Machine learning to predict Reynolds stresses in transitional boundary layer fluid flow

Shaun Harris

Motivation and objectives Computation fluid dynamics (CFD) is often used to model the fluid flow along a surface. In calculating the Reynolds averaged Navier-Stokes (RANS) equations we find that modeling the Reynolds stresses is required and calculating these terms is a matter of research and various models have been proposed and adopted by the modeling community. However, calculating these terms in laminar to turbulent transitional boundary layer has proven difficult. Slotnick et al. (2014) states that the single most critical area in CFD simulation capability is the ability to predict viscous turbulent flows with possible boundary layer transition and flow separation present. Currently, in order to accurately solve these flows, expensive direct numerical simulations must be conducted to resolve the flow (e.g. Zaki, 2013). Models have been developed to reduce this cost (e.g. Langtry and Menter, 2009; Durbin, 2018), but, for various reasons, these models have resulted in poor performance in application and have not been widely adopted by the CFD community.

Ling et al. (2016) applied a deep neural network to calculate the Reynolds stresses. The input to their model, however, required the full non-averaged velocity and pressure terms. These input features allowed for Galilean and rotational invariant coordinate system, which is an advantageous property for these systems. A major conclusion is that significant improvements over classical RANS closure solvers is possible using the deep neural network described there.

The input to the model used in this work will retain the Galilean invariant property, but will not retain the rotational invariant property. This is done such that the model will take advantage of already readily available values in a typical CFD solver. These inputs are also very similar to the inputs used by Wu et al. (2019) where these features were used to identify the interface between turbulent and laminar portions of the boundary layer. The dataset used there was the same dataset used here in this research found in the work by Zaki (2013) and published online in the John Hopkins Turbulence Databases.

Dataset and features Zaki (2013) dataset contains velocity and pressure at each Cartesian grid point location in a flow over a flat plate. This data was saved using hundreds of terabytes of data and stored online for many researchers to access and explore. Zaki (2013) also released averaged flow-fields over the spanwise dimension and in time. The variables are

$$\bar{u}, \overline{uu}, \bar{v}, \overline{vv}, \bar{w}, \overline{ww}, \overline{uv}, \overline{uw}, \overline{vw}, \bar{p}, \tag{1}$$

where $\bar{u}$ indicates the streamwise averaged velocity, $\bar{v}$ is the wall-normal averaged velocity, $\bar{w}$ is the spanwise averaged velocity, $\bar{p}$ is the averaged pressure. For this case, $\overline{w}$ is nearly zero, and is neglected as an input feature.

The input parameters were calculated by taking the gradients of the two dimensional averaged quantities at each spatial grid point, thus giving fourteen quantities for the input feature dimension, $d = 14$. These features were calculated using the given grid and a fourth-order finite difference operators. These Galilean invariant input features for a single spatial location are

$$x^{(i)} = \left[\bar{P}, \partial_x\bar{u}, \partial_x\bar{v}, \partial_x\bar{p}, \partial_y\bar{u}, \partial_y\bar{v}, \partial_y\bar{p}, \partial_{xx}\bar{u}, \partial_{xx}\bar{v}, \partial_{xx}\bar{p}, \partial_{yy}\bar{u}, \partial_{yy}\bar{v}, \partial_{yy}\bar{p}, \nu\right]^{\mathsf{T}}. \tag{2}$$

In addition to computing the input features, the output features also needed to be preprocessed by calculating

$$\overline{u'u'} = \overline{uu} - \bar{u}\bar{u}, \tag{3}$$

and similarly for the $\overline{v'v'}$, $\overline{w'w'}$, and $\overline{u'v'}$. Some of the input and output features can be seen in Figure 1. This work, however, will focus on modeling the streamwise Reynolds stress $\overline{u'u'}$ and the results and scores presented herein are regarding that output feature.

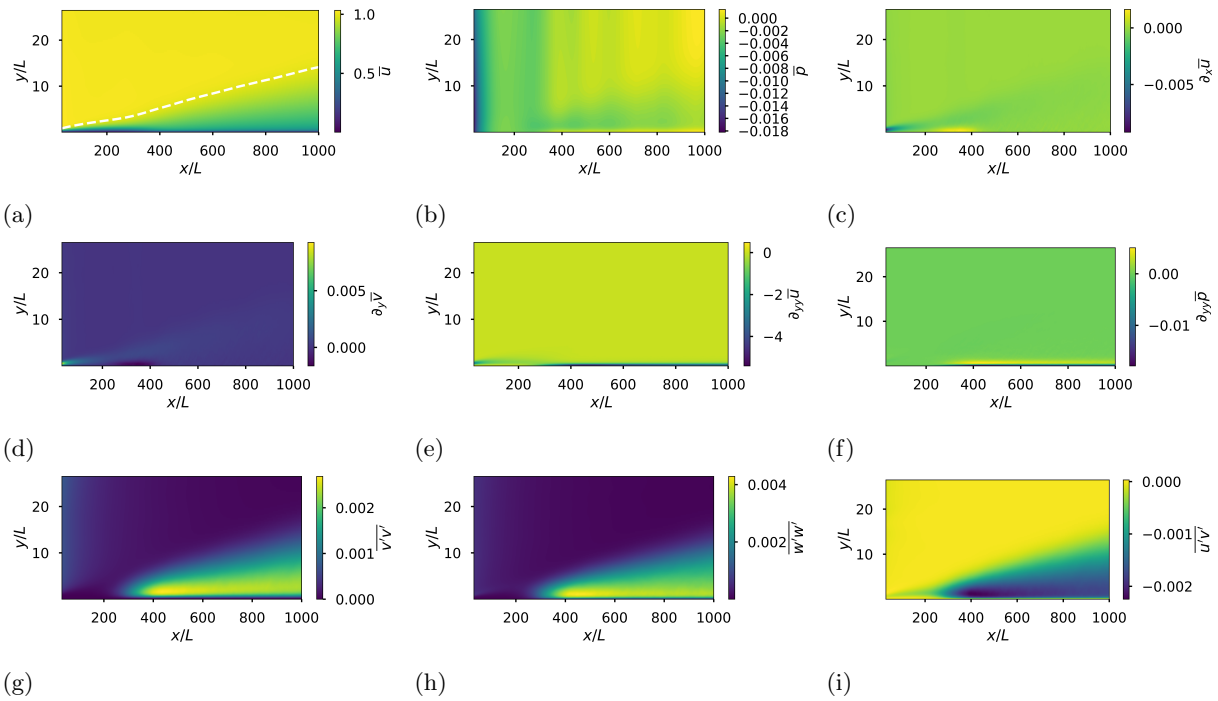

FIGURE 1. The averaged base flow over the laminar to turbulent transitional flat plate. (a)-(b) show the velocity and pressure over the domain, the white dotted line in (a) indicates the 99% boundary layer thickness. (c)-(f) show some of the preprocessed input features. (g)-(i) show some of the output features.

This dataset contained roughly $n = 739,200$ data grid points. This was split to 70/30 training and validation sets. The training set utilized some data augmentation for a couple of the models. One augmentation had to do with a polynomial feature mapping of degree two and three. The augmentation for polynomial degree two, for example, augmented the input features to contain the original dataset, and combinations of the input features. Thus, the polynomial feature map of degree two would act on a two dimensional input feature map

$$\phi(\left[x_1, x_2\right]^{\mathsf{T}}) = \left[x_1, x_2, x_1^2, x_1 x_2, x_2^2\right]^{\mathsf{T}}. \tag{4}$$

Similar results are given for polynomial feature map of degree three, but that it includes up to third order polynomials and interactions. This augmentation is useful to contain some non-linear interaction between the input features, which appears in the governing RANS equations. Additional data augmentation included adding some random noise to the input features so as reduce bias in the resulting model. This was achieved by increasing the datasize by 400% and on the extra 300% a random noise around 2% in magnitude was added to each of the original fourteen input features as shown here for one set of input features

$$x^{(i)} = x^{(i)} + (0.04\text{rand}(d) - 0.02)x^{(i)}. \tag{5}$$

Here, the function rand generats a random vector between zero and one of dimension d.

Methods Four methods are demonstrated in this work. Linear regression was tested first, it was found to give better than baseline performance, but not adequate for the desired performance. Data augmentation was then used to expand the input features using polynomial feature maps of degree two and three as well as adding the noisy input features. In addition to linear regression, a simple neural network was tested with two layers and one activation function. These methods are described in more detail below.

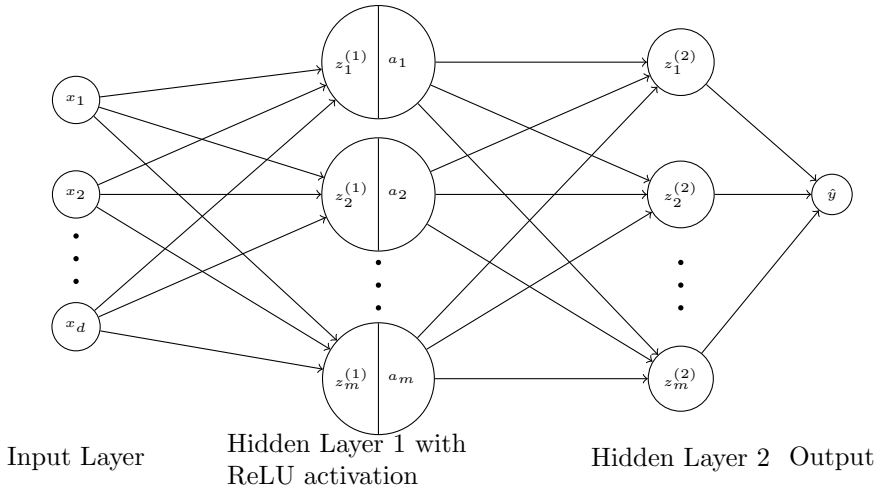

FIGURE 2. Illustration of multilayer perceptron neural network used in this work.

Linear regression Linear regression is a simple model that fits a line to the dataset. With multiple input features, this model becomes

$$h_\theta(x^{(i)}) = \hat{y}^{(i)} = \theta^T x^{(i)} = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + ... + \theta_d x_d^{(i)}, \tag{6}$$

where d is the number of input features. It is noted that the number of input features may change if we augment the data with the polynomial feature mapping. The coefficients θ are found by minimizing the cost function of the sum squared error, or minimizing

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2. \tag{7}$$

When polynomial feature maps were used, then the model became

$$h_\theta(x^{(i)}) = \hat{y}^{(i)} = \theta^T \phi(x^{(i)}) \tag{8}$$

where ϕ is the polynomial feature map of order two or three. Python package scikit-learn (Pedregosa et al., 2011) was used to calculate the linear regression and polynomial feature maps.

Neural network In addition to the linear regression, a neural network was implemented to see the performance. For this, a simple neural network was formed with minimal two layers each with and one hundred nodes with the first layer containing a rectified linear unit (ReLU) activation function. The ReLU activation function is defined by

$$a = g(z) = \max(0, z). \tag{9}$$

The neural network is shown in Figure 2 where $d = 14$, $m = 100$, and the output contained the output features, model for Reynolds the stresses, individually. Each output feature was trained on a separate neural network.

In order to optimize the weights associated with this neural network, the gradients of the cost function with respect to each of the parameters is needed. To calculate these gradients, automatic differentiation was used in PyTorch (Paszke et al., 2017). This allowed the gradients with respect to the objective function to be minimized and easily calculated in the PyTorch framework. The automatic differentiation details can be found in the work published by Paszke et al. (2017). The neural network model was trained using the mean squared error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2, \tag{10}$$

ID	Model name
LR	Linear regression
LR poly2	Linear regression with polynomial feature map of degree two
LR poly3	Linear regression with polynomial feature map of degree three
NN	Neural network

TABLE 1. Summarized description of the four models used in this work

	Training	Test
LR	0.728	0.729
LR poly2	0.951	0.942
LR poly3	0.993	0.994
NN	0.990	0.990

TABLE 2. R^2 score for each model tested for modeling the $\overline{u'u'}$ Reynolds stress.

and the Adam optimization strategy was used. The details for the Adam strategy can be found in the work by Kingma and Ba (2014). The four models described here are summarized in Table 1.

Results and discussion Here, we wish to find a model that all the variation in the model can be explained by the inputs, and that accurately describes the data. Thus, we will use an R^2 metric to display the goodness of fit for the models. So for any model a value of $R^2 = 0.5$ means that half of the observed variation in the model is explained by the model inputs. It can be seen by the equation

$$R^2 \equiv 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2} \tag{11}$$

that a value of 1 indicates a model that perfectly captures the Reynolds stresses using the given inputs. Thus, we seek a model that has a very high R^2 score.

The data used a 70/30 random split of the dataset into training and test data and the resulting R^2 score is reported in Table 2 for each model tested for the $\overline{u'u'}$ Reynolds stress term. We see that the LR poly3 model seems to perform the best, though for training and timing purposes, however, LR poly2 may be very useful as it gives good performance on training speed and expense of prediction. For qualitative purposes, the $\overline{u'u'}$ Reynolds stress can be observed in Figure 3 for each model compared to the dataset.

This is observed in Figure 3.

The exploration of which parameters were most important was also conducted. This was done using a similar procedure as outlined in Breiman (2001) using a method known as permutation importance or mean decrease accuracy. Here, each individual input feature is perturbed and run through the model, the resulting model prediction is compared to the true label in a statistical manner to produce a designed weight for each input feature. This was done for LR poly2 and LR poly3 models, and the resulting weight indicates how much of the model predictor depends on the certain input parameters. The resulting weight metric, computed using the Python module ELI5, is shown in Table 3.

Conclusion and future work We have found that we can accurately model the streamwise Reynolds stress term using the Galilean invariant averaged base flow properties at each grid point. The best model that was tested was the linear regression with a polynomial feature mapping of degree three. Future work would include modeling the other Reynolds stress terms, and possibly the gradients of the Reynolds stresses terms directly as it may be more desirable for a RANS closure model replacement. Application to other flow types would also be very beneficial. Additionally, decreasing the number of input features using the permutation importance metric and implementation of the resulting model into a simulation package would be a matter of future work. The scripts and data used in this work are available online at `https://github.com/srharris91/ML_model_for_Reynolds_stresses.git`.

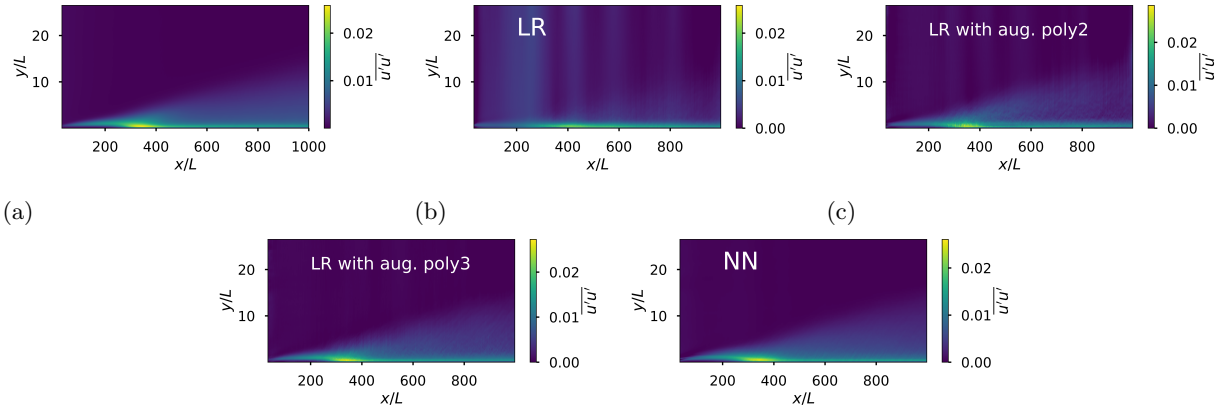

FIGURE 3. The model prediction of $\overline{u'u'}$ compared to the original dataset. (a) contains the original dataset while the models are shown by (b) LR, (c) LR poly2, (d), and (e) NN.

Weight	Feature
26.1910 ± 0.0098	$\partial_{yy}\overline{u}$
15.0344 ± 0.0669	$\partial_{yy}\overline{p}$
4.7703 ± 0.0285	$\partial_{x}\overline{u}$
3.5213 ± 0.0129	$\partial_{y}\overline{u}$
2.5225 ± 0.0083	$\partial_{y}\overline{p}$
2.4404 ± 0.0194	$\overline{p}$
2.2808 ± 0.0043	$\partial_{x}\overline{p}$
1.8278 ± 0.0133	$\partial_{y}\overline{v}$
1.3177 ± 0.0066	$\partial_{yy}\overline{v}$
0.2454 ± 0.0064	$\partial_{x}\overline{v}$
0.0634 ± 0.0003	$\partial_{xx}\overline{u}$
0.0626 ± 0.0005	$\partial_{xx}\overline{p}$
0.0017 ± 0.0000	$\partial_{xx}\overline{v}$
0.0000 ± 0.0000	ν

(a)

Weight	Feature
1294.4261 ± 25.3066	$\partial_{y}\overline{v}$
324.8356 ± 12.2379	$\partial_{x}\overline{u}$
109.1876 ± 2.1353	$\partial_{y}\overline{u}$
94.0471 ± 3.4430	$\partial_{yy}\overline{u}$
72.4699 ± 4.9553	$\partial_{x}\overline{v}$
47.3593 ± 0.6829	$\partial_{yy}\overline{p}$
21.5922 ± 0.1050	$\partial_{x}\overline{p}$
14.0522 ± 0.3132	$\partial_{y}\overline{p}$
8.5608 ± 0.1338	$\overline{p}$
2.1770 ± 0.0234	$\partial_{yy}\overline{v}$
0.3449 ± 0.0034	$\partial_{xx}\overline{p}$
0.0423 ± 0.0020	$\partial_{xx}\overline{v}$
0.0035 ± 0.0001	$\partial_{xx}\overline{u}$
0.0000 ± 0.0000	ν

(b)

TABLE 3. Weight for each input feature using permutation importance ranking for (a) LR poly2 and (b) LR poly3.

References

Breiman, L. (2001). Random forests. `https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf`.

Durbin, P. A. (2018). Some Recent Developments in Turbulence Closure Modeling. *Annual Review of Fluid Mechanics*, pages 1–47.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Langtry, R. B. and Menter, F. R. (2009). Correlation-Based Transition Modeling for Unstructured Parallelized Computational Fluid Dynamics Codes. *AIAA Journal*, 47(12):2894–2906.

Ling, J., Kurzawski, A., and Templeton, J. (2016). Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A., and et al. (2017). Automatic differentiation in pytorch.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D. (2014). CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. *Nasa Cr-2014-21878*, pages 1–73.

Wu, Z., Lee, J., Meneveau, C., and Zaki, T. (2019). Application of a self-organizing map to identify the turbulent-boundary-layer interface in a transitional flow. *Physical Review Fluids*, 4:1–18.

Zaki, T. A. (2013). From Streaks to Spots and on to Turbulence: Exploring the Dynamics of Boundary Layer Transition. *Flow, Turbulence and Combustion*, 91:451–473.