Distributed Artificial Intelligence

Taraus Renzoddy

20 August 2017

A possible solution for storing an AI model on a blockchain which has self-learning and self-stimulating features through an adaptive Proof of Intelligence protocol.

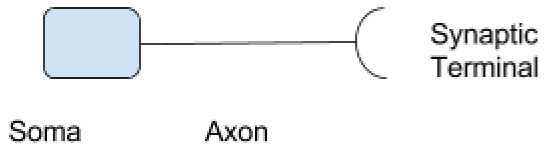
Abstract

Each year over 14 terawatts of energy is consumed in hashing and verifying bitcoin transactions. As of this paper, the network hashrate was over 8 million, trillion hashes per second. Collectively, this equates to a combined computational power of over 1 trillion, trillion computations per second. There were almost 10000 full nodes, storing almost 130 gigabytes of the blockchain each. Collectively this is a display of 1 petabyte of decentralised data capacity.
In comparison the human brain computes at around 300 trillion computations per second, with an estimated storage of over 1 petabyte.
It is entirely possible then, to build a decentralised emulation of the human brain using distributed computation and decentralised storage. Using a cryptocurrency layer and a robust economic model, a network of nodes can be coordinated and incentivised to pursue intelligence. Just like bitcoin, the ensuing intelligence will owned by everyone, and employed by all.

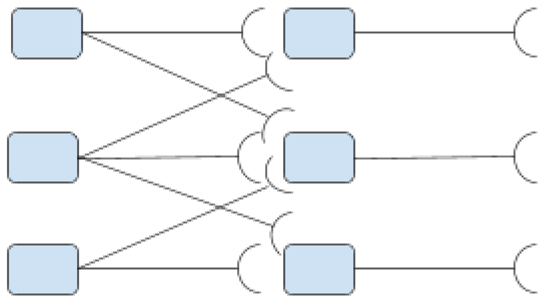
Table of Contents

Abstracting the brain

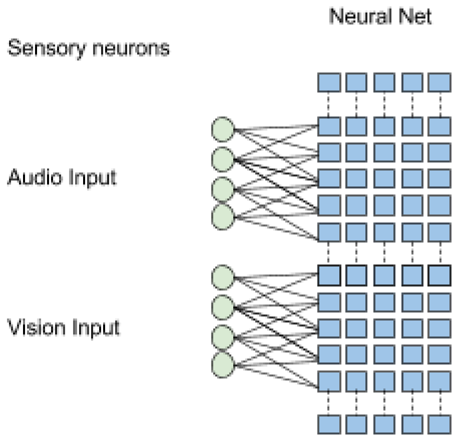
The human brain is estimated to have 100bn neurons, with each neuron having over 1000 connections, leading to a total count of upwards of 100tn neural connections [1]. Each neuron can reset its charge after 5ms. A neuron can be simplified to be comprised of a soma, an axon, and a synaptic terminal.



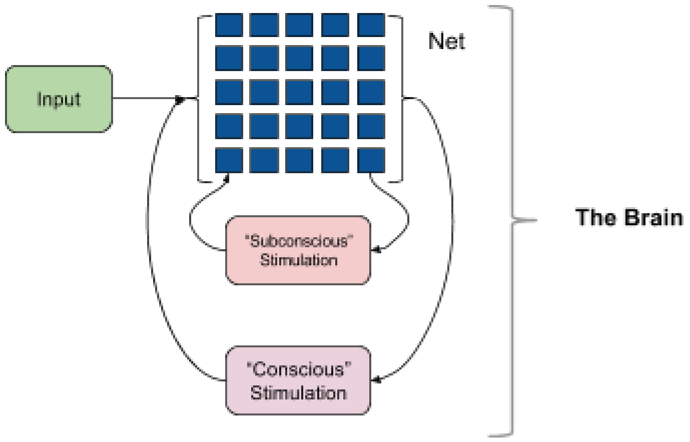
Each neuron maintains connections with other neurons. Neural communication is via "action potentials", electrical conduction that is "all-or-nothing"; a minimum level of action potential is required before a receiving neuron is excited. Neurons have inherent levels of variability and can initiate new connections. A connection that has been stimulated many times is reinforced and less likely to break. Likewise a weak or under-utilised connection may break and re-form with other neurons.


Connected Neurons

The brain is stimulated by a variety of input neurons - such as vision, audio, pressure, heat, touch, taste and smell. When stimulated, these sensory neurons then in turn stimulate distinct sections of the brain in a recognizable fashion.



The neural net is then stimulated and due to high-levels of inter-connection, patterns of stimulation emerge across the entire net. The brain recognises these patterns of stimulation to form intelligent thought. The brain has an ability to recognise these patterns of stimulation, and re-stimulate the neural net to further illicit intelligent thought. We believe there is no distinction between "conscious" and "subconscious" thought; rather subconscious thought is substantially more simple than conscious thought with a commensurate increase in speed of stimulus, and a decrease in control.

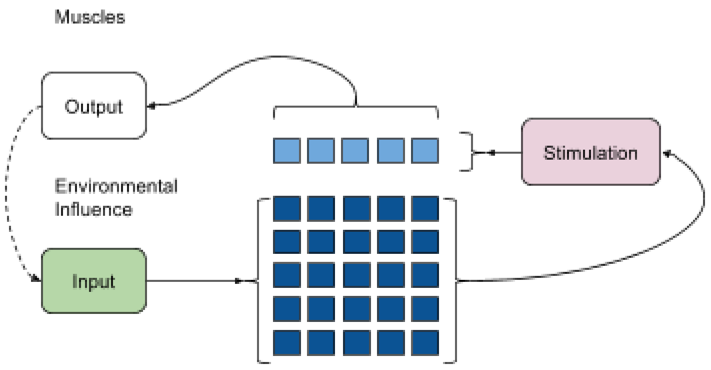


It is important to clarify that "thought" is simply a pattern of stimulation, and that "control of thought" is simply the ability to control where patterns of stimulation are initiated. Propagations of stimulus patterns are achieved by association,

[1] On Intelligence

one pattern of stimulus will stimulate other patterns of stimulus more strongly through a process of reinforcement learning.

Lastly, the brain can influence incoming sensory stimulation through activation of a muscle cells; this allows the brain's host (the body) to affect its environment. Nerve cells originating from discrete sections of the neural net are stimulated, which in turn stimulate muscle cells in a predictable fashion. Through proprioception, the outgoing signals can be refined accurately.



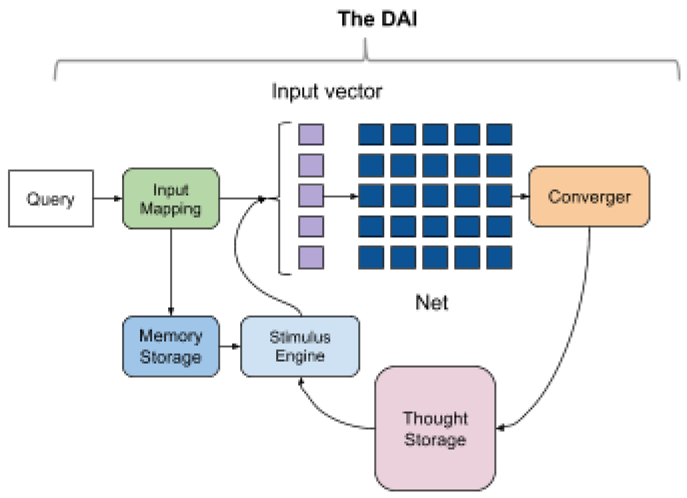
Summary

For a machine to emulate the brain a matrix of connections is required, with each connection requiring inherent variability and a discrete action potential. Input needs to be mapped discretely to the connections, and the stimulus will need to propagate across the net. The machine needs to be able to recognise patterns of stimulus, and in turn further form other patterns of stimulus through reinforced association. The machine may have the ability to influence input through outgoing stimulus. Lastly, the machine will require a very high neuron and neural connection count to be intelligent.

The DAI

A model for distributed artificial intelligence, DAI, is proposed, utilising the blockchain to store an abstraction of a neural net and coordinate nodes through an adaptive proof of intelligence protocol.

The DAI is comprised of an input mapping engine, an input vector, a neural net, a converging engine, a stimulus engine and two decentralised storage tables of memories and thoughts.



The following is the process of input computation performed by each node connected to the DAI:

1. The query is mapped to an input vector and broadcasted to the net.
2. The query is immediately content-addressed and stored as a memory, alongside its input vector.

    3. The input vector stimulates the net through pointer references.
    4. The net broadcasts the stimulus pattern to the converger, which is content-addressed and stored as a thought.
    5. The stimulus engine locates associated memories and their input vectors, and re-stimulates the net.

6. Associated memories are reported in each recursive cycle of (3, 4, 5) as the "answer" to the original query, and stored alongside the thought.
7. The DAI confirms the most correct association.

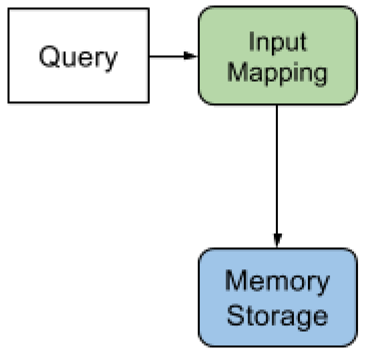
Through inherent neural variability, each node may offer different associated memories as the answer to the input query. As each memory is content-addressed, if more than two nodes report the same memory, they simply increment the count of the memory associated with the corresponding thought, and thus "confirm" each other.

Once a memory is confirmed, the DAI updates the blockchain with the model of the confirmed node, and rewards it with the block reward of DAI tokens. Similarly to the bitcoin protocol, nodes will try to work on the longest blockchain, and will discard shorter chains.

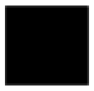
It is important to highlight that the DAI will only ever offer associated memories it has previously been exposed to. Through neural variability memories can have variations, but the DAI will never be capable of wholly original memories.

Input Mapping

As queries are submitted to the DAI, they are categorised, mapped to an input vector, and content-addressed.

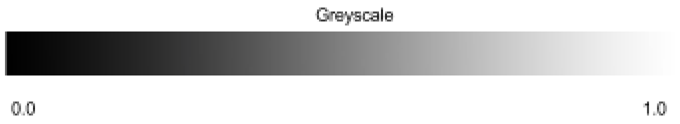


Take a single black pixel:

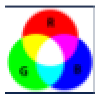


A black pixel

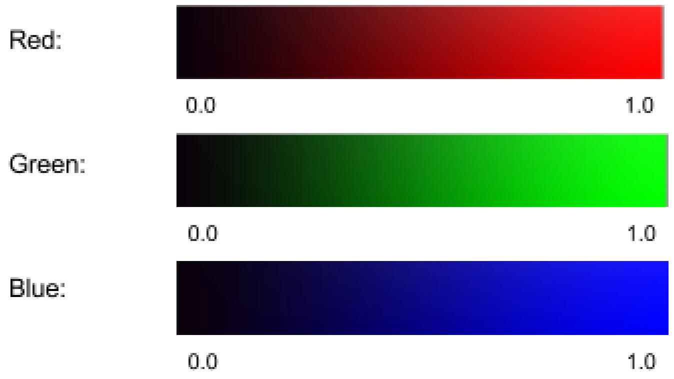
A single black pixel has 1 dimension and only 1 channel of data: whiteness. Pure black is 0.0 and pure white is 1.0, with greyscale a double of range [0.0 … 1.0]. The biological counterpart is the "rod" photoreceptor in the human retina, which is sensitive to light of a medium wavelength, but not colour.



A colour pixel would have a biological counterpart of the "cone" of which there are three variants that are responsive to short (blue), medium (green) and long (red) wavelengths of light. The corresponding three-channel data in red, green, and blue:



A colour pixel



Abstracting data channels in this manner can be continued across all data forms. Appendix A treats popular data types.

The query is processed to detect appropriate data types, then abstracted appropriately. The DAIInput object contains size, data type (greyscale, colour, audio etc), an array of input neurons which are sensitive to that data type, and corresponding stimulus values:

```
type DAIInput struct{
      Size int
      // total size of input

       type int
      // type of data

      arrayNeurons []
      // an array of mapped input neurons

       arrayStimulus []
      // an array of stimulus values
       }
```

For a black pixel, the neuron and stimulus array would be of size 1:

```
arrayNeuron[0] :
"0x56asdhg98fg2j5bas8as9ga8gna98ghasg"
arrayStimulus[0] : 0.0
```

The input is then stored inside a memory, which is persists the initiating query in decentralised storage. The DAIMemory type object can contain multiple DAIInput objects, which is useful when the DAI is submitted a query that contains multiple channels of data, such as a video (greyscale, colour, audio, text). The DAIMemory is content-addressed and stores the original query as opaque content data in decentralised storage:

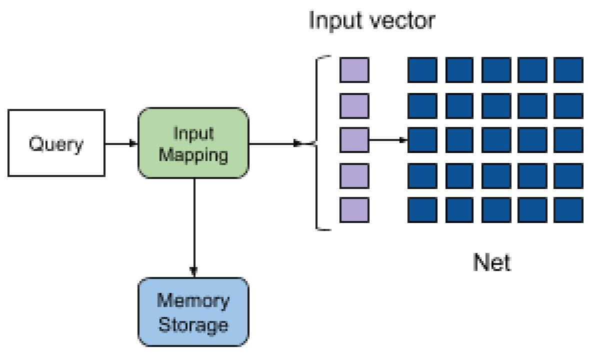
```
type DAIMemory struct{
      Hash Multihash
      // cryptographic hash of memory

       Size int
      // total size of memory

      inputVector []DAIInput
      // inputVector is an array of DAIInput
objects

       dataAddress IPFSAddress
      // opaque content data stored off-chain
      }
```

Content data (which may be very large) is stored off-chain in IPFS or an equivalent decentralised storage service. The address to the data is stored on-chain in the relevant memory object.

It is worthwhile to clarify that a query can only be mapped once and is uniquely stored in Memory Storage. Incoming queries that are already mapped and stored are discarded, and the stored memory is queried and its corresponding `inputVector` is submitted to the DAI instead.

DAI Neural Net

After mapping, the resultant input vector is used to stimulate the neural net.



Each neuron in the net needs to satisfy:

1. Contains an "address list" (pointer references to other neurons)
2. Address of the connected neuron
3. A Minimum Excitation Level (MEL), a float with range [0.0 … 1.0], that determines when the neuron at the address is excited.
4. A Connection Variability Factor (CVF), a float with range [0.0 … 1.0], that determines the variation permitted at that connection. 0.0 allows little variation, 1.0 allows universal variation.

The DAINeuron structure contains the type of neuron, the address of the neuron, persists the value of incoming stimulation as well as the ID of the query associated with the stimulus, and an address list:

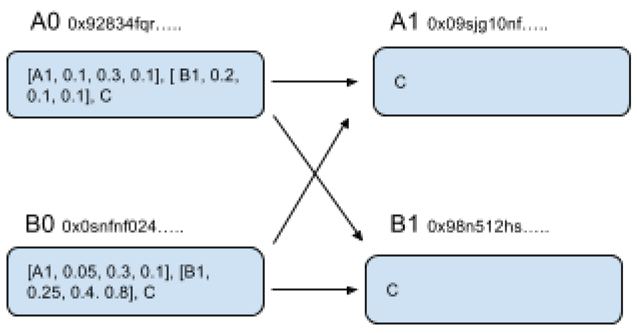
```
type DAINeuron struct {
        type int
       // type of neuron

       hash Multihash
       // cryptographic address of neuron

        stimulation double
        // value of incoming stimulation

        IDQuery Multihash
        // cryptographic hash of incoming query

        IDAnswer Multihash
        // cryptographic hash of associated
memory for learning

        addressList []DAINueralConnection
       // An array of DAI neural connections

        Size int
       // total size of addressList
       }
```

Note: the IDMemory parameter can be null, and is only used for active learning (discussed later).

The DAINeuralConnection object persists the structure of each outgoing neural connection. The size of this array may be artificially capped as it will exponentially increase the complexity of the network. The average neural connection account for human neurons is estimated to be 7000.

The DAINeuralConnection object stores outgoing neuron addresses, a MEL and a CVF double value.

```
type DAINeuralConnection struct {
        neuron Multihash
       // cryptographic address of the
connected neuron

        MEL double
       // value of the minimum level of
excitation

        CVF double
       // value of the variability factor
}
```


A0 0x92834fqr…..
[A1, 0.1, 0.3, 0.1], [B1, 0.2, 0.1, 0.1], C
A1 0x09sjg10nf…..
C
B0 0x0snfnf024…..
[A1, 0.05, 0.3, 0.1], [B1, 0.25, 0.4. 0.8], C
B1 0x98n512hs…..
C

Each neuron contains an address list, minimum levels of excitation, a stimulation factor and a variability factor

As each neuron is stimulated by the previous neuron (or input vector), it performs the following operation with the stimulation input, `stimulation`, for each neuron stored in its `addressList`:

1. Subtract the `MEL` from the `stimulation`.
2. If less than `MEL`, break, else
    a. Stimulate the neuron with `((stimulation - MEL), IDQuery)`
3. Converge with `(stimulation, IDQuery)`

The iteration function:

```
void neuron(){
  int i;
  for (i=0; addressList[i]< neuron.size; i++){
      propagate(addressList[i], stimulation,
IDQuery);
  }
converge(stimulation, IDQuery)
}
```

The propagation function:

```
void propagate(neuralConnection, stimulation,
IDQuery){
double excite = stimulation -
neuralConnection.MEL;

       if (excite > neuralConnection.MEL){
       stimulate(neuralConnection.neuron,
excite, IDQuery)
       }
}
```

The stimulation function:

```
void stimulate(neuron, inputStimulation,
IDQuery){
       neuron.stimulation = inputStimulation;
       neuron.IDQuery = IDQuery;
}
```

Propagation across the net is simply the lightweight action of addresses sending a double primitive to each other, together with the original ID of the query. Stimulation only occurs if the stimulation is higher than the connection's minimum excitation level. The biological counterpart of this is the action potential. Each stimulated neuron converges the pattern by reporting to the converger (a local address to the computing node).

After propagation is finished across the net, each neuron which was stimulated will have reported stimulation levels to the converger, and thus formed an `outputVector` which can be mapped to the original query. The `DAIOutputVector` object will have the following structure:

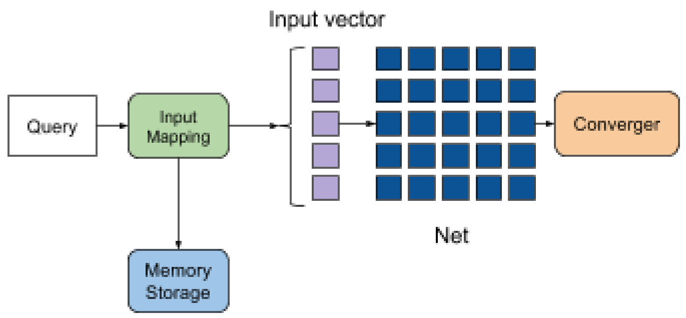
```
type DAIOutputVector struct {
        ID Multihash
       // cryptographic hash ID of the original
query
       Size int
       // total size of output

        outputVector []DAIOutput
       // a single DAIOutput object
}
```

This is a DAIOutput object, and contains two arrays of reporting neurons and their respective stimulus values:

```
type DAIOutput struct{
       arrayNeurons []
       // an array of reporting neurons

        arrayStimulus []
       // an array of reported stimulus values
}
```

Converger

The brain contains a natural ability to observe and recognise stimulation patterns and invoke memories. The DAI achieves this through collecting the `outputVector` from the neural net and uniquely marking the stimulation pattern, or "thought". When the stimulation pattern is observed, it is queried to return memories. If no thought exists, it is created and stored.



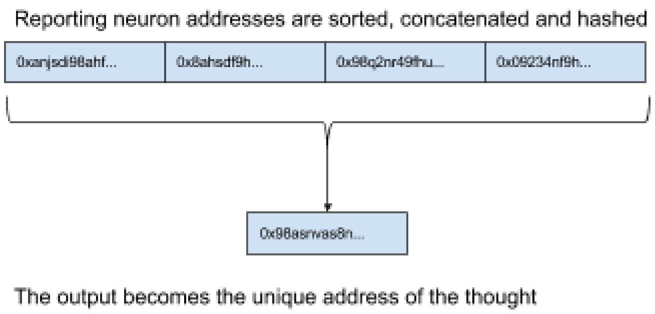
The converger then performs the following operation across the entire `outputVector`:

1. Sort and concatenate each reporting neuron address, then hash the result to form a single hash, `IDThought`
2. Query Thought Storage with the address to determine if the thought already exists,
    a. If the thought already exists, report the thought's memories
    b. If the thought does not exist, initiate and store the thought with:
        i. The cumulative total of reported stimulus, `s`, to form weight, `weight`
        ii. The thought's original query as the first memory

```
void thought(outputVector, IDQuery){
        int i;
        int weight;
        array addressArray;

        for (i=0; outputVector[i]<
outputVector.size; i++){
                addressArray[i] =
outputVector.arrayNuerons[i].address;
                weight =
outputVector.arrayStimulus[i].s + s;
        }
        IDThought = hash(addressArray);
        query(IDThought, weight, IDQuery);
}
```

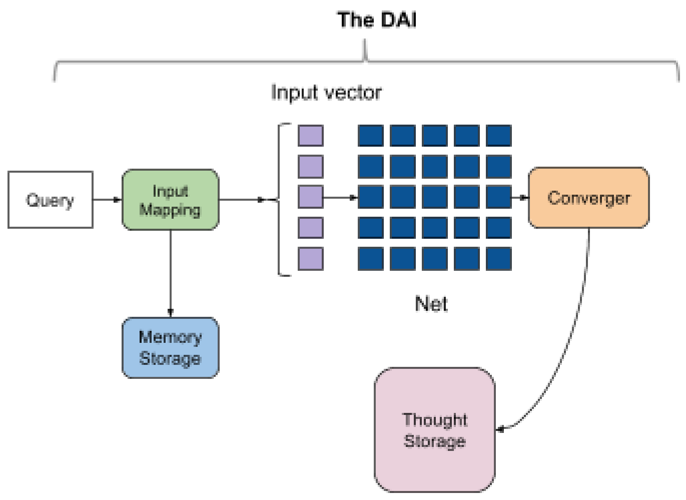
Here the outputVector is iterated over to determine the final hash and total weight. This is then sent to the query function which checks if the thought exists, before invoking memories:

```
void query(IDThought, weight, IDQuery){
        if(IDThought!){
        create(IDThought, weight, IDQuery)
        }
        remember(IDThought, weight)
}
```


Reporting neuron addresses are sorted, concatenated and hashed
0xanjsdi98ahf...
0x8ahsdf9h...
0x98q2nr49fhu...
0x09234nf9h...
0x98asnvas8n...
The output becomes the unique address of the thought

Thoughts & Remembering

When the human brain is stimulated, a stimulation pattern is produced, recognised and associated memories are invoked. Association is both natural “this seems like that” and learned “this means that”. In the DAI each stimulation pattern is mapped as a unique thought, and these thoughts invoke memories through both natural association and learned association. There is a hierarchical order in which associations are invoked.



Each thought object contains an array of associations. Each association contains a memory address, a natural value and learned value. The DAIThought object:

```
type DAIThought struct {
        IDThought Multihash;
       // cryptographic address of the thought

        associations []DAIAssociation;
       // associations is an object
}
```

The DAIAssociation object persists the associations related to the thought:

```
type DAIAssociation struct {
        DAIMemory Multihash;
       // cryptographic address of the
associated memory

        natural int;
       // represents the natural weighting of
the association

        learned int;
       // represents the learned weighting of
the association
}
```

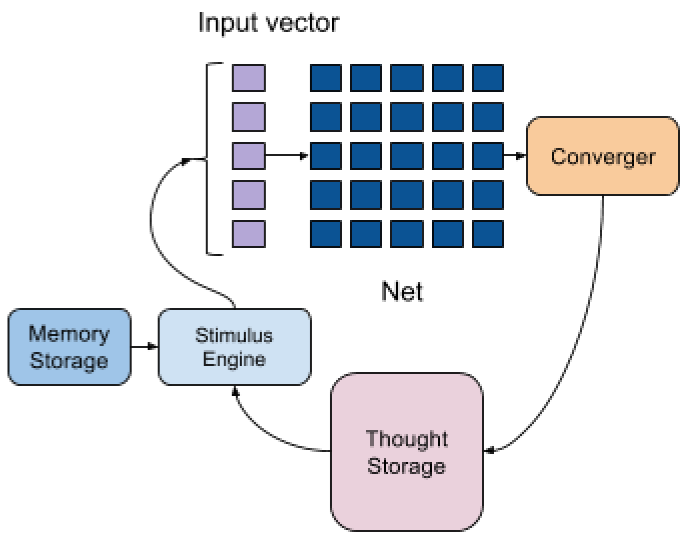
When a thought is queried, two memories are reported. The first is the memory linked to the association with the closest natural weighting to the query’s weighting (the addition of all stimulation factors), and the highest learned weighting:

```
array remember(IDThought, weight){

        memoryNatural MultiHash =
findClosest(IDThought.associations.natural);

        memoryLearned MultiHash =
findMaximum(IDThought.associations.learned);

        return[memoryNatural, memoryLearned];
}
```

The resulting returned memories (which may be the same), are then collected and suggested as answers to the original query. Memories are simply links to data objects stored in Memory Storage. This matches how the brain functions in that it simply returns associated memories from sensory input it has previously been exposed to.

Stimulation

When faced with sensory input, the brain's neural net is stimulated and a distinct stimulation pattern is observed and recognised. It then immediately invokes associated memories, and uses this to in turn stimulate the neural net again. This occurs continuously in a perpetual loop that we call "thinking".

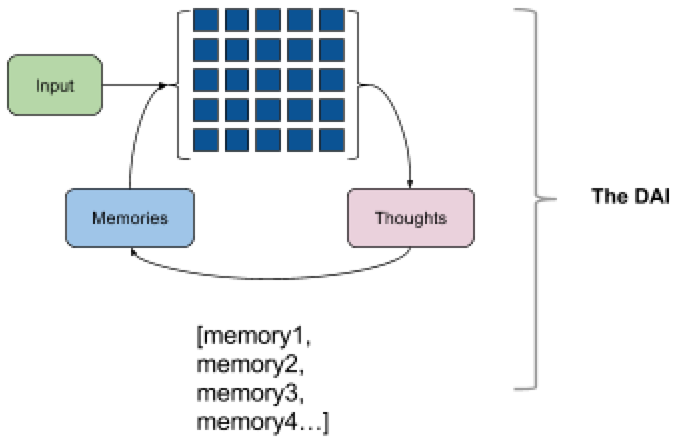


The DAI achieves this by reading the `inputVector` associated with any returned memories and stimulating the neural net again with the stored stimulus values.

```
void restimulate(IDMemory, IDQuery){
        int i;
        for (i=0; IDMemory.inputVector[i]<
IDMemory.size; i++){

stimulate(IDMemory.inputVector[i].neuron,
IDMemory.inputVector[i].stimulation, IDQuery)
        }
}
```

The DAI will then continuously loop and suggest associated memories to the original query. Due to inherent variability in the Connection Variability Factor, CVF, the memories returned will be different:

1. Variations in CVF not exceeding the MEL will cause different weightings for natural associations in each loop
2. Variations in CVF exceeding the MEL will cause different reporting neurons, and completely different thoughts



At this point, each returned memory is collected and displayed to the client who submitted the query. If the memory is rich media like an image, audio or video, this is retrieved from the decentralised off-chain storage. The client has the liberty to choose from the answers, or accept the DAI's suggestion of the most correct answer.

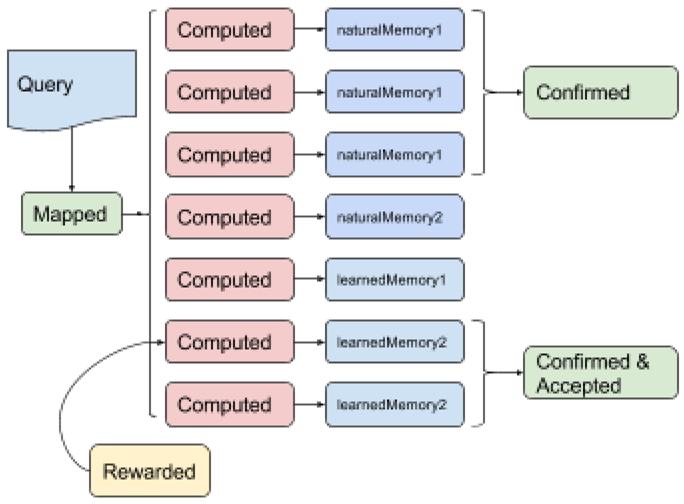
Confirmations

A large number of nodes can be part of the network and dedicate compute and storage resources. All nodes will work to compute the input and return memories that could be correct. The DAI will track the reported memories for each query and will prefer learned memories over natural memories. As soon as two nodes solve and suggest the same learned memory, the DAI assumes the confirmed memory is correct to be associated with that query and performs the following actions:

1. Updates the neural net's addressList, CVF & MEL values to that of the confirmed node's,
2. Updates Memory Storage with new DAIMemory objects,
3. Increases the learned weighting of the association of the confirmed memory and updates Thought Storage with new DAIThoughts and DAIAssociations, and
4. Awards the block reward.

If no learned memories are ever submitted, the DAI will continue to track and count the natural memories submitted from the nodes. This represents a "best guess" from the DAI. When a natural memory earns a much higher confirmation count and there is still no learned memory confirmed, the DAI will accept the natural memory as correct and execute the confirmation.

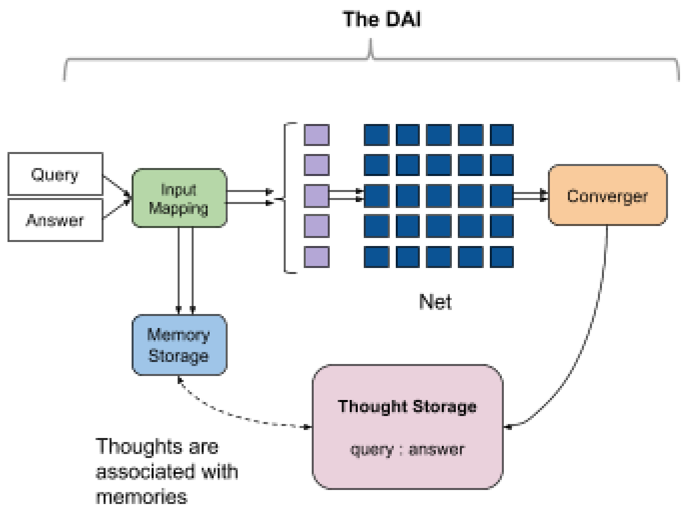
Confirmed and confirming nodes will then work to solve the next input query. All nodes will work to solve the longest chain and will discard shorter chains if they are behind.



Each time a node's reported memory is confirmed and accepted, the DAI updates the entire blockchain with that node's neural connection model and increases the strength of the connections that were used to compute the correct answer. This is done by tracking the hashed *ID* of the input that was passed through the neurons. In this way, only the neurons that were improved by computing that particular input are updated. The resulting changes are propagated across the network and all nodes benefit from the generational improvement.

Incentivised Training

The brain can learn faster if it is actively instructed to associate two stimulations. When first seeing a red colour, our brain is stimulated in a certain fashion, but we do not recognise it. We are then told the stimulation is “red”. From our previous exposure to words and language, we learn to associate a stimulation from red colours with the stimulations created by thinking of the word “red”. The DAI can mimic this with a simple adjustment by submitting the answer along with the query.



In this case, the two objects are instantiated as DAIMemory objects and cast on to input vectors. The DAI is activated with a DAIInputVector that contains both IDQuery (address of the query), as well as the IDAnswer (address of the answer) that it will be associated with. The DAI then reports to the converger and a thought is created alongside the relevant weight:

```
void thought(outputVector, IDQuery, IDAnswer){
       int i;
       int weight;
       array addressArray;

       for (i=0; outputVector[i]<
outputVector.size; i++){
              addressArray[i] =
outputVector.arrayNeurons[i].address;
              weight =
outputVector.arrayStimulus[i].s + s;
       }
       IDThought = hash(addressArray);
       learn(IDThought, weight, IDQuery,
IDMemory);
}
```

The learning function creates the DAIThought object and a DAIAssociation object with a non-zero learned parameter value:

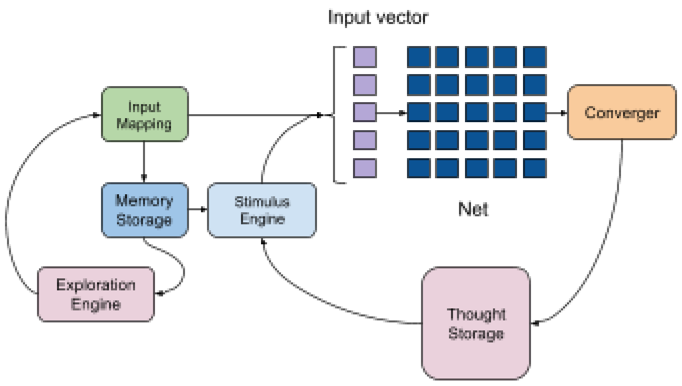
```
void learn(IDThought, weight, IDQuery,
IDAnswer){
       create(IDThought, IDQuery, weight,
IDAnswer, 1)
}
```

Once this occurs the association is created and is used to re-stimulate the DAI again to reinforce the connection and improve the learned association. Nodes that broadcast memories from a training evolution have a very high chance of being confirmed due to submitting memories with non-zero learned parameters. This will incentivise the nodes to contribute training data to maximise block rewards. As a result, the DAI will be grow towards intelligence.

Exploration

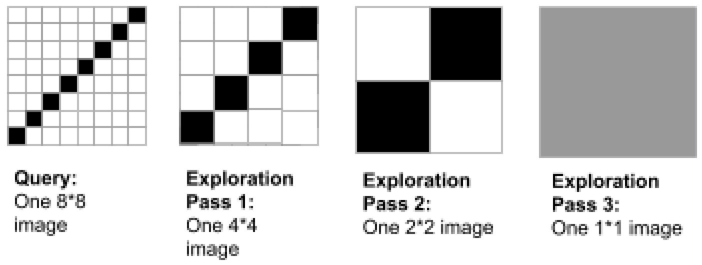
An important part of learning is our brain's ability to explore our memories in various fidelities such as through thinking, imagination and dreaming. This is helpful in exposing our brain's neural net to various simulated sensory stimulus and reinforcing learned associations. We also find the most clarity in thought when simplifying sensory input and taking a bigger perspective.

The DAI will achieve this through an exploration engine.



Simplification

When a real-life query (real stimulus) is submitted to the DAI, it is immediately mapped and stored, before being submitted to the neural net. The Exploration Engine takes the query data, and reduces the data fidelity by half and submits the simplified query again, but with the simplified data as an answer to the query.

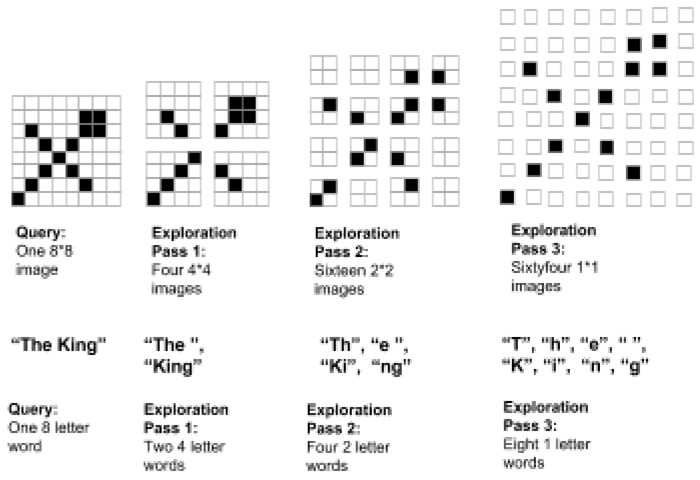


This will train the net to form associations between complex queries and their simplified versions, as often the simplest answer is the most correct.

Dissection

The DAI can also explore the data it is given by dissecting it in half if it has a single dimension (text string, audio), or quarter if it has two dimensions (image), and immediately submitting all pieces of the data as separate queries into the DAI for processing. This continues until the query data has been dissected down to the lowest data unit.

Illustrated for an original 8*8 pixel image with a diagonal stroke, and a text string:



Through exploration, the neural net is given the chance to be exposed to exponentially more data than without, and is given the chance to form learned associations between naturally related stimulation patterns and memories.

Variability

MEL & CVF

The Minimum Excitation Level, MEL, at each neuron is randomly generated, but has a ceiling of the Connection Variability Factor (CVF) for each connection. A new neural connection has a CVF of 1.0, so the activation level for that neural connection will exist in the range [0.0 … 1.0]. When the strength of that connection is improved, the CVF decreases, asymptoting to 0.0, which in turn reduces the variability of that neural connection. If the CVF of a neural connection is 0.05, then the MEL will exhibit a natural variation of 5% for each query, which will in turn influence the final weighting of the thought.

The DAI comprised of 2*2 neurons is exposed to a 2*2 pixel image and the nodes return four possible output vectors. If the excitation received by a neuron, *e,* does not exceed the random minimum excitation level for that neural connection, then the neuron can never stimulate the converging engine or consequential neurons contained in its address list. The four possible output vectors:

$\{[stimulated\ neuron : \ stimulation]\}$
$\{[A0 : \ e_{A0}],\ [A1 : \ e_{A1}],\ [B1 : \ e_{B1}]\}$
$\{[A0 : \ e_{A0}],\ [A1 : \ e_{A1}]\}$
$\{[A0 : \ e_{A0}],\ [B1 : \ e_{B1}]\}$
$\{[A0 : \ e_{A0}]\}$

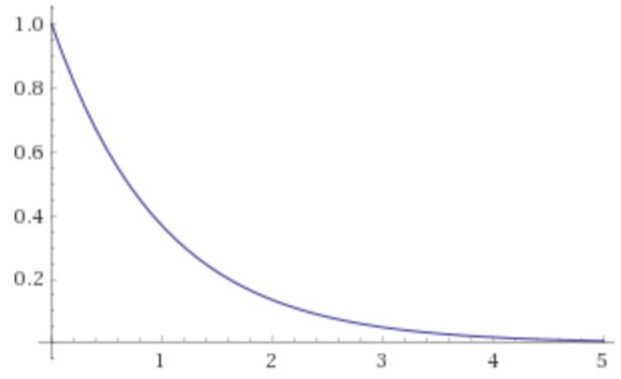
These four possible output vectors are equally valid to return a memory, so each result is hashed to create a thought address, then the excitation float is summed to determine weighting:

$\{id,\ [A0 : \ e_{A0}],\ [A1 : \ e_{A1}],\ [B1 : \ e_{B1}]\}$
$\rightarrow \{hash(A0,\ A1,\ B1) : (\ e_{A0} + e_{A1} + e_{B1})\}$
$\{id,\ [A0 : \ e_{A0}],\ [A1 : \ e_{A1}]\} \rightarrow \{hash(A0,\ A1) : (e_{A0} + e_{A1})\}$
$\{id,\ [A0 : \ e_{A0}],\ [B1 : \ e_{B1}]\} \rightarrow \{hash(A0,\ B1) : (e_{A0} + e_{B1})\}$
$\{id,\ [A0 : \ e_{A0}]\} \rightarrow \{hash(A0) : (e_{A0})\}$

When querying Thought Storage, the addresses will return the closest possible natural memories, and the most likely learned memories. The nodes will work to solve as fast as possible and that the nodes which can submit memories faster are more likely to be confirmed by other nodes and earn the block reward. Each time the DAI updates the neural net with the CVF and MEL values of the correct node, the CVF values for each neuron is reduced through the inverse exponential function:

$$factor = e^{-x}$$

where x is the number of times that the neural connection has been validated as part of a correct result.



Here the forcing function is speed; the faster the result is returned, the more likely it will be confirmed, the more times the corresponding neural connections are updated and variability is reduced from 1.0 down to 0.0. In this manner the DAI will be most likely to return the correct memories by pursuing the fastest possible path.

Neural Exploration

For further variability, each node can randomly terminate a neural connection which has a variability factor close to 1.0, and re-form it with another completely different neuron. This mimics biology and allows learning and growth. This is completed by parsing in another neuron address selected from the DAI net and giving it a starting CVF of 1.0, with a random MEL. Strong connections with low CVFs will not be removed.

CVF Decay Function

In order to prevent a previously highly confirmed connection becoming irrelevant as the DAI grows, an inverse decay function seeks to return low CVFs back to high values to encourage neural exploration. The decay function will be coupled to block count as the DAI grows.

Economic Model

The economic model cascades across the four fundamental parts of the DAI: Input Mapping, Thought Generation, Intelligent Confirmations and Client Fees.

Input Mapping

Incoming queries need to be mapped, stored (if not already) and submitted to the DAI. The work required for each query:

1. Hash the query, and check if it exists
    a. If exists, submit existing DAIMemory object
    b. If none, create a new DAIMemory:
        i. Separate data channels
        ii. Size and map data channels to an input vector
        iii. Submit to DAI

Input mapping also includes the necessary task of referencing the growth of new neurons, which is dependent on the performance of the DAI. This can be performed through consensus-based rules.

DAI Tokens are issued for each successful query processed and submitted to the DAI, and token reward amount is in proportion to the size of the memPool, incentivising nodes to work to process queries if there is a backlog. Nodes can also perform the exploration function with idle hash-power, which incentivises continual DAI growth.

Thought Generation

Input vectors submitted to the DAI are propagated across neurons and maths is performed at each neuron. Neurons can be allocated in memory “just-in-time” and de-allocated as soon as they are used, as propagation is solely one-way. Each neuron that is stimulated is collected by the converger. After propagation is finished, (or throughout), the stimulation pattern is hashed to form a thought and associations retrieved. Associations are used to submit natural and learned memories.

Through consensus-based rules, the nodes which are the most confirmed for each query are deemed to be correct and issued the block reward. The block reward is always proportional to the size of the query, so that the most complex queries are sufficiently rewarded.

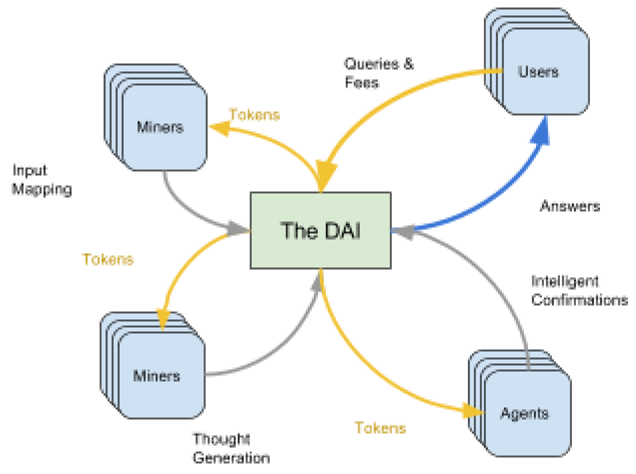
Intelligent Confirmations

The DAI returns an array of linked memories to each query submitted, and will confirm the most learned, or the closest natural memory as correct. A vital part of learning is thus intelligently selecting the correct linked memory, and this can easily be performed by existing agents, such as humans and API-based services.

Agents can thus earn DAI tokens by participating in the confirmation loop and help accelerate DAI learning and utility. An example of such would be the correct label for an image query. A number of labels are returned as valid associations and a human agent could easily select the most correct one to be confirmed by the DAI. If the selection is confirmed, the agent is rewarded. This complies with the consensus-based proof of intelligence protocol.

Client Fees

The value of the token can also be driven by the revenue earned by the DAI model in answering queries, functioning effectively as distributed AI-as-a-Service. Users purchase tokens on the market which are consumed when they field queries to the DAI - the higher the fee the more it is prioritised in the memPool. This will provider a counter to the act of mining in which new tokens are minted and issued to miners and agents. The economics of supply and demand will ensure a match between the DAI and its utility.

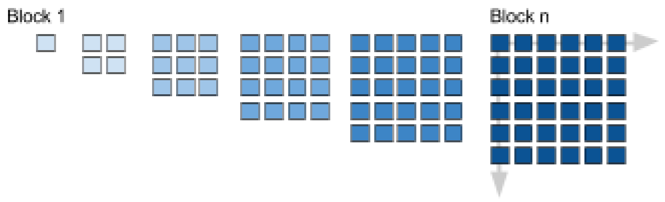


Scalability

Growing the DAI

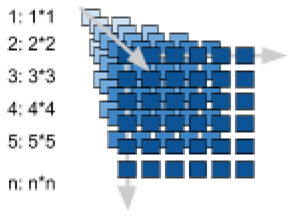
The human brain has 100bn neurons and 100tn neural connections and so is a reasonable target to have with appropriate resources. To facilitate this level of growth, the DAI will grow with increased node resources; more resources will result in larger growth.

The coupling function to expand the DAI will be based on number of confirmations. If an input receives thousands of confirmations to an input, valid or not, then it is clearly over-resourced to the net size and capacity to grow exists. This is performed by adding another layer, both deep and high, to the existing DAI net, and adding the block:

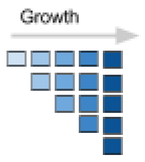


The DAI growth is coupled to network size, growing in response to node resources

To prevent regression each block is solely additive, forming a dimensionally stacked net from the origin which preserves the input vector, and the already learned layers. In this way the DAI is a comprised of *n* deep neural nets, rather than a single net. As each layer is added, it forms connections with previous neurons.



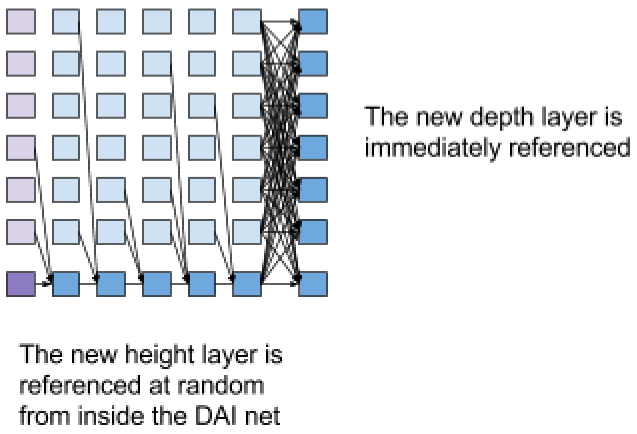
The DAI net grows from the origin



The input vector scales, but preserves existing neurons

If the DAI is passed a small 1*1 image again, the fastest way to return a memory is still to employ the original 1*1 block and return the answer off that.

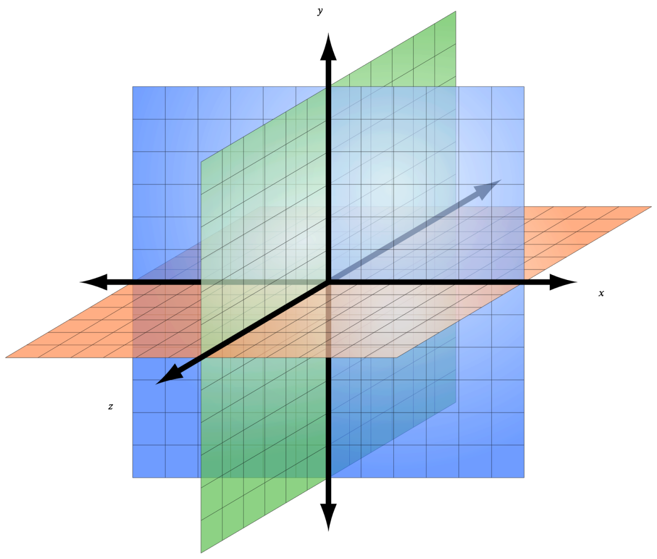
Each new block is a carbon copy of the previous block, except with the additions of the new layers which add to the depth and height of the DAI net. The new addresses of the neurons are added to the address lists of the preceding layer and each given a starting variability of 1.0.


The new depth layer is immediately referenced
The new height layer is referenced at random from inside the DAI net

Referenced Neural Location

The human brain receives sensory input at specific locations. Visual information from the retina is passed to the visual cortex located at the back of the head. The same is respectively true of other sensory input and output.

To mirror this, the DAI neuron addresses can be cast from a 3D cartesian coordinate system, (x, y, z), and octets can be allocated to each new data type that the DAI receives.



Two dimensional data (such as visual) will require three dimensions to scale in (2 dimensions to cast the input, a third dimension to grow the net), whilst single dimension data (audio, text) can scale in two dimensions.

An address of a neuron is thus established as:

```
multihash castNeuron(x, y, z){
return hash(x, y, z);
}
```

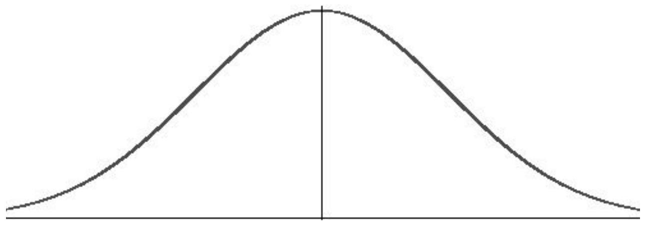
A neuron table tracks the coordinates that have already been issued to prevent overwrite, and to ensure that neurons are correctly located in octets reserved for their data type.

As an example the first data types exposed to the DAI can be grayscale images from 1*1 pixels to 8*8 pixels. Greyscale has a single channel of data, and an image has two dimensions, so the DAI will grow from a 1*1*1 block to a 8*8*8 block.

The first input neuron is established at [0, 0, 0], with total DAI size of 1*1*1. After the first block the DAI is grown to a 1*1 layer and a 2*2*2 layer. The second layer contains four input neurons and four net neurons and is cast onto:

[0, 0, 0] - [0, 1, 0] - [0, 0, 1] -[0, 1, 1]
[1, 0, 0] - [1, 1, 0] - [1, 0, 1] -[1, 1, 1]

As the DAI is exposed to various data types, input neurons are established from the centroid, before incrementing outwards. Neurons in the subsequent depth layers are randomly connected with nearby neurons with a connection distance governed by the normal probability distribution:



$$P_{distance} = \frac{1}{\sqrt{2\pi}\sigma} * e^{\frac{x^2}{2\sigma^2}}$$

This will ensure that neurons are more likely to be connected to nearby neurons which process similar inputs. If a neuron exists at the random location (incremented from the starting neuron’s location), the address of that neuron is added to the address list. If it does not exist, it is created. This will ensure continual growth in response to larger data inputs.

Conclusion

A method for storing an AI model on a blockchain was discussed. The method employs mapping and storing each query and casting it to an input vector as a “memory” object. The input vector stimulates a net of connected neurons by a propagating stimulus values. Each neuron has inherent variability and uses an “action potential”. The subsequent pattern of stimulation is hashed for unique identification as a “thought”, and associations are retrieved. Associations can be learned or natural, and link to “memories”. Memories are then submitted as potential answers for the query, and are confirmed as correct by a consensus-based proof of intelligence protocol.

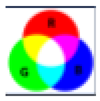
Nodes are incentivised to work for a cryptographically secure token, the DAI token, which is awarded for input mapping, thought generation and intelligent confirmations. Tokens are continually minted, but are burnt by clients utilising the DAI for intelligence as “query fees”. This establishes a supply and demand model and a source of revenue.

Importantly, nodes who perform work that is confirmed through the consensus-based proof of intelligence protocol are used to continually update the blockchain of memories, neurons and thoughts. In this way the DAI continually grows in the pursuit of intelligence. The resulting intelligence is decentralised, resilient to attacks and is owned by all, to be used by all.

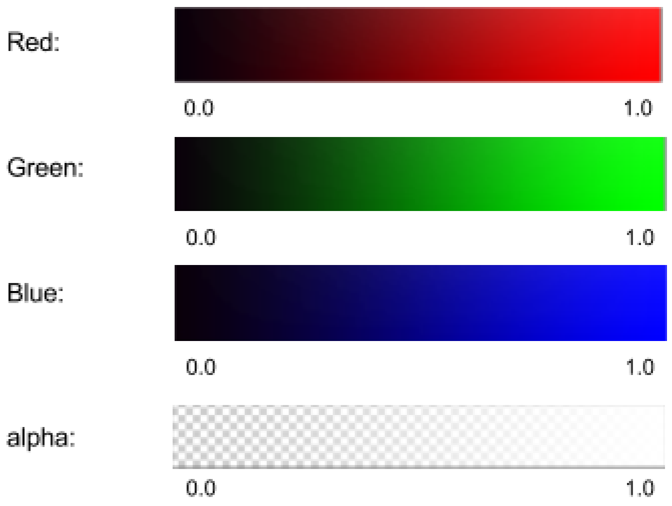
Appendix A - Data Models

Visual

Images are two-dimensional, with up to four channels of data:

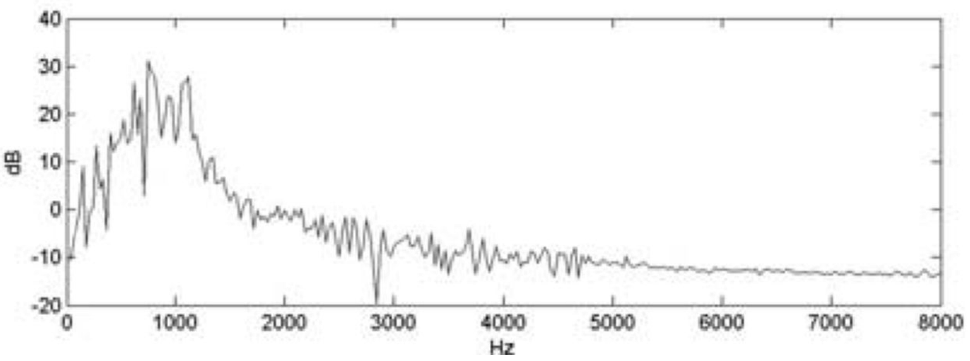


A colour pixel



Audio

Audio is two-dimensional, with a single channel of data (volume). Stereo audio would have two channels of data. An audio stream would be deconstructed down to its sample rate, and the DAIMemory object would store an array of DAInputVectors, each corresponding to the sampling.



Video

Video would be deconstructed to be comprised of an image stream and an audio stream, both with a sample rate. This would make it two-dimensional, but with an array of DAIInputVectors.

Text

Text strings can be stored as a single dimension, with one channel of data. Each character is represented it the range [0.0 … 1.0].