Bounded Round Analytic Intelligent Neural Network (BRAINN)

Spatially Bounded Fourier Analysed Polymathic Spiking Neural Network

Maximilian Powers
International School of Nice

(Research Paper)

(Dated: October 14, 2018)

Contents

I. Introduction

a. History In 1943 McCulloch and Pitts formulated the first computational model for learning, which eventually branched off into two areas, one focussed on the biological processes's within the brain, the other on the applications of machine learning. The former gained strong momentum when in 1958 the Perceptron was introduced by Rosenblatt, and then in 1980 when the Backpropagation algorithm was developed. Since then the field of machine learning has been relatively stagnant, until recently when processing power caught up and advancements in Information Theory, Data and Signal Analysis, and solving NP-Problems were made, that machine learning has been popularised through image recognition, language translation, and Netflix suggestions. As previously mentioned, machine learning was primarily focussed with understanding the brain but these more realistic time based neural networks have not seen the advancements that regular neural networks have seen, mainly due to the complication of large scale parrallel processing which is being tackled currently by companies such as Linux, Beowulf clusters, and with the rise of multi-core processors found in most computers to reduce heat however the problem of information retrieval, something done very efficiently in the brain, still has not been solved.

II. Spiking Neural Networks.

In the introduction and the appendices, Section A, the importance of the Artificial Neural Network is emphasises. However, in this paper, Spiking Neural Networks (SNNs), which are part of the third wave of neural networks will be looked at, and eventually modified.

Definition 1. Third Wave Neural Networks. These networks are at the frontier of artificial intelligence and generally deal with a more biologically accurate picture of a brain as opposed to a cost minisation problem, as used by ANNs in the Backpropagation algorithm, A iii.

i. The neurones

To understand a Spiking Neural Network (SNN), the biological representation of that needs to be understood. The dendrites are the neurones connecting to that neurone, whereas the axon is that neurone's own pathway. The soma is the area where the membrane potential is stored, a small shield that prevents any small electrical current from activating the neurone. Pre-synaptic pulses flow through the dendrites, whereas the post-synaptic pulses flow out of the neurone through the axon going into other neurones. It is the time at which these events occur that determines whether or not the strength of connectivity between two neurones increase, or decreases.

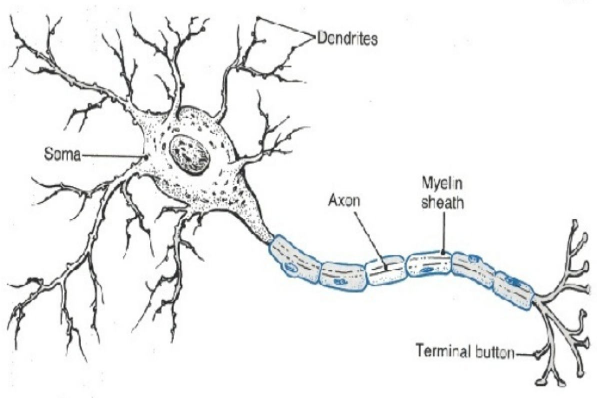


Figure 1. A Diagram of a neurone

ii. Temporal and Spatial Architecture.

Spiking Neural Networks use temporal data, information in the time domain, to learn as opposed to ANNs which used continuous, iterative weights. SNNs are also local in their learning process, as in the influence of a neurone depends on how close they are to that neurone, as well as being able to change its own position. Finally, these neural networks use energy thresholds, as in a certain neurone must break the energy threshold, otherwise known as the membrane potential, in order to fire a synapse, which is a transmission of energy between two neurones.

iii. Leaky Integrate and Fire Models.

In order to determine when this membrane potential is passed, and when the neurone fires a pulse, an elementary understanding of the theory of electricity needs to be understood. Take the variable u_i denoting the membrane potential of neurone i and then $I(t)$ representing the current passing through a neurone from an input, such as an image, or the electrical signal of a presynaptic neurone, if $I(t) = 0$ then the resting membrane potential is u_{rest}. Using elementary theories of electromagnetism a relationship between $I(t)$ and $u_i - u_{\text{rest}}$ can be postulated. The cel membrane acts as a capacitor of capacity C that can hold a charge for a certain period of time, this electrical charge is represented as $q = \int I(t')\mathrm{d}t'$ where t' is the time of the pulse. Within any imperfect insulator there is resistance, in this case it will allow this current to slowly pass through the membrane potential, and this resistance will be taken as R. A linear circuit can be constructed from this and resembles the following,

As seen in Figure 2 the set up of a neurone revolves around a resistor and a capacitor in parallel. As the input is slowly released out of the resistor due to inefficiency, in this case biology has taken advantage of imperfections in a resistor, the current $I(t)$ is gradually leaked into

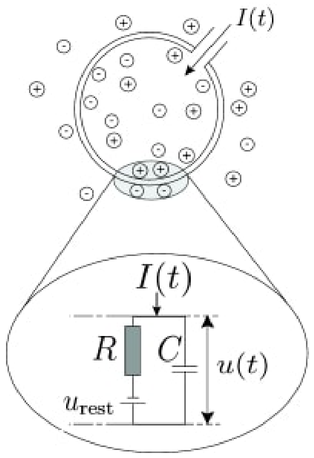


Figure 2. Resistor and Capacitor in parallel modelling a neurone with an input current of $I(t)$

the circuit causing the membrane potential to jump from $u_{\text{rest}} \to u(t)$ which resets the system so a new current can flow through the neurone, this process takes around 10ms which is long compared to the 1ms time of a spike. Then using the law of current conservation the current can be split into two components

$$I(t) = I_R + I_C \tag{1}$$

Then using Ohm's law for resistance $I_R = u_r/R$ where $u_r = u - u_{\text{rest}}$ and is the voltage running through the the resistor. The second component I_C can be broken down using the definition of the capacity as $C = q/u$ where q is the charge and u is the voltage, and therefore the capacitive current $I_C = dq/\text{d}t = Cdu/\text{d}t$ hence arises the final formula

$$I(t) = \frac{u(t) - u_{\text{rest}}}{R} + C\frac{du}{dt} \tag{2}$$

Then using some basic algebraic manipulation, and setting $\tau_m = RC$, the leaky integrator, the the following equation can be used

$$RI(t) = u(t) - u_{\text{rest}} + RC\frac{du}{\text{d}t} \tag{3}$$

$$\tau_m \frac{du}{\text{d}t} = RI(t) - [u(t) - u_{\text{rest}}] \tag{4}$$

The membrane potential at a time t is therefore u, and is referred to as the τ_m is the membrane time constant. This is an equation modelling a passive membranes, and is an ordinary differential equation (ODE).
While there are many different solutions for equation 4 only the general solution will be considered. This solution assumes there is a varying current, $I(t)$ passing through the post-synaptic neurone, $u(0) = u_{\text{rest}}$, and t must start at 0.

$$u(t) = u_{\text{rest}} + \exp\left(-\frac{t - t_0}{\tau_m}\right) + \frac{R}{\tau_m}\int_{t_0}^{t} \exp\left(-\frac{t - t'}{\tau_m}\right) I(t')\text{d}t \tag{5}$$

Then using $\tau_m = RC$.

$$u(t) = u_{\text{rest}} + \exp\left(-\frac{t - t_0}{\tau_m}\right) + \frac{1}{C}\int_{t_0}^{t} \exp\left(-\frac{t - t'}{\tau_m}\right) I(t')\text{d}t \tag{6}$$

This is a modelling equation, as in it is not derived from equation 4 so it can be broken down. The first term acts as the baseline so that all the voltages are added on top of that, typically this is $-70mV$. The second term decrease the value as soon as spike t_0, the first spike, has increased the voltage in accordance to the membrane time constant τ_m. The integral sums up all the previously added spikes that are undergoing exponential decay, and then adds the current of spike t'. As the membrane potential increases with additional pulses it will reach it's maximum, or it's energy threshold, and then Equation 5 is instantly resets to $u(t) = u_{\text{rest}}$ and $I(t) = 0$. In Section V iv where a distance based modification is introduced. As a side note, the membrane potential also decreases computational inefficiencies, as not every neurone will fire for each iteration of the network.

iv. Theoretical Learning.

While it may seem counterintuitive to use discrete temporal data, there is a lot more information in the time domain then there is the amplitude domain. However since this data is in the time domain, it a lot harder to both understand and process as patterns are extremely hard to identify through the noise of the signals. There are 4 types of learning that are used by neurones: Pulse Pair Facilitation (PPF), Vesicle Depletion, Long-term Potentiation (LTP), & Long-term Depression (LTD).

1. Short Term Weight Change.

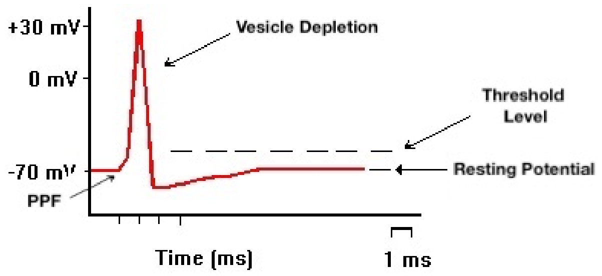


Figure 3. A neurone undergoing excitation

Both PPF and Vesicle Depletion happen in the short term, affecting the strength between two neurones and lasting around 10-100ms. If the presynaptic neurones fire before the postsynaptic neurone, then there is an overall increase in strength. This process repeats itself and if this were to happen many times, it would cap out at some maximum strength. To prevent this strength from increasing exponentially, Vesicle Depletion reduces the voltage felt within the neurone almost immediately after it has peaked so that it returns to its normal value.

v. Rate-Based Hebbian Learning.

In this section, the learning algorithm for SNNs will be developed and explained.

1. *Formalising the neurone.*

$$\nu_i = g(u_i) \tag{7}$$

Where ν_i is the rate of firing, which is related to its membrane potential u_i and by a monotonously increasing function $g(u_i)$, which could be the sigmoid function, $g(u_i) = \dfrac{1}{1+e^{-u_i}}$, where $\forall g'(u_i) > 0$ and $g(u_i)$ lies between 0 and 1. u_i will be taken as the value shown in Equation 6.

The change in a weight of two neurones i and j can be taken as Δw_{ij} between the interation of the algorithm n and $n+1$. The importance however lies in the timing of when this new trial, $n+1$ occurs, hence the rate of change of w_{ij} will be used such that.

$$\frac{dw_{ij}}{\mathrm{d}t} = \frac{\Delta w_{ij}}{\mathcal{T}} \tag{8}$$

Where $\mathcal{T}$ is the overall time for a trial to run. Following Hebb's rule, if both the post- and pre-synaptic neurone are active together, then there should be an overall increase in weight, which will be formulated now in terms of all of its effects, which are Locality, Cooperativity, Synaptic Depression, Boundedness, Competition, and Longterm Stability. Firstly,

2. *Locality*

Locality is the spatial connectivity between two neurones which is dependant on distance, the closer two neurones are to each other the easier it will be to interact leading to more of these interactions. Hence the only spatially relevant parameters are the weights, and both the pre- and post-synaptic firing rates. Our locality function therefore can be written as

$$\frac{dw_{ij}}{\mathrm{d}t} = F(w_{ij}, \nu_i, \nu_j) \tag{9}$$

Where F is an arbitrary function of those parameters.

3. *Cooperativity*

Following Hebb's rule, a pre- and post-synaptic neurone must be active to induce a change in weights and hence there is a causal relationship. These can be classified by the table above.

While positive weight change is not only induced by simultaneous activity, it will be discussed later on. Going back to the rate equation we can expand the function using a Taylor series at $\nu_i = \nu_j = 0$. An expansion to the second order gives

$$\frac{d}{\mathrm{d}t} w_{ij} \approx c_2^{corr}(w_{ij})\nu_i\nu_j/2 + c_2^{post}(w_{ij})\nu_i^2/2 + c_2^{pre}(w_{ij})\nu_j^2/2 + c_1^{pre}(w_{ij})\nu_j + c_1^{post}(w_{ij})\nu_i + c_0(w_{ij}) \tag{10}$$

Going term by term, c^{corr} is the chose coefficient for the dependent your model is on correlation between the pre- and post-synaptic neurone, c^{post} is the importance of the post-synaptic neurone on the model and c^{pre} the importance of the pre-synaptic neurone. Strong c^{pre} are generally feedforward networks, whereas a balance of both are generally recurrent networks. The simplest implementation of Hebbian plasticity would require $c^{corr} > 0$ and all other coefficients to be set to 0, hence

$$\frac{d}{\mathrm{d}t} w_{ij} = c^{corr}(w_{ij})\nu_i\nu_j \tag{11}$$

This is the basis for hebbian learning, and if $c^{corr} < 0$, this would be referred to as anti-Hebbian. If we continue our Taylor expansion we can create more and more intricate learning rules, for example ν^3 will have $\nu_i\nu_j^2$, $\nu_i^2\nu_j$ etc.

4. *Synaptic Depression*

Hebb's original idea did not comprise of a decreasing parameter, however it is clear that these networks require these depressions. The use of a weight decay, through the c_0 parameter for example would result in

$$c_0(w_{ij}) = -\gamma_0 w_{ij} \tag{12}$$

While $\gamma_0 > 0$. Synaptic depression could also be implement by combinations involving c_1^{post}, c_2^{post}, c_1^{pre} etc. For example following the previously described rule of $\frac{d}{\mathrm{d}t} w_{ij} = (\nu_i - \nu_\theta)\nu_j$ could use $c_2^{corr} = 1$, $c_1^{pre} = -\nu_\theta < 0$, and all other coefficients set to zero. This is called a presynaptically gated network, as it is dependent on presynaptic activity to develop and change and the postsynaptic activity determines solely the direction of the change, which can seen on the Table.

post i	pre j	$\frac{d}{dt}w_{ij} \propto \nu_i\nu_j$	$\frac{d}{dt}w_{ij} \propto \nu_i\nu_j - c_0$	$\frac{d}{dt}w_{ij} \propto (\nu_i - \nu_\theta)\nu_j$	$\frac{d}{dt}w_{ij} \propto \nu_i(\nu_j - \nu_\theta)$	$\frac{d}{dt}w_{ij} \propto (\nu_i - n\bar{u}_i)(\nu_j - \bar{\nu_j}.)$
ON	ON	+	+	+	+	+
ON	OFF	0	-	0	-	-
OFF	ON	0	-	-	0	-
OFF	OFF	0	-	0	0	+

Table I. ON indicates that the neurone is firing at it's maximum rate ($\nu = \nu_{max}$) whereas OFF means the neurone is inactive ($\nu = 0$). The first table is the standard Hebb rule, then Hebb with decay. then pre- and post-synaptic gating, and lastly the covariance rule. The $\bar{\nu_i}$ notation refers to the mean firing rate. The parameter c_0 lies in between 0 and v^2_{max} hence $0 < c_0 < \nu^2_{max}$. and ν_θ is an arbitrary firing rate between 0 and ν_{max} hence $0 < \nu_\theta < \nu_{max}$. From this table we can observe that using certain models will give us different results. Such as the covariance model which says that if two neurones are not firing together, they gain in weight, and they can only loose weighting if one is on and the other is off compared to the Hebb with decay model which says that when only a single neurone is firing, there is no change and it is only when both neurones are not firing that they loose weight.

5. *Boundedness*

Within a network of spiking neurones there needs to be limitations to there excitations and depressions and hence must be bound to a range such that $0 \leq w_{ij} \leq w^{max}$ where w^{max} is the maximal weight value such that the network's bio-mechanism's are stable. In order to achieve this we use the coefficients of the Taylor expansion as they can be dependent on w_{ij}, and in sure that $0 \leq w_{ij} \leq w^{max}$. By saturating the weight dependence, we take the coefficient c_2^{corr} as

$$c_2^{corr}(w_ij) = \eta_0(w^{max} - w_{ij}) \tag{13}$$

Where the constant $\eta_0 > 0$. The factor $(w^{max} - w_{ij})$ would mean as w_{ij} approached, w_{ij} changes less and less.

III. Fourier Series.

Fourier Analysis is the most important branch of harmonic analysis. Fourier Series is the basis on which several transforms are built upon, such as Fourier Transforms (FT), Discrete Time Fourier Transform(dtFT), Short Time Fourier Transforms (STFT), Fast Fourier Transform (FFT) ... etc and is important with other subjects such as Hilbert and Laplace Transforms. In regards to Spiking Neural Networks, there is a clear link between Signal Analysis and Learning Patterns from Spike Trains. If an SNN were to have a Fourier Transform based learning algorithm, particularily in regards to the spatial plasticity of a neurone as Fourier Transforms are computationally heavy (FFT has a complexity of $O(n \log n)$). This new algorithm will be presented in the last chapter of the paper. Before covering the issue of Fourier Series and Transform we must first understand the addition of periodic signals, as well as the orthog1onality of sinusoidal functions.

i. Addition of Periodic Signals

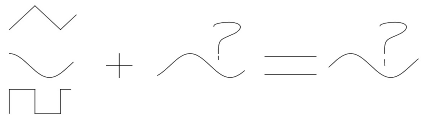


Figure 4. Addition of periodic signals.

In other words:

$$f_1\text{Hz} + f_2\text{Hz} = f_3\text{Hz}, [f_1,\ f_2]|f_3 \tag{14}$$

$$\frac{f_1}{f_2} \in R^+ \vee I^+ \tag{15}$$

Where $\vee$ is an or operator, and $[a, b]|c$ shows that c is the lowest common multiple of $[a, b]$. The periodicity of the new signal will be determined by 1 over the highest lowest common factor of the two. e.g.

$$1\text{Hz} + 0.8\text{Hz} = \frac{1}{5} = 0.2\text{Hz} \tag{16}$$

You can also notice that:

$$1\text{Hz} + 100\text{Hz} = 1\text{Hz} \tag{17}$$

$$0.2\text{Hz} + 1\text{Hz} = 0.2\text{Hz} \tag{18}$$

$$2\text{Hz} + (2^{32})\text{Hz} = 2\text{Hz} \tag{19}$$

From this we can conclude that if you were to add any signal with an integer frequency to a signal of 1Hz, the new signal would have a frequency of 1Hz.

$$1\text{Hz} + \lim_{n\to\infty} n\text{Hz} = 1\text{Hz} \tag{20}$$

Such that $n \in I^+$. All we must do to find these new signals is find their magnitudes, so now we know that any signal, can be represented by an infinite number of other signals, which is the basis of which Fourier series is built upon. Also, the added signal can never not be a multiple of the original signal without corrupting it.

Take the square wave function, $f(t)$ and give it a period of 1s. This function could also be represented as:

$$f(t) = \sum_{n=0}^{\infty} \sin(2\pi nt) \tag{21}$$

When $n = 0$ there is no function, when $n = 1$, there is 1Hz, $n = 2$, there is 2Hz etc. n can only increment 1's, or the period, or else it will corrupt the signal. Finally we can denote the function b_n, as the representation of each frequency produce ($b_1 = 1, b_2 = 2$ etc.)

ii. Orthogonality of Sinusoids.

Definition 2. Orthogonal. This term comes from vector spaces wherein if two vectors are perpendicular, hence their dot product is 0, they are considered to be orthogonal. However for function this means that when 2 different function are multiplied together as an inner product, which is defined as

$$\langle a_n, b_n \rangle = \sum_{i=1}^{n} a_i \cdot b_i \tag{22}$$

And when they are then passed under a definite integral they are equal to 0, but when computed with itself, are equal to some value other than 0. Explaining more simply, imagine a set of particles and another set of antiparticles, now when combined they annihilate leaving nothing, but when mixed with each other they're is mass. Now more formally take the two function ϕ_m, ϕ_n and they are orthogonal under some interval $[a, b]$. Therefore,

$$\begin{aligned}\langle \phi_n, \phi_m \rangle &= \int_a^b \phi_m \cdot \phi_n \mathrm{d}x \\ &\to 0 \iff n \neq m \\ &\to ||\phi_n^2|| \iff n = m\end{aligned}$$

Where $||..||$ is the square norm of a vector, which is just an absolute value but for vectors so that they're positive in all directions. Now take another function $f(x)$, such that

$$f(x) = \sum_{n=0}^{\infty} C_n \phi_n$$

$$\langle f(x), \phi_m \rangle = \sum_{n=0}^{\infty} C_n \langle \phi_n, \phi_m \rangle$$

Which can be simplified using the aforementioned property,

$$\langle \phi_n, \phi_m \rangle = 0 \iff n \neq m$$

So now we can compute to find C_m which is our coefficient,

$$\begin{aligned}\langle f, \phi_m \rangle &= C_m \langle \phi_m, \phi_m \rangle \\ C_m &= \frac{\langle f, \phi_m \rangle}{\langle \phi_m, \phi_m \rangle} \\ &= \frac{\langle f, \phi_m \rangle}{||\phi_n^2||}\end{aligned}$$

This principle is then used with sinusoidal functions throughout this chapter, and it is this key property that underlies all of Fourier Series and Transforms.

Sinusoids are orthogonal at certain frequencies regardless of amplitude, real or imaginary, and phase so long as the frequencies are different. So if you take the function,

$$f(t) = \sin(2\pi t)\sin(2\pi 2t) \tag{23}$$

Then take the integral of $f(t)$ from 0 to 1,

$$\int_0^1 f(t)\mathrm{d}t = \int_0^1 \sin(2\pi t)\sin(2\pi 2t)\mathrm{d}t \tag{24}$$

Would be equal to 0, and can be visualised by:

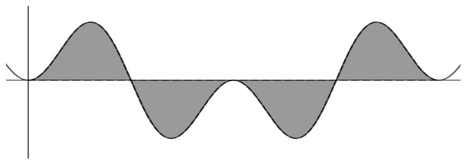


Figure 5. Area beneath the function $f(t) = \sin(2\pi t)\sin(2\pi 2t)$.

The same can be done with 1Hz + 3Hz.

$$f(t) = \sin(2\pi t)\sin(2\pi 3t) \tag{25}$$

$$\int_0^1 f(t)\mathrm{d}t = \int_0^1 \sin(2\pi t)\sin(2\pi 3t)\mathrm{d}t \tag{26}$$

This can be generalised to:

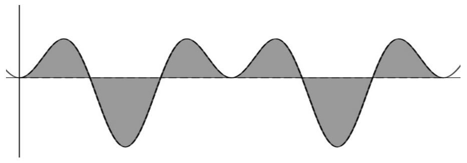


Figure 6. Area beneath the function $f(t) = \sin(2\pi t)\sin(2\pi 3t)$.

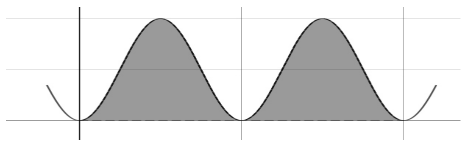


Figure 7. Area beneath the curve $f(t) = \sin(2\pi t)\sin(2\pi t)$ is not equal to 0

$$\int_0^1 \sin(2\pi t)\sin(2\pi nt)\mathrm{d}t = 0|_{n\neq 1} \tag{27}$$

As at $n = 1$ the area of integration would be equal to 0.5. You can do the same for any initial signal, so again this can be written as

$$\int_0^1 \sin(2\pi mt)\sin(2\pi nt)\mathrm{d}t|_{0\ n\neq m}^{0.5\ n=m} \tag{28}$$

So

$$f(t) = \sum_{n=1}^{\infty} b_n \sin(2\pi nt) \tag{29}$$

As at $n = 0$, $f(t) = 0$. Now you're left with a the series,

$$b_1 \sin(2\pi 1t) + b_2 \sin(2\pi 2t) + b_3 \sin(2\pi 3t) + ... \tag{30}$$

All of the above equations can be either written in terms of sin, cos.

$$\int_0^1 \cos(2\pi mt)\cos(2\pi nt)\mathrm{d}t|_{0\ n\neq m}^{0.5\ n=m} \tag{31}$$

$$\int_0^1 \cos(2\pi mt)\sin(2\pi nt)\mathrm{d}t|_{0\ n\neq m}^{0.5\ n=m} \tag{32}$$

In order to make the Fourier series simpler, use a Cosine Fourier Transform for even signals, and Sine Fourier Transform for odd signals and a mixture for those that are neither.

iii. Finding Fourier Coefficients.

In this subsection the coefficients will be found for neither odd or even signals, odd and even signals will be found in the appendices Section as are not used in the exploration. Now taking a signal such as, The Fourier

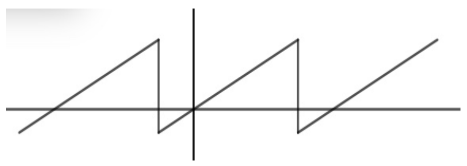


Figure 8. Signal that is neither odd or even

series for the signal in Figure **??** can be written as sine and cosine functions,

$$f(t) = \sum_{n=0}^{\infty} a_n \cos(2\pi nt) + b_n \sin(2\pi nt) \tag{33}$$

Each of the coefficients, a_n and b_n in Equation 33 can be written as integrals following the property of orthogonality, which gives the following 3 formulas wherein

$$a_n = \int_0^1 f(t)\cos(2\pi nt)\mathrm{d}t \tag{34}$$

$$b_n = \int_0^1 f(t)\sin(2\pi nt)\mathrm{d}t \tag{35}$$

$$a_0 = \int_0^1 f(t)\mathrm{d}t \tag{36}$$

iv. Final Fourier Series Formula.

In the aforementioned equations, the period of the signal was between 0 and 1s, this can further be generalised to a variable period T,

$$f(t) = \sum_{n=0}^{\infty} a_n \cos(2\pi f_0 t) + b_n \sin(2\pi f_0 t) \tag{37}$$

$$a_n = \frac{1}{T}\int_T f(t)\cos(2\pi nt)\mathrm{d}t \tag{38}$$

$$b_n = \frac{1}{T}\int_T f(t)\sin(2\pi nt)\mathrm{d}t \tag{39}$$

$$a_0 = \frac{1}{T}\int_T f(t)\mathrm{d}t \tag{40}$$

v. Complex Fourier Series.

In the final equations, there are 3 equations each representing a component of the overall Fourier series. Using Euler's formula, as shown in Equation 41, these aforementioned equations can be simplified into one.

$$\cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2} \text{ and } \sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i} \tag{41}$$

Hence.

$$\begin{aligned} f(t) &= a_0 + \sum_{n=1}^{\infty} \cos nt + \sum_{n=1}^{\infty} \sin nt \\ &= a_0 + \sum_{n=1}^{\infty} a_n(\frac{e^{int} + e^{-int}}{2}) + \sum_{n=1}^{\infty} b_n(\frac{e^{int} - e^{-int}}{2i}) \\ &= a_0 + \sum_{n=1}^{\infty} C_n e^{int} + \sum_{n=1}^{\infty} C_{-n} e^{-int} \end{aligned} \tag{42}$$

Where,

$$C_n = \frac{a_n - ib_n}{2} \tag{43}$$

This equation can be simplified by making $-n$ positive so the two exponents can be combined,

$$= a_0 + \sum_{n=1}^{\infty} C_n e^{int} + \sum_{n=-1}^{-\infty} C_n e^{int} \tag{44}$$

Since a_0 represents one frequency in an infinite amount it can be ignored, and the two sums can be added together to give,

$$= \sum_{n=-\infty}^{\infty} C_n e^{int} \tag{45}$$

Where each C_n is equal to,

$$\frac{1}{T}\int_T f(t)e^{-int}\mathrm{d}t \tag{46}$$

Hence the calculations have been decreased by 3 fold.

IV. The Discrete Fourier Transform.

While Fourier series are useful in solving differential equations, and global approximations of other functions, the true applications of the Fourier series lie in the Fourier Transform. A basic Fourier Transform is shown in Section C ii, but in this paper the more important Discrete Fourier Transform will be explored, not only because of it's numerous application in modern day acoustics such as Schorr's Prime Factorisation algorithm, infrared spectroscopy, Siri, and MRI scans. From understanding atmosphere's of distant exoplanets to creating personal assistant, the DFT, or it's more computationally efficient FFT, is most widely used Digital Signal processing tool, which is why it's listed in the top 100 most used algorithms. The DFT requires the signal to be discrete, as well as being finite in duration. In this section, N will be the number of discrete impulses, each of amplitude x.

i. Complex Geometric Series

Lemma 1. Take the finite series $S_n(z^n)$ where $z^n \in \mathbb{C}$. It's sum will be $\dfrac{1-z^N}{1-z}$.

Proof. Now writing this out,

$$\begin{aligned} S_n(z^n) &= \sum_{n=0}^{N-1} z^n = 1 + z + z^2 + ... + z^{N-1} \\ zS_n(z_n) &= z + z^2 + z^3 + ... + z^{N-1} + z^N \\ S_n(z^n) - z^n S_n(z_n) &= 1 - z^n \\ S_n(z^n) &= \frac{1-z^N}{1-z} \end{aligned} \tag{47}$$

□

ii. Proving Complex Sinusoidal Orthogonality.

To prove orthogonality, the integral of two or more waves with different frequencies is equal to 0.

$$\langle A_1 \sin(w_1 t), A_2 \sin(w_2 t)\rangle \tag{48}$$

Where $w_1 \neq w_2$ and $\perp$ symbolises orthogonality of two functions. Exact orthogonality only occurs when the harmonics divided by the sampling rate, N.

$$f_k = k\frac{f_s}{N} \tag{49}$$

Where $k = 0, 1, 2, 3, ..., N-1$ and $f_s = \dfrac{1}{T}$. These signals are then processed in the complex plane corresponding to f_k,

$$s_k(n) = e^{jw_k nT} \tag{50}$$

Where $w_k = f_k \cdot 2\pi = k\dfrac{2\pi}{N}f_s$.

$$W_N^k = e^{jk2\pi f_s T/N} \tag{51}$$

$$W_N^k = e^{jk2\pi/N} \tag{52}$$

$$[W_N^k]^N = e^{jk2\pi N/N} = e^{jk2\pi} = 1 \tag{53}$$

Regardless of k, the $[W_N^k]^N$ will always be equal to 1 since k can only be an integer, following Euler's formula:

$$e^{j2\pi k} = \cos(2\pi k) + j\sin(2\pi k), \tag{54}$$

Since $\sin(2\pi\mathbb{I}) \equiv 0$ & $\cos(2\pi\mathbb{I}) \equiv 1$.

iii. The Orthogonality of the DFT Sinusoids.

To show that the DFT are also completely orthogonal, take a signal $s_k(n)$.

$$s_k(n) = e^{jw_k nT} = e^{j2\pi kn/N} = [W_N^k]^N,\ n = 0, 1, 2, ..., N-1 \tag{55}$$

Now for the $k = 0,$ to the $N-1$ sinusoidal.

$$\begin{aligned} \langle s_k, s_l\rangle &= \sum_{n=0}^{N-1} \frac{s_k(n)}{s_l(n)} \\ &= \sum_{n=0}^{N-1} e^{j2\pi kn/N} e^{-j2\pi ln/N} \end{aligned} \tag{56}$$

$$\sum_{n=0}^{N-1} e^{j2\pi n(k-l)/N} = \frac{1 - e^{j2\pi n(k-l)}}{1 - e^{j2\pi n(k-l)/N}} \tag{57}$$

The last equation uses Lemma 1. Since the denominator is non-zero, and the numerator is 0, this proves.

$$\langle s_k, s_l \rangle \Leftrightarrow k \neq l \tag{58}$$

Now when $s_k = s_l$,

$$\sum_{n=0}^{N-1} e^{j2\pi \frac{n}{N}(k-k)} = \sum_{n=0}^{N-1} e^0 = N \tag{59}$$

iv. The Transform.

Given a signal $x(n) \in \mathbb{C}^n$, where $(C)^n$ is the complex plane of n dimensions, the dtF would be defined as,

$$s_k \perp x(n) = \sum_{n=0}^{N-1} \frac{x(n)}{s_k}, \; k = 0, 1, 2, ..., N-1 \tag{60}$$

This will be denoted as a function $\mathcal{F}(x_k)$.

$$\begin{aligned} \mathcal{F}(x_k) &= \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \\ &= \sum_{n=0}^{N-1} x(n)(\cos(\frac{2\pi kn}{N}) - j\sin(\frac{2\pi kn}{N})) \end{aligned} \tag{61}$$

For example take the signal,

$$x_k = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 3-j \\ -2j \\ -2+3j \end{bmatrix}, \tag{62}$$

In this case N is equal to 4. Now apply the Discrete Fourier Transform to each x_k to give

$$\mathcal{F}(x_k) = \begin{bmatrix} \mathcal{F}(x_0) \\ \mathcal{F}(x_1) \\ \mathcal{F}(x_2) \\ \mathcal{F}(x_3) \end{bmatrix} \tag{63}$$

As

$$\begin{aligned} \mathcal{F}(x_k) &= \sum_{n=0}^{4-1} x_n e^{\frac{2\pi kn}{4}} \\ &= \begin{bmatrix} \sum_{n=0}^{3} (x_k) e^{\frac{-j2\pi(0)(n)}{4}} \\ \sum_{n=0}^{3} (x_k) e^{\frac{-j2\pi(1)(n)}{4}} \\ \sum_{n=0}^{3} (x_k) e^{\frac{-j2\pi(2)(n)}{4}} \\ \sum_{n=0}^{3} (x_k) e^{\frac{-j2\pi(3)(n)}{4}} \end{bmatrix} = \begin{bmatrix} 5 \\ -3j \\ 3-4j \\ 8+7j \end{bmatrix} \end{aligned} \tag{64}$$

Hence,

$$\mathcal{F}(x_k) = \begin{bmatrix} 5 \\ -3j \\ 3-4j \\ 8+7j \end{bmatrix} \tag{65}$$

To convert this onto the complex plane use, this transform can be denoted $f(\omega_k)$ where each ω_k corresponds to its respective frequency.

$$\sqrt{a^2 + b^2}, \text{ where } a \in \mathbb{R} \;\&\; b \in \mathbb{C} \tag{66}$$

$$f(\omega_k) = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 5 \\ \sqrt{15} \end{bmatrix} \tag{67}$$

So to summarise:

$$x_k \rightarrow \mathcal{F}(x_k) \rightarrow f(\omega_k) \tag{68}$$

V. Spatially Bounded Polymathic Fourier Spiking Neural Networks (SCPFSNN).

The aim of this paper was to introduce a new form of neural networks aimed at brain analysis, as opposed to machine learning, and while Constrained Polymathic Spiking Neural Networks might prove to be useful when combining different input types, such as an noisy interlay of sound and visual sensors when trying to understand learning, which is often a mixture of mainly visual and auditory information. Hence if these networks are indeed capable of learning many tasks at once, these could be combined to reinforce learning.

i. Architecture

When modelling a brain, especially if it needs to be computationally efficient, the model needs to be easy to generate, and mainly rely on boolean-like operations. While this is not the goal for the architecture presented here, optimisation ideas have been presented in section VI. If the model for this neural network primarily revolves around spatial, and temporal information, there must be constraints to these learning methods, as discussed the membrane potential limits temporal learning, however a distance limiting framework, such as the skull for the brain, needs to be put in place for spatial learning. .

The brain functions not as one but as many parts, as opposed to neural networks wherein the information is sent uniformly. In order to create a structurally rigid architecture, the, as will be called here, lobe a_i will denote a fixed point in a predefined space that does not move, and will take as input that specific information type. If for example the network will be required to use tactile and motor processing in order to make a robot learn through what it feels, the neural network will be constructed with two lobes.

The points that will be used as lobes need to have there distance maximised, and the problem of maximising minimas is an NP-Complexity type problem. named the Tammes Problem, with the only current computational approach being that described by Berman and Hanes (1977). While this uses n different lobes, for a realistic model $n = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 24\}$ which are all the currently found points. Formally this can be described as an extremal configuration a of n-points on a sphere S^2 such that

$$a_i \in \operatorname{argmax}_{\mathbf{x}\in(S^2)^n}\{\min_{1\leq i<j\leq n}\{\|x_i - x_j\|\} \tag{69}$$

To further define S^2 where r_i is the size of each lobe that is chosen by the programmer and should be dependent on how many neurones will be placed within the lobe, as well as the importance of the lobe,

$$S_i^2 = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 = r_i\} \tag{70}$$

Finally the global "Brain" is the collection of all of these sub-spheres such that,

$$S_i^2 \in \mathbb{S}^2 \tag{71}$$

$$\mathbb{S}^2 = \{x, y, z \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 = d\} \tag{72}$$

Where d is a chosen value, that should be dependent on how many lobes are used, as well as how many global neurones will be used, something that will be discussed later.

ii. Modified neurone setup.

Now the architecture for the neural network has been established, the internal mechanisms need to be modified from the traditional neural networks to take advantage of the nested spheres.

1. *Input neurones*

Firstly, these neurones only use the input part of the current, hence no current from other neurones, and will be placed randomly and statically within the area given to that lobe, a_i, and are solely feed-forward neurones, as in they do not change, or evolve over time, hence they do not use Hebbian Learning or STDP, as well as not requiring a membrane potential $u(t)$. The current felt by these neurones will depend on how strong that specific input is, for example if a black on white image has an input neurone for each pixel, then for a black pixel, the neurone will have an input spike of 1, and if the pixel were white it would be 0. For clarity, these neurones will be denoted n_i^j where j is the index of each input neurone and i the index of the lobe.

$$n_i^j \in \{x_i^j, y_i^j, z_i^j \in \mathbb{R}^3 | x_{i,j}^2 + y_{i,j}^2 + z_{i,j}^2 \leq r_i\} \tag{73}$$

2. *Local neurones*

These are the neurones that will learn within the lobe a_i, and are recurrent, as in they learn overtime and are not static in their position. They will use Hebbian Learning, as well as the Fourier based spatial evolution that will be developed later. The current felt from these neurones will solely come from other neurones, however input neurones will have a much larger impact than other local neurones. The number of local neurones within the lobe is completely arbitrary, and will require code to see which configurations work best. These neurones will be denoted l_i^h and will use a modified Leaky Integrate and Fire model for their membrane potential.

$$l_i^h \in \{x_i^h, y_i^h, z_i^h \in \mathbb{R}^3 | x_{i,h}^2 + y_{i,h}^2 + z_{i,h}^2 \leq r_i\} \tag{74}$$

3. *Output neurones*

These are the neurones that will output what the network recognises, such as a word or a shape, and are located on the edge of the sphere of a lobe a_i. They are distributed randomly on the edge of their respective neurones and should be chosen in relation to how many potential choices a lobe has. For example a number recognition system between 0-9 requires 10 output neurones, one for each number. Formally these will be written as o_i^f where f is the number of output neurones.

$$o_i^f \in \{x_i^f, y_i^f, z_i^f \in \mathbb{R}^3 | x_i^2 + y_i^2 + z_i^2 = r_i\} \tag{75}$$

4. *Global neurones*

These neurones, g_m are global in the sense they are placed, and can move wherever so long as they are within the global sphere. They have a much lesser impact on output neurones, as well as local neurones as there main goal is simply to transfer information between different lobes, such as when working out the speed of an object, both Lidar and Radar will be used and the combination of these signals will be processed within the neural network, hence working together in conjunction to further validate the interpretation of the data.

$$g_m \in \mathbb{S}^2 \vee g_m \notin S^2 \tag{76}$$

iii. Visualisation of such a neural network

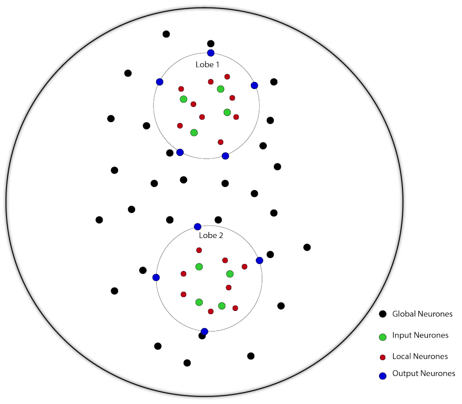


Figure 9. Illustration of a neural network with the pre-described architecture

iv. Modified Leaky Integrate and Fire

As previously discussed in subsection II iii, the Leaky Integrate and Fire models are used to simulate a resistor and capacitor in parallel, which in this case is the membrane potential $u(t)$. Each individual neurone will use the typical integrate and fire method wherein there is a variable u_i that describes the momentary membrane potential of neurone i. However, the membrane potential also depends on what neurone interaction, $\mathbb{I}$ is taking place, and by the distances those neurones are from each other.

$$\mathbb{D} = d\{(n_j l_h) \wedge (l_{h_1}, l_{h_2}) \wedge (g_{m_1} g_{m_2}) \wedge (l_h o_f) \wedge (g_m o_f)\} \tag{77}$$

Where $\wedge$ is an or operator, and $d(.)$ is a distance function, in this case a 3-Dimensional Pythagorean,

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{78}$$

. The variable $\mathbb{D}$ represents all the potential interactions between the different neurone classes, all of which will be influence by their distance. If the two neurones, p_1 and p_2, do not belong to one of those interaction, no connection is felt between those neurones hence,

$$\{p_1, p_2 \notin \mathbb{D} \mid I(t) = 0\} \tag{79}$$

Now the modified LIF model can be developed. Firstly, the input current from a pre- to a post-synaptic neurone will be dependent on the distance that pulse travels, and for each interaction, the programmer can choose an n value, which will determine how important distance is.

$$I(t) = \frac{1}{\mathbb{D}^n} \cdot I_R(t) + I_C(t) \tag{80}$$

Where n is the importance of each interaction on a neurone, hence there will be 6 different values as there are 6 neurone classes, for example the interaction between input neurones and local neurones is strong, but that between output neurones and global neurones is weak. Following this and using Equation 4

$$I(t) = C\frac{du}{dt} - \frac{[u(t) - u_{\text{rest}}]}{\mathbb{D}^n} \tag{81}$$

and finally using Equation 6

$$u(t) = u_{\text{rest}} + \exp\left(-\frac{t - t_0}{\tau_m \mathbb{D}^n}\right) + \frac{1}{C}\int_{t_0}^{t} \exp\left(-\frac{t - t'}{\tau_m \mathbb{D}^n}\right) I(t')\mathrm{d}t' \tag{82}$$

In this case, the exponential decay happens faster depending on the importance, and distance, of each interaction between neurones.

v. Modified Fourrier Rate-Based Hebbian Learning

In Section II v, we characterised the overall function F as being a Taylor expansion, however in this frequential approach to neurone interaction, a Fourier expansion will be used.

1. *Fourier Expansion of Rate Based Hebbian Learning*

Referring to Equation 10, an approximative function must be made in order to estimate how the weights between a pre- and post-synaptic neurone evolves with time. In this instance a Fourier Expansion will be used wherein ν_j and ν_i are not constrained to an approximation around 0.

$$\frac{dw_{ij}}{\mathrm{d}t} = c_0(w_{ij}) + \sum_k c_k^{pre}(w_{ij})e^{ik\nu_j} + \sum_k c_k^{post}(w_{ij})e^{ik\nu_i} + c_{l,m}^{corr}(w_{ij})e^{i(l\nu_j + m\nu_i)} \tag{83}$$

Where $k, n \in \mathbb{I}$, and for the C^{corr} term, n, k can be chosen for whatever weighting, such as more of an emphasis on pre-synaptic correlations would mean $n > k$, vice versa, or $n = k$ in which case both the pre- and post-synaptic pulse have the same impact on the Hebbian term. What makes better, despite being computationally more efficient, is that it is a local approximation within a given frequency, which will be calculated later, as opposed to a Taylor expansion which is local expansion on one point, hence $\frac{dw_{ij}}{\mathrm{d}t}$ has been approximated throughout its entire frequential domain, and it is easier to analyse using the DFT.

2. *Incorporation of DFT Analysis*

Now using the DFT on a set of spikes, $\mathbb{X}_z$, where

$$\mathbb{X}_z = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{16} \end{bmatrix} \tag{84}$$

Therefore a DFT analysis is performed every 16 spikes emitted by a neurone, this can be altered to any 2^n following the rules of an FFT which can only operate in base 2. After each batch of 16, $\mathbb{X}_z$ resets becoming an empty array after the DFT is applied. While mathematically the FFT and DFT have the same results, the FFT is a lot faster in its computation but more abstract to describe. Now since $\mathbb{X}_z \in \mathbb{R}$ and referencing equation 68,

$$\mathbb{X}_z \to \mathbb{F}(\mathbb{X}_z) \to f(\mathbb{X}_z) \tag{85}$$

$f(\mathbb{X}_z)$ is kept for each local, and global neurone hence now there needs to be an index for a local, $f(\mathbb{X}^L_{h,i})$ and global neurones $f(\mathbb{X}^G_m)$, now the interactions between these spectra and how they can add to the membrane potential will be discussed.

3. *Analysis of single neurone*

Now that we have a sample $f(\mathbb{X}_z)$ of a single neurone, take the maximum frequency such that,

$$\mathbb{Y}_1 = \text{argmax}(f(\mathbb{X}_z)) \tag{86}$$

Then this repeats every 16 times the neurone fires such that $\mathbb{Y}_1$ is the maximum frequency between 1-16 spikes, $\mathbb{Y}_2$, is the maximum frequency between 17-32 spikes until $\mathbb{Y}_n$ is the maximum frequency between spike $16(n-1)+1$ and $16n$. These are then appended into an array.

$$\mathbb{Y} = \begin{bmatrix} \mathbb{Y}_1 \\ \mathbb{Y}_2 \\ \vdots \\ \mathbb{Y}_n \end{bmatrix} \tag{87}$$

Now a frequential array for each neurone has been established, relationships between different neurones need to be established in order to influence the c^{corr} term in Equation 83,

4. *Covariance in Neuronal Relations*

Covariance is a measure of the strength between two or more random variates, the formula is given as

$$\text{Cov}(X,Y) = \sum_{i=1}^{N} \frac{(x_i - \bar{x})(y_i - \bar{y})}{N} \tag{88}$$

In order to calculate our coefficient, the variance of X and Y needs to be calculated, given as

$$\text{Var}(X) = \sum_{i=1}^{N} \frac{(x_i - \bar{x})^2}{N} \tag{89}$$

The unbiased $N-1$ formula is not used as the total neurone population will be calculated. However, if the network has an interaction size greater than 30, Central Limit Theory can be applied and the calculations can be based on samples, instead of the holistic calculation. Then dividing that by the variance, ρ gives a correlation coefficient for a total population N,

$$\rho = \frac{\text{Cov}(X,Y)}{\sqrt{\text{Var}(X)\cdot\text{Var}(Y)}} \tag{90}$$

In this case it will be a comparison between two neurones. The mean of our previously mentioned $\mathbb{Y}$ in Equation 87, can be simply calculated by the formula

$$\bar{\mathbb{Y}} = \sum_{i=1}^{n} \frac{\mathbb{Y}_i}{n} \tag{91}$$

There the comparison of two neurones with frequency arrays $\mathbb{Y}^1$ and $\mathbb{Y}^2$, using Equations 90 and 91, the correlation coefficient can be written as,

$$\rho_{(\mathbb{Y}^1,\mathbb{Y}^2)} = \frac{\text{Cov}(\mathbb{Y}^1,\mathbb{Y}^2)}{\sqrt{\text{Var}(\mathbb{Y}^1)\cdot\text{Var}(\mathbb{Y}^2)}} \tag{92}$$

Therefore, using taking the correlation coefficient from the Hebbian, Fourier Expansion Equation 83, it can now be written as,

$$c^{\text{corr}}_{l,m} = \rho_{(\mathbb{Y}^1,\mathbb{Y}^2)} \tag{93}$$

There are some limitations to using covariance, notably that it will only detect linear correlations, so it's up to the programmer to decide which correlation coefficient fits the neurones best. Using the frequency of the interaction, the next weighting can be calculated from the Inverse Fourier Series for c^{pre} and c^{post}. Hence,

$$\begin{aligned} c^{pre}_{k+1}(w_{ij}) &= \frac{1}{2\mathbb{Y}_n}\int_0^{\mathbb{Y}_n} c^{pre}_k(w_{ij})e^{int}\mathrm{d}t \\ c^{post}_{k+1}(w_{ij}) &= \frac{1}{2\mathbb{Y}_n}\int_0^{\mathbb{Y}_n} c^{post}_k(w_{ij})e^{int}\mathrm{d}t \end{aligned} \tag{94}$$

Now the weights can update iteratively in relation to the frequency the neurone emits.

5. *Spatial Plasticity through the Covariance of Fourier Analysis.*

Now that $c^{\text{corr}}_{l,m}$ has been set, and $\rho_{(\mathbb{Y}^1,\mathbb{Y}^2)}$ has been explained, the last modification of Hebbian Learning can be

made. In Section V i the framework for the neural network was set up through some spatial constraints, these will be the boundaries to our spatial evolution. If a neurone is found on one of these spheres, it is deleted from the neural network, and a new neurone of the same class is placed added to its respective location. Furthermore, when this process stops, the neural network should have finished its learning process and hence can be used, if its overall success rate is high enough, to carry out its task without supervision.
Now the interactions between each neurone classes will be developed, if an imaginary line between two neurones, $p_1 = \{x_1, y_1, z_1\}$ and $p_2 = \{x_2, y_2, z_2\}$, were to be drawn, it's distance $\mathbb{D}$, as previously described in Equation 78 will be calculated using the 3D Pythagorean. Now if there is a positive weight change between neurone p_1 and p_2 for consecutive iterations, such that $\forall \Delta w > 0$ inbetween iteration n and $n+1$ or at least for most of the time then spatial evolution will help the neural network evolve, and decrease the membrane potential, as explained in Equation 82 for those respective neurones. This distance change will be a function of $\rho_{(\mathbb{Y}^{p_1}, \mathbb{Y}^{p_2})}$, which will be written as Γ for brevity, such that,

$$\begin{aligned}\mathbb{D}_{p_1,p_2} &= d(p_1 + k \cdot \Gamma \vee p_2 - k \cdot \Gamma) \\ &= d(p_1 + k \cdot \Gamma \vee p_2 - k \cdot \Gamma) \qquad (95)\end{aligned}$$

k is a chosen constant for how much emphasis is placed on the interaction, which will be discussed later. Hence,

$$\begin{aligned}\mathbb{D}_{p_1,p_2} = d((&x_1 + k \cdot \Gamma), (y_1 + k \cdot \Gamma), (z_1 + k \cdot \Gamma) \vee \\ &(x_2 - k \cdot \Gamma), (y_2 - k \cdot \Gamma), (z_2 - k \cdot \Gamma)) \qquad (96)\end{aligned}$$

As neurones don't interact only with one other neurone, the sum of these interactions must be taken such that j is the number of neurones that the initial neurone, p_1, is connected to, hence

$$\begin{aligned}\mathbb{D} = d((x_1 + \sum_{n=i}^{j} k_i \cdot \Gamma), (y_1 + \sum_{n=i}^{j} k_i \cdot \Gamma), (z_1 + \sum_{n=i}^{j} k_i \cdot \Gamma) \\ \vee\ (x_2 - \sum_{n=i}^{j} k_i \cdot \Gamma), (y_2 - \sum_{n=i}^{j} k_i \cdot \Gamma), (z_2 - \sum_{n=i}^{j} k_i \cdot \Gamma))\end{aligned} \qquad (97)$$

As k has different values for different neural interactions, the total of which shown in Equation 77, hence k_i will be that specific value for that interaction. In the case that neurones are far apart, $\mathbb{D}$ can determine whether or not a signal will be sent between two neurones. This can be shown by using λ_c as the specific limit to that interaction,

$$\{(p_1 \vee p_2 \notin g_m) \bigwedge \mathbb{D} > \lambda_c \mid I(t) = 0\} \qquad (98)$$

vi. Categorising neurone interactions.

In this subsection, the categorisation of each neurone interaction, and the changes on n from Equation 82 describes the effect of distance, k from Equation 95 influences the importance of linear correlation and spatial plasticity, and λ_c from Equation 98 that imposes a limit on the reach of a neurone.

1. Local neurones and static neurones.

There is no spatial plasticity in this interaction due to the static nature of input neurones, and output neurones hence k is zero. However their should be no distance limit between local neurones and the static neurones, hence λ_c is high, as long as they belong to the same lobe, however n should be fairly large which will cause neurones to not cluster the static neurones, but instead only having a few neurones transfer the vast majority of the data.

2. Local neurones.

These are the most common interactions, and therefore influence the networks outcome the most. However these interactions are restricted to their lobe, a_i as well as having fairly short connections, hence λ_c should be fairly high, however n should be large as close neurones must have a much greater impact than those that are further away. Fluidity is encourages within these neurones, hence k is fairly high.

3. Global neurones.

The main goal of these neurones is to transfer information between other lobes and their outputs, hence n can be very low, and λ_c should be high to allow information flow. The importance of them firing together is very high however, so k must be low in order to ensure that neurones must be very similar in order to reorganise themselves.

4. Global neurones and output neurones.

These interactions are meant to influence the final decision of the lobes by taking data from other lobes. It is important that n is high so that only the closest neurones transfer any information to the output neurones. Fluidity should not occur as there is a static neurone involved, hence k is zero, while λ_c is low so only neurones that are close to the lobes transfer information.

vii. Overview of all interactions.

Throughout this section 3 constants have been encountered, n, k, and λ_c, which will each take different values for different interactions. An overview of what they should be is summarised in the table below.

Constants	$(n_j \vee o_f), l_h$	l_{h_1}, l_{h_2}	g_{m_1}, g_{m_2}	g_m, o_f
n	Low	High	Low	Low
λ_c	High	High	High	Low
k	Zero	High	Low	Zero

VI. Open Problems.

As previously described, BRAINN can be rather tedious, and computationally inefficient. The network however can be analysed using some fairly advanced mathematical attributes, firstly instead of having nested spheres a 4-Dimensional Hypersphere would be much less computationally heavy as it inherently has nested spheres while also allowing global boundedness. The signal analysis part of the network should use a Fast Fourier Transform(FFT) instead of a DFT, to reduce the number of operations from $O(n^2) \rightarrow O(n \log n)$ where n is the data size, in this case the recorded frequencies of every input. Another idea is to use topology and graph theory, wherein the neurones are vertices, and the synapses are edges, which are then turned into a directional simplicial complex structure. From this, neural information can be extracted from a finalised BRAINN to compare it's dimensionality to that of a biological brain undergoing the same sensory input. Now for neural optimisations, all the output neurones could be summed into one for each lobe, and in accordance to its spike train, or its frequency of fire, could give what it believes as the answer. As this network relies on many different neurones, each with different parameters and tasks, a useful approach in reducing the load of the computation could be to use game theory, specifically mean field theory to approximate, instead of compute, future movement and spikes within the network. A Monte Carlo approach can be taken as well, in which the spatial evolution of the neurones could be probabilistic instead of holistic as described previously.

[1] Gerstner, Wulfram, and L.F. Abbott. "Homeostasis and Learning through Spike-Timing Dependent Plasticity." *EPFL Scientific Publications*, Feb. 2004, infoscience.epfl.ch/record/114304/files/Abbott04.pdf.

[2] Ponulak, F, and A Kasinski. "Introduction to Spiking Neural Networks: Information Processing, Learning and Applications"*National Center for Biotechnology Information,* U.S. National Library of Medicine, 2011, www.ncbi.nlm.nih.gov/pubmed/22237491.

[3] Gerstner, Wulfram, et al. "Introduction: neurones and Mathematics."*Neuronal Dynamics: From Single neurones to Networks and Models of Cognition.*, Cambridge University Press, 2015, pp. 3-24.

[4] Soni, Devin. "Spiking Neural Networks, the Next Generation of Machine Learning" *TowardsDataScience*, 11January 2018. `https://towardsdatascience.com/spiking-neural-networks-the-next-generation-of-machine-learning-84e167f4eb2b`

[5] Denis Dmitriev. "Neural Network 3D Simulation." Online video clip. Youtube. Youtube, 15 November 2016. Web. 18 January 2018.

[6] 3Blue1Brown."But what is the Fourier Transform? A visual introduction" Online video clip. YouTube. YouTube, 26 January 2018. Web. 27 January 2018.

[7] Physics Videos by Eugene Khutoryansky. "Fourier Transform, Fourier Series, and frequency spectrum" Online video clip. YouTube. YouTube, 6 September 2015. Web. 11 October 2017.

[8] Dawkins, Paul. "Differential Equations - Fourier Series." *Pauls Online Notes*, 2018, `tutorial.math.lamar.edu/Classes/DE/FourierSeries.aspx`.

[9] Khamsi, M. "Fourier Series: Basic Results." *SOSMath*, Math Medics, LLC, 2016, `www.sosmath.com/fourier/fourier1/fourier1.html`.

[10] Pierce, Rod. "Fourier Series" *Math Is Fun*, 27 Dec 2017.`mathsisfun.com/calculus/fourier-series.html`.

[11] Brookes, Mike. "Complex Fourier Series." *Department of Electrical and Electronic Engineering*, Imperial College London, 2014, `www.ee.ic.ac.uk/hp/staff/dmb/courses/E1Fourier/00300_ComplexFourier.pdf`.

[12] MathTheBeautiful. "Complex Fourier Series" Online video clip. YouTube. YouTube, 15 March 2017. Web. 9 October 2017

[13] Ahmed Isam. "1. Understanding Fourier Series, Theory + Derivation". Online video clip. YouTube. Youtube, 25 February 2014. Web. 9 October 2017.

[14] Ahmed Isam. "2. Understanding Fourier Transform, Theory + Derivation". Online video clip. YouTube. Youtube, 11 December 2014. Web. 10 October 2017.

[15] Ahmed Isam. "3. Understanding The Discrete Fourier Transform dtFT / DFT and sampling theory". Online video clip. YouTube. Youtube, 11 December 2014. Web. 10 October 2017.

[16] Ahmed Isam. "4. Understanding The Discrete Fourier Transform DFT, Theory and Derivation". Online video clip. YouTube. Youtube, 12 December 2014. Web. 10 October 2017.

[17] Ahmed Isam. "5. Understanding The Fast Fourier Transform FFT". Online video clip. YouTube. Youtube, 12 December 2014. Web. 12 August 2018.

[18] Simon Xu. "Discrete Fourier Transform - Simple Step by Step". Online video clip. YouTube. Youtube, 3 August 2015. Web. 12 October 2017 .

[19] Azad, Kalid. "An Interactive Guide To The Fourier Transform." *BetterExplained*, 2017, betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/.

[20] Stuart Riffle. " Understanding the Fourier Transform " *AltDevBlogADay*, 2012, www.altdevblogaday.com/2011/05/17/understanding-the-fourier-transform/

[21] Weisstein, Eric W. "Discrete Fourier Transform." *MathWorld*, Wolfram Web Resource. mathworld.wolfram.com/DiscreteFourierTransform.html

[22] Arar, Steve. "Discrete Fourier Transform." *Wolfram MathWorld*, 20 July 2017, mathworld.wolfram.com/DiscreteFourierTransform.html.

[23] Khamsi, M. "Covergence of Fourier Series". *SOSMath*, Math Medics, LLC, 2016, http://www.sosmath.com/fourier/fourier3/fourier31.html.

[24] Basarab-Horwath, Peter. "Lecture 3: Fourier Series: Pointwise and Uniform Convergence." *Linkopings Universitet*, 2015, courses.mai.liu.se/GU/TATA57/Dokument/FourierSeries2.pdf.

[25] 3Blue1Brown."But what is a Neural Network? | Deep learning, chapter 1" Online video clip. Youtube. Youtube, 5 October 2017. Web. 6 October 2017.

[26] 3Blue1Brown."Gradient descent, how neural networks learn | Deep learning, chapter 2" Online video clip. Youtube. Youtube, 16 October 2017. Web. 18 October 2017.

[27] 3Blue1Brown."What is backpropagation really doing? | Deep learning, chapter 3" Online video clip. Youtube. Youtube, 3 November 2017. Web. 3 November 2017.

[28] 3Blue1Brown."Backpropagation calculus | Deep learning, chapter 4" Online video clip. Youtube. Youtube, 3 November 2017. Web. 3 November 2017.

[29] Moawad, Assaad. "Neural networks and backpropagation explained in a simple way" *Medium*, 1 February 2018, https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

[30] Anuradha Wickramarachchi. "Introduction to Neural Networks" *TowardsDataScience*, 1 July 2017. https://towardsdatascience.com/introduction-to-neural-networks-ead8ec1dc4dd

[31] Tushar Gupta. "Deep Learning: Back Propagation" *TowardsDataScience*, 25 January 2017. https://towardsdatascience.com/back-propagation-414ec0043d7

Appendices

A. Artificial Neural Networks

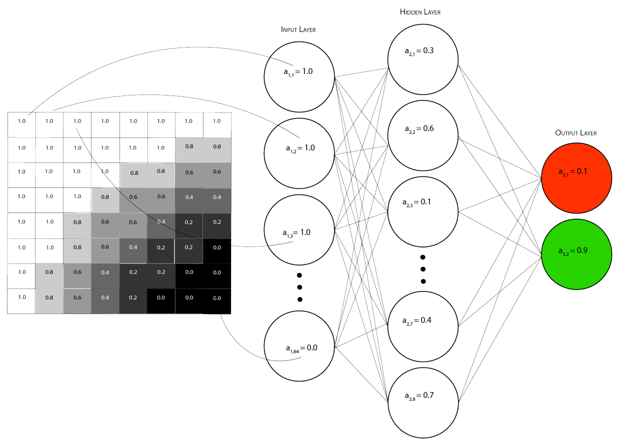
Artificial Neural Networks (ANNs) are at the forefront of 21st century computing, from being used by Netflix to recommend a new series, to identifying the probability of life on other planets, it is clear that they are not only present in almost every algorithm, but they are crucial for the functioning of our society. While there are currently other frameworks being used, such as Convolutional Neural Networks for image recognition or Long short-term memory for speech recognition, they are the extensions of ANNs, and that is why in this paper we will focus on implementing the DFT on top of an ANN, in the future however the same principles could be applied to other systems.

i. The Structure of an Artificial Neural Network

Before we begin talking about the mathematics behind Artificial Neural Networks, let's first talk about what it actually is, its structure, and to highlight its importance in both present and the future of technology. As the name suggests, the Artificial Neural Network (ANN) was inspired by the brain, and it was hypothesized in early 1940?s by Donald Hebb, and mathematically explained by McCulloch & Pitts.
For the sake of simplicity, it is easier, to begin with, an example and bring it back to the theory. If you had an 8 by 8 black and white image, and you wanted the computer to figure out if there was a curve or a straight edge.

The inputs would be each of those pixels, hence 8x8 = 64 input nodes, a node is just a hypothetical object that holds a number. Each of these input nodes will have a number between 0 and 1 dependent on its grayscale value (1 representing white, 0 representing black, and all the values in between being a shade of grey), this is called the input activation number. Now for the output nodes, you would want 2 of these, one representing a curve and the other a straight edge. If the computer were to spit out a value close to one for an edge or a curve, then that means it is fairly sure that the image has an edge or a curve, however if the values are both around 0.5, or $\frac{1}{n}$ where n is the number of output nodes, then it means the computer is not sure what the image represents. In this case, $a_{3,2}$ represents the output node of a curve, and hence the computer believes it is a curve with relative certitude.

One of the main concepts behind an ANN is its inherent connectivity. Each node has a specific weight and is dependent on the sum of all the previous weights,



hence to generate the next node in the network it would want the sum of all the weights and use a logistics curve, such as the sigmoid function, that would place this number between 0 and 1. Most of the time, however, you want to make sure the value is high enough so that is fires, or that it does not fire, in this case, you would add a bias to the end of the weights before being put into the sigmoid function unique to each node such that would make sure your node would only fire when the correlation is strong enough. Furthermore, this process happens for every node, so the number of connections within the previously mentioned networks would be $(64)_{\text{input}} \times (8)_{\text{hidden}} \times (2)_{\text{output}} = 1024$, and all of this is repeated for every iteration of the network which causes these networks to require exponentially more computing power, in fact, one of the main issues with ANN's is their scalability, especially with image and sound recognition. Now we can change our definition of a node from something that holds a number, to a function that takes in the weights of the previous layer, and outputs numbers to the next layers.

ii. Parameterising Artificial Neural Networks

As we previously mentioned, an ANN is composed of multiple nodes, each of which have a weight's dependent on the previous nodes, a bias and is passed through a logistics function, in this case we'll use the sigmoid as it is easy to differentiate.

$$\sigma = \frac{1}{1+e^{-x}} \tag{99}$$

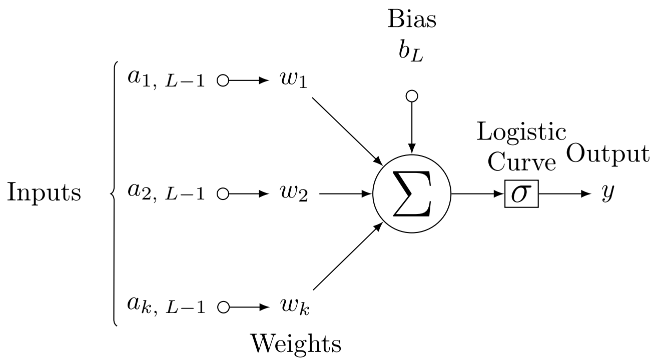
$$a_{k,i} = \sigma\Big(\sum_{L=0}^{n} w_{k,i} a_{(k-1),L} + b_k\Big) \tag{100}$$

Where a is the value held by the node, L is the layer number, hence $L-1$ is the previous layer, k is the position within that layer, b_k is the associated bias of that layer. This can be simplified in matrix form,

$$a_{k,i} = \sigma\Big(\begin{bmatrix} w_{0,L} & w_{1,L} & \cdots & w_{k,L} \end{bmatrix} \begin{bmatrix} a_{0,L-1} \\ a_{1,L-1} \\ \vdots \\ a_{k,L-1} \end{bmatrix} + b_L\Big) \tag{101}$$

and $\vec{w}$ is the sum of all of these

$$\vec{w}(a) = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,L} \\ w_{1,0} & w_{1,1} & \cdots & w_{0,L} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,2} & \cdots & w_{k,L} \end{bmatrix} \begin{bmatrix} a_{0,L-1} \\ a_{1,L-1} \\ \vdots \\ a_{k,L-1} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix} \tag{102}$$



Now that we have defined what each output of the nodes will be, we can input random initial weights and associate a cost function with what the output should be and what it is using training data (data the computer can compare to). In the beginning, the network should spit out random outputs, with uncertainty in the values (e.g. around 0.2 for each node instead of a certain 0.9+). For each output node, there will be an associated real value associated with it, in a sense, it's telling the computer what it did badly in order to make it optimise itself. The cost function will take in all the weights and biases of the nodes, and return one value, the valuation of how well the network did, keep in mind this is an extremely large input space (Often in the 10^6 in most used neural networks, which means 10^6 dimensions).

$$C(\vec{w}) = \begin{bmatrix} b_0 \\ a_{0,0} \\ \vdots \\ b_L \\ a_{k,L} \end{bmatrix} \tag{103}$$

The goal in any cost function is to minimise it, in this case, the cost will be

$$Cw = (a_k - y_k)^2 \tag{104}$$

where a_k is the inidividual activation output, and y_k is the expected value respect to the training data.

In multivariable calculus, the gradient is just that of the steepest ascent, hence you would just take the negative value of that, where each direction would represent a change in weights and biases, and the length of that vector would be the magnitude of the descent. This function will be called $\nabla C(\vec{w})$ and is the backbone of backpropagation, something we'll get to later. In essence, the computer will just calculate the $\nabla C(\vec{w})$ and subtract that to our $\vec{w}(a)$ function, which will cause respective changes to each of the weights and biases that will decrease our cost, and increase our accuracy. It is to be noted the greater the change, the more important that node is to determine the output of the artificial neural network.

$$\vec{w}_n(a) - \nabla C(\vec{w}_n) = \vec{w}_{n+1}(a) \tag{105}$$

Now the framework has been set up, we can go into how the computer minimises the cost function

iii. The Backpropagation Algorithm

In order to minimise any n-th dimension variable, we must take its derivative, find its gradient and take the negative of that value. In this case, we'll use a 1-dimensional neural network, and expand upon it later. Now imagine the most basic ANN, two nodes, and one connection between them.

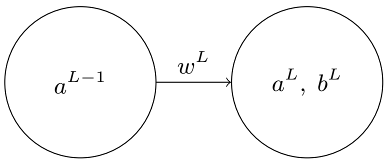


Figure 10. A basic 1 by 1 ANN showing what one connection is made of.

$$z^L = w^L a^{L-1} + b^L \tag{106}$$

$$a^L = \sigma(z^L) \tag{107}$$

$$C_0(...) = (a^L - y)^2 \tag{108}$$

The function we want to minimise is the

$$\frac{\partial C_0}{\partial w^L} \text{ and } \frac{\partial C_0}{\partial b^L} \tag{109}$$

which we can do by using the chain rule

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} \tag{110}$$

$$\frac{\partial z^L}{\partial w^L} = a^{L-1} \tag{111}$$

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L) \tag{112}$$

$$\frac{\partial C_0}{\partial a^L} = 2(a^L - y) \tag{113}$$

$$\frac{\partial C_0}{\partial w^L} = a^{L-1}\sigma'(z^L)2(a^L - y) \tag{114}$$

$$\frac{\partial C_0}{\partial b^L} = \frac{\partial z^L}{\partial b^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} \tag{115}$$

$$\frac{\partial z^L}{\partial b^L} = 1 \tag{116}$$

$$\frac{\partial C_0}{\partial b^L} = (1)\sigma'(z^L)2(a^L - y) \tag{117}$$

And all of this is just one component of the gradient vector, which is built up of

$$\nabla C(\vec{w}) = \begin{bmatrix} \frac{\partial C_0}{\partial w^1} \\ \frac{\partial C_0}{\partial b^1} \\ \vdots \\ \frac{\partial C_0}{\partial w^L} \\ \frac{\partial C_0}{\partial b^L} \end{bmatrix} \tag{118}$$

Now this is for a 1 dimensional neural network, if it were to be a 2D one like previously, you would just need to add the k subscript, and c_j is the position of the node within the previous layer.

$$z_j^L = w_{j0}^L a_0^{L-1} + w_{j1}^L a_1^{L-1} + ... + w_{jk}^L a_k^{L-1} = \sum_{k=0}^{n_L-1} w_{jk}^L a_k^{L-1} + b_j^L \tag{119}$$

$$a_j^L = \sigma(z_j^L) \tag{120}$$

$$C_0(...) = \sum_{k=0}^{n_L-1} (a_k^L - y_k)^2 \tag{121}$$

and the gradient descent function is

$$\frac{\partial C_0}{\partial a_k^{L-1}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^L}{\partial a_k^{L-1}}\frac{\partial a_j^L}{\partial z_j^L}\frac{\partial C_0}{\partial a_j^L} \tag{122}$$

$$\frac{\partial C_0}{\partial a_k^{L-1}} = a_k^{L-1}\sigma'(z_j^L)\frac{\partial C}{\partial a_j^L} \quad (123)$$

$$\frac{\partial C}{\partial a_j^L} = \sum_{j=0}^{n_L} w_j k^{L+1}\sigma'(z_j^{L+1})\frac{\partial C}{\partial a_j^{L+1}} \text{ or } 2(a_j^L - y_j) \quad (124)$$

B. Convergence of Fourier Series

While we have previously discussed the approximative insight the Fourier Series gives, we never delved into why the series converge as $n \to \infty$. In this chapter we will focus on 2π functions as opposed to $2L$ functions. This function is defined on [a,b] and is piecewise continuous if there exists partitions $[n_1, n_2, ..., n_i] \vdash [a, b]$ such that $f(x)$ is continuous in between n_{i-1} and n_i. The function is considered piecewise smooth if $f'(x)$ is also continuous between n_{i-1} and n_i. As the function is not fully continuous throughout the real or complex domain, the fundamental theorem of Calculus which needs to be adjusted to account for the redundancies of points a and b.

$$\lim_{x_a-} f(x) = f(a) \text{ and } \lim_{x_b+} = f(b) \quad (125)$$

Hence,

$$\lim_{x_a-} f(x) - \lim_{x_b+} f(x) = \int_b^a f'(x)\mathrm{d}x \quad (126)$$

Then

$$\lim_{k\to\infty}\int_b^a f(x)\sin(xk)\mathrm{d}x = 0$$
$$\text{and } \lim_{k\to\infty}\int_b^a f(x)\cos(xk)\mathrm{d}x = 0$$

i. Proof

Recall that the Fourier Series is a global approximation of a function, as opposed to a Taylor series which is a local approximation.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n\cos(2\pi nx) + b_n\sin(2\pi nx) \quad (127)$$

Which can also be written to its Nth degree as which are known as Fourier Partial sums.

$$f_N(x) = \frac{a_0}{2} + \sum_{n=1}^{n=N} a_n\cos(2\pi nx) + b_n\sin(2\pi nx) \quad (128)$$

The full Fourier sum is

$$f_N(x) = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\mathrm{d}t$$
$$+\sum_{n=1}^{n=N}\frac{1}{\pi}\int_{-\pi}^{\pi} f(t)(\cos(nt)\cos(nx) + \sin(nt)\sin(nx))\mathrm{d}t \quad (129)$$

Which simplifies to

$$f_N(x) = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)[\frac{1}{2} + \sum_{n=1}^{N}\cos(n(x-t)]\mathrm{d}t \quad (130)$$

Following $\cos(a-b) = \cos(a)\cos(b) + \sin(a)\sin(b)$.
If we substitute $s = x - t$ and assume f is 2π periodic

$$f_N(s) = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x-s)[1 + 2\sum_{n=1}^{n=N}\cos(ns)]ds \quad (131)$$

From this we can extract a trigonometric series, which called the Dirichlet Kernel,

$$D_N(s) = 1 + 2\sum_{n=1}^{N}\cos(ns) \quad (132)$$

Which has the following properties for $N, \forall s \in \mathbb{N}$

Remark. $D_N(s)$ is a 2π-periodic function.

Remark. $D_N(0) = 2N + 1$.

Lemma 2. $|D_N(s)| \leq 2N + 1$.

Proof. Since $\max(cos(x))$ occurs at $x = 0$, at all other values $cos(x)$ is smaller. Formally this can be written as

$$\forall x \neq 0, y = 0 \quad (133)$$

□

Lemma 3. $\frac{1}{2\pi}\int_{-\pi}^{\pi} D_N(s)ds = 1$.

Proof.

$$\begin{aligned}\frac{1}{2\pi}\int_{-\pi}^{\pi} D_N(s)ds &= \frac{1}{2\pi}\int_{-\pi}^{\pi} 1 + 2\sum_{n=1}^{N}\cos(ns)ds \\ &= \frac{1}{2\pi}(s - 2\sum_{n=1}^{N}\sin(ns))|_{-\pi}^{\pi} \\ &= \frac{1}{2\pi}(2\pi + 2\sum_{n=1}^{N}\sin(n\pi) - \sin(n\pi)ds \\ &= \frac{1}{2\pi}(2\pi) \\ &= 1\end{aligned}$$

□

Lemma 4. $D_N(s) = \frac{\sin(N+1/2)s}{\sin(s/2)}$.

Proof. This can be proved via some trigonometric manipulation. Firstly suppose $n \in \mathbb{N}$ & $s \neq k\pi$ for any $k \in \mathbb{Z}$

$$D_N(s) = 1 + 2\sum_{n=1}^{N}\cos(ns) \quad (134)$$

Then remembering cosine is even and sine is odd:

$$2\sum_{n=1}^{N}\cos(ns)+1=\sum_{n=-N}^{N}\cos(ns) \tag{135}$$

$$\sum_{n=-N}^{N}\sin(ns)=0 \tag{136}$$

Hence

$$\begin{aligned} D_N(s) &= \sum_{n=-N}^{N}\cos(ns)+\frac{\cos(s/2)}{\sin(s/2)}\sum_{n=-N}^{N}\sin(ns) \\ &= \frac{1}{\sin(s/2)}\sum_{n=-N}^{N}(\sin(s/2)\cos(ns) \\ &+ \cos(s/2)\sin(ns)) \\ &= \frac{1}{\sin(s/2)}\sum_{n=-N}^{N}\sin((n+1/2)s) \end{aligned} \tag{137}$$

Following $\sin(A)\cos(B)+\cos(A)\sin(B)=\sin(A+B)$.

Which is a telescoping sum, as in a sum that cancels itself with previous terms leaving a single fixed value.

$$D_N(s)=\frac{\sin((N+1/2)s)}{\sin(s/2)} \tag{138}$$

Going back to the Dirichlet Kernel, which can be defined in two domains following the 2nd trait of the Dirichlet

$$D_N(s)=\begin{cases}\dfrac{\sin((N+1/2)s)}{2\pi\sin(s/2)} & \text{if } s\neq 0,\pm 2\pi,... \\ \\ \dfrac{2N+1}{2\pi} & \text{if } s=0,\pm 2\pi,..\end{cases} \tag{139}$$

Another interesting property of the Dirichlet is

$$\int_{-\pi}^{0}D_N(s)ds=\int_{0}^{\pi}D_N(s)ds=\frac{1}{2} \tag{140}$$

□

Now the Fourier Series can be proven to be convergent as $n\to\infty$. Since $D_N(s)$ is even the integral can broken into 2,

$$\begin{aligned} f_N(x) &= \frac{1}{2\pi}\int_{-\pi}^{\pi}f(x-t)D_N(t)\mathrm{d}t \\ &= \frac{1}{2\pi}\int_{-\pi}^{0}f(x-t)D_N(t)\mathrm{d}t \\ &+ \frac{1}{2\pi}\int_{0}^{\pi}f(x-t)D_N(t)\mathrm{d}t \\ &= \frac{1}{2\pi}\int_{0}^{\pi}(f(x-t)+f(x+t))D_N(t)\mathrm{d}t \end{aligned} \tag{141}$$

Following the 4th and 5th trait of the Dirichlet kernel,

$$f_N(x)-s=\frac{1}{2\pi}\int_{0}^{\pi}(f(x-t)+f(x+t)-2s)D_N(t)\mathrm{d}t \tag{142}$$

$$f_N(x)-s=\frac{1}{2\pi}\int_{0}^{\pi}\frac{(f(x-t)+f(x+t)-2s)}{t}\cdot\frac{t}{\sin(t/2)}\sin((N+1/2)t)\mathrm{d}t \tag{143}$$

And following the final identity in the aforementioned introduction, $f_N(x)-s\to 0$ which shows that $f_N(x)$ will converge to a point s since $t/\sin(t/2)$ is bounded to the interval $(0,\pi)$ which is a finite value. This formula is used by computers to find the the Fourier Series as it does not require any formulas for the Fourier coefficients, and instead directly calculating each consecutive term.

C. Fourier Series and Transforms

i. Odd and Even coefficients

Take the signal

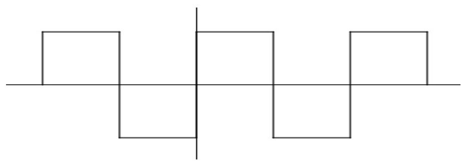


Figure 11. An odd signal, with period $\pi/2$.

In Figure 11, $f(-t)\neq f(+t)$ so this is an odd series, therefore you should use the sine function. This signal will have a period of $\pi/2$. To find b_1 multiply $f(t)$ with the first sinusoid, $\sin(2\pi 1t)$ and integrate.

$$\int_0^1 f(t)\sin(2\pi t)\mathrm{d}t \tag{144}$$

To break this down:

$$f(t)=b_1\sin(2\pi 1t)+b_2\sin(2\pi 2t)+b_3\sin(2\pi 3t)+... \tag{145}$$

So the sinusoid, $\sin(2\pi t)$ will be distributed over all the other sinusoids as well as the integral to give:

$$\int_0^1 \sin(2\pi t)(b_1\sin(2\pi t)+b_2\sin(2\pi 2t)+b_3\sin(2\pi 3t)+...\mathrm{d}t \tag{146}$$

Which can be rewritten as

$$\int_0^1\sin(2\pi t)b_1\sin(2\pi t)\mathrm{d}t+\int_0^1\sin(2\pi t)b_2\sin(2\pi 2t)\mathrm{d}t + \int_0^1\sin(2\pi t)b_3\sin(2\pi 3t)\mathrm{d}t+... \tag{147}$$

Following the orthogonality of sinusoids, all but the first trigonometric function will cancel to 0 leaving,

$$b_1 \int_0^1 \sin(2\pi t)\sin(2\pi t)\mathrm{d}t = \frac{2}{\pi} \tag{148}$$

Which is equal to,

$$\frac{b_1}{2} = \frac{2}{\pi} \tag{149}$$

$$b_1 = \frac{4}{\pi} \tag{150}$$

Now for b_2

$$\begin{aligned} &= \int_0^1 f(t)\sin(4\pi t)\mathrm{d}t \\ &= \int_0^1 b_2 \sin(4\pi t)\sin(4\pi t)\mathrm{d}t \\ &= b_2 \int_0^1 \sin(4\pi t)\sin(4\pi t)\mathrm{d}t \end{aligned} \tag{151}$$

Which would be equal to whatever value the integral is divided by 0.5. To generalize,

$$f(t) = \sum_{n=0}^{\infty} b_n \sin(2\pi n t) \tag{152}$$

$$b_n = \frac{1}{0.5}\int_0^1 f(t)\sin(2\pi n t)\mathrm{d}t \tag{153}$$

The first transform, $f(t) \rightarrow b_n$ is a forward analytic transform as it goes from the time domain to the frequency domain. Going from $b_n \rightarrow f(t)$ is an Inverse Fourier Transform going the frequency domain to the time domain.

Take another signal with duration 1s, the magnitude is not important:

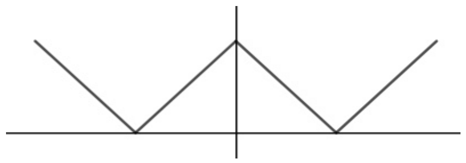


Figure 12. Even signal

In this series $f(+t) = f(-t)$ so it is an even signal, hence we should use cosine. if we were to use:

$$f(t) = \sum_{n=0}^{\infty} b_n \sin(2\pi n t) \tag{154}$$

$$b_n = 0 \rightarrow f(t) = 0 \tag{155}$$

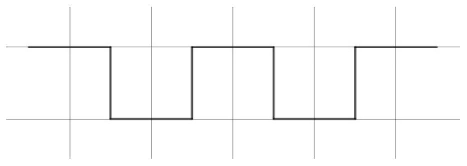
So our Fourier series for this triangular function will have no sinusoidal component, and only cosinusoidal components.

$$f(t) = \sum_{n=0}^{\infty} a_n \cos(2\pi n t) \tag{156}$$

$$a_n = \frac{1}{0.5}\int_0^1 f(t)\cos(2\pi n t)\mathrm{d}t \tag{157}$$

ii. Basic Fourier Transform

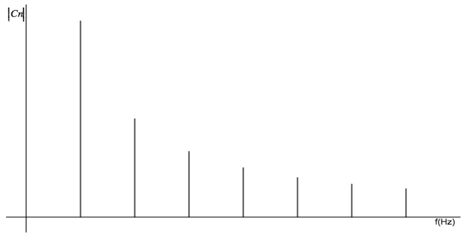
Take the signal:



Where $T = 2$ and $f_0 = 0.5$Hz. Then we can generate our Complex Fourier Series.

$$\begin{aligned} C_n &= \frac{1}{T}\int_T f(t)e^{-i2\pi n f_0 t}\mathrm{d}t \\ &= \frac{1}{2}\int_{-0.5}^{0.5} (1)e^{-i2\pi n 0.5 t}\mathrm{d}t \\ &= \frac{1}{2}\frac{1}{-j\pi n}e^{-j\pi n t}\big|_{-0.5}^{0.5} \\ &= \frac{\sin(\frac{n\pi}{2})}{n\pi} \end{aligned} \tag{158}$$

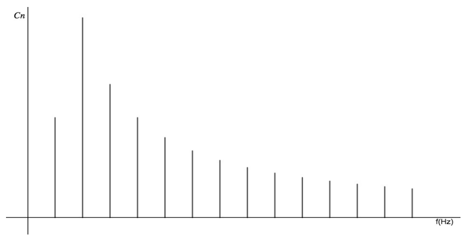
To use this in our graph, we take the absolute value of C_n. Every frequency will have an associated C_n every 0.5s, so at $n = 1$, the frequency will be $0.5Hz$, at $n = 2$ the frequency will be at $1Hz$ etc.



These values are not the expected values you would get from the function.

If we were to double the period. $T = 4$, $f_0 = 0.25$. So now the graph would look something like this

So by increasing period, fundamental frequency goes down as well as the magnitude of each component as now n_1 on first graph is equal to n_2 etc. This is because $C_n \propto \dfrac{1}{n}$. The density however increases, as in the number of individual C_n in a range of frequencies. As $T \rightarrow \infty$,



the $f_0 \to 0$, but never 0 as that is reserved for DC. So we can denote this small signal as df. The magnitude would approach 0 as well, and the density would approach infinity. The spectrum has a component in every single frequency, as $f_0 = df$. If we were to take a signal from 0 to 3 Hz then $nf_0 = 3 \to ndf = 3 \to n = \dfrac{3}{df}$, which is a very large number, since df is extremely small, therefore the spectrum is no longer discrete but continuous and hence have an associated value of C_n for every f value.