

APPARATUS AND METHOD FOR TRAINING BINARY DEEP NEURAL NETWORKS

FIELD OF THE INVENTION

5 This invention relates to machine learning models, in particular to the training of binary deep neural networks.

BACKGROUND

10 Deep learning is the origin of numerous successes in the fields of computer vision and natural language processing during the last decade. It has occupied a central spot of technological and social landscapes, with applications going far beyond computer sciences.

Deep learning uses deep neural networks (DNNs), which are complex non-linear systems vaguely inspired by biologic brains, being able to manipulate high-dimensional objects, autonomously learn from given
15 examples without being programmed with any task-specific rules and obtain state-of-the-art performance.

Figure 1 shows the basic operation scheme of a DNN, which is usually split into two phases. In the *training* phase, the DNN, noted as 'model' in the figure, learns its own parameters. A starting version of the model to be trained, shown at 101, is trained using provided training data 102 to form trained model 103. Then, in
20 the *inference* phase, the DNN is used to output one or more predictions 104 on unseen input data 105.

Deep neural networks are generally known as very intensive in terms of memory and computation. On one hand, DNNs are generally composed of a very large number of parameters, reaching hundreds of millions in today's applications. This requires a significant memory footprint for representing the model. On top of
25 that, the training phase requires a large amount of training data and incurs a lot of other temporary variables used for optimizing the model weights (also referred to as parameters), such as gradient information. As a result, training a DNN generally requires dedicated powerful infrastructure, which can limit the potential of artificial intelligence.

30 One promising approach for alleviating this memory wall issue is to design a DNN with binary parameters, meaning that the model parameters are represented by a binary number, consuming only 1 bit instead of a floating-point number of 32 bits. This would greatly reduce not only the model footprint, but also training memory and computation complexity.

35 However, binary parameters are discrete and cannot be optimized with the existing deep learning theory of gradient-descent (see, for example https://en.wikipedia.org/wiki/Gradient_descent).

40 To train binary deep neural networks, binarization is a prominent approach. Figure 2 shows a schematic operation diagram of this process, in which a floating-point model 201 is the starting DNN with floating-point parameters, and floating-point gradient-descent optimizer 202 is the block which optimizes the model

floating-point parameters using gradient-descent principles. Binarization block 203 converts model parameters from floating-point into binary number form, for instance by using sign extraction subject to some predefined performance criteria. Binarized model 204 is updated during the training process and the final model obtained at the end of the training process is to be used for inference.

5 The main limitation of this solution is that the training phase completely relies on the floating-point training. Not only does it not solve the memory and computational complexity issues, but it also adds more complexity to the training process.

10 It is desirable to develop a method which overcomes such problems.

SUMMARY

According to a first aspect, there is provided a device for training a binary deep neural network, the device being configured to: generate a training signal in dependence on an error between an output of a prototype
15 version of the binary deep neural network and an expected output, the prototype version of the binary deep neural network having multiple binary weights each having a respective value; and in dependence on the training signal, output for each binary weight of the prototype version of the binary deep neural network a respective decision to invert or maintain the respective value of the respective binary weight.

20 This may allow the device to train and optimize a DNN comprising binary weights directly in the binary domain, without the need for gradient processing methods. As a result, the training process may require reduced memory and computational power, because it avoids memory-intensive gradient signals, as well as floating-point multiplication.

25 The device may be configured to generate the training signal in dependence on a predefined optimization target. This may allow the training signal to reflect how good the prototype version of the DNN is at predicting the outcome when compared with the expected outcome.

The predefined optimization target may be the minimization of a loss function. This may be a convenient
30 implementation for training a deep neural network.

The training signal may comprise one or more quantities. The device may be configured to output the respective decision for each binary weight in dependence on the one or more quantities in order to meet the predefined optimization target. This may allow the device to use the training signal to determine whether
35 or not to invert a particular weight in order to meet the predefined optimization target, for example in order to minimize a predetermined loss function.

The device may be further configured to track the status of the predefined optimization target. This may allow the device to incorporate the status of the optimized entity into the optimization signal during an
40 iterative optimization process. This may allow the trained deep neural network to achieve better prediction accuracy during inference.

The respective decision to invert or maintain the respective value of each binary weight of the prototype version of the binary deep neural network may be based on an optimization signal computed in dependence on the training signal. This may allow the device to make a decision for inverting or maintaining each binary weight of the prototype version of the DNN.

Each binary weight may have only two possible values. This may allow the model parameters to be represented by a binary number, consuming only 1 bit instead of a floating-point number of 32 bits. This may reduce not only the model footprint, but also training memory and computation complexity.

The device may be configured to receive a set of training data for forming the output of the prototype version of the binary deep neural network, the set of training data comprising input data and respective expected outputs. The training signal may be formed in dependence on the error between the output of the prototype version of the binary deep neural network and the expected output. This may allow the prototype version of the binary DNN to process the input data to form a predicted output, which can then be compared to the respective expected output. This may allow the device to assess the performance of the prototype version of the binary DNN.

The device may further comprise a memory configured to store an accumulator updated in dependence on the predefined optimization function. This may allow control of the amount of binary weights to be inverted during each training iteration, so as to enhance the training convergence and performance.

The device may be configured to reset the memory in dependence on the respective decisions. For example, for each binary weight that is inverted in a particular iteration of the training process, the device may instruct a memory reset for a stored output of an accumulator function corresponding to that inverted weight. This may allow an accumulator to be reset each time a corresponding binary weight is inverted.

The device may be further configured to update the binary weights of the prototype version of the binary deep neural network in dependence on the respective decisions. This may allow the device to optimize the weights and update the DNN for use in the next iteration of the training process.

The device may be configured to iteratively update the binary weights of the prototype version of the binary deep neural network until a predefined level of convergence is reached. This may allow the resulting trained binary deep neural network to achieve a predefined level of performance during inference.

The binary deep neural network may be a Boolean deep neural network comprising Boolean neurons. Using a Boolean neuron design, Boolean layers and networks such as linear layers and convolutional layers may be constructed straightforwardly in the same way as floating-point layers and networks are constructed from floating-point neurons.

According to another aspect, there is provided a method for training a binary deep neural network, the binary deep neural network comprising multiple binary weights, the method comprising: generating a training signal in dependence on an error between an output of a prototype version of the binary deep neural network and an expected output, the prototype version of the binary deep neural network having multiple binary weights each having a respective value; and in dependence on the training signal, outputting for each binary weight of the prototype version of the binary deep neural network a respective decision to invert or maintain the respective value of the respective binary weight.

This method may allow for the optimization or training of a DNN comprising binary weights directly in the binary domain without the need for gradient processing methods. As a result, the training process may require reduced memory and computational power, because it avoids memory-intensive gradient signals as well as floating-point multiplication.

According to a further aspect, there is provided a computer readable storage medium having stored thereon computer readable instructions that, when executed at a computer system, cause the computer system to perform the method as set forth above. The computer system may comprise one or more processors. The computer readable storage medium may be a non-transitory computer readable storage medium.

BRIEF DESCRIPTION OF THE FIGURES

The present disclosure will now be described by way of example with reference to the accompanying drawings.

In the drawings:

Figure 1 schematically illustrates an operation scheme for a DNN.

Figure 2 schematically illustrates binarization of a model trained using a floating-point gradient-descent optimizer.

Figure 3 schematically illustrates the training of a binary DNN in the binary domain.

Figure 4 shows an example of processing flows and signals for one layer of a binary DNN.

Figure 5 shows a flowchart for an example of a method for training a binary DNN comprising multiple binary weights in accordance with embodiments of the present invention.

Figure 6 shows an example of a device for training a binary DNN.

Figures 7(a) and 7(b) show examples of results achieved using an embodiment of the present invention in terms of memory consumption and memory reduction respectively.

DETAILED DESCRIPTION

A weight is a parameter within a neural network that transforms input data within the network's hidden layers. A neural network comprises a series of nodes, or neurons. Within each neuron is a set of inputs, a set of weights and a bias value. As an input enters the node, it is multiplied by the respective weight value. It may optionally add a bias before passing the data to the next layer. The resulting output is either observed or passed to the next layer of the network.

Weights are learnable parameters within the network. A DNN may randomize the weights before learning initially begins. As training continues, the weights are adjusted toward the desired values to give the “correct” output.

- 5 Embodiments of the present invention can allow for the training and optimization of a DNN having binary weights directly in the binary domain without the need of gradient processing.

A ‘weight’ of a DNN may also be referred to as a ‘parameter’ interchangeably, having the same meaning.

- 10 Each binary weight has only two possible values. This can allow the model weights to be represented by a binary number, consuming only 1 bit instead of a floating-point number of 32 bits. This may reduce not only the model footprint, but also training memory and computation complexity.

- 15 Given a binary DNN, logic rules can be established for taking a decision to invert or to keep each binary weight. Such logic rules make decisions by using signals computed from the binary model and subject to achieving a predefined optimization target, such as minimizing a learning loss function.

- 20 When a decision for inverting a binary weight is given, the binary weight is inverted from its current binary value to the other possible binary value. This avoids the combinatorics nature of discrete optimization problems, which is NP (nondeterministic polynomial time)-Hard, and also takes the binary constraint as an advantage in that the binary parameter has only two values, such that inverting or maintaining the current value it is the ultimate question.

- 25 Figure 3 shows an exemplary embodiment of the present invention. A binary optimizer is shown at 300.

Binary deep neural network 301 is the DNN to be trained. DNN 301 comprises an input layer, an output layer and one or more hidden layers. The DNN 301 has binary weights. Initially, the weights of the DNN 301 may be randomized.

- 30 The training process follows a standard iterative process, which iterates loops of a forward pass followed by a backpropagation pass. In the forward pass, training data is injected into a prototype version of the DNN (i.e. the current version of the DNN in that iteration of the training process) for evaluating the error between the output produced by the prototype version of the DNN and the true data labels. In the backward pass, the error is propagated from the output layer throughout the network, back to the input layer. This
35 may be done in dependence on an optimization target, such as minimizing a predetermined loss function

During this backpropagation, different information is computed, including a training signal, indicated at 302. For instance, in the standard full-precision DNN, training signals include the gradient of the loss function with respect to each quantity to be optimized.

In this implementation, the training signal 302 is generated in dependence on the error between the output of the prototype version of the binary deep neural network 301 and an expected output (the true data labels of the training data).

- 5 In dependence on the training signal, a respective decision is output for each binary weight of the prototype version of the binary deep neural network 301 to invert or maintain the respective value of the respective binary weight, as will be described in more detail below.

10 The training signal can include quantities which are specifically defined to express how a predefined optimization target, such as a loss function, varies when inverting a binary weight. The quantities are not necessarily real-valued signals. These training signals are sent to and used by the processor 303. The processor 303 is configured to implement at least a function that outputs an optimization signal from the received training signal of each binary weight, which is taken as input to the function. The optimization signal is then fed to the decision logic 304, as well as the optimizer memory 305.

15 Computation of the optimization signal from the training signal can be implementation-specific. One generic way of computing the optimization signal from the training signal is to compute an accumulation of the training signal over multiple iterations of the training process.

20 The decision logic 304 uses the received optimization signal to take a decision for inverting or keeping each binary weight of the current version of the DNN. Upon instruction from decision logic 304, the binary inverter 306 can perform binary inversion of a binary weight or can maintain its value.

25 The controller 307 can take control of the optimization process. For instance, it can take into account the decision made by the decision logic 304 to instruct a memory reset, or to adapt the way that the processor 303 computes the optimization signal. The controller 307 can also track the status of the optimization target.

30 The updated binary weights 308 are sent back to the binary DNN and the weights of the prototype version of the binary DNN 301 are updated as required, i.e. to invert those for which a decision has been output to do so, or maintain them.

In one preferred embodiment, the binary DNN is a Boolean deep neural network which is made of Boolean neurons. A Boolean neuron has Boolean inputs $b_1, b_2, \dots, b_m$ and Boolean weights $w_0, w_1, \dots, w_m$ where m is the number of inputs, and in one particular example, outputs a Boolean value given as follows:

35

$$\text{output} = \begin{cases} \text{TRUE,} & \text{if } w_0 + \sum_{i=1}^m \text{XOR}(b_i, w_i) \geq T, \\ \text{FALSE,} & \text{if } w_0 + \sum_{i=1}^m \text{XOR}(b_i, w_i) < T, \end{cases}$$

wherein XOR is the Boolean exclusive-or logic, and T is a pre-defined threshold and in this example is set to $T = (m + 1)/2$.

Using this Boolean neuron design, Boolean layers and networks, such as linear layers and convolutional layers, are constructed straightforwardly in the same way as floating-point layers and networks are constructed from floating-point neurons.

With the Boolean network described above, Figure 4 shows an example of signals and flows in the processing of one layer of the binary DNN according to this preferred embodiment. The Boolean optimizer is indicated at 401. A layer of the DNN is indicated at 402.

In Figure 4, left-to-right arrows indicate forward processing, right-to-left arrows indicate backward processing, and arrows in vertical directions are between the layer 402 and the optimizer 401. In particular, where W is a weight, Z is the backpropagation signal which is received from a downstream layer and $U = \text{MAJ}(\text{XOR}(W, Z))$ is the signal to be sent to an upstream layer. X is the feedforward input to the layer and $Y = \text{MAJ}(\text{XOR}(X, W))$ is the signal to be sent to a downstream layer.

The weights W and the feedforward inputs X can be stored at the layer, as indicated at 403.

In this example, the training signal Q is given as $Q = \text{MAJ}(\text{XOR}(X, Z))$, in which $\text{XOR}(X, Z)$ is element-wise XOR in appropriate matching dimensions of X and Z , and $\text{MAJ}(A)$ is majority vote of Boolean array A which outputs TRUE if in A contains more TRUES and FALSEs, and outputs FALSE otherwise.

In this example, for a Boolean neuron of XOR logic, the rule for optimizing a binary weight is 'invert weight W if $\text{XOR}(W, \text{XOR}(X, Z)) = \text{FALSE}$ '. In this example, training signal is $Q := \text{XOR}(X, Z)$. This signal Q is a quantity used to determine whether or not to invert W in order to minimize the predefined loss function.

Other logic rules may alternatively be used to determine whether or not to invert W , as appropriate.

The training signal is therefore used to determine whether or not to invert the value of a binary weight in the prototype version of the binary DNN.

Going back to Figure 3, in one example, the memory 305 stores an accumulator M . The controller 307 controls the optimizer operation. The controller 307 can specify a first scalar parameter ALPHA and a second scalar parameter BETA, which can be fixed or adapted during the training process, as required. ALPHA provides the ability to control the amount of binary weights to be changed in each iteration. BETA behaves like a forgetting parameter, which reflects the system evolution during the training process, mimicking the brain-plasticity phenomenon.

In one embodiment, the controller 307 adapts BETA in each training epoch as the ratio between the number of not-inverted binary weights to the total number of binary weights of the layer. Here, the number of not-inverted binary weights is obtained from the decision logic 304.

- 5 For each binary weight which is inverted in a particular iteration of the training process, the controller 307 can instruct a memory reset of value M of the accumulation corresponding to that inverted weight.

The processor 303 can perform $M \leftarrow \text{BETA} * M + \text{ALPHA} * Q$, in which '*' stands for the standard real-valued multiplication. The updated M for an inverted weight is stored in the memory 305.

- 10 The decision logic 304 gives an inversion instruction of weight W if $W = \text{True}$ and $M \geq 1$, or $M = \text{False}$ and $M \leq -1$.

- 15 The binary inverter 306 executes $W = \text{NOT } W$ upon an inversion instruction. The values of the weights are then updated in the prototype version of the DNN 301 for use in the next training iteration. Alternatively, if the trained model has converged, the DNN with those updated weights is used for the inference phase.

- Figure 5 shows an example of a method for training a binary deep neural network in accordance with embodiments of the present invention. As described above, the binary deep neural network comprises multiple binary weights. At step 501, the method comprises generating a training signal in dependence on an error between an output of a prototype version of the binary deep neural network and an expected output, the prototype version of the binary deep neural network having multiple binary weights each having a respective value. At step 502, the method comprises, in dependence on the training signal, outputting for each binary weight of the prototype version of the binary deep neural network a respective decision to invert or maintain the respective value of the respective binary weight.

Figure 6 shows an example of a device configured to implement the above methods.

- The device 600 may comprise at least one processor, such as processor 601, and at least one memory, such as memory 602. The memory stores in a non-transient way code that is executable by the processor(s) to implement the device in the manner described herein.

The device 600 may also be configured to implement a binary deep neural network trained according to the method 500, with optional additional features as described with respect to the above embodiments. This may allow the device to perform inference using the binary DNN for a variety of tasks.

- 35 The main advantage of the approach described herein is that it allows to train a binary DNN directly in the binary domain without the need for a floating-point (or full-precision) gradient, resulting in multiple-time factor of memory and computational complexity reduction, while approaching prediction accuracy of the full-precision training, as illustrated in Table 1 below.

40

Type	Method	Test Acc. (%)
------	--------	---------------

BINARY-TRAINED	Present method	89.5
FULL-PRECISION TRAINED BINARIZATION	Binarized Neural Network	89.9
	XNOR-Net	89.8
	Loss-Aware Binarization	87.7
	Differentiable Soft Quantization	91.7
FULL-PRECISION	Optimized Full-Precision	93.8

Table 1: Test accuracy on the CIFAR10 (Canadian Institute for Advanced Research) image dataset with VGG (Visual Geometry Group) Small architecture, using a batch size of 100.

- 5 Figures 7(a) and 7(b) exemplify the advantages of the preferred embodiment described above in terms of memory consumption and memory reduction respectively, compared to the full-precision training.

Embodiments of the present invention can therefore allow for the training and optimization of binary DNNs directly in the binary domain without the need of a gradient. Compared to existing solutions, which require
10 DNNs to compute gradient signals, the training process requires much less memory and computational power because it avoids memory-intensive gradient signals. The approach is native to deep architectures.

The method works directly on binary weights of the binary DNN. Existing solutions require two versions of each quantity (a full-precision version and a binarized one), such as weights and a neuron's input and
15 output. Avoiding full-precision signals, as in the training process described herein, requires much less memory and computational power.

The optimizer controller reflects the decision results on the processor and memory and allows the binary training process to achieve better prediction accuracy. This is an innovative component which incorporates
20 the status of the optimized entity into the optimization signal during the iterative optimization process. This can allow the binary training process to achieve better prediction accuracy.

The applicant hereby discloses in isolation each individual feature described herein and any combination of two or more such features, to the extent that such features or combinations are capable of being carried
25 out based on the present specification as a whole in the light of the common general knowledge of a person skilled in the art, irrespective of whether such features or combinations of features solve any problems disclosed herein, and without limitation to the scope of the claims. The applicant indicates that aspects of the present invention may consist of any such individual feature or combination of features. In view of the foregoing description it will be evident to a person skilled in the art that various modifications may be made
30 within the scope of the invention.

CLAIMS

1. A device (600) for training a binary deep neural network, the device being configured to:
generate (501) a training signal (302) in dependence on an error between an output of a prototype
5 version of the binary deep neural network (301) and an expected output, the prototype version of the binary
deep neural network (301) having multiple binary weights each having a respective value; and
in dependence on the training signal (302), output (502) for each binary weight of the prototype
version of the binary deep neural network (301) a respective decision to invert or maintain the respective
value of the respective binary weight.
10
2. The device (600) as claimed in claim 1, wherein the device is configured to generate the training signal
(302) in dependence on a predefined optimization target.
3. The device (600) as claimed in claim 2, wherein the predefined optimization target is the minimization of
15 a loss function.
4. The device (600) as claimed in claim 2 or claim 3, wherein the training signal (302) comprises one or
more quantities and wherein the device (600) is configured to output the respective decision for each binary
weight in dependence on the one or more quantities in order to meet the predefined optimization target.
20
5. The device (600) as claimed in any of claims 2 to 4, wherein the device (600) is further configured to
track the status of the predefined optimization target.
6. The device (600) as claimed in any preceding claim, wherein the respective decision to invert or maintain
25 the respective value of each binary weight of the prototype version of the binary deep neural network (301)
is based on an optimization signal computed in dependence on the training signal (302).
7. The device (600) as claimed in any preceding claim, wherein each binary weight has only two possible
values.
30
8. The device (600) as claimed in any preceding claim, wherein the device (600) is configured to receive a
set of training data for forming the output of the prototype version of the binary deep neural network (301),
the set of training data comprising input data and respective expected outputs.
- 35 9. The device (600) as claimed in any preceding claim, wherein the device (600) further comprises a
memory (305, 602) configured to store an accumulator updated in dependence on the predefined
optimization function.
- 40 10. The device (600) as claimed in claim 9, wherein the device is configured to reset the memory (305, 602)
in dependence on the respective decisions.

11. The device (600) as claimed in any preceding claim, wherein the device (600) is further configured to update the binary weights of the prototype version of the binary deep neural network (301) in dependence on the respective decisions.

5 12. The device (600) as claimed in claim 11, wherein the device (600) is configured to iteratively update the binary weights of the prototype version of the binary deep neural network (301) until a predefined level of convergence is reached.

10 13. The device (600) as claimed in any preceding claim, wherein the binary deep neural network (301) is a Boolean deep neural network comprising Boolean neurons.

14. A method (500) for training a binary deep neural network, the binary deep neural network comprising multiple binary weights, the method comprising:

15 generating (501) a training signal in dependence on an error between an output of a prototype version of the binary deep neural network (301) and an expected output, the prototype version of the binary deep neural network (301) having multiple binary weights each having a respective value; and
in dependence on the training signal (302), outputting (502) for each binary weight of the prototype version of the binary deep neural network (301) a respective decision to invert or maintain the respective value of the respective binary weight.

20 15. A computer-readable storage medium (602) having stored thereon computer-readable instructions that, when executed at a computer system (601), cause the computer system to perform the method (500) of claim 14.