MACHINE LEARNING GUIDED DISCOVERY AND DESIGN FOR INERTIAL CONFINEMENT FUSION

A Dissertation

by

KELLI DENISE HUMBIRD

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Ryan G. McClarren
Committee Members,	Jayson L. Peterson
	Marvin L. Adams
	Bani K. Mallick
	Jim E. Morel
Head of Department,	Yassin Hassan

May 2019

Major Subject: Nuclear Engineering

ABSTRACT

Inertial confinement fusion (ICF) experiments at the National Ignition Facility (NIF) and their corresponding computer simulations produce an immense amount of rich data. However, quantitatively interpreting that data remains a grand challenge. Design spaces are vast, data volumes are large, and the relationship between models and experiments may be uncertain.

We propose using machine learning to aid in the design and understanding of ICF implosions by integrating simulation and experimental data into a common framework. We begin by illustrating an early success of this data-driven design approach which resulted in the discovery of a new class of high performing ovoid-shaped implosion simulations. The ovoids achieve robust performance from the generation of zonal flows within the hotspot, revealing physics that had not previously been observed in ICF capsules.

The ovoid discovery also revealed deficiencies in common machine learning algorithms for modeling ICF data. To overcome these inadequacies, we developed a novel algorithm, deep jointly-informed neural networks (DJINN), which enables non-data scientists to quickly train neural networks on their own datasets. DJINN is routinely used for modeling data ICF data and for a variety of other applications (uncertainty quantification; climate, nuclear, and atomic physics data). We demonstrate how DJINN is used to perform parameter inference tasks for NIF data, and how transfer learning with DJINN enables us to create predictive models of direct drive experiments at the Omega laser facility.

Much of this work focuses on scalar or modest-size vector data, however many ICF diagnostics produce a variety of images, spectra, and sequential data. We end with

a brief exploration of sequence-to-sequence models for emulating time-dependent multiphysics systems of varying complexity. This is a first step toward incorporating multimodal time-dependent data into our analyses to better constrain our predictive models.

DEDICATION

"I do not see the logic of rejecting data just because they seem incredible."

- Fred Hoyle

This work is dedicated to those who made it possible. Thank you.

ACKNOWLEDGEMENTS

Words are a completely inadequate form for expressing my immense gratitude toward those in my life that made this work possible. Just a few years ago I was tempted to end my scientific career altogether, and had it not been for the support system around me, I never would have found what very much feels like my calling.

Thank you first and foremost to my family. My parents have been endlessly patient with my terminal student status. Their assistance, both emotionally and financially, in my move from Texas to California was vital; I am not sure I would have made the decision to return to Livermore if they had not been so enthusiastic about the opportunity. Thank you to my sister, who has been a best friend and someone with whom I can have deep discussions about the future of artificial intelligence and the source of human consciousness while be-dazzling wine glasses and watching Harry Potter. Thank you to my brother, for keeping me humble and helping shape me into a tough woman who is not afraid to handle things on her own. Thank you to my brother's family, who have been such a big source of happiness for everyone; the best part of being a sister is getting to be an aunt.

Thank you to my advisor, Ryan McClarren, who willingly accepted me as a graduate student when I suddenly decided to return to Texas A&M after a brief stint at another university. Dr. McClarren was instrumental in getting me on the career path I have found, and I am so glad to have had the opportunity to learn from and work with him.

Thank you to those who have made my years at Lawrence Livermore National Laboratory (LLNL) endlessly enjoyable and productive. Thank you to the Livermore Graduate Scholars Program, for awarding me the opportunity to perform my doctoral

research at LLNL. Thank you to the Design Physics and ICF programs for financial support, which allowed me to attend many conferences during my time as a scholar.

Thank you to Brian Spears and the entire ICF ensembles team, who have treated me as part of the group and given me a voice since the day I first stepped on site. The team has done wonders to increase my confidence and competence as a scientist, and have helped me develop research and communication skills I never would have gained had I not moved to LLNL. The team has been a vital source of inspiration, motivation, and friendship during my PhD; it is hard to imagine I will ever find such an amazing group of coworkers elsewhere. Thank you Michael, for going on long coffee breaks with me and letting me vent about science and about life. Thank you John, for always piquing my intellectual curiosity; conversations with you remind me of why I love science: there is so much waiting to be known! And a huge thank you to Luc Peterson, who has gone above and beyond any reasonable expectations a graduate student should have for their advisor. It is challenging to summarize the impact Luc has had on my career in just a few sentences, but never have I woken up bright-eyed and eager for 14+ hour work days, as often as I have since I started working with him. Beyond being a constant source of intellectual excitement, he has been an outstanding friend, who has had my back and fought for me, even in the most ridiculous siutations. It is no exaggeration at all to say none of this would have happened had our paths not crossed. I can't wait to see what more we can accomplish together in the future!

Thank you to my partner, Arnulfo, for being endlessly supportive of every decision I have made. From transfering universities, to attempting to bail on science to get an MBA, to packing up my life and moving to California without a guaranteed source of funding; he went along with every idea I had and helped write so many application essays. He has seen me through some of my highest highs, and lowest lows; I do

not think there is anyone quite as strong as him for sticking with me through it all. He continues to lift me up and cheer me on, even when it means nights apart and countless frequent flier miles. Thank you, Arnulfo, for loving me and for believing in me fiercely and for being my biggest fan.

Thank you to all of the friends I have made over the years, both two- and four-legged, who have kept me sane throughout this entire journey. I am so grateful I found the horse community early in life, because I have found an instant group of friends in every city I have lived in since; friends who will pick you up from the ground after your horse bucked you off (but not before they caught your horse because they understand priorities), who will stay at the barn overnight helping you tend to a sick horse, and who will, without hesitation, drive 100 miles with you to get to an emergency clinic. "The barn" has been a source of stability, humility, and love that I have needed to make it through the past several years. And thank you to the equines themselves, who have taught me the most important lesson of all: every problem in life looks infinitely smaller when viewed from the back of a good horse.

NOMENCLATURE

AE	Autoencoder
AI	Artificial Intelligence
BART	Bayesian Additive Regression Trees
B-DJINN	Bayesian Deep Jointly-Informed Neural Network
CPU	Central Processing Unit
DJINN	Deep Jointly-Informed Neural Network
DNN	Deep Neural Network
DSR	Down Scatter Ratio
EV	Explained Variance
GP	Gaussian Process
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HDC	High Density Carbon
HED	High Energy Density
HL	Hidden Layer
HEDP	High Energy Density Physics
IAE	Integrated Absolute Error
ICF	Inertial Confinement Fusion
ITF	Ignition Threshold Factor
ITFX	Experimental Ignition Threshold Factor
KNN	K-Nearest Neighbors
LANL	Los Alamos National Laboratory
LEH	Laser Entrance Hole
LLE	Laboratory for Laser Energetics

LLNL	Lawrence Livermore National Laboratory
LPI	Laser-Plasma Interactions
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MARS	Multivariate Adaptive Regression Splines
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MSE	Mean Squared Error
NIF	National Ignition Facility
NN	Neural Network
NSAE	Normalized Sum of Absolute Errors
QOI	Quantity of Interest
RBF	Radial Basis Function
RF	Random Forest
RHGD	Radiation Hydrodynamics with Gray Diffusion
RM	Richtmeyer-Meshkov
RMS	Root-Mean-Square
RNN	Recurrent Neural Network
RT	Rayleigh-Taylor
Seq2seq	Sequence-to-Sequence
SD	Standard Deviation
SSD	Smoothing by Spectral Dispersion
SVM	Support Vector Machine
TF	TensorFlow
UQ	Uncertainty Quantification

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a dissertation committee consisting of Professor McClarren as the academic advisor, Dr. Peterson as the research advisor at Lawrence Livermore National Laboratory (LLNL), Professors Adams and Morel of the Department of Nuclear Engineering, and Professor Mallick of the Department of Statistics.

Many of the ICF simulation databases discussed in this work were generated by the ICF Ensembles Simulations team at LLNL. Team members include Brian Spears, Luc Peterson, John Field, Ryan Nora, Jim Gaffney, Michael Kruse, Scott Brandon, and Paul Springer. Experimental data for NIF ICF implosions was made available by the ICF and NIF programs at LLNL.

HYDRA simulations and the corresponding figures for the ovoid implosion in Chapter 2 were produced by Luc Peterson.

The LILAC database and experimental data from the Omega laser facility was provided by R. Betti, V. Gopalaswamy, and J. Knauer at the University of Rochester Laboratory for Laser Energetics (LLE).

The ICF simulation database in Chapter 7 was produced with the Merlin workflow, developed by the Machine Learning Strategic Initiative team at LLNL. The ICF simulation code, JAG, was developed by Jim Gaffney.

I also wish to thank several researchers at LLNL for fruitful discussions over the past few years: D. Callahan, N. Meezan, C. Young, R. Ward, J. Salmonson, and F. Graziani.

All other work conducted for the dissertation was completed by the student. The simulations and analyses were conducted on resources made available by LLNL and

Los Alamos National Laboratory (LANL).

Funding Sources

This work was funded by the Livermore Graduate Scholars Program and the Design Physics division at LLNL. Additional financial assistance came from the Philosophic Education Organization's Scholar Award, for which I am inexpressibly grateful.

Some of the content in this dissertation is reproduced with permission from the following journal articles:

- © 2017 AIP Publishing. Reprinted, with permission, from J. L. Peterson, K. D. Humbird, J. E. Field, S. T. Brandon, S. H. Langer, R. C. Nora, B. K. Spears, and P. T. Springer, "Zonal flow generation in inertial confinement fusion implosions", Physics of Plasmas 24, 032702 (2017).

- © 2018 IEEE. Reprinted, with permission, from K. D. Humbird, J. L. Peterson, R. G. McClarren, "Deep Neural Network Initialization With Decision Trees", IEEE Transactions of Neural Networks and Learning Systems (October 2018).

Some of the content in this dissertation is available via preprint access on arxiv.org:

- K. D. Humbird, J. L. Peterson, R. G. McClarren, "Predicting the time-evolution of multi-physics systems with sequence-to-sequence models", (2018).

- K. D. Humbird, J. L. Peterson, R. G. McClarren, "Transfer learning to model inertial confinement fusion experiments", (2018).

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Released as LLNL-TH-769065.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Nuclear fusion is the process of combining two light nuclei in order to form a heavier nucleus. This process releases immense amounts of energy as a result of the change in mass when two nuclei are fused to form a new, single nucleus. As an example, consider the reaction:

$$\mathrm{D} + \mathrm{T} \longrightarrow \mathrm{He4} + \mathrm{n} + \mathrm{Q}, \tag{1.1}$$

in which deuterium (D) and tritium (T) are fused together to form an alpha particle (He4) and a neutron, and releases energy Q. The released energy is the difference in binding energies (B) between the final and initial products in Eq. 1.1. The binding energy is the energy required to divide a nucleus into its component neutrons and protons:

$$\mathrm{B} = \mathrm{c}^2(\mathrm{Z}\mathrm{m}_\mathrm{p} + (\mathrm{A} - \mathrm{Z})\mathrm{m}_\mathrm{n} - \mathrm{m}), \tag{1.2}$$

where A and Z are the mass and atomic numbers, respectively, and m_p and m_n are the proton and neutron masses, respectively, and m is the mass of the nucleus. For the DT reaction, the difference in binding energies yields about 17.6 MeV of energy that is carried by the alpha particle (3.5 MeV) and the neutron (14.1 MeV). This is far more energy per unit mass than is released by nuclear fission, making fusion is an attractive alternative for clean and efficient energy production. However, achieving fusion energy production in a controlled, sustainable environment is extremely challenging.

There are currently two major approaches to achieving nuclear fusion in a lab-

oratory setting — magnetic and inertial confinement. In this work, we will focus exclusively on inertial confinement fusion, which relies on the inertia of the fuel mass to provide confinement to the burning fuel. We will discuss two current approaches to ICF throughout the dissertation: direct and indirect drive. In section 1.1, we will discuss these two apporachs in more detail. In section 1.2, we present challenges associated with designing and understanding ICF experiments, followed in section 1.3 by an outline of our efforts to improve how we analyze ICF data, which will be detailed throughout the remainder of this work.

1.1 Brief overview of inertial confinement fusion

The goal of ICF is to compress fusion fuel to extreme temperature, density, and pressure, such that the fuel begins to fuse and thus produces immense amounts of thermonuclear energy. Most fusion experiments use deutrium (D) and tritium (T) fuel, as DT fusion reactions have a higher cross section (σ), which can be thought of as the probability of the reaction occuring, across the range of temperatures achievable in fusion experiments. A typical ICF fuel capsule is illustrated on the left side of Fig. 1.1. The capsule is a spherical shell composed of an ablator on the outermost layer, with a shell of DT ice inside of the ablator, followed by DT gas fill in the center. The capsules are about 2 mm in diameter, with typical ablator and DT ice thicknesses of about 70 microns.

To initiate fusion reactions, the fuel capsule is compressed with a driver – a laser in the case of direct drive, or an X-ray bath created in a hohlraum for indirect drive. The driver ablates the outer layer of the fuel capsule and as the ablator material blows off, a rocket-like force acts on the inner region of the fuel shell, compressing the capsule radially inward [79, 135]. Often, the driver used to compress the fuel is varied in time to launch multiple shocks through the shell which are often designed

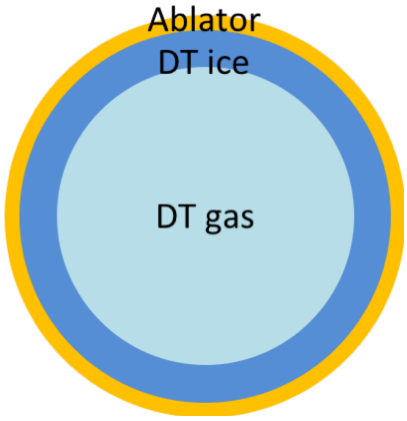


Figure 1.1: Illustration of a typical ICF fuel capsule. The outer ablator material surrounds a DT ice layer, with DT gas at the center.

to merge in the ice layer, and a single shock propagates through the interior gas fill. The laser is then turned off, and the capsule coasts inward at constant velocity until the shock reaches the center of the capsule, reflects, and begins propgating outward. When this shock hits the inner surface of the shell, the shell decelerates and a subsequent shock reflected off the shell propgates back toward the center of the capsule. The hotspot develops a unfiorm pressure profile as the reflected shocks diminish and the shell "stagnates". Under ideal stagnation conditions, the gas at the center of the capsule is heated substantially, creating a region of low density plasma where conditions are ideal for fusion to occur. The alpha particles produced in the fusion reactions deposit much of their energy into the central hotspot, further heating the gas. Thermal conduction from the hotspot and the desposition of alpha particle energy into the inner shell surface combine to create a thermonuclear burnwave that rapdily propagates through the cold shell, burning the fuel and producing immense amounts of fusion energy. Eventually, the shell begins to expand outward, cooling the hotspot and ending thermonuclear burn.

"Ignition" is typically defined as the development of a central hotspot that ignites a fusion burnwave, consuming the DT fuel and producing more energy than was required to initiate the process [79]. There are several metrics that attempt to define the requirements for ignition, such as Lawson criteria [75, 149, 16], which sets limits on the minimum density and confinement time required for ignition:

$$n\tau > \frac{12k_BT}{\langle\sigma v\rangle Q}, \tag{1.3}$$

where n is the density of DT (assuming the fuel is half deuterium, half tritium), τ is the confinement time, k_B is the Boltzmann constant, T is the temperature of the burning plasma, $\langle\sigma v\rangle$ is the DT reactivity, and Q is 17.6 MeV for DT fusion reactions. For temperatures on the order of 5-10 keV, the Lawson criteria becomes:

$$n\tau > 10^{14} - 10^{15} s/cm^3. \tag{1.4}$$

Inertial confinement fusion aims to meet the Lawson criteria by reaching extremely high densities for a short period of time, in contrast to magnetic fusion which aims to contain low density plasmas for longer times [125]. Since the confinement time and density are difficult to measure in an experiment, alternative ignition metrics based on quantities that can be measured have also been proposed, such as the ignition threshold factor [123]. In general, the conditions for ICF ignition require confinement times on the order of picoseconds (ps) and densities of 1000x the liquid density in order to meet the Lawson criteria; temperatures of many keV where the reactivity of DT is highest; and areal densities above 0.3 g/cm^2 to stop alpha particles in the shell, resulting in heating and burning of the cold fuel.

Achieving the necessary conditions for ignition is challenging in practice; the fuel must be compressed symmetrically– even small deviations from spherical compression

can lead to large losses in performance [51, 22, 72, 48]. Furthermore, any small defects or perturbations on the fuel shell amplify significantly during compression, which can lead to holes in the fuel shell or contamination in the central hotspot; this places challenging engineering tolerances on the fabrication of the capsules, the membranes ("tents") that hold the capsule in place within a hohlraum, and the filltube used to insert the DT fuel into the capsule [110, 134].

Two approaches to achieving laser-driven ICF ignition are currently being tested in laboratory settings: direct drive, in which lasers impinge directly upon the fuel capsule, and indirect drive, in which lasers heat up a hohlraum to create an X-ray bath, which impinges upon the fuel capsule. Each approach has its own set of advantages and challenges; in subsections 1.1.1 and 1.1.2 we will briefly overview indirect and direct drive experiments currently being carried out at the National Ignition Facility [93] and Omega laser facility [66], respectively.

1.1.1 Indirect drive ICF

The NIF currently carries out indirect drive ICF experiments [49]. In these experiments, 192 laser beams (totaling in approximately 2 MJ of energy) are focused into a small cylindrical hohlraum (approximately 6 mm in diameter, 10mm tall); the setup is illustrated in Fig. 1.2. The laser beams heat the hohlraum, which is typically made of gold or depleted uranium for high X-ray conversion efficiency [130]. The heated hohlraum produces a bath of X-rays that impinge upon the fuel capsule, which is held in place in the center of the hohlraum by thin membranes called capsule support tents. The X-rays ablate the outer shell of the fuel capsule – the ablator – which is often made of high density carbon (HDC), plastic (CH), or berylium (Be). As the outer shell material is ablated away, it creates an inwardly-directed force which compresses the inner portion of the fuel capsule shell as described in the

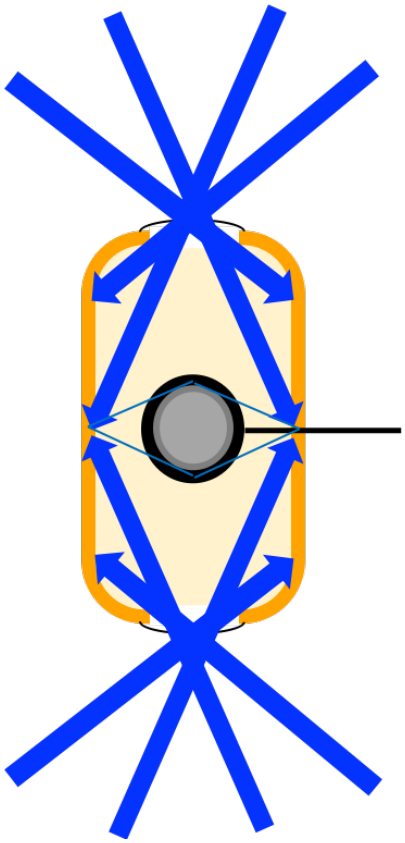


Figure 1.2: Illustration of a hohlraum with a fuel capsule in the center. The laser beams enter the hohlraum through laser entrance holes at the top and bottom of the cylinder. A filltube on the right of the figure is used to fill the fuel cpasule with DT. Thin capsule support tents hold the capsule in place in the hohlraum.

previous section.

There are many challenges that prevent indirect drive experiments from performing ideally. In particular, there are several sources of asymmetry in the system which make spherical compression difficult. The fuel is placed in a cylindrical hohlraum, and the lasers enter the hohlraum through laser entrance holes (LEH) at the top and bottom of the cylinder and illuminate discrete regions of the hohlraum walls. This setup is inherently asymmetric, and thus many efforts at NIF have been focused

on adjusting the laser pulse to offset the physical asymmetries in the hohlraum to irradiate the shell as uniformally as possible [72, 138].

Engineering features such as the capsule support tent, surface roughness of the ablator or DT ice layer, and the gas fill tube that is attached to the capsule also distort shell compression. Many efforts have been made to minimize the impact of these features, by testing alternative capsule support systems [95], putting limits on surface roughness [50], and reducing the size of the filltube [28]. These engineering features impart small perturbations which get amplified as the capsule is compressed. Shocks can cause Richtmyer-Meshkov (RM) [89, 112, 105] growth of these small-scale imperfections, which can be amplified by the Rayleigh-Taylor (RT) instability [110, 134] during the main capsule acceleration. The combined effects of asymmetric radiation drives and engineering features can lead to a distorted asymmetric shell at stagnation, which has been shown to adversely affect performance [51, 22, 72, 48].

1.1.2 Direct drive ICF

In direct drive, laser beams are arranged in a nearly-spherical configuration around the target, and the lasers directly ablate the outer surface of the fuel capsule. Many direct drive experiments are carried out at the Omega laser facility at the Laboratory for Laser Energetics (LLE), a laser system with 60 beams totaling in 30 kJ of energy.

Direct drive ICF experiments suffer from many of the same challenges as indirect drive. The capsule stalk and ice roughness lead to RM and RT instabilities, and symmetric compression is made challenging by the finite number of laser beams which create discrete regions of focused light upon the capsule surface rather than smooth, spherical irradiation. Many developments have been made to reduce irradiation non-uniformities, including smoothing by spectral dispersion (SSD) [118, 117, 113], beam

overlap, and polarization smoothing with distributed phase plates [69, 78]. These technological developments have improved irradiation uniformity at Omega to less than 1% RMS (root mean square) fluctuations averaged over 300 ps of irradiation time, which is thought to have a minimal effect on implosion performance. The major advantage of direct drive is the ability to couple the full energy of the laser to the capsule – there are no losses of energy due to the X-ray conversion eficiency of a hohlraum. However, achieving smooth irradiation and mitigating laser-plasma interactions while scaling the system up to a laser the size of NIF remains a challenge.

1.2 Designing and interpreting ICF experiments

In direct and indirect drive, a central focus of ICF research is dedicated to mitigating and minimizing sources of asymmetries and performance degradation. Overcoming these challenges is particularly difficult given the time and monetary expense of experiments. Researchers therefore rely heavily on computer simulations to search for promising experimental designs – from changing the shape of the laser pulse to modifying the power balance between laser beams to adjusting the amount of dopant in the ablator. The codes used to simulate ICF experiments are complex, integrating radiation hydrodynamics with atomic physics, nuclear burn, laser and plasma physics, magentic field effects, and more [86]. At the extreme conditions reached in ICF experiments, accurately modeling all of the physical processes in high-dimensional integrated simulations is extremely expensive. Often simplifications must be made due to limitations in computational resources, and the simplified models are not always well-validated.

Determining the accuracy of ICF codes is made even more challenging by the fact that many of the initial conditions are not necesarily known a priori, or even measurable in experiments for post-shot modeling. For example, high-fideilty 3D integrated

hohlraum simulations for indirect drive experiments are prohibitively expensive, and thus one often runs low-resolution hohlraums in 2D that place symmerty constraints on the system, however these simulations are often too low resolution to accurately model the capsule as it is compressed by a factor of 20-30 in radius. Capsule-only simulations, which can model the dynamics at high resolution with moderate computational expense, are often used for detailed post-shot modeling. Capsule-only simulations take the radiation drive that is generated by the hohlraum simulation, and translate that to the imposed radiation field on the capsule. There is a loss of information when mapping the drive generated in the hohlraum simulation into a radiation drivefor the capsule simulation, and there is a risk that the hohlraum-generated drive is not an accurate depiction of what happened in the experimnent. Therefore, drive multipliers are often introduced – parameters that vary the amplitude or the asymmetries of the drive as a function of time. These multipliers are not known nor can they be measured, but they can be inferred by using the available experimental observables. Not knowing the correct simulation inputs required to model an experiment is an important challenge addressed throughout this work – it complicates the ability to decide if a simulation does not match an experiment because the physics modeled in the code is incorrect, or if it is simply due to the fact that the simulation inputs used were incorrect. Direct drive experiments do not suffer as greatly from such complications, as they do not use hohlraums and thus the simulation inputs for capsule-only simulations are directly comparable to the inputs used in an experiment.

The process of inferring simulation inputs which are consistent with experimental observations is often referred to as a "post-shot" analysis. Finding the simulation that is most consistent with the experimental data is an important step in understanding the details of the experiment. Due to the extreme scales of the experiments,

many of the diagnostics in ICF experiments can only record a limited amount of information at a small number of time points throughout the implosion. Post-shot simulations provide detailed information about the implosion that cannot be directly measured, such as the pressure in the hotspot, residual kinetic energy, and entropy. However, current post-shot analyses often do not take into account the full set of experimental measurements. The simulation input space is prohbitively large, and finding a simulation that is consistent with all of the experimental measurements is a daunting task. Often, researchers explore a small number of unknown simulation inputs, varying them a few at a time until the simulations match a few important measurements within the experimental error bars. This process often misses a distribution of other potential simulation input settings that could result in a similar quality match to experiment; thus there is a real possibility that interpretations of such post-shot analyses could be biased, incomplete, or incorrect.

In this work, the overall goal is to develop techniques that will improve how we design and understand ICF experiments. We take a data-driven approach, leveraging modern machine learning algorithms and developing new methods when necessary to bridge the gap between simulations and experiments. In the following section, I outline my specific contributions to this effort that are discussed in detail throughout the subsequent chapters.

1.3 Contributions of this work

The traditional process for designing an ICF experiment begins with a computational model of the implosion. Design parameters are hand-tuned to optimize the implosion to achieve the outcome desired in the experiment. However, the experimental observations often differ from the predictions, and the design is iterated upon in light of the experimental observations. There are many processes in the design

loop which can be made more efficient using modern data analysis techniques. For example, simulation-based design optimization is readily automated with machine learning and high-dimensional optimization algorithms, and machine learning tools can aid in quantifying the disagreement between simulation predictions and experimental observations. Throughout this dissertation, we demonstrate several ways in which machine learning tools are improving the way we design and understand ICF experiments.

We begin by demonstrating how machine learning models, or "surrogates" enable rapid simulation-based design optimization. In Chapter 2, we show that machine learning models trained on one of the largest ICF simulation databases ever created led to discovery of a new family of high performing, ovoid-shaped implosions. The existence of such implosions challenges decades of ICF research suggesting asymmetries degrade implosion performance, and demonstrates the powerful capabilities of simple machine learning models for improving experimental design.

While traditional machine learning models, like those used in the ovoid discovery, are powerful for making point predictions for moderately complicated datasets, these models are not accurate enough to quantitatively compare simulations and experiments; such work requires surrogates that are extremely accurate and can scale well to high-dimensional, high volume datasets. Furthermore, quantifying and propagating uncertainties is a vital component for predicting the outcome of future experiments, thus the surrogates should also account for the uncertainty in their own predictions.

"Deep jointly-informed neural networks" (DJINN) was developed to address these needs [61]. DJINN is a novel algorithm for designing robustly accurate deep neural networks for arbitrary datasets without requiring manual tuning of the network or expensive hyper-parameter optimization techniques. DJINN has enabled non-data

science experts to easily train accurate deep neural networks with minimal effort, and has been a vital component in our analyses of ICF data. Chapter 3 presents the algorithm in detail, and demonstrates that DJINN often outperforms other methods for designing and initializing deep neural networks.

An advantage of using deep neural network surrogates, like DJINN, is that they are readily cast into an approximate Bayesian framework. In Chapter 4, we compare the Bayesian formulation of DJINN to other Bayesian models for a variety of datasets. We demonstrate the ability to perform parameter inference with DJINN via Markov Chain Monte Carlo (MCMC) and inverse modeling, and compare the performance of these techniques to standard MCMC inference.

In Chapter 5, we expand the use of DJINN for parameter inference to ICF databases. DJINN is combined with autoencoder neural networks, designed for unsupervised dimensionality reduction, to perform advanced post-shot analyses for NIF ICF experiments. Rather than manually searching for a single set of simulation inputs that are consistent with a few experimental observables, the neural network-based post-shot searches for all of simulations inputs that are consistent with a large number of experimental measurements. This technology is readily applied to other high energy density physics experiments; for example, the technique is also used to infer material properties from atomic emission spectra [122].

Parameter inference is an important problem for indirect drive experiments at NIF, as values of many simulation inputs are unknown and cannot be measured. The inferred values of inputs changes from one experiment to the next, thus inferring inputs based on experimental observables only aids in understanding previous experiments; it does not necessarily help create predictive models for future experiments. Researchers are working on advanced Bayesian methods [33] for model calibration, which use a collection of experiments to adjust a simulation-based surrogate model's

predictions to be more consistent with the experimental data.

As an alternative to Bayesian model calibration, we explore a machine learning technique called "transfer learning" to calibrate simulation-based surrogate models to experimental data. Transfer learning uses a neural network trained on a large set of inexpensive data, freezes many of the layers of the neural network, then retrains the last layers on a smaller, but related dataset to achieve a specific task. In ICF we have large databases of simulations, but limited quantities of experimental data. In Chapter 6, we demonstrate the ability to use transfer learning with DJINN to calibrate neural networks trained on low fidelity simulations to high fidelity simulations, then to experiments. This calibration technique is illustrated on a database of direct drive simulations and a collection of experiments performed at the Omega facility. We then use the experimentally-calibrated models to perform rapid design optimization, and discuss how this approach compares to physics-guided iterative design optimization.

We end with a chapter on time series data, which we see as the next big challenge for incorpating the full suite of diagnostic information from our experiments into our predictive models. Toward this end, we explore the use of sequence-to-sequence models to predict the evolution of complex multi-physics systems in Chapter 7.

In Chapter 8, we summarize the major contributions of this work, and propose future paths of exploration for applying machine learning to ICF data.

2. MACHINE LEARNING-GUIDED DESIGN OPTIMIZATION [1]

Inertial confinement fusion is the process by which a hollow deuterium-tritium (DT) fuel capsule is heated and compressed to extreme conditions to ignite a fusion reaction [79, 135]. Ablator material on the outer surface of the capsule is heated via impinging X-rays generated within a surrounding hohlraum (indirect drive), creating a rocket-like force that compresses the DT shell. Under ideal conditions, the gas at the center of the capsule ignites a fusion burn wave that propagates outward, consuming the fuel and releasing immense amounts of fusion energy. However, implosion performance is sensitive to a variety of instabilities and asymmetries, making ignition difficult to achieve experimentally [89, 112, 105, 110, 134].

Several processes that can lead to an asymmetric shell at stagnation are expected to negatively impact performance [22, 72, 48, 51], thus one of the major goals of ICF research is to produce a nearly spherical implosion. One way to accomplish this is via the minimization of the sources of asymmetric stagnation, for example by placing engineering tolerances on the capsule surfaces [50], reducing the effects of engineering features [95, 28], and ensuring a uniform radiation drive [72, 138].

In this chapter we use a machine learning model trained on the largest ICF simulation database ever created to search a vast design space for high-performing implosions that are resilient to laser drive perturbations [106]. The "optimal" design that the machine learning model finds reveals a nonlinear stabilization process for ICF implosions that addresses drive and shell distortions via intentionally-generated large scale flows within the hotspot. These flows, which are set up by driving the

[1] "Partially Reproduced with permission from "Zonal flow generation in inertial confinement fusion implosions", J. L. Peterson, K. D. Humbird, J. E. Field, S. T. Brandon, S. H. Langer, R. C. Nora, B. K. Spears, and P. T. Springer, Physics of Plasmas 24, 032702 (2017), with the permission of AIP Publishing."

capsule asymmetrically, shear off small scale instabilities and large scale asymmetries, making the implosions robust to a variety of performance degradation sources. A consequence of the asymmetric drive is an ovoid-shaped fuel shell at stagnation; the high performance of such implosions challenges the notion that spherical shells universally have the largest ignition margins.

We discuss the dataset, the learning algorithm, and how it predicted the existence of the ovoid in the sections 2.1 and 2.2. In section 2.3, we confirm the machine learning model's prediction with a set of new simulations, and show that the ovoid implosions are more resilient to drive and engineering feature perturbtations than implosions which are driven symmetrically.

2.1 Surrogate models of ICF simulation data

Previous work illustrates the important role machine learning models, or "surrogates", have played in improving the understanding of ICF implosions. For example, the ignition threshold factor (ITF) and its experimental counterpart (ITFX) are surrogate models fit to simulation and experimental data that quantify the margin of ignition for an implosion [123, 51]. Several studies have used ensembles of simulations to train surrogates for studying the effects of laser drive asymmetries [72], and capsule fabrication imperfections [22] on implosion performance. These models have provided valuable insight into the conditions for ignition, however they are restricted to exploring low-dimensional design spaces due to computational limitations.

Advancements in supercomputer architecture and a novel on-the-fly data management technique [106] have been used to create what was then the largest collection of two-dimensional ICF simulations using the code HYDRA [86]. Creating the ensemble of simulations consumed roughly 39 million CPU-hours of computer resources during the Open Science Phase I of the Trinity Supercomputer at Los Alamos Na-

tional Laboratory and generated 5 petabytes of raw data that were processed using a novel *in-transit* data analysis technique [74].

The dataset constitutes a nine-dimensional parameter scan of time-varying drive magnitudes, drive asymmetries (described by Legendre modes $P_1,\ P_2,\ P_4$), and capsule gas fill densities. The baseline simulation is an axisymmetric variant of a high density carbon (HDC) [57] NIF [91] implosion design, meant to ignite in 1D, with a 20 μm dopant layer of 3% Si embedded in the 75 μm-thick shell of 1108 μm outer radius. Both the DT and the HDC use tabular equations of state (LEOS 1018 and 64 respectively [68, 126]) and opacities. The ice layer is 55 μm thick and the central gas has a baseline density of 5×10^{-4} g/cm^3. The initially Arbitrary-Lagrangian-Eulerian mesh with 513 angular zones and 321 impedance matched radial zones remaps to an entirely Eulerian elliptic mesh near stagnation. The nine-dimensional study constitutes Latin hypercube sampling of a space around the baseline implosion, with linearly varying drive magnitude A and asymmetry perturbations between three time points (the end of the first shock "trough," the end of the "rise" to peak laser power and the end of "peak" radiation drive). All time-dependent perturbations ramp up from time zero and down from the end of peak power. P_1 and P_4 have the same value at the three time points, but P_2 and A can vary (see Fig. 2.2a for an example). The data points are sampled linearly between $\pm$ [2, 10, 5, 25]% for $[P_1,\ P_2,\ P_4,\ A]$. The capsule gas fill density is sampled logarithmically between 0.2$\times$ and 5$\times$, for a total of 9 independent variables.

Machine learning models are trained on the data to create a surrogate model which emulates HYDRA in the 9D design space. The surrogates enable rapid exploration throughout design space to search for an implosion that robustly ignites, even under adverse conditions.

2.2 Surrogate optimization to search for robust, high-yield implosions

Surrogate models learn to emulate computationally expensive simulations by interpolating between the available simulation data points. Several machine learning algorithms are considered for the ICF surrogates, but due to the high volume of data and the presence of many cliff- and peak-like features that are not well resolved, most algorithms are unable to accurately fit the data. The random forest algorithm [13], which consists of an ensemble of decision trees trained on bootstrapped samples of the dataset, displays the best performance for a variety of quantities of interest (QOI), as shown in Fig. 2.1. The random forest surrogate for the logarithm of the total energy yield ($\log_{10} Y$) achieves a ten-fold cross-validated mean prediction error of 8%. (Explicitly, the surrogate model is trained on ten random subsets 80% of the data and is tested on the remaining 20%: the mean error on the prediction for the 20% random hold-out points is 8%). We can also build surrogates for other physical QOI in the database, such as DT fuel areal density (ρR), the first Legendre moments of the DT shell (P_{0-8}) and an ignition threshold factor metric [121] ITFX $\doteq \mathrm{Y}(\rho\mathrm{R})^2$ at the time of peak energy production ("bang time").

Once trained, the surrogate models can estimate QOI anywhere within the design space, allowing implosions that are not in the original dataset to be studied without running additional, expensive simulations. The surrogates are particularly well-suited for scanning the vast design space of simulation inputs to search for high performing implosions. For example, due to the difficulty of controlling laser drive symmetry in experiments, it is of great interest to locate a high yield "plateau" in design space – a region of igniting implosions that are robust to variations in drive and thus might be realistic candidates for experiments.

The yield surrogate lets us define a metric for quantifying robustness to drive

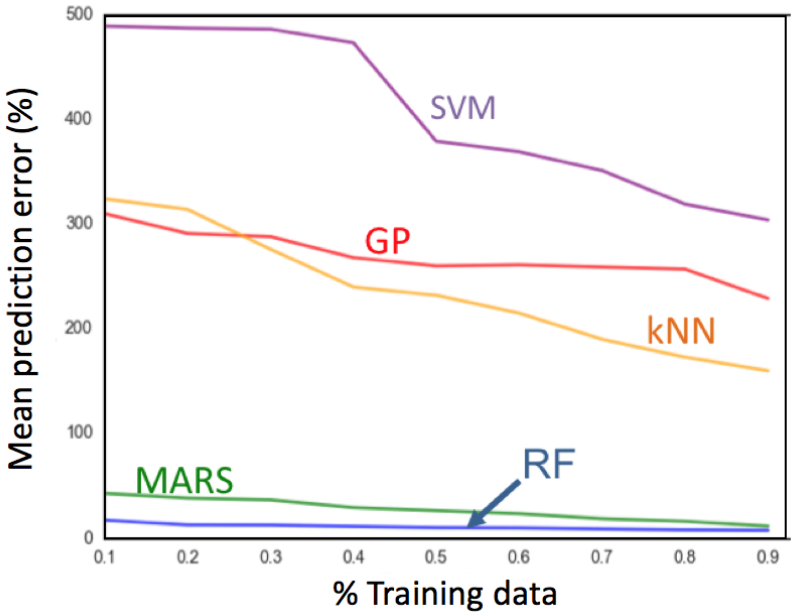


Figure 2.1: Comparison of several machine learning algorithms for fitting the yield data. The random forest (RF) shows significantly higher performance than the other algorithms– support vector machine (SVM), Gaussian process (GP), k-nearest neighbord regressor (kNN), and multivariate adaptive regression splines (MARS).

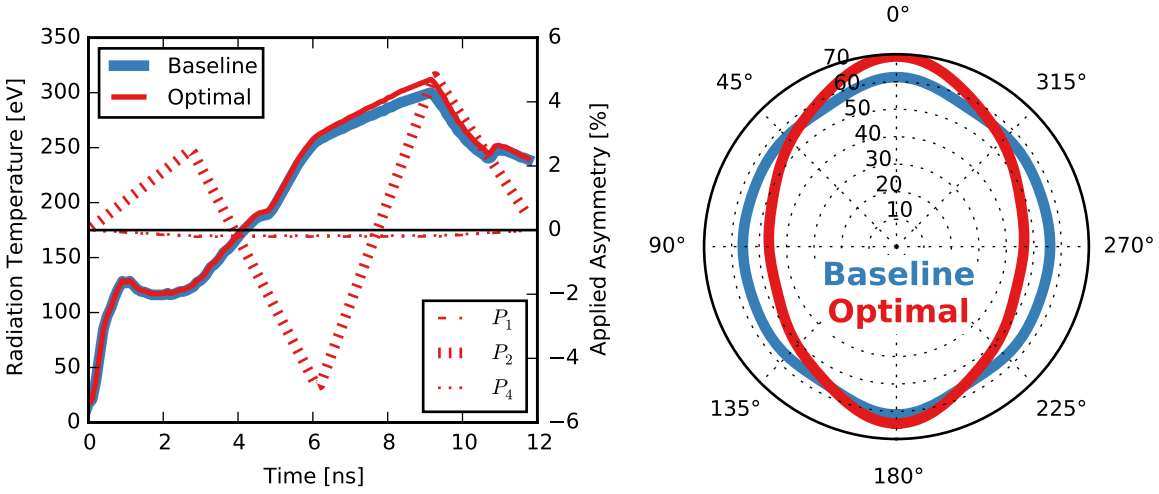
asymmetries that can locate such plateaus. For this measure, we pick a point in parameter space and with the surrogate make 1000 random input variations within a Latin hypercube centered at that location with side length Δ of 10% of the total sample space. The number of surrogate evaluations that achieve $Y > 1$ MJ within this volume serves as a local estimate of the probability of achieving high yield under variable conditions: $\mathcal{P}(Y > 1|\Delta = 0.1)$. This function serves as a smoothing operator on the yield, filtering out narrow "peaks" of high performance in favor of more broad "plateaus".

To locate a robust, high quality implosion, we define a performance metric for multi-dimensional optimization:

$$\mathcal{C} = 10\mathcal{P} + \text{ITFX}. \tag{2.1}$$

The first term in Eq. 2.1 finds broad areas of parameter space that ignite, and the second term finds locations that are high up the ignition cliff. We weight the first term higher (which has a maximum value of unity) to make it of similar order as the second term (which crosses the ignition threshold at 1, but can be over 10 for robustly burning designs). Furthermore, since our operational space is nine-dimensional and a single evaluation of $\mathcal{P}$ requires 1000 surrogate evaluations, we opt for a simplex based optimization algorithm [98] to avoid gradient evaluations in the search for a robust design.

The implosion which optimizes Eq. 2.1 is predicted to have fusion energy yield and ITFX of 15.23 MJ and 23.1 MJ$\cdot$(g/cm^2)^2, respectively, which can be compared to the baseline implosion of 15.08 MJ in yield and an ITFX of 16.9 MJ$\cdot$(g/cm^2)^2. The drive that produces the optimal implosion is shown in Fig. 2.2a (additionally, the optimal point has a 0.5$\times$ gas fill multiplier). Notably, this optimal point, which



(a) Capsule radiation drives

(b) Bang time fuel shapes (μm)

Figure 2.2: Comparison of the baseline and optimal radiation drives and surrogate-predicted bang time fuel shapes. Reproduced with permission from [106].

is predicted to robustly achieve high yield, has a time-varying P_2 applied drive asymmetry. Figure 2.2b compares the DT bang-time fuel shapes as predicted by the P_n surrogate models for both the baseline and optimal drives. Due to the time-varying asymmetry, the optimal drive's stagnated shape is predicted not to be a sphere, but an ovoid.

The surrogates also predict that the optimal ovoid implosion is more resistant than the symmetrically driven baseline to other perturbations. Figure 2.3 shows surrogate outputs for yield as the relative drive fluence $\int T_r^4 dt$ (normalized to the baseline) is varied. To eliminate the effects of the remaining input parameters, the optimal and symmetric implosions are compared with the same gas fill, P_1, and P_4 perturbations such that the change in performance is due to the P_2 drive alone. Both designs fall off in yield as the drive is reduced. However, while adding a -2.5% $P4$ to the baseline design requires a relative fluence of 1.1 for ignition, the optimal design

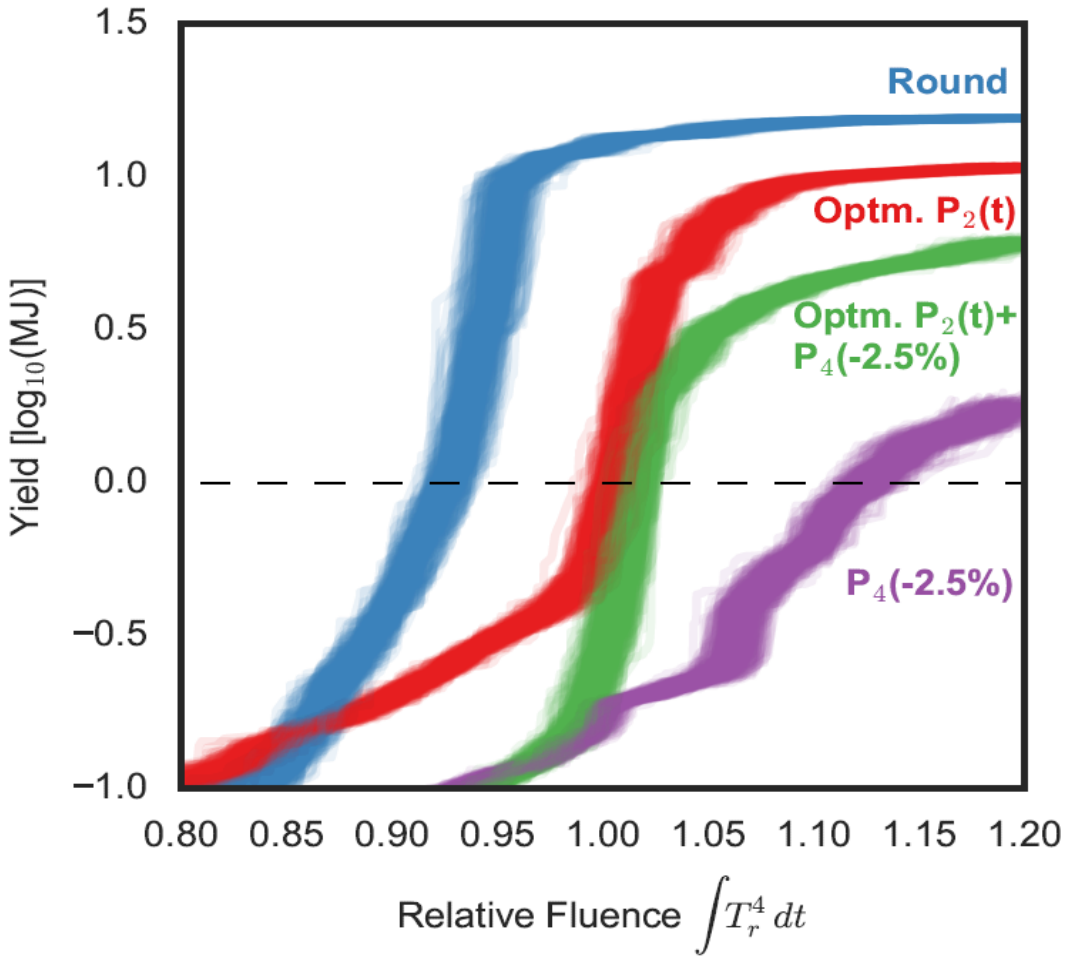


Figure 2.3: The surrogate prediction for yield under changing peak drive, as represented by the total normalized drive fluence $\int T_r^4 dt / (\int T_r^4 dt)_{baseline}$ for the baseline (round) and optimal ovoid (optm) cases. The optimal ignites with a relative fluence of slightly greater than 1.0, even under an applied P_4 perturbation. The baseline requires a relative fluence of more than 1.10 to ignite under an applied P_4. The multiple lines for each implosion represent variations in performance due to varying the relative amplitude of the drive during the trough, rise, and peak of the pulse, while ensuring that the drive integrates to the indicated relative fluence.

ignites with a relative fluence of around 1.01.

To investigate the movement of the ignition cliff further, yield contours predicted by the surrogates are shown in Fig. 2.4 for varying P_2 drives on the rise to peak power (P_2^P) and at the end of the peak radiation drive (P_2^P). The overlaying white contour lines correspond to the surrogate-predicted P_2 moment of the fuel radius at bang time, and the black point indicates the location of the optimal implosion. There is a broad, high yield ridge along a line compensating P_2^R and P_2^P drives, with higher yields favoring a negative P_2^R and positive P_2^P. This compensating drive does not result in a round implosion, but an ovoid with positive P_2, as shown by the bang time fuel P_2 contours. Under the addition of a P_1 perturbation, the high yield ridge contracts toward more extreme compensating drives which results in P_2 of up to 40%. The optimal implosion remains within the cliff boundaries while the round implosion falls to low yield. The high yield ridge appears to extend beyond the boundaries of the design space, suggesting that there may exist an implosion with higher performance than the optimal point if more extreme compensating P_2 drive perturbations are considered.

2.3 HYDRA simulations of the optimal implosion

To confirm the surrogate predictions of a robust ovoid implosion at a location not explicitly in the original simulation database, a series of 2D HYDRA simulations are performed for the optimal point and for a symmetric simulation with the same drive amplitude and gas fill, so that any differences are due solely to the time-varying drive asymmetry. To isolate alpha-particle bootstrapping from hydrodynamic effects, "burn-off" simulations with a reduced fusion cross-section are also performed.

Figure 2.5 illustrates how the ovoid shape arises from the applied time-dependent P_2 drive asymmetry. First, asymmetric shock bounce seeds vorticity in the gas.

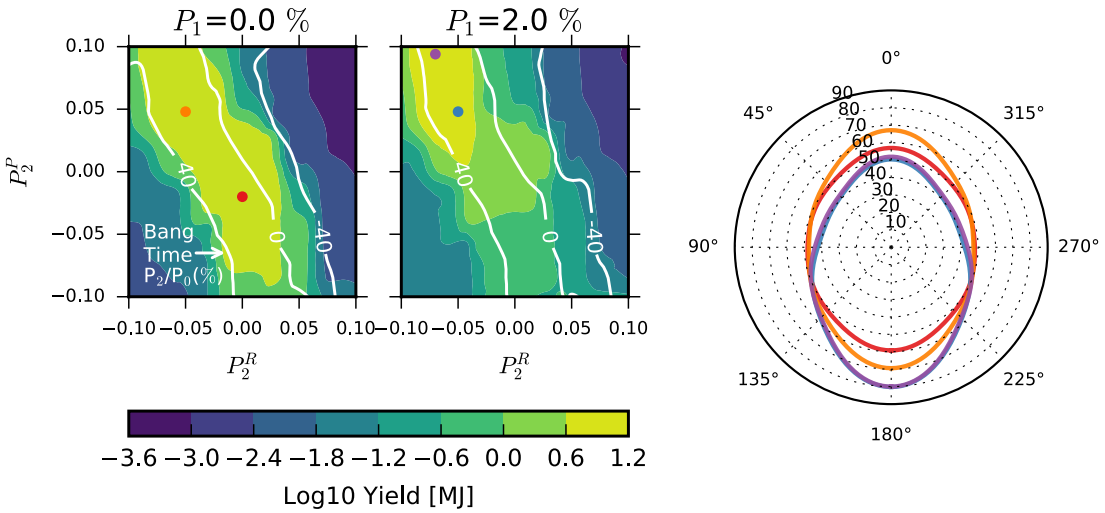


Figure 2.4: Contours of the surrogate prediction for the yield for varying P_2 on the rise to peak power (P_2^R) and at the end of the peak radiation drive (P_2^P) without (left) and with (right) an applied P_1 perturbation. The orange point indicates the optimal implosion. High yield implosions lie along a ridge of compensating P_2 drives (negative on the rise, positive at the peak), with the ridge shifting to require more extreme drives to reduce the effects of an applied P_1. Overlaying contour lines for the P_2 moment of the fuel at bang time illustrate that the P_2 drive that maximizes yield does not result in a round implosion, but an ovoid with positive P_2. The polar plot on the right illustrates the shape of the fuel shell at stagnation for the implosions indicated by the corresponding colored dots in the contour plot, confirming the implosions along the ridge of high yield are ovoids. Reproduced with permission from [106].

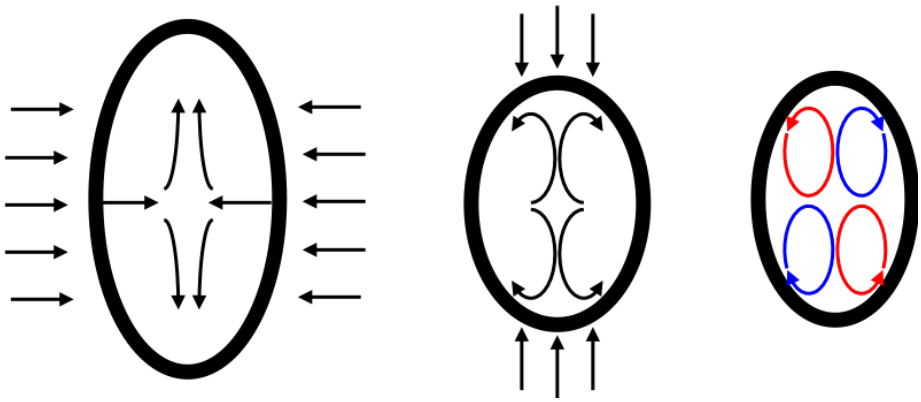


Figure 2.5: Generation of zonal flows within the capsule as a result of the time-dependent asymmetric radiation drive. During the rise to peak power, the negative P_2 drive compresses the capsule along the equator, forming axial gets as the gas meets on axis (left). At peak power, the positive P_2 drive squeezes the capsule on the poles, preventing the jets from escaping (center) and causes the flow to circle in on itself, forming coaxial, counter-propagating vortex rings (right).

Then the negative P_2 drive on the rise to peak power squeezes the capsule along the equator; as the compressed gas meets on axis, it forms axial jets (left of Fig. 2.5). At peak power, the positive P_2 drive squeezes the poles of the capsule, preventing the jets from escaping (center of Fig. 2.5). The flow circles on itself, forming two co-axial counter-propagating vortex rings (right of Fig. 2.5). Figure 2.6 illustrates the ovoid implsoion at stagnation. The exterior shell conforms to the vortex rings, forming an ovoid, and the central gas is trapped in a vorticity quadrupole. The hotspot is elongated, and does not align with the high-pressure central core. Strong coherent flows exist throughout the hotspot, so that the cold dense shell on the equator accretes into the central high-pressure region, burns, and exhausts via the poles.

The flows appear to nonlinearly suppress the growth of hydrodynamic instabilities. Figure 2.7 shows the upper right section of the stagnating shell for a burn-off

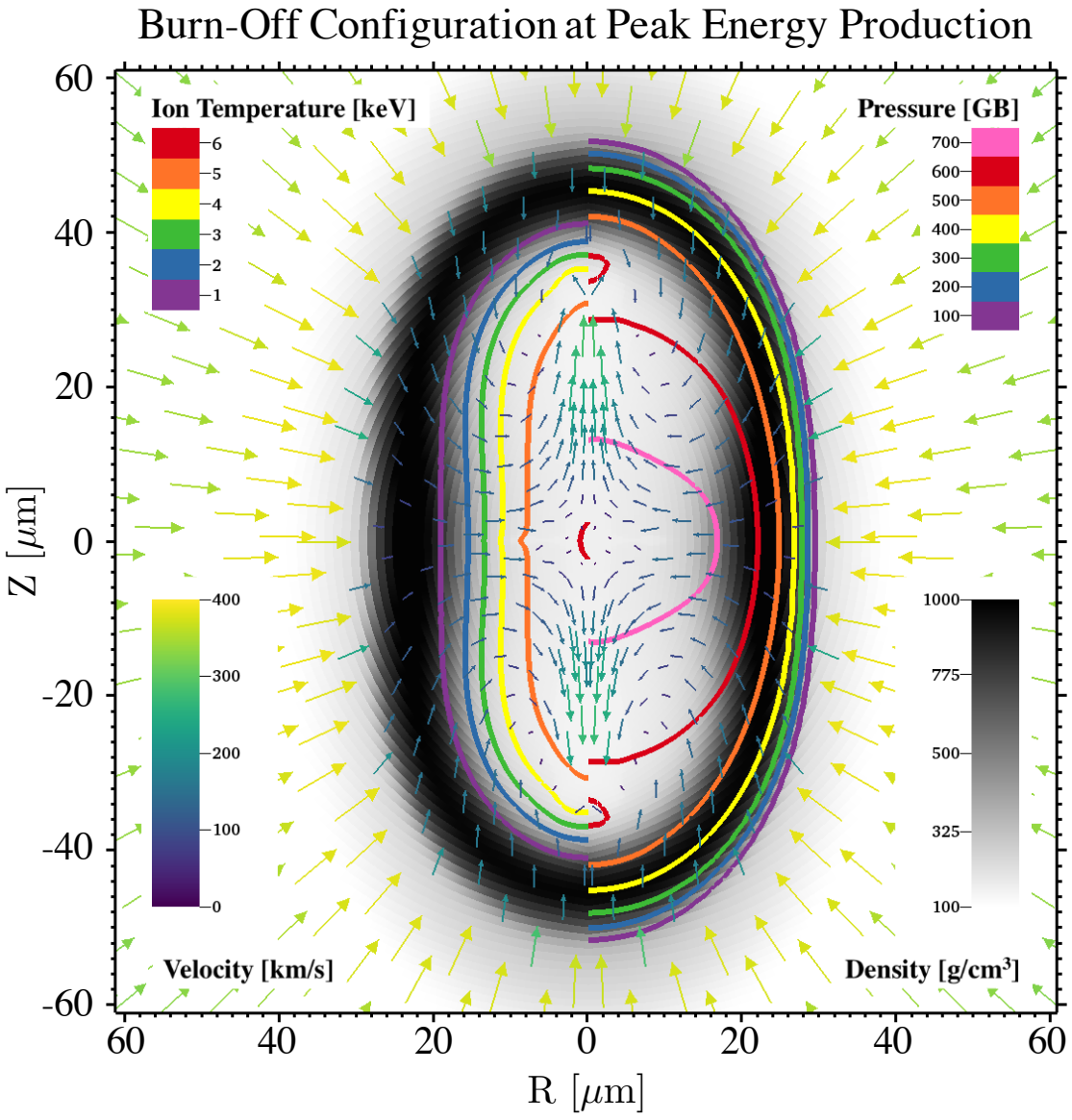


Figure 2.6: Burn-off ovoid implosion at the time of peak energy production. The arrows indicate the velocity field, illustrating the formation of counter-propagating coaxial vortex rings within the fuel shell. The colored contours indicate the ion temperature on the left, and the pressure on the right. The density of the fuel shell is shown in black; the shell is thicker near the equator, where fuel accretes into the hotspot, burns, and leaves via the poles. Reproduced with permission from [106].

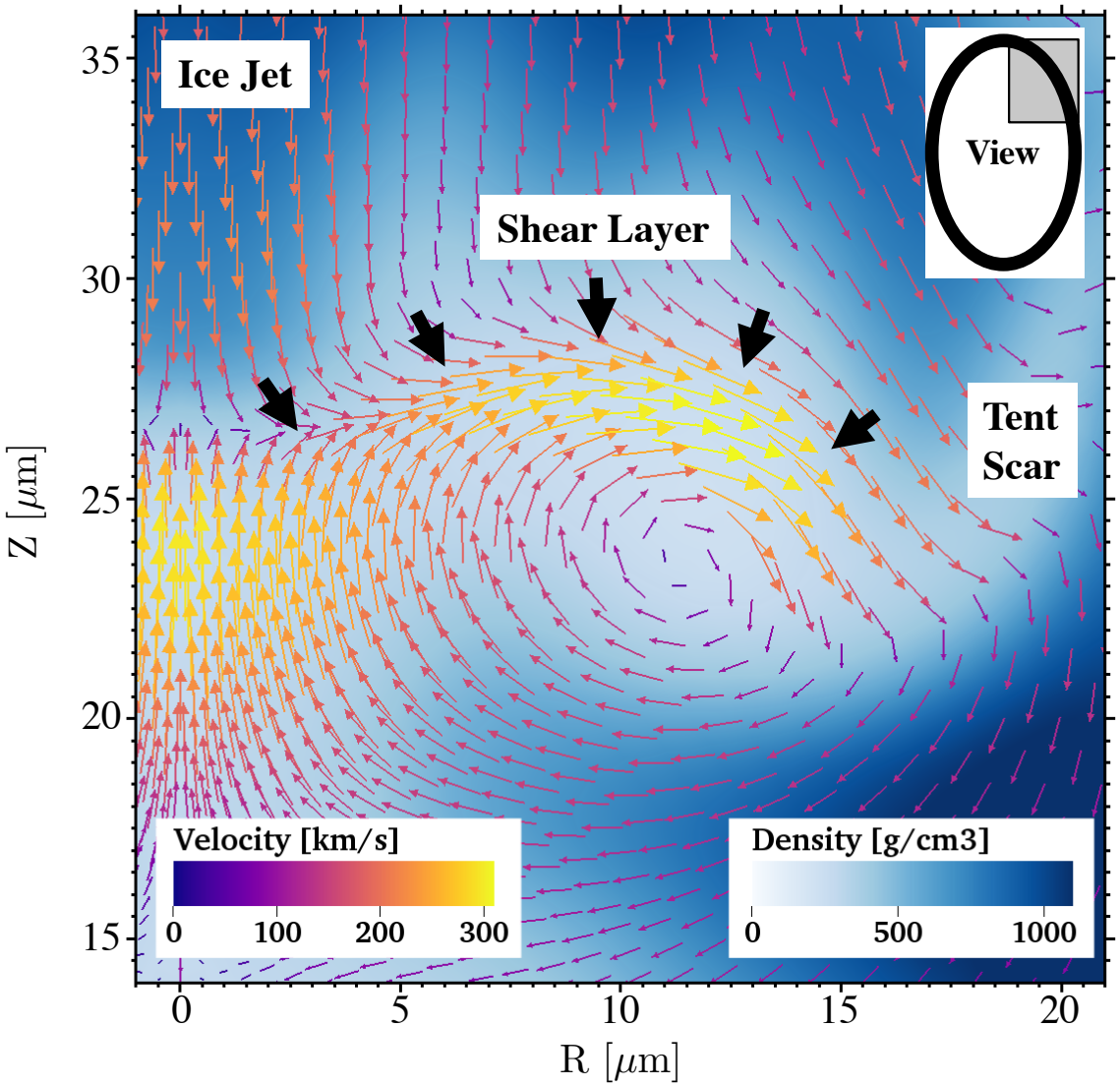


Figure 2.7: Density and velocity fields in the upper part of the stagnating shell for an ovoid burn-off HYDRA simulation perturbed with ice layer roughness and capsule support tent membrane. The background flows set up a high-velocity shear layer (thick arrows) that mitigates the effects of the perturbations during stagnation. Reproduced with permission from [106].

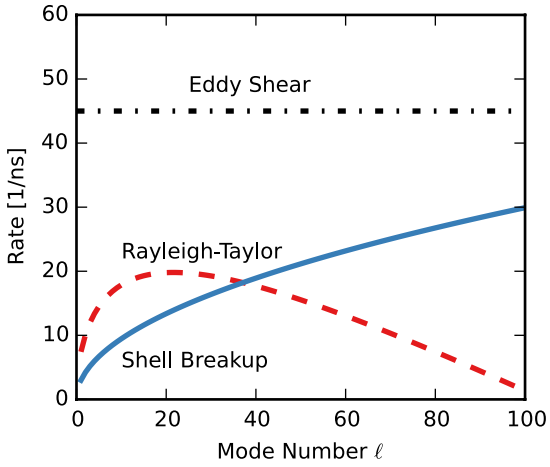


Figure 2.8: The local shearing rate of the eddy in Fig. 2.7 is larger than both the deceleration Rayleigh-Taylor growth rate (Eq. 2.2) or the shell breakup rate (Eq. 2.3). Reproduced with permission from [106].

ovoid implosion with added performance degradation sources: ice surface roughness (at NIF ignition specifications [51]) and a capsule support tent (calibrated to a 100 nm tent on an HDC capsule [143]). The roughness on the inner ice surface seeds a large on-axis jet directed down toward the central hotspot, while the capsule support tent creates a visible low-density "scar" on the shell. The background flows meets the axis jet head-on, forming a high velocity (> 300 km/s) shearing layer that directs the jet away from the hotspot. The same shear layer directs the flow field tangential to the tent scar, reducing convective loss through the hole.

The shearing in Fig. 2.7 appears strong enough to compete with shell distortions that occur during deceleration. The local shear rate can be estimated as the ratio of the local velocity to the eddy size. The eddy in Fig. 2.7 is roughly 5 μm across with an average velocity of 225 μm/ns, which gives a shearing rate of 45 ns^{-1}. Perturbations on the shell evolve at a characteristic rate that can be estimated as either the RT

growth rate or the inverse of the shell breakup time τ [62]. The RT growth rate is given by Eq. 2.2:

$$\gamma_{RT} = \alpha\sqrt{\frac{kg}{1+kL_m}} - \beta k v_a, , \tag{2.2}$$

where k is the wavenumber of the perturbation on the inner surface of the shell with characteristic density scale length L_m, v_a is the local ablation velocity, g is the deceleration, and $\alpha \simeq 0.9$ and $\beta \simeq 1.4$. The shell breakup time is estimated by:

$$\tau = \sqrt{\frac{2\pi R\left(\rho R\right)_{shell}}{l P_{stag}}}, \tag{2.3}$$

for some mode number $l = kR$ on a shell with areal density $\left(\rho R\right)_{shell}$ stagnating against a hotspot with pressure P_{stag}. Figure 2.8 shows that the eddy shear rate is larger than both rates predicted by Eqs. 2.2 and 2.3, suggesting that the shear flows present in the ovoid are potentially strong enough to impact the growth of shell perturbations during capsule deceleration.

Figure 2.9 shows contours of yield > 1 MJ for the round and ovoid implosions with varying levels of applied P_4 asymmetry and tent amplitude. The performance of both implosions falls off with increasing perturbation strength, but the ovoid implosion maintains high yield for a larger parameter range. For instance, the ovoid produces > 9 MJ with a 300 nm tent and $+3\%$ P_4, where the round implosion fails to ignite.

In all, these new simulations confirm the surrogate predictions of an asymmetric ovoid implosion that is more resilient to perturbations than symmetrically driven one-dimensional designs. The resiliency of the ovoid implosions is attributable to the generation of large scale zonal flows at stagnation, that protect the hotspot from engineering and asymmetry perturbations that are observed to degrade the performance of spherical implosions.

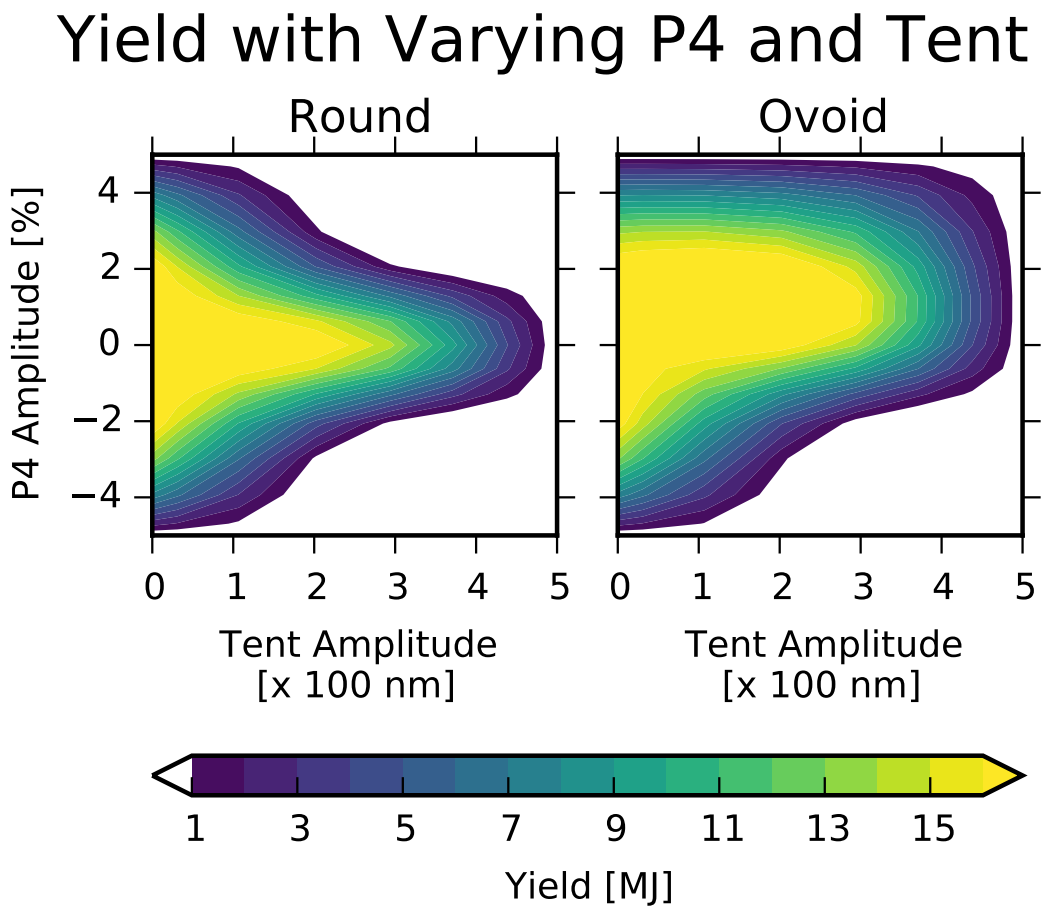


Figure 2.9: Contours of yield for the round and ovoid implosions under a combination of P_4 drive asymmetry and tent perturbations. Reproduced with permission from [106].

2.4 Conclusions

A machine learning algorithm trained on the largest ensemble of ICF implosion simulations led to the discovery of a new class of robustly igniting implosions. These implosions are ovoid in shape, challenging the well-accepted notion that spherical implosions uniformly outperform aspherical ones. The ovoids are intentionally driven by strong time-varying drive asymmetries, which serve to set up large-scale zonal flows at stagnation, making them more robust than spherical implosions to perturbations from the laser drive and shell imperfections, due to locally strong shearing rates induced by the flows.

Additional work must be completed to evaluate the possibility of achieving an ovoid implosion in experiments. The simulations in this study were 2D capsule-only simulations, subject to axisymmetric constraints; exploring the stability of the flow configurations in 3D capsule simulations is a priority. If the ovoids are robustly stable in 3D capsule simulations, we will move toward integrated hohlraum simulations to aid in determining the experimental requirements necessary for achieving the desired time-varying P_2 radiation drive asymmetry. It may be possible to experimentally create these implosions with current capabilities at the NIF. Near-vacuum hohlraums (NVH) [9], which are designed to counter a late-time polar drive with early time equatorial drive, may already operate near this regime (indeed similar flow patterns have been observed in some integrated NVH simulations of current NIF designs [8]). Mapping out the experimental signatures of zonal flow dominated hotspots is therefore another priority.

The ovoid implosions challenge decades of research that suggest drive asymmetries are detrimental to implosion performance, and present a new approach to achieving ignition in experiments. The machine-learning guided discovery and characterization

of the ovoids suggests a new paradigm for understanding and exploring complex systems via modern data analysis techniques.

3. DEEP JOINTLY-INFORMED NEURAL NETWORKS [1]

The machine learning-guided discovery of the ovoid implosion is an illustrative example of the benefits of augmenting physics studies with modern data analysis. While the random forest machine learning algorithm produces models that are accurate enough for tasks such as design optimization, this project illustrated several shortfalls of black-box machine learning methods. In particular, the random forests struggled to accurately model high gradient features in the response surfaces, such as the "ignition cliff" in which the yield rapidly changes by several orders of magnitude.

The applications in which we wish to use surrogate models (post-shot modeling of experiments, Bayesian calibration, and more) require the models to accurately capture the ignition cliff. Furthermore, we are interested in propagating and quantifying uncertainties, and it is thus important to consider surrogate models which estimate prediction uncertaintity due to the sparsity of training data. We turn our attention to deep neural networks: models that are flexible enough to capture highly complex response surfaces, and which can be readily extended into an approximate Bayesian framework via the recent advancements with dropout and other regularization techniques [34].

3.1 Introduction

Deep neural networks are quickly becoming one of the most popular tools in machine learning due to their success at solving a wide range of problems– from language translation [145, 2], to image recognition [116, 46, 77], to playing Atari [83, 36]. Neural networks trained via supervised learning are capable of discovering subtle

[1] Reproduced from "Deep Neural Network Initialization With Decision Trees', K. D. Humbird, J. L. Peterson. R. G. McClarren, IEEE TNNLS 10.1109/TNNLS.2018.2869694 (2018), with the permission of IEEE.

relationships between variables that make them well-suited for creating "surrogate" models for complex physical systems. Surrogate models approximate complicated response surfaces by interpolating between a set of sparse data that is typically expensive to acquire. The models provide a method for studying a continuum of designs rapidly, without resorting to costly computer simulations or experiments. Many machine learning algorithms can be used to create surrogates, but neural networks offer several distinct advantages: they are scalable to large volumes of high dimensional data, have low memory demands, and can be readily updated as new data becomes available.

Despite the flexibility of neural networks, the application of deep learning to study physics-based problems has been slow to increase in popularity. In part, the limited use of neural networks by non-experts is due to the difficulty of training an accurate model. There are an infinite number of design options, including the activation function, learning rate, regularization methods, and the network architecture: the number of hidden layers and the number of neurons in each layer. Often, changes in these settings can yield wildly different results. Datasets of interest for physics-based systems are often from high dimensional design spaces that are under-sampled and represent complex, nonlinear processes. The choice of neural network architecture for such data can significantly impact the training efficiency and accuracy of the model, and there exist few guidelines for determining appropriate settings that are robust across a multitude of problems.

In many cases, simple machine learning algorithms can produce reasonably accurate surrogate models with minimal effort from the user. For example, decision tree-based algorithms, such as random forests or extremely randomized trees, have been successful at modeling a variety of physics-based datasets [13, 17, 38]. Tree-based models are robustly accurate and have few hyper-parameters that need to be

tuned, making them convenient "black box" algorithms. However, traditional trees are confined to on-axis splits, limiting the accuracy of the model, and the memory demands for storing an ensemble of trees is high for complex data.

To create a black box neural network, the user-friendly features of tree-based models can be combined with the accuracy, flexibility, and scalability of deep neural networks. Several studies have explored the possibility of mapping decision trees and random forests to neural networks [115, 136, 6, 11, 142]. One particularly successful approach maps trees to equivalent two hidden layer neural networks, with the number of neurons in each layer related to the number of leaves in the decision tree [115, 11]. The mapping "warm starts" the neural network training process by initializing the network in a state that performs similarly to the decision tree; after additional training, the neural network achieves higher accuracy than the original tree-based model. Although the two hidden layer models perform well for moderately-sized datasets, the networks can become quite wide for high-dimensional nonlinear regression problems with complex decision trees, making subsequent training difficult for limited-size datasets.

While it is possible to fit any function with a sufficiently wide, shallow neural network [59], studies suggest that deep networks often perform better than wide networks with a similar number of neurons [43]. Including more hidden layers allows for higher levels of interaction between parameters, thus deep networks can discover nonlinear relationships not discernible with only two hidden layers. Based on this observation, we propose a novel mapping from decision trees to deep neural networks. The mapping produces a network with a specific number of hidden layers, neurons per hidden layer, and a set of initial weights that reflect the decision tree structure. The neural network is subsequently trained using back-propagation to optimize predictive performance. The algorithm is called "deep jointly-informed neural networks", or

DJINN, as the final neural network is informed by an underlying decision tree model and the standard training method of back-propagation.

In the following sections, DJINN is described in detail and compared to a variety of other neural network models for regression and classification datasets. In section 3.2, the algorithm for mapping from trees to initialized neural networks is presented and illustrated with a few examples. In section 3.3, DJINN is presented as a "warm start" method for training deep neural networks and is compared to other warm start and weight initialization techniques. Section 3.4 compares DJINN, which determines the neural network architecture based on the structure of a decision tree, to a Bayesian hyper-parameter optimization method for selecting an appropriate architecture. Although DJINN does not attempt to optimize the architecture of the neural network, it displays comparable performance to optimized architectures at a significantly lower computational cost. Overall, DJINN is observed to be a robustly accurate and user-friendly method for creating deep neural networks to solve a variety of classification and regression tasks.

3.2 Deep Jointly-Informed Neural Networks

The DJINN algorithm determines an appropriate deep neural network architecture and weight initialization that utilizes the dependency structure of a decision tree trained on the data. The algorithm can be broken into 3 steps: constructing the ensemble of decision trees, mapping from trees to neural networks, and fine-tuning the neural networks via back-propagation. In the following sections, each step is presented in detail and the mapping is illustrated with a few simple examples.

3.2.1 Decision tree construction

The first step of the DJINN algorithm is the construction of the decision tree-based model. This can be a single decision tree that will result in a single neural

network, or an ensemble of trees, such as random forests [13], that will produce an ensemble of neural networks. The depth of the trees is often limited to avoid the creation of excessively large neural networks; the maximum tree depth is a hyper-parameter that should be tuned for each dataset.

3.2.2 Mapping decision trees to deep neural networks

The DJINN algorithm chooses a deep neural network architecture and a set of initial weights based on the structure of a decision tree. The mapping is not intended to reproduce the decision tree, but instead takes the decision paths as guidance for the network architecture and weight initialization.

While neural networks are initialized layer by layer, decision trees are typically stored by decision path. The paths begin at the top branch of the tree, and follow the left, and then the right, side of every decision until a leaf (prediction) is reached. The manner in which trees are stored makes them difficult to navigate according to depth, but simple to traverse recursively. When mapping from tree to neural network, it is easiest if the structure of the tree is known before initializing neural network weights, thus the decision paths are recursed through twice: first to determine the structure, then to initialize the weights.

The primary branch of the tree is defined as the $l = 0$ level. The levels then increase from $l = [1, D_t]$ where D_t is the maximum tree depth, often specified by the user. The maximum branch depth is defined as $D_b = D_t - 1$, as the last level of a decision tree contains only leaves. The mapped neural network has D_t total layers: an input layer at $l = 0$, D_b hidden layers, and an output layer. The output layer contains one neuron per label for multi-label classification, or one neuron for single-output regression problems. Multi-output regression is accommodated by performing the mapping on multi-output decision trees [104], and including one neuron per target

variable in the output layer.

Algorithm 1 outlines the process of initializing the DJINN network for a single tree. If an ensemble method is desired, a random forest or extremely randomized tree model can be used, and the mapping is repeated for each tree to create an ensemble of neural networks.

The variance of the normal distribution used to initialize nonzero DJINN weights is $3/(n_{\text{prev}} + n_{\text{cur}})$, where n_{prev} and n_{cur} are the numbers of neurons in the previous and current hidden layers, respectively. Biases for each neuron are randomly sampled from the same distribution. This is a variant of the popular Xavier initializer [41, 133]. The variance of the distribution is designed to keep the scale of the gradients roughly the same in all layers of a deep neural network. Weights that are used to pass input variables or leaf values through the hidden layers are initialized to unity in order to preserve their value.

3.2.3 Optimizing the neural networks

Once the trees have been mapped into initialized neural networks, subsequent tuning of the weights is carried out using back-propagation. For the examples presented in the following sections, the neural networks are trained using Google's deep learning software Tensorflow [1]. The activation function used at each hidden layer is the rectified linear unit (ReLu), which generally performs well for deep neural networks [97, 23] and can exactly retain the values of neurons in previous hidden layers. The Adam optimizer [70] is used to minimize the cost function, which is mean squared error (MSE) for regression, and cross-entropy with logits for classification [24].

3.2.4 Examples mapping from trees to DJINN models

Figure 3.1 shows a simple example of a decision tree and the initialized DJINN neural network. Following the steps outlined in the algorithm, the mapping is per-

Algorithm 1 DJINN Tree to Neural Network Mapping

1: Recurse through paths of the decision tree:
- · Determine max branch depth (D_b)
- · Count number of branches at each level $N_b(l)$
- · Record max depth each input occurs as a branch: $L_i^{\max}$

▷ For a max branch depth D_b, there will be D_b hidden layers, an input layer with $\mathrm{N_{in}}$ neurons, and an output layer with $\mathrm{N_{out}}$ (regression) or $\mathrm{N_{class}}$ (classification) neurons in the neural network. Each hidden layer will have $n(l)$ neurons, where

$$n(l) = n(l-1) + N_b(l) \tag{3.1}$$

This "copies" the previous hidden layer and adds "new" neurons for each branch in the current level of the tree.

2: Create arrays W^l of dimension $n(l)$ x $n(l-1)$, l=1,...,D_b, and $W^{D_{b+1}}$ with dimension $n(D_b)$ x N_{out} (or N_{class}) to store initial weights. Initialize arrays to 0.

3: For each input i=0,1,...,N_{in}-1:
- · Set $W^l_{i,i} = 1$ for $l < L_i^{\max}$

▷ This ensures input values are passed through hidden layers until the decision tree no longer splits on them.

4: Recurse through decision paths of the tree:

For levels l=1,...,D_b:

For each node c in level l:

·Define p as the neuron created by the parent branch

If c = branch:

▷ According to Eq. 3.1, a new neuron has been added to layer l

· Initialize $W^l_{\mathrm{new},p} \sim \mathcal{N}(0, \sigma^2)$, connecting branch p and new neuron

· Initialize $W^l_{\mathrm{new},c} \sim \mathcal{N}(0, \sigma^2)$, connecting branch c and new neuron

If c = leaf:

· Initialize $W^l_{p,p} \sim \mathcal{N}(0, \sigma^2)$, l=l+1 ...D_b-1

· Initialize $W^{D_b}_{p,\mathrm{out}} \sim \mathcal{N}(0, \sigma^2)$

▷ Classification: out = neuron for the class

▷ Regression: out = output neurons

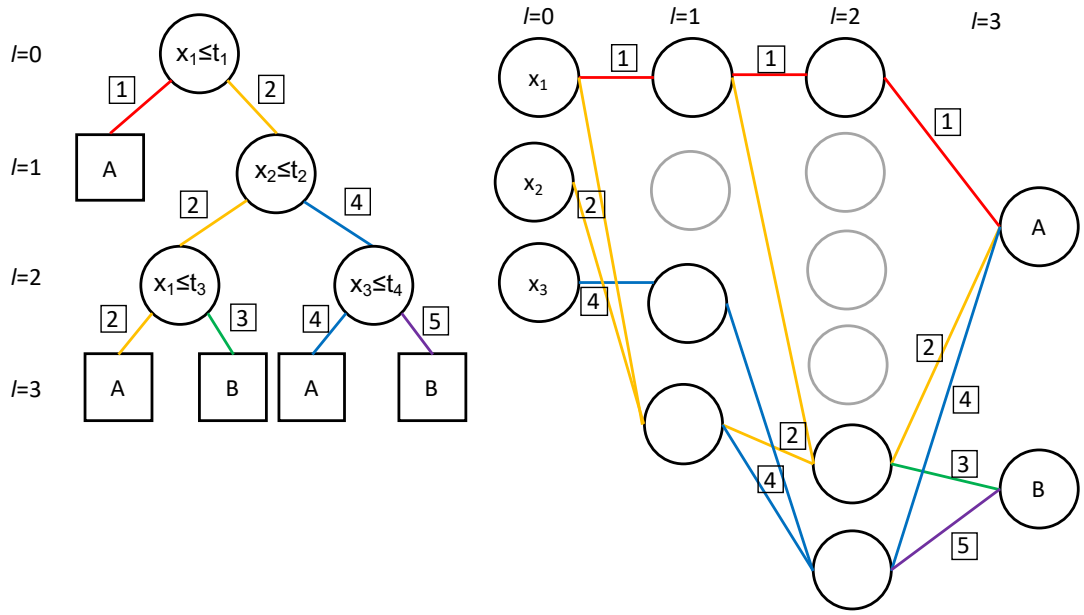


Figure 3.1: Illustration of the DJINN mapping for a simple decision tree. Gray neurons are initially unconnected; if biases are randomly initialized to negative values, the neurons cannot learn and are thus not included in the final DJINN architecture.

formed as follows:

1.
   - The maximum tree depth is $D_t = 3$, as indicated by the numbers $l = 0, ..., 3$.
   - There are (1, 1, 2, 0) branches in each level of the tree
   - The maximum depth at which each input occurs as a branch is given by: $L_i^{\max} = (2, 1, 2)$ for x_1, x_2, and x_3.

2. The neural network architecture is shown in Fig. 3.1; initially all weights are set to zero.

3. Set $W_{i,i}^l = 1$ for $l < L_i^{\max}$; this "retains" the input values through the hidden layers until they are no longer used as branches. In the figure, this step is

represented by the horizontal red (labeled 1) and blue (2) connections for x_1 and x_3, respectively.

4.
   - Start with leftmost branch (red, 1). This node is a leaf, which uses x_1 to determine if the output is class A. The horizontal red connections propagate the value of the parent, x_1, through the hidden layers to the output layer, then connect to class A.
   - Consider the yellow path (2) in the tree:
     - For l=1 the node is a branch splitting on x_2, with parent x_1. One of the "new" neurons in l=1 of the neural network represents this decision. Connect x_1 and x_2 to this neuron (yellow, 2).
     - For l=2 there is a branch splitting on x_1; connect the parent (new neuron in l=1) and x_1 to a new neuron in l=2 (yellow, 2).
     - For l=3 there are two leaves, connect the parent (new neuron in l=2) to class A (yellow, 2) and B (green, 3).
   - Move to the rightmost path of the tree:
     - The l=1 layer, which created a new neuron that accepts x_1 and x_2 in l=1 of the network, has already been mapped.
     - For l=2 there is a branch splitting on x_3; connect the parent (new neuron in l=1 of the neural network) and x_3 to a new neuron in l=2 (blue, 4).
     - For l=3 there are two leaves; connect the parent (new neuron in l=2) to class A (blue, 4) or B (purple, 5).

In step 4, all "connections" are non-zero weights initialized from the Xavier normal distribution as described previously, unless already initialized to unity. Qual-

itatively, the algorithm maps decision paths in the tree to decision paths through the network. Neurons which are not initially connected are randomly included in the final architecture; all biases are randomly initialized from a normal distribution, thus neurons with positive biases can be trained. The inclusion of extra degrees of freedom allows for the neural network to correct for inaccuracies in the decision tree during training.

As decision trees are sequences of logical operations, further insight into the mapping can be gained by considering how DJINN initializes networks to solve simple logic problems. Figure 3.2 illustrates the decision tree and DJINN mapping for three logic operations that have unique decision tree structures: the IF, OR, and XOR statements. The connections in the initialized DJINN networks indicate nonzero weights and all biases are random. Gray neurons represent those that are not initially connected, but could be included in training if randomly assigned a positive bias.

For the IF x statement, the tree contains a single decision based on the value of x. DJINN reproduces this decision path by connecting the input x to either 0 or 1; knowing the value of x alone is enough to solve the problem. Two decisions are needed to solve the OR problem: if $x \geq 0$ the answer is 1, otherwise it needs to also consider the value of y. If $x < 0$ and $y \geq 0$ the answer is 1, otherwise the answer is 0. In the DJINN mapping of this tree, the value of x is passed directly to the output class 1, as shown by the red connections; mimicking the left side of the tree. To mimic the right side of the tree, both the values of x and y are passed to the last neuron in the hidden layer, which is then connected to classes 0 and 1. The XOR operation requires knowledge of x and y to determine the correct class. The DJINN initialization has two hidden neurons that receive both x and y, which then connect to the output layer. For the OR and XOR problems, the gray neurons that are randomly included can correct for errors in the decision tree. For simple logic

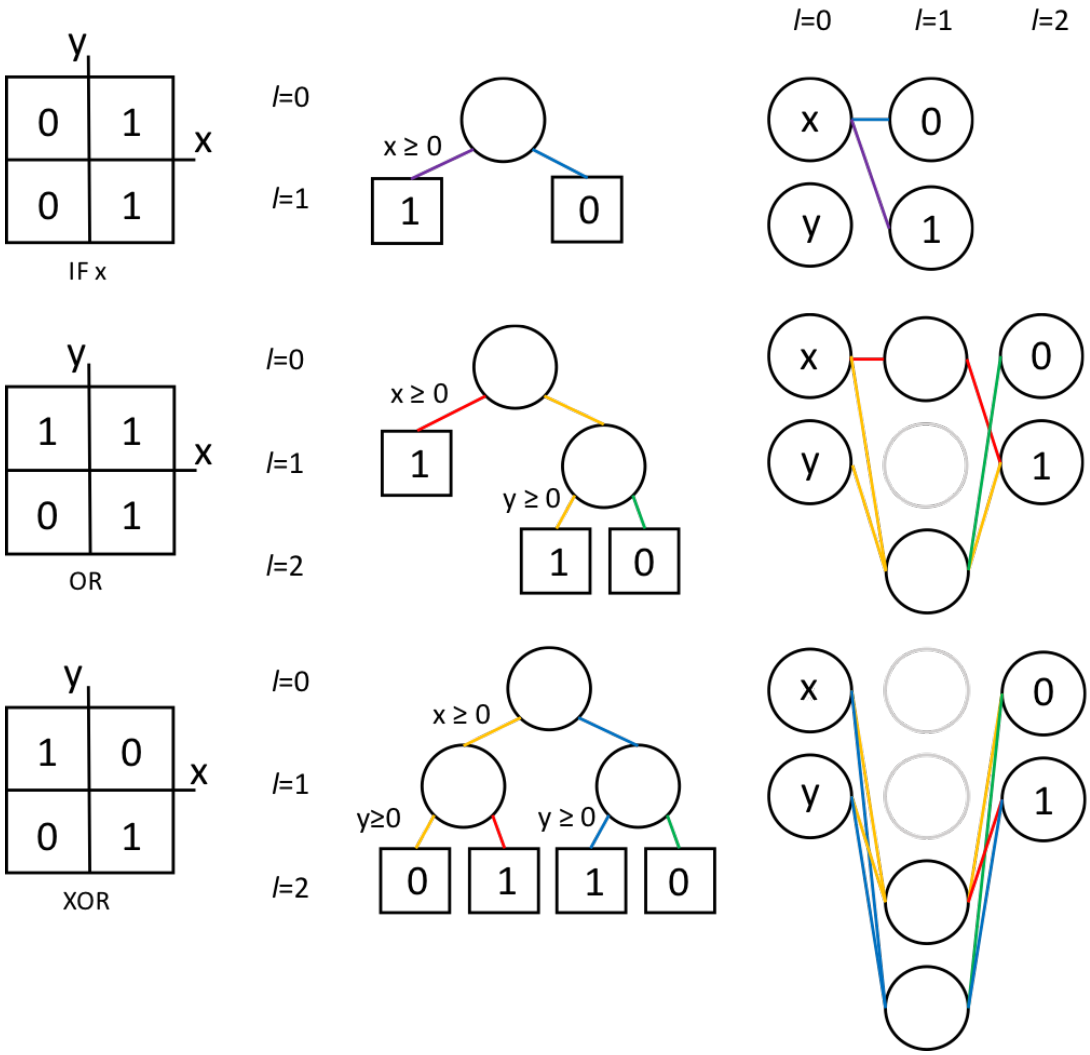


Figure 3.2: Truth tables, decision trees, and DJINN-initialized neural networks for logical operations IF(x), OR, and XOR. Decision paths in the tree are mapped to paths through the neural network, indicated by color. Gray neurons are initially unconnected; if biases are randomly initialized to negative values, the neurons cannot learn and are thus not included in the final DJINN architecture.

operations, the presence of additional neurons is not necessary, but for complicated problems the decision tree is often too simple to accurately model the data.

Since decision trees are a series of logical operations, DJINN initialization can also be viewed as such. When a branch splits into two additional branches, there is an XOR-like decision; when a branch splits into a branch and a leaf there is an OR-like decision, and when a branch splits into two leaves, there is an IF-like decision. This is illustrated in Fig. 3.1: the first decision is OR-like– the red and yellow connections in the neural network from Fig. 3.1 match those in Fig. 3.2. The next decision is XOR-like– the yellow and blue connections between l=1 and l=2 in Fig. 3.1 match those from the XOR in Fig. 3.2. Finally, there are two IF-like decisions, which connect the neurons from the final hidden layer to the outputs as shown by the blue/purple and yellow/green connections in Fig. 3.1.

Currently, the thresholds of the logical operations are tuned during training; a potential path for improving the algorithm is to encode the decision tree thresholds into the neural network initialization procedure.

3.3 DJINN Performance

The ease of use of the DJINN algorithm makes it an attractive method for general researchers to create neural network-based surrogate models for complex datasets. Unlike hyper-parameter optimization algorithms used to design neural networks [111, 150], DJINN does not require expensive searches through high-dimensional parameter spaces in order to determine a suitable neural network architecture and weight initialization.

In the following sections, the performance of DJINN is compared to alternative methods for neural network design and initialization for a variety of regression and classification datasets. In section III A, the benefits of using DJINN as an ensemble

method are explored, followed by a comparison to shallow neural networks initialized from decision trees in section III B. In section III C, the importance of the initial topology of the DJINN weights is illustrated by comparing DJINN to other initializations: densely connected topologies, and sparsely-connected initial weights that do not leverage the dependency structure of the data learned by a decision tree. The DJINN initialization is shown to provide a warm-start to the training process for a variety of datasets, allowing the models to achieve higher predictive performance than non-informative initialization techniques in a fixed amount of training time.

3.3.1 DJINN as an ensemble method

The DJINN algorithm maps a decision tree to a deep neural network with an architecture and initial weights that reflect the dependency structure of the data learned by the tree. In practice, ensembles of decision trees, such as random forests [13] or extra-trees models [38] often exhibit significantly higher performance than individual decision trees. In the ensemble approach, each tree is trained on a random subset of the data and gains complementary knowledge about the relationship between the input and target variables. Each tree makes its own prediction for the target variables, and the model reports the mean prediction of the ensemble. Increasing the number of trees in the ensemble improves predictive performance up to some maximum number of trees, at which point the model begins to over-fit to the training data.

Similar to the random forest from which DJINN is mapped, the performance of DJINN improves as the number of trees included in the ensemble increases. Figure 3.3 plots the predictive performance of DJINN as the number of tree-initialized neural networks increases; the mean squared error (MSE) is normalized by the MSE of the single-tree model. The bold line shows the mean value from a five-fold cross-

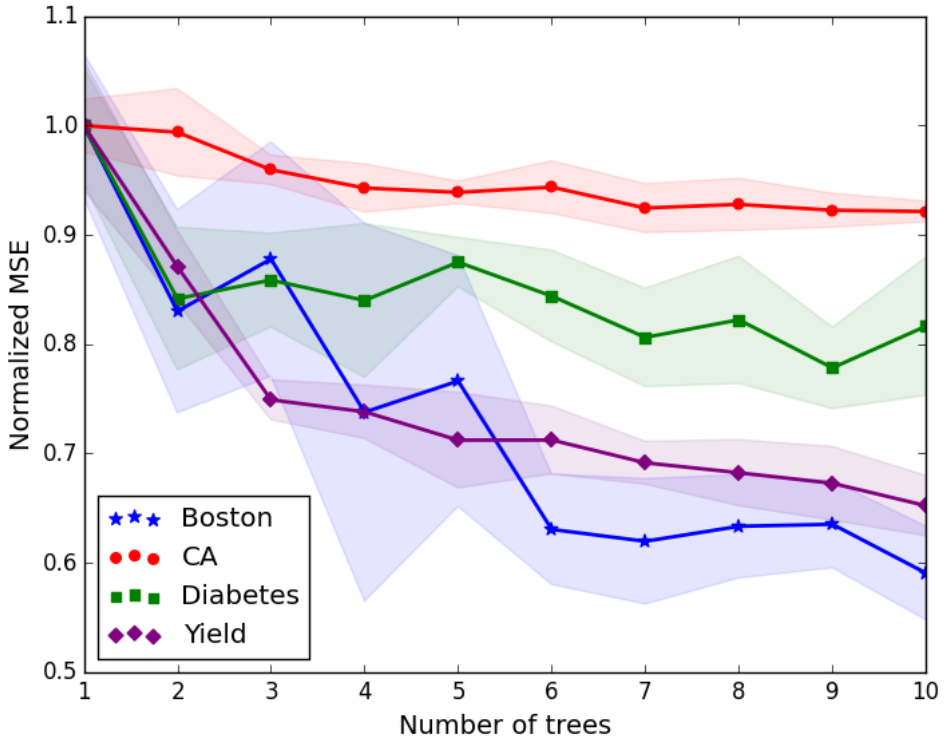


Figure 3.3: MSE (normalized by the MSE of one tree) as a function of the number of trees included in the DJINN ensemble for various regression datasets. The performance of the model improves as the number of trees is increased.

validation score, and the error bars represent the standard deviation of the score. The cross-validation is performed by randomly splitting the data into training (80%) and testing (20%) groups with a fixed random seed, such that each model sees the five same permutations of training and testing data.

Included in Figure 3.3 are three standard regression datasets: California housing prices [102], Boston housing prices [52], and diabetes disease progression [29]. DJINN is also tested on a novel database of inertial confinement nuclear fusion (ICF) implosion simulations [74] from Chapter 2. The ICF dataset consists of 46,416 points Latin

hypercube sampled from a nine-dimensional input space. The output of interest is the yield: the thermonuclear energy produced in the implosion. The yield response surface has proved challenging to fit with common machine learning algorithms [106], as there are many nonlinear cliff- and peak-like features that are not well resolved by the data. A 300-tree random forest regressor [13], mentioned in Chapter 2, proved to be the most successful model, with a mean prediction error of approximately 10% and an explained variance score of 0.92.

The models are trained with fixed hyper-parameter settings, summarized in Table 4.1. Features in each dataset are scaled between [0,1] prior to training, but performance metrics such as MSE and mean absolute error (MAE) are reported in unscaled units, unless otherwise noted.

For each of the regression datasets, the error of the DJINN model decreases with the number of trees included in the model; this behavior is characteristic of the random forest from which DJINN is mapped. In tree-based ensemble methods, there is typically a minimum number of trees that achieves low prediction error; adding more trees yields diminishing improvements in performance, and eventually leads to over-fitting. In the following sections, DJINN is always evaluated as an ensemble method with ten trees per model.

Ensemble methods are becoming popular for various neural network applications; in particular, recent work has shown that attention models exhibit improved performance when treated as ensembles. Attention models are popular for exploiting dependencies between variables, particularly for time series and sequence data [5, 140, 82]. Rather than using a single attention model, it has been observed that using an ensemble of models, in which each model is initialized with a different structure to extract complementary information from the data, often leads to superior performance [120]. This is analogous to the improvements seen in DJINN, in

Table 3.1: Neural network hyper-parameters used for each dataset.

Dataset	# Epochs	Learn. Rate	Batch Size	Max. Tree Depth
Boston housing	300	0.006	21	5
CA housing	200	0.006	826	5
Diabetes	50	0.0001	1	5
ICF Yield	300	0.008	1857	5
Iris	100	0.006	6	3
Digits	300	0.003	72	3
Wine	50	0.004	8	3
Breast cancer	100	0.006	7	4

which each neural network is initialized with a different dependency structure learned by the trees in the random forest. The importance of this dependency structure will be emphasized in the next sections.

3.3.2 Comparison to shallow tree-initialized neural networks

Other algorithms for mapping decision trees to two-hidden layer neural networks are observed to act as "warm-starts" to neural network training [115, 11]. In the first two rows of Table 3.2, DJINN is compared to two-hidden layer networks for the regression datasets presented in Figure 3.3. The first two rows of Table 3.3 show the performance of the models for four standard classification datasets: the iris flower [31], digits [64], wine [32], and breast cancer [127]. DJINN and the two-hidden layer model (abbreviated 2HL) are evaluated as ensemble methods; the trees are mapped from ten-tree random forests, and the ensemble prediction is the mean of the ten individual predictions. The networks are trained with the hyper-parameters summarized in Table 4.1 on five fixed permutations of training and testing data to produce cross-validation scores. The performance metrics include MSE, mean absolute error (MAE) and explained variance (EV) for regression, and recall, precision, and accuracy for classification. Student's t-tests between the MSE values for DJINN and the

Table 3.2: Model performances for regression tasks. The mean and standard deviation of five-fold cross-validation metrics are reported for each model. The p-value is computed with a Student's t-test between the test MSE values for DJINN and the other models. Bold blue values highlight comparisons in which DJINN has a lower error than the other method with $p<0.05$; bold red values highlight when DJINN has higher error than the other method with $p<0.05$.

	Boston				CA Housing			
Model	MSE	MAE	EV	p	MSE	MAE	EV	p
DJINN	7.289±1.541	1.840±0.105	0.915±0.014		0.233±0.011	0.318±0.006	0.826±0.010	
2HL	8.393±4.568	1.965±0.346	0.903±0.059	0.622	0.307±0.009	0.381±0.008	0.766±0.007	**3.110E-06**
Random-Dense	8.440±1.897	1.906±0.101	0.901±0.020	0.323	0.247±0.011	0.327±0.007	0.816±0.009	**0.004**
Random-Sparse	7.326±0.707	1.898±0.062	0.914±0.009	0.962	0.270±0.006	0.347±0.009	0.798±0.006	**2.009E-4**
Bayesian Opt.	7.556±0.815	2.034±0.068	0.910±0.007	0.740	0.305±0.011	0.377±0.012	0.772±0.006	**8.470E-06**

	Diabetes				Yield			
Model	MSE	MAE	EV	p	MSE	MAE	EV	p
DJINN	3154±339.9	43.391±2.006	0.455±0.100		0.018±0.002	0.063±0.003	0.990±0.001	
2HL	3108±153.3	43.456±1.381	0.421±0.043	0.787	0.031±0.005	0.088±0.012	0.983±0.003	**8.380E-4**
Random-Dense	3414±266.5	44.704±1.716	0.383±0.055	0.215	0.021±0.001	0.067±0.003	0.989±0.001	**0.045**
Random-Sparse	3045±188.5	43.783±1.268	0.461±0.061	0.547	0.049±0.007	0.111±0.011	0.973±0.003	**9.880E-06**
Bayesian Opt.	2376±107.1	38.895±1.519	0.584±0.044	**0.001**	0.023±0.003	0.081±0.008	0.988±0.001	**0.020**

2HL model give the p-value listed in the final column for each dataset.

DJINN has consistently higher predictive performance than the two hidden layer model for the regression datasets; the p-values indicate the improvements of DJINN are statistically significant for two of the four datasets. DJINN often achieves slightly higher predictive accuracy for classification tasks, but the improvements over the 2HL model are not statistically significant.

In general, the performance of DJINN is comparable to existing methods for mapping trees to initialized neural networks for simple datasets, but has higher predictive accuracy for regression tasks. As the complexity of the data increases, it is expected that the deep structure of DJINN will have advantages over the wide, shallow networks, which tend to require more data and time to train [43].

Table 3.3: Model performances on classification tasks. The mean and standard deviation of five-fold cross-validation metrics are reported for each model. The p-value is computed with a Student's t-test between the test accuracy values for DJINN and the other models. Bold blue values highlight comparisons in which DJINN has a lower error than the other method with $p<0.05$; bold red values highlight when DJINN has higher error than the other method with $p<0.05$.

	Iris				Digits			
Model	Recall	Precision	Accuracy	p	Recall	Precision	Accuracy	p
DJINN	0.987±0.020	0.980±0.029	0.983±0.025		0.973±0.010	0.977±0.008	0.976±0.009	
2HL	0.950±0.052	0.959±0.045	0.959±0.045	0.144	0.971±0.015	0.971±0.015	0.972±0.015	0.549
Random-Dense	0.982±0.019	0.975±0.027	0.978±0.023	0.289	0.976±0.011	0.979±0.009	0.978±0.010	0.667
Random-Sparse	0.988±0.011	0.979±0.019	0.983±0.015	0.289	0.971±0.005	0.972±0.004	0.972±0.004	0.303
Bayesian Opt.	0.980±0.015	0.980±0.014	0.978±0.016	0.147	0.964±0.021	0.965±0.021	0.965±0.020	0.240

	Breast Cancer				Wine			
Model	Recall	Precision	Accuracy	p	Recall	Precision	Accuracy	p
DJINN	0.960±0.012	0.954±0.019	0.960±0.013		0.982±0.019	0.975±0.027	0.978±0.023	
2HL	0.965±0.016	0.961±0.027	0.972±0.021	0.291	0.981±0.020	0.977±0.027	0.978±0.023	1.000
Random-Dense	0.959±0.014	0.958±0.018	0.960±0.016	1.000	0.982±0.019	0.975±0.027	0.978±0.023	1.000
Random-Sparse	0.958±0.009	0.954±0.021	0.958±0.011	0.829	0.990±0.014	0.987±0.019	0.989±0.015	0.397
Bayesian Opt.	0.982±0.005	0.983±0.004	0.985±0.003	**0.003**	0.989±0.016	0.992±0.011	0.989±0.015	0.397

3.3.3 DJINN as a warm-start for neural network training

Many graph-based models, including decision trees and neural networks, are trained to learn dependency structures in the data. In unsupervised applications, relationships between features are used to find lower-dimensional representations of data [63]; in supervised learning, dependency structures relate the input data to output quantities of interest via a series of latent representations formed in the hidden layers of the network [73]. If an informative structure is initially imposed on the graph, the training process can be accelerated as the imposed relationships act as a warm-start. A common method to warm-start neural networks is the use of a previously trained model to initialize a new model that will be trained on similar, or additional, data. This type of warm-start is often used in transfer learning; it leverages previously-discovered relationships between the inputs, latent representations of the data, and the outputs to accelerate the training process [100].

The DJINN and 2HL algorithms leverage the dependency structure learned by a decision tree, which has been trained on the data, to warm-start the training of a neural network. By beginning the training process in a state that is primed with dependency information between the input and output data, the tree-based models often converge to a minimum cost in fewer training epochs than randomly initialized networks with the same architecture.

To illustrate the importance of the DJINN weight initialization, the algorithm is compared to other weight initialization schemes. There are two main aspects of DJINN's initial weight topology: the sparsity of the nonzero weights, and where the nonzero weights are placed. To evaluate the importance of the dependency structure imposed by the DJINN weights, the algorithm is compared to neural networks that have no imposed dependency structure: networks with the same architecture, but densely-connected Xavier-initialized weights. To demonstrate that it is not just the sparsity of nonzero weights that is important, but the placement of these weights, DJINN is compared to a network with the same architecture, but with a random, sparse dependency structure imposed on the initial weights. The sparse-random initialization has the same number of nonzero weights per layer that the DJINN initialization utilizes, but with those weights placed randomly within the layer. The initialization guarantees that every neuron has at least one nonzero incoming and outgoing weight; this prevents the initialization from inadvertently changing the architecture by creating neurons that are unable to learn. Like DJINN, the non-zero weights are pulled from the Xavier normal distribution described in section II B.

The middle sections of Tables 3.2 and 3.3 show the performance of the random-dense and random-sparse initialization schemes for the four regression and classification datasets, respectively. Similar to the comparison between DJINN and the 2HL model, the random initializations are treated as ensemble models: each model

contains ten individual neural networks (for DJINN, this corresponds to ten trees, for random initializations this corresponds to ten random seeds used to initialize and place the weights). The prediction from the ensemble is the average of the ten individual predictions. The process of training and evaluating the performance of the random-dense and random-sparse initializations is repeated five times, with the same training and testing datasets used in the comparison between DJINN and the 2HL model. The randomly-initialized networks use the same architectures as DJINN, and are trained with the same hyper-parameters summarized in Table 4.1.

Figure 3.4 shows the training cost as a function of epoch for the regression tasks; DJINN acts as a warm-start to the training process by consistently starting at a lower cost than other initialization methods. Furthermore, DJINN often converges to the lowest cost, suggesting the network is initialized near a lower local minimum than random initializations can reach in a limited number of training epochs. The warm-start provided by the decision tree structure leads to higher predictive performance for DJINN in three of the four regression tasks. The improvements of DJINN over the other initialization schemes are statistically significant for the CA housing and yield datasets; the advantages of DJINN are less significant for Boston housing due to the noise in the training cost versus epoch, and the differences in initialization schemes are not significantly different for the diabetes progression data.

While DJINN achieves good predictive accuracy in classification tasks, the advantages of the DJINN weight initialization are less significant. The classification tasks considered are simpler than the regression datasets, thus the performance of various models is less sensitive to the choice of initial weights and hyper-parameter settings. Furthermore, the decision trees are kept shallow due to the size and dimensionality of the datasets; this limits the amount of information mapped from the decision tree into the initialized DJINN model.

The effects of limiting the depth of the decision tree for datasets with a large number of inputs are illustrated by the digit classification task. Each digit is 64-pixel image that are inputs for the decision tree. The decision tree will split first on the pixel that best separates the digits, however, it is unlikely that a single pixel can provide a significant amount of information about the class to which the image belongs. The decision tree needs to grow deep enough to consider dozens of pixel values before it can accurately classify the image as digit. For DJINN, the width of the hidden layers reflects the width of the input layer; with 64 input values, the depth of the neural network must be limited, otherwise there will not be enough data to train the model without a severe risk of over-fitting. Table 6.5 lists example DJINN architectures for each dataset; indeed, the hidden layers in the digit classification model are wide compared to models with fewer input parameters.

With a limited tree depth and a large number of input parameters, the decision paths in the tree are unlikely to contain a significant amount of information to provide a good warm-start for the neural network training procedure. This is illustrated in Fig. 3.5, where DJINN starts at a cost comparable to the other models for digit classification. In contrast, the iris dataset has four input parameters and three classes; thus the first few splits in the decision tree are able to provide valuable information for separating the classes, and DJINN starts at a slightly lower cost than the other models.

To handle datasets with a large number of inputs, it would be best to first send the data through convolutional filters or an autoencoder to compress the features into a low-dimensional, meaningful latent space. The latent variables can then be used as inputs to DJINN to build a predictive model; this idea is explored further in Chapter 5.

To summarize, the benefits of DJINN are most obvious when the trees are suf-

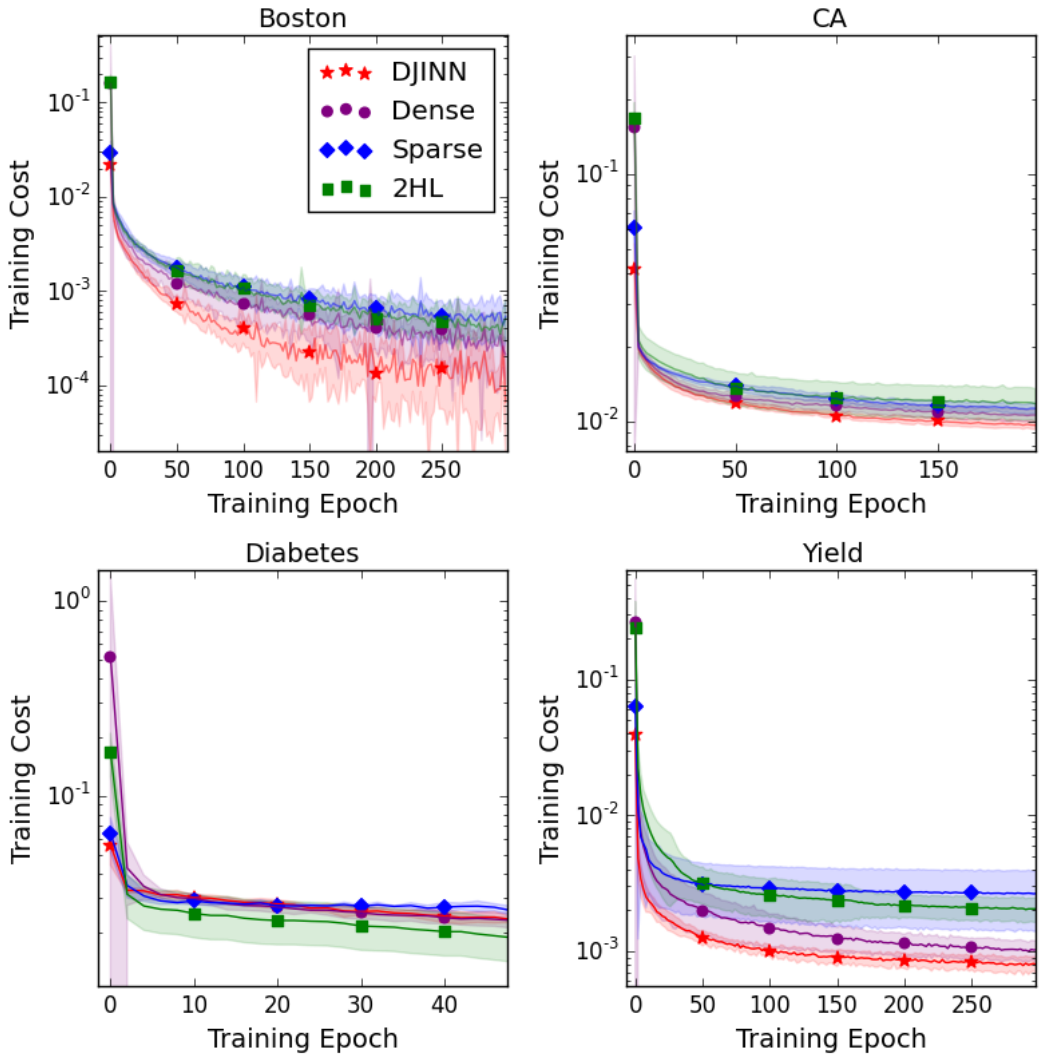


Figure 3.4: Cost (MSE for data scaled to [0,1]) as a function of training epoch for regression datasets. DJINN weights are observed to start at, and often converge to, a lower cost than the shallow network, or networks with DJINN architecture and other weight initialization techniques.

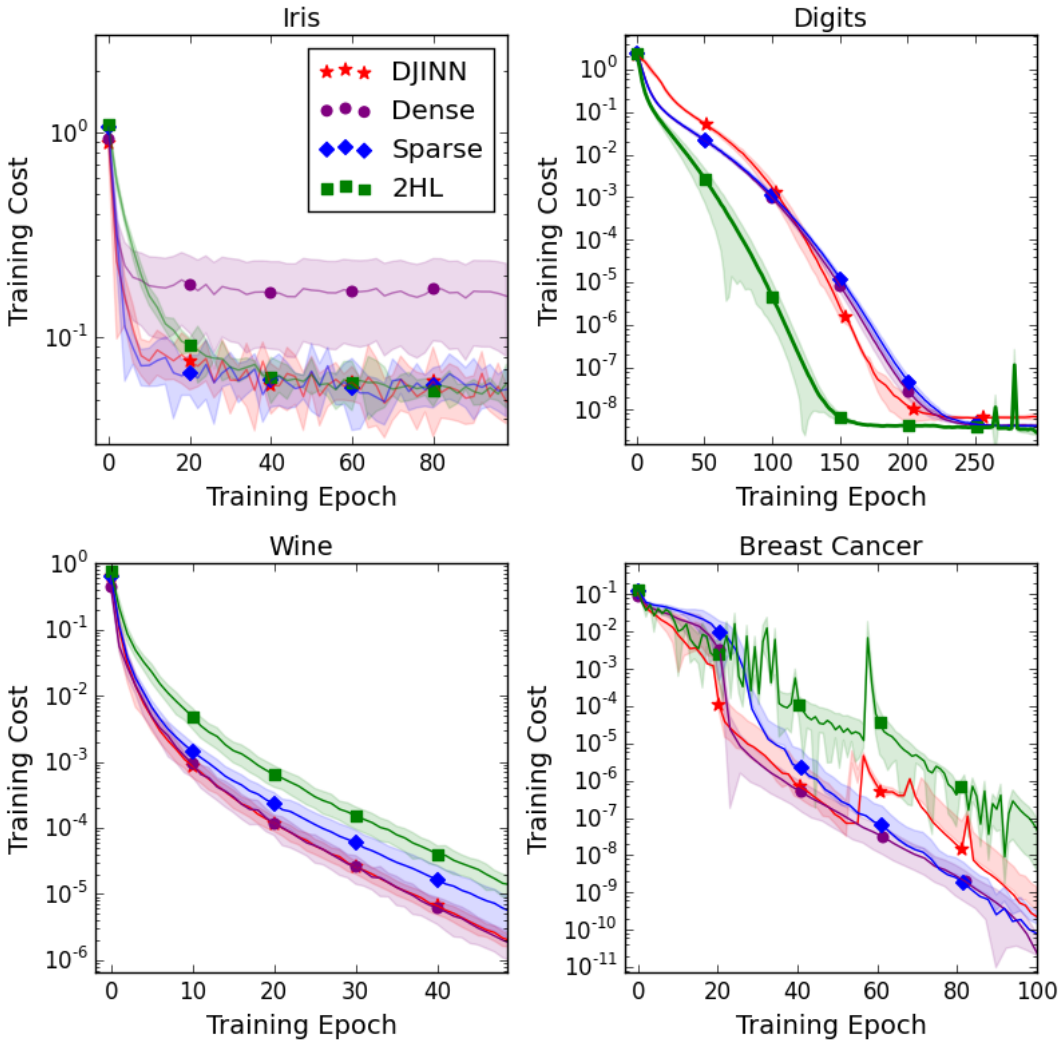


Figure 3.5: Cost (cross entropy with logits) as a function of training epoch for classification datasets.

ficiently deep, and the number of nodes in the tree exceeds the number of input parameters; this results in a meaningful dependency structure in the tree that is mapped to initial weights. These conditions are often met for regression tasks, where the warm-start provided by the decision tree allows DJINN to achieve higher predictive performance than non-informative initial weights. Although DJINN does not provide as significant of a warm-start for classification tasks, Table 3.3 shows that DJINN achieves good predictive performance and has the advantage of not requiring the user to hand-tune the architectures for each dataset.

3.4 Comparison of DJINN to hyper-parameter optimization

The utility of DJINN is its ability to be applied as a black box algorithm for efficiently creating accurate deep neural networks for arbitrary datasets. Recently, researchers have started developing a variety of hyper-parameter optimization techniques for designing deep neural networks [151, 150, 111, 4]. These algorithms eliminate the need to hand-tune architectures by searching through various combinations of hidden layers and neurons per layer to find the best architecture for a given dataset. Although effective, high-dimensional hyper-parameter searches can be prohibitively expensive. For each proposed architecture, the neural network must be trained to determine the quality of model, and unless the architecture space is restricted to a fixed number of layers or has a constraint on the number of neurons per layer, finding the optimal architecture can require training hundreds of candidate neural networks.

DJINN does not attempt to find the "optimal" architecture for a given dataset, it uses an architecture determined by the structure of a decision tree. This architecture, combined with the weight-initialization that leverages the dependency structure from the tree, is observed to produce accurate models for a variety of datasets.

Although DJINN does not solve the same problem a hyper-parameter optimiza-

tion method seeks to solve, both methods attempt to improve the usability of neural networks by reducing the number of hyper-parameters that must be specified to train a neural network. It is interesting to see how DJINN, which only requires training a decision tree to propose a suitable architecture, compares to a network designed via architecture optimization.

The final sections of Tables 3.2 and 3.3 show the performance of neural networks designed via Bayesian hyper-parameter optimization [119]. To constrain the search space, the optimizer is restricted to neural networks with the same number of layers used in the DJINN models, and searches for the optimal number of neurons for each hidden layer. Table 3.4 lists five of the architectures (resulting from the five-fold cross-validation) found via Bayesian optimization, and an example architecture from DJINN for each of the cross-validation steps. The candidate neural networks are trained with the same hyper-parameters summarized in Table 4.1 and are initialized with Xavier weights. The optimizer is stopped after it has evaluated 100 architectures, and the best model is used to compute the integrated performance metrics. Consistent with the other comparisons, the Bayesian optimizer is run for the five permutations of training/testing datasets to compute cross-validation scores recorded in Tables 3.2 and 3.3.

DJINN has a higher predictive performance than the Bayesian optimizer for three of the four regression tasks. The p-values indicate that the improvement of DJINN over the Bayesian optimizer is statistically significant for the CA housing and yield datasets, but the Bayesian optimizer performance is significantly better than DJINN for the diabetes progression data. For classification tasks, the Bayesian optimizer and DJINN perform similarly. Table 6.5 indicates that the optimization algorithm prefers smaller networks than DJINN for classification tasks; the inclusion of too many degrees of freedom in DJINN could explain its lower performance for the breast

cancer and wine classification tasks, consistent with previous discussions.

Computational efficiency is important to consider when employing hyper-parameter optimization procedures. For the examples presented above, the hyper-parameter optimization algorithm evaluates 100 neural networks; this requires approximately 10x the training time of DJINN when the ten network ensemble is trained serially, or 100x the training time of DJINN if the ten networks are trained in parallel. For the moderate-sized datasets, hyper-parameter optimization is feasible. However, for high volume, high dimensional datasets, hyper-parameter searches become prohibitively expensive. DJINN remains comparatively inexpensive as the complexity of the data increases, requiring only the construction of a small ensemble of decision trees, which are often trained in seconds, to determine an appropriate architecture and weight initialization. Subsequent training of the individual neural networks in a DJINN ensemble can then be carried out in parallel, offering significant advantages over sequential hyper-parameter optimization methods.

Overall, there is compelling empirical evidence to suggest DJINN is a robust black box algorithm for creating accurate neural networks for a wide variety of datasets. The advantages of DJINN are most evident in complex regression problems, where the choice of architecture and initialization can greatly impact the predictive performance of the model. For simple classification problems, the performance of DJINN is comparable to other network design and weight initialization techniques. Although DJINN is not attempting to find an optimal architecture, when compared against hyper-parameter optimization for designing neural networks, DJINN displays competitive performance while requiring significantly lower computational costs. DJINN successfully combines the usability of decision tree models with the flexibility of deep neural networks to produce accurate predictive models for a variety of problems.

Table 3.4: Hidden layer widths from DJINN and a Bayesian optimizer for each dataset. The width of the input layer reflects the number of features in each dataset. The output layer has a single neuron for regression tasks, and one neuron per class for classification tasks.

Dataset	DJINN	Bayesian Opt.
Boston	(15,17,20,18), (15,17,20,18), (13,15,22,27), (15,18,24,18), (14,18,22,26)	(7,10,9,14), (7,10,8,14), (13,14,10,7), (7,11,7,15), (12,9,8,11)
CA Housing	(10,12,19,23), (10,11,16,25), (10,11,19,25), (10,11,17,26), (10,14,19,21)	(12,20,5,6), (4,17,14,16), (5,13,16,5), (12,14,8,5), (20,5,20,8)
Diabetes	(12,15,19,28), (12,14,19,28), (11,15,19,28), (12,15,22,23), (11,14,20,22)	(4,1,4,7,9), (9,9,19,7), (18,9,10,4), (11,16,13,18), (13,15,16,18)
Yield	(11,15,18,23), (11,14,19,21), (10,15,22,25), (10,15,21,25), (10,13,21,23)	(5,5,30,30), (20,12,15,5), (22,10,23,19), (5,5,16,5), (14,18,11,14)
Iris	(5,4), (5,7), (5,5), (5,7), (4,5)	(14,6), (11,7), (13,7), (10,8), (11,4)
Digits	(65,48), (64,33), (63,33), (63,66), (64,23)	(17,32), (13,31), (5,49), (5,28), (13,31)
Wine	(14,15), (15,11), (15,9), (15,11), (13,9)	(4,11), (12,14), (7,12), (10,12), (12,5)
Breast Cancer	(32,33,19), (32,30,18), (32,33,23), (32,30,24), (30,31,19)	(5,3,6), (6,5,5), (2,5,6), (6,4,2), (5,6,4)

3.5 Conclusions

The flexibility and powerful predictive capabilities of neural networks are combined with user-friendly decision tree models to create scalable and accurate "deep jointly-informed neural networks" (DJINN). The DJINN algorithm maps an ensemble of decision trees trained on a dataset into an ensemble of initialized neural networks that are subsequently trained via back-propagation. The information mapped from the decision trees into initial weights provides a warm-start to the neural network training process; thus DJINN is often observed to start at, and converge to, a lower cost than other neural network initialization methods.

DJINN reduces the number of user-specified hyper-parameters needed to create a deep neural network by using the decision tree structure to determine the network architecture. When compared to hyper-parameter optimization methods for selecting an appropriate architecture, DJINN displays competitive performance at a fraction of the computational cost, demonstrating that an optimal architecture is not necessary if the weight initialization is sufficiently informative.

Although formulated for fully-connected feed-forward neural networks, DJINN could also be applied to networks that use feed-forward neural networks as part of a more complex system. For example, DJINN could be used for image analysis tasks after convolutional layers extract the important features. By combining the ease of use of decision trees with the predictive power of deep neural networks, DJINN is an attractive method for easily creating surrogate models of complex systems.

4. BAYESIAN ANALYSIS WITH DEEP JOINTLY-INFORMED NEURAL NETWORKS

Like most machine learning models, "deep jointly informed neural networks" (DJINN), presented in Chapter 3, produces point estimates for quantities of interest; standard neural networks do not automatically provide estimates of the prediction variance [43, 104]. In many situations we would prefer to have estimates of prediction uncertainty– for example in safety and risk analysis, or to aid in decision-making.

When uncertainty estimates are necessary, we often turn to Bayesian surrogate models, such as Bayesian additive regression trees [19], Gausssian processes [109], or Bayesian multivariate adaptive regression splines [27]. Obtaining surrogate uncertainty estimates using such algorithms comes at a cost— there are several challenges associated with training Bayesian surrogates which limit their usability. Rather than training a single model with optimal hyper-parameters, Bayesian surrogates often involve training a distribution of models whose hyper-parameters are found via Markov chain Monte Carlo (MCMC) sampling [40], which limits the speed at which the model can be trained. Furthermore, as Bayesian surrogates are distributions of models rather than single models, they can be expensive to store for later use.

There have been many research efforts dedicated to improving the usability of Bayesian surrogates by eliminating the need for MCMC sampling; common methods include variational inference [7] and expectation propagation [92]. In this work, an approximation to variational inference is applied to deep neural network models to produce fast, accurate, and scalable surrogates that provide prediction uncertainty estimates. In the following sections, the algorithm for training approximate Bayesian deep neural networks is introduced and applied to various prediction and

inference tasks. Section 4.1 summarizes the results of the previous chapter on "deep jointly-informed neural networks" [61], and how the inclusion of dropout in DJINN approximates variational inference of a deep Gaussian process. In section 4.2, the utility of training inverse DJINN models for parameter inference is illustrated with two examples: first with a simple one-parameter inference task, then in the next chapter, with a complex dataset of ICF simulations in which eight simulation inputs must be inferred simultaneously from experimental observations.

4.1 Deep jointly-informed neural networks with dropout

The simulations used to model ICF experiments are computationally expensive, thus MCMC sampling a large parameter space using simulations alone is unfeasible. To enable rapid evaluation of the simulator, "surrogate" models are trained on a fixed data base of simulations which span the parameter space of interest. The surrogate is essentially a model that interpolates between the available data, providing fast estimates of the QOI anywhere in the parameter space. Simple examples of surrogate models include linear regression or power law models, but as the data increases in complexity and dimensionality, more sophisticated machine learning algorithms are often required to accurately fit the data.

Deep neural networks have several properties that make them attractive surrogate models; they are accurate, flexible, scalable, and are easy to update as new data becomes available. However, the predictive accuracy of deep neural networks is often sensitive to the choice of the network architecture and training hyper-parameters. There are few robust guidelines for designing and training neural networks for arbitrary data sets, and hyper-parameter optimization techniques are prohibitively expensive unless the range of hyper-parameter values is highly constrained. "Deep jointly-informed neural networks" (DJINN), discussed in the previous chapter, is

an algorithm designed to overcome the challenges of determining an appropriate deep neural network architecture and weight initialization for arbitrary data sets. DJINN builds robustly accurate neural networks by mapping decision trees trained on the data into initialized neural networks, which are subsequently tuned via back-propagation [61]. DJINN is used in this work because it displays robustly accurate performance for a variety of data sets, and does not require extensive hyper-parameter tuning. DJINN is employed as an ensemble method, in which a random forest [13] of decision trees is mapped into an ensemble of neural networks.

An advantage of using deep neural network surrogates, such as DJINN, is that estimates on prediction uncertainties are readily obtained by employing a technique called dropout. Dropout is a popular regularization technique for deep neural network training; it requires randomly removing a small subset of neurons from the network each training epoch, preventing the network from over-fitting to the training data [124]. Gal et al. [36] recently demonstrated that sufficiently large neural networks with dropout approximate deep Gaussian processes [109, 35, 37]. The method for extracting uncertainty information from the neural networks requires dropout to be employed after each layer of the network during the training and evaluation stages. Evaluating the network many times samples the neural network model space, building up a distribution of predictions which approximates variational inference of a deep Gaussian process. Since DJINN is often employed as an ensemble method, the model space that gets sampled when using dropout with DJINN is larger than a single neural network with dropout; recent work has shown that such ensemble methods, while not strictly Bayesian, yield prediction uncertainties that behave similarly to Gaussian process methods[103].

To evaluate the efficacy of DJINN with dropout as an approximately Bayesian model, the performance of Bayesian DJINN (B-DJINN) is compared to several

other Bayesian surrogate models: Bayesian multivariate adaptive regression splines (BMARS) [27], Bayesian additive regression trees (BART) [19], and Gaussian processes (GP) with radial basis function (RBF) kernels. The optimal length scale in the RBF kernel in the GP is found by maximizing the log-marginal likelihood using the L-BFGS-B algorithm [80]. The optimal length scales for each data set are: 0.215 for logistic, 0.138 for Boston housing, 0.152 for diabetes, and 0.223 for the yield data set. Table 4.1 summarizes the hyperparameters for the B-DJINN models, including a representative architecture from the ensemble of five neural networks, and the training parameters. The dropout rate was set to 5% for all of the networks, and the weight regularization coefficient is set to 10^{-6}.

The models are compared on four regression tasks: the Boston housing [52] and diabetes progression [29] data sets are common test problems for regression. The logistic data set is generated by varying x and k in the logistic function and adding Gaussian distributed noise to compute $f(x,k)$:

$$f(x,k) = \frac{1}{1-\exp(-kx)} + N(\mu = 0, \sigma^2 = 0.01). \tag{4.1}$$

The data set contains 1000 randomly sampled points with x sampled between (-1,1) and k sampled between (0,10). The yield data set is a set of 5000 Latin hypercube sampled [87] ICF implosion simulations that span a 4D simulation input space. The implosions are simulated with the multi-physics code HYDRA [86], and the primary QOI in the database is the yield– the thermonuclear energy produced in the implosion. Each data set is split into an 80% training set and 20% test set, and the inputs are scaled between [0,1] prior to training.

To evaluate the accuracy of each model, the mean squared error (MSE), mean absolute error (MAE), and explained variance ratio (EV) are computed using the

Table 4.1: Hyperparameter for B-DJINN models.

	Architecture	Learning rate	Epochs	Batch size
Logistic	4, 6, 13, 20	0.004	500	30
Boston	13, 12, 20, 23	0.004	600	10
Diabetes	10, 13, 18, 3	0.004	300	18
Yield	5, 7, 13, 23	0.002	1000	400

mean predictions for the test data set. These metrics do not take into account the uncertainties in the predictions, thus to compare the overall accuracy and precision of each model, the normalized sum of absolute errors is computed using the following equation:

$$\frac{1}{N_{data} \cdot N_{pred}} \sum_{i=1}^{N_{data}} \sum_{j=1}^{N_{pred}} |Y_j(x_i) - T_i|, \tag{4.2}$$

where N_{data} and N_{pred} are the number of data points and predictions, respectively, $Y_j(x_i)$ is the jth prediction for the data point x_i, and T_i is the true value at point x_i. This metric is large for models that are inaccurate and have wide distributions of predictions, and small for accurate models with low uncertainties. Table 7.1 summarizes the performance metrics for each of the regression data sets.

B-DJINN with dropout displays similar performance to other Bayesian regression algorithms, both in terms of its average accuracy and according to integrated metrics which take into account the uncertainty in the model predictions. B-DJINN offers some distinct advantages over models such as BMARS and BART – it does not required MCMC sampling and thus B-DJINN can be trained more efficiently, and it is easier to save and evaluate at a later time as it does not require the full distribution of models to be saved. To illustrate the quality of the B-DJINN model fits, Fig. 4.1 compares the Gaussian process to B-DJINN for the logistic function

Table 4.2: Performance metrics for B-DJINN and several Bayesian regression models for four data sets. The MSE, MAE, and EV are computed using the mean predictions for the test data sets. Bold values indicate the best value of each metric. B-DJINN shows similar performance to the other Bayesian surrogates.

	Logistic				Boston			
	MSE	MAE	EV	NSAE	MSE	MAE	EV	NSAE
B-DJINN	3.171e-4	0.014	0.998	0.038	**5.359**	**1.743**	**0.913**	**2.068**
GP	9.023e-4	0.008	0.992	0.098	22.62	3.042	0.716	3.122
BART	1.292e-4	0.008	0.999	0.098	15.42	2.574	0.690	3.274
BMARS	**2.495e-6**	**0.002**	**0.999**	**0.01**	18.57	2.867	0.637	2.602
	Diabetes				Yield			
	MSE	MAE	EV	NSAE	MSE	MAE	EV	NSAE
B-DJINN	2935	40.82	0.405	**45.82**	0.002	0.034	**0.998**	0.065
GP	5451	60.65	0.257	60.66	**0.001**	**0.021**	0.996	0.127
BART	3379	47.11	0.466	47.02	0.002	0.032	0.995	0.038
BMARS	**2142**	**38.47**	**0.638**	49.70	**0.001**	0.032	**0.998**	**0.015**

data set. B-DJINN fits the logistic data set with lower mean error than the Gaussian process and displays lower levels of uncertainty, particularly at the boundaries of the data set. Unlike many Bayesian surrogates that require MCMC sampling the model space, Gaussian processes and B-DJINN are not limited by the scalability and computational costs of MCMC, and are thus efficient to train and evaluate. However, B-DJINN requires significantly less storage space (880 KB) than GP model (18 MB) and scales well to large databases. Traditional Gaussian process models are often limited to a few thousand training points, as the standard model requires a matrix inversion that becomes prohibitively expensive for large data sets, though there have been efforts to overcome this challenge [144, 55]. The speed, scalability, and accuracy of B-DJINN makes it an attractive model for performing challenging parameter inference tasks. In the next section, B-DJINN inverse models are compared to Monte Carlo sampling of B-DJINN and Gaussian process forward models for inference tasks

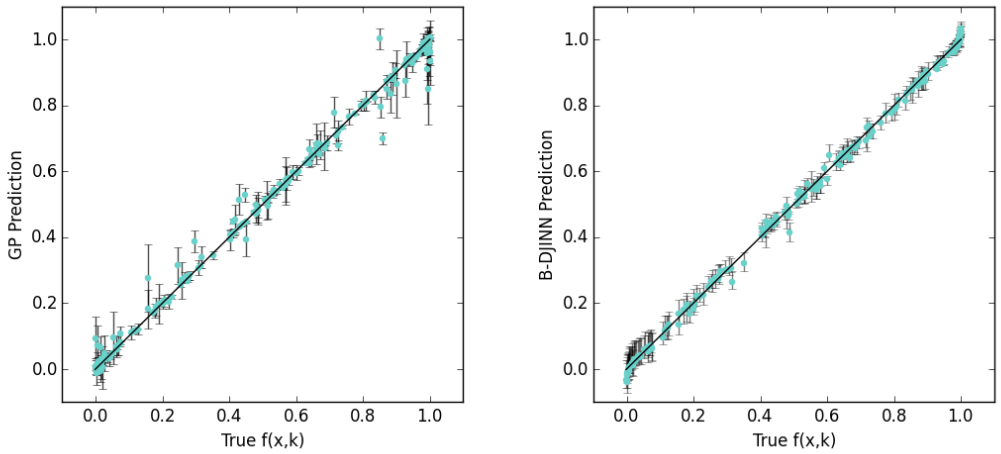


Figure 4.1: Predicted values of the logistic function with added noise plotted against the true values. The error bars indicate the 25th and 75th percentiles, the colored points indicates the 50th percentile. B-DJINN (right) is more accurate and precise than the Gaussian process (left), particularly at the boundaries of the data.

for inertial confinement fusion data.

4.2 Inverse models with B-DJINN

In complex experiments there are often input conditions to the system that are not directly controlled or measured, and must be inferred with physical models that relate the unobservable and observable quantities. Determining a set of unknown inputs which correspond to a set of observed outputs is a common task in Bayesian analysis; unknown parameters are typically inferred by MCMC sampling the "forward" surrogate model– the model that maps from input to output space– to identify sets of inputs that are consistent with observations.

Rather than MCMC sampling the forward model, B-DJINN models can be trained to map directly from output to input space, this concept is illustrated in Figure 4.2. The inverse mapping is often degenerate; this is reflected in the uncertainty of the

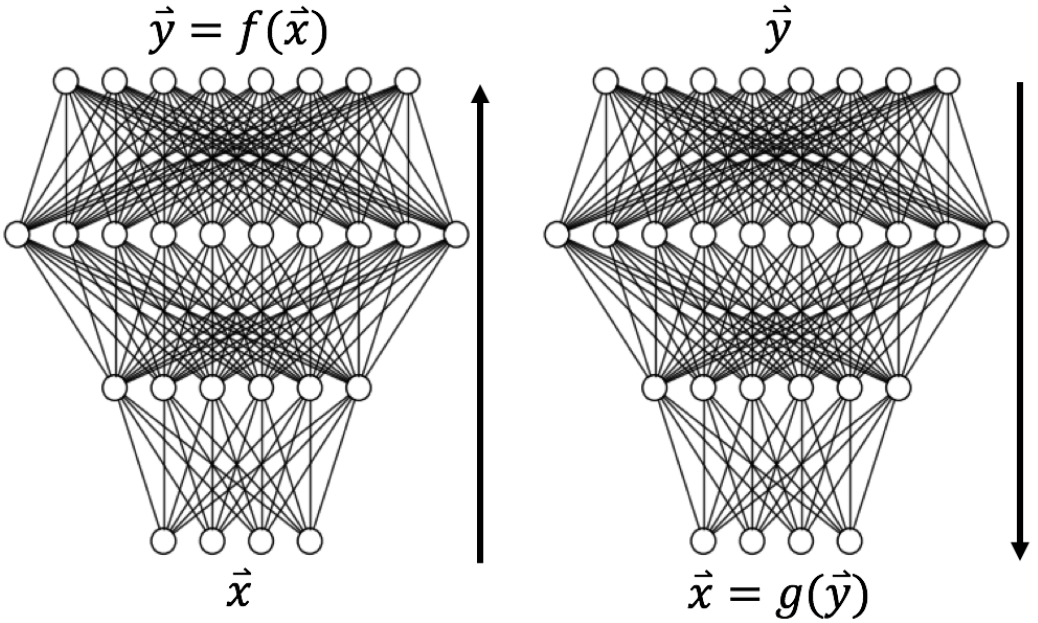


Figure 4.2: Forward (left) and inverse (right) deep neural network models. The forward model maps from simulation and experimental inputs to observables; the inverse model maps from observables to inputs.

B-DJINN predictions. The degeneracy of the inverse model can be reduced by including a large number of observables in the output space– knowing more about the experimental outcomes aids in constraining where the experiment is located in simulation input space. Although the inverse models can suffer from large uncertainties, they provide rapid estimates of the simulation inputs consistent with the observables, which can then be confirmed by evaluating the forward surrogate.

To demonstrate the utility of training inverse models for inference tasks, MCMC sampling the forward model is compared to the inverse model predictions for the logistic function dataset. Given ten random values of x and the corresponding value of $f(x, k)$, the task is to infer the value of k. The forward B-DJINN and Gaussian process models are MCMC sampled using the Metropolis-Hastings algorithm [53]

to find the values of k consistent with the observations of x, $f(x,k)$; the resulting distributions are shown in blue (B-DJINN) and green (GP) for various true values of k in the left side of Fig. 4.3. These distributions can be compared to the inverse model predictions of k that are generated by training a B-DJINN model to map directly from $(x,\ f(x,k))$ to k, then evaluating the inverse model with the ten available observations and collecting the predicted values of k into a distribution. The black points on the plot indicate the true value of k.

The MCMC sampled DJINN model provides the most accurate and precise predictions for k. Although the inverse model displays high uncertainties, the mean value of its prediction is close to the true value of k away from the boundaries of the dataset. The inverse model prediction is acquired in a fraction of the time required for MCMC inference, and although uncertain, the mean prediction is accurate enough to be used as a starting point for the MCMC sampling to accelerate convergence. The right side of Fig. 4.3 demonstrates the utility of using the inverse model prediction to warm-start sampling for k=9. The MCMC chain converges within 3000 MCMC samples when initializing the chain at the inverse model mean prediction for k; randomly initializing the chain takes over 100,000 samples to converge.

Although this is a simple inference task in which only a single parameter is unknown, the methodology is readily extended to high-dimensional inference tasks. As the number of parameters that need to be inferred increases, the advantages of using an inverse model to approximate the values of the unknown parameters become significant. MCMC sampling in high-dimensional spaces can be prohibitively expensive, and starting the chain near the true value can greatly reduce the time spent searching the space.

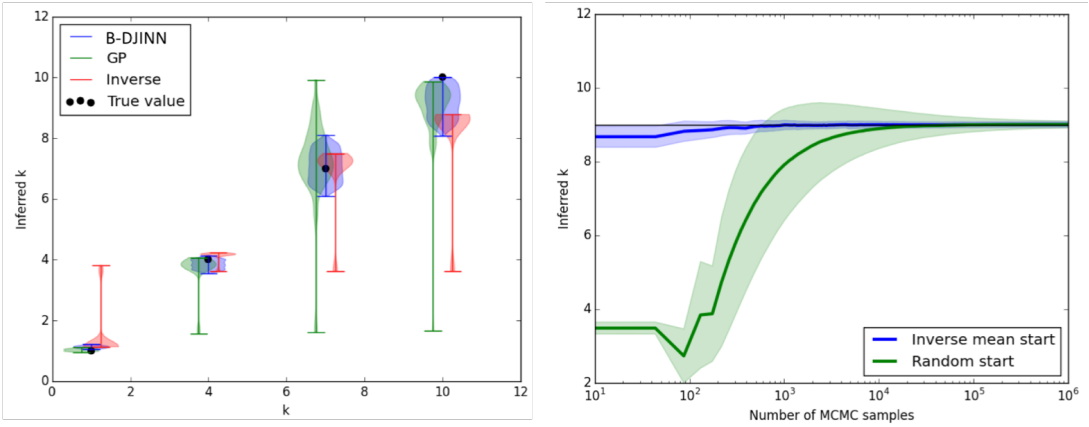


Figure 4.3: Parameter inference with inverse models and MCMC sampling forward models. Left: B-DJINN, Gaussian process (GP), and an inverse model are used to infer k given values of x, $f(x, k)$. Right: Inverse model estimates accelerate the convergence of a Markov chain by initializing the search near the true value. The shaded region indicates the 25th and 75th percentile from the distribution of inferred values of k, and the solid line indicates the median value.

4.3 Conclusions

"Deep jointly-informed neural networks" is a novel algorithm for designing and initializing neural networks by leveraging information contained in a decision tree trained on the data. DJINN is readily cast into an approximate Bayesian framework via the inclusion of dropout, which has been shown to approximate deep Gaussian process models. The Bayesian formulation of DJINN displays competitive performance with several standard Bayesian regression algorithms, but is significantly faster to train, easier to store and reload for later use, and faster to evaluate than traditional Bayesian models.

B-DJINN is used to train inverse models for efficient parameter inference, and provides accurate mean estimates of inferred parametes without requiring expensive MCMC sampling. In the next chapter, we apply this methodology to per-

form advanced post-shot analyses of ICF experiments. Post-shot analyses are high-dimensional parameter inference tasks for which MCMC sampling is prohobitively expensive; inverse models offer an efficient and accurate approach to parameter inference for ICF data with low computational cost.

5. AUGMENTING POST-SHOT ANALYSIS WITH DEEP NEURAL NETWORKS

In the previous chapter, DJINN with dropout is used for parameter inference tasks for simple examples. Parameter inference is a common challenge for indirect drive ICF, as many simulation inputs are not directly controlled or measured, and thus must be inferred based on experimental observations. The process of inferring the simulation inputs that are consistent with experiential measurements is often referred to as a "post-shot" analysis for ICF experiments [60, 99].

Post-shot analyses are critical to understanding ICF implosions; many physical quantities (such as hotspot pressure, or applied laser drive asymmetries) cannot be explicitly measured, and thus must be inferred from post-shot simulations. However, there are often multiple sets of simulation inputs that produce results that are comparable to experimental observations, and analyzing a single post-shot solution can lead to incorrect conclusions about the experiment.

Traditionally, post-shot analyses are performed by varying a small number of simulation inputs manually until the simulation outputs fall within the experimental uncertainties for a few important quantities of interest. This is a slow procedure and often only one set of simulation inputs is found, when in fact there could be several combinations of simulation inputs that are consistent with the experimental measurements. Inverse models can improve the current approach to post-shot analysis by finding the distribution of simulation inputs that are consistent with an experiment. Inverse models trained on the appropriate database of ICF simulations– one which encompasses the experimental observations– are easy to evaluate and can include a large number of observables to better constrain the distributions of inferred

inputs.

In the following sections, deep neural network inverse models created with DJINN are used to perform efficient post-shot analysis of NIF [93] experiments. In section 5.1, we present a straightforward application of inverse models to infer eight unknown input parameters of a specific experiment. In section 5.2, we introduce deep neural network auto-encoders for dimensionality reduction and describe how they can be used to improve inverse models for ICF data. We then present an example post-shot analysis of a NIF experiment using auto-encoders coupled with DJINN forward and inverse models, and conclude with avenues of future work analyzing other data from other high energy density experiments in sections 5.3 and 5.4.

5.1 Inverse model inference of unknown inputs in NIF experiments

Post-shot analysis with B-DJINN is tested using a database of ICF implosion simulations designed to encapsulate the NIF experiment N170109 [137, 14]. The database used to train the B-DJINN models contains 60,000 HYDRA simulations that are Latin hypercube sampled from an 8D input space. The inputs include several engineering features that are present in the experiment: the amplitude and width of a fill tube, a small tube used to fill the capsule with DT gas [85], and a tent– a thin membrane that holds the capsule in place within the hohlraum [96]. The inputs also include effects that could cause performance degradation in the implosion, including carbon mix of the ablator into the DT fuel, and extra energy in the fuel prior to the implosion (preheat). The final three inputs are characteristics of the experiment that are controllable– the energy scale, which is a scale factor for the laser energy and fuel capsule size, the peak multiplier, which describe the energy and power of the laser, and the amount of dopant added to the ablator.

As a demonstration, four observables commonly matched in post-shot analyses

are used to infer the simulation inputs. The observables include: neutron yield (log10(Neutrons)), which is the number of fusion neutrons produced during the implosion, ion temperature (T_ion), which is inferred using the width of the neutron birth spectra [94], bang time, which is the time of peak neutron production, and the down scatter ratio (DSR), which is the ratio of lower to higher energy neutrons in the observed spectrum [54]. The observables are used to train inverse models that map from the 4D output vector to the 8D simulation input vector. The mapping is degenerate, but it serves as an illustrative example as to how unconstrained the simulation inputs are when only matching a small number of experimental observables.

The inverse models are tested on a simulation for which the true inputs and outputs are known. The four observables are Gaussian distributed about the true value with uncertainties typical for experiments; these distributions are shown in blue in Fig. 5.1. The distributions of observables map into distributions of 8D input vectors via the inverse B-DJINN model. Figure 5.2 illustrates the resulting simulation inputs from the inverse model, shown in red; the black lines in the figure indicate the true values of the simulation inputs. The inverse model has non-negligible error in its predictions; it is therefore expected that some of the simulation inputs are not actually consistent with the known outputs. The red input distributions are passed through the forward model, resulting in the red output distributions shown in Fig. 5.1. As expected, the red "inverse + forward" distributions are broader than the true output distributions due to the error in the surrogate models.

An alternative approach to inverse modeling is MCMC sampling the forward distribution to locate simulations whose outputs fall within the blue distributions indicated in Fig. 5.1; the resulting set of inputs are shown in blue in Fig. 5.2. The inputs are much more constrained in the inverse mapping predictions, which is attributed to the error of the inverse surrogate. Due to the degeneracy of the problem,

it is challenging to train an accurate model to predict eight inputs given values of four observables, and the model predictions trend toward the mean of the training data when relationships between the inputs and outputs are not constrained. This is shown clearly in the inference of the fill tube width, which is unconstrained according to the MCMC result, but is predicted to be the mean value of the training data by the neural network.

The same procedure can be used to infer simulation inputs most consistent with observations from an actual ICF experiment, referred to as N170109 [14]. Figure 5.3 shows the distributions of the outputs, and Fig. 5.4 the corresponding simulation inputs. As shown in Fig. 5.3, in order to find simulations that are consistent with the observations, the range of acceptance was expanded to be within 2% of the mean value for yield, and within 3 standard deviations of the mean for the ion temperature– otherwise it is not possible to find simulations that match all four observables simultaneously. The mean values of the inverse model predictions are similar to the MC matches for many of the inputs, though the inverse models underestimate the uncertainties. The energy scale and dopant have known values of 1.0 and 0.0023 in the experiment; the inverse model predicts both quantities with low error.

5.2 Neural network-based post-shot analysis of ICF experiments

In the previous example, eight simulation inputs are inferred using four scalar observables from an experiment. The inverse problem is highly degenerate; four observables are not sufficient to constrain the eight inputs, thus the inferred input distributions are broad. To better constrain the input parameters, more observables need to be included in the post-shot analysis. However, adding more inputs to a neural network adds more degrees of freedom via the increased capacity of the input

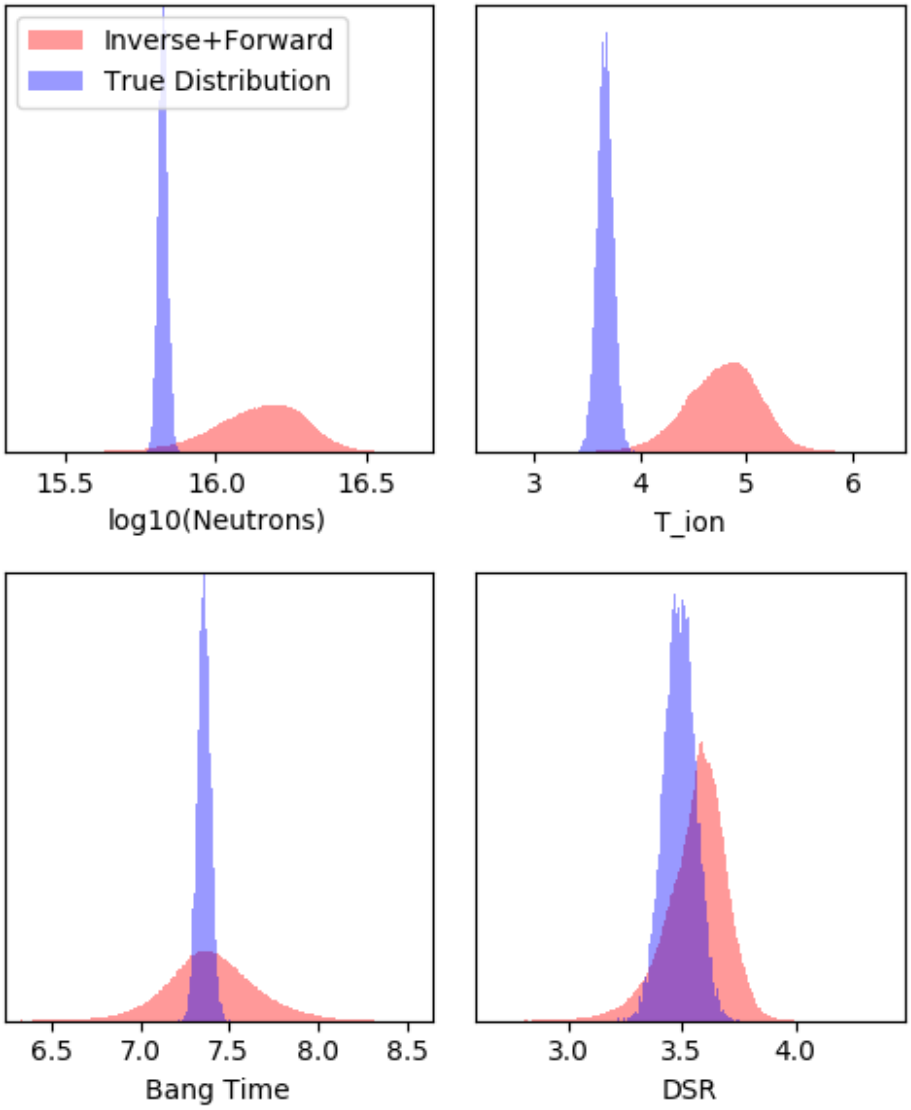


Figure 5.1: Distributions of simulation outputs from a simulation-only inference test. Blue distributions indicate the true values, red distributions are those found by first using the inverse model to map the blue distributions to simulation inputs, then using the forward model to map back to simulation outputs; error in the inverse model leads to broadening of the output distributions.

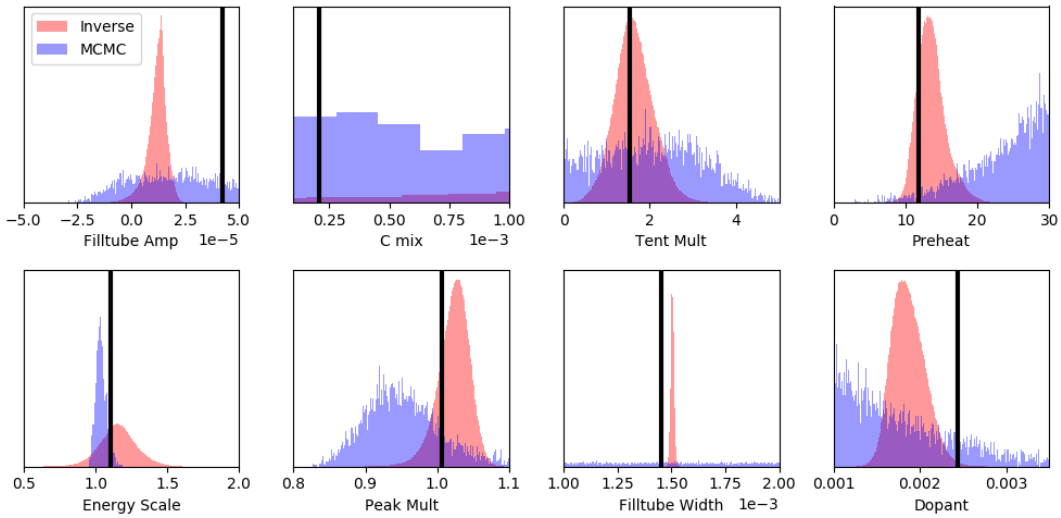


Figure 5.2: Distributions of simulation inputs. Red distributions are those found via the inverse model, blue distributions are those found by MC sampling the forward model, and accepting simulations whose outputs fall within the black dashed lines in Fig.5.1. The bold black line indicates the true value of the input.

layer, and therefore requires more training data that may not be available due to the computational expense of ICF simulations.

Many observables available in ICF experiments contain redundant information; X-ray images and neutron spectra are recorded from various lines of sight, temperatures and other QOI are measured using various diagnostics, etc. . We therefore expect that the suite of several dozen observables can be compressed down to fewer more fundamental numbers that efficiently summarize the experimental measurements. There are many dimensional reduction algorithms that leverage correlations in datasets to compress information into a low-dimensional representation. We consider the use of autoencoders, a particular type of neural network, to compress a set of experimental observables from ICF experiments into a low dimensional "latent space". The next subsection describes autoencoders in detail, followed by an exam-

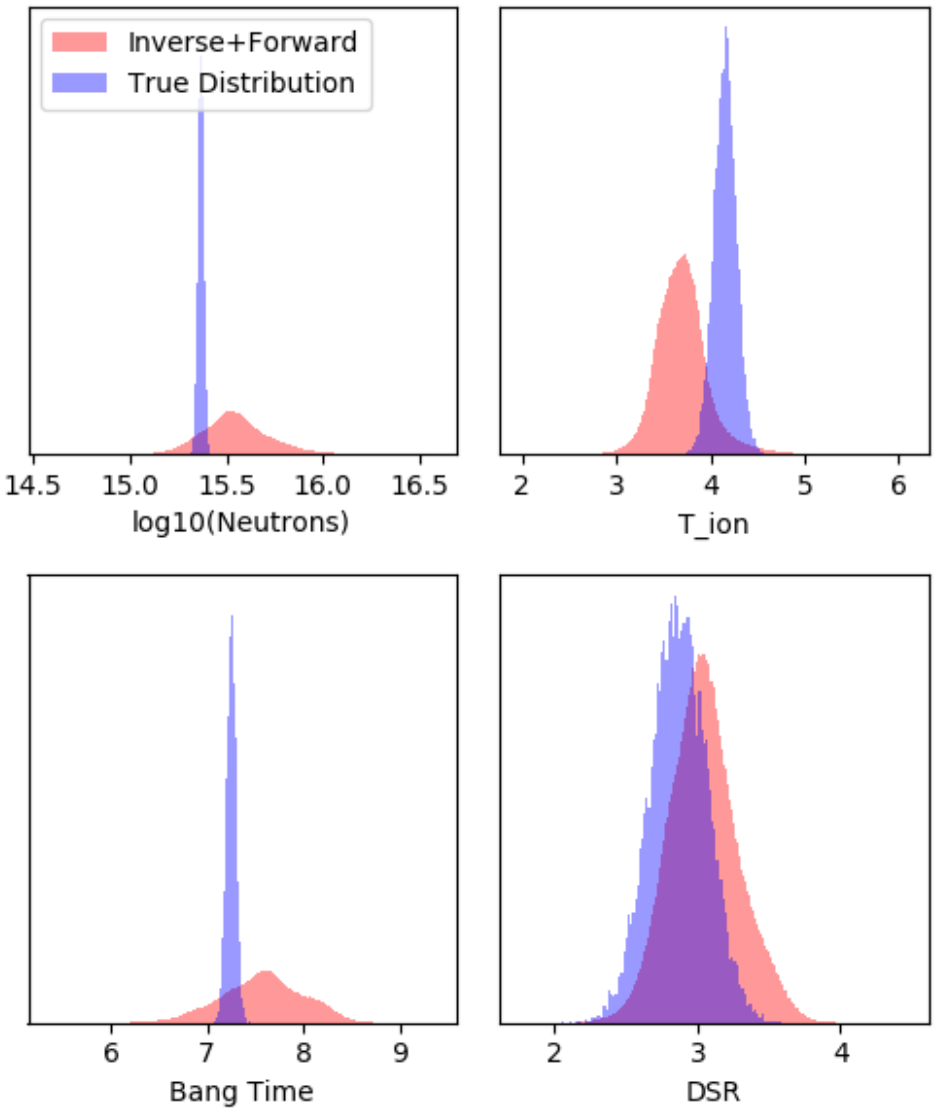


Figure 5.3: Distributions of observables for N170109. Blue distributions indicate the experimental values, red distributions are those found by first using the inverse model to map the blue distributions to simulation inputs, then using the forward model to map back to simulation outputs; error in the inverse model leads to broadening of the output distributions.

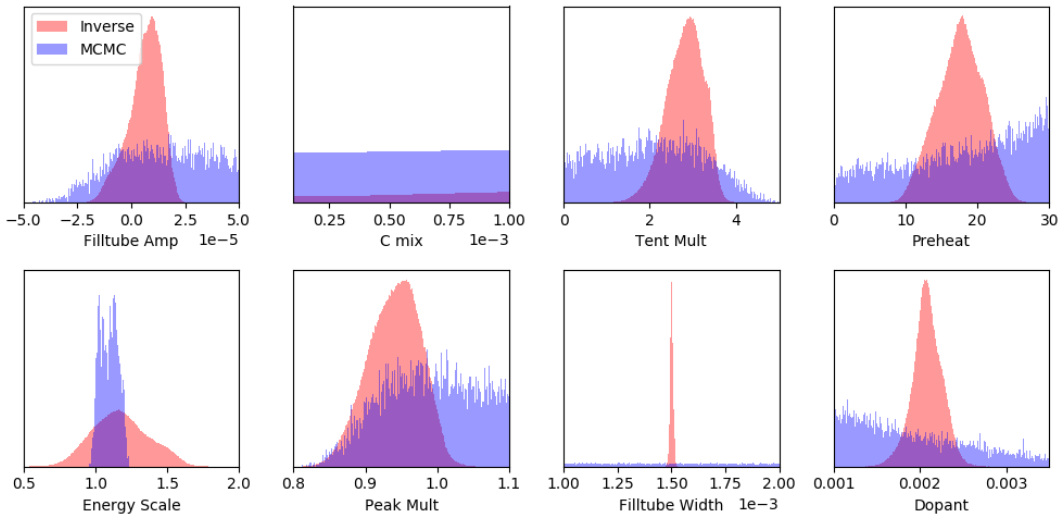


Figure 5.4: Distributions of simulation inputs consistent with observables for N170109. Red distributions are those found via the inverse model, blue distributions are those found by MCMC sampling the forward model. The limits on the x axes indicate the boundaries of the simulation data upon which the models were trained; values outside of these limits are extrapolations and thus are not considered reliable.

ple of how autoencoders coupled with DJINN models can be used to improve neural network post-shot analyses by reducing the degeneracy of the inverse models.

5.2.1 Autoencoders

An autoencoder is a neural network designed to nonlinearly compress data with minimal information loss. Autoencoders are straightforward variations of traditional neural networks that have a characteristic hourglass-like shape, illustrated in Fig. 5.5. For example, a simple autoencoder is a fully-connected feed forward neural network that takes as an input a large vector of data, and nonlinearly compresses the data through hidden layers with a decreasing number of neurons per layer until a bottleneck layer is reached. This bottleneck layer is often referred to as the "latent space"; it is a low-dimensional representation of the original vector of data that was input to the network. The network from the input layer to the bottleneck layer is the "encoder"; it encodes the large vector of data into the low-dimensional latent space. The second half of the network is often a mirror image of the first half, but with different weights; it decompresses the data one layer at a time until the output layer, which is the same size as the input layer. This half of the network is the "decoder"; it takes the latent space vector and decompresses it to get back the original vector that was input to the encoder. The autoencoder is trained by minimizing the reconstruction error between the input vector and the output vector, while forcing the data to go through the low-dimensional latent space.

Autoencoders do not have to be fully-connected neural networks; often convolutional layers are used to construct autoencoders for image data, and recurrent layers for sequential data. We will explore the use of recurrent autoencoders for ICF time series data in a later chapter, but for scalar ICF data we will focus on standard fully-connected autoencoders like that illustrated in Fig. 5.5.

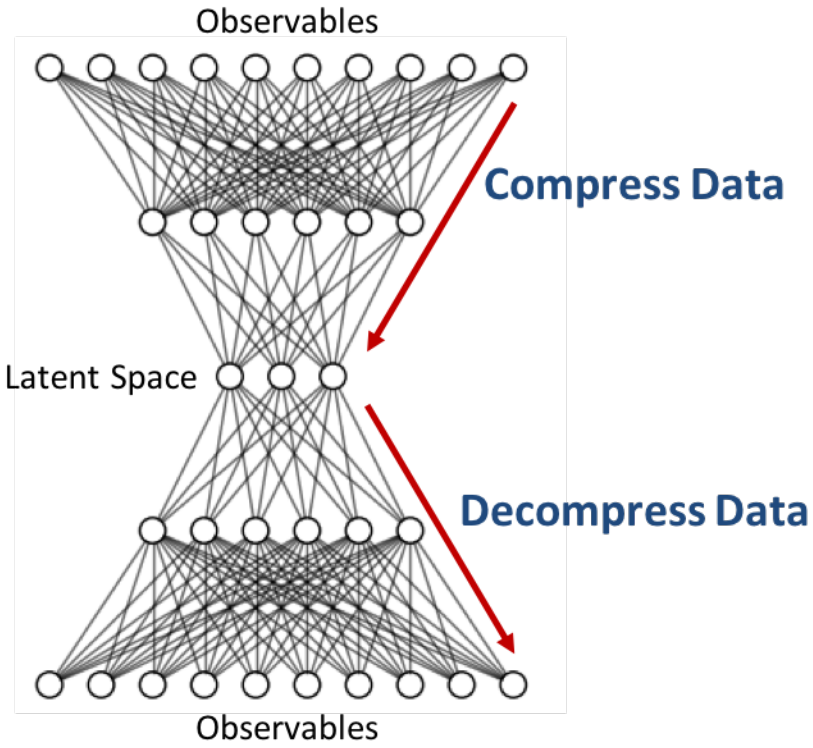


Figure 5.5: Example autoencoder architecture. The top half of the network, the encoder, compresses the observables into a lower-dimensional latent space. The second half of the network, the decoder, decompresses the latent space to get back the original vector of observables.

5.2.2 Augmenting post-shot analyses with autoencoders and DJINN

Inverse models for post-shot analysis suffer from the degeneracy of the inverse problem. To reduce degeneracy, many observables need to be included to constrain the distributions of inferred input parameters. The challenge of including several dozen observables is the amount of data required to train a neural network with a large number of input parameters, and therefore a large number of degrees of freedom. Autoencoders enable us to include many observables in the inverse models without adding a significant number of inputs to the inverse DJINN model, keeping the training data demand moderate.

We can combine autoencoders and DJINN models to perform neural network-augmented post-shot analysis for NIF experiments. Figure 5.6 illustrates the workflow for performing post-shots. Experimental observables are passed through the encoder to compress the data into a low-dimensional latent space. An inverse DJINN model is trained to map from the latent space ($\mathbf{y}$) to the input space ($\mathbf{x}$) to produce a distribution of predictions for the simulation inputs that are consistent with the experimental observations. The mapping from experimental observations to simulation inputs is the standard post-shot problem. Next, the simulation inputs are passed through a forward DJINN model to get back a distribution of latent space vectors, which can be decoded to produce distributions of simulation observables. Ideally, the distribution of simulation observables would match the initial distributions of experimental observables; however, in practice these distributions will be slightly different. The DJINN models and autoencoder introduce error which will increase the variance of the observable distributions, in addition to error that reflects the fact that the simulations and experiments are often inconsistent with one another. Although it is not possible to decouple the error due to the neural networks and the error due

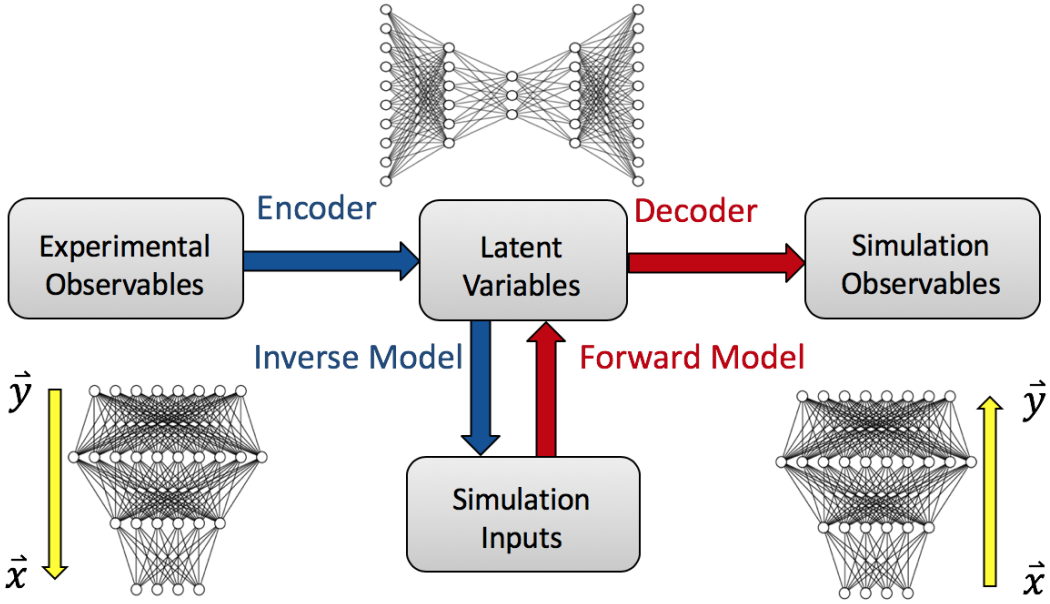


Figure 5.6: Workflow for performing neural network-augmented post-shot analysis. Experimental observables are mapped into the latent space using the encoder network. An inverse DJINN model maps from the latent space representation to the simulation inputs, producing the set of inputs that are consistent with the experimental observations. The inputs can then be passed through the forward DJINN model and decoded via the decoder to give back the set of simulation observables that are most consistent with the experimental observations.

to the simulations themselves, this workflow enables us to quantify the error between our best-matching post-shot simulations and the experimental observations.

To illustrate the improvements including a large number of observables provides to a post-shot analysis, we will compare the post-shot analysis using the above workflow to a simple inverse model post-shot analysis with the four scalar observables included in Figures 5.4, 5.3– the yield, ion temperature, bang time, and down scatter ratio. The two workflows are summarized in Fig. 5.7.

To illustrate the improvements autoencoders offer for post-shot analysis, consider the same test simulation from Figures 5.1-5.2. Rather than using inverse models to

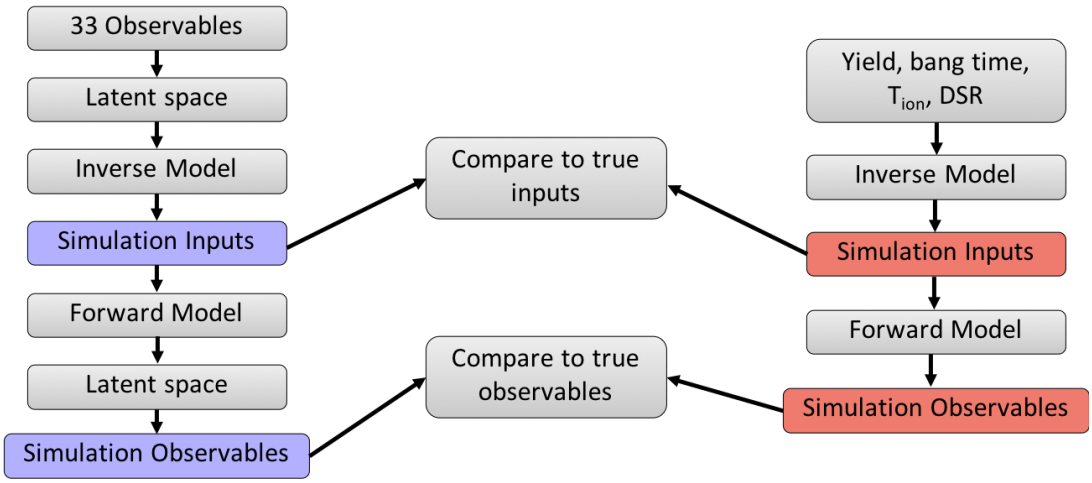


Figure 5.7: Autoencoder-based post-shot analysis (left) and simple inverse model post-shot analysis (right).

find post-shot simulations which are consistent with only four observables, inverse models built with the autoencoder find simulations which are consistent with all 45 observables included in the latent space. Figure 5.8 shows the distributions of four of the observables, and Fig. 5.9 shows the corresponding post-shot simulation inputs.

Requiring post-shot simulations to be consistent with a set of 45 observables results in more accurate and precise predictions of the simulation inputs from the inverse models, as shown in Fig. 5.9. Although there is still error accumulation from the forward and inverse models, resulting in some post-shot simulations which have observable values slightly outside the range of the true experimental uncertainties, shown in Fig. 5.8, the error is significantly lower than that in Fig 5.1.

The inverse modeling approach to post-shot analysis performs best when a large number of observables are used to constrain the simulation inputs; when matching only a small number of observations the mapping is highly degenerate, and MCMC sampling the forward model yields better results. Although including a large num-

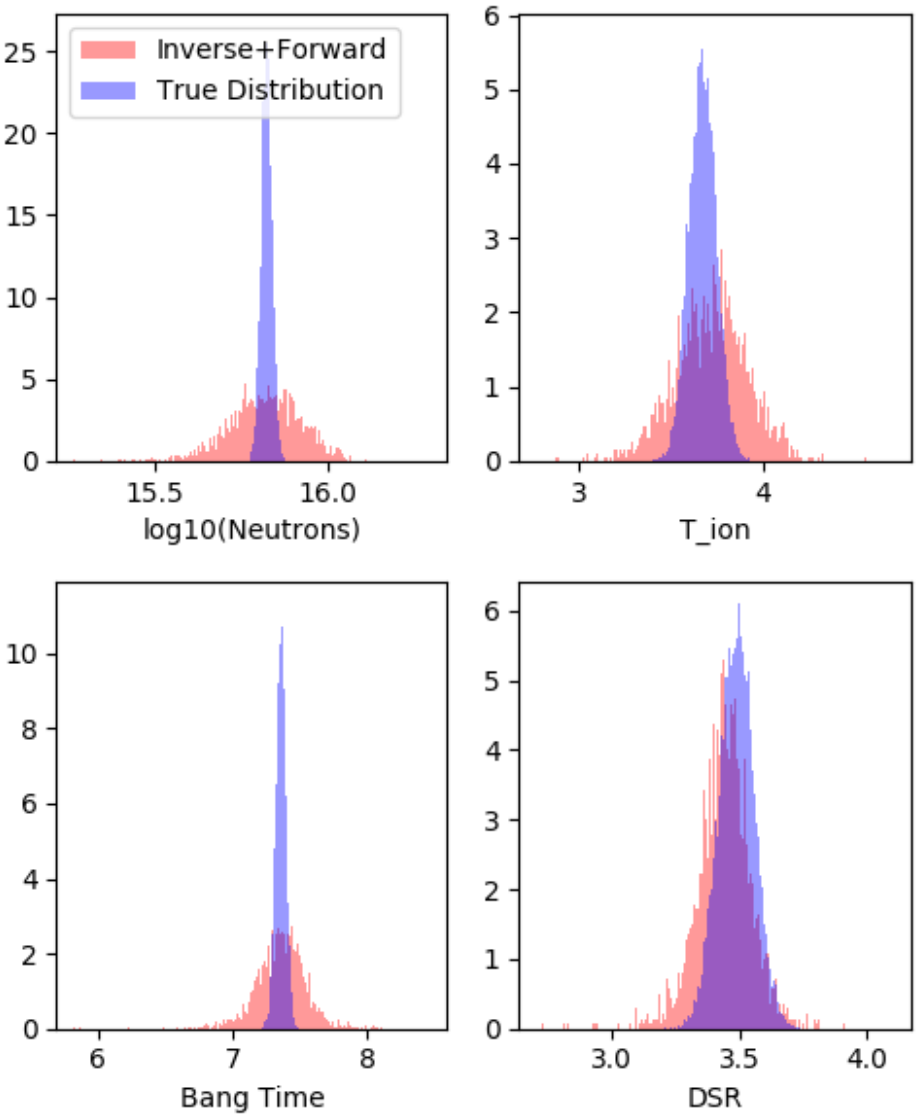


Figure 5.8: Distributions of simulation outputs from a simulation-only inference test. Blue distributions indicate the true values, red distributions are those found by first using the latent space inverse model to map the blue distributions to simulation inputs, then using the forward model to map back to the latent space, which is decoded to produce simulation outputs. The latent space post-shot analysis locates simulations which agree well with experimental observations, without requiring expensive MCMC sampling.

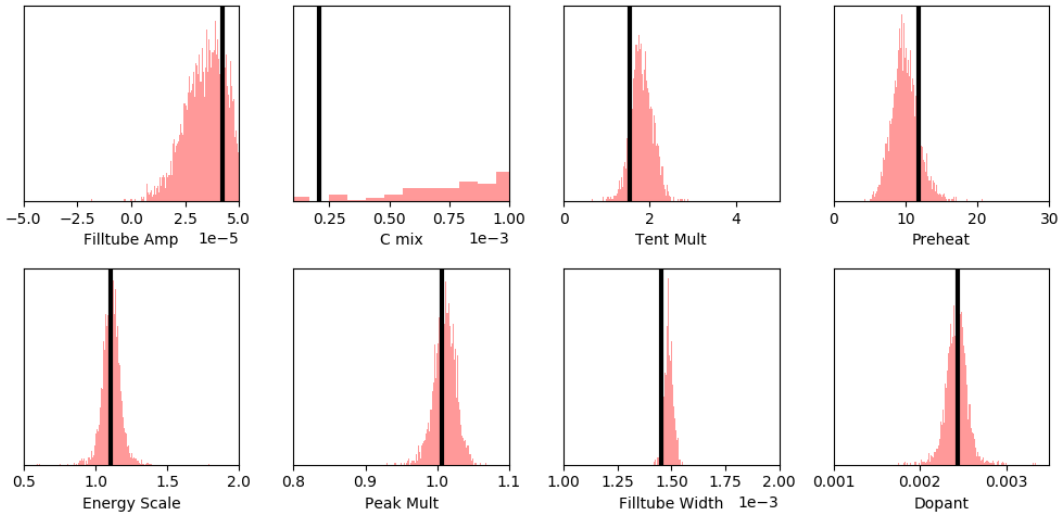


Figure 5.9: Distributions of simulation inputs. Red distributions are those found via the inverse model which maps from the latent space (composed of 45 observables compressed to 10 latent parameters) to the simulation input space. The bold black line indicates the true value of the input. The latent space post-shot analysis results in better-constrained and more accurate values for the simulation inputs which are consistent with experimental observations, and does not require expensive MCMC sampling of the forward model.

ber of observables in the inverse models does not remove all of the degeneracy, it does improve the inverse model accuracy significantly. Furthermore, when matching several dozen observables MCMC sampling can be prohibitively expensive, thus inverse models augmented with autoencoders offer a promising alternative to MCMC sampling when searching high dimensional spaces for inverse solutions.

5.3 Post-shot analyses of HED experiments

The same methodology applied to post-shot analysis of indirect drive ICF experiments can be extended to other experimental platforms which rely heavily on computer simulations for design, such as high energy density experiments carried out at the NIF. The autoencoder-based post-shot analysis is being applied to the analysis of emission spectra from materials at extreme conditions to infer physical quantities, such as temperature or density. This work is ongoing, and the results presented are preliminary.

Autoencoders are used to compress emission spectra to low-dimensional latent space representations, which can then be used to train a DJINN model which relates the latent space to physical quantities of interest. Figure 5.10 illustrates an example spectrum and its reconstruction from a database of simple ICF implosions with Ge in the ablator, modeled with the multiphysics code Cretin [114]. Cretin uses detailed atomic physics calculations to determine the expected emission spectra that would be observed in an experiment with specified physical conditions. Thus far, we have demonstrated the ability to train autoencoders and DJINN models using a database of approximately 70,000 Cretin simulations which span a 10D input parameter space. The autoencoders compress the 250-energy bin spectra into a 10D latent space with less than 3% mean reconstruction error. The inverse DJINN models which map the 10D latent space to the 10D input space are not as accurate as necessary for mean-

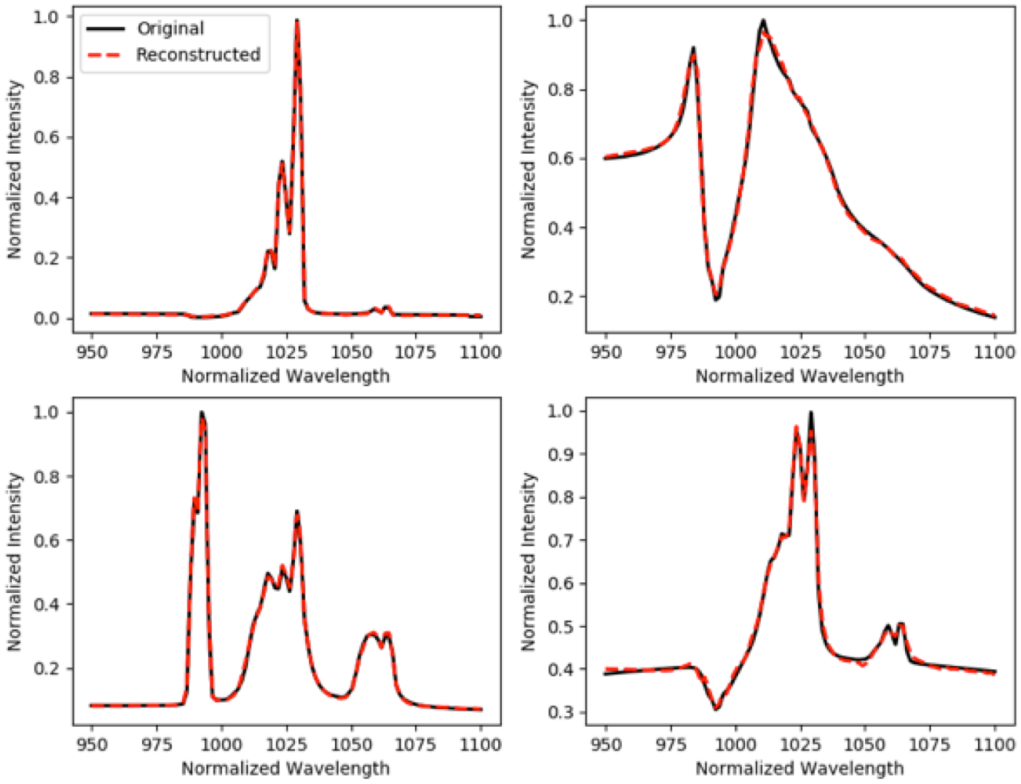


Figure 5.10: A few examples of the true Ge emission spectra (solid black) and the reconstructed spectra (dashed red) after passing through a 10D latent space autoencoder.

ingful applications to experimental data; however, recent improvements to the Cretin code could improve the quality of these models in the future. Figure 5.11 illustrates the quality of the inverse model predictions for the physics parameters given an input spectrum. The true values of the physics parameters are shown in black, the blue distributions are the DJINN predictions. DJINN is able to consistently predict many of the physics parameters accurately (such as tempatures, and mix regions), but is unable to predict features such as the filltube, or the outer radius of the fuel.

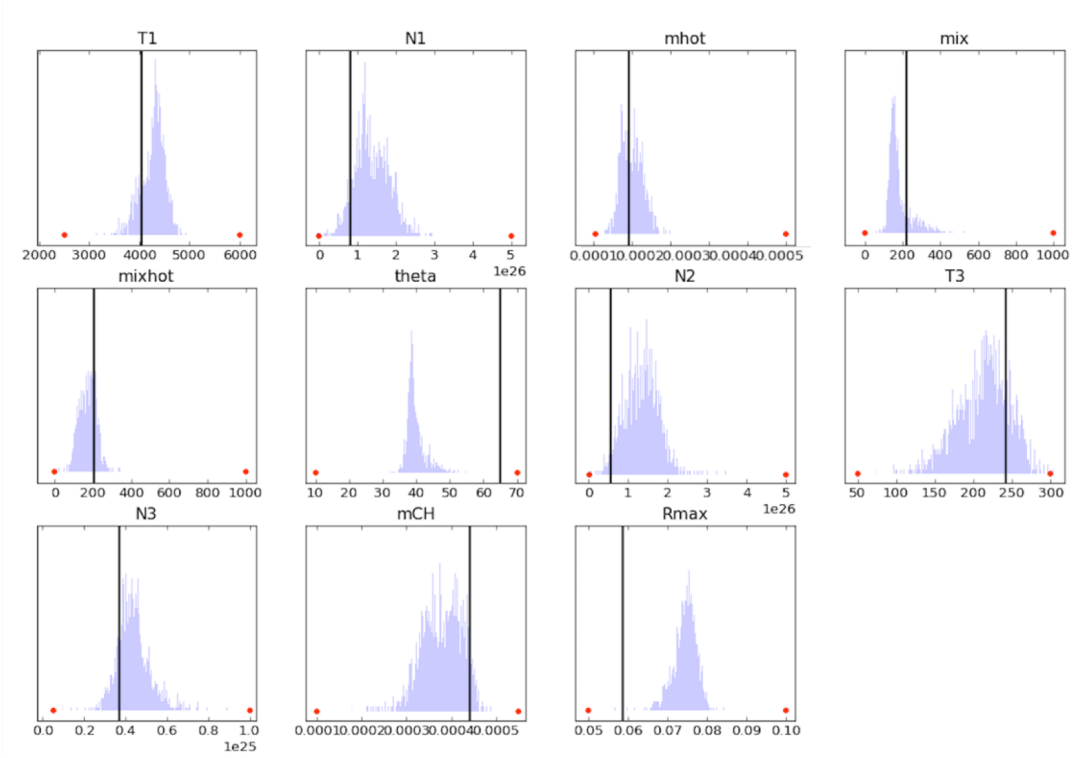


Figure 5.11: Example set of inferred inputs given the emission spectrum in the top left of Figure 5.10. N and T are the densities and temperatures in three distinct regions of the fuel, mhot is the mass of the hotspot, mix is the amount of mix in the region outside of the hotspot, mixhot is the mix in the hotspot, theta is the size of a filltube perturbation, mCH is the mass of the ablator, and Rmax is the outer radius. The red dots indicate the boundaries of the 10D space that the Cretin database spans, the true values of the parameters are indicated in black, and the blue distributions are the inferred values from the DJINN inverse model.

5.4 Conclusions

Ensembles of deep neural networks with dropout are used as approximate Bayesian models to perform post-shot analyses for inertial confinement fusion experiments. Inverse models, which map directly from output to input space, are compared to traditional MCMC sampling of the forward model for post-shot analysis. The inverse models suffer from large uncertainties when constrained by only a few outputs, but the benefits of using inverse models become evident when searching for post-shot simulations which match a large number of experimental outputs.

Autoencoder neural networks are used to compress a large number of observables into a lower dimensional representation of the data, and inverse models are trained to map from this low dimensional "latent" space to the simulation inputs directly. Autoencoder augmented post-shot analyses enable the best post-shot simulations to be found rapidly, by simply evaluating a neural network multiple times. This approach is much more efficient than MCMC sampling a forward model, which can be prohibitively expensive when searching high dimensional parameter spaces for solutions that are consistent with several dozen experimental outputs.

Neural network-based post-shot analyses offer a promising approach to interpreting ICF experiments, as they provide uncertainty estimates on the inferred simulation inputs, and thus can be used to establish the relative likelihoods of certain hypotheses explaining the experimental data. Furthermore, neural network-based inverse modeling can be applied to a variety of problems beyond ICF; as an example we have applied these technologies to other high energy density experimental platforms. However, such analyses will not yield reliable results if the simulation database does not properly encapsulate the experiments of interest. Advanced sampling techniques that specifically search for regions of parameter space that are consistent with the

experimental data could improve the design of databases upon which the surrogate models are trained for future post-shot analyses. Alternatively, it could be the case that the simulator is incapable of producing results consistent with observations from an experiment; in this case it might be more useful to pursue model calibration in order to make more predictive models; this is explored in the next chapter.

6. TRANSFER LEARNING TO MODEL INERTIAL CONFINEMENT FUSION EXPERIMENTS

In the previous chapter we presented neural network-based post-shot analysis of ICF and other HED experiments performed at NIF. The goal of a post-shot analysis is to find the simulation inputs that are most consistent with experimental measurements, carried out after the experiment has been performed. Post-shot analyses are often necessary for NIF experiments as most high fidelity simulations only model the capsule, and inputs for these simulations are not controllable or directly measurable in experiments. The primary unknown input for capsule-only simulations is the radiation drive. Hohlraum simulations produce an approximate drive for the capsule, but due to the expense of high resolution simulations, this drive is often an approximation of what occurs in experiments, and must be adjusted by inferring drive multipliers using post-shot techniques.

Post-shot analyses assume that the simulator is correct and capable of making predictions consistent with the experiment, *if* the correct inputs can be determined. This assumption does not always hold, and due to the vast number of inputs to the simulation and the large uncertainties in some experimental measurements, it is difficult to define what it means to find a set of inputs that are "consistent" with experimental observations. It is possible that the simulator is incorrect, and an exhaustive search for the best post-shot simulation will yield outputs that are inconsistent with at least some of the experimental measurements. Currently, well-tuned post-shot simulations often match on the order of 10 experimental scalars within the experimental uncertainties [33]. Post-shots are useful for understanding an experiment after the fact, but because simulation inputs are inferred on a shot-

by-shot basis and the inputs are not necessarily consistent across shots, they do not always aid in predicting the outcome of future experiments.

If instead we assume the simulator has error, we can take an alternative approach to creating a more predictive model. This approach is often referred to as model calibration – using experimental data to "calibrate" an inaccurate surrogate to produce a model which is more consistent with reality. Model calibration is a broadly-researched topic [139, 128, 12], with the most popular technique developed by Kennedy and O'Hagan [67]. In this approach, the true model ($Y_{true}(\mathbf{x})$) is assumed to be an additive combination of a simulator ($Y_{sim}(\mathbf{x})$) and an unknown discrepancy term ($\delta(\mathbf{x})$), as shown in Eq. 6.1, which must be determined with experimental data. The form of this discrepancy term must be specified by the user, however, and the complexity is limited by the amount of experimental data that is available.

$$Y_{true}(\mathbf{x}) = Y_{sim}(\mathbf{x}) + \delta(\mathbf{x}) \tag{6.1}$$

Researchers have explored Bayesian calibration with discrepancy terms for ICF data [33]; however, these techniques require the simulation and experiment share the same inputs. In the case of NIF data, a subset of experimental observables are used to infer the simulation inputs, while observables not included in this inference step are calibrated with a discrepancy term. This method suffers from high degrees of degeneracy, as the input parameter inference and discrepancy term calibration terms can compensate for one another, increasing overall uncertainties in predictions. In this work, we propose a non-parametric model calibration technique, referred to as "transfer learning", for calibrating ICF models.

Traditional machine learning models gain knowledge by observing large quantities of labeled data. For example, if the task is to classify photos of animals, a model

will need to be exposed to millions of labeled images of all the animals it is expected to classify, in a variety of different scenarios, colors, perspectives, etc, in order to classify the animals correctly. Supervised learning tasks are straightforward to solve when large quantities of data are available; however, many of these techniques break down when limited to small sets of labeled data.

Transfer learning is an alternative learning technique that can help overcome the challenge of training on small datasets. Transfer learning is a method for using knowledge gained while solving one problem, and applying it to a different, but related problem. This approach is most commonly used for image classification tasks [58, 101, 147] , for which there are many large databases of labeled images [26, 30, 47]. There are several neural networks that have already been trained on such datasets that are available for download, such as AlexNet [73] and Inception [132], which can take several weeks on dozens of GPUs to train from scratch. These neural networks have been studied extensively, and they appear to learn how to recognize images in a logical series of steps as the hidden layers of the network are traversed. First, the networks often search for edges in the images, then for simple geometric patterns, and eventually they begin to recognize specific characterstics, such as eyes, arms, ears, etc. In general, the deeper layers of the network focus on finer details in the images. It is thus expected that neural networks trained on any large image dataset are learning about features common to all images (edges, shapes, contrast) in the early layers of the model. It thus seems reasonable to take the first few layers of one of these pretrained neural networks, freezing them, and then retraining the last few layers of the network on a new set of images for a distinct classification task. The old frozen layers teach the network to "see"; the last few layers teach the network to "recognize" specific types of images. Transfer learning is the process of taking a network trained on a large dataset, freezing the first several layers of the network,

then retraining the last layers on a different, often smaller dataset. As an example relevant to ICF, researchers in the NIF optics group have used transfer learning on AlexNet and Inception to classify images of different types of damage that occur on the optics at NIF. There are not enough labeled optics images to train a network from scratch, but transfer learning on a network trained on ImageNet [26] has proved to be a huge success. This methodology has enabled the group to automate their damage inspection by letting the network process the images of optical componnets, rather than having an optics expert inspect each image manually [65].

In ICF it is often the case that we do not have enough experimental measurements to train a machine learning model on the experimental data alone. We are not interested in image classification for our implosions, so traditional transfer learning using a pre-trained open-source model is not appropriate. However, we do have the ability to create massive databases of ICF simulations, which we suspect are a good reflection of reality, but need to be tuned to be more consistent with experiments. We therefore propose the use of transfer learning as a non-parametric approach for calibrating a simulation-based neural network to experiments. The general approach is illustrated in Figure 6.1: train a neural network on simulation data to relate simulation inputs to observable outputs. Freeze many of the layers of the neural network, but leave some open for re-training. Retrain these available weights using the sparse set of experimental data for which the inputs and output observables are known. Similar to how image-based neural networks learn to "see" in early layers of the network, the first few layers on a network trained on ICF simulation data might learn the low-order shape of the response surface. The last few layers would then be minor perturbations to this surface, which can be adjusted to align with experimental observables when experimental data is limited.

We will test the feasibility of using transfer learning to produce neural networks

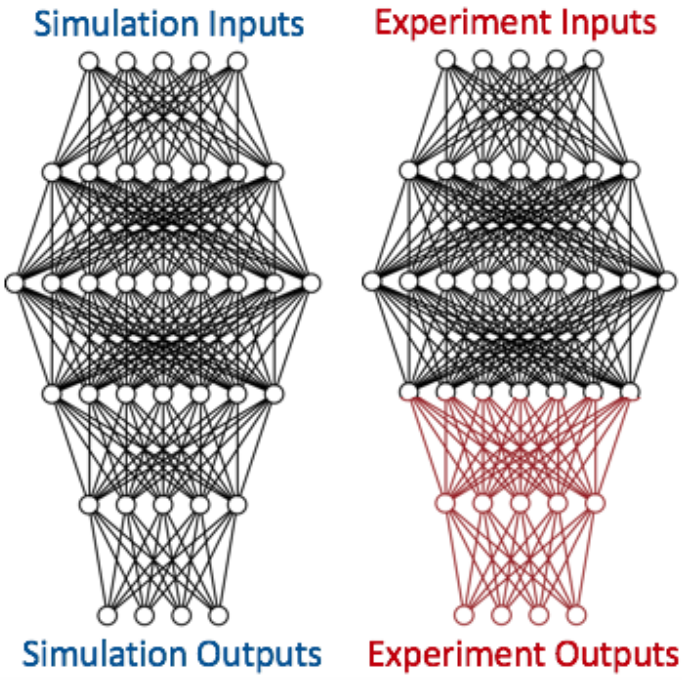


Figure 6.1: To transfer learn from simulations to experiments, the first three layers of the simulation-based network are frozen, and the remaining two layers are available for retraining with the experimental data.

that accurately predict the outcomes of ICF experiments. This idea requires the inputs and outputs be similar for both simulations and experiments, which makes it challenging to apply this technique to NIF datasets. We often model NIF experiments using capsule-only simulations, as they can be modeled with much higher resolution than integrated hohlraum simulations. However, capsule-only simulations have a set of inputs that are different from those specified in an experiment. In order to test transfer learning on NIF data, we would need to run an ensemble of hohlraum simulations which span the design space of multiple NIF experiments, however this would require exploring a 5D or larger design space, meaning we would need to run on the order of tens of thousands of hohlraum simulations. Since this would require massive computational resources, we opt to study the efficacy of transfer learning for ICF data using a dataset that is already available for analysis. This data is provided by researchers at the Laboratory of Laser Energetics (LLE), where they perform direct drive ICF experiments at the Omega laser facility [66]. In direct drive, the capsule-only simulations and experiments have the same inputs and can produce the same observable outputs, making calibration via transfer learning possible. Furthermore, the Omega facility has performed a large number of experiments [10] that span a modest design space, making it feasible to create the simulation database that spans the space of the experiments.

In section 6.1 we will introduce a proposed hierarchical approach to transfer learning for numerical simulation data. In section 6.2, we will compare standard and hierarchical transfer learning for the Omega dataset, and use the transfer learned models to study the discrepancies between simulations and experiments in section 6.3.

6.1 Hierarchical transfer learning

Computer simulations of complex physical systems are often modeled at varying levels of fidelity. Quick, low fidelity models are used to explore vast design spaces for optimal settings, and expensive high fidelity models might be used in interesting regions of design space to compute predictions of planned experiments. The high fidelity simulations are often more accurate and reliable than the inexpensive, approximate models, but the expense of running the simulation often prevents their use in large parameter scans. It might, however, be possible to create models that emulate high fidelity simulations with reduced computational cost by using transfer learning. For example, a model trained on a dense Latin hypercube sampled set of 1D simulations could be calibrated to a sparse Latin hypercube of 2D simulations that fill the same design space. Rather than running a dense Latin hypercube of 2D simulations to train a 2D surrogate model, one could obtain an accurate surrogate by transfer learning from 1D to 2D with a relatively small number of 2D simulations, saving substantial computational resources. Furthermore, this 2D-calibrated model can be subsequently calibrated to experimental data. If the 2D model is a better reflection of reality than the 1D model, the transfer learning step between 2D and the experiment should be easier to train than jumping from 1D directly to the experimental data. We refer to this technique of transfer learning from low to subsequently higher fidelity simulations to the experimental data as "hierarchical transfer learning".

To demonstrate the utility of hierarchical transfer learning, consider the simple function:

$$f(a, x) = xe^{ax}, \tag{6.2}$$

where x and a are random variables; x between [-1,1] and a between [0,1]. This expression will be the "experiment" or true function. We also have a low and high fidelity approximation of the experiment:

$$f_{low}(a, x) = x, \tag{6.3}$$

$$f_{high}(a, x) = x + ax^2, \tag{6.4}$$

where these are the first (low fidelity) and second (high fidelity) order Taylor expansions for the true function. This problem will be used to study the benefits of hierarchical transfer learning; specifically we are interested in whether or not stepping through the hierarchy results in better models than calibrating directly from low fidelity data to the experiments.

For this comparison, we begin with DJINN models with 3 hidden layers that take (x, a) as inputs and predict $f(x, a)$ or one of the approximations to $f(x, a)$. First we compute the average explained variance score (averaged over 5 random training/testing data splits of 80/20%) for DJINN models trained on experiments alone; this is the standard to which we will compare various transfer learning techniques, as we do not expect them to exceed the performance of a DJINN model trained purely on experimental data. Next, we transfer learn from high fidelity simulations to experiments, then from low fidelity simulations to experiments. Finally, we transfer learn from low to high fidelity, and then to experiments to test the hierarchical approach. The results are summarized in Table 6.2; the neural network hyperparameters are noted in Table 6.1 and are kept the same for all of the models.

For this simple example, there is not a statistically significant difference between a model trained exclusively on a large dataset of experiments, models that are trained

Table 6.1: Hyper-parameters for original model and transfer learning for the Taylor expansion example.

Original Model Parameters	
Number of models	5
Hidden layer widths	4-8-14; 4-7-15, 4-8-14; 4-7-15; 4-7-14
Learning rate	0.004
Batch size	50
Epochs	300
Transfer Learning Parameters	
Retrained layers	Last 2
Learning rate	0.0001
Batch size	1
Epochs	300

Table 6.2: Comparison of hierarchical and one-step transfer learning to direct modeling of experiments.

Model	Mean±SD Explained Variance	p-value
Train with 100 exp.	0.994 ± 0.004	-
Train with 100 high fid.; TL with 25 exp.	0.994 ± 0.006	0.957
Train with 100 low fid.; TL with 25 exp.	0.954 ± 0.041	0.016
Train with 100 low fid.; TL with 50 high fid. + TL with 25 exp.	0.981 ± 0.025	0.279

on high fidelity simulations and calibrated to experiments, and models that are hierarchically calibrated. However, these three models are statistically significantly better than the model which is calibrated directly from low fidelity to the experiment, illustrating that there is an advantage of informing the model of high fidelity data prior to experimental calibration. In order to make an accurate emulator of the experiments, the cost of each of these routes should be the determining factor in which method to choose; however, it is expected in most cases that the hierarchical method, which requires the least number of experiments and/or high fidelity simulations, is cheapest. For this simple example the computational cost is negligible, but for applications in which complex multi-physics systems are being modeled, the computational cost difference between low and high fidelity simulations can be substantial. For such systems experiments are also often costly and limited in number.

In Table 6.2 only a single size dataset is considered for the hierarchical model. To determine the minimum number of high fidelity simulations and experiments that is needed to get a model that is not significantly different than the baseline (experiment only) surrogate, we can compute the mean explained variance score as the dataset sizes are varied. First, we determine the minimum number of high-fidelity points that are required to produce a transfer-learned model that is of similar performance to a model trained exclusively on 100 high fidelity simulations. Then, we determine how many experiments are needed to recalibrate this model to be on par with the experiment-only baseline model. As in the models from Table 6.2, the number of points in the dataset includes training and testing data, which are split into 80/20% sets. The transfer learning parameters are held constant as recorded in Table 6.1, and each model starts with the same low fidelity surrogate trained on 80 data points and validated on the remaining 20 points.

Figure 6.2 illustrates how the transfer learning quality improves as the number

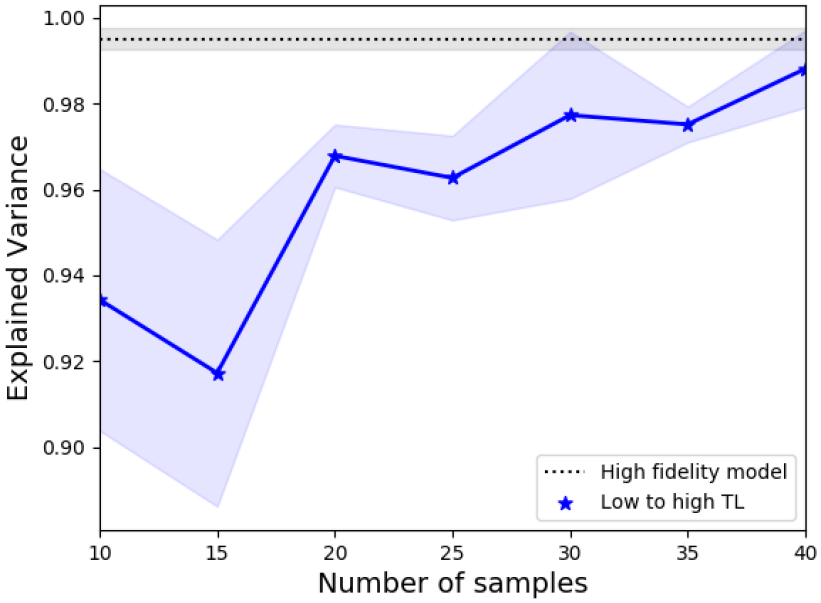


Figure 6.2: "High fidelity" prediction quality as the number of high fidelity data points used for transfer learning from low fidelity data is increased. 30-40 high fidelity data points with transfer learning produce a model that is comparable in quality to one trained exclusively on 100 high fidelity simulations.

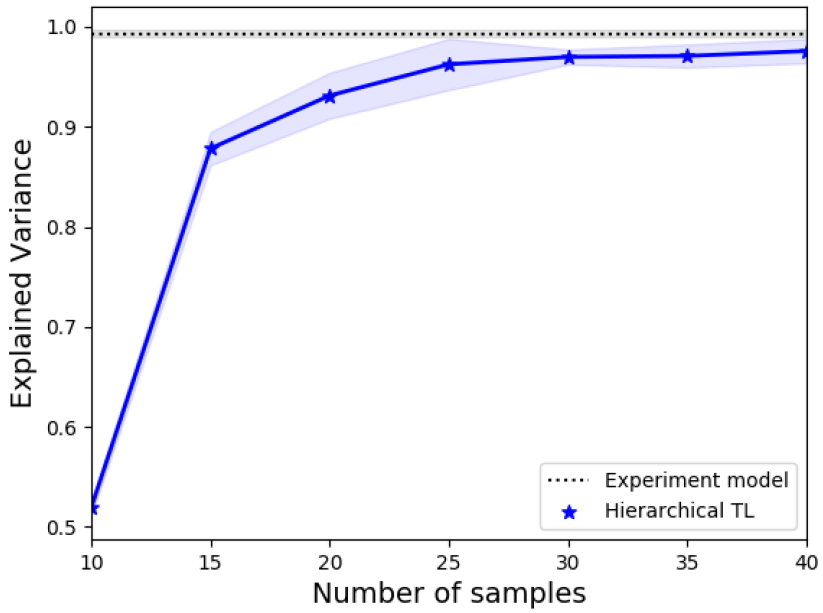


Figure 6.3: Experimental prediction accuracy as the number of experimental data points is increased in hierarchical transfer learning. Models are first trained on 100 low fidelity simulations, calibrated to high fidelity with 30 high fidelity data points, then subsequently calibrated to the experiments. The model quality converges with about 25 experiments, and is comparable to the performance of a model trained on 100 experiments alone.

of high fidelity data points is increased; the error bars reflect the variation in performance when the testing/training datasets are shuffled. Around 30 high fidelity samples, the transfer learned model begins to perform similarly to the model trained only on 100 high fidelity simulations. We use 30 high fidelity simulations in the hierarchical model, and next determine the minimum number of experiments necessary to subsequently calibrate the model to experiments. A model trained on 100 experimental values is taken as the baseline to which we are comparing the transfer learned model. Figure 6.3 illustrates the quality of the transfer learned model predictions as the number of experiments is varied. For this example, good performance is achieved with 25 experiments, beyond this the improvement in model quality is minimal.

Whether or not it is beneficial to perform heirarchical transfer learning depends on the relative expense of the varying levels of fidelity in simulations and the experiments. If running 100 low fidelity and 30 high fidelity simulations is cheaper than 100 high fidelity simulations, it is worth taking the hierarchical approach for subsequently calibrating to 25 experiments; however there may be situations in which running a high fidelity database is preferred. There are many other factors that could optimize performance of the hierarchical model, such as where the high fidelity simulations and experiments are placed in the design space; future work will explore how to optimize sampling for hierarchical transfer learning.

The previous results are for a very simple, low dimensional function that is fast to evalaute. In the next sections, we apply the same techniques to a real-world application: direct drive inertial confinement fusion experiments from the Omega laser facility.

6.2 Transfer learning for ICF model calibration

The database we use to test hierarchical transfer learning for ICF data is composed of 23 experiments from the Omega laser facility that roughly fall into a 9D design space. Researchers at Omega provided a database of 30k LILAC [25] simulations Latin hypercube sampled [87] in the 9D input space encompassing the experiments. The nine inputs include laser pulse parameters: the average drive, drive rise time, energy on the target, the first picket power, foot power, foot width, and foot picket width, and capsule geometry parameters: the ice thickness and the outer radius of the capsule. The 30k simulations are low fidelity; they are 1D, do not account of laser-plasma interactions (LPI), and use approximate equations of state. Each simulation takes about 10 wall-clock minutes to run. Each of the 23 experiments is accompanied by their best post-shot simulation, which is a high-fidelity LILAC simulation with LPI, more accurate equations of state, and nonlocal electron transport; these simulations require approximately 8-10 wall-clock hours to run. Both the low and high fidelity simulations produce a large number of scalar outputs; 19 of which are included in the following analysis. There are only 5 observables available for all 23 experiments that are common to the simulation database that will be used to test transfer learning with experimental data.

In section 6.2.1, we train the low fidelity simulation-based neural networks using DJINN. We will refer to the models trained only on the low fidelity LILAC simulations as "low fidelity DJINN" models. In section 6.2.2, we use transfer learning to calibrate from low fidelity to high fidelity (post-shot) simulations, and to experiments.

Table 6.3: Hyper-parameters for original model and transfer learning for the Omega dataset.

Original Model Parameters	
Number of models	5
Hidden layer widths	11-13-22-16; 11-14-19-26; 11-14-24-16; 11-15-29-30; 11-14-22-29
Learning rate	0.004
Batch size	1500
Epochs	400
Transfer Learning Parameters	
Retrained layers	Last 2
Learning rate	0.0003
Batch size	1
Epochs	2300

6.2.1 Low fidelity simulation-based DJINN models

The low-fidelity simulation database is used to train an ensemble of five low-fidelity DJINN models, which predict all 19 observables simultaneously and are individually cross-validated. The variance between DJINN models, each of which have been trained on a different random 80% subset of the data, will provide uncertainty estimates on the model predictions. The hyperparameters used to train the networks are summarized in Table 6.3. The mean explained variance score for each output is given in Table 6.4.

The same low fidelity DJINN models are used for standard (low fidelity simulations to experiments) and hierarchical transfer learning (low to high fidelity simulations to experiments).

6.2.2 Hierarchical transfer learning with the Omega dataset

For the Omega database, we compare the hierarchical approach, shown in Figure 6.4, to transfer learning directly from low fidelity simulations to experiments. We

Table 6.4: Mean explained variance scores on the test datasets for five DJINN models trained on random 80% subsets of the 30k low-fidelity LILAC simulations.

AbsorptionFraction: 0.991	PeakKineticEnergy: 0.981	Tion_DD: 0.959
Adiabat: 0.886	Pressure: 0.958	Vi: 0.971
BW: 0.869	R0: 0.962	Yield: 0.944
BangTime: 0.990	RhoR: 0.968	Yield_DD: 0.949
ConvergenceInner: 0.964	ShockMass: 0.856	Rhomaxbt: 0.962
ConvergenceOuter: 0.962	Rhonave: 0.967	
IFAR: 0.885	Tion: 0.956	

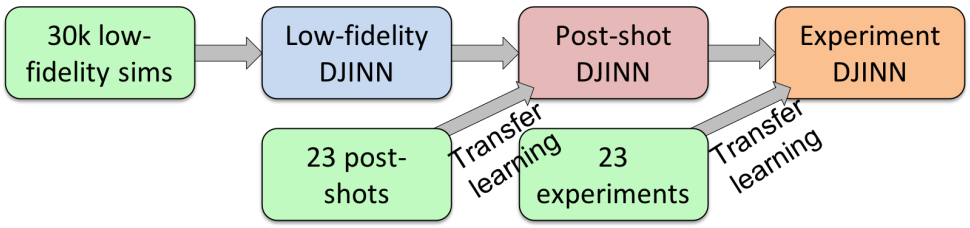


Figure 6.4: Hierarchical transfer learning work flow for the Omega data. DJINN models are trained on the low fidelity simulation database. They are next calibrated to the high fidelity post-shot simulations via transfer learning and are subsequently calibrated to the experiments. This approach is compared to direct transfer learning from low-fidelity simulations to experiments, to evaluate the advantages high-fidelity simulations might offer.

refer to the models transfer learned to post-shot simulations as "post-shot DJINN" models, and those that are subsequently transfer learned to experiments as "experiment DJINN" models. If the post-shots are more accurate depictions of reality than the low fidelity simulations, priming the DJINN model with post-shot information prior to calibrating to the experiments could improve the ability of the model to adapt to the experimental data, as in the example of Section 6.1.

The low fidelity DJINN models described in the previous section are calibrated

independently, each on a different random subset of the post-shot or experimental data. For each of the models, the first three layers of weights are frozen, and the remaining two layers are available for retraining, as shown in the cartoon in Figure 6.1. Note that the architecture of the networks is not reflected in this cartoon; the true architectures are given in Table 6.5 for the ensemble of five DJINN models.

The last two layers of weights are retrained to convergence for 2000 epochs with a batch size of one, and a learning rate of 0.0003 in the Adam optimizer. Each model is trained on a random 90% of the experimental data (20 points) and tested on the remaining 10% (3 points). The low fidelity and post-shot simulations have 19 outputs, but the experiments only have 5 available observables. To calibrate to the experiments, the cost function, which is the MSE of the 19 scaled outputs, is modified such that the missing 14 outputs are not weighted. More explicitly, the cost becomes a weighted MSE where the weights are zero for outputs not measured in the experiment, and unity for those that are observed in the experiment. The predictions for the remaining 14 observables are not constrained by the experimental data and could change in non-physically motivated ways, thus we will focus only on the 5 available observables. An equivalent approach to the weighted cost function would be to train the low fidelity and post-shot DJINN models with only the 5 outputs available in the experiment. We choose to retain all 19 outputs so we can evaluate the accuracy of the post-shot calibration for all 19 observables in the hierarchical modeling approach. Note that all input and output data is scaled [0,1], using the parameter ranges set by the database of 30k simulations, prior to training. This prevents the cost function from being biased toward outputs that are larger in magnitude due to the choice of units.

First we consider transfer learning from the low fidelity simulations to the post-shot simulations. Figure 6.5 illustrates the prediction error (calculated on training

Table 6.5: Architectures of DJINN models trained on the Omega databases. There are nine inputs and nineteen outputs for the baseline, low fidelity models.

Architectures
(9, 10, 13, 20, 20, 19)
(9, 10, 11, 23, 21, 19)
(9, 11, 11, 13, 20, 19)
(9, 11, 12, 19, 25, 19)
(9, 10, 13, 22, 25, 19)

and testing data combined), computed as:

$$\text{Error} = \frac{\text{Prediction} - \text{Post shot}}{\text{Post shot}}, \tag{6.5}$$

for all nineteen available outputs for the uncalibrated (low fidelity) and calibrated (post-shot) DJINN models. The error bars reflect the standard deviation in prediction errors from the ensemble of DJINN models; the points on Fig. 6.5 illustrate the mean error.

The low fidelity and post-shot simulations differ significantly in their predictions of the nineteen observables, shown by the error in the blue points of Figure 6.5. The mean error of less than 5% in the red points indicates that the networks are able to successfully learn the post-shot outputs via transfer learning.

The post-shot calibrated models are next calibrated to the experimental data, again by transfer learning the last two layers of the network – the same layers that were modified to calibrate to the post-shot data. The mean and standard deviation of final prediction error is now computed using the experiment as the ground truth:

$$\text{Error} = \frac{\text{Prediction} - \text{Experiment}}{\text{Experiment}}. \tag{6.6}$$

The prediction quality is illustrated in Fig. 6.6 for the low-fidelity, post-shot, and

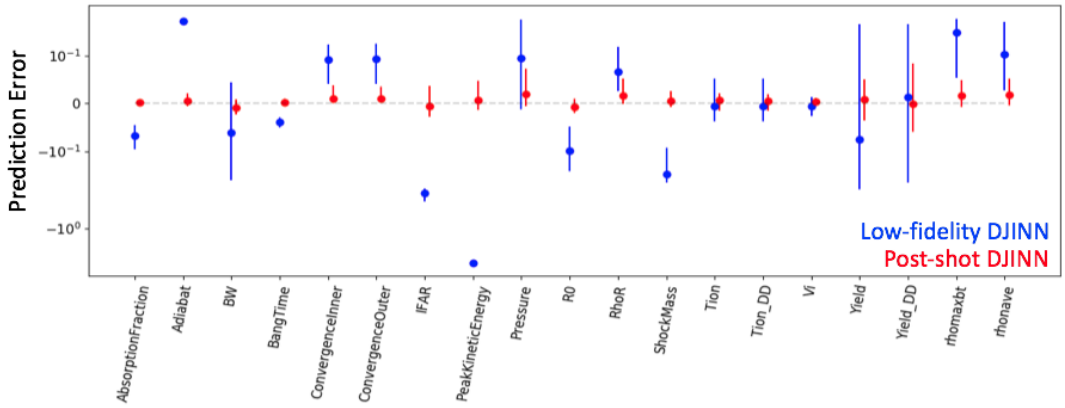


Figure 6.5: Prediction error (with the post-shot as the ground truth) for calibrated and uncalibrated DJINN models. The low fidelity model predicts post-shot observables with significant error, as the models contain different physics. The models calibrated to post-shot data are able to predict all 19 post-shot observables with high accuracy.

experiment DJINN models.

Figure 6.6 illustrates that the post-shot is not necessarily more predictive of reality than the low fidelity simulations; however, transfer learning to experiments is still successful. The largest error for transfer learning is in the yield, perhaps due to the model needing to adjust its predictions by over an order of magnitude for most experiments, however the mean prediction error for all observables is close to zero.

Since the post-shot models are no closer to the experiments than the low fidelity models, hierarchical modeling does not offer significant benefits for this dataset; comparable results are achieved by transfer learning directly from low fidelity simulations to the experiments. Table 6.6 records the average explained variance ratio for each the experimental observables for the hierarchical models, and those that calibrated directly from low fidelity to experiments. The explained variance ratios are computed on the test dataset, and are averaged for the five models.

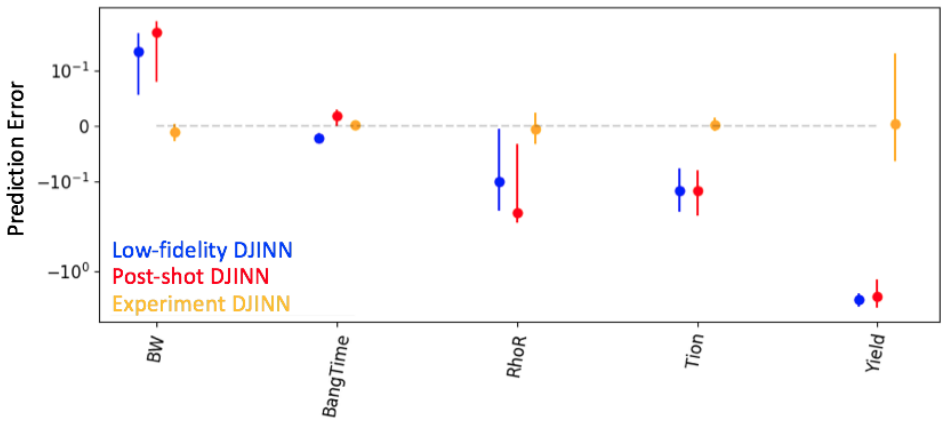


Figure 6.6: Prediction error (with the experiment as the ground truth) for low fidelity DJINN models, DJINN models calibrated to the post-shot data, and DJINN models calibrated to the experimental data. The experimentally calibrated models predict the five experimental observables with high accuracy.

Table 6.6: Explained variance scores for models calibrated from low fidelity to post-shot simulations to experimental data, and for models calibrated directly from low fidelity simulations to experiments. The post-shot simulations are not an accurate picture of reality, and thus there are no significant benefits of first calibrating to the post-shot data for this particular dataset.

	Explained Variance (Mean +/- Std)		
Observable	Hierarchical	Low fid. - Exp.	p-value
Burnwidth	0.975 ± 0.023	0.889 ± 0.079	0.139
Bangtime	0.987 ± 0.015	0.942 ± 0.092	0.364
ρR	0.874 ± 0.097	0.835 ± 0.179	0.712
T_{ion}	0.988 ± 0.009	0.924 ± 0.094	0.211
Yield	0.818 ± 0.143	0.956 ± 0.034	0.096

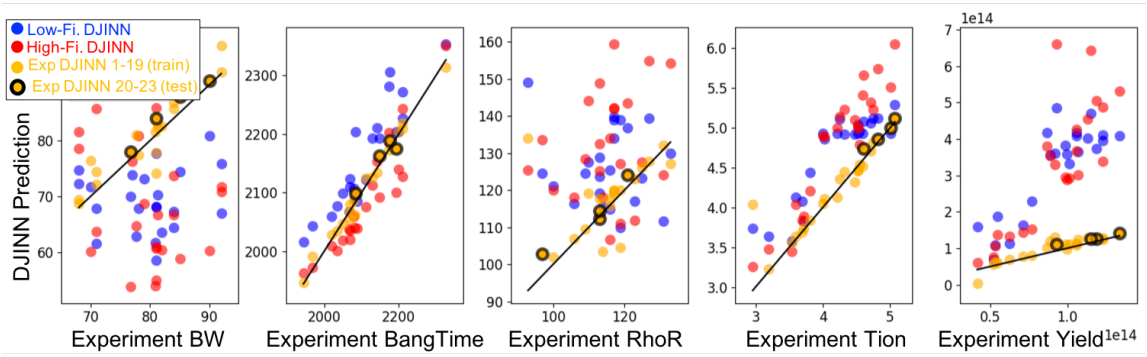


Figure 6.7: Actual experimental observations plotted against the three DJINN models predictions. The blue and red points indicate the low and high fidelity model predictions, respectively; both of which have high prediction error. Predictions of the four most recent experiments are shown in yellow circled in bold black, after transfer learning using previous experimental data (uncircled yellow). The experiment-calibrated model is able to accurately predict future experiments with significantly higher accuracy than the simulation-based models.

The previous analyses always involve randomly choosing the training and testing data for model calibration. To illustrate how these models can be used to predict the outcomes of future experiments, we take the models calibrated to the post-shot data (which is just a high fidelity calculation at desired experimental locations in design space) and calibrate this model to experiments using only the oldest 19 experiments in the data set. We then test the models on the 4 most recent experiments; the predictions are shown in Figure 6.7. Training on the old data and predicting the four most recent experiments requires the model to extrapolate in input space, away from the old experimental data. The model is able to successfully predict the outcomes of the newest four experiments, demonstrating it does have the ability to successfully extrapolate away from the training data.

6.3 Exploring discrepancies between simulations and Omega experiments with transfer learning

A result of hierarchical transfer learning is that we now have models that emulate low fidelity simulations, post-shot simulations, and experiments. We can use these three models to explore the 9D design space and study the discrepancies between the two types of simulations and the experiments.

A primary use of ICF implosion simulations is to find optimal design settings for experiments. An interesting application of the three models is thus to search for "optimal" designs using each fidelity surrogate and determine if the simulation-based models suggest a similar "optimal" design as the experiment-informed model. For this exercise, we define an optimal design as one that maximizes the experimental ignition threshold factor (ITFX):

$$\text{ITFX} \propto \text{Yield} \cdot \rho\text{R}^2. \tag{6.7}$$

where ρR is the areal density. The resulting optimal designs are illustrated in Fig. 6.8.

There are several important differences between the three optimal designs. First, consider the differences between the low fidelity (blue) and post-shot (red) designs. The low fidelity design prefers high compression of a thick capsule, and achieves this by driving the capsule with a very high power. This differs from the high fidelity design, which prefers a lower power and thinner shell in order to achieve a similar implosion velocity and yield. The high fidelity design includes LPI effects, so it is reasonable for this design to lower the peak power to avoid LPI [84]. Next consider the experimental design: this design adjusts the picket and the foot of the pulse. Lowering the foot of the pulse (around 0.75 ns) lowers the adiabat inside the shell, allowing for higher compression and therefore higher areal density. To

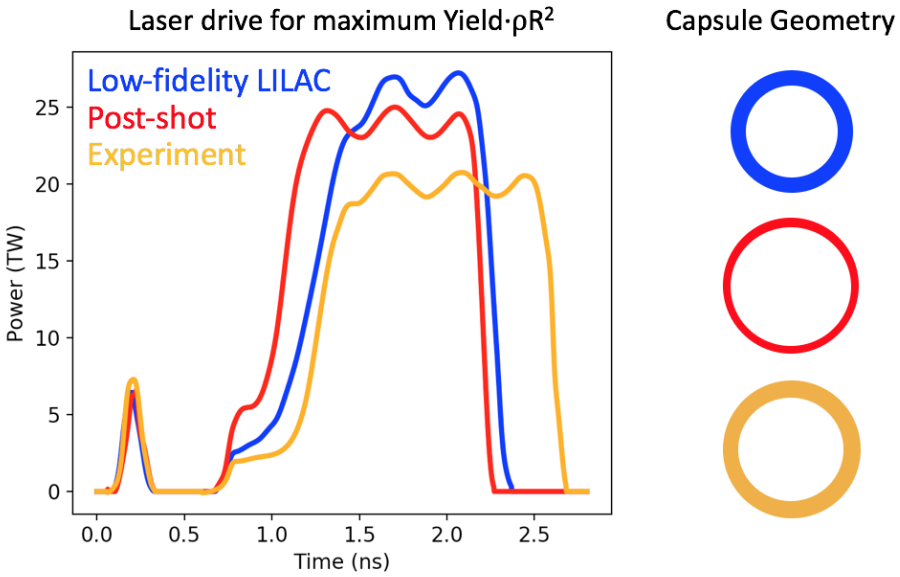


Figure 6.8: Designs which optimize ITFX according to the low fidelity, post-shot, and experiment DJINN models. The three designs are distinct due to the lack of accurate physics models, asymmetries, and other experimental sources of performance degradation not included in the simulations.

Table 6.7: Predictions of the low fidelity, high fidelity, and experiment DJINN models at the point of optimal Yield$(\rho R)^2$ according to each of the three models.

log10(Yield$(\rho R)^2$) at:	Low Fi. Optimal	High Fi. Optimal	Exp. Optimal
Low Fi. DJINN	19.764	19.586	19.570
High Fi. DJINN	19.421	19.598	19.273
Exp. DJINN	18.951	19.019	19.070

mitigate instabilities associated with higher compression, the picket of the pulse is increased to increase the adiabat on the outer surface of the shell. The higher adibat stabilizes hydrodynamic instbailites by increasing the ablation velocity [42, 71, 3]. The experimental capsule is thicker and is driven at an even lower power than the post shot for a longer period of time; perhaps due to underestimation of the LPI effects by the post-shot simulations.

It is interesting to compare the predictions for each fidelity of DJINN model for the three optimal designs, given in Table 6.7. The low fidelity model optimal design is expected to perform extremely well according to the low fidelity model, but the experiment-calibrated model expects it to perform 24% worse than the experimental optimal. Thus, relying on simulations alone to search for optimal designs leads to incorrect conclusions about how to optimize the design.

The maximum ITFX design according to the experiment-calibrated model is consistent with other analyses of this database [45, 44, 10]. The researchers at Omega have been training power law-based models to relate simulation outputs and experimental measurements; through this process they found that to optimize yield, they should increase the thickness of the ice in the capsule. A series of shots confirmed this prediction; they used the power law models to predict their next experiment, each time increasing the thickness of the capsule by just a few percent, then using the new experimental data to update their model before designing the next experiment.

In this process their goal was to maximize yield only, and a separate campaign was developed to optimize areal density (ρR) independently, by modifying the picket and foot of the pulse, which sets and shapes the adiabat of the implosion. After optimizing the areal density, they plan to use an iterative procedure to optimize yield and areal density together– targeting yield through capsule geometry and ρR through the laser pulse. This approach is largely physics-guided, and treats the pulse and capsule independently to independently optimize yield and areal density seperately, and will not explicitly account for interaction terms between the pulse and capsule geometry.

The neural network-based optimization can consider nonlinear interactions between the inputs, unlike power laws, this optimization procedure tunes the pulse and capsule simultaneously to maximize ITFX. However, the neural networks might be inaccurate far from the experimental data, thus caution should be taken to make small extrapolations from the data with this technique as well. However, the fact that two independent and very distinct methods for combining simulation and experimental data leads to similar suggestions for optimizing performance is encouraging, and shows that there is promise in using transfer learning to calibrate ICF simulations to experiments.

6.4 Conclusions

Transfer learning with deep jointly-informed neural networks has enabled the creation of surrogate models which emulate more expensive simulations and experiments, without requiring massive quantities of expensive data. Transfer learning uses low fidelity simulations to learn the approximate responses surfaces, then uses a sparse collection of expensive high fidelity simulations or experimental data to modify a limited number of weights in the network. The resulting networks accu-

rately emulate the expensive data, without requiring a large database of high fidelity simulations or experiments to train the neural network from scratch. Hierarchical transfer learning, the process of calibrating from low to high fidelity simulations to experiments (or between levels of fidelity of simulations) enables the creation of accurate emulators for the highest fidelity simulations or experiments, with lower computational cost than creating a neural network on the highest fidelity data alone. Transfer learning with DJINN enables the creation of neural network models that are predictive of direct drive ICF experiments at the Omega laser facility, which are used to design optimal implosions for future experiments.

Applying this methodology to indirect drive ICF experiments at NIF presents additional challenges. In order to perform transfer learning with NIF data, ensembles of hohlraum simulations are required, as the inputs for the hohlraum simulation and the experiments are the same. Hohlraum simulations are significantly more expensive than the capsule-only simulations used for the Omega simulations, and thus only small parameter spaces (fewer than 5 parameters) can be mapped out with a modest number of simulations (fewer than 50k). However, there are few NIF experiments that span such a small dimensional parameter space; several parmeters are changed from one experiment to the next, and there may only be a few (fewer than 10) experiments that span a 5D design space. Thus to perform transfer learning for NIF successfully, a series of (10 or more) experiments that span a 5D or fewer space must be performed. Alternatively, constructing a model that maps from the real experimental input space to capsule simulation inputs is being considered; this mapping from experimental inputs to capsule inputs could be learned with hohlraum simulations. The model that maps from experimental inputs to capsule inputs can be joined with the model than maps from capsule inputs to observables, and the combined model can be adjusted via transfer learning with experimental data. This

would still require an estimated ten or more experiments for a 5D or fewer space.

7. PREDICTING THE TIME-EVOLUTION OF MULTI-PHYSICS SYSTEMS WITH SEQUENCE-TO-SEQUENCE MODELS

The previous chapters focus primarily on end-time scalar quantities of interest from ICF simulations and experiments. However, much of the diagnostic data collected in ICF experiments is time-dependent and comes in the form of images, spectra, and scalars that are often measured from multiple angles. Incorporating the large amount of multimodal data collected in experiments into predictive models can better constrain the predictions to be more consistent with reality. The use of autoencoders for spectral data analysis is explored in Chapter 5, and images can be compressed and incorporated into feed forward neural networks in a similar fashion, but time-series data has not been incorporated into these analyses. Much of the data collected during ICF experiments are time-dependent; incorporating this information into the models can better constrain their predictions.

Feed forward neural networks (FFNNs) are not formulated to take advantage of correlations in sequential data, and they are only amenable to fixed-length vector data. In many multi-physics simulations time series data or data from discretized meshes often varies in size between simulations depending on the boundary conditions. For example, in hydrodynamics codes, simulations with slightly different initial conditions can require a different number of time steps and spatial discretization cells, as the resolution of the simulation can dynamically change to ensure physical processes are modeled correctly [107, 90]. Furthermore, the number of time points and the specific time at which data are collected in experiments is also often variable. To incorporate such data into predictive models, we need networks which are able to handle arbitrary length sequential data.

FFNNs have been successfully applied to sequential data by training the model to learn the "state transition" for the quantities of interest from one time step to the next [148]. The model can be iterated upon to predict the entire evolution of QOIs by using the prediction from one time step as the input to the model for the next time step. A trained state transition model can thus be given the initial conditions of a system and predict the entire trajectory of the QOIs. Although this allows for predictions of arbitrary sequence length, state transition models are incapable of learning long-term relationships in the data.

Recurrent neural networks (RNNs) are simple generalizations of FFNNs that allow the network to retain memory of its previous states, making them attractive options for modeling sequential data [88]. In an RNN, a neuron is replaced with a "cell"; long short-term memory (LSTM) cells [129, 39] and gated recurrent units (GRU) [21, 20] are commonly used cell structures as they are particularly well-suited for learning long-range dependencies in the data. RNNs are capable of mapping arbitrary length input sequences to arbitrary length output sequences. These models are applied to a wide range of problems, from image captioning to machine translation [76, 146, 141]. In this work, we focus on sequence-to-sequence (seq2seq) architectures, originally applied to machine translation [131, 81].

Standard RNN architectures can map input sequences to output sequences when the alignment between inputs and outputs is known; however, it is unclear how to handle situations in which the input and output sequences have lengths which differ from one example to the next within the same set of training data, such as variable time step simulation data. Seq2seq models handle variable sequence lengths by mapping an input sequence into a fixed-length "latent" vector via an "encoder" network. The latent vector is then mapped into a variable length output sequence via a "decoder". For example, in language translation the input sequence could be an English

phrase, which is encoded into a latent "thought vector" that captures the meaning of the phrase, and is then decoded into the equivalent French phrase. For time series data, the input sequence is the first several time steps in the evolution of a system, which gets encoded into the fixed-dimensional latent space. The latent space representation of the data is decoded to produce a series of predictions for the subsequent evolution of the system. This technique has been applied to simple time series with success (such as oscillating or monotonic functions or modest dimensionality), but has not been documented for real world applications [18].

The architecture of a seq2seq model, unrolled to illustrate each step of the input and output sequences, is shown in Fig. 7.1, where the blue and red cells represent a stack of LSTM or GRU cells for the encoder and decoder, respectively. The observed portion of the time series is input to the encoder (blue cells), and the cell state is passed from one time step to the next until this information arrives at the latent space– a fixed-dimensional vector that represents the entire input time series. The latent vector is then decoded to produce the output prediction for the unobserved portion of the time series. The decoder (red cells) predicts the output sequence one unit at a time, using its prediction for the current values of the QOIs and the cell state as inputs to predict the next value of the QOIs.

The ability of seq2seq models to handle arbitrary and variable length sequential data makes them attractive for emulating dynamic multi-physics simulations that evolve in a variable number of time steps within the same data set. Such simulations can be computationally expensive, making emulators that can be evaluated rapidly an attractive option for applications such as design optimization or hypothesis testing.

We are also interested in using seq2seq models to efficiently explore the parameter space of computationally expensive multi-physics codes. In this application,

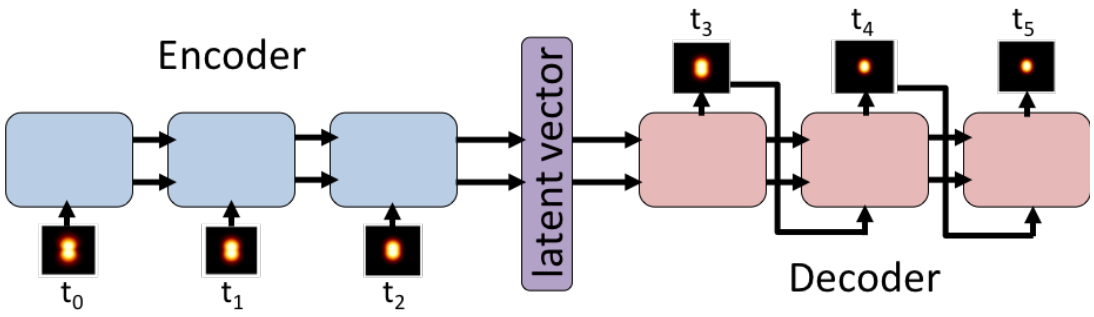


Figure 7.1: Unrolled seq2seq architecture. The blue and red cells represent stacks of recurrent cells (such as GRUs) that make up the encoder and decoder, respectively. The input image time series is compressed into a latent vector via the encoder, which is then decoded to make a prediction for the output image time series.

a seq2seq model trained on a set of simulations that are scattered throughout the parameter space is evaluated on-the-fly while running simulations at new locations. If the seq2seq model predicts the time evolution of the QOIs in the new simulations accurately, the predictable simulations are terminated in flight, and computational resources are allocated to regions of parameter space that are less predictable; this can result in significant computational savings for expensive simulations. This idea has been explored with state transition models [15], in which the author efficiently trained a state transition model by terminating simulations that were predictable, and investing computer resources in areas of parameter space where the state transition model did not perform well.

In this work, we evaluate the ability of seq2seq models to accurately emulate dynamic multi-physics simulations. We compare seq2seq models to state transition models for datasets generated with multi-physics codes that vary in complexity, and produce QOIs with dimensionality that spans several orders of magnitude. The datasets are introduced in section 7.1, followed by the performance comparison in

section 7.2. The disadvantages of state transition models are highlighted, along with the benefits seq2seq models offer for high-dimensional QOIs. Applications of seq2seq models for physics simulation data are discussed in section 7.3, including the benefits seq2seq models could offer for advanced sampling techniques, and how they can be used for numerical convergence studies.

7.1 Predicting the time evolution of multi-physics systems

In the following sections, seq2seq models are compared to neural network state transition models for three datasets generated with computer simulations of varying complexity. The multi-physics codes and corresponding datasets are presented in section 7.1.1, followed by the performance comparison in section 7.2.

7.1.1 Databases

Seq2seq models are compared to state transition models for three sets of simulation data of increasing complexity. The databases are generated using a 1D time-dependent diffusion code, a 1D Lagrangian radiation-hydrodynamics code, and a 3D multi-physics code that simulates inertial confinement fusion implosions. Along with the increasing complexity of the underlying physics models, the outputs generated in each code also become more complicated. In the following subsections, each dataset and the codes used to generate them are described in detail.

7.1.1.1 1D Diffusion

The diffusion database contains solutions to a simple 1D diffusion problem that determines the spatially-integrated concentration in a system as a function of time. The diffusion equation is given by Eq, 7.1:

$$\frac{\partial}{\partial t}\phi - \nabla D \nabla \phi = Q, \tag{7.1}$$

where D is the diffusion coefficient, ϕ is the concentration, and Q is a source.

The time evolution of the concentration is found by solving a discretized diffusion equation, Eq 7.2, on a spatial grid of arbitrary resolution, dx, with time step dt. The resolution of the grid has a significant impact on the accuracy of the solution; a coarse grid will have high error, and as the grid is refined the solution will approach the true answer at an asymptotic rate.

$$\frac{\phi^n - \phi^{n-1}}{\Delta t} - \frac{D}{\Delta x^2}(\phi^n_{i+1} - 2\phi^n_i + \phi^n_{i-1}) = Q^n_i, \tag{7.2}$$

where i, n indicate the mesh indices for space and time, respectively.

To generate a database of diffusion solutions, the problem is set up as follows: the spatially-integrated concentration as a function of time is solved for in a 1D slab of unit length with a constant diffusion coefficient D. There is a source with value unity at the left boundary, which is turned on at time t=0. This problem has an analytic solution, given by Eq. 7.3:

$$y(t) = 1 - 2\sqrt{Dt/\pi} \cdot e^{1/(2\sqrt{Dt})^2} - \text{erf}\left(\frac{1}{2\sqrt{Dt}}\right). \tag{7.3}$$

An implicit finite difference solver (centered-difference in space, backward Euler in time [108]) is used to compute the total concentration for varying values of the diffusion coefficient (D, between 1.0 and 3.0) and the spatial discretization step size (dx between 1.0e-3 and 1.0e-5). The spatially-integrated concentration as a function of time is recorded for 1000 time steps (step size of 1.0e-6) for 1000 random combinations of D and dx.

7.1.1.2 1D radiation hydrodynamics

The radiation hydrodynamics dataset tests the ability of the models to predict high-dimensional time histories generated by a more expensive multi-physics code with non-analytic solutions. In this task, the models are trained to predict the spatial and temporal evolution of multiple QOIs for systems described by the radiation hydrodynamics equations with grey diffusion (RHGD) [107], given by Eqs. 7.4-7.7:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \overrightarrow{u}) = 0, \tag{7.4}$$

$$\frac{\partial}{\partial t}(\rho \overrightarrow{u}) + \nabla \cdot (\rho \overrightarrow{u} \otimes \overrightarrow{u}) + \nabla p = -\frac{1}{3}\nabla \varepsilon, \tag{7.5}$$

$$\frac{\partial}{\partial t}(\frac{1}{2}\rho u^2 + \rho e) + \nabla \cdot ((\frac{1}{2}\rho u^2 + \rho e + p)\overrightarrow{u}) = \sigma_a c(\varepsilon - aT^4) - \frac{1}{3}\nabla \varepsilon \cdot \overrightarrow{u} \tag{7.6}$$

$$\frac{\partial \varepsilon}{\partial t} - \nabla \cdot \frac{c}{3\langle \sigma_t \rangle}\nabla \varepsilon + \frac{4}{3}\nabla \cdot \varepsilon \overrightarrow{u} = \sigma_a c(aT^4 - \varepsilon) + \frac{1}{3}\nabla \varepsilon \cdot \overrightarrow{u} \tag{7.7}$$

where ρ is the density, $\overrightarrow{u}$ is the velocity, p is the pressure, ε is the radiation energy density, e is the specific internal energy, a is a constant, c is the speed of light, T is the temperature, and σ_a, σ_t are the absorption and total opacities, respectively.

A Lagrangian RHGD code is used to generate solutions on a 1D grid of 100 spatial cells for 100 time steps. Three parameters are varied to generate 500 Latin hypercube-sampled [87] data points: the initial temperature is varied between 1.0e-6 and 0.01, the initial density between 0.75 and 1.25, and the temperature of a source on the left boundary of the slab between 0.1 and 0.6. The seq2seq models are trained

to predict the evolution of the temperature, pressure, and density as a function of time and position on the grid, totaling 300 time-varying parameters.

7.1.1.3 3D inertial confinement fusion implosion simulations

The final dataset used to compare state transition and seq2seq models is generated with a semi-analytic ICF implosion code [135, 79]. Three parameters are varied to create a database of 300 simulations. The parameters are applied shape asymmetry multipliers expressed as spherical harmonic modes (Y_l^m for l=2, m=[-1,0,1]) which vary in magnitude from [1.0,1.5], and degrade the implosion performance by distorting the shell of the ICF fuel capsule. The code generates multiple outputs; for this application the primary QOI is the X-ray emission image of the imploding capsule as a function of time. The images are 64x64 pixels with one color channel, and are recorded at 10 points in time throughout the implosion.

7.2 Comparison of seq2seq and state transition models for multi-physics systems

The three datasets are used to train state transition models and seq2seq models to predict the time evolution of the QOIs given only the initial conditions. Seq2seq models receive an "input" sequence that contains the initial condition only (the simulation inputs and the QOIs at the first time step), and the "output" sequence is the subsequent evolution of the QOIs (the simulation inputs, which are unchanged, and the evolving QOIs for the remaining time steps). The state transition models are trained using input vectors that include the simulation inputs and the QOIs at a time step t, and the corresponding output vectors contain the QOIs at time step $t + 1$. For evaluation, the state transition model is provided the simulation inputs and the initial value of the QOIs to predict the state at the first time step, and this solution is iterated upon to generate the complete time series.

The state transition model is a fully connected "deep jointly-informed neural net-

work" (DJINN) [61] from Chapter 3. The DJINN models are trained with optimized hyper-parameters chosen by the software; these settings are summarized in Table 7.1. The seq2seq model architectures are similar for all three datasets. The encoder and decoder share the same architecture, which is composed of two stacked GRU cells with a fixed number of hidden units per cell. The learning rate, batch size, and number of training epochs are chosen to optimize the performance on the test dataset; these parameters are also summarized in Table 7.1.

Each of the three datasets is split into an 80% training set and 20% test set, the latter is used to compare the performance of the two models. For the state transition models, the full sequences are sectioned into two time step slices (transitions), with time step t as an input, and $t+1$ as the corresponding output. For the seq2seq models, each sequence is a single training example with time step $t=0$ as the input, and time steps $t = [1, t_{final}]$ as the output sequence. Each time series is scaled between [0,1] prior to training the networks.

To compare the performance of the two models, the integrated absolute error (IAE) is computed, normalized by the sequence length, N_s:

$$IAE = \frac{1}{N_s} \sum_{n=1}^{N_s} |Y_n^{true} - Y_n^{pred}|, \tag{7.8}$$

where Y_n^{pred} and Y_n^{true} are the predicted and true values of the QOI at sequence step n, respectively. Reported in Table 7.1 are the mean, median, and standard deviation for test set IAE.

The seq2seq model consistently out-performs the state transition model, particularly as the dimensionality of the QOIs increases. To illustrate the quality of predictions from the seq2seq and state transition models, Figure 7.2 shows an example

Table 7.1: Model hyper-parameters and performance metrics. The state transition model, DJINN, is a fully connected neural network with the neurons per layer specified in the architecture column (this includes input and output layers). The seq2seq models are composed of two stacked GRU cells with the number of hidden units per cell specified in the architecture column. The batch size is specified in units of transitions (trans) for the state transition model, and sequences (seq) for the seq2seq model.

Dataset	Model	Architecture	Learn rate	Batch size	# Epochs	Mean IAE	Median IAE	SD IAE
Diffusion	DJINN	3,4,8,13,1	0.006	4000 trans	400	0.0329	0.0327	0.0059
	seq2seq	14 units	0.001	20 seq	3000	0.0257	0.0243	0.0086
Rad-hydro	DJINN	303,302,159,300	0.002	2000 trans	400	0.0632	0.0607	0.0155
	seq2seq	26 units	0.001	10 seq	5500	0.0419	0.0391	0.0086
ICF	DJINN	–	–	–	–	–	–	–
	seq2seq	16 units	0.001	30 seq	1500	0.0046	0.0037	0.0020

from the radiation hydrodynamics dataset. The seq2seq model predicts a smoother and more accurate evolution of the system than the DJINN state transition model, which displays visible discontinuities between time steps. The recurrent connections in the seq2seq decoder allow the model to process the entire output sequence simultaneously and leverage long and short-term correlations in the data, resulting in a smooth prediction for the temporal and spatial evolution of the system.

The ICF X-ray emission image dataset includes the highest number of input dimensions (4096 pixels) and the training data is limited to only 240 complete time series. The DJINN state transition model performs significantly worse than the seq2seq model, as fully-connected networks are not well-suited for high-dimensional image data unless preceded by convolutional layers or other dimensional reduction techniques [56, 73, 116]. Fig. 7.3 shows an example seq2seq and DJINN prediction for this dataset. The seq2seq model demonstrates low integrated error, but displays regions of up to 20% prediction error near steep gradients in the pixel values; additional training data may be required to reduce the error further. The DJINN state transition model also suffers near high gradients, and displays particularly high er-

ror in the final frames of the sequence. The first seven transitions in each sequence show the X-ray emission region compressing, and only the final two frames illustrate the subsequent expansion of the emission region. The state transition model, which only learns the transition between frames, is therefore much better at predicting the first several frames of the sequence, as the model has been exposed to significantly more data that illustrates compression from one state to the next. The seq2seq model, however, is able to predict the compression and expansion equally well, as this model learns the full evolution of the sequence, and can thus recognize that the X-ray emission region often expands toward the end of the sequence.

7.3 Using seq2seq models to improve physics simulations

Seq2seq models are attractive for multi-physics codes in which simulations can produce data of variable sequence length within a single database, as they readily handle variable length input and output sequences. As mentioned in the introduction, this enables a trained seq2seq model to predict the evolution of a system at various points in time as a simulation is progressing. For example, a simulation runs for 10 time steps, passes the data to a seq2seq model to predict the next 10 time steps, and once the simulation reaches time step 20, the error in the model's prediction is evaluated. If the seq2seq model is accurately predicting the evolution of the system, computational time can be saved by stopping the simulation early, and relying on the seq2seq model to predict the remainder of the evolution. It is reasonable to expect that there is a minimum number of time steps the simulation must complete, at which point the seq2seq model can accurately predict the remainder of the evolution. This idea is tested on the diffusion dataset. The 800 time histories from the training dataset are split into random-length input and output sequences, both varying in length from 10 to 90 time steps. The model is trained on the variable sequence length

data, and is evaluated on the test dataset, which has a fixed input sequence length. To determine the optimal input sequence length required to accurately predict the remainder of the diffusion solution, the fixed input sequence length for the test data is varied from 10 to 90 time steps. The test error as a function of input sequence length is shown in Fig. 7.4.

As expected, the average prediction error per time step decreases as the sequence length of the prediction decreases. The rate of error reduction slows around an input sequence length of 50 time steps, suggesting that it is sufficient to run half of a new diffusion simulation, at which point the seq2seq model can predict the remainder of the evolution with minimal error. In the case of the diffusion code which can be evaluated quickly, the computational time savings gained by stopping the simulation halfway through is small. However, for expensive multi-physics codes that take several hours to run, the amount of computational time that can be saved by training a seq2seq model might be significant. Future work will explore the possibility of saving computational time required to create databases of complex simulations by using a seq2seq model to complete trajectories of QOIs given only a small fraction of the system's evolution.

Another interesting application of the diffusion seq2seq model is its ability to predict solutions for various values of the discretization resolution, dx, given only the initial conditions of the system. Although it requires extrapolation, the model can be evaluated with increasingly small dx and compared to the analytic solution. In Figure 7.5, the predicted, analytic, and numerical solutions with a finite dx are shown on the left for D=1.34. On the right, the black points indicate the error in the numerical solution as a function of dx is computed for the same value of D, which shows the expected convergence rate of dx^2 for small dx. The stars indicate the seq2seq model error as dx is decreased beyond the boundary of the training

data. The seq2seq model predictions for small dx do not follow the second order convergence rate of the numerical solution, but the predictions have significantly lower error than solutions from the training data, suggesting the model does have a limited ability to extrapolate to smaller dx.

In this simple example, the diffusion equation can be solved quickly on a high resolution grid to estimate the converged solution without relying on seq2seq models. However, for more complicated systems it might be necessary to run a set of simulations with increasing levels of resolution to estimate the convergence rate of the solution. The data generated in the convergence study can be used to train a seq2seq model which can estimate converged solutions for systems which cannot be solved analytically.

7.4 Conclusions

In this work, we have demonstrated the ability of seq2seq models to predict the temporal evolution multi-physics simulations of varying complexity. The encoder-decoder structure with recurrent connections offers many advantages over simple state transition models; the seq2seq models can learn long-term dependencies in data, can handle arbitrary sequence lengths, and can process high dimensional quantities of interest effectively. By learning to accurately emulate multi-physics design codes, seq2seq models enable rapid estimation of the time trajectory of quantities of interest at a fraction of the cost of a full simulation. Accurately emulating the evolution of ICF implosions with seq2seq models is the first step toward incorporating time-dependent experimental data into a predictive framework; transfer learning with seq2seq models is a logical next step in the progression of this work.

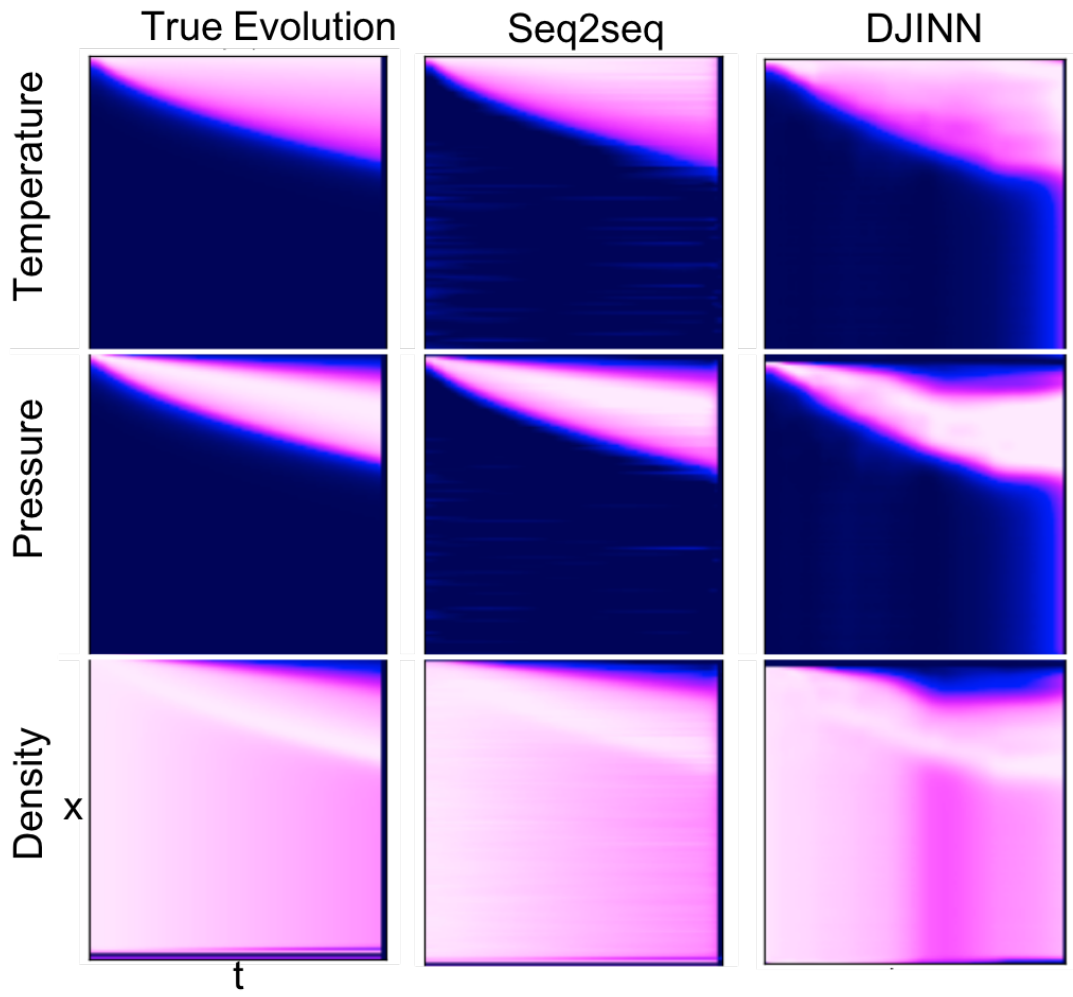


Figure 7.2: Example prediction from the radiation hydrodynamics dataset. Color maps on all panels are scaled between [0,1]. The true evolution of the system is illustrated on the left, followed by the predictions from the seq2seq (middle) and state transition DJINN model (right). The seq2seq model produces a smoother and more accurate evolution than the state transition model.

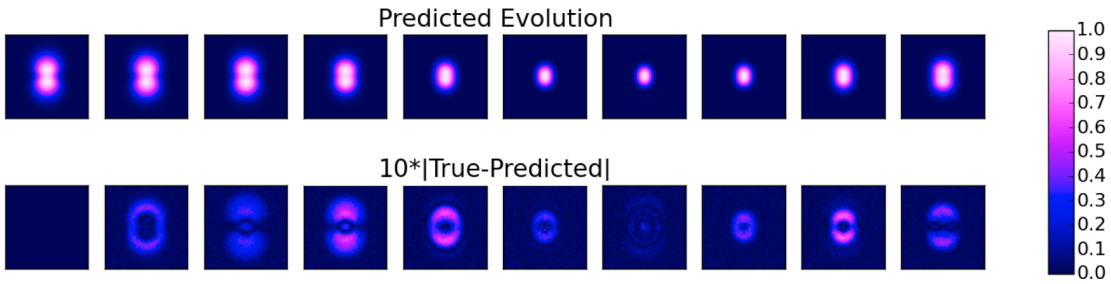


Figure 7.3: Example prediction from the ICF database. The first row is the seq2seq model's predictions evolution of the X-ray image, given the first image in the row as the input to the model. The second row illustrates the absolute error in the model's predictions multiplied by a factor of 10 for visibility. The regions of highest error occur near gradients in the X-ray intensity.

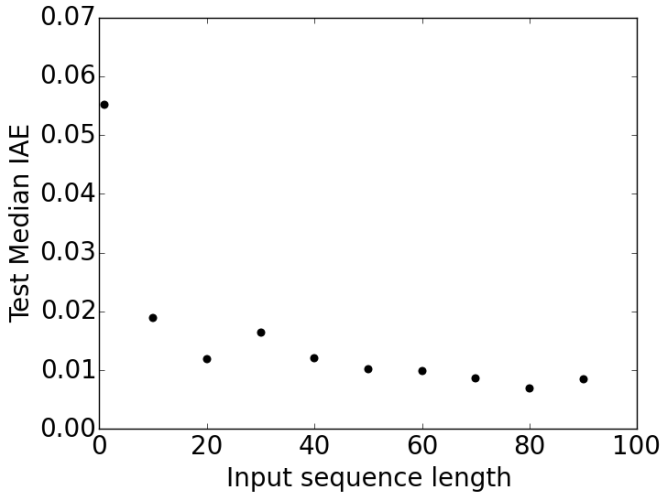


Figure 7.4: Median IAE for the test dataset as a function of the input sequence length. The model predicts the evolution with lower error per time step as the length of the input sequence increases. At an input sequence length of 50, the rate of the error reduction slows, suggesting the model only needs to observe half of the evolution before it can predict the remainder of the trajectory with minimal error.

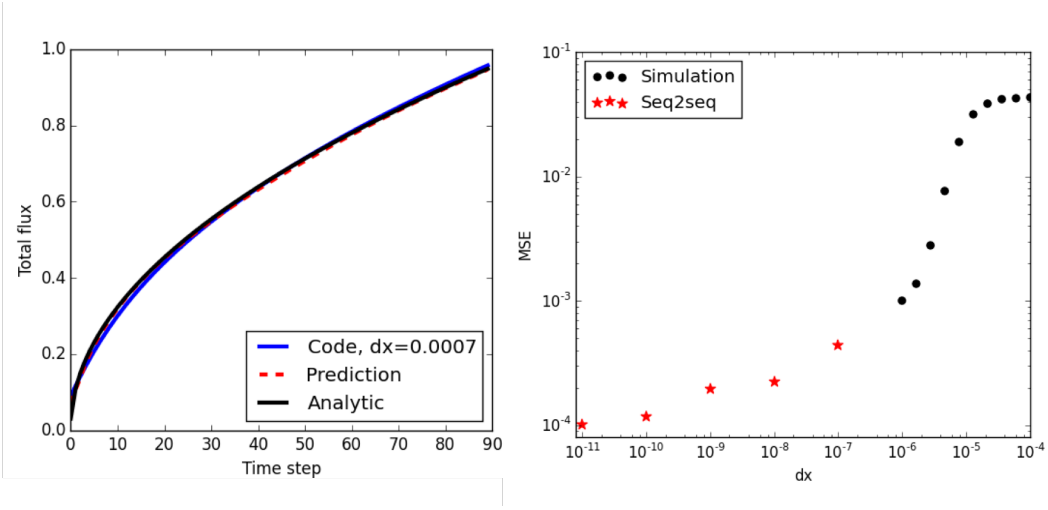


Figure 7.5: Left: Normalized total concentration for the 1D diffusion problem with a diffusion coefficient of 1.34. The black line shows the analytic solution, the blue is the numerical solution with dx=7×10^{-4}, and the red line is the prediction of the seq2seq model with dx=0. Right: Error of the finite element solution as dx is decreased is shown in black; the red stars indicates the seq2seq prediction error (compared to the analytic solution) as dx is decreased toward zero.

8. CONCLUSIONS

Achieivng inertial fusion ignition in a laboratory setting remains an elusive goal, and requires better use of the simulation capabilities, physics experiments, and diagnostic instruments available at facilities such as the NIF and Omega. In this thesis, we have explored the use of machine learning to bridge the gap between simulations and experiments for inertial confinement fusion.

We illustrate how machine learning models built on simulation databases are used to rapidly optimize implosion designs; the application of this methodology to one of the largest databases of 2D ICF simulations ever created results in the discovery of an ovoid-shaped implosion, revealing rich physics that had never before been observed in ICF capsules. The ovoid implosions are robust to a variety of performance degradation sources common in ICF experiments; their robustness is attributable to the generation of large scale zonal flows within the capsule, induced by a specific applied radiation drive asymmetry. The existence of such high performing, asymmetric implosions calls into questions decades of research that suggest deviations from spherical compression degrade implosion performance. Not only does the discovery of the ovoid point toward new physics and a potential path toward achieivng high yield in experiments, but it serves as an illustrative example of the powerful role machine learning can play in aiding experimental design.

The machine learning methods used to find the ovoid are simple, black-box algorithms that work well for optimization, but are not accurate enough for comparing simulation and experimental data. Furthermore, these models lack estimates of prediction uncertainties, which play an important role when deciding which new experiments to run, determining the value of improving particular diagnostics, and

inferring unknown quantities. To meet the need for an easy to use, highly accurate machine learning model that is equipped with uncertainty estimates, we developed "deep jointly-informed neural networks", or DJINN. DJINN is inspired by our early success with decision tree-based models for finding the ovoid; it uses decision trees trained on the data to determine an appropriate deep neural network architecture and set of initial weights. The algorithm overcomes many of the challenges of training deep neural networks on arbitrary datasets by determining many of the hyperparameters in an automated manner. Furthermore, we can leverage the work of other deep learning researchers and employee techniques such as dropout in the DJINN models to obtain prediction uncertainty estimates.

DJINN enables the efficient creation of surrogate models for a wide variety of datasets, including ICF, atomic, and nuclear physics data, and climate data. As an approximate Bayesian model that is quick to query, DJINN is easy to use for parameter inference; either by constructing inverse models that map from output to input space directly, or via Markov Chain Monte Carlo sampling. When coupled with autoencoders– neural networks designed for unsupervised data compression– DJINN is able to produce accurate and self-consistent forward and inverse models for ICF simulation databases. The inverse model methodology is applied to other high energy density science experiments, including the interpretation of atomic emission spectra to infer physical properties, such as temperature and density.

DJINN is also used to demonstrate a promising path forward for creating predictive, data-driven models of complex physics experiments. Unlike typical discrepancy term calibration, we explore the use of transfer learning with DJINN for calibrating low fidelity to high fidelity simulations, and calibrating from simulations to experiments for Omega ICF data. The idea of transfer learning is that a neural network can be trained on a task for which there are large amounts of data (such as simulations),

and then be partially retrained on a small set of typically expenssive data (such as experiments) to shift its predictions to align better with the expensive data. Partial retraining involves freezing many of the weights in the network, only allowing a small number of weights to change when exposed to the small set of expensive data. We introduce the idea of hierarchical transfer learning for numerical simulation and experimental data. In hierarchical transfer learning, a neural network is first trained on a large database of low fidelity simulation data. Many of the weights of this network are frozen, and then the model is updated with a smaller database of higher fidelity simulations; if the higher fidelity simulations are a more accurate representation of reality, this makes the final step – transfer learning to the sparse set of experimental data – easier for the network to learn. The cost of low and high fidelity simulations and the cost of experiments can be balanced to determine how to create a sufficiently accurate predictive model with the lowest investment of computational and experimental resources. We successfully create DJINN models that are transfer learned to experimental ICF data for Omega experiments, and demonstrate that these models can be used to accurately predict the outcome of future experiments. We then use these experimentally-informed models to optimize an Omega implosion design, which results in a design that is consistent with experiments researchers have found to perform best, via an iterative, physics-guided optimization procedure.

Finally, we extend our predictive modeling techniques to include time series data. We are interested in exploring whether its possible to predict the evolution of a simulation in-flight, and saving computational resources by terminating simulations that can be predicted forward by a machine learning model with high confidence. Furthermore, much of the data collected in ICF experiments is time dependent; incorporating this information into our calibrated models can better constrain the model predictions. To explore the feasibility of modeling time-dependent ICF data

with machine learning techniques, we use sequence-to-sequence models to predict the evolution of multiphysics simulations. Seq2seq models are primarily used in natural language processing tasks; we demonstrate the ability of such models to accurately predict time series data generated with computer simulations of varying complexity.

There are dozens of diagnostics that collect vast quantities data in ICF experiments; machine learning offers a promising approach for analyzing the full suite of data to improve our understanding of ICF implosions. The work presented in this dissertation is just a small glimpse at how machine learning can be used to improve how we model and interpret ICF and other high energy density science experiments. From simulation-only based design optimization, to parameter inference and model calibration, to optimization of models constrained to be consistent with experimental data, we have presented a variety of applications of machine learning to improve the ICF modeling and design process. Some interesting areas for future exploration include: hierarchical calibration for indirect drive experiments – moving from hohlraum simulations to high resolution capsule simulations to experiments; and calibrating time series predictions with mutlimodal, time-dependent experimental data.

REFERENCES

[1] M. Abadi and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[2] R. Akmeliawati, M. P. L. Ooi, and Y. C. Kuang. Real-time Malaysian sign language translation using colour segmentation and neural network. In *2007 IEEE Instrumentation Measurement Technology Conf.*, pages 1–6, May 2007.

[3] K. Anderson and R. Betti. Laser-induced adiabat shaping by relaxation in inertial fusion implosions. *Physics of Plasmas*, 11(1):5–8, 2004.

[4] B. O. Ayinde and J. M. Zurada. Nonredundant sparse feature extraction using autoencoders with receptive fields clustering. *Neural Networks*, 93:99 – 109, 2017.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *ArXiv e-prints*, September 2014.

[6] R. Balestriero. Neural Decision Trees. *ArXiv e-prints*, February 2017.

[7] M. J. Beal and et al. *Variational algorithms for approximate Bayesian inference*. University of London, 2003.

[8] L. F. Berzak Hopkins. Private communication, 2016.

[9] L. F. Berzak Hopkins, S. Le Pape, L. Divol, N. B. Meezan, A. J. Mackinnon, D. D. Ho, O. S. Jones, S. Khan, J. L. Milovich, J. S. Ross, P. Amendt, D. Casey, P. M. Celliers, A. Pak, J. L. Peterson, J. Ralph, and J. R. Rygg. Near-vacuum hohlraums for driving fusion implosions with high density carbon ablators. *Physics of Plasmas*, 22(5), 2015.

[10] R. Betti. The one-dimensional cryogenic implosion campaign on OMEGA: Modeling, experiments, and a statistical approach to predict and understand direct-drive implosions. In *American Physical Society Division of Plasma Physics Meeting*. APS, 2017.

[11] G. Biau, E. Scornet, and J. Welbl. Neural Random Forests. *ArXiv e-prints*, April 2016.

[12] A. T. Booth, R. Choudhary, and D. J. Spiegelhalter. A hierarchical Bayesian framework for calibrating micro-level models with macro-level data. *Journal of Building Performance Simulation*, 6(4):293–318, 2013.

[13] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[14] D. T. Casey and et al. The high velocity, high adiabat, "Bigfoot" campaign and tests of indirect-drive implosion scaling. *Physics of Plasmas*, 25(5):056308, 2018.

[15] V. Castillo. Enhancing experimental design and understanding with deep learning/AI. In *GPU Technology Conference*, 2018.

[16] P. Y. Chang, R. Betti, B. K. Spears, K. S. Anderson, J. Edwards, M. Fatenejad, J. D. Lindl, R. L. McCrory, R. Nora, and D. Shvarts. Generalized measurable ignition criterion for inertial confinement fusion. *Physical Review Letters*, 104(13):135002, 2010.

[17] X. Chen and H. Ishwaran. Random forests for genomic data analysis. *Genomics*, 99(6):323 – 329, 2012.

[18] Guillaume Chevalier. Sequence to sequence (seq2seq) recurrent neural network (RNN) for time series prediction. github.com/guillaume-chevalier/seq2seq-signal-prediction, 2016.

[19] H. A. Chipman, E. I. George, R. E. McCulloch, and et al. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010.

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[21] J. Chung, C. Gulcehre, K. H. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[22] D. S. Clark, S. W. Haan, and J. D. Salmonson. Robustness studies of ignition targets for the National Ignition Facility in two dimensions. *Physics of Plasmas*, 15(5), 2008.

[23] G. E. Dahl, T. N. Sainath, and G. E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE Intl. Conf. on Acoustics, Speech and Signal Processing*, pages 8609–8613, 2013.

[24] P.-T. de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

[25] J. Delettrez, R. Epstein, M. C. Richardson, P. A. Jaanimagi, and B. L. Henke. Effect of laser illumination nonuniformity on the analysis of time-resolved x-ray measurements in uv spherical transport experiments. *Physical Review A*, 36(8):3926, 1987.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[27] D. G. T. Denison, B. K. Mallick, and A. F. M. Smith. Bayesian MARS. *Statistics and Computing*, 8(4):337–346, 1998.

[28] T. R. Dittrich, O. A. Hurricane, L. F. Berzak-Hopkins, D. A. Callahan, D. T. Casey, D. Clark, E. L. Dewald, T. Doeppner, S. W. Haan, B. A. Hammel, J. A. Harte, D. E. Hinkel, B. J. Kozioziemski, A. L. Kritcher, T. Ma, A. Nikroo, A. E. Pak, T. G. Parham, H.-S. Park, P. K. Patel, B. A. Remington, J. D. Salmonson, P. T. Springer, C. R. Weber, G. B. Zimmerman, and J. L. Kline. Simulations of fill tube effects on the implosion of high-foot NIF ignition capsules. *Journal of Physics: Conference Series*, 717(1):012013, 2016.

[29] B. Efron and et al. Least angle regression. *Annals of Statistics*, pages 407–499, 2004.

[30] M. Everingham, S. Eslami, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[31] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7, Part II:179–188, 1936.

[32] M. Forina and et al. An Extendible Package for Data Exploration, Classification and Correlation. *Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno,16147 Genoa, Italy.*

[33] J. A. Gaffney. Making ICF Models More Predictive: Combining Simulations, Experiments and Expert Knowledge using Machine Learning and Bayesian Statistics. *APS Division of Plasma Physics Conference*, 2018.

[34] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, 2015.

[35] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, 2015.

[36] Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *ArXiv e-prints*, June 2015.

[37] Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

[38] S. Galelli and A. Castelletti. Assessing the predictive capability of randomized tree-based ensembles in streamflow modeling. *Hydrology and Earth System Sciences*, 17:2669–2684, 2013.

[39] F. A. Gers, D. Eck, and J. Schmidhuber. Applying LSTM to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

[40] W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[41] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Int. Conf. on Artificial Intelligence and Statistics*, 2010.

[42] V. N. Goncharov, J. P. Knauer, P. W. McKenty, P. B. Radha, T. C. Sangster, S. Skupsky, R. Betti, R. L. McCrory, and D. D. Meyerhofer. Improved performance of direct-drive inertial confinement fusion target designs with adiabat shaping using an intensity picket. *Physics of Plasmas*, 10(5):1906–1918, 2003.

[43] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[44] V. Gopalaswamy. Optimization of direct-drive inertial fusion implosions through predictive statistical modeling. In *American Physical Society Division of Plasma Physics Meeting*, 2018.

[45] V. Gopalaswamy, R. Betti, J. P. Knauer, N. Luciani, D. Patel, K. M. Woo, A. Bose, I. V. Igumenshchev, E. M. Campbell, K. S. Anderson, and et al. Tripled yield in direct-drive laser fusion through statistical modelling. *Nature*, 565(7741):581, 2019.

[46] K. Gregor, I. Danihelka, A. Graves, D. Jimenez Rezende, and D. Wierstra. DRAW: A Recurrent Neural Network For Image Generation. *ArXiv e-prints*, February 2015.

[47] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007.

[48] J. Gu, Z. Dai, Z. Fan, S. Zou, W. Ye, W. Pei, and S. Zhu. A new metric of the low-mode asymmetry for ignition target designs. *Physics of Plasmas*, 21(1), 2014.

[49] S. W. Haan, P. A. Amendt, T. R. Dittrich, B. A. Hammel, S. P. Hatchett, M. C. Herrmann, O. A. Hurricane, O. S. Jones, J. D. Lindl, M. M. Marinak, and et al. Design and simulations of indirect drive ignition targets for NIF. *Nuclear fusion*, 44(12):S171, 2004.

[50] S. W. Haan, H. Huang, M. A. Johnson, M. Stadermann, S. Baxamusa, S. Bhandarkar, D. S. Clark, V. Smalyuk, and H. F. Robey. Instability growth seeded by oxygen in CH shells on the National Ignition Facility. *Physics of Plasmas*, 22(3), 2015.

[51] S.W. Haan and et al. Point design targets, specifications, and requirements for the 2010 ignition campaign on the National Ignition Facility. *Physics of Plasmas*, 18, 2011.

[52] D. Harrison and D.L. Rubinfeld. Hedonic prices and the demand for clean air. *J. Environ. Economics & Management*, 5:81–102, 1978.

[53] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[54] R. Hatarik, D. B. Sayre, J. A. Caggiano, T. Phillips, M. J. Eckart, E. J. Bond, C. Cerjan, G. P. Grim, E. P. Hartouni, J. P. Knauer, J. M. Mcnaney, and D. H. Munro. Analysis of the neutron time-of-flight spectra from inertial confinement fusion experiments. *Journal of Applied Physics*, 118(18):184502, 2015.

[55] D. Hernández-Lobato and J. M. Hernández-Lobato. Scalable gaussian process classification via expectation propagation. In *Artificial Intelligence and Statistics*, pages 168–176, 2016.

[56] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[57] D. D.-M. Ho, S. W. Haan, J. D. Salmonson, D. S. Clark, J. D. Lindl, J. L. Milovich, C. A. Thomas, L. F. Berzak Hopkins, and N. B. Meezan. Implosion configurations for robust ignition using high- density carbon (diamond) ablator for indirect-drive ICF at the National Ignition Facility. *Journal of Physics: Conference Series*, 717(1):012023, 2016.

[58] S. Hoo-Chang, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285, 2016.

[59] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[60] K. Humbird, L. Peterson, R. McClarren, J. Field, J. Gaffney, M. Kruse, R. Nora, and B. Spears. Using deep neural networks to augment NIF post-shot

analysis. In *APS Meeting Abstracts*, page YO7.013, October 2017.

[61] K. D. Humbird, J. L. Peterson, and R. G. McClarren. Deep Neural Network Initialization With Decision Trees. *IEEE Transactions on Neural Networks and Learning Systems*, (10.1109/TNNLS.2018.2869694), 2018.

[62] O. A. Hurricane and J. Hammer. Pressure driven instability of a finite thickness fluid layer. Technical Report UCRL-JRNL-203840, Lawrence Livermore National Laboratory, 2004.

[63] A. Jalali and S. Sanghavi. Learning the Dependence Graph of Time Series with Latent Factors. *ArXiv e-prints*, June 2011.

[64] C. Kaynak. Methods of combining multiple classifiers and their applications to handwritten digit recognition. Master's thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University, 1995.

[65] L Kegelmeyer. Image analysis and machine learning for NIF optics inspection. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2018.

[66] J. H. Kelly, L. J. Waxer, V. Bagnoud, I. A. Begishev, J. Bromage, B. E. Kruschwitz, T. J. Kessler, S. J. Loucks, D. N. Maywar, R. L. McCrory, and et al. OMEGA EP: High-energy petawatt capability for the OMEGA laser facility. In *Journal de Physique IV (Proceedings)*, volume 133, pages 75–80. EDP sciences, 2006.

[67] M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[68] G. I. Kerley. Equations of state for hydrogen and deuterium. Technical Report SAND 2003-3613, Sandia National Laboratories, 2004.

[69] T. J. Kessler. Phase conversion of lasers with low-loss distributed phase plates. *SPIE*, 1870(95-104), 1993.

[70] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, 2014.

[71] J. P. Knauer, K. Anderson, R. Betti, T. J. B. Collins, V. N. Goncharov, P. W. McKenty, D. D. Meyerhofer, P. B. Radha, S. P. Regan, T. C. Sangster, V. A. Smalyuk, J. A. Frenje, C. K. Li, R. D. Petrasso, and F. H. Séguin. Improved target stability using picket pulses to increase and shape the ablator adiabat. *Physics of Plasmas*, 12(5), 2005.

[72] A. L. Kritcher, R. Town, D. Bradley, D. Clark, B. Spears, O. Jones, S. Haan, P. T. Springer, J. Lindl, R. H. H. Scott, D. Callahan, M. J. Edwards, and O. L. Landen. Metrics for long wavelength asymmetries in inertial confinement fusion implosions on the National Ignition Facility. *Physics of Plasmas*, 21(4):042708, 2014.

[73] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[74] S. H. Langer, B. K. Spears, J. L. Peterson, J. E. Field, R. Nora, and S. Brandon. A HYDRA UQ workflow for NIF ignition experiments. In *Proceedings of ISAV 2016: Second Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization*. IEEE Computer Society, 2016.

[75] J. D. Lawson. Some criteria for a power producing thermonuclear reactor. *Proceedings of the Physical Society. Section B*, 70(1):6, 1957.

[76] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan. A persona-based neural conversation model. *CoRR*, abs/1603.06155, 2016.

[77] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *The IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[78] Y. Lin, T. J. Kessler, and G. N. Lawrence. Design of continuous surface-relief phase plates by surface-based simulated annealing to achieve control of focal-plane irradiance. *Opt. Lett.*, 21(17031705), 1996.

[79] J. Lindl. *Inertial Confinement Fusion: The Quest for Ignition and Energy Gain Using Indirect Drive*. AIP-Press, College Park, MD, 1998.

[80] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[81] M. T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[82] M.-T. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *ArXiv e-prints*, August 2015.

[83] Volodymyr M. and et al. Human-level control through deep reinforcement learning. *Nature*, 518:529 – 533, 2015.

[84] Brian J MacGowan, BB Afeyan, CA Back, RL Berger, G Bonnaud, M Casanova, BI Cohen, DE Desenne, DF DuBois, AG Dulieu, et al. Laser–plasma interactions in ignition-scale hohlraum plasmas. *Physics of Plasmas*, 3(5):2029–2040, 1996.

[85] A. G. MacPhee and et al. X-ray shadow imprint of hydrodynamic instabilities on the surface of inertial confinement fusion capsules by the fuel fill tube. *Phys. Rev. E*, 95:031204, Mar 2017.

[86] M. Marinak, G. Kerbel, N. Gentile, O. Jones, D. Munro, S. Pollaine, T. Dittrich, and S. Haan. Three-dimensional HYDRA simulations of national ignition facility targets. *Physics of Plasmas*, 8(5):2275–2280, 2001.

[87] M. D. McKay, R. J. Beckman, and W. J. Conover. *A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code*, volume 21. Springer, 1979.

[88] L. R. Medsker and L. C. Jain. Recurrent neural networks. *Design and Applications*, 5, 2001.

[89] E. E. Meshkov. Instability of the interface of two gases accelerated by a shock wave. *Fluid Dynamics*, 4:101–104, 1969.

[90] D. Mihalas and B. W. Mihalas. *Foundations of radiation hydrodynamics*. Courier Corporation, 2013.

[91] G. H. Miller, E. I. Moses, and C. R. Wuest. The National Ignition Facility: enabling fusion ignition for the 21st century. *Nuclear Fusion*, 44(12), 2004.

[92] T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[93] E. I. Moses, R. N. Boyd, B. A. Remington, C. J. Keane, and R. Al-Ayat. The National Ignition Facility: Ushering in a new age for high energy density science. *Physics of Plasmas*, 16(4):–, 2009.

[94] D. H. Munro. Interpreting inertial fusion neutron spectra. *Nuclear Fusion*, 56(3):036001, 2016.

[95] S. R. Nagel, S. W. Haan, J. R. Rygg, M. Barrios, L. R. Benedetti, D. K. Bradley, J. E. Field, B. A. Hammel, N. Izumi, O. S. Jones, S. F. Khan, T. Ma, A. E. Pak, R. Tommasini, and R. P. J. Town. Effect of the mounting membrane on shape in inertial confinement fusion implosions. *Physics of Plasmas*, 22(2), 2015.

[96] S. R. Nagel, J. R. Rygg, L. R. Benedetti, T. Ma, M. A. Barrios, S. W. Haan, B. A. Hammel, T. Doeppner, A. E. Pak, R. Tommasini, and et al. The impact of capsule "tent" thickness on interpreting low mode shape. In *APS Meeting Abstracts*, 2013.

[97] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In Thorsten Joachims, editor, *Proceedings of the 27th Intl. Conf. on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010.

[98] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[99] R. Nora, J. Field, B. Spears, and C. Thomas. Using ensembles of simulations to find high-fidelity post-shot models of inertial confinement implosions at the National Ignition Facility. In *APS Meeting Abstracts*, October 2016.

[100] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1717–1724. IEEE, 2014.

[101] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[102] R. K. Pace and R. Barry. Sparse spatial autoregressions. *Statistics and Probability Letters*, 33:291–297, 1997.

[103] Tim Pearce, Mohamed Zaki, Alexandra Brintrup, and Andy Neel. Uncertainty in neural networks: Bayesian ensembling. *arXiv preprint arXiv:1810.05546*, 2018.

[104] 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, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[105] J. L. Peterson, D. S. Clark, L. P. Masse, and L. J. Suter. The effects of early time laser drive on hydrodynamic instability growth in National Ignition Facility implosions. *Physics of Plasmas*, 21(9):092710, 2014.

[106] J. L. Peterson, K. D. Humbird, J. E. Field, S. T. Brandon, S. H. Langer, R. C. Nora, B. K. Spears, and P. T. Springer. Zonal flow generation in inertial confinement fusion implosions. *Physics of Plasmas*, 24(3):032702, 2017.

[107] Gerald C Pomraning. *The equations of radiation hydrodynamics*. Courier Corporation, 1973.

[108] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge

university press, 2007.

[109] C. E. Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[110] Rayleigh. Investigation of the character of the equilibrium of an incompressible heavy fluid of variable density. *Proceedings of the London Mathematical Society*, s1-14(1):170–177, 1882.

[111] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-Scale Evolution of Image Classifiers. *ArXiv e-prints*, March 2017.

[112] R. D. Richtmyer. Taylor instability in shock acceleration of compressible fluids. *Communications on Pure and Applied Mathematics*, 13(2):297–319, 1960.

[113] J. E. Rothenberg. Comparison of beam-smoothing methods for direct-drive inertial confinement fusion. *J. Opt. Soc. Am. B*, 14(1664-1671), 1997.

[114] H. A. Scott. Cretin: radiative transfer capability for laboratory plasmas. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 71(2-6):689–701, 2001.

[115] R. Setiono and W.K. Leow. On mapping decision trees and neural networks. *Knowledge-Based Systems*, 12(3):95 – 99, 1999.

[116] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, 2014.

[117] S. Skupsky and R. S. Craxton. Irradiation uniformity for high-compression laser fusion experiments. *Physics of Plasmas*, 6(21572163), 1999.

[118] S. Skupsky and et al. Improved laser-beam uniformity using the angular dispersion of frequency-modulated light. *J. Appl. Phys.*, 66(34563462), 1989.

[119] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, pages 2951–2959, USA, 2012. Curran Associates Inc.

[120] H. Song, D. Rajan, J. J. Thiagarajan, and A. Spanias. Attend and Diagnose: Clinical Time Series Analysis using Attention Models. *ArXiv e-prints*, November 2017.

[121] B. K. Spears, S. Brandon, D. Clark, C. Cerjan, J. Edwards, O. Landen, J. Lindl, S. Haan, S. Hatchett, J. Salmonson, P. Springer, S. V. Weber, and D. Wilson. Prediction of ignition implosion performance using measurements of low-deuterium surrogates. *Journal of Physics: Conference Series*, 244(2):022014, 2010.

[122] B. K. Spears and et al. Deep learning: A guide for practitioners in the physical sciences. *Physics of Plasmas*, 25(8):080901, 2018.

[123] B. K. Spears, S. Glenzer, M. J. Edwards, S. Brandon, D. Clark, R. Town, C. Cerjan, R. Dylla-Spears, E. Mapoles, D. Munro, J. Salmonson, S. Sepke, S. Weber, S. Hatchett, S. Haan, P. Springer, E. Moses, J. Kline, G. Kyrala, and D. Wilson. Performance metrics for inertial confinement fusion implosions: Aspects of the technical framework for measuring progress in the National Ignition Campaign). *Physics of Plasmas*, 19(5), 2012.

[124] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[125] Weston M Stacey. *Fusion: An introduction to the physics and technology of magnetic confinement fusion*. John Wiley & Sons, 2010.

[126] P. A. Sterne, L. X. Benedict, S. Hamel, A. A. Correa, J. L. Milovich, M. M. Marinak, P. M. Celliers, and D. E. Fratanduono. Equations of state for ablator materials in inertial confinement fusion simulations. *Journal of Physics: Conference Series*, 717(1):012082, 2016.

[127] W. N. Street, W. H. Wolberg, and O. L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. volume 1905, pages 861–870, 1993.

[128] H. F. Stripling, R. G. McClarren, C. C. Kuranz, M. J. Grosskopf, E. Rutter, and B. R. Torralva. A calibration and data assimilation method using the Bayesian MARS emulator. *Annals of Nuclear Energy*, 52:103–112, 2013.

[129] M. Sundermeyer, R. Schlüter, and H. Ney. LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech communication Association*, 2012.

[130] L. J. Suter, R. L. Kauffman, C. B. Darrow, A. A. Hauer, H. Kornblum, O. L. Landen, T. J. Orzechowski, D. W. Phillion, J. L. Porter, L. V. Powers, and et al. Radiation drive in laser-heated hohlraums. *Physics of Plasmas*, 3(5):2057–2062, 1996.

[131] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[132] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[133] Y. Tang. TF Learn: TensorFlow's high-level module for distributed machine learning. *arXiv preprint 1612.04251*, 2016.

[134] R. Taylor. The instability of liquid surfaces when accelerated in a direction perpendicular to their planes. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 201(1065):192–196, 1950.

[135] J. Meyer ter Vehn and S. Atzeni. *The Physics of Inertial Fusion*. Oxford University Press, Oxford, UK, 2009.

[136] G. Thimm and E. Fiesler. *Neural network initialization*, pages 535–542. Springer Berlin Heidelberg, 1995.

[137] C. Thomas. BigFoot, a program to reduce risk for indirect drive laser fusion. *58th Annual Meeting of the APS Division of Plasma Physics*, 61:18, 2016.

[138] R. P. J. Town, D. K. Bradley, A. Kritcher, O. S. Jones, J. R. Rygg, R. Tommasini, M. Barrios, L. R. Benedetti, L. F. Berzak Hopkins, P. M. Celliers, T. Döppner, E. L. Dewald, D. C. Eder, J. E. Field, S. M. Glenn, N. Izumi, S. W. Haan, S. F. Khan, J. L. Kline, G. A. Kyrala, T. Ma, J. L. Milovich, J. D. Moody, S. R. Nagel, A. Pak, J. L. Peterson, H. F. Robey, J. S. Ross, R. H. H. Scott, B. K. Spears, M. J. Edwards, J. D. Kilkenny, and O. L. Landen. Dynamic symmetry of indirectly driven inertial confinement fusion capsules on the National Ignition Facility. *Physics of Plasmas*, 21:056313, 2014.

[139] M. Van Oijen, J. Rougier, and R. Smith. Bayesian calibration of process-based forest models: bridging the gap between models and data. *Tree Physiology*, 25(7):915–927, 2005.

[140] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. *ArXiv e-prints*, June

2017.

[141] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3156–3164. IEEE, 2015.

[142] S. Wang, C. Aggarwal, and H. Liu. Using a random forest to inspire a neural network and improving on it. *Available at: http://www.public.asu.edu/ swang187/publications/NNRF.pdf*, 2017.

[143] C. R. Weber and B A Hammel. Private communication, 2016.

[144] A. Wilson and H. Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International Conference on Machine Learning*, pages 1775–1784, 2015.

[145] Y. Wu and et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv e-prints*, September 2016.

[146] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

[147] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[148] D. Yuan, Z. Qian, and Y. Pei. State transition ANNs for hourly wind speed forecasting. In *2017 Chinese Automation Congress (CAC)*, pages 6934–6938, Oct 2017.

[149] C. D. Zhou and R. Betti. A measurable Lawson criterion and hydro-equivalent curves for inertial confinement fusion. *Physics of Plasmas*, 15(10):102707, 2008.

[150] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. 2017.

[151] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning Transferable Architectures for Scalable Image Recognition. *ArXiv e-prints*, July 2017.