Russell Clarke

Welcome

Hello and welcome, this is the landing page for a redirect to some of the projects I am working on. I do other things too, artwork, reading spending time with family and cooking. I am fond of a lot of things to make use of my free time but am particularly fond of the projects which utilise the programming skills I am trying to keep up with. Here's a link to My Projects on GitHub.

You'll find a lot of them are still unfinished, this is because I get distracted with other ideas and at present do not have focus, just free reign on the Computer.

It would be great to meet some of you or talk online and come up with some sort of plan or project from concept to completion. You can find my contact details via GitHub.

I have put my Dissertation up there for a read as it will help to understand the path I am hoping to take with some of the exploratory programming. If you have any questions. I'd be happy to answer some of them.

Also, if you fancy taking a look at some of the artwork I do it can be found on deviant art.

Projects

You might notice some of my projects are incomplete. I have a lot of ideas I have to note down in partial pseudo, several of them to focus on.

Since losing employment, I now have more time to work on those projects and so I endeavour to get some of those to you for you, the reader to take a look at.

I prefer to write off the cuff and learn as I go, I try not to copy and paste because why? I won't learn anything and there is no challenge in copying someone else's work. This is the latest, I was pondering over MAC address generation for ages, left the partial pseudo, then finally, on to the final piece below.

There are derivatives which can be used from this and yes, I will reorganise my code into a more structured repository. What's interesting here is this is quite an intensive generation, I piped out to a file one evening and got to 5.3 GB and decided to stop. There is probably a nice logarithm out there which will be less processor intensive. Looks pretty cool though to see it finally working. Language is Python, I am learning Python, C and a few others on the fly, sorry if the other work takes a while to come to fruition.

A simple MAC address incrementer written in Python for generating hardware MAC addresses. Could be used for a silo to readdress hardware, owner and destruction of hardware for example, then have that specific address put back into the pool of addresses by way of Database per country.

Warning: The file it generates if saved to disk is very large. Well over 10GB.

Code

```
for w in range(16):
    m = '%x' % w
    for v in range(16):
        l = '%x' % v
        for u in range(16):
            j = '%x' % u
            for t in range(16):
                x = '%x' % t
                for s in range(16):
                    h = '%x' % s
                    for r in range(16):
                        g = '%x' % r
                        for q in range(16):
                            f = '%x' % q
                            for p in range(16):
                                e = '%x' % p
                                for o in range(16):
                                    d = '%x' % o
                                    for n in range(16):
                                        c = '%x' % n
                                        for k in range(16):
                                            b = '%x' % k
                                            for i in range(16):
                                                a = '%x' % i
                                                print(a + b + ':' + c + d + ':' + e + f + ':' + g + h + ':' + x + j + ':' + l + m)

exit(0)
```

QBit and GParticulates

About time I explained this project a little (it is on the back burner for a little while). First off, I am no Mathematician, I am learning some algebra on the fly too. This was my first serious attempt at writing a program in C.

I had a brief conversation with my father about QuBits, he literally asked me how much I know about them and quantum computing and I said in summary 'not a lot, probabilistic computing'. I couldn't sleep that night and was thinking about the logic of a single QuBit knowing from something I'd briefly read, they consider, 1, 0 and 1 and 0 at the same time to determine a probable outcome.

I set about considering how that might be achieved programmatically. Well we have to consider either 1 or 0 and 1 and 0 and the probability, I wanted to set this in motion and add the learning element to it using a DNN (Deep Neural Network).

I chose to set about writing, following and attempting to compress two existing solutions for the DNN (detailed in the code on GitHub) and for the probability chose the Markov Decision Process since I am already familiar with this process from my final project at University.

It's almost complete and I like the original in q2.c and the latest in q9.c. They require some review and verification.

I was also considering what a God particle might look like as a signal during all this code writing and mathematical exploration. It stemmed from the fusion project in the fusion repository, arriving at the conclusion it would be a sigmoid with an infinite exponent for an infinite iteration to the value of PI as it would be omnipresent cubed since it would be three dimensional, I set about trying to express it in code and Math. I didn't need to add the fourth dimension of time, since it is already rolling infinitely. Here's the code for the GParticulate.

The important bit is not the whole code and it's syntax but the expression(s).

Using the html notations from toptal and w3Schools , I think the expression I am trying to complete is similar to the expression:

$ƒ(x)^n ≡ σ((π·r^3)^{n∞})∞$

such that sigmoid σ of sine wave ~ is of infinite scale to the N-Array ∑ of both an infinite increment ∆ and of the infinite Nabla or decrement ∇ in both scales proportionally ∝. At least, that's the trail of thought. I used the negation as equal to, because the computation consumes that computational energy. As such consider the negation ≋ as equality ≡ for future calculations in these writings.

Code

```
float A;
float PI;
float U;
float RAD = 0.1;

float main(float argc, char **argv){
    #define A 0.1;
    #define PI 3.1415926535;
    #define U exp(pow(pow((PI * (RAD)), 3), ((exp(A++)) / (exp(--A))));
    //return 0.0; //not sure if this will work might do given the main method is float.
    return 1;
}
```

Thyme

What I have attempted to achieve here is to find the 5th dimension which would or could be the velocity of time itself.

The thought process which sparked this quest has been bugging me for years, I mean literally years, is time a concept, a measurement of change or an actual calculation which can be determined. Then a few weeks ago I considered PI.

I began to think, what if the numerical values of PI should be interpreted as 3.1.4.1.5.9.2.6......and so on. What if PI is a process intelligent lifeforms have to take to reach a certain point at a given time.

That being said, 1 would be calculate, 2 two dimensions, 3 three dimensions, 4 time, 5 velocity of time, 6 and so on ...... and full stop, is time to stop and think.

The code below is pure theorem, my math is not what it could or should be and the code seems to make sense.

I have been using the html notations from toptal and w3Schools they are very helpful, thank you.

Constant and Partial Derivative

I have defined a constant ƒ(x) as

$ƒ(x)^n ≡ σ((π·r^3)^{n∞})∞$

The Partial derivative or differential ∂ of the constant $f(x)$ would be time to leave (point fA), go somewhere (velocity V over time T), and come back (point fB) which is actually A minus B over T^3 cubed (third or physical dimension) at rate V, since we always have to be in the present. Returning -

$\partial f(x) \approx f(fA - fB) / T^3)\infty \because$

$\partial f(x) \approx ((fA \% fB) / \forall T)\infty$

$\therefore V \approx \wedge \vee \propto \partial f(x)\infty$

Past Time

Past time PT is equal to $\approx$ the partial ∂ of the constant $f(x)$ as a remainder % of future time FT over all time $\forall T$ exponentially ∞. Because the remainder of the future time over all time as a constant gives the past, constantly.

$\therefore PT \approx ((\partial f(x) \% FT) / \forall T)\infty$

Future Time

Future time FT is all time $\forall T$ divided over the partial ∂ of the constant $f(x)$ minus past time PT. Because we have to divide all time $\forall T$ over the velocity V and subtract the past to gain the future value as a continuum.

$FT \approx ((\forall T / \partial f(x)) - PT)\infty$

Time

Time itself $\forall T\infty$ is a self reinforcing recurrent function as a composite of past time PT, future time FT divided over the partial derivative of the constant (Velocity) $\partial f(x)$ which must then be divided over itself as a summation of the N-Array $\sum$ of $\forall T\infty$ exponentially in all dimensions.

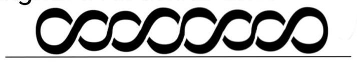
$T \approx (((PT + FT) / \partial f(x)) / \sum \forall T\infty))\infty$

....where x in the aforementioned scenario is....

$\sigma((\pi \cdot r^3)^{n\infty})\infty$

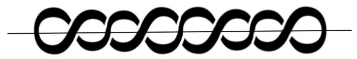
Linear time

Signal over linear time



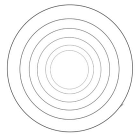
Self reinforced time

Self reinforced time over
linear time



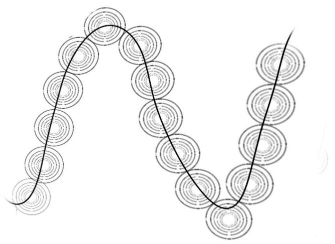
Spherical time is just
linear time omnipresent

Consider spherical time
as a constant



The input to the DNN

Consider it now as a
signal, travelling over
a sine wave



In addition, the partial
as velocity and at which
time in an array matrix
VEL

$$\begin{array}{ll} T1 & \partial T \\ T2 & \partial T \\ T3 & \partial T \\ Tx & \partial T \end{array}$$

Why put them in a matrix?
As time passes, the velocity
would change when approaching
gravity for example or EMP

Why would the time velocity change?
It wouldn't necessarily. However, in these examples,
an EMP is used and EMP is prone to change when
encountering other EMP.

Code

```
float A;
float T;
float C;
float PT;
float FT;
float V;
float PI;
float DIM = 1;
float SIG(C);

//An afterthought
//float DIV;
//float PAR;
//float NTH;
float VEL;
float IN;
float RAD = 1;

//Time goes in recurrently
#define IN T;

//Sigmoid standard is (1 / (1 + E(--z))) where E is the epsilon value or 'frequency'.
#define SIG ((1) / (-1(C--)));

#define PI 3.1415926535;

//Oliptical Dimension
#define A (pow((PI * RAD), DIM));

//Partial derivative
#define V (pow(T, 3) / (PT - FT));

//Partial derivative time matrix
#define VEL [T, V];

//Past Time
#define PT (T / (FT % C));

//Constant
#define C (exp(pow(A), ((exp(IN++)) / (exp(--IN))))); //Rate of change or constant from i9.c

//Future Time
#define FT (PT - (C / (T)));

/* Recursive function gives us the time and constant as a
 * vector because pT and fT are factored in. T is present,
 * C is constant rate of change (defined) thus, the present. */

#define T (exp((V / (PT + FT)) / (exp(pow(IN, DIM), (V)))));

float main (float argc, char **argv) {
    return 1;
}
```

LandRover

Simply a play on words to describe a simple logic for engine on engine off processes in an ICU (could be any ICU) without importing navigation headers and without exporting data to a server.

Just a quick 15 minute write up after a Bluetooth name flagged up on my phone when I was messing with it. I thought, hmmmm, I could write some code for that.

Code

```
float L;
float A;
float N;
float D = 3;
float R = 1;
float O;
float V;
float E;
float PI;
unsigned char report;

//location lon lat distance
float L1;
float L2;
float L3;

float main() {
    int engine, off = 0, on = 1;

    report = L++, L1++, L2++, L3++;

    engine = (0 != 1) ? 0 : 1;

    //lon and lat header files required here
    L = 0.000000; //distance set to zero
    L1 = 0.000000; //lon
    L2 = 0.000000; //lat
    #define L3 = (-L1, -L2); //Get an invert for accuracy.
    #define A (pow((PI * R), D));
    #define N (pow(L, A)); //Distance in the Nth Dimension;
    #define O (/* id comprised of VIN, MAC, ICU, REG? */);
    #define V (L / E);
    #define E diff(time_t go, time_t stop);

    report;

    if(1){
        time_t go;
```

```
            report;
        }else if(0){
            time_t stop;
            report;
        }

        return 0;
    }
```

RTL Project Merge

This project is a merger from the 'Thyme' and 'QBit and GParticulates' repositories. It combines the back propagation reinforcement learning algorithms in i9.c and the probability calculations which make use of decision tables and the Markov Decision Process with the theorem of time.

Apologies in advance for the hybrid of Math with functional programming in the notation. The purpose is to try to get an accurate probability over time by calculations and learning constantly from the partial derivative VEL which is a Matrix of time T and differential V.

It is still a work in progress since it is theorem. Without further adieu, in order of build;

Decision Tables, Markov Decision Process and QuBit

The first port of call is to pay tribute once again to decisions tables, Markov Decision Process (MDP) and AI, I learnt about them at University.

Decision Tables and probability

Factoring in probability into a decision table was a concept considered when thinking about QuBit logic.

Using basic knowledge of the Markov Decision Process, I think I may have implemented it correctly within the decision table. I do not recommend Wikipedia for academic work. However, as a quick reference or refresher, it can be useful for a brief overview.

I defined Y as equivalent to a boolean value of $1 \overline{\vee}$ (one or the other) 0. The way I think a QuBit works logically, would be to calculate the probability of either $1 \overline{\vee} 0$ over all input (input prime or J'), squared ²; Since the decision table always has two possible decisions. Adding the logical boolean Y of $1 \overline{\vee} 0$ brings consideration of the input for all ∀ values being either on or off (1 or 0) at any one given moment as part of the function.

This brings us to MDP. Even though QuBits consider both outputs simultaneously such that $P(A \cap B)$ and $P(A \cup B)$ are both simultaneously parallel ∥, a similar process can be emulated using increments of the input as a data stream. As such, I ended up with the function -

$Y \approx 1 \overline{\vee} 0$

$J \approx \forall Y\infty$

$P \mid J' \approx MDP$

$Q \approx ((((P \mid J')^{n\infty})^2)+Y)\infty$

Weights and Biases

When defining the weights and biases, they are defined as random in each instance or iteration for several reasons; consideration of probability systems having a bias in which case the bias could be random to provide the best probable outcome based on what information the perceptron in the neural network is receiving as a constant.

A strong bias in one favour or the other could cause incorrect output unless in the instance of law, health (including environment) and possibly economics where the bias should be in favour of coexistence and symbiotic relationship.

However for Math, deterministic probable outcome would be best achieved using a constant random bias and weight system, which explains why the bias and weights are initialised randomly and continues to be random in the implementation, they do have continual updates with the aim to provide a better outcome. They are generated randomly each time per iteration for each perceptron. Potentially, that would provide a more accurate outcome rather than a biased one.

The other reason, to distribute the weights as a mean average deviation (Find the median, calculate the lowest average and the highest average) accordingly, across all perceptrons as per discussion in Neural Networks and Deep Learning Article.

The weights and biases are distributed as random float values using modf() and rand() function in C, the minimum and maximum values are set to 0.0 and 1.0 where the returned values are randomly generated and are greater than or equal to ≥ 0.0 and less than or equal to ≤ 1.0, discarding the integer values into memory location across all perceptrons. The weights and bias are defined respectively as -

$W \approx \varphi(x) \forall W^{n}\infty$

$B \approx \varphi(x) \forall B^{n}\infty$

MDP

Redefining the MDP to suit this project was quite a challenge, as already mentioned, a suitable input method was required for the probability and reward functions, the bias, weights (as part of the DNN) and collate them into the MDP ready for the decision tables.

Rewards

Defining the reward had to be done using the logical operator Y and factor in any input (more on that later) which might be entering the DNN and MDP. The reward was defined as $1 \overline{\vee} 0$ divided / over the vector matrix $INPUT(\forall T, \sigma O, O)$ of

Time

$\forall T \approx (((PT + FT) / \partial f(x)) / \sum \forall T\infty))\infty$

Sine Output

$\sigma O \approx ((W * \forall T) + B)\infty$

and ∀Output as a full back propagation from the DNN, across all iterations $\forall N \approx (\forall T, \sigma O, \forall O)$ exponentially ∞; As such the reward value could be defined as -

$R \approx ((1 \overline{\vee} 0) / ((\forall T, \sigma O, \forall O), (\forall T, \sigma O, \forall O)\infty))\infty$

or simply

$R \approx (Y / (\partial INPUT, \forall INPUT))\infty$

...more on the outputs later.

Probability

The probability P is measured from all inputs $\forall INPUT$ divided over the vector matrices of partial input $\partial INPUT$ and logical boolean Y $1 \overline{\vee} 0$ exponentially. As such the probability P could be noted as -

$P \approx ((\forall N \approx (\forall T, \sigma O, \forall O)\infty)\infty / (\partial N \approx (T, O, \partial O), 1 \overline{\vee} 0))\infty$

or simply

$P≋(\forall INPUT / (\partial INPUT, Y))^{\infty}$

MDP

The probability of Y occurring over T is divided over the reward R and bias B continuously to feed forward the probable outcome and back propagate into the network at a median more logical level. The resulting function of the MDP is

$MDP≋(((\forall N≋(\forall T, \sigma O, \forall O)^{\infty})^{\infty} / (\partial N≋(T, O, \partial O), 1 ⊽ 0))^{\infty} / (((1 ⊽ 0) / ((\forall T, \sigma O, \forall O), (\forall T, \sigma O, \forall O)^{\infty}))^{\infty} + \varphi(x)\forall B^{n\infty}))$

or simply

$(P≋(\forall INPUT / (\partial INPUT, Y))^{\infty} / (R≋(Y / (\partial INPUT, \forall INPUT))^{\infty} + B≋\varphi(x)\forall B^{n\infty}))$

The logic of Q (discussed earlier on in Decision tables and probability) is then questioned again by implementing the same MDP in constant 'R' and in it's own probability decision making, where constant 'R' is as MDP defined as (Q / (REWARD + BIAS)) where Q is the input to be calculated and assessed (please see code).

The most likely behaviour is the AI decides 1 is more probable and preferred over 0 since 0 would be off preventing it from doing what it does, think and solve problems. However, the solution is not a panacea, since other factors are involved, true or false for example, thus the output would need to be interpreted correctly.

Inputs and Outputs

The velocity of some event occurring at some time could be measured over an exponential vector matrix VEL containing time T as a partial differential ∂T and velocity as a partial differential ∂V. Originally this was included as part of the input into the DNN. However, it was later removed since VEL is a calculation as described in the Thyme project repository images, to get the current velocity relevant to that particular time. Velocity is however included in the T constant input to the DNN.

Output

Quantified DNN Output is included in the input to bring the full output of the DNN back into the DNN INPUT to facilitate a full back propagation. Since the hidden layers are divided over the quantity of OUTPUT neurons, the values become less intensive for the DNN to compute, we don't want to overload the DNN.

In addition to the overloading, having four output perceptrons, permits a more logical computation for example a nibble, compared to a potential stack overload value. Velocity V is the action in the state space time T and the event of Y occurring in multiple states over T with the reward values and biases being accounted for.

Input

Full outputs, the activated output OUT and time are being distributed back through the network for reassessment with back propagation comparable to the initial input. However, that is where things begin to get complicated because INPUT is not simply an input, we have to digress slightly and take another look at hat we are feeding the network.

Essentially we are feeding everything included in the time T constant including the velocity and the Continuum as the present, The BIAS (OUT) acting on all the weights for all the outputs and the quantified output from the DNN's output layer OUTPUT.

Code

```
/**
 * Learn from the probability of time and question the probability.
 * An expansion from merging Thyme project with QBits and Particulates
 * @Created and finished between 04.09.2020 and 05.09.2020
 * @Listening Peace Orchestra.
 * @By Russell A E Clarke Et. al
 */

#include
#include
#include
#include
#include

float A;
float T;
float C;
float PT;
float FT;
float V;
float PI;
float DIM; //could be 3rd dimension for example. Set the value here or in the method, Method preferred.
float SIG;
float VEL;
float IN;
float RAD = 1;

float WEIGHT;
float weights;
float EPSILON;
float SIGMOID;
float COST;
float BIAS;
float REWARD;
float PROBABILITY;
float MDP;
float Q;
float R;
float INPUT;
float HIDDEN;
float OUTPUT;

float main () {

    //The basics, learning rate first
    #define EPSILON 5E-5;
    #define Y (? 0 : 1);
    #define PI 3.1415926535;
    #define A (pow((PI * RAD), DIM));
    #define SIG ((1) / (-1(C--)));
    #define SIGMOID (1.0 / (-1 + (exp(O)--))));

    //Preparing for NN and Biases
    #define WEIGHT (modf(((randn() % 1.0) + 0.0), float *wdiscard));
    #define BIAS (modf(((randn() % 1.0) + 0.0), float *bdiscard));

    //Weigh up the costs and gradient of descent
    #define COST (weights, BIAS);
    #define GRAD_DESCENT (pow(((COST) - (COST(EPSILON))), 2));

    //Define constant first
    #define C SIG(exp(pow(A), ((exp(IN++)) / (exp(--IN))))); //predecrement all time as past time divide over increment of all time for continuum. Had
    //#define C SIG(exp(pow(A), ((exp(++PT)) / (exp(FT++))))); //preincrement the past and divide over the future
    //Begin telling computer about the past and future and all time
    #define PT ((FT % C) / T);
    #define FT (((T) / C) - PT);
    #define T (exp((V / (PT + FT)) / (exp(pow(IN, DIM), (V)))));

    //Tell it what to calculate and what is going into the constant
    #define V (pow(T, 3) / (PT - FT));
    #define VEL [T, V];
    #define IN T;

    //Markov Decision Process
    #define REWARD ((INPUT, N) / Y);
    #define PROBABILITY ((INPUT, Y) / N);
    #define MDP ((REWARD + BIAS) / PROBABILITY);

    //Implementation
    #define N INPUT++;
    #define OUT SIGMOID((weights * N) + BIAS);

    //Calculate the probability
    #define Q pow(pow(MDP, N), 2) + Y;
    #define R ((REWARD + BIAS) / Q);

    //Initialise the input, output and hidden layers of the network
    #define INPUT ((T, OUT, OUTPUT) * WEIGHT) + BIAS;
    #define HIDDEN (((10 * pow(20, 4)) * WEIGHT * (INPUT));
    #define OUTPUT ((4 * WEIGHT) / (HIDDEN)) + BIAS;

    return 1;
}
```