(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 September 2020 (10.09.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/177863 A1

(51) International Patent Classification:
G06N 3/063 (2006.01) *G06N 3/08* (2006.01)
G06N 3/04 (2006.01)

(21) International Application Number:
PCT/EP2019/055537

(22) International Filing Date:
06 March 2019 (06.03.2019)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **NOKIA TECHNOLOGIES OY** [FI/FI]; Karakaari 7, 02610 Espoo (FI).

(72) Inventors: **AIT AOUDIA, Faycal**; 3 Parc de la Bérengère, 92210 Saint-Cloud (FR). **HOYDIS, Jakob**; 47 rue Sarrette, 75014 Paris (FR).

(74) Agent: **AARNIO, Ari** et al.; NOKIA TECHNOLOGIES OY, IPR Department, Karakaari 7, 02610 Espoo (FI).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

WO 2020/177863 A1

(54) Title: TRAINING OF ALGORITHMS

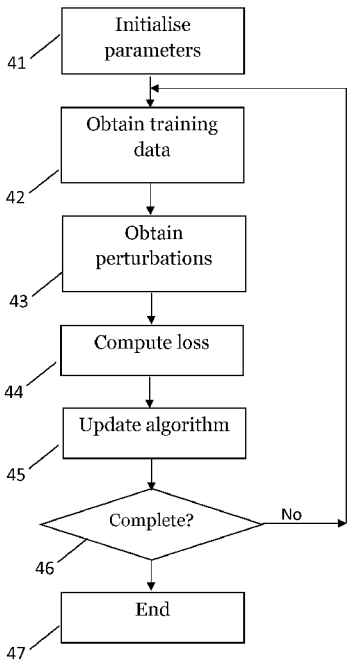

Fig. 4

(57) Abstract: An apparatus, method and computer program is described comprising: initialising parameters of an algorithm; generating or obtaining training data comprising inputs and desired target outputs for training said algorithm; generating or obtaining a set of random perturbation vectors to be applied after at least some arithmetic operations of the algorithm; updating at least some trainable weights of the algorithm based on a loss function; and repeating the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached.

Training of Algorithms

Field

The present specification relates to training of algorithms, such as neural networks, having at least some trainable weights.

Background

An algorithm, such as a neural network, can be trained using a loss function. Obstacles to practical hardware implementations of such algorithms include high memory requirements and computational complexity. There remains scope for further developments in this area.

Summary

In a first aspect, this specification describes an apparatus comprising: means for initialising parameters of an algorithm, wherein said algorithm having at least some trainable weights and at least some arithmetic operations; means for generating or obtaining training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs; means for generating or obtaining a set of random (e.g. pseudo-random) perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm (e.g. all mathematical operations that might introduce an error, such as due to the use of fixed-point arithmetic), wherein said perturbation vectors have a distribution (e.g. a probability density function) depending on properties of a target device implementing said algorithm and depending on the arithmetic operations; means for updating at least some of the trainable weights of the algorithm based on a loss function; and means for repeating the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached. The perturbation vector may be random or pseudo-random perturbations within a defined distribution. For example, the perturbation vectors may be evenly distributed within a defined set (wherein the defined set defines the distribution of the perturbation vectors).

The algorithm may comprise a neural network. Alternatively, or in addition, the algorithm may have a plurality of layers (e.g. a plurality of dense layers), each layer having at least some trainable weights and each layer including one or more arithmetic operations.

The arithmetic operations may include multiplications. In some embodiments, the perturbation vectors are applied after each multiplication.

Some embodiments comprise means for determining an error distribution of the target device for use in defining said distribution of said perturbation vectors. The error distribution may be determined by measurement. One example is a histogram approach. The distribution of the perturbation vectors may be based on a "best fit" to the measured error distribution.

The target device may be one of an application-specific integrated circuit and a field-programmable gate array.

The target device may implement a fixed-point arithmetic. In one embodiment, training may be conducted using a floating-point arithmetic.

Some embodiments comprise means for quantizing said trainable weights, such that said weights can only take values within a codebook having a finite number of entries that is a subset of the possible values available during updating. The said means for repeating may comprise means for repeating the generation or obtaining of training data, the generating or obtaining of random perturbation vectors, the updating of said trainable weights and the quantizing of said trainable weights until the first condition is reached.

The said loss function may comprise a penalty term related to the quantization of the trainable weights. The said penalty term may include a variable that is adjusted on each repetition of the updating and quantizing such that, on each repetition, more weight is given in the loss function to a difference between the trainable weights and the quantized trainable weights.

The first condition may be met when a performance metric for the algorithm is met. Alternatively, or in addition, the first condition may be met when a performance metric has been unchanged (e.g. unchanged within a margin) for a defined number of iterations. Alternatively, or in addition, the first condition may comprise a defined number of iterations (e.g. a maximum number of iterations).

The loss function may be related to one or more of mean squared error, block error rate and categorical cross-entropy.

The said at least some weights of the algorithm may be trained using stochastic gradient descent (or methods based on stochastic gradient descent or some similar algorithm).

In some embodiments, the algorithm implements at least part of a transmission system comprising a transmitter, a channel and a receiver.

The said means may comprise: at least one processor; and at least one memory including computer program code, the at least one memory and the computer program configured, with the at least one processor, to cause the performance of the apparatus.

In a second aspect, this specification describes a method (e.g. a method of training an algorithm, such as a neural network) comprising: initialising parameters of an algorithm, wherein said algorithm has at least some trainable weights and at least some arithmetic operations; generating or obtaining training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs; generating or obtaining a set of random (e.g. pseudo random) perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm (e.g. all mathematical operations that might introduce an error), wherein said perturbation vectors have a distribution (e.g. a probability density function) depending on properties of a target device implementing said algorithm and depending on the arithmetic operations; updating at least some of the trainable weights of the algorithm based on a loss function; and repeating the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached. The perturbation vector may be random or pseudo-random perturbations within a defined distribution. For example, the perturbation vectors may be evenly distributed within a defined set (wherein the defined set defines the distribution of the perturbation vectors).

The algorithm may comprise a neural network. Alternatively, or in addition, the algorithm may have a plurality of layers (e.g. a plurality of dense layers), each layer

having at least some trainable weights and each layer including one or more arithmetic operations.

Some embodiments comprise determining an error distribution of the target device for use in defining said distribution of said perturbation vectors. The error distribution may be determined by measurement.

Some embodiments comprise quantizing said trainable weights, such that said weights can only take values within a codebook having a finite number of entries that is a subset of the possible values available during updating.

The said loss function may comprise a penalty term related to the quantization of the trainable weights. The said penalty term may include a variable that is adjusted on each repetition of the updating and quantizing such that, on each repetition, more weight is given in the loss function to a difference between the trainable weights and the quantized trainable weights.

The first condition may be met when a performance metric for the algorithm is met. Alternatively, or in addition, the first condition may be met when a performance metric has been unchanged (e.g. unchanged within a margin) for a defined number of iterations. Alternatively, or in addition, the first condition may comprise a defined number of iterations (e.g. a maximum number of iterations).

The said at least some weights of the algorithm may be trained using stochastic gradient descent (or methods based on stochastic gradient descent).

In a third aspect, this specification describes any apparatus configured to perform any method as described with reference to the second aspect.

In a fourth aspect, this specification describes computer-readable instructions which, when executed by computing apparatus, cause the computing apparatus to perform any method as described with reference to the second aspect.

In a fifth aspect, this specification describes a computer program comprising instructions for causing an apparatus to perform at least the following: initialise parameters of an algorithm, wherein said algorithm has at least some trainable weights

and at least some arithmetic operations; generate or obtain training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs; generate or obtain a set of random perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm, wherein said perturbation vectors have a distribution depending on properties of a target device implementing said algorithm and depending on the arithmetic operations; update at least some of the trainable weights of the algorithm based on a loss function; and repeat the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached.

In a sixth aspect, this specification describes a computer-readable medium (such as a non-transitory computer readable medium) comprising program instructions stored thereon for performing at least the following: initialising parameters of an algorithm, wherein said algorithm has at least some trainable weights and at least some arithmetic operations; generating or obtaining training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs; generating or obtaining a set of random (e.g. pseudo random) perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm (e.g. all mathematical operations that might introduce an error), wherein said perturbation vectors have a distribution (e.g. a probability density function) depending on properties of a target device implementing said algorithm and depending on the arithmetic operations; updating at least some of the trainable weights of the algorithm based on a loss function; and repeating the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached.

In a seventh aspect, this specification describes an apparatus comprising: at least one processor; and at least one memory including computer program code which, when executed by the at least one processor, causes the apparatus to: initialise parameters of an algorithm, wherein said algorithm has at least some trainable weights and at least some arithmetic operations; generate or obtain training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs; generate or obtain a set of random perturbation vectors to be applied after at least some of said

arithmetic operations of the algorithm, wherein said perturbation vectors have a distribution depending on properties of a target device implementing said algorithm and depending on the arithmetic operations; update at least some of the trainable weights of the algorithm based on a loss function; and repeat the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached.

In an eighth aspect, this specification describes an apparatus comprising: an initialisation module for initialising parameters of an algorithm, wherein said algorithm having at least some trainable weights and at least some arithmetic operations; a training data module for generating or obtaining training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs; a perturbation vector generating module for generating or obtaining a set of random (e.g. pseudo-random) perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm (e.g. all mathematical operations that might introduce an error, such as due to the use of fixed-point arithmetic), wherein said perturbation vectors have a distribution (e.g. a probability density function) depending on properties of a target device implementing said algorithm and depending on the arithmetic operations; a training module for updating at least some of the trainable weights of the algorithm based on a loss function; and a control module for repeating the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached. The perturbation vector may be random or pseudo-random perturbations within a defined distribution. For example, the perturbation vectors may be evenly distributed within a defined set (wherein the defined set defines the distribution of the perturbation vectors).

Brief description of the drawings

Example embodiments will now be described, by way of non-limiting examples, with reference to the following schematic drawings, in which:

FIG. 1 is a block diagram of a system in accordance with an example embodiment;
FIG. 2 is a block diagram of an example deep neural network;
FIG. 3 is a flow chart showing an algorithm in accordance with an example embodiment;

FIG. 4 is a flow chart showing an algorithm in accordance with an example embodiment;

FIG. 5 is a flow chart showing an algorithm in accordance with an example embodiment;

FIG. 6 is a block diagram of an example communication system in which example embodiments may be implemented;

FIG. 7 is a block diagram of a components of a system in accordance with an example embodiment; and

FIGS. 8A and 8B show tangible media, respectively a removable memory unit and a compact disc (CD) storing computer-readable code which when run by a computer perform operations according to embodiments.

Detailed description

FIG. 1 is a block diagram of a system, indicated generally by the reference numeral 10, in accordance with an example embodiment. The system 10 comprising a training data module 12, an algorithm 14 and a loss function module 16. The algorithm 14 includes a number of trainable weights and implements at least some arithmetic functions. The training data module 12 and the loss function module 16 are used to train the algorithm 14, as described further below.

The training data module 12 provides training data including inputs and corresponding desired outputs for training the algorithm 14. The inputs are provided to the algorithm 14 and the outputs provided to the loss function module 16. The output of the algorithm 14 in response to the inputs received from the training data module 12 is also provided to the loss function module, such that the loss function module can generate an output based on a comparison of actual outputs and expected outputs of the algorithm 14.

The trainable weights of the algorithm 14 are trained based on the output of the loss function module 16, for example using stochastic gradient descent, or some similar approach.

By way of example, the algorithm 14 may be implemented using a neural network, such as a deep neural network.

FIG. 2 shows an example deep neural network 20 comprising a plurality of interconnected nodes arranged in a plurality of layers. As is well known in the art, a

neural network, such as the network 20, can be trained by adjusting the connections between nodes and the relative weights of those connections.

One obstacle to practical hardware implementation of the system 10 is the high memory requirement and computational complexity of the involved algorithm (which may, for example, be implemented using a neural network). Hardware acceleration may be needed to achieve reasonable interference time. However, graphical processing units (GPUs) that can be used to accelerate neural network evaluations come at a high monetary and energy cost that may not be viable in many systems. Moreover, GPUs do not always enable the low inference time requires in many practical applications. These facts may motivate the implementation of algorithms using field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs).

Neural networks are often trained and exploited on computationally powerful platforms with general processing units (GPU) acceleration, supporting high precision floating point arithmetic (e.g. 32-bit or 64-bit floating point arithmetic). Such hardware may not be available for many applications. Accordingly, the neural networks implementing the algorithm 14 of the system 10 described above may be compressed to meet practical constraints. This may be achieved by using a more compact representation of neural network parameters, at the cost of reduced precision. For example, as described further below, compression of weights and/or biases of the neural networks may be achieved through quantization, such that the weights are forced to take values within a codebook with a finite number of entries (e.g. at a lower precision that that provided by a training module). Thus, the capabilities of the systems used for training an algorithm (such as the algorithm 14) may be different to the capabilities of the algorithm in use.

FIG. 3 is a flow chart showing an algorithm, indicated generally by the reference numeral 30, in accordance with an example embodiment.

The algorithm 30 starts at operation 31, where the algorithm 14 described above is trained. For example, the algorithm 14 may be a neural network that is trained in a supervised manner using a stochastic gradient descent (SGD) algorithm.

At operation 32, parameters of the algorithm 14 are quantized. The algorithm 14 can then be implemented using the quantized parameters. This approach can significantly

reduce memory requirements within the algorithm 14 and the complexity of arithmetic operations implemented by the algorithm 14, but typically results in reduced precision.

Example embodiments are described below with reference to a neural network including a number of dense layers (such as the neural network 20 referred to above). It should be noted that this is one example implementation and that other embodiments are possible.

A dense layer may be made of $U \geq 1$ units, each implementing the following equation:

$$a_u = \sigma\left(\sum_{i=1}^{I} w_{u,i}x_i + b_u\right), \; u = 1, \ldots, U \qquad (1)$$

where:

a_u is the u^{th} activation (i.e. the output of the u^{th} unit)

I is the dimension of the input vector x,

$w_{u,i}$ are layer weights

b_u is a bias

σ is an activation function (e.g. ReLu, sigmoid etc.).

The basic arithmetic operations required by the implementation of such as layer typically include addition, multiplication and an activation function (e.g. identity, ReLu etc.).

Note that standard layers of a neural network implementation (e.g. dense, convolutional or recurrent layers) require only simple arithmetic operations such as the addition, multiplication and activation functions referred to above. When performing inference on an erroneous arithmetic (e.g. erroneous due to quantization errors), each operation potentially introduces an error. For example, when using fixed point arithmetic with K_I bits for the integer part, K_F bits for the fractional part, and one sign bit, only real scalars that can be written as:

$$z = (-1)^{b_s} \left(\sum_{i=0}^{K_I - 1} b_i 2^i + \sum_{j=1}^{K_F} b_{-j} 2^{-j} \right) \quad (2)$$

where the sign bit b_s and b_i take values $\{0, 1\}$ can be represented without errors. These numbers form a finite set called a codebook, which is denoted by $\mathcal{C}$ herein. Any real number not in the codebook, for example resulting from the multiplication of two numbers within the codebook, will only be approximated by a number from the codebook, leading to the previously mentioned errors.

Assuming one has some knowledge of the targeted hardware and arithmetic, it may be possible to define a quantization function Q, which maps each real number to an element from the codebook $\mathcal{C}$ in order to simulate the targeted hardware arithmetic. With the knowledge of Q, one could define the neural network, at training, so that the neural network behaves as the targeted hardware. For example, considering a dense layer defined in equation (1) above, the neural network could be implemented in training as:

$$a_u = \sigma \left(\sum_{i=1}^{I} \mathcal{Q}(w_{u,i} x_i) + b_u \right) \quad (3)$$

assuming only multiplications cause errors.

However, a drawback with this approach is that Q is typically not differentiable. Thus, it may not be possible to train a neural network trained in this way using typical deep learning algorithms, such as stochastic gradient descent.

One further approach is to consider the quantization error:

$$e = \mathcal{Q}(z) - z \quad (4)$$

for a given real number z and relax it to a random perturbation. Therefore, in the implementation of a neural network during training, a random perturbation can be added to the result of each operation that can potentially cause errors (e.g. multiplications). As an example, a dense layer as defined in (1) above may be implemented in training as:

$$a_u = \sigma\left(\sum_{i=1}^{I}(w_{u,i}x_i + p_i) + b_u\right) \quad (5)$$

where p_i is the random perturbation, and assuming that only multiplications cause errors. As discussed further below, such an approach provides a differentiable error function.

As an additional example, and under the same assumptions, a 2-dimensional convolutional layer may be implemented at training as:

$$Y_{i,j,l} = \sigma\left(\sum_{m=-M}^{M}\sum_{n=-N}^{N}\sum_{q=1}^{Q}\left(W^l_{m,n,q}X_{i+m,j+n,q} + P^l_{i+,j+n,q}\right)\right) \quad (6)$$

where $Z_{i,j,l}$ denotes the (i,j,l) component of a 3-D tensor $\mathbf{Z}$. $Y_{i,j,l}$ is the output of the l^{th} filter with co-ordinate (i,j), e.g. a pixel in a picture, M and N are the filter width and length, and Q is the number of filters in the layer's input. $\mathbf{X}$ is the layer's input and $\boldsymbol{W}^l$ is the weights of the filter of the l^{th} layers. Finally, $\boldsymbol{P}^l$ is the added perturbation.

Regarding non-trivially implemented activation functions, such as hyperbolic tangent and sigmoid, these are often implemented by piece-wise linear approximations, for which perturbations can also be added for increased robustness.

Since a neural network obtained using the principles of random perturbation outlined above is differentiable, such a neural network can be training using stochastic gradient descent (SGD) or some similar methodology, as follows.

Let's denote by $f_{\theta,p}: \mathbb{R}^N \mapsto \mathbb{R}^M$ the mapping defined by a neural network with training parameters θ, and where $\mathbf{p}$ is the vector of perturbations added to the operations that generate errors on the targeted hardware. If the targeted hardware is assumed to not be prone to arithmetic errors, then $\mathbf{p}$ can be set to the null vector. Let's define by $l: \mathbb{R}^N \times \mathbb{R}^M \mapsto \mathbb{R}$ the per-example loss function. Then, the loss function can be defined as:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})}\left\{\mathbb{E}_{\mathbf{p}}\left\{l\left(f_{\boldsymbol{\theta},\mathbf{p}}(\mathbf{x}),\mathbf{y}\right)\right\}\right\} \quad (7)$$

where the first expectation is taken over the input-output pairs used for training, and the second expectation over the perturbations.

FIG. 4 is a flow chart showing an algorithm, indicated generally by the reference numeral 40, in accordance with an example embodiment. The algorithm 40 shows an example training method for an algorithm (such as a neural network) having at least some trainable weights, wherein the algorithm is to be implemented on a target device (such as an application-specific integrated circuit or a field-programmable gate array).

The algorithm 40 starts at operation 41 where parameters of the algorithm 14 are initialised. The initialised parameters may be set to θ_0 and a variable j (if used) set to 0. The training parameters θ may be initialised randomly.

At operation 42, training data is obtained. The training data may include a number of inputs together with target desired outputs of the algorithm in response to said inputs. Thus, for example, a set of N input-output pairs may be generated $\{(x_i, y_i), i = 1, \ldots N\}$. For example, the input-output pairs may be generated randomly.

At operation 43, a set of N perturbation vectors is generated (e.g. randomly) $\{\boldsymbol{p}_i, i = 1, \ldots N\}$. As discussed further below, the perturbation vectors may be applied after at least some arithmetic operations of the algorithm being trained. The perturbation vectors may be random (e.g. random within a defined distribution). The perturbation vectors may have a distribution depending on properties of a target device implementing said algorithm and/or depending on the arithmetic operations of the algorithm.

At operation 44, a loss function estimate is generated based on outputs y_i of the input-output pairs and estimated outputs $\{\hat{\mathbf{y}}_i = f_{\boldsymbol{\theta}_j,\mathbf{p}_i}(\mathbf{x}_i), i = 1, \ldots, N\}$, where the loss function estimate is given by:

$$\hat{L} = \frac{1}{N}\sum_{i=1}^{N} l\left(\hat{\mathbf{y}}_i, \mathbf{y}_i\right) \quad (8)$$

At operation 45, parameters of the algorithm (e.g. a neural network) are updated based on the loss function (e.g. using stochastic gradient descent or some similar process).

At operation 46, it is determined whether the algorithm 40 is complete (e.g. by determining whether a first condition has been reached). If so, the algorithm terminates at operation 47. If not, the variable j (if used) is incremented and the algorithm returns to operation 42, and the training processes repeated.

The operation 46 may be implemented in many ways. For example, the algorithm 40 may be deemed to complete when a defined number of iterations (e.g. indicated by the variable j) have been carried out. Other conditions for terminating the algorithm 40 are possible, such as if performance metric has been met or that a performance metric has been unchanged (e.g. within a margin) for a defined number of iterations, suggesting that the algorithm has settled. Of course, multiple metrics could be considered in combination.

The algorithm 40 may be used to train a neural network (or some other algorithm) to be more robust to arithmetic errors, and therefore more suitable to the hardware on which it is deployed. As noted above, the hardware may be an ASIC or an FPGA, although this is not essential to all embodiments. Moreover, the target hardware may implement a fixed-point arithmetic and may, for example, be training using a floating-point arithmetic.

The choice of a probability density function (pdf) from which the perturbation of operation 43 is drawn may depend on a number of factors, such as a quantization function and assumptions on the potential errors. By way of example, consider a fixed point arithmetic system with K_E bits for the integer part and K_F bits for the fractional part and with the following assumptions:

- Quantization is implemented by truncating extra bits
- No overflow occurs. (This is the case if K_E is large enough.)
- The "error bits" (i.e. the bits that are truncated) are assumed to be independent and identically distributed and drawn with equal probabilities.

In this case, the error is uniformly distributed on $[-2^{K_F}, 0]$. Note that the probability density function (pdf) could be different for different layers of a neural network (e.g. if different resolutions are used in each layer).

In addition to making an algorithm (such as a neural network) robust to errors (such as arithmetic errors), one may want to compress the algorithm. This may be achieved, for example, by quantizing the weights of the algorithm itself (i.e. by forcing the weights to take values in a finite codebook $\mathcal{W}$). Note that, in general the codebook $\mathcal{W}$ is different to the codebook $\mathcal{C}$ discussed above.

Quantizing the weights can significant reduce the complexity of an algorithm. For example, by forcing the weights to take values within the codebook $\{-1,0,1\}$, all multiplications can be reduced to zeroing or a sign change. Similarly, by forcing the weights to take values within the codebook of powers of two $\{2^n | n \in \mathbb{Z}\}$, all multiplications can be reduced to bit shifts.

FIG. 5 is a flow chart showing an algorithm, indicated generally by the reference numeral 50, in accordance with an example embodiment. The algorithm 50 combines the teaching described above with learning-compression principles.

The algorithm 50 starts at operation 51, where an algorithm (such as the algorithm 14) is initialised, thereby providing a means for initialising parameters of an algorithm.

Operations 52 to 54 implement a learning operation 52, a compression operation 53 and a parameter updating operation 54 that collectively implement a learning-compression algorithm, as discussed further below. At operation 55, it is determined whether or not the learning-compression algorithm is complete. If not, the operations 52 to 54 are repeated. When the operation 55 determines that the learning-compression algorithm is complete (e.g. if a sufficient number of iterations of the operations 52 to 54 have been completed or if a given performance threshold has been met), the algorithm 50 terminates at operation 56.

The learning operation 52 adjusts the weights of the algorithm 14 (e.g. neural networks) in the non-quantized space. The operation 52 may be similar to the algorithm 40 described above, but includes an additional penalty term, related to the allowed codebook values in the quantized space. Thus, the learning operation 52 implements a

means for updating trainable parameters of the transmission system based on a loss function.

More specifically, we denote by $\mathcal{L}$ the cross-entropy, which is the loss function minimized during training defined by:

$$\mathcal{L}(\theta) = \mathbb{E}_s[-\log([\boldsymbol{p}]_s)]$$

The learning operation 52 solves the following optimization problem:

$$\theta = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\theta) + \frac{\mu}{2}\left\|\theta - \theta_q - \frac{1}{\mu}\lambda\right\|^2$$

Where:

- μ is a parameter which increases as the training progresses (such that $\frac{1}{\mu}$ reduces);
- θ_q is the projection of the solution of the previous learning step on the codebook; and
- λ are Lagrange multiplier estimates.

At compression operation 53, a compression process is carried out in which the adjusted weights are quantized. Thus, the compression operation 53 implements a means for quantizing trainable parameters of the transmission system.

More specifically, the compression operation 53 may perform the element-wise quantization of $\theta - \frac{1}{\mu}\lambda$:

$$\theta_q = \Pi(\theta - \frac{1}{\mu}\lambda)$$

Where θ was computed during the learning operation 52. We denote the codebook by $\mathcal{C}$. The quantizing the scalar $\theta - \frac{1}{\mu}\lambda$ can be expressed as:

$$\theta_j = \Pi(\theta - \frac{1}{\mu}\lambda) \triangleq \underset{c\in\mathcal{C}}{\operatorname{argmin}} \left|\theta_j - \frac{1}{\mu}\lambda - c\right|$$

Which simply means to find the closest element within the quantization codebook $\mathcal{C}$.

At operation 54, various parameters of the algorithm 14 are updated, as discussed further below.

Thus, the algorithm 50 iterates between the learning operation 52 and the compression operation 53. At the end of each operation, the Lagrange multiplier elements are updated. The algorithm 50 can be expressed mathematically as set out below.

The initialisation operation 51 may comprise:

1. $\theta^{(0)} \leftarrow \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta)$
2. ${\theta_q}^{(0)} \leftarrow \Pi(\theta^{(0)})$
3. $\lambda^{(0)} \leftarrow 0$
4. Initialize $\mu^{(0)}$ with a positive value
5. $i \leftarrow 1$

The learning operation 52 may comprise:

1. $\theta^{(i)} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta) + \frac{\mu^{(i-1)}}{2} \left\| \theta - {\theta_q}^{(i-1)} - \frac{1}{\mu^{(i-1)}} \lambda^{(i-1)} \right\|^2$

The compression operation 53 may comprise:

1. ${\theta_q}^{(i)} = \Pi(\theta^{(i)} - \frac{1}{\mu^{(i-1)}} \lambda)$

The update parameters operation 54 may comprise:

1. Update Lagrange multiplier estimates:

   $\lambda^{(i)} \leftarrow \lambda^{(i-1)} - \frac{1}{\mu^{(i-1)}} (\theta^{(i)} - {\theta_q}^{(i)})$
2. Set $\mu^{(i)}$ such that $\mu^{(i)} > \mu^{(i+1)}$
3. $i \leftarrow i + 1$

The complete operation 55 may comprise:

1. If $\left\| \theta^i - {\theta_q}^{(i)} \right\|$ is small enough (i.e. below a threshold), then proceed to operation 56, otherwise return to operation 52.

However, the criterion for the complete operation 55 could be implemented in other ways.

The algorithm 50 converges to a local solution as $\mu^{(i)} \rightarrow \infty$. This could be achieved, for example, by following a multiplicative schedule $\mu^{(i)} \leftarrow \mu^{(i-1)}a$, where $a > 1$ and $\mu^{(0)}$ are parameters of the algorithm.

It should be noted that the sequence $\mu^{(i)}$ can be generated in many ways. Using a multiplicative schedule (as discussed above), the initial value of $\mu^{(0)}$, as well as a are optimisation parameters (and may be optimised as part of the training operation 52 described above).

It should also be noted that the batch size N as well as the learning rate (and possibly other parameters of the chosen SGD variant, e,g, ADAM, RMSProp, Momentum) could be optimization parameters of the algorithms 40 and 50 described above.

The principles described herein may be used many applications (e.g. relating to many example algorithms 14). By way of example, the principles described herein may be used in a communication system.

FIG. 6 is a block diagram of an example communication system, indicated generally by the reference numeral 60, in which example embodiments may be implemented. The system 60 includes a transmitter 62, a channel 63 and a receiver 64. Viewed at a system level, the system 60 converts an input symbol (s) (also called a message) received at the input to the transmitter 62 into an output symbol ($\hat{s}$) at the output of the receiver 64.

The transmitter 62 implements a transmitter algorithm. Similarly, the receiver 64 implements a receiver algorithm. The algorithms of the transmitter 62 and the receiver 64 may be trained using the principles described herein in order to optimise the performance of the system 60 as a whole.

The transmitter 62 may include a dense layer of one or more units 70, 71 (e.g. including one or more neural networks) and a normalization module 72. The dense layers 70, 71 may include an embedding module. The modules within the transmitter 62 are provided by way of example and modifications are possible.

Similarly, the receiver 64 may include a dense layer of one or more units 74, 75 (e.g. including one or more neural networks), a softmax module 76 and an arg max module 77. The output of the softmax module is a probability vector that is provided to the input of an arg max module 77. The modules within the receiver 64 are provided by way of example and modifications are possible.

The system 60 therefore provides an autoencoder implementing an end-to-end communication system. The autoencoder can be trained with respect to an arbitrary loss function that is relevant for a certain performance metric, such as block error rate (BLER). (The terms 'autoencoder' and 'communication system' can both be used to describe the system 60.)

As noted above, one obstacle to practical hardware implementation of systems (such as the communication system 60) is the high memory requirement and computational complexity of the involved neural networks. Hardware acceleration may be provided to achieve reasonable inference time. However, graphical processing units (GPUs) that can be used to accelerate neural network evaluations come at a high monetary and energy cost that may not be viable in many communication systems.

By way of example, neural networks (or some other parametric algorithm) may be trained and exploited on computationally powerful platforms with graphic processing units (GPU) acceleration, supporting high precision floating point arithmetic (e.g. 32-bit or 64-bit floating point arithmetic). Such hardware may not be available for some systems (such as some implementations of the communication system 60). Accordingly, such neural networks may be compressed to meet practical constraints. This may be achieved by using a more compact representation of neural network parameters, at the cost of reduced precision. For example, as described further above, compression of weights and/or biases of the neural networks may be achieved through quantization, such that the weights are forced to take values within a codebook with a finite number of entries (e.g. at a lower precision that that provided by a training module). In extreme cases, each weight may take a binary value (e.g. either -1 or +1).

The principles described herein may therefore be used to train a more robust communication system.

For completeness, FIG. 7 is a schematic diagram of components of one or more of the modules described previously (e.g. the transmitter or receiver neural networks), which hereafter are referred to generically as processing systems 110. A processing system 110 may have a processor 112, a memory 114 closely coupled to the processor and comprised of a RAM 124 and ROM 122, and, optionally, hardware keys 120 and a display 128. The processing system 110 may comprise one or more network interfaces 118 for connection to a network, e.g. a modem which may be wired or wireless.

The processor 112 is connected to each of the other components in order to control operation thereof.

The memory 114 may comprise a non-volatile memory, a hard disk drive (HDD) or a solid state drive (SSD). The ROM 122 of the memory 114 stores, amongst other things, an operating system 125 and may store software applications 126. The RAM 124 of the memory 114 is used by the processor 112 for the temporary storage of data. The operating system 125 may contain code which, when executed by the processor, implements aspects of the algorithms 30, 40 and 50.

The processor 112 may take any suitable form. For instance, it may be a microcontroller, plural microcontrollers, a processor, or plural processors.

The processing system 110 may be a standalone computer, a server, a console, or a network thereof.

In some embodiments, the processing system 110 may also be associated with external software applications. These may be applications stored on a remote server device and may run partly or exclusively on the remote server device. These applications may be termed cloud-hosted applications. The processing system 110 may be in communication with the remote server device in order to utilize the software application stored there.

FIGS. 8A and 8B show tangible media, respectively a removable memory unit 165 and a compact disc (CD) 168, storing computer-readable code which when run by a computer may perform methods according to embodiments described above. The removable memory unit 165 may be a memory stick, e.g. a USB memory stick, having internal memory 166 storing the computer-readable code. The memory 166 may be accessed by

a computer system via a connector 167. The CD 168 may be a CD-ROM or a DVD or similar. Other forms of tangible storage media may be used.

Embodiments of the present invention may be implemented in software, hardware, application logic or a combination of software, hardware and application logic. The software, application logic and/or hardware may reside on memory, or any computer media. In an example embodiment, the application logic, software or an instruction set is maintained on any one of various conventional computer-readable media. In the context of this document, a "memory" or "computer-readable medium" may be any non-transitory media or means that can contain, store, communicate, propagate or transport the instructions for use by or in connection with an instruction execution system, apparatus, or device, such as a computer.

Reference to, where relevant, "computer-readable storage medium", "computer program product", "tangibly embodied computer program" etc., or a "processor" or "processing circuitry" etc. should be understood to encompass not only computers having differing architectures such as single/multi-processor architectures and sequencers/parallel architectures, but also specialised circuits such as field programmable gate arrays FPGA, application specify circuits ASIC, signal processing devices and other devices. References to computer program, instructions, code etc. should be understood to express software for a programmable processor firmware such as the programmable content of a hardware device as instructions for a processor or configured or configuration settings for a fixed function device, gate array, programmable logic device, etc.

As used in this application, the term "circuitry" refers to all of the following: (a) hardware-only circuit implementations (such as implementations in only analogue and/or digital circuitry) and (b) to combinations of circuits and software (and/or firmware), such as (as applicable): (i) to a combination of processor(s) or (ii) to portions of processor(s)/software (including digital signal processor(s)), software, and memory(ies) that work together to cause an apparatus, such as a server, to perform various functions) and (c) to circuits, such as a microprocessor(s) or a portion of a microprocessor(s), that require software or firmware for operation, even if the software or firmware is not physically present.

If desired, the different functions discussed herein may be performed in a different order and/or concurrently with each other. Furthermore, if desired, one or more of the above-described functions may be optional or may be combined. Similarly, it will also be appreciated that the flow diagram of FIGS. 3 to 5 are examples only and that various operations depicted therein may be omitted, reordered and/or combined.

The example embodiments have generally been described above with reference to populations of neural networks. This is not essential to all embodiments. For example, other forms of algorithms (such as a parametric algorithm) may be used. Moreover, although example embodiments are described in which trainable parameters of the algorithm (e.g. parameters of a neural network) are updated, this is not essential to all embodiments. For example, a population of algorithms may be updated by updating one or more parameters and/or one or more structures of one or more of the algorithms of a population.

It will be appreciated that the above described example embodiments are purely illustrative and are not limiting on the scope of the invention. Other variations and modifications will be apparent to persons skilled in the art upon reading the present specification.

Moreover, the disclosure of the present application should be understood to include any novel features or any novel combination of features either explicitly or implicitly disclosed herein or any generalization thereof and during the prosecution of the present application or of any application derived therefrom, new claims may be formulated to cover any such features and/or combination of such features.

Although various aspects of the invention are set out in the independent claims, other aspects of the invention comprise other combinations of features from the described embodiments and/or the dependent claims with the features of the independent claims, and not solely the combinations explicitly set out in the claims.

It is also noted herein that while the above describes various examples, these descriptions should not be viewed in a limiting sense. Rather, there are several variations and modifications which may be made without departing from the scope of the present invention as defined in the appended claims.

Claims

1. An apparatus comprising:

means for initialising parameters of an algorithm, wherein said algorithm having at least some trainable weights and at least some arithmetic operations;

means for generating or obtaining training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs;

means for generating or obtaining a set of random perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm, wherein said perturbation vectors have a distribution depending on properties of a target device implementing said algorithm and depending on the arithmetic operations;

means for updating at least some of the trainable weights of the algorithm based on a loss function; and

means for repeating the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached.

2. An apparatus as claimed in claim 1, wherein said perturbation vectors are evenly distributed within a defined set.

3. An apparatus as claimed in claim 1 or claim 2, wherein said algorithm comprises a neural network.

4. An apparatus as claimed in any one of claims 1 to 3, wherein the algorithm has a plurality of layers, each layer having at least some trainable weights and each layer including one or more arithmetic operations.

5. An apparatus as claimed in any one of the preceding claims, wherein the arithmetic operations include multiplications.

6. An apparatus as claimed in any one of the preceding claims, further comprising means for determining an error distribution of the target device for use in defining said distribution of said perturbation vectors.

7. An apparatus as claimed in any one of the preceding claims, wherein said target device implements a fixed-point arithmetic.

8. An apparatus as claimed in any one of the preceding claims, further comprising means for quantizing said trainable weights, such that said weights can only take values within a codebook having a finite number of entries that is a subset of the possible values available during updating.

9. An apparatus as claimed in claim 8, wherein said means for repeating comprises means for repeating the generation or obtaining of training data, the generating or obtaining of random perturbation vectors, the updating of said trainable weights and the quantizing of said trainable weights until the first condition is reached.

10. An apparatus as claimed in claim 8 or claim 9, wherein the loss function comprises a penalty term related to the quantization of the trainable weights.

11. An apparatus as claimed in claim 10, wherein said penalty term includes a variable that is adjusted on each repetition of the updating and quantizing such that, on each repetition, more weight is given in the loss function to a difference between the trainable weights and the quantized trainable weights.

12. An apparatus as claimed in any one of the preceding claims, wherein the first condition is met when a performance metric for the algorithm is met.

13. An apparatus as claimed in any one of the preceding claims, wherein the first condition is met when a performance metric has been unchanged for a defined number of iterations.

14. An apparatus as claimed in any one of the preceding claims, wherein the first condition comprises a defined number of iterations.

15. An apparatus as claimed in any one of the preceding claims, wherein the loss function is related to one or more of mean squared error, block error rate and categorical cross-entropy.

16. An apparatus as claimed in any one of the preceding claims, wherein the algorithm implements at least part of a transmission system comprising a transmitter, a channel and a receiver.

17. An apparatus as claimed in any one of the preceding claims, wherein the means comprise:

at least one processor; and

at least one memory including computer program code, the at least one memory and the computer program configured, with the at least one processor, to cause the performance of the apparatus.

18. A method comprising:

initialising parameters of an algorithm, wherein said algorithm has at least some trainable weights and at least some arithmetic operations;

generating or obtaining training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs;

generating or obtaining a set of random perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm, wherein said perturbation vectors have a distribution depending on properties of a target device implementing said algorithm and depending on the arithmetic operations;

updating at least some of the trainable weights of the algorithm based on a loss function; and

repeating the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached.

19. A computer program comprising instructions for causing an apparatus to perform at least the following:

initialise parameters of an algorithm, wherein said algorithm has at least some trainable weights and at least some arithmetic operations;

generate or obtain training data comprising inputs and desired target outputs for training said algorithm, wherein the target outputs are expected outputs of the algorithm in response to the respective inputs;

generate or obtain a set of random perturbation vectors to be applied after at least some of said arithmetic operations of the algorithm, wherein said perturbation vectors have a distribution depending on properties of a target device implementing said algorithm and depending on the arithmetic operations;

update at least some of the trainable weights of the algorithm based on a loss function; and

repeat the generation or obtaining of training data, generation or obtaining of random perturbation vectors and updating of said trainable weights until a first condition is reached.

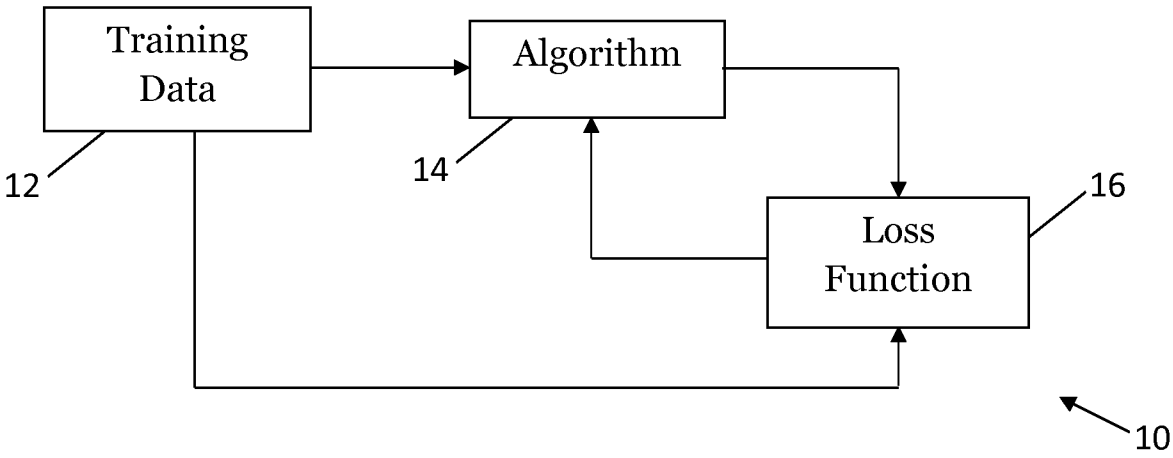

Fig. 1

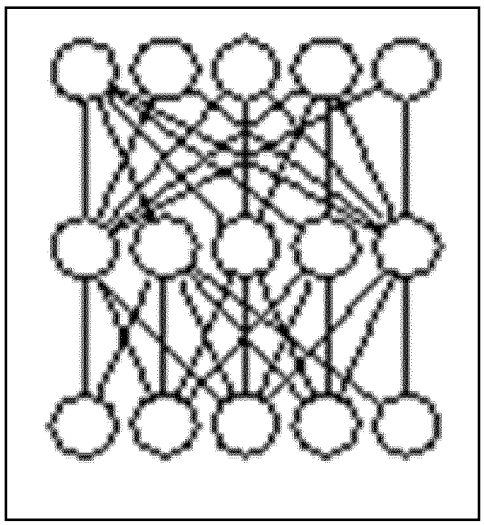

Fig. 2

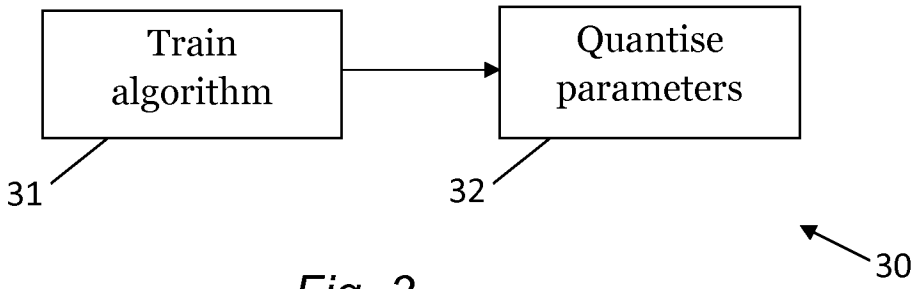
Train algorithm
Quantise parameters
31
32
30

Fig. 3

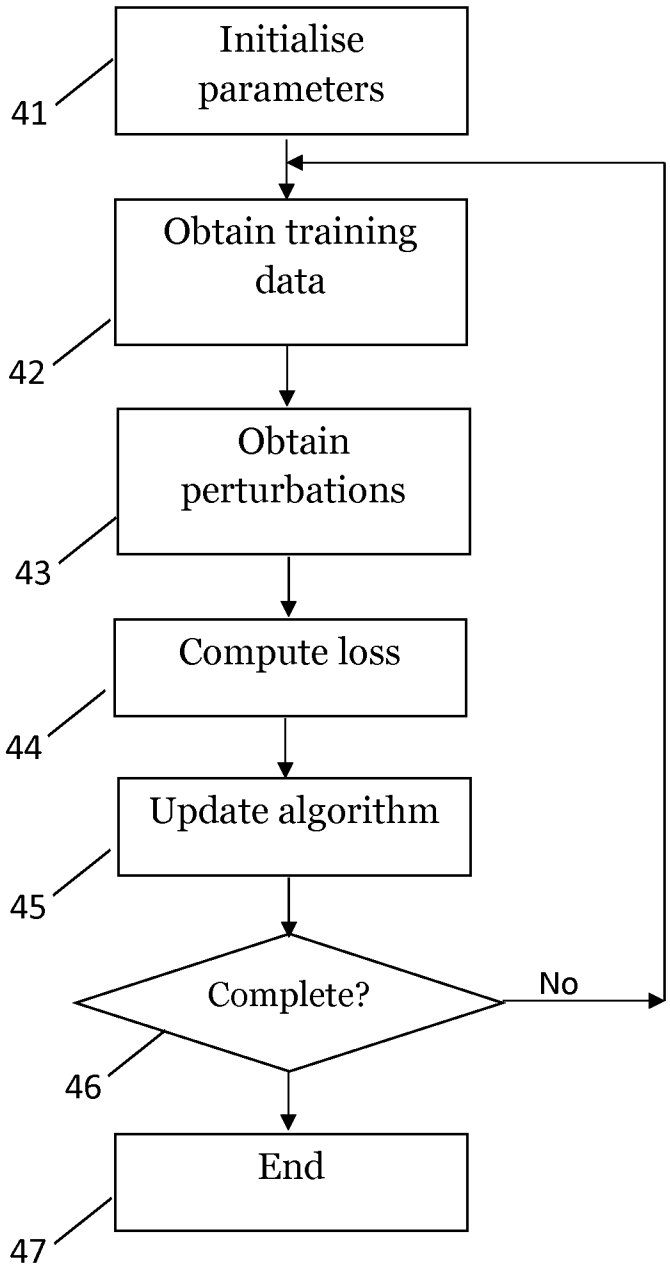
Initialise parameters
41
Obtain training data
42
Obtain perturbations
43
Compute loss
44
Update algorithm
45
Complete?
No
46
End
47
40

Fig. 4

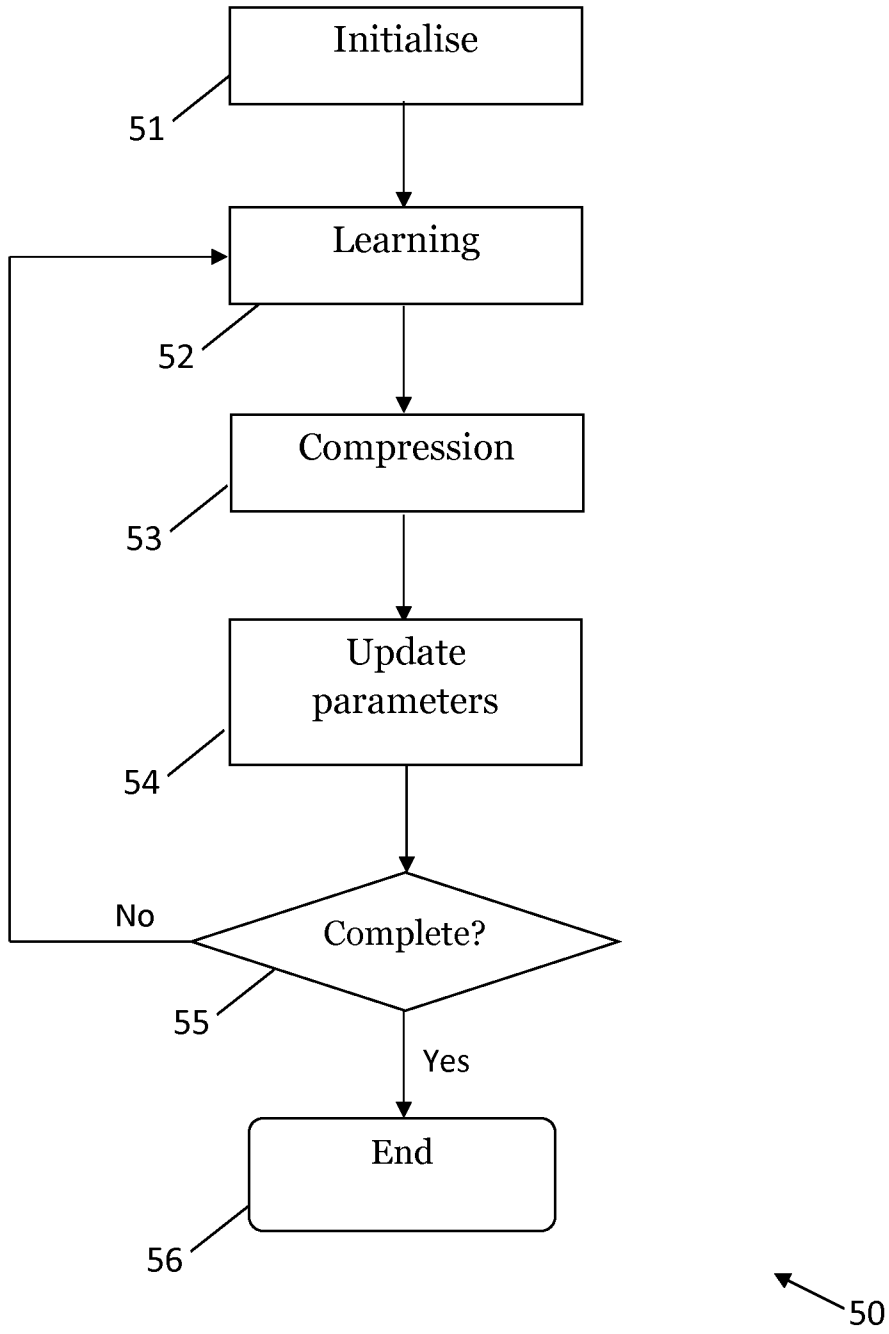
Initialise
51
Learning
52
Compression
53
Update parameters
54
Complete?
No
55
Yes
End
56
50

Fig. 5

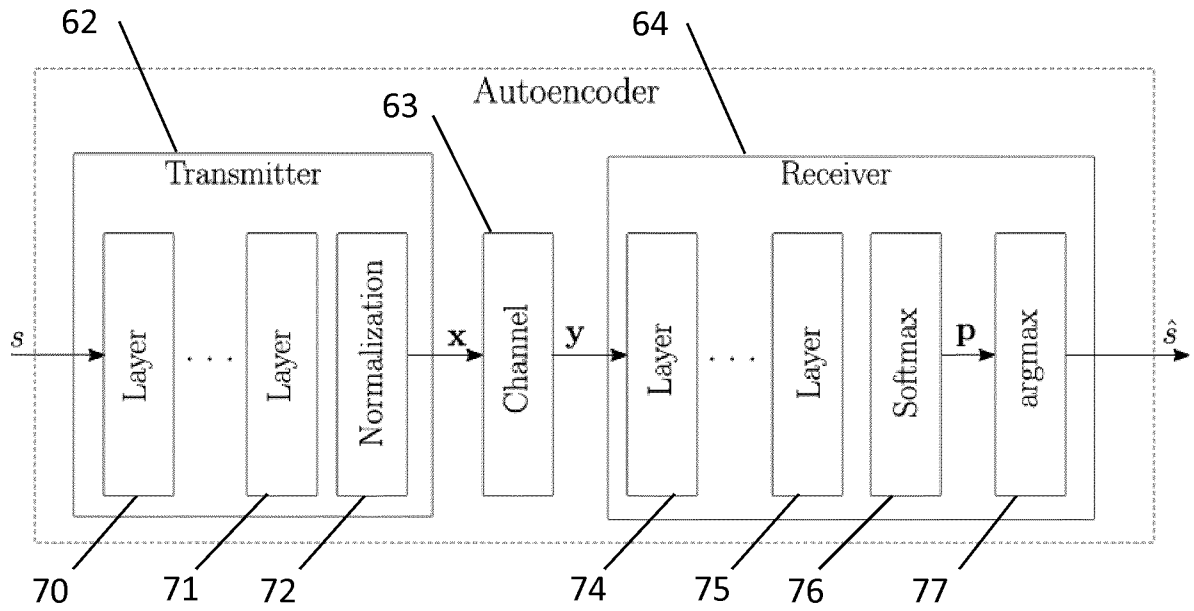
62
64
Autoencoder
63
Transmitter
Receiver
s
Layer
Layer
Normalization
x
Channel
y
Layer
Layer
Softmax
p
argmax
ŝ
70
71
72
74
75
76
77
60

Fig. 6

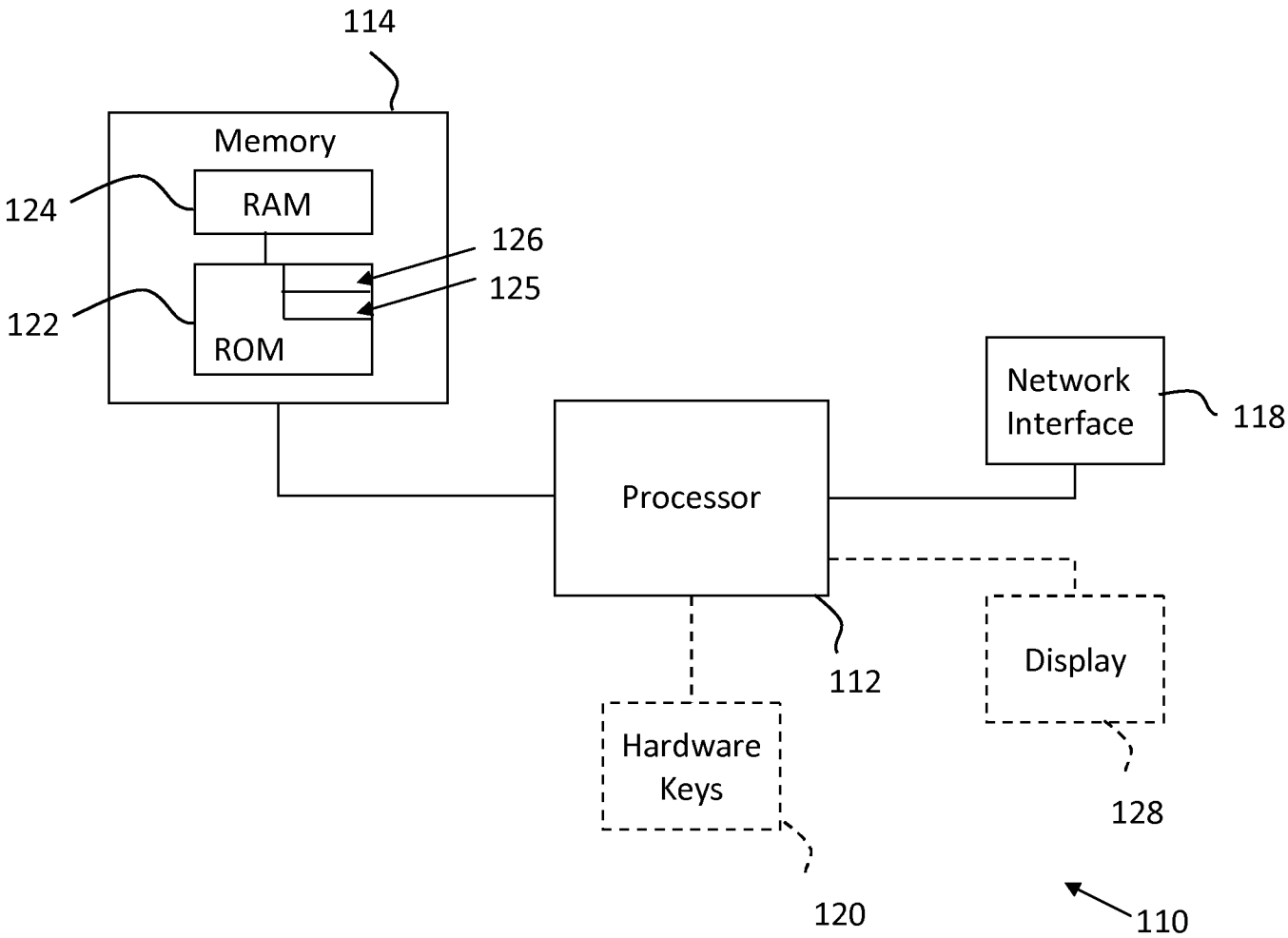
114
Memory
124
RAM
126
125
122
ROM
Network Interface
118
Processor
112
Display
Hardware Keys
128
120
110

Fig. 7

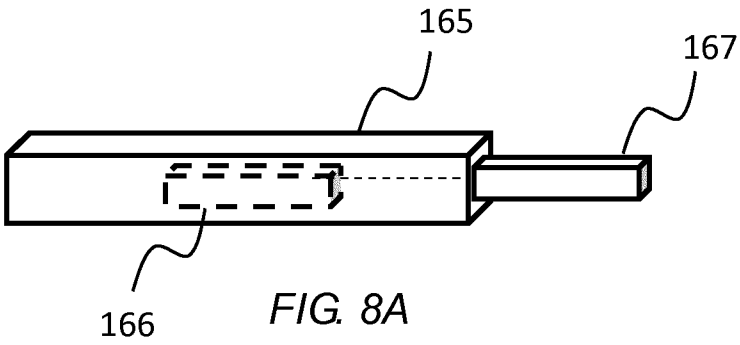

FIG. 8A

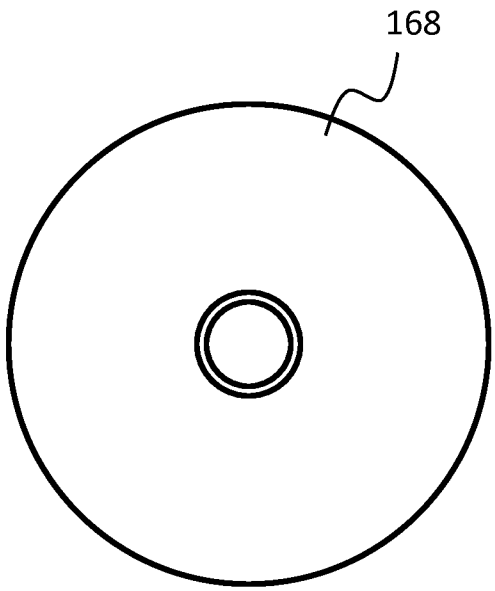

FIG. 8B

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2019/055537

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06N3/063 G06N3/04 G06N3/08
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CHAIM BASKIN ET AL: "NICE: Noise Injection and Clamping Estimation for Neural Network Quantization", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 29 September 2018 (2018-09-29), XP081422330, the whole document ----- -/--	1-19

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 November 2019	06/12/2019

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Moro Pérez, Gonzalo

2

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Chaim J Baskin ET AL: "UNIQ: Uniform Noise Injection for Non-uniform Quantization of Neural Networks", , 2 October 2018 (2018-10-02), XP055641141, DOI: 10.1016/j.addbeh.2016.11.010 Retrieved from the Internet: URL:https://arxiv.org/pdf/1804.10969.pdf [retrieved on 2019-11-11] the whole document	1-19
A	FAYCAL AIT AOUDIA ET AL: "Towards Hardware Implementation of Neural Network-based Communication Algorithms", 2019 IEEE 20TH INTERNATIONAL WORKSHOP ON SIGNAL PROCESSING ADVANCES IN WIRELESS COMMUNICATIONS (SPAWC), 19 February 2019 (2019-02-19), pages 1-5, XP055641018, DOI: 10.1109/SPAWC.2019.8815398 ISBN: 978-1-5386-6528-2 the whole document	1-19
A	US 2016/328647 A1 (LIN DEXU [US] ET AL) 10 November 2016 (2016-11-10) paragraph [0002] - paragraph [0014] paragraph [0077] - paragraph [0087]; figures 4, 5	1-19
A	Darryl D Lin ET AL: "Fixed Point Quantization of Deep Convolutional Networks", , 2 June 2016 (2016-06-02), pages 1-10, XP055561866, Retrieved from the Internet: URL:https://arxiv.org/pdf/1511.06393.pdf [retrieved on 2019-02-26] the whole document	1-19
A	HAO LI ET AL: "Training Quantized Nets: A Deeper Understanding", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 7 June 2017 (2017-06-07), XP081281619, the whole document	1-19

2

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2019/055537

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2016328647 A1	10-11-2016	CN 107646116 A	30-01-2018
		EP 3295382 A1	21-03-2018
		US 2016328647 A1	10-11-2016
		WO 2016182659 A1	17-11-2016