Investigation of Bond Strain Effects on XANES Spectra via Artificial Neural Networks

by
Jeremy K. Thaller

Professor Anatoly Frenkel, Advisor
Brookhaven National Laboratory
Chemistry Division
New York, USA

Professor Wolfgang Schmahl, Advisor
Ludwig-Maximilians-Universität
Fakultät Geowissenschaft
München, Germany

A Master thesis submitted to the Faculty of Earth- and Environmental Sciences
of Ludwig-Maximilians-Universität München in the framework of MaMaSELF

September 1, 2021

Abstract

A recently published method [1] enables the decoding of X-ray absorption near edge structure (XANES) spectra of nanoparticles to obtain important structural descriptors: coordination numbers and bond distances. Utilizing supervised machine learning (ML), the method trains an artificial neural network (ANN) to recognize a relationship between the nanoparticle structure and the XANES spectrum. Once trained, the ANN is used to "invert" an unknown spectrum to obtain the corresponding descriptors of the catalyst structure. Bond strain is known to be an important catalytic descriptor, yet, its accurate determination in reaction conditions is hampered by high temperature and low weight loading of real catalysts. ML–assisted XANES analysis offers a promising new direction for extracting the bond strain information from XANES—and not from extended x-ray absorption fine structure (EXAFS) analysis. Using simulated XANES spectra of Au nanoparticles, we have developed an ANN capable of "inverting" an unseen XANES spectrum and predicting structural disorder in the form of mean-squared displacement. The utility of the method was demonstrated on computer-simulated nanoparticles of degrees of disorder. Further work to test the model onto particles of varying sizes and bridge the model's predictive domain onto experimental absorption spectra is required.

Executive Summary

This thesis presents work toward developing an artificial neural network (ANN) capable of ingesting a XANES spectrum and predicting the mean squared displacement (MSD) of the structure. A schematic of this network is presented in Figure 1.

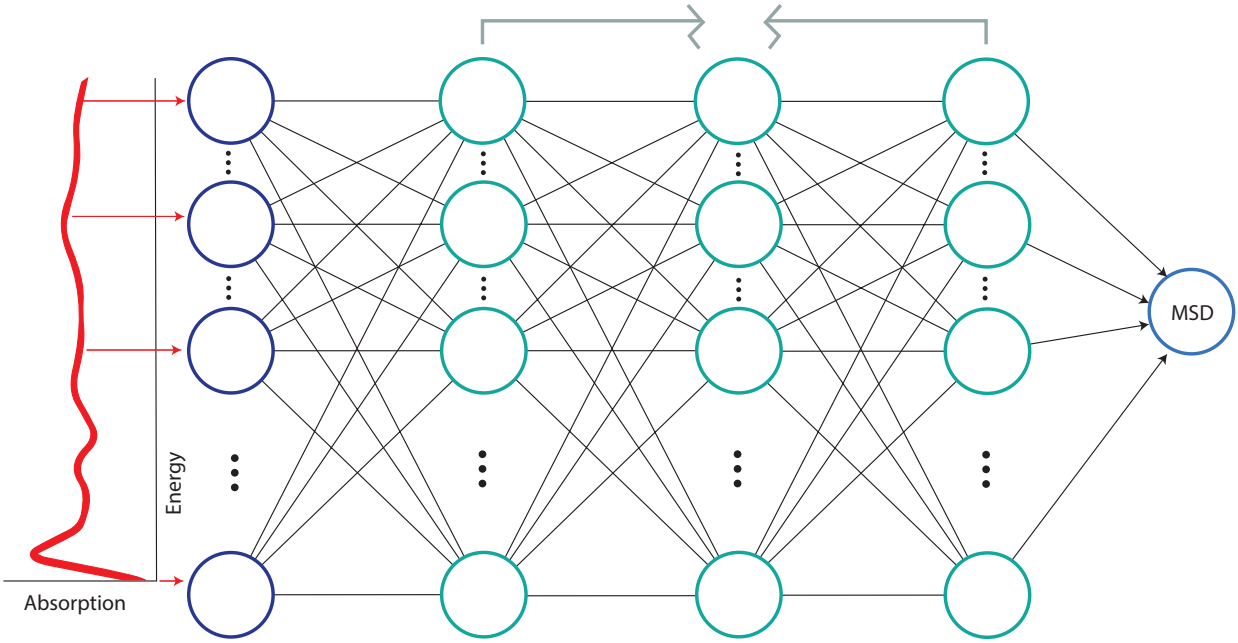


Figure 1: The above figure describes the general structure of the neural network described in this thesis. The model inputs the absorption values for a XANES spectrum and predicts the mean squared displacement (MSD) of the structure. For this thesis, we train the network exclusively on large, bulk-like gold nanoparticles.

In order to train a neural network, we rely on a large quantity of data obtained through FEFF9 [2], an *ab initio* x-ray absorption simulation software capable of simulating XANES spectra. We first present work on developing a new, statistical-based methodology for simulating disordered structures. The goal of this new approach was to reduce the number of required simulations for creating a dataset of FEFF-simulated structures. Instead of simulating the disorder structures themselves, different isotropically "expanded" or "contracted"

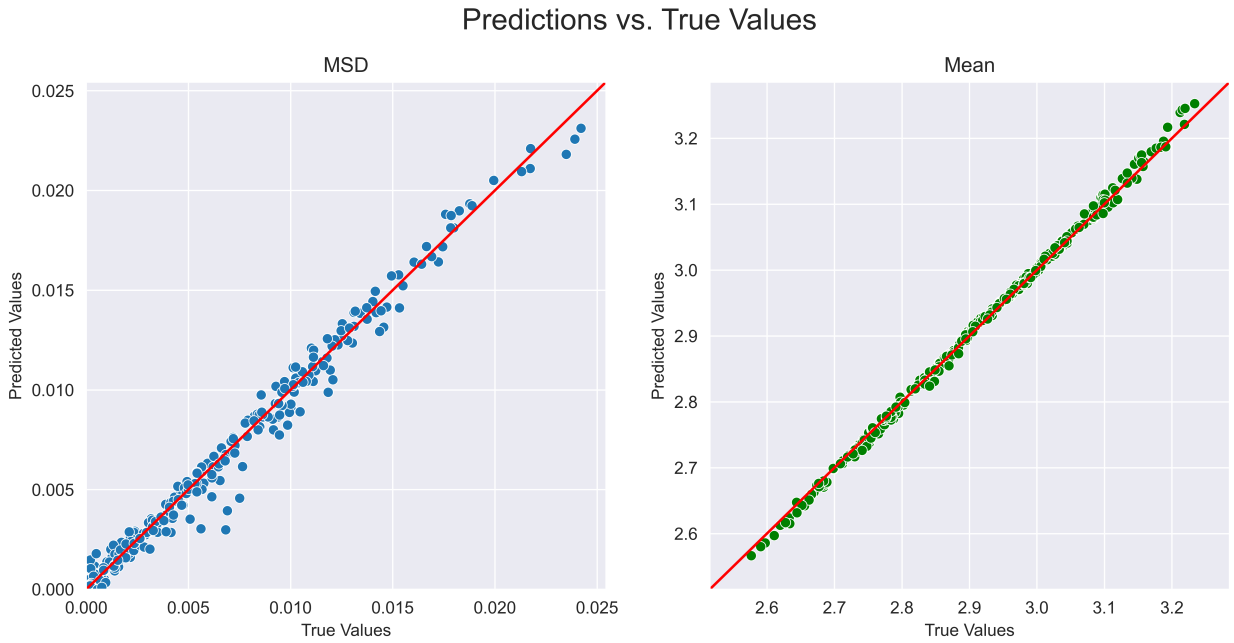


Figure 2: The predictions for a train-test split on particle-averaged simulation data are presented above. Each point in each subplot represents a FEFF simulated spectrum in the test set. For each spectrum, the y-axis represents the MSD value predicted by the NN, and the x-axis represents the true label (MSD for the left figure and mean for the right) for that spectrum. Hence, points on the $y = x$ red line are perfect predictions.

configurations were created and simulated. A tunable, distributed weighted average of these resulted was then used to approximate the absorption spectrum of disordered structures. This method, however, was ultimately abandoned for the remainder of this thesis due to systematic discrepancies between its results and those of the traditional simulation approach.

Trained on the simulation data, the neural network shows great predictive accuracy and promise for predicting the mean squared displacement and mean location of nearest neighbor bond lengths in Au, bulk-like nanoparticles. In order for the neural network to make predictions from experimental data—as opposed to simulation data—we employ a technique in machine learning known as "one-shot transfer learning." The process is the topic of current work, and we end the thesis with a discussion on the transfer-leaning plan and progress made towards this goal.

Acknowledgments

I would like to thank my advisor Professor Anatoly Frenkel and my professors in Germany and Poland over the past two years. I also owe a great debt of gratitude towards those who took the time to help proofread my work, as well as the free and open-source software (FOSS) community.

Contents

Abstract **i**

Executive Summary **ii**

Acknowledgments **iv**

1 Introduction **1**

1.1 X-ray Absorption Spectroscopy 1

1.1.1 EXAFS 5

1.1.2 XANES 8

1.2 Goals of the Thesis and Approach 8

1.3 Outline of the Thesis 9

2 Simulating Disorder **11**

2.1 Traditional Particle-Averaged Simulations 11

2.2 Statistical-Averaged Simulations 13

2.2.1 Generating Distortion Not Disorder 13

2.2.2 Generating Disorder via Probability Distribution Averaging 15

2.2.3 Simulation vs. Experimental Data 19

2.3 Particle-Averaged FEFF vs. Skewnorm-Averaged Structures 21

3 Machine Learning **25**

3.1 Feedforward and Backpropagation in ANNs 26

3.1.1 Feedforward 26

3.1.2 Loss Metrics and Regularization 29

3.1.3 Backpropagation 31

3.2 Optimizers 34

3.3 Normalization 37

3.4 Data Sparsity 39
3.4.1 Data Augmentation 39
3.4.2 Transfer Learning 40
3.5 Covolutional Neural Networks 41
3.6 How to Train a Neural Network 44

4 Results **46**
4.1 Training with Simulation Data 46
4.2 Experimental Data 46
4.2.1 Data Augmentation 48
4.2.2 Transfer Learning 49

5 Outlook **56**

A Select Python Code **58**
A.1 distortionator.py 58
A.2 gaussianator.py 59
A.3 generate-training-data.py 62
A.4 create-g(r).ipynb 65
A.5 nn.ipynb 66
A.6 nn-buddy.py 66

B Simulating XANES Spectra **67**
B.1 FEFF9 Design 67
B.2 FEFF Cards 69

List of Figures

1 Project Approach ii
2 Simulation Test Set Predictions iii

1.1 Example XAFS Experiment Setup 3
1.2 Raw XAFS Spectrum 4
1.3 XANES vs. EXAFS Regions 5
1.4 ANN for Metallic Nanoparticle Coordination Numbers 9
1.5 Project Approach 10

2.1 Simulation vs. Experimental 3 14
2.2 2D Distortion 15
2.3 FEFF Simulations Results 16
2.4 Simulated Spectrum Gaussian Weighting 17
2.5 Simulated Disordered Spectrum Weightings 18
2.6 Simulation vs. Experimental XANES: Disorder Comparison 20
2.7 Bulk-nanoparticle difference: Simulation vs. Experimental data 20
2.8 Simulation vs. Experimental XANES: Size Comparison 21
2.9 Particle-Averaged vs. Skewnorm-Averaged FEFF Low-Disorder 23
2.10 Particle-Averaged vs. Skewnorm-Averaged FEFF High-Disorder 24

3.1 Common Activation Functions 28
3.2 Overfitting 29
3.3 Neural Network Example 30
3.4 Toy Absorption Spectrum 41
3.5 1D Convolution Result 43
3.6 Example ANN Training Curve 45

4.1 Simulation Test Set Predictions 47

4.2 Experimental Data Interpolation 49
4.3 Data Augmentation: Gaussian Noise 50
4.4 Data Augmentation: Horizontal Shift 50
4.5 Transfer Learning Process 51
4.6 Hyperparamater Sweep: Cost Curve 53
4.7 Hyperpamater Sweep: Graphical Representation 54
4.8 Learning Curve for the best hyperparameters 55

B.1 FEFF Diagram 68

List of Abbreviations

AdaGrad	Adaptive Gradients
Adam	Adaptive Moment Estimator
ANN	Artificial Neural Network
BCC	Body-Centered Cubic
BNL	Brookhaven National Laboratory
CDF	Cumulative Distribution Function
CNN	Convolutional Neural Network
DFT	Density Functional theory
EMA	Exponential Moving Average
DOS	Density of states
DW	Debye-Waller
EXAFS	Extended X-ray Absorption Fine Structure
FDMNES	Finite Difference Method Near Edge Structure
FMS	Full Multiple Scattering
LASSO	Least Absolute Shrinkage and Selection Operator
MAE	Mean Absolute Error
MD	Molecular Dynamics
ML	Machine Learning
MSD	Mean Squared Displacement
MSE	Mean Squared Error
NN	Neural Network
NP	Nanoparticle
PDF	Probability Density Functional
RDF	Radial Distribution Functional
ReLU	Rectified Linear Unit
RMC	Reverse Monte Carlo
RMSprop	Root Mean Squared Propagation
SCF	Self Consistency Field
SGD	Stochastic Gradient Descent
XAFS	X-ray Absorption Fine Structure
XANES	X-ray Absorption Near Edge Structure

Chapter 1

Introduction

Synchrotron radiation was first observed by General Electric in Schenectady, New York [3]. Initially just a side effect of particle accelerator experiments, it has since grown to be an important and powerful source of high-energy electromagnetic radiation for structural determination. Compared to lab-scale x-ray generation for diffraction experiments, arguably the most important benefit of synchrotron radiation is its high brilliance. Synchrotron radiation creates a highly collimated beam of photons characterized by small divergence and spatial coherence. Additionally, synchrotron radiation is tunable across a wide spectrum (microwaves to hard X-rays) and capable of high flux, useful for short time-scale-dependent experiments or weak scatterers. Synchrotron radiation can be produced in a pulsed structure. Importantly, the incoming photons are highly polarized, either linearly or circularly, depending on where the measurement system lies with respect to the plane of the synchrotron. The advent of a technique to manufacture such a high-quality source of x-rays allowed for new, advanced methods of x-ray absorption spectroscopy, and the two fields were developed in parallel [4].

1.1 X-ray Absorption Spectroscopy

X-ray absorption spectroscopy (XAS) measures the absorption of high-energy photons by a sample as a function of energy [5]. X-ray Absorption Fine-Structure (XAFS) spectroscopy refers to the study of absorption spectra created from high-intensity x-ray interactions. An incident photon is only absorbed by a bound electron in cases where the photon's initial energy is at least as great as the energy difference between the electron's initial quantum state and the next allowed quantum state (or unoccupied continuum level) [6]. The attenuation, or change in transmitted light intensity as a result of inelastic processes, is characterized by

the Beer-Lambert Law (1.1). For an incident beam of intensity I_0, the transmitted intensity after interacting with an attenuation coefficient of μ and a sample of thickness x is:

$$I_t = I_0 e^{-\mu x} \tag{1.1}$$

For x-rays (.5–50 keV), absorption of the incident photon by core-level electrons is possible. At this energy, nearly all the photon's energy is absorbed by the core electron, resulting in the characteristic absorption-edges first observed in 1920 [7, 8]. When an x-ray with energy greater than the binding energy of a core level electron is absorbed, the electron is "ejected" (i.e., "excited") and imparted with the excess energy in the form of momentum. Electrons ejected in this manner are referred to as "photoelectrons" or electrons in an excited state, and result in a "core-hole" at the core level of the electronic shell. This core-hole is filled either by an electron in a higher level and the emission of an Auger electron or fluorescence. In either case, the original excited electron decays rapidly—on the order of femtoseconds—and leads to the emergence of a "fine structure" in the absorption vs. incident photon energy plot. As the energy of the incident radiation increases, the photon's energy will eventually match the binding energy of a core-level electron. As a result, an "edge" in the spectrum will be observed. The location of these edges depends on the chemical and physical structure, as well as the electronic and vibrational states of the material. Absorption edges are like fingerprints used to identify elements, distinguish oxidation states, and even probe short-range order from the characteristic peaks and oscillations in the spectrum. XAFS spectroscopy can be performed on virtually any stable element since all atoms contain core-level electrons. Although a high-quality source of x-rays such as synchrotron radiation is required for the analysis, the ubiquity and utility of XAFS spectroscopy have made it an indispensable technique in fields such as materials biology, chemistry, and materials science [9] [10].

In experimental setups, particular energy photons are selected from the broad spectrum of synchrotron radiation via a pair of monochromators. The primary monochromator is a crystal with interplanar spacing (d) chosen specify to satisfy the Bragg equation 1.2, reflecting photons of wavelength, λ at angle θ.

$$n\lambda = 2d\sin(\theta) \tag{1.2}$$

The secondary monochromator removes higher-order harmonics that satisfy the Bragg equation ($2\lambda,\ 3\lambda,\ etc.$). Different wavelengths of light are selected by changing the angle. A

common method for XAFS analysis involves scanning the monochromators over an energy range. In the XAFS setup, two ionization chambers are used to measure the incident and transmitted light intensity. For any experiment, the absorption spectrum for a reference sample is also measured to calibrate the energy scale [11]. Two common sample holder cells are the the Nashner-Adler cell and the Clausen flow cell [12]. A diagram depicting a simplified experimental setup for an XAFS experiment is presented in Figure 1.1.

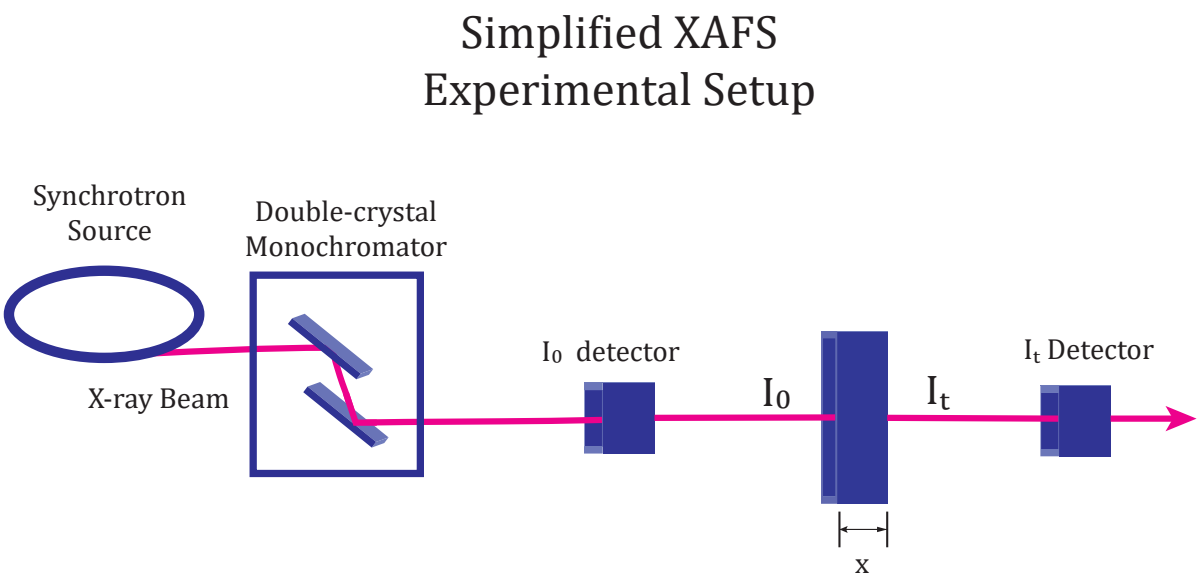


Figure 1.1: A simplified diagram for an x-ray absorption spectroscopy experiment. The high brilliance x-rays are produced from the synchrotron and then collimated through a monochromator. The beam is then split, with one half sent to an ionization chamber to measure the initial x-ray intensity, I_0. The other half is sent towards the sample, where a second ionization chamber awaits the beam on the other side to measure the intensity of the transmitted beam, I_t.

An example XAFS spectrum is plotted in Figure 1.2. The overall shape of the spectra is characterized by a large jump followed by oscillations. The absorption steadily decreases as a function of increasing energy. The slope of this decrease is often approximated by the Victoreen formula (1.3).

$$\mu_b(E) = aE^{-3} + bE^{-4} \tag{1.3}$$

The coefficients α and β can be found via a simple regression on a spectrum measured at pre-edge energies [13]. Typically XAFS spectra are normalized to affix the decaying absorption beyond the first edge to a value of one (See Figure 1.3). The large jump in absorption is

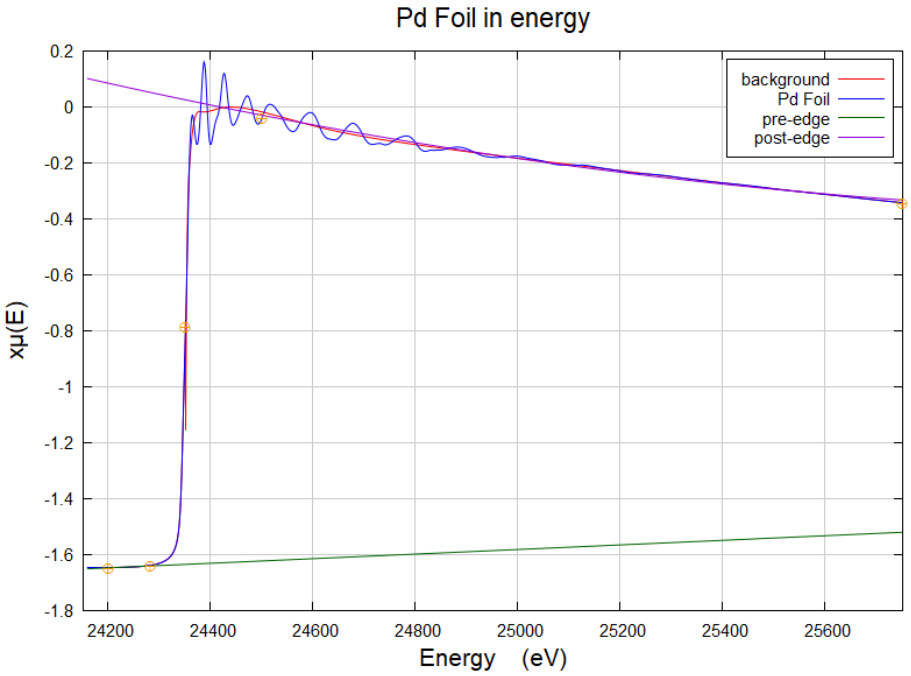


Figure 1.2: The transmission data for the k-edge of palladium (Pd) foil is plotted above. The data was collected at the as beamline at NSLSII and the figure from [12]. The raw absorption coefficient $x\mu(E)$ is the exponent found in the Beer-Lambert law in equation (1.1), where x is the sample thickness.

referred to as the "edge" and is caused by the sudden complete absorption of X-rays by electrons at the core binding energy. New edges are introduced when the incident energy matches the binding energy of a different energy electron. Each edge is labeled according to the quantum numbers: the K-edge corresponds to the 1_s level, L_1 to the 2s level, and L_2 and L_3 refer to the relativistically-split $P_{1/2}$ and $P_{3/2}$ level. This degeneracy causes the L_3 absorption edge to have a greater edge jump than L_1 or L_2 [14]. Regions far from the edge are relatively smooth and caused by the background absorption of x-rays. At energies beyond the edge, the absorption spectrum is characterized by oscillations referred to as the fine structure.

The XAFS spectrum is typically divided into two regions of study: the area near the first absorption peak (XANES) and the area after (EXAFS). XANES has a strong sensitivity to the oxidation state and coordination chemistry of the absorbing atom, while the EXAFS can be used to determine the bond lengths, coordination numbers, and atomic species of the absorbing atom's neighbors.

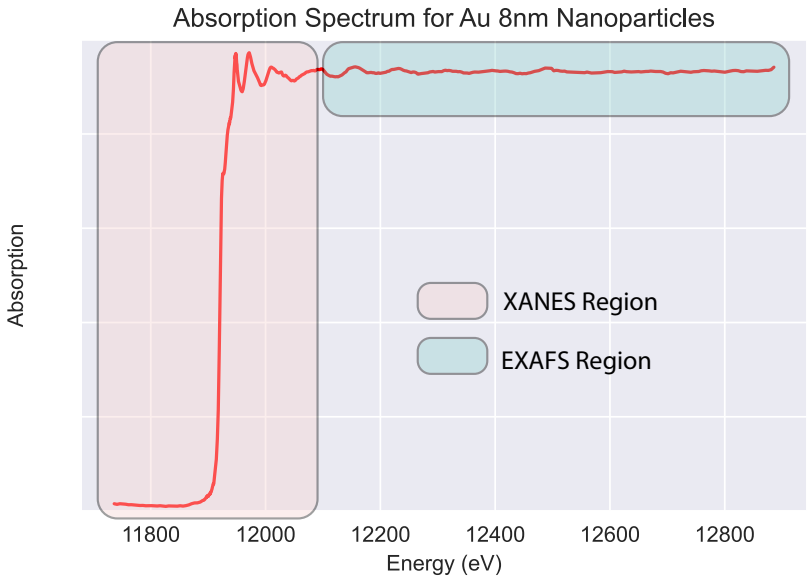


Figure 1.3: The absorption spectrum is divided into two regions: XANES and EXAFS. The XANES (x-ray absorption near edge structure) includes the lower energy region in the direct vicinity of the leading edge. The EXAFS (extended x-ray absorption fine structure) region includes all the energies beyond the XANES.

1.1.1 EXAFS

Beginning approximately 50 eV beyond the edge of the absorption spectrum lies the Extended X-ray Absorption Fine Structure (EXAFS) region. The spectral shape of this domain is determined by the multiple scattering of the photoelectron, interference of the incoming and outgoing waves of the photon, and electronic energy level splitting of the local structure. The oscillations in the EXAFS region are extremely sensitive to local bond lengths, coordination numbers, and atomic species of the surrounding elements. The EXAFS fine structure is the consequence of the photoelectrons' self-interference as they are scattered by the local atomic environment surrounding the absorbing atoms from which they were excited.

EXAFS Data Reduction

Preparing the absorption data for fitting requires series of steps in preparation. These preparation steps are known as data reduction, and the process of regularizing sample prepartion, thickness, and other experiment-specific measurements is known as "normalization." First, the pre-edge background must be removed using the Victoreen formula (1.3) or an alternative polynomial. Next, the atomic background, $\mu_0(E)$, is removed, and the measured absorption is normalized accordingly. This is a non-trivial process, as the absorption coefficient is not

that of a single, isolated atomic absorber; instead, it represents that of an atom and its surrounding neighbors [15]. High-order spline fittings must be used to remove the low-frequency, immeasurable oscillations caused by photoelectron scattering with nearby valence electrons. The energy-dependent absorption, $\mu(E)$ is related to $\chi(E)$ via equation (1.4):

$$\chi(E) = \frac{\mu(E) - \mu_0(E)}{\Delta\mu_0(E)} \tag{1.4}$$

For EXAFS fitting, the EXAFS function is typically transformed into k-space and plotted in terms of the photoelectron momentum, k [12]. Alternatively, it can be Fourier transformed into a radial distribution-like function (R-space) via (1.5) [16].

$$\widetilde{\chi}(r) = \frac{1}{2\pi}\int_0^\infty k^n \chi(k) e^{2ikr} dk \tag{1.5}$$

Two of the most popular EXAFS fitting programs, Artemis and Athena, convert to R-space via this technique.

The EXAFS Equation

With the data reduction complete, a multi-parameter fitting can be performed via the EXAFS equation. This equation is an approximation based principally on the single electron approximation of Fermi's golden rule (1.6).

$$\mu(\omega) \propto \sum_f^{E_f > E_F} \langle\psi_i|\Delta|\psi_f\rangle^2 \, \delta(E_F - E_i - \hbar\omega) \tag{1.6}$$

Fermi's golden rule describes the probability of a transition in state occurring. Here (1.6), $\mu(\omega)$ is the absorption coefficient, f and i are the final and initial eigenstates of the photoelectron, respectively, Δ is the dipole transition operator, and the delta function δ ensures conservation of energy. The transition operator Δ represents the interaction between the electrons and photon and is constrained by the dipole approximation. The constructive and descructive interference of the outgoing and backscattered waves modulates the transition matrix element coupling coupling the initial and final states, resulting in the fine structure in XAS. The summation of all energy values approximates the many-bodied problem as a single particle theory.

At energies above the core electron binding energy, excess energy is transferred to the photoelectron, which may then undergo multiple scattering with the surrounding atoms. The final wavefunction of the photoelectron is the superposition of the outgoing photoelectron

and the scattered wave [17]. EXAFS only takes into account the local region around the absorber at a distance r_j from the backscattering atom. The absorption coefficient, μ, is proportional to the transition rate given by Fermi's golden rule. Therefore, μ depends on the energy and symmetry of both the initial and final electron states [18].

For a system with multiple atoms, one must sum over all the backscattered waves. The total EXAFS signal is the sum of all possible photoelectron scattering paths; this includes both single scattering and multiple scattering. The summation over all single scattering paths in the first nearest-neighbor shell can be written as [19] [20]:

$$\chi(k) = \sum_i S_0^2 \int_0^\infty g_i(R) f_i(k,R) e^{-2R/\lambda(k)} \sin[2kR + \phi_i(k,R)] \frac{dR}{kR^2} \tag{1.7}$$

where χ is the total strength of the EXAFS signal and S_0 is an amplitude correction factor. Here (1.7), $g_i(R)$ is the radial distribution function (RDF). The terms S_0^2, $f_i(k,R)$, $\lambda(k)$, and $\phi_i(k,R)$ are related to the local structure described by the RDF and material specific. They are determined by factors such as interatomic distances, degree of structural disorder, and the number of nearest neighbors.

To account for the inevitable variation in bond lengths, the Debye-Waller factor σ_j is introduced; σ_j describes the standard deviation of bond-lengths of the sample.[1]. The lifetime of the photoelectron's excited state is taken into account by introducing an exponential term to account for the mean-free-path, $\exp(-2r_j/\lambda)$. Combining all this information, we arrive at the EXAFS equation (1.8).

$$\chi(k) = \sum_j \frac{N_j}{kr_j^2} F_j(k) \exp(-2\sigma_j^2 k^2) \exp(-\tfrac{2r_j}{\lambda(k)}) \sin[2kr_j + \phi_{ij}(k)] \tag{1.8}$$

To recapitulate all the terms in (1.8): N_j is the coordination number of atom j; k is the wavenumber; r_j is the distance to neighboring atom j;$F_j(k)$ is a scattering property of the neighboring atoms; $\exp(-2\sigma_j^2 k^2)$ is an exponential dampening term dependent on the variance in bond length σ_j^2; ϕ_{ij} includes the phase shifts for the incoming and outgoing waves and accounts for scatting off non-point-like atoms.

Although popular, the EXAFS equation is still a first-order approximation with assumptions and limitations. For example, the fitted disorder parameter, the Debye-Waller factor, explicitly assumes a Gaussian distribution of nearest-neighbor bond length distances. Other more accurate but computationally intensive alternatives are increasing in popularity. Such

[1]Note, this differs from the Debye-Waller factor used for x-ray absorption, which describes the broadening of a diffraction peak due to variations in inter-planar spacing [21]

alternatives include molecular dynamics (MD) [22], reverse Monte Carlo (RMC) simulations [23] and neural networks (NNs) [24] [25].

1.1.2 XANES

The XANES region, or the near-edge region, encodes the chemical and electronic structural information of the sample within its shape. The XANES shape reflects the lower energy photons that scatter much more strongly than in the EXAFS region. XANES also has the advantage of a higher signal-to-noise ratio than the subtle interference-determined oscillations found in EXAFS. While there is an "EXAFS Equation," there is no "XANES Equation" equivalent; however, this does not mean that there is no structural information encoded in the XANES, only that the theory is underdeveloped. Mainly, the strong errors in potential and many-body effects limit the development of a "XANES equation."

With the recent explosion in the popularity of machine learning, the investigation of the XANES latent space has become a topic of modern research is. In other words, *how much information is encoded in XANES?*

Recent work at Brookhaven National Laboratory [1] has shown that a model can learn structural descriptors from the XANES spectrum. Specifically, their method enables the decoding of XANES spectra to obtain the coordination number of metallic nanoparticles. In this 2017 paper, the group trained an artificial neural network (ANN) to recognize a relationship between the nanoparticle structure and the XANES spectrum. Once trained, the ANN is used to "invert" an unknown spectrum to obtain the corresponding structural descriptors of the catalyst. These descriptors, the coordination numbers, are used to calculate the number of shells (nanoparticle size) and shape (Archimedian solid) of the sample. While this model can determine the structure of nanoparticles from XANES —a feat previously only possible with the full EXAFS spectrum —it still has one major limitation: the ANN does not predict the disorder of the structure. The bond-length disorder is known to be an important descriptor for catalysts [26] [27]. Thus, discerning the catalyst size, shape, and disorder soly from XANES would provide an efficient and comprehensive tool for studying catalysis.

1.2 Goals of the Thesis and Approach

Building on the previous work [1], the goal of this thesis is comprised of two parts. First, we seek to determine whether bond-length information is encoded in the XANES spectrum.

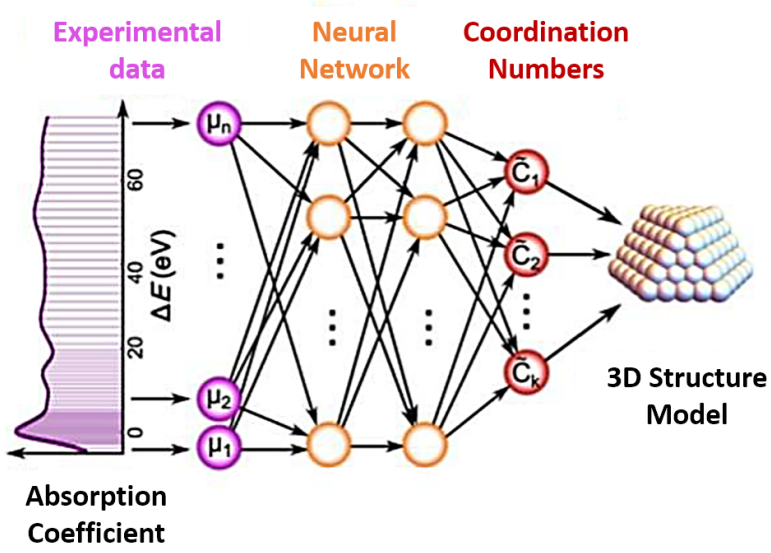


Figure 1.4: From [1], the neural network is trained to take a XANES spectrum from a metallic nanoparticle and predict the coordination number of the structure. This coordination number of nanoparticles is a known descriptor, which allows for easy calculation of the nanoparticle's size and shape.

Second, we utilize machine learning to predict the bond-length static disorder from a XANES spectrum. As with the 2017 BNL paper [1], the work in this thesis was conducted with gold (Au) due to the availability of relevant experimental data. While an ongoing goal is to expand this thesis's work to Au nanoparticles (among other elements), we primarily choose to simulate bulk Au to mitigate additional complexity driven by surface effects. Machine learning requires a substantial amount of training data, far more than could be experimentally obtained. Instead, we rely on absorption simulation software to create the training data: a collection of XANES spectra for large, bulk-like Au nanoparticles with known disorders. The network is then trained on the theoretical XANES spectra. Because of the systematic differences between the simulation and experimental data, the network must then be adjusted in order to be able to predict experimental data.

1.3 Outline of the Thesis

Often, the most time-intensive part of a machine learning-based project is the process of collecting and preprocessing the data. Chapter 2 is dedicated entirely to the process of

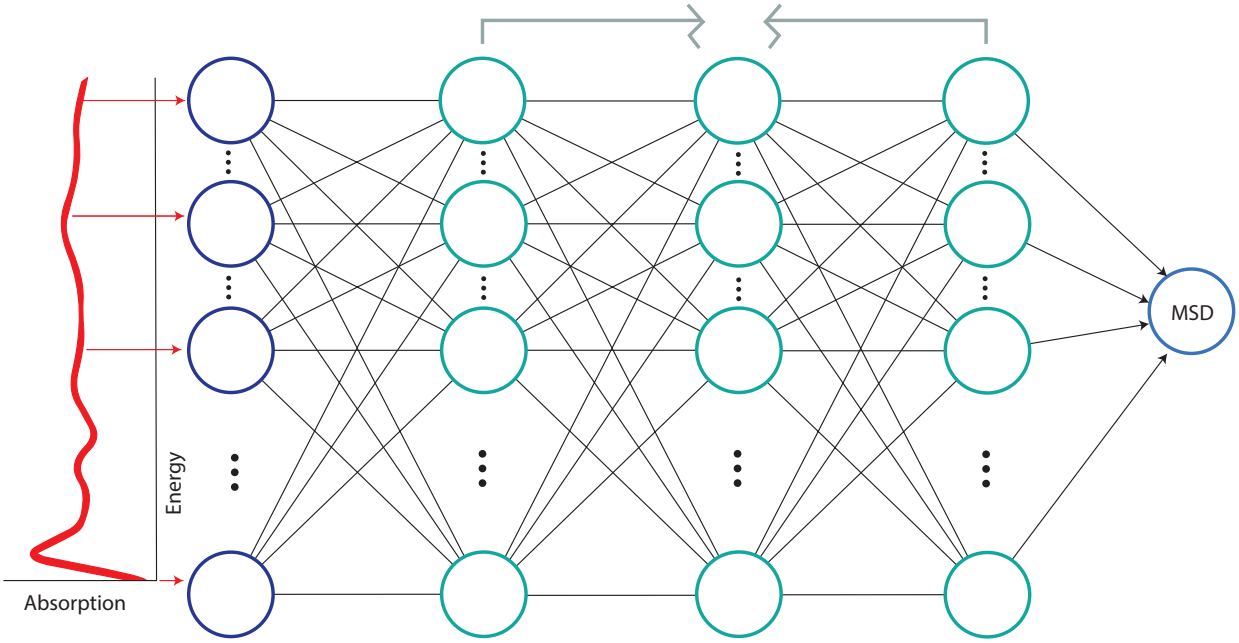


Figure 1.5: The goal of this thesis is to develop a neural network that inputs an absorption spectrum as an input—where each node represents the absorption at a given energy value—and predicts the mean squared disorder of the structure.

generating XANES spectra via simulations. Next, a solid foundational understanding of machine learning is integral in understanding the approach. All the machine learning terms present in later chapters are defined here. Chapter 4 describes the specif1ic model architecture and results of the training process. Further discussion and comments on future work are included in chapter 5. Appendix A includes a description of the main Python scripts written for this thesis, which are necessary for replicating the work. The file structure and some individual functions are included to clarify the calculation or creation of selected parameters and files. The files and scripts are available upon request by contacting the author. Finally, Appendix B is included to provide a more comprehensive explanation of the FEFF simulation software.

Chapter 2

Simulating Disorder

Before making any predictions, neural networks must first be trained on a large quantity of data. In the context of this thesis, to teach our neural network to predict the mean squared displacement (MSD) from XANES, we must first assemble a collection of training data comprised of XANES spectra, each labeled with a known MSD. Gathering such a large quantity of high-quality experimental data would be impractically time-intensive and expensive. Instead, simulations provide a practical alternative, though even simulating each possible disordered structure is time-intensive. This process has been standard practice in theoretical XAFS work and discussed in section 2.1.

We have conducted a significant amount of work in developing a new approach for simulating disordered XANES spectra. A discussion on the development of this new, statistical approach is presented in sections 2.2.1–2.2.3. This new process is based on the statistical averaging of non-disordered structures. Instead of simulating hundreds of defined, disordered structures, we run many XANES simulations of simple, non-disordered structures and generate the disordered spectra via purposeful statistical averaging. In sections 2.2.1–2.2.3, we explain this statistical weighting process in-depth, beginning with the creation of simple, non-disordered spectra for the FEFF input files and culminating in the creation of many possible disordered spectra with known MSDs. Finally, the efficacy and limitations of this approach are discussed in section 2.3.

2.1 Traditional Particle-Averaged Simulations

We refer to the traditional method for running FEFF simulations as particle-averaged FEFF. The simulation software, FEFF9 [2], only simulates the absorption spectrum for one absorber

at a time. For this project, we are interested, at first, in simulating bulk Au. Accordingly, we attempt to remove the surface effects on the spectrum by simulating the absorption of only the first shell atoms and averaging (arithmetic mean) the results. This way, the first shell absorber atoms are surrounded by bulk structure, allowing FEFF to minimize the contribution of surface effects which affect the potential and multiple scatterings differently than the bulk. We made the decision to first simulate bulk Au in order to reduce the number of confounding variables. If the statistical averaging technique works for simulating disordered bulk structures, the next stage would be to expand the process to nanoparticles.

Simulating absorption spectra via FEFF requires the creation of FEFF input files which include various user-defined parameters and the Euclidean coordinates of the structure. Determining how to create these structures is a project in and of itself. The approach taken for this experiment was to start with the perfectly ordered crystal of Au atoms. Then, for each structure, each individual atom is shifted by a random distance in a random direction. For each atom (in a given structure), the direction to be shifted is chosen from a uniform distribution, whereas the distance by which to shift the atom from its original location is chosen from a Gaussian. The standard deviation of the Gaussian from which the shifted distance is drawn differs from structure to structure. Thus, with a narrow width Gaussian, the shifted distances on average tend to be smaller than for a gaussian with a large width. Therefore, structures with small MSDs tend to be created when the width of the Gaussian for shift distances is narrow, whereas high MSD structures tend to be created when the width of the Gaussian is wide. The code for this script can be found in Appendix A.3.

Absorption simulation software is far from perfect, especially for XANES. There are different ways to treat the electronic potential and Fermi energies of the structure, none of which work best in all circumstances. FEFF works by simulating the green's function. A more in-depth description of how FEFF works can be found in Appendix B. FEFF includes many user-defined parameters which can be tweaked to alter the resulting function. By comparing the resulting FEFF spectrum to a reference experimental spectrum, an exhaustive grid search (parameter sweep) can be run to determine the best FEFF parameters for simulating a given structure. Each FEFF input file was run with the following paramters:

```
1                SCF 4.6 0 30 .5 1
2                EDGE    L3
3                EXCHANGE    5   0.2 0.5
4                S02 1.
5                XANES   3.7 0.05    0.1
6                FMS 7
7
8                POTENTIALS
9                0       79      Au      -1      -1      0.
10               1       79      Au      -1      -1      0.
```

An explanation for the meaning behind these parameters can be found in Appendix section B.2.

While particle averaging is the traditional method for simulating XANES via FEFF, the method is computationally intensive for large structures with many absorbers. One advantage of developing a statistical averaging method would be the ability to create an infinite number of disordered XANES spectra from a limited number of simulations. For particle averaging, to create 1000 disordered structures requires 13,000 simulations[1], a process that necessitates several days of computation time on a dedicated distributed computing cluster.

2.2 Statistical-Averaged Simulations

To expedite the process of simulating XANES spectra of disordered structures, we have developed a new methodology build on a statistical averaging of a few easily simulated structures.

2.2.1 Generating Distortion Not Disorder

Instead of creating structures with a range of *disorder*, we begin by generating structures with a range of *distortion*. Whereas *disorder* refers to a stochastic displacement of atoms from their original positions characterized by MSD and the width σ^2 of a radial distribution function, *distortion* refers to isotropic expansion or contraction of the subject. Equivalently, we define distortion as a radial shift in all atomic positions away from (or towards) the center atomic absorber.

[1] 1000 structures $\times$ 13 absorbers per structure = 13,000 simulations

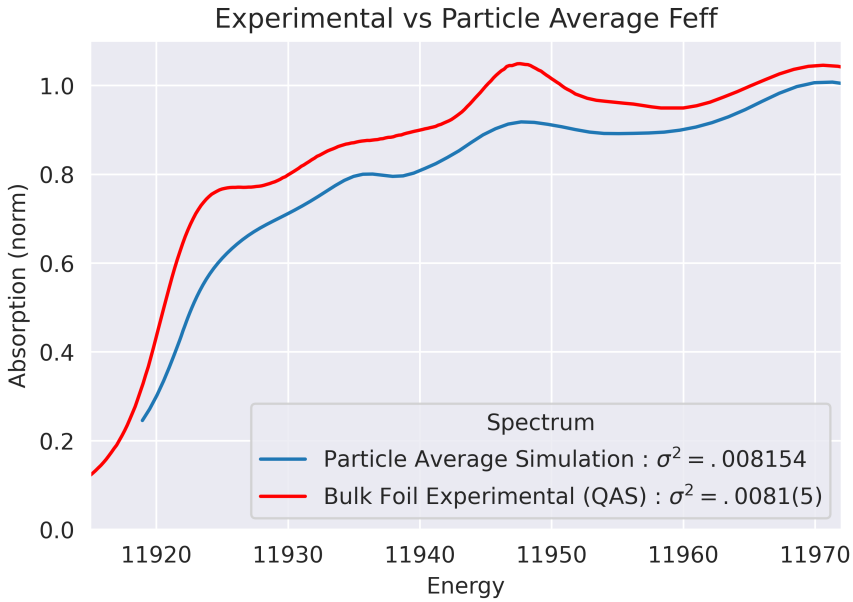


Figure 2.1: Comparing the bulk foil (red) measurement to a simulated large, bulk-like nanoparticle with the same disorder. The FEFF-simulated version underestimates the magnitude of the oscillations at lower energies.

A 2-dimensional projection of this isotropic distortion is presented in Figure 2.2. Though the figure only shows the xy-plane projection of the first 12 nearest neighbors, the actual structure consists of either 55 atoms to simulate nanoparticles, or the first four shells (561 atoms) to simulate bulk materials. Both structures were created with a lattice constant of 4.0782 Å to match that of bulk Au [28]. In reality, the nearest-neighbor distances for Au nanoparticles are likely smaller than for the bulk structure; this can be accounted for later on in the averaging process since the original coordinates will only be one structure out of many. The important part is that the BCC crystal structure is correct.

We generate a total of 91 FEFF input files with different levels of distortion. Each file contains the same center absorber located at $(0, 0, 0)$ with all the Euclidean distances from the center expanded or contracted radially. All the first shell atomic coordinates are shifted on the range of -0.45 Å to $+0.45$ Å in increments of 0.01 Å. For example, the FEFF input file with the greatest inward shift contains all first nearest neighbor atoms shifted 0.45 Å radially inwards towards the center absorber, and the FEFF input file with the largest outwards shift has the first nearest neighbor coordinates shifted 0.45 Å radially outwards away from the center absorber. The atoms in the outer shells are scaled accordingly to preserve the crystal structure according to equation 2.1

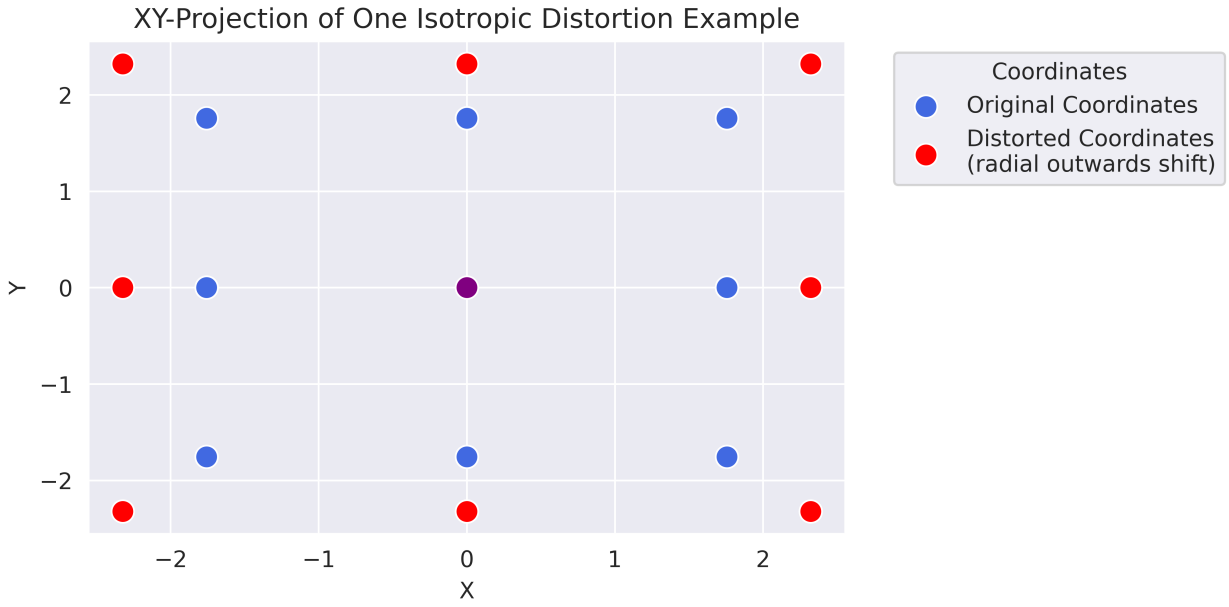


Figure 2.2: Each point represents one of the 12 nearest neighbor atoms surrounded the center of an Au cluster projected onto the xy-plane. The four corner points actually represent two atoms each because of the projection. The blue atoms represent the original coordinates, and the red atoms represent the radially shifted coordinates. The center absorber atom is purple since its original position is the same as its distorted position.

$$\boldsymbol{\rho}_{shifted}^{(i)} = \boldsymbol{\rho}_{\mathbf{0}}^{(i)} \left(\frac{a+\delta}{a} \right) \tag{2.1}$$

where $\boldsymbol{\rho}_0^{(i)}$ is the vector from the center absorber to the original (lattice) position of atom i, a is the lattice constant of Au, and $\delta \in [-0.45,\ 0.45]$ and is the distance the atoms in the first shell will be shifted.

Running the 91 simulations (one for each of the distorted structures) takes approximately 30 minutes. Were we to generate thousands more or employ RMC or MD, this process could take days or weeks of computation time. We plot the resulting XANES spectra from the FEFF simulations in Figure 2.3.

2.2.2 Generating Disorder via Probability Distribution Averaging

One way to characterize system disorder is with the Gaussian standard deviation, σ, of the radial distribution function. The idea of our statistical averaging method is to emulate this width by weighting the simulated XANES spectra according to this distribution. An illustration of this relative weighting is depicted in Figure 2.4 as a histogram with $\sigma = 0.1$ Å.

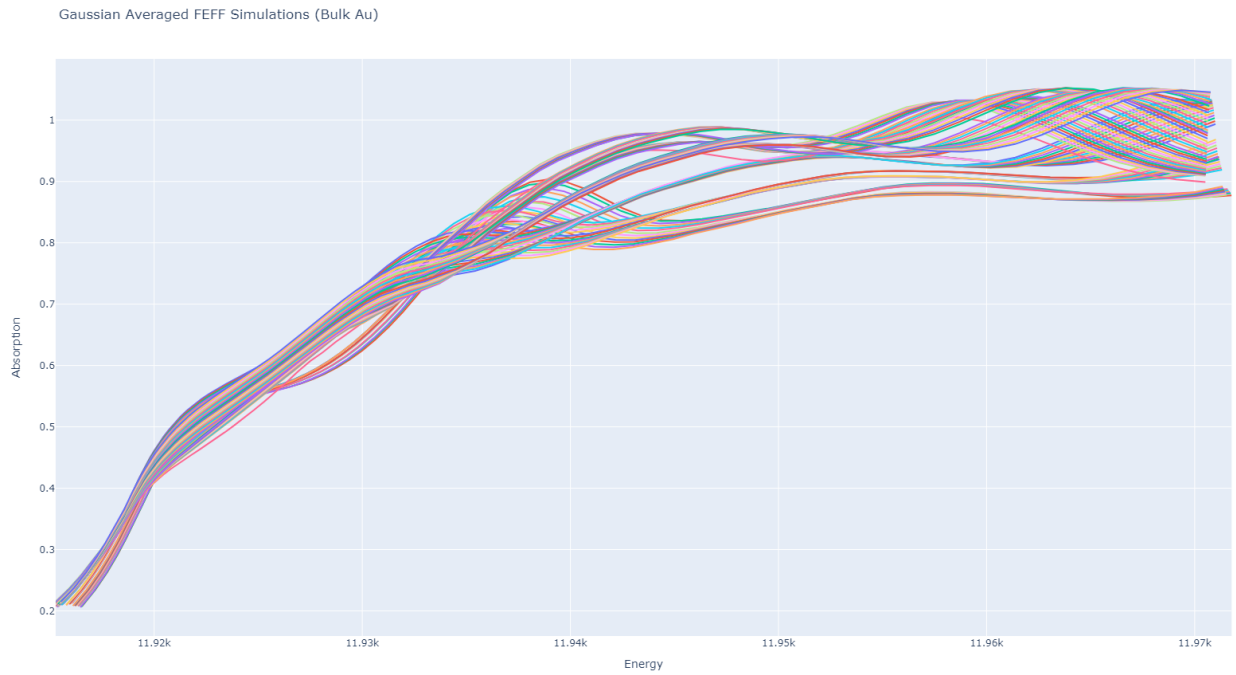


Figure 2.3: Each spectrum represents the FEFF simulation results for a different distorted structure. For each spectrum, the crystal structure and center absorber remain constant, the only parameter that varies is the euclidean distance from the center to the other coordinates.

Each histogram bin represents a simulated XANES spectrum with a different isotropic displacement. For example, the bin at $\Delta\rho = 0.0$ Å represents the simulated XANES spectrum with no distortion, and the bin at $\Delta\rho = -0.2$ Å represents the simulated XANES spectrum with all the atomic coordinates shifted isotropically inwards towards the center absorber by 0.2 Å. The height of each bin, $f(\Delta\rho)$, represents the relative contribution of each simulated XANES spectrum towards the resulting weighted spectrum. For visual clarity, Figure 2.4 depicts only 40 bins; the actual weighting includes 91 bins ranging from -0.45 Å to $+0.45$ Å.

The disordered, Gaussian-averaged XANES spectrum, $\langle\mu(E)\rangle$, using the histogram weighting of the Gaussian in Figure 2.4 is calculate via equation (2.2):

$$\langle\mu(E)\rangle = \frac{1}{S}\sum_{\Delta\rho=-.45}^{+.45} g\left(\Delta\rho \mid \mu = 0, \sigma = 0.1\right)\mu(E \mid \Delta\rho) \tag{2.2}$$

In the above equation, $\Delta\rho$ is the isotropic radial displacement of each atom from its original position, $\mu(E \mid \Delta\rho)$ is the simulated FEFF spectrum for the given $\Delta\rho$ configuration, and S represents a standardization factor included to negate the effect of the changing Gaussian height as a function of the variance, σ^2. With the inclusion of S, only the relative height of

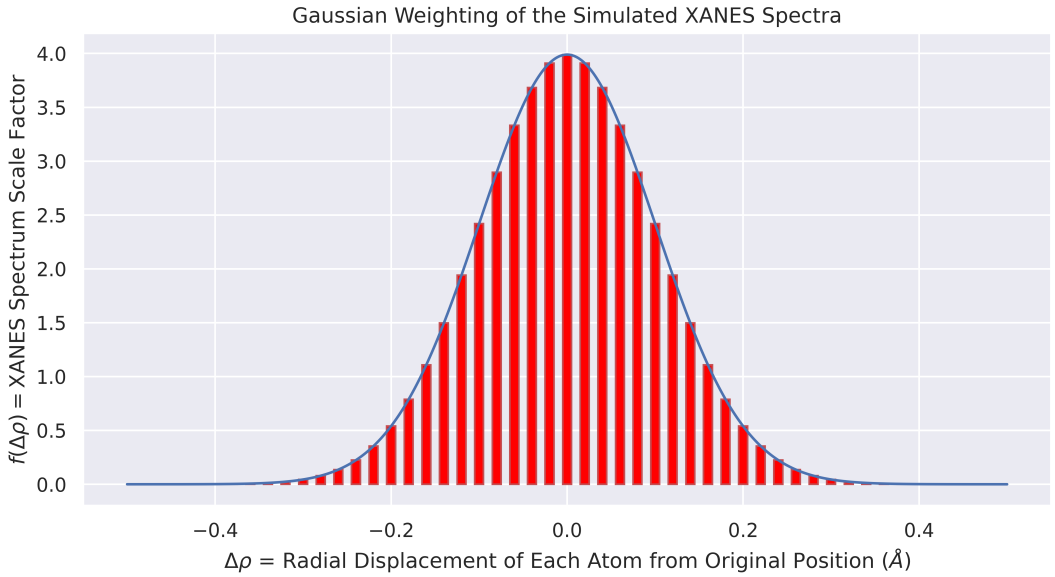


Figure 2.4: A Gaussian distribution probability density function can be used to calculate the relative weight of each FEFF-generated XANES spectrum towards one simulated, disordered spectrum. Each bin (red bar) represents a FEFF-generated spectrum; the x-axis is the isotropic shift of the first nearest neightbor atomic positions, and the y-axis is the relative weight factor.

each bin matters for producing the averaged XANES spectrum. This standardization factor is defined in Equation (2.3):

$$S = \sum_{\Delta\rho=-.45}^{+.45} g\left(\Delta\rho \mid \mu = 0, \sigma = 0.01\right) \tag{2.3}$$

In both equations (2.2) and (2.3), the function g is just the typical Gaussian distribution probability density function (Equation 2.4):

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{2.4}$$

The above example only generates one (simulated) disordered XANES spectrum and does so via weighting of a Gaussian distribution with mean and variance equal to 0 and 0.01, respectively. To simulate systems with different degrees of disorder, we can vary the shape of the probability density function. With a Gaussian distribution, we can only vary the mean and variance; to simulate even more conditions, however, we can instead use the multivariate skew normal (skew-norm) distribution, $f(x)$ [29] [30], written in equation (2.5).

$$f(x) = 2\phi(x)\Phi(\alpha x) \tag{2.5}$$

where $\phi(x)$ is the Gaussian PDF:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{2.6}$$

and $\Phi(x)$ is the Gaussian CDF:

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)\ dt \tag{2.7}$$

Equation (2.5) includes the shape parameter, α, which has the nice property of producing a right-skewed distribution when positive and a left-skewed distribution when negative. When $\alpha = 0$, the distribution simply produces the typical Gaussian distribution (2.4). Utilizing equation (2.5), we can vary μ, σ, and α to alter the first four moments of the function: the mean, standard deviation, skew, and kurtosis. Eighteen possible skew-norm weighting functions are plotted in Figure 2.5, however, 1000+ unique combinations of weightings would be required to produce the neural network's training data.

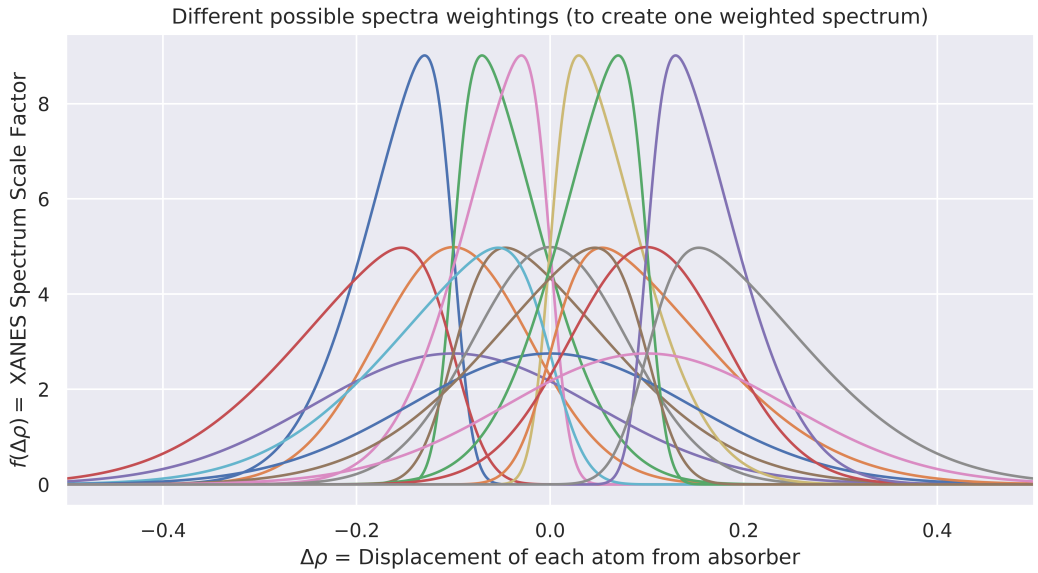


Figure 2.5: Eighteen skew-norm distributions plotted with all possible combinations of $\sigma \in \{.08, .145\}$, $\mu \in \{-.1, 0, .1\}$, and $\alpha \in \{-5, 0, 5\}$. Each represents a possible way to produce a simulated, disordered spectrum from many FEFF-simulated, distorted spectra

The disorder of the skew-norm-generated, disordered spectrum is characterized by the

mean squared displacement of each atom from its original position ($\Delta\rho$), weighted in the same manner as the spectra. Note, this is different than the standard deviation of the Gaussian used to create it. The weighted mean squared displacement, MSD, is calculated via as follows:

$$S = \sum_{\Delta\rho=-.45}^{+.45} f\left(\Delta\rho \mid \mu, \sigma^2, \alpha\right) \tag{2.8}$$

$$\mu_{weighted} = \frac{1}{S} \sum_{\Delta\rho=-.45}^{+.45} \Delta\rho \left(f(\Delta\rho \mid \mu, \sigma^2, \alpha)\right) \tag{2.9}$$

$$MSD = \frac{1}{S} \sum_{\Delta\rho=-.45}^{+.45} \Delta\rho \left(f(\Delta\rho \mid \mu, \sigma^2, \alpha) - \mu_{weighted}\right)^2 \tag{2.10}$$

Here, f is the skew-norm function from equation (2.5). The code for this equation can be found in Appendix A.2, written in Python and optimized with NumPy [31].

2.2.3 Simulation vs. Experimental Data

To check our FEFF simulation parameters, as well as the validity of the Gaussian-weighted disorder technique, we compare the simulation data to experimental data [32] [33] [34]. In Figure 2.6, both experimental and simulation spectra for bulk-like and nanoparticle scenarios are plotted. EXAFS fitting was used to characterize the disorder in the experimental measurements. For the bulk foil (unpublished data), this parameter was found to be $\sigma^2 = 0.0081(5)$ $\mathring{A}^2$, and for the 8 nm disordered particle, $\sigma^2 = 0.0102(8)$ $\mathring{A}^2$. One simulated disordered spectrum was weighted according to the Gaussian $N(0, 0.09)$ to represent the disordered nanoparticle, and the other was weighted according to the Gaussian $N(0, 0.038)$ to represent the bulk. These weightings correspond to MSD values that match the measured σ^2 values for the experimental data.

In Figure 2.6, the bulk Au foil spectrum is above the 8 nm nanoparticle spectrum (more absorption) until the peak around 11937 eV, where the nanoparticle's absorption becomes higher. The two spectra criss-cross again over the next two peaks, changing which material absorbs more at a given energy range. This change is more easily seen in Figure 2.7, which plots the difference between the bulk material and the nanoparticle absorption for both the experimental measurements and the simulations. The experimental and simulation difference-spectra follow the same trend with the exception of the peak around 11947 eV.

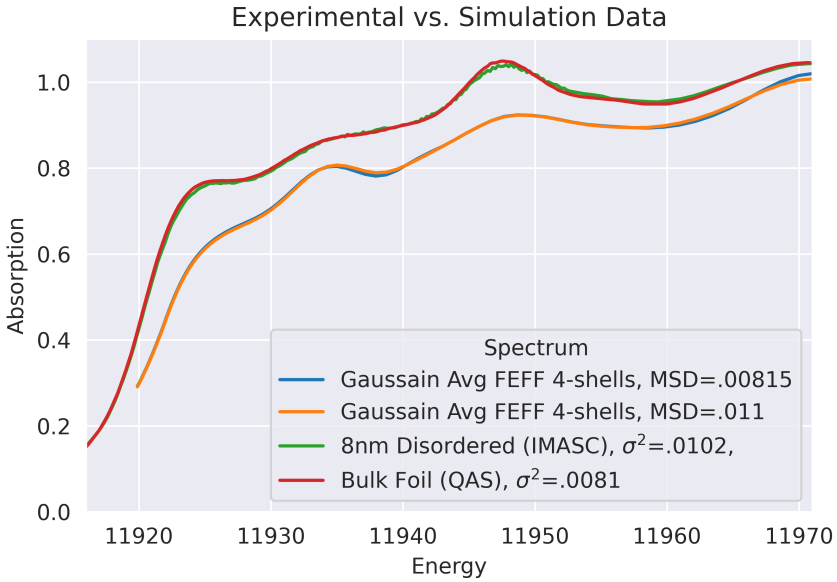


Figure 2.6: Comparing the bulk foil (red) measurement to the 8 nm disordered nanoparticle (green) measurement is an analog to comparing the simulated, non disordered FEFF spectrum (blue) to the simulated disordered spectrum (orange).

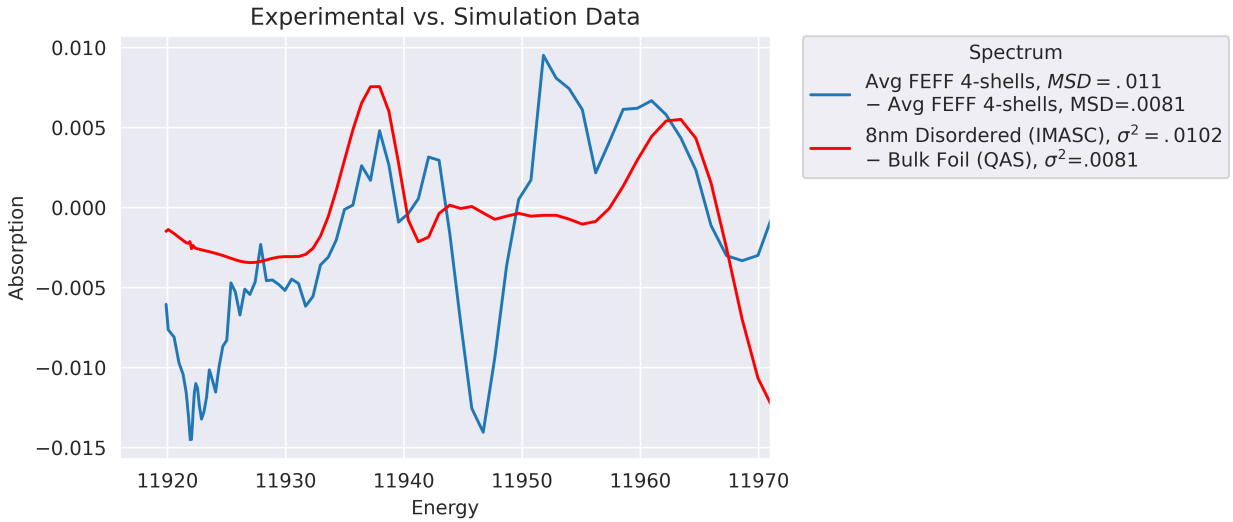


Figure 2.7: The difference between the nanoparticle spectrum and the bulk spectrum are plotted for the same data as in Figure 2.6. It is easier to see where the bulk and the nanoparticle absorption crisscross by plotting the difference.

Figures 2.6 and 2.7 are not meant to be perfect comparisons of simulations vs. experimental data. For one, the experimental data includes a comparison between a bulk spectrum to a nanoparticle. By contrast, both the simulation spectra are from identically sized 55 atom Au clusters. Still, much of the disorder trends are coded in the simulation approach.

To test if nanoparticle size information is also coded in our Gaussian-averaged simulations, we compare different size nanoparticle simulations to experimental data in Figure 2.8. We first simulated two different size nanoparticle gold clusters, one with 13 Au atoms and one with 55. As expected, including more atoms in the simulation produces more bulk-like spectrum characteristics, such as larger amplitude peaks.

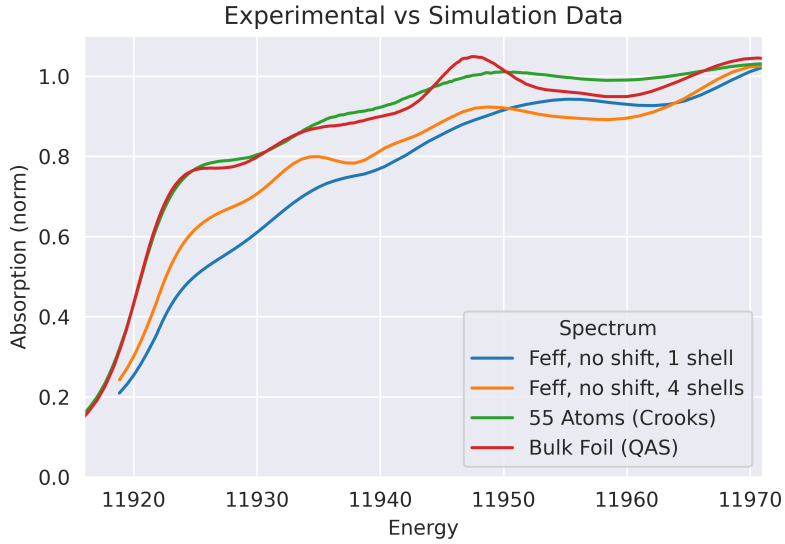


Figure 2.8: Comparing the bulk foil (red) measurement to the 55 atom nanoparticle (green) measurement is an analog to comparing the 13 atom simulated spectrum (blue) to the 55 atom simulated spectrum (orange).

These preliminary indications suggest the statistical approach may provide useful insights given the similarity in trends; however, this does not yet provide any indication as to whether the statistical averaged simulations are equivalent to the particle average simulations, nor do they tell us whether these spectra would work as training data for predicting the mean squared disorder from experimental spectra. This broader discussion is the topic of the next section.

2.3 Particle-Averaged FEFF vs. Skewnorm-Averaged Structures

While it is important to see that the skew-norm-averaging methodology maintains the systematic changes in XANES spectra as disorder increases (as seen in section 2.2.3), it is more important to compare this methodology to the widely-used particle-averaged methodology

(seen in section 2.1). Unfortunately, this is where the efficacy of the statistical methodology breaks down and the limitations of the method are revealed. As seen in figures 2.9 and 2.10, the particle averaged FEFF spectra and the skew-norm averaged spectra are distinctly different. We showcase two examples using skew-norm averaged, one for a bulk-like nanoparticle with low-disorder (figure 2.9) and one for bulk-like nanoparticle with high-disorder (2.10). Averaging with a Gaussian instead of the skew-norm produced the same problematic results. This is likely a result of the false assumption that the linear combinations of $\mu(E)$ calculated on ordered structures with different lattice parameters contribute equivalently to that of actual disordered structures.

Because of the discrepancy in spectral shape between particle-averaged and skew-norm averaged FEFF simulations for structures with the same disorder, the statistical-based methodology work was abandoned for this thesis. This does not mean, however, that the skew-norm approach is without merit. Preliminary work has shown it is possible to create identical spectra through selective weighting of the distorted (non-disordered) spectra. One current limitation, however, is that the shape of the weightings required to produce these identical spectra remains unpredictable. Sometimes the weightings resemble Gaussian distribution similar to the radial distribution function; however, this is not always the case. Sometimes jagged, multimodal weightings can produce very similar spectra. While it is in theory possible in experimental data for wildly different RDFs to have the same MSD value, we expect these strangely shaped distributions to be exceedingly unlikely in nature (from an entropic, configurational perspective), and for disorder in bond lengths to be generally Gaussian in shape. Further work to understand the relationship between different distorted spectral weightings, FEFF simulations, and the mean squared disorder of the structure may provide insight into the contribution of atomic displacement in a XANES spectrum for disordered structures. Consequently, for the remainder of this thesis, particle-averaged FEFF spectra were used as the training data for the neural network.

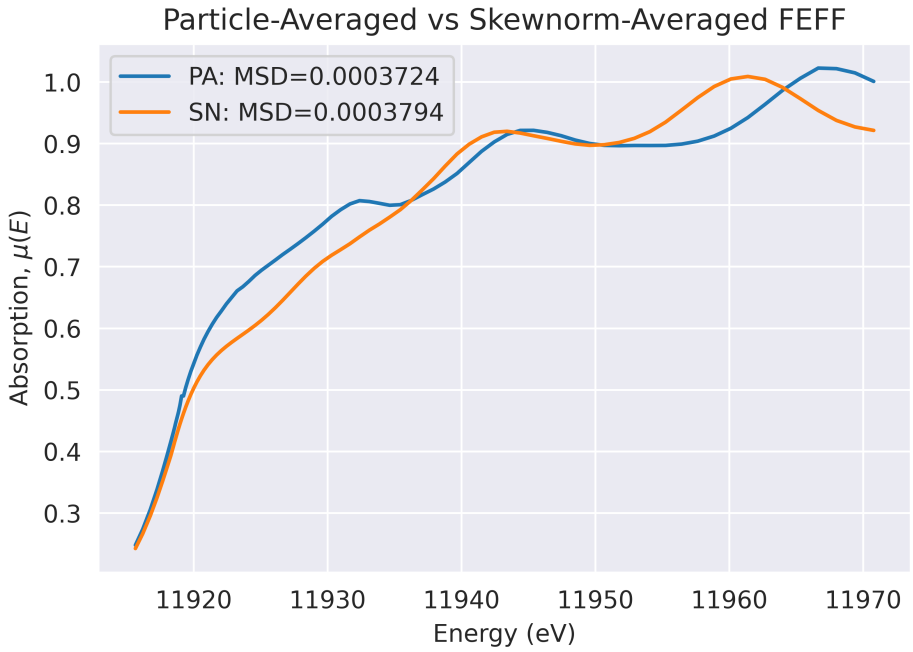


Figure 2.9: The particle Averaged FEFF (PA) and Skewnorm-Averaged FEFF (PA) for low-disorered bulk-like Au nanoparticles are plotted above. The systematic difference reveal a flawed assumption in the skew-norm-averaged methodology

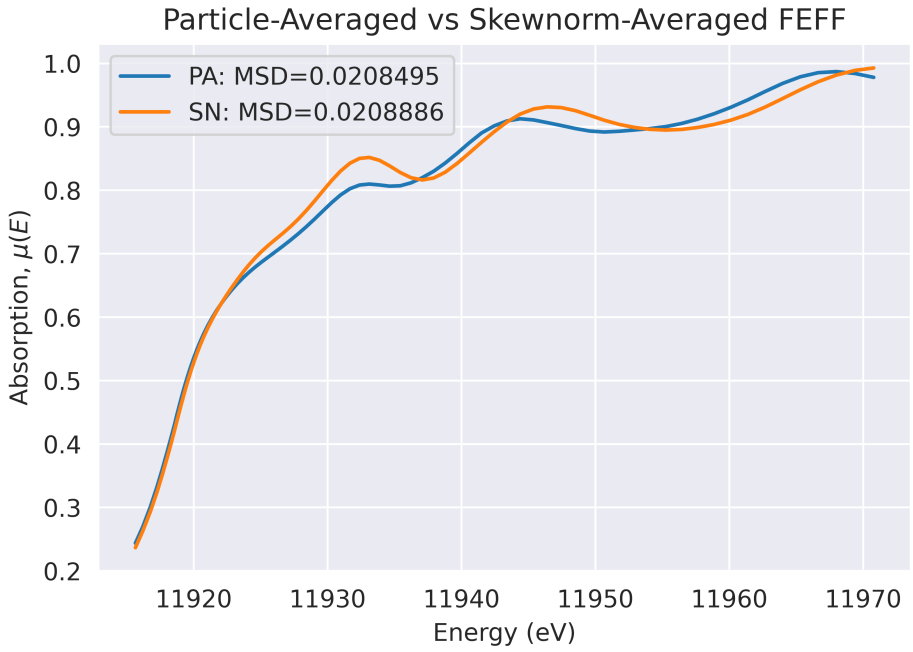


Figure 2.10: The particle Averaged FEFF (PA) and Skewnorm-Averaged FEFF (SN) for particles high-disordered bulk-like Au nanoparticles are plotted above. The systematic differences between the particle-averaged FEFF and the statistical averaged methods are less significant for particles with a high degree of disorder. The persistent differences, however, suggest a false assumption in the statistical averaging methodology.

Chapter 3

Machine Learning

Predicting disorder from XANES spectra requires a sophisticated model or algorithm capable of extracting non-linear features from the data. Increasing the disorder in the structure does not simply shift or scale the spectrum by a scalar; instead, disorder alters the spectrum in a complex and unknown way, for which we rely on machine learning (ML) to discern. The general goal of ML is to recognize patterns in data, iteratively learning to solve complex, non-linear problems and make powerful predictions on new, unseen data [35]. Due to the integral role ML plays in this thesis, the following chapter serves to establish how neural networks fundamentally operate and define all the terms necessary for understanding the neural network implementations discussed in chapter 4.2.

We begin by clarifying and distinguishing the following commonly misused terms: machine learning (ML), deep learning, and artificial intelligence (AI). ML is the most general term out of the three, referring to the computational technique of fitting a model on a dataset via an iterative training process. The models created in ML can either be regressors or classifiers: the former predicts a continuous range of values, whereas the latter is a discrete predictor. Artificial Neural networks (ANNs), or neural networks (NNs) for short, are one example of a machine learning model that tends to be complex and computationally intensive to train. As a result, ANNs are often highly non-linear models capable of solving complex tasks such as object detection [36] or speech recognition [37] [38]. Neural networks were originlly inspired by biological nervous systems, and fittingly, their graphical representations include terms such as "nodes" and "connections." The field of ML involving ANNs with many layers is referred to as deep learning [39]. AI is a subfield of deep learning where a neural network is trained to generate human-like responses. Common examples of AI are generative chatbots [40] and a virtual assistants [41] [42].

Using the above terminology, we can reframe the goal of this thesis: to predict disorder from XANES, we utilize deep learning to train a regression-based artificial neural network. The following sections walk through the mathematical process of training a simple neural network. In practice, sophisticated APIs such as Google's TensorFlow [43] or Facebook's PyTorch [44] handle the mathematical backend and optimization; however, one must first build a fundamental understanding of the methods these frameworks are executing before attempting to implement them.

3.1 Feedforward and Backpropagation in ANNs

The process where an ANN passes information from the input to the subsequent layers to make a prediction is called the "feedforward process." The process of updating the parameters of a neural network is called backpropagation. Whereas feedforward is essentially a chain of linear algebra operations, backpropagation relies principally on vector calculus. Neither action is particularly mathematically complicated; however, there are many parts, and it is easy to get lost in the sea of similar-looking partial derivatives. In this next section (3.1.1), we explicitly walk through the mathematics of the feedforward process for a fully connected (affine) neural network.

3.1.1 Feedforward

Consider the neural network in Figure 3.3: it contains an input layer with three nodes, a single hidden layer with five nodes, and an output layer with two nodes. The input layer (zeroth layer) has a cardinality of $\mu = 3$ and is represented in Einstein notation[1] as the covector (row vector) x_μ. Each edge in the graph represents a weight that will be multiplied by the connecting node on its left in the feedforward process. First, each node in the input layer is multiplied by the weight of the connecting edge and added together. Applying this operation for all input nodes and weights can thus be represented as the inner (dot) product of the input layer row vector and a weights matrix. The hidden layer (first layer) has a cardinality of $\nu = 5$. Thus, the resulting inner product is $x_\mu W_\nu^{\mu(1)}$, where $W_\nu^{\mu(1)}$ represents the matrix of weights connecting the zeroth and first layer. While this result has the correct dimensionality for the hidden layer, there are still two operations required to produce the actual values for the nodes $h_\nu^{(1)}$. First, a small trainable parameter, b_ν is added

[1]Recall that in Einstein notation repeated indices are implicitly summed over. For example, $u^i = A^i_j x_j = \sum_{j=1}^{5} A_{ij} x_j$

to every element in the resulting vector from the previous calculation. The values in this row vector are called biases and introduced to prevent overfitting—i.e. the phenomenon where a model predicts the training data well but does not generalize to reliably predict unseen data. Biases are a regularization parameter. Regularization techniques are discussed below, and an example is provided in Figure 3.2. The final operation applied to produce the first hidden layer's values is known as an activation function. These functions are applied element-wise to the layer, and without them, neural networks would be unable to learn non-linear features. There are several types of activation functions, the three most common being sigmoid, tanh, and ReLU.

Sigmoid and Tanh

The sigmoid and tanh activation functions are defined as the following:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.2}$$

Note, sometimes the term "sigmoid" is used to refer to the shape an s-shaped curve, so both equations (3.1) and (3.2) are considered sigmoids. In machine learning, however, the sigmoid function always refers to equation (3.1). Note that the sigmoid function maps the input between zero and one. Hence, the it is often used in the final layer of ANNs to output a probability. Note that sigmoid asymptoically approach their minimum and maximum values (for sigmoid, 0 to 1; for tanh -1 to 1) around $x = -4$ and $x = 4$ respectively, meaning that the sigmoid activation function is only useful within that limited range of input values. One issue with both the sigmoid and tanh activation functions has the potential for creating a vanishing gradient. The gradient of either of these functions approaches zero for values above four. This asymptotic approach hurts the ability of the NN to meaningfully update its trainable parameters [45]. The importance of calculating the gradients of these activation functions will be discussed in section 3.1.3 within the context of backpropagation.

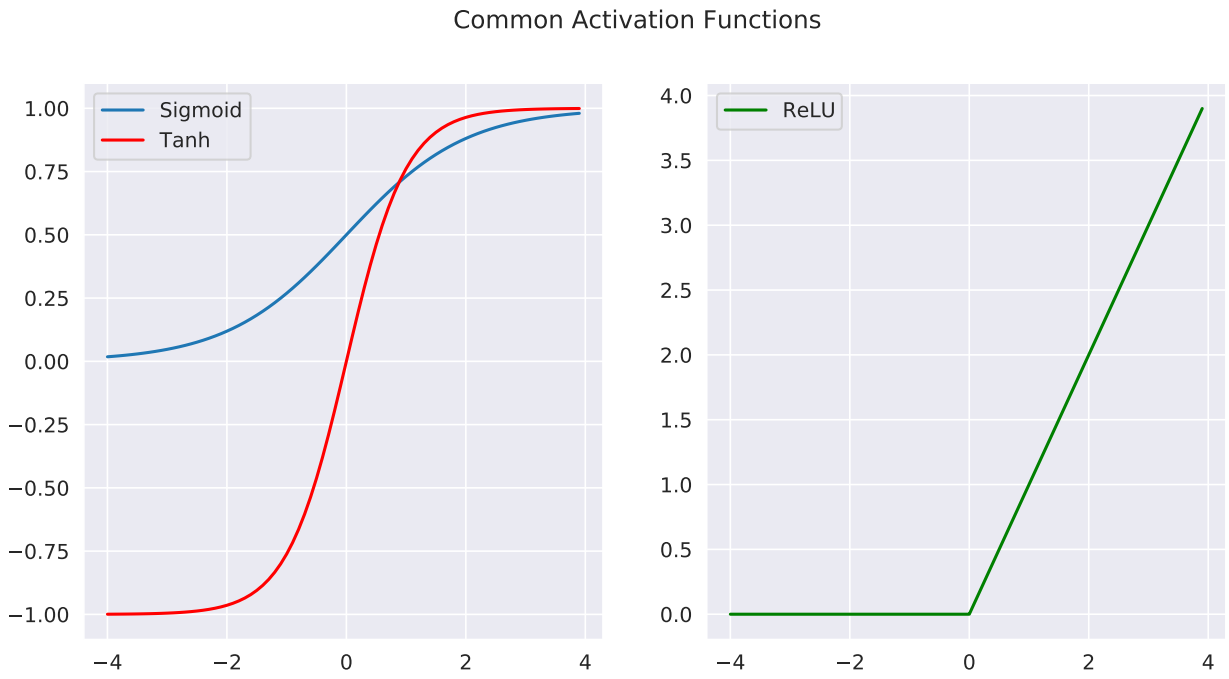


Figure 3.1: Plotted above are three common activation functions: sigmoid, tanh, and ReLU. Sigmoid and tanh are particularly useful for scaling the output of a neural network layer to be within a given range. ReLU and its variations are useful for deep ANNs, where vanishing gradients are problematic.

ReLU

The **Re**ctified **L**inear **U**nit activation function (ReLU) has become an important staple of machine learning. Conventially, it is written as $f(x)$ and defined as:

$$f(x) = \begin{cases} 0 & \text{if } x <= 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.3}$$

ReLU is important because it provides a much greater range in values as outputs. Whereas sigmoid and tanh saturate around $x = 4$, ReLU never saturates for linear values. Additionally, ReLU is simple to calculate and tends to help neural networks converge quickly. Further, because ReLU returns 0 for any negative value fed forward into the node, many ReLU activation functions in a given model help lead to sparser layers, reducing the overall complexity of the model and helping to prevent overfitting. Arguably its greatest benefit is the reduced likelihood of creating a vanishing gradient [46].

With the inner product of the input nodes weight matrix calculated, the baises added, and then the activation function applied to each node, we arrive at the final final vector

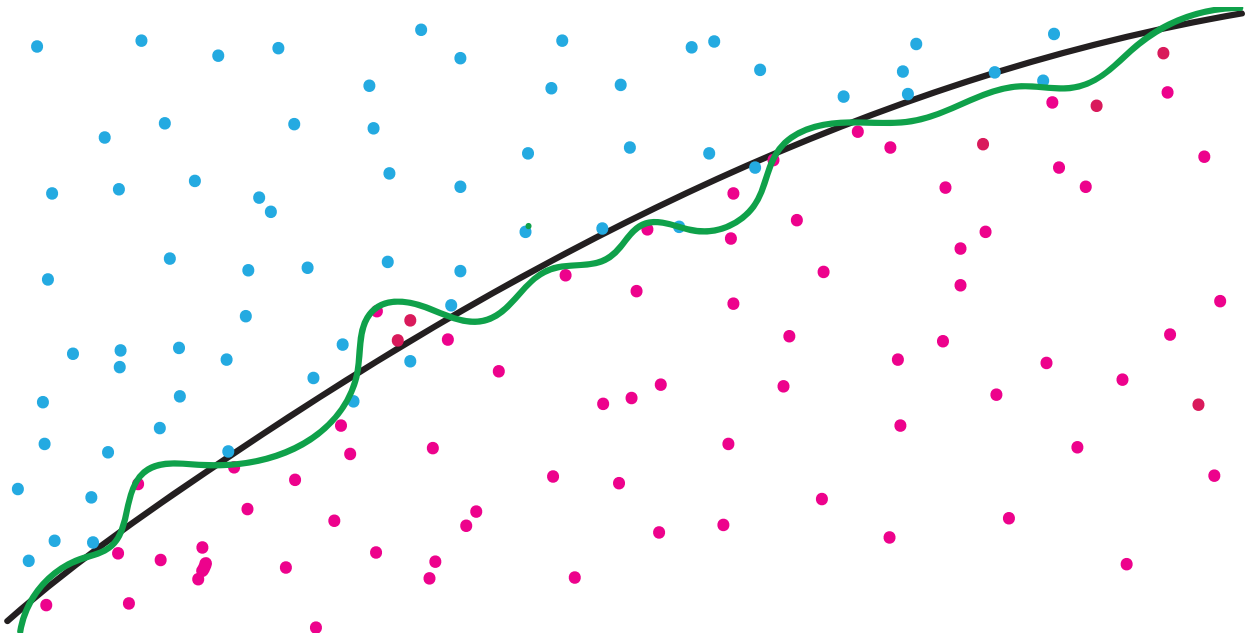


Figure 3.2: The green curve represents a model overfitting a binary classification problem. Although it makes near-perfect predictions for the training data in this figure, the model will not generalize as well as the simpler black curve when it tries to predict new, unseen data. Introducing biases and dropout layers in neural networks are strategies to prevent overfitting to reduce the model variance and fit the data like the black curve.

for the first hidden layer: $h_\nu^{(1)}$. Mathematically, $h_\nu^{(1)} = \sigma\left(x_\mu W_\nu^\mu + b_\nu\right)$. To calculate the next layer, the process is now repeated—only $h_\nu^{(1)}$ is used instead of the input layer, and the output $\hat{y} = \sigma\left(h_\nu^{(1)} W_\kappa^\nu + b_\kappa\right)$ is the final output of the neural network. The equations for each step as well as the dimensionality of each layer can be found in Figure 3.3.

3.1.2 Loss Metrics and Regularization

In order to update the network parameters, it is necessary to evaluate the quality of every prediction the neural network makes. The measure of error for prediction is referred to as the *loss*, whereas the summed total of all the losses is called the *cost*. For regression problems, the two most common cost functions are the mean squared error and the mean absolute error [47]. Without regularization, they are defined as:

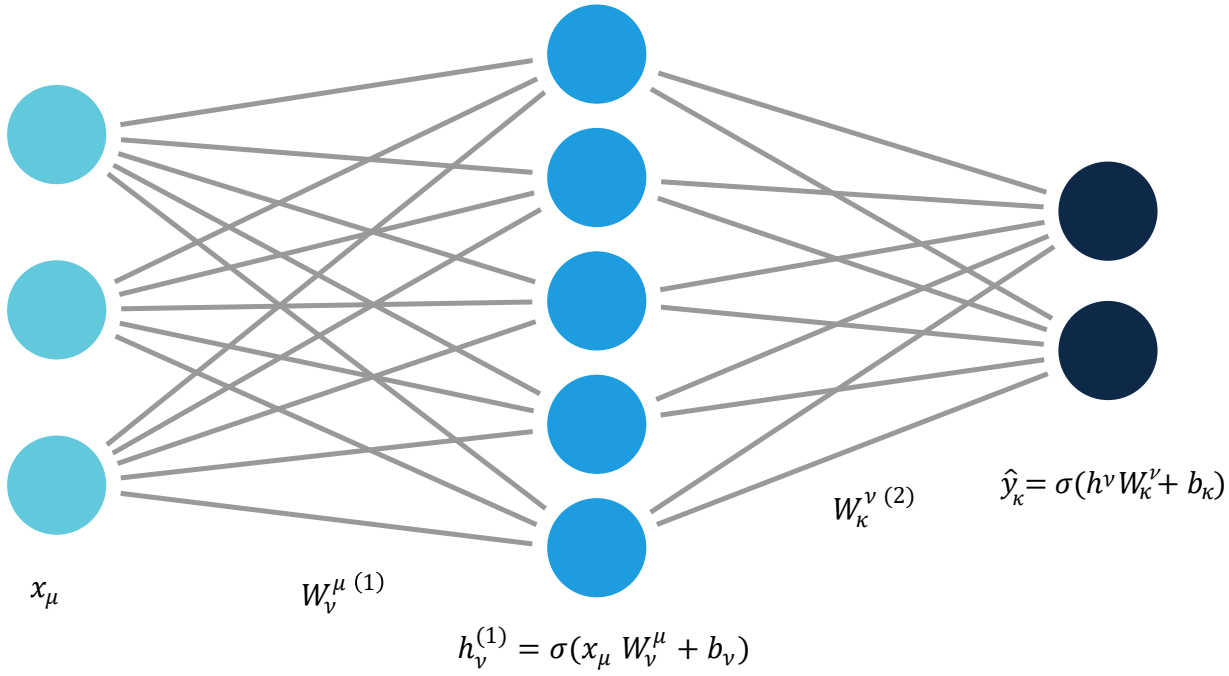


Figure 3.3: This diagram of a fully connected (affine) neural network has a single hidden layer and two output nodes. The tensors for each hidden layer are written in Einstein notation. The implicitly summed over greek letters and dimensionality are written below the tensors for clarification.

$$\text{MSE} = \frac{1}{n}\sum_i (\hat{y}_i - y_i)^2 \tag{3.4}$$

$$\text{MAE} = \frac{1}{n}\sum_i |\hat{y}_i - y_i| \tag{3.5}$$

where n is the number of training samples. Either cost metric can be regulated. The two most common regularizations are L1 (LASSO) and L2 (Ridge). Applied to the MSE, equation (3.4) with regularization becomes:

$$\text{L1 MSE: } J = \frac{1}{n}\sum_i (\hat{y}_i - y_i)^2 + \lambda \sum_j |W_j| \tag{3.6}$$

$$\text{L2 MSE: } J = \frac{1}{n}\sum_i (\hat{y}_i - y_i)^2 + \lambda \sum_j (W_j)^2 \tag{3.7}$$

where W_j is the j^{th} weight for training sample i, and λ is the regularization hyper-parameter.

One simple equation[2] for updating the weights for loss L is as follows:

$$W_j := W_j - \alpha \frac{\partial L}{\partial W_j} \tag{3.8}$$

where α is the learning rate. L2 regularization is often referred to as weight decay. Every iteration, the weights are pushed closer to zero due to the multiplication of the weights by a value < 1. L1 is known as LASSO (least absolute shrinkage and selection operator) because it shrinks the less important features' coefficients to zero. This is because for small values, $|W_i|$ is a much stiffer penalty than $(W_i)^2$. Thus, L1 is a good choice when there are dozens of features [45].

Including biases in the loss function is not the only way to regularize a model. Dropout layers are an entirely different method of regularization used exclusively for neural networks [48]. The idea is to introduce a hidden layer with a probability "dropping out," i.e. ignored. Large weights in a neural network are indicative of a high-variance network, likely to be overfitting the data. By introducing layers with a probability of dropping out, the inward connections to the next layer change stochastically from batch to batch. The result of this behavior has the effect of adding noise to the network, similar to the inclusion of biases [49] [50].

3.1.3 Backpropagation

Backpropagation is the process of updating all the trainable parameters of the machine learning model, including weights, biases, and any other trainable parameters. The partial derivative requires repeated use of the chain rule. The output of the neural network in Figure 3.3 can be written as a functional:

$$\hat{y} = \sigma\left(h_\nu^{(1)}\left(x_\nu\right)\right) \tag{3.9}$$

[2]This is a simple version of stochastic gradient descent, which will be discussed in depth in section 3.2

Here (3.9), the output layer $\hat{y}$ is a function of the hidden layer $h_\nu^{(1)}$, which in turn is a function of the input layer x_μ. Consider the MSE cost without regularization:

$$J = \text{MSE} = \frac{1}{n}\sum_i (\hat{y}_i - y_i)^2 \tag{3.10}$$

Note that J is really $J(\hat{y})$, meaning that it is a function of the output functional (3.9). For well-behaved functions, such as (3.10), the derivative of a summation is equal to the summation of the derivatives of each term. To see how much to shift the weights, calculate the gradients for each layer. The first partial derivative is trivial:

$$\frac{\partial J}{\partial J} = 1 \tag{3.11}$$

and the next partial derivative is also straightforward[3]:

$$\frac{\partial J}{\partial \hat{y}} = \frac{2}{n}\sum_i (\hat{y}_i - y_i) \tag{3.12}$$

Applying the chain rule yields:

$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial J}{\partial J}\frac{\partial J}{\partial \hat{y}} = 1 \cdot \frac{2}{n}\sum_i (\hat{y}_i - y_i) \tag{3.13}$$

The next required term in the chain is the derivative of the loss with respect to the sigmoid:

$$\frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \sigma} \tag{3.14}$$

We already found the first term (3.13), and the second term is trivial.

$$\frac{\partial \hat{y}}{\partial \sigma} = 1 \implies \frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \sigma} = \frac{2}{n}\sum_i (\hat{y}_i - y_i) \cdot 1 \tag{3.15}$$

At this point, we have the derivative $(\partial J/\partial\sigma)$ for the σ in the final layer $\hat{y} = \sigma\left(h_\nu W_\kappa^\nu + b_\kappa\right)$. The next derivative in the chain will be $(\partial J/\partial g)$ where $g = h_\nu W_\kappa^\nu + b_\kappa$. Continuing the chain,

[3]The astute may notice that if we instead chose MAE, we encounter a problem taking the derivative when $\hat{y} = y$. Usually zero is returned instead or the MAE is approximated with a differentiable function. Otherwise using MAE is straightforward: $\dfrac{\partial J_{\text{MAE}}}{\partial \hat{y}} = \begin{cases} +1, & \hat{y} > y \\ -1, & \hat{y} < y \end{cases}$

$$\frac{\partial J}{\partial g} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial g} \tag{3.16}$$

where

$$\sigma(g) = \frac{1}{1 + e^{-g}} \tag{3.17}$$

$$\implies \frac{\partial \sigma}{\partial g} = \sigma(g)(1 - \sigma(g)) \tag{3.18}$$

Combining these previously calculated terms yields:

$$\frac{\partial J}{\partial g} = \frac{2}{n} \left(\sum_i (\hat{y}_i - y_i) \right) \cdot 1 \cdot \sigma(z)(1 - \sigma(z)) \tag{3.19}$$

Now comes the good part. Recall that the trainable parameters in the network are the weights W and biases b. The next step is to calculate the gradients with respect to each of these parameters. This, in turn, will be used to update the parameter.

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial g} \frac{\partial g}{\partial W} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial g} \frac{\partial g}{\partial W} \tag{3.20}$$

The last partial derivative in the chain is

$$\frac{\partial g}{\partial W} = W \tag{3.21}$$

So,

$$\frac{\partial J}{\partial W} = \frac{2}{n} \left(\sum_i (\hat{y}_i - y_i) \right) \cdot 1 \cdot \sigma(z)(1 - \sigma(g)) \cdot W \tag{3.22}$$

For the baises,

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial g} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial g} \frac{\partial g}{\partial b} \tag{3.23}$$

$$\frac{\partial g}{\partial b} = 1 \tag{3.24}$$

$$\implies \frac{\partial J}{\partial b} = \frac{2}{n} \left(\sum_i (\hat{y}_i - y_i) \right) \cdot 1 \cdot \sigma(g)(1 - \sigma(g)) \cdot 1 \tag{3.25}$$

Note that W is actually $W_\kappa^{\nu(2)}$, a matrix of weights, and b is actually b_κ, a row vector of biases. Thus, the above equation (3.23) is just the partial derivative for one term in the weight matrix or bias covector. Repeating the process for each term in the matrix W and vector b yields the gradients $\boldsymbol{\nabla}_w L$ and $\boldsymbol{\nabla}_B L$, which represent the gradient of the loss function with respect to the weights and biases, respectively. This is the origin of the term, "gradient descent," an optimization algorithm discussed in the next section. This was just the process to calculate the gradients need to update weights for the final layer, but one can see how continuing the process of chaining partial derivatives will yield the gradients for earlier layers in the network.

3.2 Optimizers

Having calculated all the gradients via backpropagation, the weights and biases of the network can now be adjusted. The general idea of gradient descent relies on the fact that the gradient of any function points in the direction of the steepest increase. Thus, to optimize the network—which is equivalent to finding the parameters that minimize the value of the loss function—the weights and biases are updated by shifting their values in the opposite direction of the gradient of the loss function with respect to the weights, $\boldsymbol{\nabla}_w L$. Gradient descent is the core principle of machine learning; this efficient algorithm for systematically updating model parameters made it possible to develop deep neural networks and train them with large datasets. Numerous improvements have been made to gradient descent since its inception [51], AdamW [52] being the current state-of-the-art. In this section, we introduce several optimization algorithms to provide context for Adam, the optimizer used for training our model. In the previous section (3.1.3), the gradients of the weights W and biases b were written explicitly. For simplicity, the variable θ is introduced to refer to either parameter. As before, $J(\theta)$ is the cost function; it could be the MAE, MSE, or any other differentiable measurement of fit quality.

Gradient Descent

Vanilla gradient descent [53] updates the parameters in the following way:

$$\theta := \theta - \eta \cdot \boldsymbol{\nabla}_\theta J(\theta) \tag{3.26}$$

Here (3.26), η is a *hyperparameter* known as the learning rate. A hyperparameter is a user-defined parameter that must be chosen before the training process begins; it is not

a trainable parameter. Hyperparameters can be "tuned" by repeating the entire training process with various hyperparameters set. Typically, one trains the model with a variety of hyperparameters over a short number of epochs. Once satisfied, the number of epochs is increased, and the training is repeated with the best-found hyperparameters. One limitation to gradient descent is the need for the entire cost J to be calculated. For large datasets, this can become impractical. Two common variants are batch gradient descent and stochastic gradient descent (SGD). In the former, the training set is divided into batches, and the gradient is updated after each batch. One iteration through all the batches is called an *epoch*. In the latter, the gradient is calculated using the loss function instead of the cost function—i.e. the gradient is calculated, and the parameters are updated after each training sample. Both methods greatly reduced training time with the help of optimized, parallel computing [54]. By updating the parameters after every training sample, SGD will move in the direction of the true gradient. The major limit to these methods, however, is the fixed learning rate [55]. If the learning rate η is too large, the algorithm will be unstable and "bounce" around the global minimum of the cost function. If η is too small, the algorithm will, at best, take a long time to train, and at worst, end up stuck in a local minimum.

Stochastic Gradient Descent with Momentum

Compared to regular SGD, stochastic gradient descent with momentum [56] can greatly reduce the time to convergence. The general idea is to add a fraction of the previous parameter update to the current update. The exponential moving average (EMA) is an averaging of points within a period that puts greater weight on more recent points[4]. Here, S_t is the t^{th} value in the sequence S, and V_t is the t^{th} value in the new exponential moving averaged sequence, V .

$$V_t = \beta V_{t-1} + (1 - \beta)S_t \tag{3.27}$$

where $\beta \epsilon [0, 1]$, a hyperparameter which partly defines how much weight the previous $1/(1-\beta)$ terms of S contribute[5]. EMA's are common in market forecasting, so often S_t is the price at time t. The continuous update for SGD with momentum is as follows:

[4]In contrast a simple moving average treats each point as equally significant

[5]Typically 0.90 is a good starting point

$$V_t = \beta V_{t-1} + (1 - \beta) \nabla_\theta J(\theta) \tag{3.28}$$

$$w = W - \alpha V_t \tag{3.29}$$

Here, α is the learning rate, as always. To be clear, $\nabla_w L$ is the gradient of the loss function with respect to the weights. Note that the cost function $J(\theta)$ may instead be the loss function $L(\theta)$ if updates are performed after each training sample (i.e. a batch size of one). SGD with momentum tends to perform better than SGD because it gives a closer estimate of the full gradient from the batch than SGD. Additionally, the momentum helps push the update through ravine-shaped local minima in the correct direction, whereas SGD tends to oscillate back and forth along the ravine's steeper dimension [57].

Root Mean Squared Propagation

Root Mean Squared Propagation (RMSprop)[6] is another variant of SGD designed to improve convergence speed and remedy Adagrad's [58] tendency to rapidly diminishing gradients [59][60]. The idea is to dampen oscillations in directions when the predictions are close to the cost function's minimum and accelerate movement when far away. In RMSprop, we keep a moving average of the squared gradients for each weight and use these to divide the learning rate by an exponentially decaying average. As before, $\nabla_\theta J$ is the gradient of the cost with respect to weights.

$$S_{k+1} = \beta S_k + (1 - \beta)(\nabla_\theta J \cdot \nabla_\theta J) \tag{3.30}$$

$$\theta_{k+1} = \theta_k - \alpha \frac{\nabla_\theta J}{\sqrt{S_{k+1}} + \epsilon} \tag{3.31}$$

The hyperparameter ϵ is included in the denominator to prevent a possible division by zero as well as provide more stability.[7]

[6] RMSprop has an interesting history. It is an unpublished algorithm, first introduced by Geoff Hinton in an online series of lectures. Nevertheless, it is an incredibly popular algorithm and included in most ML platforms.

[7] Typical values for α and β are 0.001 and 0.9, respectively.

Adaptive Moment Estimator

Adaptive Moment Estimator (Adam) is a combination of RMSprop and SGD with momentum. Adam uses the squared gradients to scale the learning rate for each parameter (similar to RMS prop), and it uses a moving average of the gradient (similar to SGD with momentum).

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)(\boldsymbol{\nabla}_\theta J) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\boldsymbol{\nabla}_\theta J \cdot \boldsymbol{\nabla}_\theta J) \end{aligned} \tag{3.32}$$

The new parameters in this algorithm, m_{t-1} and v_{t-1} are the first and second moments of the gradient (the mean and variance), respectively. Adam's 80,000 citations in the six years since its publication gives some indication of the importance and power of this algorithm [61]. This was the chosen algorithm for training our neural network for predicting disorder in XANES.

3.3 Normalization

Normalization is an important step for any machine learning algorithm. There are three types of normalization utilized in our training process: feature normalization, label normalization, and batch normalization. Without feature normalization, a model will put greater weight on features with larger values. For example, if a model is trained to predict surface stress of a silica bead in silicone gel given the diameter of the bead in microns and the adhesion energy in mNm^{-1}, the model would learn to ignore the bead's size in its predictions. This is because the particle's size is $1000\times$ smaller than the adhesion energy [62]. To correct this scaling issue each feature is "normalized" on the training data to be on the same scale. Often a z-score is used to center the features around a mean of zero with a standard deviation of one. This is known as standardizing or applying a standard-scalar [63].

$$Z_{norm}^{(i)} = \frac{x_i - \mu}{\sqrt{\sigma^2 - \epsilon}} \tag{3.33}$$

In our neural network, the training features are normalized in this way.

For the same reasons feature normalization is important, the training labels must also be normalized. Instead of using the standard scalar, the labels are normalized using min-max normalization, which normalizes the values between zero and one. This is useful when the

labels are known to be evenly distributed over a range. To scale the i^{th} label (y) via min-max scaling:

$$Z^{(i)}_{norm} = \frac{x_i - \text{Min}(y)}{\text{Max}(y) - \text{Min}(y)} \tag{3.34}$$

Scaling the training features means the neural network will predict the scaled values. The prediction can be "un-scaled" to retrieve the real, predicted values via:

$$y^{(i)} = -\frac{Z^{(i)}_{norm}\,(\text{Max}(y) - \text{Min}(y))}{\text{Min(y)}} \tag{3.35}$$

It is important that the scaling parameters μ, σ, $\text{Min}(y)$ and $\text{Max}(y)$ come from the training set—not the testing or validation set. Using the values from the entirety of data constitutes a form of *data leakage*, where the training process is given a hint of the validation or test data. It is important that the model never sees the testing or validation until testing or validation time; otherwise, the model is unlikely to generalize as well to unseen data as one might expect given a loss curve.

Batch normalization is a technique designed to make NN's more robust to internal covariate shift [64]. Covariate shift refers to a systematic difference between the training and validation data, or in the context of a training batch, a systematic shift in the distribution of data from batch to another [65]. For example, if a NN is trained for binary classification to predict whether or not an image includes a cat—and the network is trained on images of only black cats—the network is unlikely to make a correct prediction when it encounters an image of an orange cat.

The idea of batch normalization is to normalize each hidden layer similar to how training data or labels are normalized; however, whereas normalizing the training data centers the dataset or labels using fixed parameters such as the mean and variance, in batch normalization the mean and variance of the batch normalization are learnable parameters. In batch normalization, the values for a hidden layer are scaled via:

$$\widetilde{Z}_i = \gamma Z^{(i)}_{norm} - \beta \tag{3.36}$$

Notice that if $\sqrt{\sigma^2 + \epsilon}$ and $\gamma = \mu$, we get equation (3.33), i.e. the hidden layer is normalized in the same way as the input layer. This is generally not useful, however, because normalizing all parameters to be centered around zero causes the sigmoid-like activation functions to be mostly focused on the linear regime.

The output of the batch normalization is passed forward to the next hidden layer of

the network, while the normalized input is retained in the current layer. Normalizing the hidden layers for each batch means that later hidden layers do not have to adapt as much to the earlier hidden layers. Consequently, this allows the deeper layers to do a better job tuning themselves a little more independently of the other layers, improving performance and speeding up the learning process. Note, because each batch is scaled ($z \rightarrow \widetilde{z}$), a small amount of noise is added, which acts as a slight form of regularization[8]. Recently, several papers [66] [67] [68] have been published disputing the reason batch normalization improves the model performance; none, however, dispute its efficacy. Data augmentation may be vital in building the neural network trained on simulation data to predict experimental data, for which there is a sparsity of data for training.

3.4 Data Sparsity

Unfortunately, not all project goals include a plethora of diverse training data. This section introduces two techniques for dealing with training data: data augmentation and transfer learning. Data augmentation can be a valuable technique even with an abundance of training data. Transfer learning, on the other hand, aims to solve a complex problem with little data by first training the model on an easier problem with ample data.

3.4.1 Data Augmentation

Data augmentation is a technique for expanding the size and variance of the training data for machine learning purposes. It has been critically important for developing powerful deep neural networks, particularly in the domain of image processing [69][70] [71] [72]. Consider the example in section 3.3 of a cat-vs-not-cat binary classifier. The idea of data augmentation is to take the dataset containing only images of the black cats and add new images created from the original dataset. Common methods of data augmentation are image cropping, image rotation, and introducing image filters that alter the color, sharpness, or contrast. In the case of the black-cats-only dataset, color filtering may train the network to become color agnostic and correctly classify an image of an orange cat without ever having seen one. For signal processing, including absorption spectroscopy, common methods for expanding the training data size include the introduction of Gaussian noise and shifting the spectra horizontally [73]. By artificially expanding the size and complexity of training data, data

[8]Note, while batch-norm adds regularization, this is not intended to be used as a form of regularization. L1, L2 regularization or dropout layers should be used instead.

augmentation helps prevent models from overfitting.

3.4.2 Transfer Learning

Transfer learning was first introduced in 1976 to [74] [75] [76]. The idea is to alter a model trained on one task to be able to solve a new but similar task. One famous example involves a neural network originally trained to classify pastries, which, utilizing transfer learning, was re-purposed for detecting cancer cells [77]. Consider a model first trained on dataset A with the final goal of predicting dataset B. Note that A and B in this example are not a train-test split but, instead, inherently different problems. By pre-training the model on A to solve a similar task with B, the model learns inductive biases which encourage the model's parameters θ_B to be similar to θ_A. This may have the effect of training the model to learn low-level features that may not have been learned from B alone [78]. Transfer learning using simulations as training data is a cutting-edge topic of research in ML and particularly in AI. Modern autonomous driving systems rely on testing and training their models using driving simulations to bolster their practice time beyond what would be possible from real-world driving tests alone [79]. A significant time has been invested in creating platforms specifically for developing autonomous driving systems [80].

Applying transfer learning to incredibly sparse datasets—attempting to teach the model from just a few examples—is called few-shot learning[9] [83]. This is a very hot area of modern research, as there are many instances in which real-world data is incredibly expensive to acquire, but simulating or obtaining similar data is possible. In the context of this thesis, transfer learning will be an important tool for creating a neural network capable of predicting disorder from an experimental XANES spectrum. Because of the sparsity of experimental data, it is unfeasible to train the neural network on experimental data alone. Instead, we rely on simulated XANES spectra created with FEFF. The systematic differences between the simulated XANES spectra and the experimental counterparts, however, suggest a neural network trained purely off simulation spectra will not perform well when it encounters an experimental spectrum for the first time. Applying the principles of transfer learning: the neural network can first be trained with the simulated XANES spectra, for which there are ample examples. Then, using the limited number of experimental data—included extra examples created via data augmentation—the network can be trained again via transfer learning.

[9]Other common terms are "one-shot" [81] and "zero-shot" [82] learning. These both refer to the same concept but with only one or even zero training examples, respectively

3.5 Covolutional Neural Networks

Convolution is a mathematical operation for combining two functions, the result of which is a third function revealing the effect of the second function on the first [84]. The convolution of two continuous functions, f and g, is a special type of integral transformation. The result is the integral of the product of f and the shifted inverse of g, where f can be thought of as the input function and g is often referred to as the kernel.

$$f \otimes g = \int_{-\infty}^{\infty} f(j)g(i-j)\,dj \tag{3.37}$$

The variable i (no relation to the imaginary number) is represents the weighted-shift in the function $g(\tau)$. Different values of i emphasize different parts of the other function, $f(\tau)$.

In computer science, convolutions are an important and powerful tool for signal and image processing [85] [86]. Because images and signals—which can be thought of as a 1D image—are comprised of a discrete number of points (e.g. pixels), a modified formula is required to perform the convolution. The convolution of the signal f with the kernel g can be written as [87]:

$$f \otimes g = \sum_{j=1}^{m} g(j) \cdot f(i-j+m/2) \tag{3.38}$$

Here (3.38), m is the length of the kernel g, and i and j are hyperparameters. To demonstrate visually, consider a simple, example absorption spectrum with only 10 data points (3.4).

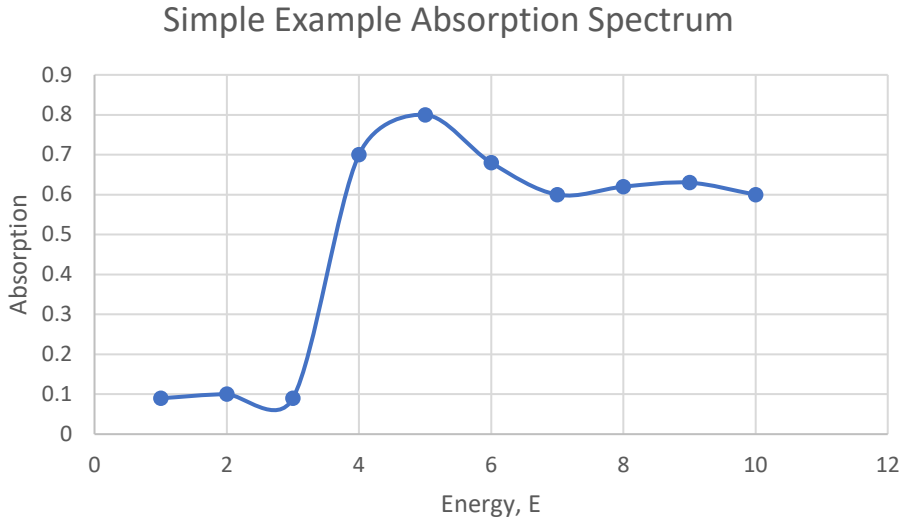


Figure 3.4: A simple abosrption spectrum for demonstration purposes.

Each data point, (E, μ) in the spectrum is described as a feature vector, where the feature is the energy value for a given point (E, μ). The vector, f, is depicted below with boxes to represent each element in the vector. Zeros are padded on both sides for reasons that will become clear soon.

$f =$

0	.09	.10	.09	.70	.80	.68	.60	.62	.63	.60	0

Additionally, consider the kernel, g

$g =$

.1	.1	.1

The convolution works by multiplying each element in the input vector f by the corresponding element in the kernel g and summing the results. The kernel then moves to be centered around the next element in f. One way to think about this process is a kernel or filter sliding over an input signal. Applying the kernel g onto the first index of f yields:

0	.09	.10	.09	.70	.80	.68	.60	.62	.63	.60	0
.1	.1	.1									

$$h(1) = (0)(.1) + (.09)(.1) + (.10)(.1) = 0.019$$

where $h(1)$ is 1st index of the resulting vector. For the next point, the kernel shifts to be centered around it.

0	.09	.10	.09	.70	.80	.68	.60	.62	.63	.60	0
	.1	.1	.1								

$$h(2) = (.09)(.1) + (.10)(.1) + (.09)(.1) = 0.028$$

The final resulting vector is:

$h =$

.019	.028	.089	.159	.218	.208	.190	.185	.185	.123

The toy example was chosen to demonstrate the basics of a 1D convolution. In this example, the input spectrum was a vector of length ten and the kernel of length three. In the context of applying a 1D convolution to a neural network, the size of the input vector is the cardinality of the hidden layer directly preceding the convolutional layer. Often the hidden layer's output, represented as a vector, is reshaped into an n-dimensional tensor before

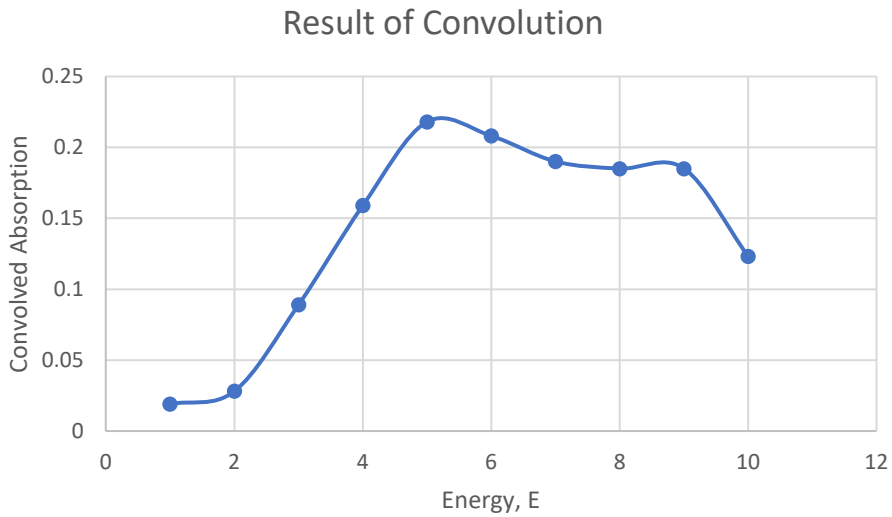


Figure 3.5: The result of the 1D convolution of kernel $g = (.1, .1, .1)$ on the spectrum in Figure 3.4 is plotted above.

applying the convolution. To apply a 1D-convolution to a Tensor of rank n, simply apply the convolution to each of the n-vectors separately, ensuring to pad the ends of each vector with zeros. Layers with dimensionality greater than one can be "un-raveled". Alternatively, the layers of each dimension can be truncated or "pooled" by averaging the layers that are stacked on one another (or taking the max of each layer) until the desired dimensionality is achieved. A common way to achieve this is with a max-pooling layer or an average pooling layer. Max pooling and average pooling layers also have the additional benefit of downsampling the feature space, tending to make the network more robust to slight variations in the position of features in the input image or signal. This is referred to as "local translation invariance" [88].

Another important possible change in the above example is the stride length. In the above example, we "slid" the kernel across the input spectrum one point at a time. This is referred to as a stride size or stride length of one. The stride could be any integer less than the length of the input vector f, though in practice, typically stride lengths tend to be one or close to one. Lastly, in the above example, we only applied one convolutional kernel, or "filter." In practice, many filters are applied sequentially. In the above example, the filter $g = (.1, .1, .1)$ was simply decided upon *a priori*. In practice, the values for each filter in the convolutional layer are initialized randomly or according to the specified initialization function. The default in Keras is Glorot Uniform. The values of each filter are trainable parameters that evolve to produce the best final prediction given the subsequent layers in

the neural network.

3.6 How to Train a Neural Network

Building a successful neural network requires a combination of intuition and procedural know-how. The first key is to start with a simple model, perhaps a single hidden. *If you start with a complex model including data augmentation and regularization, you will never be able to tune the hyperparameters and find a good solution.* A good strategy is to pick a simple architecture and reasonable hyperparameters and train the model on a small subset of training samples, say 1–10 samples. Then, train the model over ten or so epochs and see if the training cost decreases and whether you can overfit it. If you can overfit the small sample size, it means the code is working, and the network architecture makes sense. Now is the time to increase the number of samples in the training data, either to the full train-test split or a subset of if working with "big data."[10] Next, you can run a broad hyperparameter search. Afterward, run maybe 20 epochs and see how the training and validation loss is moving. If they both are going down, the architecture looks good. If not, start over. Ideally, the training and validation loss curves will be decreasing together like in figure 3.6. If the model still does not predict well after hyperparameter tuning, the model is underfitting, and a more complex architecture is required (add more layers).

Ideally, the training and validation loss will decrease together over many epochs. It is more likely, however, that after many epochs, the training loss will continue to decrease while the validation loss plateaus. This is the point to start introducing regularization such as dropout layers as well and considering data augmentation. These techniques will allow the model to continue decreasing the validation loss and prevent overfitting. The main goal of training is to minimize the validation loss. The placement of dropout layers and the type of data augmentation are subject to trial and error. Generally, it is best to include dropout layers after a ReLU activation and before an affine layer. There has been some research suggesting the inclusion of low-probability dropout layers after convolutional layers tends to improve model performance [49] [50], but these rules do not work for every network, and it is still worth trying many options while training.

[10]Big data has become a nebulous term, but a reasonable example is when there is so much data that the dataset cannot be loaded into the computer's memory (RAM) at the same time.

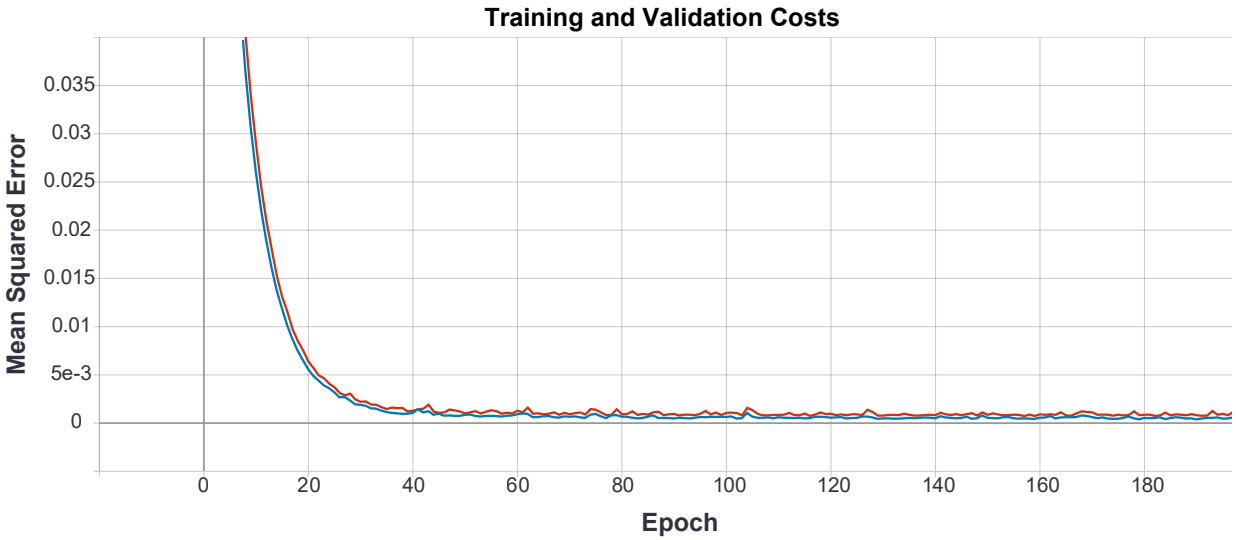


Figure 3.6: In this example loss curve, the x-axis is the epoch, and the y-axis is the mean squared error of the predictions. The red curve is the training data, and the blue curve is the validation set. Notice how both curves decrease together, but the training curve has a smaller error than the validation curve. This is the expected behavior of a model that is not overfitting and training properly.

Chapter 4

Results

The results of the training process are presented in this chapter. We begin by first outlining the training network architecture and training process on simulated XANES in section 4.1. Then, we discuss the work on expanding the model to make solid predictions on experimental data.

4.1 Training with Simulation Data

The 1000 simulated XANES spectra were first loaded into a Pandas dataframe [89] [90] of shape 1000×82. Each of the 82 columns represents a discrete energy value, and each row represents the absorption for a given spectrum at those energies. The dataset was split into training and testing groups according to an 80-20 random split, respectively. All absorption columns were then scaled via the standard scalar (3.33) and the training labels scaled via a min-max scaler (3.34). First, the model was trained to predict four descriptors: the mean squared displacement (MSD), the mean bond length distance, the standard deviation of the bond length distributions, and the skew of the bond length distribution. Note that the standard deviation is equal to the square root of the MSD. This feature was only included preliminarily in order to better understand the correlation in the network's predictions.

4.2 Experimental Data

Training a neural network entirely on simulation data and then making predictions on experimental data is unlikely to provide quality results. Using the trained network from Table (4.1) that predicted the test set values in Figure 4.1, we predicted the MSD values from two

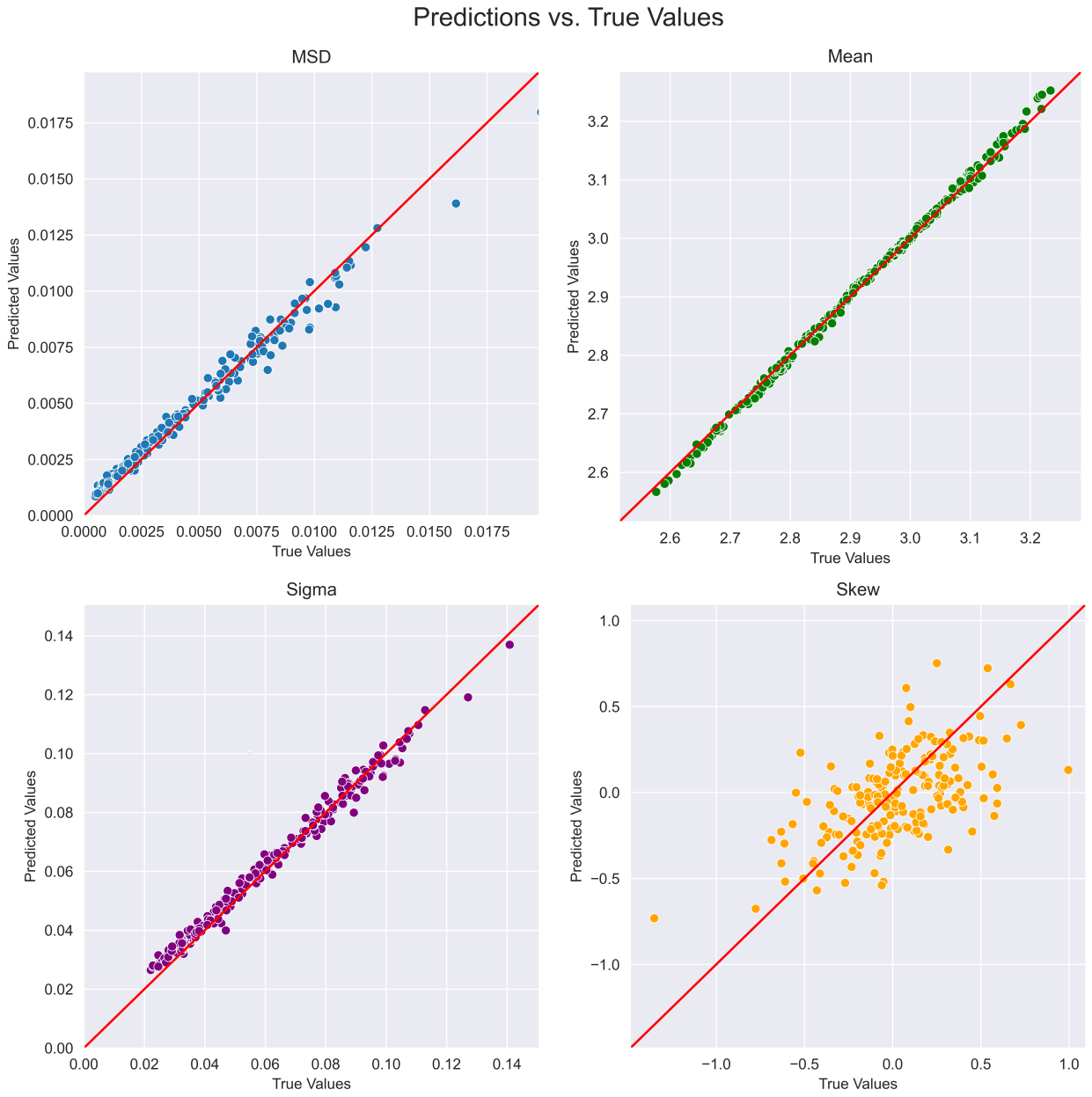


Figure 4.1: One configuration of the trained neural network includes four output nodes: MSD, Sigma, Mean, and Kurtosis. Sigma is the square root of the MSD and was included during training to confirm the patterns recognized by the network. Each point in each subplot represents a FEFF simulated spectrum in the test set. For each spectrum, the y-axis represents the MSD value predicted by the NN, and the x-axis represents the true MSD (label) for that spectrum. Hence, points on the $y = x$ red line are perfect predictions.

Layer Type	**Output Shape**	**# of Parameters**
Normalization	(None, 82)	165
Dense	(None, 128)	10,624
Dense	(None, 128)	16,512
Dense	(None, 512)	25,088
Reshape	(None, 8, 64)	0
1D-Convolution	(None, 6, 32)	6,176
Dropout	(None, 6, 32)	0
1D-Max-Pooling	(None, 3, 32)	0
Flatten	(None, 96)	0
Dense	(None, 128)	12,416
Dense	(None, 48)	6,192
Dense	(None, 512)	25,088
Flatten	(None, 512)	0
Dense	(None, 4)	513

Table 4.1: The model architecture for the network trained entirely on simulation data (which performed poorly on experimental data) relies primarily on affine (Dense) and convolutional layers. The model includes 108,966 total parameters, 108,801 of which are trainable.

experimental spectra. On the unpublished IMASC data, the network predicted an MSD of 0.0003724 Å^2 instead of the EXAFS equation fitted value of $\sigma^2 = 0.0102(8)$ Å^2. This poor prediction suggests the network considers the experimental spectrum to look most similar to the lowest disorder FEFF spectra; the model is not generalizing to understand the disorder encoded in the spectral shape.

4.2.1 Data Augmentation

While there is ample data for training and predicting on only simulation data, we only have two experimental spectra. In order to create more training and testing data for the neural network, two types of data augmentation were utilized: Gaussian noise inclusion and horizontal spectral shifting. While the motivation for utilizing data augmentation is to expand the size of the experimental training and testing set, the neural network must be

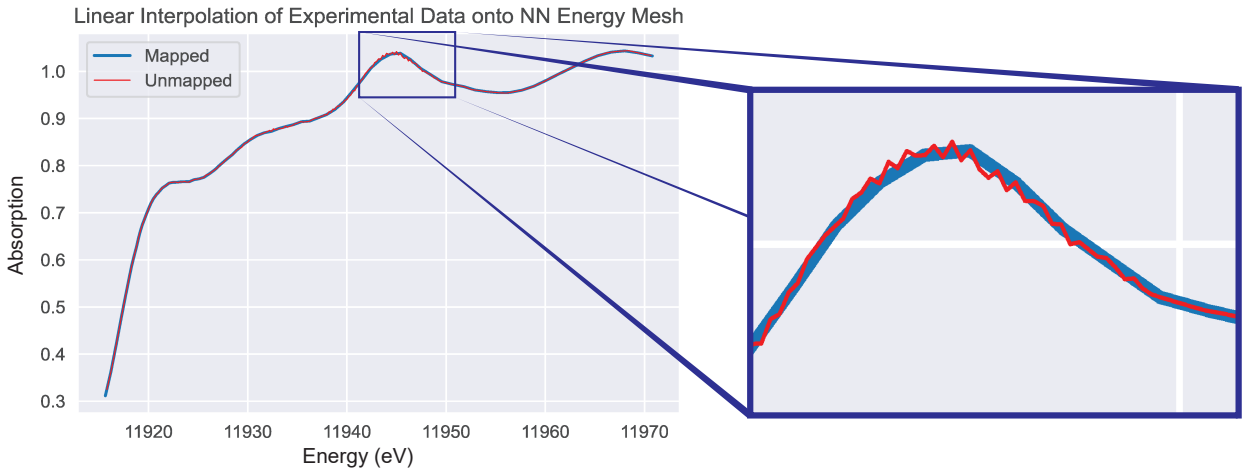


Figure 4.2: The experimental data is measured as a function of different energy values than those on which the neural network is trained. Consequently, the experimental spectrum must be mapped onto the proper energy mesh via linear interpolation.

trained to recognize the augmentation types prior to training or testing on the experimental data. As such, both the FEFF simulated dataset and experimental dataset are augmented.

Noise is artificially injected into the spectra by randomly shifting each absorption coefficient vertically. The shifted value for each energy level is selected from a Gaussian distribution with standard deviation $\sigma = 0.01$. An exaggerated example of the injected noise is shown in Figure 4.3.

The second form of data augmentation utilizes horizontal shifts. While this is common for signal processing and time series analysis, the inclusion here is more controversial. In XAFS, the edge location is dependent on the oxidation/reduction state of the species. Shifting the horizontal location is akin to shifting the species of the model; however, the neural network is not being tasked to determine the oxidation state of the sample. Instead, the model is merely tasked with predicting the mean squared displacement of the nanoparticle's bond lengths. The theory is that the disorder information is encoded throughout the entire XANES spectrum, not from just a simple feature such as the edge placement.

4.2.2 Transfer Learning

In building the transfer learning model, we opted to begin with a new network architecture, which can be found in Table (4.2). We hypothesized that applying convolutional layers

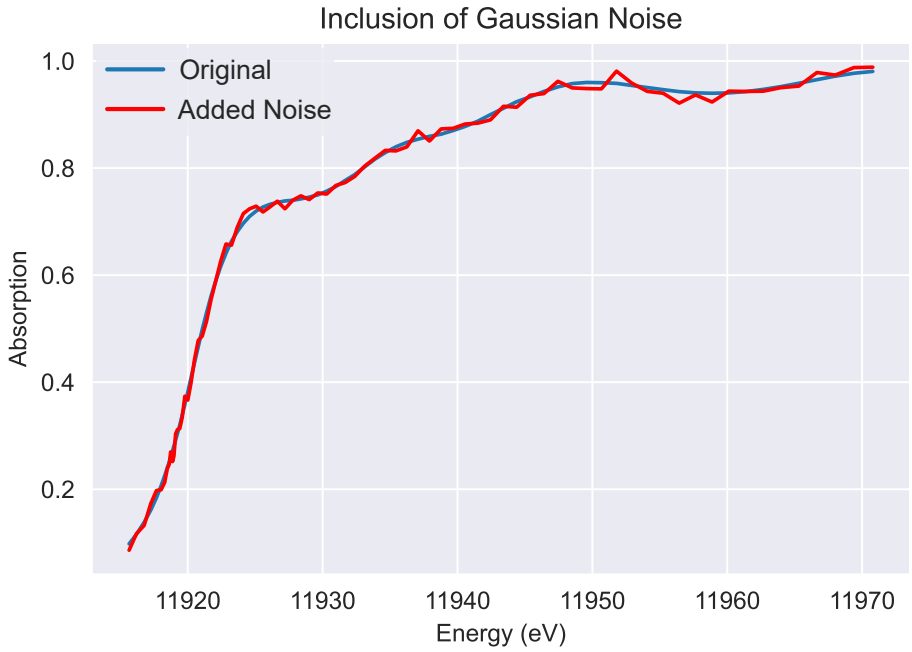


Figure 4.3: Gaussian noise is added to the spectra to increase the variance of the training data. This helps the network to learn the low-level features of the spectra and ignore artifacts not caused by the structural disorder. For demonstration purposes, the scale of the noise in this figure has been increased beyond what was used in training.

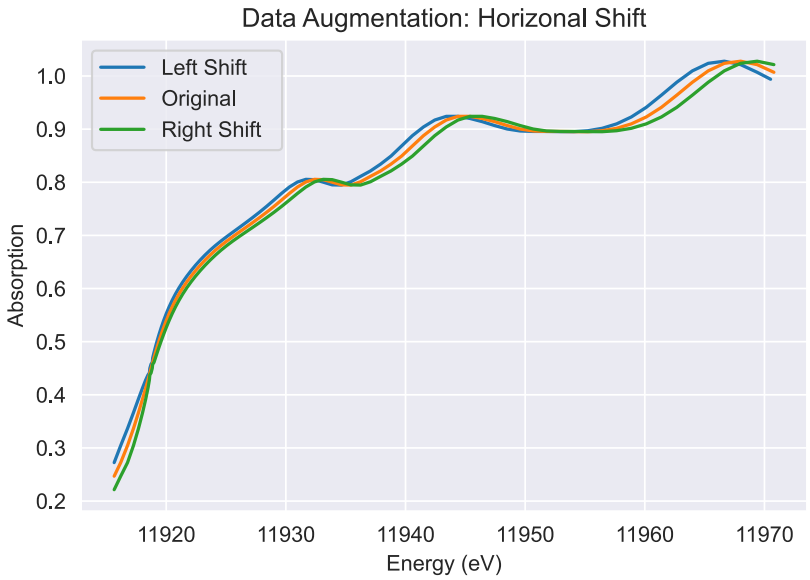


Figure 4.4: In order to train the network to predict disorder from the overall shape of the spectra—as opposed to fixating on the edge location—we introduce horizontal-shift as a data augmentation technique.

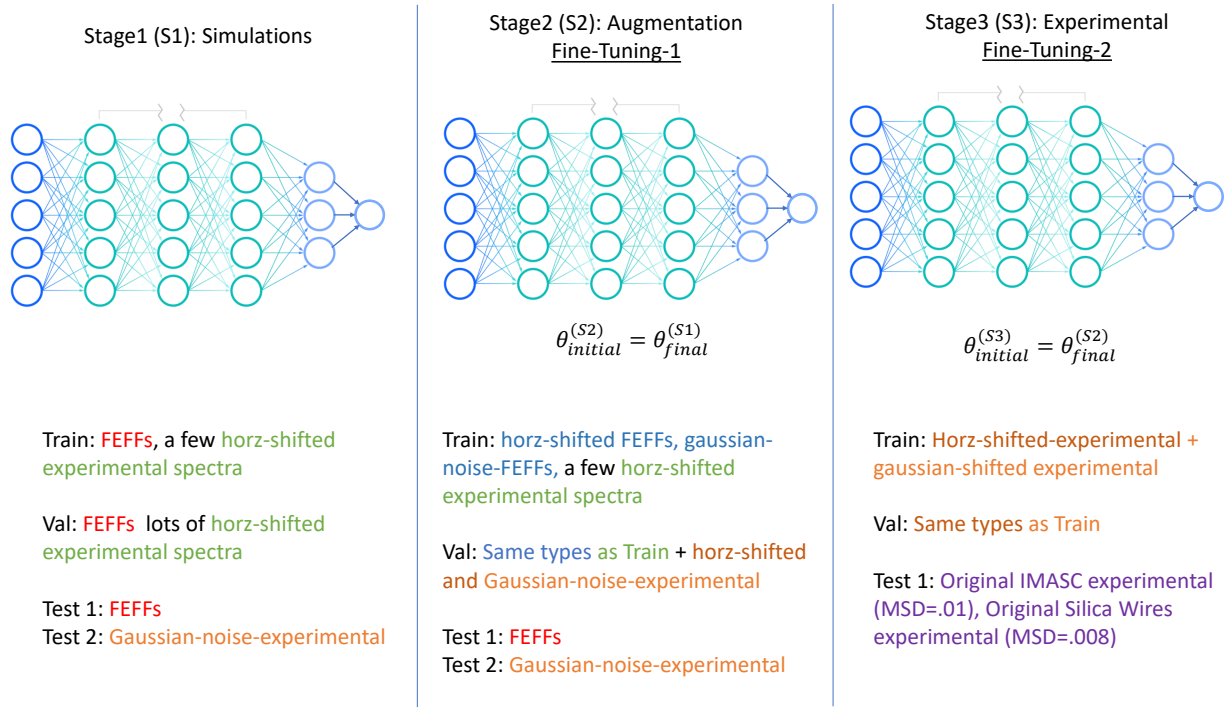


Figure 4.5: The allocation of training and testing data is summarized above. At each stage, the network architecture remains unchanged; however, some of the network's parameters are retrained on new data. The different sets of data are color-coded for readability. Note that the network is never trained on the original experimental data—it is only used for the final stage of testing.

before any affine layers may lead to a more consistent prediction between simulation and experimental spectra.

Our approach for applying transfer learning involves two stages of fine-tuning. First, we primarily train the model on FEFF simulated spectra, selecting an architecture and hyperparameters which are likely to be compatible with fine-tuning. We achieve this by weighting the validation set heavily (around 10%) with augmented experimental spectra and choosing a model which predicts unseen data-augmented experimental spectra as well as it predicts unseen simulated FEFF spectra. The training loss curves and selection process can be found in Figure 4.6. One concern with this approach is that we are injecting biases into our model selection; however, we take this into account through the utilization of a third, unseen test set and the fact that the model does not update its parameters based on its validation set predictions. When training a model in machine learning, the parameters are continuously updated until a minimum is reached in the loss function. While the loss landscape will have a global minimum, it also contains local minima, some of which will be more agnostic to the differences in experimental spectra and be better candidates for transfer

Name	Type	# Parameters	Output Shape
normalization_input	InputLayer	0	
normalization	Normalization	165	None, 82
reshape	Reshape	0	None, 82, 1
conv1	Conv1D	128	None, 82, 32
max_pooling1d	MaxPooling1D	0	None, 41, 32
conv2	Conv1D	3104	None, 39, 32
max_pooling1d_1	MaxPooling1D	0	None, 19, 32
flatten	Flatten	0	None, 608
dense1	Dense	58464	None, 96
dout	DropOut	0	None, 96
dense2	Dense	30264	None, 312
output	Dense	313	None, 1

Table 4.2: A new network architecture was constructed for the transfer learning process. The new network only has one output node, representing the MSD of the input spectrum.

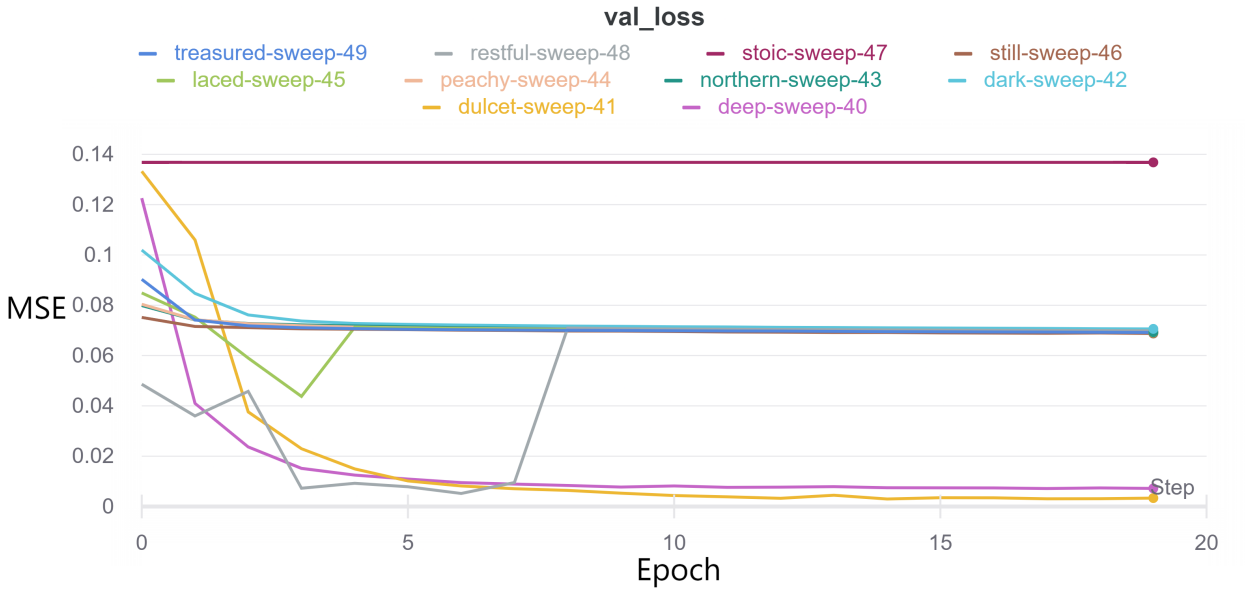


Figure 4.6: The validation cost (mean-squared-error) for 10 out of the 50 hyperparameter combinations searched in this sweep are plotted above. Most hyperparameters result in a loss curve stabilizing around 0.08, which is significantly higher than the training cost (not shown) that stabilized around 0.01. A few of the validation loss curves, instead, stabilized around 0.01 (dulcet-sweep-41 and deep-sweep-40). Because the validation set is heavily weighted with augmented experimental spectra, these two spectra are likely to be good candidates for transfer learning onto experimental data.

learning. In this first stage of learning, the intention is to teach the model to find broad predictive features from the simulation set while selecting a model at a local minimum that is likely to be a successful transfer-learning candidate.

The next stage of transfer learning seeks to teach the model to ignore noise and focus on the broader shape of the spectrum. We achieve this by freezing the early stages of the model and retraining the later parameters on a new dataset comprised primarily entirely of data-augmented spectra. Often, models are trained with the augmented dataset in the initial stage, which helps act as a form of regularization. In the case of transfer learning, however, applying this regularization so early on in the process may lead to undesirable local minima for which transfer learning onto the experimental dataset would be impossible. By applying an initial stage of fine-tuning to a well-tuned base model, we increase the likelihood of a successful second fine-tuning stage.

The last stage of the fine-tuning process is to freeze even more layers and reduce the learning rate, then train the model using all of the data-augmented experimental spectra. If the process is successful, the model will have learned to predict the MSD and ignore horizontal and Gaussian noise in the spectra from the first stage. In this way, we have

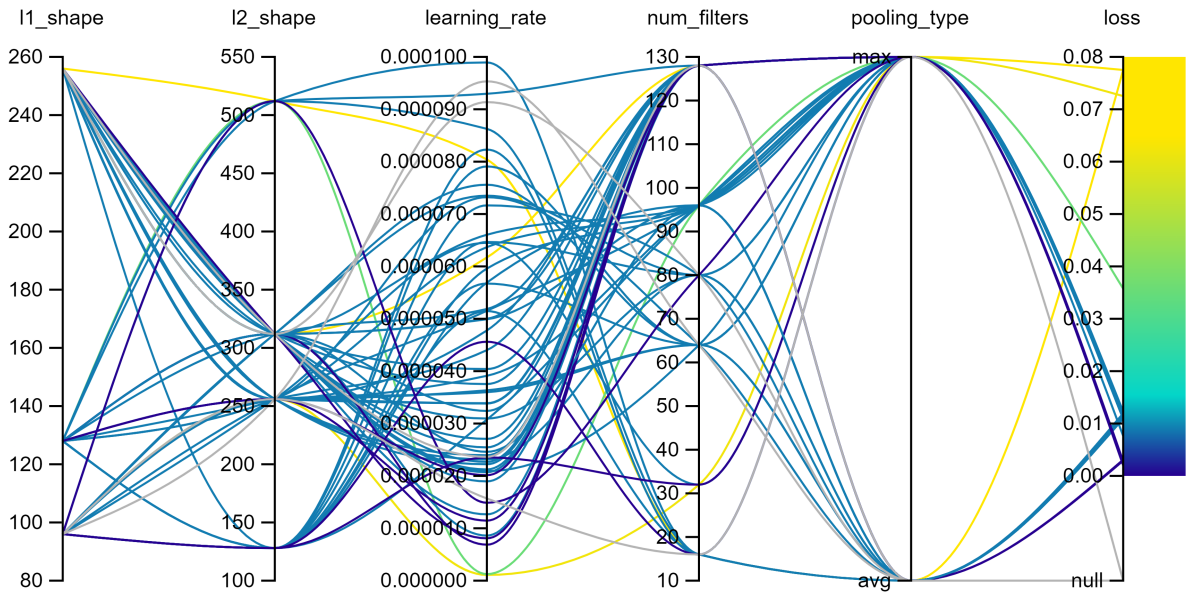


Figure 4.7: The hyperparameters are obtained through a "sweep," where the entire training process (20 epochs) is repeated with different hyperparameters each time. The hyperparameter training process and figure generation was conducted with the aid of [91].

increased the number of possible training samples to use for the final fine-tuning stage from one to many, allowing us to withhold both of the un-altered experimental spectra from the training process and use them to evaluate the success of the transfer learning process. A visualization and specific breakdown of which data is allocated into each stage of the training process can be found in Figure 4.5.

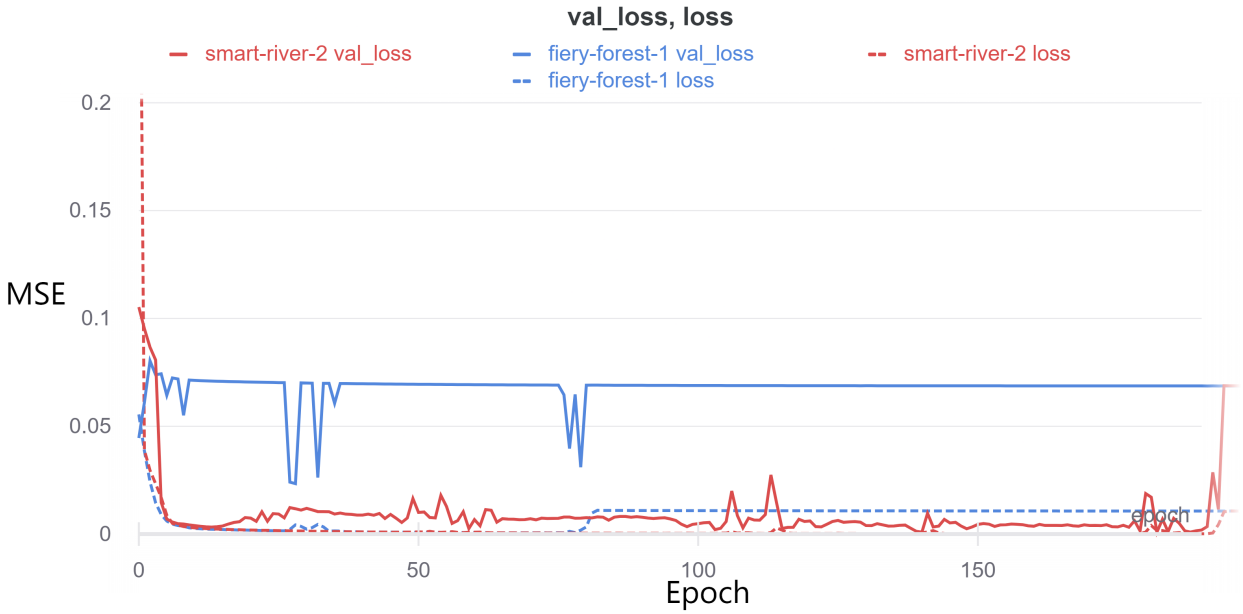


Figure 4.8: Both iterations of the training process were run with identical network architectures and hyperparameters; however, these two runs have significantly different validation losses. The discrepancy is due to the random weights initialization process. We mitigate this problem by setting global random seeds throughout the training process to select consistent pseudo-random values. The slight difference in initial parameters causes one iteration to become "stuck" in a different local minimum, which performed similarly with the training set but made substantially worse predictions on the validation set. The goal of the first stage of the transfer learning process is to identify hyperparameters that will lead to the best transfer learning candidates.

Chapter 5

Outlook

One-shot learning is arguably the most challenging problem in machine learning [76], and continued work is required to improve the model's ability to predict disorder in experimental spectra. While more time should be dedicated to tuning the hyperparameters and experimenting with different architectures, the approach taken in this thesis—as described in Chapter 4—is by no means the only technique to apply transfer learning. Recent work [92] [93] [94] on meta-learning, the process by which a model learns to solve many sub-problems and is optimized for learning new problems, offers a promising direction that may be pursued. Alternatively, there has been interesting work on few-shot learning for regression problems involving deep embeddings [95]

While current limitations in the quantity of experimental data stymie the efforts to complete a model for predicting the mean squared disorder of experimental bulk gold XANES spectra, we present reasonable evidence that bond-length disorder is encoded within the spectra. Further work on transfer learning, improved simulation quality, or a much larger corpus of experimental data may yield the desired predictive capabilities. A successful neural network capable of predicting bulk gold should, in theory, be able to learn to predict disorder from other structures via a similar transfer learning process as described in Chapter 4.

Even with successful predictions on the original experimental spectra, more evidence is required to test the network's predictive validity. This evidence should be in the form of predicting the MSD from new, experimental spectra. It is important that the model has never seen these spectra, nor a data-augmented version of these spectra. Even though the model is trained only on a simulation and data-augmented experimental spectra, there is a possibility that the network is biased beyond what we expect (through transfer learning model selection). Thus, correctly predicting the mean squared displacement from novel Au XANES

spectra would build confidence that our model has learned—rather than memorized—the encoded disorder within XANES spectra.

Appendix A

Select Python Code

The code for this thesis is well-documented and stored in a GitHub^{TM} repository. This appendix includes short descriptions of the major scripts written for this thesis, as well as the inclusion of a selected few major functions. The mathematical formulas and descriptions for how these scripts work are included in the main chapters of this thesis; however, there are instances where looking at the code can be useful in understanding the approach. Also included are the file structures that the scripts generate or expect to find. This is important to know if actually running the scripts.

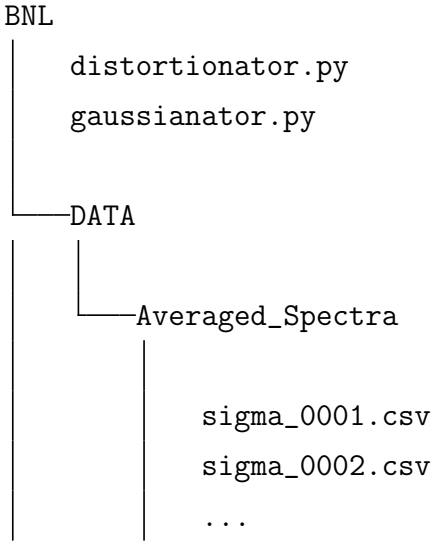
A.1 distortionator.py

Given `feff.inp` file, generate many `feff.inp` files — each with a structure slightly shifted radially outwards (or inwards) from the original structure. File structure is organized as the following:

```
BNL
|   distortionator.py
|   gaussianator.py
|
|
└──DATA
|   |
|   |
|   └──Original_Structure
|   |   |   feff.inp
|   |
|   └──minus_pt_0001
|   |   |   feff.inp
|   |
|   └──plus_pt_0001
|   |   |   feff.inp
|   |
|   ....
```

A.2 gaussianator.py

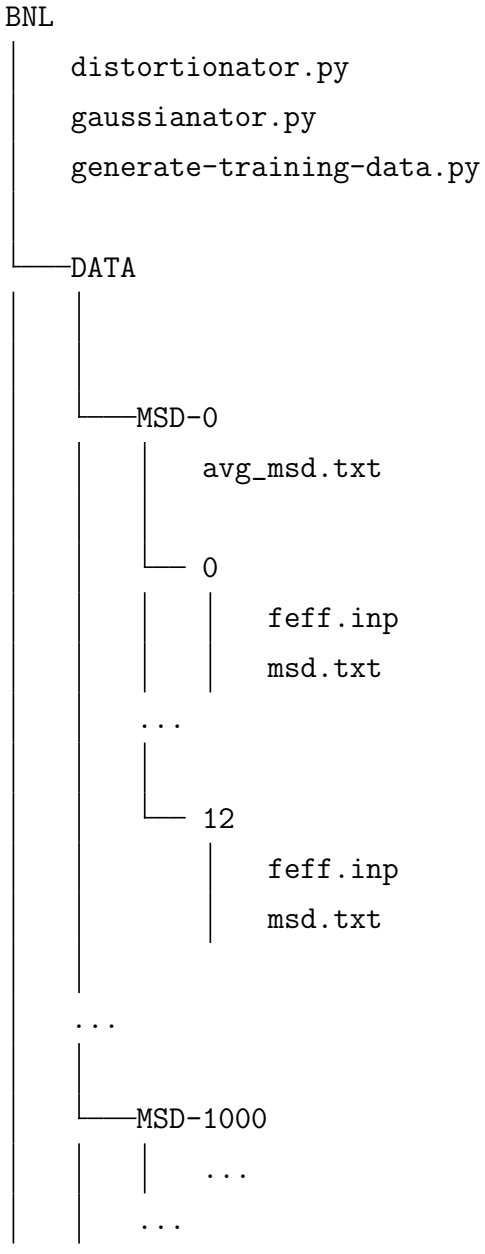
Take all the `xmu.dat` files (each one represents the sprectrum from the $\Delta\rho$ shifted crystrals) and generates many gaussian averaged XANES spectra. One file per different standard deviation of the gaussian. The File structure is organized as follows:


BNL
distortionator.py
gaussianator.py
DATA
Averaged_Spectra
sigma_0001.csv
sigma_0002.csv
...

```
# Inputs: -----
# dataframe df = the already mapped concat dataframe of all the different
    delta_rho shifted feff xanes
# float64 mean = mean of gaussian.
# float64 std = standard deviation of gaussian
# float64 skewness = skew parameter of stats.skewnorm
# Outputs: ----
# dataframe df_weighted2 = one dataframe. It is one distribution-weighted
    spectra with cols=['omega','mu']
# float64 avg_MSD = the mean squared displacement of the skewnorm-averaged
    spectrum
# Note a skewness of 0, sigma 1, and mean 0 is a standardized normal
    distribution.
def weight_by_distribution2(df_concat, mean, std, skewness):
# calculate the MSD -------------------
# get the bin heights for all the bins
bin_heights = np.array([stats.skewnorm.pdf(nn_dist, loc=mean, scale=std,
    a=skewness) for nn_dist in BINS])
normalization_factor = np.sum(bin_heights)
# same as sum(bin_height_i * nn_bond_dist)/(sum(bin_heights))
weighted_nn_dist_mean = np.dot(bin_heights, BINS) / normalization_factor
# same as sum(bin_height_i * ( nn_bond_dist_i - mean_bond_dist )^2
    )/sum(bin_heights)
sq_dif = np.square(np.subtract(BINS, weighted_nn_dist_mean))
avg_msd = np.divide(np.dot(bin_heights, sq_dif), normalization_factor)
# now do the spectrum -----------------
df_weighted2 = pd.DataFrame(data={'omega': df_concat.loc['0'].omega, 'mu':
    np.zeros(df_concat.loc['0'].omega.shape[0])})
for shift, bin_height in zip(SHIFTS, bin_heights):
    df_weighted2.mu += df_concat.loc[shift]['mu'].multiply(bin_height)
df_weighted2.mu /= normalization_factor # correct for the sum, so the area
    under the PDF=1
return df_weighted2, avg_msd
```

A.3 generate-training-data.py

This script generates the FEFF input files (`feff.inp`) for the disordered structures—i.e. the true-disordered structures, NOT the distorted structures used for the skew-norm averaging.



```
# Inputs: ----
# pandas dataframe df = the unshifted dataframe with spherical coordinates
# float shift_sigma = the width of the np.random.normal distribution from which
    shift distances are chosen
# Outpts: ----
# np arrays x, y, z = the shifted coorinates
# Notes: -----
# shift_val = radius of sphere project new point onto = distance of new
    disordered atom from original location
def gen_random_delta_rho_shift(df, shift_sigma):
    df_temp = df.copy()
    # SHIFT
    df_temp['shift_val'] = np.random.normal(loc=0, scale=shift_sigma,
        size=df_temp.shape[0])
    df_temp['theta'] = 6.28 * np.random.random_sample(df_temp.shape[0])
    df_temp['phi'] = 6.28 * np.random.random_sample(df_temp.shape[0])
    # Calculate the new coordintes
    df_temp['x'] +=
        round(df_temp.shift_val*np.sin(df_temp.phi)*np.cos(df_temp.theta), 5)
    df_temp['y'] +=
        round(df_temp.shift_val*np.sin(df_temp.phi)*np.sin(df_temp.theta), 5)
    df_temp['z'] += round(df_temp.shift_val*np.cos(df_temp.phi), 5)
    # turn to numpy array
    x1 = df_temp.loc[:, 'x'].values
    y1 = df_temp.loc[:, 'y'].values
    z1 = df_temp.loc[:, 'z'].values
    return x1, y1, z1
```

```
# Inputs: -----
# str folder path of one structure (contains 13 subfolders, one for each absober)
# Outputs: ----
# Returns float64 MSD, the mean-squared-displacement of the structure.
def do_one_structure(folder):
    bonds = set()
    rhos = []
    duplicates = 0
    for i in range(13):
        subfolder_path = os.path.join(folder, str(i))
        file = os.path.join(subfolder_path, 'feff.inp')
        df_absorbers = (load_initial_file(file)
                        .pipe(to_spherical)
                        .query('rho < 3.5')
                        )
        for index, row in df_absorbers.iterrows():
            option1 = (df_absorbers[df_absorbers.absorber==0].index[0], index)
            option2 = (index, df_absorbers[df_absorbers.absorber==0].index[0])
            if df_absorbers[df_absorbers.absorber==0].index[0] == index:
                pass
            elif option1 in bonds or option2 in bonds: # duplicate bond found
                duplicates += 1
            elif option1 not in bonds or option2 not in bonds: # new bond found
                bonds.add(option1)
                rhos.append(row.rho)
    if len(rhos) != 120 or len(bonds) != 120:
        raise Not120BondsException(len(rhos), len(bonds))
    dif = np.array(rhos) - np.mean(arr)
    squared = np.square(dif)
    summed = np.sum(squared)
    msd = summed/len(rhos)
    with open(os.path.join(folder, 'fixed_avg_msd.txt'), "w") as f:
        f.write(str(msd))
    return msd
```

A.4 create-g(r).ipynb

This iPython notebook loops through all the disordered structures and creates a histogram of nearest neighbor distances for each structure. Because there are 13 absorbers, each of which has 13 nearest neighbors, there are a total of 169 bond lengths. Many of these bonds are shared with absorbers and would be counted twice if one were not careful. There are only 120 unique bonds for the nearest neighbors of each atom in the first shell. This script keeps track of all the unique bonds to ensure no bond length is counted twice. We achieve this by utilizing python's set exclusivity and checking that the set contains the expected number of bonds when complete.

```
# inputs: ----
# str folder = the folder path which includes 13 subfolders within
# save_as = an optional boolean for saving the histogram plot.
# returns: ----
# msd, mean, sigma, skew
def do_one_structure(folder, save_as=None):
    rhos = set()
    for i in range(13):
        subfolder_path = os.path.join(folder, str(i))
        df_absorbers = (load_initial_file(os.path.join(subfolder_path,
            'feff.inp')).
              pipe(to_spherical).
              query('rho < 3.5 and rho > 0')
              )
        for rho in df_absorbers.rho:
            rhos.add(rho)
    if len(rhos) != 120:
        raise Not120BondsException(len(rhos))
    rhos = np.array(list(rhos))
    difs = rhos - np.mean(rhos)
    msd = np.sum(np.square(difs))/len(rhos)
    rhos = list(rhos)
    skew, mean, sigma = stats.skew(rhos), np.mean(rhos), np.std(rhos)
    # code for plotting removed for brevity
    if save_as is not None:
        plt.savefig(save_as, dpi=600, bbox_inches='tight')
```

```
    return msd, mean, sigma, skew
```

A.5 nn.ipynb

This Jupyter notebook contains the neural network, built with TensorFlow [43] and Keras [96].

A.6 nn-buddy.py

The sole purpose of this python script is to be imported by `nn.ipynb`. The script contains many useful helper functions that take care of data-loading, plotting, and linear interpolation of experimental data on the same energy mesh used for the training sample. This way, the

Appendix B

Simulating XANES Spectra

B.1 FEFF9 Design

FEFF is an *ab initio* absorption simulation software based on real-space multiple scattering (RSMS) and the Greene's function. The current version, FEFF9, calculates a self-consistent density function over a wide range of structures by considering the excited state properties within an all-electron framework [97]. Like other density-functional theory (DFT) software, FEFF begins with an initial guess for the electronic density. Then, using an optimization algorithm like gradient descent (See 3.2), FEFF iterates through a series of calculations until the resulting potential matches the initial potential (self-consistency). This process is depicted in Figure B.1.

FEFF first makes an initial guess for the electronic density, $\rho^{(0)}$ then calculates the initial potential, $V_{eff}^{(i)}$, which is a functional of that density. Using this potential, FEFF either solves the Schrodinger equation or the Green's function [98]. Finally, using Green's function, FEFF calculates the new potential, $\rho^{(i+1)}$, and checks if the potential is self-consistent. If not, the potential is updated, and the iterative process repeats. Once self-consistency is achieve, FEFF uses the potential to calculate the absorption spectrum using Fermi's Golden Rule 1.6. Alternatively, if using Green's function, FEFF calculates the density from Green's function and solves for the absorption through a series of approximations [2] [99]. The exact mathematical formalism is beyond the scope of this thesis.

The different calculations for FEFF are executed by a series of sub-programs, coordinated to run in order through a BASH script. The input file (typically `FEFF.inp`) file is written by the user. The card parameters are discussed in the next section. The `FEFF.inp` file is first read by the program `rdinp` and the *ab initio* Debye-Waller factor is calculated by `dmdw` if an

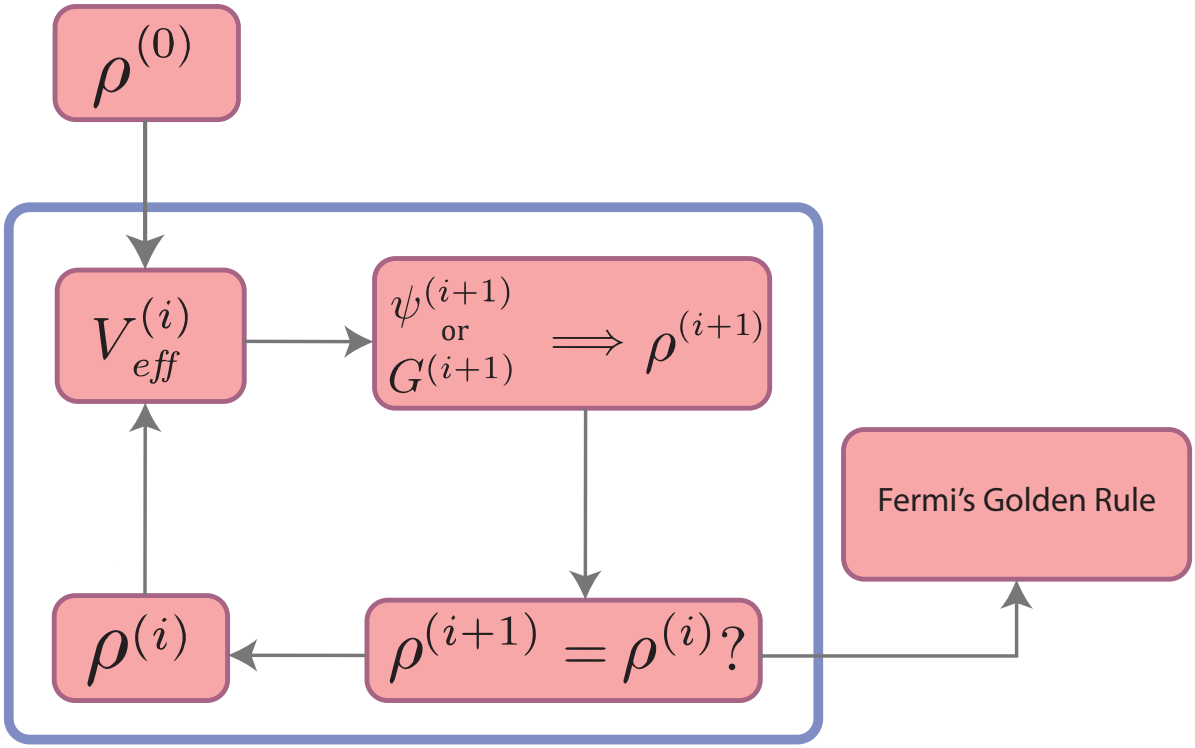


Figure B.1: Feff Diagram works by starting with an initaly guess for the density and calculating a resulting potential. From this, either the Schrodinger or Green

optional matrix is specified [100]. Then the muffin tin (overlap) potentials are calculated via `potph`. FEFF calculates the free atom potentials via a relativistic Dirac-Fock model [101]. Then Hedin-Lundqvist/Quinn self-energy is used for calculating the excited states, and the scattering potentials are calculated by overlapping the individual and muffin tin and free atom densities [102]. The last program run for the self-consistency scheme is `pot`. The core holes are calculated via `screen`, the angular momentum density of states is calculated wiht `ldos`, and `xsph` calculates the scattering phase shifts along with dipole matrix elements and the x-ray cross section [103]. Finally the full muliple scattering [104] is calculated and the Green's function is obtained via `fms`. The matrix elements are multiplied with results from the Green's function in `mkgtr` and the multiple scattering path are written via `path`. The effective scattering amplitutes f_{eff} (from which FEFF gets its name) and XAFS parameters for each path are calculated via `genfmt`, and the `ff2x` is used to obtain the final XANES spectrum, the result of which is convolved with the many-body spectral function `sfconv` and written in the file *xmu.dat*.

B.2 FEFF Cards

Calculating an absorption spectrum using FEFF9 requires an input file which is comprised of the coordinates of the structure as well as a series of parameters known as cards. Extensive explanations for each card can be found in the FEFF manual [99]. Here, we present only a brief description and suggestions for how to alter the cards in order to help fine the best parameters for replicating an experimental spectrum. The `SCF` card helps FEFF find the self-consistency potential and better estimate the Fermi level. The `EDGE` card specifies which edge is being simulate. In our case, we are interested in the L_3 edge. The `EXCHANGE` card specifies the exchange correlation potential used for both the fine structure and background calculations. Setting S02 to 1 ensures just alters the amplitude of the absorption spectrum via S_0^2, the squared determinant of overlap integrals for core orbitals [99]. The amplitude can be adjust later on, so a large value such as one is just by convention. The `XANES` card's parameters are all optional, but and are used to alter the output energy mesh. We select parameters that include a greater density of measurements near the regions of greatest change (the peaks). The `FMS` card provides information about the full multiple scattering. Finally, The `POTENTIAL` card describes the elements present in the material. One absorber potential must be specified as zero, while the other non-absorbing potentials specified are labeled one. Other non-absorbing elements present in the sample would be labeled as 2, 3, and so on.

References

[1] J. Timoshenko, D. Lu, Y. Lin, and A. I. Frenkel. *Supervised machine-learning-based determination of three-dimensional structure of metallic nanoparticles.* The Journal of Physical Chemistry Letters, 8(20):5091–5098 (2017).

[2] M. Newville. *EXAFS Analysis Using feff and feffit.* Journal of synchrotron radiation, 8(2):96–100 (2001).

[3] F. R. Elder, A. M. Gurewitsch, R. V. Langmuir, and H. C. Pollock. *Radiation from electrons in a synchrotron.* Phys. Rev., 71:829–830 (1947).

[4] S. Mobilio, F. Boscherini, and C. Meneghini. *Synchrotron Radiation.* Springer (2016).

[5] D. J. Gardenghi et al. *Synchrotron radiation-based spectroscopic investigation of the electronic and geometric structures of iron-sulfur clusters, particles, and minerals.* Ph.D. thesis, Montana State University-Bozeman, College of Letters & Science (2012).

[6] A. Einstein. *On a heuristic point of view concerning the production and transformation of light.* Annalen der Physik, 17(132):1–16 (1905).

[7] H. Fricke. *The k-characteristic absorption frequencies for the chemical elements magnesium to chromium.* Physical Review, 16(3):202 (1920).

[8] G. Hertz. *ueber die absorptionsgrenzen in derl-serie.* Zeitschrift fuer Physik, 3(1):19–25 (1920).

[9] J. J. Rehr and R. C. Albers. *Theoretical approaches to x-ray absorption fine structure.* Reviews of modern physics, 72(3):621 (2000).

[10] M. Newville. *Fundamentals of xafs.* Reviews in Mineralogy and Geochemistry, 78(1):33–74 (2014).

[11] D. Koningsberger and R. Prins. *X-ray absorption: principles, applications, techniques of exafs, sexafs, and xanes* (1988).

[12] N. J. Marcella. *Decoding Reactive Structures in Bimetallic Catalysts*. Ph.D. thesis, Stony Brook University (2021).

[13] K. Klementev. *Xafs spectroscopy. i. extracting the fine structure from the absorption spectra*. arXiv preprint physics/0003086 (2000).

[14] J. E. Penner-Hahn et al. *X-ray absorption spectroscopy*. Comprehensive Coordination Chemistry II, 2:159–186 (2003).

[15] J. J. Rehr and R. C. Albers. *Theoretical approaches to x-ray absorption fine structure*. Reviews of modern physics, 72(3):621 (2000).

[16] B. K. Teo. *EXAFS: Basic Principles and Data Analysis*, volume 9. Springer Science & Business Media (2012).

[17] J. Rehr, R. Albers, and S. Zabinsky. *High-order multiple-scattering calculations of x-ray-absorption fine structure*. Physical review letters, 69(23):3397 (1992).

[18] A. Ankudinov, J. Rehr, J. Low, and S. Bare. *Theoretical interpretation of xafs and xanes in pt clusters*. Topics in catalysis, 18(1):3–7 (2002).

[19] A. Filipponi, A. Di Cicco, and C. R. Natoli. *X-ray-absorption spectroscopy and n-body distribution functions in condensed matter. i. theory*. Physical Review B, 52(21):15122 (1995).

[20] A. Filipponi and A. Di Cicco. *X-ray-absorption spectroscopy and n-body distribution functions in condensed matter. ii. data analysis and applications*. Physical Review B, 52(21):15135 (1995).

[21] M. Rühle and M. Wilkens. *Transmission electron microscopy*. In R. W. Cahn and P. Haasenae (editors), *Physical Metallurgy (Fourth Edition)*, chapter 11, pp. 1033–1113. North-Holland, Oxford, fourth edition edition (1996).

[22] B. J. Palmer, D. M. Pfund, and J. L. Fulton. *Direct modeling of EXAFS spectra from molecular dynamics simulations*. The Journal of Physical Chemistry, 100(32):13393–13398 (1996).

[23] S. Gurman and R. McGreevy. *Reverse monte carlo simulation for the analysis of exafs data.* Journal of Physics: Condensed Matter, 2(48):9463 (1990).

[24] Y. Lin, M. Topsakal, J. Timoshenko, D. Lu, S. Yoo, and A. I. Frenkel. *Machine-learning assisted structure determination of metallic nanoparticles: A benchmark.* In *Handbook on big data and machine learning in the physical sciences: Volume 2. Advanced Analysis Solutions for Leading Experimental Techniques*, pp. 127–140. World Scientific (2020).

[25] J. Timoshenko, A. Anspoks, A. Cintins, A. Kuzmin, J. Purans, and A. I. Frenkel. *Neural network approach for characterizing structural transformations by x-ray absorption fine structure spectroscopy.* Physical review letters, 120(22):225502 (2018).

[26] D. Uzio and G. Berhault. *Factors governing the catalytic reactivity of metallic nanoparticles.* Catalysis Reviews, 52(1):106–131 (2010).

[27] M. Jørgensen and H. Grönbeck. *Strain affects co oxidation on metallic nanoparticles non-linearly.* Topics in Catalysis, 62(7):660–668 (2019).

[28] W. Research. *ElementData.* `https://reference.wolfram.com/language/ref/ElementData.html` (2014).

[29] A. Azzalini and A. Capitanio. *Statistical applications of the multivariate skew normal distribution.* Journal of the Royal Statistical Society: Series B (Statistical Methodology), 61(3):579–602 (1999).

[30] P. Virtanen et al. *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.* Nature Methods, 17:261–272 (2020).

[31] C. R. Harris et al. *Array programming with NumPy.* Nature, 585(7825):357–362 (2020).

[32] C.-W. Pao et al. *Photoconduction and the electronic structure of silica nanowires embedded with gold nanoparticles.* Physical Review B, 84(16):165412 (2011).

[33] Y.-G. Kim, S.-K. Oh, and R. M. Crooks. *Preparation and characterization of 1- 2 nm dendrimer-encapsulated gold nanoparticles having very narrow size distributions.* Chemistry of Materials, 16(1):167–172 (2004).

[34] Y. Guo et al. *Uniform 2 nm gold nanoparticles supported on iron oxides as active catalysts for co oxidation reaction: structure–activity relationship.* Nanoscale, 7(11):4920–4928 (2015).

[35] G. Carleo et al. *Machine learning and the physical sciences.* Reviews of Modern Physics, 91(4):045002 (2019).

[36] C. Szegedy, A. Toshev, and D. Erhan. *Deep Neural Networks for Object Detection* (2013).

[37] G. Hinton et al. *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.* IEEE Signal Processing Magazine, 29(6):82–97 (2012).

[38] S. Ruan, J. O. Wobbrock, K. Liou, A. Ng, and J. Landay. *Speech is 3x faster than typing for english and mandarin text entry on mobile devices.* arXiv preprint arXiv:1608.07323 (2016).

[39] J. Schmidhuber. *Deep learning in neural networks: An overview.* Neural networks, 61:85–117 (2015).

[40] A. S. Lokman and M. A. Ameedeen. *Modern chatbot systems: A technical review.* In *Proceedings of the future technologies conference*, pp. 1012–1023. Springer (2018).

[41] G. Lopez, L. Quesada, and L. A. Guerrero. *Alexa vs. Siri vs. Cortana vs. Google Assistant: a comparison of speech-based natural user interfaces.* In *International Conference on Applied Human Factors and Ergonomics*, pp. 241–250. Springer (2017).

[42] R. Sarikaya. *The technology behind personal digital assistants: An overview of the system architecture and key components.* IEEE Signal Processing Magazine, 34(1):67–81 (2017).

[43] M. Abadi et al. *TensorFlow: Large-scale machine learning on heterogeneous systems* (2015). Software available from tensorflow.org.

[44] A. Paszke et al. *Pytorch: An imperative style, high-performance deep learning library.* In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alche-Buc, E. Fox, and R. Garnett (editors), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc. (2019).

[45] F. Girosi, M. Jones, and T. Poggio. *Regularization theory and neural networks architectures.* Neural computation, 7(2):219–269 (1995).

[46] V. Nair and G. E. Hinton. *Rectified linear units improve restricted boltzmann machines.* In *Icml* (2010).

[47] J. Kukavcka, V. Golkov, and D. Cremers. *Regularization for deep learning: A taxonomy.* arXiv preprint arXiv:1710.10686 (2017).

[48] 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).

[49] S. Cai, Y. Shu, G. Chen, B. C. Ooi, W. Wang, and M. Zhang. *Effective and efficient dropout for deep convolutional neural networks.* arXiv preprint arXiv:1904.03392 (2019).

[50] H. Wu and X. Gu. *Towards dropout training for convolutional neural networks.* Neural Networks, 71:1–10 (2015).

[51] A. Cauchy et al. *Méthode générale pour la résolution des systemes d'équations simultanées.* Comp. Rend. Sci. Paris, 25(1847):536–538 (1847).

[52] I. Loshchilov and F. Hutter. *Decoupled weight decay regularization.* Conference Paper at ICLR 2019 (2019).

[53] S. Ruder. *An overview of gradient descent optimization algorithms.* CoRR, abs/1609.04747 (2016).

[54] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li. *Parallelized stochastic gradient descent.* In *NIPS*, volume 4, p. 4. Citeseer (2010).

[55] D. R. Wilson and T. R. Martinez. *The general inefficiency of batch training for gradient descent learning.* Neural networks, 16(10):1429–1451 (2003).

[56] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning representations by back-propagating errors.* nature, 323(6088):533–536 (1986).

[57] N. Qian. *On the momentum term in gradient descent learning algorithms.* Neural networks, 12(1):145–151 (1999).

[58] J. Duchi, E. Hazan, and Y. Singer. *Adaptive subgradient methods for online learning and stochastic optimization.* Journal of machine learning research, 12(7) (2011).

[59] C. Igel and M. Hüsken. *Improving the rprop learning algorithm.* In *Proceedings of the second international ICSC symposium on neural computation (NC 2000)*, volume 2000, pp. 115–121. Citeseer (2000).

[60] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. *The marginal value of adaptive gradient methods in machine learning.* arXiv preprint arXiv:1705.08292 (2017).

[61] D. P. Kingma and J. Ba. *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980 (2014).

[62] J. Thaller. *Toward a direct measurement of strain-dependent surface stress in soft solids* (2019).

[63] R. Larson and B. Farber. *Elementary statistics.* Pearson Education Canada (2019).

[64] S. Ioffe and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* In *International conference on machine learning*, pp. 448–456. PMLR (2015).

[65] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. *How does batch normalization help optimization?* In *Proceedings of the 32nd international conference on neural information processing systems*, pp. 2488–2498 (2018).

[66] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. *How does batch normalization help optimization?* In *Proceedings of the 32nd international conference on neural information processing systems*, pp. 2488–2498 (2018).

[67] J. Kohler, H. Daneshmand, A. Lucchi, T. Hofmann, M. Zhou, and K. Neymeyr. *Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization.* In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 806–815. PMLR (2019).

[68] G. Yang, J. Pennington, V. Rao, J. Sohl-Dickstein, and Schoenholz. *A mean field theory of batch normalization.* arXiv preprint arXiv:1902.08129 (2019).

[69] D. A. Van Dyk and X.-L. Meng. *The art of data augmentation.* Journal of Computational and Graphical Statistics, 10(1):1–50 (2001).

[70] S. Frühwirth-Schnatter. *Data augmentation and dynamic linear models.* Journal of time series analysis, 15(2):183–202 (1994).

[71] L. Perez and J. Wang. *The effectiveness of data augmentation in image classification using deep learning.* arXiv preprint arXiv:1712.04621 (2017).

[72] M. A. Tanner and W. H. Wong. *The calculation of posterior distributions by data augmentation.* Journal of the American statistical Association, 82(398):528–540 (1987).

[73] C. Shorten and T. M. Khoshgoftaar. *A survey on image data augmentation for deep learning.* Journal of Big Data, 6(1):1–48 (2019).

[74] S. Bozinovski. *Reminder of the first paper on transfer learning in neural networks, 1976.* Informatica, 44(3) (2020).

[75] S. J. Pan and Q. Yang. *A survey on transfer learning.* IEEE Transactions on knowledge and data engineering, 22(10):1345–1359 (2009).

[76] L. Torrey and J. Shavlik. *Transfer learning.* In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264. IGI global (2010).

[77] J. Somers. *The Pastry A.I. That Learned to Fight Cancer.* The New Yorker (2021).

[78] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. *To transfer or not to transfer.* In *NIPS 2005 workshop on transfer learning*, volume 898, pp. 1–4 (2005).

[79] A. Amini et al. *Learning robust control policies for end-to-end autonomous driving from data-driven simulation.* IEEE Robotics and Automation Letters, 5(2):1143–1150 (2020).

[80] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. *Carla: An open urban driving simulator.* In *Conference on robot learning*, pp. 1–16. PMLR (2017).

[81] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra et al. *Matching networks for one shot learning.* Advances in neural information processing systems, 29:3630–3638 (2016).

[82] Y. Xian, B. Schiele, and Z. Akata. *Zero-shot learning-the good, the bad and the ugly.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4582–4591 (2017).

[83] S. Ravi and H. Larochelle. *Optimization as a model for few-shot learning* (2016).

[84] M. L. Boas. *Mathematical methods in the physical sciences; 3rd ed.* Wiley, Hoboken, NJ (2006).

[85] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman. *1d convolutional neural networks and applications: A survey.* Mechanical Systems and Signal Processing, 151:107398 (2021).

[86] W. Rawat and Z. Wang. *Deep convolutional neural networks for image classification: A comprehensive review.* Neural computation, 29(9):2352–2449 (2017).

[87] R. Zabih. *Cs1114 section 6: Convolution.* Cornell University (2013). Cornell University.

[88] O. S. Kayhan and J. C. v. Gemert. *On translation invariance in cnns: Convolutional layers can exploit absolute spatial location.* In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14274–14285 (2020).

[89] Wes McKinney. *Data Structures for Statistical Computing in Python.* In Stéfan van der Walt and Jarrod Millman (editors), *Proceedings of the 9th Python in Science Conference*, pp. 56 – 61 (2010).

[90] T. pandas development team. *pandas-dev/pandas: Pandas* (2020).

[91] L. Biewald. *Experiment tracking with weights and biases* (2020). Software available from wandb.com.

[92] C. Finn, P. Abbeel, and S. Levine. *Model-agnostic meta-learning for fast adaptation of deep networks.* In *International Conference on Machine Learning*, pp. 1126–1135. PMLR (2017).

[93] F. Zhou, B. Wu, and Z. Li. *Deep meta-learning: Learning to learn in the concept space* (2018).

[94] A. Nichol, J. Achiam, and J. Schulman. *On first-order meta-learning algorithms.* arXiv preprint arXiv:1803.02999 (2018).

[95] Y.-X. Wang and M. Hebert. *Learning to learn: Model regression networks for easy small sample learning.* In *European Conference on Computer Vision*, pp. 616–634. Springer (2016).

[96] F. Chollet et al. *Keras* (2015).

[97] K. Jorissen and J. J. Rehr. *New Developments in FEFF: FEFF9 and JFEFF.* In *Journal of Physics: Conference Series*, volume 430, p. 012001. IOP Publishing (2013).

[98] T. S. Tan, J. J. Kas, and J. J. Rehr. *Real-space green's function approach for x-ray spectra at high temperature.* Phys. Rev. B, 104:035144 (2021).

[99] J. J. Rehr, J. J. Kas, F. D. Vila, M. P. Prange, and K. Jorissen. *Parameter-free Calculations of X-ray Spectra with FEFF9.* Physical Chemistry Chemical Physics, 12(21):5503–5513 (2010).

[100] F. D. Vila, J. Rehr, H. Rossner, and H. Krappe. *Theoretical x-ray absorption debye-waller factors.* Physical Review B, 76(1):014301 (2007).

[101] A. Ankudinov, S. Zabinsky, and J. Rehr. *Single configuration dirac-fock atom code.* Computer physics communications, 98(3):359–364 (1996).

[102] J. J. Rehr, J. J. Kas, M. P. Prange, A. P. Sorini, Y. Takimoto, and F. Vila. *Ab initio theory and calculations of x-ray spectra.* Comptes Rendus Physique, 10(6):548–559 (2009).

[103] A. Sorini, J. Kas, J. Rehr, M. Prange, and Z. H. Levine. *Ab initio calculations of electron inelastic mean free paths and stopping powers.* Physical Review B, 74(16):165111 (2006).

[104] J. J. Rehr and R. C. Albers. *Scattering-matrix formulation of curved-wave multiple-scattering theory: Application to x-ray-absorption fine structure.* Physical Review B, 41(12):8139 (1990).

I declare that I prepared and wrote this thesis work independently and with no other means than those referenced in the text.

Jeremy K. Thaller	Date