Neural Mesh: Introducing a Notion of Space and Conservation of Energy to Neural Nets

Zoe D. Papakipos and Jacob A. Beck

I. ABSTRACT

Neural nets are based on a simplified model of the brain. In this project, we wanted to slightly unsimplify this model and make a neural network that more closely emulates the low-level interactions of neurons. We accomplished this by making a new model, called a neural mesh, which takes into account a concept of space. In our model, neurons can only fire to adjacent neurons, as in the brain. Like in an RNN, our model has a state that persists between time steps, so the neurons' energies stay around. Like in the brain, we only allow neurons to fire in a time step if they contain enough energy, or excitement. We also enforce a notion of conservation of energy, so that a neuron cannot excite its neighbors more than the excitement it already contained at that time step. Thanks to these features, our neural mesh more closely models the brain compared to a classic neural net. We found that our neural mesh performed better on classifying MNIST digits than a one-layer feed-forward neural net with a small number of neurons, although this advantage disappeared with more neurons.

II. BACKGROUND

Neural nets were created to emulate a basic idea of how the brain works: neurons fire and excite other neurons, and how much one neuron excites another gets stronger the more they fire together. The details of the optimization algorithm have changed since then, with the use of things like gradient descent and annealed learning rates. But the idea remains: to loosely model the brain, in the hopes that emulating one optimizer which produces intelligence will help us get closer to producing actual intelligence. And neural nets do work much better than their predecessors at many tasks. But this does not mean they are actually close to producing intelligence itself.

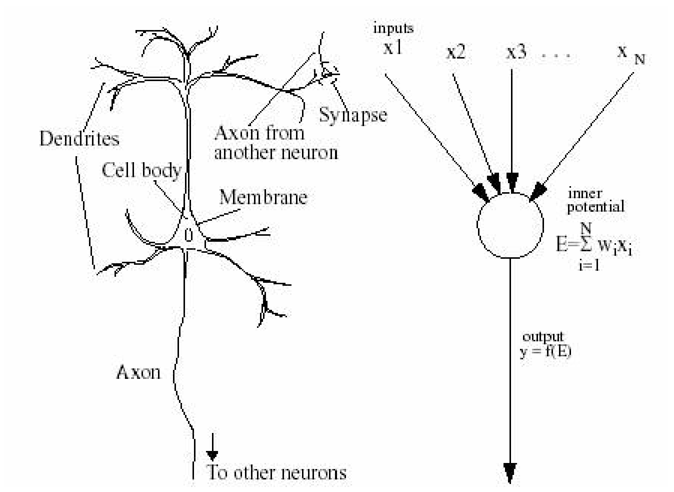

Fig. 1. Classic neural nets emulate a basic model of the brain. [1]

Recurrent neural nets, or RNNs, took neural nets a step closer to how our brain works: an RNN keeps around a persistent state throughout time steps, thus keeping temporally disjoint information around. RNNs also use masking mechanisms to determine which information to keep around at each step. Both of these improvements resemble how a network of actual neurons works: energy doesn't just reset in the neurons after each time step or inference. The same energy levels persist, and the state of the network keeps some information around, depending on the state at previous time steps. In these ways, RNNs are closer to a true model of the brain, but they still dont get it quite right. We wanted to improve upon the RNN by imposing a notion of space and forcing the neurons to obey laws of physics. If deep neural nets and RNNs were the first steps toward emulating neurons learning in the brain, then we think that our neural mesh is another step along the same path.

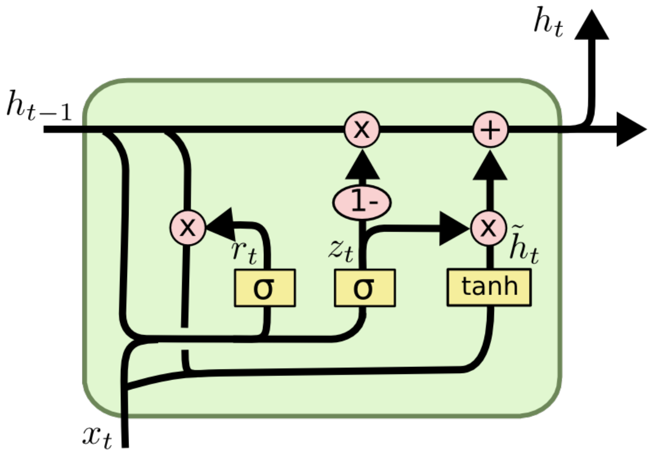

Fig. 2. An RNN, like this GRU, has a state that persists between time steps, or inputs. [2]

So how do neurons actually learn? How are connections formed, and how do neurons pass energy to other neurons? Neurons in the brain exist in a network in 3D space. Each neuron can fire to up to 10,000 other nearby neurons. When a neuron gets "excited", ions move into it, and it "fires", meaning it releases some number of vesicles of neurotransmitters across each synapse to adjacent neurons. [3] How many vesicles are released varies depending on the neurons involved, but also within the same neuron-to-neuron connection (or synapse). Unfortunately, its not yet well understood in the neuroscientific community when and why a neuron releases a certain number of vesicles. It may be somewhat random, but it seems more likely the reason is something we haven't been able to identify yet.

We model how much potential one neuron has to "excite" another as a weight between them, as in the standard neural net model. We had to make another assumption to introduce variability into how much exciting happens in each synapse. We chose to model this by enforcing a kind of conservation of energy: however much excitement a neuron has in a given time step, that neuron cannot excite other neurons any more than that amount. That is, for a neuron i with activation a_i and neighbors in set N_i:

$$\sum_{j \in N_i} |w_{ij} a_i| \leq a_i$$

Thus, how much a neuron excites another is based not only on the learned weight between them, but also on the current excitement of the source neuron. It is worth noting that this is an assumption that classic neural nets already make. The part we are introducing is the conservation of energy, which reflects the fact that in our brains, neurons have to "build up" excitement and cannot fire if they are not excited.

Each neuron releases one of two types of neurotransmitters: inhibitory or excitory. We model this by positive or negative weights, respectively. Inhibitory neurotransmitters inhibit excitement, whereas excitory neurotransmitters increase excitement. Thus, the activations in the neurons of our mesh represent the excitement of the neurons. In our brains, neurons have to build up excitement to a certain level before they can fire again. An obvious way for us to model this is to only allow a neuron to fire if its activation is past some threshold. We decided for the sake of ease to choose this threshold to be 0.

III. Methods

Our neural mesh is a network of neurons, where the location of neurons determines which neurons it is connected to. While neurons in the brain are admittedly arranged in a 3D network, we chose to model it in 2D. We simplified the geometry to a square grid, where each neuron can fire to the neurons above, below, right, and left of itself. However, the grid "wraps around", meaning the neurons at the top can fire to the ones on the bottom row, and vice versa, and the neurons at the right can fire to the ones on the leftmost row, and vice versa. Thus, our network can be seen as a 3D mesh - a torus, to be precise.

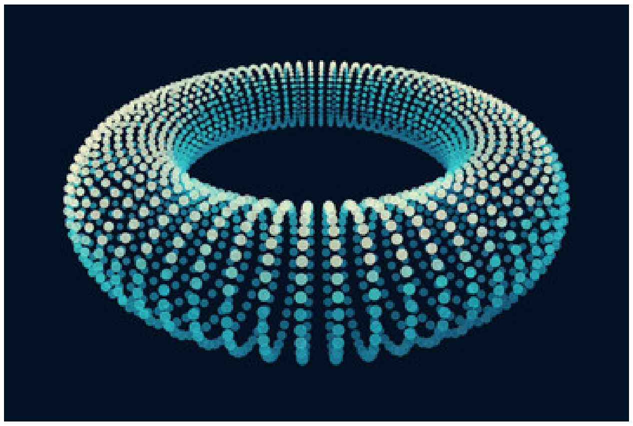

Fig. 3. The effective shape of our neural mesh is a torus. [4]

Neurons in the neural mesh fire in basically the same way as normal neural nets: for each neuron-to-neuron connection, there is a weight that is trained via gradient descent. When a neuron i fires to neuron j at time t, neuron j's activation a_j for time $t + 1$ is set to $w_{ij}a_i$. Thus, at each time step, all the neurons update simultaneously based on other neurons firing into it. However, there are many things about our neural mesh that diverge from classic neural nets. As previously mentioned, only the four adjacent neurons can fire to a given neuron. We also had several other ideas for mechanics that might work well for our model, so we tried out all the following options (along with a few others that did not pan out and were uninteresting enough we didn't feel them necessary to include):

1) Keep showing the image as input at every time step, rather than just on the first one.
2) Max the activations with 0 at every time step (i.e. apply relu to the activations).
3) Include a bias in the feed-forward layer for the input.
4) Clip the activations to [-1, 1] at each time step.

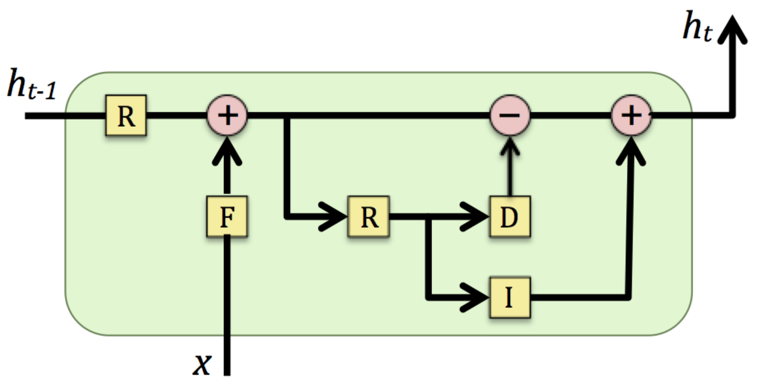

Fig. 4. The neural mesh architecture includes a persistent state and gating mechanisms, like an RNN.

Note that in figure 4, R refers to performing a relu operation, F to applying a feed-forward layer, D to a decrement operation, and I to an increment operation. A decrement operation computes the amount of energy leaving each neuron as follows, where M_u is the matrix that transforms an input up, i.e. the identity matrix shifted one row up. M_d, M_r, and M_l are defined similarly for shifted down, right, and left. $*$ represents pointwise multiplication, and W_u is a matrix of the weights from each neuron to its neighbor above.

$$D(s) = \frac{1}{4}M_u \cdot (W_u * s) + \frac{1}{4}M_d \cdot (W_d * s) + \frac{1}{4}M_r \cdot (W_r * s) + \frac{1}{4}M_l \cdot (W_l * s)$$

An increment operation computes the amount of energy leaving each neuron, as follows:

$$I(s) = \frac{1}{4}M_u \cdot (W_u * s) + \frac{1}{4}M_d \cdot (W_d * s) + \frac{1}{4}M_r \cdot (W_r * s) + \frac{1}{4}M_l \cdot (W_l * s)$$

Ideally, our neural mesh would be evaluated as a general AI, since our aim in coming up with this architecture was to make something that can learn like a human. But there are currently no datasets or standard benchmarks for evaluating general AIs, and we also don't have a good loss function to use even if we had such a dataset. If we were to try to evaluate our neural mesh as a general AI, we might use an evolutionary algorithm to learn a reward function for the environment that it is placed in. However, finding a way to evaluate general AI learning models is outside the scope of this paper. In this paper, we are merely trying out what we think is a good internal structure to eventually use as part of an artificial brain. To evaluate this piece, we want to make sure it can learn low-level tasks. So we can basically evaluate it on any of the normal machine learning datasets.

We decided to test our neural mesh on a classic machine learning task, but one that is not generally associated with recurrent models: pattern recognition in images (specifically, MNIST). Our hope for the neural mesh on this task is that instead of looking at an item in a sequence at each time step, and using the state to keep around information about previous items in the sequence, our model processing an image is more like a human looking at the same image over many time steps, and allowing computation to keep happening. You can kind of see our neural mesh as a kind of general form of a feed forward net: information flows into the mesh, and then flows around the mesh between nodes, being processed as in a multi-layer feed-forward neural net. This seems like a useful abstraction: rather than defining a fixed number of layers and a fixed size for each layer, we are letting the model decide for itself how many times it flows the information through the mesh, and how many neurons it uses.

In fact, it is possible that our neural mesh could even emulate a classic one-layer feed-forward neural net. If we run the neural mesh without repeatedly showing the input, and choose to either relu the activations at each time step or clip our activations to [-1, 1], then our neural mesh can exactly emulate a one-layer feed-forward neural net if it sets the internal weights to 0, since it just takes in the input, applies some weights to it, applies a non-linearity, then applies the output weights. It therefore makes a lot of sense to compare our neural mesh to a one-layer feed-forward neural net, since the latter is a realizable subset of our hypothesis space. We expect to do no worse than a one-layer feed-forward neural net since our neural mesh can emulate one, but hopefully will choose to do something more clever.

IV. RESULTS

We evaluated our neural mesh against a one-layer feed-forward neural net on a classic image recognition problem: classification of MNIST handwritten digits. We found that with the following settings, our neural mesh outperformed a one-layer feed-forward neural net on classification accuracy:

1) Do not show the image as input as every time step; only on the first one.
2) Do not relu the activations at each time step.
3) Do not include a bias on the feed-forward input layer.
4) Do clip the activations to [-1, 1] at each time step.

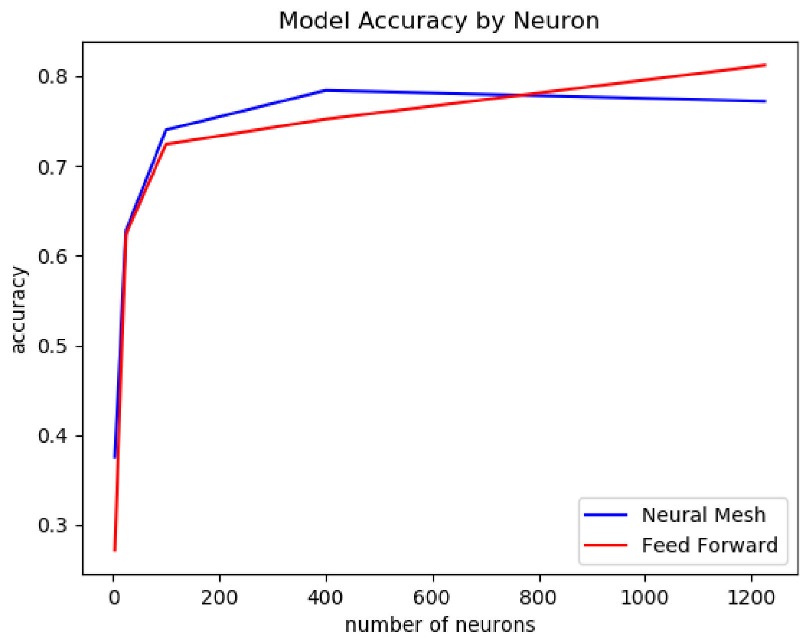

Fig. 5. Our neural mesh outperforms a one-layer feed-forward neural net with smaller numbers of neurons.

It is interesting that reluing the activations at each time step, although introducing a non-linearity, did not help training. However, clipping the activations to [-1, 1] is also a non-linear operation, so this probably makes up for the lack of relu or other classic activation function. The fact that we are not using input bias or showing the input at every time step means that after the first time step when the image is first shown and fired into the neural mesh, no energy is entering the mesh again. This is significant because it means that for computation to happen, the energy already in the system must be circulated by firing. In a classic neural net, on the other hand, input is generally shown at every time step or computation step, so new energy is constantly entering the system.

We also found that reshaping the neural mesh without varying the number of neurons (like 1200 x 1 versus 30 x 40) did not significantly change the behavior. Given the wrap-around nature of the neural mesh, this makes some sense since energy can get from any neuron to any other neuron in the net given enough time steps because the entire net is fully connected. The other hyperparameters, including number of epochs and learning rate, were tuned in a standard way.

While exploring whether clipping the activations helped, we found some interesting trends, shown in figures 6 and 7. If we did not clip the activations, then increasing the window size did improve the model's accuracy, although the accuracy was still lower than the accuracies achieved with clipping. One reason for this might be that using a larger window size increases the number of times relu is applied within the neural mesh cell. (As shown back in figure 4, the first relu on the incoming state is variable and in case turned off, but the internal relu is always applied at each time step). So even though we are not introducing a non-linearity by clipping the activations, we are applying relu (to part of the state) more and more times, so the neural mesh might be leveraging these non-linearities with a large window.

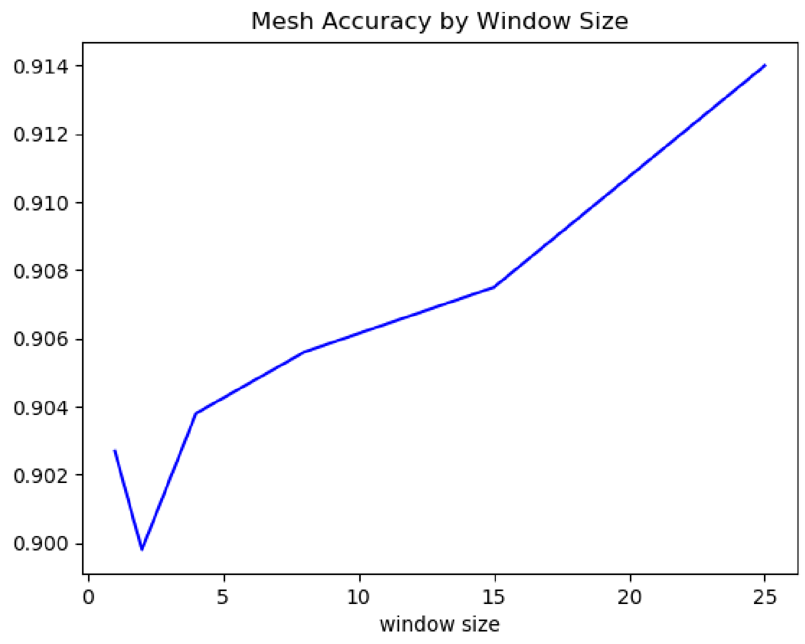

Fig. 6. When we don't clip the activations, increasing the window size does improve the model, but it still does worse than clipping.

If we did clip the activations, then changing the window size did not significantly improve the model's accuracy. This would fit our hypothesis that increasing the window size introducing a non-linearity like clipping. Thus, since we are already clipping, increasing the window size does not carry the same benefit. However, we are still surprised that increasing the window size does not help at all. It seems plausible that increasing the window size (and therefore the number of time steps the neural mesh gets to process each input) would always help the accuracy. It is possible with too large a window size when the accuracy is already good makes the model overfit to the training data, since the neural mesh is acting more like a classic neural net with more neurons.

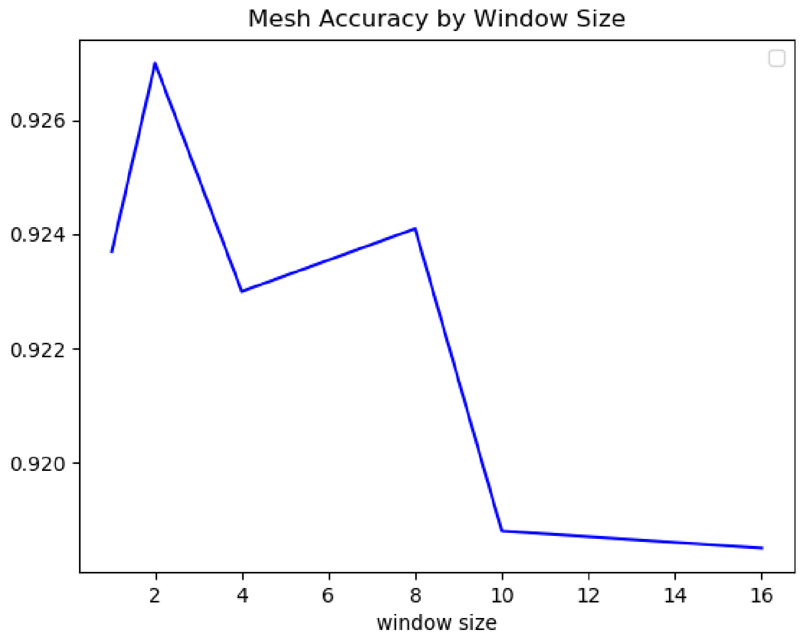

Fig. 7. Increasing the window size doesn't improve the model when we clip the activations.

Given the above settings that we landed on, we then tuned the number of neurons. We compared the test accuracies of both the neural mesh and the one-layer feed-forward net with many different numbers of neurons. Interestingly, we found that the neural mesh outperformed or did similarly to the feed-forward net with smaller numbers of neurons. Our hypothesis for why this might be is that the neural mesh is less limited by the number of neurons, since it can move energies around and even reuse neurons over many time steps. In this way, and given the non-linearity of clipping activations to [-1, 1] the neural mesh can kind of mimic a deeper neural net.

We also visualized the activations (i.e. energies) in all the neurons in the mesh over time, to give a sense of what is going on. Figures 8 and 9 show a window (of size 10) from three randomly selected inputs. In each square, a pixel represents a neuron in the neural mesh at that time step. The color of each pixel shows the activation level, where 0 is black, yellow is 1, and pink is in between. Figure 8 was produced using our settings as described above, and figure 9 was produced using the same settings but also showing the image as input at every time step, rather than just the first one.

As we can see in figure 8, if we only show the image once, the total energy (i.e. "excitement") in the mesh decreases over time. This makes sense: when we show the image as input and fire it into the neural mesh through the input feed-forward layer, we are adding energy to the system through the values in the image and the input weights. Note that if the input weights are 0, we might remove some energy, but we still have the potential to add energy when showing the input. On the time steps when the input is not shown, there is no way to add energy to the system. We can only move energy if a neuron fires with a positive weight, or remove energy if a neuron fires with a negative weight (i.e. expends energy to decrease the energy of, or inhibit, another neuron).

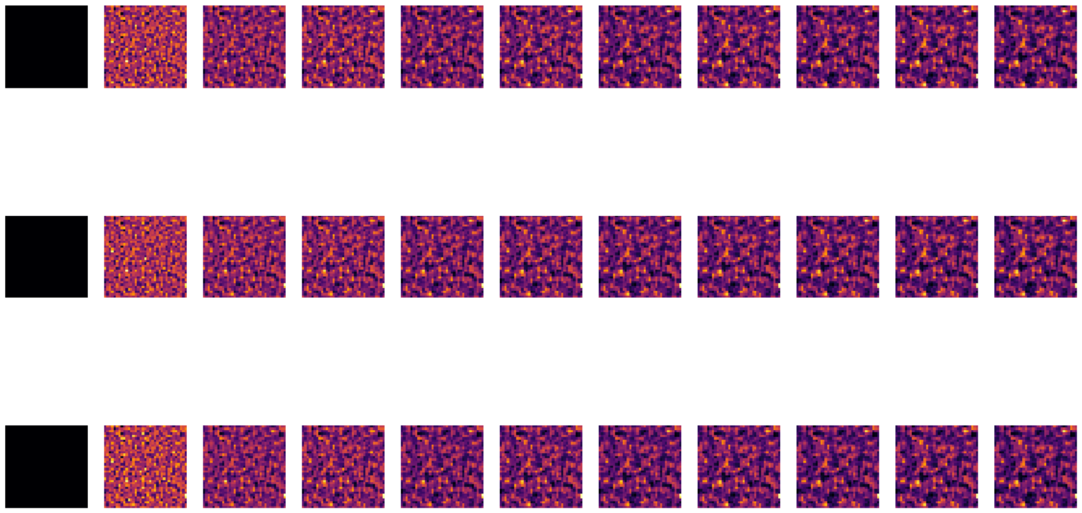

Fig. 8. With our settings, the total energy in the mesh decreases over time.

With the settings in figure 9, on the other hand, we show the image to the neural mesh at every time step, so if the model learns sufficiently positive input weights, the neural mesh can gain energy at every time step. That clearly happens and is illustrated nicely in figure 9, by the neural mesh becoming more yellow at every time step.

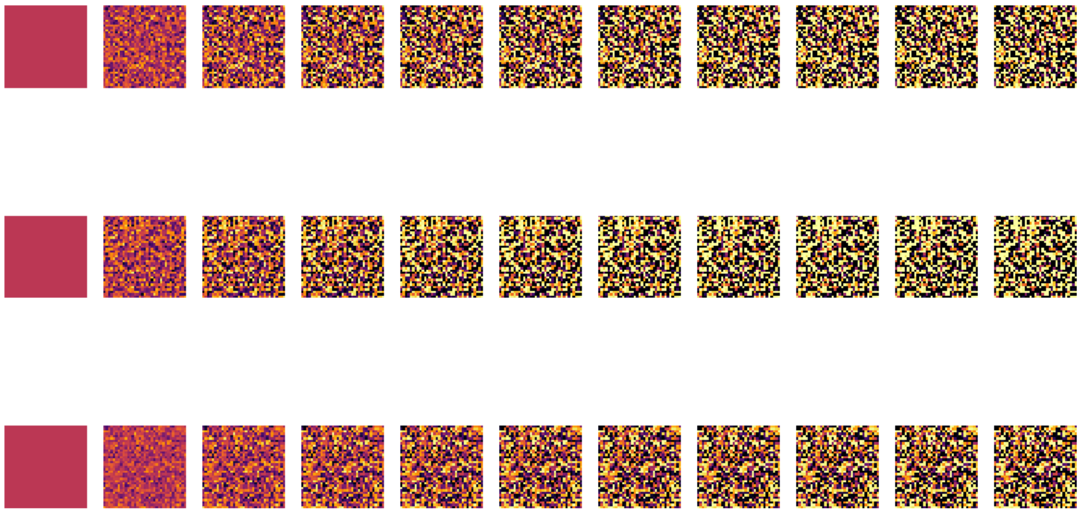

Fig. 9. If we show the input every time step, the total energy increases over time.

V. CONCLUSION

We have demonstrated that our neural mesh can do at least about as well on basic machine learning tasks, and can seemingly even get more performance out of a small number of neurons. Thus, our model might be useful even just as a lightweight learner. However, our hope is that by designing an architecture to more closely emulate the brain, we have made something that might be useful as a general learner. But as we mentioned before, we do not believe this is possible to benchmark yet. While we do not yet have a robust enough reward function that would allow us to test our model like that, we believe that once we find such a reward function, it will be good to already have architectures that emulate the brain to test on general learning.

In introducing the neural mesh, we hope we have demonstrated the utility of going further down the path of designing machine learning architectures that more closely resemble the brain. We recognize that it is impossible to bring this path to its logical conclusion by making an architecture that fully resembles the brain until we understand every step of how the brain learns. However, we believe that it is useful to think about these things even before we get there, so that when we have all the requisite information, we have models to work from that are closer to what we want.

VI. REFERENCES

1. Hemming, Cecilia. Using Neural Network to Retrieve Lexical Data. Swedish National Graduate School of Language Technology, www.hemming.se/gslt/LingRes/NeuralNetworks.htm.

2. Olah, Christopher. Understanding LSTM Networks. Understanding LSTM Networks, Colah Blog, 27 Aug. 2015, colah.github.io/posts/2015-08-Understanding-LSTMs/.

3. Signal Propagation: The Movement of Signals between Neurons. Khan Academy, www.khanacademy.org/test-prep/mcat/organ-systems/neural-synapses/a/signal-propagation-the-movement-of-signals-between-neurons.

4. © Adobe Stock