T(emporal) C(onnection) N(etworks)

## Historical N(eural) N(etworks)

TCN is an entirely novel way to create an A(rtificial) I(intelligence) simulation that moves beyond the traditional neural network Hebbian-learning networks. In the traditional NNs, each layer is connected forward and backward to the adjacent layers. Learning is achieved by adjusting the weights between the adjacent layers to achieve categorization of the inputs through training or self-organization. For supervised training, the NN output is compared with known correct values, and training sessions are used to compare the predictions of the NN with the training data set. The NN then determines the deviations of the NN prediction from the correct answer and goes back through the NN connections, adjusting them to reduce the error between the NN prediction and the correct result presented for the training session.

Early Perceptrons and NNs had significant limitations in solving certain problems that were resolved by implementing backward connections between adjacent NN layers. NN training was slow and computational intensive. As computer resources grew rapidly, NNs with many more nodes per layer and higher layer counts were created, yet the basic method of operation remained largely the same.

Various self-organizing NNs were contrived that didn’t require a training set, but were designed to recognize patterns in the input data without the need for a training set of examples.

These early NNs had a common base. Neural nodes were arranged in layers with every node on each layer connected to all the nodes in the layers above and below. Further, there were no intra-layer connections, forward connections dominated over backward connections, and connections did not skip beyond adjacent layers. Further, the connections were always of fixed geometric distance from each other. All connections had the same immediacy, only the weights were varied through the training process.

The inter-connections were modulated using a logistic or similar function that provided symmetrical non-linearity to the inter-connections’ weights. Different non-linear modulation functions provided changes in NN performance providing a better outcome for various problem domains.

These NNs have been very successful across a broad range of problems requiring categorization of complex pattern recognition but they remain unrepresentative of the way actual brains work.

## T(emporal) C(onnection) N(etworks)

### Physical Structure

TCNs are constructed and modeled after the way a real brain is built. Individual nodes, or neurons, are not arranged in layers, but are inter-connected to multiple other nodes depending on their temporal distance, rather than in some artificial layered structure. The temporal distance between neurons depends on their physical interconnectivity and the speed of axon spike propagation.

There is no restriction conferred by the artificial layering of neurons, and forward and backward connections are equally likely. Further, some connections carry positive weights – known as excitory connections – whereas other connections carry negative weights – and are known as inhibitory connections.

Any neuron in the TCN can connect with any other and is not limited by any layering as is the case with traditional NNs.

There are three elements in a TCN; neurons, connections, and signals. Each will be examined in turn.

### TCN Neurons

These have three simple roles in a TCN. They act as the anchor for connections, they integrate incoming signals, and when the incoming signals exceed a threshold, they cascade and create an outgoing signal of their own. The outgoing signals are transmitted to any other neurons that are connected to the cascading neuron.

Incoming signals arrive at a neuron via connections with a signal size modulated by the connection and at a time determined by the temporal distance from the neuron at the originating end of the connection. The neurons accept incoming signals within a relatively short time window before any incoming signal declines, fades, and is forgotten. Enough signals have to arrive at a neuron within this aggregation window to cause a neuron to cascade.

When a neuron cascades, it enters a refractory period during which any newly arriving signals are ignored and discarded. Once the neuron exits its refractory period it will again start accepting, queueing and aggregating incoming signals.

### TCN Connections

All neurons have connections to other neurons; if a neuron does not have any connections it is of no use in a TCN or a brain. In a TCN the connections are analogs for the synapses in a real brain. Each connection has a temporal distance specification which determines how far in the future a signal sent down the connection will arrive at the destination neuron. The signals exert either a positive –excitory influence or a negative—inhibitory influence—on the target neuron. The sum of positive and negative signals are combined by the neuron to determine if the neuron has accumulated enough signal inputs to cascade and create an ongoing signal of its own.

Each connection determines the temporal distance from the origination neuron to the target neuron. Any signal injected into a connection by a cascading neuron is delivered to the target neuron, delayed by the temporal distance between the connected neurons.

The connections also perform another vital function. As multiple signals are sent down a connection in a short space of time, the strength of the connection is quickly enhanced, and then declines over time if no more signals or very few signals are delivered over the same connection. The signal arrival rate is termed ‘tetanic’ and is typically ten or twenty signals arriving within less than a second. This simulates S(hort) T(erm) P(otentiation) which is the basis of short-term memory in the brain.

There is a secondary, similar mechanism that increases the amplitude of the connections when signals arrive regularly over an extended period of time, typically ten or twenty signals with two to four hours. If the signal is enhanced to a certain level, then signal amplitude is permanently enhanced, and this is L(ong) T(erm) P(otentiation) which simulates long term memory in the brain.

In traditional NNs signal scaling was implemented using the symmetric non-linear logistic or similar function that prevented network over-saturation by very strong or very weak signals. The change of inter-neuron NN weights between the layers, modulated by the non-linear functions was the way that NN learning was implemented.

But this is not how learning occurs in a real brain. STP and LTP are the result of ionic infusion and diffusion processes that change the signal weight scaling for synapse. These are embodied in the connections using variants of the ‘leaky bucket’ algorithm. The STP algorithm allows rapid build up to a maximum scaling factor followed by a steady decline back to the base value without any new arrivals.

The LTP is different. The infusion and signal build-up takes place over a longer period of time than STP. If it ever reaches a maximum, the LTP scaling does not decline back to the base, bur remains elevated for ever, simulating long-term memory formation.

### TCN Signals

When a neuron exceeds a threshold, it cascades and creates a signal. The signal is propagated to all the neurons that are connected to the cascading neuron. The signals are sent via the connections, which determine how far in the future the signal will arrive at the neuron that is the target of the connection.

The connection will modify the size of the signal based on any STP or LTP modifications.

## Implementation

### Signals

These are the simplest of the objects. They have a size and a time. When they are delivered to a connection, the connection adds the connection temporal distance to the signal origination time and sets that as the delivery time to the target neuron.

Likewise, the connections scales the signal size by the connections STP or LTP modifiers, and also makes the signal positive or negative.

The signal is then enqueued on the target neuron’s incoming signal queue with the modified amplitude.

### Neurons

Their roles are relatively simple. Incoming signals are enqueued on the neurons and the neuron aggregate within a small aggregation time window to determine if sum of the incoming signals exceed the cascade threshold.

If they do, the neuron creates a signal of its own and forwards it to all dependent connected neurons. The connection definition is responsible for any modifications to the emitted signal.

Once a neuron has emitted a signal it enters a refractory period. During the refractory period, any enqueued signals are ignored through the end of the refractory. Any new incoming signals ae also ignored during the neuron refractory period.

### Connections

Connections are at the very center of the TCN—and the brain. They are the most numerous of the elements in the TCN. They record the temporal distance between neurons, they record the sign of the signals, and they keep track of the current state of enhancement of the connection for STP and LTP.

STP and LTP signal enhancement are at the focus of TCN learning.

When a signal is delivered to a connection for delivery, the connection applies the scaling and temporal displacement to the signal and delivers it to the target neuron where it is enqueued for future summation.

## Processing

All processing takes place at a given digital clock time. Any signal generations, STP or LTP summations, signal ageing are computed at the current digital clock time. Any results of the computations at the current digital clock time will occur in the future.

If signals in a queue are no longer relevant, because they have aged out or fall within a refractory period, the cheapest solution is to just ignore the signals and drop them from any queue as they are no longer relevant.

Signals are ephemeral, being continuously created, processed, and discarded. Neurons and connections live for ever. Signals are implemented using a ring buffer; they are created, consumed, and dropped.

## Timing

The TCN is a digital clock simulation of a brain. The smallest clock time is the oldest signal time within a TCN and this is the first time at which activity will be processed.

When a TCN is swept for activity at a given clock time, any signals or events created as a result of the current clock time will, by definition, occur at some future clock time. It may be at the next tick of the clock, but it will always be some relative time in the future. This permits multiple worker threads to sweep across the TCN in parallel, looking for work. No thread can create any event that must be handled by any other thread at the current clock tick.

Once the TCN has been fully swept for work, the clock is advanced to the next clock tick that has any useful work to be performed. This is optimized by each worker thread keeping track of the smallest (i.e oldest) future clock time it encounters during the look for work. The next clock time will be set to this smallest future clock time, effectively skipping over all clock times that have no useful work to be enacted.

## Storage

Any useful TCN must be capable of handling millions of simulated neurons and tens of millions of connections between the neurons. This has dictated close attention to how these entities are implemented.

Using higher lever languages to represent dynamic objects – such as Perl, Python, Java, or even C++ -- imposes on every object the overhead of pointers to default constructors, destructors, and other automatic storage management functions. This burdens an object-oriented provisioning implementation with many megabtyes of unnecessary overhead. In the case of signals this memory overhead would exceed the storage needed for the signals themselves. The default implementation for object creation in each of these languages is to use the stack which is limited to megabytes of memory. The requirements of a TCN is for gigabtyes of memory.

The implementation chosen is to create arrays of neurons, connections, and signals, provisioned from the heap area of the operating system and having iterators sweep across all of the arrays inverting the normal object-oriented paradigm.

Neurons and connections are statically created and provisioned and remain for the life of the TCN. Signals are different. As the simulation proceeds, signals are created by cascading neurons, are transmitted via the connections and enqueued on the target neurons. Signals remain enqueued until their future clock time arrives when they are either processed or dropped. In some cases signals arrive at a neuron which is in its refractory period and is summarily dropped.

The ephemeral nature of signals dictates a dynamic rather than static pool of signals. This is achieved by a large ring-buffer of signal-capable slots. Slots remain active until consumed or dropped, when they are effectively returned to the ring-buffer pool. When the ring-buffer has reached the end of its capacity, it wraps.

## Sizing and Computation

The advent of modern 6nm fabrication for computer memories has opened the possibility to create digital brain simulations comparable in size to biological neurons. Previously the size and power requirements for the huge memory requirements to digitally simulate brains have been prohibitive.

Neurons range in size from 4 – 100 micrometers in diameter, or 4,000-10,000 nanometers. Modern computer memories use 6-10 nanometer technology, yielding eight-bit memory units of about 500 nanometers. This means an eight-byte TCN storage unit is comparable to the size of the smallest of neurons, which opens the way to building a TCN digital brain simulation comparable in size to a biological brain.

Biological brains are also well known for low power consumption and highly parallel operations. As described above under “Timing” the TCN, being governed by a master clock, allows many iterators to work in parallel across a distributed TCN. Only iterators that find useful work to be performed at the current clock tick will have any computational load. All other iterators will simply skip all queued signals that have not yet reached their future signal time. Iterators will return the earliest clock signal for which they have discovered signals to be processed.

The minimum future clock time will become the next tick of the master clock from which all iterators commence their next scan for work. Most iterators should find little or no work to be performed for a given clock tick leading to very fast and efficient processing of the TCN brain simulation.

## STP and LTP

As noted above, these are implemented similarly to the classic ‘leaky bucket’ algorithm to mimic the ionic infusion and diffusion processes that implement these functionalities in nature. For speed of implementation the TCN does not use complicated run-time formulas to compute the STP and LTP levels.

Instead, all STP and LTP have pre-computed tables for their results. Connections STP and LTP implementations use an efficient look-up procedure to determine the nearest fit for the require STP and LTP multipliers. These tables can be pre-computed and created to any required level of clock granularity and the STP and LTP resolution times remain constant, regardless of the granularity of the tables.

## Summary

TCNs are designed to mimic the processes found in real-life brains. Great attention has been paid to the efficiency of the implementation for both memory and processor requirements.

Given the ability to process all parts of a TCN in parallel for a given digital clock time, in the future it will be possible to create TCNs distributed across networks of inter-connected computers with global clock management and neuron-connection labeling. At the end of a multi-computer clock tick sweep of the distributed TCN, any generated inter-computer signals must be delivered across the network and the next clock sweep initiated. This will allow the TCN architecture to be extended across a huge number of cooperating processors, maybe approaching the scale of real-life brains.