)}, \qquad \frac{\partial L}{\partial c_m^{(3)}} \;=\; \sum_{i=1}^N e_{im}^{(3)}.$$

Matrix form Let $H^{(2)} = [h_{ij}^{(2)}] \in \mathbb{R}^{N\times J}$ and $E^{(3)} = [e_{im}^{(3)}] \in \mathbb{R}^{N\times M}$. Then

$$\boxed{\nabla_{B^{(3)}} L \;=\; (H^{(2)})^\top E^{(3)}} \quad \in \mathbb{R}^{J\times M}, \qquad \boxed{\nabla_{c^{(3)}} L \;=\; \mathbf{1}^\top E^{(3)}} \quad \in \mathbb{R}^{1\times M}.$$

with

$$(H^{(2)})^\top = \begin{bmatrix} h_{11}^{(2)} & h_{21}^{(2)} & \cdots & h_{N1}^{(2)} \\ h_{12}^{(2)} & h_{22}^{(2)} & \cdots & h_{N2}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ h_{1J}^{(2)} & h_{2J}^{(2)} & \cdots & h_{NJ}^{(2)} \end{bmatrix}, \qquad E^{(3)} = \begin{bmatrix} e_{11}^{(3)} & e_{12}^{(3)} & \cdots & e_{1M}^{(3)} \\ e_{21}^{(3)} & e_{22}^{(3)} & \cdots & e_{2M}^{(3)} \\ \vdots & \vdots & \ddots & \vdots \\ e_{N1}^{(3)} & e_{N2}^{(3)} & \cdots & e_{NM}^{(3)} \end{bmatrix}.$$

Backprop error into layer 2. For each sample i and unit j,

$$e_{ij}^{(2)} := \frac{\partial \ell_i}{\partial z_{ij}^{(2)}} = \left(\sum_{m=1}^{M} e_{im}^{(3)} B_{jm}^{(3)} \right) u'\big(z_{ij}^{(2)}\big).$$

Matrix form (collect all $e_{ij}^{(2)}$ into $E^{(2)} \in \mathbb{R}^{N\times J}$). Let $E^{(3)} = [e_{im}^{(3)}] \in \mathbb{R}^{N\times M}$, $B^{(3)} = [B_{jm}^{(3)}] \in \mathbb{R}^{J\times M}$, and $U_2' := u'(Z^{(2)}) = [\, u'(z_{ij}^{(2)})\,] \in \mathbb{R}^{N\times J}$. Then

$$\boxed{E^{(2)} \;=\; \big(E^{(3)}(B^{(3)})^\top\big) \;\odot\; U_2'} \quad \in \mathbb{R}^{N\times J}.$$

by similar reasoning

Thus, we can easily generalise on L layers:

$\forall l \in \{L, \dots, 1\}$.

$$E^{(l)} = \begin{cases} (E^{(l+1)} B^{(l+1)\top}) \odot u'\big(Z^{(l)}\big), & \in \mathbb{R}^{N\times \mathrm{idx}(l)}, \quad l < L, \\ (\delta\, a^\top) \;\odot\; u'\big(Z^{(l)}\big), & \in \mathbb{R}^{N\times \mathrm{idx}(L)}, \quad l = L. \end{cases} \tag{1}$$

$$\frac{\partial L}{\partial B^{(l)}} = \begin{cases} (H^{(l-1)})^\top E^{(l)} & \in \mathbb{R}^{\mathrm{idx}(l-1)\times \mathrm{idx}(l)}, \quad l > 1, \\ X^\top E^{(1)} & \in \mathbb{R}^{\mathrm{idx}(X.\mathrm{col})\times \mathrm{idx}(1)}, \quad l = 1. \end{cases} \tag{2}$$

$$\frac{\partial L}{\partial c^{(l)}} \;=\; \mathbf{1}^\top E^{(l)} \;\in\; \mathbb{R}^{1\times \mathrm{idx}(l)}, \qquad \forall\, l.$$

$$\frac{\partial L}{\partial a} \;=\; (H^{(L)})^\top \delta \;\in\; \mathbb{R}^{\mathrm{idx}(L)\times 1}, \qquad \frac{\partial L}{\partial b} \;=\; \mathbf{1}^\top \delta \;\in\; \mathbb{R}.$$

3 Softmax MLP

In multi-class classification, for the output layer we use the softmax function:

$$P_{i,v} = \mathrm{softmax}(f(x_i)_v) = \frac{\exp\big(f(x_i)_v\big)}{\sum_{j=1}^{V} \exp\big(f(x_i)_j\big)}, \qquad i = 1, \dots, n, \; v = 1, \dots, V,$$

Thus, as in logistic regression, we maximise our log-likelihood:

$$L = \sum_{i=1}^{N}\sum_{v=1}^{V} \mathbf{1}(y_i = v)\,\log\!\left(\frac{\exp\big(f(x_i)_v\big)}{\sum_{j=1}^{V}\exp\big(f(x_i)_j\big)}\right), \quad \forall i \in \{1,\dots,N\} \quad \forall v \in \{1,\dots,V\}.$$

$$\frac{\partial l_i}{\partial f(x_i)_v} = \mathbf{1}(y_i = v) - P_{i,v} = \delta_{iv} \quad \forall i \in \{1,\dots,N\} \quad \forall v \in \{1,\dots,V\}. \tag{3}$$

Let

$$\Delta \;=\; \begin{pmatrix} \mathbf{1}(y_1 = 1) - P_{1,1} & \cdots & \mathbf{1}(y_1 = V) - P_{1,V} \\ \vdots & \ddots & \vdots \\ \mathbf{1}(y_N = 1) - P_{N,1} & \cdots & \mathbf{1}(y_N = V) - P_{N,V} \end{pmatrix} \;\in\; \mathbb{R}^{N\times V}.$$

Also, the output layer is simply

$$f(x_i)_v = \sum_{m=1}^{M} h^{(3)}_{im}\,A_{mv} \;+\; b^{(3)}_v, \qquad \forall v \in \{1,\dots,V\}.$$

The gradients are then given by

$$\frac{\partial L}{\partial A} \;=\; H^{(3)\top}\Delta \;\in\; \mathbb{R}^{M\times V}, \qquad \frac{\partial L}{\partial b^{(3)}} \;=\; \mathbf{1}_N^{\top}\Delta \;\in\; \mathbb{R}^{1\times V}.$$

$$e^{(3)}_{im} = \sum_{v=1}^{V} \delta_{iv}\,A_{mv}\,u'\big(z^{(3)}_{im}\big) = \Delta A^{\top} \odot\; u'(Z^{(3)}) \in \mathbb{R}^{N\times M},$$

other gradients stay the same
so the generalisation rule becomes:
$\forall l \in \{L,\dots,1\}$.

$$E^{(l)} = \begin{cases} (E^{(l+1)}B^{(l+1)\top}) \odot u'\big(Z^{(l)}\big)\,, & \in \mathbb{R}^{N\times \mathrm{idx}(l)}, \quad l < L, \\ (\Delta\, A^{\top}) \;\odot\; u'\big(Z^{(l)}\big)\,, & \in \mathbb{R}^{N\times \mathrm{idx}(L)}, \quad l = L. \end{cases} \tag{4}$$

$$\frac{\partial L}{\partial B^{(l)}} = \begin{cases} (H^{(l-1)})^{\top}\,E^{(l)} & \in \mathbb{R}^{\mathrm{idx}(l-1)\times \mathrm{idx}(l)}, \quad l > 1, \\ X^{\top}\,E^{(1)} & \in \mathbb{R}^{\mathrm{idx}(X.\mathrm{col})\times \mathrm{idx}(1)}, \quad l = 1. \end{cases} \tag{5}$$

$$\frac{\partial L}{\partial c^{(l)}} \;=\; \mathbf{1}^{\top}E^{(l)} \;\in\; \mathbb{R}^{1\times \mathrm{idx}(l)}, \qquad \forall\, l.$$

$$\frac{\partial L}{\partial A} \;=\; (H^{(L)})^{\top}\Delta \;\in\; \mathbb{R}^{\mathrm{idx}(L)\times V}, \qquad \frac{\partial L}{\partial b} \;=\; \mathbf{1}^{\top}\Delta \;\in\; \mathbb{R}^{1\times V}.$$

Remark. If the objective is the *mean* log-loss instead of the total log-likelihood, then final layer E must be scaled by a factor of $\frac{1}{N}$.

4 Penalizations

4.1 L2

Neural network training as a constrained program. We maximize the log-likelihood subject to an ℓ_2 constraint on the weights:

$$\max_{W} \; L(W) \tag{6}$$

$$\text{s.t.} \; \|W\|^2 \; \leq \; C, \tag{7}$$

where $L(W) = \sum_{i=1}^{N} \ell_i(W)$ is the total log-likelihood, $W = \{B^{(1)}, B^{(2)}, B^{(3)}, a\}$ are the weights, and $\|W\|^2 = \sum_{l=1}^{3} \|B^{(l)}\|_F^2 + \|a\|_2^2$ (binary head) or $\|W\|^2 = \sum_{l=1}^{3} \|B^{(l)}\|_F^2 + \|A\|_F^2$ (softmax)

Lagrangian. The associated Lagrangian is

$$\mathcal{L}(W, \mu) = L(W) \; - \; \mu\big(\|W\|^2 - C\big), \qquad \mu \geq 0.$$

Then the gradients will be :

$$\frac{\partial L_\lambda}{\partial B^{(l)}} = \begin{cases} (H^{(l-1)})^\top E^{(l)} \; - \; \lambda B^{(l)}, & l > 1, \\ X^\top E^{(1)} \; - \; \lambda B^{(1)}, & l = 1, \end{cases}. \tag{8}$$

Output layer (binary).

$$\frac{\partial L_\lambda}{\partial a} = (H^{(L)})^\top \delta \; - \; \lambda a,$$

Output layer (softmax).

$$\frac{\partial L_\lambda}{\partial A} = (H^{(L)})^\top \Delta \; - \; \lambda A,$$

4.2 Dropout

Dropout. For each layer l, define a dropout mask

$$m_{i,idx(l)}^{(l)} \sim \text{Bernoulli}(p) \qquad \forall i \in \{1, \ldots, N\}, \forall idx(l) \in \{1, \ldots, IDX(L)\}.$$

where $p \in [0, 1]$ is the dropout probability and entries $M_{ij}^{(l)} \in \{0, 1\}$ are independent samples. The dropout-modified hidden activations are

$$\tilde{H}^{(l)} = \frac{1}{p}\big(H^{(l)} \odot M^{(l)}\big),$$

where $\odot$ denotes elementwise multiplication. This scaling ensures that $\mathbb{E}[\tilde{H}^{(l)}] = H^{(l)}$, since $\mathbb{E}[m_{i,idx(l)}^{(l)}] = p$

$$\text{Var}(\tilde{h}) = h^2 \, \frac{1 - p}{p}$$

Backpropagation with Dropout. The error propagation becomes

$$E^{(l)} = \begin{cases} \left(E^{(l+1)} B^{(l+1)\top}\right) \odot u'\big(Z^{(l)}\big) \odot M^{(l)} \frac{1}{p}, & l < L, \\ \left(\Delta A^\top\right) \odot u'\big(Z^{(L)}\big) \odot M^{(L)} \frac{1}{p}, & l = L. \end{cases}$$

$$\frac{\partial L}{\partial B^{(l)}} = \begin{cases} \big(\tilde{H}^{(l-1)}\big)^\top E^{(l)}, & l > 1, \\ X^\top E^{(1)}, & l = 1, \end{cases}$$

$$\frac{\partial L}{\partial c^{(l)}} = \mathbf{1}^\top E^{(l)}, \qquad \forall l.$$

Finally, for softmax or sigmoid (suppose here softmax, but formulas are same)

$$\frac{\partial L}{\partial A} = \big(\tilde{H}^{(L)}\big)^\top \Delta, \qquad \frac{\partial L}{\partial b} = \mathbf{1}^\top \Delta.$$

Example. Suppose

$$H^{(l-1)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad M^{(l-1)} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B^{(l)} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

Applying dropout:

$$\tilde{H}^{(l-1)} = H^{(l-1)} \odot M^{(l-1)} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 5 & 0 \end{bmatrix}.$$

Now compute the pre-activation:

$$Z^{(l)} = \tilde{H}^{(l-1)} B^{(l)} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 5 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

...

4.3 Early Stopping

Early stopping is a regularization technique where training is stopped before the loss function fully converges, in order to prevent overfitting.

- **Standard training:** The usual approach is to minimize the training loss function (or maximise in my case loglikelihood or minimise negative log likelihood (cross-entropy)

$$L_{\text{train}}(\theta) \quad \text{until convergence.}$$

  This often leads to overfitting, where the training error decreases but the validation error increases.

- **Early stopping:** Instead of training until convergence, we stop earlier, i.e.

$$t^* = \arg\min_t L_{\text{val}}(\theta_t),$$

  where t is the training epoch and L_{val} is the validation loss.

- **Stopping criteria:** Several rules can be applied:

1. *Validation-based:* stop if validation error does not decrease for k consecutive epochs, i.e.
$$L_{\text{val}}(\theta_t) \geq L_{\text{val}}(\theta_{t-k}), \quad \forall\, t > T.$$
2. *Training-based:* stop if training loss does not decrease "enough" between epochs, for example
$$L_{\text{train}}(\theta_{t-1}) - L_{\text{train}}(\theta_t) < \varepsilon,$$
where $\varepsilon > 0$ is a small threshold.

4.4 Data Augmentation

Data augmentation is a regularization technique in which the training set is artificially enlarged by applying label-preserving transformations to the input data. Examples include random flips, rotations, crops, or color jittering for images; synonym replacement or back-translation for text; and time-shifting or pitch-shifting for audio.

- **Motivation:** Prevents overfitting by forcing the model to learn invariant, generalizable features instead of memorizing the training samples.
- **Formal definition:** Given a dataset $\{(x_i, y_i)\}_{i=1}^N$, define a random transformation $T \sim \mathcal{D}$ drawn from a distribution of augmentation operations. Each training pair is transformed as
$$(x_i, y_i) \;\mapsto\; (T(x_i), y_i).$$
- **Training objective with augmentation:**
$$\min_\theta \; \mathbb{E}_{T\sim\mathcal{D}}\big[L(f_\theta(T(x)), y)\big],$$
where f_θ is the model, L is the loss function, and θ are the trainable parameters.

Thus, data augmentation can be seen as a form of stochastic regularization that injects noise into the inputs, complementing techniques like dropout (noise in the activations) or weight decay (constraints on parameters).

5 Optimisations

5.1 Gradient Descent/ascent

$$\text{(Gradient descent)} \quad \theta_{t+1} \to \theta_t + / - \frac{\alpha_t}{N}\sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

$$\text{(Stochastic gradient descent (SGD))} \quad \theta_{t+1} \to \theta_t + / - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$$

- With appropriate sampling, stochastic gradients are equal in expectancy to full gradient $\to$ No bias
- **Pros:**
  - Does not calculate N gradients
  - Works on infinite data or online

- **Cons:** Introduces variance in the gradient

$$\text{(SGD with mini-batches)} \quad \theta_{t+1} \to \theta_t + / - \frac{\alpha_t}{K} \sum_{i=1}^{K} \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

6 Initialisations

Sampling in $W^{(l)}$ *independently and identically* from either a normal or uniform distribution.

6.1 Zero Initialization.

Zero initialization is usually used for initializing biases, but it is not used for initializing weights, as it leads to symmetry in the network, causing all neurons to learn the same features.

6.2 Random Initialization.

Random initialization means sampling the weights from a normal distribution or a uniform distribution, usually independently.

6.3 LeCun Initialization.

LeCun initialization is designed to preserve the variance of activations across layers, especially when using activation functions such as **sigmoid**, **tanh**, or **SELU**, linear or identity. It scales the weights based on the number of input units to each layer.

Let fan_{in} denote the number of input units to a neuron.

- **LeCun Normal:**

$$W \sim \mathcal{N}\left(0, \frac{1}{fan_{in}}\right)$$

- **LeCun Uniform:**

$$W \sim U\left(-\sqrt{\frac{3}{fan_{in}}}, \sqrt{\frac{3}{fan_{in}}}\right)$$

Note

ReLU and Variants: Prefer **He initialization**.

Example: For Layer 2 with weights $\mathbf{B}^{(2)} \in \mathbb{R}^{K \times J}$, we have $fan_{in} = K$ (inputs) and $fan_{out} = J$ (outputs).

6.4 Xavier (Glorot) Initialization

Xavier initialization (also called Glorot initialization) was introduced by Glorot and Bengio (2010) to maintain the variance of activations and backpropagated gradients across layers. It is especially suitable for activation functions such as the **tanh**, **sigmoid** .

- **Xavier Normal:**

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{fan_{in} + fan_{out}}}\right)$$

- **Xavier Uniform:**

$$W \sim U\left(-\sqrt{\frac{6}{fan_{in} + fan_{out}}}, \sqrt{\frac{6}{fan_{in} + fan_{out}}}\right)$$

Note

ReLU and Variants: Prefer **He initialization**.

When the fan-in and fan-out are equal, then Glorot initialization is the same as LeCun initialization.

6.5 He Initialization

He initialization (also called Kaiming initialization) was introduced by He et al. (2015) for networks with **ReLU** activations.

- **He Normal:**
$$W \sim \mathcal{N}\left(0, \frac{2}{fan_{in}}\right)$$
- **He Uniform:**
$$W \sim U\left(-\sqrt{\frac{6}{fan_{in}}}, \sqrt{\frac{6}{fan_{in}}}\right)$$

7 Batch normalisation