Simulating Euler Fluids with Neural Nets

Alvaro Zamora, Andrew Eberhardt

September 23, 2019

1 Motivation

Fluid simulations are ubiquitous in physics. New simulation techniques that run efficiently and accuracy are of great interest. However, the community currently regards many machine learning techniques with suspicion. With this is mind we attempt to demonstrate a convolutional neural network that can act as an accurate and fast fluid simulation tool. This network would be given the state of a fluid and one time and predict the state at a future time.

2 Previous Work

Previous work by Google [1], Pixar [2], and Xie et.al. [3] shows promising results on the use of convolutional networks and GANs as accelerated solvers. The work in [2] inspired our use of an autoencoder to access a lower-dimensional space in which we can evolve the state forward. However their exact approach is not appropriate for our application because they only considered parameterizable fluid simulations. The work in [1] describes a clever way to predict the pressure to then update the grid, but is also not appropriate for our application as it focuses on incompressible fluids. Although the work in [3] generates temporally coherent results from a physics-based approach, the fact that every generated example needs to be tuned is unacceptable for our application.

Enzo, a state-of-the-art AMR solver [4], inspired the solver we used to generate the examples. In particular, we adopted their use of the HLL interface flux and used a piecewise linear reconstruction method with an RK3 time integrator. To generate resolution-stable initial conditions for our Kelvin-Helmholtz metric, we adopted the strategy in [5] of using a ramp function to capture the fluid interfaces.

3 Objective and Metrics

Our motivation merits tests of two types of objectives. **The first will be to demonstrate the speed and accuracy of the neural net as an alternative fluid solver**. We will measure the success of this objective with the following metrics

1. The extent to which the error of the prediction be minimized when compared with conventional solvers (accuracy)
2. The extent to which the simulation can be carried out with fewer operations and less memory (efficiency)

Our second objective will **demonstrate the ability of the solver to make phenomenologically correct predictions**. This will be accomplished by training the net on examples of fluid initial conditions that should not create Kelvin Helmholtz instability (a common fluid instability, e.g. ocean waves, clouds, etc) and then testing the net on initial conditions that should create a Kelvin Helmholtz instability. The ability of the net to predict the existence of this phenomena without being shown it will then be evidence that it is accurately representing the physics of the system. We will measure the success of this objective with the following metric

1. The extent to which the net shows evidence for Kelvin Helmholtz instability for the appropriate initial conditions (prediction)

4 Data Generation

Data will consist of a set of initial and final conditions (fig 1). An initial/final condition pair will be written x^t, x^{t+dt}. Where each represents the conditions of four real valued fields (N by N grids) corresponding to relevant fluid quantities (density, momentum density, and energy), i.e. $x^t, x^{t+dt} \in \mathbb{R}^{4\times N\times N}$.

As mentioned above the **initial conditions** will be generated such that they will not explicitly contain Kelvin Helmholtz instability. We would like to train the network on a large variety of initial conditions covering a range of length scales. In order to achieve this we generate Gaussian random fields from random isotropic power spectra. Fields corresponding to a single data point are generated from the same power spectrum such that all quantities for a given data point vary on the same length scales.

Final conditions would be generated using a traditional fluid solving technique (E) that advances initial conditions using Euler's equations. Final conditions would be produced for some fixed number of time steps in advance of the initial conditions. i.e.

$$x^{t+dt} \equiv E(x^t)$$

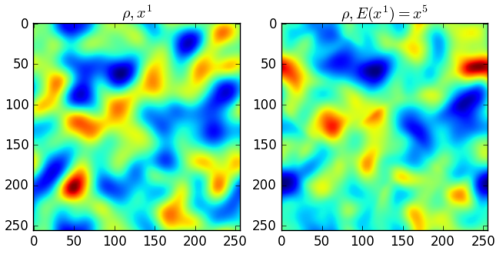

Figure 1: A pair of initial and evolved conditions for some x at $t = 1$ and $t = 5$. The density is the field being plotted.

5 The Network

5.1 Structure

The structure of our net will contain three main components which act as mappings as follows (fig 2)

1. encoder (ϕ). $\phi : x \to f$
2. decoder (ψ). $\psi : f \to x$
3. simulator (S). $S : f \to f'$

The job of the encoder is to map initial or final conditions at a given time (x^t for conditions x at time t) to a feature vector representing those conditions (f^t, for features representing conditions x at time t). The decoder then maps feature vectors to the conditions that produced them (i.e. it is the inverse mapping of ϕ). The simulator then takes feature vectors corresponding to a set of conditions at time t to a feature vector representing conditions at time $t + dt$. The mappings then are given as follows

where specifically

$$\phi(x^t) = \hat{f}^t \qquad \psi(f^t) = \hat{x}^t \qquad S(f^t) = \hat{f}^{t+dt}$$

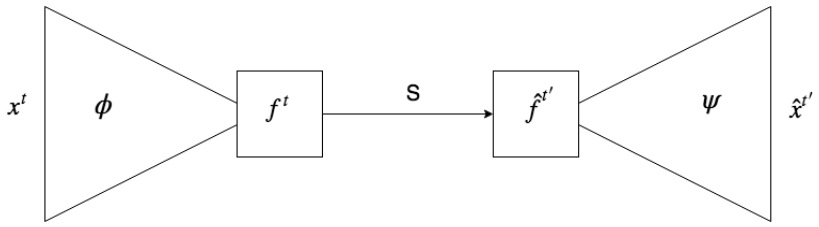

Figure 2: A layout of the net structure. ϕ, S, ψ are the components of the actual net. Read left to right we start with a data point x^t and compress it using the compressing portion of the net (ϕ) to f^t. We then evolve our compressed data point to some new time $\hat{f}^{t'}$ (representing the prediction of the compressed verion of the evolved data point $x^{t'}$) using the simulator portion of the net (S). We then use the decompressing portion of the next (ψ) to recover the predicted evolved conditions $\hat{x}^{t'}$

Where hats indicate predicted values. The network then functions as a map from conditions to predicted conditions ($C : x \to x'$)

$$\psi(S(\phi(x^t))) = \hat{x}^{t+dt}$$

The accuracy metric can then be evaluated as the degree to which x^{t+dt} and $\hat{x}^{t+dt}$ differ (where we are trying to minimize the difference). The efficiency metric can be evaluated as the extent to which the run time and memory cost for the operation $\psi S \phi$ is less than the operation E. This involves both reducing the size of the feature vector f and the complexity of the net. We define a value $R \equiv (size\ of\ x)/(size\ of\ f)$ to represent the reduction in size the a given net achieves.

Finally, the prediction metric can be evaluated by showing the trained net initial conditions that should generate Kelvin Helmholtz instability and looking for the instability in the output.

5.2 Training the encoder-decoder net

The first step in producing our net is to train the encoder and decoder. Notice that a well trained encoder and decoder should produce the following results

$$\psi(\phi(x^t)) = \hat{x}^t \approx x^t$$

i.e. the encoder and decoder should reproduce the identity mapping ($\psi\phi \approx I$). Therefore we can train the the encoder and decoder on our set of conditions (either initial or final) where the truth value is simply the input conditions (fig 3). At this stage we try and maximize the compressing ability of our network. We can split the combined network $\psi\phi$ at its narrowest point (i.e. the layer with the fewest outputs).

This output represents the encoded data point. Note that our simulator (S) will only advance encoded data meaning that the larger we make R, the more efficient our solver will be. Though it seems reasonable to suspect that a larger R will produce less accurate results.

Once the encoder and decoder are trained to approximate the identity we can train the simulator using the compressed initial and final conditions. At this time we kept initial and final conditions at some fixed time step apart. We trained a map from the first to the fifth time step. i.e.

$$S : f^1 \to f^5$$

where the training data and truth values were f^1 and f^5 respectively. The simulation net includes flattening and 1-dimensional convolutional layers. The net was trained in sequences. We started with a simple architecture and then slowly added new layers overtime redoing the training at each layer. Results of this net in fig 4.

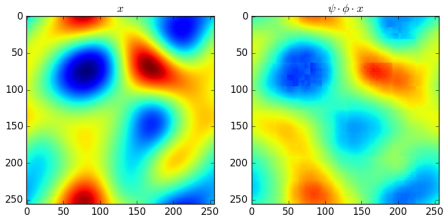

Figure 3: Left: some data point x. Right the result of encoding and decoding $\psi \cdot \phi \cdot x$ the data point with a net for which the compression factor is $R = 16$.

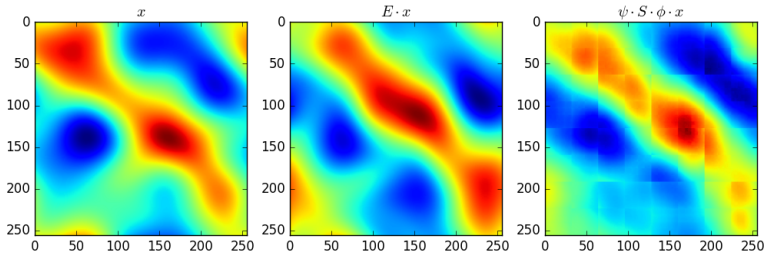

Figure 4: Left: Initial conditions for some data point. Middle: final conditions for the same data point. Right: the result of our nets prediction given the initial conditions.

5.3 Training a complete net

It is instructive to look at the results of nets trained where $R = 1$. This has the disadvantage that no compression is done and so you do not get a greater memory efficiency when performing the training. However, it is still possible for this method to be a more time efficient solver and potentially a more accurate one as well.

$$C : x^t \to x^{t'}$$

5.4 Loss function

The loss function used was the DSSIM loss. Given a kernel size K, the loss function breaks up the target and predicted image into patches of size KxK and compute the mean of the following quantity over the patches:

$$\text{DSSIM} = \frac{1}{2} - \frac{1}{2}\frac{(2\mu_1\mu_2 + c_1)(2\sigma_{12} + c_2)}{(\mu_1^2 + \mu_2^2 + c1)(\sigma_1^2 + \sigma_2^2 + c_2)}$$

where μ_i is the mean of the patch and σ_i^2 is the variance for either the target or label image ($i = 1, 2$), and σ_{12} is the covariance of the target and label patches. Here, c_1 and c_2 are nudge factors to prevent division by zero.

6 Evaluation of metrics

With respect to our accuracy metric, we investigate the relationship between the prediction error and length scales in the input (for which entropy is a proxy). The encoder-decoder net struggles to successfully compress

high entropy (information dense) conditions. While the complete net struggles at predicting low entropy (large length scale) conditions (Fig 5). This is probably due to the difficulty inherent in encoding a more information dense system. Likewise the complete net probably fails at large length scales because it is better at representing the lengths scales of the kernels in the convolutional layers which have some maximum size.

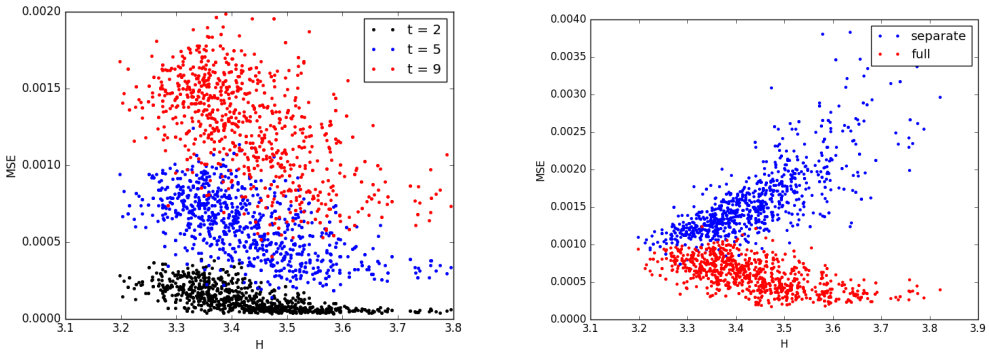

Figure 5: Left shows the error on iterative predictions as a function of entropy for 1 iteration, 4 iterations, and 8 iterations. Right shows the error on predictions as a function for entropy for a complete and encoder-decoder nets.

With respect to our **efficiency** metric, the compression factor we were able to achieve for our compressed conditions was $R = 16$ and still maintain a fairly high level of accuracy. The trained neural network produces results over 100 times faster on a GPU than a traditional solver running in parallel on 4 cores.

With respect to our **prediction** metric, the net failed to predict Kelvin Helmholtz instability. Despite the fact that the initial conditions producing KHI contained values and length scales that were within the range our nets were trained on the net failed to reproduce KHI. (Fig 6)

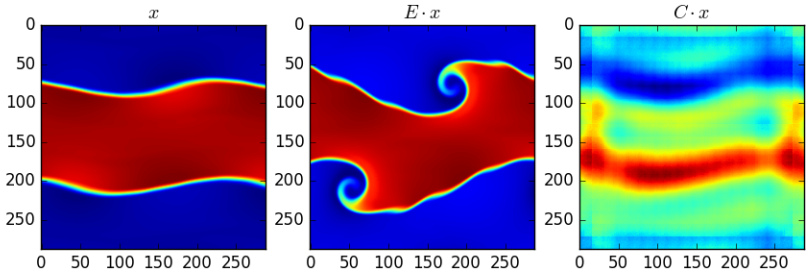

Figure 6: Left: Initial conditions producing KHI. Middle: final conditions containing clear KHI. Right: Our nets prediction on these initial conditions.

7 Conclusion and future work

We had a reasonable deal of success according to each metric except for our prediction metric. The most logical place to proceed with this work is to figure out how we may satisfy this goal without simply showing it the problem we are trying to predict, i.e. how similar do we need to make the train problems before it can handle this test? Additionally, we have a lot of room to make larger nets with the results of our efficiency metric being satisfied to the degree it is. This means we could experiment with a lot of new and more complex architectures.

8 Contributions

Andrew Eberhardt: Wrote a code that produced initial conditions (x^0). Worked on training/designing the simulator (S) structure we ended up using. Alvaro Zamora: Wrote a code that evolved the initial conditions (E). Worked on training/designing the encoder decoder net ($\psi\phi$) structure we ended up using. Both: We both worked on several many attempted structures for every part of the nets (ψ, ϕ, S) many of which we did not use. We also both worked on training complete a few different complete nets (C). Also dozens of utility and support scripts had to be written which was done by both of us.

Code link: https://github.com/expwnential/MLProj

References

[1] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating Eulerian Fluid Simulation with Convolutional Networks," *CoRR*, vol. abs/1607.03597, 2016.

[2] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep Fluids: A Generative Network for Parameterized Fluid Simulations," *ArXiv e-prints*, June 2018.

[3] Y. Xie, E. Franz, M. Chu, and N. Thuerey, "tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow," *CoRR*, vol. abs/1801.09710, 2018.

[4] G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J.-h. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C. So, F. Zhao, R. Cen, Y. Li, and Enzo Collaboration, "ENZO: An Adaptive Mesh Refinement Code for Astrophysics," *The Astrophysical Journal Supplement Series*, vol. 211, p. 19, Apr. 2014.

[5] B. E. Robertson, A. V. Kravtsov, N. Y. Gnedin, T. Abel, and D. H. Rudd, "Computational Eulerian hydrodynamics and Galilean invariance," , vol. 401, pp. 2463–2476, Feb. 2010.