2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics

Solving Level Set Evolving Using Fully Convolution Network

Ran Wei*, Futing Bao, Yang Liu, Weihua Hui
School of Astronautics, Northwestern Polytechnical University
127 West Youyi Road, Xian Shaanxi,710072,P.R.China
Email: *RanWei.NWPU@outlook.com

***Abstract*—We have designed and trained a fully convolution neural network to calculate geometry evolving in discontinuous velocity fields. Train data set is generated by numerically solving level set equations. The equation-based regression problem is innovatively transformed to a series of classification problem to make the training easier. A network structure similar to deep residual net is used to improve accuracy. Test shows that our network is 18x faster compared to standard level set method, and error bound of our work is 0.5% for single frame prediction, and 9% for iterative prediction. Accuracy may be further improved by using priori knowledge or introducing generative adversarial network.**

***Index Terms*—CNN; Level set; Partial differential equation**

I. Introduction

Level Set Method (LSM) has been widely used in fields such as Computational Fluid Dynamics (CFD), earthquake simulation and computer vision. The method maps an evolving surface in $\mathbb{R}$ dimension space to an isoline in $\mathbb{R}+1$ dimension space. High-dimension mapping provides LSM excellent compatibility. Merging and splitting of geometries are perfectly handled. On the other hand, this also results in huge computational needs. To compute evolving of an $\mathbb{R}$ dimensional geometry, we have to process $\mathbb{R}+1$ dimensional volume data. Evolving of a high-resolution 3D surface may take hours even on GPU.

While solving LSM equations requires a large computational resource, the final effect of the equations is not complicated: the surface of geometry moves inward or outward at local defined propagating speed. Essentially, propagation of surfaces is a graphics problem, and humans can easily solve this on grid papers. This implies that artificial intelligence may provide an agile route to solve the problem. Viewed from a different angle, surface evolving problem is actually predicting the next image frame using the previous frame.

There has been a number of attempts to solve Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs) using artificial intelligence methods. In [1], the authors tested feed forward neural networks on several ODEs and PDEs. According to their test, neural network gives even better results than finite element method, on both accuracy and efficiency. In [2], neural network and stochastic simulation were combined to solve viscoelastic flow problem. In [3], the authors tried to solve Stokes problem using neural network. The error bound of their work was in the order of 10^{-5}, which make it possible to be used in optimization systems. In [4], the authors used regress forest to perform Smoothed Particle Hydrodynamics (SPH) simulation. Interactive real-time CFD simulation was achieved using the model they have trained. In [5], an end-to-end Convolutional Neural Networks (CNN) was trained to compute steady flow field for cars.

There are basically two advantages to use machine learning rather than numerical methods. First of all, efficiency of trained neural networks is much higher than directly solving equations. Most trained artificial intelligence model only requires constant execution time, while numerical solving often relies on iterative solving until the output converges. Secondly, it is unlikely to trigger numerical divergence in trained models. The users can get rid of tuning CFL or other computing parameters, which is often empirical.

In this article, we tried different ways building a neural network to regress level set equations. Training data was generated by directly solving level set equations with randomly generated initial states. Our trained model accurately predicts the shape of propagating geometry at next time step, costing only $1/18$ the execution time of directly solving.

II. Method

A. Level set method

LSM uses Signed Distance Field (SDF) to describe geometry. SDF is defined as:

$$\phi(\vec{x}) = S(\vec{x})D(\vec{x}) \tag{1}$$

where $D(\vec{x})$ is the nearest distance from $\vec{x}$ to target geometry, and $S(\vec{x}) = 1$ when $\vec{x}$ is outside the geometry and $S(\vec{x}) = -1$ when $\vec{x}$ is inside. The $\phi = 0$ isoline is the surface of the target geometry.

In the same space we define velocity field $\vec{v}(\vec{x})$. It is the velocity vector which indicates movement of geometry surface. $\vec{v}(\vec{x})$ means that if point $\vec{x}$ is on geometry surface, the infinitesimal surface facet near $\vec{x}$ moves at velocity $\vec{v}$.

In [6], an equation was proposed to calculate ϕ after one infinitesimal time step:

$$\phi_t + V_n \cdot |\nabla\phi| = 0 \tag{2}$$

where V_n is the normal component of $\vec{v}(\vec{x})$. A positive V_n indicates the surface is moving outward and negative V_n means inward. $\nabla\phi$ can be calculated following WENO5 scheme [7] and Godunov's scheme [8]. As LSM is a mature technology,

978-1-5386-3022-8/17 $31.00 © 2017 IEEE
DOI 10.1109/IHMSC.2017.61

trivia details about solving equation 2 will not be described in this article.

B. Data generating

Training a machine learning model requires a large amount of data. Training data shall have the same probability distributions with predicting case. LSM applies to so many fields that it is impossible to train a single model which fits with any case. In this article, we focus on 2D cases with the following properties:

1) Target geometry can be any shape in any position.
2) $V_n > 0$ all over the space.
3) The space is divided into two areas. Inside each partition, V_n is uniform.
4) The ratio of two values of V_n shall be in the interval $[0.5, 0.8]$.
5) Shape of partitions can be any closed polygon. One partition contains another is allowed.

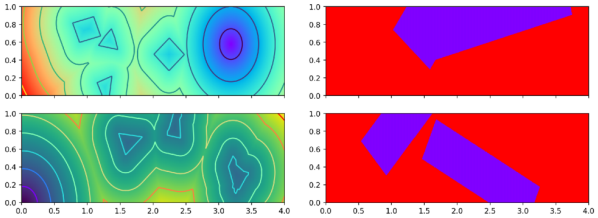
We take the above case as example because it is important in solid rocket simulation, where different solid propellants burn and ignite each other (flame front can be regarded as an expanding geometry). Even though the work presented in this article is focused on the above case, the training process and net structure below can be easily promoted to other cases.

In general machine learning tasks, training data are often collected and labelled by humans. To train an equation solver, however, the data can be automatically generated from actual equation solving. All we have to do is to produce random initial conditions and boundary conditions, and record the output of numerical solving. Specifically for LSM, we shall generate random initial geometry (represented by ϕ) and random velocity field (represented by V_n). Here is the procedure to generate a random initial SDF:

Procedure 1 Building a random initial SDF

Input: space dimension, mesh dimension, shape templates
  Discretize space to grid point $\vec{X}$
  Initial $\phi = \infty$
  for Random number $\in [4, 8]$ **do**
    Fetch a random shape R_s from shape templates
    Perform random transformation on R_s
    Calculate SDF $\phi_{rs}(\vec{X})$ of R_s following equation 1
    for all $\vec{X}$ in field **do**
      $\phi(\vec{X}) = \min(\phi_{rs}(\vec{X}), \phi(\vec{X}))$
    end for
  end for
  return $\phi(\vec{X})$

In this article, *space dimension* in above procedure is normalize to $[0, 4], [0, 1]$ on x, y axis respectively, *mesh dimension* is $[1024, 256]$ on x, y axis respectively, and *shape templates* contains various triangle, rectangle, polygon and circles. Each transformed shape has it's owns SDF, and $\phi(\vec{X}) = \min(\phi_{rs}(\vec{X}), \phi(\vec{X}))$ operation is actually a boolean *and* operation to join them. Procedure 1 is proved to ensure enough randomness in generated data set.



(a) ϕ (b) V_m

Fig. 1: Sample ϕ and V_n

Velocity field V_n is generate using a similar procedure, of which details will not be repeated. In figure 1 we present two examples of generated ϕ and V_n.

Once the initial status is built, iteratively apply equation 2 on ϕ, and record each status of ϕ. Then we can use the $(n+1)-th$ frame as the label of $n-th$ frame. A frame sequence of length N provides us $N-1$ train data items. As the generated data set is large, we suggest using a database to store them.

C. Net structure

1) layers: Obviously, LSM evolving relies on local information. From equation 2 it is easy to see that updating $\phi(\vec{x})$ only needs local $V_n(\vec{(x)})$ and $|\nabla\phi(\vec{x})|$. V_n is predefined and can be read directly, while calculating $|\nabla\phi|$ requires ϕ values from a range of ± 4 grids on each axis. As a result, predicting a single node in next frame exactly needs information from its neighboring nodes (radius = 4). Further nodes are completely of no use.

Convolutional neural networks are designed to discover local rules and reuse them among the whole field. Since we only need local information, there shouldn't be any dense (fully connected) layer all over the entire network. Therefore, we decided to use Fully Convolutional Network (FCN) on this problem.

One key difference between our LSM solving network and commonly used neural networks is that we need the output to be as accurate as possible on every pixel. Capturing advanced features requires deep network, but during feeding forward, the original input will be blurred. Under this occasion, it is hard for the rear layers to get exact information about the input. To solve this, a "short cut" is needed to transfer information directly to rear layers. We used a net structure similar to ResNet [9] to do the work: add the input of the $n-th$ layer to activation output of the $(n+1)-th$ layer. Structure of this Res-like module (referee as "R-block" below) is shown in figure 2. All these modules have a dropout ratio of 0.4 during training.

Usage of R-block ensures that the distance from most layers to original input are not further than 2 layers. Detailed net structure is shown in figure 3.

In convolutional networks, a common method to reduce network dimension is using pooling layers, which select evident features and abandons non-significant ones. Pooling

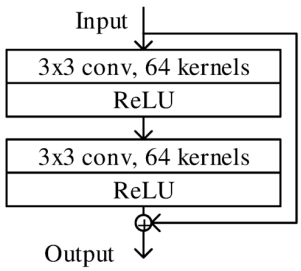


Fig. 2: Structure of R-block

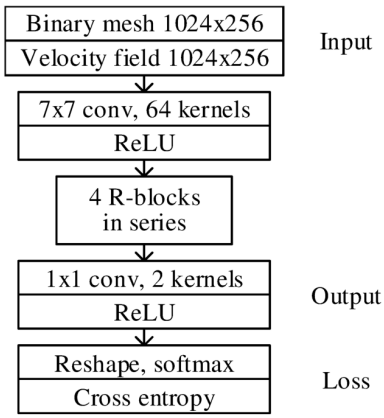


Fig. 3: Net structure

operation often works well in classification networks, because classifying images requires highly abstract representation to make decisions. In this article, however, we haven't used any pooling layer, because our output is an image which is exactly the same size with network input. Pooling or downsampling results in losing information, but in our problem, each node is of the same importance. It is also difficult to reasonably recovering image size, which is reduced in pooling operation. The problem about pooling is another reason to chose FCN, for any dense layer would be at least of size $(1024\times256, 1024\times256)$. Such a dense layer would be difficult to train, easy to get overfitted and become performance bottleneck of the entire network.

2) Loss function: The next step to design a reasonable loss function. Loss function is the key factor which actually tells net optimizer what we want. Machine learning problems can be divided into two categories in general: regression and classification. Loss function of each category has their own characteristics.

Intuitively, solving PDEs such as LSM equations is a regression problem, so we can simply use ϕ of next frame as label, and $(\phi_{predict} - \phi)^2$ as loss function. However, our research shows that regression model has poor performance on predicting an SDF. Reason for this is, processing SDF requires higher accuracy than ordinary neural networks can achieve. Take our problem as example, we have placed 256 grids in $[0, 1]$ range, which means the maximum possible difference between values of ϕ on two neighbouring nodes is $l_g = 1/256{\approx}0.0039$. CFL condition limit propagation amount under l_g. Suppose CFL factor is 0.8, the amount of change after one time step would be $0.8l{\approx}0.0031$. To approximate equation solving, the difference between equation output and network output shall be at most $0.8l/20{\approx}1.55e-4$. The required error bound is close to the precision limit of ordinary 32-bit float variable $(1e-6)$. If we take reinitialization [10] into account, required error bound would be even more stringent. Neural network is essentially an approximation method, training a network having such accuracy is extremely difficult.

From another perspective, our purpose is predicting shape of target geometry instead of obtaining an exact SDF. Other than SDF, a commonly used discretized description for arbitrary shape is binary mesh: for each position $\vec{x}$ in the space, simply record whether it is inside the geometry. By converting SDF representation to binary representation, we have succeeded in translating the regression problem into classification issues. Accuracy of binary representation is naturally lower than SDF representation, but the accuracy loss can be easily compensated by increasing mesh dimension.

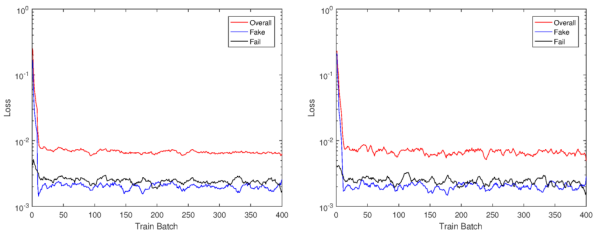
For each instance, our net has 1024×256 objects to classify. The number is much larger than ordinary neural networks, and target objects are organized by pixels, so we need to make some transformation to get the network eventually working. Net input contains two fields (binary mesh and V_n) of the same dimension, and we encode them into a two-channel image. Corresponding label only contains binary mesh, and we convert it to one-hot format, and again we have a two-channel image. The output layer in figure 3 outputs an array of size $(1024, 256, 2)$, we then resize both this array and label to $(1024\times256, 2)$, and calculate softmax - cross entropy loss function on each pixel. Now we get a (1024×256) sized array which contains cross entropy loss of each pixel. By taking their average, we have a scalar loss value, which can be used by any optimizer.

III. Result

A. Training

In order to make the difference between two velocity partitions more significant, we chose $n-th$ and $(n+10)-th$ frame as a training data item. Following this rule, we have got 38052 valid training instances. 8% of these instances are randomly selected as testing data set. Batch size during training is set as 8. Bigger batch may result in better results, but it is impossible to apply bigger batch size on the authors' K80 GPU, due to GPU RAM limitation.

We have defined three metrics to monitor training: overall loss L_o, fail-true ratio R_{fail}, and fake-true ratio R_{fake}. L_o is simply the value of averaged cross entropy loss. R_{fail} is the ratio of nodes which should have been predicted as "inside" but failed. R_{fake} is the ratio of nodes which shouldn't have been predicted as "inside" but actually did. Define two boolean arrays $T_i(\vec{x})$ and $T_l(\vec{x})$ as following:



(a) Training (b) Testing

Fig. 4: Loss

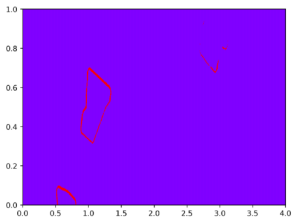


Fig. 5: Predict test

$$\begin{aligned} T_i(\vec{x}) &= \phi_{input}(\vec{x}) < 0 \\ T_l(\vec{x}) &= \phi_{label}(\vec{x}) < 0 \end{aligned} \tag{3}$$

Then R_{fail} and R_{fake} can be calculated following equation 4:

$$\begin{aligned} L_{per-node}(\vec{x}) &= NetOut - Label \\ R_{fake} &= \frac{1}{\sum \neg T_i} \sum (\neg T_i(\vec{x})) L_{per-node}(\vec{x}) \\ InBand(\vec{x}) &= \neg T_i(\vec{x}) \wedge T_l(\vec{x}) \\ R_{fail} &= \frac{1}{\sum InBand} \sum InBand(\vec{x}) L_{per-node}(\vec{x}) \end{aligned} \tag{4}$$

where values of $T_i(\vec{x})$ and $T_l(\vec{x})$ were converted to 0 for *False* or 1 for *True*.

In figure 4 we have shown the three metrics during training and testing. It easy to see that training converges fast, and all three metrics dive to less than 0.5% rapidly. On testing dataset the metrics are almost the same with training, indicates that our net has strong generalization ability.

B. Testing

1) Accuracy: We then applied our trained network on several newly generated shapes. None of them had appeared in training or testing. Figure 5 shows the comparison of network output and standard LSM result (wrongly predicted pixels are shown in red).

Obviously, only a few pixels near the boundary were incorrectly predicted. In order to further validate ability of our network, we iteratively use net output as the next input, to see if error accumulates during iterative predicting. In figure 6 we have plotted R_{fail} and R_{fake} during this iterative procedure.

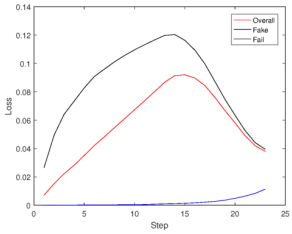


Fig. 6: Loss function in iteratively testing

TABLE I: Efficiency comparison

	Time per step /s	CFL	Equivalent speed
LSM	9.3461993	0.8	1
FCN	5.165529	8	18.0934

The error metrics increases slightly as iteration number grows, especially R_{fail}. However, the overall loss kept under 9%. As the maximum allowed iteration number is less than $256/8 = 32$ (we have 256 nodes on y axis, and each iteration the geometry expands approximately 8 grids), overall error would be under control. An error bound of 9% is acceptable to be used in rapid and rough simulation systems (optimization systems, for example).

2) Efficiency: The fundamental purpose of our research is looking for an efficient route to perform LSM evolving with acceptable accuracy. We have tested the efficiency of our network and compared with original LSM. The test is performed on python/tensorflow/CPU platform. Batch size is set as 1 because the main using environment of our network is iteratively predicting. As training data is $n-th$ and $(n+10)-th$ frames in a sequence generated by LSM, equivalent CFL number of the neural work is 10 times larger than LSM.

Averaged result of testing in shown in table I. From the table it is clear that our route is 18x faster than directly solving. On GPU platform, accelerating ratio of a well designed program may be even bigger, because calculating WENO5 scheme requires complex memory accessing pattern on GPU, while inferring networks only needs standard matrix operation.

IV. Discussion

In this article, we have designed an FCN to efficiently imitate LSM evolving in acceptable accuracy. The reason why regression model gives bad accuracy is discovered, and we then innovatively transformed LSM evolving problem to a classification problem, and designed loss function organization for such a special classification task. Our work evolves geometries in discontinuous velocity fields in 18x faster speed than ordinary equation solving. The error bound is 0.5% for single prediction, and 9% for iterative prediction.

Further work of this research is developing network for 3D case, and applying it to real optimization systems. Processing 3D cases requires 3D convolution, which is a challenge to both net designation and computing power.

Possible methods to further improve accuracy include using priori knowledge and Generative Adversarial Networks

(GAN). In specific problems, some micro structure in output is obviously unreasonable (e.g. an isolated pixel emerges far from main geometry), therefore we can fix them before outputting final result. Such rules shall be designed on the base of priori knowledge about target tasks. For GAN, Predicting a frame can be seen as generating an image, therefore it is possible for our network to benefit from GAN.

REFERENCES

[1] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[2] D. Tran-Canh and T. Tran-Cong, "Computation of viscoelastic flow using neural networks and stochastic simulation," *Korea-Australia Rheology Journal*, vol. 14, no. 4, pp. 161–174, 2002.

[3] M. Baymani, A. Kerayechian, and S. Effati, "Artificial Neural Networks Approach for Solving Stokes Problem," *Applied Mathematics*, vol. 01, no. 04, pp. 288–292, 2010. [Online]. Available: http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/am.2010.14037

[4] L. u. Ladick, E. Zurich, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross, "Data-driven Fluid Simulations using Regression Forests," *ACM Trans. Graph. Article*, vol. 34, no. 9, pp. 2–5, 2015. [Online]. Available: http://doi.acm.org/10.1145/2816795.2818129%5Cnhttp://people.inf.ethz.ch/sjeong/physicsforests/

[5] X. Guo, W. Li, and F. Iorio, "Convolutional Neural Networks for Steady Flow Approximation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. New York, New York, USA: ACM Press, 2016, pp. 481–490. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2939672.2939738

[6] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988. [Online]. Available: http://dx.doi.org/10.1016/0021-9991(88)90002-2

[7] X.-D. Liu, S. Osher, and T. Chan, "Weighted Essentially Non-oscillatory Schemes," *Journal of Computational Physics*, vol. 115, no. 1, pp. 200–212, 1994. [Online]. Available: http://dx.doi.org/10.1006/jcph.1994.1187

[8] S. K. Godunov, "A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics," *Matematicheskii Sbornik*, vol. 89, no. 3, pp. 271–306, 1959.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Arxiv.Org*, vol. 7, no. 3, pp. 171–180, 2015. [Online]. Available: http://arxiv.org/pdf/1512.03385v1.pdf

[10] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-Based Fast Local Level Set Method," *Journal of Computational Physics*, vol. 155, no. 2, pp. 410–438, 1999. [Online]. Available: http://dx.doi.org/10.1006/jcph.1999.6345