OpenBrain: Massively Asynchronous Neurocomputation

William H. Guss
Machine Learning at Berkeley
2650 Durant Ave, Berkeley CA, 94720
wguss@ml.berkeley.edu

Mike Zhong
Machine Learning at Berkeley
Berkeley CA, 94720
lol@gmail.com

Phillip Kuznetsov
Machine Learning at Berkeley
Berkeley CA, 94720
philkuz@ml.berkeley.edu

Jacky Liang
Machine Learning at Berkeley
Berkeley CA, 94720
jackyliang@berkeley.edu

Abstract

In this paper we introduce a new framework and philosophy for recurrent neurocomputation. By requiring that all neurons act asynchrynously and independently, we introduce a new metric for evaluating the universal intelligence of continuous time agents. We proved representation and universal approximation results in this context which lead to a set of learning rules in the spirt of John Conway's game of life. Finally evaluate this framework against different intelligent agent algorithms by implementing an approximate universal intelligence measure for agents embedded in turing computable environments in Minecraft, BF, and a variety of other reference machines.

1 Introduction

Most standard neural network models depart from the biological neuron in a number of ways. Our asynchronous continuous-time model draws inspiration from physiological and biological principles for neuron dynamics.

In a biological neuron, there are leak Na+ and K+ current channels that constantly push the neuron's voltage towards its equilibrium voltage. We have included an exponential decay term to model this homeostatic tendency towards equilibrium. Likewise, we have also loosely modelled the refractory period, the period immediately after an action period in which a neuron cannot send a signal, instead of having resetting the voltage of a neuron immediately back to zero.

There are many features of a biological neuron that we have chosen not to implement (most notably the dynamics of a neuron's voltage during a action potential, by implementing a Hodgkins-Huxley model (see wikipedia)) in that there is a trade-off between having a model that is more biological realisticity and one that is mathematically and computationally simple. There are projects like BlueBrain that aim for the most biologically realistic model, and a plethora of projects (deep learning, etc.) that are vaguely biologically inspired but have been massively implemented and successful at a small class of tasks. Our hope is by striving for a balance between the two ideals by implementing a system that is both biologically inspired and computationally simple, we would be able to make a massive implementation that succeeds in a wide variety of tasks that require both creativity and intelligence.

2 Asynchronous Neurocomputation

2.1 The Core Framework

We begin by giving basic abstractions on neurons.

Definition 2.1.1. *A **neuron** $n \in N$ is defined by*

- *a voltage $V_n(t)$*
- *a decay time τ_n*
- *a refactory period ρ_n*
- *a voltaic threshold θ_n*

Definition 2.1.2. *A **connection** $c \in C$ is a tuple $(n_i, n_j, w_{ij}) \in N \times N \times \mathbb{R}$ where n_i is the **anterior neuron**, n_j is the **posterior neuron**, and w_ij is the standard synaptic weight.*

For a neuron n, we denote the set of anterior neurons A_n and the dendritic connections, C_n^a. In the same light we will use the notations P_n and C_n^p to denote the sets of posterior neurons and posterior connections for n respectively.

Because it is impossible to implement time as a continuous parameter on any computer, we introduce the time-step, Δt, as a numerical simulation step-size in time. The conversion between discrete algorithm iterations, with $k = 0, 1, 2, ...$ and continuous time t is $V(t_k) = V(k\Delta t) = V[k]$. As computational power permits, a smaller Δt gives a more accurate simulation.

Definition 2.1.3. *We say that a neuron n experiences **voltage decay** so that for all k,*

$$V_n[k+1] \leftarrow V_n[k]e^{-\Delta t/\tau}. \tag{2.1.1}$$

so that unless it obtains voltage from anterior neurons' firing, its voltage will decay exponentially to 0.

Definition 2.1.4. *A neuron n is said to **fire** if it is not in its refractory period and $V_n(t_k) = V_n[k] > \theta_n$. Then for all $m \in P_n$,*

$$V_m[k+1] += w_{nm}\sigma(V_n[k]); \tag{2.1.2}$$

*that is, voltage is propagated to the posterior neurons. Immediately after neuron n fires, it enters a **refractory period** until time $t_k + \rho_n$, or iteration $k + \frac{\rho_n}{\Delta t}$.*

Combining (2.1.2) and (2.1.1) we get that for a neuron m at time $k+1$

$$V_m[k+1] = V_m[k]e^{-\Delta t/\tau} + \sum_{n \in A'_m} w_{nm}\sigma(V_n[k]) \tag{2.1.3}$$

such that A'_m is the set of anterior neurons which fired at time k.

2.2 Continuous Time Universal Intelligence Measure

In alignment with the philosophy which predicates OpenBrain, we wish to extend the evaluation of our algorithm well beyond its performance in supervised learning tasks; that is, what can be said about the intelligence of OpenBrain as an agent in an environment? A metric more condusive to implementing general intelligence is needed. The answer will motivate an important discussion of representation theory annd learning rules.

2.3 Universal Intelligence

A well established[1, 3, 4] machine intelligence measure is the Universal Intelligence Measure proposed by Legg and Hutter [2]. Drawing from a large amount of disparate literature on the subject they develop a consise definition of the intelligence of an *agent* in an *environment*.

Environment-agent interaction is defined with respect to an observation space, $\mathcal{O}$, and an action space, $\mathcal{A}$, both of which consist of abstract symbols. The perception space, $\mathcal{P} \subset \mathcal{O} \times \mathbb{R}$, is the combination of observations and rewards.

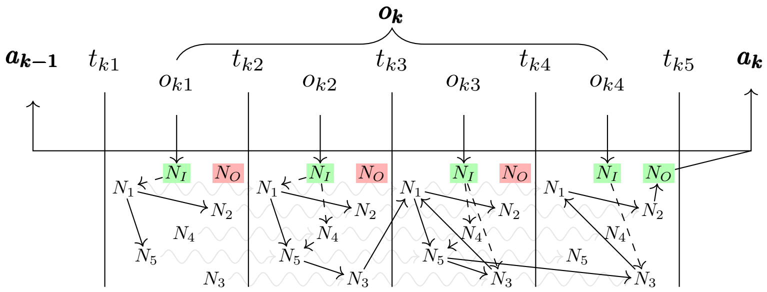

Figure 1: The diagram of sub-observation neural interaction for π_O.

Definition 2.3.1. *An* ***environment*** μ *is a probability measure, specifically defining*

$$\mu(o_k r_k | o_1 r_1 a_1 \dots o_{k-1} r_{k-1} a_{k-1}), \tag{2.3.1}$$

the probability of observing $o_k r_k \in \mathcal{P}$ *given a history, a string* $o_1 \dots a_{k-1} \in \bigtimes_{i=1}^{k} \mathcal{P} \times \mathcal{A}$.

In the same light the agent definition is given.

Definition 2.3.2. *An* ***agent*** π *is a probability measure, giving*

$$\pi(a_k | o_1 r_1 a_1 \dots a_{k-1} o_k r_k) \tag{2.3.2}$$

the probability of the possible action a_k *being enacted by* π *being the environment-agent interaction history.*

Having defined the basic framework, [2] gives a definition for Universal Intelligence. Let E be the space of all turing complete reward environments, and $K : E \to \mathbb{R}$ be the Kolmogorov complexity of an environment. This complexity is calculated with respect to the length of the string with which a reference machine $\mathcal{U}$ generates the environment.

Definition 2.3.3. *If* π *is an agent then we say that the* ***universal intelligence*** *of* π *is*

$$\Upsilon(\pi) = \sum_{\mu \in E} 2^{-K(\mu)} V_\mu^\pi \tag{2.3.3}$$

where V_μ^π *is the expected reward of the agent in* μ,

$$V_\mu^\pi = \mathbb{E}\left(\sum_{i=1}^{\infty} r_i \right) \leqslant 1. \tag{2.3.4}$$

The definition is satisfactory for agents which act synchronously with their environments; that is, the environment waits for the agent to act before giving a new observation. Therefore in Hutter's sense, the framework of [2] describes an agent π embedded in μ.

Despite this, the environments which we normally consider an intelligent agent to 'act well' in are often chaotic and operate with noise which is temporally independent from the agent-environment interaction itself.

In order to integrate OpenBrain with this framework, we will propose a continuous time universal intelligence measure.

2.4 Continuous Time Intelligence

To make a continuous time intelligence measure which is compatible with agents who act instantaneously within an environment, we will define a completed perception space.

Since different agents

Definition 2.4.1. *Given an environment* μ *with an associated perception space* $\mathcal{P}$ *we define the* ***completion*** $\tilde{\mathcal{P}}$ *with respect to the admissible sequences of observations in* μ*; that is,*

$$\tilde{\mathcal{P}} = \bigsqcup_{k=1}^{\infty} \mathcal{P}^k. \tag{2.4.1}$$

2.5 Universal Approximation and Representation

Conjecture 2.5.1. *Given a sequence of symbols $a_1r_1o_2a_2r_2$... there exists an agent π functioning according to some OpenBrain O such that the probability of taking an action a_k given $o_1r_1a_1$... is 1. Formally,*

$$\exists\pi_O : P[\pi_O(a_k|o_1r_1a_1...)] = 1$$

But this particular agent might not necessarily work well in other environments; it can have a low overall intelligence score. The ideal Openbrain agent would take the same actions as Hutter's AIXI in every environment μ, that is, $\forall\mu,\ \pi_o = \pi^*$. However, due to the finite nature of this agent, it is unreasonable to expect a single openbrain agent to match AIXI in every possible environment at every time step. This motivates a learning nature in which the Openbrain agent makes mistakes in the beginning, but slowly converges to Hutter's AIXI, and starts maximising the sum of expected rewards after some time step t_o.

Conjecture 2.5.2. *$\exists\pi_o$ such that $\forall\mu\ \exists t_o$ s.t. $V_\mu^{\pi_o}[t_o] = V_\mu^*[t_o]$, where $V_\mu^\pi[t_o]$ is the expected sum of rewards for agent π after time step t_o*

This is a nice property to have, but this conjecture falls apart in environments that are chaotic or have absorbing states. For example, consider a simple MDP with an absorbing state neighbouring the initial state. If the agent accidentally steps into the absorbing state, then by definition it is stuck there forever, and it can never get good rewards. Since every learning agent is bound to make mistakes at some point, we limit our environments to those where it is possible to recover from mistakes.

Definition 2.5.3. *An environment μ is* ***recoverable*** *iff for every state $s, \epsilon > 0, \exists t_o > 0\ s.t.\ |V_\mu^*[t_o] - V_{\mu,s}^*[t_o]| < \epsilon$, where $V_{\mu,s}^*$ is the expected sum of rewards starting from state s and acting optimally*

Conjecture 2.5.4. *There is a π_o such that for all recoverable environments $\mu, \epsilon > 0$, $\exists t_o$ such that $|V_\mu^{\pi_o}[t_o] - V_\mu^*[t_o]| < \epsilon$*

2.6 Multiprocess Turing Completeness [Omitted]

3 Asynchronous Signal Error Backpropagation

In this section we introduce an asynchronous signal error backpropagation method, as a general framework for updating parameters using the policy gradient. In traditional supervised learning, neural networks propagate an error gradient to almost every neuron on every layer. This method is adventageous since it it is easily optimized using GPUs and tensor multiplication, but measures must be taken to introduce sparsity and prevent overfitting such as dropout and ReLU activations.

Our framework immediately invokes sparsity of both neuronal connections and neuronal firings and therefore would perform suboptimally by such methods. Fortunately our framework can take advantage of chain rule as a graph transition process. Furthermore we wish to optimize the networks ability to produce an output a vector y at time $t_0 + t_e$ given that at time t_0 the network was fed some x.

As is standard with recurrent neural networks we define a method to unroll the neural activity of an OpenBrain between two points in time.

Definition 3.0.1. *We say that neural activity of an openbrain O over time can be* ***unrolled*** *as a function $\mathfrak{U}_O(a, b)$ so that the following diagram commutes*

$$\begin{tikzcd}
 & C(a) \arrow[r, "V_N(a)"] & \mathbb{R}^{|N|} \arrow[d, squiggly, "D_N(a)"] & \\
\bigtimes_{t\in[a,b]_{\Delta t}} C(t) \arrow[ur, "\pi_a"] \arrow[r, dotted, "\pi_{\dots}"] \arrow[dr, "\pi_b"] \arrow[drrr, bend right, "\mathfrak{U}_O(a,b)"'] & \vdots \arrow[r, dotted, "V_N(\dots)"] & \vdots \arrow[d, squiggly, "D_N(b-\Delta t)"] & \\
 & C(b) \arrow[r, "V_N(b)"] & \mathbb{R}^{|N|} \arrow[r, "\pi_{N_O}"] & \mathbb{R}^{|N_O|}
\end{tikzcd}$$

where $C(t)$ is the set of connections at time t, $D_N(t - \Delta t) : \mathbb{R}^{|N|} \to \mathbb{R}^{|N|}$ is the voltage decay from one timestep to the next, and π_t is the cannonical projection. We will sometimes use the shorthand $\mathfrak{U}_O(I)$.

Lastly, we must define a lost function to validate the activity of the openbrain agent against some desired output.

Definition 3.0.2. *Select the some neurons N_I, N_O for an openbrain O. Then given a dataset D of datapairs (x, y) where $x \in \mathbb{R}^{|N_I|}, y \in \mathbb{R}^{|N_O|}$, we define the **loss** of O on D as*

$$\begin{aligned} V_{N_I}(t_0) &:= x \\ E(\tau, y) &:= \frac{1}{2}\left\|\mathfrak{U}_O(I)\big|_\tau - y\right\|^2 \\ J(I) &= \int_I E(\tau, y) e^{\frac{-(\tau - t_0)^2}{|I|^2}}\, d\tau \end{aligned} \tag{3.0.1}$$

*where $(x, y) \in D$ and $[t_0, t_0 + t_e] = I$ is the **evaluation interval**.*

Learning is then defined as the optimization of the error function over the evaluation interval with respect to the connection parameters which lead to the final behaviour. One might optimize learning so that the most optimal behaviour minimizes the evaluation interval by letting $t_e \to 0$ as $t_0 \to \infty$, but we hold t_e constant in the initial formulation.

3.1 Signal Derivation

Observing Figure 1, it is intuitive that neural dynamics are sparse and moreover that error signals should be backpropagated along the very transitions that lead to the actions themselves. Not surprisingly the derivation of the gradient with respect to connection strength at a fixed time τ is similar to that of standard feed forward networks with a few caveats.

The gradient of J integrated over time is simply

$$D_{c_{nm}}[J] = \int_I D_{c_{nm}} E(\tau, y) \cdot e^{\frac{-(t - t_0)^2}{|I|^2}}\, d\tau. \tag{3.1.1}$$

For simplicity we will just calculuate the Frechet derivative of E w.r.t c_{nm} at a timestep τ and then worry about integration later. Using the unrolled commutative diagram for $\mathfrak{U}_O$ we get the following equivalent diagram by the universal property of tensor products

$$\begin{tikzcd} C(\tau) \arrow[r, "\pi_{nm}"] \arrow[rrr, bend left, "\mathfrak{U}_O(I)\big|_\tau"] & \mathbb{R} \arrow[r, hook, "V^\gamma_{N_O}(\tau)"] & \bigotimes\limits_{\substack{k=(k_1,\ldots,k_q)\\ k \in \text{path}(\gamma)}} \mathbb{R}^{|N_O|} \arrow[r, "T"] & \mathbb{R}^{|N_O|} \arrow[r, "J'(\tau)"] & \mathbb{R} \end{tikzcd}$$

where T is a linear map, path(γ) is the path $c_{nm}(\tau) \to n \in N_O$ through the neural activity diagram, and $V^\gamma_{N_O}$ is a function which parameterizes the voltage of the output neurons at time $t_0 + t_e$ with respect to $c_{nm}(\tau)$.

This formulation then lets us differentiate the diagram to calculate $D_{c_{nm}(\tau)}[\mathfrak{U}_O(t_0, t_0 + t_e)]$ as a linear combination over path(γ).

$$\begin{tikzcd} C(\tau) \arrow[r, "D\pi_{nm}"] \arrow[rrr, bend left, "D_{c_{nm}}\mathfrak{U}_O(I)\big|_\tau"] & \mathbb{R} \arrow[r, hook, "DV^\gamma_{N_O}(\tau)"] & \bigotimes\limits_{\substack{k=(k_1,\ldots,k_q)\\ k \in \text{path}(\gamma)}} \mathbb{R}^{|N_O|} \arrow[r, "DT"] & \mathbb{R}^{|N_O|} \arrow[r, "DE"] & \mathbb{R} \end{tikzcd}$$

Therefore $\frac{\partial E}{\partial c_{nm}(\tau)} = DE \circ DT \circ DV^{\gamma}_{N_O}(\tau) \circ D\pi_{nm}$. Using the voltage definition and the fact that derivative of T is just a linear map $\bigotimes \mathbb{R}^{|N_O|} \to \mathbb{R}^{|N_O|}$,

$$\begin{aligned} D\pi_{nm} &= \frac{\partial V_m[\tau]}{\partial c_{nm}} = \chi_{n \in A'_m} \sigma(V_n[\tau - \Delta t]). \\ DT \circ DV^{\gamma}_{N_O}(\tau) &= \sum_{\substack{k=(k_1,\ldots,k_q) \\ k \in \text{path}(\gamma)}} \prod_{j=2}^{q} \frac{\partial V_{k_j}[t_{k_j}]}{\partial V_{k_{j-1}}[t_{k_{j-1}}]} \qquad (3.1.2) \\ DE &= \langle\, V_{N_O}[t_0 + t_e] - y, \cdot \rangle . \end{aligned}$$

This path formulation essentially only requires that at every timestep we calculate the affect of an anterior neuron on its posterior and the affect of all anterior connections on the voltage of the neuron. Then when weight update occurs, every neuron would pass its error signal to its anterior neurons to be aggregated and finally updated. However, this process immediately gives infinite gradients if indeed there are cycles and requires that weigh updates occur synchronously, even before integrating over τ.

3.2 Towards Decentralized Gradient Descent

Given our exposition into supervised learning for O, it would seem that such a task is not suited to current optimization methods. However, recalling methods from quantum physics $DT \circ DV^{\gamma}_{N_O}$ appears analygous to the Feynman path integral. We conjecture that there is actually a reasonable space-time path distribution for the gradient over path(γ) which can be renormalized. If such a distribution exists we conjecture that it suffices to draw a discrete set of paths in the calculation of the gradient.

4 Experimentation [Omitted]

4.1 Implementation [Omitted]

4.2 Results [Omitted]

5 Conclusion [Omitted]

5.1 Future Work [Omitted]

References

[1] Shane Legg. “Machine super intelligence”. PhD thesis. University of Lugano, 2008.

[2] Shane Legg and Marcus Hutter. “Universal intelligence: A definition of machine intelligence”. In: *Minds and Machines* 17.4 (2007), pp. 391–444.

[3] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[4] Samuel Rathmanner and Marcus Hutter. “A philosophical treatise of universal induction”. In: *Entropy* 13.6 (2011), pp. 1076–1136.

Appendices

A Universal Intelligence Definitions