

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 15, 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

Abstract	i
Executive Summary	ii
Acknowledgments	iii
1 Introduction	1
1.1 X-ray Absorption Spectroscopy	1
1.1.1 XAFS	2
1.1.2 EXAFS	3
1.1.3 XANES	4
1.2 Disorder in XAFS	5
1.3 Nanoparticles?	5
1.4 Machine Learning?	5
1.5 Outline of the Thesis	5
2 Simulating Disorder	6
2.1 Generating Distortion Not Disorder	6
2.2 Generating Disorder via Probability Distribution Averaging	8
2.3 Simulation vs. Experimental Data	11
2.4 Particle-Averaged FEFF Simulated Disordered Structures	13
3 Machine Learning	15
3.1 Feedforward and Backpropagation in ANNs	15
3.1.1 Feedforward	16
3.1.2 Loss Metrics and Regularization	18
3.1.3 Backpropagation	19
3.1.4 Concrete Example	20
3.2 Optimizers	22
3.3 Batch normalization	23
3.4 How to Train a Neural Network	24

4	Results	26
4.1	temp	26
A	An appendix	27

List of Figures

2.1	2D Distortion	7
2.2	FEFF Simulations Results	9
2.3	Simulated Spectrum Gaussian Weighting	10
2.4	Simulated Disordered Spectrum Weightings	11
2.5	Simulation vs. Experimental	12
2.6	Bulk-nanoparticle difference: Simulation vs. Experimental data	13
2.7	Simulation vs. Experimental 2	14
2.8	Simulation vs. Experimental 3	14
3.1	Activation-Functions	17
3.2	Overfitting	18
3.3	Neural Network Example	19

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, selecting photons of wavelength, λ at angle θ .

$$n\lambda = 2d \sin(\theta) \quad (1.2)$$

The secondary monochromater removes higher order harmonics that satisfy the Bragg equation ($2\lambda, 3\lambda$ etc.). Different wavelengths 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 absorption 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

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

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).

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

The coefficients α and β can be found via a simple regression on a spectrum 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].

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

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).

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

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 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. EXAFS only takes into account the local region around the absorber at a distance r_j from the backscattering atom.

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

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 photoelectron'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).

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

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.

1.1.3 XANES

The XANES region, or the near-edge region, contains the chemical and electronic structural information of the sample encoded 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 structural descriptors can be learned from the XANES spectrum. Specifically, their method enables the decoding of XANES spectra to obtain the coordination number of bi-metallic nanoparticles. In this 2017 paper, the group trained an artificial neural network (ANN) to recognize a relationship

¹Note, 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]

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.

1.2 Disorder in XAFS

1.3 Nanoparticles?

1.4 Machine Learning?

Probably want to talk about these papers in this section [11] [1].

1.5 Outline of the Thesis

Building off the historical context of measuring disordered nanoparticles in XANES spectra, this thesis begins with an in-depth description of

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) *Of just Au NPs? Can we generalize this to the sigma of the PRDF?*, 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-intensive and expensive. Rather, simulations provide a practical alternative, though even simulating each possible disordered structure individually, would be extremely time-intensive. Instead, we run many XANES simulations of simple, non-disordered structures and generate the neural network training data or 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 MSD. *Maybe give a brief outline of the process so you can understand the point of the sections as a walk-through before the very end*

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*. Whereas *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 distortion 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 on in the averaging process since the original coordinates will only be one structure out of

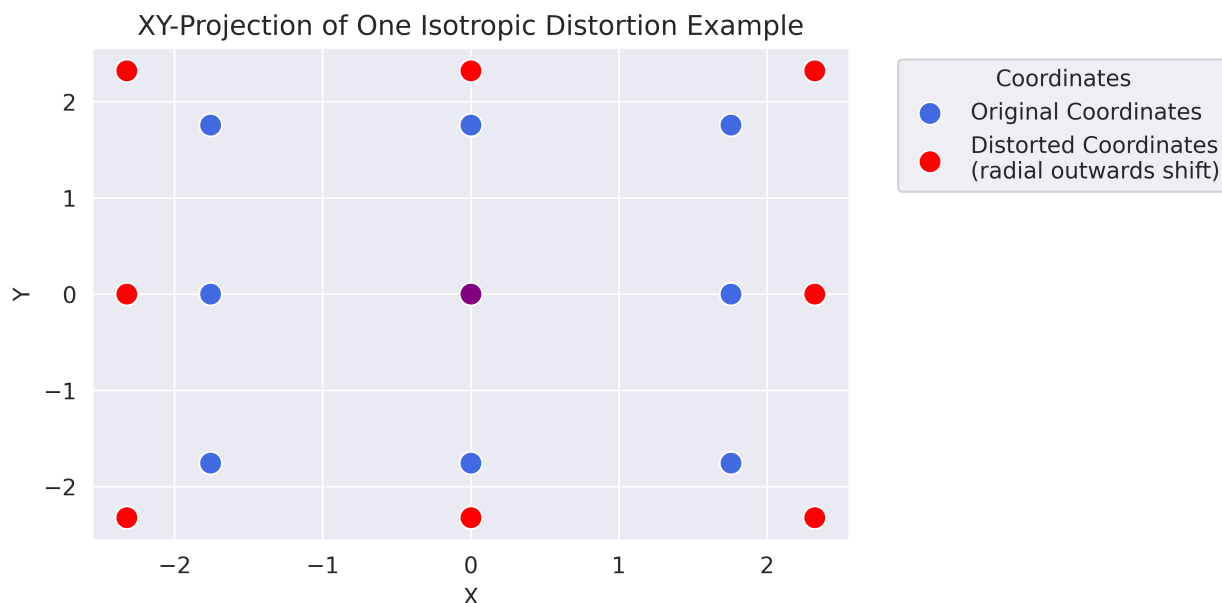


Figure 2.1: Each point represents an atom of first 12 nearest neighbors of a Au cluster projected onto the xy -plane. The four corner points actually represent two atoms because of the projection. The blue atoms represent the original coordinates, and the red atoms represent the radially shifted coordinates.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

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 \AA to $+0.45 \text{ \AA}$ in increments of 0.01 \AA . For example, the FEFF input file with the greatest inward shift has all coordinates shifted 0.45 \AA radially inwards towards the center absorber, and the FEFF input file with the largest outwards shift has coordinates shifted 0.45 \AA 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 minutes. 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):

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

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

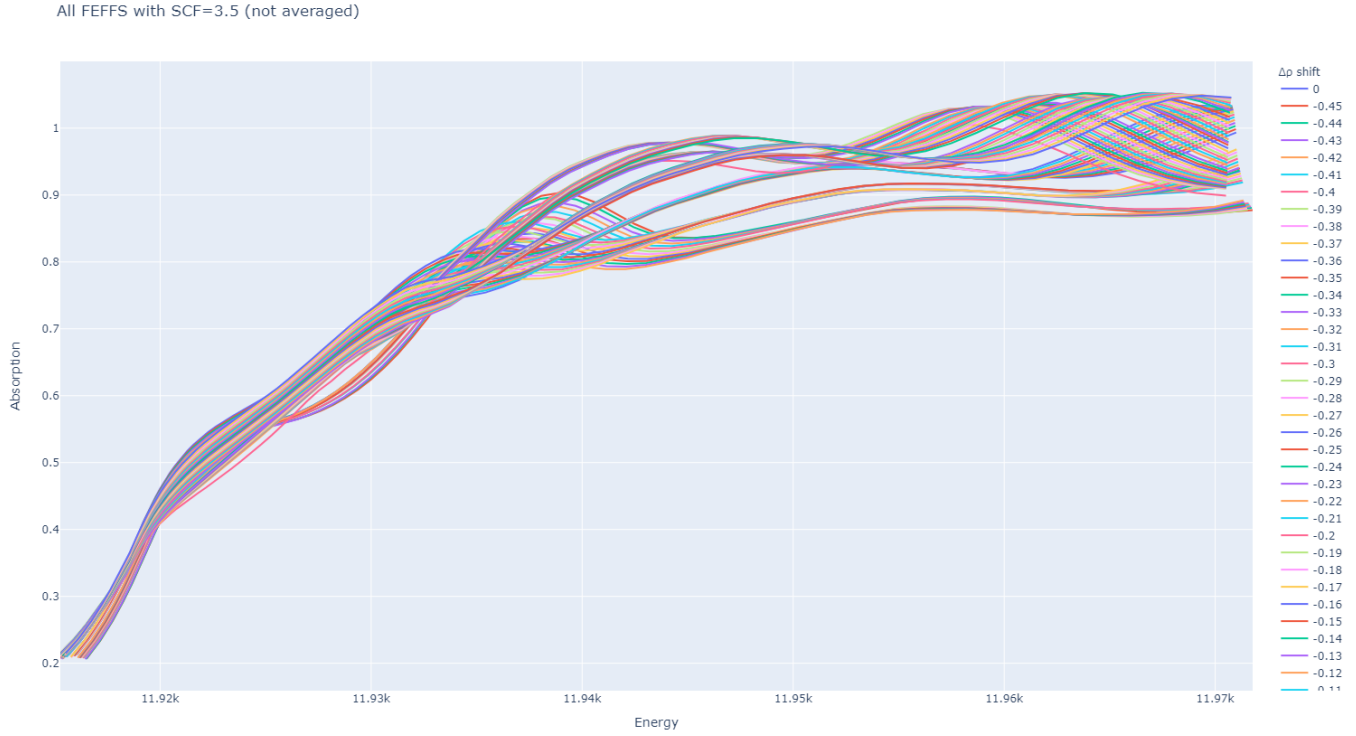


Figure 2.2: *TEMPORARY - way too much info. I'll select a few.* Each spectrum represents the FEFF simulation results for a different distorted structure. For each spectrum, the crystal structure and center absorber remain constant, the only parameter that varies is the euclidean distance from the center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

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

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

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

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

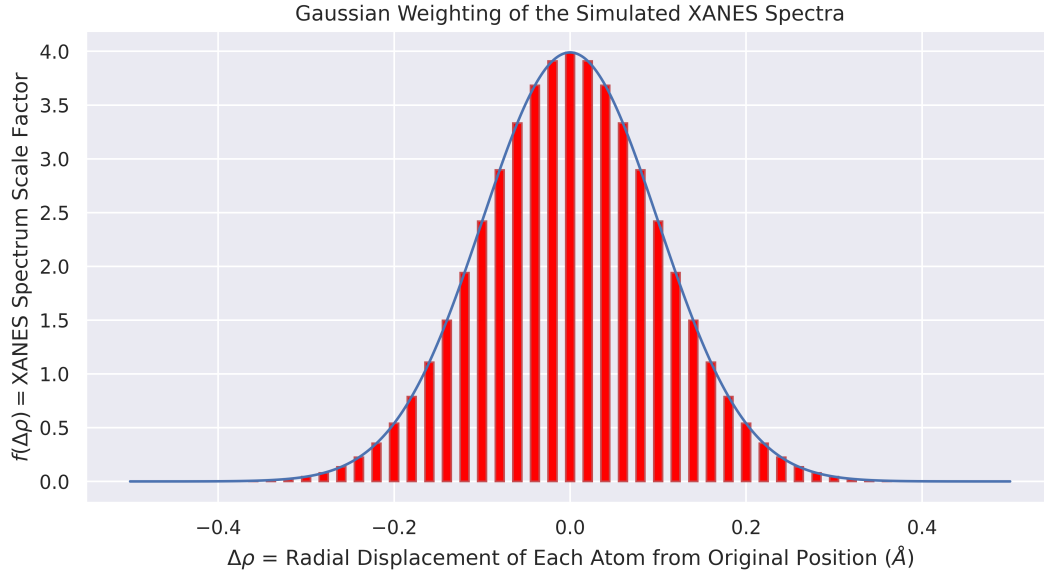


Figure 2.3: A Gaussian distribution probability density function can be used to calculate the relative weight of each FEFF generated XANES spectrum towards one simulated, disordered spectrum. Each bin (red bar) represents a FEFF generated spectrum; the x -axis is the isotropic shift of the atomic positions, 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) [12, 13], $f(x)$.

$$f(x) = 2\phi(x)\Phi(\alpha x) \quad (2.4)$$

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

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

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

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

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 distribution 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.

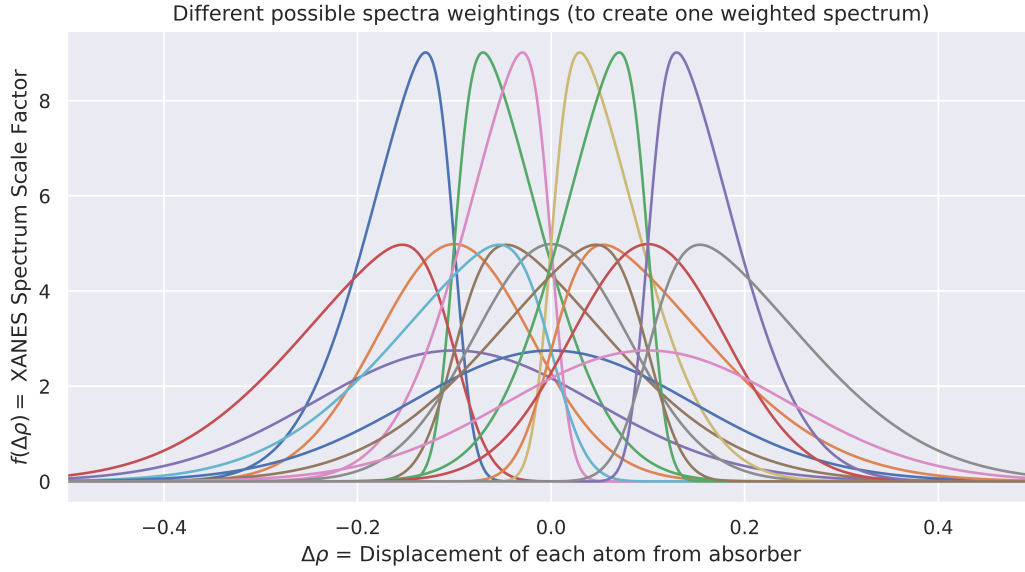


Figure 2.4: Eighteen skew-normal distributions plotted with all possible combinations of $\sigma = \{.08, .145\}$, $\mu = \{-.1, 0, .1\}$, and $\alpha = \{-5, 0, 5\}$. Each represents a possible way to produce a simulated, disordered spectrum from many FEFF-simulated, distorted spectra

The disorder of the skew-normal 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):

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

Here, $f(x)$ is the skew-normal 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 $\sigma^2 = 0.0081(5) \text{ \AA}^2$, and for the 8 nm disordered particle, $\sigma^2 = 0.0102(8) \text{ \AA}^2$. 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.

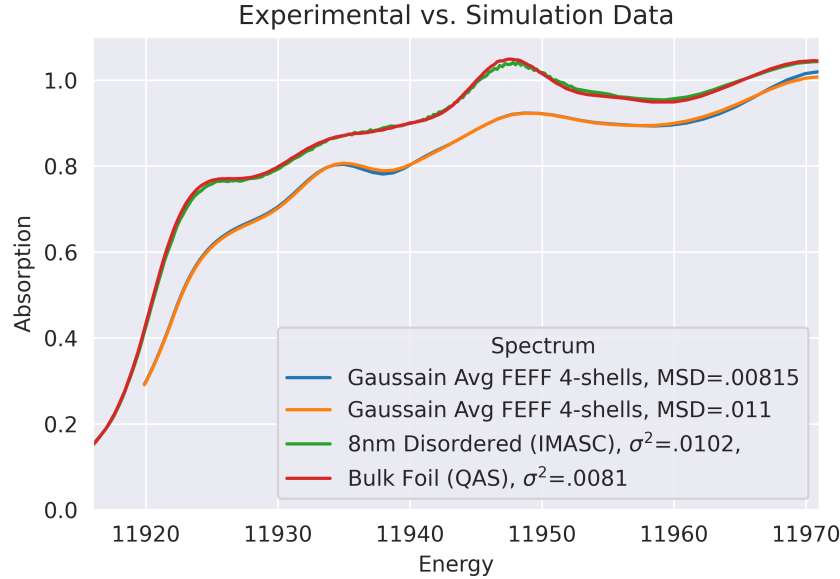


Figure 2.5: Comparing the bulk foil (red) measurement to the 8 nm disordered nanoparticle (green) measurement is an analog to comparing the simulated, non disordered FEFF spectrum (blue) to the simulated disordered spectrum (orange).

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.

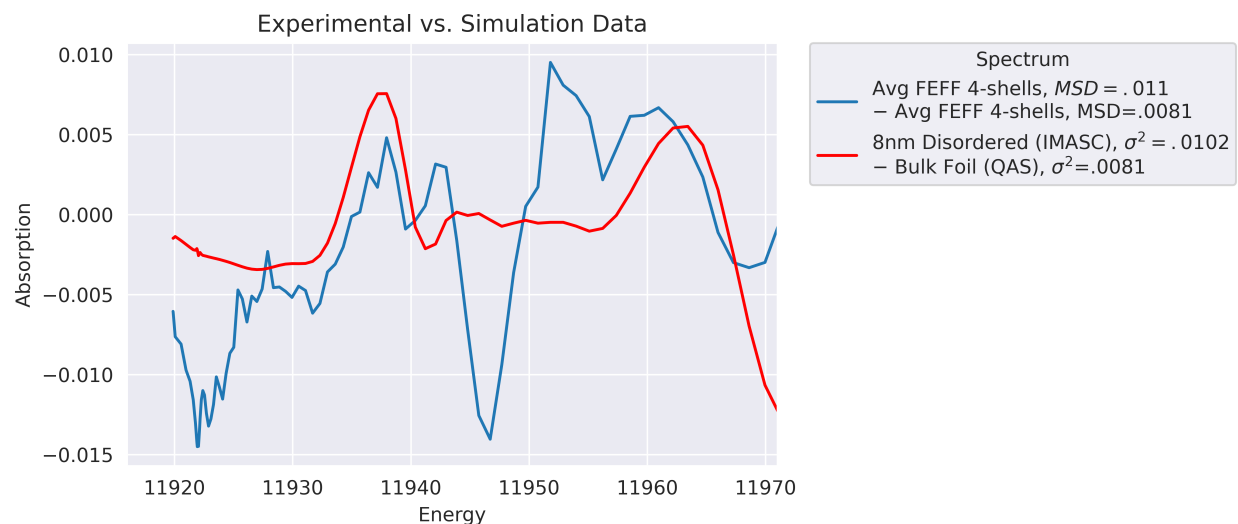


Figure 2.6: The difference between the nanoparticle spectrum and the bulk spectrum are plotted for the same data as in Figure 2.5. It is easier to see where the bulk and the nanoparticle absorption crisscross by plotting the difference.

2.4 Particle-Averaged FEFF Simulated Disordered Structures

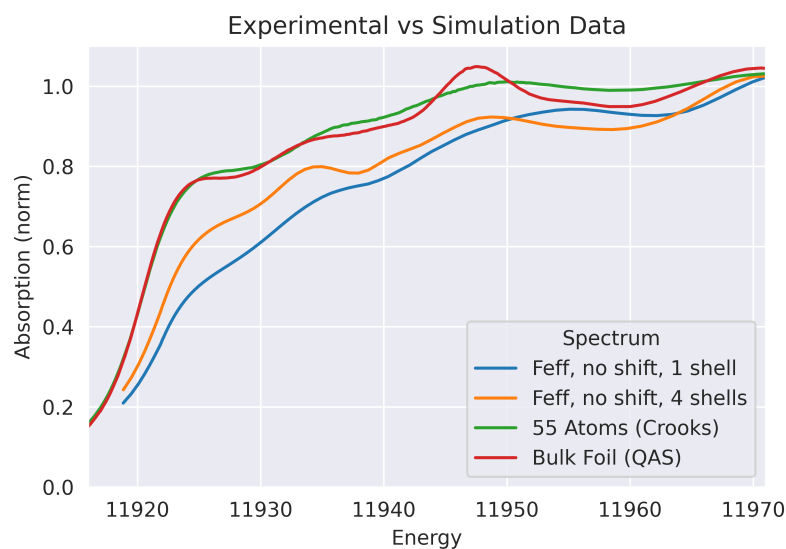


Figure 2.7: Comparing the bulk foil (red) measurement to the 55 atom nanoparticle (green) measurement is an analog to comparing the 13 atom simulated spectrum (blue) to the 55 atom simulated spectrum (orange).

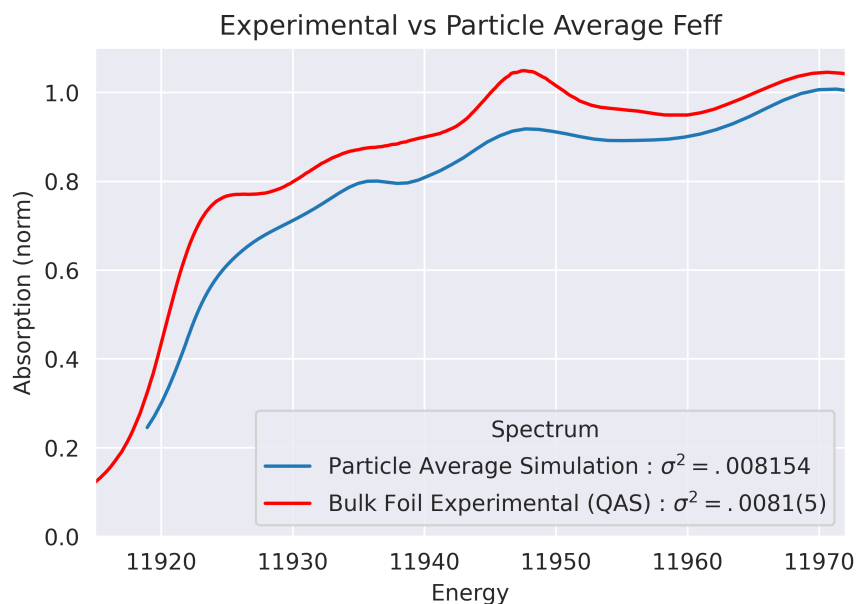


Figure 2.8: Comparing the bulk foil (red) measurement to a simulated large, bulk-like nanoparticle with the same disorder. The FEFF-

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 networks fundamentally work, and then the specific implementations 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 [14]. 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 notation¹ 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 required 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:

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

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 approaches 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 gradients will be discussed in next section in the context of backpropagation.

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

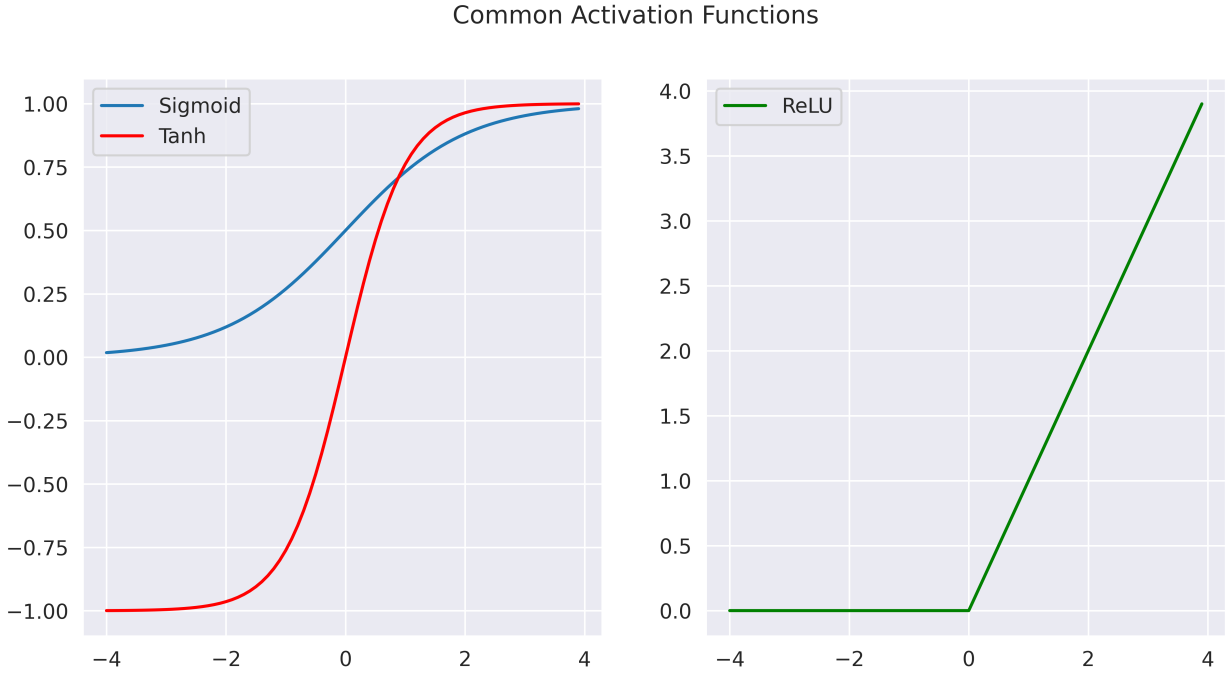


Figure 3.1: Plotted are three common activation functions: sigmoid, tanh, and ReLU.

ReLU

The **R**ectified **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:

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

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 calculated for each node, we arrive at the final final for the first hidden layer. Mathematically, $h_{\nu}^{(1)} = \sigma(x_{\mu}W_{\nu}^{\mu} + b_{\nu})$. This process is now repeated, only with the starting layer $h_{\nu}^{(1)}$ and the output $\hat{y} = \sigma(h_{\nu}W_{\kappa}^{\nu} + b_{\kappa})$ 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.

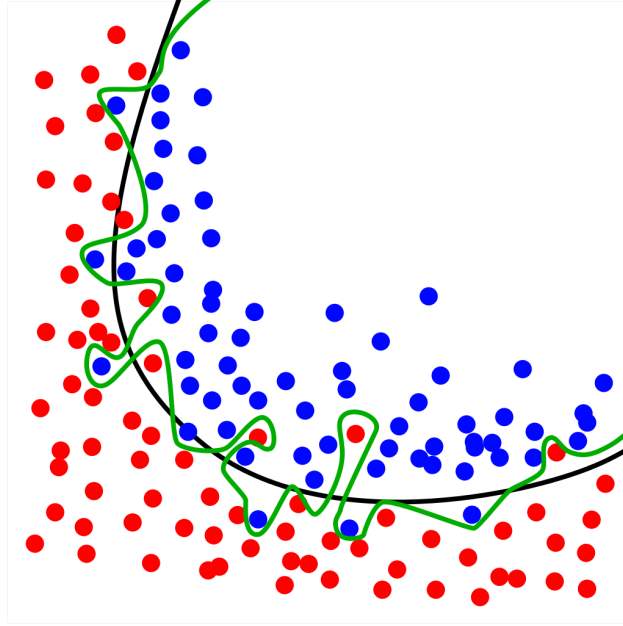


Figure 3.2: The green curve represents a model that is overfitting the data in a binary classification problem. Although it makes perfect predictions for the training data in this figure, the model will not generalize well. Introducing biases and including dropout layers in neural networks are strategies 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 network parameters. The measure of the error in one prediction is referred to as the *cost*, and the summed total of all the errors in predictions is called the *loss*. 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 losses, respectively, and defined as:

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

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

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

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

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

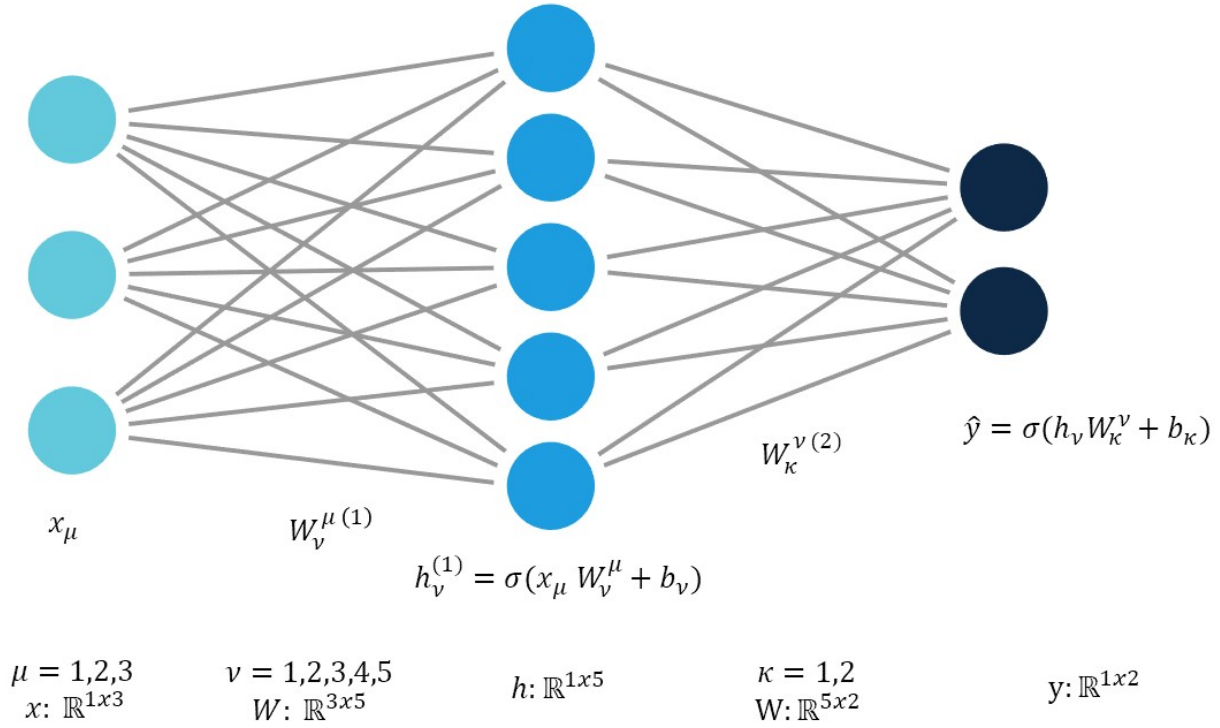


Figure 3.3: Figure first made by me for my website

closer to zero since you're multiplying the weights by a number < 1 . For L2, the equation for the total cost (loss) is:

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

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 $\text{abs}(w)$ 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 derivatives require repeated use of the chain rule. Representing the above neural network as a functional yields the equation:

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

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, calculate the gradients for each layer. The first partial derivative is trivial:

$$\frac{\partial L}{\partial L} = 1 \quad (3.8)$$

For the next layer (the output layer):

$$\frac{\partial \hat{y}}{\partial L} = \frac{\partial L}{\partial L} \frac{\partial \hat{y}}{\partial L} \quad (3.9)$$

For the next nested-function (the sigmoid):

$$\frac{\partial L}{\partial \sigma} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \quad (3.10)$$

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

$$\frac{\partial L}{\partial g} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial g} \quad (3.11)$$

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 simply 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).

The pattern should be clear. You multiply the gradient calculated in the previous layer by the partial the next layer w.r.t the previous layer.

3.1.4 Concrete Example

Finally, we will repeat the process above as explicitly as possible. Consider again the functional \hat{y} in equation (3.7).

$$\hat{y} = \sigma(f(x)) \quad (3.12)$$

$$z = f(x) = W \cdot x + b \quad (3.13)$$

$$\sigma = \frac{1}{1 + e^{-z}} \quad (3.14)$$

The sigmoid function is applied element-wise to the values calculated in $f(x)$. I tried to indicate this by using z . As before:

$$Loss = L = \frac{1}{m} \sum (\hat{y} - y)^2 \quad (3.15)$$

$$\frac{\partial L}{\partial \hat{y}} = 1 \quad (3.16)$$

What we want to know how to shift the weights of the functions. We want to know $\frac{\partial L}{\partial W}$, but it'll take some work to get there. Now, explicitly,

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y) \quad (3.17)$$

So,

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial L}{\partial L} \frac{\partial L}{\partial \hat{y}} = 1 \cdot 2(\hat{y} - y) \quad (3.18)$$

$$\frac{\partial L}{\partial \sigma} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \quad (3.19)$$

We already found the first part, but how about the second term in the chain? Well,

$$\frac{\partial \hat{y}}{\partial \sigma} = 1 \implies \frac{\partial L}{\partial \sigma} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} = 2(\hat{y} - y) \cdot 1 \quad (3.20)$$

Okay, now, let's go a layer deeper.

$$\frac{\partial L}{\partial f} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial f} \quad (3.21)$$

So how about this next chain?

$$\frac{\partial \sigma}{\partial f} = \sigma(z)(1 - \sigma(z)) \quad (3.22)$$

where z is the value of $f(x)$. So,

$$\frac{\partial L}{\partial f} = 2(\hat{y} - y) \cdot 1 \cdot \sigma(z)(1 - \sigma(z)) \quad (3.23)$$

Now we're up to the good part. The parameters we want to "wobble" to change our network are the weights and biases (W and b). So,

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial f} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial f} \frac{\partial f}{\partial W} \quad (3.24)$$

This last chain

$$\frac{\partial f}{\partial W} = x \quad (3.25)$$

So,

$$\frac{\partial L}{\partial W} = 2(\hat{y} - y) \cdot 1 \cdot \sigma(z)(1 - \sigma(z)) \cdot x \quad (3.26)$$

And also,

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial f} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial f} \frac{\partial f}{\partial b} \quad (3.27)$$

$$\frac{\partial f}{\partial b} = 1 \quad (3.28)$$

$$\frac{\partial L}{\partial b} = 2(\hat{y} - y) \cdot 1 \cdot \sigma(z)(1 - \sigma(z)) \cdot 1 \quad (3.29)$$

This was just the process for a neural network with one hidden layer and a sigmoid activation function, but one can see how you could continue the process if you had more than one hidden layer. For example, after $\frac{\partial L}{\partial f}$ you would chain $\frac{\partial f}{\partial g}$ where g is the next hidden layer (f and g are swapped here with how I wrote it in the first part). If there were an activation function for this layer, which there often is, instead of $\frac{\partial f}{\partial g}$, you would have to chain another $\frac{\partial f}{\partial \sigma}$ first.

3.2 Optimizers

Adaptive moment Estimator

Adam is a combination of RMS prop and SGD with momentum. It uses the squared gradients to scale the learning rate (like RMS prop), and it uses a moving average of the gradient (like SGD with momentum).

RMS Propagation

RMS prop is another variant of SGD to improve convergence speed. Once again, the idea is to dampen oscillations in directions where you are close to the minimum, and accelerate movement when you are far from the minimum. Here, we keep a moving average of the squared gradients for each weight. As before, $\nabla_{\theta} L$ is the gradient of the cost with respect to weights.

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

$$\theta_{k+1} = \theta_k - \alpha \frac{\nabla_{\theta} L}{\sqrt{S_{k+1}} + \epsilon} \quad (3.31)$$

Stochastic Gradient Descent with Momentum

Compared to regular SGD, this can greatly reduce time to convergence. The general idea is add a fraction of the previous update to the current update. The exponential moving average is an averaging of points within a period that puts greater weight on more recent points (a simple moving average treats each point as equally significant). Here, V is the EMA.

$$V_1 = \beta V_0 + (1 - \beta) S_1 \quad (3.32)$$

$$V_2 = \beta V_1 + (1 - \beta) S_2 \quad (3.33)$$

$$\dots \quad (3.34)$$

where S is the sequence. EMA's are common in market forecasting, so often S is price at time t . β is defined from $[0,1]$, and usually a value around 0.90 is used. The continuous update for SGD with momentum, using Andrew Ng's notation, is as follows:

$$V_1 = \beta V_0 + (1 - \beta) \nabla_w L(W, X, y) \quad (3.35)$$

$$w = W - \alpha V_1 \quad (3.36)$$

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. SGD with momentum tends to work 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 a ravine in the right direction, where SGD tends to oscillate back and forth on in the short way. In my course's notation:

3.3 Batch normalization

Batch normalization is a way to make your network more robust to covariance shift². 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:

$$Z_{norm}^{(i)} = \frac{x_i - \mu}{\sqrt{\sigma^2 - \epsilon}} \quad (3.37)$$

²Covariate 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.

Normalize the values for hidden layer l like:

$$\tilde{Z}_i = \gamma Z_{norm}^{(i)} - \beta \quad (3.38)$$

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 mostly be focused on the linear regime of the sigmoid.

The structure of implementing batch normalization looks something like this:

$$\begin{aligned} \text{first pass: } & x \cdot \theta^T \rightarrow z \\ \text{batch normalize: } & z \rightarrow \tilde{z} \\ \text{apply activation function: } & g(\tilde{z}) = a \\ \text{second pass: } & a \cdot \theta^T \end{aligned}$$

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 ($z \rightarrow \tilde{z}$), 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.

3.4 How to Train a Neural Network

Here the process for building a NN for cifar10 [Permalink](#) 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. Afterwards, 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

Appendices are a good idea for almost any thesis. Your main thesis body will likely contain perhaps 40-60 pages of text and figures. You may well write a larger document than this, but chances are that some of the information contained therein, while important, does *not* merit a place in the main body of the document. This sort of content - peripheral clarifying details, computer code, information of use to future students but not critical to understanding your work ... - should be allocated to one or several appendices.

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] 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>.
- [13] 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).
- [14] J. Schmidhuber, Neural networks **61**, 85 (2015), URL <https://arxiv.org/abs/1404.7828>.