Emulating alpha and beta waves to achieve parallel processess attention control in memory

amaggdy12@gmail.com

Abstract: Observing the current world of computing ,and the single core scaling stalling behind the number of cores growth manufacturing direction ,together with the fact that NNs are unaware of the concept of time(only sequences & batches) giving rise to many troubles both on the hardware level; not being able to take full benefit of all cores ,and on the memory-related NNs design level. **through this study**, it is intended to introduce the concept of time to machines independent of a processor's performance/clock rate and give rise to a unit of measurement for a layer's performance with regards to Time only, alongside suggesting a Design structure to make full use of every core's potential(occupancy) constructing a dynamic working memory.

Introduction:

Neural networks initially were intended to replace classic and fuzzy based algorithms as they required too much lab our , were computationally expensive and never coped well with the chaotic (non-linear) real world.

RNNs (especially LSTM) were then used to construct memory-based dynamic moving primitives but RNNs are Gradient-based which meant "sequential" expensive computation beside suffering other problems as the vanishing gradient. LSTM was the one that maintained a manageable backwards flow of error through methods such as Hessian-free second-order optimization method preserving the gradient by estimating its curvature; or using informed random initialization, then a Kalman filter was added on top of that; so Accelerating inference in RNNs became even more difficult due to their increasing complexity and being of an inherently sequential nature, making LSTM more computationally expensive; abusing only a percent of the G.P.U.'s cores and consequently the rest are resting at idle (wimpy cores).

Although Reservoir computing has been around for a while now, it seems to be frowned upon still as it seems a bit chaotic and that is what enticed ~~the author~~(me) to use Echo state networks as they appear to be strongly related to the "order out of chaos" and the "chaos gives birth to a star" ideas.

ESNs are computationally inexpensive, not memory straining and do not suffer the vanishing gradient problem , in fact, they are notorious for not converging easily if not tuned correctly as they retain all past memory so they were used here to construct the basic deep network, and then concepts of "working memory" were built on it.

"Working memory " is not a storage per se, but a mental workspace utilized during planning, reasoning and solving problems. Most psychologists differentiate WM from "short-term" memory because it can involve the manipulation of information rather than being a passive storage [1].

Along the same lines, Engle et al. [2] argued that W.M. is all about the capacity for controlled, sustained attention in the face of interference or distraction.

Attention-control is a fundamental component of the WM system and probably the main limiting factor for capacity [3, 4]. Consequently, the inability to effectively parallel process two-attention demanding tasks limits our multitasking performance severely.

Over the past several decades psychologists have developed tests to measure the individual differences in WM capacity and better understand the underlying mechanisms. These tests have been carefully crafted to focus on the specific aspects of WM such as task-driven attention control, interference and capacity limits [5]. The best known and successfully applied class of tasks for measuring WM capacity is the “complex span” paradigm.

The challenge presented by complex span tasks is recalling the list of items, despite being distracted by the processing task.

Studies show that individuals with high WM capacity are less likely to store irrelevant distractors [6] and they are better at retaining task-relevant information [7].

Developing task-driven strategies for cognitive control are essential for the effective use of WM[8].

Using gates to control admittance of data into deep memory is a task-driven attention control.

Conceptions:

“ESN” and “reservoir” terms are used interchangeably denoting a network comprising a no. of randomly connected units.

ESN’s size must be as small as possible so it could minimize the next reservoir’s “wait time” but call overhead must be in head as well.

ESNs are notorious for not converging easily if not tuned correctly as they retain all past memory so we use a single ESN as an input related oscillating function with as low as 40 units constituting it i.e., when input is “big” or “slow” , output frequency is proportionally slow obeying a linear law and we call it ClockESN.

We use two ClockESNs, supplying the memory cortex with two regulating different sine waves, for each sine wave: [0,1] denotes allow and [1,0] denotes block.

One clock generates a relatively fast wave controlling the “primal” working memory and the other relatively slow one controls the “primitive” long-term memory.

ClockESN supplies frequency to a number of gates each controlling exogenous input to a stack of ESNs

ClockESN supplies frequency to a number of gates each controlling input from the previous stack to the next stack

ESN stacks grow bigger in number of reservoirs and no. of units per reservoir as we go deeper and by deeper we mean closer to the most superficial of the decision related networks

Big-memo comprises two hemispheres :

- one hemisphere of parallel connected stacks of ESNs corresponding to memory cortex

- one hemisphere of a sequential stack of ESNs corresponding to behavioral cortex/scratch pad(or muscle memory)

To achieve basic attention we could adopt a few concepts and strategies:

The input to every cortex is a time-indexed stream of items. At a higher level, we can view the input as a concatenation of various subsequences that represent and require different functional units of processing[8].

we use a constant-sized set of “command markers” to both mark the beginning of a subsequence as well as to indicate its functional type.

Researchers seem to adopt one of two methods of training :

Some seek to add context parameters to dynamical movement primitives (DMP) to generalize to new objects.

Others seek to associate these actions with the different contexts and get an overlap.

I tried to fuse both approaches by associating time to the shape of the pulses sidebyside with actions while sticking to a relatively deep complex ESN.

To catch something , we move in its general direction based on an action indicating the start of sequence (Ex: baseball player) before getting hold on real info about its current position using prediction which is basically memory of past time-coupled events fed to it previously .

This implies that the Deepest "most primitive" (biggest and slowest)in the memory cortex is connected to the most superficial action network to construct reflexes ,then faster acting yet more complex (going through more stages) networks signal comes later to correct .

Big-memo Design strategy:

Let us start with some basic notations:

H_1= memory -hemisphere

H_2= action-hemisphere

C_1 = cortex1 C_2 = cortex2

CK_1 = clock1 CK_2 = clock2

R_{ij} is reservoir R in position i and stack j

Input is a tuple of (I, N)

I is the input value

N is the number of times input is allowed to propagate through reservoirs and change their states before being terminated which basically resembles TTL(time to live) , to achieve a selective memory layer on top of normal memory .

what we need now is a definition of time, independent of a processor's clock, to decouple reservoir's execution time from c.p.u. 's frequency:

Assuming constant core performance and no recurrence overhead:

Assume a reservoir sync clock (base clock) produces a frequency ($1/RS$) such that it produces 10 Giga oscillations in 1 second.

So a frequency of ($1/RS$) is unrelated to c.p.u.'s frequency (a software layer can make sure it stays irrelevant).

One tick (T) equals one "instruction" completed; time between the admittance of input I(n) to a reservoir and first output O(n) sprung .

Example: assume a tick on one processor's core is 0.1RS which is a lot , then that corresponds to:

- Time between the admittance of input to the reservoir and its first output is T= 0.1RS

- Time to achieve one T is $\frac{1}{0.1 x 10^{10}}$ seconds.

- output max freq is 0.1 $1/RS$ = 0.1x10^{10} seconds.

Now we need to benchmark our system independent of the hardware, control our response for time-relevant task and control our memory propagation frequency independent of any hardware.
A tick is our benchmark of a reservoir's performance.

So the next step is to tune a reservoir sync clock(modulator) and optionally tie it to input so if input is slower/faster then the modulator will be proportionally slower/faster ,hence response speed varies for time-relevant tasks.

Sync clock controls the gate G(I) releasing input to gate G(R) (optional).

Sync clock controls the gate G(R) releasing input to reservoir $(R_{Ij})_C$ where i denotes position in stack , j denotes stack No. & c denotes Cortex No.

Sync clock controls the gate G(t) allowing traversal of memory through different cortices/stacks .

Now we try to emulate the alpha and beta waves in the human brain,

So we tune the primitive cortex C_2 to a frequency of 50T $1/RS$

,and we tune the primal cortex C_1 to a frequency of 60T $1/RS$ (60 ticks per RS).

One advantage of connecting input to the RS clock is constraining "reflexes" speed to be proportional to input speed & size and selective focus on memory regions keeping their dynamics intact.

Gate G(I) withholds external input I(n)and can be controlled by RS clock or can be normally open.

G(I) sends input to G(s) and RS clock (optional)

$G(s)_i$ is a gate on each stack of memory reservoirs for i= 1, … n

Each gate is a parent, if it gets an allow signal then it activates all underlying stack reservoirs

Each reservoir receives input I(n) from other nodes which is a tuple of (I, N) ;

I denoting input and N denoting number of steps data is allowed to move before death (N = 1) , if (N=0) then it is subject to memory fading effects only.

En is the enable bit : '0' is enable , '1' is disable. Which is checked by a parent function to activate all underlying stack reservoirs .

Input processing structure is constructed such that long-term memory is connected to the most superficial layer of the behavioral cortex constructing "instinct" while scratch-pad memory is connected to the deepest of the behavior cortex.

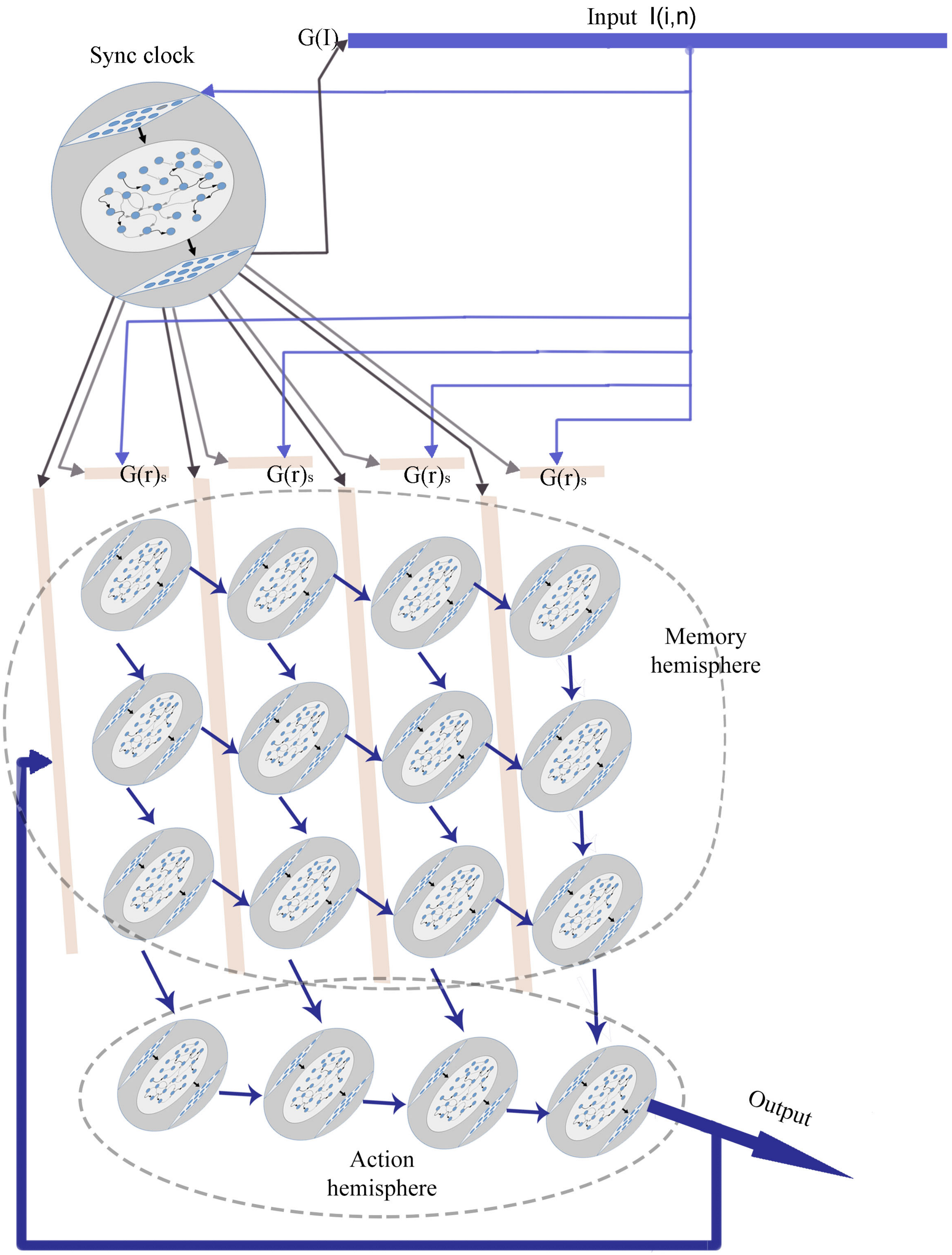
A consequence of RS definition is that the slowest reservoir will tie the rest of the hemisphere to it and we get wasted processor cycles, which again is a desirable consequence since we need to deflect from complex reservoirs and reside to complex stacks, cortices and hemispheres so we could exploit the parallel architecture.

So far, we have concluded that size doesn't matter, in fact it could be a turn-off for a number of processors leading to the "wimpy core" phenomenon.

Next we go for a bold move and take the whole memory hemisphere as a big reservoir i.e. all connecting weights are not changed once intialized.

So the only weights to be trained are exogenous input weights, action hemisphere connecting weights and output feedback weights.

And we go by another nature-inspired method , which is millions in and only one scores , so we increase the number of memory hemisphere reservoirs but keep the output (action hemisphere) minimal.


Input I(i,n)
G(I)
Sync clock
G(r)s
G(r)s
G(r)s
G(r)s
Memory hemisphere
Action hemisphere
Output

Implementation:

Basic guidelines for constructing a stable ESN are[9]:

-The spectral radius of the matrix must be less than for a stable matrix but a recent study supported that this constraint does not hold all the time a local lyapunov exponent serves better.

-If data < 1+ No. of RD units + No. of input units ,then expect overfitting.

-Think sparse not big , connections rather than No. of units.

-Input data normalization and scaling is always a good thing to avoid saturation in the activation nonlinearities(memory loss).

-Leaking rate controls reservoir dynamics hence increasing duration of short term memory.

-First training steps (like a 100) results are harmful if collected , ignore them and collect afterwards.

-A time step weighting array can be used to assign different importances at different time steps while training .

-Use output feedback only if necessary.

-Teacher forcing is more efficient and helps to a faster converge .

-Regularization or noise addition is always a good idea.

-ESNs seem to benefit the most from initial chaotic activations while being teacher forced.

-Feedback weights chosen from [-0.5 , 0.5] proved to achieve convergence the fastest.

-Noise term insertion was sampled from a uniform distribution from over [-0.001 , 0.001] to [-0.0000001 , 0.00000001]

-Due to the relative cheapness with regards to computing power requirement , many Methods of error compensation can be incorporated

-Noise term range of values can be changed overtime and is preferably high then lowered as time passes.

-if large noise was added to output feedback , larger noise was added to input to achieve stability but generally more noise leaded to better stability.

-Some Dimensionality reduction algorithms are also incorporated in the framework such as (PCA) and (KPCA) to be used instead of ridge regression or LMS.

An ESN layer script is available on github and can be integrated into tensorflow.

The philosophy behind doing it this way is to minimize the need for different layers , scaling input to be dealt with internally and scaling output to get desirable dimensions.

The first and last matrices are adapters and the middle one is the real sparse reservoir.

Next we model our cores performance:

Assuming constant internal recurrent units kernel ratio across all reservoirs

Assuming constant connectivity ratio across all reservoirs.

Assuming G.P.U. performance is stable

so a single reservoir is created and a single input pulse is presented to it then we time the first output pulse as ESNs can oscillate for a while.

redo this step multiple times and average over the results.

Now we could experiment with two choices:

one is to clone the reservoir created and construct H1 and H2 of its clones

the other is to create new reservoirs for as much as needed respecting the above assumptions.

depending on the target memory and multiplied

I go with option2 as its more chaotic

probability of getting two alike reservoirs in the same hemisphere:
P(x | H)= P(z | x) P(y | x) P(N | x)

No. of possible combinations for a reservoir of a no. of units N is P(N | x) = 2^N-1

No. of possible permutations for connectivity connections ratio c of a reservoir is :

P(c | x) = $\frac{N!}{(N-c)!}$

No. of possible permutations for recurrence connections ratio r of a reservoir is:

p(r | x) = $\frac{N!}{(N-r)!}$

basically the probability of getting two alike reservoirs in the same hemisphere is zero.

We try to investigate the theoretical speedup in latency of the execution of a task at a fixed workload that can be expected of a system whose resource usage is optimized to rely on parallelism instead of single core usage.

Exploring the Tensorflow LSTM benchmark on different “rigs”

a G.P.U. of lower core clock but more number of cores is generally slower and sometimes much slower than a G.P.U. of a faster core clock but less number of cores

while parallelism can achieve better occupancy, point-wise multiplication can cause a memory bottleneck by doing these in separate kernels; fusing them into a single kernel reduces data transfers to and from global memory and significantly reduces kernel launch overhead.

Two measures can be taken against that:

firstly, experimenting with the optimal reservoir size second, on introduction of exogenous input to many top reservoirs of different stacks , these reservoirs can be concatenated into one big matrix or more by input which should achieve less matrix multiplications leading to better occupancy.

Although parallelism comes naturally to point-wise operations, it was found that they were being executed inefficiently. This is for two reasons: firstly because there is a cost associated with launching a kernel to the GPU; secondly because it is inefficient to move the output of one point-wise operation all the way out to GPU main memory before reading it in again moments later for the next. By their nature point-wise operations are independent, and as such, it is possible to fuse all of the point-wise kernels into one larger kernel[10].

According to Amdahl, the theoretical speedup of the execution of the whole task increases with the improvement of the resources of the system and that regardless of the magnitude of the improvement, the theoretical speedup is always limited by the part of the task that cannot benefit from the improvement.

Amdahl's law gives the theoretical speedup in latency of the execution of the whole task at fixed workload.

$$S'_{\text{latency}}(O', s) = \frac{T'(O')}{T'(O', s)} = \frac{1}{(1-p)\frac{1}{O'} + \left(1 - \frac{1-p}{O'}\right)\frac{1}{s}}.$$

S_{latency} : theoretical speedup of the execution of the whole task.

s : speedup of the part of the task that benefits from improved system resources.

p : proportion of execution time that the part benefiting from improved resources originally occupied.

O : non-parallelizable part factor of reduction.

[1] Nelson Cowan. The many faces of working memory and short-term storage. Psycho-nomic Bulletin and Review, 24(4):1158–1170, 2017.

[2] Randall W Engle, Stephen W Tuholski, James E Laughlin, and Andrew RA Conway.
Working memory, short-term memory, and general fluid intelligence: a latent-variable
approach. Journal of experimental psychology: General, 128(3):309, 1999.

[3] Andrew RA Conway and Randall W Engle. Working memory and retrieval: A
resource-dependent inhibition model. Journal of Experimental Psychology: General,
123(4):354, 1994

[4] Randall W Engle and Michael J Kane. Executive attention, working memory capacity,
and a two-factor theory of cognitive control. Psychology of learning and motivation,
44:145–200, 2004.

[5] Klaus Oberauer and Hsuan-yu Lin. An interference model of visual working memory.
Psychological Review, 124(1):1–39, 2017.

[6] Edward K Vogel, Andrew W McCollough, and Maro G Machizawa. Neural mea-
sures reveal individual differences in controlling access to working memory. Nature,
438(7067):500, 2005.

[7] Ashleigh M Maxcey-Richard and Andrew Hollingworth. The strategic retention of
task-relevant objects in visual working memory. Journal of Experimental Psychology:
Learning, Memory, and Cognition, 39(3):760, 2013.

[8] Learning to Remember, Forget, and Ignore using Attention Control in Memory
IBM Research AI, Almaden Research Center, San Jose, USA † 28 sep2018

[9] A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach

herbert jaeger

[10]Optimizing Performance of Recurrent Neural Networks on GPUs

Jeremy Appleyard ⋆ Tomáš Kočiský †‡ Phil Blunsom