Simulated annealing to optimize binary weights in multilayer feedforward NN

20602 - COMPUTER SCIENCE (ALGORITHMS)

Giosuè Migliorini

October 20, 2021

DSBA - Bocconi University

Outline

Neural Network structure

Multilayer Perceptron for K-class Classification[1]

- Start with an **input matrix** X of dimensions $N \times p$ and an **output vector** $y = (y_1, \ldots, y_N)$ where $y_i \in \{1, \ldots, K\}$.

[1] Hastie et al. (2009)

Multilayer Perceptron for K-class Classification[1]

- Start with an **input matrix** X of dimensions $N \times p$ and an **output vector** $y = (y_1, \dots, y_N)$ where $y_i \in \{1, \dots, K\}$.
- Pass the input X through H layers of matrix multiplication (with **weight matrices**), applying an **activation function** $\sigma(\cdot)$ element-wise. The matrices $Z^{(h)},\ h = 1, \dots, H$ computed at each pass are the **hidden layers**.

[1] Hastie et al. (2009)

Multilayer Perceptron for K-class Classification[1]

- Start with an **input matrix** X of dimensions $N \times p$ and an **output vector** $y = (y_1, \ldots, y_N)$ where $y_i \in \{1, \ldots, K\}$.
- Pass the input X through H layers of matrix multiplication (with **weight matrices**), applying an **activation function** $\sigma(\cdot)$ element-wise. The matrices $Z^{(h)},\ h = 1, \ldots, H$ computed at each pass are the **hidden layers**.
- Obtain a final layer by computing $T = Z^{(H)} \cdot \beta$, where β is a weight matrix of dimensions $M \times K$, and pass it through the **softmax function** to obtain class probabilities for each class k:

[1] Hastie et al. (2009)

Multilayer Perceptron for K-class Classification[1]

- Start with an **input matrix** X of dimensions $N \times p$ and an **output vector** $y = (y_1, \ldots, y_N)$ where $y_i \in \{1, \ldots, K\}$.
- Pass the input X through H layers of matrix multiplication (with **weight matrices**), applying an **activation function** $\sigma(\cdot)$ element-wise. The matrices $Z^{(h)},\ h = 1, \ldots, H$ computed at each pass are the **hidden layers**.
- Obtain a final layer by computing $T = Z^{(H)} \cdot \beta$, where β is a weight matrix of dimensions $M \times K$, and pass it through the **softmax function** to obtain class probabilities for each class k:

$$g(T_k) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

[1] Hastie et al. (2009)

Multilayer Perceptron for K-class Classification[1]

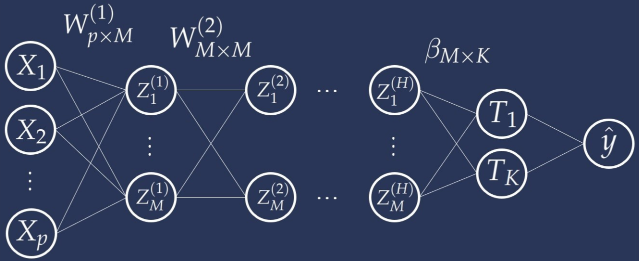
- Start with an **input matrix** X of dimensions $N \times p$ and an **output vector** $y = (y_1, \ldots, y_N)$ where $y_i \in \{1, \ldots, K\}$.
- Pass the input X through H layers of matrix multiplication (with **weight matrices**), applying an **activation function** $\sigma(\cdot)$ element-wise. The matrices $Z^{(h)},\ h = 1, \ldots, H$ computed at each pass are the **hidden layers**.
- Obtain a final layer by computing $T = Z^{(H)} \cdot \beta$, where β is a weight matrix of dimensions $M \times K$, and pass it through the **softmax function** to obtain class probabilities for each class k:

$$g(T_k) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

- Predict by taking each $\hat{y}_i$ equal to the class k with highest probability

[1] Hastie et al. (2009)

NN Architecture Cartoon



$$Z^{(1)} = \sigma(X\cdot W^{(1)}) \qquad Z^{(h)} = \sigma(Z^{(h-1)}\cdot W^{(h)}), \quad h = 2,\ldots,H$$

$$T = Z^{(H)}\cdot\beta \qquad \hat{y} = \arg\max_k\,(T_1,\ldots,T_K)$$

$$Z^{(h)} = \begin{bmatrix} z^{(h)}_{1,1} & \ldots & z^{(h)}_{1,M} \\ \vdots & \ddots & \\ z^{(h)}_{N,1} & & z^{(h)}_{N,M} \end{bmatrix} \qquad W^{(h)} = \begin{bmatrix} \omega^{(h)}_{1,1} & \ldots & \omega^{(h)}_{1,M} \\ \vdots & \ddots & \\ \omega^{(h)}_{M,1} & & \omega^{(h)}_{M,M} \end{bmatrix}$$

BINARY NEURAL NETWORKS[2]

The **Binary Neural Network** (BNN) has architecture identical to a multilayer perceptron, and is characterized by:

- Weights taking values in $\{$-1, 1$\}$, final layer included

[2]Courbariaux et al. (2016)

Binary Neural Networks[2]

The **Binary Neural Network** (BNN) has architecture identical to a multilayer perceptron, and is characterized by:

- Weights taking values in {-1, 1}, final layer included
- Deterministic sign functions as activation, making each unit in the hidden layers constrained to {-1, 1}

$$\sigma(x) = \begin{cases} 1, & \text{if } x > 0. \\ -1, & \text{otherwise.} \end{cases}$$

[2]Courbariaux et al. (2016)

BINARY NEURAL NETWORKS[2]

The **Binary Neural Network** (BNN) has architecture identical to a multilayer perceptron, and is characterized by:

- Weights taking values in {-1, 1}, final layer included
- Deterministic sign functions as activation, making each unit in the hidden layers constrained to {-1, 1}

$$\sigma(x) = \begin{cases} 1, & \text{if } x > 0. \\ -1, & \text{otherwise.} \end{cases}$$

Weights are initialized uniformly at random.

[2]Courbariaux et al. (2016)

Optimization by Simulated Annealing

Framing the problem

Fitting as a Combinatorial Optimization Problem:

- The spins in the weight matrices take values in a finite set $\mathcal{X}$ of possible configurations.

Framing the problem

FITTING AS A COMBINATORIAL OPTIMIZATION PROBLEM:

- The spins in the weight matrices take values in a finite set $\mathcal{X}$ of possible configurations.
- The cost function can be defined as the **cross-entropy loss**, commonly used for classification problems in machine learning:

FRAMING THE PROBLEM

FITTING AS A COMBINATORIAL OPTIMIZATION PROBLEM:

- The spins in the weight matrices take values in a finite set $\mathcal{X}$ of possible configurations.
- The cost function can be defined as the **cross-entropy loss**, commonly used for classification problems in machine learning:

$$E(c) = \sum_{i}^{N} CE_i(c)$$

$$CE_i(c) = -\sum_{k=1}^{K} \mathbb{1}_{[y_i=k]} \, log(\hat{p}(y_i = k))$$

Framing the problem

FITTING AS A COMBINATORIAL OPTIMIZATION PROBLEM:

- The spins in the weight matrices take values in a finite set $\mathcal{X}$ of possible configurations.
- The cost function can be defined as the **cross-entropy loss**, commonly used for classification problems in machine learning:

$$E(c) = \sum_{i}^{N} CE_i(c)$$

$$CE_i(c) = -\sum_{k=1}^{K} \mathbb{1}_{[y_i=k]} \, log(\hat{p}(y_i = k))$$

The solution is the configuration of weights s.t. the cost function is minimized.

Weight matrix

For ease of implementation one big weight matrix is initialized, computations are performed by slicing it at each layer.
The matrix has $(p + M(H-1) + K) \times M$ elements, being the composition of one matrix of size $p \times M$, $H-1$ matrices $M \times M$, and one $K \times M$ matrix.

WEIGHT MATRIX

For ease of implementation one big weight matrix is initialized, computations are performed by slicing it at each layer.
The matrix has $(p + M(H-1) + K) \times M$ elements, being the composition of one matrix of size $p \times M$, $H-1$ matrices $M \times M$, and one $K \times M$ matrix.

SIZE OF THE COP:
The set $\mathcal{X}$ of possible configurations has cardinality $2^{(p+M(H-1)+K)M}$

$$W^T = \begin{bmatrix} \omega_{1,1}^{(1)} & \dots & \omega_{1,p}^{(1)} & \dots & \omega_{1,1}^{(H)} & \dots & \omega_{1,M}^{(H)} & \beta_{1,1} & \dots & \beta_{1,K} \\ \vdots & \ddots & & & \vdots & \ddots & & \vdots & \ddots & \\ \omega_{M,1}^{(1)} & & \omega_{M,p}^{(1)} & \dots & \omega_{M,1}^{(H)} & & \omega_{M,M}^{(H)} & \beta_{M,1} & & \beta_{M,K} \end{bmatrix}$$

Framing the problem

We can translate the COP to a statistical mechanics problem by:

Framing the problem

We can translate the COP to a statistical mechanics problem by:

- Introducing a **Boltzmann-Gibbs** probability measure over the space of configurations $\mathcal{X}$:

$$P_\beta(c) = \frac{1}{Z(\beta)} e^{-\beta E(c)} \qquad Z(\beta) = \sum_{c \in \mathcal{X}} e^{-\beta E(c)}$$

Framing the problem

We can translate the COP to a statistical mechanics problem by:

- Introducing a **Boltzmann-Gibbs** probability measure over the space of configurations $\mathcal{X}$:

$$P_\beta(c) = \frac{1}{Z(\beta)} e^{-\beta E(c)} \qquad Z(\beta) = \sum_{c \in \mathcal{X}} e^{-\beta E(c)}$$

- Interpreting the cost function of the COP as the energy of the BG distribution

FRAMING THE PROBLEM

We can translate the COP to a statistical mechanics problem by:

- Introducing a **Boltzmann-Gibbs** probability measure over the space of configurations $\mathcal{X}$:

$$P_\beta(c) = \frac{1}{Z(\beta)} e^{-\beta E(c)} \qquad Z(\beta) = \sum_{c \in \mathcal{X}} e^{-\beta E(c)}$$

- Interpreting the cost function of the COP as the energy of the BG distribution

As $\beta \to +\infty$, $P_\beta(c)$ concentrates around the minimum cost (energy) configurations.

Simulated Annealing

In **simulated annealing** the temperature $T = 1/\beta$ is slowly lowered and samples are drawn from $P_\beta(c)$ through MCMC iterations[3].

[3]Kirkpatrick et al. (1983)

Simulated Annealing

In **simulated annealing** the temperature $T = 1/\beta$ is slowly lowered and samples are drawn from $P_\beta(c)$ through MCMC iterations[3].

A **cooling schedule** is set a priori, it will determine how temperature is lowered and how many Monte Carlo steps will be performed for each level. Starting from T_0, at each step k:

- exponential schedule: $T_k = T_0\, a^k, \quad 0.8 < a < 0.9$
- linear schedule: $T_k = T_k - a\, k, \quad 0 < 1 - a \ll 1$

[3]Kirkpatrick et al. (1983)

Simulated Annealing

At each step, starting from an initial configuration W_0 of the weight matrix with associated level of cross entropy (energy) $E(W_0)$:

1. **Switch sign to a random element** in the weight matrix to obtain a new configuration W_1

SIMULATED ANNEALING

At each step, starting from an initial configuration W_0 of the weight matrix with associated level of cross entropy (energy) $E(W_0)$:

1. **Switch sign to a random element** in the weight matrix to obtain a new configuration W_1
2. Compute $E[W_1]$, the cross-entropy of the model's predictions with the new configuration

Simulated Annealing

At each step, starting from an initial configuration W_0 of the weight matrix with associated level of cross entropy (energy) $E(W_0)$:

1. **Switch sign to a random element** in the weight matrix to obtain a new configuration W_1
2. Compute $E[W_1]$, the cross-entropy of the model's predictions with the new configuration
3. **Metropolis Step:** Accept the new configuration with probability:

$$P_{\text{acc}} = \begin{cases} 1 & \text{if } E[W_1] < E[W_0]. \\ \exp\left\{ \frac{E[W_1]-E[W_0]}{T} \right\} & \text{otherwise.} \end{cases}$$

Simulated Annealing

At each step, starting from an initial configuration W_0 of the weight matrix with associated level of cross entropy (energy) $E(W_0)$:

1. **Switch sign to a random element** in the weight matrix to obtain a new configuration W_1
2. Compute $E[W_1]$, the cross-entropy of the model's predictions with the new configuration
3. **Metropolis Step:** Accept the new configuration with probability:

$$P_{\text{acc}} = \begin{cases} 1 & \text{if } E[W_1] < E[W_0]. \\ \exp\left\{\frac{E[W_1]-E[W_0]}{T}\right\} & \text{otherwise.} \end{cases}$$

This enables the algorithm to escape local minima by allowing perturbations with decreasing probability.
As $T \approx 0$, SA reduces to a local search.

Simulated Annealing

A desirable property for a solution is **generalizability**, that translates to low test error.

- Solution belonging to **wide flat regions** of the loss landscape will have better generalization properties. Such solutions have an high density of nearby configurations with similar performance in terms of loss.[4][5]

For BNNs, it is possible to explore the **solution landscape** by changing sign to a given proportion of weights at random, and checking how energy (or accuracy) changes consequently. The procedure is repeated and then averaged.

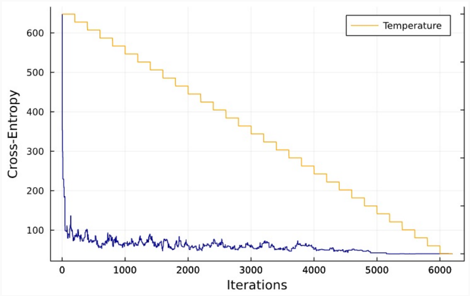
[4]Baldassi et al.(2015)
[5]Baldassi et al. (2019)

Performance on experiments

Binary Classification

For binary classification, the **Breast Cancer Wisconsin** dataset was used:

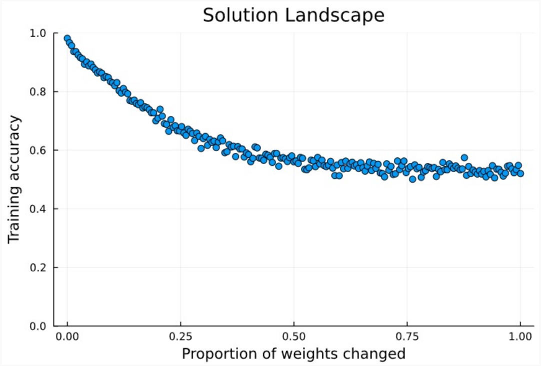
- Sample size N = 699, 10 real-valued input features
- The BNN was trained on 70% of the dataset, tested on 30%
- H=2 layers, each having M = 10 units
- **Linear cooling schedule**



Training set accuracy: 98.16%; **Test set accuracy:** 96.19%

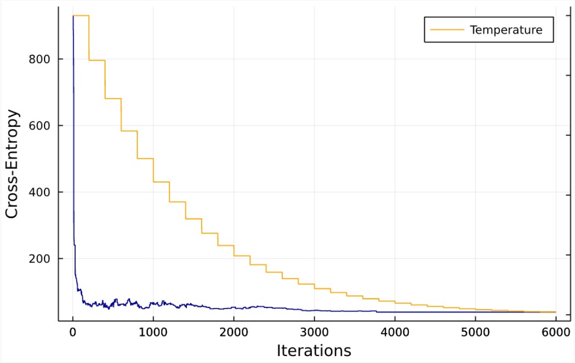
Binary Classification

The **minimum energy configuration** was reached after 3771 iterations. High accuracy on the test set is motivated by the minimum being in a region of high solution density, as can be seen from the **solution landscape**:



Binary Classification

If an **exponential cooling schedule** is used, the performance is slightly worse.

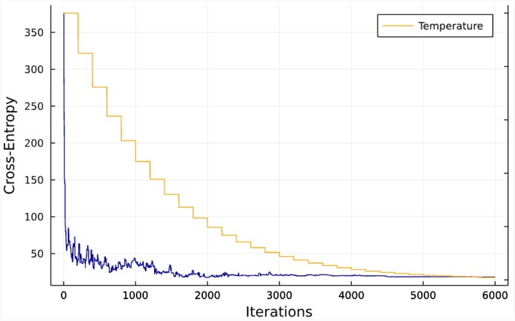


Training set accuracy: 96.73%; **Test set accuracy:** 94.29%

Multiclass Classification

For multiclass classification, the **Iris dataset** was used:

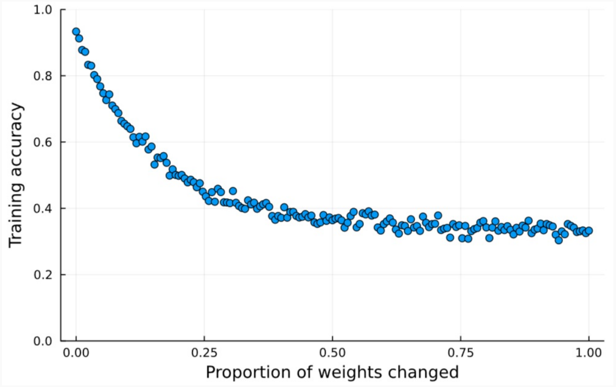
- Sample size N = 150, 4 real-valued input features, classes $K = 3$
- The BNN was trained on 70% of the dataset, tested on 30%
- H=2 layers, each having M = 10 units
- **Exponential cooling schedule**



Training set accuracy: 93.33%; **Test set accuracy:** 80%

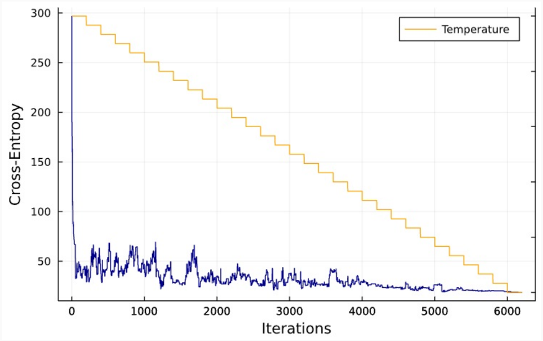
Multiclass Classification

The **minimum energy configuration** was reached after 1998 iterations. Now the minimum is in a region of **low solution density**, this motivates the poor generalization ability of the model:



Multiclass Classification

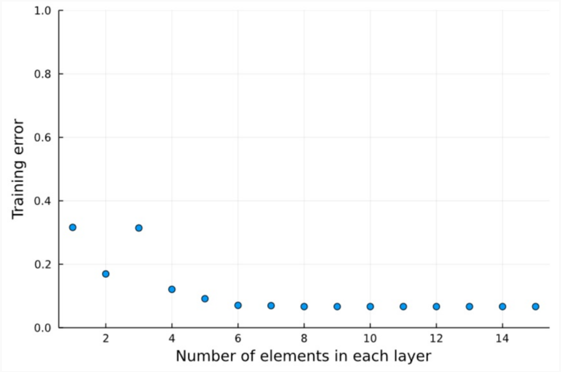
If a **linear cooling schedule** is used, the performance is slightly worse on the testing set.



Training set accuracy: 93.34%; **Test set accuracy:** 77.78%

Lower bounds on train error

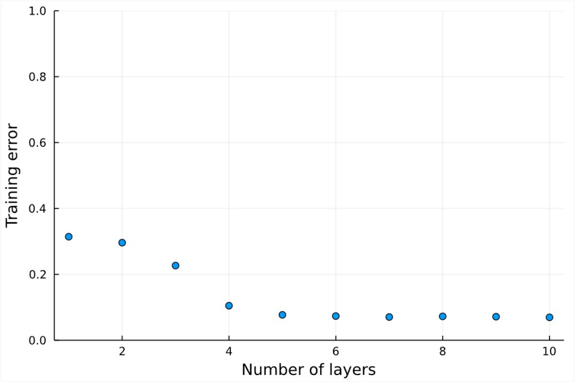
In practice, BNNs exhibit a lower bound on misclassification error on the training set, that rarely reaches zero even with many and/or bigger layers. This **resistance to over-fitting** could be attributable to the binary constraint.



Training error (averaged on 10 random seeds) of taking bigger layers (increasing M): solutions don't improve significantly. Same happens as more layers are added.

LOWER BOUNDS ON TRAIN ERROR

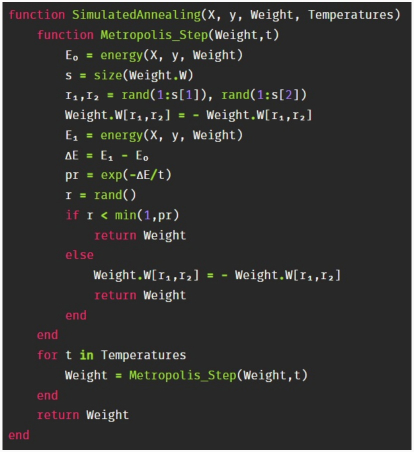
In practice, BNNs exhibit a lower bound on misclassification error on the training set, that rarely reaches zero even with many and/or bigger layers. This **resistance to over-fitting** could be attributable to the binary constraint.



Training error (averaged on 10 random seeds) of taking more layers (increasing H): solutions don't improve significantly.

Implementation details

Complexity of Metropolis iterations



```
function SimulatedAnnealing(X, y, Weight, Temperatures)
    function Metropolis_Step(Weight,t)
        E₀ = energy(X, y, Weight)
        s = size(Weight.W)
        r₁,r₂ = rand(1:s[1]), rand(1:s[2])
        Weight.W[r₁,r₂] = - Weight.W[r₁,r₂]
        E₁ = energy(X, y, Weight)
        ΔE = E₁ - E₀
        pr = exp(-ΔE/t)
        r = rand()
        if r < min(1,pr)
            return Weight
        else
            Weight.W[r₁,r₂] = - Weight.W[r₁,r₂]
            return Weight
        end
    end
    for t in Temperatures
        Weight = Metropolis_Step(Weight,t)
    end
    return Weight
end
```

- Energy has to be computed at each iteration. This means fitting a new model with the new configuration. Since this requires matrix multiplication and sequential passes through the activation, time complexity (with respect to the four parameters defining the size of the weight matrix) is:

$$O(Npm) + O(HNM^2) + O(NM)$$

XNOR FOR MATRIX MULTIPLICATION

According to [6], a computational advantage of BNNs is the possibility to compute multiplications between matrices of bits instead of matrices of integers:

- Conversion of matrices of integers $\in \{-1, 1\}$ to Julia's BitArrays

[6] Courbariaux et al. (2016)

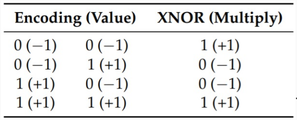
[7] Simons, Lee (2019)

[8] https://sushscience.wordpress.com/2017/10/01/understanding-binary-neural-networks/

XNOR FOR MATRIX MULTIPLICATION

According to [6], a computational advantage of BNNs is the possibility to compute multiplications between matrices of bits instead of matrices of integers:

- Conversion of matrices of integers $\in \{-1, 1\}$ to Julia's BitArrays
- Application of XNOR operator for couples of entries. The operation is equivalent to a multiplication:



Encoding (Value)		XNOR (Multiply)
0 (−1)	0 (−1)	1 (+1)
0 (−1)	1 (+1)	0 (−1)
1 (+1)	0 (−1)	0 (−1)
1 (+1)	1 (+1)	1 (+1)

[7]

[6] Courbariaux et al. (2016)

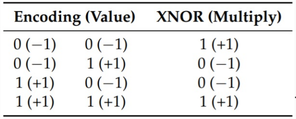
[7] Simons, Lee (2019)

[8] https://sushscience.wordpress.com/2017/10/01/understanding-binary-neural-networks/

XNOR FOR MATRIX MULTIPLICATION

According to [6], a computational advantage of BNNs is the possibility to compute multiplications between matrices of bits instead of matrices of integers:

- Conversion of matrices of integers $\in \{-1, 1\}$ to Julia's BitArrays
- Application of XNOR operator for couples of entries. The operation is equivalent to a multiplication:



Encoding (Value)		**XNOR (Multiply)**
0 (-1)	0 (-1)	1 $(+1)$
0 (-1)	1 $(+1)$	0 (-1)
1 $(+1)$	0 (-1)	0 (-1)
1 $(+1)$	1 $(+1)$	1 $(+1)$

[7]

- Sum the n XNOR outputs for each entry of the new matrix (call it s)

[6] Courbariaux et al. (2016)

[7] Simons, Lee (2019)

[8] https://sushscience.wordpress.com/2017/10/01/understanding-binary-neural-networks/

XNOR for matrix multiplication

According to [6], a computational advantage of BNNs is the possibility to compute multiplications between matrices of bits instead of matrices of integers:

- Conversion of matrices of integers $\in \{-1, 1\}$ to Julia's BitArrays
- Application of XNOR operator for couples of entries. The operation is equivalent to a multiplication:

Encoding (Value)		XNOR (Multiply)
$0\ (-1)$	$0\ (-1)$	$1\ (+1)$
$0\ (-1)$	$1\ (+1)$	$0\ (-1)$
$1\ (+1)$	$0\ (-1)$	$0\ (-1)$
$1\ (+1)$	$1\ (+1)$	$1\ (+1)$

[7]

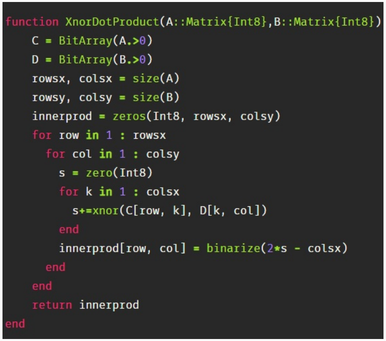
- Sum the n XNOR outputs for each entry of the new matrix (call it s)
- Apply the following function[8]: $$b(x) = \begin{cases} 1, & \text{if } 2 \cdot s - n > 0. \\ -1, & \text{otherwise.} \end{cases}$$

[6] Courbariaux et al. (2016)

[7] Simons, Lee (2019)

[8] https://sushscience.wordpress.com/2017/10/01/understanding-binary-neural-networks/

XNOR FOR MATRIX MULTIPLICATION



```
function XnorDotProduct(A::Matrix{Int8},B::Matrix{Int8})
    C = BitArray(A.>0)
    D = BitArray(B.>0)
    rowsx, colsx = size(A)
    rowsy, colsy = size(B)
    innerprod = zeros(Int8, rowsx, colsy)
    for row in 1 : rowsx
      for col in 1 : colsy
        s = zero(Int8)
        for k in 1 : colsx
          s+=xnor(C[row, k], D[k, col])
        end
        innerprod[row, col] = binarize(2*s - colsx)
      end
    end
    return innerprod
end
```

With respect to the original matrix multiplication:

- Time complexity stays unchanged: $\Theta(M^3)$ for matrices $M \times M$
- Memory requirements are reduced
- In practice... built-in method is faster

References

References

- **Baldassi, C., Pittorino, F., Zecchina, R.** (2019), Shaping the learning landscape in neural networks around wide flat minima. *Proceedings of the National Academy of Sciences.*
- **Baldassi, C. Ingrosso, A., Lucibello, C., Saglietti, L., Zecchina, R.**(2015), Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses. *Phys. Rev. Lett. 115, 128101*
- **Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y.** (2016), Binarized Neural Networks. *Proceedings of the Advances in Neural Information Processing Systems*
- **Hastie, T.; Tibshirani, R. Friedman, J.** (2009), The Elements of Statistical Learning - 2nd Ed., *Springer New York.*
- **Simons, T., Lee, D.** (2019), A Review of Binarized Neural Networks. *Electronics*.

Thank you!