Convolutional Neural Network for Fluid Dynamics

Thesis Project

Sylle Hoogeveen
4445082

MSc Applied Mathematics Student

Department of Electrical Engineering,
Mathematics & Computer Science

February 15, 2022

Contents

1 Introduction

1.1 Relevance

1.2 Aim and Objectives

The aim of this study is to construct a single reduced order model for time-dependent incompressible flow that can account for different boundary conditions, material parameters and geometries (computational domains). As test case, blood flow during one heartbeat will be used. The objectives are high (need to specify?) accuracy, measured as difference between the predicted velocity field and simulated velocity field, and speed-up between the OpenFoam simulations and machine learning predictions.

2 | Preliminaries & Related work

In this chapter relevant theory and related research is discussed to set the framework for this study.

2.1 Classic reduced order methods

Besides simplified physics (or operational based reduction methods) most classic reduced order methods (ROMs) are projection-based:

- proper orthogonal decomposition (POD) methods
- reduced basis methods

  Principal Component Analysis (PCA)
- balancing methods(?)

Con of using classic methods are they usually rely on linear basis functions constructed by Singular Value Decomposition (SVD) [REF: deep fluids] or eigenvalue decomposition (PCA). Also they depend on fixed computational domain (fixed geometry), although work has been done on moving objects in the domain [REF: https://arxiv.org/abs/2106.02338] and [REF: http://grail.cs.washington.edu/projects/model-reduction/41-treuille.pdf].

2.2 ML for reducing order

Instead of using classical reduced order methods, non-linear manifolds can be found to represent the model in reduced form by using ML.

- Neural Network (NN) [REF: https://www.sciencedirect.com/science/article/pii/S0045794917313202]
- Physics Informed NN (PINN) / Physics Reinforced NN (PRNN) [REF: https://www.sciencedirect.com/science/artic
- encoder: deep CNN [REF: deepfluids] or
  shallow masked [REF: https://www.sciencedirect.com/science/article/pii/S0021999121007361]

2.3 Time evolution as time series forecasting

Instead using Euler implicit method or other classical numerical methods for time evolution of a PDE solution, we can view this part of the problem as a time series forecasting problem. There are two machine learning techniques for time series forecasting, the previously discussed general NNs and Recurrent NNs. Recurrent NNs can be built with different unit types: standard units, long short-term memory (LSTM) units or Gated Recurrent Units (GRUs). Also RNNs can have a specific architecture called an Encoder-Decoder network with or without attention mechanism. All these techniques will be discussed in the following sections.

2.3.1 Neural Network

The most naive way of predicting future time steps with machine learning is using a standard neural network. With a NN, no assumption on the input data is made and there is no mechanism for 'memorizing' previous input data. However, this approach is still valuable to explore, as making no assumptions on the input data also means the network has all flexibility to learn undiscovered patterns and less complexity in the network is desirable as it is computationally and memory-wise more efficient.

Multiple studies were done using a NN for time evolution in combination with using ML techniques for reducing order. Kim et al. used an autoencoder to create a latent space representation of smoke and fluid simulations and thereafter a NN to find the subsequent latent representation. As input for the NN, they used the latent representation found by the autoencoder concatenated with a control vector difference between user input parameters. The output of the network is the difference between the current latent space representation and the next latent representation. This approach led to significant speedup while maintaining accuracy for a variety of fluid behaviour. However, the ability to reconstruct physically accurate scenarios depended heavily on the how closely the scenarios matched the input training data [8] .

Lopez-Martin et al. created one Deep NN with 3D convolutional layers to reduce dimensionality of velocity fields generated by a synthetic jet simulation. Three dimensions in this case are x,y and time. The final layers of the network were used for time evolution. This was achieved by adding a convolutional layer that controls the final depth dimension of the feature space to be the number of time steps to be predicted. This tensor is reshaped such that it can be broken into the number of time steps to be predicted vectors. These vectors are then used as input to the fully connected layers, which perform the time evolution [9].

2.3.2 Recurrent Neural Network

Recurrent Neural Networks have input, hidden and output neurons similar to a standard neural network. However, the neurons are divided in groups belonging to a fixed time step. The input neurons are connected to neurons in the hidden layer belonging to the same time step. Also the neurons within the hidden layer are fully connected to hidden neurons with the same assigned time step and one way connected to hidden neurons of the next time step. The output neurons are again only connected to hidden neurons belonging to the same time step.

make image representation

Standard Unit

A standard unit or standard RNN cell has two inputs: x_t, the new part of a sequence and h_{t-1} the hidden state of the previous part of the sequence. These are concatenated, multiplied by the weight matrix, added with the bias vector and the results is scaled between [-1,1] by the hyperbolic tangent activation function. The outcome is a new hidden state: $h_t = tanh(W * x_t^\frown h_{t-1} + b)$, which will be used in the next time step. Note that the weight matrix is constant for each time step.

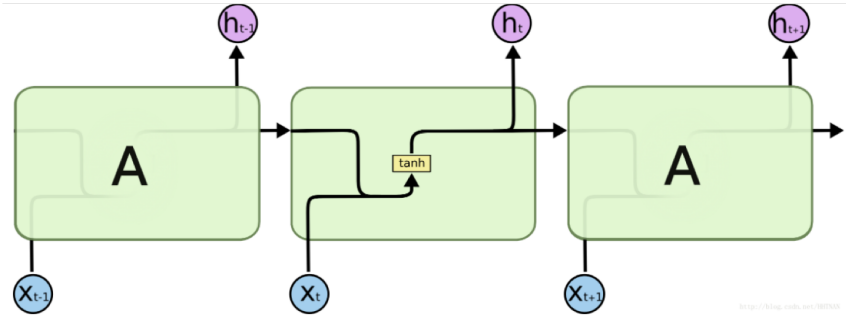

Figure 2.1: Standard unit RNN, figure from:

However, a problem occurs when the sequence is very long. A large sequence length is similar to a very deep NN. Due to the large number of steps in backpropagation the gradient vanishes and results in no update of the weights. In other words, learning halts.

Long Short-Term Memory Unit

The LSTM unit was developed in ... to solve the vanishing gradient problem of the RNNs with standard units. The LSTM unit has three inputs: again the new part of the sequence x_t and hidden state of the previous part of the sequence h_{t-1}. In addition, there is a second state vector which is the cell state or LSTMs memory c_{t-1}. The cell state passes through the LSTM unit with less computation, making it easier to pass through unchanged. Similar to the idea of adding residual connections in deep NNs, this helps preserve the gradient. In the LSTM unit three gates are computed from the concatenation $x = x_t^\frown h_{t-1}$, each having their own weight matrix:

- Forget gate $f = \sigma(W_f x + b_f)$
- Update gate $u = \sigma(W_u x + b_u)$
- Output gate $o = \sigma(W_o x + b_o)$

Here, σ is the sigmoid function which returns values between 0 and 1, killing input close to 0 and letting input close to 1 pass nearly unchanged. The new cell state is computed by multiplying the previous cell state by what was learned to forget (the forget gate) and adding this to what was learned to be remembered (update gate times $tanh(W_c * x + b_c)$): $c_t = f * c_{t-1} + u * tanh(W_c * x + b_c)$. The new hidden state is computed by the cell state scaled between [-1,1] by tanh function multiplied by what was learned to be exposed to the hidden state (output gate): $h_t = o * tanh(c_t)$. This gives the ability to control what to forget from the cell state, what to store from the input in the cell state and what part of the cell state to expose to the hidden state at a given point in time.

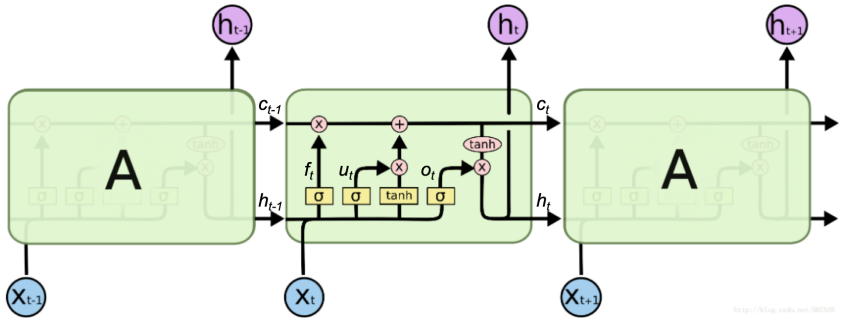

Figure 2.2: LSTM unit RNN, figure from:

RNNs with LSTM units have been used in combination with classical reduced order methods for temporal evolution in simulations of physical phenomena. For example, Hu et al. [7] used POD and SVD together with an LSTM network for predicting flood induced conditions. They reported a maintained accuracy in comparison to the full model while CPU cost was reduced three orders of magnitude. Also RNNs with LSTM units have been used in combination with machine learning techniques for order reduction. Simpson et al. [10] trained an autoencoder for order reduction and LSTM network to predict the response of dynamical systems in the latent space based on forcing time histories. However, to the author's knowledge this approach has not been tested on fluid dynamics simulations.

Gated Recurrent Unit

GRUs are an evolution of LSTM units and introduced quite recently in 2014 [REF: CHO 2014]. By dropping the output gate and only using two gates:

- Reset gate $r = \sigma(W_r x + b_r)$
- Update gate $z = \sigma(W_z x + b_z)$

The reset gate is similar to the forget gate in the LSTM unit and is to decide what to forget. The update gate is to decide what should be passed to the output. First a proposed new hidden state is computed by multiplying the old hidden state with the reset gate, concatenating this with the new part of the sequence and scaling between [-1,1] with the tanh function: $\hat{h}_t = tanh(W\widehat{x_r}(r * h_{t-1})$. Thereafter, the actual new hidden state is computed by taking a linear sum between the previous hidden state and the proposed new hidden state, in which the reset gate determines how much of the hidden state is updated: $h_t = (1 - z_t)h_{t-1} + z_t\hat{h}_t$.

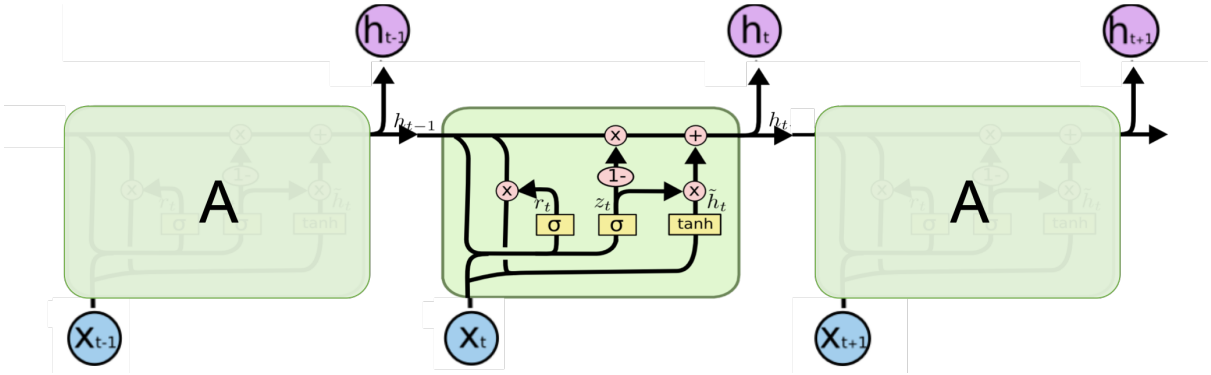

Figure 2.3: GRU RNN. figure from

With only two gates, there is no mechanism to control which parts of the state is exposed to the output, but the GRU has a less complex structure. This makes the units computationally more efficient while it was proven to have comparable performance to LSTM units [REF: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling].

To the author's knowledge, an RNN with GRUs for time evolution has not been used in combination with classical or machine learning reduced order techniques. Hence, this is a novel approach which will be explored in this research. *no papers found, novel approach?*

Encoder-decoder, attention mechanism

no papers found, novel approach?

2.4 Other ML in CFD

improve accuracy of coarse grid CFD simulation [REF: https://www.pnas.org/content/pnas/118/21/e2101784118.full.pdf]

2.5 Blood Flow modelling

2.5.1 Mathematical representation

To create a realistic synthetic data set, the flow of blood was modeled by Navier-Stokes equations as proposed in [11], adapted from 3D to 2D.

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial}{\partial t}(\mathbf{u}) + (\nabla \mathbf{u})\mathbf{u} + \nabla \frac{p}{\rho} - \nabla \cdot (2\nu D(\mathbf{u})) &= 0 \end{aligned} \tag{2.1}$$

In which $\mathbf{u}$ is the velocity, p is the mean normal stress ($p = -\frac{1}{2}(tr\mathbf{T})$ with $\mathbf{T} = -p\mathbf{I} + 2\mu\mathbf{D}$ the fluid stress), ν is the kinematic viscosity ($\nu = \frac{\mu}{\rho}$, with μ the dynamic viscosity) and $\mathbf{D}$ the strain-rate tensor ($\mathbf{D} = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)$)).

3 | Methodology

3.1 Data Generation

3.1.1 Numerical Method

To solve the mathematical problem stated in the previous section. OpenFoam software was used. In particular the IcoFoam solver, for incompressible, Newtonian fluids was implemented. The IcoFoam solver solves the following equation:

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial}{\partial t}(\mathbf{u}) + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) - \nabla \cdot (\nu \nabla \mathbf{u}) &= -\nabla p \end{aligned} \tag{3.1}$$

In which $\mathbf{u}$ is the velocity in $\frac{m}{s}$ and p is the kinematic pressure in $\frac{m^2}{s^2}$. Taking $\nu = ...$ and $p = ...$ makes equations 2.1 and 3.1 NOT?! equivalent as shown below.

$$\nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = \nabla \cdot \begin{bmatrix} u_x \\ u_y \end{bmatrix} [u_x \ u_y] = \nabla \cdot \begin{bmatrix} u_x^2 & u_x u_y \\ u_x u_y & u_y^2 \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x}(u_x^2) + \frac{\partial}{\partial y}(u_x u_y) \\ \frac{\partial}{\partial y}(u_y^2) + \frac{\partial}{\partial x}(u_x u_y) \end{bmatrix} = \begin{bmatrix} 2u_x \frac{\partial u_x}{\partial x} + u_x \frac{\partial u_y}{\partial y} + u_y \frac{\partial u_x}{\partial y} \\ 2u_y \frac{\partial u_y}{\partial y} + u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial x} \end{bmatrix}$$

$$\nabla \cdot (\nu \nabla \mathbf{u}) = \nabla \cdot (\nu \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix}) = \nu \begin{bmatrix} \frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_y}{\partial x \partial y} \\ \frac{\partial^2 u_y}{\partial y^2} + \frac{\partial^2 u_x}{\partial x \partial y} \end{bmatrix}$$

$$(\nabla \mathbf{u})\mathbf{u} = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} \\ u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} \end{bmatrix}$$

$$\nabla \cdot (2\nu \mathbf{D}(\mathbf{u})) = \nabla \cdot (2\nu * \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T))) = \nabla \cdot (\nu \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix} + \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_y}{\partial x} \\ \frac{\partial u_x}{\partial y} & \frac{\partial u_y}{\partial y} \end{bmatrix}) = \nabla \cdot (\nu \begin{bmatrix} 2\frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \\ \frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} & \frac{\partial u_y}{\partial y} \end{bmatrix})$$
$$= \nu \begin{bmatrix} 2\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_y}{\partial x \partial y} + \frac{\partial^2 u_x}{\partial y^2} \\ 2\frac{\partial^2 u_y}{\partial y^2} + \frac{\partial^2 u_x}{\partial x \partial y} + \frac{\partial^2 u_y}{\partial x^2} \end{bmatrix}$$

3.1.2 Geometries

Four different geometries were created with the GMSH python library by the following steps:

1. define 2D mesh by creating points, lines and a surface in the xy-plane
2. extrude the surface in the z-direction by one cell (this is necessary as OpenFoam only takes 3D meshes)
3. add all surfaces to appropriate Physical Groups and name these groups
4. save the mesh as .msh2 file (this is compatible with gmshToFoam function)
5. in openfoam, run gmshToFoam on mesh file
6. set front and back patches to empty type in constant/boundary file, set inflow and outlflow boundaries to patch type and set other boundaries to wall type
7. define all input parameters in the 0 folder
8. run icoFoam as solver
9. convert output to VTK files for visualization with Paraview by the "foamToVTK -allPatches" command

All code to create the meshes can be found in [REF APPENDIX MESH CODE].

3.2 Network Architectures

Taking into account the possibilities discussed in chapter 2, two network architectures are proposed: an auto-encoder for order reduction with a NN for timestepping and an auto-encoder for order reduction with a RNN using GRUs for timestepping.

3.2.1 Auto-encoder

3.2.2 Neural Network

3.2.3 Recurrent Neural Network with GRUs

3.3 Tuning, Training and Testing protocols

4 | Results

5 | Discussion

6 | Conclusions and Recommendations

Bibliography

[1] Amirhossein Arzani and Scott T.M. Dawson. *Data-driven cardiovascular flow modelling: Examples and opportunities*. 2021. DOI: 10.1098/rsif.2020.0802.

[2] Wenqian Chen et al. "Physics-informed machine learning for reduced-order modeling of nonlinear problems". In: *Journal of Computational Physics* 446 (2021). ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110666.

[3] M. Cheng et al. "An advanced hybrid deep adversarial autoencoder for parameterized nonlinear fluid flow modelling". In: *Computer Methods in Applied Mechanics and Engineering* 372 (Dec. 2020). ISSN: 00457825. DOI: 10.1016/j.cma.2020.113375.

[4] My Ha Dao. *Projection-Based Reduced Order Model for Simulations of Nonlinear Flows with Multiple Moving Objects*.

[5] Matthias Eichinger et al. *Technical Report Series Center for Data and Simulation Science Surrogate Convolutional Neural Network Models for Steady Com-putational Fluid Dynamics Simulations SURROGATE CONVOLUTIONAL NEURAL NETWORK MODELS 1 FOR STEADY COMPUTATIONAL FLUID DYNAMICS 2 SIMULATIONS*. 2016.

[6] Amir Barati Farimani, Joseph Gomes, and Vijay S Pande. *Deep Learning the Physics of Transport Phenomena*. 2017.

[7] R. Hu et al. "Rapid spatio-temporal flood prediction and uncertainty quantification using a deep learning method". In: *Journal of Hydrology* 575 (Aug. 2019), pp. 911–920. ISSN: 00221694. DOI: 10.1016/j.jhydrol.2019.05.087.

[8] Byungsoo Kim et al. "Deep Fluids: A Generative Network for Parameterized Fluid Simulations". In: *Computer Graphics Forum* 38 (2 May 2019), pp. 59–70. ISSN: 14678659. DOI: 10.1111/cgf.13619.

[9] Manuel Lopez-Martin, Soledad Le Clainche, and Belen Carro. "Model-free short-term fluid dynamics estimator with a deep 3D-convolutional neural network". In: *Expert Systems with Applications* 177 (2021). ISSN: 09574174. DOI: 10.1016/j.eswa.2021.114924.

[10] Thomas Simpson, Nikolaos Dervilis, and Eleni Chatzi. "Machine Learning Approach to Model Order Reduction of Nonlinear Systems via Autoencoder and LSTM Networks". In: *Journal of Engineering Mechanics* 147 (10 2021). ISSN: 0733-9399. DOI: 10.1061/(asce)em.1943-7889.0001971.

[11] H. Švihlová et al. "Determination of pressure data from velocity data with a view toward its application in cardiovascular mechanics. Part 1. Theoretical considerations". In: *International Journal of Engineering Science* 105 (2016). ISSN: 00207225. DOI: 10.1016/j.ijengsci.2015.11.002.

[12] H. Švihlová et al. "Determination of pressure data from velocity data with a view towards its application in cardiovascular mechanics. Part 2. A study of aortic valve stenosis". In: *International Journal of Engineering Science* 114 (2017). ISSN: 00207225. DOI: 10.1016/j.ijengsci.2017.01.002.