Incorporating Prior Domain Knowledge into Deep Neural Networks

Nikhil Muralidhar*‡°, Mohammad Raihanul Islam*‡°, Manish Marwah+,
Anuj Karpatne*‡, and Naren Ramakrishnan*‡
*Department of Computer Science, Virginia Tech, VA, USA
‡Discovery Analytics Center, Virginia Tech, USA
+Micro Focus, Sunnyvale, CA, USA
Email: {nik90, raihan8, karpatne, naren}@cs.vt.edu, manish.marwah@gmail.com
°(equal contribution)

***Abstract*—In recent years, the large amount of labeled data available has also helped tend research toward using minimal domain knowledge, e.g., in deep neural network research. However, in many situations, data is limited and of poor quality. Can domain knowledge be useful in such a setting? In this paper, we propose domain adapted neural networks (DANN) to explore how domain knowledge can be integrated into model training for deep networks. In particular, we incorporate loss terms for knowledge available as monotonicity constraints and approximation constraints. We evaluate our model on both synthetic data generated using the popular Bohachevsky function and a real-world dataset for predicting oxygen solubility in water. In both situations, we find that our *DANN* model outperforms its domain-agnostic counterpart yielding an overall mean performance improvement of 19.5% with a worst- and best-case performance improvement of 4% and 42.7%, respectively.**

***Keywords*-Noisy Data; Domain Knowledge; Neural Networks; Deep Learning; Limited Training Data;**

I. Introduction

Deep learning has witnessed tremendous success in recent years in areas such as computer vision [1], natural language understanding [2], and game playing [3]. In each of these areas, considerable improvements have been made in tasks such as image recognition [4], machine translation [5], [6], and in games such as *Go* where top human players have been roundly defeated [7].

A common philosophy behind these machine learning successes has been use of end-to-end models with minimally processed input features and minimal use of domain or innate knowledge[1], so as not to introduce user bias into the system; and, instead let the models learn mostly from data, in contrast to the past where domain knowledge played a central role in engineering model features.

There is an ongoing debate [8] on how much domain knowledge is necessary for efficient learning. At one extreme is "blank slate" (or *tabula rasa*) learning where no domain knowledge is assumed *a priori* and everything is induced from data, including model structure and hyperparameters.

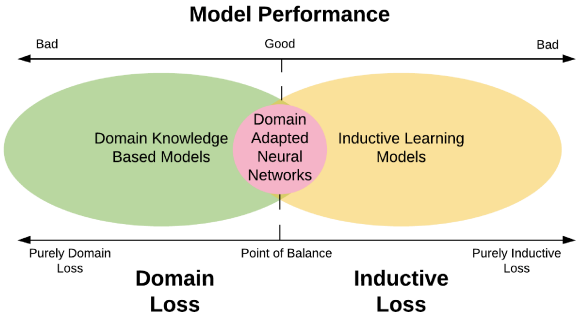


Figure 1: **Advantages of hybrid models like domain adapted neural networks (*DANN*) as opposed to using purely inductive or purely domain based models.**

At the other end is the approach where everything is manually hard-wired based on domain expertise with little help from data.

While researchers agree that these extremes lead to poor models, it is unclear where the sweet spot lies. In deep learning, domain knowledge often contributes to selection of network architecture. The most successful example of this idea pertains to the use of convolutional neural networks for tasks involving images or videos, because images exhibit translational invariance. Similarly, recurrent neural networks are preferred for data with sequential structure. However, in these situations, large amounts of training data are available. What about cases where data may be limited or sparse[2] (i.e. limited training data that is not fully representative of the entire data distribution) and of poor quality? In fact, while in general data has become abundant in recent years, there are several applications where sufficient and representative data is hard to come by for building machine learning models, e.g., in modeling of physical processes in critical infrastructure such as power plants or nuclear reactors. There are several impediments in collecting data from such systems: 1) **limited data**: data available is limited in terms

[1]We use the terms domain knowledge or innate knowledge interchangeably here to refer to anything not learned using data.

[2]Note that we use the terms limited data and sparse data interchangeably here.

of feature coverage since these systems typically run in an operationally optimized setting and to collect data outside this narrow range is usually expensive or even unsafe, if at all possible; 2) **expensive data**: in some instances, for example manufacturing facilities, collection of data may be disruptive or require destructive measurements; 3) **poor quality data**: quality of data collected from physical infrastructure systems is usually poor (e.g., missing, corrupted, or noisy data) since they typically have old and legacy components.

We posit that in these situations, model performance can be significantly improved by integrating domain knowledge, which might readily be available for these physical processes in the form of physical models, constraints, dependencies relationships, and knowledge of valid ranges of features. In particular, we ask:

1) When data is limited or noisy, can model performance be improved by incorporation of domain knowledge?
2) When data is expensive, can satisfactory model performance be achieved with reduced data sizes through incorporation of domain knowledge?

To address these questions, in this paper, we propose DANN (domain adapted neural networks), where domain-based constraints are integrated into the training process. As shown in Fig. 1, DANN attempts to find a balance between inductive loss and domain loss. Specifically, we address the problem of incorporating monotonic relationships between process variables (*monotonicity constraints* [9]) as well as incorporating knowledge relating to the normal quantitative range of operation of process variables (*approximation constraints* [9]). We also study the change in model performance when multiple domain constraints are incorporated into the learning model. In each case, we show that our proposed domain adapted neural network model is able to achieve significant performance improvements over domain agnostic models.

Our main contributions are as follows:

1) We propose DANN which augments the methodology in [10] to incorporate both monotonicity constraints and approximation constraints in the training of deep neural networks.
2) We conduct a rigorous analysis by characterizing the performance of domain based models with increasing data corruption and decreasing training data size on synthetic and real data sets.
3) Finally, we also showcase the effect of incorporating multiple domain constraints into the training process of a single learning model.

II. Related Work

In recent times, with the permeation of machine learning into various physical sciences, there has been an increasing attempt to leverage the power of learning models to augment, simplify experimentation and otherwise replace costly simulations in these fields. However, owing to the underlying complexity of the function space and the corresponding lack of representative datasets, there have been a number of attempts at incorporating already existing domain knowledge about a system into a machine learning framework or to overcome drawbacks of existing simulation frameworks using mahcine learning models. In [11], the authors utilize a stacked generalization approach to incorporate domain knowledge into a logistic regression classifier for predicting 30 day hospital readmission. In [12], the authors utilize random forests for reconstructing discrepancies in a Reynolds-Averaged Navier-Stokes system (RANS) for modeling industrial fluid flows. It is a well known problem that the predictive capabilities of RANS models exhibit large discrepancies. Wang et al. try to reconstruct these discrepancies through generalization of machine learning models in contexts where data is not available. There have also been efforts to utilize machine learning techniques to quantify and reduce model-form uncertainty in decisions made by physics driven simulation models. In [13], [14] the authors achieve this goal using a Bayesian network modeling approach incorporating physics-based priors. From a Bayesian perspective, our approach to integrating domain knowledge into the loss function is equivalent to adding it as a prior.

In addition to incorporating domain knowledge, there have also been attempts to develop models that are capable of performing more fundamental operations like sequential number counting, and other related tasks which require the system to generalize beyond the data presented during the training phase. Trask et al. [15] propose a new deep learning computational unit called the Neural Arithmetic Logic Unit (NALU) which is designed to perform arithmetic operations like addition, subtraction, multiplication, division, exponentiation etc. and posit that NALUs help vastly improve the generalization capabilities of deep learning models. Another related research work is the paper by Arabshahi et al. [16] in which the authors employ black-box function evaluations and incorporate domain knowledge through symbolic expressions that define relationships between the given functions using tree LSTMs. Bongard et al. [17] propose the inverse problem of uncovering domain knowledge given time-series data in a framework for automatically reverse-engineering the functioning of a system. Their model learns domain rules through the intrusive approach of intelligently perturbing the operation of a system and analyzing the resulting consequences. In addition, they assume that all the data variables are available for observation which is quite often not the case in many machine learning and physical system settings.

Mustafa in [9] proposes a framework for learning from hints in inductive learning systems. The proposed framework incorporates different types of hints using a data assimilation process wherein data is generated in accordance with a

particular domain rule and fed into a machine learning model as an extension of the normal training process. Each such domain based data point is considered one of the hints that guides the model toward more domain amenable solutions. Generating data that is truly representative of a particular piece of innate knowledge without overtly biasing the model is costly and non-trivial. Also, as stated in [9], direct implementation of hints in the learning process is much more beneficial than are methods of incorporating domain knowledge through data assimilation. Hence, we develop methods wherein innate knowledge about a system is directly incorporated into the learning process and not through external costly means like data assimilation. We show that incorporating domain constraints directly into the loss function can be used to greatly improve model quality of a learning algorithm like a deep neural network (NN) even if it is trained using a sparse, noisy dataset that is not completely representative of the spectrum of operational characteristics of a system.

Research closest to ours has been conducted by Karpatne et al. [10]. Here, the authors propose a physics guided neural network model for modeling lake temperature. They utilize the increasing monotonic relationship of water density measurements with increasing depth as the physical domain knowledge that is incorporated into the loss function. They predict the density of water in a lake at different depths and utilize the predicted densities to calculate the corresponding water temperature at those depths using a well established physical relationship between water temperature and density. However, they incorporate only a single type of domain knowledge (i.e., monotonic relationships). In this work, we have augmented the approach in [10] to model other types of domain rules and characterize model behavior in many challenging circumstances (to be detailed in later sections).

III. Problem Formulation and Solution Approach

Problem Statement: Leverage domain knowledge to train a robust, accurate learning model that yields good model performance even with sparse, noisy training data.

Innate knowledge about the functioning of a system S may be available in several forms. One of the most common forms of knowledge is a quantitative range of normal operation for a particular process variable Y in S. Another type of domain knowledge could be incorporating monotonically increasing or decreasing relationships between different process variables or measurements of the same process variable taken in different contexts. To incorporate these domain based constraints into the inductive learning process, we develop **domain adapted neural networks** (*DANN*).

We select deep neural network models as the inductive learner owing to their ability to model complex relationships and adopt the framework proposed in [10] for incorporating domain knowledge in the training of deep neural network models.

The generic hybrid loss function of the deep learning model is depicted in Eqn. 1. Here, $\text{Loss}(Y, \hat{Y})$ is a mean squared error loss used in many inductive learning applications for regression and Y, $\hat{Y}$ are the ground-truth and predicted values, respectively, of the target system variable. $R(f)$ is an L_2 regularization term used to control model complexity of the model f. The $\text{Loss}_D(\hat{Y})$ term is the domain loss directly incorporated into the neural network loss function used to enforce that the model learned from training data is also in accordance with certain accepted domain rules.

$$\underset{f}{\text{argmin}} \ \text{Loss}(Y, \hat{Y}) + \lambda_D \text{Loss}_D(\hat{Y}) + \lambda R(f) \tag{1}$$

Here λ_D is a hyper-parameter determining the weight of domain loss in the objective function. We chose the value of λ_D empirically (see Fig. 3). λ is another hyper-parameter determining the weight of the regularizer. We model two types of constraints: 1) Approximation Constraints; and, 2) Monotonicity Constraints.

A. Approximation Constraints

Noisy measurements quite often cause significant deviation in model quality. In such cases, the insights domain experts possess about reasonable ranges of normal operation of the target variable could help in training higher quality models. We wish to incorporate these *approximation constraints* during model training, to produce more robust models.

Such constraints may be specified as a quantitative range of operation of the target variable Y. Let (y_l, y_u) be the range of normal operation of a particular target variable $Y \in R^{m \times 1}$, i.e., $Y \in [y_l, y_u]$ (y_l, y_u can be provided by a domain expert or estimated empirically). Then, $g(\hat{Y})$ in Eqn. 2 represents the functional form of the approximation constraint while Eqn. 3 depicts how we incorporate $g(\hat{Y})$ directly into the training loss function of a deep feed-forward neural network.

$$g(\hat{Y}) = \begin{cases} 0 & \text{if } \hat{Y} \in [y_l, y_u] \\ |y_l - \hat{Y}| & \text{if } \hat{Y} < y_l \\ |y_u - \hat{Y}| & \text{if } \hat{Y} > y_u \end{cases} \tag{2}$$

$$\text{Loss}_D(\hat{Y}) = \sum_{i=1}^{m} ReLU(y_l - y^i) + ReLU(y^i - y_u) \tag{3}$$

$$ReLU(z) = z^+ = max(0, z) \tag{4}$$

A $ReLU$ term is appropriate here as its output is non-zero when the input is positive and thus suitable for modeling the constraints.

B. Monotonicity Constraint

Physical, chemical, and biological processes quite often have facets which are related monotonically. Let x_1, x_2 represent

measurements of a single phenomenon in different contexts in a system (e.g., x_1, x_2 could be pressure at different heights, air temperature at different times of the day). If we consider a function $h(x) = y$ such that $x_1 > x_2 \Rightarrow h(x_1) > h(x_2)$, then x_1, x_2 and $h(x_1)$, $h(x_2)$ are said to share a monotonic relationship. We can incorporate such *monotonicity constraints* using the formulation represented in Eqn. 5. Here, $\text{Loss}_D(\hat{Y_1}, \hat{Y_2})$ represents the domain loss calculated by enforcing the monotonicity constraint $\hat{Y_1} < \hat{Y_2}$.

In Eqn. 5, $\mathbb{I}(\cdot)$ represents the identity function which evaluates to true if the result of the logical AND ($\wedge$) operation evaluates to true and is false otherwise. The identity function essentially serves to produce a boolean mask of cases where measurements obey the monotonicity constraint being enforced while the predictions by the neural network model violate the constraint. Applying this mask to the ReLU function (described in Eqn. 4) allows us to capture errors only of the instances wherein the domain constraint is violated. Formulating the domain loss $\text{Loss}_D(\cdot)$ in this manner causes the model to change course to a region in the (learned) function space more amenable to the injected domain constraint.

$$\text{Loss}_D(\hat{Y_1}, \hat{Y_2}) = \sum_{i=1}^{m} \mathbb{I}\Big((x_1^i < x_2^i) \wedge (\hat{y}_1^i > \hat{y}_2^i)\Big) \cdot ReLU(\hat{y}_1^i - \hat{y}_2^i) \quad (5)$$

IV. Dataset Description

A. Synthetic Datasets

We use the popular Bohachevsky function as the basis for generating synthetic datasets to evaluate the effectiveness of incorporating domain knowledge in our experiments. A Bohachevsky function is typically given by an expression similar to Eqn. 6. In our experiments we use a variant with positive amplitudes for the cosine functions i.e $a_1 = 0.3$, $a_2 = 0.4$. Similarly, we set $p_1 = 3$, $p_2 = 4$ and $k_1 = 1$, $k_2 = 2, K = 0.7$.

$$f(x_1, x_2) = k_1 x_1^2 + k_2 x_2^2 + a_1 \cos(p_1 \pi x_1) + a_2 \cos(p_2 \pi x_2) + K \quad (6)$$

The values of x_1, x_2 are randomly sampled positive values from a normal distribution. We sample m values each of x_1 and x_2 to form our input data vector $X \in \mathbb{R}^{m \times 2}$. For each row $\mathbf{x}_i \in \mathbb{R}^{1 \times 2}$ in X, we generate the corresponding target value y_i using Eqn. 6 to form our target vector $Y \in \mathbb{R}^{m \times 1}$. The dataset X, Y is used for experiments involving *approximation constraints*.

In order to conduct experiments to test the effectiveness of incorporating *monotonicity constraints*, we create two more synthetic datasets X', X'' such that each $x'_{i,1} = 6x_{i,1}$, $x''_{i,1} = 12x_{i,1}$. Hence, for the monotonicity constraint experiments, we generate three datasets X, X', X'' such that $x_{i,1} < x'_{i,1} < x''_{i,1}$ and the outputs calculated for X, X', X'' using Eqn. 6 are Y, Y', Y'' respectively with $y_i < y'_i < y''_i$.

B. Real Datasets

We also demonstrate the performance of our models on a real-world application for prediction of oxygen solubility in water. The solubility of oxygen in water is primarily governed by three factors, the water temperature, salinity and pressure. We obtained temperature (t), salinity (s) and pressure (p) samples for the North Atlantic and Iceland Basin *Biofloat 48*[3]. This data is then used to calculate the amount of dissolved O_2 ($f(p, s, t)$) using the physical relationship detailed in Eqn. 7. We also compute the amount of O_2 solubility by increasing the pressure by 5.0 decibar and 10.0 decibar while keeping the temperature and salinity levels the same, thus once again obtaining three datasets X, X', X'' and $X(p) < X'(p) < X''(p)$.

$$\begin{aligned} f_1 &= \alpha_1 + \alpha_2\,(100/t) + \alpha_3\, ln(t/100) - \alpha_4\,(t/100) \\ f_2 &= f_1 + s(-\alpha_5 + \alpha_6\,(t/100) - \alpha_7\,((t/100)^2)) \\ f(p, s, t) &= e^{f_2}(p/100) \end{aligned} \quad (7)$$

Here α_1–α_7 are the constant terms. These terms are defined by researchers who measured the O_2 solubility by empirical evaluation.

V. Experimental Findings

Objective: We test our *DANN* framework on *Monotonicity* and *Approximation* constraints by trying to answer two questions:

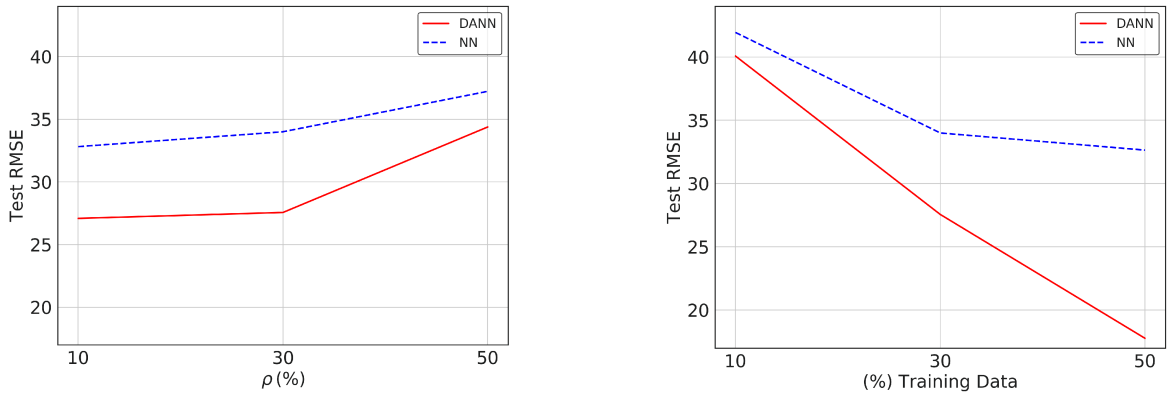
1) How well does *DANN* perform when available training data is noisy?
2) Can *DANN* perform well even if it is trained with limited training data?

A. Performance With approximation constraints in sparse and noisy contexts.

Experimental Setup: For the purposes of this experiment, we consider the dataset $X \in \mathbb{R}^{m \times 2}$ as defined in section IV-A. Each row $\mathbf{x}_i$ of X can be denoted as $\mathbf{x}_i = [x_{i,1}, x_{i,2}]$. The values of x_1 and x_2 in each row are then used to calculate the corresponding output function value $f(x_1, x_2)$ as described in Eqn. 6 to yield $Y \in R^{m \times 1}$. It must be noted that for the purposes of this experiment, x_1 and x_2 are randomly sampled and $x_1 \in \mathcal{N}(5, 1)$, $x_2 \in \mathcal{N}(20, 1)$. The location and scale for random sampling were chosen arbitrarily, ensuring only that x_1 and x_2 distributions were distinct.

Imputing Noise: We randomly select a subset of rows in X and interchange the values of x_1 and x_2 in those rows leading to the calculated value of $f(x_1, x_2)$ in Y for those rows being outside an expert-determined approximate normal range. This is done to intentionally corrupt a subset

[3]https://www.bco-dmo.org/dataset/3426



(a) **Approximation Constraint - Noise Percentage vs. RMSE** (b) **Approximation Constraint - Train Percentage vs. RMSE**

Figure 2: **Comparison between *DANN* and *NN* in noisy validation experiments indicates that the *DANN* model significantly outperforms the vanilla *NN* model both in the case of varying the amount of training data available as well as the noise percentage of the training data. In Fig. 2a, we varied the noise percentage (ρ) from 10% to 50% while keeping the percentage of training data used constant at 30%. In Fig. 2b, we similarly varied the training data percentage while keeping the noise percentage constant at ρ = 30%.**

of the dataset where the imposed domain constraint is violated. Let us term the percentage of rows sampled for such corruption as ρ (a.k.a. noise percentage). We vary ρ from 10% –50% of the total size (m rows) of the dataset X in order to evaluate the performance of *DANN* augmented with an approximation constraint on increasingly noisy data. At the end of this noise imputation stage, we have $X \in \mathbb{R}^{m\times 2}, Y \in \mathbb{R}^{m\times 2}$ where ρ% of rows in Y are outside the expert suggested approximation constraint i.e ρ% of rows in Y violate the constraint.

Approximation Constraint Injection: In order to inject the approximation constraint detailed in Eqn. 2 and Eqn. 3 into *DANN*, it is first required for a domain expert to provide a range of normal operation of the prediction variable i.e Y in our case. For this, we considered a subset of clean data from Y before corruption and evaluated the mean (μ) and standard deviation (σ). The range of normal operation i.e. the approximation constraint was chosen as $[\mu - \sigma, \mu + \sigma]$.

Model Architecture: We compare *DANN* with a standard feed-forward *NN*. In order to ensure fair comparison, both networks employ the same network architecture with two hidden layers where the first layer has a size of 64 units and the second layer has a size of 128 units. We apply an l_2 regularization on both models to curtail model complexity. Both models were trained for a standard 200 epochs with a learning rate of 0.001. The difference between the *DANN* model and the *NN* model is that the *DANN* model incorporates the approximation constraint directly into the loss function during training while the *NN* model does not.

Evaluation Strategy: We employ a 60-20-20 split for training, validation, and testing respectively. We use the RMSE score which calculates the prediction error using predictions

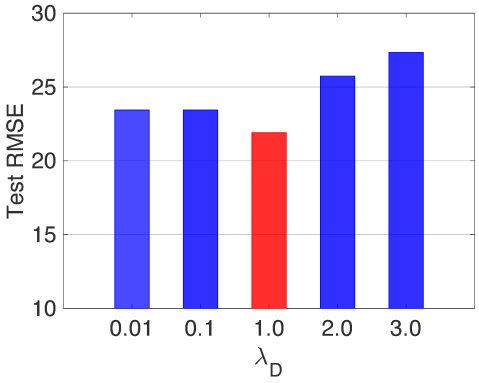


Figure 3: **RMSE score for different values of λ_D (lower the better). Results shown are for *noise-free* model selection. We infer from the figure that $\lambda_D = 1.0$ results in the best performance. Therefore, we set $\lambda_D = 1.0$ for *DANN*.**

$\hat{Y}$ and target values Y to evaluate model performance. The model is trained for the aforementioned 200 epochs and the best model is then selected based on the performance on the validation set. We then compute the performance of the selected model on the test set and report it. The test set is always noise-free. Noise-free in this context implies that none of the Y values in the test set violate the approximation constraint.

Evaluation on two Validation Sets: The quality of the validation set influences the model selection significantly. Hence, two types of validation experiments have been performed, i.e., we validate the model on a validation set with noisy data (i.e. corrupted data) and separately on a validation

set with noise-free data.

- **Noisy Model Selection:** Model selection is done using a noisy validation set.
- **Noise-Free Model Selection:** Model selection is done using a noisy-free validation set

Discussion of Results: We evaluated our DANN model using approximation constraints and the results are depicted in Fig. 2a and 2b. Fig. 2a shows the comparative performance of the DANN model relative to the domain knowledge agnostic neural network (NN) model at different levels of data corruption (ρ). We vary the percentage of data corruption from 10%–50% and observe that the DANN model outperforms the NN model significantly in all cases. We observe an expected upward trend of both models with increasing ρ with a significant increase at ρ= 30%. The *DANN* model shows a mean percentage improvement of 14.67% over the domain agnostic *NN* model. We see that even at extreme noise levels close to ρ = 50%, the *DANN* model significantly outperforms the *NN* model.

We also study the characteristics of the *DANN* and *NN* models in sparse data settings by varying the available data used to train the models from 10%–50% and observe once again that *DANN* comfortably outperforms the domain-agnostic *NN* model with a mean percentage improvement of 22.98%. In fact, we observe in Fig. 2b that the *DANN* model is able to incorporate more training data and improve its performance indicating that there exists no strong bias of the applied domain constraint preventing the model from assimilating useful inductive signals if any, as more data becomes available.

Frequently, in noisy and sparse settings, there is a minimum acceptable error threshold that is desired and a method that learns a stronger representation of the function space in such contexts is highly valuable. For example, if an RMSE value of 25.0 was desired, from Fig. 2b, we observe that it is possible to achieve with just about 40% of available training data using the *DANN* model while the *NN* model never breaches this error threshold.

Hyperparameter Selection: One of the primary concerns in the context of the *DANN* model is how the hyperparameter λ_D which controls the influence of the domain constraint is set. We set the value of λ_D empirically. We plot the RMSE score for several values of λ_D in Fig. 3. As $\lambda_D = 1.0$ rewards us with the best performance, we set the value $\lambda_D = 1.0$ in all our experiments. We also set the value of $\lambda = 1.0$ for this experiment.

B. Performance with monotonicity constraints in sparse and noisy contexts.

We also evaluate the performance of our *DANN* model augmented with *monotonicity constraints* on the synthetic dataset described in section IV-A as well as a real-world dataset described in section IV-B. We first discuss the experimental setup and results for the synthetic data case and then proceed with the experimental setup and discussion of results for the real world application.

Experimental Setup - Synthetic Data

Bohachevsky Function Value Prediction: For this experiment, we consider the datasets X, X', X'' which are as described in section IV-A. Each row i of X, X', X'' has the monotonic relationship $x_{i,1} < x'_{i,1}, < x''_{i,1}$ in tact and consequently $y_i < y'_i < y''_i$ where each of y_i, y'_i, y''_i corresponds to the i^{th} target value of Y, Y', Y'' respectively.

Imputing Noise: Let us first consider Y, Y'. We know that there exists a monotonic relationship of the form $y_{i,1} < y'_{i,1}$ for each instance of the two target datasets. We randomly sample a subset of rows and switch the values of y and y' in those rows in order to create the effect of noisy data where expected monotonicity constraints are violated. Let us call the percentage of randomly sampled indices ρ. For the purposes of our experiment, we vary the value of ρ between 10%–50% to evaluate the performance of the *DANN* model on increasingly noisy datasets. We repeat the same process of randomly exchanging values of a subset of rows for Y' and Y'' in order to violate the monotonicity constraint $y'_i < y''_i$. The two sets of randomly sampled indices are disjoint.

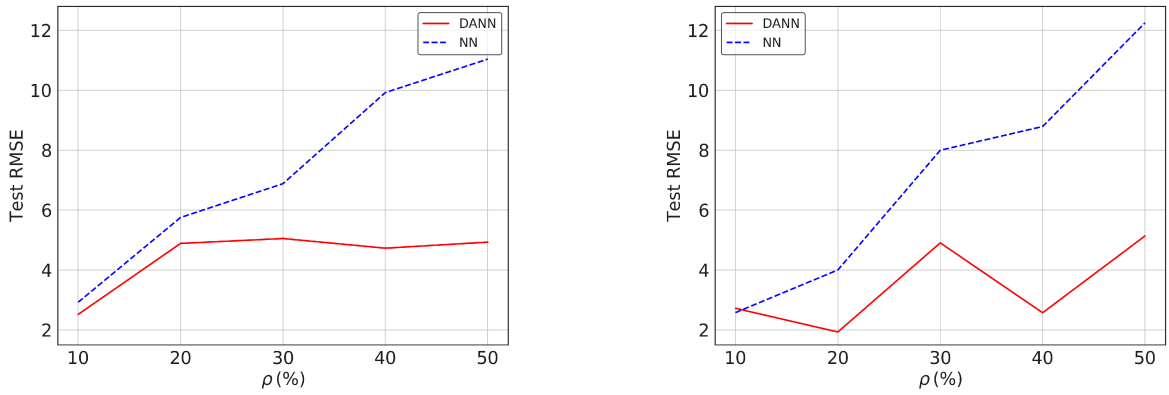
Domain Knowledge Injection: The domain knowledge being injected in this case checks for consistency in monotonicity between input measurements and model predictions i.e for a particular input instance i, the constraints enforced are $x_{i,1} < x'_{i,1} \Rightarrow \hat{y}_i < \hat{y}'_i$ & $x'_{i,1} < x''_{i,1} \Rightarrow \hat{y}'_i < \hat{y}''_i$

Model Architecture: We once again compare *DANN* with a vanilla neural network (*NN*) model. *DANN* and *NN* both employ an architecture with two hidden layers where the first hidden layer has a size of 512 units and 256 units is the size of the second hidden layer. The rest of the architecture and model design is similar to the architecture described in the *Model Architecture* part of section V-A.

Evaluation Strategy: We once again employ a 60-20-20 split for training, validation and testing respectively. The RMSE score of the predictions $\hat{Y}, \hat{Y}', \hat{Y}''$ compared against the true values Y, Y', Y'' respectively is used for evaluating model performance. Both *NN* and *DANN* are trained for 200 epochs and the best *NN* and *DANN* model are selected based on the performance on the validation set. We then calculate the performance of the selected model on the test set and report results. The test and validation experiments are organized similar to those described in the *Evaluation Strategy* of section V-A.

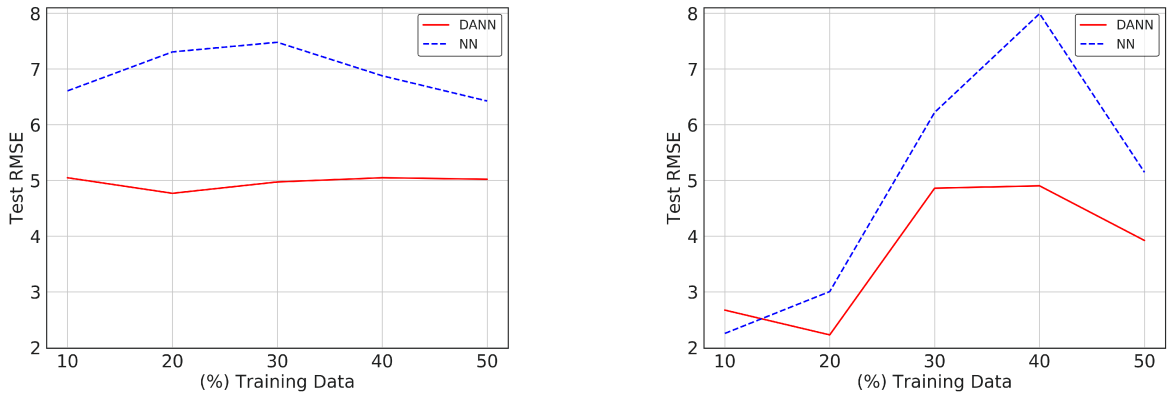
Discussion of Results: We run two sets of experiments, one wherein we evaluate the model performance of *NN* and *DANN* models with increasingly noisy training data and the other where we evaluate the model performance by continuously decreasing the amount of training data available for the *NN* and *DANN* models to be trained on, to simulate sparse data settings.

Fig. 4 shows the prediction performance comparison of *NN*



(a) **Monotonicity Constraint - Noisy Validation Set.** (b) **Monotonicity Constraint - Noise-free Validation Set.**

Figure 4: **Comparison between *DANN* and *NN* in both the noisy and noise-free validation experiments indicates that the *DANN* model significantly outperforms the vanilla *NN* model as far as robustness to noisy training data is concerned. We observe that even when $\rho = 50\%$ and with a noisy validation set, the *DANN* model is still able to outperform the *NN* model. We used 40% of the available training data to maintain a relatively sparse dataset.**



(a) **Monotonicity Constraint - Noisy Validation Set.** (b) **Monotonicity Constraint - Noise-free validation Set.**

Figure 5: **Comparison between *DANN* and *NN* in noisy validation set for Bohachevsky function value prediction. The noise-level used for the experiments is $\rho = 30\%$. In both the case of the noisy and noise free validation sets, we see that *DANN* significantly outperforms *NN*.**

and *DANN* with increasing noise percentage ρ, using noisy and noise free validation sets during model training. We observe that in Fig. 4a, both models show a rising trend with increasing amount of noise while this trend is very noticeable in the case of the domain agnostic *NN* model, the deterioration of model performance of the *DANN* model is relatively gradual almost plateauing after about $\rho = 30\%$. As a result of this, the performance improvement of the *DANN* model over the *NN* model increases significantly with increase in noise percentage with *DANN* showing an average performance improvement of 32.65% over the vanilla *NN* model. A similar rising trend can be observed in the *NN* model in Fig. 4b wherein a noise-free validation set was used during training. Once again, the *DANN* model significantly outperforms the domain agnostic *NN* model, this time with an average performance improvement of 42.7%.

Fig. 5a showcases the performance of the domain adapted and agnostic models in sparse data settings trained using a noisy validation set. Here, we observe that the domain adapted model *DANN* is able to learn a significantly better representation of the target function space relative to the *NN* model using just about 20% of the training data. The model performance of the *DANN* model stays almost constant (after about 30% training data) with increasing training data percentage indicating the ability of the domain aware *DANN* model to effectively learn accurate representations even using very little training data. The *DANN* registers a mean performance improvement of 28.06%. Fig. 5b depicts that once again the *DANN* model outperforms the *NN* model with increasing training data in most cases except for the

10% training data case, with a mean average performance improvement of 18.32%.

Experimental Setup - Real Data:

O_2 **solubility prediction:** The solubility of O_2 generally depends on three factors. The first two are the temperature of the water and its salinity level. It has been observed that O_2 solubility in water is inversely proportional to both temperature and salinity [18]. The other factor is pressure. Higher pressure leads to increased O_2 solubility [19].

Dataset generation by changing one factor: We use samples of temperature, salinity and pressure taken from North Atlantic and Iceland Basin *Biofloat 48*[4]. This dataset contains the temperature, salinity and pressure levels for a period of time. We calculate the amount of O_2 dissolved in the water using Eqn. 7 given the above parameters. Apart from this computation, we also compute the amount of dissolved O_2 by raising the pressure level by 5.0 and 10.0 decibar while keeping the temperature, salinity level same as defined in the dataset. In this way, we obtain three datasets Y, Y', and Y'' of different O_2 levels.

Dataset generation by changing multiple factors: To conduct a separate experiment to test multiple domain constraints applied simultaneously, we generate another dataset by changing temperature and salinity level in addition to pressure. Specifically, we increase the temperature by 5° and 10° Celsius and the salinity level by 5 and 10 units to obtain Y, Y', and Y''.

Impute Noise: Noise imputation has been carried out on Y, Y', Y'' datasets in a similar manner by exchanging values between the datasets as indicated in the *Impute Noise* segment of section V-B. Through such noise imputation, we achieve a set of noisy datasets with explicit violation of the physical relationships.

Model Architecture: The architecture of both the *DANN* and *NN* models is similar to that used for the synthetic datasets as described earlier in section V-B. The only difference is that λ is set to 0.01 (recall that it is set to 1.0 for synthetic datasets). However λ_D remains constant (i.e. 1.0)

Evaluation Strategy: We once again employ the same 60-20-20 split for train,validation and test sets respectively. We compute the RMSE score by comparing the predicted O_2 solubility $\hat{Y}, \hat{Y'}, \hat{Y''}$ with the target values Y, Y', Y'' respectively and use this score for evaluating model performance. We run the model for 300 epochs and select the best model based its performance on the validation set. We then compute the performance of the selected model on the test set report results. The test and validation experiments are organized similar to those described in the *Evaluation Strategy* of section V-A.

Discussion of Results: We run two types of empirical evaluations to evaluate the performance of *DANN*. In the first experiment we evaluate how *DANN* behaves in the presence of increasing noisy data. The result in shown in Fig. 6. Our observation is *DANN* outperforms the vanilla *NN* in both types of evaluation. The improvement is generally higher when ρ is larger. For example when $\rho = 50\%$, the gain is 17% for *noise free model* selection, whereas it is only 8% when $\rho = 10\%$. We observe similar phenomena in *noisy model* selection. Here gain in 9% when $\rho = 50\%$ while it is 2% for $\rho = 10\%$. Another observation is overall RMSE is lower for *noise-free* model selection. Since the test data remains noise free for both cases, the model based on noise-free validation set provides better understanding of the test data rather than the model trained on noisy validation set. From this experiment we can conclude that *DANN* leverages the domain constraint so that it can estimate the output better than other model.

To evaluate how *DANN* performs when limited training data is available, we train the model with fewer number of instances (10%–50%) and compute the test RMSE same as before. The result is shown in Fig. 7. We can see that here also *DANN* outshines *NN* model by a good margin. For both models RMSE score is the highest when the size of the train data is lowest i.e. 10%. RMSE improves as more training data is available.

Finally we test the efficacy of *DANN* using the dataset where multiple factors related to O_2 solubility are changed. Since here more than one variable is changed we can also include more than one constraint in the objective function. The result for *noisy model selection* where $\rho = 20\%$ is shown in Fig: 8. Here *DANN*-One leverages just one domain constraint i.e. pressure. Similarly *DANN*-Two uses two domain constraints i.e. (pressure + temperature). Finally *DANN*-Three or *DANN*-All uses all three domain constraints (i.e. pressure + temperature + salinity). We observe that *DANN* models i.e. *DANN*-One, *DANN*-Two and *DANN*-Three easily outperform *NN*. Moreover, *DANN*-Three achieves the best result as it exploits all three domain constraints.

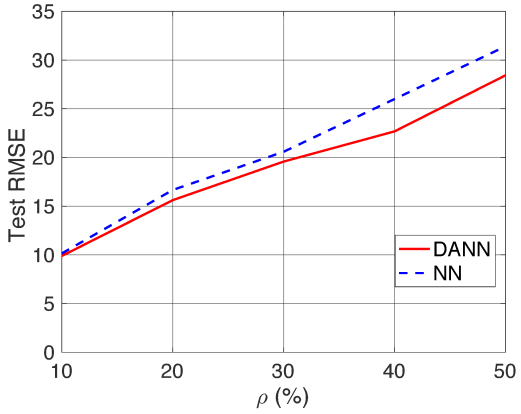
C. Summary of Results

The results depicted in section V-A and section V-B showcase that the domain-aware *DANN* model significantly outperforms the domain-agnostic *NN* model. A summary of our experimental results depicting mean performance improvement of *DANN* over *NN* is shown in Table I.

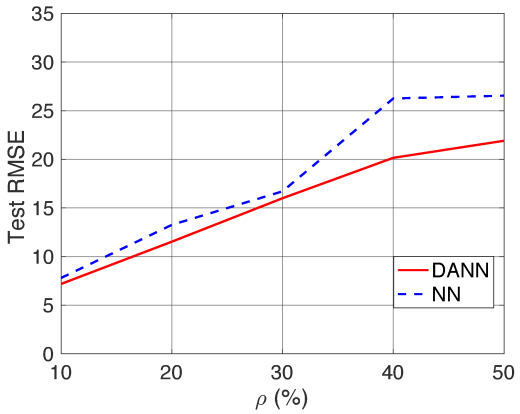
How does *DANN* perform when the available training data is noisy?

DANN models trained with approximation and monotonicity constraints, perform well even in extremely noisy settings highlighting the robustness that domain based constraints bring to inductive learning models. The superior performance of *DANN* relative to the *NN* model tells us that they are able to extract relevant signals from noisy data and are able to model the underlying function space in an accurate manner. The *DANN* model shows a mean performance

[4]https://www.bco-dmo.org/dataset/3426

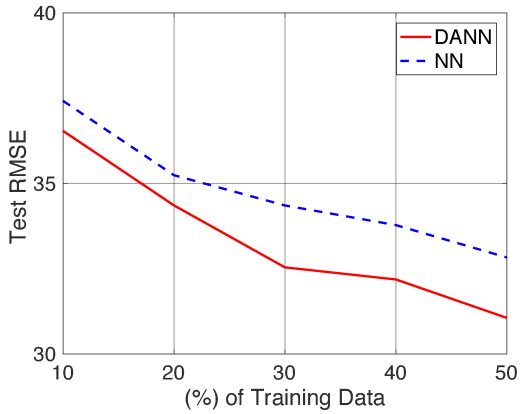


(a) **Monotonicity Constraint - Noisy Validation Set.**

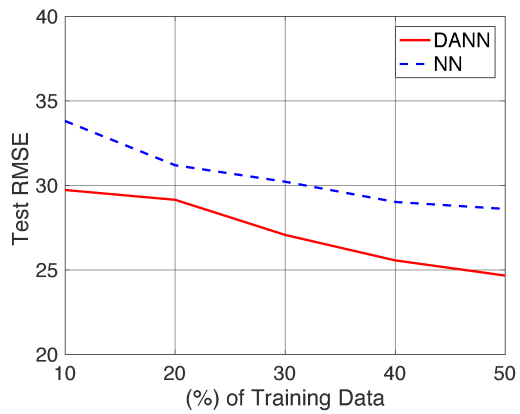


(b) **Monotonicity Constraint - Noise Free Validation Set.**

Figure 6: **Comparison between *DANN* and *NN* in noisy and noise-free validation experiments for O_2 solubility prediction (lower the better). *DANN* outperforms *NN* in both cases, especially when ρ is higher.**



(a) **Monotonicitiy Constraint- Noisy Validation Set.**



(b) **Monotonicity Constraint - Noise-Free Validation Set.**

Figure 7: **Comparison of *DANN* and *NN* with reduced training data (lower the better). We use the dataset with $\rho = 50\%$. We can see that *DANN* clearly outperforms the *NN* model.**

Table I: **Summary of Results.**

Type of Constraint	Dataset	Validation Set	Experiment Type	Mean % Improvement (*DANN* over *NN*)
Approximation	Bohachevsky	Noisy	ρ vs. RMSE	14.67%
Approximation	Bohachevsky	Noisy	(%)Training Data vs. RMSE	22.98%
Monotonicity	Bohachevsky	Noisy	ρ vs. RMSE	32.65%
Monotonicity	Bohachevsky	Noise-Free	ρ vs. RMSE	42.70%
Monotonicity	Bohachevsky	Noisy	(%)Training Data vs. RMSE	28.06%
Monotonicity	Bohachevsky	Noise-Free	(%)Training Data vs. RMSE	18.32%
Monotonicity	O_2 Sol.	Noisy	ρ vs. RMSE	7.19%
Monotonicity	O_2 Sol.	Noise-Free	ρ vs. RMSE	13.18%
Monotonicity	O_2 Sol.	Noisy	(%)Training Data vs. RMSE	4.05%
Monotonicity	O_2 Sol.	Noise-Free	(%)Training Data vs. RMSE	10.94%

improvement of 22.08% over the *NN* model across all the experiments performed with noisy training data.

Can *DANN* perform well even with limited training data?

In the case of the approximation constraint, we can observe a clear trend of the *DANN* model being able to assimilate relevant signals even from noisy training data as more data becomes available. In both the case of the monotonicity constraint and the approximation constraint, the *DANN* model is able to learn a good representation of the target function even with sparse training data without being biased by the domain

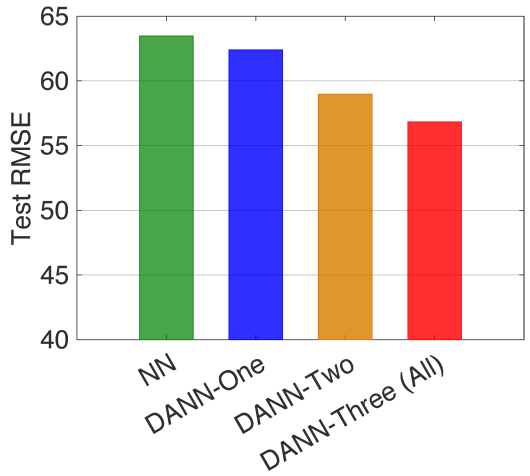


Figure 8: **Test RMSE score while extending the functionality to incorporate more than one constraint (lower the better). We use the noisy validation set with $\rho = 20\%$. As we see, while applying no constraints (i.e. *NN*) the RMSE is the highest. Applying all three domain constraints gives the best performance.**

constraint. The *DANN* model shows a mean performance improvement of 16.87% over the *NN* model across all the experiments performed with limited training data.

VI. Conclusion

In this paper, we have introduced domain adapted neural networks to leverage prior domain knowledge and learn good model representations in sparse and noisy settings. We incorporate the innate domain knowledge into the model during training through the introduction of specially formulated domain loss terms. Specifically, *DANN* encodes *monotonicity* and *approximation* domain constraints into the loss function, learning from both data and domain knowledge during training. We demonstrate the effectiveness of *DANN* on synthetic and real data, with significant performance gains in both cases. We plan to extend the *DANN* model to incorporate more sophisticated deep learning architectures and domain rules in the future.

Acknowledgements: This work is supported in part by the National Science Foundation via grants DGE-1545362, IIS-1633363, and by the Army Research Laboratory under grant W911NF-17-1-0021.

References

[1] S. Srinivas *et al.*, "A taxonomy of deep convolutional neural nets for computer vision," *Frontiers in Robotics and AI*, vol. 2, p. 36, 2016.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *arXiv preprint arXiv:1708.02709*, 2017.

[3] L. Galway, D. Charles, and M. Black, "Machine learning in digital games: a survey," *Artificial Intelligence Review*, vol. 29, no. 2, pp. 123–161, 2008.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[5] J. Zhang and C. Zong, "Deep neural networks in machine translation: An overview," *IEEE Intelligent Systems*, vol. 30, no. 5, pp. 16–25, 2015.

[6] C. Chu and R. Wang, "A survey of domain adaptation for neural machine translation," *arXiv preprint arXiv:1806.00258*, 2018.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[8] G. Marcus, "Innateness, alphazero, and artificial intelligence," *CoRR*, vol. abs/1801.05667, 2018.

[9] Y. S. Abu-Mostafa, "A method for learning from hints," in *NIPS*, 1993, pp. 73–80.

[10] A. Karpatne, W. Watkins, J. Read, and V. Kumar, "Physics-guided neural networks (pgnn): An application in lake temperature modeling," *arXiv preprint arXiv:1710.11431*, 2017.

[11] S. Radovanović, B. Delibašić, M. Jovanović, M. Vukićević, and M. Suknović, "Framework for integration of domain knowledge into logistic regression," in *WIMS*, 2018.

[12] J.-X. Wang, J.-L. Wu, and H. Xiao, "Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data," *Physical Review Fluids*, vol. 2, no. 3, p. 034603, 2017.

[13] H. Xiao *et al.*, "Quantifying and reducing model-form uncertainties in reynolds-averaged navier–stokes simulations: A data-driven, physics-informed bayesian approach," *Journal of Computational Physics*, vol. 324, pp. 115–136, 2016.

[14] J. Wang, J.-L. Wu, and H. Xiao, "Incorporating prior knowledge for quantifying and reducing model-form uncertainty in rans simulations," *Int J Uncertain Quantif.*, vol. 6, no. 2, 2016.

[15] A. Trask *et al.*, "Neural arithmetic logic units," *arXiv preprint arXiv:1808.00508*, 2018.

[16] F. Arabshahi, S. Singh, and A. Anandkumar, "Combining symbolic and function evaluation expressions in neural programs," *arXiv preprint arXiv:1801.04342*, 2018.

[17] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *PNAS*, vol. 104, no. 24, pp. 9943–9948, 2007.

[18] G. Truesdale, A. Downing, and G. Lowden, "The solubility of oxygen in pure water and sea-water," *Journal of Applied Chemistry*, vol. 5, no. 2, pp. 53–62, 1955.

[19] M. Geng *et al.*, "Prediction of oxygen solubility in pure water and brines up to high temperatures and pressures," *Geochim Cosmochim Ac*, vol. 74, no. 19, pp. 5631–5640, 2010.