(19) DANMARK

(10) DK/EP 3298545 T3

(12) Oversættelse af europæisk patentskrift

Patent- og Varemærkestyrelsen

(51) Int.Cl.: *G 06 N 3/0464 (2023.01)* *G 06 N 3/048 (2023.01)* *G 06 N 3/063 (2023.01)*

(45) Oversættelsen bekendtgjort den: **2024-01-02**

(80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2023-10-25**

(86) Europæisk ansøgning nr.: **16724517.4**

(86) Europæisk indleveringsdag: **2016-04-29**

(87) Den europæiske ansøgnings publiceringsdag: **2018-03-28**

(86) International ansøgning nr.: **US2016029986**

(87) Internationalt publikationsnr.: **WO2016186813**

(30) Prioritet: **2015-05-21 US 201562165022 P** **2015-09-03 US 201514845117**

(84) Designerede stater: **AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR**

(73) Patenthaver: **Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA**

(72) Opfinder: **THORSON, Gregory Michael, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA**
CLARK, Christopher Aaron, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA
LUU, Dan, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA

(74) Fuldmægtig i Danmark: **Ijon AB, Nordenskiöldsgatan 11A, 21119 Malmö, Sverige**

(54) Benævnelse: **VEKTORBEHANDLINGSENHED I EN PROCESSOR TIL ET NEURALT NETVÆRK**

(56) Fremdragne publikationer:
EP-A2- 0 422 348
US-A1- 2007 086 655
US-A1- 2011 029 471
US-A1- 2014 180 989
US-A1- 2014 288 928
US-A1- 2014 337 262
US-B1- 8 924 455
KUNG S: "VLSI Array processors", IEEE ASSP MAGAZINE, IEEE, US, vol. 2, no. 3, 1 July 1985 (1985-07-01), pages 4-22, XP011370547, ISSN: 0740-7467, DOI: 10.1109/MASSP.1985.1163741
Alex Krizhevsky ET AL: "ImageNet classification with deep convolutional neural networks", The 26th annual conference on Neural Information Processing Systems (NIPS'25): 3-8 December 2012, 6 December 2012 (2012-12-06), XP055113686, Retrieved from the Internet: URL:http://books.nips.cc/papers/files/nips25/NIPS2012_0534.pdf [retrieved on 2014-04-11]
LECUN YANN A ET AL: "Efficient BackProp", 2012, BIG DATA ANALYTICS IN THE SOCIAL AND UBIQUITOUS CONTEXT : 5TH INTERNATIONAL WORKSHOP ON MODELING SOCIAL MEDIA, MSM 2014, 5TH INTERNATIONAL WORKSHOP ON MINING UBIQUITOUS AND SOCIAL ENVIRONMENTS, MUSE 2014 AND FIRST INTERNATIONAL WORKSHOP ON MACHINE LE, XP047195726, ISBN: 978-3-642-17318-9

Fortsættes ...

DESCRIPTION

BACKGROUND

[0001] This specification relates to computing neural network inferences in hardware.

[0002] Neural networks are machine learning models that employ one or more layers to generate an output, e.g., a classification, for a received input. Some neural networks include one or more hidden layers in addition to an output layer. The output of each hidden layer is used as input to the next layer in the network, i.e., the next hidden layer or the output layer of the network. Each layer of the network generates an output from a received input in accordance with current values of a respective set of parameters.

[0003] EP 0,422,348 discloses a two dimensional array of processing elements connected to receive weight inputs and multiplexed data inputs for operation in feedforward, partially or fully connected neural network mode or in cooperative, competitive neutral network mode.

[0004] US 2014/180989 A1 discloses a parallel convolutional neural network. The CNN is implemented by a plurality of convolutional neural networks each on a respective processing node. Each CNN has a plurality of layers. A subset of the layers are interconnected between processing nodes such that activations are fed forward across nodes. The remaining subset is not so interconnected

SUMMARY

[0005] The invention is defined by the independent claims. The dependent claims define useful embodiments,

[0006] In general, this specification describes a special-purpose hardware circuit that computes neural network inferences.

[0007] In general, one innovative aspect of the subject matter described in this specification can be embodied in a circuit for performing neural network computations for a neural network comprising a plurality of layers, the circuit comprising: activation circuitry configured to receive a vector of accumulated values and configured to apply a function to each accumulated value to generate a vector of activation values; and normalization circuitry coupled to the activation circuitry and configured to generate a respective normalized value for each activation value.
Implementations can include one or more of the following features. The activation circuitry receives the vector of accumulated values from a systolic array in the circuit. The normalization circuitry comprises a plurality of normalization register columns, each normalization register column comprising a plurality of normalization registers connected in series, each

normalization register column configured to receive a distinct activation value, a respective normalization unit in the normalization register column configured to calculate a respective normalized value. Each normalization unit is configured to pass the distinct activation value to an adjacent normalization unit. Each normalization unit is configured to: receive a respective activation value; generate a respective intermediate normalized value from the respective activation value; and send the respective intermediate normalized value to one or more neighboring normalization units. Generating the respective intermediate normalized value comprises generating a square of the respective activation value. Each normalization unit is further configured to:
receive, from one or more neighboring normalization units, one or more intermediate normalized values generated from activation values; sum each intermediate normalized value to generate an index; use the index to access one or more values from a lookup table; generate a scaling factor from the one or more values and the index; and generate the respective normalized value from the scaling factor and the respective activation value. Pooling circuitry configured to receive the normalized values and configured to pool the normalized values to generate a pooled value. The pooling circuitry is configured to store the plurality of normalized values in a plurality of registers and a plurality of memory units, where the plurality of registers and the plurality of memory units are connected in series, where each register stores one normalized value and each memory unit stores a plurality of normalized values, where the pooling circuitry is configured to, after every clock cycle, shift a given normalized value to a subsequent register or memory unit, and where the pooling circuitry is configured to generate the pooled value from the normalized values. Pooling circuitry configured to receive the activation values and configured to pool the activation values to generate a pooled value. The pooling circuitry is configured to store the plurality of activation values in a plurality of registers and a plurality of memory units, where the plurality of registers and the plurality of memory units are connected in series, where each register stores one normalized value and each memory unit stores a plurality of activation values, where the pooling circuitry is configured to, after every clock cycle, shift a given activation value to a subsequent register or memory unit, and where the pooling circuitry is configured to generate the pooled value from the activation values.

[0008] Particular embodiments of the subject matter described in this specification can be implemented so as to realize one or more of the following advantages. Multiple activation values for each neural network layer of a neural network can be computed during a given clock cycle. Optionally, the processor can generate multiple normalized values from the activation values during another given clock cycle. The processor can also optionally generate pooled values from the normalized values or the activation values. The processor is capable of taking a new accumulated sum in each clock cycle and producing an activated, normalized, and pooled result in each clock cycle, thereby pipelining computations.

[0009] The details of one or more embodiments of the subject matter of this specification are set forth in the accompanying drawings and the description below. Other features, aspects, and advantages of the subject matter will become apparent from the description, the drawings, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010]

FIG. 1 is a flow diagram of an example method for performing a computation for a given layer of a neural network.

FIG. 2 shows an example neural network processing system.

FIG. 3 shows an example architecture including a matrix computation unit.

FIG. 4 shows an example architecture of a cell inside a systolic array.

FIG. 5 shows an example architecture of a vector computation unit.

FIG. 6 shows an example architecture for normalization circuitry.

FIG. 7 shows another example architecture for normalization circuitry with sample activation values.

FIG. 8 shows an example architecture for a normalization unit inside the normalization circuitry.

FIG. 9 shows an example architecture for pooling circuitry.

[0011] Like reference numbers and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0012] A neural network having multiple layers can be used to compute inferences. For example, given an input, the neural network can compute an inference for the input. The neural network computes this inference by processing the input through each of the layers of the neural network. In particular, the layers of the neural network are arranged in a sequence, each with a respective set of weights. Each layer receives an input and processes the input in accordance with the set of weights for the layer to generate an output.

[0013] Therefore, in order to compute an inference from a received input, the neural network receives the input and processes it through each of the neural network layers in the sequence to generate the inference, with the output from one neural network layer being provided as input to the next neural network layer. Data inputs to a neural network layer, e.g., either the input to the neural network or the outputs of the layer below the layer in the sequence, to a neural network layer can be referred to as activation inputs to the layer.

[0014] In some implementations, the layers of the neural network are arranged in a directed graph. That is, any particular layer can receive multiple inputs, multiple outputs, or both. The layers of the neural network can also be arranged such that an output of a layer can be sent back as an input to a previous layer.

[0015] Some neural networks normalize outputs from one or more neural network layers to generate normalized values that are used as inputs to subsequent neural network layers. Normalizing the outputs can help ensure the normalized values remain within expected domains for the inputs of the subsequent neural network layers. This can reduce errors in inference calculations.

[0016] Some neural networks pool outputs from one or more neural network layers to generate pooled values that are used as inputs to subsequent neural network layers. In some implementations, the neural network pools a group of outputs by determining a maximum or average of the group of outputs and using the maximum or average as the pooled output for the group. Pooling the outputs can maintain some spatial invariance so the outputs arranged in various configurations can be processed to have the same inference. Pooling the outputs can also reduce dimensionality of inputs received at the subsequent neural network layers while maintaining desired characteristics of the outputs before pooling, which can improve efficiency without significantly compromising the quality of inferences generated by the neural networks.

[0017] This specification describes special-purpose hardware circuitry that optionally performs normalization, pooling, or both on outputs of one or more neural network layers.

[0018] FIG. 1 is a flow diagram of an example process 100 for performing a computation for a given layer of a neural network using a special-purpose hardware circuit. For convenience, the method 100 will be described with respect to a system having one or more circuits that performs the method 100.The method 100 can be performed for each layer of the neural network in order to compute an inference from a received input.

[0019] The system receives sets of weight inputs (step 102) and sets of activation inputs (step 104) for the given layer. The sets of weight inputs and the sets of activation inputs can be received from dynamic memory and a unified buffer, respectively, of the special-purpose hardware circuit. In some implementations, both the sets of weight inputs and the sets of activation inputs can be received from the unified buffer.

[0020] The system generates accumulated values from the weight inputs and the activation inputs using a matrix multiplication unit of the special-purpose hardware circuit (step 106). In some implementations, the accumulated values are dot products of the sets of weight inputs and the sets of activation inputs. That is, for one set of weights, which is a subset of all weights in the layer, the system can multiply each weight input with each activation input and sum the products together to form an accumulated value. The system can then compute dot products of other set of weights with other sets of activation inputs.

[0021] The system can generate a layer output from the accumulation values (step 108) using a vector computation unit of the special-purpose hardware circuit. In some implementations, the vector computation unit applies an activation function to the accumulated values, which will be described further below in reference to FIG. 5. The output of the layer can be stored in the unified buffer for use as an input to a subsequent layer in the neural network or can be used to determine the inference. The system finishes processing the neural network when a received input has been processed through each layer of the neural network to generate the inference for the received input.

[0022] FIG. 2 shows an example special-purpose integrated circuit 200 for performing neural network computations. The system 200 includes a host interface 202. The host interface 202 can receive instructions that include parameters for a neural network computation. The parameters can include one or more of the following: how many layers should be processed, corresponding sets of weight inputs for each layer of the model, an initial set of activation inputs, i.e., the input to the neural network from which the inference is to be computed, corresponding input and output sizes of each layer, a stride value for the neural network computation, and a type of layer to be processed, e.g., a convolutional layer or a fully connected layer.

[0023] The host interface 202 can send the instructions to a sequencer 206, which converts the instructions into low level control signals that control the circuit to perform the neural network computations. In some implementations, the control signals regulate dataflow in the circuit, e.g., how the sets of weight inputs and the sets of activation inputs flow through the circuit. The sequencer 206 can send the control signals to a unified buffer 208, a matrix computation unit 212, and a vector computation unit 214. In some implementations, the sequencer 206 also sends control signals to a direct memory access engine 204 and dynamic memory 210. In some implementations, the sequencer 206 is a processor that generates control signals. The sequencer 206 can use timing of the control signals to, at appropriate times, send the control signals to each component of the circuit 200. In some other implementations, the host interface 202 passes in a control signal from an external processor.

[0024] The host interface 202 can send the sets of weight inputs and the initial set of activation inputs to the direct memory access engine 204. The direct memory access engine 204 can store the sets of activation inputs at the unified buffer 208. In some implementations, the direct memory access stores the sets of weights to dynamic memory 210, which can be a memory unit. In some implementations, the dynamic memory is located off of the circuit.

[0025] The unified buffer 208 is a memory buffer. It can be used to store the set of activation inputs from the direct memory access engine 204 and outputs of the vector computation unit 214. The vector computation unit will be described in more detail below with reference to FIG. 5. The direct memory access engine 204 can also read the outputs of the vector computation unit 214 from the unified buffer 208.

[0026] The dynamic memory 210 and the unified buffer 208 can send the sets of weight inputs

and the sets of activation inputs, respectively, to the matrix computation unit 212. In some implementations, the matrix computation unit 212 is a two-dimensional systolic array. The matrix computation unit 212 can also be a one-dimensional systolic array or other circuitry that can perform mathematical operations, e.g., multiplication and addition. In some implementations, the matrix computation unit 212 is a general purpose matrix processor.

[0027] The matrix computation unit 212 can process the weight inputs and the activation inputs and provide a vector of outputs to the vector computation unit 214. In some implementations, the matrix computation unit sends the vector of outputs to the unified buffer 208, which sends the vector of outputs to the vector computation unit 214. The vector computation unit can process the vector of outputs and store a vector of processed outputs to the unified buffer 208. The vector of processed outputs can be used as activation inputs to the matrix computation unit 212, e.g., for use in a subsequent layer in the neural network. The matrix computation unit 212 and the vector computation unit 214 will be described in more detail below with reference to FIG. 3 and FIG. 5, respectively.

[0028] FIG. 3 shows an example architecture 300 including a matrix computation unit. The matrix computation unit is a two-dimensional systolic array 306. The array 306 includes multiple cells 304. In some implementations, a first dimension 320 of the systolic array 306 corresponds to columns of cells and a second dimension 322 of the systolic array 306 corresponds to rows of cells. The systolic array can have more rows than columns, more columns than rows, or an equal number of columns and rows.

[0029] In the illustrated example, value loaders 302 send activation inputs to rows of the array 306 and a weight fetcher interface 308 sends weight inputs to columns of the array 306. In some other implementations, however, activation inputs are transferred to the columns and weight inputs are transferred to the rows of the array 306.

[0030] The value loaders 302 can receive the activation inputs from a unified buffer, e.g., the unified buffer 208 of FIG. 2. Each value loader can send a corresponding activation input to a distinct left-most cell of the array 306. For example, value loader 312 can send an activation input to cell 314. The value loader can also send the activation input to an adjacent value loader, and the activation input can be used at another left-most cell of the array 306. This allows activation inputs to be shifted for use in another particular cell of the array 306.

[0031] The weight fetcher interface 308 can receive the weight input from a memory unit, e.g., the dynamic memory 210 of FIG. 2. The weight fetcher interface 308 can send a corresponding weight input to a distinct top-most cell of the array 306. For example, the weight fetcher interface 308 can send weight inputs to cells 314 and 316.

[0032] In some implementations, a host interface, e.g., the host interface 202 of FIG. 2, shifts activation inputs throughout the array 306 along one dimension, e.g., to the right, while shifting weight inputs throughout the array 306 along another dimension, e.g., to the bottom. For example, over one clock cycle, the activation input at cell 314 can shift to an activation register

in cell 316, which is to the right of cell 314. Similarly, the weight input at cell 316 can shift to a weight register at cell 318, which is below cell 314.

[0033] On each clock cycle, each cell can process a given weight input, a given activation input, and an accumulated output from an adjacent cell to generate an accumulated output. The accumulated output can also be passed to the adjacent cell along the same dimension as the given weight input. An individual cell is described further below with reference FIG. 4.

[0034] The accumulated output can be passed along the same column as the weight input, e.g., towards the bottom of the column in the array 306. In some implementations, at the bottom of each column, the array 306 can include accumulator units 310 that store and accumulate each accumulated output from each column when performing calculations with layers having more activation inputs than rows. In some implementations, each accumulator unit stores multiple parallel accumulations. This will be described further below with reference to FIG. 6. The accumulator units 310 can accumulate each accumulated output to generate a final accumulated value. The final accumulated value can be transferred to a vector computation unit, e.g., the vector computation unit 502 of FIG. 5. In some other implementations, the accumulator units 310 passes the accumulated values to the vector computation unit without performing any accumulations when processing layers with layers having fewer activating inputs than rows.

[0035] FIG. 4 shows an example architecture 400 of a cell inside a systolic array, e.g., the systolic array 306 of FIG. 3.

[0036] The cell can include an activation register 406 that stores an activation input. The activation register can receive the activation input from a left adjacent cell, i.e., an adjacent cell located to the left of the given cell, or from a unified buffer, depending on the position of the cell within the systolic array. The cell can include a weight register 402 that stores a weight input. The weight input can be transferred from a top adjacent cell or from a weight fetcher interface, depending on the position of the cell within the systolic array. The cell can also include a sum in register 404. The sum in register 404 can store an accumulated value from the top adjacent cell. Multiplication circuitry 408 can be used to multiply the weight input from the weight register 402 with the activation input from the activation register 406. The multiplication circuitry 408 can output the product to summation circuitry 410.

[0037] The summation circuitry can sum the product and the accumulated value from the sum in register 404 to generate a new accumulated value. The summation circuitry 410 can then send the new accumulated value to another sum in register located in a bottom adjacent cell. The new accumulated value can be used as an operand for a summation in the bottom adjacent cell.

[0038] The cell can also shift the weight input and the activation input to adjacent cells for processing. For example, the weight register 402 can send the weight input to another weight register in the bottom adjacent cell. The activation register 406 can send the activation input to

another activation register in the right adjacent cell. Both the weight input and the activation input can therefore be reused by other cells in the array at a subsequent clock cycle.

[0039] In some implementations, the cell also includes a control register. The control register can store a control signal that determines whether the cell should shift either the weight input or the activation input to adjacent cells. In some implementations, shifting the weight input or the activation input takes one or more clock cycles. The control signal can also determine whether the activation input or weight inputs are transferred to the multiplication circuitry 408, or can determine whether the multiplication circuitry 408 operates on the activation and weight inputs. The control signal can also be passed to one or more adjacent cells, e.g., using a wire.

[0040] In some implementations, weights are pre-shifted into a weight path register 412. The weight path register 412 can receive the weight input, e.g., from a top adjacent cell, and transfer the weight input to the weight register 402 based on the control signal. The weight register 402 can statically store the weight input such that as activation inputs are transferred to the cell, e.g., through the activation register 406, over multiple clock cycles, the weight input remains within the cell and is not transferred to an adjacent cell. Therefore, the weight input can be applied to multiple activation inputs, e.g., using the multiplication circuitry 408, and respective accumulated values can be transferred to an adjacent cell.

[0041] FIG. 5 shows an example architecture 500 of a vector computation unit 502. The vector computation unit 502 can receive a vector of accumulated values from a matrix computation unit, e.g., the matrix computation unit described in reference to FIG. 2.

[0042] The vector computation unit 502 can process the vector of accumulated values at the activation unit 404. In some implementations, the activation unit includes circuitry that applies a non-linear function to each accumulated value to generate activation values. For example, the non-linear function can be tanh(x), where x is an accumulated value.

[0043] Optionally, the vector computation unit 502 can normalize the activation values in normalization circuitry 506 that generates normalized values from the activation values.

[0044] Also optionally, the vector computation unit 502 can pool values, either activation values or normalized values, using pooling circuitry 508. The pooling circuitry 508 can apply an aggregation function to one or more of the normalized values to generate pooled values. In some implementations, the aggregation functions are functions that return a maximum, minimum, or average of the normalized values or of a subset of the normalized values.

[0045] Control signals 510 can be transferred, e.g., by the sequencer 206 of FIG. 2, and can regulate how the vector computation unit 502 processes the vector of accumulated values. That is, the control signals 510 can regulate whether the activation values are pooled, normalized, or both. The control signals 510 can also specify the activation, normalization, or pooling functions, as well as other parameters for normalization and pooling, e.g., a stride value.

[0046] The vector computation unit 502 can send values, e.g., activation values, normalized values, or pooled values, to a unified buffer, e.g., the unified buffer 208 of FIG. 2.

[0047] In some implementations, the pooling unit 508 receives the activation values instead of the normalization circuitry 506 and stores the pooled values in the unified buffer. In some implementations, the pooling unit 508 sends the pooled values to the normalization circuitry 506, which generates normalized values to be stored in the unified buffer.

[0048] FIG. 6 shows an example architecture 600 for normalization circuitry, e.g., the normalization circuitry 506 of FIG. 5. The normalization circuitry can, for each clock cycle, receive a vector of activated values from activation circuitry 602, e.g., the activation circuitry 504 of FIG. 5. Depending on the value of a system parameter, the normalization circuitry can either pass the vector of activated values to pooling circuitry, i.e., without normalizing the activated values, or generate a vector of normalized values from the vector of activated values. For example, if the system parameter, e.g., provided by a user, instructs the circuit to pass the vector of activated values to pooling circuitry, e.g., the user does not want to normalize the values, the system parameter can be a signal to a multiplexor that passes the values directly to the pooling circuitry and skips the normalization circuitry.

[0049] In some implementations, the vector of activated values includes activated values generated by applying an activation function to accumulated values generated from activation inputs based on a set of weights inputs.

[0050] In some other implementations, the activated values for the set of weight inputs are staggered across multiple vectors of activated values because of delays caused when shifting activation and weight inputs. For example, a matrix computation unit can generate accumulated values A_0-A_n from a set of activation inputs and a set of weight inputs from Kernel A, accumulated values B_0-B_n from a set of activation inputs and a set of weight inputs from Kernel B, and accumulated values C_0-C_n from a set of activation inputs and a set of weight inputs from Kernel C. The accumulated values A_0-A_n and B_0-B_n can be generated over subsequent clock cycles because weight inputs and activation inputs are shifted across the matrix computation unit before corresponding accumulated values are computed, as described above in reference to FIG. 4. A_0 can be generated on clock cycle 0, A_1 and B_0 can be generated on clock cycle 1, A_2, B_1, and C_0 can be generated on clock cycle 2, A_n, B_{n-1}, and C_{n-2}can be generated on clock cycle n, and so forth. The matrix computation unit can generate a vector of accumulated values including A_0 and B_0 for clock cycle X and another vector of accumulated values including A_1 and B_1 for clock cycle X+1. Therefore, the accumulated values for a given kernel, e.g., A_0-A_n from Kernel A, can be spread out across multiple vectors of accumulated values over subsequent clock cycles in a staggered fashion.

[0051] As a result, the multiple vectors of accumulated values can become multiple vectors of activated values, e.g., after processing by the activation circuitry 504 of FIG. 5, and each of the

multiple vectors of activated values can be sent to a distinct normalization register column. In particular, the activation circuitry 602 can send each activated value from a vector of activated values to a distinct normalization register column 604-610. In particular, normalization registers 616-622 can each receive a respective activated value. A normalization register column can include a set of normalization registers connected in series. That is, an output of a first normalization register in the column can be sent as an input to a second normalization register in the column. In some implementations, each normalization register stores an activated value. In some other implementations, each normalization register also stores a square of the activated value. In some implementations, the normalization circuitry has as many normalization register columns as there are columns in the activation circuitry or in the systolic array.

[0052] In some implementations, before providing the vectors of activated values to the normalization register columns, the circuit sends the vectors to a squaring unit. The squaring unit can calculate a square of each activated value for use in computing normalized values, which will be described further below. The squaring unit can generate vectors of squared activated values, i.e., one for each vector of activated values, and send the vectors of squared activated values to the normalization register columns. In some other implementations, the squaring unit sends both the vectors of activated values and the vectors of squared activated values to the normalization register columns.

[0053] In some implementations, the normalization circuitry forms staggered groups, e.g., staggered groups 624 and 628, based on a normalization radius parameter. The normalization radius parameter can indicate a number of outputs from surrounding normalization registers to use when calculating a normalized value. The number of outputs can be equal to two times the normalization radius parameter. By way of illustration, staggered groups 624 and 628 are formed from a normalization radius parameter of 1. The staggered group 624 includes normalization units 632 and 618, and also includes zero register 636. Zero register 636 can always output a value of 0 and can serve as a buffer when calculating normalized values on edges of the normalization circuitry. The zero registers 635 and 638 can be included in a column of zero registers 612. An example of values inside the staggered groups will be described further below in reference to FIG. 7.

[0054] In some implementations, normalization units, e.g., normalization units 626, 630, use outputs from the staggered groups to generate a corresponding component, e.g., a square of activation values inside registers of the staggered group, used to compute a normalized value. For example, the components can be used to generate a sum of squares of all activated values. Normalization units can use the sum of squares to compute the normalized value, which will be described further below. In some implementations, there is a corresponding normalization unit for each staggered group.

[0055] The normalization circuitry can generate a normalized value for an activated value based on the staggered groups. For example, the normalized value for an activation value stored in normalization register 632 can be stored in normalization unit 626. In particular,

based on the staggered group 624, the normalization circuitry can compute a sum, e.g., using summation circuitry, of all of the squares generated by normalization registers inside the staggered group 624. The sum can be stored in the normalization unit 626. The sum can be a normalized value corresponding to an activated value. The normalization circuitry can continue to generate another corresponding normalized value for staggered group 628, which includes normalization registers 634, 640, and zero register 620, and the corresponding normalized value can be stored in normalization unit 630.

[0056] The normalization circuitry can form a vector of normalized values from the generated normalized values, e.g., which can be stored in the normalization units, and can send the vector of normalized values to pooling circuitry, if determined by a neural network parameter, or a unified buffer.

[0057] FIG. 7 shows another example architecture 700 for normalization circuitry with sample activated values inside normalization registers. The normalization radius parameter can be 1, as demonstrated in the staggered groups 724 and 728. In particular, staggered group 724 includes normalization registers 732 and 718 and zero register 736. Staggered group 728 includes zero register 738 and normalization registers 734 and 740.

[0058] The normalization registers 716-720, 732, 734, and 740 can store activated values, e.g., corresponding to columns from a systolic array. The notation AX,Y, e.g., A0,0 of normalization register 740, denotes an activated value corresponding to Column X in Clock Cycle Y.

[0059] As demonstrated in the figure, activated values are loaded in a staggered manner. For example, on Clock Cycle 0, activated values A0,0, A1,0, and A2,0 can be computed, but the normalization circuitry loads the three activated values over three clock cycles. In some implementations, the activated values are loaded in a non-staggered manner. That is, A0, 0, A1,0, and A2,0 can be loaded in one clock cycle.

[0060] N0 can be a normalized value for A0,1 stored in normalization register 726. N0 can be calculated based on the sum of squares of A0,1 and A1,1 and 0 (from zero register 736), which will be described below in reference to FIG. 8. Similarly, N1 can be a normalized value for A0,0 that is calculated based on the sum of squares of A0,0 and A1,0 and A2,0 (from register 720).

[0061] The normalization circuitry can compute normalized values for each activated value using a radius of 1. Other radii are possible. If the normalization circuitry has not yet loaded the activated values necessary for a normalization calculation, the normalization circuitry can shift the activated value to a subsequent normalization register until the necessary activated values are loaded. For example, to calculate a normalized value for activated value A0,2 stored in normalization register 716 requires an activated value A1,2 in light of a radius of 1. Activated value A1,2 can be loaded into normalization register 718 on a subsequent clock cycle, at which point, the normalization circuitry can compute a normalized value for activated value A0,2.

[0062] FIG. 8 shows an example architecture 800 for a normalization unit inside the normalization circuitry. The normalization unit can receive an activated value 802. In some implementations, the activated value 802 is passed to a subsequent normalization unit through a multiplexor 814, e.g., when the circuit determines the activated value 802 is in an incorrect position, i.e., the activated value needs to be stored at a subsequent normalization unit for a normalization calculation. The normalization circuitry can send a control signal to the multiplexor 814 to pass through a particular output, e.g., either a normalized value or an unaffected activation value.

[0063] In some implementations, the activated value is passed to square circuitry 804. The square circuitry 804 can generate a squared activated value 808, i.e., raise the activated value to the power of two. The square circuitry 804 can send the squared activated value 808 to neighboring normalization units, e.g., other normalization units in the same staggered group of the normalization unit.

[0064] In some implementations, the received activated value is already squared before being provided to the normalization register columns, as described above with reference to FIG. 6.

[0065] The normalization unit can also receive squared activated values 810 from the neighboring normalization units at summation circuitry 806. The summation circuitry 806 can generate a sum of the squared activated value 808 and the received squared activated values 810.

[0066] The sum can be sent to a memory unit 812. In some implementations, the memory unit 812 includes a look up table and interpolation unit. The normalization unit can use a portion of the sum, e.g., a set of high bits of the sum, as an address to look up one or more coefficients provided by a system parameter. The memory and interpolation unit 812 can generate a normalization scaling factor based on the coefficients and the sum of squared activated values. The normalization scaling factor can be sent to multiplication unit 816.

[0067] In some implementations, the sum of squares is a 12 bit value. The normalization unit can use the top 4 bits of the sum of squares as an index to the lookup table. The top 4 bits can be used to access coefficients, e.g. which are specified by a user, from the lookup table. In some implementations, the top 4 bits access 2 12-bit coefficients: A & B. The bottom eight bits can be a delta used in an equation to calculate the normalization scaling factor. An example equation is given by Scaling factor = *minimum* (1048575, [A * delta + B * 256 + 2^7]) >> 8, where *minimum* processes two arguments and returns the argument with the minimum value.

[0068] The normalization unit can multiply, using the multiplication unit 816, the normalization scaling factor with the activated value 802 to generate a normalized value. In some implementations, the normalized value is then sent to pooling circuitry, e.g., the pooling circuitry 508 of FIG. 5.

[0069] FIG. 9 shows an example architecture 900 for pooling circuitry. The pooling circuitry can apply an aggregation function to one or more normalized or activated values to generate pooled values. By way of illustration, the architecture 900 can perform a pooling of a 4 x 4 set of activated or normalized values. Although the pooling shown in FIG. 9 has a square region, i.e., 4x4, rectangular regions are possible. For example, if the region has a window of n x m, the architecture 900 can have n * m registers, i.e., n columns and m rows.

[0070] The pooling circuitry can receive a sequence of elements from the vector of normalized values, e.g., from normalization circuitry 506 of FIG. 5. For example, the sequence can represent pixels of an 8 x 8 portion of an image, and the pooling circuitry architecture 900 can pool values from a 4 x 4 subset of the 8 x 8 portion. In some implementations, normalized values are appended to the sequence once computed by normalization circuitry coupled to the pooling circuitry. In some implementations, the neural network processor includes multiple parallel pooling circuitries. Over each clock cycle, each pooling circuitry can receive a respective element from the vector of normalized values from normalization circuitry. Each pooling circuitry can interpret elements received from the normalization circuitry as a two-dimensional image arriving in raster order.

[0071] The pooling circuitry can include a series of registers and memory units. Each register can send an output to aggregation circuitry 906 that applies an aggregation function across the values stored inside the registers. The aggregation function can return a minimum, maximum, or average value from a set of values.

[0072] A first normalized value can be sent to and stored inside register 902. On a subsequent clock cycle, the first normalized value can shift to a subsequent register 908 and be stored in memory 904, and a second normalized value can be sent to and stored inside register 902.

[0073] After four clock cycles, four normalized values are stored inside the first four registers 902, 908-912. In some implementations, the memory unit 904 operates under first-in-first-out (FIFO). Each memory unit can store up to eight normalized values. After the memory unit 904 contains a complete row of pixels, the memory unit 904 can send a normalized value to register 914.

[0074] At any given point in time, the aggregation circuitry 906 can access normalized values from each register. The normalized values in the registers should represent normalized values for a 4 x 4 portion of the image.

[0075] The pooling circuitry can generate a pooled value from the accessed normalized values by using the aggregation circuitry 906, e.g., a maximum, a minimum, or an average normalized value. The pooled value can be sent to a unified buffer, e.g., the unified buffer 208 of Fig. 2.

[0076] After generating the first pooled value, the pooling circuitry can continue to generate pooled values by shifting the normalized values through each register so that new normalized values are stored in the registers and can be pooled by the aggregation circuitry 906. For

example, in architecture 900, the pooling circuitry can shift the normalized values over 4 more clock cycles, thereby shifting the normalized values in the memory units into the registers. In some implementations, the pooling circuitry shifts the new normalized values until a new normalized value is stored in a last topmost register, e.g., register 916.

[0077] The aggregation circuitry 906 can then pool the new normalized values stored in the registers.

[0078] In some implementations, instead of receiving a vector of normalized values, the pooling circuitry receives a vector of activated values, as described above in reference to FIG. 5.

[0079] Embodiments of the subject matter and the functional operations described in this specification can be implemented in digital electronic circuitry, in tangibly-embodied computer software or firmware, in computer hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Embodiments of the subject matter described in this specification can be implemented as one or more computer programs, i.e., one or more modules of computer program instructions encoded on a tangible non transitory program carrier for execution by, or to control the operation of, data processing apparatus. Alternatively or in addition, the program instructions can be encoded on an artificially generated propagated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus for execution by a data processing apparatus. The computer storage medium can be a machine-readable storage device, a machine-readable storage substrate, a random or serial access memory device, or a combination of one or more of them.

[0080] The term "data processing apparatus" encompasses all kinds of apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit). The apparatus can also include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them.

[0081] A computer program (which may also be referred to or described as a program, software, a software application, a module, a software module, a script, or code) can be written in any form of programming language, including compiled or interpreted languages, or declarative or procedural languages, and it can be deployed in any form, including as a standalone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data, e.g., one or more scripts stored in a markup language document, in a single file dedicated to the

program in question, or in multiple coordinated files, e.g., files that store one or more modules, sub programs, or portions of code. A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[0082] The processes and logic flows described in this specification can be performed by one or more programmable computers executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[0083] Computers suitable for the execution of a computer program include, by way of example, can be based on general or special purpose microprocessors or both, or any other kind of central processing unit. Generally, a central processing unit will receive instructions and data from a read only memory or a random access memory or both. The essential elements of a computer are a central processing unit for performing or executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Moreover, a computer can be embedded in another device, e.g., a mobile telephone, a personal digital assistant (PDA), a mobile audio or video player, a game console, a Global Positioning System (GPS) receiver, or a portable storage device, e.g., a universal serial bus (USB) flash drive, to name just a few.

[0084] Computer readable media suitable for storing computer program instructions and data include all forms of nonvolatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[0085] To send for interaction with a user, embodiments of the subject matter described in this specification can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can send input to the computer. Other kinds of devices can be used to send for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input. In addition, a computer can interact with a user by sending documents to and receiving documents from a device that is used by the user; for example, by sending web pages to a web browser on a user's client device in response to requests received from the web browser.

[0086] Embodiments of the subject matter described in this specification can be implemented in a computing system that includes a back end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the subject matter described in this specification, or any combination of one or more such back end, middleware, or front end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

[0087] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

[0088] While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any invention or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular inventions. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination.

[0089] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system modules and components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

[0090] Particular embodiments of the subject matter have been described. Other embodiments are within the scope of the following claims. For example, the actions recited in the claims can be performed in a different order and still achieve desirable results. As one example, the processes depicted in the accompanying figures do not necessarily require the particular order shown, or sequential order, to achieve desirable results. In certain implementations, multitasking and parallel processing may be advantageous.

REFERENCES CITED IN THE DESCRIPTION

Cited references

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- EP0422348A [0003]
- US2014180989A1 [0004]

Patentkrav

1. Kredsløb (502) til udførelse af neurale netværks beregninger for et neuralt netværk, der omfatter en flerhed af lag, hvilket kredsløb omfatter:

et aktiveringskredsløb (504), der er konfigureret til at modtage en vektor af akkumulerede værdier, og som er konfigureret til at anvende en funktion på hver akkumuleret værdi for at generere en vektor af aktiveringsværdier, og

et normaliseringskredsløb (506), der er koblet til aktiveringskredsløbet, og som er konfigureret til at modtage aktiveringsværdier af vektoren af aktiveringsværdier og generere en respektiv normaliseret værdi for hver aktiveringsværdi, idet normaliseringskredsløbet omfatter en flerhed af normaliseringsregisterkolonner (604, 606, 608), idet hver normaliseringsregisterkolonne omfatter en flerhed af normaliseringsregistre (616, 632, 640), og hvor mindst ét normaliseringsregister i hver normaliseringsregisterkolonne er konfigureret til at modtage en respektiv aktiveringsværdi for at generere i det mindste den respektive normaliserede værdi for aktiveringsværdien,

hvor flerheden af normaliseringsregistre (616, 632, 640) i hver normaliseringsregisterkolonne (604, 606, 608) er serieforbundet, hver normaliseringsregisterkolonne er konfigureret til at modtage den respektive aktiveringsværdi, hvor normaliseringskredsløbet er konfigureret til at danne grupper (624, 628) omkring ét eller flere normaliseringsregistre, idet hver gruppe svarer til en normaliseringsenhed, og hver normaliseringsenhed er konfigureret til at beregne en respektiv normaliseret værdi for den respektive aktiveringsværdi,

hvor normaliseringskredsløbet (506) er beregnet til at forskyde den aktiverede værdi til et efterfølgende normaliseringsregister (632, 634, 640) i

normaliseringsregisterkolonnen (604, 606), indtil alle de aktiverede værdier af den respektive gruppe, der er nødvendige for at beregne den respektive normaliserede værdi for aktiveringsværdien, er indlæst.

2. Kredsløb ifølge krav 1, hvor aktiveringskredsløbet (504) modtager vektoren af akkumulerede værdier fra et systolisk array i kredsløbet.

3. Kredsløb ifølge krav 1 eller 2, hvor hvert normaliseringsregister (616, 632, 640) er konfigureret til at videregive den forskellige aktiveringsværdi til en tilstødende normaliseringskolonne.

4. Kredsløb ifølge krav 1 eller 3, hvor hver gruppe (624, 628) dannes ved anvendelse af en normaliseringsradiusparameter.

5. Kredsløb ifølge et af kravene 1 til 4, hvor hver normaliseringsenhed er konfigureret til at:

modtage den respektive aktiveringsværdi

generere en respektiv normaliseret mellemværdi ud fra den respektive aktiveringsværdi, og

sende den respektive normaliserede mellemværdi til én eller flere nabonormaliseringsenheder.

6. Kredsløb ifølge krav 1, hvor genereringen af den respektive normaliserede mellemværdi omfatter generering af et kvadrat af den respektive aktiveringsværdi.

7. Kredsløb ifølge krav 5 eller 6, hvor hver normaliseringsenhed ydermere er konfigureret til:

fra en eller flere nabonormaliseringsenheder at modtage én eller flere normaliserede mellemværdier, der genereres ud fra aktiveringsværdier

at summere hver normaliseret mellemværdi for at generere et indeks

at bruge indekset til at få adgang til én eller flere

værdier fra en opslagstabel
at generere en skaleringsfaktor ud fra den ene eller de flere værdier og indekset, og
at generere den respektive normaliserede værdi ud fra skaleringsfaktoren og den respektive aktiveringsværdi.

8. Kredsløb ifølge et af kravene 1 til 7, der ydermere omfatter et poolingskredsløb (508), som er konfigureret til at modtage de normaliserede værdier og pulje de normaliserede værdier for at generere en puljet værdi.

9. Kredsløb ifølge krav 8, hvor poolingskredsløbet (508) er konfigureret til at gemme flerheden af normaliserede værdier i en flerhed af registre og en flerhed af hukommelsesenheder,
hvor flerheden af registre og flerheden af hukommelsesenheder er serieforbundet, hvor hvert register gemmer en normaliseret værdi, og hver hukommelsesenhed gemmer en flerhed af normaliserede værdier,
hvor poolingskredsløbet er konfigureret til efter hver clockcyklus at forskyde en given normaliseret værdi til et efterfølgende register eller en hukommelsesenhed, og
hvor poolingskredsløbet er konfigureret til at generere den puljede værdi ud fra de normaliserede værdier.

10. Kredsløb ifølge et af kravene 1 til 7, der ydermere omfatter et poolingskredsløb (508), som er konfigureret til at modtage aktiveringsværdierne og pulje aktiveringsværdierne for at generere en puljet værdi.

11. Kredsløb ifølge krav 10, hvor poolingskredsløbet (508) er konfigureret til at gemme flerheden af aktiveringsværdier i en flerhed af registre (616, 632, 640) og en flerhed af hukommelsesenheder,
hvor flerheden af registre og flerheden af hukommelsesenheder er serieforbundet, hvor hvert register gemmer en normaliseret værdi, og hver hukommelsesenhed gemmer en flerhed af aktiveringsværdier,
hvor poolingskredsløbet er konfigureret til efter hver

clockcyklus at forskyde en given aktiveringsværdi til et efterfølgende register eller en hukommelsesenhed, og hvor poolingskredsløbet er konfigureret til at generere den puljede værdi ud fra aktiveringsværdierne.

DRAWINGS

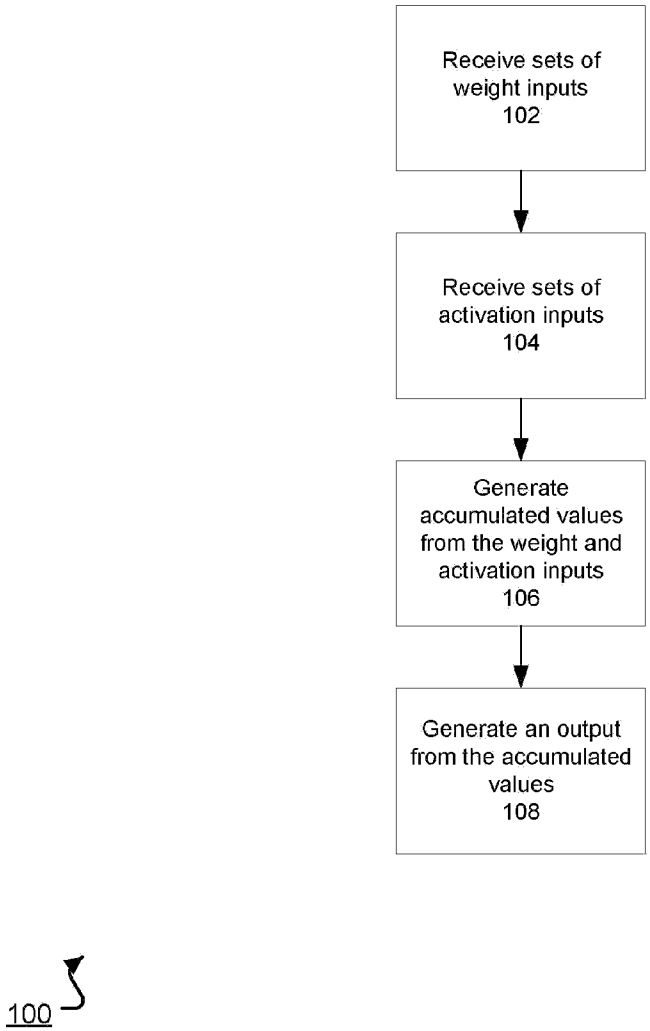

FIG. 1

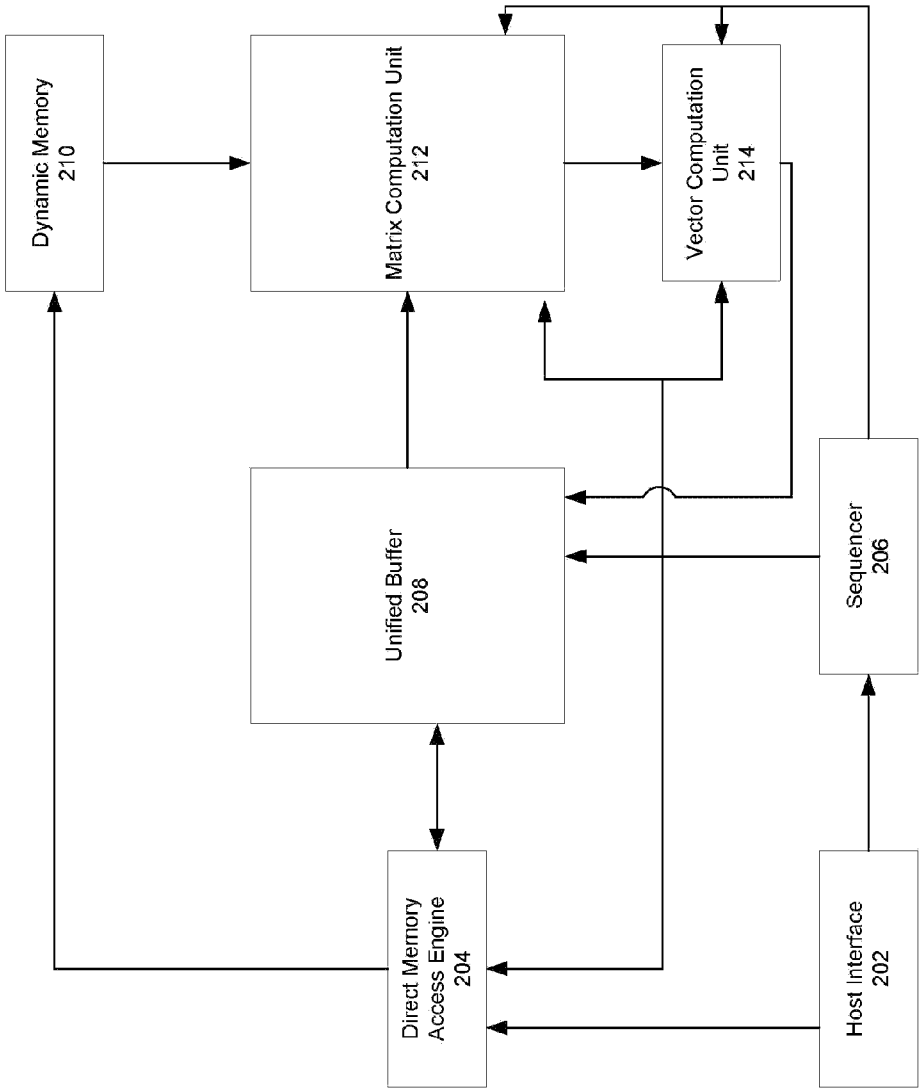
Dynamic Memory
210
Matrix Computation Unit
212
Vector Computation
Unit
214
Unified Buffer
208
Sequencer
206
Direct Memory
Access Engine
204
Host Interface
202
200

FIG. 2

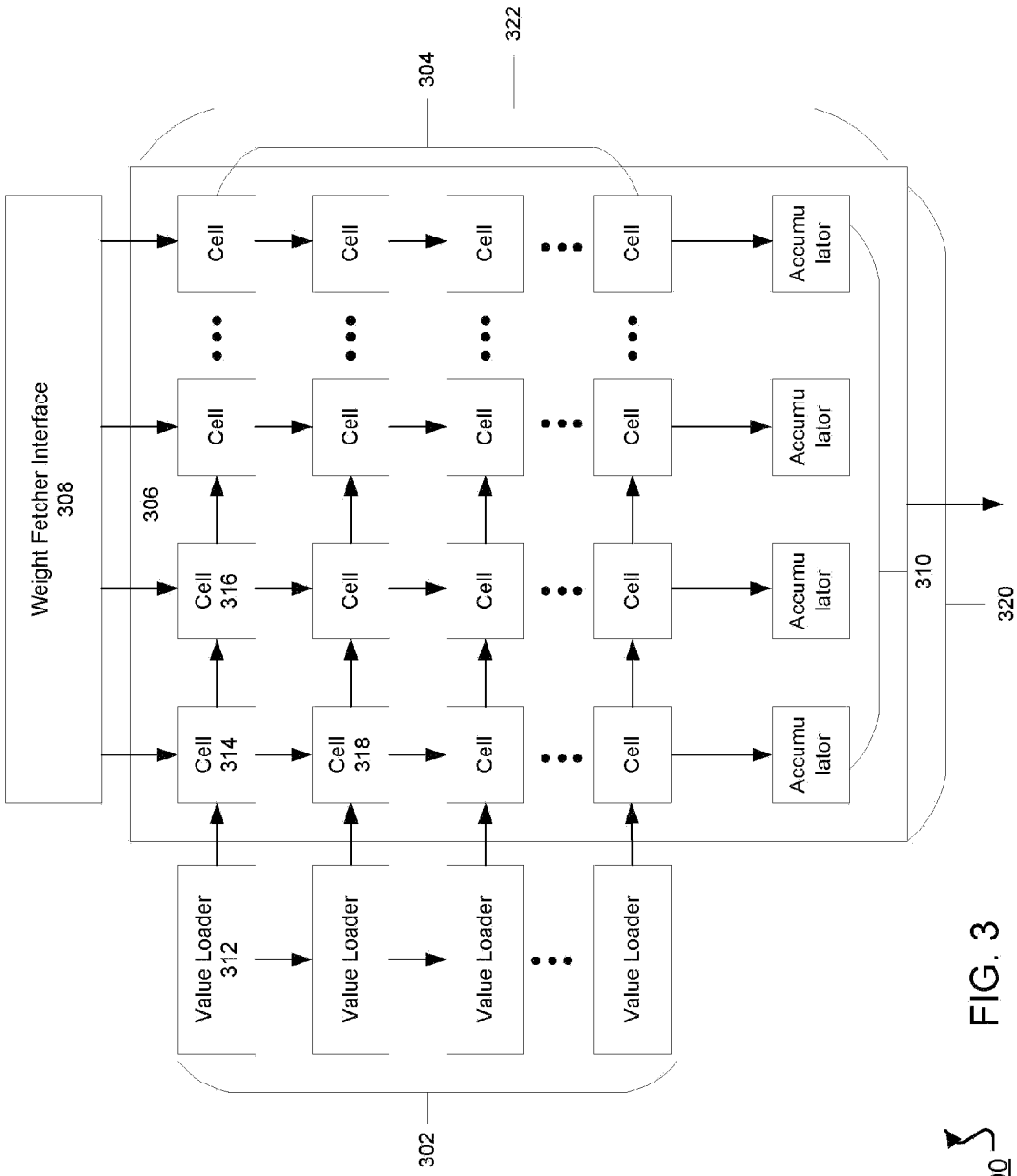
Weight Fetcher Interface
308
306
Value Loader
312
Cell
314
Cell
316
Cell
318
Cell
Accumu
lator
310
320
302
304
322
300

FIG. 3

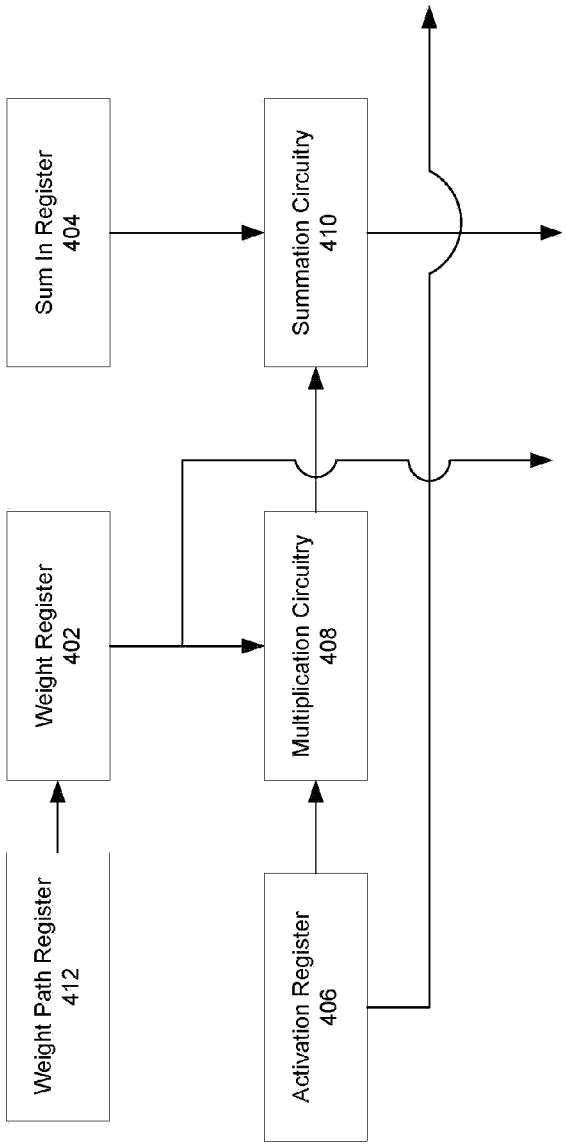
Sum In Register
404
Summation Circuitry
410
Weight Register
402
Multiplication Circuitry
408
Weight Path Register
412
Activation Register
406
400

FIG. 4

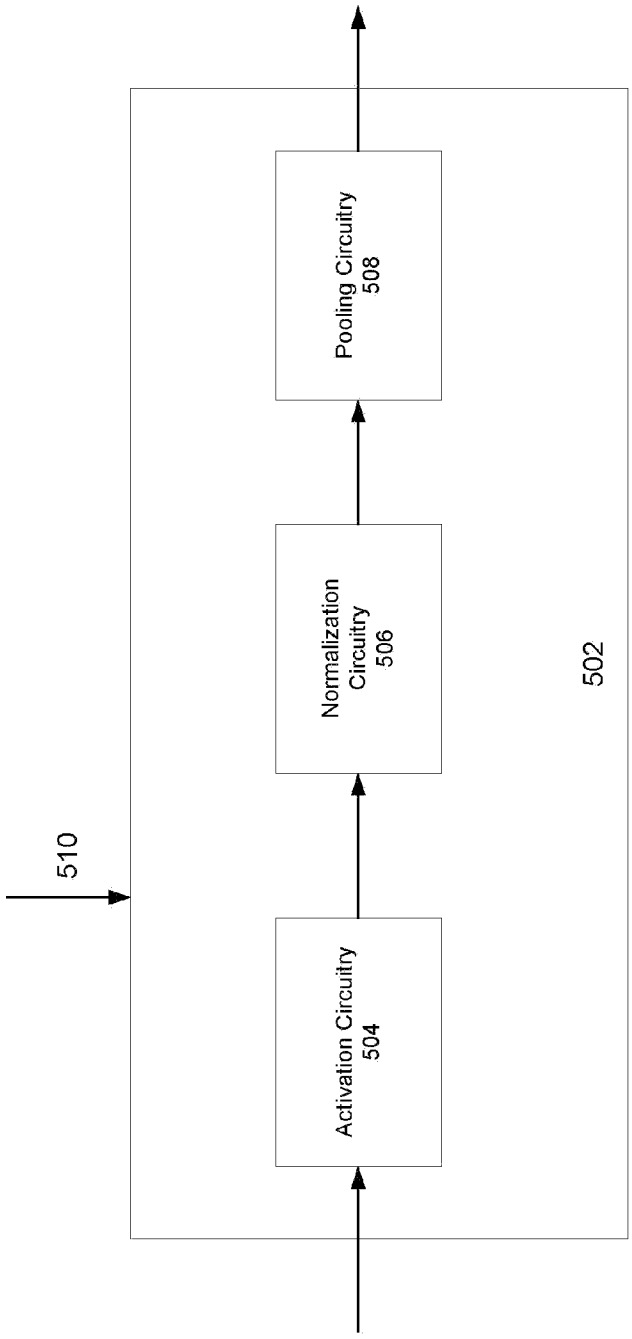
510
Activation Circuitry
504
Normalization
Circuitry
506
Pooling Circuitry
508
502
500

FIG. 5

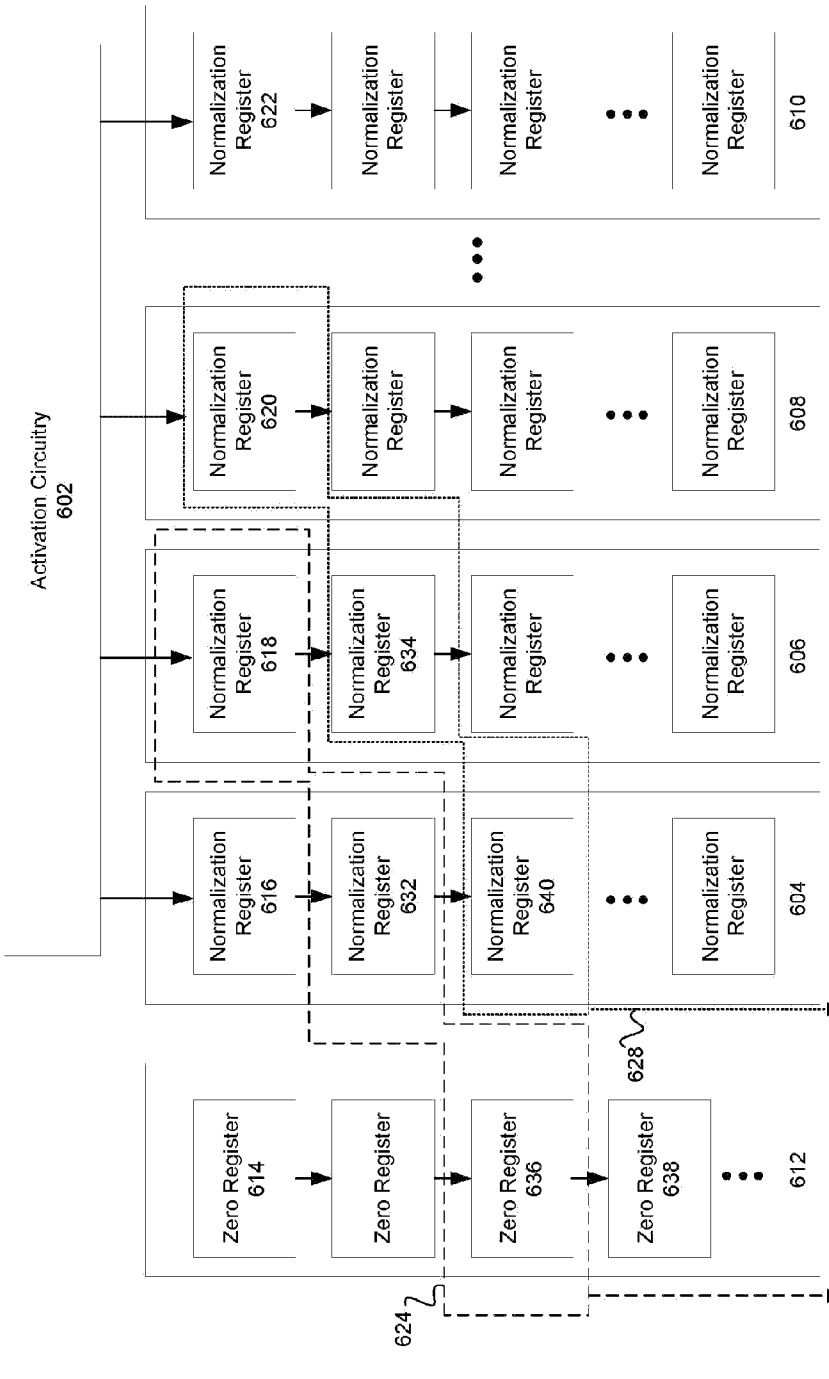
Activation Circuitry
602
Normalization Register 622
Normalization Register
Normalization Register
Normalization Register 610
Normalization Register 620
Normalization Register
Normalization Register
Normalization Register 608
Normalization Register 618
Normalization Register 634
Normalization Register
Normalization Register 606
Normalization Register 616
Normalization Register 632
Normalization Register 640
Normalization Register 604
Zero Register 614
Zero Register
Zero Register 636
Zero Register 638
612
624
628
Normalization Unit 630
Normalization Unit 626
600

FIG. 6

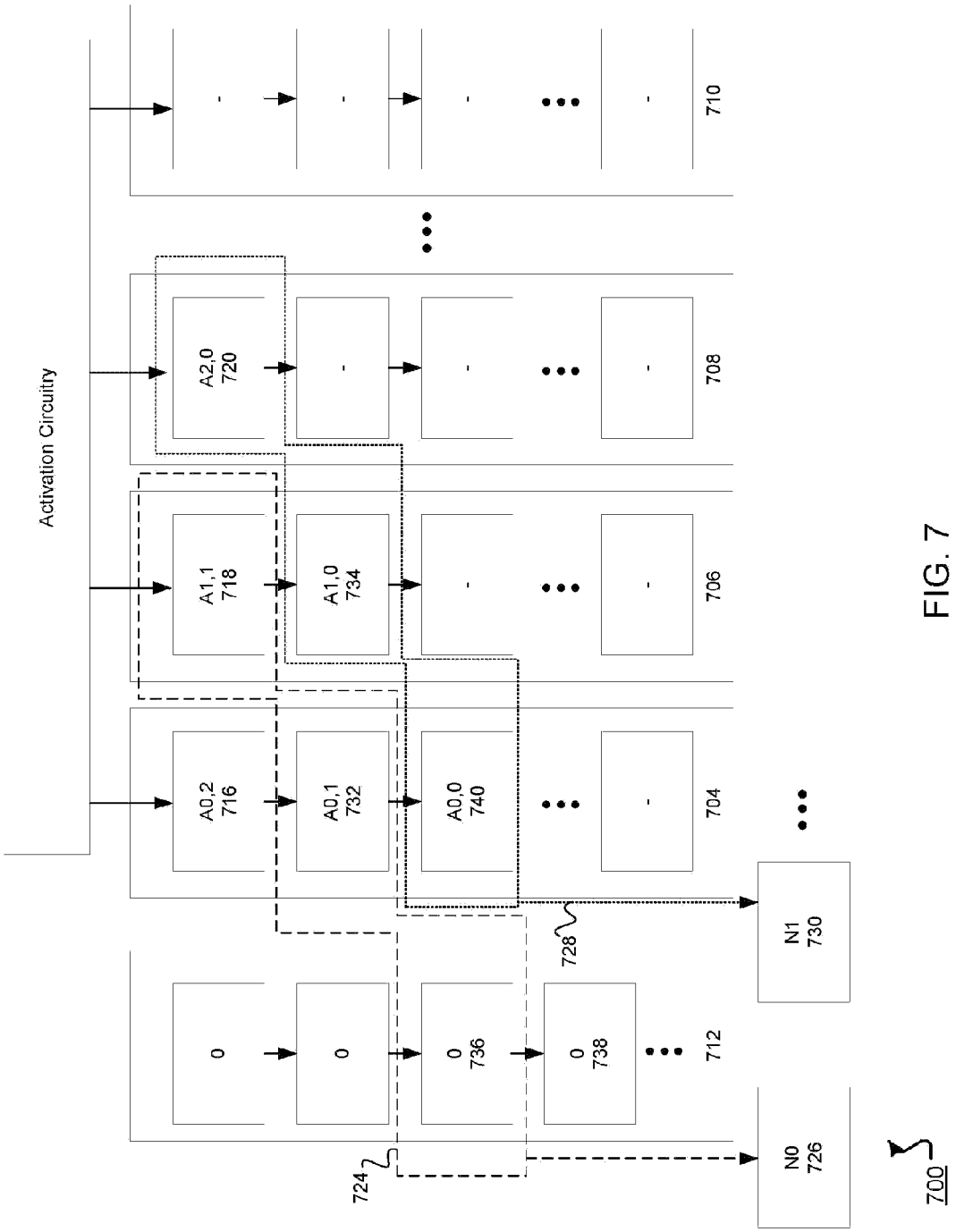
Activation Circuitry
A0,2 716
A0,1 732
A0,0 740
704
A1,1 718
A1,0 734
706
A2,0 720
708
710
0
0
0 736
0 738
712
724
728
N1 730
N0 726
700

FIG. 7

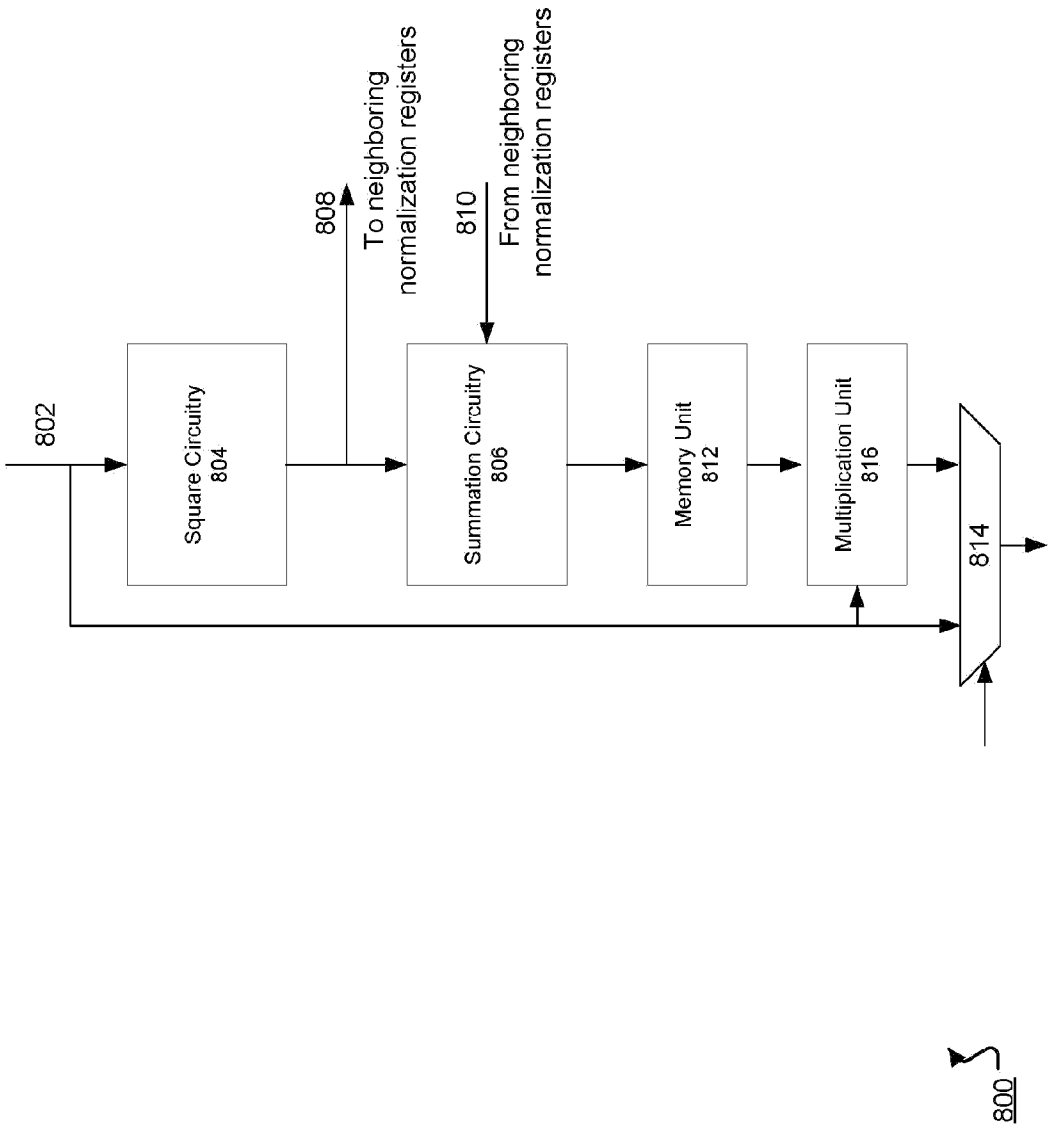
802
Square Circuitry
804
808
To neighboring
normalization registers
810
From neighboring
normalization registers
Summation Circuitry
806
Memory Unit
812
Multiplication Unit
816
814
800

FIG. 8

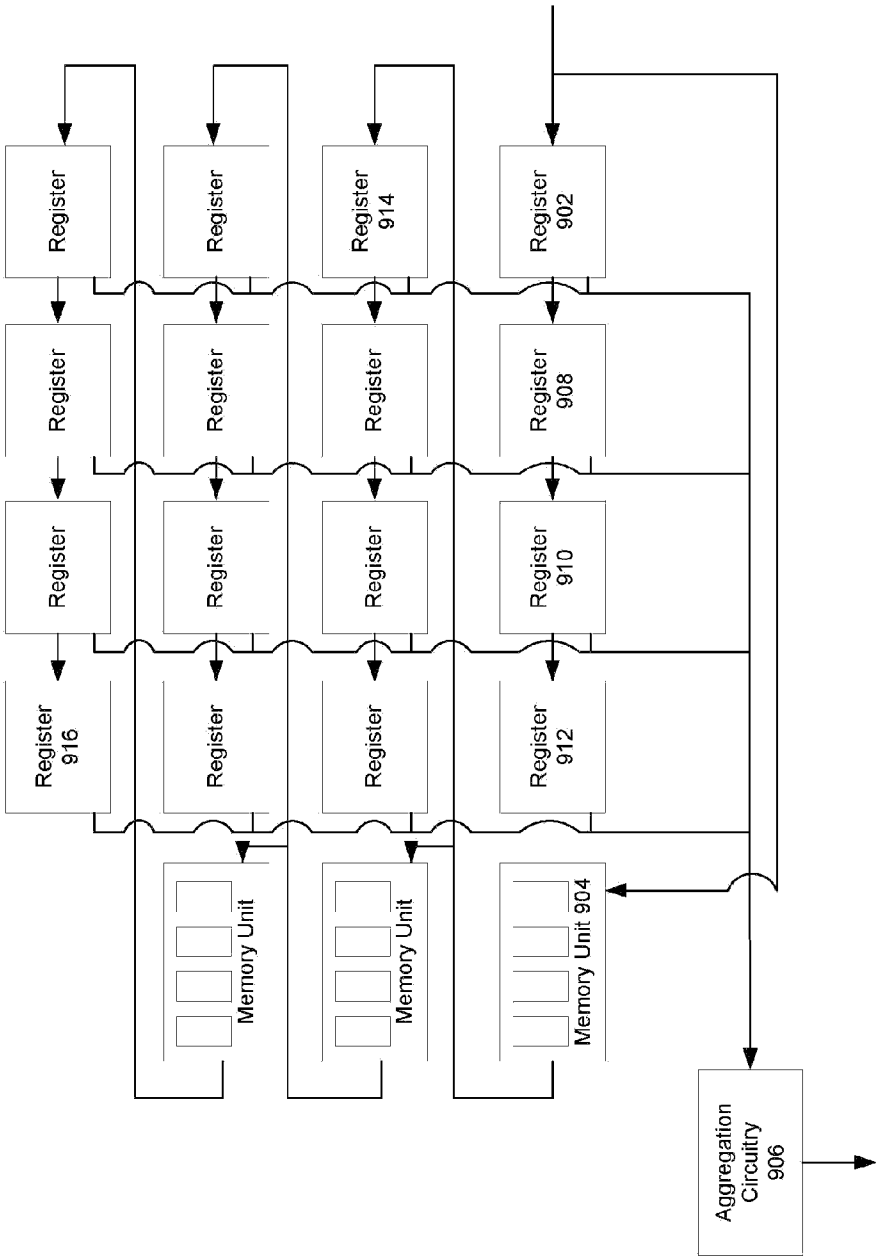
Register
Register
Register 914
Register 902
Register
Register
Register
Register 908
Register
Register
Register
Register 910
Register 916
Register
Register
Register 912
Memory Unit
Memory Unit
Memory Unit 904
Aggregation Circuitry 906
900

FIG. 9