# Basic MLP with manually-derived Backprop

Teo Asinari

November 2, 2023

## 1 Introduction

**Goal:** To design, train and use a simple 3-layer MLP for binary classification of size-2 vectors.

**Design:** of the form

$$[(layer\_size, Activation)...]$$

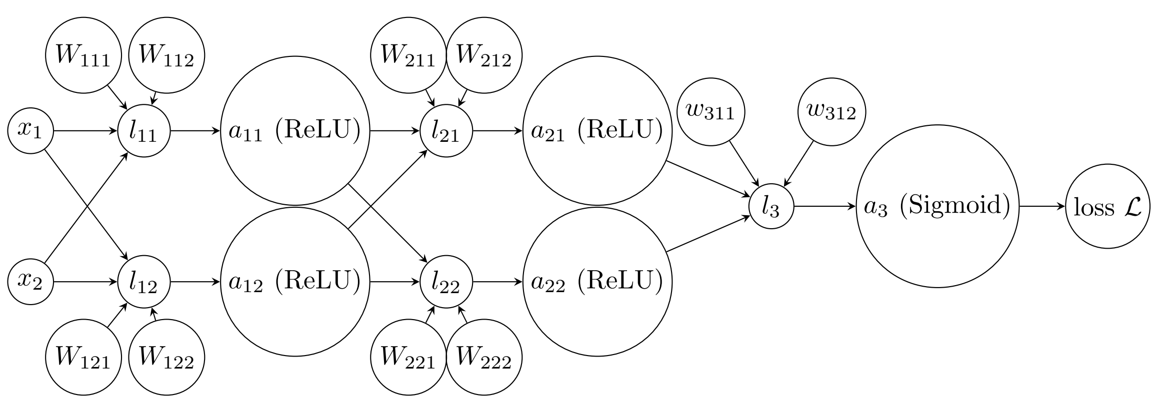
: [(2, ReLU), (2, ReLU), (1, Sigmoid)]

### 1.1 Diagrams

#### 1.1.1 Vectorized Diagram (Equiv to Roger Grosse' 'Computational Graph')

$$\begin{tikzcd}
\boldsymbol{W_1} \arrow[dr] & & \boldsymbol{W_2} \arrow[dr] & & \boldsymbol{w_3} \arrow[dr] & & t \arrow[dr] & \\
\boldsymbol{x} \arrow[r] & \boldsymbol{l_1} \arrow[r] & \boldsymbol{a_1} \arrow[r] & \boldsymbol{l_2} \arrow[r] & \boldsymbol{a_2} \arrow[r] & l_3 \arrow[r] & a_3 \arrow[r] & \mathcal{L}
\end{tikzcd}$$

#### 1.1.2 Expanded Diagram (Equiv. to Roger Grosse' 'Network Architecture')



## 1.2 Definitions

### 1.2.1 Remark on general notation

Since it seems that mathematical notation in this field tends to suffer from overloading, imprecision/lack of specificity, a lack of convention, poor readability, and a generally poor aesthetic/deisgn sense, I am going to try to avoid worsening the situation. So, for the purpose of this work, all non-bold variables denote scalars. A bold lower-case variable denotes a vector, a bold upper-case a Matrix. A non-bold lower-case may denotes either an arbitrary scalar variable or if subscripted typically an element of a vector. A non-bold upper-case, if subscripted, will typically denote a matrix element.

### 1.2.2 Remark on weight notation

$w_{i,j,k}$ is to say the weight at the $i$-th layer, $j$-th neuron, $k$-th weight. Hence $w_{111}$ is the first weight of the first neuron in the first layer, etc.

### 1.2.3 Remark on layer notation

This is a sub-case of the weight notation. I.e., $L_{ij}$ is the scalar value of the $j$-th neuron at the $i$-th layer, etc.

### 1.2.4 Neuron firing calculation

This is just a straightforward dot-product. We have:

$$l_{ij} = \boldsymbol{W}_{ij}\boldsymbol{x}_i$$

Where $\boldsymbol{x}_i$ in this case is referring to a more general notion of 'layer input', not necessarily just the first input to the network as in the diagrams above.

## 1.3 BackPropagation Derivation

**Notation for derivative of loss w.r.t. to a function** I will be using the following: $\overline{f} = \frac{\partial \mathcal{L}}{\partial f}$. This notation was introduced by Roger Grosse from the University of Toronto.

**Pa(x) and Ch(x)** these refer to the sets of parent and child vertices of a vertex in a graph.

**General Approach** Let's label the computational graph nodes as $v_1, ..., v_N$ with some topological ordering. Then, our general goal for backprop is to compute $\overline{v_i}$ for $i \in 1, ...N$. With these, we can trivially calculate the weight updates. We compute a forward pass of the network, then set $v_N = 1$, then, for $i = N-1, ..., 1$, we have:

$$\overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j}\frac{\partial v_j}{\partial v_i} \quad \text{(The Backprop Rule)} \tag{1}$$

### 1.3.1 Applying the backprop rule

**Loss and final activation** Then, going backwards through the 'computational graph', starting at the end:

$$\overline{\mathcal{L}} = 1 \tag{2}$$

$$\overline{a_3} = \overline{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial a_3}$$

$$\overline{a_3} = (1) \frac{\partial \mathcal{L}}{\partial a_3}$$

$$\overline{a_3} = \frac{\partial \mathcal{L}}{\partial a_3}$$

$$\overline{a_3} = \frac{\partial}{\partial a_3} \frac{1}{2}(a_3 - t)^2$$

$$\overline{a_3} = (a_3 - t) \frac{\partial}{\partial a_3}(a_3 - t)$$

$$\overline{a_3} = (a_3 - t)(1)$$

$$\overline{a_3} = (a_3 - t) \tag{3}$$

**Final layer** ***N.B.*** I use $\sigma$ to denote the sigmoid function here, not an activation function.

$$\overline{l_3} = \overline{a_3} \frac{\partial a_3}{\partial l_3}$$

$$\overline{l_3} = \overline{a_3} \frac{\partial}{\partial l_3} \sigma(l_3)$$

$$\overline{l_3} = \overline{a_3}\ \sigma(l_3)(1 - \sigma(l_3)) \tag{4}$$

**Final layer weights**

$$\overline{w_{31i}} = \overline{l_3} \frac{\partial}{\partial w_{31i}} l_3$$

$$\overline{w_{31i}} = \overline{l_3} \frac{\partial}{\partial w_{31i}} \sum_j w_{31j} a_{2j}$$

$$\overline{w_{31i}} = \overline{l_3} a_{2i} \tag{5}$$

**Second layer activation**

$$\overline{a_{2i}} = \overline{l_3} \frac{\partial}{\partial a_{2i}} l_3$$

$$\overline{a_{2i}} = \overline{l_3} \frac{\partial}{\partial a_{2i}} \sum_j w_{31j} a_{2j}$$

$$\overline{a_{2i}} = \overline{l_3} w_{31i} \tag{6}$$

**Second layer**

$$\overline{l_{2i}} = \overline{a_{2i}} \frac{\partial}{\partial l_{2i}} a_{2i}$$

$$\overline{l_{2i}} = \overline{a_{2i}} \frac{\partial}{\partial l_{2i}} \mathrm{ReLU}(l_{2i})$$

Note that $d/dx(\mathrm{ReLU}(x))$ is the heaviside step function $\theta(x)$:

$$\begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\overline{l_{2i}} = \overline{a_{2i}} \quad \theta(l_{2i}) \cdot (1)$$

$$\overline{l_{2i}} = \overline{a_{2i}} \quad \theta(l_{2i}) \tag{7}$$

**Second layer weights**

$$\overline{W_{2ij}} = \overline{l_{2i}} \frac{\partial}{\partial W_{2ij}} l_{2i}$$

$$\overline{W_{2ij}} = \overline{l_{2i}} \frac{\partial}{\partial W_{2ij}} \sum_k W_{2ik} a_{1k}$$

$$\overline{W_{2ij}} = \overline{l_{2i}} a_{1j} \tag{8}$$

**Input layer activation**

$$\overline{a_{1i}} = \overline{l_{21}} \frac{\partial}{\partial a_{1i}} l_{21} + \overline{l_{22}} \frac{\partial}{\partial a_{1i}} l_{22}$$

$$\overline{a_{1i}} = \overline{l_{21}} \frac{\partial}{\partial a_{1i}} \sum_j W_{21j} a_{1j} + \overline{l_{22}} \frac{\partial}{\partial a_{1i}} \sum_j W_{22j} a_{1j}$$

$$\overline{a_{1i}} = \overline{l_{21}} W_{21i} + \overline{l_{22}} W_{22i} \tag{9}$$

**Input layer**

$$\overline{l_{1i}} = \overline{a_{1i}} \frac{\partial}{\partial l_{1i}} a_{1i}$$

$$\overline{l_{1i}} = \overline{a_{1i}} \frac{\partial}{\partial l_{1i}} \mathrm{ReLU}(l_{1i})$$

$$\overline{l_{1i}} = \overline{a_{1i}} \ \theta(l_{1i}) \cdot (1)$$

$$\overline{l_{1i}} = \overline{a_{1i}} \ \theta(l_{1i}) \tag{10}$$

**Input layer weights**

$$\overline{W_{1ij}} = \overline{l_{1i}} \frac{\partial}{\partial W_{1ij}} l_{1i}$$

$$\overline{W_{1ij}} = \overline{l_{1i}} \frac{\partial}{\partial W_{1ij}} \sum_k W_{1ik} x_{1k}$$

$$\overline{W_{1ij}} = \overline{l_{1i}} x_{1j} \tag{11}$$

**Notes to self** Grosse follows a per-element approach first, then somehow transformed those results into a vectorized form involving (in some cases) re-arranged multiplications and matrix transpose. I am somewhat confused/overwhelmed by this.

#### 1.3.2 Vectorized derivation:

#### 1.3.3 Forward pass vectorized:

$$\boldsymbol{l}_1 = \boldsymbol{W}_1 \cdot \boldsymbol{x}$$

$$\boldsymbol{a}_1 = \mathrm{ReLU}(\boldsymbol{l}_1)$$

$$\boldsymbol{l}_2 = \boldsymbol{W}_2 \cdot \boldsymbol{a}_1$$

$$\boldsymbol{a}_2 = \mathrm{ReLU}(\boldsymbol{l}_2)$$

$$l_3 = \boldsymbol{w}_3 \cdot \boldsymbol{a}_2$$

$$a_3 = \sigma(l_3)$$

$$\mathcal{L} = \frac{1}{2}(a_3 - t)^2$$

#### 1.3.4 Backpropagation vectorized:

$$\overline{\mathcal{L}} = 1 \tag{12}$$

$$\overline{a_3} = \overline{\mathcal{L}}(a_3 - t)$$

$$\overline{a_3} = (a_3 - t) \text{ (Scalar)}$$

$$\overline{l_3} = \overline{a_3} \times \frac{\partial}{\partial l_3} a_3$$

$$\overline{l_3} = \overline{a_3} \times \frac{\partial}{\partial l_3} \sigma(l_3)$$

$$\overline{l_3} = \overline{a_3} \times \sigma'(l_3) \text{ (scalar)} \tag{13}$$

$$\overline{\boldsymbol{w}_3} = \overline{l_3} \frac{\partial}{\partial \boldsymbol{w}_3} l_3$$

$$\overline{\boldsymbol{w}_3} = \overline{l_3} \frac{\partial}{\partial \boldsymbol{w}_3} \boldsymbol{w}_3 \cdot \boldsymbol{a}_2$$

$$\overline{\boldsymbol{w}_3} = \overline{l_3}\begin{bmatrix}\frac{\partial}{\partial w_{311}}(w_{311}a_{21} + w_{312}a_{22}) & \frac{\partial}{\partial w_{312}}(w_{311}a_{21} + w_{312}a_{22})\end{bmatrix}$$

$$\overline{\boldsymbol{w}_3} = \overline{l_3}\begin{bmatrix}a_{21} & a_{22}\end{bmatrix}$$

$$\overline{\boldsymbol{w}_3} = \overline{l_3}\boldsymbol{a}_2^T \text{ (1,2 vector)} \tag{14}$$

$$\overline{\boldsymbol{a}_2} = \overline{l_3}\frac{\partial}{\partial \boldsymbol{a}_2} l_3$$

$$\overline{\boldsymbol{a}_2} = \overline{l_3}\frac{\partial}{\partial \boldsymbol{a}_2} \boldsymbol{w}_3 \cdot \boldsymbol{a}_2$$

$$\overline{\boldsymbol{a}_2} = \overline{l_3}\begin{bmatrix}\frac{\partial}{\partial a_{21}}(w_{311}a_{21} + w_{312}a_{22}) \\ \frac{\partial}{\partial a_{22}}(w_{311}a_{21} + w_{312}a_{22})\end{bmatrix}$$

$$\overline{\boldsymbol{a}_2} = \overline{l_3}\begin{bmatrix}w_{311} \\ w_{312}\end{bmatrix}$$

$$\overline{\boldsymbol{a}_2} = \overline{l_3}\boldsymbol{w}_3^T \text{ (2,1 vector)} \tag{15}$$

$$\overline{\boldsymbol{l}_2} = \begin{bmatrix}\overline{a_{21}}\theta(l_{21}) \\ \overline{a_{22}}\theta(l_{22})\end{bmatrix}$$

$$\overline{\boldsymbol{l}_2} = \overline{\boldsymbol{a}_2} \circ \theta(\boldsymbol{l}_2) \text{ ($\circ$ is the hadamard/element-wise product)} \tag{16}$$

$$\overline{\boldsymbol{W}_2} = \begin{bmatrix}\overline{l_{21}}a_{11} & \overline{l_{21}}a_{12} \\ \overline{l_{22}}a_{11} & \overline{l_{22}}a_{12}\end{bmatrix}$$

$$\overline{\boldsymbol{W}_2} = \begin{bmatrix}\overline{l_{21}} & \overline{l_{21}} \\ \overline{l_{22}} & \overline{l_{22}}\end{bmatrix} \circ \begin{bmatrix}a_{11} & a_{12} \\ a_{11} & a_{12}\end{bmatrix}$$

$$\text{or, more importantly } \overline{\boldsymbol{W}_2} = \begin{bmatrix}\overline{l_{21}} \\ \overline{l_{22}}\end{bmatrix} \times \begin{bmatrix}a_{11} & a_{12}\end{bmatrix} = \overline{\boldsymbol{l_2}} \times \boldsymbol{a_1}^T \tag{17}$$

$$\overline{\boldsymbol{a}_1} = \begin{bmatrix}\overline{l_{21}}W_{211} + \overline{l_{22}}W_{221} \\ \overline{l_{21}}W_{212} + \overline{l_{22}}W_{222}\end{bmatrix}$$

$$\overline{\boldsymbol{a}_1} = \begin{bmatrix}W_{211} & W_{221} \\ W_{212} & W_{222}\end{bmatrix} \times \begin{bmatrix}\overline{l_{21}} \\ \overline{l_{22}}\end{bmatrix}$$

$$\overline{\boldsymbol{a}_1} = \boldsymbol{W_2}^T \times \overline{\boldsymbol{l_2}} \tag{18}$$

$$\overline{\boldsymbol{l}_1} = \begin{bmatrix}\overline{a_{11}}\theta(l_{11}) \\ \overline{a_{12}}\theta(l_{12})\end{bmatrix}$$

$$\overline{\boldsymbol{l}_1} = \overline{\boldsymbol{a}_1} \circ \theta(\boldsymbol{l}_1) \tag{19}$$

$$\overline{\boldsymbol{W}_1} = \begin{bmatrix}\overline{l_{11}}x_{11} & \overline{l_{11}}x_{12} \\ \overline{l_{12}}x_{11} & \overline{l_{12}}x_{12}\end{bmatrix}$$

$$\overline{\boldsymbol{W}_1} = \begin{bmatrix}\overline{l_{11}} \\ \overline{l_{12}}\end{bmatrix} \times \begin{bmatrix}x_{11} & x_{12}\end{bmatrix}$$

$$\overline{\boldsymbol{W}_1} = \overline{\boldsymbol{l}_1} \times \boldsymbol{x}_1^T \tag{20}$$

## 1.4 Misc. Remarks

### 1.4.1 Rounding: Training vs Inference

Since we aim to train a binary classifier, the round() would be necessary for the correct output range. However since round() is not differentiable, we omit it during training, calculating fractional losses instead. We only include round() during inference.

### 1.4.2 2023-10-18 Remaining points of confusion

How does one go about, concretely, on an element-by-element level, determining say the matrix equivalent of derivative of a function applied to a matrix? Is the derivative applied element-wise to the existing matrix, yielding a matrix of the same dimension as the original? Then we need to have a clean notation for that without getting confusing/ambiguous. I find some of the notational conventions here unclear and confusing. The current approach is vague and imprecise, which I find bothersome.

### 1.4.3 2023-10-19 Remaining questions

- What is the meaning of $\circ$ in the context of these vectorized equations? What is the difference between $\circ$ and $\cdot$? Answer: According to ChatGPT, it is the composition of the linear transformations represented by the matrices. Apparently ChatGPT can also be coaxed into thinking it's the same as matrix multiplication, i.e., $A \circ B = AB$. OK, so if one looks at `https://math.libretexts.org/Bookshelves/Linear_Algebra/Interactive_Linear_Algebra_(Margalit_and_Rabinoff)/03%3A_Linear_Transformations_and_Matrix_Algebra/3.04%3A_Matrix_Multiplication#:~:text=As%20we%20will%20see%2C%20composition,of%20transformations%20and%20of%20matrices`. It does seem like it really IS matrix multiplication, but then WHY bother to use this symbol? I am somewhat confused but am still feeling reassured that I am justified in assuming it's equiv. to matrix multiplication.

- Why are some of the matrices in these vectorized backprops transposed? Examples from Roger Grosse:

- $$\overline{\boldsymbol{W}^{(2)}} = \overline{\boldsymbol{y}}\boldsymbol{h}^T$$

- This would be so much easier if they stated the dimensions of the various matrices/vectors in the equations

- Other questions: What does it mean to take for example: $\frac{\partial}{\partial \boldsymbol{w_3}}(\boldsymbol{w}_3 \cdot \boldsymbol{a}_2)$? Answer: There are two observations. One, that in general, for $\boldsymbol{x} \in \mathbb{R}^n, f(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}$, we have that $\frac{\partial f}{\partial \boldsymbol{x}} = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ...]$ Also, recall that in our MLP: $\boldsymbol{w}_3 = [w_{311}, w_{312}]$ and $\boldsymbol{a}_2 = [a_{21}, a_{22}]$ so $\boldsymbol{w}_3 \cdot \boldsymbol{a}_2 = w_{311}a_{21} +$

$w_{312}a_{22}$, so $\frac{\partial \boldsymbol{w}_3 \cdot \boldsymbol{a}_2}{\partial \boldsymbol{w}_3} = [\frac{\partial}{\partial w_{311}}(w_{311}a_{21} + w_{312}a_{22}), \frac{\partial}{\partial w_{312}}(w_{311}a_{21} + w_{312}a_{22})]$ so $\frac{\partial \boldsymbol{w}_3 \cdot \boldsymbol{a}_2}{\partial \boldsymbol{w}_3} = [a_{21} + \frac{\partial}{\partial w_{311}}(w_{312}a_{22}), \frac{\partial}{\partial w_{312}}(w_{311}a_{21}) + a_{22}]$ so $\frac{\partial \boldsymbol{w}_3 \cdot \boldsymbol{a}_2}{\partial \boldsymbol{w}_3} = [a_{21} + 0, 0 + a_{22}]$ so $\frac{\partial \boldsymbol{w}_3 \cdot \boldsymbol{a}_2}{\partial \boldsymbol{w}_3} = [a_{21}, a_{22}]$ so $\frac{\partial \boldsymbol{w}_3 \cdot \boldsymbol{a}_2}{\partial \boldsymbol{w}_3} = \boldsymbol{a}_2$

### 1.4.4 2023-10-22 Remaining questions

- what is the derivative of an $n \times m$ vector/matrix with respect to another matrix of the same dimension? ChatGPT: in general, the derivative of one vector w.r.t. another is called the Jacobian. The generalized answer for derivative of a matrix w.r.t another is still unanswered.

### 1.4.5 2023-10-24 Remaining questions

- the 'type' of the LHS in these backprop calculations is not clearly stated. Quite irritating.
- is the vectorized solution really even necessary? It's quite confusing and unclear to work with algebraically. Example: what is the type of $\overline{\boldsymbol{l}_2}$? It seems like it's the product of two identically-sized (2,1) vectors? How is that even defined? I am confused. The per-element approach seems easier to program into code.
- I think the vectorized approach might just be overkill. I will read `https://brilliant.org/wiki/backpropagation/` and think more about this.

### 1.4.6 2023-10-25 Notes

- I am trying to think through the implementation of backpropagation and it seems like the main advantage of this approach is to minimize redundant calculations.
  The approach will have to involve somehow storing the value of the derivative of each 'node'
  (in the sense of vertex in the full computation graph, where each vertex represents a single scalar)
  in order to reuse that value. I could implement this per-element but I am nearly certain it would be way more computationally expensive than a vectorized approach.
- I am looking at the per-element equations and it seems that based off of them, since the derivative w.r.t. loss (overline) calculations are per-element, that they are at-most the same dimension as the original vectors/matrices, (e.g., $\overline{\boldsymbol{W}_{ijk}}$ has the same dimension as $\boldsymbol{W}_{ijk}$), only being further 'compressible' IFF they have algebraically equivalent values.
- the other issue with backprop is my confusion about the relationship between the algebraic expressions derived here and exact update values applied to the weights.

- So, I think a better approach to obtain the vectorized definitions for backprop is to 'derive' them by finding vector/matrix expressions that are 'equivalent' to the per-element derivation, with the added constraint that the 'overline' vector/matrix has to have the SAME dimension as the original. ANSWER TO THE QUESTION: We first analytically determine the derivative of the error function w.r.t. to all weights in the network using backprop/chain rule. THEN, we evaluate the error function at a given 'point' (namely, the error function is a function of the expected output and generated outputs), and apply and from there backtrack to get the 'derivative WRT the weight' applied at that specific training point.
- The naive application of the chain-rule for backprop might not apply cleanly in the vectorized approach, I think, because the derivatives WRT loss of prev layers can be non-scalar.
- Tomorrow I will go through and 'matirixize/vectorize' the per-element derivations by inspection.
- I think one possibly MAJOR mistake that I have made is to have treated $w_{11i}$ and $w_{12i}$ as part of the same matrix? That might have caused issues in the backprop calculation

#### 1.4.7 2023-10-28 Observations

- So, in order to computate the full backprop, I not only need to store extra numpy arrays for the updates themselves (to accelerate the update operation), but I need to also store every activation (and post-activation) as they're needed for computing the values of the updates.
- I modified the code to use a tuple of two numpy ndarrays (one for the weights, the other for the updates) and I suspect I will need to add yet another two for activations and post-activation.
- Then, I will be storing the complete 'state' of the network.

#### 1.4.8 2023-10-31 Remarks

- Got confused about the vectorized derivation of the loss w.r.t $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. The general approach: use the per-element equations and the constraint that deriv of Loss w.r.t to X must have X's same dimensions Then, use either knowledge of hadamard product, transpose and (recall that matrix product of a column vector by row vector is a matrix) to derive (by inspection) the vectorized forms of the derivatives. This will allow straight forward coding with numpy.