Artificial Intelligence and Machine Learning for Nuclear Physics

Andreas Isene*, Christian D. Nguyen*, and Daniel Fremming*

Advances in artificial intelligence/machine learning methods provide tools that have broad applicability in scientific research. These techniques are being applied across the diversity of nuclear physics research topics, leading to advances that will facilitate scientific discoveries and societal applications. This review provides a snapshot of nuclear physics research which has been transformed by artificial intelligence and machine learning techniques.

CONTENTS

I. INTRODUCTION

This Review represents an up-to-date summary of work in the application of artificial intelligence (AI) and machine learning (ML) in nuclear science, covering topics in nuclear theory, experimental methods, accelerator technology, and nuclear data.

Nuclear physics is a well-established field, with more than a century of fundamental discoveries covering a huge span of degrees of freedom, energy scales and length scales, ranging from our basic understanding of fundamental constituents of matter to the structure of stars and the synthesis of the elements in the Cosmos. Experiments produce data volumes that range in complexity and heterogeneity, thereby posing enormous challenges to their design, their execution, and the statistical data analysis.

Theoretical modeling of nuclear properties is, in most physical cases of interest, limited by the large amount of degrees of freedom in quantum-mechanical calculations. The analysis of experimental data and the theoretical modeling of nuclear systems aims, as is the case in all fields of physics, at uncovering the basic laws of motion in order to make predictions and estimations, as well as finding correlations and causations for strongly interacting matter. The broad aims of nuclear physics as a field correspond to a highly distributed scientific enterprise. Experimental efforts utilize many laboratories worldwide, each with unique operation, data acquisition, and analysis methods. Similarly, the scales of focus spanned in theoretical nuclear physics lead to broad needs for algorithmic methods and uncertainty quantification. These efforts, utilizing arrays of data types across size and energy scales, create a perfect environment for applications of AI/ML methods.

II. ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR NUCLEAR PHYSICS IN BROAD STROKES

Statistics, data science, and AI/ML form important fields of research in modern science. They describe how to learn and make predictions from data, and enable the extraction of key information about physical processes

* These authors contributed equally to this work

and the underlying scientific laws based on large datasets. As such, recent advances in AI capabilities are being applied to advance scientific discoveries in the physical sciences.

Ideally, AI represents the science of building models to perform a task without being explicitly programmed. ML tasks fall under the broader AI umbrella. We will henceforth refer to the methods discussed as "AI/ML". The idea is that there exist generic algorithms which can be used to find patterns in a broad class of datasets without having to write code specifically for each problem. The algorithm builds its own logic based on the data. The attentive reader should however always keep in mind that machines and algorithms are to a large extent developed by humans. The choice of a specific AI/ML algorithm is governed by the insights and knowledge about a specific system.

There exist many AI/ML approaches; they are often split into two main categories, supervised and unsupervised. In supervised learning, data are labeled and one lets a specific ML algorithm learn and deduce patterns in the datasets. This allows one to make predictions about future events and/or data not included in the training set. On the other hand, unsupervised learning is a method for finding patterns and relationship in datasets without any prior knowledge of the system. Many researchers also operate with a third category, dubbed reinforcement learning. This is a paradigm of learning inspired by behavioral psychology, where actions are learned to maximize reward. One may encounter reinforcement learning being accompanied by supervised deep learning methods. Furthermore, what is often referred to as semi-supervised learning, entails developing algorithms that aim at learning from a dataset that includes both labeled and unlabeled data.

Another way to categorize AI/ML tasks is to consider the desired output of a system. Some of the most common tasks are:

III. EXERCISE 1: EXPECTATION VALUES FOR ORDINARY LEAST SQUARES EXPRESSIONS

Assumptions:

1. ε is independent from x which gives: $\mathbb{E}[\varepsilon] = 0$)
2. $y = f(x) + \varepsilon$. Here we will simplify f=f(x).
3. $f(x)$ is a fixed, deterministic function of x, hence $\mathbb{E}[f(x)] = f(x)$

A. The mean-square error of our model can be written:

$$\begin{aligned}\text{MSE} &= \mathbb{E}[(y-\tilde{y})^2] \\ &= \mathbb{E}[y^2 - 2y\tilde{y} + \tilde{y}^2] \\ &= \mathbb{E}[y^2] - 2\mathbb{E}[y\tilde{y}] + \mathbb{E}[\tilde{y}^2]\end{aligned}$$

We look into each term:

$$\mathbb{E}[y^2] \tag{1}$$

$$\mathbb{E}[y\tilde{y}] \tag{2}$$

$$\mathbb{E}[\tilde{y}^2] \tag{3}$$

From (1) we have:

$$\begin{aligned}\mathbb{E}[y^2] &= \mathbb{E}[(f+\varepsilon)^2] = \mathbb{E}[f^2 + 2f\varepsilon + \epsilon^2] \\ &= \mathbb{E}[f^2] + \mathbb{E}[\varepsilon^2] \\ &= \mathbb{E}[f^2] + \sigma^2 = f^2 + \sigma^2\end{aligned}$$

From (2):

$$\begin{aligned}\mathbb{E}[y\tilde{y}] &= \mathbb{E}[y]\mathbb{E}[\tilde{y}] = \mathbb{E}[f+\varepsilon]\mathbb{E}[\tilde{y}] \\ &= \mathbb{E}[f]\mathbb{E}[\tilde{y}] = f\mathbb{E}[\tilde{y}]\end{aligned}$$

We have:

$$\begin{aligned}Var[\tilde{y}] &= \mathbb{E}[(\tilde{y} - \mathbb{E}[\tilde{y}])^2] \\ &= \mathbb{E}[\tilde{y}^2 - 2\tilde{y}\mathbb{E}[\tilde{y}] + (\mathbb{E}[\tilde{y}])^2] \\ &= \mathbb{E}[\tilde{y}^2] - 2\mathbb{E}[\tilde{y}]\mathbb{E}[\tilde{y}] + (\mathbb{E}[\tilde{y}])^2 \\ &= \mathbb{E}[\tilde{y}^2] - (\mathbb{E}[\tilde{y}])^2 \\ &\implies \mathbb{E}[\tilde{y}^2] = Var[\tilde{y}] + (\mathbb{E}[\tilde{y}])^2\end{aligned}$$

Put the terms together:

$$\begin{aligned}&\mathbb{E}[y^2] - 2\mathbb{E}[y\tilde{y}] + \mathbb{E}[\tilde{y}^2] \\ &= f^2 + \sigma^2 - 2f\mathbb{E}[\tilde{y}] + (Var(\tilde{y}) + (\mathbb{E}[\tilde{y}])^2) \\ &= f^2 - 2f\mathbb{E}[\tilde{y}] + (\mathbb{E}[\tilde{y}])^2 + Var(\tilde{y}) + \sigma^2 \\ &\implies \mathbb{E}[(f - \mathbb{E}[\tilde{y}])^2] + Var(\tilde{y}) + \sigma^2\end{aligned}$$

First term in above expression can be approximated:

$$\mathbb{E}[(f - \mathbb{E}[\tilde{y}])^2] \simeq \frac{1}{n}\sum_i (y_i - \mathbb{E}[\tilde{y}])^2 \text{ ,where } f_i \simeq y_i \tag{4}$$

From (4) we have that MSE can be written:

$$\mathbb{E}[(f - \mathbb{E}[\tilde{y}])^2] \simeq \frac{1}{n}\sum_i (y_i - \mathbb{E}[\tilde{y}])^2 = Bias[\tilde{y}] \tag{5}$$

Similarly, the variance can be expressed as:

$$\mathbb{E}[(\tilde{y} - \mathbb{E}[\tilde{y}])^2] \simeq \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{y}])^2 = Var[\tilde{y}]$$

Setting in both the bias and variance terms into the equation we then obtain:

$MSE = Bias(\tilde{y}) + Var(\tilde{y}) + \sigma^2$

B. Discussing the terms

The bias term represents the error caused by in-built assumptions in our used method. It consists of the average prediction over all data sets subtracted with the desired regression function. The variance term describes the difference between solutions for the data sets and their average(variance of the model arounds its mean) (**?**).

As for the sigma term, it represents the variance of the error, i.e. the noise of our method, where this quantity is expected to be irreducible. Since the sigma term is irreducible we will omit its interpretation in this section. The equation of minimum squared error(MSE) can be regarded as the expected loss when using our regression model, where we desire to minimize the error. This minimization can be dubbed the bias-variance trade-off, since it is about striking a balance between the bias and variance term. As for interpreting the terms(bias and variance), we will look into 2 scenarios as to how model complexity affects their behaviour. Let $\tilde{y}$ denote our model. The more complex it is, the better it is to go through the data points, hence we will acquire a lower bias. Respectively, the variance will be higher because the a higher model complexity makes the model "move" through more data points and it learns the noise of the training data. This flexibility in the model will lead run into the problem of overfitting our model, it works well on the training data we input, but it will run into problems once given new data it has never "seen", i.e. overfitting makes the model less able to generalize.

If the model is too rigid, it will output a high bias and low variance, and the model is not sufficient to fit the data points - we say it will underfit. Similarly to the case with a complex model, it will result in a poor generalization.

IV. STUDYING MSE VALUES THROUGH ANALYSIS OF BIAS-VARIANCE

The main code is obtained from: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter3.html#the-bias-variance-tradeoff.
Modified code uses $y = sin(x) + log(x^2)$ as the one-dimensional function.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.utils import resample

np.random.seed(2018)

n = 40
n_boostraps = 100
maxdegree = 14


# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.sin(x) + np.random.normal(0, 0.1, x.sha
error = np.zeros(maxdegree)
bias = np.zeros(maxdegree)
variance = np.zeros(maxdegree)
polydegree = np.zeros(maxdegree)
x_train, x_test, y_train, y_test = train_test

for degree in range(maxdegree):
    model = make_pipeline(
        PolynomialFeatures(degree=degree), Li
    )
    y_pred = np.empty((y_test.shape[0], n_boo
    for i in range(n_boostraps):
        x_, y_ = resample(x_train, y_train)
        y_pred[:, i] = model.fit(x_, y_).pred

    polydegree[degree] = degree
    error[degree] = np.mean(np.mean((y_test -
    bias[degree] = np.mean((y_test - np.mean(
    variance[degree] = np.mean(np.var(y_pred,
    print("Polynomial degree:", degree)
    print("Error:", error[degree])
    print("Bias^2:", bias[degree])
    print("Var:", variance[degree])
    print(
        "{} >= {} + {} = {}".format(
            error[degree],
            bias[degree],
            variance[degree],
            bias[degree] + variance[degree],
        )
    )

plt.plot(polydegree, error, label="Error")
plt.plot(polydegree, bias, label="bias")
plt.plot(polydegree, variance, label="Variance
plt.legend()
plt.show()
```

Results from the bias and variance trade-off as function of the model complexity can be seen in the table **??** below. Here we set the number of data points to 40 and bootstrap to 100 iterations. Blue rows highlights where the model complexity yields the lowest error:

Order	Error	Bias^2	Variance
0	3.222694	3.104592	0.118102
1	2.637989	2.492254	0.145734
2	1.234957	1.095855	0.139102
3	1.142538	0.978143	0.164395
4	0.604439	0.475311	0.129127
5	blue!250.578585	blue!250.4131747	blue!250.165410
6	blue!250.370845	blue!250.214435	blue!250.156410
7	blue!250.435129	blue!250.229928	blue!250.205200
8	0.739773	0.360638	0.379134
9	1.179098	0.368951	0.810147
10	0.917037	0.390615	0.526421
11	7.420075	0.278572	7.141503
12	9.530186	0.267499	9.262687
13	16.252874	0.147212	16.105661

TABLE I Bias and variance trade-off as function of polynomial complexity.

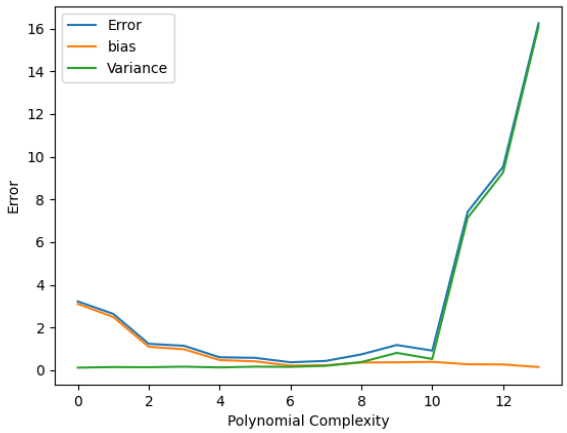

FIG. 1 Plot Analysis of plot of error, bias, and variance as function of increasing complexity

V. BIAS-VARIANCE TRADE-OFF

- **Low Complexity(< 5):** High bias and low error, the model underfits our test data and is not able to capture the complexity of our target function. This gives us a somewhat higher error.
- **Optimal Complexity($5 - 7$):** Our error is relatively low meaning that our model does a decent job in describing underlying patterns in the target function. This is provided by the bias-term. The variance on the other hand, is low enough so our model does not overfit. For the polynomial degree between 4-7 we can see that the trade-off between bias and variance is minimizes the error.
- **High Complexity(> 7):** The low bias describes now that our model is exceptional at fitting to our target function, but our model is now too specific - it is restricted to perform well with the data set we provided, but will struggle with new data sets. The high variance shows that our model has been overfitted and the higher error describes that our model is no longer suitable for generalization.

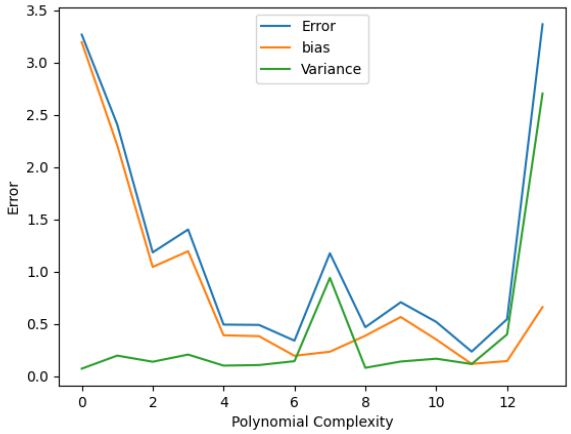

FIG. 2 Plot Analysis of plot of error, bias, and variance as function of increasing complexity

From the table we can see the error of the model is at its lowest when the degree of the polynomial is 6. This also holds true for visual inspection of the graph in figure **??**.

VI. EFFECT OF NUMBER OF DATA POINTS

From quick plots we observed that for low function complexity, the error, bias and variance stayed low (closer to 0, seemingly). A major difference is the magnitude of error. On the y-axis we observed the error for (data points = 10) to take values between 0-700. When setting number of data points to 70, the error decreased and was within the range 0-3.5, but the error, bias and variance stayed at an all time low of zero until a polynomial degree of 10.

VII. BOOTSTRAP RESAMPLING

As for bootstrap resampling we worked with the following number of bootstraps while holding number of samples fixed: (10,200). For number of bootstraps= 10 the error varied where the bias was approximate close to the error to begin with. The graph of bias and errors stayed the same until degree 11 where error and variance were dominant and closer to each other and suddenly made a jump(see **??**). As for number bootstrap= 200, we had another behaviour. As depicted in figure **??**.

Classification: Outputs are divided into two or more classes. The goal is to produce a model that assigns

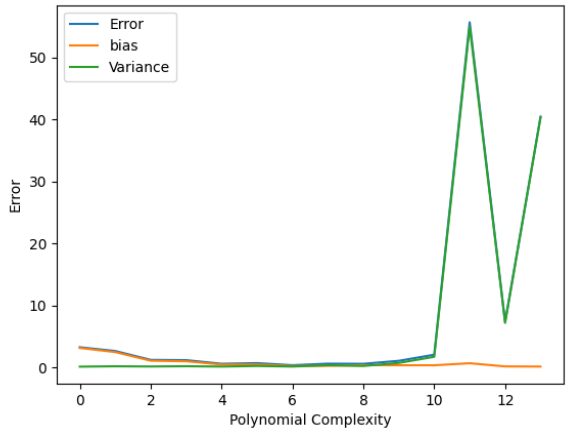

FIG. 3 Plot Analysis of plot of error, bias, and variance as function of increasing complexity

inputs into one of these classes. An example is to identify digits based on pictures of hand-written numbers.

Regression: Finding a functional relationship between an input dataset and a reference dataset. The goal is to construct a function that maps input data into continuous output values.

Clustering: Data are divided into groups with certain common traits, without knowing the different groups beforehand. This AI/ML task falls under the category of unsupervised learning.

Generation: Building a model to generate data that are akin to a training dataset in both examples and distributions of examples. Most generative models are types of unsupervised learning.

In Table **??** we list many of the methods encountered in this work, with their respective abbreviations.

The methods we cover here have three central elements in common, irrespective of whether we deal with supervised, unsupervised, or semi-supervised learning. The first element is some dataset (which can be subdivided into training, validation, and test data), the second element is a model, which is normally a function of some parameters to be determined by the chosen optimization process. The model reflects our prior knowledge of the system (or lack thereof). As an example, if we know that our data show a behavior similar to what would be predicted by a polynomial, fitting the data to a polynomial of some degree would determine our model. The last element is a so-called cost (or loss, error, penalty, or risk) function which allows us to present an estimate on how good our model is in reproducing the data it is supposed to train. This is the function which is optimized in order to obtain the best prediction for the data under study. The simplest cost function in a regression analysis (fitting a continuous function to the data) is the so-called mean squared error function while for a binary classification problem, the so-called cross entropy is widely used, see, e.g., (**???**) for more details. We will henceforth refer to this element as the assessment of a given method.

Traditionally, the field of AI/ML has had its main focus on predictions and correlations. In AI/ML and prediction-based tasks, we are often interested in developing algorithms that are capable of learning patterns from existing data in an automated fashion, and then using these learned patterns to make predictions or assessments of new data. In some cases, our primary concern is the quality of the predictions or assessments, with perhaps less focus on the underlying patterns (and probability distributions) that were learned in order to make these predictions. However, in many nuclear physics studies, we are equally interested in being able to estimate errors and find causations. In this Colloquium, we emphasize the role of predictions and correlations as well as error estimation and causations in statistical learning and ML. For general references on these topics and discussions of frequentist and Bayesian methodologies, see, e.g., (**????**).

REFERENCES

TABLE II Table of AI/ML with indication on the main type of learning (S: supervised, U: unsupervised, Semi-S: semi-supervised).

Acronym	Method	Type of Learning
AE	Autoencoders	U
ANN	Artificial Neural Networks	S
BED	Bayesian Experimental Design	S
BM	Boltzmann Machines	U
BMA	Bayesian Model Averaging	S
BMM	Bayesian Model Mixing	Semi-S
BO	Bayesian Optimization	S
BNN	Bayesian Neural Networks	S
CNN	Convolutional Neural Networks	S
EMB	Ensemble Methods and Boosting, including Decision Trees and Random Forests	S
GAN	Generative Adversarial Networks	U
GP	Gaussian Processes	Semi-S
KNN	k-nearest neighbors	U
KR	Kernel Regression	S
LR	Logistic Regression	S
LSTM	Long short-term memory	S
PCA	Principal Component Analysis & Dimensionality Reduction	U
REG	Linear Regression	S
RL	Reinforcement Learning	Neither S nor U
RNN	Recurrent Neural Networks	S
SVM	Support Vector Machines	S
VAE	Variational Auto Encoders	U