A neural network for breaking time dependent problems into parts

Tina White

What is the problem?

I demonstrate how a cognitively inspired neural network architecture performs on a given dataset and compare its performance to a recently developed baseline. The dataset is a small collection of solutions to Burgers' equation, shown in Figure 1, a one-dimensional application of an initial-boundary-value problem that models the movement of a shockwave in a tube.

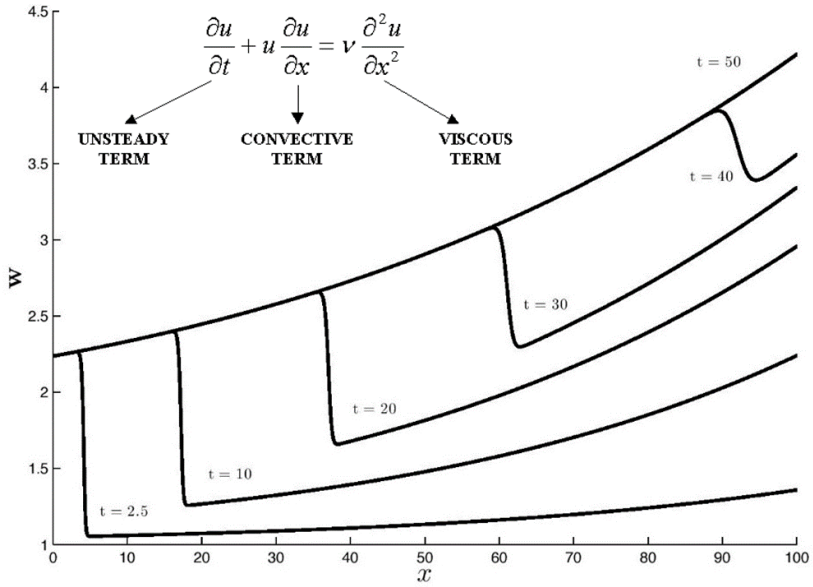


Figure 1: Typical Burgers' equation solutions for velocity, *w* or *u*, as a function of space, *x*, and time, *t*

In problems where the time-dependent behavior of a fluids problem is of interest, like Burgers' equation, the inclusion of LSTMs is expected to improve accuracy, reduce computational expense, or both, since in the baseline architecture, time step uniformity was not being exploited. The performance of the baseline model may be improved using LSTM nodes. The problem is, simply, can the predictions be improved if it is known that one of the inputs is time, uniformly spaced?

The baseline model

The baseline architecture is a simple feed-forward network, except with clearly delineated connected clusters shown in Figure 2. The network performs a mapping from (x,t,μ) tuples to (f) where x represents space, t represents time, and y represents the output of the network, which approximates f, the function to be learned. In this case, f is the velocity w or u. Also, μ represents a hyperparameter, which could refer to any parameter than has been changed from one simulation to the next. A hyperparameter could be, for example, a boundary condition, or a constant in the governing equation, like viscosity.

This architecture is unique in that different clusters automatically identify zones in the perceived data that behave according to different functions (the function networks). The separation is similar to a mixture of experts layer. [3] A separate part of the architecture determines when and by how much to turn the other parts on or off (the context networks). Different loss functions are used for each part of the network to train them as either function or context networks. The Burgers' equation test case is well suited to this architecture because the problem is more easily solved by breaking it into parts – solutions in front of the

shockwave and behind it can be thought of as two functions that behave almost independently, with a boundary (the shockwave) between them.

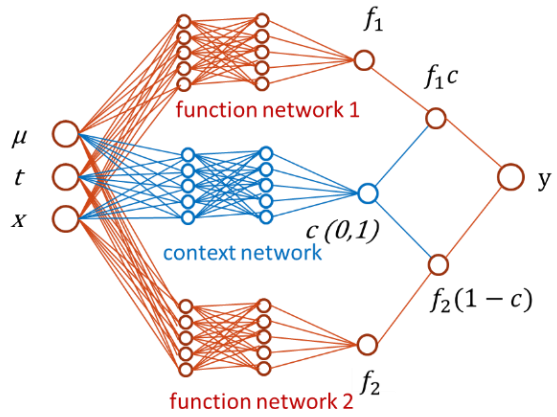


Figure 2: The clustered network architecture

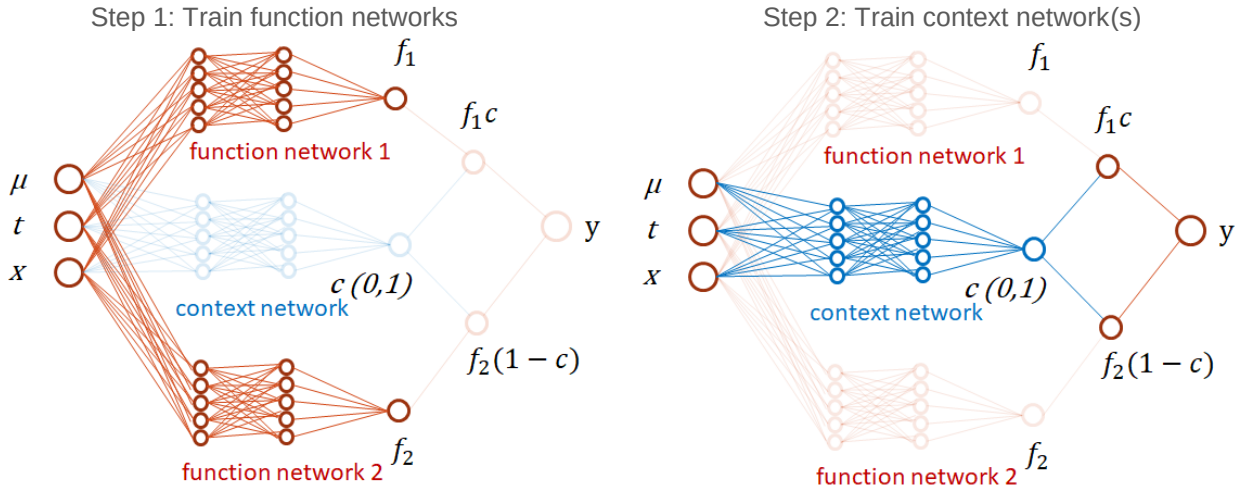


Figure 3: The clustered network training procedure

The training procedure for this network is also unique in that the function networks are trained separately from the context network. The training procedure for the simple 2-cluster network in the diagram is (1) Train the network using a loss function defined as the minimum of abs(*f-f1*) and abs(*f-f2*) (2) While holding the function network weights constant, train the cluster networks using the loss function defined as abs(*f-y*) and (3) Train both networks simultaneously using the loss function from (2).

The limitations of this network are somewhat obvious – scaling up to more complicated PDEs and fluids problems becomes a challenge quickly. I don't expect this overly-simple network to capture the behavior of a wake, for example. More insight is needed to build up to that point, if this type of network can be used as a building block at all.

The architecture might be overly simple, but it is a good starting point because it is the only architecture found that works well for this problem, as shown in Figure 4b, given very examples of Burgers' simulations, in a way that many earlier attempts did not (including variational autoencoders, more complex LSTM networks for video sequences, and fully connected layers). Earlier version like Figure 4a

tended to either over-generalize, smoothing out the shockwave entirely, or overfit, fitting perfectly to training data, but giving garbage predictions at nearby hyperparameters not part of the training set.

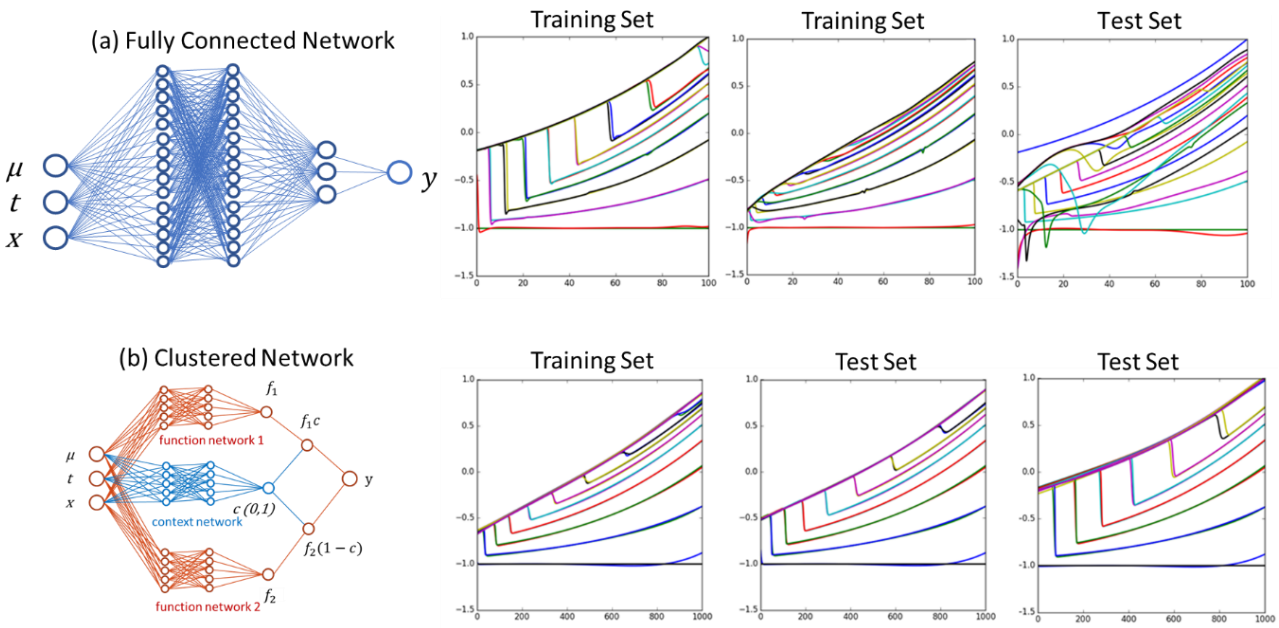


Figure 4: Two different network architectures including a fully connected network and the baseline clustered network and their predictions overlaid on actual solutions from either the training or test sets

A paper [1] from an earlier class project for CS221 includes a more details on (1) the reason why the Burgers' dataset was chosen, (2) the types of engineering problems where this type of neural network architecture would be useful (3) some related work, and (4) a description of other more standard architectures from Fall 2016 that failed to solve the problem. The clustered network that was successful on this problem (the baseline) was developed in Winter 2018.

How will the problem be addressed?

I will modify the code that contains the baseline network and determine how to best combine the baseline architecture with LSTM nodes. The simplest modification would be to change all the current tanh nodes to LSTM nodes, while handling the inputs and outputs appropriately, leaving a network diagram little different from Figure 2. I'll compare the baseline architecture, which performs well, as shown in Figure 3, on the dataset, with the LSTM architecture, which I expect will perform better.

How does the project address computational and cognitive issues?

The baseline network architecture was loosely inspired by the shape of pyramidal cells in the neocortex as shown in Figure 5. Another inspiration came from the way the interaction between neurons and glial cells was presented in [2]. Inspiration for making the LSTM modification for this project came from some points made about "learning to learn" recently in CS379C. I had nearly ruled out using LSTMs due to poor performance with earlier architectures but appreciated the idea that short timescale learning/dynamics can emerge from the behavior of the already-learned weights in a network. Again, while the connection is distant, incorporating LSTMs into the network is a step in that direction.

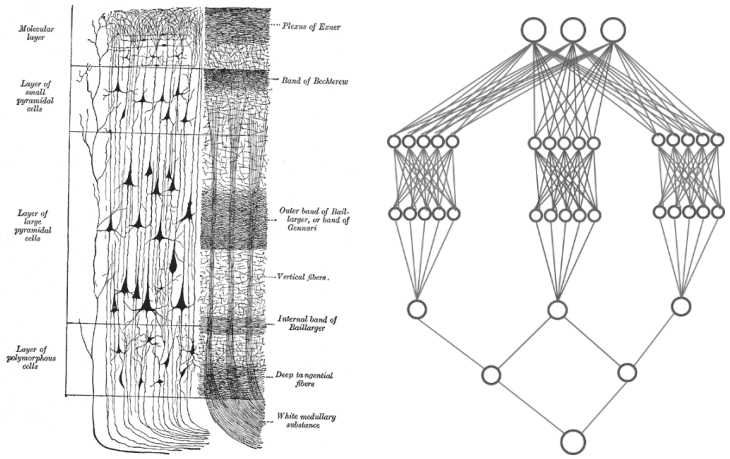


Figure 5: Pyramidal cells in the neocortex vs. baseline clustered network architecture

Also, humans solve and simplify problems by breaking them into parts. This network is designed explicitly to only do that well. Each training point is assumed to fall into one context or another, or along a boundary in between. The function and context networks are thus analogous in some ways to real concepts and context.

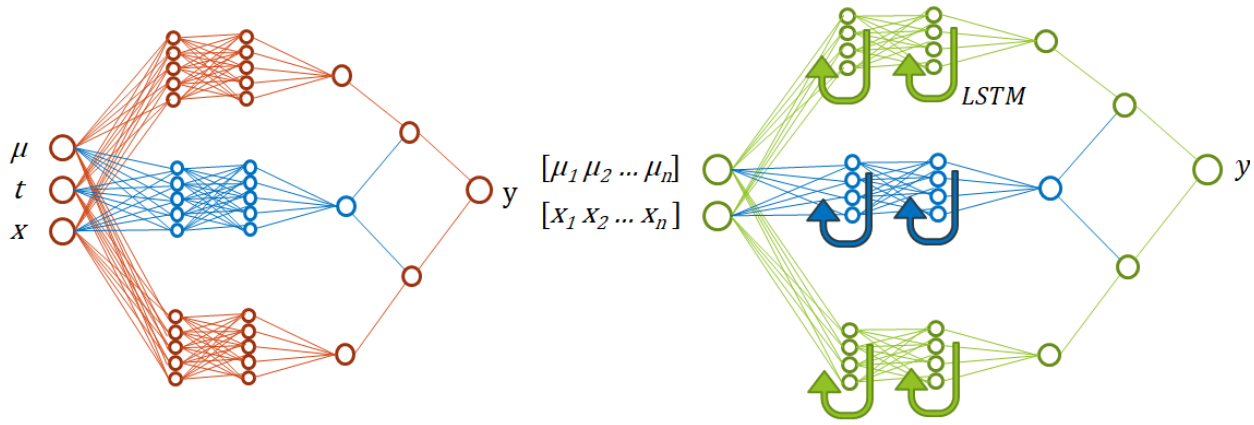
Finally, for the programmer's apprentice, there are a great number of technical subproblems. One such problem is how networks can learn to represent separate and independent concepts from initially jumbled and undifferentiated input data. I assume that trying to fit one curve through the universe is hard, and that separating and categorizing things isn't just something humans do, but something that is necessary for making good predictions. Literally separate clusters of networks for separate concepts is one direct method for accomplishing this, and the method is at least consistent with observations that different parts of the cortex perform unique functions (which, if there is local damage, can be learned and performed elsewhere in the cortex). It's also consistent with observations that intelligence improves with greater synaptic pruning.

Metric for success

The benefit of the baseline network architecture is that it generalizes very well. However, the (offline) training process is slow compared to the offline costs of other methods that address the same industry problems – like reduced order modeling. Therefore, the metric for success would be to see an improvement in training time for the time-dependent Burgers' equation, while not suffering from loss of generalization.

Implementation

As shown in Figure 6, there are two important differences between the baseline architecture and the modified LSTM architecture (1) the tanh nodes have been replaced with LSTM nodes and (2) the input batches are now sequences of (x,μ) tuples instead of (x,t,μ) tuples.



(a) Clustered Network (b) Clustered LSTM Network

Figure 6: Diagram of baseline clustered network vs. clustered LSTM network showing architecture differences, with arrows representing LSTM nodes

Results

The addition of LSTMs did not lead to any improvement, as shown in Figure 7. Surprisingly, the version of the network with LSTMs lost much of what was gained by the original clustered architecture. The version with LSTMs (1) trained far more slowly, even with one layer and very few nodes and (2) did not generalize as well. While both methods fit the training data, only the baseline clustered network learned to accurately extrapolate. Many variations of the clustered LSTM network were explored (changing the number of nodes in each layer, the number of layers, etc.) But the issues persisted. It does not appear to be a bug, and the problem is likely with the model, as discussed in the following section.

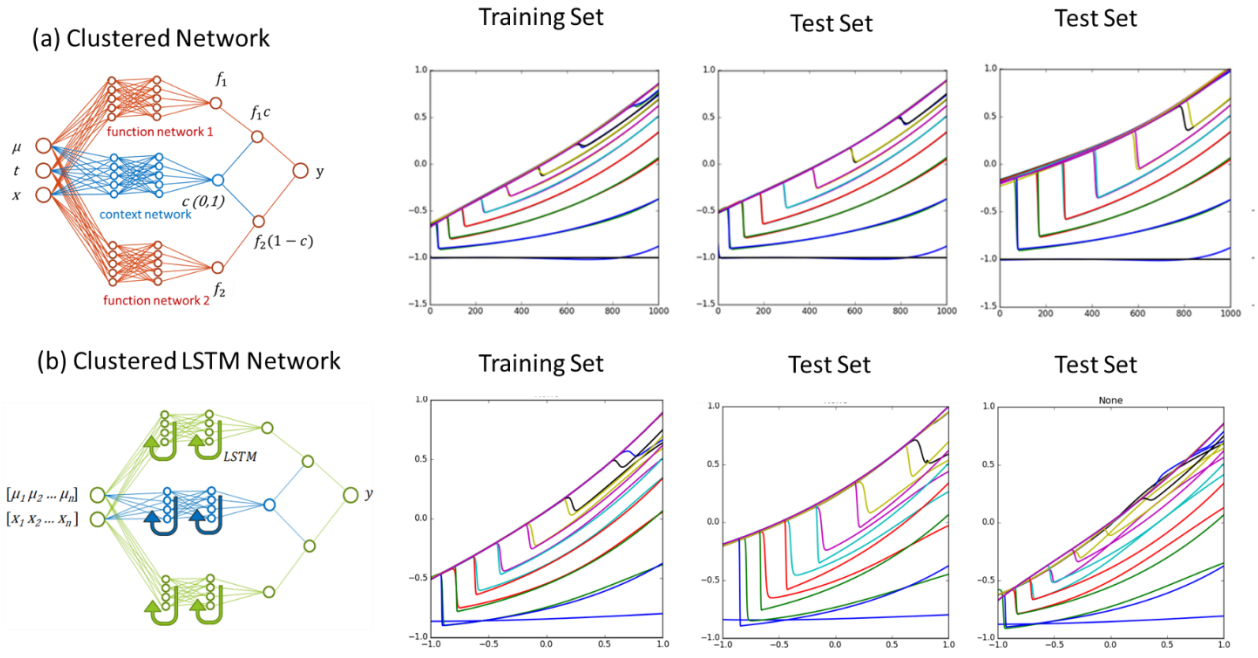


Figure 7: Results for the baseline clustered network compared to the clustered LSTM network with predictions overlaid on actual solutions from either the training or test sets

Discussion

Since the brain receives a steady stream of perceptual data, and since PDE's are often time dependent, with uniformly spaced time steps, the addition of LSTMs to this architecture seemed like a natural extension to the baseline architecture for time dependent problems, which also could apply to the way the brain processes information. However, because LSTM's eliminate the gains seen from the clustered network, the best option going forward might be to step back and determine whether further improvements to the treatment of temporal data in this architecture are necessary, and whether the current setup already treats time in a way suited well enough to the problem.

One reason the LSTM network did not improve the results may be that it is unnecessary for the network to remember anything more than a step or two back in time. With smooth functions, perhaps only memory of one previous time step is necessary. Therefore, an option going forward might be to try a convolution over time instead of a complex structure like an LSTM. Also, part of the reason the original networks generalizes so well is that it has so few parameters (weights and biases) to learn. The LSTM introduces more parameters into the system, which change with time. Part of the benefit of the baseline network may have been its simplicity, some of which is lost with an LSTM.

Another reason there was no improvement may be that the current bias of the network already handles time appropriately – perhaps there is no need to treat time differently from space, given that the datasets are fully solved problems in both time in space. So, there is no need to "step" forward in time. The current network fits to the simplest possible function in time, and deviations from this bias may only lead to poorer performance.

[1] White, Tina. "A neural network architecture for reduced order modeling of PDEs." https://web.stanford.edu/class/cs221/2017/restricted/p-final/crwhite/final.pdf (2016).
[2] Samborska, Veronika, et al. "Mammalian Brain As a Network of Networks." Opera Medica & Physiologica (2016).
[3] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. CoRR, arXiv:1701.06538, 2017.