A neural network architecture for reduced order modeling of PDEs

Tina White
Department of Mechanical Engineering
Stanford University
Stanford, CA 94305
crwhite@stanford.edu

Abstract

Neural networks are powerful tools for solving problems for which the governing equations are unknown. This project will assess the suitability of neural networks for solving problems for which the governing equations are known, but time intensive to compute. It will attempt to compress training data from previous engineering simulations to find a simpler representation of a given problem. The training data will come from a finite element analysis, which is a numerical simulation technique using a large system of equations to find approximate solutions to boundary value problems for partial differential equations. A computational fluid dynamics (CFD) problem would be a prime example of a finite element-like problem that computes accurate results, but in engineering problems that are of interest to industry, the results often require extensive computational power to achieve.

1 Background

1.1 Baseline

The current state of the art for this problem is a ROM (reduced order model). In a ROM, a training set exists with pre-computed CFD simulations at various parameters. In the first stage, the training set is split into sub-regions using k-means clustering. A singular value decomposition (SVD) is computed to reduce the size of the sub-matrices and create a local reduced order basis (ROB). In the second stage of the method, a residual minimization problem minimizes the error in the original PDE using a linear combination of the basis vectors, which gives an estimate of the solution at target hyper parameters in real time. ROMs can be accurate, but their implementation is an art, and solutions frequently suffer from highly inaccurate predictions and Gibbs' oscillations near discontinuities, as shown in Figure 1, which represents a typical output given by the baseline method for a Burgers' equation test case.

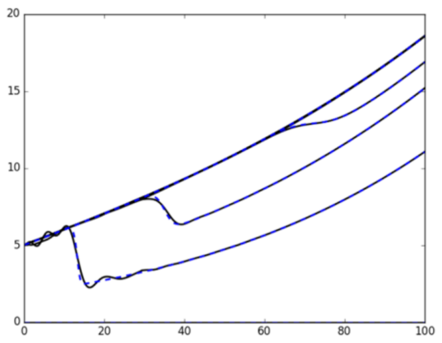

Figure 1: Gibbs' oscillations in ROM outputs represented by solid black lines

1.2 Oracle

If you optimize for accuracy, a full order simulation can be run at every parameter of interest. The downside is computational expense, so that optimization according to parameters of interest frequently becomes infeasible. Also, an oracle optimized for speed would be an engineer familiar with the problem with a few CFD simulations at different parameters. The engineer can make an educated guess at parameters near the optimum. This is what usually happens for very high dimensional CFD problems in engineering. An ideal solution, a true oracle, optimized for both accuracy and time, would instantaneously compute accurate results for a new parameter that exactly match the results of a full order simulation.

1.3 Related Work

There are alternatives that are budding areas of research. Neural networks have been recently used to directly estimate design quantities of interest in a fluid simulation directly from parameters, skipping over the simulation stage. This option would be less time intensive, but for engineers to trust these kinds of simulations, an autoencoder is an important aspect of the solution, because the representation is possible to decode and visualize, and therefore the engineer or scientist can be confident that the approximation retains some fidelity to the physics of the problem. Other related work applies neural networks to simplified and smooth fluids equations that do not include discontinuities. [1] [3] While these are visually appealing results, simulation results of interest to engineers often involve discontinuities like shock waves or crack propagation, and so a method that cannot predict a shockwave location would not be useful from an engineering standpoint.

1.4 Problem Summary

Input: Training data from previous CFD simulations at various parameters.

Output: Approximations of CFD solutions at target parameters.

Evaluation metrics for success: (1) A sufficiently small root mean squared error between the full CFD simulation output and system output at the target parameters and (2) A runtime on the order of a reduced order modeling implementation.

2 Dataset

The Burgers' equation (1) can be used for testing, and is typically used for testing reduced order modeling (ROM) methods. ROM methods that can be successfully applied to predict Burgers' equation solutions, in most circumstances, can be scaled up and applied to the full Navier Stokes equations. While the Burgers' equation test case is a toy problem, and trivial, it is useful and is complex enough to translate to much harder problems. It is a one-dimensional application of an initial-boundary-value problem that models the movement of a shockwave in a fluid. Figure 2 shows the fluid velocity w (or u) vs. location x as a shockwave moves from left to right in time across the domain.

$$\underset{\substack{\swarrow\\ \text{UNSTEADY}\\ \text{TERM}}}{\frac{\partial u}{\partial t}} + \underset{\substack{\downarrow\\ \text{CONVECTIVE}\\ \text{TERM}}}{u\frac{\partial u}{\partial x}} = \underset{\substack{\searrow\\ \text{VISCOUS}\\ \text{TERM}}}{\nu\frac{\partial^2 u}{\partial x^2}} \tag{1}$$

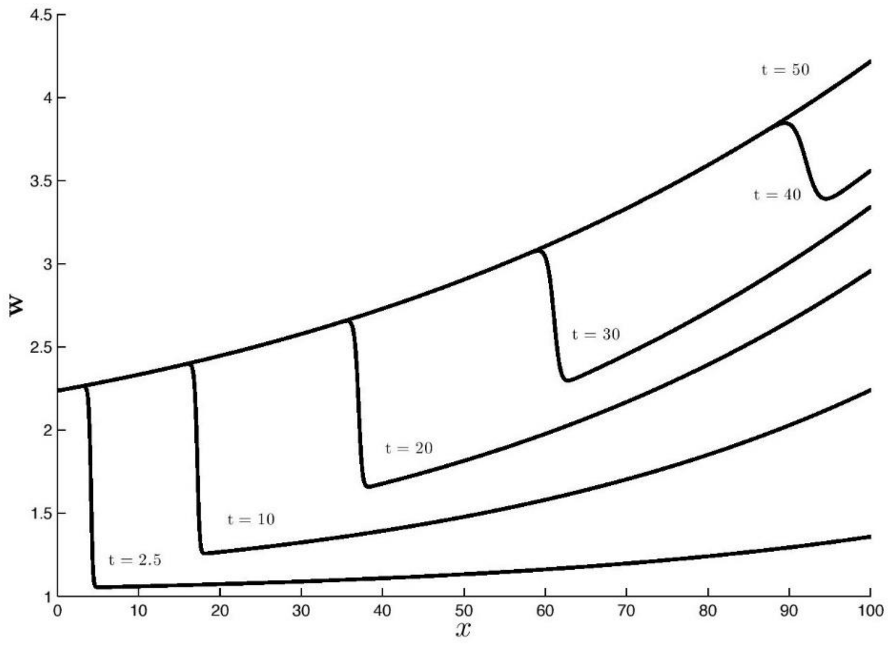

Figure 2: Typical Burgers' equation solutions

Several Burgers' equation analyses were run using a small python code for ROMs. For each case, given a specified parameter, 501 snapshots in time were generated. These cases were separated into a training set and a validation set represented by Figure 3.

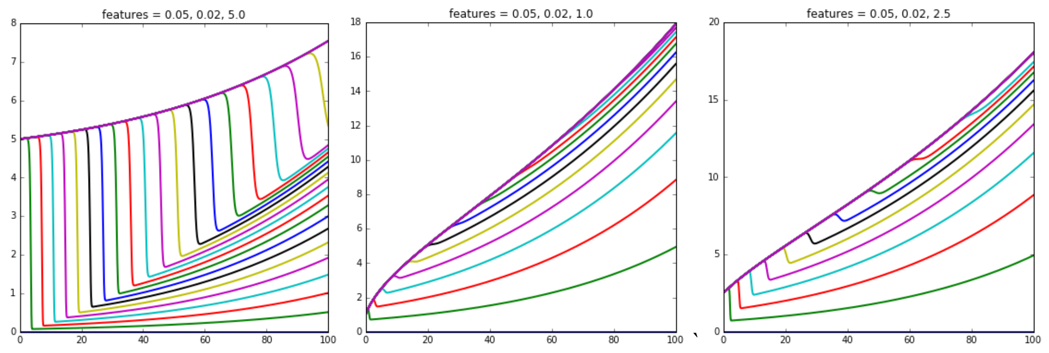

Figure 3: The training set (left two images) and validation set (right image)

3 Model and Algorithm

The model consists of two parts (1) an autoencoder and (2) an LSTM network.

3.1 Autoencoder

An autoencoder determines features from the solution inputs in order to compress the data prior to being fed into the LSTM network in section 3.2. Without the autoencoder, the full Burgers' solution is input into the LSTM network, and the run time of both the network training and testing is unnecessarily increased. With an autoencoder, the information can be highly compressed prior to being fed into the LSTM.

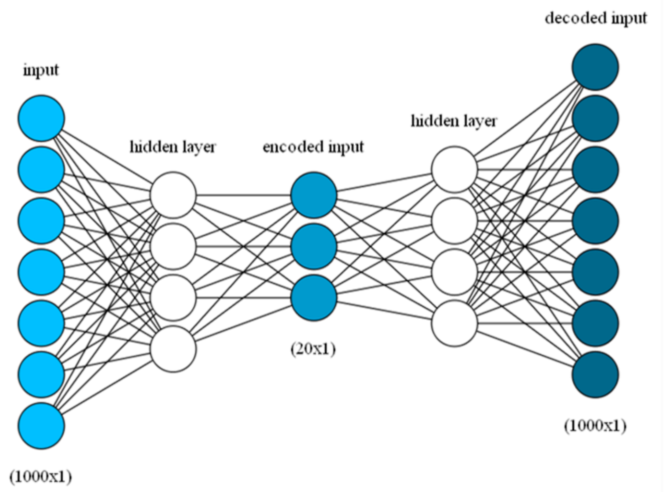

Figure 4: Autoencoder

The autoencoder only outputs representations of the original inputs, and does not output new solutions in time. It only compresses the data.

3.2 LSTM Network Architecture

When searching for possible network configurations, a neural network architecture was found that could address the problem of the proposed PDE solving system. The image captioning LSTM network from the CS231N homework assignments has nearly the appropriate architecture, and therefore that LSTM network was modified to complete the project as shown in Figure 5. The idea is similar to video learning methods [2].

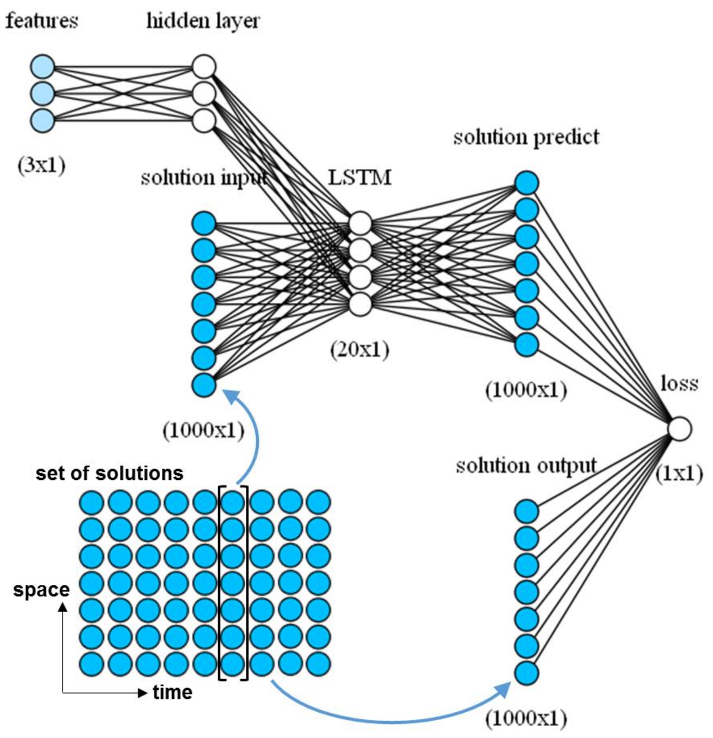

Figure 5: LSTM Network Architecture

An image captioning LSTM network takes an image a feature vector and output captions, which are "time-dependent" in the sense that each word depends on the previous word. Analogously, as shown in Figure 6, the PDE modeling network would take the small set hyperparameters (in the Burgers' equation case, there are only 3) as the feature vector and output the full Burgers' equation solutions (in this case a 1000x501 matrix representing the 1000 spatial points and at the 501 time steps. Because the setup of the image captioning network already includes a fully connected layer from the hyperparameters (features) to the hidden layer, my proposed model no longer needs a step (3), because the step is necessarily included in the network.

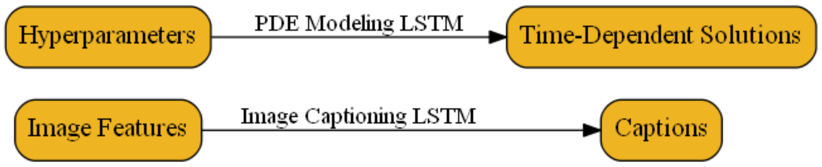

Figure 6: Inputs and outputs of image captioning and PDE modeling systems

Since this is a regression problem and not a classification problem, modifications were made to the original network – most importantly the loss function. In the original image captioning network, a Softmax loss function was employed and a single word was chosen for the next word. This would not be acceptable for the PDE solving system, since the loss function should be a measure of the total error between the predicted solutions (solution_predict) in Figure 5 and the original output solutions (solution_output) in Figure 5.

The inputs will be multiple sets of 1000x501 matrices, each representing a Burgers' equation solution at specific hyperparameters. The simplest version of this system would need at least two points in the dataset, two 1000x501 matrices representing the solutions for two sets of two hyperparameters. The validation set for the network would then be a single hyperparameters case at a point between the two sets. These inputs are shown in Figure 3.

Each long short term memory (LSTM unit) from in Figure 5 has a cell diagram represented by Figure 7, which has a state c_t at time t. The cell is like a memory unit. Access to the memory unit for reading or modifying is controlled through sigmoidal gates – the input gate i_t, forget gate f_t and output gate o_t. The values of the cells are given by the set of equations (2).

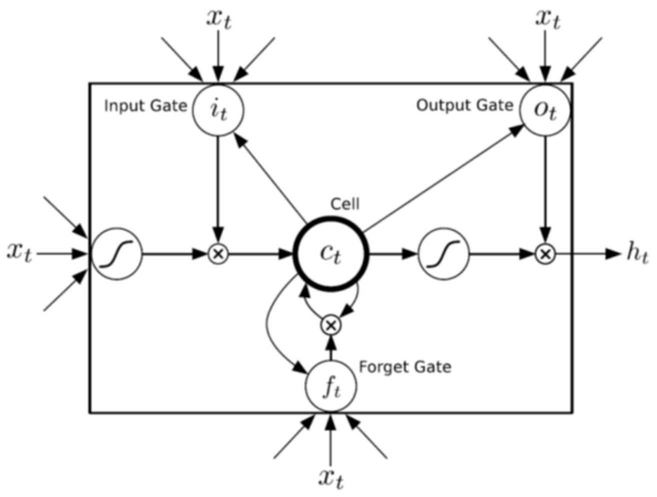

Figure 7: LSTM unit diagram

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right), \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right), \\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\right), \\
\mathbf{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + W_{co}\mathbf{c}_t + \mathbf{b}_o\right), \\
\mathbf{h}_t &= \mathbf{o}_t \tanh(\mathbf{c}_t).
\end{aligned} \tag{2}
$$

4 Results

4.1 Autoencoder

The decoded inputs from the autoencoder reveal that it is unable to learn the locations of the discontinuities according to Figures 8. This occurs even though the model converges for both the training and validation data. This has implications for the LSTM network, which will also be unlikely to learn the locations and therefore will not converge well or accurately predict the future. Instead of learning the discontinuity location, the network tends to learn a smooth average value for the inputs. For smooth CFD solutions, this may not present an issue, but for any flows of engineering interest involving shockwaves, this is a considerable hurdle to overcome. Most likely, the network is unable to learn the location of the discontinuity because it knows no information about the node locations in space. It does not know that nodes to the left or right are even nearby and thus useful for determining the current node value.

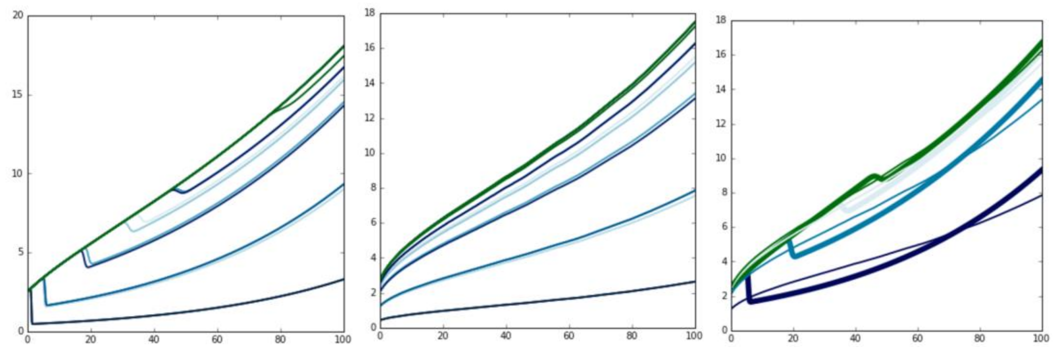

Figure 8: Sample inputs (left, bold), decoded inputs (middle, thin), and overlay (right)

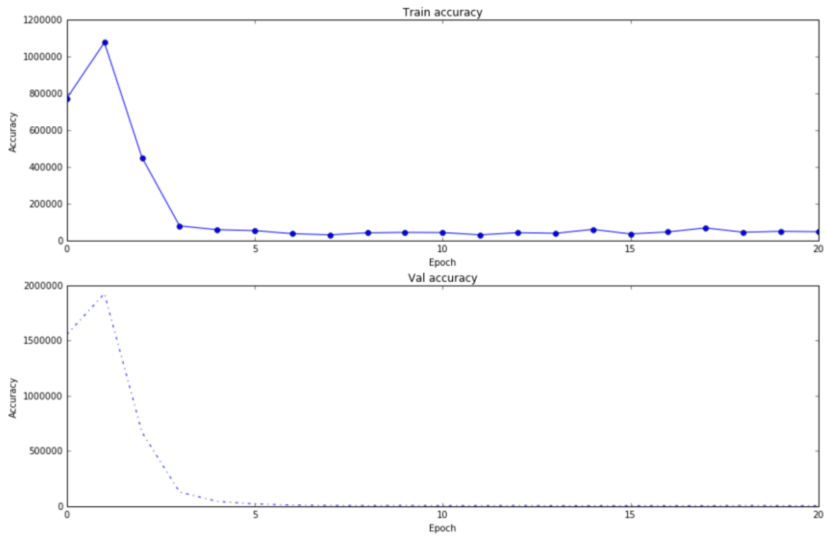

Figure 9: Autoencoder training and validation accuracy loss

To improve the autoencoder, several methods were explored, including changing the number of hidden layers – one, two, and three layer models were tested for both encoding and decoding. Also, the number of nodes in the hidden and encoded layers were changed, including very small representation sizes like 5 and 15, to large representations equal to the size of the input set of 1000 and greater >1500. Regardless of the architecture and number of nodes, the autoencoder continued to learn only a smooth approximation of the function.

4.2 LSTM Network

Because the autoencoder was unable to learn the locations of the discontinuities, the encoded representations were not used in the LSTM network. Instead, the full Burgers' equations solutions were input into the LSTM network to test whether the network alone could learn how to feed forward in time. The training is slow, as expected, since it is off the original design, but the results may reveal more about how to modify the current implementation going forward.

The training history loss for the LSTM network can be plotted to check if the model is converging. Figure 10 shows the training history loss given the full dataset, with a learning rate of 0.02, a learning rate decay of 0.995, a batch size of 1, and number of epochs equal to 300. The jumps in loss are expected, since there are only two set of 501 time steps in the training set, and each will have its own associated loss with a given network. A successful model, however, once fully converged, should no longer show such pronounced jumps. There should also be a much smaller loss relative to the initial loss (the loss given random weights). From the loss plot alone, it's possible to see that the network has not yet learned the representation well by the final epoch.

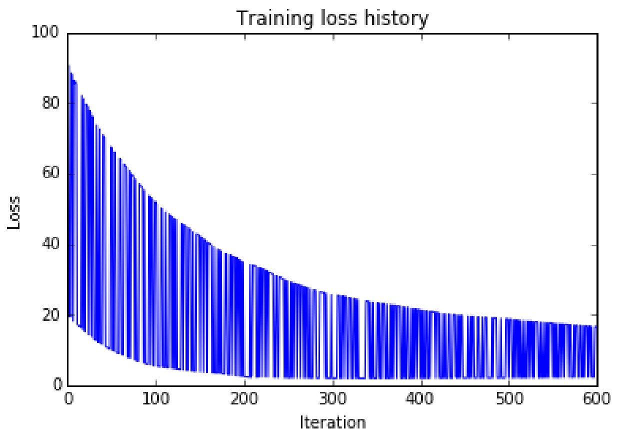

Figure 10: Training loss history for preliminary model

The network can output both (1) reproductions of the original solutions and (2) predictions for new solutions at different hyperparameters, which in this case would be single point in the validation set. Figure 11 shows both the reproduction of the original solutions and the prediction for the validation set. The network has not yet been optimized, and it may have some implementation errors, so the original solutions are poorly reproduced, and so, predictably, are the predictions for the validation set. Most likely, the results are poor for the same reasons that the autoencoder is not able to learn the function – it is unable to learn the location of the discontinuities, and therefore predicting the future given poor representations of the present is an even more difficult problem.

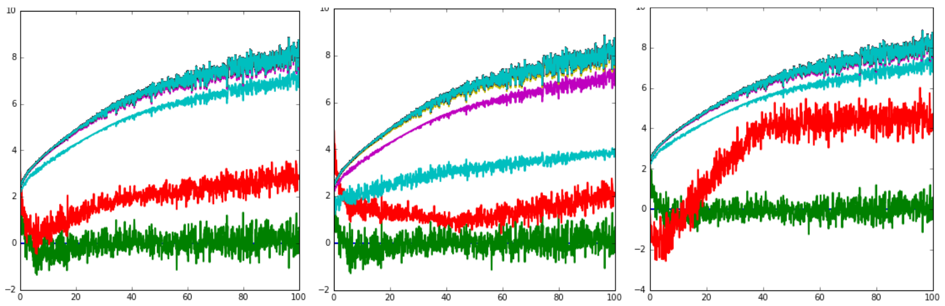
Figure 11: The training set (left two images) and the validation set (right image)

Although the current solutions are extremely poor representations of the original solutions, there is no reason why a neural network with a different architecture cannot be trained to be reasonably accurate. There are many possibilities for why the implementation is not yet performing well. The following modifications were attempted, but did not result in improvements to the preliminary results: (1) regularization (2) dropout (2) sigmoid nonlinearity instead of relu and (4) loss function calculated with square of L2 norm instead of L2 norm. Several possibilities will be explored, outlined in the next section, until a predictive system is achieved.

5 Alternative Models and Future Work

Given the inaccurate predictions of the original model, three additional models are proposed for future work – Model 1, Model 2, and Model 3.

5.1 Model 1: More Hidden Layers in LSTM network

Model 1 combines encoding with the LSTM network architecture. It is the same as the original LSTM network architecture, except additional layers have been added, which are equivalent to encoding the LSTM inputs and decoding the LSTM outputs. It's possible that the LSTM may remember the locations of the discontinuities in time if it is fed correctly encoded data.

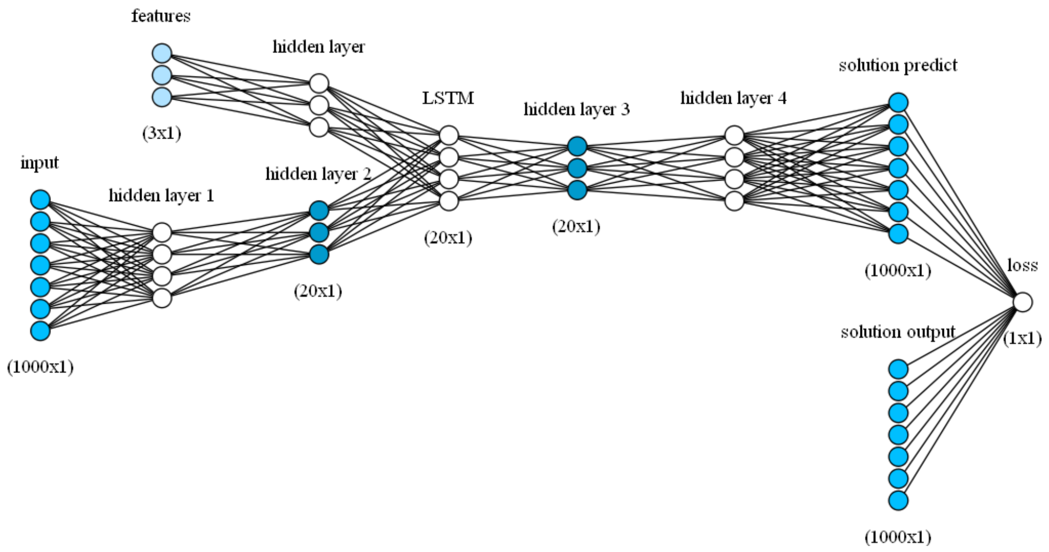

Figure 12: Model 1 diagram

Although additional testing will need to be completed, preliminary results for Model 1 are not promising. Model 1 converges much like the original model, producing smooth results that do not "see" the discontinuity, much like the initial autoencoder results. Figure 13 shows the convergence and results of the initial Model 1 implementation.

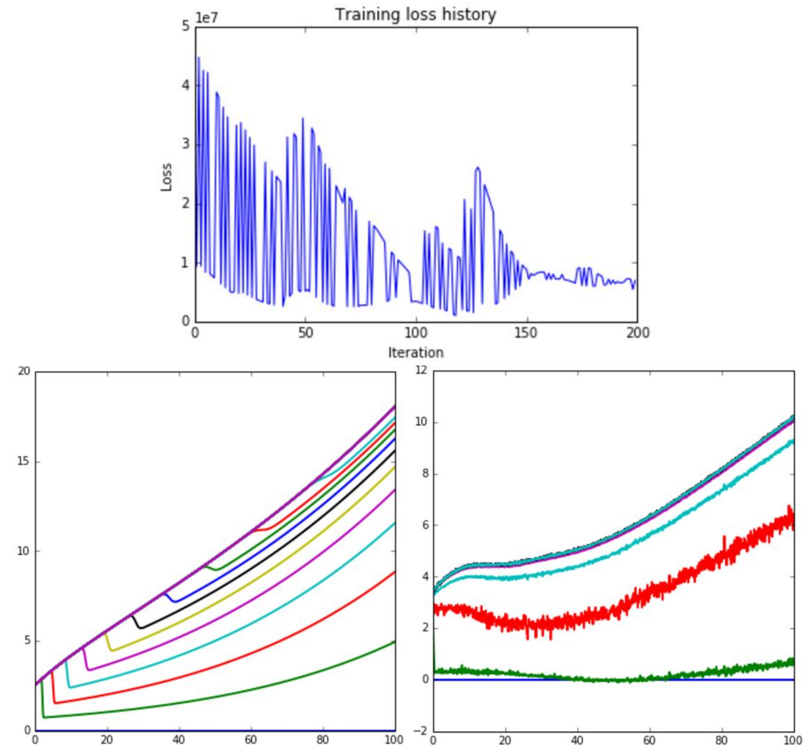

Figure 13: Model 1 convergence (top), sample inputs (left) and decoded outputs (right)

5.2 Model 2: Local Receptive Field in Autoencoder

While a convolutional neural network may appear to be an ideal candidate to address this problem because it forces nearby nodes to be connected to each other, the inputs in CFD cases are not spatially invariant, which is a requirement for a convolution. Figure 14 visually demonstrates how a CFD grid differs from an image and is therefore not an ideal candidate for a convolution.

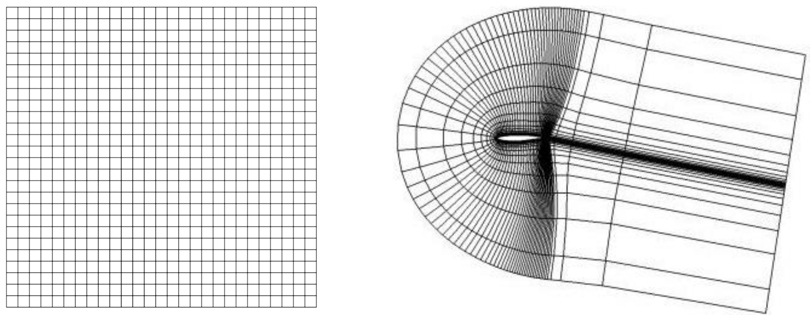

Figure 14: Spatial variation in a grid of pixels (left) compared to a CFD grid (right)

Therefore, instead of a convolution, a local receptive layer is proposed to add to the first and last hidden layer of the autoencoder as shown in Figure 15. This layer would be insensitive to spacing, because the weights in the local receptive layer are free to vary in space, while the weights in a convolution are forced to be the same regardless of their location.

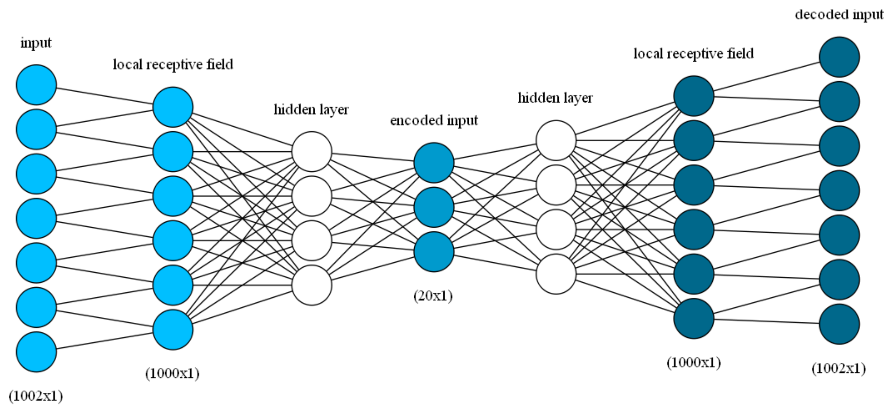

Figure 15: Model 2 diagram

5.3 Model 3: LSTM Over Both Time and Space

In this model, instead of feeding the network an entire vector across space, only a few nodes (2-3) are fed into the LSTM at a time, as shown in Figure 16 (far left). With this architecture, the LSTM memory may "remember" the discontinuity locations as if they were discontinuities in time. The downside of this method is that it may suffer from the same problem as a convolution since the same weights will be applied at each spatial step within this architecture.

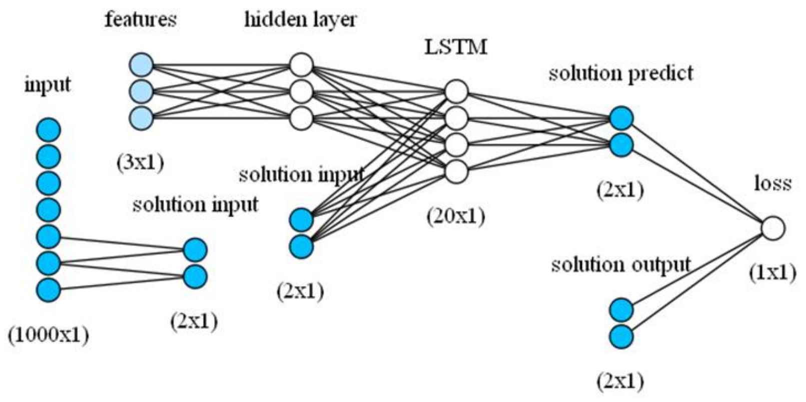

Figure 16: Model 3 diagram

6 Conclusions

While the original model was unsuccessful at predicting future time steps, there is no reason why neural networks cannot be used to predict the results of PDEs for which the governing equations are known, but time intensive to compute. The data was compressed using an autoencoder, but the compression was computed in such a way that important information was lost - specifically the discontinuity locations. The newly proposed models may resolve the issues encountered in the original model, and if so, many engineering problems of interest to industry could be solved efficiently.

References

[1] Guo, Xiaoxiao, Wei Li, and Francesco Iorio. "Convolutional Neural Networks for Steady Flow Approximation."

[2] Srivastava, Nitish, Elman Mansimov, and Ruslan Salakhutdinov. "Unsupervised learning of video representations using lstms." CoRR, abs/1502.04681 2 (2015).

[3] Tompson, Jonathan, et al. "Accelerating Eulerian Fluid Simulation With Convolutional Networks." arXiv preprint arXiv:1607.03597 (2016).