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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 December 2018 (13.12.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/223330 A1

(51) International Patent Classification:
G06F 19/24 (2011.01)

(21) International Application Number:
PCT/CN2017/087570

(22) International Filing Date:
08 June 2017 (08.06.2017)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **ALIBABA GROUP HOLDING LIMITED** [—/CN]; Fourth Floor, One Capital Place, P.O. Box 847, George Town, Grand Cayman (KY).

(72) Inventors: **ZOU, Youzhi**; 383 King Street, Apt 1215, San Francisco, California 94158 (US). **ZHANG, Jiaxing**; Building B, Huanglong Times Plaza, No.18 Wantang Road, Xihu District, Hangzhou, Zhejiang (CN). **CUI, Xiaoyuan**; City Center Bellevue, Suite 800, 500 108th Ave NE, Bellevue WA 98004 (US). **LI, Xiaolong**; 229 216th PL SE Sammamish, WA 98074 (US). **YUAN, Qi**; Building B, Huanglong Times Plaza, No.18 Wantang Road, Xihu District, Hangzhou, Zhejiang (CN).

(74) Agent: **BEIJING TSINGYUANHUI INTELLECTUAL PROPERTY LAW FIRM**; Suzhou Street No. 362, Haidian District, Beijing 100080 (CN).

(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, 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).

(54) Title: METHOD AND APPARATUS FOR DISTRIBUTED MACHINE LEARNING SYSTEM

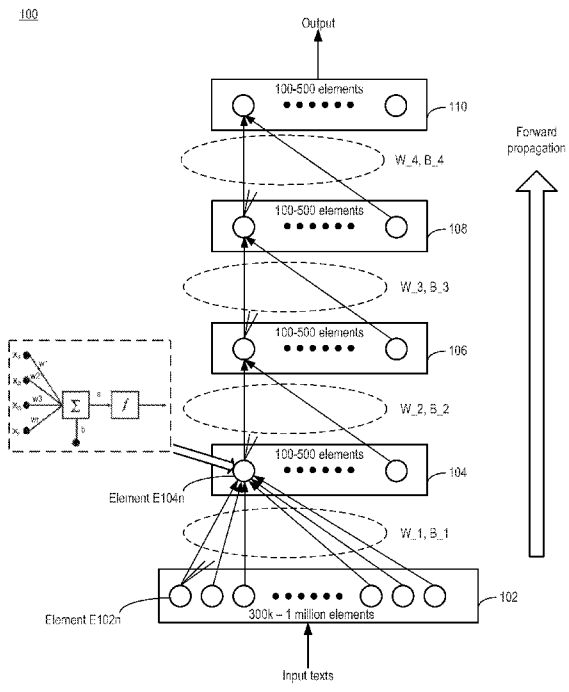

FIG. 1A

(57) Abstract: A server for facilitating dependency-based execution of a set of machine learning tasks is provided. The server stores data representing storage elements in a machine learning model and connections between the storage elements. The server also comprises a task module configured to receive processing assignments for some of the storage elements, and determine a set of machine learning tasks and dependencies among the set of machine learning. The server also comprises a scheduler module configured to determine a resolution status for each of the dependencies, and determine, based on the resolution status, a sequence of execution of the set of machine learning tasks. The server also comprises one or more hardware processor configured to execute a set of machine learning tasks based on the determined sequence.

WO 2018/223330 A1

[Continued on next page]

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

METHOD AND APPARATUS FOR DISTRIBUTED MACHINE LEARNING SYSTEM

TECHNICAL FIELD

[001] The present disclosure generally relates to the field of computer software, and more particularly, to a method and an apparatus for a distributed machine learning system.

BACKGROUND

[002] A machine learning system may broadly refer to a system that learns from and makes predictions based on a set of data. A machine learning system may employ a model such as, for example, an artificial neural network, which supports vector machines, etc., to perform the processing. Machine learning systems can process hundreds of millions of data points at very high speed, and have contributed substantially to the advance of various technology fields, such as image recognition, speech recognition, natural language processing, information acquisition, etc.

[003] Some machine learning systems can be trained with a set of training data to produce a target output. In a training process, various configuration parameters of the machine learning system can be modified based on a relationship between an actual output of the system and a target output. For example, through a training process, an artificial neural network can be configured, for example, to maximize the likelihood of generating a target output from a particular set of input data, to minimize an error between the target output and actual output, etc.

[004] Despite the high-speed data processing capability provided by machine learning systems, the training process for these machine learning systems is typically iterative, and requires a substantial amount of processing time. As a result, it may become impossible to constantly train the machine learning systems with new data, which can degrade these systems'

performances.

SUMMARY

[005] Embodiments of the present disclosure provide a server for facilitating dependency-based execution of a set of machine learning tasks. The server may comprise a model data storage configured to receive data representing a machine learning model, the data including information related to storage elements in the machine learning model and connections between the storage elements. The server may also comprise a task module configured to: receive processing assignments for some of the storage elements in the machine learning model, and determine a set of machine learning tasks and dependencies among the set of machine learning tasks based on the data and the processing assignments. The server may also comprise a scheduler module configured to determine a resolution status for each of the dependencies and determine, based on the resolution statuses, a sequence of execution of the set of machine learning tasks. The server may also comprise one or more hardware processors configured to execute the set of machine learning tasks based on the determined sequence.

[006] Embodiments of the present disclosure also provide a method for facilitating dependency-based execution of a set of machine learning tasks. The method may comprise receiving data representing a machine learning network model, the data including information related to storage elements in the model and connections between the storage elements; receiving processing assignments for some of the storage elements in the machine learning model; determining a set of machine learning tasks and dependencies among the set of machine learning tasks based on the data and the processing assignments; determining a resolution status for each of the dependencies; determining, based on the resolution statuses, a sequence of execution of the set of machine learning tasks; and executing the set of machine learning tasks based on the

determined sequence.

[007] Embodiments of the present disclosure also provide a non-transitory computer readable medium that stores a set of instructions that is executable by at least one hardware processor of an apparatus to cause the apparatus to perform a method of facilitating dependency-based execution of a set of machine learning tasks. The method may comprise receiving data representing a machine learning network model, the data including information related to storage elements in the model and connections between the storage elements; receiving processing assignments for some of the storage elements in the machine learning model; determining a set of machine learning tasks and dependencies among the set of machine learning tasks based on the data and the processing assignments; determining, based on the resolution statuses, a sequence of execution of the set of machine learning tasks; and executing the set of machine learning tasks based on the determined sequence.

[008] Additional objects and advantages of the disclosed embodiments will be set forth in part in the following description, and in part will be apparent from the description, or may be learned by practice of the embodiments. The objects and advantages of the disclosed embodiments may be realized and attained by the elements and combinations set forth in the claims.

[009] It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the disclosed embodiments, as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

[010] **FIGs. 1A-1B** are diagrams illustrating the operations of an exemplary machine learning model consistent with embodiments of the present disclosure.

[011] **FIGs. 2A-2C** are diagrams illustrating an exemplary distributed machine learning system 200 in which embodiments of the present disclosure can be used.

[012] **FIGs. 3A-3C** are graphs illustrating dependencies among the tasks illustrated in **FIG. 2C**.

[013] **FIGs. 4A-4B** are block diagrams illustrating an exemplary task processor for facilitating dependency-based execution of the machine learning tasks according to embodiments of the present disclosure.

[014] **FIG. 5** is a flowchart illustrating an exemplary method for facilitating dependency-based execution of the machine learning tasks according to embodiments of the present disclosure.

[015] **FIG. 6** is a block diagram illustrating an exemplary computer system on which embodiments described herein can be implemented.

DESCRIPTION OF THE EMBODIMENTS

[016] Reference will now be made in detail to exemplary embodiments, examples of which are illustrated in the accompanying drawings. The following description refers to the accompanying drawings in which the same numbers in different drawings represent the same or similar elements unless otherwise represented. The implementations set forth in the following description of exemplary embodiments do not represent all implementations consistent with the invention. Instead, they are merely examples of apparatuses and methods consistent with aspects related to the invention as recited in the appended claims.

[017] With embodiments of the present disclosure, a computer that performs a set of machine learning tasks can determine dependencies among the set of machine learning tasks, and create a task status record that stores information related to the dependencies, indication of

execution statuses of the set of machine learning tasks, and resolution statuses of the dependencies of the set of machine learning tasks. The computer can determine a sequence of execution of the set of machine learning tasks based on the resolution statuses of the dependencies, and execute the set of machine learning tasks based on the determined sequence. The execution can be arranged to facilitate concurrent execution of tasks with resolved dependencies (or no dependencies). As a result, the execution time for these tasks can be minimized, and the performance of the machine learning tasks can be improved.

[018] According to some embodiments, the operations, techniques, and/or components described herein can be implemented by an electronic device, which can include one or more special-purpose computing devices. The special-purpose computing devices can be hard-wired to perform the operations, techniques, and/or components described herein, or can include digital electronic devices such as one or more application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs) that are persistently programmed to perform the operations, techniques and/or components described herein, or can include one or more hardware processors programmed to perform such features of the present disclosure pursuant to program instructions in firmware, memory, other storage, or a combination. Such special-purpose computing devices can also combine custom hard-wired logic, ASICs, or FPGAs with custom programming to accomplish the technique and other features of the present disclosure. The special-purpose computing devices can be desktop computer systems, portable computer systems, handheld devices, networking devices, or any other device that incorporates hard-wired and/or program logic to implement the techniques and other features of the present disclosure.

[019] The one or more special-purpose computing devices can be generally controlled and coordinated by operating system software, such as iOS, Android, Blackberry, Chrome OS,

Windows XP, Windows Vista, Windows 7, Windows 8, Windows Server, Windows CE, Unix, Linux, SunOS, Solaris, VxWorks, or other compatible operating systems. In other embodiments, the computing device can be controlled by a proprietary operating system. Operating systems control and schedule computer processes for execution, perform memory management, provide file system, networking, I/O services, and provide a user interface functionality, such as a graphical user interface ("GUI"), among other things.

[020] Reference is now made to **FIG. 1A**, which illustrates an exemplary machine learning model 100 with which embodiments of the present disclosure can be used. Machine learning model 100 can be configured to map a set of text data into a vector space to perform certain types of analysis. Machine learning model 100 can be used in, for example, a search query auto-completion system, a search engine, etc.

[021] As shown in **FIG. 1A**, machine learning model 100 can include a deep neural network topology with multiple layers, including layer 102, layer 104, layer 106, layer 108, and layer 110. Each layer includes a predetermined number of storage elements. In a case where machine learning model 100 is used to map text data into a vector space, each storage element can correspond to a vector value associated with a set of reference text. For example, layer 102 includes 300k – 1 million storage elements, which can correspond to a set of reference texts numbered between 300k to 1 million. Also, each of layers 104, 106, 108, and 110 includes 100-500 storage elements, which can correspond to a set of 100-500 reference texts.

[022] Each storage element of a layer may store an output value determined based on a set of weights w, a bias value b, and a set of input values. As to be discussed below, each storage element can also store an error value. For layer 102, the set of input values can be a set of numbers representing a collection of input texts (e.g., a phrase). For each of layers 104, 106, 108,

and 110, the set of input values can be some or all of the output values stored in a preceding layer. For example, as shown in **FIG. 1A**, the output value stored in storage element E104n of layer 104 can be determined based on the output values stored in each storage element of layer 102, a weighing matrix W_1, and a bias matrix B_1, according to the following exemplary expression:

$$\text{E104n} = f(\sum_{i=1}^{N}(\text{wi} \times \text{E102i}) + \text{bn}) \qquad \text{(Expression 1)}$$

[023] Here, wi can be a matrix value included in weighing matrix W_1 that corresponds to a particular storage element E102i of layer 102 out of N storage elements in layer 102. Moreover, bn can be a matrix value included in bias matrix B_1 that corresponds to storage element E104n.

[024] Further, function f can be in the form of an activation function according to the following exemplary expression:

$$f(x) = \frac{1}{(1+e^{-x})} \qquad \text{(Expression 2)}$$

[025] The output values stored in each of layers 102, 104, 106, 108, and 110 can be determined based on Expressions 1 and 2 above, in a forward propagation process. For example, after the output values stored in the storage elements of layer 102 are generated, these output values can be propagated to layer 104 where these values are combined with weighing matrix W_1 and bias matrix B_1 according to Expressions 1 and 2 to generate the values stored in layer 104. Subsequently, the output values of layer 104 can be propagated to layer 106 to be combined with weighing matrix W_2 and bias matrix B_2 to generate the output values stored at layer 106. The output values of layer 106 can then be propagated to layer 108 to be combined with weighing matrix W_3 and bias matrix B_3 to generate the output values stored at layer 108. Finally, the output values of layer 108 can be propagated to layer 110 and combined with

weighing matrix W_3 and bias matrix B_3 to generate the output values stored at layer 110. The output values of layer 110 can be a set of vectors that represent the input texts. The set of vectors can then be used to determine an attribute of the input texts. As an illustrative example, the set of vectors can be used to determine a semantic relationship between the input texts and a set of documents, which can then be provided as a search result in response to a query containing the input texts. The semantic relationship can be determined based on, for example, a cosine distance between the set of vectors output by machine learning model 100 and a set of vectors representing the set of documents.

[026] Referring to **FIG. 1B**, as a part of the training process, the output values of machine learning model 100 can be compared with a target to determine an error. The weighing matrices W_1, W_2, W_3, and W_4, as well as the bias matrices B_1, B_2, B_3, and B_4, can be updated based on the error. Using the illustrative example, machine learning model 100 can be associated with an objective function to minimize a cosine distance between the output vectors of a predetermined input text and the vectors of a predetermined set of documents. In this example, the weighing matrices and the bias matrices can be updated according to the objective function.

[027] The updating of the weighing matrices and the bias matrices can occur in a process of backward propagation. In the process of backward propagation, an error value can be determined for each storage element at the topmost layer 110. The error values can then be propagated back to layer 108, and the error values stored at each storage element of layer 108 can be determined according to the following exemplary expression:

$$\mathrm{Er108n} = Cn \times \sum_{i=1}^{N} (\mathrm{wi} \times \mathrm{Er110i}) \quad \text{(Expression 3)}$$

Here, Er108n is an error value stored at storage element E108n of layer 108. Further, wi can be a

matrix value included in weighing matrix W_4 that corresponds to a particular storage element E110i of layer 110 out of N elements in layer 110. Er110i is an error value calculated based on a comparison between a value stored at storage element E110i and the target. The parameter Cn can be a value based on the output value stored in storage element E108n. In some cases, as a part of the training process, a gradient (e.g., a rate of change of a value) of the objective function with respect to weighing matrix W_4 can also be determined for storage element E108n. The gradient can be determined based on the error value stored at storage element E108n, and can be used to update weighing matrix W_4. The gradient value can also be stored at storage element E108n.

[028] The error values and gradient values stored in each of layers 102, 104, and 106 can be determined based on Expression 3 and by the backward propagation process. For example, after the error values and gradient values stored in the storage elements of layer 108 are generated, the error values can be propagated back to layer 106, where these values are combined with weighing matrix W_3 according to Expression 3 to generate the error values and gradient values stored in layer 106. Subsequently, the error values of layer 106 can be propagated back to layer 104 to be combined with weighing matrix W_2 to generate the error values and gradient values stored at layer 104. The error values of layer 104 can then be propagated back to layer 102 to be combined with weighing matrix W_1 to generate the error values and gradient values stored at layer 102.

[029] After the updating of the weighing matrices, multiple iterations of forward propagation and backward propagation can be performed until the error at the output layer falls within a predetermined threshold.

[030] Reference is now made to **FIG. 2A**, which illustrates an exemplary distributed

machine learning system 200 in which embodiments of the present disclosure can be used. As shown in **FIG. 2A**, distributed machine learning system 200 includes a server cluster 201 and a server cluster 203. Server cluster 201 includes a plurality of task processors 202a-202n, whereas server cluster 203 includes a parameters processor 204 and a resource coordinator 206. Each task processor may be hosted on a server, or on a plurality of servers. Parameters processor 204 and resource coordinator 206 can be hosted on a server, or on different servers. Parameters processor 204 may also be hosted on a server or on a plurality of servers. Each server may include one or more memory devices and one or more hardware processors. The hardware processors may include a single processing core or multiple processing cores. The hardware processors can be based on various architectures, such as being a general purpose central processing unit (CPU), a graphical processing unit (GPU), etc. The servers can be communicatively coupled with each other either over a network via a set of network interfaces (both of which are not shown in **FIG. 2A**). Although FIG. 2A illustrates that distributed machine learning system 200 includes two server clusters, it is understood that distributed machine learning system 200 may also include a single server cluster or more than two server clusters, with task processors 202a-202n, parameters processor 204 and resource coordinator 206 distributed among the servers within the cluster.

[031] Task processors 202a-202n and parameters processor 204 can, under the coordination of resource coordinator 206, collaborate to perform the aforementioned training process. For example, a task processor (or a hardware processor core associated with the task processor) can be assigned to perform the aforementioned training process on some of the storage elements of a machine learning model, such as machine learning model 100 of **FIGs. 1A** and **1B**. The assignment can be performed by resource coordinator 206 based on, for example,

available computing resources at task processors 202a-202n. Resource coordinator 206 can also keep a record of which of the storage elements of the machine learning model are being processed by each of task processors 202a-202n.

[032] Each of task processors 202a-202n can store a replica of data representing the machine learning model in the memory devices included in the task processors. The data may include, for example, a data structure (e.g., a search tree) that defines the storage elements in each of layers 102-110 and the connections between these storage elements. The data may also include the output values and error values stored in each of the storage elements, as well as parameters (e.g., the weighing matrices and bias matrices) associated with the model. Each task processor can obtain the parameters from the parameters processor 204 over the network, and perform the forward and backward propagations with respect to the assigned storage elements of the model. The performance of forward and backward propagations may include, for example, computing the output and error values for each storage element, as well as a gradient value of an objective function with respect to the weighing matrices for each storage element based on the output and error values. Each of task processors 202a-202n can then transfer the computed gradient values to parameters processor 204, which can update the weighing matrices based on the computed gradients, and transfer the updated weighing matrices back to task processors 202a-202n to perform new iterations of the forward and backward propagations. The hardware processors in each of task processors 202a-202n can also execute one or more tasks associated with the performances of the aforementioned forward and backward propagations.

[033] Reference is now made to **FIG. 2B**, which illustrates the tasks associated with the performances of the aforementioned forward and backward propagations. As shown in **FIG. 2B**, operation 250 may be associated with a forward propagation operation according to Expressions

1 and 2, or a backward propagation operation according to Expression 3. For example, the inputs (in1, in2, in3, in4) can be the output values or error values stored in the storage elements of a preceding or subsequent layer. The output can be an output value (for forward propagation) or a gradient value (for backward propagation). Operation 250 can include a set of computation tasks 252 such as, for example, arithmetic operations including multiplication and addition, read operations to a mapping table that represent the function f, etc., to generate the output from the inputs.

[034] Operation 250 can also include a set of communication tasks. For example, the model parameters (e.g., weighing factors W1, W2, W3, and W4) may be obtained from parameters processor 204 over the network in a communication task 254. Also, in a case where the computation of output values and error values of the same layer is distributed among a number of task processors, a task processor may also obtain, as inputs to computation tasks 252, the computed output values from another task processor to perform the computations for a subsequent layer. As an illustrative example, a task processor handling the computation of some of the output values at layer 104 may have to obtain the output values of some of layer 102 storage elements computed by another task processor over the network in a communication task 254. Also, the output of operation 250 (e.g., gradient values) can also be transmitted to parameters processor 204 over the network in a communication task 256. These communication tasks may include, for example, generating packets to include the parameter and gradient data, writing the packets data to a buffer queue in a network interface, etc.

[035] For forward and backward propagations, the tasks processors 202a-202n can execute (using the hardware processors) a set of operations 250 to determine the output values and error values at each layer of machine learning model 100. There can be dependencies among

the set of operations 250. For example, as discussed above, during forward propagation, the output values are propagated and updated sequentially by layers 102, 104, 106, 108, and 110. Also, during backward propagation, the error values (generated based on the output values from forward propagation) are propagated and updated sequentially by layers 110, 108, 106, 104, and 102.

[036] In some embodiments, operation 250 can be executed sequentially, starting with layer 102 followed by layer 104, 106, 108, and 110 for forward propagation, followed by backward propagation starting with layer 110 followed by layer 108, 106, 104, and 102. Timing diagrams 260 and 262 in **FIG. 2C** illustrate the timing of sequential executions of operations 250. Timing diagram 260 can correspond to the timing of sequential execution of operations 250 for forward propagation. For example, starting with time t0, a task processor (e.g., task processor 202a) can first execute a communication task 254a to obtain input data. After the communication task 254a completes and the input data become available, the task processor can then execute computation tasks 252a to determine output values of some of the storage elements (assigned by resource coordinator) at layer 102. After the computation tasks complete, the work machine can then execute a second communication task 254b to obtain data for the weighing matrix W_1 from parameter processor 204, as well as output values at layer 102 from other task processors. When the communication task completes, and weighing matrix W_1 and output values at layer 102 become available, the work machine can execute another set of computation tasks 252b to determine the output values at layer 104. This is then followed by, sequentially, a communication task 254c for obtaining data for weighing matrix W_2 and layer 104 output values, computation tasks 252c for computing the output values at layer 106, communication task 254d for obtaining data for weighing matrix W_3 and layer 106 output values, computation

tasks 252d for computing the output values at layer 108, communication task 254e for obtaining data for weighing matrix W_4 and layer 108 output values, and computation tasks 252e for computing the assigned output values at layer 110, which completes at time t10. Other task processors (e.g., task processors 202b, 202c, etc.) can also sequentially execute operations 250 sequentially to perform forward propagation for their assigned storage elements at each layer.

[037] On the other hand, timing diagram 262 can correspond to the timing of sequential execution of operations 250 for backward propagation, after forward propagation completes. Starting at time t11, the task processor can first execute computation tasks 252f to compute error values and gradient values for some of the assigned storage elements at layer 110. When the computation tasks complete, the work machine can execute a communication task 254f to transmit the gradient values at layer 110 to parameter processor 204, which can update the parameters based on the gradient values for the next forward propagation process. Also, after the computation of error values at layer 110 completes and the error values become available, the task processor can execute another set of computation tasks 252g to determine the error values and gradient values at layer 108. This is then followed by, sequentially, communication task 254g for transmitting the gradient values of layer 108 to parameter processor 204, computation tasks 252h for computing error values and gradient values at layer 106, communication task 254h for transmitting the gradient values of layer 106, computation tasks 252i for computing error values and gradient values at layer 104, communication task 254i for transmitting the gradient values of layer 104, computation tasks 252j for computing error values and gradient values at layer 102, and communication tasks 254j for transmitting the gradient values of layer 102, which completes at time t21. Other task processors can also sequentially execute operations 250 sequentially to perform backward propagation for their assigned storage elements at each

layer.

[038] With the arrangements shown in **FIG. 2C**, where operations 250 are executed sequentially for each layer, the data dependencies between layers and between tasks can be resolved before the tasks are executed. However, the execution times for each of operations 250 adds up and can contribute to substantial latency to the forward and backward propagation processes.

[039] According to embodiments of the present disclosure, the dependencies among the computation and communication tasks can be further refined, which can facilitate concurrent execution of some of these tasks by a multi-core computer processor that supports concurrent execution of multiple threads. Reference is now made to **FIGs. 3A** and **3B**, which are graphs that illustrate refined dependencies among the computation and communication tasks illustrated in **FIG. 2C**.

[040] Graph 300 of **FIG. 3A** illustrates the dependencies among the computation and communication tasks for a forward propagation process executed by a task processor (e.g., task processor 202a). As shown in **FIG. 3A**, the execution of computation tasks 252a, which computes output values for layer 102, depends on the completion of communication task 254a, because communication task 254a provides the input data for the execution of computation tasks 252a. Moreover, the execution of computation tasks 252b, which computes output values for layer 104, also depends on the completion of communication task 254b and computation tasks 252a, because each provides the input (e.g., output values at layer 102, weighing matrices W_1, etc.) for the execution of computation tasks 252b. Therefore, computation tasks 252a may be executed after the completion of communication task 254a, and computation tasks 252b may be executed after the completion of communication task 254b and computation tasks 252a.

[041] However, the computation tasks 252a and communication task 254b are independent from each other. For example, the acquisition of the W_1 matrices from parameter processor 204 can be independent from computation tasks 252a, which neither provides nor uses the W_1 matrices. Moreover, the acquisition of output values of layer 102 computed by other task processors (e.g., task processors 202b, 202n, etc.) are also independent from computation tasks 252a, since each task processor is assigned to compute output values of a specific set of storage elements of the same layer, and the computations of output values of storage elements among the same layer can occur independently from each other. These task processors can also be configured to execute their own computation tasks 252a and generate their layer 102 output values before task processor 202a does. Therefore, computation tasks 252a and communication task 254b can be executed concurrently.

[042] Similar dependency relationships also exist for other tasks in the forward propagation process. For example, computation tasks 252c may be executed after the completion of communication task 254c and computation tasks 252b, but communication task 254c and computation tasks 252b can be executed concurrently. Moreover, computation tasks 252d may be executed after the completion of communication task 254d and computation tasks 252c, but communication task 254d and computation tasks 252c can be executed concurrently. Further, computation tasks 252e may be executed after the completion of communication task 254e and computation tasks 252d, but communication task 254e and computation tasks 252d can be executed concurrently.

[043] Graph 320 of **FIG. 3B** illustrates the dependencies among the computation and communication tasks for a backward propagation process executed by a task processor (e.g., task processor 202a). As shown in **FIG. 3B**, the execution of computation tasks 252f for computing

error and gradient values for layer 110 depends on the completion of computation tasks 252e (the computation of output values at layer 110), because the error values are computed based on a difference between the output values and target values. Also, the execution of computation tasks 252g for computing error and gradient values for layer 108 also depends on the completion of computation tasks 252e, because computation tasks 252e provide the errors values to computation tasks 252g as inputs. For similar reasons, the execution of computation tasks 252h starts after computation tasks 252g complete, the execution of computation tasks 252i starts after computation tasks 252h complete, and the execution of computation tasks 252j starts after computation tasks 252i complete.

[044] Moreover, the execution of communication task 254f for transferring layer 110 gradient values depends on the completion of computation tasks 252f, which generates the layer 110 gradient values. Therefore, the execution of communication task 254f starts from the completion of computation tasks 252f. For similar reasons, the execution of communication task 254g starts after completion of computation tasks 252g, the execution of communication task 254h starts after completion of computation tasks 252h, the execution of communication task 254i starts after completion of computation tasks 252i, and the execution of communication task 254j starts after completion of computation tasks 252j.

[045] However, the execution of communication task 254f for transferring layer 110 gradient values (to parameter processor 204) can be independent from the execution of computation tasks 252g (at task processor 202a) for computing error and gradient values for layer 108, at least because computation tasks 252g do not use gradient values for layer 110. Therefore, the execution of communication task 254f and computation tasks 252g can occur concurrently. For similar reasons, the execution of communication task 254g and computation

tasks 252h can occur concurrently, the execution of communication task 254h and computation tasks 252i can occur concurrently, and the execution of communication task 254i and computation tasks 252j can also occur concurrently.

[046] Reference is now made to **FIG. 3C**, which illustrate timing diagrams 360 and 362. Timing diagrams 360 and 362 illustrate the timing of the aforementioned concurrent and sequential executions of the computation and communication tasks. As discussed above, for forward propagation, the executions of computation tasks 254a-e occur sequentially, but each of computation tasks 254a-e can also be executed concurrently with one of communication tasks 252b-e. Also, for backward propagation, the executions of computation tasks 254f-j occur sequentially, but each of computation tasks 254f-j can also be executed concurrently with one of communication tasks 252f-j. Comparing with timing diagrams 260 and 262 of **FIG. 2C**, the total execution time for forward and backward propagations can be substantially reduced.

[047] Reference is now made to **FIG. 4A**, which illustrates an exemplary task processor 400 for facilitating dependency-based execution of machine learning tasks, according to embodiments of the present disclosure. Task processor 400 can be used in distributed machine learning system 200 of **FIG. 2A** and can interact with parameters processor 204 and resource coordinator 206 to perform a training process on learning model 100 of **FIG. 1A**. As shown in **FIG. 4A**, task processor 400 includes a model data storage 402, a task module 404, a scheduler module 406, a queue buffer 408, and a processing unit 410.

[048] Queue buffer 408 may include a set of queue buffers 408a-408e for storing a set of tasks to be provided to task processor 408. Each of queue buffers 408a-408e may be associated with a predetermined task attribute, such as a task type and a task criticality. For example, a task can be stored in a particular queue buffer based on whether the task is a

communication task or a computation task, whether the task is to be completed within a predetermined time period, etc.

[049] Processing unit 410 may include a multi-core hardware processor (e.g., a central processing unit, a graphical processing unit, etc.) that supports multithreading, and may associate a thread of execution with each queue included in queue buffer 408. As to be discussed below, task processor 400 can determine the dependencies among the computation and communication tasks associated with the training process, and control the queuing of these tasks in queue buffer to provide concurrent and sequential executions of these tasks based on the determined dependencies.

[050] In general, the words "module," as used herein, can be a packaged functional hardware unit designed for use with other components (e.g., portions of an integrated circuit) or a part of a program (stored on a computer readable medium) that performs a particular function of related functions. The module can have entry and exit points and can be written in a programming language, such as, for example, Java, Lua, C or C++. A software module can be compiled and linked into an executable program, installed in a dynamic link library, or written in an interpreted programming language such as, for example, BASIC, Perl, or Python. It will be appreciated that software modules can be callable from other modules or from themselves, and/or can be invoked in response to detected events or interrupts. Software modules configured for execution on computing devices can be provided on a computer readable medium, such as a compact disc, digital video disc, flash drive, magnetic disc, or any other non-transitory medium, or as a digital download (and can be originally stored in a compressed or installable format that requires installation, decompression, or decryption prior to execution). Such software code can be stored, partially or fully, on a memory device of the executing computing device, for

execution by the computing device. Software instructions can be embedded in firmware, such as an EPROM. It will be further appreciated that hardware modules can be comprised of connected logic units, such as gates and flip-flops, and/or can be comprised of programmable units, such as programmable gate arrays or processors. The modules or computing device functionality described herein are preferably implemented as software modules, but can be represented in hardware or firmware. Generally, the modules described herein refer to logical modules that can be combined with other modules or divided into sub-modules despite their physical organization or storage.

[051] Model data storage 402 can store a data structure that represents a machine learning model (e.g., machine learning model 100 of **FIG. 1A**). The data structure may include, for example, a search tree that defines the storage elements in each of layers 102-110 of machine learning model 100, and the connections between these storage elements.

[052] Task module 404 can determine, based on the data structure stored in model data storage 402, a set of tasks for the training process, as well as dependencies among these tasks. The determination of the set of tasks and dependencies among these tasks can be triggered by, for example, receiving an assignment instruction from a resource coordinator (e.g., resource coordinator 206) to perform the aforementioned training process on the assigned storage elements of machine learning model 100. Task module 404 can create, for each storage element, a set of tasks for forward propagation, and a set of tasks for backward propagation. For forward propagation, the set of tasks may include communication tasks for obtaining input values from other task processors and for obtaining model parameters (e.g., weighing matrix) from parameter processor 204, as well as computation tasks for computing output values. For backward propagation, the set of tasks may include computation tasks for computing error values and

gradient values, and a communication task for transferring the gradient values to parameter processor 204.

[053] Task module 404 can also determine, based on the connections between these storage elements, dependencies for these tasks, and create a record of dependency statuses of these tasks. Reference is now made to **FIG. 4B**, which illustrates an example of task status record 412 created by task module 404. As shown in **FIG. 4B**, task status record 412 includes a graph representation (e.g., in the form of a search tree) of the dependencies among communication tasks 252a-d and computation tasks 254a-d of **FIG. 3A**. Each of these tasks is also associated with an execution status indicator. Moreover, record 412 also includes a set of dependency resolution status indicators 420a-d, each of which can provide an indication about whether a dependency resolved between two tasks has been resolved. The indication can be determined based on an execution status of the tasks. As an example, as discussed above, the execution of computation tasks 252a for layer 102 output values depend on the completion of communication task 254a for bringing in the input values for computation tasks 252a. Dependency resolution status indicator 420a may be set, based on the status indicator of communication task 254a indicating completion, that the dependency on communication task 254a has been cleared. This information allows the execution of computation tasks 252a to proceed. Similarly, dependency resolution status indicator 420b may also be set based on the status indicators of communication task 254b and computation tasks 252a.

[054] Referring back to **FIG. 4A**, scheduler module 406 can control which task is to be executed, and the order by which the tasks are to be executed, based on task status record 412 of **FIG. 4B**. For example, scheduler module 406 can determine a sequence of execution of tasks according to a sequence of clearances of dependency resolution status indicators 402a-d.

Scheduler 406 may also allow the execution of all of the tasks with resolved dependencies (or no dependencies) to proceed concurrently. For example, when dependency resolution status 420a indicates that the dependency has been resolved, scheduler module 406 may allow the executions of computation tasks 252a and communication task 254b to proceed. Next, when dependency resolution status 420b indicates that the dependency has been resolved (which may occur after computation tasks 252a complete), scheduler 406 may allow the execution of computation tasks 252b and communication task 254c to proceed.

[055] Scheduler module 406 can control which task is to be executed, and the order by which the tasks are to be executed, by determining which of the tasks are to be queued in queue buffer 408, after dependency resolution status 420a indicates that the dependency for a particular task has been resolved. The determination can be based on, for example, matching the task attributes with the a task attribute associated with the queue. For example, queues 408a, 408c, and 408e may be associated with a communication task, and queues 408b and 408d may be associated with a computation task. Based on these associations, scheduler module 406 may store communication task 254b in one of queues 408a, 408c, and 408e. Scheduler module 406 may also store computation tasks 252a in one of queues 408b and 408d.

[056] Moreover, scheduler module 406 may also determine which queue to store a task based on a criticality of the task, to avoid the task becoming a bottleneck. For example, as discussed above with respect to **FIG. 3A**, the computation tasks 252a and communication task 254b are independent from each other, and can be executed concurrently. On the other hand, computation task 252b depends on completion of both of these tasks. Therefore, to avoid one of the tasks (e.g., computation tasks 252a or communication task 254b) becoming a bottleneck for another task (e.g., computation task 252b), scheduler module 406 may also determine to store

computation tasks 252a and communication task 254b into two queues of similar waiting time. The waiting time can be determined based on, for example, a number of tasks pending in the queues.

[057] Each of these tasks can then be executed concurrently when multiple threads associated with the queues that store these tasks become available. After storing communication task 254b and computation tasks 252a in queues 408b-e, scheduler module 406 may also update the execution status indicators for these tasks, to allow the next set of tasks (e.g., communication task 254c and computation tasks 252b) to be executed by at least clearing the dependency statuses of these tasks.

[058] **FIG. 5** is a flowchart representing an exemplary method 500 for facilitating dependency-based execution of the machine learning tasks, consistent with embodiments of the present disclosure. It will be readily appreciated that the illustrated procedure can be altered to delete steps or further include additional steps. Method 500 can be performed by a server configured as a task processor (e.g., task processor 400 of **FIG. 4A**).

[059] After an initial start, the task processor receives data representing a machine learning network model, in step 502. The data may include, for example, a data structure (e.g., a search tree) that defines the storage elements in each of layers 102-110 of machine learning model 100, and the connections between these storage elements.

[060] Based on the model data, the task processor can determine a set of tasks to be executed for a training process, as well as dependencies among these tasks, in step 504. The set of tasks may include communication and computation tasks for forward propagation and backward propagation. The dependencies can be determined based on the connection between the elements of the machine learning model.

[061] After determining the set of tasks and their dependency relationship, the task processor can create a task status record that stores the dependency relationship information, indications of the execution statuses, as well as dependency resolution statuses of the set of tasks, in step 506. The task status record can include similar information as task status record 412 of **FIG. 4B**, and may include a set of dependency resolution status indicators (e.g., dependency resolution status indicators 402a-d).

[062] Based on the dependency resolution status indicators, the task processor can determine a dependency resolution status for each of the set of tasks, in step 508. The task processor can also determine a sequence of execution of tasks according to a sequence of clearances of the dependency resolution status indicators, in step 510. The sequence may also include arranging a set of tasks with resolved dependencies (or no dependencies) to be executed concurrently (if multiple idle threads are available). The task processor can then execute the tasks based on the determined sequence by storing the tasks in a queue buffer (e.g., queue buffer 406 of **FIG. 4A**) for execution, in step 512.

[063] **FIG. 6** is a block diagram of an exemplary computer system 600 with which embodiments described herein can be implemented. Computer system 600 includes a bus 602 or other communication mechanism for communicating information, and one or more hardware processors 604 (denoted as processor 604 for purposes of simplicity) coupled with bus 602 for processing information. Hardware processor 604 can be, for example, a central processing unit, a graphical processing unit, etc., with multiple processing cores. Computer system 600 can be a part of server clusters 201 and 203 that hosts task processors 202a-202n, parameters processor 204, and resource coordinator 206.

[064] Computer system 600 also includes a main memory 606, such as a random access

memory (RAM) or other dynamic storage device, coupled to bus 602 for storing information and instructions to be executed by processor 604. Main memory 606 also can be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 604. Such instructions, after being stored in non-transitory storage media accessible to processor 604, render computer system 600 into a special-purpose machine that is customized to perform the operations specified in the instructions.

[065] Computer system 600 further includes a read only memory (ROM) 608 or other static storage device coupled to bus 602 for storing static information and instructions for processor 504. A storage device 610, such as a magnetic disk, optical disk, or USB thumb drive (Flash drive), etc., is provided and coupled to bus 602 for storing information and instructions.

[066] Computer system 600 can be coupled via bus 602 to a display 612, such as a cathode ray tube (CRT), an liquid crystal display (LCD), or a touch screen, for displaying information to a computer user. An input device 614, including alphanumeric and other keys, is coupled to bus 602 for communicating information and command selections to processor 604. Another type of user input device is cursor control 616, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 604 and for controlling cursor movement on display 612. The input device typically has two degrees of freedom in two axes, a first axis (for example, x) and a second axis (for example, y), that allows the device to specify positions in a plane. In some embodiments, the same direction information and command selections as cursor control may be implemented via receiving touches on a touch screen without a cursor.

[067] Computing system 600 can include a user interface module to implement a graphical user interface (GUI) that can be stored in a mass storage device as executable software

codes that are executed by the one or more computing devices. This and other modules can include, by way of example, components, such as software components, object-oriented software components, class components and task components, processes, functions, fields, procedures, subroutines, segments of program code, drivers, firmware, microcode, circuitry, data, databases, data structures, tables, arrays, and variables. The modules may include, for example, components of system 200 of **FIG. 2A** and task processor 400 of **FIG. 4A**.

[068] Computer system 600 can implement the techniques described herein using customized hard-wired logic, one or more ASICs or FPGAs, firmware and/or program logic which in combination with the computer system causes or programs computer system 600 to be a special-purpose machine. According to some embodiments, the operations, functionalities, and techniques and other features described herein are performed by computer system 600 in response to processor 604 executing one or more sequences of one or more instructions contained in main memory 606. Such instructions can be read into main memory 606 from another storage medium, such as storage device 610. Execution of the sequences of instructions contained in main memory 606 causes processor 604 to perform the method steps (e.g., method 500 of **FIG. 5**) described herein. In alternative embodiments, hard-wired circuitry can be used in place of or in combination with software instructions.

[069] The term "non-transitory media" as used herein refers to any non-transitory media storing data and/or instructions that cause a machine to operate in a specific fashion. Such non-transitory media can comprise non-volatile media and/or volatile media. Non-volatile media can include, for example, optical or magnetic disks, such as storage device 610. Volatile media can include dynamic memory, such as main memory 606. Non-transitory media include, for example, a floppy disk, a flexible disk, hard disk, solid state drive, magnetic tape, or any other

magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, flash memory, register, cache, any other memory chip or cartridge, and networked versions of the same.

[070] Non-transitory media is distinct from, but can be used in conjunction with, transmission media. Transmission media can participate in transferring information between storage media. For example, transmission media can include coaxial cables, copper wire and fiber optics, including the wires that comprise bus 602. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[071] Various forms of media can be involved in carrying one or more sequences of one or more instructions to processor 604 for execution. For example, the instructions can initially be carried on a magnetic disk or solid state drive of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 600 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 602. Bus 602 carries the data to main memory 606, from which processor 604 retrieves and executes the instructions. The instructions received by main memory 606 can optionally be stored on storage device 610 either before or after execution by processor 604.

[072] Computer system 600 can also include a communication interface 618 coupled to bus 602. Communication interface 618 can provide a two-way data communication coupling to a network link 620 that can be connected to a local network 622. For example, communication

interface 618 can be an integrated services digital network (ISDN) card, cable modem, satellite modem, or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 618 can be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links can also be implemented. In any such implementation, communication interface 618 can send and receive electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[073] Network link 620 can typically provide data communication through one or more networks to other data devices. For example, network link 620 can provide a connection through local network 622 to a host computer 624 or to data equipment operated by an Internet Service Provider (ISP) 626. ISP 626 in turn can provide data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 628. Local network 622 and Internet 628 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 620 and through communication interface 618, which carry the digital data to and from computer system 600, can be example forms of transmission media.

[074] Computer system 600 can send messages and receive data, including program code, through the network(s), network link 620 and communication interface 618. In the Internet example, a server 630 can transmit a requested code for an application program through Internet 628, ISP 626, local network 622 and communication interface 618.

[075] The received code can be executed by processor 604 as it is received, and/or stored in storage device 610, or other non-volatile storage for later execution. In some embodiments, server 630 can provide information for being displayed on a display.

[076] It will be appreciated that the present invention is not limited to the exact construction that has been described above and illustrated in the accompanying drawings, and that various modifications and changes can be made without departing from the scope thereof. It is intended that the scope of the invention should only be limited by the appended claims.

WHAT IS CLAIMED IS:

1. A server for facilitating dependency-based execution of a set of machine learning tasks, comprising:

a model data storage configured to receive data representing a machine learning model, the data including information related to storage elements in the machine learning model and connections between the storage elements;

a task module configured to:

receive processing assignments for some of the storage elements in the machine learning model, and

determine a set of machine learning tasks and dependencies among the set of machine learning tasks based on the data and the processing assignments;

a scheduler module configured to:

determine a resolution status for each of the dependencies, and

determine, based on the resolution statuses, a sequence of execution of the set of machine learning tasks; and

one or more hardware processors configured to execute the set of machine learning tasks based on the determined sequence.

2. The server of claim 1, wherein the task module is further configured to create a task status record that stores a representation of the dependencies among the set of machine

learning tasks, a status indicator associated with each of the set of machine learning tasks, and a set of dependency resolution status indicators determined based on the dependencies and the status indicators;

wherein the scheduler module is further configured to determine a resolution status for each of the dependencies based on the dependency resolution status indicators in the task status record.

3. The server of claim 2, wherein the determination of a sequence of execution of the set of machine learning tasks comprises the task module being configured to: determine the sequence of execution of the set of machine learning tasks based on a sequence of clearance of the set of dependency resolution status indicators.

4. The server of claim 2, wherein one of the set of dependency resolution status indicators is associated with a plurality of tasks of the set of machine learning tasks;

wherein the determination of a sequence of execution of the machine learning tasks comprises the task module being configured to: determine that the plurality of tasks are to be concurrently executed based on a clearance of the one of the set of dependency resolution status indicators.

5. The server of claim 1, wherein the scheduler module is configured to further determine,

based on attributes of the set of machine learning tasks, the sequence of execution of the set of machine learning tasks.

6. The server of claim 5, wherein the attributes comprise a task criticality;

wherein the task processor further comprises a queue buffer comprising a set of queues associated with a set of threads of execution;

wherein the scheduler module is configured to assign the set of the machine learning tasks among the set of queues based on task criticalities associated with tasks of the set of the machine learning tasks, and waiting times at queues of the set of queues.

7. The server of claim 5, wherein the attributes comprise a task type;

wherein the task processor further comprises a queue buffer comprising a set of queues associated with a set of threads of execution;

wherein the scheduler module is configured to assign the set of the machine learning tasks among the set of queues based on task types associated with tasks of the set of the machine learning tasks, and task types associated with queues of the set of queues.

8. A method for facilitating dependency-based execution of a set of machine learning tasks, the method comprising:

receiving data representing a machine learning network model, the data including

information related to storage elements in the model and connections between the storage elements;

receiving processing assignments for some of the storage elements in the machine learning model;

determining a set of machine learning tasks and dependencies among the set of machine learning tasks based on the data and the processing assignments;

determining a resolution status for each of the dependencies;

determining, based on the resolution statuses, a sequence of execution of the set of machine learning tasks; and

executing the set of machine learning tasks based on the determined sequence.

9. The method of claim 8, further comprising:

creating a task status record that stores a representation of the dependencies among the set of machine learning tasks, a status indicator associated with each of the set of machine learning tasks, and a set of dependency resolution status indicators determined based on the dependencies and the status indicators;

wherein the sequence of execution is determined based on the dependency resolution status indicators in the task status record.

10. The method of claim 9, wherein determining a sequence of execution of the set of

machine learning tasks comprises: determining the sequence of execution of the set of machine learning tasks based on a sequence of clearance of the set of dependency resolution status indicators.

11. The method of claim 9, wherein one of the set of dependency resolution status indicators is associated with a plurality of tasks of the set of machine learning tasks;

wherein determining a sequence of execution of the machine learning tasks comprises: determining that the plurality of tasks are to be concurrently executed based on a clearance of the one of the set of dependency resolution status indicators.

12. The method of claim 8, wherein the sequence of execution of the set of machine learning tasks is further determined based on attributes of the set of machine learning tasks.

13. The method of claim 12, wherein the attributes comprise a task criticality;

wherein the method further comprises: assigning the set of the machine learning tasks among a set of queues associated with a set of threads of execution, the assignment being based on task criticalities associated with tasks of the set of the machine learning tasks and waiting times at queues of the set of queues.

14. The method of claim 11, wherein the attributes comprise a task type;

wherein the method further comprises: assigning the set of the machine learning tasks among a set of queues associated with a set of threads of execution, the assignment being based on task types associated with tasks of the set of the machine learning tasks and task types associated with queues of the set of queues.

15. A non-transitory computer readable medium that stores a set of instructions that is executable by at least one hardware processor of an apparatus to cause the apparatus to perform a method of facilitating dependency-based execution of a set of machine learning tasks, the method comprising:

receiving data representing a machine learning network model, the data including information related to storage elements in the model and connections between the storage elements;

receiving processing assignments for some of the storage elements in the machine learning model;

determining a set of machine learning tasks and dependencies among the set of machine learning tasks based on the data and the processing assignments;

determining a resolution status for each of the dependencies;

determining, based on the resolution statuses, a sequence of execution of the set of machine learning tasks; and

executing the set of machine learning tasks based on the determined sequence.

16. The non-transitory computer readable medium of claim 15, further storing the set of instructions executable by at least one hardware processor of the apparatus to perform: creating a task status record that stores a representation of the dependencies among the set of machine learning tasks, a status indicator associated with each of the set of machine learning tasks, and a set of dependency resolution status indicators determined based on the dependencies and the status indicators;

wherein the sequence of execution is determined based on the dependency resolution status indicators in the task status record.

17. The non-transitory computer readable medium of claim 16, wherein the sequence of execution of the set of machine learning tasks is determined based on a sequence of clearance of the set of dependency resolution status indicators.

18. The non-transitory computer readable medium of claim 16, wherein one of the set of dependency resolution status indicators is associated with a plurality of tasks of the set of machine learning tasks;

wherein determining a sequence of execution of the machine learning tasks comprises non-transitory computer readable medium storing the set of instructions

executable by at least one hardware processor of the apparatus to perform: determining that the plurality of tasks are to be concurrently executed based on a clearance of the one of the set of dependency resolution status indicators.

19. The non-transitory computer readable medium of claim 15, wherein the non-transitory computer readable medium further stores the set of instructions executable by at least one hardware processor of the apparatus to cause the apparatus to perform: assigning the set of the machine learning tasks among a set of queues associated with a set of threads of execution, the assignment being based on task criticalities associated with tasks of the set of the machine learning tasks and waiting times at queues of the set of queues.

20. The non-transitory computer readable medium of claim 15, wherein the non-transitory computer readable medium further stores the set of instructions executable by at least one hardware processor of the apparatus to cause the apparatus to perform: assigning the set of the machine learning tasks among a set of queues associated with a set of threads of execution, the assignment being based on task types associated with tasks of the set of the machine learning tasks and task types associated with queues of the set of queues.

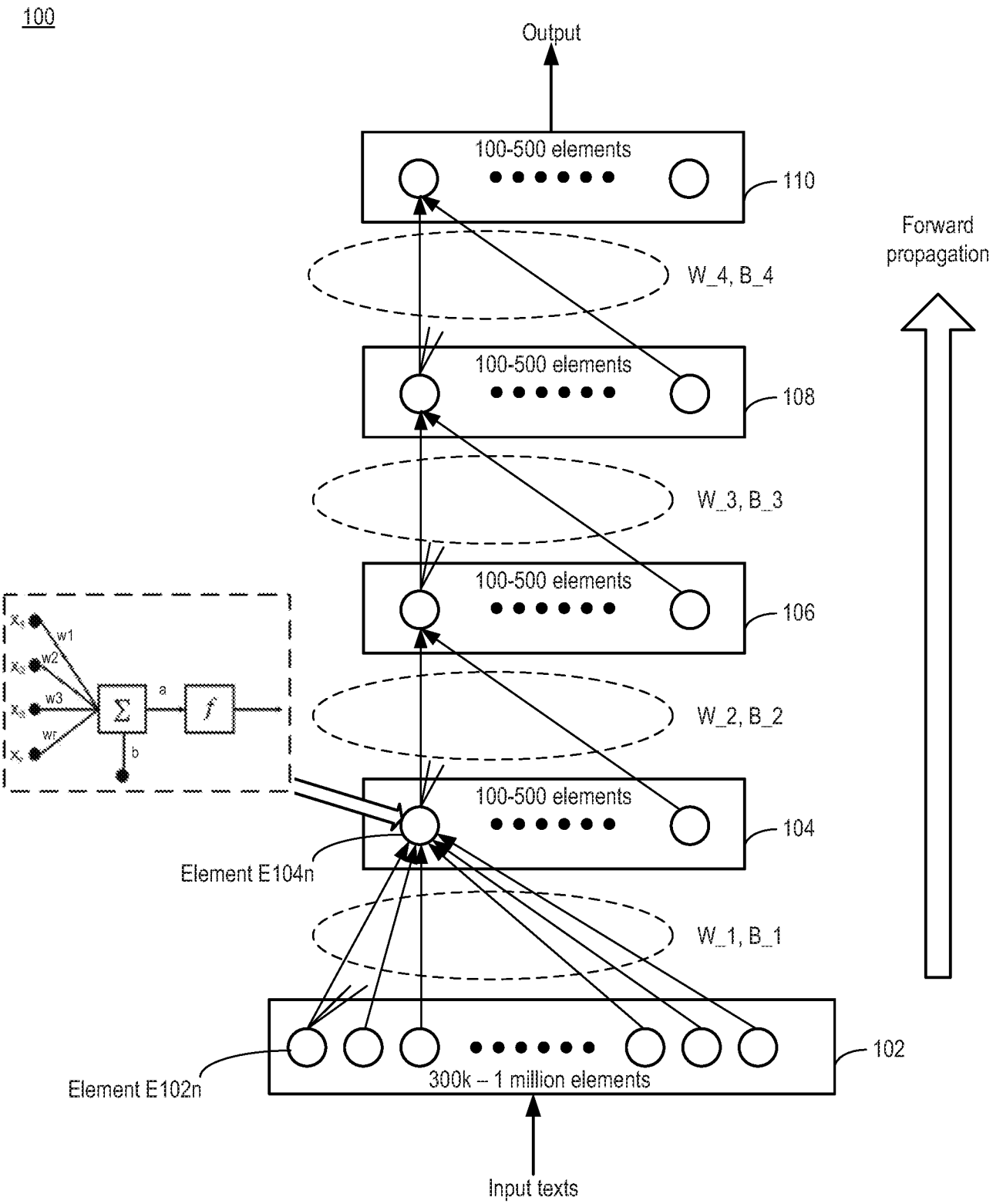
100
Output
100-500 elements
110
Forward propagation
W_4, B_4
100-500 elements
108
W_3, B_3
100-500 elements
106
W_2, B_2
100-500 elements
104
Element E104n
W_1, B_1
102
Element E102n
300k – 1 million elements
Input texts
w1
w2
w3
wr
a
b

FIG. 1A

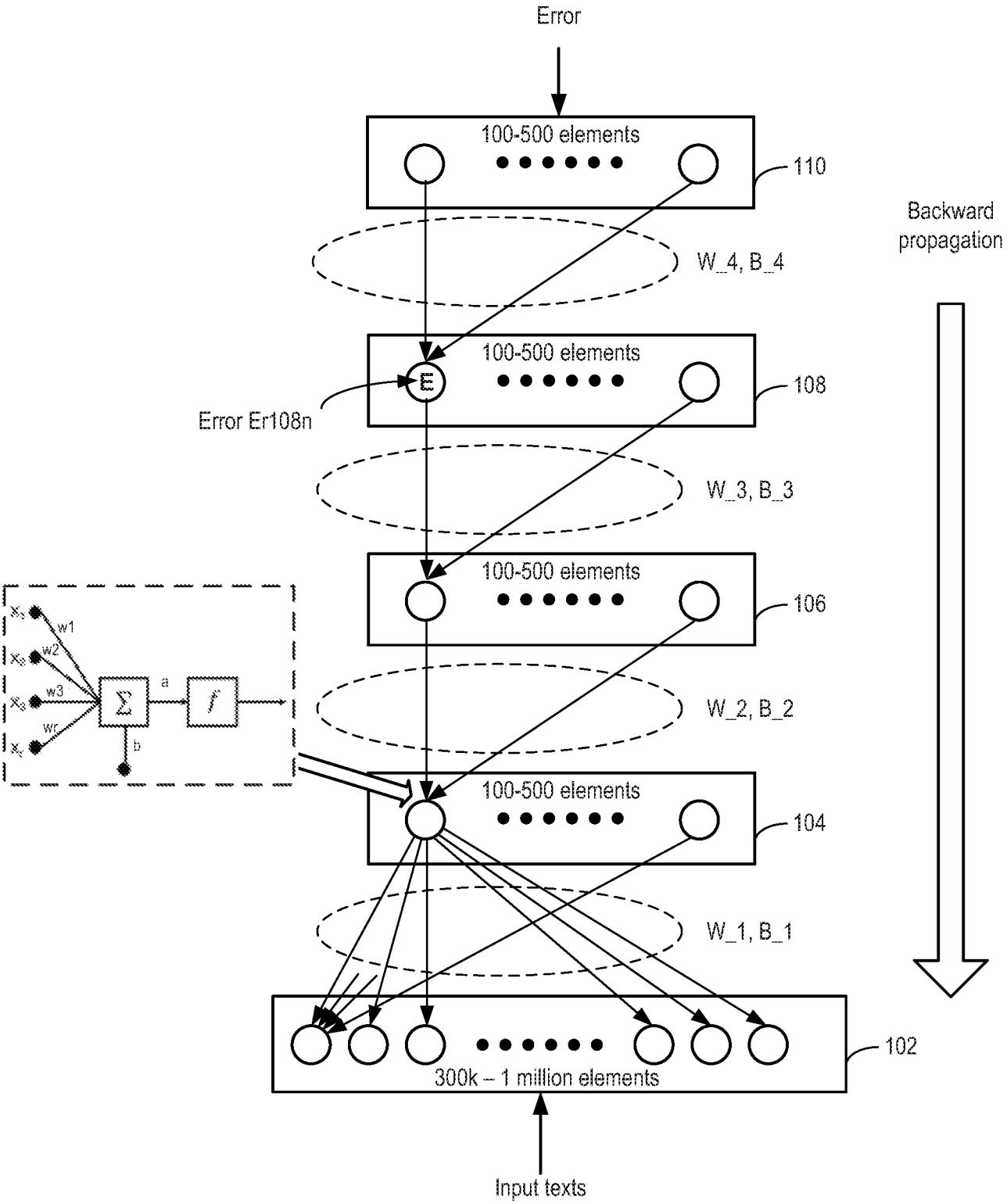
Error
100-500 elements
110
Backward propagation
W_4, B_4
100-500 elements
108
Error Er108n
W_3, B_3
100-500 elements
106
W_2, B_2
100-500 elements
104
W_1, B_1
102
300k – 1 million elements
Input texts
w1
w2
w3
a
b

FIG. 1B

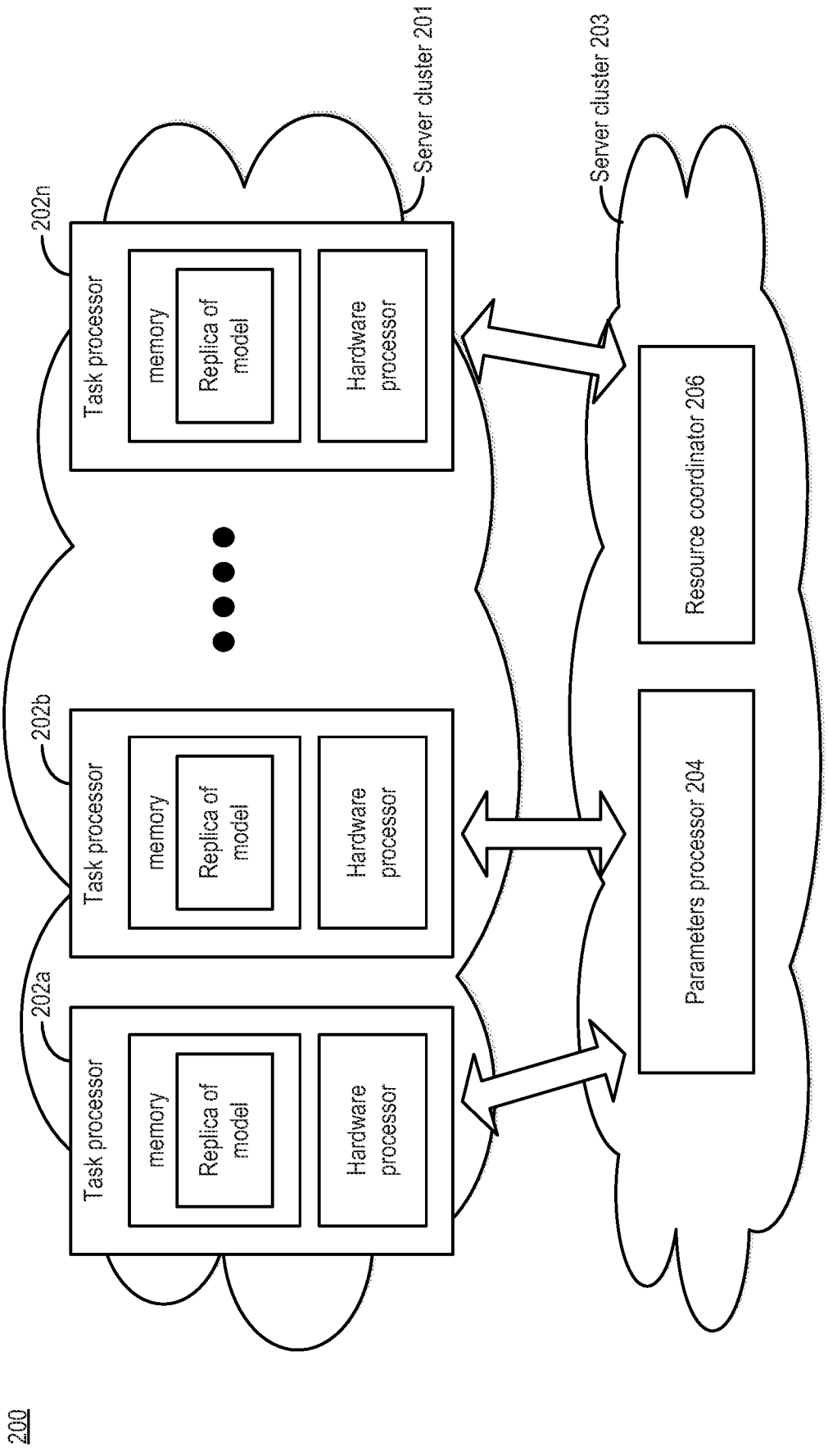
200
Server cluster 201
Server cluster 203
202n
202b
202a
Task processor
memory
Replica of model
Hardware processor
Resource coordinator 206
Parameters processor 204

FIG. 2A

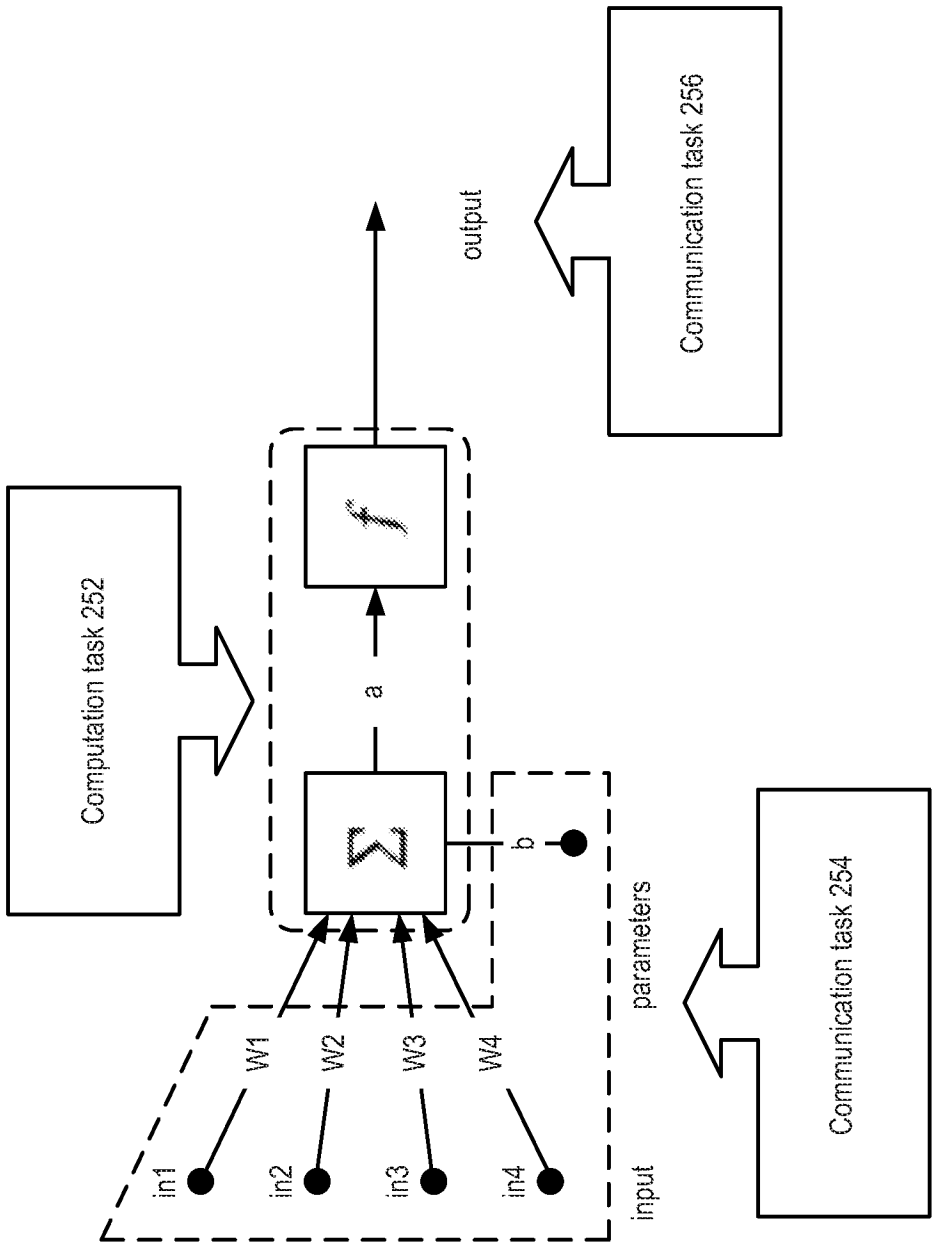
250
Computation task 252
Communication task 256
Communication task 254
output
a
b
parameters
input
in1
in2
in3
in4
W1
W2
W3
W4

FIG. 2B

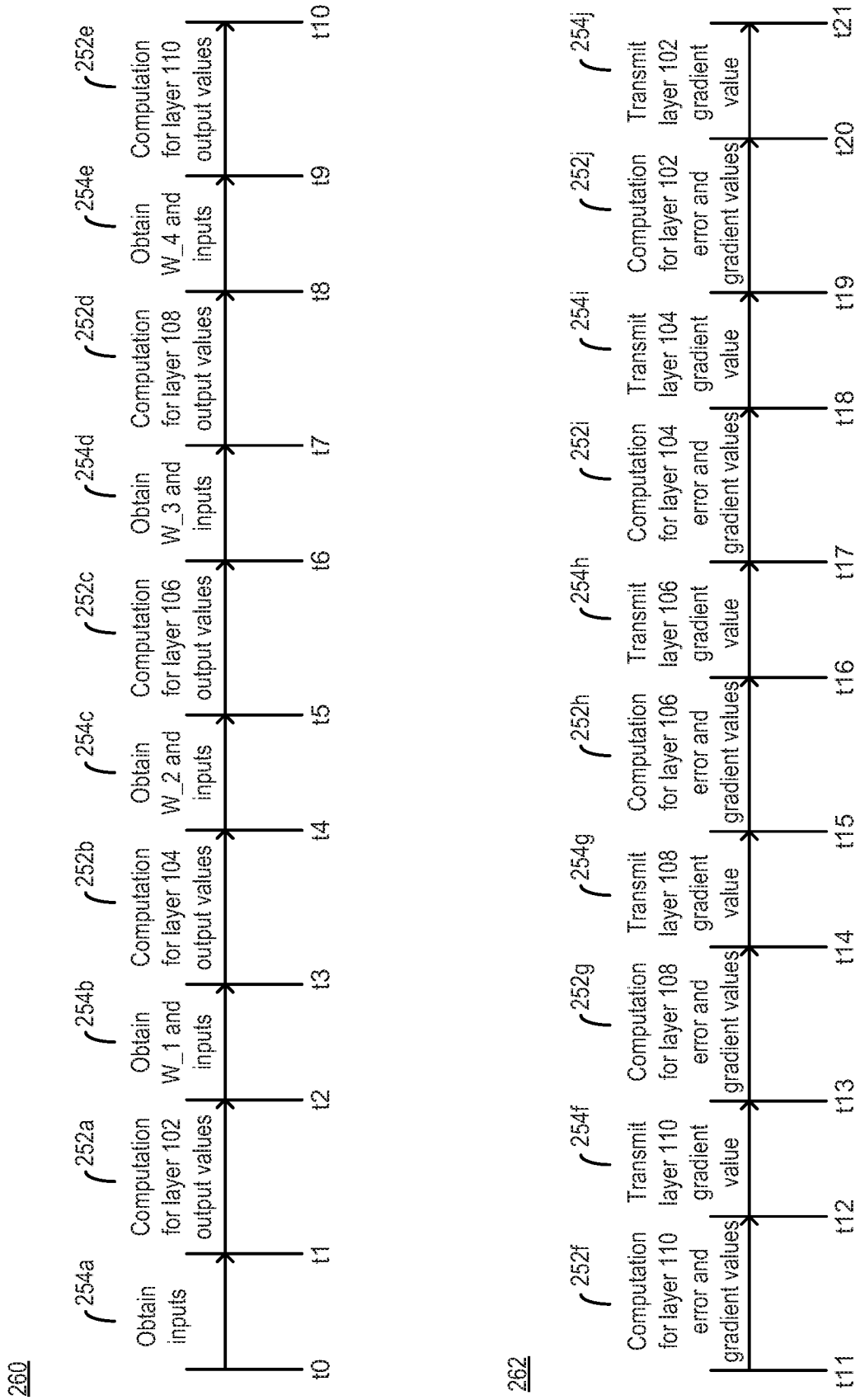
260
Obtain inputs
254a
Computation for layer 102 output values
252a
Obtain W_1 and inputs
254b
Computation for layer 104 output values
252b
Obtain W_2 and inputs
254c
Computation for layer 106 output values
252c
Obtain W_3 and inputs
254d
Computation for layer 108 output values
252d
Obtain W_4 and inputs
254e
Computation for layer 110 output values
252e
t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10
262
Computation for layer 110 error and gradient values
252f
Transmit layer 110 gradient value
254f
Computation for layer 108 error and gradient values
252g
Transmit layer 108 gradient value
254g
Computation for layer 106 error and gradient values
252h
Transmit layer 106 gradient value
254h
Computation for layer 104 error and gradient values
252i
Transmit layer 104 gradient value
254i
Computation for layer 102 error and gradient values
252j
Transmit layer 102 gradient value
254j
t11 t12 t13 t14 t15 t16 t17 t18 t19 t20 t21

FIG. 2C

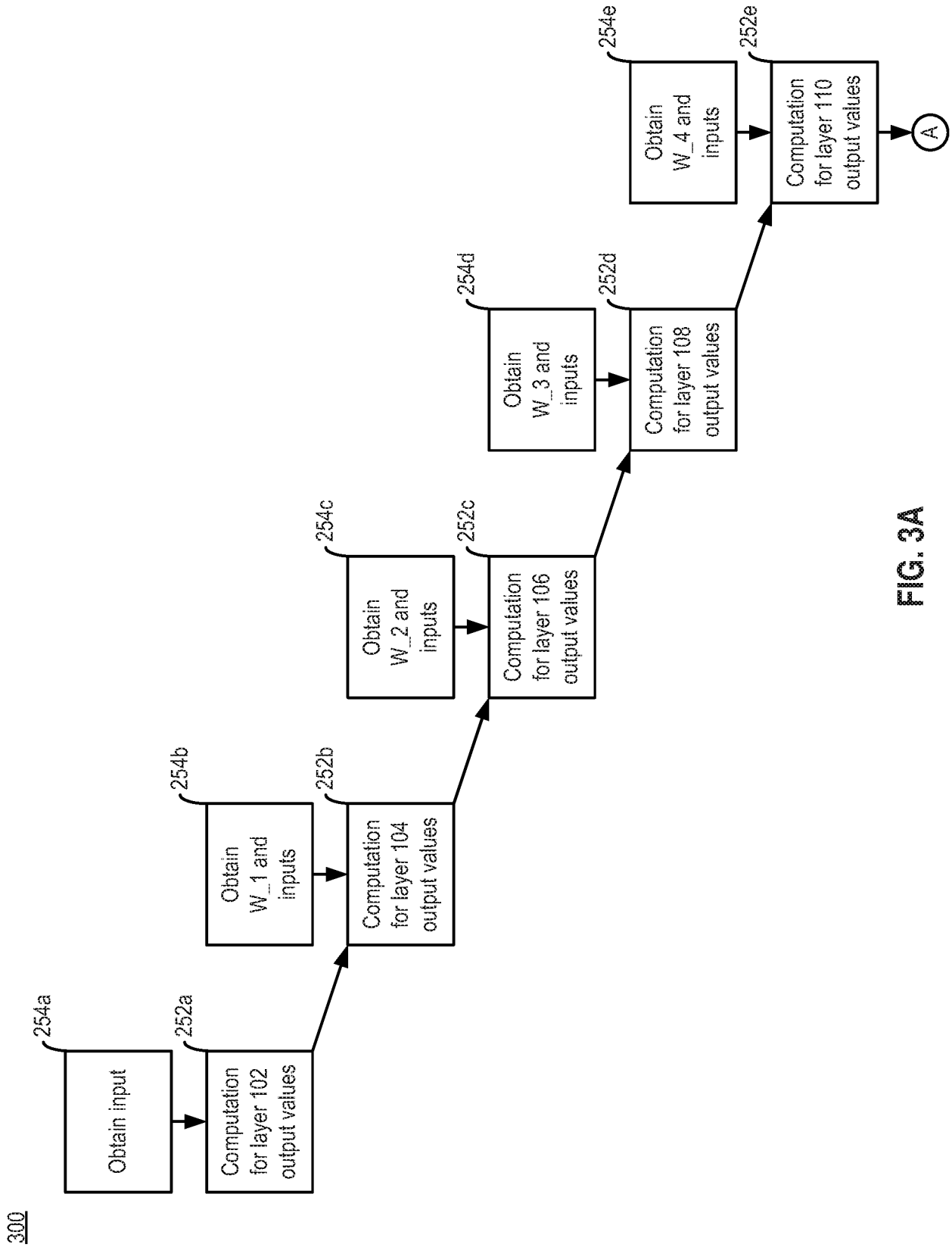
300
Obtain input
254a
Computation for layer 102 output values
252a
Obtain W_1 and inputs
254b
Computation for layer 104 output values
252b
Obtain W_2 and inputs
254c
Computation for layer 106 output values
252c
Obtain W_3 and inputs
254d
Computation for layer 108 output values
252d
Obtain W_4 and inputs
254e
Computation for layer 110 output values
252e
A

FIG. 3A

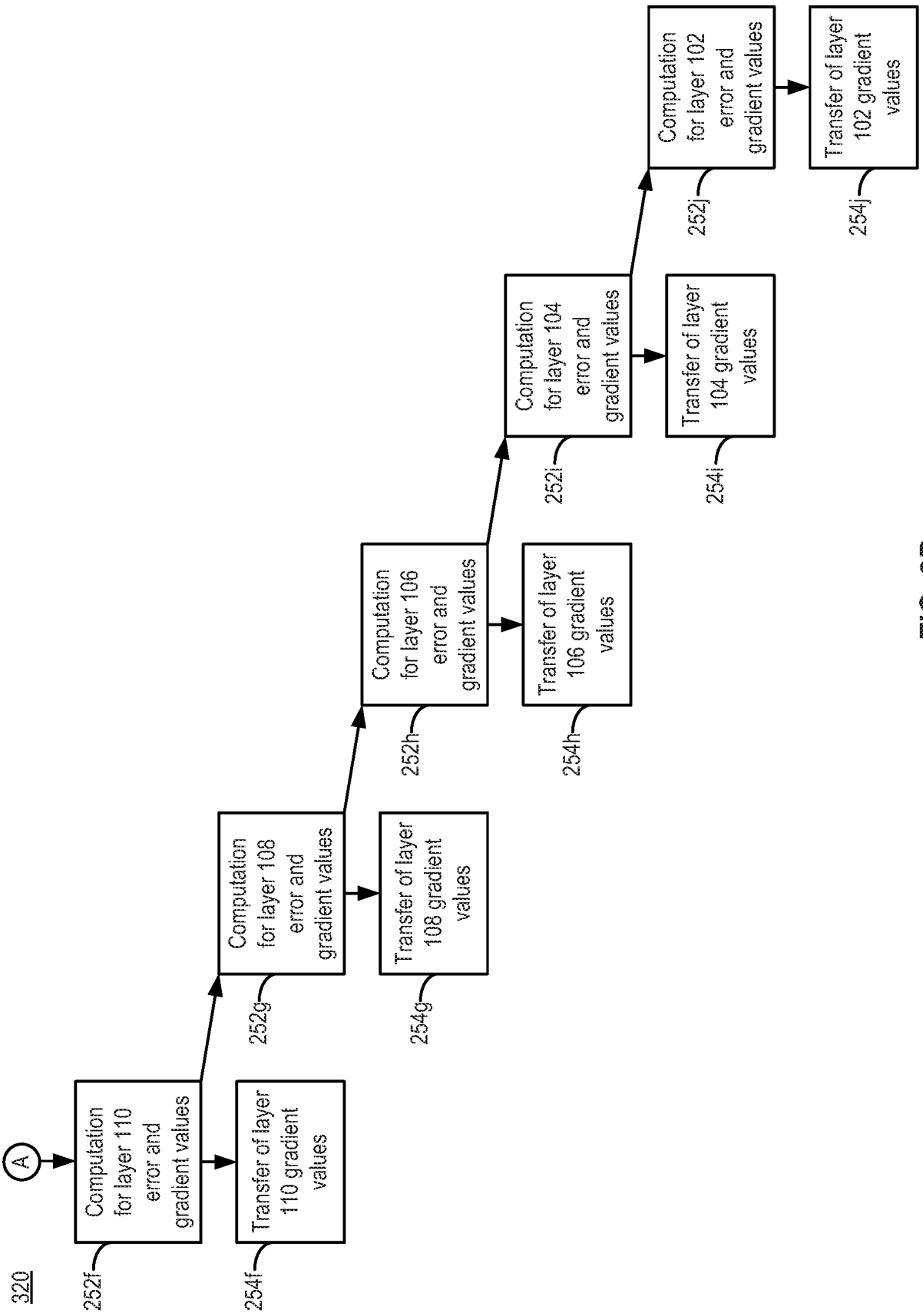
320
A
Computation for layer 110 error and gradient values
252f
Transfer of layer 110 gradient values
254f
Computation for layer 108 error and gradient values
252g
Transfer of layer 108 gradient values
254g
Computation for layer 106 error and gradient values
252h
Transfer of layer 106 gradient values
254h
Computation for layer 104 error and gradient values
252i
Transfer of layer 104 gradient values
254i
Computation for layer 102 error and gradient values
252j
Transfer of layer 102 gradient values
254j

FIG. 3B

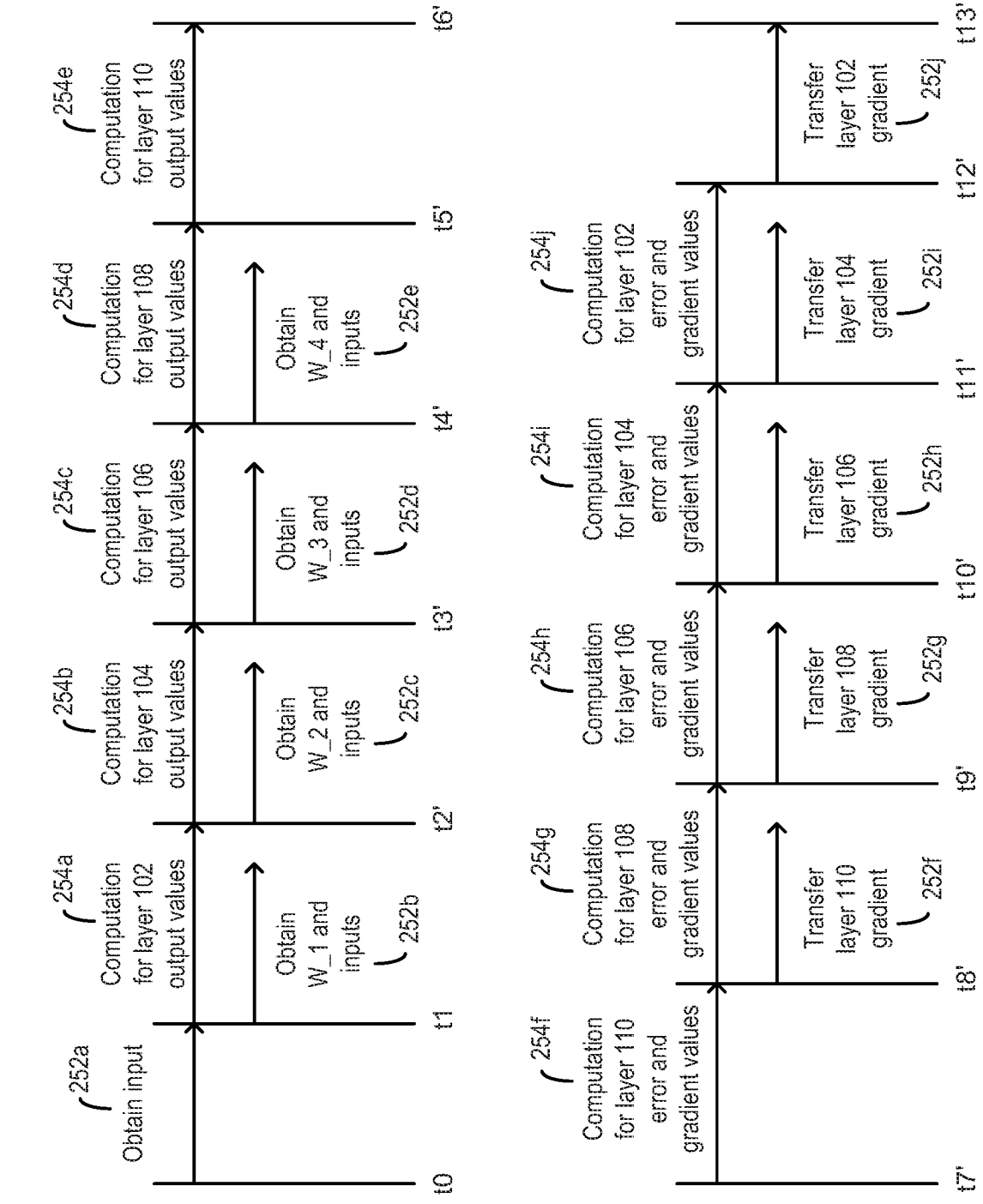
360
252a
Obtain input
254a
Computation for layer 102 output values
252b
Obtain W_1 and inputs
254b
Computation for layer 104 output values
252c
Obtain W_2 and inputs
254c
Computation for layer 106 output values
252d
Obtain W_3 and inputs
254d
Computation for layer 108 output values
252e
Obtain W_4 and inputs
254e
Computation for layer 110 output values
t0
t1
t2'
t3'
t4'
t5'
t6'
254f
Computation for layer 110 error and gradient values
252f
Transfer layer 110 gradient
254g
Computation for layer 108 error and gradient values
252g
Transfer layer 108 gradient
254h
Computation for layer 106 error and gradient values
252h
Transfer layer 106 gradient
254i
Computation for layer 104 error and gradient values
252i
Transfer layer 104 gradient
254j
Computation for layer 102 error and gradient values
252j
Transfer layer 102 gradient
t7'
t8'
t9'
t10'
t11'
t12'
t13'

FIG. 3C

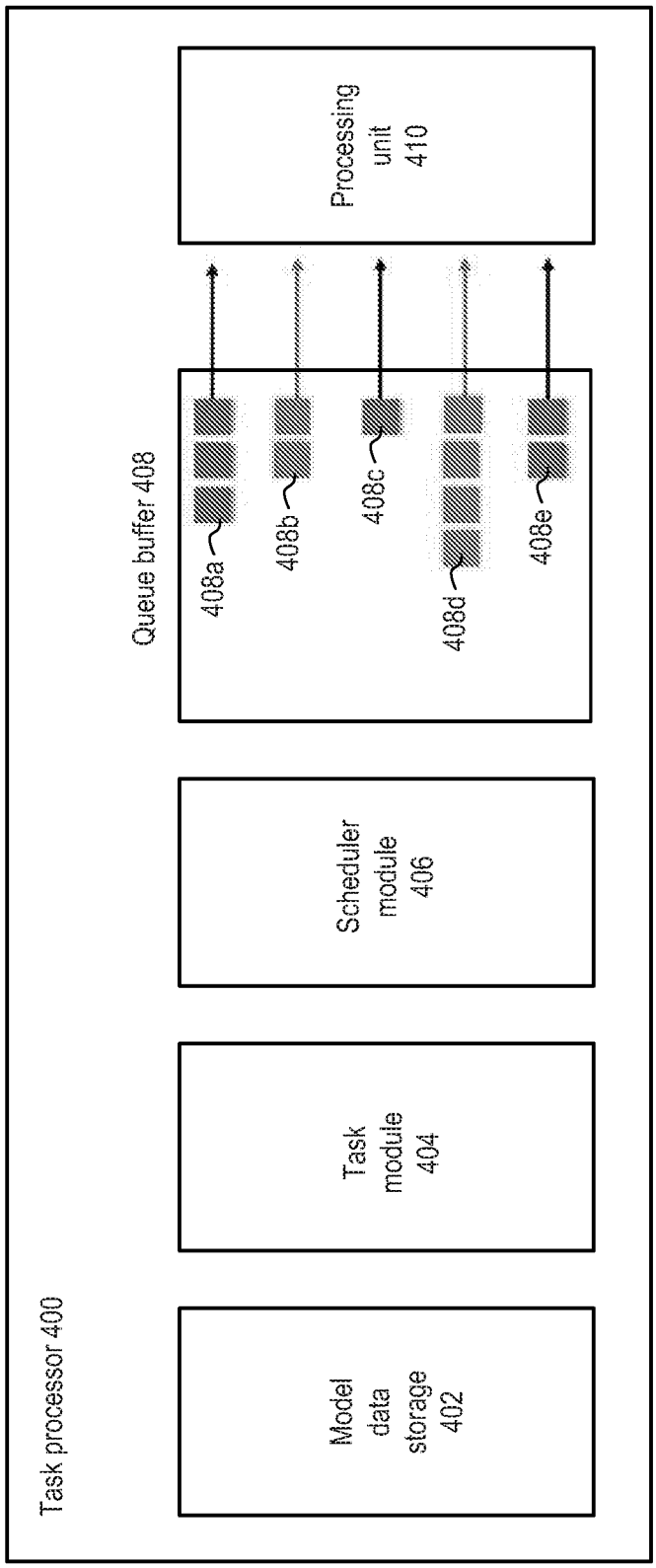
Task processor 400
Model data storage 402
Task module 404
Scheduler module 406
Queue buffer 408
408a
408b
408c
408d
408e
Processing unit 410

FIG. 4A

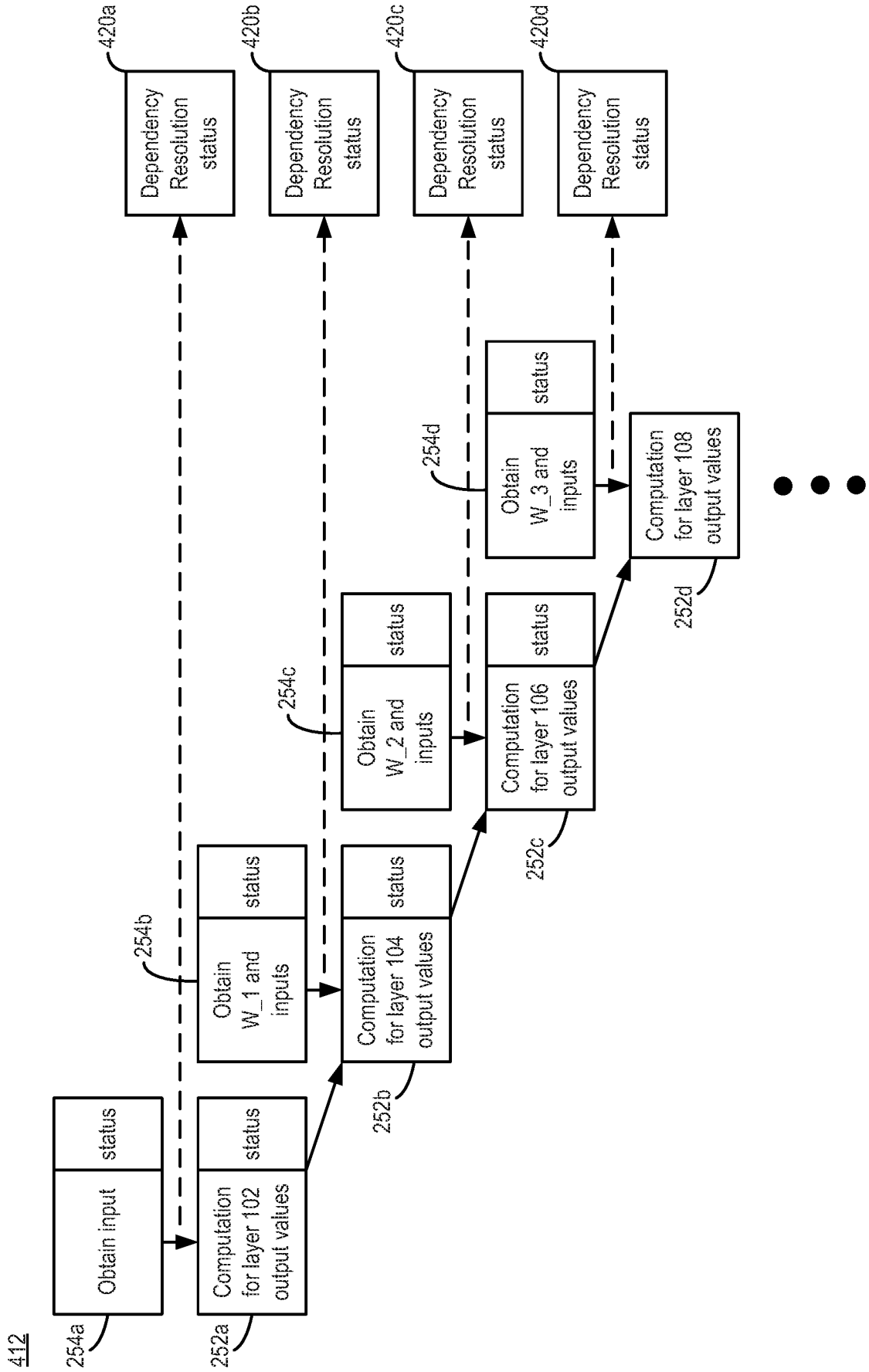
412
Obtain input
status
254a
Computation for layer 102 output values
status
252a
Obtain W_1 and inputs
status
254b
Computation for layer 104 output values
status
252b
Obtain W_2 and inputs
status
254c
Computation for layer 106 output values
status
252c
Obtain W_3 and inputs
status
254d
Computation for layer 108 output values
252d
Dependency Resolution status
420a
Dependency Resolution status
420b
Dependency Resolution status
420c
Dependency Resolution status
420d

FIG. 4B

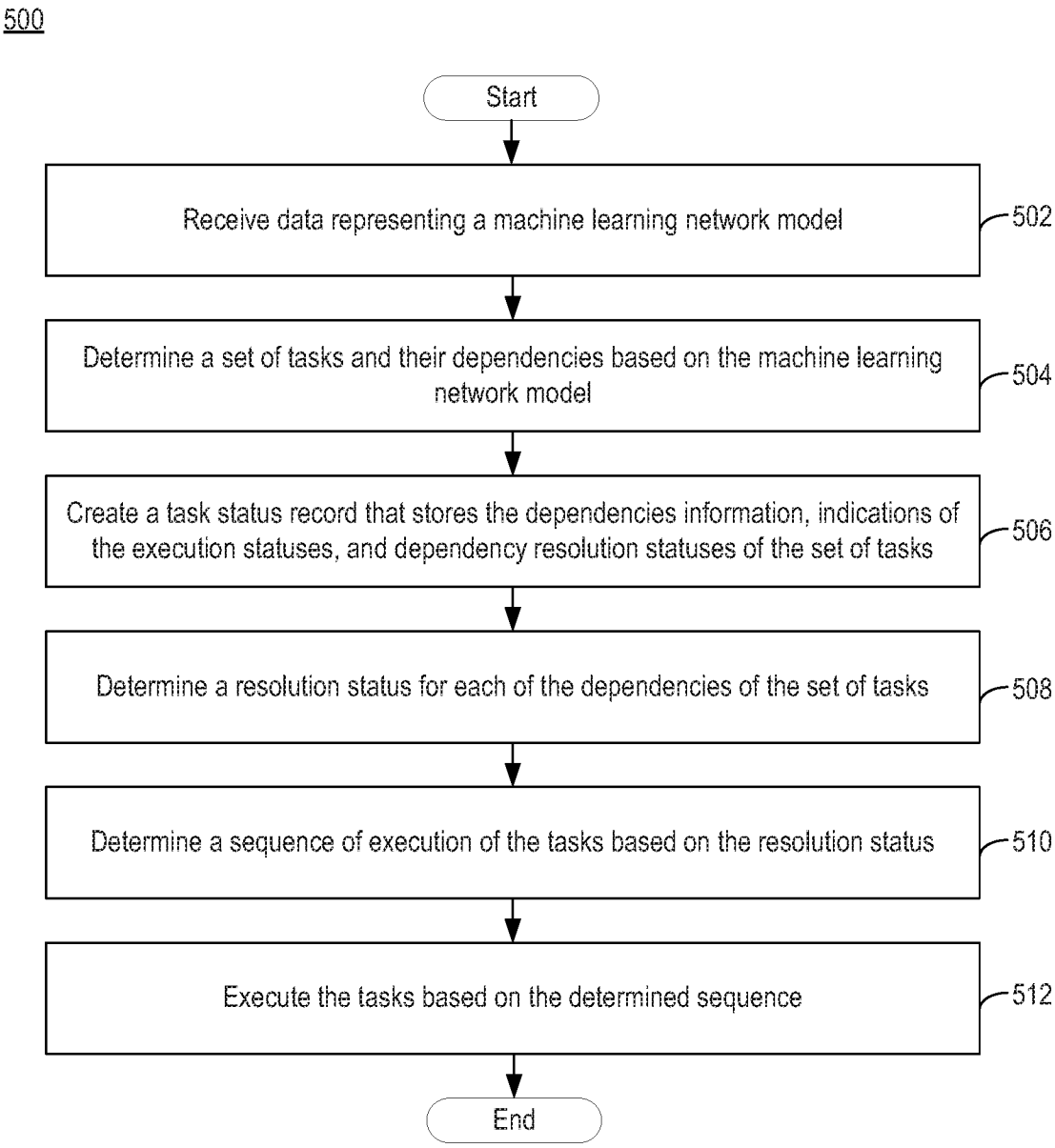
500
Start
Receive data representing a machine learning network model
502
Determine a set of tasks and their dependencies based on the machine learning network model
504
Create a task status record that stores the dependencies information, indications of the execution statuses, and dependency resolution statuses of the set of tasks
506
Determine a resolution status for each of the dependencies of the set of tasks
508
Determine a sequence of execution of the tasks based on the resolution status
510
Execute the tasks based on the determined sequence
512
End

FIG. 5

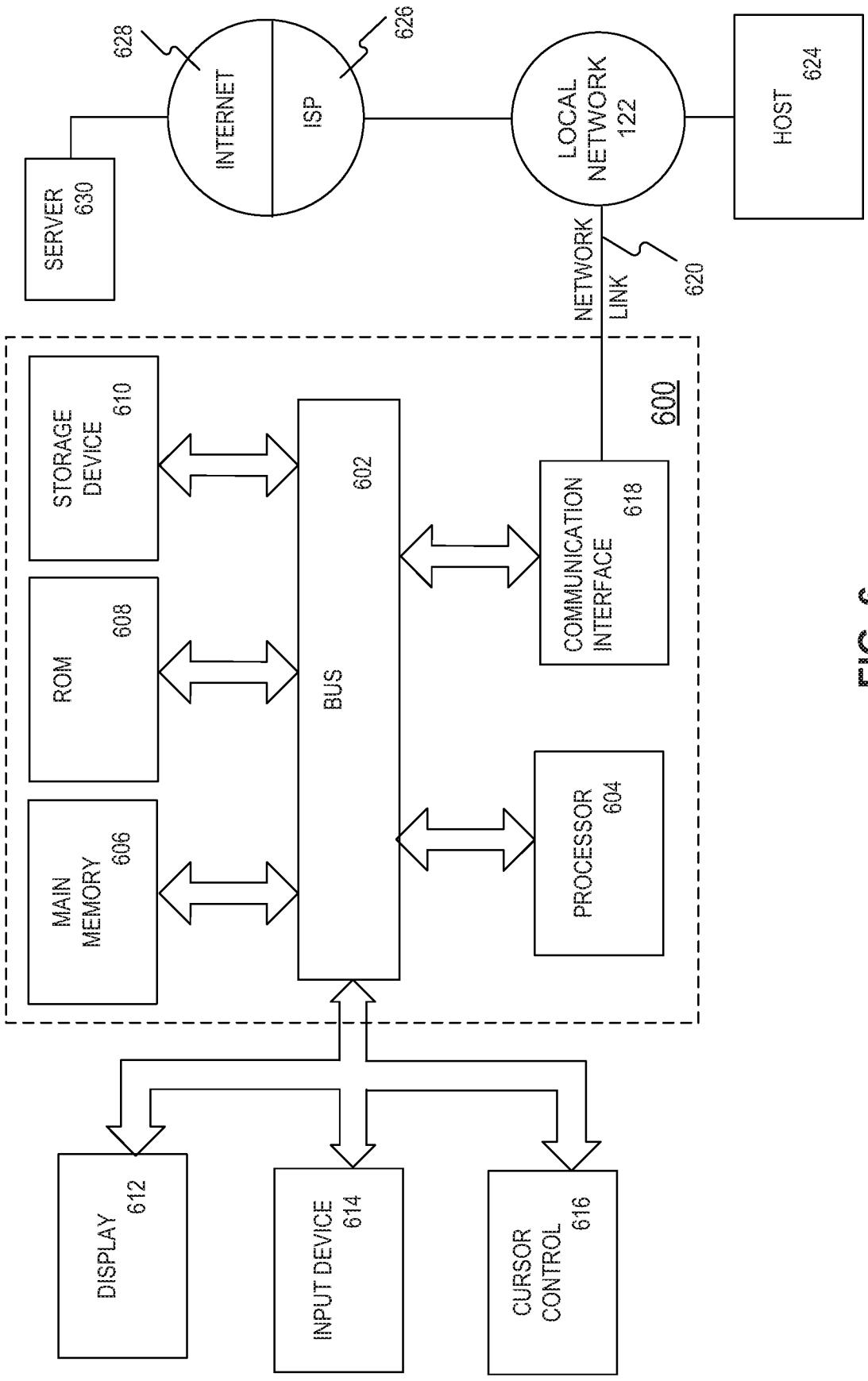
628
INTERNET
ISP
626
SERVER
630
LOCAL
NETWORK
122
HOST
624
NETWORK
LINK
620
600
STORAGE
DEVICE
610
602
COMMUNICATION
INTERFACE
618
ROM
608
BUS
MAIN
MEMORY
606
PROCESSOR
604
DISPLAY
612
INPUT DEVICE
614
CURSOR
CONTROL
616

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2017/087570

A. CLASSIFICATION OF SUBJECT MATTER

G06F 19/24(2011.01)i

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)

G06F

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)

CNKI,CNPAT,WPI,EPODOC:machine,learning,distribut+,dependen+,status+,assign+, task+,RNN,RBM

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2011131163 A1 (MICROSOFT CORP.) 02 June 2011 (2011-06-02) abstract, description paragraphs [0023], [0025], [0078], figures 1-3, 10, 12	1-20
A	US 2013290223 A1 (YAHOO! INC.) 31 October 2013 (2013-10-31) the whole document	1-20
A	US 2014222730 A1 (CISCO TECH., INC.) 07 August 2014 (2014-08-07) the whole document	1-20
A	HECKERMAN, David et al. Dependency Networks for Inference, Collaborative Filtering,and Data Visualization. ". Journal of Machine Learning Research 1 (2000)." *31 Dec.2000(31.12.2000), page 49-75,* ISSN: 1532-4435,	1-20

☐ Further documents are listed in the continuation of Box C. ☑ 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
27 February 2018	**12 March 2018**
Name and mailing address of the ISA/CN **STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088** **China** Facsimile No. **(86-10)62019451**	Authorized officer **GAO,Dandan** Telephone No. **(86-10)53961301**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2017/087570

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 2011131163 A1	02 June 2011	None	
US 2013290223 A1	31 October 2013	None	
US 2014222730 A1	07 August 2014	None	