# Binarized Neural Network

## Knowledge and Data Mining project

Irene Caria

Matricola: 2040639

Pietro Sittoni

Matricola: 2040637

February 22, 2023

# Table of contents

# Table of contents

# Table of contents

# Table of contents

# Table of contents

# Table of contents

Binarized neural networks are nets in which both the weights and activations are binary.

- The unknown model parameter $w_i \in \{-1, 1\}$
- The input variables are $x^{(i)}$ for $i = 1, ..., m$, where each $x^{(i)} = (x_1^{(i)}, ..., x_n^{(i)}) \in \{-1, 1\}^n$
- The labels $y^{(i)} = f(x^{(i)})$ for $i = 1, ..., m$, where $f : \{-1, 1\}^n \longrightarrow \{-1, 1\}$
- The activation function is $sign\left(\sum_{i=1}^{n} x_i w_i\right)$

Binarized neural networks are nets in which both the weights and activations are binary.

- The unknown model parameter $w_i \in \{-1, 1\}$
- The input variables are $x^{(i)}$ for $i = 1, ..., m$, where each $x^{(i)} = (x_1^{(i)}, ..., x_n^{(i)}) \in \{-1, 1\}^n$
- The labels $y^{(i)} = f(x^{(i)})$ for $i = 1, ..., m$, where $f : \{-1, 1\}^n \longrightarrow \{-1, 1\}$
- The activation function is $sign\left(\sum_{i=1}^{n} x_i w_i\right)$

Binarized neural networks are nets in which both the weights and activations are binary.

- The unknown model parameter $w_i \in \{-1, 1\}$
- The input variables are $x^{(i)}$ for $i = 1, ..., m$, where each $x^{(i)} = (x_1^{(i)}, ..., x_n^{(i)}) \in \{-1, 1\}^n$
- The labels $y^{(i)} = f(x^{(i)})$ for $i = 1, ..., m$, where $f : \{-1, 1\}^n \longrightarrow \{-1, 1\}$
- The activation function is $sign\left(\sum_{i=1}^{n} x_i w_i\right)$

Binarized neural networks are nets in which both the weights and activations are binary.

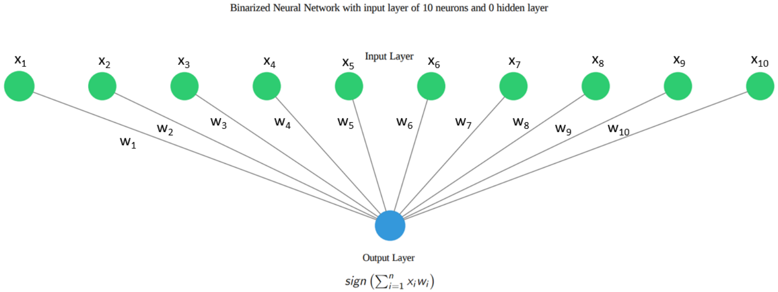
- The unknown model parameter $w_i \in \{-1, 1\}$
- The input variables are $x^{(i)}$ for $i = 1, ..., m$, where each $x^{(i)} = (x_1^{(i)}, ..., x_n^{(i)}) \in \{-1, 1\}^n$
- The labels $y^{(i)} = f(x^{(i)})$ for $i = 1, ..., m$, where $f : \{-1, 1\}^n \longrightarrow \{-1, 1\}$
- The activation function is *sign* $\left(\sum_{i=1}^{n} x_i w_i\right)$

Binarized neural networks are nets in which both the weights and activations are binary.

- The unknown model parameter $w_i \in \{-1, 1\}$
- The input variables are $x^{(i)}$ for $i = 1, ..., m$, where each $x^{(i)} = (x_1^{(i)}, ..., x_n^{(i)}) \in \{-1, 1\}^n$
- The labels $y^{(i)} = f(x^{(i)})$ for $i = 1, ..., m$, where $f : \{-1, 1\}^n \longrightarrow \{-1, 1\}$
- The activation function is $sign\left(\sum_{i=1}^{n} x_i w_i\right)$

We can transform this problem into a MAX-SAT.
We used Python in particular PySAT and RC2 algorithm in order to solve the MAX-SAT problem.

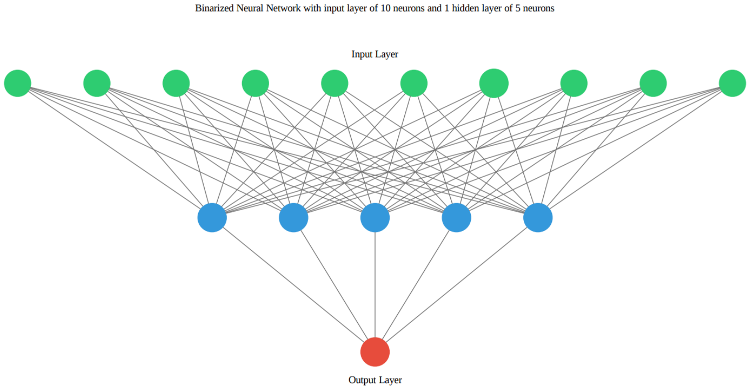
This is an example of Binarized Neural Network with the input layer of 10 neurons and 0 hidden layer.



# BNN - 1 hidden layer



Here an example of Binarized Neural Network with the input layer of 10 neurons and 1 hidden layer of 5 neurons.



We try to maximize

$$\sum_{i=1}^{m} y_i o_i,$$

where $y_i$ is the i-th target and $o_i$ is the i-th output of the model.

# BNN -Encoding- 1 layer



We use $n \times m$ propositional variables:

i $W_1, \dots, W_n$

ii $o_1, \dots, o_m$

If $x, y \in \{-1, 1\}$ the product is one when $x, y$ have the same sign minus one otherwise.
In propositional logic if we have the i-th entry of the k-th observation the product between $W_i$ and $x_i^k$ is equivalent to $\neg^{x_i^k} W_i = W_i$ when $x_i^k = 1$, $\neg^{x_i^k} W_i = \neg W_i$.

The main idea in order to encode $\text{sign}(\sum_i w_i x_i)$ is: the sign is positive if there are at least $\lceil \frac{n}{2} \rceil$ positive terms in the sum so

$$\bigwedge_{\substack{I \subseteq [n] \\ |I| = n - \lceil \frac{n}{2} \rceil + 1}} \bigvee_{i \in I} \neg^{x_i} W_i, \text{the negation (in CNF) is:}$$

$$\bigwedge_{\substack{I \subseteq [n] \\ |I| = \lceil \frac{n}{2} \rceil}} \bigvee_{i \in I} \neg\neg^{x_i} W_i.$$

The hard clause can easily obtain by

$$o_k \equiv \bigwedge_{\substack{I \subseteq [n] \\ |I| = n - \lceil \frac{n}{2} \rceil + 1}} \bigvee_{i \in I} \neg^{x_i^k} W_i,$$

$$\left( \bigwedge_{\substack{I \subseteq [n] \\ |I| = n - \lceil \frac{n}{2} \rceil + 1}} \bigvee_{i \in I} \neg^{x_i^k} W_i \vee \neg o_k \right) \bigwedge$$

$$\bigwedge \left( \bigwedge_{\substack{I \subseteq [n] \\ |I| = \lceil \frac{n}{2} \rceil}} \bigvee_{i \in I} \neg\neg^{x_i^k} W_i \vee o_k \right).$$

The soft Clause for each observation in the data set are:

$$(o_k, y_k).$$

We try to maximize the number of output that have the same sign of $y_k$.

The Clause are:

i $(\bigvee_{i \in I} \neg^{x_i^k} W_i \vee \neg o_k, \infty)$, for each $I \subseteq [n]$ such that $|I| = n - \lceil \frac{n}{2} \rceil + 1$

ii $(\bigvee_{i \in I'} \neg\neg^{x_i^k} W_i \vee \neg\neg o_k, \infty)$, for each $I^{'} \subseteq [n]$ such that $|I^{'}| = \lceil \frac{n}{2} \rceil$

iii $(o_k, y_k)$.

$k = 1, \ldots, m$

We use $(n + m + 1) \times h + m$ propositional variables:

i $W_1^1, \dots, W_n^1, W_1^2, \dots, W_n^h$

ii $\bar{w}_1, \dots, \bar{w}_h$

iii $H_1^1, \dots, H_h^1, H_1^2, \dots, H_h^m$

iv $o_1, \dots, o_m$

We encode the product with the element $(W_i^j)$ and the input with the same reasoning as before. But we need to encode the hidden literals with the output weight $(\overline{w}_j)$.

$$H_j^k \equiv \overline{w}_j$$

The hard clause can easily derive from

$$H_j^k \equiv \bigwedge_{\substack{I\subseteq[n]\\|I|=n-\lceil\frac{n}{2}\rceil+1}} \bigvee_{i\in I} \neg^{x_i^k} W_i^j,$$

$$\bigwedge_{\substack{I\subseteq[n]\\|I|=n-\lceil\frac{n}{2}\rceil+1}} \bigvee_{i\in I} \neg^{x_i^k} W_i^j \vee \neg H_j^k \bigwedge$$

$$\bigwedge_{\substack{I\subseteq[n]\\|I|=\lceil\frac{n}{2}\rceil}} \bigvee_{i\in I} \neg\neg^{x_i^k} W_i^j \vee H_j.$$

$$o_k \equiv \bigwedge_{\substack{I \subseteq [h] \\ |I| = n - \lceil \frac{h}{2} \rceil + 1}} \bigvee_{j \in J} (\overline{w}_j \equiv H_j^k),$$

$$\left( \bigwedge_{\substack{I \subseteq [h] \\ |I| = h - \lceil \frac{h}{2} \rceil + 1}} \bigvee_{j \in J} \left( (\overline{w}_j \vee \neg H_j^k \vee \neg o_k) \wedge (\neg \overline{w}_j \vee H_j^k \vee \neg o_k) \right) \right) \bigwedge$$

$$\bigwedge \left( \bigwedge_{\substack{I \subseteq [h] \\ |I| = \lceil \frac{h}{2} \rceil}} \bigvee_{j \in J} \left( (\neg \overline{w}_j \vee \neg H_j^k \vee o_k) \wedge (\overline{w}_j \vee H_j^k \vee o_k) \right) \right).$$

Then using the distributive property of the $\wedge$ and $\vee$ we obtain the remaining hard clause.

The soft clause are equal to the previous case $(o_k, y_k)$

Majority function

$$f(x) = \begin{cases} 1 & \textit{if sum}(x) \geq 0, \\ -1 & \textit{otherwise} \end{cases}$$

.

Majority function

$$f(x) = \begin{cases} 1 & \textit{if sum}(x) \geq 0, \\ -1 & \textit{otherwise} \end{cases}$$

XOR function

$$f(x) = \begin{cases} 1 & \textit{if } \#1 \textit{ in } x \textit{ is odd}, \\ -1 & \textit{otherwise} \end{cases}$$

.

Majority function

$$f(x) = \begin{cases} 1 & \textit{if } sum(x) \geq 0, \\ -1 & \textit{otherwise} \end{cases}$$

XOR function

$$f(x) = \begin{cases} 1 & \textit{if \#1 in } x \textit{ is odd}, \\ -1 & \textit{otherwise} \end{cases}$$

Parity function

$$f(x) = \begin{cases} 1 & \textit{if \#1 in } x \textit{ is even}, \\ -1 & \textit{otherwise} \end{cases}$$

.

Majority function

$$f(x) = \begin{cases} 1 & \text{if } sum(x) \geq 0, \\ -1 & \text{otherwise} \end{cases}$$

XOR function

$$f(x) = \begin{cases} 1 & \text{if } \#1 \text{ in } x \text{ is odd}, \\ -1 & \text{otherwise} \end{cases}$$

Parity function

$$f(x) = \begin{cases} 1 & \text{if } \#1 \text{ in } x \text{ is even}, \\ -1 & \text{otherwise} \end{cases}$$

Inner product function

$$f(x) = \begin{cases} 1 & \text{if } prod(x) = 1, \\ -1 & \text{otherwise} \end{cases}$$

.

Majority function

$$f(x) = \begin{cases} 1 & \textit{if } sum(x) \geq 0, \\ -1 & \textit{otherwise} \end{cases}$$

XOR function

$$f(x) = \begin{cases} 1 & \textit{if } \#1 \textit{ in } x \textit{ is odd}, \\ -1 & \textit{otherwise} \end{cases}$$

Parity function

$$f(x) = \begin{cases} 1 & \textit{if } \#1 \textit{ in } x \textit{ is even}, \\ -1 & \textit{otherwise} \end{cases}$$

Inner product function

$$f(x) = \begin{cases} 1 & \textit{if } prod(x) = 1, \\ -1 & \textit{otherwise} \end{cases}$$

We also implemented a *random* function which assigns randomly a value in $\{-1,1\}$.

# Dataset creation



- Set a random seed
- Given the dimensions $m \times n$, we randomly generate $m$ observations with $n$ values in $\{-1, 1\}$
- For each observation we calculate the corresponding label by applying one of the binary functions shown before (Majority, XOR, Parity, Inner product, Random)
- Split in train and test sets

# Dataset creation

- Set a random seed
- Given the dimensions $m \times n$, we randomly generate $m$ observations with $n$ values in $\{-1, 1\}$
- For each observation we calculate the corresponding label by applying one of the binary functions shown before (Majority, XOR, Parity, Inner product, Random)
- Split in train and test sets

# Dataset creation

- Set a random seed
- Given the dimensions $m \times n$, we randomly generate $m$ observations with $n$ values in $\{-1, 1\}$
- For each observation we calculate the corresponding label by applying one of the binary functions shown before (Majority, XOR, Parity, Inner product, Random)
- Split in train and test sets

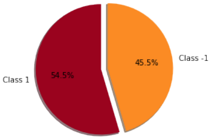
# Dataset creation

- Set a random seed
- Given the dimensions $m \times n$, we randomly generate $m$ observations with $n$ values in $\{-1, 1\}$
- For each observation we calculate the corresponding label by applying one of the binary functions shown before (Majority, XOR, Parity, Inner product, Random)
- Split in train and test sets

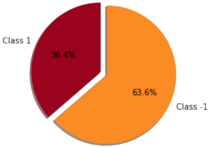
# Pie charts - balance or unbalanced data?



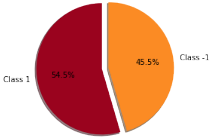
### Majority function



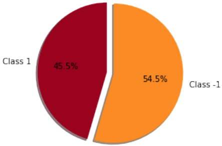

### Random function



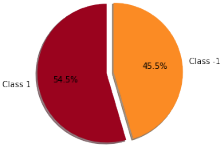

### XOR function




### Parity function




### Inner product function




Note: these pie charts refer to training labels of 22 observations (see next slide)

# BNN - Performance - 1-layer vs 2-layer ($n = 5$)



- The following table summarizes the train and test performances of both 1-layer and 2-layer BNN with dimension $n = 5$

| BNN type | Hidden dim | Binary f | Train dim | Test dim | Train acc | Test acc |
|---|---|---|---|---|---|---|
| $1-$layer | 0 | majority | $(22, 5)$ | $(10, 5)$ | 1.0 | 1.0 |
| $2-$layer | 5 | majority | $(22, 5)$ | $(10, 5)$ | 1.0 | 1.0 |
| $1-$layer | 0 | XOR | $(22, 5)$ | $(10, 5)$ | $0.\overline{81}$ | 0.8 |
| $2-$layer | 5 | XOR | $(22, 5)$ | $(10, 5)$ | 1.0 | 1.0 |
| $1-$layer | 0 | parity | $(22, 5)$ | $(10, 5)$ | $0.\overline{81}$ | 0.8 |
| $2-$layer | 5 | parity | $(22, 5)$ | $(10, 5)$ | 1.0 | 1.0 |
| $1-$layer | 0 | inner product | $(22, 5)$ | $(10, 5)$ | $0.\overline{81}$ | 0.8 |
| $2-$layer | 5 | inner product | $(22, 5)$ | $(10, 5)$ | 1.0 | 1.0 |
| $1-$layer | 0 | random | $(22, 5)$ | $(10, 5)$ | $0.\overline{72}$ | 0.6 |
| $2-$layer | 5 | random | $(22, 5)$ | $(10, 5)$ | $0.8\overline{63}$ | 0.6 |

# BNN - Performance - 1-layer ($n = \max$)



- BNN perfomance - only input layer with the maximum dimension $n$ of the dataset

| **Binary f** | **Train dim** | **Test dim** | **Train acc** | **Test acc** |
|---|---|---|---|---|
| majority | $(44, 20)$ | $(20, 20)$ | $1.0$ | $1.0$ |
| XOR | $(44, 20)$ | $(20, 20)$ | $0.88\overline{63}$ | $0.65$ |
| parity | $(44, 20)$ | $(20, 20)$ | $0.86\overline{36}$ | $0.45$ |
| inner product | $(44, 20)$ | $(20, 20)$ | $0.8\overline{63}$ | $0.45$ |
| random | $(44, 20)$ | $(20, 20)$ | $0.\overline{81}$ | $0.25$ |

# BNN - Performance - 2-layer ($n = \max$)



- BNN performance - with 1 hidden layer of 10 neurons and the maximum dimension $n$ of the dataset

| **Binary f** | **Train dim** | **Test dim** | **Train acc** | **Test acc** |
|---|---|---|---|---|
| majority | $(89, 15)$ | $(39, 15)$ | 1.0 | $0.\overline{743589}$ |
| XOR | $(89, 15)$ | $(39, 15)$ | 1.0 | $0.\overline{871794}$ |
| parity | $(89, 15)$ | $(39, 15)$ | 1.0 | $0.\overline{769230}$ |
| inner product | $(89, 15)$ | $(39, 15)$ | 1.0 | $0.\overline{871794}$ |
| random | $(89, 15)$ | $(39, 15)$ | 0.83 | 0.4 |

In the Python code, we will show the implementation of the BNN with 1 and 2 layer, using different binary functions and dataset dimensions.
https://colab.research.google.com/drive/1IhwUSa4mlCOJPS7gfna3hRF61_Jvc4Ns

# Thanks for the attention!