<u>Binarized Neural Networks On Tensorflow</u>

La Sapienza Università di Roma   Neural Networks

Michele Ciciolla   Flavio Lorenzi

1869990   1662963

prof. Uncini   prof. Scardapane

Concept

Is it possible to run a deep neural network under the hardware limitations of small devices with limited battieries and memory space in an efficient way? What if you want to do some real-time detection/recognition using a deep learning system running on a images database?

Deep Neural Networks (DNNs) require powerful GPUs and memory for training and storing the trained model; small devices cannot supply such luxuries.

A BNN is defined by all binary numbers. Those are very cheap to store and it's obvious that the single bit representation uses 1/32 of the space required by the 32-bit representation comparing a 32-bit floating-point representation (used in most DNNs) with a single bit.

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 apoach using the Sign(x) function

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

± 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**
  $W_k^b \leftarrow \mathrm{Binarize}(W_k)$
  $s_k \leftarrow a_{k-1}^b W_k^b$
  $a_k \leftarrow \mathrm{BatchNorm}(s_k, \theta_k)$
  if $k < L$ **then**
    $a_k^b \leftarrow \mathrm{Binarize}(a_k)$
  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 $\ll\gg$ stands for **both** left and right binary shift.

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

The network for CIFAR-10

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

1. a binary convolutional network using the classical batch normalization and Vanilla Adam optimizer
2. a binary convolutional network using shift based batch normalization and shift based AdaMax optimizer

Both type are Convolutional Networks with operations of clip to remain between +1 and -1 values. They are characterized by an input layer, multiple hidden layers [1024] and an output layer [10] based on Sign function: convolutional layers convolve the input and pass its result to the next layer for the max_pooling operation, a discretization process to down-sample the input representation (reducing dimensionality at each step). Finally fully connected layers connect every neuron and prepare them for the output.

Adamax Adamax is an extension of Adam (Adaptive moment estimation) , a very excellent alternative based on estimates of lower-order moments: the main difference compared to Adam is the different composition of the u t moment. The idea is to look this value as the Euclidean l2 norm (it's than replaced with infinity norm to avoid instability) of the individual current and past gradients: so we will take the maximum value.

$$
\begin{aligned}
u_t &= \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty)|g_t|^\infty \\
&= \max(\beta_2 \cdot v_{t-1}, |g_t|)
\end{aligned}
$$

So then we can update the classic Adam equations replacing v t with u t obtaining the following rules:

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t$$

$$v_t = \max(\beta_2 . v_{t-1}, |g_t|)$$

$$\theta_t = \theta_{t-1} - (\alpha \oslash (1 - \beta_1^t)) . (m_t \oslash v_t^{-1})$$

mt and vt are moments, ϑt is the gradient descent equation; gt is the gradient, ϑt −1 is the previous parameter; α,β1,β2 are learning rate and the betas of Adam optimizer.

In our work we used the shift-based version of Adamax, so we can see the presence of ⊘ symbol to show the shifting operation (no more a simple multiplication between float values). Shift operation is essential in this kind of networks because division and multiplication can be easily replaced by right/left shift over binaries.

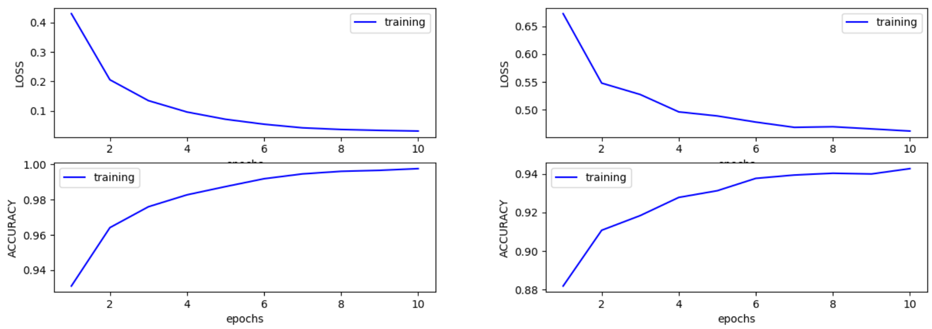

Figure 1: Results for MNIST: Vanilla training (left) - ShiftBased (right)

Results and conclusion

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

In particular for MNIST datasets the Figure 1 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 (Figure 2 below):

1. Binary network with Vanilla Adam optimizer and batch normalization trained over 20 epochs with a batchsize of 100 shows to achieve a very optimal accuracy on classification: the monitored average is **0.985** in training phase and **0.746** in test phase that are similar to the guideline paper ones.

2. Binary network with Shift Based Normalization and AdaMax resulted to be more slow in training phase, in fact after 20 epochs we achieved a suboptimal model (the accuracy is **0.85** in training phase and **0.746** in test phase, after about 9 hours of training).

For the second example it was thought to add the plot of the test too, to have further confirmation on the results achieved: it is interesting to see how

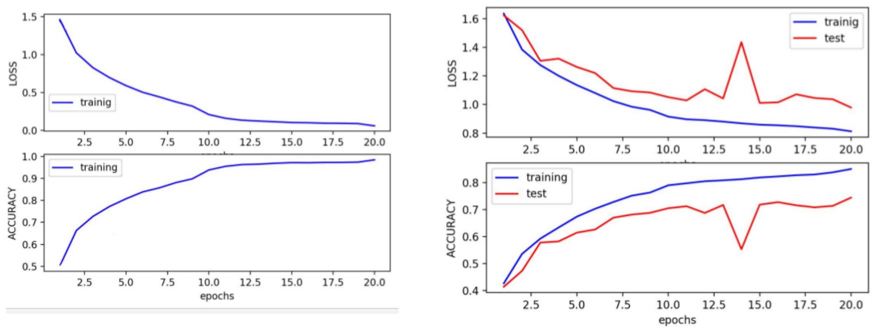

Figure 2: Results for CIFAR:Vanilla training (left) - ShiftBased (right)

it varies with respect to the training phase, resulting in being less efficient and more oscillating.

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).

The implementation of this work is based on two frameworks like Theano and Torch. Instead we have chosen to detach ourselves from these and start working with Tensorflow thanks to its ease of implementation and versatility. It also enjoys excellent popularity, it has very accurate documentation and it is followed by active discussion groups.

To conclude we noticed that our implementation reflects the results and aims presented in the paper about MNIST and CIFAR, obviously it has not been possible to exactly reproduce the all the experiments and results, due to unavailability of the GPU.

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