Journal notice

/*Special Issue on Neural Network Learning in Big Data*/

Title

Kanban Cell Neuron Network

Pre publication copyright notice

Correspoinding author

Colin James III, Director, Ersatz Systems Machine Cognition, LLC
3925 Elisa Ct, Colorado Springs CO 80904-1056
info@cec-services.com +011 (719) 210.9534

Abstract

To map and process "big data" requires new types of development for machine cognition and signal processing of combinatorial logic in a network. A novel system for machine cognition includes Kanban cells (KC), Kanban cell neurons (KCN), and Kanban cell neuron networks (KCNN), patent pending. The KC is an asynchronous AND-OR gate without feedback that is self-timing by the input data to process until input is equal to output. For the KCN, the input example is a four-valued logic (4VL) based on the 2-tuple as four logical values in the set {"", 01, 10, 11} of four-valued bit code (4vbc) where "" is equivalent to 00. Access to a sparsely filled look up table (LUT) is minimized in hardware with a 2-bit value per logical signal. The unique algorithm of multiple KCNs in the KCNN emulates the human neuron with nine logical inputs and one output. The KCNN model in parallel is scalable for large data sets and is adaptable as a forward-looking rules-engine as based on bivalent trial and error. The real-time algorithm is implemented by a look up table (LUT). In software the LUT occupies 64 KB, and on a desktop processes 1MM KCNs per second. In hardware the LUT complex occupies 194 KB, and on a $40 device processes at 1.8 BB KCNs per second, or about 1600 times faster. The immediate applications KCNNs are analytics for high velocity streaming data such as in time series for meteorologics and econometrics, for which examples with results are presented.

Keywords

analytics, AND-OR gate, high velocity stream, Kanban cell neuron, linear network, multi-valued logic

Historical background

Originally the Kanban cell (KC) was the production part of a linear pull system used to minimize the size, change, and turnover of parts inventory for the Just in Time (JIT) manufacturing of automobiles. The KC was used by Toyota in about 1964.

Fig. 1 shows the KC in the Petri net, a bipartite directed graph, and with or without a failure or idle subnet as an abstraction of the generalized stochastic Petri net (GSPN) of flexible manufacturing systems (FMS), which are push production systems (for example, using pallets to load incomplete parts and to unload completed parts by continuous transportation as by conveyer or automatic guided vehicle (AGV)).

The system in Fig. 1 is a Petri net of a KC [James 1999]. Step 104 is a transition. Step 101 is the input and output place. Steps 105 and 106 are feedback paths of the feedback loops of the paths 103 to 104 and 102 to 104. In this context, feedback paths serve as decision branches in the logic of the KC, and are commonly referred to in their totality as feedback loops. (Steps marked as m1, m2, m6, and t2 are true to the original labels and equivalent to the respective Steps 101, 102, 103, and 104.)

Fig. 2 shows similarity between the structure of the KC and the design of a universal accounting arithmetic system in N-dimensions [James 1998]. Steps 201, 202, and 203 are descriptive of the respective symbols of Steps 101, 102, and 103 in Fig. 1. Steps 204, 205, and 206 are descriptive of the respective symbol of Step 104 in Fig. 1. Step 207 is keyed to the respective Step 107 in Fig. 1. Steps in Fig. 2 describe how a KC system performs the identical functionality of a disparate system not related solely to the field of manufacturing. This indicates that the KC is a universal mechanism for abstract processing of any kind.

Fig. 2 also shows an accounting arithmetic system [James 1998]. Step 201 inputs a transaction type and the amount on which to operate. Step 202 uses a look up table (LUT) to translate the transaction type into an output index code. Step 203 uses another LUT to obtain the series of sequential logic switches by which to operate on the amount from Step 201. The input index of Step 203 is the output index of Step 202. Step 204 is the output of the account indexes and respective operators. The input index of Step 204 is the output from Step 203. Step 205 operates the series of sequential accounts by which the respective operators process the amount from Step 201. The accounts contain only a single balance value which is updated and overwritten. Step 206 is the update step by which an unbounded transaction log records the transaction type and amount from Step 201 and also a unique time stamp that verifies when Steps 201 through Step 205 are completed. Step 207 is a feedback loop from Steps 206 to Step 201 to restart the process for subsequent inputs.

Fig. 3 shows a network that is synchronous, self-timing neural network as a series of feedback loops. The network consists of data places, through which data flows as data places 301, 311, and 321, and of timing places as timing places 302, 312, and 322, which stimulate the data places as 301, 311, and 321. The direction the data flows is bidirectional as in paths 305 and 306. The direction of timing paths is bidirectional as in paths 303 and 304. The timing places 302, 312, and 322 effectively open and close the data places 301, 311, and 321 to control when waiting data is allowed to flow. The timing places 302, 312, and 322 may be either physical clock cycles or logical looping structures, the duration for which constitutes a delay in that network.

In biology, a neuron is a cell nucleus and body with multiple dendrites as input paths, and a single axon as output path. The entry pathway of the dendrite to the neuron cell is a synapse and receptor where in the neurotransmitter fluid such as serotonin, ion transfers occur with calcium (Ca+), potassium (K+), and sodium (Na+).

Neural components can be represented as vector spaces such as an adaptive linear neuron or adaptive linear element (Adaline) composed of weight and bias vectors with a summation function (OR circuit), and also a multi-layered Adaline (Madaline) where two such units emulate a logical XOR function (exclusive OR circuit). Such components are examples of probabilistic methodology applied as an apparatus to map and mimic the biological neuron.

A perceptron can be represented as a binary classifier using a dot product to convert a real-valued vector to a single binary value which serves as a probabilistic methodology and apparatus to map and mimic the biological neuron.

A spike neuron or spiking neuron can be represented as a conductive network based on temporal or time bias and differential equations or calculus which serves as a probabilistic methodology and apparatus to map and mimic the biological neuron.

The first deficiency with these neural networks is that as they are based on a vector space, so a solution is ultimately not bivalent, is probabilistic, and hence is undecidable. (Bivalency is not a vector space [James 2010].)

The second deficiency with these networks is that an exclusive OR (XOR) function is sometimes mistakenly developed in them to mimic a neuron. The logical XOR connective is orthogonal or effectively perpendicular as a mathematical operator. However, biological bodies are not rectilinear, but rather based on a phi or Phi ratio of (1 ± (5 ^ 0.5)) / 2. This means that there are no right angles (90-degree arcs) in biology per se. While the logical XOR connective may be constructed from the NOR or NAND logical connectives, there is no evidence that the XOR function is built into the neuron, or necessarily should be.

The third deficiency with these networks is that the perceptron and spike neuron can accept any input without discrimination.

Descriptive summary

The Kanban cell (KC), Kanban cell neuron (KCN), and Kanban cell neuron network (KCNN) are improvements on the above background components.

For example, the KC processes three input values and one output value. The values are only in a multi-valued logic (MVL) such as the 2-tuple set of { "00", "01", "10", "11"}. This is a four-valued logic (4VL) of four-valued bit code (4vbc). It also includes a four-valued logic with null (4VLN) and four-valued bit code with null (4vbcn).

The KC is a logic gate using solely the OR and AND logical connectives, that is, with only the arithmetical operations of addition and multiplication. The KC performs self-timing and terminates processing when the first of the three input values equals the output value.

The KCN maps three such KCs into one output, that is, nine inputs as dendrites into one output as axon. The KCNN consists of KCNs in sequential and/or parallel operation.

The purpose of the KCNN is: to process input and output more quickly than the ion transfers; not to require synchronous, repetitive feedback or timing paths; and to use the attributes of the input and output data to self-time itself internally. The KCNN goal is faster asynchronous machine cognition of itself. A benefit of KCNN is also to map neurons into a non-vector space that is not probabilistic and hence to ensure bivalency and decidability.

Descriptive details

The system described below is a massive forest or collection of trees with branches or bundles of leaves or nodes. A node is the Kanban cell (KC). A bundle is a Kanban cell neuron (KCN). A tree is cascaded KCNs, each with a Kanban cell level (KCL). A forest is a Kanban cell neuron network (KCNN, KCN2, or KCN^2, and pronounced KCN-Two or KCN-Squared). The KC is effectively a sieve that processes signals in a highly selective way. The KC is the atomic and logical mechanism of the system. Three KCs make up a KCN. Hence the KCN is a radix-3 tree. Multiple KCs as nodes at the same level form a KCL to better describe the KCN to which the KCs belong. The KCNN contains nine KCNs to produce one KCN result. Hence the KCNN is also a radix-9 tree.

The KCN maps the biological mechanism of the human neuron as a series of 9-dendrites that process input signals concurrently into 1-axon, the path of the output signal. The input

signals are equivalent to the logical values of null, contradiction, true, false, and tautology. These are respectively assigned as a 2-tuple of the set {"", "00", "01", "11"} or as the set of {"", 0, 1, 2, 3}, depending on which set of valid results is required.

As a sieve, the KCN filters three input signals into one output signal. The number of all input signals to produce one valid output signals as a result is in the ratio of about 15 to 1.

The computational or machine mechanism to accomplish this is basically the same for software or hardware implementation. The core concept is that LUTs produce output results at a faster rate than by the brute force of computational arithmetic.

The theory of input signals is presented here. The Kanban cell (KC) is defined as

$$(ii_1 \wedge pp_1) \vee qq_1 = kk_1,$$

where $\wedge$ is AND and $\vee$ is OR. Equation 1

The Kanban cell neuron (KCN) is four of these connected KCs and is defined as

$$(((ii_1 \wedge pp_1) \vee qq_1 = kk_1) \wedge ((ii_2 \wedge pp_2) \vee qq_2 = kk_2))$$
$$\vee ((ii_3 \wedge pp_3) \vee qq_3) = kk_3) = kk_4. \quad \text{Equation 2}$$

The utility of the KCN is that it matches the human neuron with 9-inputs as dendrites and 1-output as axon. The 9-inputs are: ii_1, pp_1, qq_1; ii_2, pp_2, qq_2; ii_3, pp_3, qq_3. The 1-output is: kk_4. This algorithm is the program for the KC.

For example, pseudo code to process input values of ii, pp, qq into kk is presented here.

```
LET kk_lut = lut( ii, pp, qq)          ! 3-D LUT indexed by ii, pp, qq for kk
LET kk_output = ii                     ! Preset output kk to ii if test fails below
IF kk_lut ≠ ii THEN LET kk_output = kk_lut      ! iff LUT result <> ii
```

The input to the KC is in the form of 3 x 2-tuples or three dibit values as effectively a 2-tuple set of {ii, pp, qq}. To produce a single dibit value as kk output, the three input values are required. Hence the expression "3-inputs to 1-output" accurately describes the KC. When KCs are chained, three inputs are required to produce each of the three outputs accepted into the next consecutive KC, for a total of 9-input signals. This defines the KCN as the expression of "9-inputs to 1-output."

The number of inputs required in KC_1 to produce KC_n is given by the formula of

$$KC_n = 3^n,$$

where $n > 0$. Equation 3

It follows that KCs in parallel and chained in succession represent a permutation of exponential complexities.

Each successively complex level of KCs hence has its number of KCs as a power of 9 (= 3^2), such as (3^2) ^0 = 1, (3^2)^1 = 9, (3^2)^2 = 81, … , (3^2)^n.

The number of groups of signals of {ii, pp, qq} required for levels of KCs as KCLs may be tabulated. The number of groups where three groups are processed concurrently for KCL is the result of reducing the cumulative signals by a factor of three of the cumulative number of the groups of signals. For KCL-12 or 3 ^ 13, the number of discrete signals is 1,594,323, and the cumulative number of KCNs is 531,444. For sizing in hardware implementation, each such result occupies 2-bits for a respective KCL-12 storage requirement of 1,062,882 bits.

A commonly published statistic is that on average there are seven to nine dendrites per neuron. This means two to three complete groups of signals (six to nine discrete signals) could be processed concurrently at receptor points for the dendrites along a neuron.

In the formula (ii AND pp) OR qq = kk of a KC in Equation (1), there are 64-combinations of 2-tuples (or dibits) of the set of {00, 01, 10, 11}. When ii and kk are the same value (ii = kk), this represents the condition where input is processed to the same output result. This also means there was no change to the input after processing. Consequently, this event marks the termination of processing for that particular signal, and hence the result is final. Such an event or condition produces no new logical result.

When ii, pp, or qq is "00", a contradiction is present to mean something is both true and false. This has the same effect as a null value because no logical information is disclosed other than that the there is a void result. Consequently the values "00" and "" are folded together into "". Hence the four-valued logical set of values as the 2-tuple of the set {"00", "01", "10", "11"} becomes {"", "01", "10", "11"}. This is named four-valued logic with null (4VLN) composed of four-valued bit code with null (4vbcn).

All possible combinations of the values within the 2-tuple produce 64-values. A LUT is presented for these values and an index in the inclusive interval range of [0, 63]. It is named LUT-64. It appears in Table 1 and in the source code below, in which the DATA statements set forth the table by rank order.

This represents a sparsely filled LUT of three inputs {ii, pp, qq} to produce one output {kk} for a KC. When three KCs are combined to make a KCN, there are nine inputs to produce one output. This is named LUT-9 and is built by combining three LUTs of 64 entries each into 64 ^ 3 entries or 262,144 entries. The sparsely filled LUT-9S is indexed as (0 ... 63, 0 ... 63, 0 ... 63).

Statistics for the percentage of signals processed from KCN-18 are disclosed. The signals accepted and rejected for the KCL-18 cascade are for input of 129,140,163 discrete random signals to a single result as a dibit (2-tuple) where 8,494.377 signals are accepted at about 7%, for a ratio of accepted signals to rejected signals of 1 to 13. This also indicates how the KCN overcomes the deficiencies of accepting all signals as in the Background section above.

Of interest is the relative distribution of the four-valued bit code (4vbc) for contradiction (00), true (01), false (10), and tautology (11). The logical results of the KCN favor the tautology (11) by about 34560 / 50432 = 69% over the other frequency of the other combined logical values. In the same way, true (01) and false (10) represent about (7936 + 7936) / 50432 = 31%. These statistics imply that the KCN filters about: 2-valid assertions to 9-invalid assertions; equal numbers of true- and false-assertions; and 1-true or 1-false to 2-tautologies. By extension, this signifies that the KCN places an onus on rejecting invalid assertions and on finding tautologies. This also indicates how the KCN overcomes the problem of accepting all signals as in the historical neurons described in the Background section above.

A software implementation is presented here. The values in a LUT of 64-entries may be represented as the same respective values but in three different formats such as a natural number, the character string of that natural number, or character digits representing exponential powers of arbitrary radix bases. As numeric symbols, the four valid results are in the set of {0, 1, 2, 3}, and the invalid results are specified as a null set {-1}. As character string symbols, the four valid results are in the set of {"0", "1", "2", "3"}, and the invalid results are specified as a null set {""}. As character string exponents, the four valid results are in the set of {"00", "01", "10", "11"} or {00, 01, 10, 11}.

The representation of the data elements within a LUT is important because the type of format affects the size of the LUT and the speed at which the data is manipulated. For example, to test a number for invalid result requires determining if it is a negative number, that is, is less than zero. The pseudo code for this is: IF n < 0 THEN; or in the negative as IF NOT(n = 0) THEN. (In logic it is often a more direct strategy to test for the negation of a proposition or an assertion.) To test a character string for an invalid result requires determining if it is a null character {""}, that is, not within the set {"1", "2", "3"}. The pseudo code for this is: IF n$ <> "" THEN; or IF NOT(n$ = 0) THEN. A faster method is to test the length of the character string because a null string has a length of zero. The pseudo code for this is: IF LEN(n$) = 0; or IF NOT(LEN(n$) = 0). The main reason it is faster to test a character string rather than a number is the fact that in this context a number is represented by 8-characters of the IEEE format rather than by one literal character. The size of the LUT is also smaller for a literal character string: 64-elements as numbers occupies 64 * 8 = 512-characters, whereas 64-elements as characters occupy 64 * 1 = 64-characters or 1/8 less. A LUT for 9-inputs [LUT-9] consists of a 2-tuple each (2-bits) to make 9 * 2 = 18-bits. The binary number 2^18 is decimal 262,144 or those numbers in the range of the inclusive interval of [0, 262143]. This means LUT-9 is an array indexed from 0 to 262,143 that is sparsely populated with kk results as {"", "1", "2", "3"} for binary "", 01, 10, 11. (The null symbol means that if it is used in a multiplication or exponential calculation, the resulting number is likely to raise exceptions.) The fill rate for the sparsely populated LUT-9 also shows that a single KCN rejects about 93% of all signals, and accepts about 7%. This reiterates how the KCN overcomes the deficiencies of accepting all signals described int he Background section above.

The design flow of the software implementation consists of three parts: build the LUT (as above), populate the top-tier of the KCL with input values, and process the subsequent lower-tier KCLs. For testing purposes, the input values are generated randomly in the range interval of [0, 2462143], that is, at the rate of 9-input signals at once. These are checked for results (valid or invalid) and used to populate the top-tier of KCL. The size of the top-tier level is determined by the maximum memory available in the software programming environment. In the case of True BASIC®, the maximum array size is determined by the maximum natural number in the IEEE-format which is (2 ^ 32) – 1. The largest radix-3 number to fit within that maximum is (3 ^ 20) – 1. However the compiler system allows two exponents less at 3 ^ 18 (3 ^ 18.5, to be exact). Hence the top-tier KCL is set as KCL-18. Subsequent lower-tier KCLs are processed by string manipulation. Consecutive blocks of 9-signal inputs are evaluated for all valid results. The valid results as single ordinal characters are multiplied to the respective exponent power of four and summed into an index value for the LUT. If the indexed LUT result is a valid result, namely not null, then the result is stored at the point in the KCL tier. This phase constitutes KCN performance.

A method of building the LUT according to an embodiment of the invention is disclosed here. A single dimensioned, 64-element array is filled with the 14-valid results as single character literals and interspersed with the 50-invalid results as null values. The array is indexed sequentially in the interval range of [0, 63] by result. Three such arrays are combined into a three dimensional array of 64 * 64 * 64 = 262,144 elements. This serves to index the three inputs of {ii, pp, qq} into all possible combinations of results. This array is then evaluated for null results where NOT(ii$ = kk$), NOT(LEN(ii$) = 0), and NOT(LEN (kk$) = 0). The respective elements are then designated as invalid results. To access the three dimensional array faster, it may be rewritten in a one dimensional array. This is because while the three indexes of ii, pp, and qq are conceptually easier to digest, a single index of 262,144 elements in the range interval [0, 262143] requires only one index value. Incidental arrays used to perfect the LUT may re-indexed to zero or null to reclaim memory space. This table is named LUT-241K and is implemented in software as 2-tuples or 2-bits for 262,144 * 2-bits or 524,288 bits at 8-bits per byte for 64K bytes.

A LUT with a 6-character key according to an embodiment of the invention is disclosed. The values of searchable element values for "iippqq" should also have the same key string-length, to enhance a radix search or binary search. Hence the numerical value of the index in the interval of [0, 63] is converted into a 2-character string value of the index in the interval ["00", "63"]. The subsequent indexes for the remaining two array dimensions are concatenated onto the first string index to form a 6-character key. The interval of digital search keys as [000000, 636363] contains potentially 636,364 keys. However, this is not exactly the case as some keys are impossible because the range is sparsely occupied. Excluding consecutive null values at the extrema of the range, the interval range of valid keys is [010002, 586353], but again not all key combinations therein are possible as the frequency or cycle of valid results is in runs of four separated by blocks of seven nulls. This is named LUT-636K.

The Phi-hash LUT as a variation is presented here. Using a 1,148,219-element Phi-hash table versus the LUT-636K with 636,364-elements above occupies about 55% more memory space and is slower to process because the character string result values must be converted back to natural integers. The Phi-hash formula uses upper case Phi = ((5 ^ 0.5) - 1) / 2 and hash_m = 2 ^ 21 - (2 ^ n), where n = 19.8559, and hash (key) = Floor_INT(hash_m * ((Phi * key) MOD 1)). The choice of n here is best determined by test so as to minimize the size of the hash space. In other words, the hash space is just large enough to accommodate all hash values with finding a unique index within that space.

Software performance is presented here. Software simulation uses pseudo code of the educator's language of True BASIC® on a no-load, quad-core laptop with 8 GB RAM. Processing a LUT of 4VLN with the "00" values folded into the null "" is about 1% faster than without combining those values. Software performance favors the LUT-241K design using radix-4 arithmetic to build index values. This executes at the rate of 1,087,752 KCNs per second. The rate is based on processing 21,523,361 KCNs. This number of KCNs processed is derived from the Sigma permutation of

$$\sum_{i=1}^{17} (3 \wedge (i - 2)) \qquad \text{Equation (4).}$$

A hardware implementation is presented here. Of critical importance to implementation in hardware is the particular hardware device(s) chosen. This is based on hardware sizing. In the case of LUT-241K, the size is 262,166 elements of 2-bits each or 524,588 bits (512Kb) or 65,336 bytes (64KB). However, additional bits are required to represent the KCLs of the KCNNs. These are calculated as a radix-3 function where 3^12 or 531,441 entries at 2-bits each is 1,062,882 bits. The LUT and data structure occupy a total requirement of 1,587,470 bits. This example is directed to the use of many field programmable gate arrays (FPGAs) to build the KCNN system at a lower cost of less than $40 per target device. In addition, performance becomes a factor for faster or slower devices. On average in hardware, one access to LUT-241K takes 13.25 nanoseconds for processing at the rate of 1.8 BB KCNs per second, which is about 1,600 times faster than in software.

Detailed descriptions of the figures are presented here.

Fig. 4 illustrates the synchronous KC model with feedback loops for external clock timing, named τ-propagation delays (lower-case tau). Steps 401 and 404 are the respective input and output places. Step 402 is a process transition. Step 403 is a feedback transition. Step 405 is a tau-propagation place, such as an external clock. Step 406 is an untimed feedback path from 403 to 401. Step 407 is a process path from 403 to 404. Step 408 is a feedback path from 404 to 405. Step 409 is a continuation of the feedback path of 404 to 401 from 405 to

401. The two feedback paths are 406 and 408/409. Feedback path 406 is synchronous in that it is controlled by logical result from 403 to route data that is timing delay based on a decision, but not by a synchronous timing delay. Feedback path 408/409 is synchronous in that it is controlled by a propagation delay in 405.

Fig. 5 illustrates a logical circuit for the KC that consists of multiple inputs, one output, and synchronous feedback loops. Steps 505 and 506 represent back propagating paths, respectively analogous to 405 and 406 in Fig. 4. Exactly how these feedback paths are stimulated is a matter of sequencing, either by decision based on data, by external clock, or by both. This KC model is ultimately synchronous, and not asynchronous.

Fig. 6 illustrates a functional process diagram of the KC using symbols for nodes and processes according to an asynchronous model. Steps 601 and 604 are the respective input and output places. Step 602 is a circuit for the logical connective of AND. Step 603 is a circuit for the logical connective of OR.

This KC model includes two logical connective gates, an AND gate 602 and an OR gate 603. The signals specified to be processed are the set {ii, pp, qq} in 601 where ii AND pp is processed in 602, and that result is OR qq in 603 to produce the result renamed as kk in 604.

Note that the instant KC model in Fig. 6 is a simplification of the previous KC model in Fig. 5 in that the instant shows no feedback loops connecting from 603 to 601 or from 604 to 602, as is the case in the previous from 503 to 501 and from 504 to 502. There are no feedback paths present in the instant model. This means it is asynchronous or untimed by its data and with straight through data flow.

This model is made further unique by the method to terminate the input of signals to 601. If kk in 604 is determined to be equal to ii from 601, then this instance of system 60 stops processing. This feature inhibits the model from the otherwise potentially endless processing of results kk to 604. This method effectively makes the model into a self-timing KC where the input ii of 601 and the output kk of 604 determine the next state of system 60 as active or dormant. This unique feature means that the model is immune to external timing constraints and is wholly self-reliant for control of its asynchronous operation on its input and output data.

Fig. 7 illustrates an electrical diagram of the KC using symbols of Deutsches Institut fűr Normung (DIN) according to an asynchronous model. Inputs of 701a, 701b, and 701c are processed to the output result of 704. The AND circuit is 702. The OR circuit is 703. Inputs 701a and 701b are processed in the circuit 702. Along with input 701c, the intermediate result of 701d is input into the circuit 703 and processed. The result is output as 704.

The numbered labels are keyed in the last digits to the respective labels in the KC model of Fig. 6. The results from gate 702 as 701d and gate 703 as 704 may be determined by binary arithmetic (using the logical connectives of AND and OR) or by LUT result based on the inputs 701a, 701b, and 701c, and in either method to produce the result 704. It is the unique feature of this model that results are obtained directly by arithmetic computation or indirectly by LUT access. In the case of computation, there is no memory overhead for the size of a LUT. In the case of LUT access, there is no computational overhead for arithmetic. Therefore, this model a generic embodiment of the KC for faster computation-less or slower memory-less requirements.

Fig. 8 illustrates a flow diagram of the KCN (Kanban cell neuron) using symbols for nodes according to an asynchronous model of the KC. The three inputs into one output is known as 3-to-1 processing. From previous operations, the respective kk results are in 801, 802, and

803. These serve as the subsequent inputs of ii, pp, and qq into 804 as a kk result. That in turn serves as an input to the next subsequent operations if any. This model maps the paths of signals named as nodes in a network tree. The input nodes are for ii in 801, pp in 802, and qq in 803, and serve as three inputs in this model to produce one output for kk in 804. The signal values at any labeled location are automatically stored there as data which is persistent for the duration of the electrical life of the model or until reassigned or erased.

Fig. 9A illustrates a flow diagram of the KCNN (Kanban cell neuron network) using symbols for nodes according to an asynchronous model. This network implements the KCN as nine inputs into one output, and named as 9-to-1 processing. Inputs 901a, 902a, and 903a result in 911a. Inputs 904a, 905a, and 906a result in 912a. Inputs 907a, 908a, and 909a result in 913a. Results 911a, 912a, and 913a are renamed respectively as 921a, 922a, and 923a. The input set of {901a, ... , 909a} is a set of 9-values that produce 924a as a single output result. The result of 924a is renamed to 931a as one of three prospective inputs to the result in 934a.

This network maps a multitude of the paths of signals named as nodes in a network tree for the model of the KCN in Fig. 8. A unique feature is how consecutive outputs of kk in the set $\{kk_n, kk_{n+1}, kk_{n+2}\}$ serve as inputs of the set $\{ii_{n+3}, pp_{n+3}, qq_{n+3}\}$ to produce kk_{n+3}. This method effectively passes results from one level of the nodes in a network tree into the next level of the nodes in the cascade of the nodes in a network tree. This mechanism is inherently combined with the method from the Introduction above where individual KCNs become dormant when input ii is equal to output kk, to make this KCNN terminate when input signals are exhausted. For example, if ii_1 in 901a is equal to kk_1 in 911a, then ii_4 in 921a is null, not set to kk_1 of 911a, and 911a terminates that KCN. This means that for a cascade of network paths to proceed requires no interruptions in the consecutive sequence of valid input values. Therefore if kk_1 in 911a is null, then any subsequent output value, such as kk_4 in 924a and kk_5 in 934a are null paths and no longer active. What follows from this is that kk_2 in 912a and kk_3 in 913a are also ignored by this network, and the next set of input signals, beginning with the unattached potential node 9nn, are potentially processed.

Fig. 9B illustrates an abstraction of Fig. 9A. A unique feature of this method is that if all input and output signals are acceptable and not null, then the output result of kk_4 in 931b may be obtained directly by one access to a LUT as indexed by the nine input signals in the set $\{ii_1, pp_1, qq_1, ii_2, pp_2, qq_2, ii_3, pp_3, qq_3\}$. In other words, this KCNN is based on nine-input signals to one-output signal as in the ratio of (3 ^ 2) to 1 or 9:1. It is KCNN model that performs most quickly, and hence is ideally suited for implementation in hardware over software.

Fig. 10 illustrates a system level diagram of a multitude of KCNNs using blocks for individual KCNNs according to an asynchronous model. There are 729 constituent KCNs named 10001, 10002, ... , 10728, 10729. The total number of KCNNs is calculated by the summation formula named 10000 and read as the sum of KCNNs from 1 to 729. Fig. 10 is an example of KCNNs mapping many biological neuron cells in Fig. 9B. Here about 94 billion KCNs (3 ^ 23 KCNs) are mapped by 729 KCNNs of the inclusive range of [10001, 10729]. This example is based on the limitation that the largest natural number of indexes to a LUT, as easily represented in 32-bit computing machinery, is about 2 ^ 32. That in turn limits the number of KCNs processed per KCNN with a LUT of 524,288 bits (65,536 bytes or 64 KB) to 21,523,361 KCNs per KCNN, as determined by the running sum of (3 ^ (i - 2)) for i from 1 to 17. If a complex programmable logic device (CPLD) or a FPGA contains about 3 ^ 17 KCNs, then such devices as KCNNs arranged in parallel sequence may easily map and quickly process large numbers of corresponding biological neuron cells. Hence Fig. 10 represents the *digital framework* for processing about 94 billion biological neuron cells. This model may be assembled and programmed into a serial or parallel cascade as a computer system network of KCNNs to perform the processes of the KCN model.

Fig. 11 illustrates a behavioral diagram of the KCNN using symbols for blocks of computer programming tasks according to an asynchronous model. Fig. 11 contains the flow chart steps to program the KCNN. In 1101, data structures and variables are initialized. In 1102, LUT(s) are built by arithmetic from primitives or by reading from a constant list of predetermined values. In 1103, signals to process are input. In 1104, results from input values are processed by LUT, by arithmetical calculation, or by both. The results from 1104 are output in 1105.

Of unique interest is the method to build LUTs in 1102. Results may be obtained by logical arithmetic, or LUTs may be constructed by either logical arithmetic or by reading data directly from a specification list, or a combination of both. However the size or extent of the LUT may be limited by the number and type of datum. The MVL chosen for exposition here by example is 4VL or a 4vbc where the values are in the set {00, 01, 10, 11} and taken to express respectively in words the logical states of {contradiction, true, false, tautology} and the decimal digits of {0, 1, 2, 3}. A fifth value of null or "" is helpful for ease in programming.

Of unique interest to this method is the rationale behind folding the 2-tuple 00 into null "". Contradiction or 00 means "not false and not true" or in other words "true and false" as absurdum. Null on the other hand has the meaning of nothing or no value. Because absurdum imparts no information about the the state of true (01), false (10), or tautology as "false or true" (11), then the informational value of absurdum is as void as to the state of falsifiability as is null. Hence the 4VL adopted in this exposition is the set {"", 01, 10, 11}. This means that the whenever an input signal of 00 or "" is encountered, it short circuits and voids that KCN processing it.

The methodology for building the LUT for three inputs to one output is presented here. For the three input variables as in the set of {ii, pp, qq}, each variable of which is a 2-tuple as in the set of {"", "01", "10", "11"}, there are 2 ^ 6 or 64-combinations possible, and typically indexed as in the inclusive interval range of [0, 63]. Of these 64-combinations, there are 14-combinations that do not include the value "", as presented in Table 1 for the 14-combinations excluding "".

Connective No.	((ii	& qq)	| pp)	= kk
090	01	01	10	11
095	01	01	11	11
106	01	10	10	10
111	01	10	11	11
122	01	11	10	11
127	01	11	11	11
149	10	01	01	01
159	10	01	11	11
165	10	10	01	11
175	10	10	11	11
181	10	11	01	11
191	10	11	11	11
213	11	01	01	01
234	11	10	10	10

Table 1. Connective assignments to 4vbc

The connective number is the decimal equivalent of the binary digits. For example, binary "11 10 10 10", with most significant on the left, is decimal 234. The connective number is meaningful as an identifier in the mathematical theory of 4vbc which has 256 8-bit logical connectives as < 0, 1, ... , 254, 255>. When the 14-combinations of Table 1 are placed in the LUT of 64-entries, the frequency of distribution is sparse.

The implementation methodology for building the LUT for nine inputs to one output is presented here. Three instances of the table of 64-elements are manipulated to produce all possible combinations. Each combination of three inputs and one output is further checked for the exclusive condition of ii = kk for which that combination is subsequently excluded as null "". The resulting LUT consists of 64 ^ 3 or 262,144 entries, each of which is a 2-tuple. In the source code in True BASIC®, this LUT is populated by manipulation of input arrays from minimal DATA statements. In the source code in VHDL, this LUT is enumerated bit-by-bit and occupies over 300-pages of text.

Forward looking rules engine

The KCNN makes use of clusters to assign logical values to statistics in times series.
The data set of a variable is sorted in rank order then divided evenly into the number of logical values of the multi valued logic. For example with four logical values as in 4VL, the sorted list of <10, 20, 30, 40, 50, 60, 70, 80> is assigned respectively to logic values of <00, 01, 10, 11> as clusters: <00> for <10, 20> ; <01> for <30, 40>; <10> for <50, 60>; and <11> for <70, 80>. Alternatively the numeric values are assigned as < 1, 2, 3, 4>. Similarly, the assignment respectively of logical values to clusters could be in a reverse or different order as clusters: <11> for <10, 20> ; <10> for <30, 40>; <01> for <50, 60>; and <00> for <70, 80>. In this example, the assignment of values does not include weighting, such that all clusters do not have the same count of statistical values as cluster assignments of: <00> for <10> ; <01> for <20, 30, 40>; <10> for <50, 60, 70>; <11> for <80>.

To determine cluster assignments the sorted order of values for variables may be sorted in ascending or descending order. For example with meteorological time series by collection date and/or time, cluster values may be selected for dry temperature C, wet temperature or dew point C, relative humidity %, barometric pressure kPa, and wind speed km/h. Similarly for econometric time series by trading date and/or time, cluster values may be selected for volume and the price at open, close, high, and low. The justification for which type of sorting order is determined by trial and error tests of the data.

To determine parameters for the forward looking rules engine, a comprehensive tabulation of all possible combinations of variable values indicates the logical signals of interest. For example in either data set above, there are five variables each in ascending and descending order but only three such variables taken at a time, and excluding repetition of the same variable in opposite sort order, to serve as valid inputs. For n = 5 * 2 = 10 and k = 3, from (k - 1)!(n - (k - 1))! there are 80,640 possible combinations.

Some of these combinations are irrelevant because they do not make sense. For example with meteorological time series, if the variable to predict is wind speed, then four of the five variables are of interest. For n = 4 * 2 = 8 and k = 3, from (k - 1)!(n - (k - 1))! there are 1440 possible combinations. Furthermore if temperature is consistently taken as either dry or wet, then n = 3 * 2 = 6 and k = 3 for 48 possible combinations. If temperature and relative humidity are consistently taken as one sort order and pressure taken as the opposite sort order (because there is a known inverse relation between pressure and temperature), then n = 3 and k = 3 for 2 possible combinations depending on the sort order.

Similarly for example with econometric data, if the variables for price at open and close are deemed irrelevant (except to show return on a sell or buy position), then there are three remaining variables of price at high and low and volume. Because there is no known inverse relation among these variables, they are all either in ascending or descending sort order. Hence from n = 3 and k = 3 there are 2 possible combinations depending on sort order.

These procedures establish the forward looking rules engines for the respective examples. It is the assignment of the time series variables to the input variables in the KCN formula and especially the subsequent evaluation of output signals that is left for the analyst.

Application to meteorologics

The KCNN as a model is applied to meteorologics. The four types of systems to predict wind speed are classified as: statistical; physics-based; data mining; and hybrid [Kusiak 2013]. This model is not statistical because autoregressive analysis such as Grey systems [Zhang 2012] does not apply (see discussion of Fig. 1 in Section V). This model is not physics-based because numerical weather prediction models or Kalman filters are not used. This model is not based on data mining because recursive neural networks or support vector machines are not used. This model is not a hybrid ensemble because the foregoing types are not modified. This model is rather a distinct fifth type based on cluster logic to mean logical arithmetic applied to data clusters.

The variables evaluated are published as dew point (wet bulb temperature) and dry temperature in degrees C, relative humidity in %, barometric pressure in kPa, and wind speed in km/h. In general, low pressure implies higher wind speed, and high pressure implies lower wind speed. The variables are recorded by date at an hourly rate on the hour. The hourly period is known as medium-term wind speed prediction [ncdc.]. The temperature values are accurate to the nearest tenth degree Centigrade [climate.], as opposed to the nearest rounded degree Fahrenheit [ncdc.]. The wind speed values are accurate to the nearest km/h [climate.], as opposed to the nearest rounded mph [ncdc.].

The population is published by government weather services. The populations examined are for North America in Canada and USA. The population retrieved is for hourly statistics for the month of October, 2013 for the weather station at Ottawa International Airport, Canada. At 24 hourly statistics for 31 days, this equals 744 potential points of time.

For October, 2013 the published population of October 7 excluded ten records for 09:00 - 18:00 hours. Therefore the sample size is reduced to 734 points of time. The sample is available from [climate.]. An example of eight consecutive points of time appears in Table 2. Observed bits are those predicted.

Date	Dew Point C	Pressure kPa	Relative Humidity %	Temp Dry C	Speed km/h	Observed	Expected
2013.1031.03:00	2.3	100.43	97	2.7	9	01	01
2013.1031.02:00	1.4	100.52	97	1.8	10	01	01
2013.1031.01:00	1.0	100.58	97	1.4	10	01	01
2013.1031.00:00	1.1	100.64	98	1.4	8	01	01
2013.1030.23:00	1.1	100.75	94	2.0	5	00	01
2013.1030.22:00	0.8	100.83	93	1.8	10	01	01
2013.1030.21:00	0.3	100.80	89	1.9	8	01	01
2013.1030.20:00	-0.2	100.82	83	2.4	6	00	01

Table 2. Example of hourly statistics and derived logic

Because all such points are evaluated, there is no consideration for selection with replacement or selection without replacement.

To evaluate wind speed prediction, the literature relies on statistical metrics based on margins of error [Kusiak 2013, Zhang 2012]. Absolute error (AE) measures the magnitude of the absolute value of the difference between the predicted (approximate) and the observed (exact) values. Mean average error (MAE) is an average of such absolute errors. Both AE and MAE are not necessarily assumed to be based on a ratio scale which has a true point with a meaningful zero value.

By extension, the mean absolute percentage error (MAPE) depicts the percentage error of the predicted values. However, some problems may arise. For zero as the observed value, an instance of the absolute percentage error is undefined from division by zero. For a perfect prediction fit, MAPE is zero. However for an imperfect prediction fit, MAPE has no upper limit value. Taking an average of MAPE for multiple time series produced by different methods also may be distorted by MAPE with high values.

The statistical expression of (1.0 – MAPE) converts a method to a scale where a higher percentage means a more accurate prediction. However the number of points evaluated from a time series is washed by the averaging process so that a record of degrees of freedom for the independently observed data points is not required.

Consequently the statistical method adopted in this study is a collection of observed and expected values for an N-by-M contingency test. The Chi-square test (χ^2) is a subset of the contingency test.

1) Observed Values: These are the logical values of applying the instant KCN formula to the input variables, excluding wind speed which is to be predicted. The observed values are the 2-tuples of < 00, 01, 10, 11 >. There are four observed values with none needing correction for sample size.

2) Expected Values: The expected values are the 2-tuples of < 00, 01, 10, 11 > which represent four equally random numbers of prediction of wind speed. For example with 734 points of time, one-fourth or 183.5 points of time are expected to be predicted randomly. Four possible expected values are within each such cluster. Hence any given 2-tuple predicting that same 2-tuple is one fourth of 183.5 or 45.8 points of time. This is rounded up to 46 points of time per cluster of 2-tuple.

3) Calculation of Raw Results: The N-by-M contingency test here is a table of 4-rows and 2-columns. This is the format of a χ^2 test. The degrees of freedom are df = 3.

4) Presentation of Statistical Results: The Fisher P is used in the format of "P <= 0.n" to be read as "statistically significant at a level equal to or less than zero point n".

The χ^2 matrix in four clusters and result is in Table 3.

Cluster by 2-tuple	Observed prediction	Expected prediction
11	96	46
10	46	46
01	53	46
00	47	46
	$\chi^2 = 11.84$; df = 3	P <= 0.01

Table 3. χ^2 Matrix

The test results are statistically significant at a level equal to or less than 0.01.

The model predicts only the highest wind speed within the range of one of four clusters. The model can predict wind speed within the range of one of four clusters. In fact, the wind speed is predicted in the highest speed range within the four clusters.

Fig. 1 shows with increasing time on the x-axis the distribution of clusters of wind speed predicted for the sample. The clusters < 00, 01, 10, 11 > are scaled up by one unit and numbered < 1, 2, 3, 4 >. This reserves a cluster named < 0.1 > for the incorrect predictions.

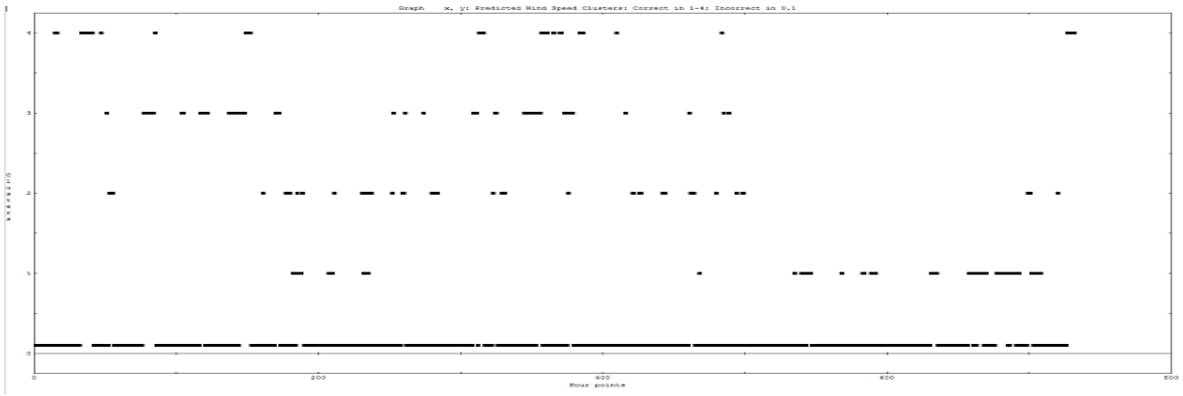

Figure 12. Distribution of wind speed predicted by cluster for increasing time

In Fig. 12, it is mistaken to connect the points with a line because the points do not constitute a function. Each point is a vertical, directed ray with undefined slope (not infinite slope). What follows from the model not being a function is that no calculus is associated with it. Therefore regression analysis of the model is not meaningful.

Fig. 13 shows with increasing time on the log x-axis the frequency of clusters of wind speed predicted for the sample. Cluster < 4 > as the highest wind speed is the most frequent. Incorrect predictions are in cluster < 0.1 > and not in cluster < 0 > because log(0) is undefined for a graph of log x-axis.

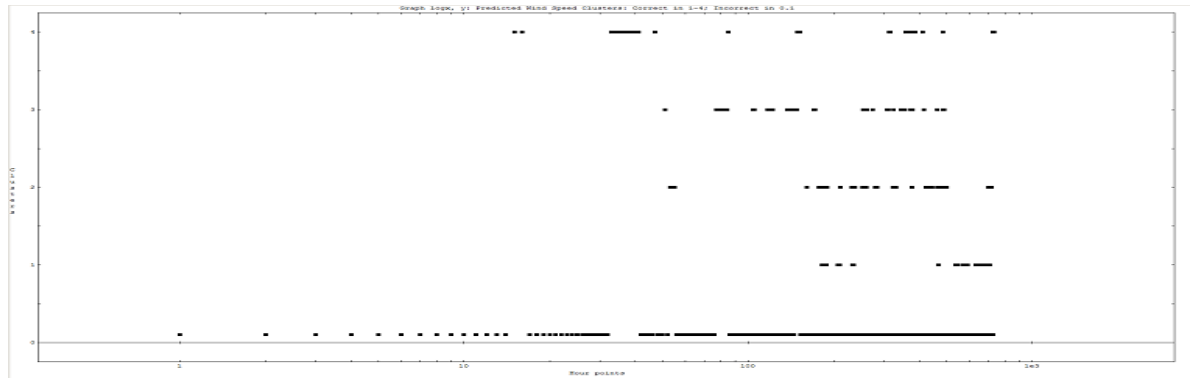

Figure 13. Frequency of wind speed predicted by cluster for increasing time in log x-axis

From historical weather data of one month of hourly statistics, the model predicts wind speed at a statistically significant level. It is conjectured that subsequent months of historical data produce nearly identical results.

Application to econometrics

An example of how the KCN is applied to prediction of stock trading signals is in Table 3. This is a virtual book of performance. The equities traded are Chinese exchange traded funds (ETFs) which are tied to one or more indexes such as Standard and Poors SPY. Such indexes in general always rise slowly in time. These ETFs are noteworthy because their holdings are in a market manipulated by the Chinese government. Nonetheless after about 60 trading days there is an annualized profit above the market.

The buy and sell signals are from the KCN and based on volume and prices for high and low. The particular trading signals and evaluation are not disclosed and patent pending.

59 Consecutive trading days from 2014.09.22 back to 2014.06.30

Issue symbol	Buy date	Buy price	Buy signals	Sell signals	Sell date	Sell price	Net profit	Profit %	Annual %
ASHR	07/11/14	22.19			07/21/14	22.62	0.43	1.94	
ASHS			none						
CHLC	08/11/14	25.5			08/18/14	25.6	0.1	0.39	
CHXF			none						
CHXX	06/30/14	16.88			07/21/14	18.25	1.37	8.12	
CN	07/21/14	26.72			08/11/14	28.68	1.96	7.34	
DSUM	07/11/14	24.84			08/11/14	25.04	0.2	0.81	
ECNS	06/30/14	45.54			07/21/14	45.96	0.42	0.92	
FCA	09/22/14	[22.99]							
FCHI	06/30/14	46.78			07/21/14	48.48	1.7	3.63	
FXI	06/30/14	37.04			07/11/14	37.9	0.86	2.32	
	09/02/14	40.47			09/08/14	42.52	2.05	5.07	
	09/22/14	[39.87]							
GXC			none						
MCHI			none						
PEK	07/11/14	28.1			08/11/14	31.14	3.04	10.82	
PGJ			none						
QQQC			none						
YANG			none						
YINN	09/02/14	33.89			09/08/14	39.12	5.23	15.43	
	09/22/14	[32.17]							
Totals		347.95				365.31	17.36	4.99	21.31
		Buy price				Sell price		Profit %	Annual %

Table 3. 59 consecutive trading days from 2014.09.22 back to 2014.06.30

Acknowledgments

Thanks are due for helpful discussions to Sang Du, Tony Story, and the referees.

References

climate.weather.gc.ca/climateData/hourlydata_e.html?timeframe=1&Prov=ONT &StationID =49568&hlyRange=2011-12-14

James III, C. (1998). A reusable database engine for accounting arithmetic, Proceedings of the Third Biennial World Conference on Integrated Design & Process Technology, 2: 25-30.

James III, C. (1999). Recent advances in logic tables for reusable database engines. Proceedings of the American Society of Mechanical Engineers International, Petroleum Division, Energy Sources Technology Conference & Exhibition, ETCE99-6628.

James III, C. (2010). Proof of four valued bit code (4vbc) as a group, ring, and module. World Congress and School on Universal Logic III.

James III, C. (2013). Recent advances in algorithmic learning theory of the Kanban cell neuron network. IEEE Proceedings of International Joint Conference on Neural Networks. August, 2158-63.

Kusiak, A., Zhang, Z., Vermas, A. (2013). Prediction, operations, and condition monitoring in wind energy. Energy. 60:1-12.

ncdc.noaa.gov

Zhang, J., Cheng, M., Cai, X. (2012). Short-term wind speed prediction based on Grey system theory model in the region of China. PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review). 88:7a:67-71.

Caption text of figures

Figure 1. A Kanban cell in a Petri net

Figure 2. Accounting arithmetic system in N-dimensions

Figure 3. Synchronous, self-timing neural network as feedback loops

Figure 4. Synchronous Kanban cell in feedback loops as external propagation delays

Figure 5. Logical circuit for the Kanban cell

Figure 6. Functional diagram of the Kanban cell in symbols for nodes and processes

Figure 7. Electrical diagram of the Kanban cell in symbols of Deutsches Institut für Normung (DIN

Figure 8. Flow diagram of Kanban cell neuron

Figure 9A. Flow diagram of a Kanban cell neuron network

Figure 9B. Flow diagram of a Kanban cell neuron network as a LUT

Figure 10. System level diagram of multiple Kanban cell neuron networks in blocks of tasks

Figure 11. Behavioral diagram of the Kanban cell neuron network in blocks of tasks

Figure 12. Distribution of wind speed predicted by cluster for increasing time

Figure 13. Frequency of wind speed predicted by cluster for increasing time in log x-axis