Basic MLP with manually-derived Backprop

Teo Asinari

October 10, 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

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

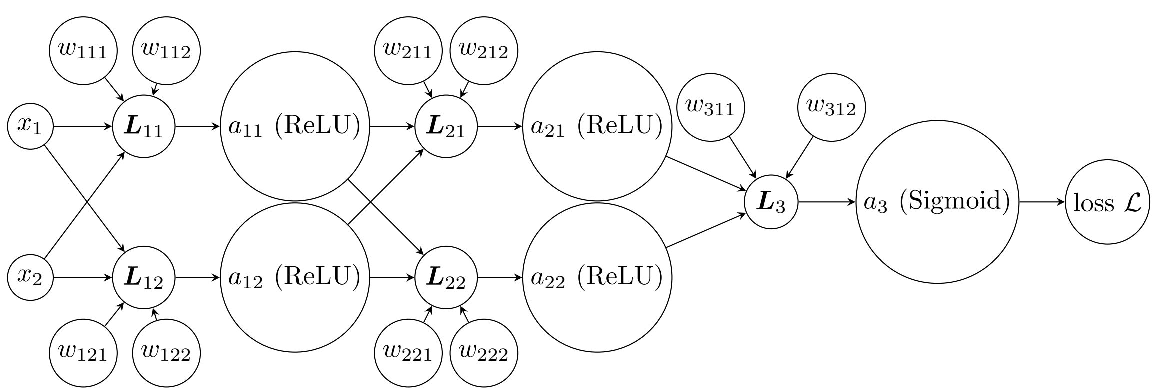
: [(2, ReLU), (2, ReLU), (1, Sigmoid)]

1.1 Diagrams

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

$$\begin{tikzcd}
\boldsymbol{w_1} \arrow[dr] & & \boldsymbol{w_2} \arrow[dr] & & \boldsymbol{w_3} \arrow[dr] & & \boldsymbol{t} \arrow[dr] & \\
\boldsymbol{x} \arrow[r] & \boldsymbol{L_1} \arrow[r] & \boldsymbol{a_1} \arrow[r] & \boldsymbol{L_2} \arrow[r] & \boldsymbol{a_2} \arrow[r] & \boldsymbol{L_3} \arrow[r] & \boldsymbol{a_3} \arrow[r] & \mathcal{L}
\end{tikzcd}$$

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



1.2 Definitions

1.2.1 Remark on 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.

1.2.2 Remark on layer notation

This is a sub-case of the weight notation. I.e., $\boldsymbol{L}_{ij}$ is the j-th neuron at the i-th layer, etc.

1.2.3 Neuron firing calculation

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

$$\boldsymbol{L}_{ij} = \boldsymbol{w}_{ij}\boldsymbol{x}_i$$

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:

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

1.3.1 Applying the backprop rule

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

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

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

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

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

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

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

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

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

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

$$\overline{\boldsymbol{L}_3} = \overline{\boldsymbol{a_3}}\frac{\partial \boldsymbol{a_3}}{\partial \boldsymbol{L}_3}$$

$$\overline{\boldsymbol{L}_3} = \overline{\boldsymbol{a_3}}\frac{\partial}{\partial \boldsymbol{L}_3}\sigma(\boldsymbol{L}_3)$$

$$\overline{\boldsymbol{L}_3} = \overline{\boldsymbol{a_3}}\ \sigma(\boldsymbol{L}_3)(1 - \sigma(\boldsymbol{L}_3)) \tag{4}$$

Final layer weights

$$\overline{w_{3i}} = \overline{\boldsymbol{L}_3}\frac{\partial}{\partial w_{3i}}\boldsymbol{L}_3$$

$$\overline{w_{3i}} = \overline{\boldsymbol{L}_3}\frac{\partial}{\partial w_{3i}}\sum_j w_{3j}a_{2j}$$

$$\overline{w_{3i}} = \overline{\boldsymbol{L}_3}a_{2i} \tag{5}$$

Second layer activation

Second layer

Second layer weights

Input layer activation

Input layer

Input layer weights

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.