Investigation of Bond Strain Effects on XANES Spectra by Supervised Machine Learning

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

July 22, 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 both the computer-simulated nanoparticles of different sizes and degrees of disorder, as well as on experimental data of disordered nanoparticles.

Executive Summary

Your executive summary will give a detailed summary of your thesis, hitting the high points and perhaps including a figure or two. This should have all of the important take-home messages; though details will of course be left for the thesis itself, here you should give enough detail for a reader to have a good idea of the content of the full document. Importantly, this summary should be able to stand alone, separate from the rest of the document, so although you will be emphasizing the key results of your work, you will probably also want to include a sentence or two of introduction and context for the work you have done.

Acknowledgments

The acknowledgment section is optional, but most theses will include one. Feel free to thank anyone who contributed to your effort if the mood strikes you. Inside jokes and small pieces of humor are fairly common here ...

Contents

List of Figures

Chapter 1

Introduction

Synchrotron radiation was first observed by General Electric in Schenectady, New York [2]. 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.

1.1 X-ray Absorption Spectroscopy

X-ray absorption spectroscopy measures the absorption of high-energy photons by a sample as a function of energy [3]. 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:

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

Above the absorption edge, the condensed state has characteristic absorption jumps where the incident photon's energy matches the binding energy of a core electron. 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 [4, 5].

In experimental setups, particular energy photons are selected from the broad spectrum of synchrotron radiation via a pair of monochromaters. The primary monochromater is a

crystal with interplanar spacing, d, chosen specify to satisfy the Bragg equation 1.2, relecting photons of wavelngth, λ at angle θ.

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

The secondary monochromater removes higher order harmonics that satisfy the Bragg equation ($2\lambda, 3\lambda etc.$). Different wavelngths of light can be selected by changing the angle. In the XAFS setup, two ionization chambers are used to measure the incident and transmitted light intensity. For any study, the absoprion spectrum for a reference sample is also measured to calibrate the energy scale.

1.1.1 XAFS

X-ray Absorption Fine-Structure (XAFS) spectroscopy refers to the study of absorption spectra created from high-intensity x-ray interactions. 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 has made it an indispensable technique in fields such as materials biology, chemistry, and materials science [6] [7].

The XAFS equation is

$$\chi = \frac{\mu(E) - \mu_0(E)}{\mu_0(E) - \mu_b(E)} \tag{1.3}$$

where μ is the measured absorption, μ_0 is the "atomic" absorption due to .... specific electrons, and μ_b is the absorption of other processes [8], typically approximated with the Victoreen polynomial (1.4).

$$\mu_b(E) = aE^{-3} + bE^{-4} \tag{1.4}$$

The coefficients α and β can be found via a simple regression on a sprectrum measured at pre-edge energies [8].

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.

1.1.2 EXAFS

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.

EXAFS Data Reduction

To prepare the absorption data for fitting (data reduction), several steps must be performed. First the pre-edge background must be removed using the Victoreen formula (1.4) or an alternative polynomial. Next, the atomic background, $\mu_0(E)$ is removed and the absorption 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. High order spline fittings must be used to remove the low frequency, immeasurable oscillations due to photoelectron scattering with nearby valence electrons. For EXAFS Fitting, the EXAFS function is often transformed into k-space, and Fourier transformed into a radial distribution-like function via (**??**) [9].

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

Alternatively, one can convert the wave to R-space, which is what Artemis and Athena, two of the most common EXAFS fitting software, do.

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 many-body Fermi golden rule (1.6).

$$\mu(E) \propto \sum_f^{E_f > E_F} |\langle f|H_{int}|i\rangle|^2 \, \delta(E - E_F - Ef) \tag{1.6}$$

Fermi's golden rule describes the probability of a transition in state occurring. Here (1.6), $\mu(E)$ is the absorption coefficient, f and i are the final and initial states of the photoelectron, respectively. H_{int} is the matter-light interaction operator, and the delta function ensures conservation of energy. The summation over all enegy 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. EXAFS only takes into account the local region around the absorber at a distance r_j from the backscattering atom.

$$\chi(k) \approx F_j(k)\frac{\sin[2kr_j + \phi_{ij}(k)]}{2r_j^2} \tag{1.7}$$

The above equation (1.7) describes the EXAFS signal for only one backscattering atom. For a system with many atoms, one must sum over all the backscattered waves. To account for the inevitable variation in bond lengths, the Debye-Waller factor σ_j is introduced; this describes the standard deviation in bond-length of the sample.[1]. The lifetime of the photo-electron's excited state is taken into account by introducing an exponential term to account for the mean-free-path, $e^{-2r_j/\lambda}$. Combining all this information with an amplitude correction factor of S_0, we arrive at the EXAFS equation (1.8.

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

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 alternatives include molecular dynamics, reverse Monte Carlo simulations and neural networks [11].

1.1.3 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 which 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 Lab [1] has shown that it is possible for a model to 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

[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 [10]

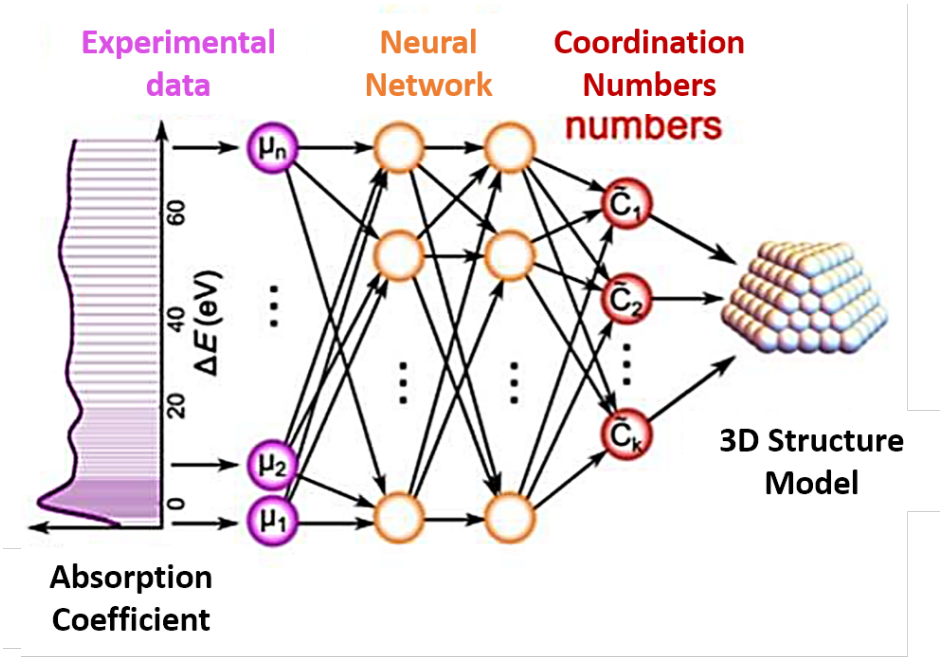

Figure 1.1: From [1], the neural netowrk is trained to take a XANES spectrum from a metallic nanoparticle, and predict the coordination number of the structure. This coordination number of nanoparticles is a known discriptor, which allows for easy calculation of the the nanoparticle's size and shape.

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 catalyst [12] [13].

1.2 Diagram for collecting experimental XAFS data???

1.3 Goals of the Thesis and Approach

Building of the previous work [1], the goal of this thesis is comprised of two parts: first, determine whether bond-length information is encoded in the XANES spectrum; and second, utilize machine learning to predict the bond-length disorder of metallic nanoparticles from a XANES spectrum. As with the 2017 paper [1], the work is conducted with gold (Au) nanoparticles, in part to expand on the previous paper, and in part due to the access to

the relevant experimental data. Machine learning requires a substantial amount of training data, far more than could be experimentally obtained. Consequently, we will rely on absorption simulation software to create the training data, a collection of XANES spectra for Au nanoparticles with known disorder. Finally, once the network is trained, the network must be modified to abstract to experimental data to compenstate for the systematic differences between the simulation and experimental data.

1.4 Outline of the Thesis

Often the most time-intensive part of any machine learning-based project is the process of collecting and preprocessing the data. Chapter 2 is dedicated entirely to the process of generating XANES spectra via simulations. Next, a solid foundational understanding of machine learning is important in understanding the approach. All machine learning terms present in later chapters are defined here. Chapter 4 describes the exact model architecture and results of the training process. Further discussions and future work are included in chpater 5. *Appendix A includes a description of the main Python script written for this thesis and necessary for replicated the work. The files can be obtained upon request by contacting the author.*

Chapter 2

Simulating Disorder

Before making any predictions, neural networks must first be trained on a large quantity of data. Specifically, to teach our neural network to predict the mean squared displacement (MSD), we must first generate a large quantity 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-intensitive and expensive. Rather, simulations provide a practical alternative, though even simulating each possible disordered structure individually, would be extremely time-intensive. This process is discussed in section 2.5. First, a discussion on the development of a new method for simulating disordered nanoparticles is presented in sections 2.1–2.3. The new process utilizes 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 clever statistical averaging. In this chapter, 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. The efficacy and limitations of this approach are discussed in section 2.4.

2.1 Generating Distortion Not Disorder

Instead of creating structures with a range of *disorder*, we instead begin by generating structures with a range of *distortion*. Wheres *disorder* refers to a statistical average of atomic displacement from their original position, characterized by MSD and the width σ^2 of a partial radial distribution function, *distortion* refers only 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.

A 2-dimensional projection of this isotropic distorion is presented in Figure 2.1. Though the figure only shows the xy-plane projection of the first 12 nearest neighbors, the actual structure used consists of the first four shells (55 atoms) with a lattice constant of 4.0782 Å to match that of bulk Au. *Citation? Wolfram Element Data?* In reality, the nearest-neighbor distances for Au nanoparticles are likely smaller *citations?*; this can be accounted for later

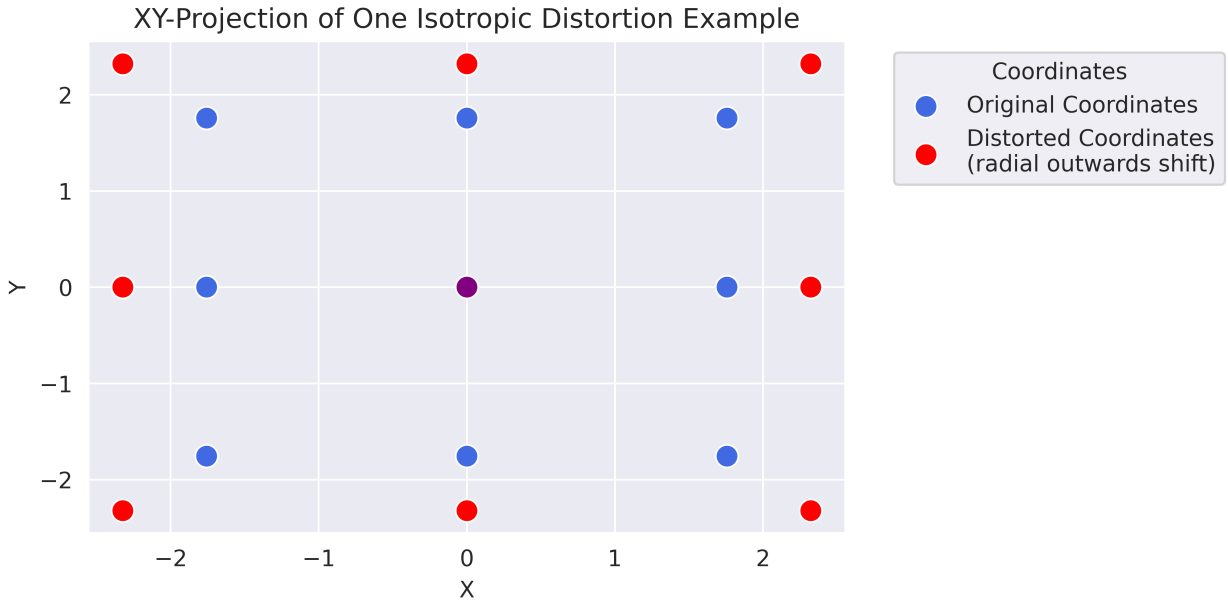

Figure 2.1: Each point represents an atom of first 12 nearest neighbors of a Au cluster projected onto the xy-plane. The four corner points actually represent two atoms because of the projection. The blue atoms represent the original coordinates, and the red atoms represent the radially shifted coordinates. The center absorber atom is purple since its original position is the same as its distorted position.

on in the averaging process since the original coordinates will only be one structure out of many. The important part is that the 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 $(0, 0, 0)$, but all other atomic coordinates are in a shifted location 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 has all coordinates shifted 0.45 Å radially inwards towards the center absorber, and the FEFF input file with the largest outwards shift has coordinates shifted 0.45 Å radially outwards away from the center absorber.

Each FEFF input file is run with the following parameters:

```
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.
```

Running the 91 simulations (one for each of the distorted structure) takes approximately 30 minutues. Were we to generate thousands more or employ RMC or MD, this process could take weeks of compute time. We plot the resulting XANES spectra from the FEFF simulations in figure 2.2.

2.2 Generating Disorder via Probability Distribution Averaging

One way to characterize system disorder is with the Gaussian width, σ , of the partial radial distribution function. The idea of our statistical averaging method is to emulate this width by weighting the simulated XANES spectra accordingly. For example, Figure 2.3 depicts a histogram with $\sigma = 0.1$ Å. 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 2.3 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 2.3 is calculate via Equation (2.1):

$$\langle\mu(E)\rangle = \frac{1}{S}\sum_{\Delta\rho=-.45}^{+.45} g\left(\Delta\rho \mid \mu = 0, \sigma = 0.1\right)\mu(E \mid \Delta\rho) \tag{2.1}$$

In the above equation, $\Delta\rho$ is the isotropic, radial displacement of each atom from its original position, and $\mu(E \mid \Delta\rho)$ is the simulated FEFF spectrum for the given $\Delta\rho$ configuration. Furthermore, in Equation (2.1), S represents a standardization factor needed to negate the effect of the changing Gaussian height as a function of the variance, σ^2. With the inclusion

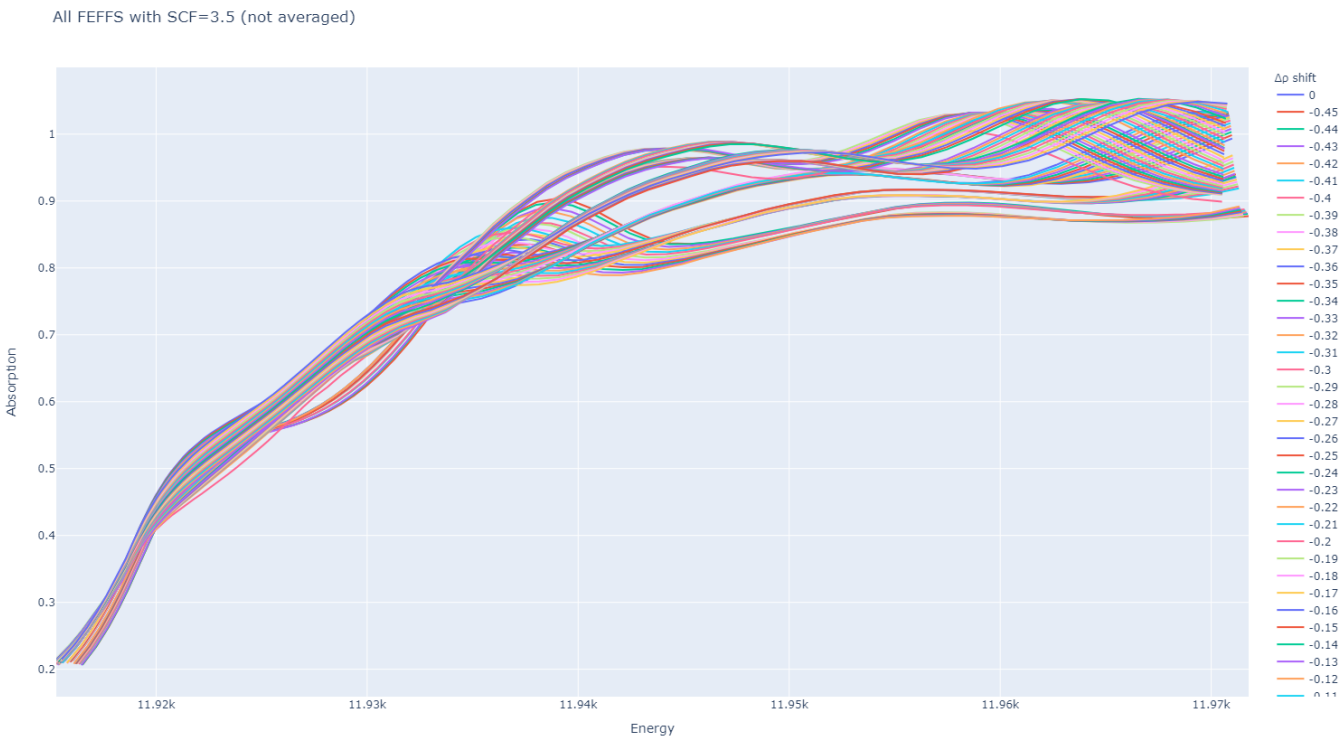

Figure 2.2: *TEMPORARY - way too much info. I'll select a few.* Each spectrum represents the FEFF simulation results for a different distorted structure. For each spectrum, the crystal structure and center absorber remain constant, the only parameter that varies is the euclidean distance from the center to the other coordinates.

of S , only the relative heights of each bin matters for producing the averaged XANES spectrum. This standardization factor is defined in Eqation (2.2):

$$S = \sum_{\Delta\rho=-.45}^{+.45} g\left(\Delta\rho \mid \mu = 0, \sigma = 0.01\right) \tag{2.2}$$

In both equations (2.1) and (2.2), the function g is just the typical Gaussian distribution probability density function (Equation 2.3):

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{2.3}$$

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

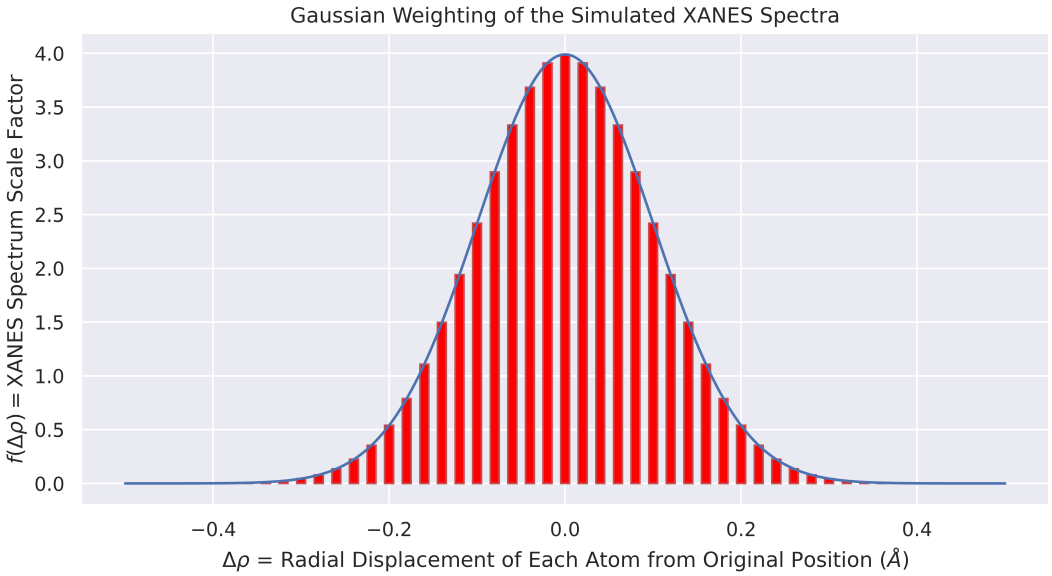

Figure 2.3: A Gaussian distribution probability density function can be used to calculate the relative weight of each FEFF generated XANES spectrum towards one simulated, disordered spectrum. Each bin (red bar) represents a FEFF generated spectrum; the x-axis is the isotropic shift of the atomic positions, and the y-axis is the relative weight factor.

the mean and variance; to simulate even more conditions, however, we can instead use the multivariate skew-normal distribution (2.4) [14, 15], *f(x)*.

$$f(x) = 2\phi(x)\Phi(\alpha x) \tag{2.4}$$

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

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{2.5}$$

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

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)\ dt \tag{2.6}$$

Equation (2.4) includes the shape parameter, α, which has the nice property of producing a right-skewed distribution when positive and a left-skewed distibution when negative. When $\alpha = 0$, the distribution simply produces the typical Gaussian distribution (eq. 2.3). Utilizing equation (2.4), we can vary μ, σ, and α to alter the first four moments of the function: mean, standard deviation, skew, and kurtosis. Eighteen possible skew-norm weighting functions are plotted in Figure 2.4. To produce the neural network training data, 1000 such weightings are used.

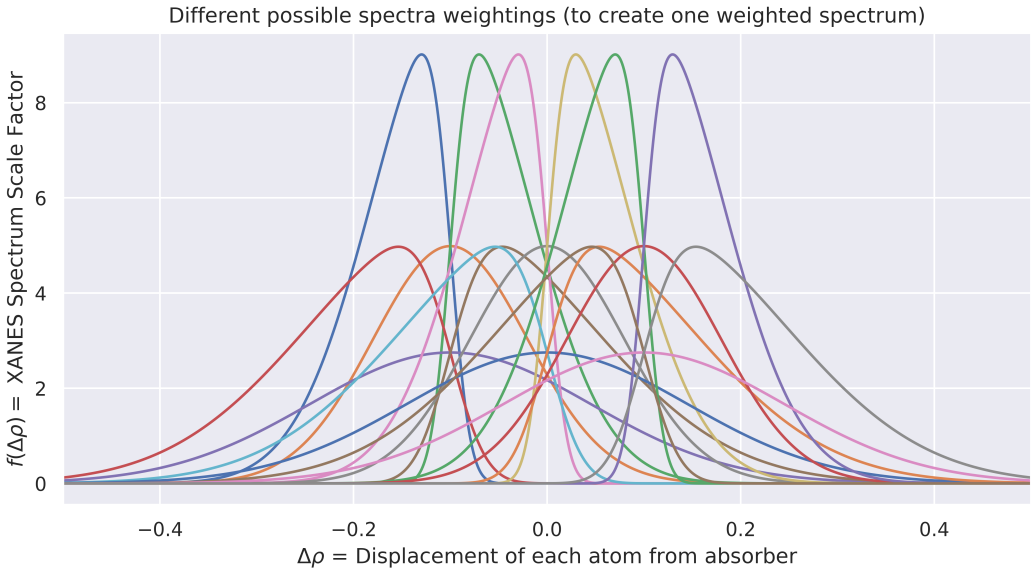

Figure 2.4: Eighteen skew-norm distributions plotted with all possible combinations of $\sigma = \{.08, .145\}$, $\mu = \{-.1, 0, .1\}$, and $\alpha = \{-5, 0, 5\}$. Each represents a possible way to produce a simulated, disordered spectrum from many FEFF-simulated, distorted spectra

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. *Instead of characterizing the disordered spectrum by the standard deviation of the gaussian used to create it.* The weighted mean squared displacement, MSD, is calculated via equation (2.7):

$$MSD = \frac{1}{S} \sum_{\Delta\rho=-.45}^{+.45} f\left(\Delta\rho \mid \mu, \sigma^2, \alpha\right) \tag{2.7}$$

Here, $f(x)$ is the skew-norm function from equation (2.4), and the MSD of each individual FEFF spectrum is equal to the isotropic distortion, $\Delta\rho$.

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. *Can someone give me citations for these?* In Figure 2.5, 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, this parameter was found to be $\sigma^2 = 0.0081(5)$ Å^2, and for the 8 nm disordered particle, $\sigma^2 = 0.0102(8)$ Å^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.

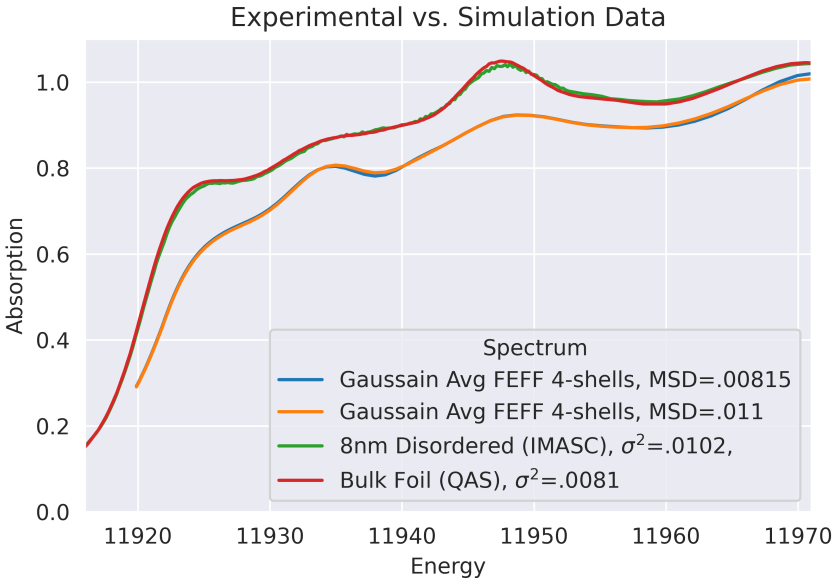

Figure 2.5: Comparing the bulk foil (red) measurement to the 8 nm disordered nanoparticle (green) measurement is an analog to comparing the simulated, non disordered FEFF spectrum (blue) to the simulated disordered spectrum (orange).

In Figure 2.5, the bulk Au foil spectrum is above the 8 nm nanoparticle spectrum (more absorption) until the peak around 11937 eV, where the NP absorption becomes higher. The two criss-cross again over the next two peaks, changing which material has the higher absorbance in an energy range. This change is more easily seen in Figure 2.6, which plots the difference between the 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.

Figures 2.5 and 2.6 aren't meant to be perfect comparisons of simulations vs. experimental data. For one, the experimental data compares a bulk spectrum to a nanoparticle. By contrast, both the simulation spectra are of the same size 55 atom cluster. Still, much of the disorder trends are coded in the simulation approach.

To test if the size information is also coded in our simulations, we compare different size simulations to experimental data in Figure 2.8. As expected, including more atoms in the simulation produces more bulk-like spectrum characteristics, such as larger amplitude peaks.

I NEED A PLOT COMPARING THIS METHOD TO PARTICLE AVERAGED METHOD HERE.

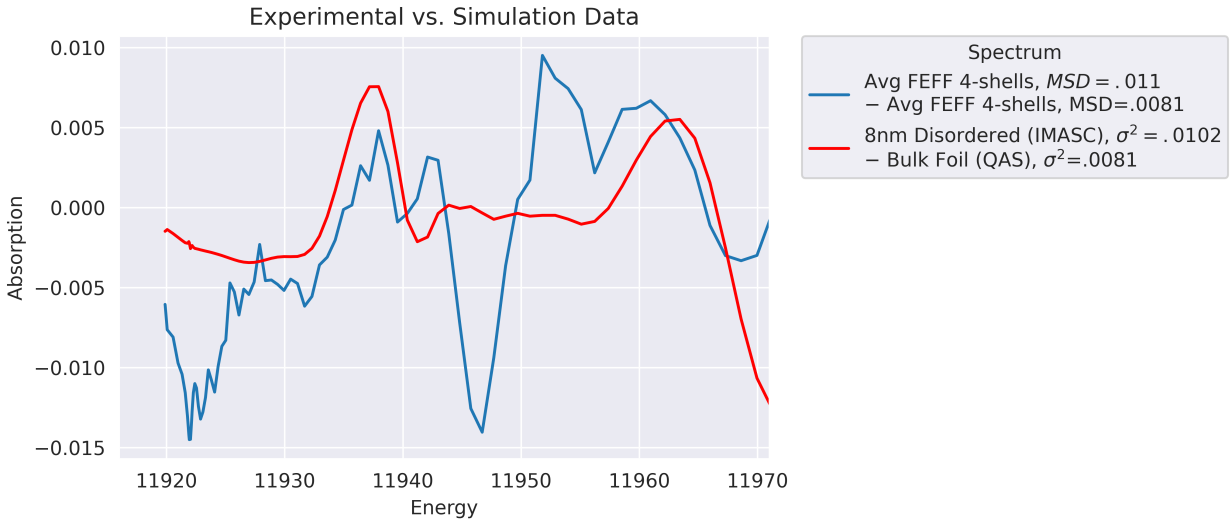

Figure 2.6: The difference between the nanoparticle spectrum and the bulk spectrum are plotted for the same data as in Figure 2.5. It is easier to see where the bulk and the nanoparticle absorption crisscross by plotting the difference.

2.4 Particle-Averaged FEFF Simulated Disordered Structures

2.5 Traditional Particle-Averaged Simulations

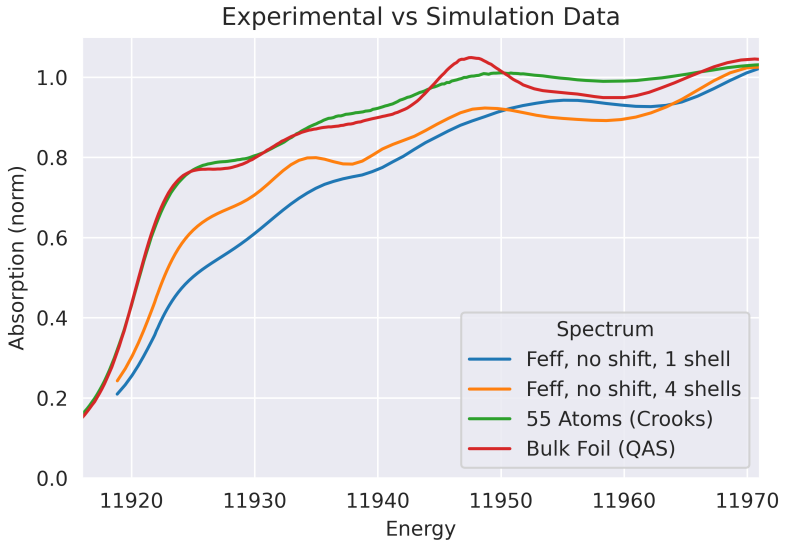

Figure 2.7: Comparing the bulk foil (red) measurement to the 55 atom nanoparticle (green) measurement is an analog to comparing the 13 atom simulated spectrum (blue) to the 55 atom simulated spectrum (orange).

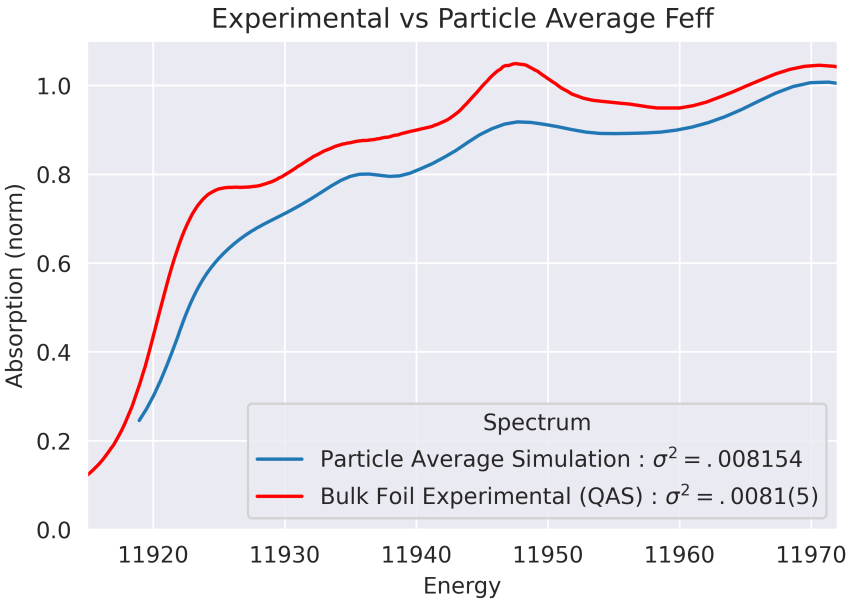

Figure 2.8: Comparing the bulk foil (red) measurement to a simulated large, bulk-like nanoparticle with the same disorder. The FEFF-

Chapter 3

Machine Learning

In order to predict disorder from XANES spectra, we rely on machine learning, a technique capable of building highly non-linear models from a large collection of data. Due to the integral part of machine learning, specifically deep learning, this chapter serves as a basis for defining, first, how neural netorks fundementally work, and then the specific implimentations used in the project.

First we clarify the terms machine learning (ML), deep learning, and artificial intelligence (AI). Machine learning refers to the statistical technique of fitting a model to a large collection of data. These models can either be regressors or classifiers, the former being able to predict a continuous range of values and the latter being a discrete predictor. Neural networks are one example of a machine learning model that tends to be computationally intensive. They are useful for creating highly non-linear models capable of making difficult predictions and solving difficult tasks. Neural networks were inspired by biological processing systems, and the graphical representation includes common terms such as "nodes" and "connections." The field of ML involving ANNs with many layers is referred to as deep learning [16]. AI is a subfield of deep learning where a neural network is trained to generate human-like responses, such as a chatbot or a virtual assistant.

With these definitions, to predict disorder from XANES we utilize deep learning to train a regression-based neural network. The following sections will walk through the mathematical process of training a simple neural network. In practice, sophisticated APIs such as Google's TensorFlow or Facebook's PyTorch handle the mathematical backend; however, one should fundamentally understand what these frameworks are doing.

3.1 Feedforward and Backpropagation in ANNs

Understanding how a neural network makes a prediction requires a solid grasp of linear algebra. The process where a NN passes information from the input to the subsequent layers to make a prediction is called the feedforward process. The name comes from the operation where each layer of the NN passes information to the next layer until the reaching output layer. The process of updating the weights of the NN is called backpropagation and relies

principally on vector calculus. Neither action is particularly mathematically complicated; however, there are so many parts that it is easy to get lost in the sea of similar-sounding partial derivatives. In this next section, we explicitly walk through the math for the feedforward process of a fully connected (affine) neural network.

3.1.1 Feedforward

Consider the neural network in figure 3.3 with an input layer of 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 a row vector x_μ. Each edge in the graph represents a weight that is multiplied with the connected node on the left. Each node in the input layer is multiplied by the weight of the connected edge and added together. This operation for all input nodes and weights can thus be represented as a dot product between the input layer row vector and a weight matrix. The hidden layer (first layer) has a cardinality of $\nu = 5$. Thus, the resulting dot product is $x_\mu W_\nu^{\mu(1)}$ where $W_\nu^{\mu(1)}$ represents the matrix of weights to create the first layer. While this result has the correct dimensionality for the hidden layer, there are still two operations reuired to produce the actual values for the nodes $h_\nu^{(1)}$. First, a small trainable parameter, b_ν is added to every value from the previous calculation. The values in this row vector are called biases and are introduced to prevent overfitting —i.e. the phenomenon where a model predicts the training data well, but is unable to generalize and predict un-seen data. Biases are a regularization parameter. Regularization techniques are discussed below. 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. Without an activation function, a neural network 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 activation function is defined as the following:

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

The sigmoid activation function maps the input between zero and one. Hence, sigmoid activation functions are often used in the final layer to output a probability. The sigmoid approaches 1 and -1 around $x = 4$ and $x = -4$ respectively, meaning that the sigmoid activation function is only useful within that limited range. One issue with both these activation functions is the potential for creating a vanishing gradient. The gradient of either of these functions appoaches zero for values above 4. This hurts the ability for the NN to learn how to meaningfully update its trainable parameters. The importance of calculating gradiets will be discussed in next section in the context of backpropagation.

[1]Recall that in Einstein notation repeated indices are implicitly summed over.

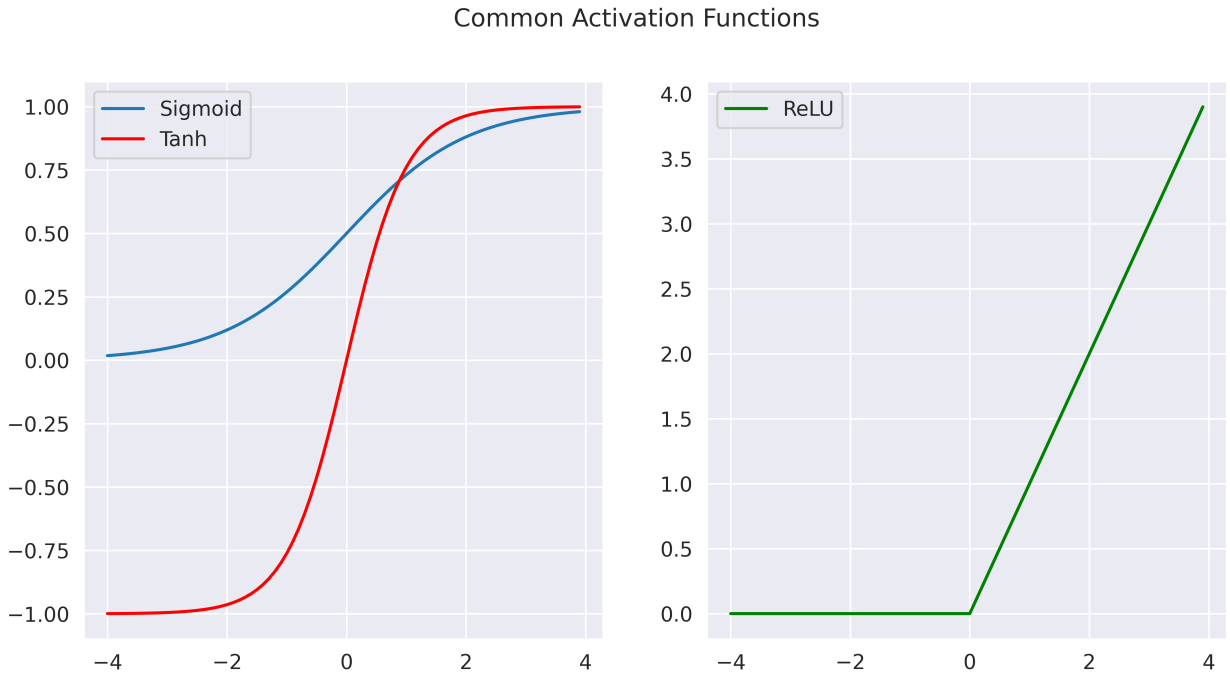

Figure 3.1: Plotted are three common activation functions: sigmoid, tanh, and ReLu.

ReLU

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

$$f(x) = \begin{cases} 0 & \text{if } x <= 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.2}$$

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

With the input nodes dotted with the weights, the baises added, and then the activation function calcuated for each node, we arrive at the final final for the first hidden layer. Mathematically, $h_\nu^{(1)} = \sigma\left(x_\mu W_\nu^\mu + b_\nu\right)$. This process is now repeated, only with the starting layer $h_\nu^{(1)}$ and the output $\hat{y} = \sigma\left(h_\nu W_\kappa^\nu + b_\kappa\right)$ 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.

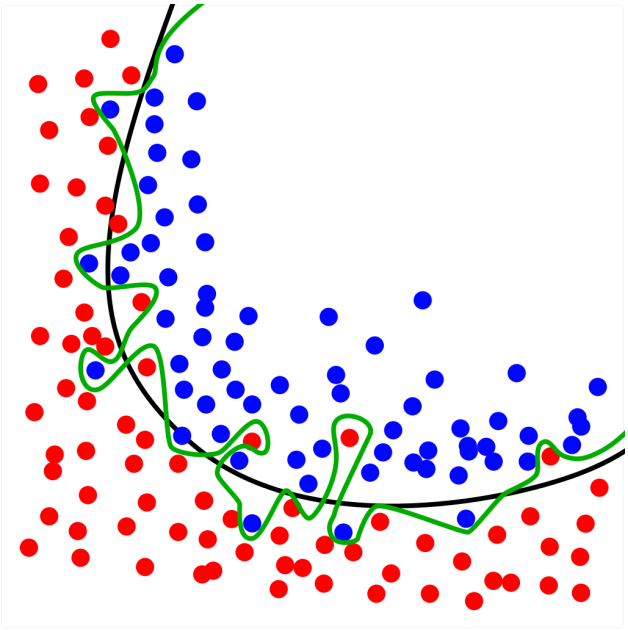

Figure 3.2: The green curve represents a model that is overfitting the data in a binary classification problem. Although it makes perfect predictions for the training data in this figure, the model will not generalize well. Introducing biases and including dropout layers in neural networks are strategies to prevent overfitting.

3.1.2 Loss Metrics and Regularization

It is necessary to evaluate the quality of every prediction the neural network makes in order to update the netork parameters. The measure of the error in one prediction is referred to as the *loss*, and the summed total of all the errors in predictions is called the *cost*. For regression problems, the two most common cost functions are the mean squared error and the mean absolute error. These are also referred to as the L2 and L1 costs, respectively, and defined as:

$$\text{L1 Cost} = \frac{1}{n}\sum(\hat{y} - y) \tag{3.3}$$

$$\text{L2 Cost} = \frac{1}{n}\sum(\hat{y} - y)^2 \tag{3.4}$$

where n is the number of training sampels. The equation for updating the weights under L2 regularization is as follows:

$$W := (1 - \alpha\lambda)W - \frac{\partial J}{\partial W} \tag{3.5}$$

where α is the learning rate, λ is the regularization hyper-parameter, and J is the cost. L2 regularization is often referred to as weight decay. Every iteration the weights are pushed

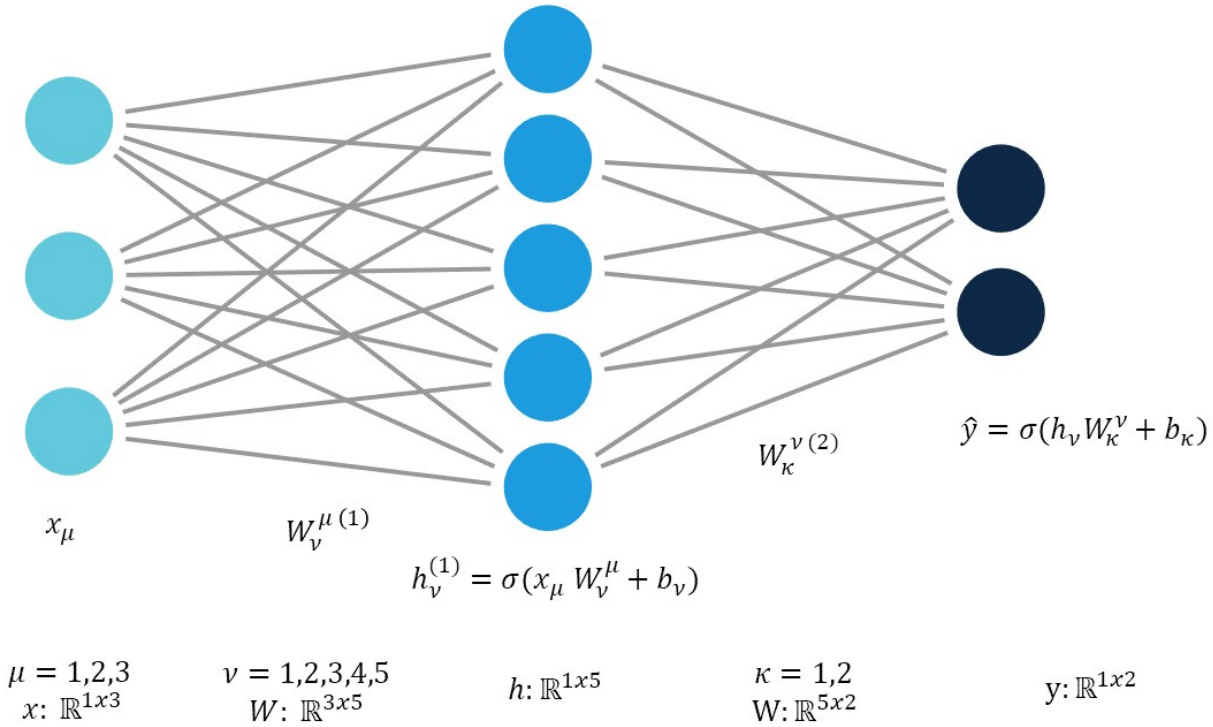

Figure 3.3: Figure first made by me for my website

closer to zero due to the multiplicatiion of the weights by a value < 1. For L2, the equation for the cost is:

$$\text{L2 cost} = \sum(y_i - y)^2 + \lambda \sum(W)^2 \tag{3.6}$$

L1 regularization is just the same as above, but with an absolute value for the regularization term instead of a square. 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 $abs(w)$ is a much stiffer penalty than w^2. Thus, L1 is thus a good choice when you have a ton of features.
The process can be repeated ad-nauseam for networks with more layers.

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 require repeated use of the chain rule. Representing the above neural network as a functional yields the equation:

$$\hat{y} = \sigma\left(h_\nu^{(1)}\left(x_\nu\right)\right) \tag{3.7}$$

where the 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 L2 loss (3.3). Note that the loss function is a function of the previous functional (3.7). To see how much to shift the weights, calcuate the gradients for each layer. The first partial derivative is trivial:

$$\frac{\partial J}{\partial J} = 1 \tag{3.8}$$

For the next layer (the output layer):

$$\frac{\partial \hat{y}}{\partial J} = \frac{\partial J}{\partial J}\frac{\partial \hat{y}}{\partial J} \tag{3.9}$$

For the next nested-function (the sigmoid):

$$\frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \sigma} \tag{3.10}$$

The next layer is the hidden layer before the activation function. For simplicity it will be referred to as g where $g = x_\mu W_\nu^\mu + b_\nu$.

$$\frac{\partial J}{\partial g} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \sigma}\frac{\partial \sigma}{\partial g} \tag{3.11}$$

Next layer is the input layer, X_μ has no trainable parameters, so the process for this network architecture is complete. If there were more hidden layers, the chaining rule would continue by multiplying the gradient calculated in the previous step by the gradient of the next layer with respect to the previous layer (towards the input layer).

3.1.4 Concrete Example

Finally, we will repeat the backpropogation process of updating the weights for the previous example as explicitly as possible. Consider again the functional $\hat{y}$ in equation (3.7) and the L2 loss in equation (3.3), rewritten here as L . To determine how do shift the weights in the function, want to know $\frac{\partial J}{\partial W}$, where W is short for $W_\kappa^{\nu(2)}$, the weight matrix connected to the second layer (the output layer).

$$L = \frac{1}{m}\sum(\hat{y} - y)^2 \tag{3.12}$$

$$\tag{3.13}$$

As before, the first partial derivative is trivial

$$\frac{\partial J}{\partial J} = 1 \tag{3.14}$$

and the next partial derivative is also strait-foward:

$$\frac{\partial J}{\partial \hat{y}} = 2(\hat{y} - y) \tag{3.15}$$

Applying the chain rule yields:

$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial J}{\partial J}\frac{\partial J}{\partial \hat{y}} = 1 \cdot 2(\hat{y} - y) \tag{3.16}$$

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

$$\frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \sigma} \tag{3.17}$$

We already found the first term (3.16), and the second term in trivial.

$$\frac{\partial \hat{y}}{\partial \sigma} = 1 \implies \frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \sigma} = 2(\hat{y} - y) \cdot 1 \tag{3.18}$$

At this point, we have the derivative $\frac{\partial J}{\partial \sigma}$ 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 $\frac{\partial J}{\partial g}$ where $g = h_\nu W_\kappa^\nu + b_\kappa$ as before. Continuing the chain,

$$\frac{\partial J}{\partial g} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \sigma}\frac{\partial \sigma}{\partial g} \tag{3.19}$$

where

$$\sigma(g) = \frac{1}{1 + e^{-g}} \tag{3.20}$$

$$\implies \frac{\partial \sigma}{\partial g} = \sigma(g)(1 - \sigma(g)) \tag{3.21}$$

Combining these previously calculated terms yields:

$$\frac{\partial J}{\partial f} = 2(\hat{y} - y) \cdot 1 \cdot \sigma(z)(1 - \sigma(z)) \tag{3.22}$$

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

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

The last partial in the chain is

$$\frac{\partial g}{\partial W} = W \tag{3.24}$$

So,

$$\frac{\partial J}{\partial W} = 2(\hat{y} - y) \cdot 1 \cdot \sigma(z)(1 - \sigma(g)) \cdot W \tag{3.25}$$

For the baises,

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial g} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial g} \frac{\partial g}{\partial b} \tag{3.26}$$

$$\frac{\partial g}{\partial b} = 1 \tag{3.27}$$

$$\implies \frac{\partial J}{\partial b} = 2(\hat{y} - y) \cdot 1 \cdot \sigma(g)(1 - \sigma(g)) \cdot 1 \tag{3.28}$$

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 (3.26) is just the partial derivative for one term in the weight matrix or bias vector. Repeating the process for each term in the matrix W and vector b yields the gradients $\nabla_w L$ and $\nabla_B L$, 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 a function points in the direction of greatest 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, $\nabla_w L$. 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. Several improvements have been made to gradient descent since its inception [17], with the state of the art being AdamW [18]. In this section, several optimization algorithms are introduced in order to provide context for Adam, the optimizer used for training our model. In the previous section, the gradients for the weights and biases were written explicitely. For simplicity, the variable θ is introduced to refer to either parameter. As before, $J(\theta)$ is the cost function; it could be the L1 or L2 cost, or any other differentiable measurement of fit quality.

Gradient Descent

Vanilla gradient descent [19] updates the parameters in the following way:

$$\theta := \theta - \eta \cdot \nabla_\theta J(\theta) \tag{3.29}$$

Here (3.29), 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. 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 [20]. 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 [21]. 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 [22] 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[2]. 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 .

$$V_t = \beta V_{t-1} + (1 - \beta) S_t \tag{3.30}$$

where $\beta \epsilon [0, 1]$, a hyperparameter which partly defines how much weight the previous $1/(1-\beta)$ terms of S contribute[3]. 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:

$$V_t = \beta V_{t-1} + (1 - \beta) \nabla_\theta J(\theta) \tag{3.31}$$

$$w = W - \alpha V_t \tag{3.32}$$

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

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

[3]Typically 0.90 is a good starting point

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 [23].

Root Mean Squared Propagation

Root Mean Squared Propogation (RMSprop)[4] is another variant of SGD designed to improve convergence speed and remedy adagrad's [24] tendency to rapidly diminishing gradients [25][26]. 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.

$$S_{k+1} = \beta S_k + (1-\beta)(\nabla_\theta J \cdot \nabla_\theta J) \tag{3.33}$$

$$\theta_{k+1} = \theta_k - \alpha \frac{\nabla_\theta J}{\sqrt{S_{k+1}} + \epsilon} \tag{3.34}$$

The hyperparameter ϵ is included in the denominator to prevent a possible division by zero.[5]

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).

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)(\nabla_\theta J) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J \cdot \nabla_\theta J) \end{aligned} \tag{3.35}$$

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 ciations in the 6 years since its publication gives some indication of the importance and power of this algorithm [27]. This was the chosen algorithm for training our neural network for predicting disorder in XANES.

[4]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.

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

3.3 Batch normalization

Batch normalization is a way to make your network more robust to covariance shift[6] [28] [29]. For example, if you train your cat vs. not cat identifier with only images of black cats, your network won't make good predictions when it comes to orange cats.

The idea is to normalize each hidden layer, similar to how you normalize the input dataset. However, whereas with normalization of training data centers the dataset around a mean of zero and STD of 1, the mean and variance of the batch normalization are trainable/learnable parameters.

Normalize input data i like:

$$Z_{norm}^{(i)} = \frac{x_i - \mu}{\sqrt{\sigma^2 - \epsilon}} \tag{3.36}$$

Normalize the values for hidden layer l like:

$$\widetilde{Z}_i = \gamma Z_{norm}^{(i)} - \beta \tag{3.37}$$

You can see that if $\sqrt{\sigma^2 + \epsilon}$ and $\gamma = \mu$, you get the first equation, i.e. you're normalizing the hidden layer in the same way as the input layer. You generally don't want to do this, however, because if you're normalizing everything to be centered around zero, your activation function will be mostly focused on the linear regime of the sigmoid.

The structure of implementing batch normalization looks something like this:

$$\text{first pass: } x \cdot \theta^T \rightarrow z$$

$$\text{batch normalize: } z \rightarrow \widetilde{z}$$

$$\text{apply activation function: } g(\widetilde{z}) = a$$

$$\text{second pass: } a \cdot \theta^T$$

etc.

Note, this is for one mini-batch, so x is the i^{th} mini-batch. Normalizing the hidden layers for each batch means that later hidden layers don't have to adapt as much to the earlier hidden layers. This means that the later layers can do a better job tuning themselves a little more independently of the other layers, and it speeds up the learning process. Note, because you're scaling each mini-batch ($z \rightarrow \widetilde{z}$), you add a little bit of noise, which is a little bit of regularization. But don't use batch-norm as a form of regularization. That's not its purpose.

[6] Covariate shift is a type of dataset shift where the distribution of training data differs from that of testing, or in this context, batch to batch.

3.4 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 [30]. 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.

$$f \otimes g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{3.38}$$

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 [31] [32]. 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 foruma is required. The convolution of the signal f with the kernel g can be written as [33]:

$$f \otimes g = \sum_{j=1}^{m} g(j) \cdot f(i-j+m/2) \tag{3.39}$$

Here (3.39), 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).

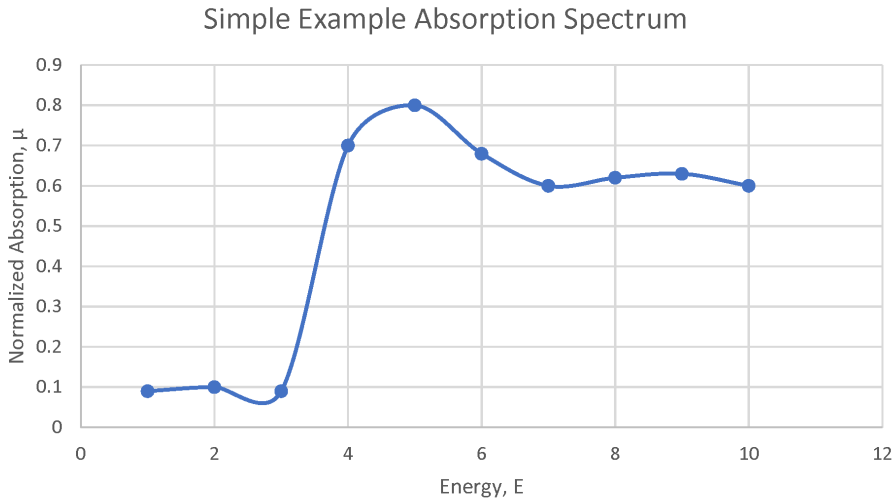

Figure 3.4: A simple abosrption spectrum for demonstration purposes.

Each data point, (E, μ) in the spectrum is described as a feature vector, where the feature is the the energy value for a given point (E, μ). The vector is depicted in (**??**) with zeros padded on both sides. Additionally, consider a kernel, g

$$f = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & .09 & .10 & .09 & .70 & .80 & .68 & .60 & .62 & .63 & .60 & 0 \\ \hline \end{array}$$

$$g = \begin{array}{|c|c|c|} \hline .1 & .1 & .1 \\ \hline \end{array}$$

The convolution works by mutliplying 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:

$$h = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline .019 & .028 & .089 & .159 & .218 & .208 & .190 & .185 & .185 & .123 \\ \hline \end{array}$$

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 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 downsampleing 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" [34].

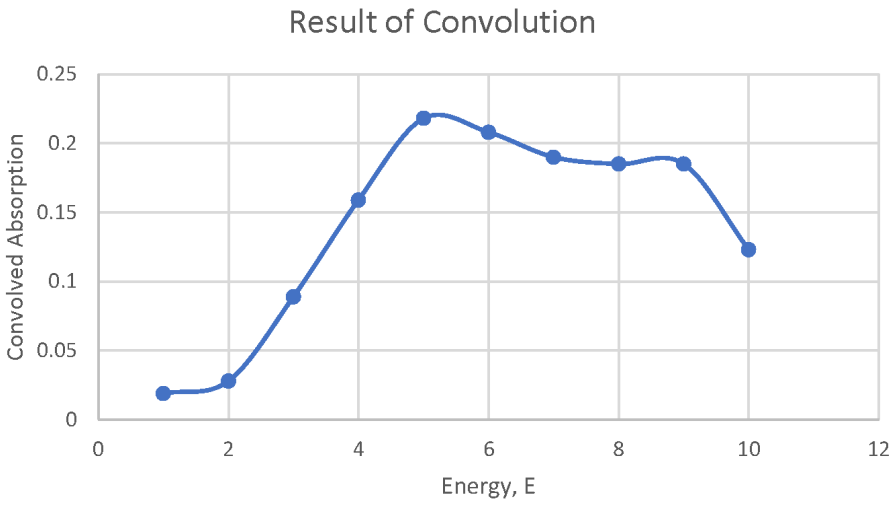

Figure 3.5: The result of the 1D convolution of kernel $g = (.1, .1, .1)$ on the spectrum in Figure 3.4.

Another important possible change, 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 of one. The stride could be any integrer less than the length of the input vector f, though in practice, typically a 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 funciton. The default in Keras is Glorot Uniform. The values of each filter are trainable parameters which evolve to produce the best final prediction given the subsequent layers in the neural network.

3.5 How to Train a Neural Network

Here is the process for building a NN for cifar10Permalink. You need to start with a simple model first. Pick some hype hyperparameters to see if you can overfit it (say, make the training set 1 - 10 images). If you can overfit it, then it means your code is working and your architecture makes sense. Next you can up the images to say 1000 and run a broad hyperparameter search. Afterward, run maybe 20 epochs and see how the training and validation loss are moving. If they both are going down, your architecture looks good. If not, start over.

Once it looks like your training loss is continuing to decrease but your validation loss plateaus, it's time to start introducing regularization (say nn.Dropout(.5)) and data augmentation. This way you won't continue to overfit the data and can keep dropping the validation loss.

I haven't done it yet with the cifar10 set, but I suspect that you should build up a more complex architecture without regularization and make sure you can overfit it to death before tuning hyperparameters and introducing regularization.

If you start with a complex model with data augmentation and regularization you'll never be able to tune the hyperparameters and find a good solution.

It's best to put dropout layers after a ReLU activation and before an affine (nn.Linear()) layer

Chapter 4

Results

Here I expect to showcase lots of nice figures and data to show how well the neural network works

4.1 temp

Appendix A

An appendix

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
│       │   ...
```

A.3 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.

A.4 nn.ipynb

The neural network, a Jupyter notebook.

A.5 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 dataloading, plotting, and linear interpolation of experimental data on the same energy mesh used for the training sample.

Bibliography

[1] J. Timoshenko, D. Lu, Y. Lin, and A. I. Frenkel, The Journal of Physical Chemistry Letters **8**, 5091 (2017).

[2] F. R. Elder, A. M. Gurewitsch, R. V. Langmuir, and H. C. Pollock, Phys. Rev. **71**, 829 (1947), URL https://link.aps.org/doi/10.1103/PhysRev.71.829.5.

[3] D. J. Gardenghi et al., Ph.D. thesis, Montana State University-Bozeman, College of Letters & Science (2012).

[4] H. Fricke, Physical Review **16**, 202 (1920).

[5] G. Hertz, Zeitschrift fuer Physik **3**, 19 (1920).

[6] J. J. Rehr and R. C. Albers, Reviews of modern physics **72**, 621 (2000).

[7] M. Newville, Reviews in Mineralogy and Geochemistry **78**, 33 (2014).

[8] K. Klementev, arXiv preprint physics/0003086 (2000).

[9] D. B. K. Teo (1986).

[10] M. Rühle and M. Wilkens, in *Physical Metallurgy (Fourth Edition)*, edited by R. W. Cahn and P. Haasenae (North-Holland, Oxford, 1996), chap. 11, pp. 1033–1113, fourth edition ed., ISBN 978-0-444-89875-3, URL https://www.sciencedirect.com/science/article/pii/B9780444898753500168.

[11] J. Timoshenko, A. Anspoks, A. Cintins, A. Kuzmin, J. Purans, and A. I. Frenkel, Physical review letters **120**, 225502 (2018).

[12] D. Uzio and G. Berhault, Catalysis Reviews **52**, 106 (2010).

[13] M. Jørgensen and H. Grönbeck, Topics in Catalysis **62**, 660 (2019).

[14] A. Azzalini and A. Capitanio, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **61**, 579–602 (1999), ISSN 1467-9868, URL http://dx.doi.org/10.1111/1467-9868.00194.

[15] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al., Nature Methods **17**, 261 (2020).

[16] J. Schmidhuber, Neural networks **61**, 85 (2015), URL `https://arxiv.org/abs/1404.7828`.

[17] A. Cauchy et al., Comp. Rend. Sci. Paris **25**, 536 (1847).

[18] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, Conference Paper at ICLR 2019 (2019), `1711.05101`, URL `https://arxiv.org/pdf/1711.05101.pdf`.

[19] S. Ruder, CoRR **abs/1609.04747** (2016), `1609.04747`, URL `http://arxiv.org/abs/1609.04747`.

[20] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, in *NIPS* (Citeseer, 2010), vol. 4, p. 4.

[21] D. R. Wilson and T. R. Martinez, Neural networks **16**, 1429 (2003).

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, nature **323**, 533 (1986).

[23] N. Qian, Neural networks **12**, 145 (1999).

[24] J. Duchi, E. Hazan, and Y. Singer, Journal of machine learning research **12** (2011).

[25] C. Igel and M. Hüsken, in *Proceedings of the second international ICSC symposium on neural computation (NC 2000)* (Citeseer, 2000), vol. 2000, pp. 115–121.

[26] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, arXiv preprint arXiv:1705.08292 (2017).

[27] D. P. Kingma and J. Ba, arXiv preprint arXiv:1412.6980 (2014).

[28] S. Ioffe and C. Szegedy, in *International conference on machine learning* (PMLR, 2015), pp. 448–456.

[29] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, in *Proceedings of the 32nd international conference on neural information processing systems* (2018), pp. 2488–2498.

[30] M. L. Boas, *Mathematical methods in the physical sciences; 3rd ed.* (Wiley, Hoboken, NJ, 2006), URL `https://cds.cern.ch/record/913305`.

[31] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, Mechanical Systems and Signal Processing **151**, 107398 (2021), ISSN 0888-3270, URL `https://www.sciencedirect.com/science/article/pii/S0888327020307846`.

[32] W. Rawat and Z. Wang, Neural computation **29**, 2352 (2017).

[33] R. Zabih, Cornell University (2013), cornell University, URL `https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf`.

[34] O. S. Kayhan and J. C. v. Gemert, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 14274–14285.