# Binarized Neural Networks On Tensorflow

La Sapienza Università di Roma    Neural Networks

Michele Ciciolla    Flavio Lorenzi

1869990    1662963

prof. Uncini    prof. Scardapane

## Brief state of art

words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words words

## Introduction

In this work we studied the foundamentals of neural networks with binary weights and activations constrained to either $+1$ or $-1$ via a deterministic appoach using the Sign(x) function

$$x^b = Sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & else \end{cases}$$

$\pm 1$ values are very advantageous from a hardware perspective because they drastically reduce memory size and accesses, and replace most arithmetic operations with bit-wise operations, which is expected to substantially improvepower-efficiency ( `https://arxiv.org/abs/1602.02830` ).

In BNNs, floating point multiplications are reduced to binary operations or approximated power of two binary shifts; according to this fact these solutions are introduced:

1. Shift Based Batch Normalizations: used to accellerate the training, it almost aproximates the Vanilla BN without multiplications and not loosing accuracy.

2. Approximated power of two binary shifts: used to replace multiplications during scale and updating parameters in the learning rule.

3. Shift Based Adamax: Adamax optimizer with multiplications replaced by binary shift operations.

Given the main concepts of BNNs, it's now explained how they're applied for MNIST and CIFAR-10 datasets.

# The network for MNIST

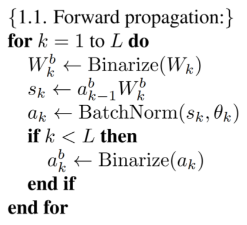
The script MNISTtrain.py gives the possibility to train two networks:

1. a binary fully connected neural network using the classical batch normalization and Vanilla Adam optimizer

2. a binary fully connected neural network using shift based batch normalization and shift based AdaMax optimizer

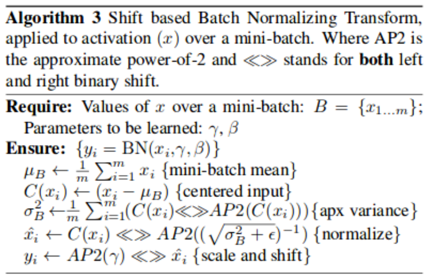
Both are built as a MLP of 2 hidden layers [2048] + input layer [784] + output layer [10] based on Sign(x) activation function explained in the layer.binarize method and sparse_softmax for computing cross entropy.

**Vanilla binary dense network** This network adopts a classical MLP architecture with weight tensor initialized using Xavier initializer. During the forward pass the activation tensor is normalized using tf.layers.batch_normalization function of Tensorflow. As soon as this pass is completed, loss is computed and the backward propagation starts with the goal of updating all the parameters according to the direction computed by the optimizer Adam.



```
{1.1. Forward propagation:}
for $k = 1$ to $L$ do
  $W_k^b \leftarrow \text{Binarize}(W_k)$
  $s_k \leftarrow a_{k-1}^b W_k^b$
  $a_k \leftarrow \text{BatchNorm}(s_k, \theta_k)$
  if $k < L$ then
    $a_k^b \leftarrow \text{Binarize}(a_k)$
  end if
end for
```

**Shift-based batch normalization** The aim of shift-based batch normalization (sbn) is to emulate a classical normalization without using floating multiplication. This should be done using a right/left binary shift when getting the apx variance, xdot and the normalized activation output. In our work we tried to emulate this behaviour although we mainteined some multiplication that cannot be excluded due to int32 and float32 tensor type error. AP2(x) usage is respected.



**Algorithm 3** Shift based Batch Normalizing Transform, applied to activation $(x)$ over a mini-batch. Where AP2 is the approximate power-of-2 and $\lll\ggg$ stands for **both** left and right binary shift.

```
Require: Values of $x$ over a mini-batch: $B = \{x_{1\dots m}\}$;
  Parameters to be learned: $\gamma$, $\beta$
Ensure: $\{y_i = \text{BN}(x_i,\gamma,\beta)\}$
  $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ {mini-batch mean}
  $C(x_i) \leftarrow (x_i - \mu_B)$ {centered input}
  $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(C(x_i) \lll\ggg AP2(C(x_i)))$ {apx variance}
  $\hat{x}_i \leftarrow C(x_i) \lll\ggg AP2((\sqrt{\sigma_B^2 + \epsilon})^{-1})$ {normalize}
  $y_i \leftarrow AP2(\gamma) \lll\ggg \hat{x}_i$ {scale and shift}
```

# The network for CIFAR-10

1. Qui dai una panoramica della tua rete seguendo le mie linee guida (9 righe)
2. Visto che la normalizzazione lho introdotta io, descrivi poi brevemente adamax e come si differenzia da adam (3-4 righe).

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t \leftarrow \max\left(\beta_2 \cdot v_{t-1}\,, |g_t|\right)$$
$$\theta_t \leftarrow \theta_{t-1} - \left(\alpha \lll\ggg \mathrm{AP2}\left(\left(1 - \beta_1^t\right)^{-1}\right)\right) \cdot \left(m_t \lll\ggg \mathrm{AP2}\left(v_t^{-1}\right)\right)$$

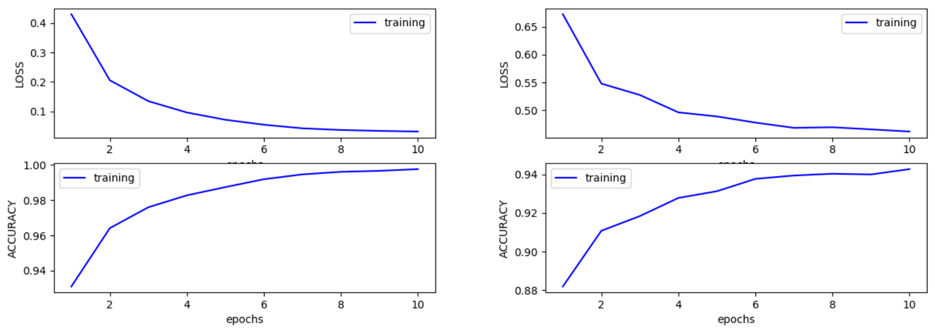


**Figure 1: Vanilla training (left) - ShiftBased (right)**

## Results and conclusion

We managed to achieve quite good results applying our BNN on the given datasets.

In particular for MNIST datasets the graphs below shows the trend of loss and accuracy during training epochs and final accuracy over a batch test.

1. Binary network with Vanilla Adam optimizer and batch normalization trained over 10 epochs shows to achieve an high accuracy on classification. During test phase avarage result is **0.97.**

2. Binary network with Shift Based Normalization and AdaMax resulted to be more slow in training phase, in fact after 10 epochs we achieved a sub-optimal model (in 20 epochs accuracy would increase for sure). Test phase got **0.95** of performance.

The same has been done over CIFAR-10 following the explained setups:

1. versione classica

2. versione con shift based

**Conclusion** Working on binarized neural networks, we've touched with hands the great power of this resources. Their light computation weight may open the way to an integration of deep learning in small devices with limited hardware/software availability like smartphones or small motherboard (Arduino,Raspberry Pi).

aggiungi qualcosa su come sia semplice e funzionale lavorare con tensorflow INTERAGENDO CON UN PROGETTO GIA' FATTO

## References

[1] https://software.intel.com/en-us/articles/accelerating-neural-networks-with-binary-arithmetic

[2] https://pdfs.semanticscholar.org/3a34/aa848678ff939dfcb2effd00300717037836.pdf

[3] https://github.com/MatthieuCourbariaux/BinaryNet

[4] https://github.com/itayhubara/BinaryNet.pytorch

[5] https://github.com/ivanbergonzani/binarized-neural-network