SAND2016-0249J

Machine Learning Strategies for Systems with Invariance Properties

Julia Ling[a], Reese Jones[a], Jeremy Templeton[a]

[a]*Sandia National Laboratories, 7011 East Avenue, Livermore, CA, 94550*

Abstract

In many scientific fields, empirical models are employed to facilitate computational simulations of engineering systems. For example, in fluid mechanics, empirical Reynolds stress closures enable computationally-efficient Reynolds Averaged Navier Stokes simulations. Likewise, in solid mechanics, constitutive relations between the stress and strain in a material are required in deformation analysis. Traditional methods for developing and tuning empirical models usually combine physical intuition with simple regression techniques on limited data sets. The rise of high performance computing has led to a growing availability of high fidelity simulation data. These data open up the possibility of using machine learning algorithms, such as random forests or neural networks, to develop more accurate and general empirical models. A key question when using data-driven algorithms to develop these empirical models is how domain knowledge should be incorporated into the machine learning process. This paper will specifically address physical systems that possess symmetry or invariance properties. Two different methods for teaching a machine learning model an invariance property are compared. In the first method, a basis of invariant inputs is constructed, and the machine learning model is trained upon this basis, thereby embedding the invariance into the model. In the second method, the algorithm is trained on multiple transformations of the raw input data until the model learns invariance to that transformation. Results are discussed for two case studies: one in turbulence modeling and one in crystal elasticity. It is shown that in both cases embedding the invariance property into the input features yields higher performance at significantly reduced computational training costs.

Keywords: machine learning, turbulence models, constitutive models, tensor invariants

1. Introduction

In many scientific domains, macroscopic phenomena are governed by microscopic interactions. In order to enable efficient simulations of such systems, empirical models are often employed to approximate the effect of the microscopic interactions. For example, in turbulence modeling, Reynolds Averaged Navier Stokes (RANS) simulations use empirical models for the Reynolds stresses to approximate the effect of turbulent transport on the mean momentum field. These simulations provide significant savings in computational cost in comparison to Direct Numerical Simulations (DNS), which directly resolve the turbulent fluctuations all the way down to the smallest scales. Empirical models

are also used in solid mechanics in the form of constitutive models to relate stress to deformation that is potentially complex at the atomic scale, and in thermodynamic relations for non-ideal gases, and in electromagnetism to define material properties such as conductivity.

Traditionally, these models have been developed through a combination of theoretical knowledge and calibration to a limited set of data from simplified test cases. For example, the widely-used Spalart-Allmaras RANS turbulence model was initially tuned to reproduce experimental results for two-dimensional mixing layers and wakes [1]. The wall functions commonly employed to model the effects of walls on turbulent flows were developed based on results for a flat plate boundary layer [2]. When defining the constitutive relations for hyper-elastic materials, Ogden [3] used physical arguments to determine the key terms, then fit the coefficients to experimental data. Valanis and Landel [4] used a similar methodology for forming their constitutive relation for hyper-elastic materials. These examples are typical of data-limited regimes, where simple models with only a few free parameters are desirable because they can avoid over-fitting when tuned to the limited amount of available experimental data.

More recently, the rise of high performance computing has led to the availability of large high-fidelity data sets in increasingly complex configurations. In fluid mechanics, Direct Numerical Simulations (DNS) resolve all scales of the time-dependent flow. In solid mechanics, Molecular Dynamics (MD) simulations resolve complex material deformation with atomic resolution. While it is not yet computationally feasible to run these high fidelity simulations for many full-scale applications, these simulations output huge quantities of data that can be mined to develop and improve empirical models. Furthermore, advances in experimental techniques, such as digital volume correlation [5, 6], tomographic particle image velocimetry [7], and magnetic resonance velocimetry [8], are yielding experimental data sets of unprecedented size and detail.

There have already been several efforts to use machine learning algorithms to leverage these large high-fidelity data sets to construct more accurate model closures. Milano and Koumoutsakos [9] used DNS results for a turbulent channel flow to train a neural network to reconstruct the near wall flow. Tracey et al. [10, 11] used machine learning algorithms to model the Reynolds stress anisotropy, and also to mimic the source terms from the Spalart-Allmaras turbulence model. Duraisamy et al. [12] used neural networks and Gaussian processes to model intermittency in transitional turbulence, and Zhang et al. [13] used these algorithms to model turbulence production in channel flow. Data driven models have also been used to calibrate models for sub-surface flows [14]. Ling and Templeton [15] used Random Forests to predict regions of high model form uncertainty in RANS results.

Data-driven methods have also been adopted to develop advanced constitutive models in solid mechanics. Ghaboussi [16, 17] proposed the use of neural networks for constitutive modeling. Neural networks have been used to model the stress-strain relation in a variety of materials, including sand [18], viscoplastic materials [19], metal alloys [20], and concrete [21]. Feng and Yang [22] also used data-driven genetic evolution methods to derive a constitutive relation for composite materials. Shen et al. [23] and Liang et al. [24] implemented neural networks to model the strain energy function for hyper-elastic materials and elastomeric foams, respectively. Clearly, data-driven methods are gaining popularity in the context of solid mechanics as well.

One of the key questions when applying data-driven techniques to physical systems is

to what extent domain knowledge can and should be built into the algorithms. One school of thought in the machine learning community is that very little domain knowledge needs to be injected into the algorithm, and that given sufficient data, the algorithm should be capable of discerning patterns and structure on its own [25, 26]. One drawback of embedding domain knowledge into the machine learning model is that if that domain knowledge proves faulty, it can lead to diminished performance in the machine-learned model. In many physical systems, however, definitive and exact domain knowledge does exist in the form of known invariances and symmetries. A function is an invariant under a given transformation if its value does not change when its inputs are subjected to that transformation. A more thorough discussion of invariants is given in Sec. 2.1.

For example, the classical laws of motion are known to obey Galilean invariance, which states that these laws do not change in different inertial frames of reference. In fact, the laws of motion are intricately linked to invariance properties. Noether's theorem [27, 28] describes how symmetry properties represent fundamental constraints that can be used to derive the laws of motion, underlining how central these properties are in physical systems. Given the importance of these symmetry properties, it would be desirable, both for model accuracy and physical realizability, for any machine learning model of a system with an invariance property to respect those invariances. In the context of fluid mechanics, Galilean invariance dictates that any scalar flow variable, such as pressure or velocity magnitude, will be invariant to rotations, reflections, or translations of the frame of reference. Likewise, the same principle dictates that solid deformation response will depend only on stretches, not on local rotations. Similarly, many crystalline materials exhibit material symmetry properties, such that scalar functions like the internal strain energy will be invariant to rotations within the symmetry group associated with the crystal.

While there has been increasing interest in applying data-driven methods to closure models for physical systems, there is no real consensus in the literature on how the input feature sets for these models should be chosen, and whether these input features should respect the invariance properties of the system. Whereas in traditional closure modeling, invariance properties are typically enforced explicitly, machine learning closure models have the option of using non-invariant inputs and then training the model to learn the invariance property. Milano and Koumoutsakos [9] used the instantaneous velocity field as an input to their neural network to predict the velocity components in the near wall region; they made no effort to ensure rotational invariance in their neural network, since it was designed to make predictions in only a single flow configuration. When using neural networks to construct a model for the turbulence intermittency factor, Duraisamy et al. [12] used a hill-climbing feature selection technique to determine the key input features. The chosen features did not respect rotational invariance, and the model was trained and tested on only a single flow configuration. Lefik and Schrefler [29] used a neural network to model the constitutive relation for a super-conducting cable. Instead of using invariant input features, they trained their neural network on several rotations of their original training data to try to enforce invariance properties in the model. While Tracey et al. [10] and Ling and Templeton [15] both used Galilean invariant input features for their machine learning models, both studies used physical intuition to guide their choice of input features. A more thorough understanding of the benefits of using invariant features and a comprehensive methodology for determining a complete input feature basis, would serve to significantly aid efforts to apply machine learning to

constitutive modeling applications.

There has been some research in the machine learning community on learning invariance properties, particularly in the domain of image recognition. For example, in face detection, it would be desirable to be able to detect a face in an image regardless of the orientation of the face. Similarly, in text processing, models should be able to interpret handwritten letters even if they are slightly distorted, scaled, or rotated. In the context of text processing for documents with some mis-printed letters, Baird [30] suggested expanding the training set to including distorted versions of the initial training data. For the application of hand-written digit recognition, Simard et al. [31] developed the Tprop algorithm, which made use of cost function derivative information to better learn invariance to distortion. Decoste et al. [32], working with support vector machines on the same problem of digit recognition, suggested the idea of using jittering kernels to find the closest match between any two examples within an invariance space. Other researchers have proposed using more complex machine learning models, such as deep convolutional neural networks [33] or weight-sharing neural networks [34] for image recognition systems with invariance properties.

There are several distinguishing factors between the invariance properties of image recognition and those considered in the present study. First, the invariance properties considered here are exact. In digit recognition, for example, exact rotational invariance is not desirable, as it would lead to confusion in differentiating a 6 from a 9. Second, in many physical systems, it is important for invariance properties be respected perfectly. While 95% invariance to rotation may be tolerable in face detection, it can lead to non-realizable results or numerical divergence in many physical applications. Lastly, the invariance properties and inputs are fundamentally different. Whereas in image recognition, the input is an N×N (or N×N×3 in the case of color images) image matrix, in physical systems, the relevant inputs are often 3×1 vectors or 3×3 tensors which represent local properties at a given point in 3-dimensional space and are invariant under a group action. In these physical systems, it is possible to derive a finite invariant basis of inputs.

The main question addressed in this paper is how to teach a machine learning model about an invariance or symmetry property in a physical system. Two main approaches will be explored. In the first approach, a basis of invariant inputs is constructed and the algorithm is trained on this basis. This approach uses domain knowledge to embed the invariance into the input basis, such that the output will necessarily obey the invariance. This basis is complete: any invariant function can be expressed in terms of this basis, as will be discussed further in Section 2. In the second approach, the machine learning algorithm is trained on the raw data, not on a basis of invariants. In order to teach the model to be invariant to a given transformation, the training data set must be expanded to include transformed states of the system. For example, to teach a model to be rotationally invariant, the algorithm would be trained on data from multiple angular orientations of the system. In this way, the model has the opportunity to learn the invariance property on its own, without that property being specifically embedded into the algorithm or inputs.

The purpose of this paper is two-fold. First, it presents the procedure for building a basis of invariant inputs that can serve as a systematic methodology for constructing a complete input feature set in which no relevant information has been omitted. This procedure will be of use to researchers hoping to develop data-driven models for physical systems with invariance properties. Second, it compares the performance of a model

trained on this invariant basis to that of a model trained on raw, non-invariant inputs. It is expected that the model trained on the invariant basis will be more effective at enforcing this invariance, since it is an embedded property, not just a learned property. However, a key question of interest is how much training data and computational resources are needed in order for a model to learn an invariance property. While the case studies presented in this paper come from the fields of fluid and solid mechanics, the results have broader implications for the application of data-driven techniques to any system with a symmetry property. Section 2 will give an overview of how a basis of invariants can be constructed, and of the overall methodology by which the machine learning algorithms were trained and tested. Section 3 will describe case studies in turbulence modeling and crystal elasticity modeling. In both cases, results will be presented that quantify the ability of the machine learning algorithms to learn the relevant invariance properties. Section 4 will present conclusions and ideas for future research.

2. Methodology

2.1. Construction of an Invariant Basis

One method of embedding an invariance property into a machine learning model is to formulate all of the inputs to the model such that they are invariant. Necessarily, then, the output prediction will also be invariant. Given an arbitrary set of tensors, vectors, and scalars that are potential raw inputs, this method requires formulating these into invariant inputs. Many previous researchers have relied on physical intuition to formulate these invariant inputs [10, 12, 15], but this method risks excluding key information from the input set if the modeler omits a relevant invariant. It would be more useful to have a systematic process for creating a complete functional basis of invariants. The relevant questions then become whether or not such a basis exists, whether it is finite, and how it can be constructed. To answer these questions, this section will give a very brief overview of pertinent results from group theory, representation theory, and invariant theory.

A group is a set of elements and a binary operator which satisfy the following conditions: i) associativity, ii) every element has an inverse also in the group, iii) the group contains an identity element which is its own inverse, and iv) the group is closed under the action of the operator (i.e. if x and y are in a group with operator $\bigoplus$, then $x \bigoplus y$ is also in the group) [35]. In representation theory, these groups are represented as linear transformations, such that each element of the group is represented by a matrix and the operator is matrix multiplication [36]. For example, the set of all invertible n×n matrices and the operation of matrix multiplication form a group representation, called the general linear group of degree n, GL(n). Symmetry and invariance properties can often be formulated conveniently in terms of algebraic groups and their matrix representations.

A function is invariant under a given group if its output value does not change when its inputs are subjected to the transformation specified by any element in that group. For example, a function of a vector and a tensor $f = f(\mathbf{v}, \mathbf{A})$ is rotationally invariant if:

$$f(\mathbf{Qv}, \mathbf{QAQ}^T) = f(\mathbf{v}, \mathbf{A}), \quad \forall \mathbf{Q} \in \mathrm{SO}(3) \tag{1}$$

The proper orthogonal group, SO(3), is the group of all rotation matrices in 3 dimensions. A function that is invariant to the proper orthogonal group is called *isotropic*. A polynomial invariant is an invariant polynomial function of the tensor elements. A

given second-order tensor has 3 independent isotropic polynomial invariants: its trace $Tr(\mathbf{A})$, the trace of its square $Tr(\mathbf{A}^2)$, and the trace of its cube $Tr(\mathbf{A}^3)$. Any isotropic polynomial function of $\mathbf{A}$ can be written as a function of these three invariants, so these invariants form an *integrity basis*. These invariants also form a *functional basis*, meaning that any general isotropic function of $\mathbf{A}$ can be formulated as a function of these invariants [37]. Notably, if we have two tensors $\mathbf{A}$ and $\mathbf{B}$, the functional basis of these two tensors is more than just the union of the basis of each individual tensor–the basis will also include invariants that depend on the product of these tensors. The question then becomes how to construct a basis of invariants for an arbitrary collection of tensors and vectors.

In fact, it can be shown that such a basis can be constructed in most cases of interest. For an arbitrary finite collection of vectors and tensors and a given symmetry property defined by either a finite group or the orthogonal group, the Hilbert Basis Theorem [38, 39] states that a finite integrity basis of invariants can be constructed. Any invariant function of that collection of tensors and vectors can be formulated as a function of the corresponding invariant basis [39].

In order to construct an invariant basis, the easiest approach is to refer to the literature: invariant bases have been thoroughly tabulated for arbitrary combinations of vectors and tensors for most common symmetry groups, including the orthogonal group [39, 40, 41, 42, 43]. Therefore, the construction of an invariant basis consists simply of referencing the table corresponding to the desired number of tensors and vectors for the relevant symmetry group. Thus, a finite basis of invariants both exists and is simple to construct for most symmetry properties of interest. This systematic procedure for creating an input basis mitigates the risk of omitting a key invariant from the input set, given the correct set of raw tensor and vector inputs. However, if the modeler neglects to include a key tensor in the input set, creating a complete basis of an incomplete set of tensor inputs will still result in missing information and poor model performance. It should also be noted that these invariant bases are not unique, and the machine learning model performance may vary with the choice of basis. Therefore, physical intuition can still play a role in developing the final input basis.

2.2. Machine Learning Model Construction

Training a machine learning model is the process by which the model is fit to the data; this same process is often termed calibration or fitting in the context of regression or Bayesian inference. The training data are all the data points used in this training process. Two different approaches for teaching the machine learning model about a system invariance were compared. In the first approach, the raw inputs were converted into an invariant basis for the known system invariances. The machine learning model was then trained on this basis, effectively embedding the invariance in the model. In the second approach, the raw inputs were transformed according to the known invariance property and the model was trained on this transformed data. For example, in the case of rotational invariance, the original data were rotated a specified number of times, and all the rotated data were compiled into one larger training set that was used to learn the invariance. The data were rotated in evenly spaced extrinsic rotations of the Euler angles around the cardinal axes: $\mathbf{Q}(\mathbf{e_3}, \theta_3)\mathbf{Q}(\mathbf{e_2}, \theta_2)\mathbf{Q}(\mathbf{e_1}, \theta_1)$, where $\theta_3, \theta_1 \in [0, 2\pi]$ and $\theta_2 \in [0, \pi]$. Figure 1 shows this work flow in a block diagram format.

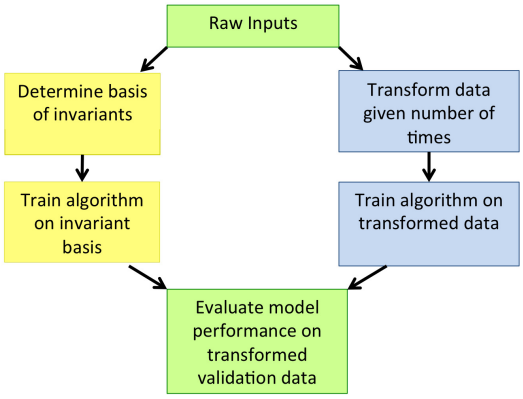


Figure 1: Block diagram of two approaches for training and evaluating machine learning models.

2.3. Machine Learning Algorithms

Two different machine learning algorithms were investigated. The first algorithm was a Random Forest (RF) regressor, implemented using the sci-kit learn open source Python library [44]. RFs are composed of an ensemble of binary decision trees. Each decision tree is trained on a random subset of the training data, obtained through random sampling with replacement (i.e. bagging). Each branch of the decision tree uses an if-then logic that determines an input and a threshold value at which to split the data, based on minimizing the Gini impurity score [45]. The result is that data points that end up at the same leaf node tend to have similar output values. In Random Forests, extra randomness is incorporated into the tree generation process by only using a random subset of the input features to determine the split criterion at each branch [46]. The final prediction of an RF regressor is given by the average prediction value among all the trees in the ensemble. RFs are an attractive option because of their robust behavior and easy implementation [47]. There are only two tunable hyper-parameters in RF implementation: the ensemble size and the maximum tree depth.

The maximum tree depth is a key parameter because it trades off between model performance and memory usage: full-depth trees will have maximal performance, but may result in excessive memory usage. For RFs with no cap on the maximum tree depth, the number of tree leaves tends to scale linearly with the training data set size (if each data point adds new information to the data set), and the tree depth scales roughly logarithmically with the number of tree leaves [48]. Therefore, in the case of the RF trained on multiple transformations of the original training data set, the RF size and memory usage would be expected to increase with the number of transformations included in the training data set. In the extreme case, this increase in tree size can cause memory overflow issues.

In order to explore the effect of imposing a maximum tree depth, in Section 3.1 for the turbulence modeling case study, the maximum tree depth was set at 15. This maximum depth was chosen because it was shown to be sufficient for the RF trained on the basis of invariants, causing only a 2.5% increase in error relative to using full-depth trees. On

the other hand, in Section 3.2, the case study in crystal elasticity, no maximum tree depth was imposed, and trees were allowed to grow to their full depth. In this case, the RFs trained on many transformations of the training data grew deeper and fuller to accommodate the large training data size. Results from these two case studies can therefore be compared to assess the effect of prescribing a maximum tree depth.

In general, the RF performance will improve as the number of trees in the ensemble increases, but with diminishing returns. An ensemble size of 500 was prescribed for both case studies as it was determined that at this size, the model performance was no long strongly dependent on the ensemble size.

The second algorithm that was evaluated was the Neural Network (NN) [49, 50]. While NNs are neither as robust nor as easy to implement as RFs, they are capable of deep learning, in which low-level features are combined and transformed into higher-level features. This capability could allow them to learn meta-properties like symmetry or invariance more easily. Multi-layer perceptrons were implemented through the ffnet open source Python package [51]. These networks consist of layers of neurons. Each neuron non-linearly combines its inputs through the sigmoid function, and its output is used as an input to the next layer. The layers are densely connected: the output of each neuron from one layer is used as an input for every neuron in the following layer.

The network architecture was determined through a constructive method, starting with a small network with only one hidden layer. In general, a network with more hidden nodes can capture more complex behavior but with increased risk of overfitting the data [52]. The performance of the network was monitored as more nodes and hidden layers were added until the addition of more nodes or hidden layers did not improve model performance on validation points. The number of neurons in each layer was directly proportional to the number of input features, N_{in}. In the turbulence modeling case study, the network architecture was $N_{in} - 6N_{in} - 12N_{in} - 6N_{in} - 2N_{in} - 1$ (meaning N_{in} nodes in the first layer, $6N_{in}$ nodes in the second layer, $12N_{in}$ nodes in the third layer, etc.), and in the crystal elasticity case study it was $N_{in} - 3N_{in} - N_{in} - 1$. Network training was accomplished through a stochastic truncated Newton optimization scheme, implemented in parallel in the ffnet python library [51].

Over-fitting was avoided through the use of an early stopping criterion [53]. Overfitting occurs when the model error rate is lower on the training set than on the validation set. 10% of the training data was held out and not used for training, and the model performance on this subset was monitored during training. An increase in model error rate on this subset during training indicated the onset of overfitting, and triggered the halt of the training process, thereby preventing over-fitting. It should be noted that this held-out subset of the training data was separate from the validation data later used to evaluate model performance.

3. Case Studies

Two different case studies are presented to demonstrate the process of constructing an invariant basis and to investigate the comparative performance of models that use invariant inputs versus models that use non-invariant inputs. The first case study is from turbulence modeling, and discusses the prediction of the anisotropy of the Reynolds stress tensor from the mean strain rate and rotation rate tensors. The second case study is from solid mechanics, and examines the strain energy function of a material exhibiting

crystalline symmetry. Both case studies begin by presenting an overview of the quantity of interest and how it is typically modeled, then discuss the physical invariances of the system and detail the construction of an invariant basis. Finally, the accuracy and computational cost of the machine learning algorithms are presented and discussed.

3.1. Case Study from Turbulence Modeling

3.1.1. Problem Statement

Reynolds Averaged Navier Stokes (RANS) turbulence models rely on empirical closures for the Reynolds stresses $\overline{\mathbf{u} \otimes \mathbf{u}}$, the transport of momentum due to turbulent fluctuations. A very common closure for the Reynolds stresses is the Boussinesq hypothesis [2]:

$$\overline{\mathbf{u} \otimes \mathbf{u}} = \frac{2}{3}k\mathbf{I} - 2\nu_t\mathbf{S} \tag{2}$$

In Eqn. 2, $\mathbf{u}$ is the fluctuating component of the velocity field, k is the turbulent kinetic energy, $\mathbf{I}$ is the identity matrix, ν_t is the eddy viscosity, and $\mathbf{S}$ is the symmetric mean strain rate tensor. This closure has the advantages of simplicity and aiding in convergence. However, in many flows, it gives poor predictions of the Reynolds stress anisotropy. The non-dimensional Reynolds stress anisotropy tensor is given by $\mathbf{A} = \frac{1}{k}\overline{\mathbf{u} \otimes \mathbf{u}} - \frac{2}{3}\mathbf{I}$. In flows with significant anisotropy, RANS models relying on the Boussinesq hypothesis are prone to elevated uncertainty [54]. Ling and Templeton [15] developed a machine learning model to predict regions of the flow with high anisotropy as an indicator of where RANS predictions would have high uncertainty. They used the second invariant of the anisotropy tensor, $\mathrm{II}_a = \frac{1}{2}[(Tr(\mathbf{A}))^2 - Tr(\mathbf{A^2})]$, to quantify the degree of anisotropy [2].

In order to quantify the uncertainty of a given RANS simulation, it would be useful to be able to predict the Reynolds stress anisotropy, given RANS simulation results for the flow field. Therefore, in this case study, the target quantity of the machine learning models was the second invariant of the Reynolds stress anisotropy II_a. The true value of II_a, required to train and evaluate the machine learning model, was calculated from DNS data. The input tensor to the machine learning model was the mean velocity field gradient $\boldsymbol{\nabla}\mathbf{U}$ as predicted by RANS. For modeling purposes, it is often convenient to decompose the velocity gradient into its symmetric and anti-symmetric components, $\mathbf{S}$ and $\mathbf{R}$, respectively. Pope [55] suggested the following non-dimensionalization for these tensors: $\mathbf{S} = \frac{1}{2}\frac{k}{\epsilon}(\boldsymbol{\nabla}_\mathbf{x}\mathbf{U} + \boldsymbol{\nabla}_\mathbf{x}^T\mathbf{U})$ and $\mathbf{R} = \frac{1}{2}\frac{k}{\epsilon}(\boldsymbol{\nabla}_\mathbf{x}\mathbf{U} - \boldsymbol{\nabla}_\mathbf{x}^T\mathbf{U})$. In the interest of consistency with the non-linear eddy viscosity modeling efforts of Craft et al. [56], Pope [55], and others [57, 58], the inputs in this case study were formulated in terms of these non-dimensionalized $\mathbf{S}$ and $\mathbf{R}$.

3.1.2. Invariance Properties

The Navier-Stokes equations obey Galilean invariance: the laws of motion do not change for inertial transformations of the frame of reference. In the context of this problem, this invariance dictates that the Reynolds stress anisotropy invariant at a given point in the flow will not change if the reference frame is translated, moved at constant velocity, rotated or reflected. $\mathbf{S}$ and $\mathbf{R}$ are invariant to translations and constant velocity offsets, but not to reflections or rotations. Therefore, to construct a basis of invariants

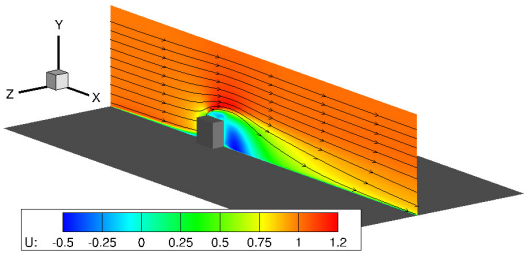


Figure 2: Schematic of wall-mounted cube in crossflow flow configuration. Solid bottom wall and cube shown in gray. Contours of streamwise velocity shown on mid-plane with streamlines overlaid in black.

to Galilean transformations from these two tensors, that basis must be invariant to the orthogonal group. The full orthogonal group is comprised of matrices for which the transpose is equal to the inverse ($\mathbf{Q}^T = \mathbf{Q}^{-1}$). This group includes all reflection and rotation matrices.

The basis of invariants of $\mathbf{S}$ and $\mathbf{R}$ to the orthogonal group can be constructed using tables from Johnson [39] which enumerate the invariants to the orthogonal group for a symmetric and anti-symmetric tensor:

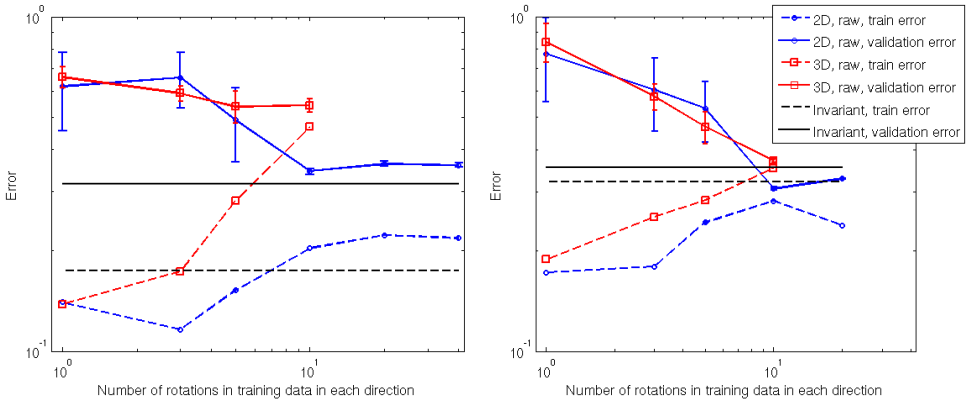
$$\begin{aligned} I_0 &= Tr(\mathbf{S}) \\ I_1 &= Tr(\mathbf{S}^2) \\ I_2 &= Tr(\mathbf{S}^3) \\ I_3 &= Tr(\mathbf{R}^2) \\ I_4 &= Tr(\mathbf{R}^2\mathbf{S}) \\ I_5 &= Tr(\mathbf{R}^2\mathbf{S}^2) \\ I_6 &= Tr(\mathbf{R}^2\mathbf{S}\mathbf{R}\mathbf{S}^2) \end{aligned} \tag{3}$$

In incompressible flows, continuity dictates that $Tr(\mathbf{S}) = 0$, so this integrity basis can be reduced to 6 non-zero invariants: $I_1, ..., I_6$.

Therefore, when applying the two approaches described in Section 2.1, the machine learning model can either be trained using the 9 distinct tensor components (6 from the symmetric $\mathbf{S}$, and 3 from the skew symmetric $\mathbf{R}$) or using the integrity basis of 6 invariants $I_1, \ldots, I_6$.

3.1.3. Data set

The data sets that were used to train and evaluate the machine learning model in this case study were a RANS and DNS of a wall-mounted cube in crossflow. These simulation results have been previously presented by Rossi et al., and details of the computational set-up are contained in Refs. [59, 60]. The RANS simulation was performed using the k-ϵ model, as implemented in FLUENT 12.0. The Reynolds number based on the free stream velocity and cube height was 5000. Figure 2 shows a schematic of this flow configuration with contours of mean streamwise velocity at the mid-plane, as calculated using DNS. As this figure shows, there is a substantial recirculation region downstream of



(a) Random Forest (b) Neural Network

Figure 3: Error as a function of number of training rotations for (a) Random Forests and (b) Neural Networks in the turbulence modeling case study. The models trained on a basis of invariants (black line, no symbols) are not trained on any rotated data, since they are invariant to rotations. The models trained on raw tensor components are trained on evenly spaced rotations of the training data in 2D (circles) and in 3D (squares). Training error shown with dashed lines, validation error shown with solid lines. The error bars represent the standard deviation of the error.

the cube. Ling and Templeton [15] showed that there are regions of significant Reynolds stress anisotropy in this flow, particularly in the stagnation region immediately upstream of the cube. With its three-dimensionality, flow curvature, stagnation and separation, this flow configuration presents a challenging case on which to train and test a machine learning model for Reynolds stress anisotropy.

15,000 points were randomly sampled from the wall-mounted cube data set. At each of these points, the tensors $\mathbf{S}$ and $\mathbf{R}$ were calculated, as was the Reynolds stress anisotropy invariant II_a. These 15,000 points were then randomly partitioned such that 10,000 were used for training the machine learning models and the other 5,000 were used for validation to test the model performance.

3.1.4. Results

The performance of the Random Forest algorithm was evaluated first. An RF was trained on the basis of invariants from Eqn. 3, and its performance was evaluated both on the training set (yielding the training error) as well as on the validation set (yielding the validation error) using the normalized error metric:

$$E = \frac{\sum |\mathrm{II}_{a,\mathrm{DNS}} - \mathrm{II}_{a,\mathrm{ML}}|}{\sum \mathrm{II}_{a,\mathrm{DNS}}} \tag{4}$$

In Eqn. 4, $\mathrm{II}_{a,\mathrm{DNS}}$ is the DNS value of the anisotropy invariant, $\mathrm{II}_{a,\mathrm{ML}}$ is the predicted value of the invariant from the machine learning model, and the sum is over all the data points. In effect, this error metric gives the mean difference magnitude between the true and predicted anisotropy, normalized by the mean value of the anisotropy.

Figure 3(a) plots the error E of the RF trained on the basis of invariants, as well as the RF trained on the raw tensor components. The RF trained on the raw tensor components uses a specified number of rotations of the training data in order to teach the algorithm rotational invariance. As a first step, the RF was trained only on 2D rotations (only rotations in the θ_3 direction), and was tested on 20 realizations of random θ_3 rotations of the validation data. The case of only 2D rotations is interesting because learning 2D rotational invariance is an easier problem that reveals how performance and computational requirements vary with the complexity of the invariance property. As a second step, the RF was trained on full 3D rotations, and tested on 20 realizations of random 3D rotations of the validation data. Results for both of these steps are shown in Figure 3. In the case of 3D rotations, the plot shows the number of training rotations in each direction, so 10 rotations in each direction corresponds to 1000 total rotations of the training data.

As this figure shows, the validation error of the RF trained on the basis of invariants is approximately 0.32. This relatively high error level reflects the fact that local values of $\mathbf{S}$ and $\mathbf{R}$ will never be able to completely predict the Reynolds stresses, since non-local effects are non-negligible in turbulent flows. In the case of the RF trained on 2D rotations of the raw tensor components, the RF is able to learn the invariance relatively well ($\frac{E_{\text{raw,2D}}-E_{\text{invariant}}}{E_{\text{invariant}}} = 9\%$) when the training data is rotated at least 10 times. In the case of 3D rotations, the RF never successfully learns rotational invariance. The training error increases significantly as the number of training rotations increases, most likely due to the cap on the maximum depth of the trees which constrains the model complexity.

Figure 3(b) presents the error of Neural Networks trained on the basis of invariants and on the raw tensor components. The RF and NN trained on the basis of invariants have similar performance to each other ($\frac{E_{\text{NN}}-E_{\text{RF}}}{E_{\text{RF}}} = 12\%$). The NN trained on raw tensor components learns invariance to 2D rotations after 10 rotations of the training data. In fact, this NN is able to achieve a validation error lower than the NN trained on the invariant basis. This is possible because the invariant basis is over-constrained in the case of 2D invariance, since this basis is invariant to the full 3D rotation group. In the case of 3D rotations, the NN trained on 10 rotations in each direction of the raw tensor components is able to achieve a validation error only 5% higher than the NN trained on the invariant basis. These results demonstrate that the NNs are capable of learning rotational invariance if their training data is rotated a sufficient number of times.

It is noteworthy that training the NN on 1000 times the initial training data set to achieve this 3D rotational invariance drastically increased the computational cost of

Table 1: Computational Requirements of Various Algorithms in the Turbulence Modeling Case Study

Algorithm	**Compute Time (CPU hours)**	**Memory Usage**
Invariant RF	0.03	0.4 GB
Raw RF, 10 rotations in 2D	0.3	0.9 GB
Raw RF, 10 rotations in 3D	46	13 GB
Invariant NN	7	0.1 MB
Raw NN, 10 rotations in 2D	105	2 MB
Raw NN, 10 rotations in 3D	1176	2 MB

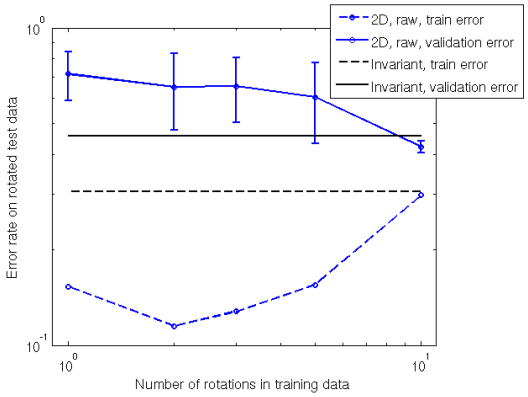


Figure 4: Error as a function of number of training rotations in 2D for Random Forests using only information from **S**, not **R**. Results are shown for an RF trained on the invariants of **S** (black line, no symbols) as well as the raw tensor components (circles). Training error shown with dashed lines, validation error shown with solid lines. The error bars represent the standard deviation of the error.

the network training. Table 1 details the computational requirements for both the RF and NN, including the CPU hours needed to train the algorithm and the final memory requirement to store the trained model. The training was performed in parallel on a system with 32 GB of RAM and 12 2.2 GHz processors per node. As this table shows, the computational cost of training the NNs was, in general, much higher than that of training the RFs. While RF training follows a greedy algorithm that lends itself to computational efficiency, training NNs requires iteratively solving a non-convex optimization problem. The RF and NN trained on 10 3D rotations of the raw tensor components required orders of magnitude longer to train than the RF and NN trained on the basis of invariants, respectively.

Even with the maximum tree depth capped at 15, the memory usage of the RFs significantly exceeded that of the NNs. In particular, when the RF was trained on 3D rotations of the training data, the memory required to store the trained RF exceeded 10 GB. This increase in memory usage is due to an increase in the number of tree nodes. The depth of a tree is given by the maximum number of branches connecting any leaf node to the root node. However, it is very possible for one leaf node to be connected to the root through 15 branchings, and for another leaf node to be connected to the root through only 2 branchings. While a tree of depth 15 could have as many as $2^{15} = 32,768$ leaves, the invariant RF trees had only 2,370 leaves on average, indicating that many leaves connected to the root through fewer than 15 branchings. The RF trained on 3D rotations of the data, on the other hand, had trees with 17,500 leaves on average, accounting for the much higher memory usage in this case. In comparison, all of the NNs used less than 0.01 GB of memory.

An interesting question is what governs the number of training rotations that are required before the machine learning model learns the invariance property. In this case study, approximately 10 training rotations were needed in each direction for either algorithm to learn rotational invariance. In order to better understand this dependence, a

sub-problem was investigated in which only dependence on **S**, not **R**, was permitted. In this case, one RF was trained on the 6 distinct tensor components of **S**, and another RF was trained on the 2 non-zero invariants of **S**. The results for this case are shown in Fig. 4 for 2D rotations. As this figure shows, the RF trained on the tensor invariants of **S** has a higher error rate than when it was trained on those of both **S** and **R**, confirming that **R** contains useful information content. The RF trained on the raw tensor components of **S** learns the invariance property after 10 training rotations–the same number of rotations required for learning invariance when using the components of both **S** and **R**. Therefore, the number of required training rotations appears to be somewhat independent of both the problem complexity and the algorithm used. Perhaps approximately 10 rotations corresponds to the point at which interpolating between the rotated training data yields accurate results.

Up to this point, the focus has been on rotational invariance. For full Galilean invariance, invariance under reflection of the reference frame is also required. The invariant basis proposed in Eqn. 3 is invariant under the full orthogonal group, including reflections. The full orthogonal group can be generated by products of the proper orthogonal group (rotation matrices only) and central inversion (the identity matrix times -1) [61]. Therefore, training a machine learning model to obey invariance to reflection required doubling the training data set size to include centrally inverted transformations of the rotated training data, as well. The NNs were evaluated for their ability to learn invariance to the full orthogonal group, including reflections. A NN was trained on 10 training rotations in each direction in 3D, as well as central inversions of each of those training rotations (2,000 transformed versions of the original training data in all). The NN was then tested on 20 random rotations and reflections of the validation data. The validation error of the NN trained and tested in this manner was $E = 0.40$: 8% higher than the error of the NN trained and tested on only 3D rotations without reflections, and 14% higher than the NN trained on the basis of invariants. Therefore, training the NN to obey invariance to reflections and rotations in 3D requires a huge increase in training data set size, and is not able to fully recover the performance of the NN trained on the basis of invariants.

3.2. Case Study from Crystal Elasticity

3.2.1. Problem Statement

In solid mechanics, constitutive relations between the stress and strain in a material are required to predict material deformation. In the case of elastic materials, a strain energy function, W, can be introduced, where W is a function of the deformation gradient, $\mathbf{F}$, which is the derivative of the current material positions with respect to their reference positions. The strain energy function can be differentiated with respect to $\mathbf{F}$ to recover the stress [62]. Thus, for elastic materials, the problem of defining a constitutive relation can be reduced to the problem of determining the value of W as a function of $\mathbf{F}$. Hence, the target quantity for the machine learning models to predict is the strain energy function, and the tensor input is the deformation gradient.

3.2.2. Invariance and Symmetry Properties

In this case, the strain energy function W exhibits two invariance properties. First, W is invariant to rigid body rotations. Therefore, $W(\mathbf{F}) = W(\mathbf{QF})$ for any $\mathbf{Q}$ in the

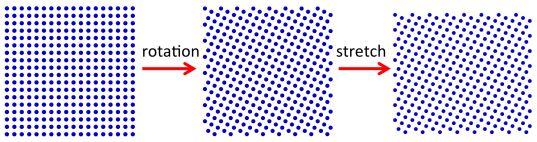


Figure 5: 2D Schematic of the data generation process for the crystal elasticity case study.

proper orthogonal group. Second, the material of interest in this case study has a cubic symmetry, so the W will not change if the crystal is rotated by an element $\mathbf{G}$ of the cubic symmetry group: $W(\mathbf{F}) = W(\mathbf{FG})$. The cubic symmetry group, also called the octahedral group, has 48 members which generate quarter rotations and inversions about the coordinate axes.

A basis of six invariants to both rigid body rotations and the cubic symmetry group was constructed using the tables in Ref. [63]:

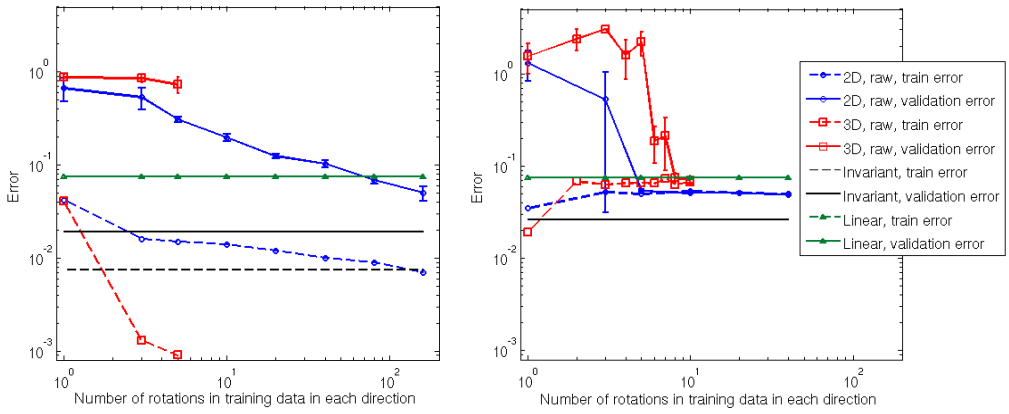
$$\begin{aligned} I_1 &= \mathrm{Tr}\,\mathbf{C} \\ I_2 &= \mathrm{Tr}\,\mathbf{C}^2 \\ I_3 &= \mathrm{Tr}\,\mathbf{C}^3 \\ I_4 &= \mathbf{C} : \mathbb{M} : \mathbf{C} \\ I_5 &= \mathbf{C} : \mathbb{M} : \mathbf{C}^2 \\ I_6 &= \mathbf{C}^2 : \mathbb{M} : \mathbf{C}^2 \end{aligned} \tag{5}$$

In Eqn. 5, $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ is the right Cauchy-Green deformation gradient tensor. $\mathbb{M} = \sum_{i=1}^{3} \mathbf{e}_i \otimes \mathbf{e}_i \otimes \mathbf{e}_i \otimes \mathbf{e}_i$ is the fourth order structure tensor characteristic of the cubic symmetry group. The operator : denotes the double inner product. For example, for tensors $\mathbf{A}$ and $\mathbf{B}$ and structure tensor $\mathbf{M}$, $\mathbf{A} : \mathbb{M} : \mathbf{B} = \sum_{i,j,k,l=1}^{3} \mathbf{A}_{ij}\mathbf{M}_{ijkl}\mathbf{B}_{kl}$. In the case of the cubic structure tensor, this expression can be further simplified to: $\mathbf{A} : \mathbf{M} : \mathbf{B} = \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{22} + \mathbf{A}_{33}\mathbf{B}_{33}$.

Therefore, the machine learning models will either be trained on the 6 invariants $I_1, \ldots, I_6$ specified in Eqn. 5, or on the 9 components of $\mathbf{F}$.

3.2.3. Data set

The data used to train and validate the machine learning models were generated using small face-centered cubic lattices interacting with an Embedded Atom Method (EAM) potential for Nickel [64]. In the interest of giving the machine learning models a simplified data set on which to learn, the temperature was set to zero. Whereas in the turbulence modeling case study, the training set was generated through a single simulation run with the different data points representing different locations in the flow, in this case study, the training set was constructed of individual samples of the strain energy function for varying crystal orientations and stretches. Perfect lattices were first rotated then stretched triaxially and the sum of the pair-wise bond energies was equated to strain energy. This process is shown schematically in Figure 5.



(a) Random Forest (b) Neural Network

Figure 6: Error as a function of number of training rotations for Random Forests (a) and Neural Networks (b) in the solid mechanics case study. The models trained on a basis of invariants (black line, no symbols) are not trained on any rotated data, since they are invariant to rotations. The results for a linear regression model using the invariant basis as inputs are shown for comparison (triangles). The models trained on raw tensor components are trained on evenly spaced rotations of the training data in 2D (circles) and in 3D (squares). Training error shown with dashed lines, validation error shown with solid lines. The error bars represent the standard deviation of the error.

Using the same method as in the turbulence modeling case study, 15,000 points were randomly sampled from this data set, of which 10,000 were used for training and the remaining 5,000 were used for validation.

3.2.4. Results

The two invariance properties were studied separately, by either left multiplying the deformation gradient tensor by an element of the proper orthogonal group, or by right multiplying the deformation gradient tensor by an element of the cubic symmetry group. Figure 6 shows the performance of the RFs and NNs in learning the invariance to rigid body rotations. Since the temperature was set to zero, the error in the NN and RF trained on the basis of invariants is very low ($E = \frac{\sum |W_{\mathrm{MD}} - W_{\mathrm{ML}}|}{\sum W_{\mathrm{MD}}} < 3\%$). Unlike in the turbulence modeling case study, an exact functional form exists for $W(\mathbf{F})$, so the error could theoretically go to zero. A Cauchy-Born model can exactly represent the strain energy response to homogeneous deformations of a perfect crystal. The strain energy of a Cauchy-Born model is the sum of the bond energies given by the particular potential each stretched by the deformation gradient. Figure 6 also shows the error for a linear model trained on the tensor invariants. In the case of the linear model, the training error differed by less than 1% from the validation error, indicating that no over-fitting occurred for this very simple model. Because this material has a non-linear constitutive equation due to the form of the EAM potential for the range of forces being applied, the linear model has a higher error than the RF and NN trained on the invariant basis, as these algorithms are able to better capture this non-linearity.

Because the RFs did not have a prescribed maximum depth in this case, the training

error remained low even as the number of training rotations increased. By 160 2D training rotations, the RF had learned invariance to 2D rigid body rotations with only a 5% validation error. However, the RFs were not able to learn invariance to 3D rigid body rotations before memory overflow occurred. Table 2 presents the computational costs of the various algorithms in this case study. Once again, the compute time for the RFs was significantly less than that of the corresponding NNs. However, in this case, because the trees were allowed to grow to full depth, the memory usage of the RFs quickly became excessive when they were trained on multiple rotations of the training data. For the RF trained on the basis of invariants, the average tree depth was 20 and the average number of leaves per tree was 2650. For the RF trained on 5 rotations in each direction in 3D, the average tree depth was 24 and the average number of leaves per tree was 238,000. This difference in tree size manifested in an increase of memory usage to over 60 GB for the RF trained on the 3D rotations. Models with such high memory usage would be unwieldy to deploy in many practical applications.

The NNs, on the other hand, were able to learn invariance to both 2D and 3D rigid body rotations, as shown in Figure 6(b). The NN error is lower than that of the linear model, indicating that the NN has also captured the non-linear behavior to some extent. However, the NNs trained on the rotated raw data were not able to recover as low an error as the NN trained on the tensor invariants.

Figure 7 compares the error in the predicted values of W to the true values. The predictions are shown for the NN trained on the invariant basis, the linear fit trained on the invariant basis, and the NN trained on raw tensor components rotated 10 times in each direction in 3D. As this figure shows, none of the models are perfect–all of them have non-zero errors. However, the NN trained on the invariant basis has by far the most accurate predictions at very high and very low values of W. While the NN trained on the raw tensor components has slightly better performance than the linear fit, neither of these two models is able to fully capture the non-linear behavior in W. Overall, the relative difference in error between the linear fit to the invariants and the NN trained on the rotated raw tensor components is $\frac{E_{\text{linear}} - E_{\text{raw, NN}}}{E_{\text{raw, NN}}} = 7\%$. Therefore, even after training on 1000 rotations of the training data, the NN is only barely able to surpass the performance of a simple linear regression on the invariant basis.

A relevant question is why the NN trained on the raw tensor components fails to fully capture the non-linear dependence of W on $\mathbf{F}$. One possible explanation is that the NN has a fixed number of neurons, so as it is trained, it uses those neurons to minimize the cost function as efficiently as possible. If the NN does not learn rotational invariance,

Table 2: Computational Requirements of Various Algorithms in the Crystal Elasticity Case Study

Algorithm	**Compute Time (CPU hours)**	**Memory Usage**
Invariant RF	0.03	1.1 GB
Raw RF, 160 rotations in 2D	5	185 GB
Raw RF, 5 rotations in 3D	5	61.5 GB
Invariant NN	0.6	0.04 MB
Raw NN, 10 rotations in 2D	34	0.08 MB
Raw NN, 10 rotations in 3D	434	0.08 MB

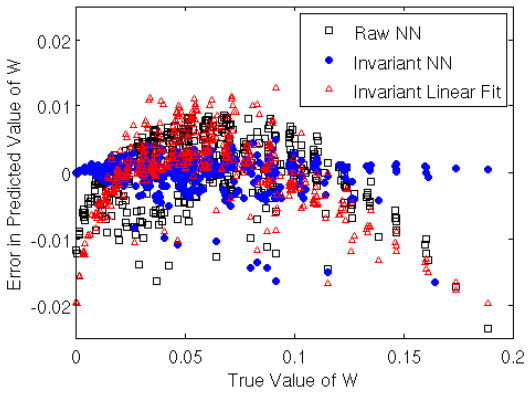


Figure 7: Scatter plot of error in the predicted value of the strain energy function, $W_{\text{predicted}} - W_{\text{true}}$, as a function of W_{true}. Results shown for the NNs trained on the basis of invariants (filled circles) and the raw tensor components rotated in 3D 10 times in each direction (empty squares) on a validation data set. Results are also shown for a linear fit to the invariant basis (triangles).

Fig. 6 shows that the resulting model has very high error ($> 100\%$). On the other hand, the non-linearity is a relatively minor effect ($< 10\%$ contribution to error, as the linear model results show). Therefore, it makes sense that in minimizing its cost function, the NN would learn rotational invariance before the non-linearity in W.

Separately, the NN was also investigated for its ability to learn cubic symmetry. The cubic symmetry group has 48 elements. If the NN was not trained on any of the cubic transformations of the training data, but was tested on cubic transformations of the validation data, the error was very high ($E > 2$). When trained on all 48 cubic transformations of the training data, the validation error on cubically transformed data was reduced to $E = 0.085$ (still 3.3 times as high as the error of the NN trained on the invariant basis). It has therefore been shown that in order for the NN to learn just rigid body rotation invariance, it required approximately 1,000 total training rotations in 3 directions (applied by left-multiplication of the deformation gradient tensor by elements of the proper orthogonal group), and for the NN to learn just cubic symmetry, it had to be trained on 48 cubic rotations and reflections of the training data (applied by right-multiplication of the deformation gradient tensor by elements of the octahedral group). Overall, then, to learn both the rigid body invariance and the cubic symmetry, approximately 48,000 times the original training data set would be required. This huge increase in training data set size would result in a significant increase in network training time.

4. Conclusions

The rise of high performance computing is opening up new possibilities for applying machine learning to physical systems to develop advanced data-driven empirical models based on high fidelity simulation results. A key question when using data driven models is to what extent domain knowledge should be embedded in the model versus learned through exposure to training data. In many physical systems, symmetry and invariance

properties are known *a priori*, and it is important that the empirical model respect these properties. This paper explored the ability of two different machine learning algorithms, Random Forests and Neural Networks, to learn invariance properties.

Two different approaches for incorporating the invariance properties into the machine learning models were explored. One approach was to build a basis of invariant inputs, such that the invariance property was embedded into the model. This paper discussed the theory and methodology of constructing such a basis of invariants using concepts from group theory and invariant theory. Several previous papers [10, 12, 15] have presented data-driven turbulence models in which physical intuition was used to craft the input feature set. The methodology of constructing an invariant basis provides a more systematic and exhaustive process for building the input feature set which is of immediate utility to machine learning practitioners building models for physical systems. While this process of building an invariant basis is not new, it is novel in the context of machine learning. The second approach was training the machine learning model on many transformed versions of the data until it had learned the invariance property. These two approaches were compared for two different case studies: one in turbulence modeling and one in crystal elasticity.

Comparisons between the results from the Random Forest and the Neural Network showed that when trained on an invariant basis, the two algorithms gave very similar performance in both case studies. However, NNs had better performance than RFs when trained on the raw tensor components. When a maximum tree depth was set for the RFs, both the training and validation error quickly rose as rotated versions of the data were added to the training data set. On the other hand, when no maximum tree depth was imposed, the RF memory usage became excessive. It is not surprising that the NNs were better able to learn invariance properties, since NNs are well known for their deep learning capabilities. These results show that when the algorithms are trained on a basis of invariants, the algorithmic requirements are significantly relaxed, and even simple models can have good performance. Indeed, in the solid mechanics case study, it was shown that a linear regression model trained on the basis of invariants had almost as good performance as the NN trained on the raw tensor components. Since both linear regression and RFs are significantly easier to implement than NNs, the good performance of these algorithms when trained on an invariant basis is a significant advantage of the invariant basis approach to embedding symmetries.

In both case studies, the NN required approximately 10 training rotations in each direction to learn rotational invariance. As discussed in Section 3.1, the exact dependence of this required number of training rotations is not fully understood, but appears to be largely independent of the problem complexity. This number of rotations could correspond to the number of training data points required to build an accurate interpolant using the sigmoidal activation functions of the NN neurons. The requirement of 10 training rotations in each direction led to an increase of training data set size of 3 orders of magnitude in order for the networks to learn rotational invariance in 3D. Moreover, in both case studies, additional symmetry properties led to further increases in training data set size. This increase in training data set size results in commensurate increases in training time. Therefore, while these case studies demonstrated that NNs are capable of learning invariance properties given an extensive enough training database, the computational requirements can become excessive. Furthermore, the resulting neural network performance was poorer than that of the network trained on the basis of invariants. It

was not unexpected that using a basis of invariants would yield better invariance properties than teaching a model invariance through repeated exposure to rotated training sets, since using the invariant inputs will exactly and directly embed the invariance property. What is more interesting is how much better the models trained on the invariant basis perform. They use less than 0.01% of the computational power to train, yet yield higher accuracy in both of the case studies explored.

It is important to note that no attempt was made to optimize the sampling of the training rotations. The rotations of the training set were evenly spaced in the three Euler angles. More complex sampling strategies would most likely be able to train the NN to learn rotational invariance with fewer training rotations. Nevertheless, these results demonstrate that for systems with multiple invariance properties, training the NN to learn these properties by expanding the training set quickly becomes intractable because of the blow up in training time. These results demonstrate the significant utility of constructing a basis of invariants and training the machine learning algorithm on this basis.

An interesting area for future work would be to apply feature selection algorithms to the invariant basis, and to compare the top ranking features from the invariant basis to the features selected through physical intuition in previous studies, such as that of Duraisamy et al. [12] and Ling and Templeton [15]. It would also be interesting to explore optimal sampling schemes for the training rotations in order to train the data-driven models to be rotationally invariant with as few training rotations as possible.

Acknowledgments

The authors wish to thank J. Ostein and K. Matulef for valuable comments on a draft of this paper. Funding for this work was provided by the Sandia LDRD program, and its support is gratefully acknowledged. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND

[1] P. Spalart, S. Allmaras, A one-equation turbulence model for aerodynamic flows, AIAA Paper 92-0439.

[2] S. Pope, Turbulent Flows, Cambridge University Press, 2000.

[3] R. Ogden, Large deformation isotropic elasticity-on the correlation of theory and experiment for incompressible rubberlike solids, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 326.

[4] K. Valanis, R. Landel, The strain energy function of a hyperelastic material in terms of the extension ratios, Journal of Applied Physics 38.

[5] H. Leclerc, J. Perie, S. Roux, F. Hild, Voxel-scale digital volume correlation, Experimental Mechanics 51 479–490.

[6] B. Bay, T. Smith, D. Fyhrie, M. Saad, Digital volume correlation: three-dimensional strain mapping using X-ray tomography, Experimental mechanics 39 217–226.

[7] F. Scarano, Tomographic PIV: principles and practice, Measurement Science and Technology.

[8] C. Elkins, M. Alley, Magnetic resonance velocimetry: applications of magnetic resonance imaging in the measurement of fluid motion, Experiments in Fluids 43 (2007) 823–858.

[9] M. Milano, P. Koumoutsakos, Neural network modeling for near wall turbulent flow, Journal of Computational Physics 182 (2002) 1–26.

[10] B. Tracey, K. Duraisamy, J. Alonso, Application of supervised learning to quantify uncertainties in turbulence and combustion modeling, AIAA Aerospace Sciences Meetingdoi:AIAA-2013-0259.

[11] B. Tracey, K. Duraisamy, J. Alonso, A machine learning strategy to assist turbulence model development, AIAA SciTechdoi:AIAA-2015-1287.

[12] K. Duraisamy, Z. Shang, A. Singh, New approaches in turbulence and transition modeling using data-driven techniques, AIAA SciTechdoi:AIAA-2015-1284.
[13] Z. Zhang, K. Duraisamy, Machine learning methods for data-driven turbulence modeling, AIAA Aviationdoi:AIAA-2015-2460.
[14] A. Elsheikh, R. Tavakoli, M. Wheeler, I. Hoteit, Boosting iterative stochastic ensemble method for nonlinear calibration of subsurface flow models, Computer Methods in Applied Mechanics and Engineering 259 (2013) 10–23.
[15] J. Ling, J. Templeton, Evaluation of machine learning algorithms for prediction of regions of high RANS uncertainty, Physics of Fluids (2015) 085103.
[16] J. Ghaboussi, J. G. Jr., X. Wu, Knowledge based modeling of material behavior with neural networks, Journal of Engineering Mechanics 117 (1991) 132–153.
[17] J. Ghaboussi, D. Sidarta, New Nested Adaptive Neural Networks (NANN) for Constitutive Modeling, Computers and Geotechnics 22 (1998) 29–52.
[18] G. Ellis, C. Yao, R. Zhao, D. Penumadu, Stress-strain modeling of sands using artificial neural networks, Journal of Geotechnical Engineering 121 (1995) 429–435.
[19] T. Furukawa, G. Yagawa, Implicit constitutive modelling for viscoplasticity using neural networks, International Journal for Numerical Methods in Engineering 43 (1998) 195–219.
[20] Y. Sun, W. Zeng, Y. Zhao, Y. Qi, X. Ma, Y. Han, Development of constitutive relationship model of Ti600 alloy using artificial neural network, Computational Materials Science 48 (2010) 686–691.
[21] S. Jung, J. Ghaboussi, Neural network constitutive model for rate-dependent materials, Computers and Structures 84 (2006) 955–963.
[22] X. Feng, C. Yang, Genetic evolution of nonlinear material constitutive models, Computer Methods in Applied Mechanics and Engineering 190 (2001) 5957–5973.
[23] Y. Shen, K. Chandrashekhara, W. Breig, L. Oliver, Finite element analysis of V-ribbed belts using neural network based hyperelastic material model, International Journal of Non-linear Mechanics 40 (2005) 875–890.
[24] G. Liang, K. Chandrashekhara, Neural network based constitutive model for elastomeric foams, Engineering Structures 30 (2008) 2002–2011.
[25] G. Piatetsky-Shapiro, Knowledge discovery in real databases: A report on the IJCAI-89 Workshop, AI magazine.
[26] R. Michalski, Understanding the nature of learning: Issues and research directions, Machine learning: An artificial intelligence approach 2 (1986) 3–25.
[27] E. Noether, Invariante variationsprobleme, Nachrichten von der Gesellschaft der Wissenschaften zu Gttingen, mathematisch-physikalische Klasse (1918) 235–257.
[28] t. A. Deriglazov.
[29] M. Lefik, B. Schrefler, Artificial neural network as an incremental non-linear constitutive model for a finite element code, Computer methods in applied mechanics and engineering 192 (2003) 3265–3283.
[30] H. Baird, Document image defect models, in: Structure Document Image Analysis, Springer, Berlin, 1992.
[31] P. Simard, B. Victorri, Y. L. Cun, J. Denker, Tangent prop–a formalism for specifying selected invariances in an adaptive network, Advances in neural information processing systems (1992) 895–903.
[32] D. Decoste, B. Scholkopf, Training invariant support vector machines, Machine Learning.
[33] S. Farfade, M. Saberian, L. Li, Multi-view face detection using deep convolutional neural networks, International Conference on Multimedia Retrieval.
[34] J. Wood, Invariant pattern recognition: A review, Pattern Recognition 29 (1996) 1–17.
[35] M. Hamermesh, Group Theory and Its Application to Physical Problems, Courier Corporation, 1989.
[36] R. Cahn, Semi-Simple Lie Algebras and Their Representations, Courier Corporation, 2014.
[37] J. Boehler, Applications of tensor functions in solid mechanics, Springer, Wien, 1987.
[38] D. Hilbert, B. Sturmfels, Theory of algebraic invariants, Cambridge University Press, Cambridge, 1993.
[39] R. Johnson, The Handbook of Fluid Dynamics, CRC Press, 1998.
[40] A. Spencer, R. Rivlin, Isotropic integrity bases for vectors and second-order tensors, Archive for Rational Mechanics and Analysis 9 (1962) 45–63.
[41] G. Smith, On isotropic integrity bases, Archive for rational mechanics and analysis 18 (1965) 282–292.
[42] A. Spencer, Isotropic polynomial invariants and tensor functions, in: Applications of tensor func-

tions in solid mechanics, Springer, Vienna, 1987.
[43] Q. Zheng, Theory of representations for tensor functions–a unified invariant approach to constitutive equations, Applied Mechanics Reviews 47 545–587.
[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[45] D. Steinberg, P. Colla, CART: classification and regression trees, in: The top ten algorithms in data mining, Chapman and Hall, Boca Raton, FL, 2009.
[46] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32.
[47] R. Banfield, L. Hall, K. Bowyer, D. Bhadoria, W. Kegelmeyer, S. Eschrich, A comparison of ensemble creation techniques, Multiple Classifier Systems (2004) 223–232.
[48] F. Provost, V. Kolluri, A survey of methods for scaling up inductive algorithms, Data Mining and Knowledge Discovery 2 (1999) 131–169.
[49] C. Bishop, Neural networks for pattern recognition, Oxford university press, Cambridge, UK, 1995.
[50] K. Gurney, An introduction to neural networks, CRC press, New York, 1997.
[51] M. Wojciechowski, Feed-forward neural network for python, Tech. rep., Technical University of Lodz, Lodz, Poland (2011).
[52] S. Gallant, Neural network learning and expert systems, MIT Press, Boston, 1993.
[53] L. Prechelt, Automatic early stopping using cross validation: quantifying the criteria, Neural Networks 11 (1998) 761–767.
[54] C.Gorle, J. Larsson, M. Emory, G. Iaccarino, The deviation from parallel shear flow as an indicator of linear eddy-viscosity model inaccuracy, Physics of Fluids 26.
[55] S. Pope, A more general effective-viscosity hypothesis, Journal of Fluid Mechanics 72 (1975) 331–340.
[56] T. Craft, B. Launder, K. Suga, Development and application of a cubic eddy-viscosity model of turbulence, International Journal of Heat and Fluid Flow 17 (1996) 108–115.
[57] K. Abe, Y.-J. Jang, M. Leschziner, An investigation of wall-anisotropy expressions and length-scale equations for non-linear eddy-viscosity models, International Journal of Heat and Fluid Flow 24 (2003) 181–198.
[58] C. Speziale, A consistency condition for non-linear algebraic Reynolds stress models in turbulence.
[59] R. Rossi, D. Philips, G. Iaccarino, A numerical study of scalar dispersion downstream of a wall-mounted cube using direct simulations and algebraic flux models, International Journal of Heat and Fluid Flow 31 (2010) 805–819.
[60] R. Rossi, G. Iaccarino, Numerical analysis and modeling of a plume meandering in passive scalar dispersion downstream of a wall-mounted cube, International Journal of Heat and Fluid Flow 43 (2013) 137–148.
[61] R. Rivlin, Non-linear continuum theories in mechanics and physics and their applications, Springer, Heidelberg, Germany, 2011.
[62] A. Spencer, Continuum Mechanics, Longman Group UK Limited, Essex, England, 1980.
[63] N. Kambouchev, J. Fernandez, R. Radovitzky, A polyconvex model for materials with cubic symmetry, Modelling and Simulation in Materials Science and Engineering 15.
[64] S. Foiles, M. Baskes, M. Daw, Embedded-atom-method functions for the fcc metals Cu, Ag, Au, Ni, Pd, Pt, and their alloys, Physical Review B 33 (12) (1986) 7983.