Report: Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows
Master-Seminar - Deep Learning in Physics

JULIAN HOHENADEL, Technical University of Munich

This report aims to critically assess a deep learning approach to infer Reynolds-Averaged Navier-Stokes solutions. Following the thoughts of [Thuerey et al. 2020] a fully convolutional U-Net architecture is used to classify the performance on deriving pressure and velocity fields and their distributions. With an evaluation of different hyper parameter regarding network and training data size, the best models yield a mean error relative to pressure and velocity channels of less than 3% on a test set containing 90 samples. [Thuerey et al. 2020] states that due to this generic approach other PDE boundary value problems on Cartesian grids can also be tackled with this setup. In this report different adaptations of the network architecture are compared to each other with different data normalization in form of vector norms. Lastly the architecture is used to predict wave propagation on shallow water governed by the Saint-Venant equations.

Additional Key Words and Phrases: RANS, Reynolds-Averaged Navier-Stokes, CNN, U-Net, FCNN, PDE, CFD, Cartesian grid, pressure, velocity, airfoil, flow prediction, Eulerian field functions

1 INTRODUCTION

Since the last few years machine learning as well as deep learning in the form of neural networks enjoy great popularity. Their broad spectrum of application fields paired with the competitiveness of the researchers and fast available hardware (GPUs) allow the scientific community to squeeze out every bit of performance of their neural networks.

This hype also affects the computational fluid dynamics (CFD) field as [Viquerat and Hachem 2019] shows with an exponential growth in published papers over the last 20 years. Solving flow prediction problems traditionally is coupled with a deep understanding of physics and modeling these complex physical constrains. A neural network with a high enough capacity is capable to capture these high-dimensional, nonlinear constrains. Neural networks are universal approximators [Csáji 2001] i.e. there is always a trade-off regarding accuracy to the ground truth function (actual dynamics) and the number of parameter in the network.

[Thuerey et al. 2020] explains the first goal is to alleviate concerns regarding achievable accuracy and analyzability of deep learning approaches by showing a relation between accuracy and changes in the hyper-parameter of a network. As a second goal the paper provides a testing environment hosted on GitHub. The code is clean and easy to read. Training data is also available which is a jump start for own experiments with computational fluid dynamics. Different training dataset sizes and a flexible network capacity prevents long training time which can arise from a potential hardware bottleneck. Using Google Colab different kinds of ReLu activation functions and their impact on the networks' accuracy are compared based on their mean errors regarding pressure, velocity and combined. Data preprocessing is one of the most important aspects to make the training fast and converge to good minima. For that reason different vector and pseudo norms are used to profit from dimensionless training data with different scaling. To prove that this generic setup as [Thuerey et al. 2020] claims is applicable to other partial differential equations (PDE) boundary value problems the network architecture is used to train on inferring the Saint-Venant equations, which [Fotiadis et al. 2020] uses for the modeling of surface wave propagation.

This report is a part of the lecture, Master-Seminar - Deep Learning in Computer Graphics, Informatics 15, Technical University of Munich.

2 BACKGROUND

Reynolds-averaged Navier-Stokes (RANS) equations are used for the modeling of turbulent incompressible flows [Alfonsi 2009].
The Navier-Stokes equation is a nonlinear partial differential equation which can be seen as Newtons's 2nd law of motion [Bistafa 2018]:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla p + \nu \nabla^2 u \tag{1}$$

Where u is the velocity vector, g is the acceleration vector due to a body force, p is pressure, ν is the kinematic viscosity and ρ is the density. [Bistafa 2018] rewrites the left side of the equation as the particles acceleration and the right side as the external forces that affect the particle per unit of mass which yields Newtons's 2nd law of motion. The idea behind RANS is Reynolds decomposition which splits dependent model variables into a mean and fluctuating part [Alfonsi 2009]. The Reynolds number Re is a dimensionless constant which is needed for the calculation of turbulence models. Depending on the magnitude of the Reynolds number the flow ranges from laminar to turbulent [Lissaman 1983]. The angle of attack on an airfoil as well as the Reynolds number are important parameter for the creation of the physical model because they affect lift and drag coefficients [Lissaman 1983].

3 METHOD

In the following the methodology used by [Thuerey et al. 2020] to infer the RANS solutions is summarized and extended by experiments evaluated for this report. For consistency reasons the structure is kept equal to the original paper.

3.1 Data Generation

The UIUC database [Selig 1996] holds a variety of airfoil shapes, shown in Figure 1, which are used to generate training data. The provided data is only a 2D coordinate representation of the airfoil shape. To acquire the ground truth velocity and pressure maps the open source package OpenFOAM is used which contains solvers for CFD problems. The range of Reynolds numbers $[0.5, 5] \cdot 10^6$ used for generating the ground truth indicates turbulent airflow.

Table 1. Norm comparision wrt. error, L2 default (in %).

Norm	pressure	velocity	combined
L1	**14.19**	**2.251**	**2.646**
L2	14.76	2.291	2.780
L inf	15.09	2.348	2.865
L 0.25	31.03	3.106	3.272
L 0.5	15.64	2.448	2.705
L 0.75	31.03	3.106	3.272

L1 normalization achieves the best error rates (lower is better).
Trained on [Thuerey et al. 2020] U-Net implementation with $1.9 \cdot 10^6$ weights.
Seeds used for training: 0, 2^{10}, 2^{20}, 2^{30}

Also, an angle of attack of ±22.5 degrees is used. RANS turbulence models distinguish between different classes [Alfonsi 2009]. In this case the one-equation model is used. The solutions are cropped to a resolution of $128x128$ pixels around the airfoil to speed up training while keeping the most relevant information. A test set is generated separately with unseen airfoil shapes.

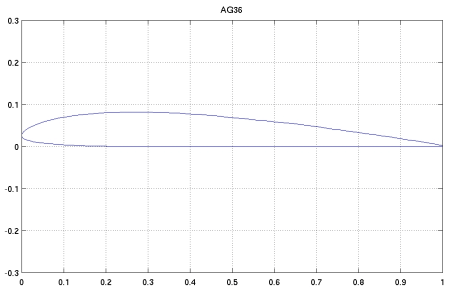

Fig. 1. A sample airfoil shape in 2D.

3.2 Pre-processing

One training data sample consists of 3 $128x128$ grids wrapped into a tensor. The channels are a bit mask representing the position and shape of the airfoil, x and y velocity. Inside the airfoil shape all elements contain 0 in the velocity fields. The velocity maps are initialized with differently scaled freestream velocity vectors which have the corresponding Reynolds number for the sample encoded in them with respect to the magnitude of the elements.
For the inferred solution of the neural network the shape stays the same as the input. [Thuerey et al. 2020] use the solution obtained from OpenFOAM as ground truth with channels split into pressure, x and y velocity. The first normalization step is to divide the ground truth data with the magnitude of the freestream velocity (default: L2 vector norm). The pressure channel is divided with the vector norm squared to make the data dimensionless. This originates in the physical unit of pressure: In an incompressible model the density is constant so to make the pressure dimensionless it needs to be divided with the squared velocity.
Additionally, to norm based normalization the pressure channel is shifted into the origin by subtraction the pressure mean for each sample respectively. By doing this the network does not have to waste resources on compensating for an off centered pressure map. Lastly [Thuerey et al. 2020] clamps each channel into an $[-1, 1]$ interval to ensure a maximum of numerical precision.

Table 1 shows how different vector norms which are only used for normalization of the target data in the pre-processing step change the averaged error of neural networks which are all trained under the same conditions and capacity. Figure 7 in the Appendix visualizes the relations of the p-norms on the unit circle: $L1 \geq L2 \geq L_{inf}$
Interestingly the same inequality relation holds for every error channel shown in Table 1 as well. The other norms are quasi norms because they violate the required convexity but follow the same definition for a vector x:

$$||x||_p := \left(\sum_{i=0}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{2}$$

Surprisingly L0.25 and L0.75 normalization hold exactly the same results, although the absolute norm values on the same sample differ vastly. They also fail to infer target pressure values by around $2x$ when compared with the other norms.
To explain the error margin between L1 and L2 normalization this report takes a closer look at the trained weights in the network. The filter used in the convolutional layers should still capture the same patterns to identify regions of interest, independent of the used norm. Figure 2 shows the difference in weights created in the first layer.

Fig. 2. Convolution weight (first layer) difference: L1 and L2 normalization Filter kernel size: 4x4.

The diversity in the first layer is shown by the different color shades. Descending deeper into the U-Net convolutional filter get far more heterogen (see Figure 8 in the Appendix). After the bottleneck this effect progressively fades out. The weights in the last layer are besides a few pixels e.g. single weights mostly identical. This can be seen in Figure 3.

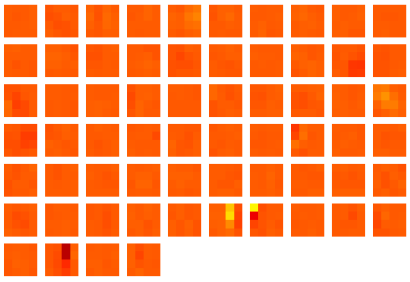

Fig. 3. Convolution weight (last layer) difference: L1 and L2 normalization Filter kernel size: 4x4.

Batch normalization is used to reduce internal covariance shift [Ioffe and Szegedy 2015]. γ and β are two learnable parameters to influence variance and mean of the input data to improve training. Because [Thuerey et al. 2020] works with shuffled datasets it is not possible to directly compare the learned weights. The γ and β per layer mean differences between L1 and L2 normalization are in a range of $[10^{-3}, 10^{-2}]$.
It is not apparent why and how the weight changes affect the accuracy. Most likely the error improvement is far from being significant enough for a network with in this case close to $2 \cdot 10^6$ parameter. With bigger filter kernel sizes (e.g. VGG nets) in the convolutional layers the detected structures would be more comparable, while a maximal size of 4x4 pixels is limiting a visual comparison.

3.3 Neural Network Architecture

[Thuerey et al. 2020] use a modified version of the U-Net architecture proposed by [Ronneberger et al. 2015] (Figure 6) which was first successfully used for biomedical image segmentation. The U-Net architecture consists of two parts. The left side of the "U" functions as an encoder. It captures the context of the image. While width and height of the image get smaller the depth of the channels increases. The right side acts as a decoder. It upsamples the feature maps to recover spatial locations. The behavior of the image size and depth is contrary to the encoder part. Every block is implemented similarly: activation function (leaky ReLu in encoder, ReLu in decoder), convolution layer, batch normalization and finally dropout for regularization [Thuerey et al. 2020]. Without the help of skip connections information captured in the initial layers of the U-Net which is needed for reconstruction in the decoding part may get lost. This can be avoided by concatenating the horizontal aligned blocks.

Table 2. Activation function comparision wrt. error after 80k iterations (in %).

Norm	pressure	velocity	combined
Std	**14.76**	**2.291**	**2.780**
PReLu	18.17	2.453	2.871
ReLu	34.13	3.369	3.547

The activation functions proposed by [Thuerey et al. 2020] achieve the best error rates (lower is better).
Seeds used for training: 0, 2^{10}, 2^{20}, 2^{30}

This report compares the use of different activation functions as they not only act as a nonlinearity. They should also control the flow of the gradients through the network which is essential for proper learning.
Rectified Linear Units (ReLu) and variants of ReLu are nowadays the standard choice as an activation function. ReLu does not saturate and yield large and consistent gradients. Both are important for fast convergence. The problem with ReLu is that it can "die" (e.g. the output is 0) which lowers the capacity of the network by deactivating neurons as their output is multiplied with 0.
Leaky ReLu does have the same benefits as regular ReLu but does not suffer from the "dying out" problem. For this reason the same network architecture is trained with the activation setup [Thuerey et al. 2020] used as well as all activations set to vanilla ReLu and PReLu respectively which can be seen in Table 2. PReLu makes use of a learnable parameter which sets the negative slope through backpropagation while leaky ReLu has a fixed hard coded parameter for that reason (0.2 is used by [Thuerey et al. 2020]). Table 2 shows the results with the activations used by [Thuerey et al. 2020] showing the best results with regard to accuracy.
Regular ReLu for both encoder and decoder is not a beneficial choice as information which is lost to the "dead" ReLu in the encoder can only be partially (with skip connections) or not at all be recovered and forwarded to the decoder. This shows in higher error rates especially for the pressure map.
PReLu is close to the performance of the standard setup but as the values suggest still needs more training time. PReLu also adds more weights to the network thus increasing its capacity.

Table 3. Activation function comparision wrt. error after 160k iterations (in %).

Norm	pressure	velocity	combined
Std	14.76	2.296	2.787
PReLu	**14.69**	**2.216**	**2.676**

The PReLu activation function achieves the best error rates (lower is better).
Seed used for training: 0

Table 3 shows that the standard setup could not achieve a better result while PReLu is able to reduce its error by 0.2%. These results were obtained with a single seed and therefore may not be that conclusive but rather indicate a trend. PReLu activations on the encoder part have a mean parameter value close to the 0.2 proposed by [Thuerey et al. 2020]. PReLu is also able to train one parameter for each input channel adding even more flexibility with the cost of additional weights. Despite [Thuerey et al. 2020] using a learning rate decay (reduction to 10% of the initial value after 50% of the training) which PyTorch states is not well suited to combine with PReLu the activation function does not seem to be affected by it.

3.4 Supervised Training

The training setup used by [Thuerey et al. 2020] consists of an L_1 loss, learning rate decay and the use of the Adam optimizer [Kingma and Ba 2014] with default PyTorch settings except β_1 (set to 0.5). Adam makes use of first and second momentum β_1 and β_2 respectively. The main idea is to accelerate into directions where the gradient accumulates (β_1) and also dampen the gradients' variance (β_2). Lowering β_1 slows down acceleration but can be useful if you overshoot good local minima. [Thuerey et al. 2020] also found that generating more training data is preferred over using adversarial training.
[Thuerey et al. 2020] evaluated their accuracy with multiple U-Nets with the same architecture but different capacity ranging from $122k$ up to $30.9m$ over training data ranging from 100 up to $12.8k$ samples. While the $30.9m$ model suffers from its huge capacity in form of overfitting small data sets it achieves a combined relative error of 2.6% using the $12.8k$ sample training dataset. The $1.9m$ and $7.7m$

model also show close results wrt. the highest capacity model.
[Thuerey et al. 2020] explored generalization with an additional dataset in which the airfoil shapes were sheared by ±15%. The datasets are used pure or mixed with equal contribution. The $30.9m$ model is able to use its capacity to bring the error down to 2.32% when training with a $51k$ mixed dataset. Compared with the regular dataset containing the same amount of samples the added complexity (i.e. regularization) benefits the training. [Thuerey et al. 2020] achieves a speed up of ca. $1000x$ when using an NVidia GTX 1080 GPU for inference of the neural networks compared to OpenFOAM.

4 TRANSFER

To show another use case for the models created by [Thuerey et al. 2020] they will be used to infer the propagation of surface waves governed by the Saint-Venant equations as they are related with the Navier-Stokes equations which is examined in [Fotiadis et al. 2020]. For this experiment the U-Net architecture will be the same except the input channels of the first layer which now contain the last n time steps. The same is applied to the output of the last layer to predict the next m time steps. As a testing environment the GitHub repository provided by [Fotiadis et al. 2020] is used. To be consistent with [Fotiadis et al. 2020] U-Net architecture and to make the results comparable $n = 5$ and $m = 20$ without dropout is used in this setup. Due to Google Colab hardware and runtime restrictions both models were only trained for 50 epochs and both neural nets have a capacity between $[7.7 \cdot 10^6, 7.8 \cdot 10^6]$.
The main difference to [Thuerey et al. 2020] architecture is that [Fotiadis et al. 2020] U-Net is more shallow with only 3 convolution blocks in the encoder and decoder respectively as well as the bottleneck resolution being 16x16. There is also no batch normalization layer after every convolution layer. Downsampling is done with max pooling instead of strided convolutions and all activation functions use ReLu.
Comparing Figure 5 to Figure 4 both RMSE are almost similar up to 35 time-steps ahead and after 80 time-steps the gap is still relatively small. [Fotiadis et al. 2020] achieve with their U-Net a speed-up of 241x compared with numerical simulations. This again shows the enormous potential of CNN's when compared with classical solvers for CFD and PDE.

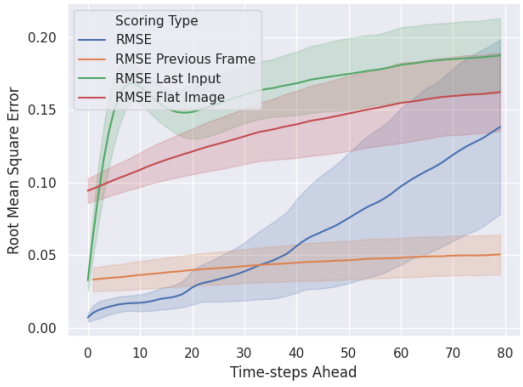

Fig. 4. RMSE error on test set with [Thuerey et al. 2020] architecture.

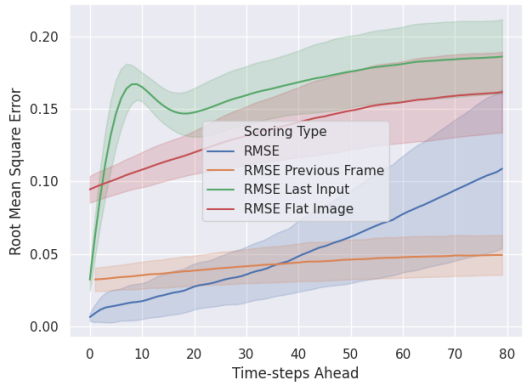

Fig. 5. RMSE error on test set with [Fotiadis et al. 2020] architecture.

5 DISCUSSION

[Fotiadis et al. 2020] show that U-Nets can outperform LSTM's in accuracy as well as in speed up with a fraction of capacity (in time-series problems). Convolutions paired with an encoder decoder structure seem to catch regions of interest fast and reliable. Looking at the validation and mean error plots shown in [Thuerey et al. 2020] the accuracy does not suffer too much from models with a lot less capacity. Their natural regularization effect should make improvement possible even with a small mixed dataset which would again increase inference speed. It is not clear why problem specific solver should infer airfoil flows if the accuracy of neural networks can be improved even more. [Thuerey et al. 2020] show that the data pre-processing (i.e. reducing the space of solutions) is almost more important than the actual fine-tuning of the models. This research is important because it questions the use of solvers and shows the rapid progress wrt. accuracy of convolutional neural networks in CFD problems.

6 CONCLUSION

[Thuerey et al. 2020] present an out-of-the-box well training derivative of the U-Net architecture proposed by [Ronneberger et al. 2015]. Despite being firstly designed for image segmentation it is applicable in a wide variety of research fields. This report compared different vector and pseudo norms for the pre-processing step and their influence on the training process. PReLu as an activation function is compared to classic ReLu and leaky ReLu. While hard coded activation parameter can provide good training results after hyper parameter tuning it can be worth testing activations that train themselves through backpropagation. For the cost of some additional weights given enough iterations it may be worth considering. The U-Net being applicable to other PDE problems leaves a lot of possible future work especially because convolutional neural networks are able to outperform LSTM's in time-series predictions.

REFERENCES

Giancarlo Alfonsi. 2009. Reynolds-Averaged Navier-Stokes Equations for Turbulence Modeling. *Applied Mechanics Reviews - APPL MECH REV* 62 (07 2009). DOI:https://doi.org/10.1115/1.3124648

Sylvio R. Bistafa. 2018. On the development of the Navier-Stokes equation by Navier. *Revista Brasileira de Ensino de FÃsica* 40 (00 2018). http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1806-11172018000200703&nrm=iso

Balázs Csanád Csáji. 2001. Approximation with Artificial Neural Networks.

Stathi Fotiadis, Eduardo Pignatelli, Mario Lino Valencia, Chris Cantwell, Amos Storkey, and Anil A. Bharath. 2020. Comparing recurrent and convolutional neural networks for predicting wave propagation. In *ICLR Workshop on Deep Neural Models and Differential Equations*. http://arxiv.org/abs/2002.08981

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. (2015). arXiv:cs.LG/1502.03167

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. (2014). arXiv:cs.LG/1412.6980

PBS Lissaman. 1983. Low-Reynolds-number airfoils. *Annual review of fluid mechanics* 15, 1 (1983), 223–239.

Quartl. 2011. Vector-p-Norms qtl1.svg. (2011). https://upload.wikimedia.org/wikipedia/commons/d/d4/Vector-p-Norms_qtl1.svg https://creativecommons.org/licenses/by-sa/3.0/legalcode.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. (2015). arXiv:cs.CV/1505.04597

Michael S. Selig. 1996. *UIUC airfoil data site*. Department of Aeronautical and Astronautical Engineering University of Illinois at Urbana-Champaign.

Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. 2020. Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows. *AIAA Journal* 58, 1 (2020), 25–36. DOI:https://doi.org/10.2514/1.J058291 arXiv:https://doi.org/10.2514/1.J058291

Jonathan Viquerat and Elie Hachem. 2019. A supervised neural network for drag prediction of arbitrary 2D shapes in low Reynolds number flows. (2019). arXiv:physics.comp-ph/1907.05090

7 APPENDIX

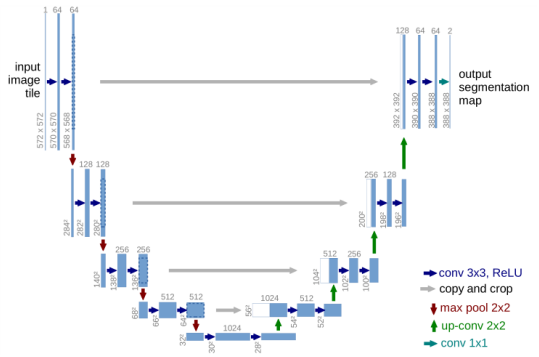

Fig. 6. U-Net architecture, taken from [Ronneberger et al. 2015].

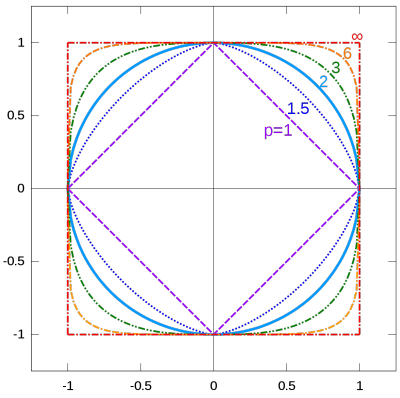

Fig. 7. Vector norms visualized on unit circle, taken from [Quartl 2011].

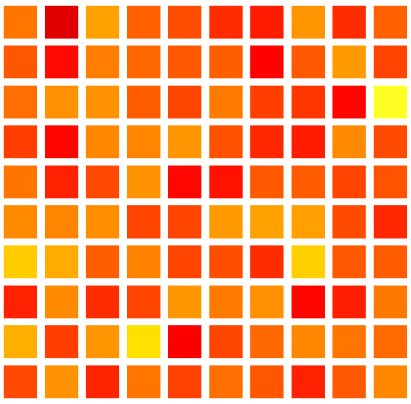

Fig. 8. Convolution weight (bottleneck) difference between L1 and L2 normalization, first 100 weights. Filter kernel size: 1x1.