Binarized Neural Network

Alberto Sinigaglia

ID: 2044712

alberto.sinigaglia@studenti.unipd.it

Binarized neural network

Binarized neural network are composed by inputs, output, and weights $\in \{-1, 1\}$.

Their aim is, given a training set $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}$ of size $\boldsymbol{n}$, where $\boldsymbol{x}^{(i)} \in \{-1, 1\}^{m}$, to find the best combination of $\boldsymbol{w} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n\}$ to maximize the classification of the training set.

Requirements

For this project, the requirements are the following:

- the cardinality of the training set must be selectable
- the number of features of the training set must be selectable
- the problem must be encoded as a MaxSAT problem

Activation function

The activation function of the output neuron is the following:

$$out^{(i)} = sign(\sum_{j=0}^{m} w_j x_j^{(i)}) \in \{-1, 1\}$$

However, there is no concept of "sum" and "sign" in logic, therefore to be able to encode it in as a MaxSat problem, the activation function is been reinterpreted as follows:

$$\boldsymbol{o}^{(i)} = \begin{cases} 1 & \#\boldsymbol{w}_j\boldsymbol{x}_j^{(i)} > 0 \geq \#\boldsymbol{w}_j\boldsymbol{x}_j^{(i)} < 0 \\ -1 & \text{otherwise} \end{cases}$$

Encoding of a single connection

From the previous formulation, it's still not feasible to identify the concept of product in logic.
To overcome this problem, taking advantage of the input encoding ($\boldsymbol{x}_j^{(i)} \in \{-1, 1\}$), we can see that the single connection gets to 1 if the input and the weight have the same sign, and -1 in case they have opposite sign.

From what just said, we can build the following truth table

$\boldsymbol{w_i}$	$\boldsymbol{x_j^{(i)}}$	$\boldsymbol{w_i\ x_j^{(i)}}$
1	1	1
1	−1	−1
−1	1	−1
−1	−1	1

which corresponds to the XNOR truth table (or equivalently, the equivalence truth table)

Encoding of the # operator

For the new formulation of the activation, there is the need to find a way to count the number of $\boldsymbol{w_i x_j^{(i)}} > 0$, which with the new XNOR interpretation is $\boldsymbol{w_i} \odot \boldsymbol{x_j^{(i)}} = \boldsymbol{True}$.

To do so, it's been considered that if $\#\boldsymbol{w_j x_j^{(i)}} > 0 \geq \#\boldsymbol{w_j x_j^{(i)}} < 0$, given that each sample has $\boldsymbol{m}$ features, then $\boldsymbol{x_i}$ is composed by 2 subgroups, $\boldsymbol{a_i^T}$ (with all the $\boldsymbol{w_j} \odot \boldsymbol{x_j^{(i)}} = \boldsymbol{True}$) and $\boldsymbol{a_i^F}$ (with all the $\boldsymbol{w_j} \odot \boldsymbol{x_j^{(i)}} = \boldsymbol{False}$).

Then either $|\boldsymbol{a}_i^T| > \frac{m}{2}$ or $|\boldsymbol{a}_i^F| > \frac{m}{2}$ (supposing $\boldsymbol{m}$ odd, for even the $=$ is assigned to one of the 2 groups arbitrarily).

However, at this point if, for example $|\boldsymbol{a}_i^T| > \frac{m}{2}$, then $\exists\left((\boldsymbol{a}_i^T)_1, (\boldsymbol{a}_i^T)_2, \ldots, (\boldsymbol{a}_i^T)_{\frac{m}{2}}\right) \subseteq \boldsymbol{a}_i^T$, and therefore we just need to check if there is a combination of $\frac{m}{2}$ variables in $\boldsymbol{a_i}$ that combined gives 1, which means that for a $\boldsymbol{J} \subset \{\text{index of } \boldsymbol{x_i}\}$ of size $\frac{m}{2}$:

$$\left(\boldsymbol{w}_{J_1} \odot \boldsymbol{x}_{J_1}^{(i)}\right) \wedge \left(\boldsymbol{w}_{J_2} \odot \boldsymbol{x}_{J_2}^{(i)}\right) \wedge \ldots \wedge \left(\boldsymbol{w}_{J_{\frac{m}{2}}} \odot \boldsymbol{x}_{J_{\frac{m}{2}}}^{(i)}\right) = \boldsymbol{True}$$

Reformulation of the problem

Given what stated before, we can reformulate the activation function from:

$$\boldsymbol{o}^{(i)} = \boldsymbol{sign}\Big(\sum_{j=0}^{m} \boldsymbol{w}_j \boldsymbol{x}_j^{(i)}\Big)$$

to:

$$\boldsymbol{o}^{(i)} = \begin{cases} 1 & \exists \boldsymbol{J} \subset \text{index of features of } \boldsymbol{x}^{(i)} : \bigwedge_{i \in J} \boldsymbol{w}_j \odot \boldsymbol{x}_j^{(i)} \\ -1 & \text{otherwise} \end{cases}$$

Therefore, the only missing part to translate to propositional logic, is the existence of such J, but we can encode as *combination of all the possible* J *connected with* $\bigvee$

Final formulation

Considering everything said so far, the formulation of the activation function, given

$$|x^{(i)}| = m$$

$$C = \left\{ \binom{m}{m/2}_1, \binom{m}{m/2}_2, \dots, \binom{m}{m/2}_k \right\}$$

is the following:

$$o^{(t)} = \bigvee_{J \in C} \bigwedge_{i \in J} w_i \odot x_i^{(t)}$$

The last missing part required to be able to train the network, is having a way to encode that the target of a sample should be equal to the output of the network, which can be easily described as follows:

$$\boldsymbol{y}^{(t)} \equiv \boldsymbol{o}^{(t)}$$

Encoding as MaxSAT problem

Now that we have a way to encode our problem as a propositional logic statement, the MaxSAT problem can be easily phrased as follow:

- as hard constrain, forcing $\boldsymbol{x}_j^{(i)}$ to be *True* or *False* depending what is inside the dataset, and same for $\boldsymbol{y}^{(i)}$
- as soft constrain having $\boldsymbol{y}^{(i)} \equiv \boldsymbol{o}^{(i)}$ with an arbitrary weight > 0

Translation to CNF

A crucial part is translating the soft constrain to CNF. However, in order to have simpler equations, the samples of the training set with target 1 will have different equation respect to those with 0 as target.

Translation to CNF for target = -1

Given that the CNF requires a conjunction of disjunction, the target function encoding given before it's the opposite of what we should have.
However, if *exist* $\boldsymbol{m}/2$ *pairs s.t.* $\boldsymbol{x_i^{(t)} w_i} = \boldsymbol{False}$, then we can translate it also as *does not exist* $\boldsymbol{m}/2$ *pairs s.t.* $\boldsymbol{x_i^{(t)} w_i} = \boldsymbol{True}$, and by doing so, the soft constrains can be transformed in a simpler form.

$$\neg\Big(\bigvee_{J\in C}\bigwedge_{i\in J} w_i \odot x_i^{(t)}\Big) = \bigwedge_{J\in C}\neg\Big(\bigwedge_{i\in J} w_i \odot x_i^{(t)}\Big)$$

$$= \bigwedge_{J\in C}\bigvee_{i\in J}\neg\big(w_i \odot x_i^{(t)}\big) = \bigwedge_{J\in C}\bigvee_{i\in J}\neg\neg\Big[\big(w_i \vee x_i^{(t)}\big) \wedge \big(\neg w_i \vee \neg x_i^{(t)}\big)\Big]$$

$$= \bigwedge_{J\in C}\bigvee_{i\in J}\big(w_i \wedge \neg x_i^{(t)}\big) \vee \big(\neg w_i \wedge x_i^{(t)}\big)$$

Expanded, becomes the following:

$$[(\neg x_1 \land w_1) \lor (x_1 \land \neg w_1)] \land [(\neg x_2 \land w_2) \lor (x_2 \land \neg w_2)] \land [(\neg x_3 \land w_3) \lor (x_3 \land \neg w_3)]\ldots$$

if we take each [...] and transform it to CNF becomes:

$$(x_i \lor \neg x_i) \land (x_i \lor w_i) \land (\neg x_i \lor \neg w_i) \land (w_i \lor \neg w_i)$$
$$\equiv$$
$$(\neg x_i \lor \neg w_i) \land (x_i \lor w_i)$$

therefore becomes:

$$\bigwedge_{J \in C} \bigvee_{i \in J} (\neg x_i^{(t)} \lor \neg w_i) \land (x_i^{(t)} \lor w_i)$$

Since each element of the big-OR is in CNF, we need to fix that big-OR, and the solution is having the cartesian product of all the terms of each element of the OR, that becomes as follows:

$$\begin{gathered}
\left(x_1 \lor x_2 \lor x_3 \lor w_1 \lor w_2 \lor w_3\right) \land \\
\left(x_1 \lor x_2 \lor \neg x_3 \lor w_1 \lor w_2 \lor \neg w_3\right) \land \\
\left(x_1 \lor \neg x_2 \lor x_3 \lor w_1 \lor \neg w_2 \lor w_3\right) \land \\
\left(x_1 \lor \neg x_2 \lor \neg x_3 \lor w_1 \lor \neg w_2 \lor \neg w_3\right) \land \\
\left(\neg x_1 \lor x_2 \lor x_3 \lor \neg w_1 \lor w_2 \lor w_3\right) \land \\
\ldots
\end{gathered}$$

In other words, we loop over all the possible combinations of the x_i, put them in $\lor$, together with the corresponding weight, with the same "sign" in front.

To recap, given a single instance from the training set with $\boldsymbol{m}$ features and $\boldsymbol{y_i}$ = False:

1. we go over all the possible subsets of size $\boldsymbol{m}/2$,
2. we go over all the possible integer from 0 to $\boldsymbol{m}/2$, convert to binary, and considering the i-th binary value as the i-th variable of the subset, consider $\boldsymbol{x_i}$ if that digit is 1, else $\neg\boldsymbol{x_i}$, and the same for the weight

Translation to CNF for target = 1

In this case, we have the same from when transforming the activation function to CNF, however, as before, we can rephrase it in a way that allows us to get a simpler transformation.

In fact, saying that *exists a subset of* $\boldsymbol{x_i}$ *of which* $\boldsymbol{x_i w_i} = \boldsymbol{True}$ is the same as saying that *does not exists a subset of* $\boldsymbol{x_i}$ *of which* $\boldsymbol{x_i w_i} = \boldsymbol{False}$, which in our case is the following:

$$\bigvee_{J\in C}\bigwedge_{i\in J} w_i \odot x_i^{(t)} = \neg \bigvee_{J\in C}\bigwedge_{i\in J} \neg\left(w_i \odot x_i^{(t)}\right)$$

However now, bringing the negation inside, we swap the AND and OR operator:

$$\neg \bigvee_{J \in C} \bigwedge_{i \in J} \neg\left(w_i \odot x_i^{(t)}\right) = \bigwedge_{J \in C} \bigvee_{i \in J} \neg\neg\left(w_i \odot x_i^{(t)}\right) = \bigwedge_{J \in C} \bigvee_{i \in J} \left(w_i \odot x_i^{(t)}\right)$$

$$= \bigwedge_{J \in C} \bigvee_{i \in J} \neg\left[\left(\neg w_i \wedge x_i^{(t)}\right) \vee \left(w_i \wedge \neg x_i^{(t)}\right)\right] = \bigwedge_{J \in C} \bigvee_{i \in J} \left(\neg w_i \vee x_i^{(t)}\right) \wedge \left(w_i \vee \neg x_i^{(t)}\right)$$

Since each element of the big-OR is in CNF, we need to fix that big-OR, and the solution is having the cartesian product of all the terms of each element of the OR, that becomes as follows:

$$\begin{gathered}
\left(\boldsymbol{x}_1 \lor \boldsymbol{x}_2 \lor \boldsymbol{x}_3 \lor \neg\boldsymbol{w}_1 \lor \neg\boldsymbol{w}_2 \lor \neg\boldsymbol{w}_3\right) \land \\
\left(\boldsymbol{x}_1 \lor \boldsymbol{x}_2 \lor \neg\boldsymbol{x}_3 \lor \neg\boldsymbol{w}_1 \lor \neg\boldsymbol{w}_2 \lor \boldsymbol{w}_3\right) \land \\
\left(\boldsymbol{x}_1 \lor \neg\boldsymbol{x}_2 \lor \boldsymbol{x}_3 \lor \neg\boldsymbol{w}_1 \lor \boldsymbol{w}_2 \lor \neg\boldsymbol{w}_3\right) \land \\
\left(\boldsymbol{x}_1 \lor \neg\boldsymbol{x}_2 \lor \neg\boldsymbol{x}_3 \lor \neg\boldsymbol{w}_1 \lor \boldsymbol{w}_2 \lor \boldsymbol{w}_3\right) \land \\
\left(\neg\boldsymbol{x}_1 \lor \boldsymbol{x}_2 \lor \boldsymbol{x}_3 \lor \boldsymbol{w}_1 \lor \neg\boldsymbol{w}_2 \lor \neg\boldsymbol{w}_3\right) \land \\
\ldots
\end{gathered}$$

In other words, we loop over all the possible combinations of the $\boldsymbol{x}_i$, put them in $\lor$, together with the corresponding weight with the opposite "sign" in front.

To recap, given a single instance from the training set with $\boldsymbol{m}$ features and $\boldsymbol{y_i}$ = True:

1. we go over all the possible subsets of size $\boldsymbol{m}/2$,
2. we go over all the possible integer from 0 to $\boldsymbol{m}/2$, convert to binary, and considering the i-th binary digit as the i-th variable of the subset, consider $\boldsymbol{x_i}$ if that digit is 1, else $\neg\boldsymbol{x_i}$, and same for the weights, with the opposite "sign" as the corresponding input

Final composition of the problem

Since now we have a CNF formula that is $\boldsymbol{True}$ in case an $\boldsymbol{x}^{(t)}$ has more than $\boldsymbol{m}/2$ values of $\boldsymbol{w}_i\boldsymbol{x}_i^{(t)} = \boldsymbol{True}$, and another that is $\boldsymbol{False}$ in case an $\boldsymbol{x}^{(t)}$ has more than $\boldsymbol{m}/2$ values of $\boldsymbol{w}_i\boldsymbol{x}_i^{(t)} = \boldsymbol{False}$, the binarized neural network can be encoded as MaxSAT problem using the following procedure.

1. as hard constraint, force the $\boldsymbol{x_i^{(t)}}$ to be $\boldsymbol{True/False}$ as they are in the dataset
2. as soft constraint, for each sample $\boldsymbol{j}$ of the dataset, with an arbitrary weight associated:
   - if $\boldsymbol{y^{(t)}} = 1$ then add the formula for the case where there are at least $\boldsymbol{m}/2$ values of $\boldsymbol{w_i x_i^{(t)}} = \boldsymbol{True}$
   - else then add the formula for the case where there are at least $\boldsymbol{m}/2$ values of $\boldsymbol{w_i x_i^{(t)}} = \boldsymbol{False}$

Evaluation

For the final evaluation, the following dataset is been generated:

- 7 features
- full cartesian product of all the possible combinations of the 7 features
- 70% training data and 30% test data
- $(\boldsymbol{x}_0 \lor \boldsymbol{x}_1 \lor \boldsymbol{x}_2) \land (\boldsymbol{x}_3 \lor \neg \boldsymbol{x}_4) \land (\neg \boldsymbol{x}_5 \lor \boldsymbol{x}_6)$ function to predict/approximate

The final accuracies of the model are:

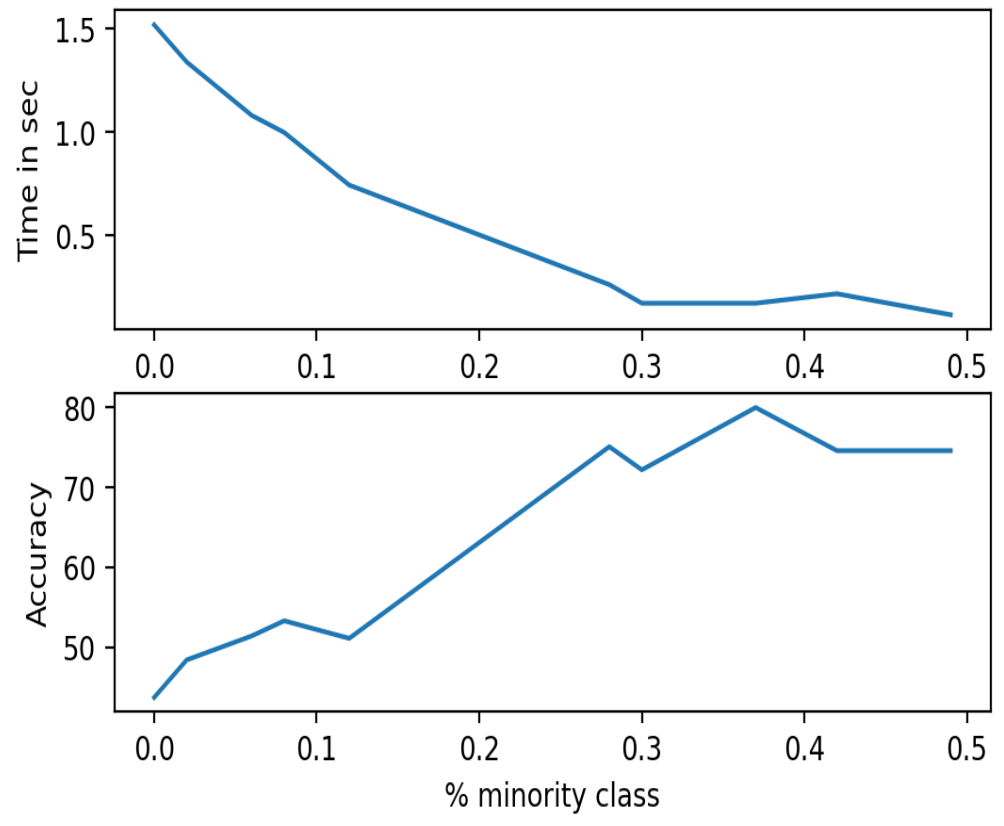
- 79.47% Training accuracy
- 81.95% Test accuracy

However, potential real dataset might have entries that are duplicates, therefore the model will be pushed to satisfy them, since with a single correct classification, would gain 2x the weight, therefore the performance might be even better than the tested one.

Analysis

The implementation is being done using `PySAT`, using `WCFN` in order to create the set of formulas with the soft and hard constraint, and using `RC2` (*relaxable cardinality constraints*) to find a solution to the MaxSAT problem.
As reported in the documentation, the algorithm takes advantage of several optimization techniques which aim to reduce the average time complexity, but not the worst case complexity.

As can be seen in the following graph, given 7 features, the ratio between 1 and -1 in the targets plays a huge role in the time complexity, if using the Fu-Malik algorithm (the same is not observed using RC2)



% of minority	time	acc
49.22%	1.79s	80.5%
45.31%	2.31s	77.1%
35.16%	2.16s	78.5%
30.47%	3.70s	74.2%
27.34%	4.20s	68.0%
22.66%	6.36s	67.9%
16.41%	5.06s	66.7%
09.38%	9.45s	59.4%

*Data acquired using 7 features, using a generating function with 6/7 binary operators

Introducing biases

From the previous slide it's been shown that the model has very poor performance for very skewed distribution. This probem appears also in the normal neural networks, if we are not using biases. Infact, if we have a classification problem (in case of logistic regression), that has always 1 as target, for the model the best solution will be not to rely on the data, but on the bias term, therefore putting the data-weights to 0 and the bias towords +infinity.

Using this fact, what we are missing is a "shortcut for our model" to do this trick, and in order to fix this problem, there is only the need of inserting enough columns such that they are enough to completely determine the classification (just like in normal models how the bias is implemented adding an additional column set to 1).

In our case, the classification is based if there exists at least half $\boldsymbol{x}_i \equiv \boldsymbol{w}_i$, and therefore we need to add a numer additional columns equal to the number of columns in the original dataset.

However, introducing a number of columns equal to the number of columns in the original dataset adds a non-ignorable amount of complexity to the problem.

Results

Using as reference the code provided which has 7 features in the dataset, even though the number of columns just doubles, the size of each soft clause passes from 560 conjuction of 8 disjunctions, to 768768 conjunction of 16 disjunctions.

This makes the computation almost impossibile, as already just the creation of the soft clauses takes 10 minutes; however, an empirical good approximation is obtained adding a number of columns equal to half the number of column in the original dataset, which creates softclauses with "just" 13440 conjunction.

Following are some test results obtained with this approximation:

- having the minority class as 9.1% of targets passes from 53% to 81%
- having the minority class as 1.2% of targets passes from 56% to 82%
- having the minority class as 0.7% of targets passes from 45% to 80%

By adding a number number of columns equal to 3/4 the number of original columns, the model improves up to 95% accuracy having the minority class being just 0.7% of the targets

Extra

During the development of this project, it's been founded a problem in the library, which is that the soft clauses can only be inserted as conjunction of literal, where instead the project required to have a weight associated with a whole CNF (if the first N $\boldsymbol{x_i w_i}$ are the same, or if the second N are the same …)

The project using the library as is, can only give an approximation since a weight would be only associated to one combination, therefore it would not be fair since a single sample on the training set might count more than one (for example, if we have 3 features, and an element of the training set is 111 and same for the weights, than there would have been a soft clause with weight 1 if $\boldsymbol{x}_1 \equiv \boldsymbol{w}_1 \wedge \boldsymbol{x}_2 \equiv \boldsymbol{w}_2$, another one if $\boldsymbol{x}_1 \equiv \boldsymbol{w}_1 \wedge \boldsymbol{x}_3 \equiv \boldsymbol{w}_3$, and another one if $\boldsymbol{x}_2 \equiv \boldsymbol{w}_2 \wedge \boldsymbol{x}_3 \equiv \boldsymbol{w}_3$, leading to 3 weights if the model satisfies this sample, which is unfair)

Solution

In order to overcome this problem, it's been used the Gentzen system GCNF, since using that we can translate a CNF soft clause to a combination of a new soft clause with a new variable, and some new hard clauses which bounds this new variable to the disjunctions of the corresponding soft clause.

This would allow to transform a WCNF model with CNF soft clause, to the original representation of a WCNF, therefore not requiring any changes to the solvers.

I've personally coded that algorithm and proposed to the owner of the PySAT library here, and it's working correctly.

The end

thank you