UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DATA-EFFICIENT SURROGATE MODELS FOR HIGH-THROUGHPUT DENSITY FUNCTIONAL THEORY

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

SCIENTIFIC COMPUTING & APPLIED MATHEMATICS

by

Schuyler Krawczuk

June 2020

The Thesis of Schuyler Krawczuk
is approved:

Professor Daniele Venturi, Chair

Professor Qi Gong

Professor Marcella Gomez

Dean Quentin Williams
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures **v**

List of Tables **ix**

Abstract **x**

Acknowledgments **xi**

1 Density Functional Theory **1**
1.1 Quantum Physics Background 2
1.1.1 Bra And Ket Vectors 2
1.1.2 The Wave Function, Schrodinger Equation & The Hamiltonian . 2
1.2 Hohenberg-Kohn Theorems 4
1.3 Kohn-Sham Equations 6
1.3.1 Exchange-Correlation Functionals 8
1.4 Solving The Kohn-Sham Equations 10
1.4.1 The Eigenvalue Problem 10
1.4.2 Plane Wave Basis 12
1.4.3 Pseudopotential 14
1.4.4 Reaching Self-Consistency 15
1.5 Numerical Methods 16
1.5.1 Davidson Algorithm 17
1.5.2 Conjugate Gradient 18
1.5.3 Computational Benchmark 19

2 Neural Network Modeling For Density Functional Theory **21**
2.1 Input-Output Maps 21
2.2 Neural Networks 24
2.2.1 Motivation For Neural Networks 25
2.2.2 Activation Functions 26
2.2.3 Convolutional Neural Network 28
2.3 Assessing Accuracy 31
2.4 Training 33

2.4.1 Training Neural Networks & Backpropagation 40
2.4.2 Regularization Methods 42
2.5 Transfer Learning 43
2.6 Neural Networks For Modeling Atomic Systems 46
2.6.1 Atomistic Neural Networks 46
2.6.2 SchNet 49
2.6.3 Transfer Learning For Modeling Molecular Properties 51

3 Predicting Free Energy Of Transition Metal Oxides With SchNet 53
3.1 Data Set Creation 54
3.2 SchNet Direct Training Results 54
3.3 Transfer Learning With SchNet 55
3.3.1 Results 57
3.3.2 Error Analysis 58
3.4 Computational Cost Comparison 60
3.5 Summary 63

List of Figures

1.1 Top: From top left moving clockwise: plot of V_{eff}, V_{ext}, V_{XC}, and V_{Hartree} in a 2-dimensional slice of a Si structure. Spatial units are in Bohrs ($5.29 \times 10^{-11} m$) and potential is in Rydberg units ($13.6eV$). Bottom: The resulting electron density (in electrons per cubic Bohr) for the above potential. 9
1.2 Algorithm for performing a DFT calculation 11
1.3 Comparison of a potential and its wave function (solid lines) to the pseudopotential and its resulting wave function (dashed lines). The pseudopotential results in a smoother wave function that is identical to the true wave function outside of the core radius, r_c. 15
1.4 Distribution computational time of 1,300 DFT calculations. Calculations were done on a compute node with 2 Intel 2.1GHz Xeon E5-2620v4 processors using 16 cores and 64GB RAM. 19

2.1 Left: Linear regression. The red line minimizes the error function in Equation (2.3) to find the line that best fits the relationship between hours studied and test grade.. Right: Logistic regression gives the percent chance of getting a passing (above 70) grade on the test. A 50% chance is indicated by the dashed horizontal line. 23
2.2 Activation functions. It can be seen that $tanh(z)$ is a rescaled version of the sigmoid function $\sigma(z)$, which both mimic the behavior of the step function in a continuous form. The ReLu is a piecewise linear function, which applies a nonlinear transformation to z. 27
2.3 Examples of convolutional filters applied to an image of an apple. From left to right is the original image, then that image convolved with a sharpening kernel, a blurring kernel, and an edge detection kernel called the Sobel filter. 29
2.4 A convolutional layer with six kernels, also called filters. Each kernel is convolved with the three-dimensional input tensor, outputting a single channel for each of these operations. 31

2.5 Top: The data from Figure 2.1 is split into a training set (blue circles) and a test set (green squares). The training data is fit with linear regression on the left and a fifth-degree polynomial on the right. Bottom: Probability distribution of the absolute error of the training set on the left and test set on the right. The polynomial fit achieves lower mean error (E) on the training set but performs much differently on the test set, while linear regression performs consistently on both. The polynomial overfits the training data, making it unable to generalize well. 34
2.6 A plot of a function and its derivative. It can be seen that the maxima and minima of f correspond to the points where its derivative are equal to zero. Only the points where $\frac{df}{dx}$ is increasing are the minima. 35
2.7 Top: State space of batch gradient descent and stochastic gradient descent used to train linear regression, where $y = -4x - 4$. Here, stochastic gradient descent is updating each step with the gradient of only one data point at a time. This leads to a path that does not follow the direction of steepest descent as closely as batch gradient descent, but arrives at the same minimum. Each method uses the constant learning rate $\alpha = 0.1$. Bottom: The two gradient descent methods used on a non-convex loss function. It can be seen that stochastic gradient descent does not get stuck in the smaller local minima as batch gradient descent does due to the variation in its direction. 38
2.8 Training of the linear regression problem from Figure 2.7 using stochastic gradient descent with and without learning rate decay. An initial step size of $\alpha = 0.25$ is used. As each converges on the minimum, it can be seen the learning rate decay allows a more precise minimization. The larger step size does not allow the flexibility to get as close to the minimum, leading to the oscillations seen crossing over it. 39
2.9 Gradient descent with momentum gradually increases the step size during the descent, allowing it to escape the local minimum at 5, which normal gradient descent converges to. 40
2.10 A simple computational graph. Equation (2.50) describes back-propagation used to get the gradient of x with respect to w. 42
2.11 L1 and L2 regularization visualized. The red contour line can be seen as the bound placed on the two parameters β_0 and β_1 by the regularization, L2 on the left and L1 on the right. The blue/green contour shows the minimum parameter values for the cost function by itself. The red point shows where the regularized cost function is minimized. 43
2.12 Comparison of polynomial regression on a small number of data points without regularization and with L2 regularization where $\lambda = 0.1$. 44

2.13 A visualization of transfer learning. A neural network is trained for source Task 1 given a data set of input-output pairs $\{x^{(1)}, y^{(1)}\}$. The shaded boxes indicate portions whose weights are updated during training. The trained hidden layer is then used to train a model for target Task 2, a similar task to the source task with a different domain, $\{x^{(2)}, y^{(2)}\}$. In this model, the hidden layers are frozen and only the output layer's parameters are optimized. 45
2.14 Atomistic neural network approach to predicting properties. The structure of a material can be represented by a matrix consisting of each atom's atomic number (Z) and its Cartesian coordinates (R) with respect to a reference point. This representation will be used as the input to the neural network, which maps it to a target property, such as an energy. . 47
2.15 The atomistic neural network proposed by Behler and Parrinello, shown with a three atom system modeling the total energy E. Each atom, from its atomic number and position, are embedded into symmetry functions G, describing its local environment relative to its neighbors. These symmetry functions are then each passed to a subnet S, each of which is a neural network with the same weights. The result of this subnet is the individual contribution to the total energy from each atom, which is then aggregated to get the total energy. 48
2.16 Architecture of SchNet using a feature size of 64 and three interaction blocks. The interaction block is shown in the middle, and the continuous-filter convolutional layer on the right. 50

3.1 Visualization of the three transfer learning schemes used. The embedding layer, the interaction blocks, and the output layer are either frozen or fine-tuned. 56
3.2 Results of the three best transfer learning methods compared to best direct training method without transfer learning with the original TMO data set. Left: Mean absolute error of the validation set evaluation for the models trained on each of the training set sizes. Right: Percent of evaluations within chemical accuracy of the value computed with DFT. 58
3.3 Results of the two best transfer learning methods compared to best direct training method without transfer learning with the extended TMO data set. Left: Mean absolute error of the validation set evaluation for the models trained on each of the training set sizes. Right: Percent of evaluations within chemical accuracy of the value computed with DFT. 59
3.4 Relative error for each method on the original and extended TMO data set. Left: Original data set. Right: Extended data set. 60

3.5 Error distribution in cases where TL1 and direct training had similar MAE with a significant gap in predictions within chemical accuracy. In each case, TL1 errors make up a larger portion of both the ends of the distribution, with the larger values making a significant impact to the MAE. Left: Predictions from models trained on 100 length training sets of original TMO data set. Right: Predictions from models trained on 500 length training set of extended TMO data set. 61
3.6 Top: Cumulative distribution of validation set error for direct training and TL1 methods trained with the 100 and 400 length training sets of the TMO data. In each case, over 20% more of the predictions from transfer learning are within chemical accuracy (1 kcal/mol). Bottom: Cumulative distribution of validation set error for direct training and TL2 methods trained with the 125 and 500 length training sets of the extended TMO data. 62
3.7 Total hypothetical computational time for screening properties with only DFT and neural networks. Using 500 samples for training, using either direct training or transfer learning would save 135 or 78 hours respectively in computational time over DFT in predicting properties of just 300 more materials. 63

List of Tables

2.1 Validation set Mean absolute error of SchNet on the U_0 from the QM9 dataset with varying amounts of training data. For 50,000 data points and up, error remains within chemical accuracy. 52

3.1 Validation set mean absolute error in eV of SchNet models trained on the full TMO training set. Arch 1 is the larger architecture described and arch 2 is the smaller architecture. L2 signifies the inclusion of L2 regularization with a coefficient of 10^{-3} in the loss during training. . . . 55
3.2 Validation set mean absolute error in eV of each of the transfer learning schemes along with the best performing direct training method across the different-sized splits of the original TMO data set. The bold numbers are the best result for the row. 57
3.3 Validation set mean absolute error in eV of the previously best performing direct training and transfer learning methods applied to the extended data TMO data set. 59
3.4 Total computational time in seconds to train direct and transfer learning models per size of training set. Transfer learning models also take into account the training time of their source model, accounting for the large difference between methods. 60

Abstract

Data-Efficient Surrogate Models for High-Throughput Density Functional Theory

by

Schuyler Krawczuk

High-throughput screening of compounds for desirable electronic properties can allow for accelerated discovery and design of materials. Density functional theory (DFT) is the popular approach used for these quantum chemical calculations, but it can be computationally expensive on a large scale. Recently, machine learning methods have gained traction as a supplementation to DFT, with well-trained models achieving similar accuracy as DFT itself. However, training a machine learning model to be accurate and generalizable to unseen materials requires a large amount of training data. This work proposes a method to minimize the need for novel data creation for training by using transfer learning and publicly-available databases, allowing for both data-efficient and accurate machine learning to replace density functional theory.

Acknowledgments

I'd like to thank my advisor Daniele Venturi for guiding me on this thesis and helping me grow as a researcher over the last year. I also received guidance from Tyler Smart of the Ping computational material science group, and would like to thank him for helping me find my direction for this thesis. I would also like to acknowledge support from the NSF-TRIPODS grant 81389-444168.

Chapter 1

Density Functional Theory

Density functional theory is used to find properties of a many-body system. With a single-body system, the Schrodinger equation can be solved to find the wave function, which contains all of the information about a system. However, the Schrodinger equation quickly becomes more difficult to solve as the number of bodies increases. The number of variables is $3N$ for a 3-dimensional N-bodied system. With more than one electron, the Hamiltonian will also include a term for electron interaction which prevents this problem from being separated into N single-body systems and causes the number of terms to grow exponentially with N [20]. For these reasons, the many-body problem becomes computationally infeasible and a different approach is required to find the wave function in practice.

Density functional theory allows for an alternative approach to solving the Schrodinger equation for many-bodied systems. Density functional theory is based on the idea that any property of a system of interacting particles is a functional of its ground state density. This limits the dimensionality of the problem to 3 regardless of the number of bodies. This is originally shown by the work of Hohenberg and Kohn [12]. Before this is covered, some basic concepts and mathematics of quantum physics will be reviewed.

1.1 Quantum Physics Background

To understand the methods used in performing density functional theory calculation, some knowledge in quantum mechanics is required. This section contains the necessary background information, including the description of common notation and theoretical concepts.

1.1.1 Bra And Ket Vectors

Bra-ket notation is the common way to represent vectors in quantum physics. A vector $\boldsymbol{A}$ can be represented in this notation as a row vector using a "bra", $\langle A|$, and as a column vector with a "ket", $|A\rangle$. The inner product of vectors $\boldsymbol{A}$ and $\boldsymbol{B}$ is written as $\langle A|B\rangle$, and the outer product is $|A\rangle\langle B|$. Any complete bracket expression, such as the preceding inner product, represents a scalar, while any incomplete bracket expression, such as an individual bra or ket, or the outer product, represents a vector. Distributive properties also hold for bra and ket vectors such that,

$$\alpha(|A\rangle + |B\rangle) = \alpha|A\rangle + \alpha|B\rangle, \tag{1.1}$$

where α is a linear operator [7]. A linear operator may operate on either a bra as $\langle A|\alpha$ or a ket, as $\alpha|A\rangle$. The associative property applies to these operations such that,

$$(\langle A|\alpha)|B\rangle = \langle A|(\alpha|B\rangle). \tag{1.2}$$

Since the order these two operators are multiplied in does not matter, this expression is written as $\langle A|\alpha|B\rangle$. The inner product of two functions $f(x)$ and $g(x)$, is the same as,

$$\langle f|g\rangle = \int_{-\infty}^{\infty} f^*(x)g(x)dx, \tag{1.3}$$

where $f^*(x)$ is the complex conjugate of $f(x)$ [7].

1.1.2 The Wave Function, Schrodinger Equation & The Hamiltonian

Contrary to classical mechanics, particles at the atomic scale behave as waves, not just particles, so their dynamics are treated as such. In quantum mechanics, a

system is described by the wave function. The wave function is related to the system's physical state by the probability density. The probability density, or the probability of a particle being at some position $\boldsymbol{r}$, is equal to the square of the amplitude of the wave function,

$$P(\boldsymbol{r}) = |\Psi(\boldsymbol{r})|^2. \tag{1.4}$$

The wave function is typically represented by $\Psi(\boldsymbol{r})$, with $\boldsymbol{r}$ being variables in the Euclidean space. In the case of a system of electrons, the probability can be thought of as the electron density, $n_i(\boldsymbol{r})$, of an electron in state Ψ_i [11].

$$n_i(\boldsymbol{r}) = |\Psi_i(\boldsymbol{r})|^2. \tag{1.5}$$

The system's electron density $n(\boldsymbol{r})$ is the sum of the electron density of each state,

$$n(\boldsymbol{r}) = \sum_{i=1}^{N} |\Psi_i(\boldsymbol{r})|^2. \tag{1.6}$$

A wave function's amplitude is not significant as much as its direction. A state multiplied by a constant coefficient, $\alpha\Psi$, still represents the same state as Ψ. A wave function is typically normalized such that,

$$\langle\Psi|\Psi\rangle^2 = 1. \tag{1.7}$$

The square of the inner product of two different states represents the probability of one state transforming to the other, justifying this normalization. The wave function behaves according to the Schrodinger equation. This partial differential equation, in the time-independent case which we will focus on, is written as,

$$\left[\frac{\hbar}{2m}\nabla^2 + V(\boldsymbol{r})\right]\Psi(\boldsymbol{r}) = E\Psi(\boldsymbol{r}). \tag{1.8}$$

The left-hand coefficient of the wave function in the Schrodinger equation is known as the Hamiltonian operator, which is the sum of kinetic and potential energy operators T and V,

$$H = T + V. \tag{1.9}$$

E is the total energy of the state. This leads to a simplified notation for the Schrodinger equation,

$$H\Psi(\boldsymbol{r}) = E\Psi(\boldsymbol{r}). \tag{1.10}$$

Since E is a scalar quantity, H is a linear operator and E is an eigenvalue of H. We can re-write the equation in bracket notation as,

$$H\left|\Psi\right\rangle = E\left|\Psi\right\rangle. \tag{1.11}$$

E can be found by multiplying both sides on the left by $\langle\Psi|$,

$$\langle\Psi|H|\Psi\rangle = \langle\Psi|E|\Psi\rangle = E\left\langle\Psi|\Psi\right\rangle = E. \tag{1.12}$$

$\langle\Psi|H|\Psi\rangle$ is known as the expectation value, or the expected value of E when H operates on Ψ. The expectation value of the Hamiltonian can also be written as,

$$E = \langle\Psi|H|\Psi\rangle = \int_{\mathbb{R}^3} \Psi^*(\boldsymbol{r})H\Psi(\boldsymbol{r})d\boldsymbol{r}. \tag{1.13}$$

For the case of a system of interacting particles, Equation (1.13) can be written in a more specific form. In the following, V_{ext} is the external potential; the electric potential on electrons because of atoms' nuclei. U is the internal potential caused by electron interaction. $n(r)$ is the electron density, which in $\mathbb{R}^3$ is the electric charge per unit volume.

$$E = \langle\Psi|H|\Psi\rangle = \langle\Psi|(T+U)|\Psi\rangle + \int_{\mathbb{R}^3} V_{ext}(\boldsymbol{r})n(\boldsymbol{r})d\boldsymbol{r}. \tag{1.14}$$

1.2 Hohenberg-Kohn Theorems

Hohenberg and Kohn's two proofs lay out the theoretical groundwork for modern density functional theory. First, they show that external potential is a unique functional of electron density. From this it follows that the total energy functional is also unique for a electron density.

Theorem 1. *The external potential of an interacting system of particles is uniquely determined by the ground state particle density.*

Proof. Given two unique external potentials $V_{ext}^{(1)}$ and $V_{ext}^{(2)}$, where $V_{ext}^{(1)}$ is not $V_{ext}^{(2)}$ + a constant, suppose they lead to the same ground state density $n(\boldsymbol{r})$. The two potentials lead to different Hamiltonians $H^{(1)}$ and $H^{(2)}$ with ground state wave functions $\Psi^{(1)}$ and $\Psi^{(2)}$, which are assumed to have the same ground state density. U is the potential due to electron interaction within the system. The two energies can be written as,

$$E^{(1)} = \left\langle \Psi^{(1)} \middle| H_1 \middle| \Psi^{(1)} \right\rangle = \left\langle \Psi^{(1)} \middle| (T+U) \middle| \Psi^{(1)} \right\rangle + \int_{\mathbb{R}^3} V_{ext}^{(1)}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r} \tag{1.15}$$

and

$$E^{(2)} = \left\langle \Psi^{(2)} \middle| H_2 \middle| \Psi^{(2)} \right\rangle = \left\langle \Psi^{(2)} \middle| (T+U) \middle| \Psi^{(2)} \right\rangle + \int_{\mathbb{R}^3} V_{ext}^{(2)}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r}. \tag{1.16}$$

Since $\Psi^{(2)}$ is not the ground state,

$$E^{(1)} < \left\langle \Psi^{(2)} \middle| H_1 \middle| \Psi^{(2)} \right\rangle, \tag{1.17}$$

$$E^{(1)} < \left\langle \Psi^{(2)} \middle| (T+U) \middle| \Psi^{(2)} \right\rangle + \int_{\mathbb{R}^3} V_{ext}^{(1)}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r}, \tag{1.18}$$

$$E^{(1)} < \left\langle \Psi^{(2)} \middle| (T+U) \middle| \Psi^{(2)} \right\rangle + \int_{\mathbb{R}^3} V_{ext}^{(2)}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r}) + \int_{\mathbb{R}^3} V_{ext}^{(1)}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r} \tag{1.19}$$

$$- \int_{\mathbb{R}^3} V_{ext}^{(2)}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r}, \tag{1.20}$$

$$E^{(1)} < E^{(2)} + \int_{\mathbb{R}^3} (V_{ext}^{(1)}(\boldsymbol{r}) - V_{ext}^{(2)}(\boldsymbol{r})) n(\boldsymbol{r}) d\boldsymbol{r}. \tag{1.21}$$

And likewise for $E^{(2)}$,

$$E^{(2)} < E^{(1)} + \int_{\mathbb{R}^3} (V_{ext}^{(2)}(\boldsymbol{r}) - V_{ext}^{(1)}(\boldsymbol{r})) n(\boldsymbol{r}) d\boldsymbol{r}. \tag{1.22}$$

Adding these two together gives us a contradictory inequality,

$$E^{(1)} + E^{(2)} < E^{(1)} + E^{(2)}. \tag{1.23}$$

This shows that two unique potentials will not have a same corresponding density. This means a potential is uniquely determined by the density. □

In their second proof, it is shown that this unique energy functional is minimized by the ground state density.

Theorem 2. *There exists a universal functional for energy $E[n]$ that is valid for any external potential. The global minimum of this functional corresponds to the ground state energy, and the density n that minimizes it is the ground state density.*

Proof. Theorem 1 shows that the ground state density is unique for each external potential, the energy can be written as a functional of density:

$$E[n] = T[n] + E_{\text{int}}[n] + E_{\text{II}} + \int_{\mathbb{R}^3} V_{\text{ext}}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r}, \tag{1.24}$$

$$F_{\text{HK}}[n] = T[n] + E_{\text{int}}[n], \tag{1.25}$$

$$E[n] = F_{\text{HK}}[n] + \int_{\mathbb{R}^3} V_{\text{ext}}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r} + E_{II}. \tag{1.26}$$

E_{II} refers to the interaction energy of the nuclei. The functional F_{HK} contains all of the terms of the interacting *electron* system and is only a function of n, so will be treated the same regardless of external potential. Given the ground state density $n^{(1)}$, the ground state energy is defined by,

$$E^{(1)}[n^{(1)}] = \left\langle \Psi^{(1)} \middle| H^{(1)} \middle| \Psi^{(1)} \right\rangle. \tag{1.27}$$

Given a different density $n^{(2)}$ with corresponding wave function $\Psi^{(2)}$, the energy is,

$$E^{(2)} = \left\langle \Psi^{(2)} \middle| H^{(1)} \middle| \Psi^{(2)} \right\rangle. \tag{1.28}$$

We also know,

$$\left\langle \Psi^{(1)} \middle| H^{(1)} \middle| \Psi^{(1)} \right\rangle < \left\langle \Psi^{(2)} \middle| H^{(1)} \middle| \Psi^{(2)} \right\rangle. \tag{1.29}$$

So from this, it can be seen that the energy corresponding to the ground state density is lower than that of any other density, minimizing the functional $E[n]$. If $F_{\text{HK}}[n]$ is known, the system can be minimized with respect with respect to n to find the ground state energy.

□

1.3 Kohn-Sham Equations

The Kohn-Sham method is based on the Hohenberg-Kohn theorems, and is the common way modern density functional theory calculations are done. This approach

replaces the interacting many-body system with an auxiliary system of non-interacting particles. The Kohn-Sham ansatz assumes that the ground state of this system is equivalent to the ground state of the interacting system [12]. To solve the Schrodinger equation for a many-body interacting system, there is a number of interaction terms that exponentially increases with the number of atoms. Solving the non-interacting system in its place is computationally feasible because of the removal of the interaction terms.

The energy functional is rewritten as,

$$E_{\mathrm{KS}} = T[n] + E_{\mathrm{Hartree}}[n] + \int_{\mathbb{R}^3} V_{\mathrm{ext}}(\boldsymbol{r}) n(\boldsymbol{r}) d\boldsymbol{r} + E_{\mathrm{XC}}[n], \tag{1.30}$$

where E_{Hartree} describes the self-interaction of the density, defined as,

$$E_{\mathrm{Hartree}}[n] = \frac{1}{2} \int \frac{n(\boldsymbol{r}) n(\boldsymbol{r}')}{|\boldsymbol{r} - \boldsymbol{r}'|} d\boldsymbol{r} d\boldsymbol{r}', \tag{1.31}$$

and the potential energy, T is defined as the expectation value of the Laplacian for each particle:

$$T[n] = \frac{1}{2} \sum_{i=1}^{N} \langle \Psi_i[n] | \nabla^2 | \Psi_i[n] \rangle \,. \tag{1.32}$$

Each of the terms of this new energy functional is well-defined, except for the exchange-correlation energy term E_{XC}. This functional accounts for the interaction between electrons and the difference between the kinetic energy of the interacting and the non-interacting system. An exact form of the exchange-correlation functional is not known exactly. However, this term's contribution is small enough compared to the others that it can be inexactly approximated to still obtain an accurate result. There are a number of approximation techniques based on the local density that will be talked about later on. With these terms, an effective potential and Hamiltonian for the auxiliary system can be written respectively as,

$$V_{\mathrm{eff}}(\boldsymbol{r}) = V_{\mathrm{ext}}(\boldsymbol{r}) + \frac{\delta E_{\mathrm{Hartree}}[n]}{\delta n(\boldsymbol{r})} + \frac{\delta E_{\mathrm{XC}}[n]}{\delta n(\boldsymbol{r})} = V_{ext}(\boldsymbol{r}) + V_{\mathrm{Hartree}}[n] + V_{\mathrm{XC}}[n] \tag{1.33}$$

and

$$H_{\mathrm{eff}}(\boldsymbol{r}) = -\frac{1}{2}\nabla^2 + V_{\mathrm{eff}}(\boldsymbol{r}). \tag{1.34}$$

Using this new Hamiltonian, the Kohn-Sham equations can be written very similarly to the Schrodinger equation,

$$H_{\text{eff}}\psi_i(\boldsymbol{r}) = \epsilon_i\psi_i(\boldsymbol{r}), \tag{1.35}$$

where ψ_i is the wave function for each particle in the non-interacting system and ϵ_i are the eigenvalues of the Hamiltonian. While in the original Schrodinger equation these eigenvalues represent the allowed energies of the system, that is not the case in the Kohn-Sham equations, since the auxiliary system of non-interacting particles is not true to its real interacting form. The significance of these eigenvalues is an open area of research.

1.3.1 Exchange-Correlation Functionals

The exchange-correlation functional's exact form is unknown, but it is also the smallest contribution to the total energy. Because of this, it can be approximated and still lead to an accurate solution. The simplest approximation used for exchange-correlation is the local density approximation (LDA). Stated by Kohn and Sham, solids can be considered close to the limit of the uniform electron gas [12]. The local exchange-correlation energy for the uniform electron gas is known, written as ϵ_{xc}. The LDA exchange-correlation function can be written as,

$$E_{\text{XC}}[n] = \int n(\boldsymbol{r})\epsilon_{\text{xc}}(n(\boldsymbol{r}))d\boldsymbol{r}. \tag{1.36}$$

A more accurate class of functionals that build off of LDA is the generalized-gradient approximation (GGA). These are exchange-correlation functionals that include a term F_{XC} that is in terms of the gradient of the density.

$$E_{\text{XC}}[n] = \int n(\boldsymbol{r})\epsilon_{\text{xc}}(n(\boldsymbol{r}))F_{\text{XC}}(\nabla n(\boldsymbol{r}))d\boldsymbol{r}. \tag{1.37}$$

The improvement in accuracy given by GGA functionals led to the wider adoption of density functional theory across chemistry and material science [20].

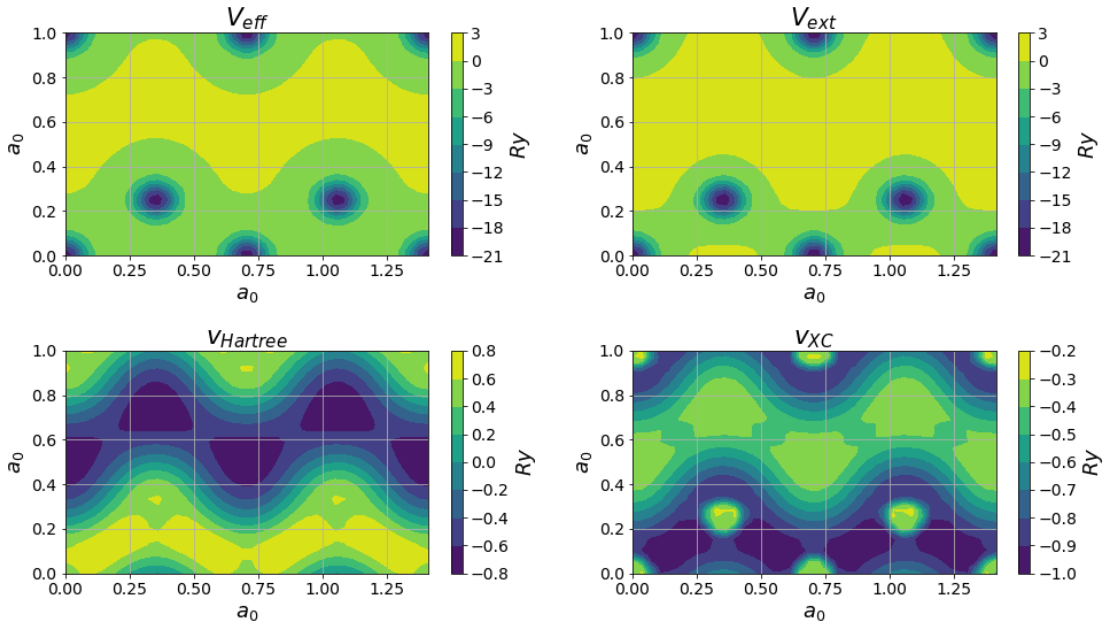
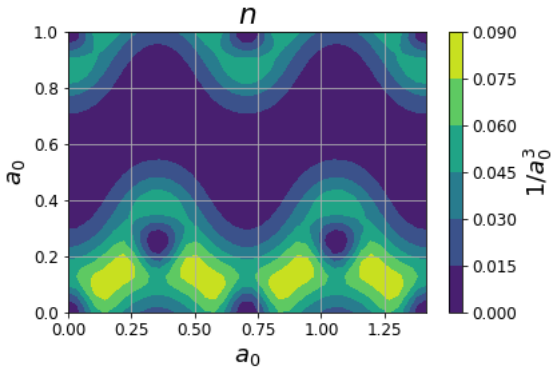


Figure 1.1: Top: From top left moving clockwise: plot of V_{eff}, V_{ext}, V_{XC}, and V_{Hartree} in a 2-dimensional slice of a Si structure. Spatial units are in Bohrs ($5.29 \times 10^{-11} m$) and potential is in Rydberg units ($13.6eV$). Bottom: The resulting electron density (in electrons per cubic Bohr) for the above potential.



1.4 Solving The Kohn-Sham Equations

To solve the Kohn-Sham equations, two quantities are needed: the external potential V_{ext} and n. V_{ext} can be constructed based on the structure of the system. n can also be approximated based on the structure of the system, but will not be accurate enough. n, if correct, will be self-consistent through the algorithm shown in Figure 1. In this process shown, an initial value for n is used to construct the effective potential, for which the Kohn-Sham equations are solved. The resulting wave functions can then be used to recalculate n. If this final n is close enough to the initial guess for n, then it is considered self-consistent and is the correct density for the system. If not, it is adjusted and another iteration with a new initial n is computed. With this self-consistent density, other important properties of the system can be found.

1.4.1 The Eigenvalue Problem

Once a value for n is given and the Hamiltonian is created, there comes the issue of solving the Kohn-Sham equations to obtain each wave function. We are left with the eigenvalue problem,

$$H\psi_i(\boldsymbol{r}) - \epsilon_i\psi_i(\boldsymbol{r}) = 0. \tag{1.38}$$

In order to solve for the wave function, the Hamiltonian will need to be diagonalized. Using an exact method to do this is not computationally efficient for a large matrix, which is the typically the case in DFT calculations, so an inexact, iterative method is used [2] [9]. The widely used approach to solving this problem is representing ψ with a finite expansion in a basis set ϕ [29], where

$$\psi_i = \sum_{j=1}^{N} c_j\phi_j. \tag{1.39}$$

The wave function in this form can be substituted into the eigenvalue problem, which will then take on another term called the overlap matrix (S) to become a generalized eigenvalue problem. The overlap matrix describes the interaction between

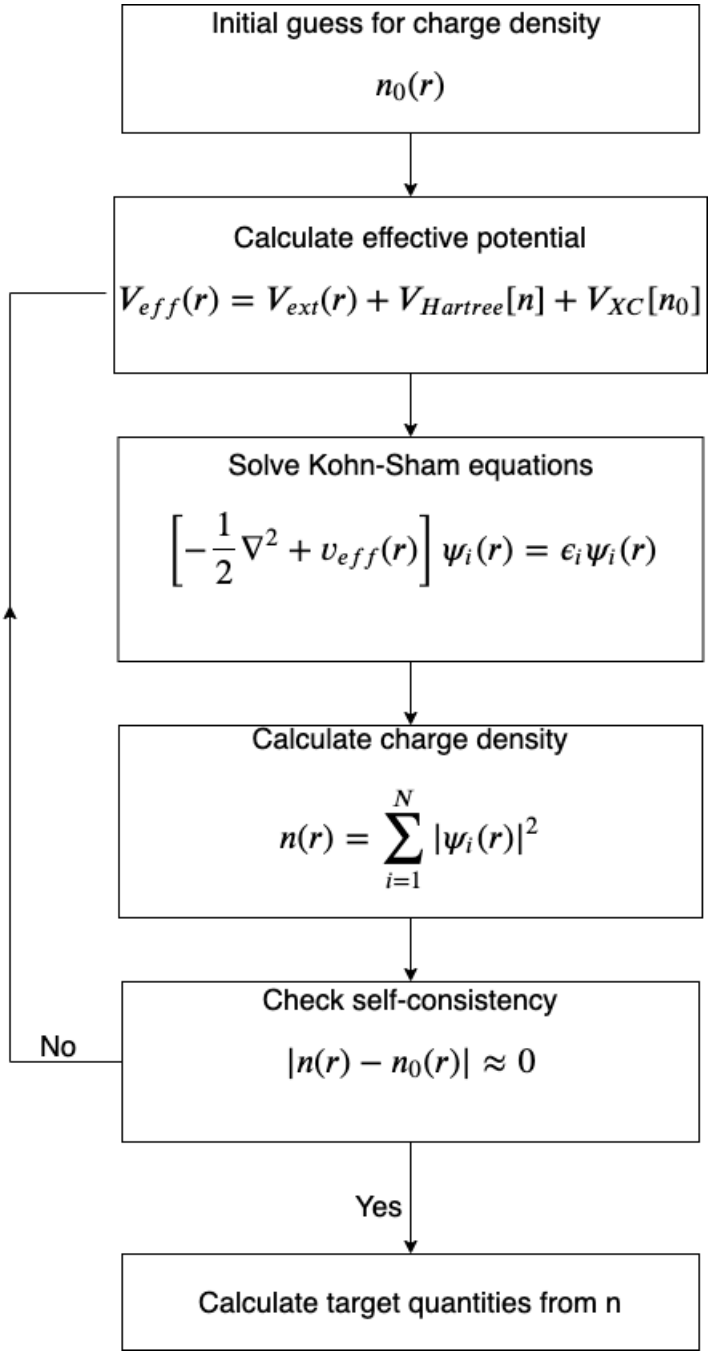


Figure 1.2: Algorithm for performing a DFT calculation

basis functions, written as,

$$S_{jk} = \langle \phi_j | \phi_k \rangle \,. \tag{1.40}$$

The generalized eigenvalue problem is now written as,

$$\sum_{jk} H_{ij} c_j c_k - \sum_{jk} \epsilon_i S_{jk} c_j c_k = 0, \tag{1.41}$$

$$\sum_{jk} c_j c_k (H_{jk} - \epsilon_i S_{jk}) = 0. \tag{1.42}$$

Freezing the change of this expression with respect to coefficient c_j, we are left with the condition,

$$\sum_{k} c_k (H_{jk} - \epsilon_i S_{jk}) = 0. \tag{1.43}$$

1.4.2 Plane Wave Basis

To solve the eigenvalue problem, the wave function is usually expanded into a set of basis functions, as seen in the previous section. The most commonly used basis is the plane wave basis. The plane wave basis is physics terminology for the Fourier basis. Plane wave basis is convenient because the orthonormality of Fourier functions is taken advantage of to simplify calculations, effectively setting the overlap matrix S from Equation (1.43) to the identity matrix. The trade-off between convergence and efficiency is also easily controlled by setting a cutoff energy, which corresponds to the number of plane waves [23]. The physical intuition behind this is motivated by Bloch's theorem, which says that a wave function ψ in a periodic potential can be written as a plane wave modulated by a periodic function [23]. In plane wave basis, the wave function can be written as,

$$\psi_i(\boldsymbol{r}) = \sum_{q} c_{i,q} e^{i\boldsymbol{q}\cdot\boldsymbol{r}}. \tag{1.44}$$

Since plane waves $\boldsymbol{q}$ are orthonormal,

$$\langle \boldsymbol{q'} | \boldsymbol{q} \rangle = \delta_{\boldsymbol{q'},\boldsymbol{q}}. \tag{1.45}$$

Rewriting the Schrodinger equation in Fourier space we get,

$$\sum_{q} \langle \boldsymbol{q'} | H | \boldsymbol{q} \rangle \, c_{i,\boldsymbol{q}} = \epsilon_i \sum_{q} \langle \boldsymbol{q'} | \boldsymbol{q} \rangle \, c_{i,\boldsymbol{q'}} = \epsilon_i c_{i,\boldsymbol{q'}}. \tag{1.46}$$

The two terms of the Hamiltonian (T and V_{eff}) can be rewritten for the plane wave basis. The kinetic energy matrix operator will become,

$$\langle \boldsymbol{q'} | \frac{1}{2} \nabla^2 | \boldsymbol{q} \rangle = \frac{1}{2} |\boldsymbol{q}|^2 \delta_{\boldsymbol{q'},\boldsymbol{q}}. \tag{1.47}$$

In a crystal with a periodic potential, $V_{\text{eff}}(\boldsymbol{r})$ is expanded in terms of Fourier components,

$$V_{\text{eff}}(\boldsymbol{r}) = \sum_{m} V_{\text{eff}}(\boldsymbol{G}_m) e^{i\boldsymbol{G_m}\cdot\boldsymbol{r}}, \tag{1.48}$$

where $\boldsymbol{G}$ are lattice vectors. $V_{\text{eff}}(\boldsymbol{G})$ is then,

$$V_{\text{eff}}(\boldsymbol{G}) = \frac{1}{\Omega} \int_{\Omega} V_{\text{eff}}(\boldsymbol{r}) e^{i\boldsymbol{G_m}\cdot\boldsymbol{r}} d\boldsymbol{r}, \tag{1.49}$$

Where Ω is the volume of a cell of the lattice. The matrix form of the effective potential is then,

$$\langle \boldsymbol{q'} | V_{\text{eff}} | \boldsymbol{q} \rangle = \sum_{m} V_{\text{eff}}(\boldsymbol{G}) \delta_{\boldsymbol{q'}-\boldsymbol{q},\boldsymbol{G}_m}, \tag{1.50}$$

so the elements are zero unless $\boldsymbol{q}$ and $\boldsymbol{q'}$ are separated by a lattice vector $\boldsymbol{G}_m$ [20]. Defining $\boldsymbol{q} = \boldsymbol{k} + \boldsymbol{G}_m$ and $\boldsymbol{q'} = \boldsymbol{k} + \boldsymbol{G}_{m'}$, the Schrodinger equation is written in terms of $\boldsymbol{k}$ as,

$$\sum_{m'} H_{m,m'}(\boldsymbol{k}) c_{i,m'}(\boldsymbol{k}) = \epsilon_i c_{i,m}(\boldsymbol{k}), \tag{1.51}$$

where the full Hamiltonian in terms of $\boldsymbol{k}$ is,

$$H_{m,m'}(\boldsymbol{k}) = \frac{1}{2} |\boldsymbol{k} + \boldsymbol{G}_m|^2 \delta_{m,m'} + V_{\text{eff}}(\boldsymbol{G}_m - \boldsymbol{G}_{m'}). \tag{1.52}$$

The number of plane waves used is set by a cutoff energy, E_{cut}, limiting the plane waves to a finite basis. Only plane waves satisfying the inequality,

$$\frac{1}{2} |\boldsymbol{G}|^2 < E_{\text{cut}}, \tag{1.53}$$

are kept. A higher cutoff energy will allow for more plane waves and in turn a more accurate representation, but is more computationally expensive.

1.4.3 Pseudopotential

Pseudopotentials are a surrogate potential used in density functional theory calculations to increase computational efficiency.The nucleus of an atom causes an electric potential that is felt by electrons in the form of

$$V_{\text{nucleus}} = \frac{Z}{|\boldsymbol{r}|}, \tag{1.54}$$

where Z is the net positive charge of the nucleus. To construct the external potential of a molecule or system for the DFT calculation, one can simply sum these potentials for the given $\boldsymbol{r}$, be it 1, 2, or 3-dimensional:

$$V_{\text{ext}} = \sum_{i=1}^{N} \frac{Z_i}{|\boldsymbol{r} - \boldsymbol{r}_i|}. \tag{1.55}$$

In practice, however, this is not the most efficient method. Close to the nucleus, there are core electrons, or electrons that are bonded as opposed to the unbonded valence electrons. The higher density of electrons in the core region leads to a less smooth function and more plane waves needed to model the wave function in this region. More components leads to more computational cost. These core electrons, since they are already bonded, can be removed from the potential to approximate the wave function [16]. They should contribute very little to the interaction between atoms. This allows for accurate modeling of the wave function outside of the core region. This creates a new potential that is the same as the true potential outside of the core region, called a pseudopotential.

The aim of a pseudopotential is to increase the speed of calculations without hurting their accuracy. Given the valence state ψ, the aim is to replace it with a smoother state inside the core region while remaining the same outside of it. The valence portion of this state can be replaced by a smooth pseudo-wave function ϕ, while the core region can be expanded in terms of the core states, χ.

$$|\psi\rangle = |\phi\rangle + \sum_{n=1}^{N_{\text{core}}} a_n |\chi_n\rangle . \tag{1.56}$$

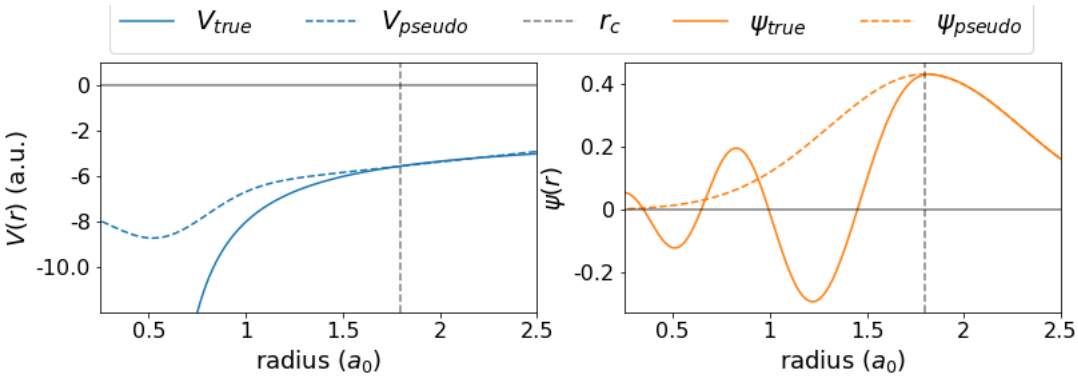


Figure 1.3: Comparison of a potential and its wave function (solid lines) to the pseudopotential and its resulting wave function (dashed lines). The pseudopotential results in a smoother wave function that is identical to the true wave function outside of the core radius, r_c.

The valence state must be orthogonal to the core states, so taking the inner product of the two leaves us with,

$$\langle \chi_n | \psi \rangle = \langle \chi_n | \phi \rangle + \sum_{n=1}^{N_{\text{core}}} a_n \langle \chi_m | \chi_n \rangle = 0. \tag{1.57}$$

Writing the state in terms of ϕ we get,

$$|\psi\rangle = |\phi\rangle - \sum_{n=1}^{N_{\text{core}}} \langle \chi_n | \psi \rangle |\chi_n\rangle . \tag{1.58}$$

If the Hamiltonian is then applied to the state in this form, the resulting eigenvalue will be the same as that of the true state,

$$H|\phi\rangle + \sum_{n=1}^{N_{\text{core}}} (\epsilon - \epsilon_n) |\chi_m\rangle \langle \chi_n | \phi \rangle = \epsilon |\phi\rangle , \tag{1.59}$$

where ϵ_n are the core eigenvalues.

1.4.4 Reaching Self-Consistency

At each iteration of the algorithm in Figure 2, a choice of a new guess for n is made. Initially, the density is approximated. This can be done using the atomic

densities [23],

$$n_0(\boldsymbol{r}) = \sum_{\alpha=1}^{N} n_\alpha(\boldsymbol{r} - \boldsymbol{R}_\alpha), \tag{1.60}$$

where R_α and n_α are the position and atomic density of atom α. The Kohn-Sham equations are then solved and density is calculated by,

$$n(\boldsymbol{r}) = \sum_{i}^{N} |\psi_i(\boldsymbol{r})|^2. \tag{1.61}$$

For the following steps it is not as simple as using the last output a the next input, however. To choose the best next step, we look at the error from the optimal density, n_{KS}:

$$\delta n = n - n_{\mathrm{KS}}$$

. Taking the output density as a function of the input density, we can solve for the optimal density n_{KS}, which would ideally be the next step.

$$\delta n^{\mathrm{out}}[n^{\mathrm{in}}] = n^{\mathrm{out}} - n_{\mathrm{KS}} \tag{1.62}$$

$$= \frac{n^{\mathrm{out}} - n_{\mathrm{KS}}}{n^{\mathrm{in}} - n_{\mathrm{KS}}}(n^{\mathrm{in}} - n_{\mathrm{KS}}) \tag{1.63}$$

$$= \frac{\delta n^{\mathrm{out}}}{\delta n^{\mathrm{in}}}(n^{\mathrm{in}} - n_{\mathrm{KS}}), \tag{1.64}$$

$$n_{\mathrm{KS}} = n^{\mathrm{in}} - (\frac{\delta n^{\mathrm{out}}}{\delta n^{\mathrm{in}}})^{-1}(n^{\mathrm{out}} - n^{\mathrm{in}}). \tag{1.65}$$

This does not give us an exact solution for n_{KS} since it is just based off the change in previous iterations, but it does give the optimal input for the following input. This formula for finding the next step is simplified with the linear mixing formula. the error ratio term is replaced with a constant α so that the next n is a linear combination of the last two steps' n.

$${n^{\mathrm{in}}}_{i+1} = {n^{\mathrm{in}}}_i + \alpha({n^{\mathrm{out}}}_i - {n^{\mathrm{in}}}_i). \tag{1.66}$$

1.5 Numerical Methods

The most computationally expensive part of the self-consistent calculation is the solving of the Kohn-Sham eigenvalue problem. This problem requires the repeated

diagonalization of the Hamiltonian matrix. Two methods that are commonly implemented in DFT codes, such as Quantum Espresso and VASP, are the Davidson algorithm and conjugate gradient method [25]. The Davidson algorithm is more computationally efficient, but has a higher memory requirement. Conjugate gradient is slower, but more memory-efficient [4].

1.5.1 Davidson Algorithm

Going back to Equation (1.61), the calculation of density is only dependent on the lowest N orbitals, where N may be much lower than the number of plane waves m that define the size of the Hamiltonian (1.61). The Davidson algorithm allows the diagonalization of a subspace of the matrix, finding just the lowest few eigenvalues [5]. This algorithm starts with a set of trial orbitals and eigenvectors, $\psi_i^{(0)}$ and $\epsilon_i^{(0)}$. The eigenvalue problem is solved in a 2N-dimensional subspace spanned by a reduced basis set $\phi^{(0)}$, where $\phi_i^{(0)} = \psi_i^{(0)}$ and $\phi_{i+N}^{(0)} = \delta\psi_i^{(0)}$ [4]. For each step, the Hamiltonian is written in terms of the basis,

$$H_{jk} = \left\langle \phi_j^{(0)} \middle| H \middle| \phi_k^{(0)} \right\rangle, S_{jk} = \left\langle \phi_j^{(0)} \middle| S \middle| \phi_k^{(0)} \right\rangle, \tag{1.67}$$

then diagonalized using a conventional algorithm. The trial eigenpairs are then updated by,

$$\psi_i^{(1)} = \sum_{j=1}^{2N} c_j^{(i)} \phi_j^{(0)}, \epsilon_i^{(1)} = \left\langle \psi_i^{(1)} \middle| H \middle| \psi_i^{(1)} \right\rangle. \tag{1.68}$$

This process is repeated until sufficient convergence is achieved in the residual g_i of the eigenvalue problem,

$$g_i^{(0)} = (H - \epsilon_i^{(0)} S)\psi_i^{(0)}. \tag{1.69}$$

1.5.2 Conjugate Gradient

In this approach, the eigenvalue problem is solved as a constrained minimization problem in the form,

$$min \left[\langle \psi_i | H | \psi_i \rangle - \sum_{j \leq i} \lambda_j (\langle \psi_i | S | \psi_j \rangle - \delta_{ij}) \right], \tag{1.70}$$

where λ_j are Lagrange multipliers. We assume the first j eigenvectors are already calculated, where $j = i - 1$. An initial guess $\psi^{(0)}$ is made for the ith eigenvector such that $\langle \psi^{(0)} | S | \psi^{(0)} \rangle = 1$ and $\langle \psi^{(0)} | S | \psi_j \rangle = 0$. A preconditioned diagonal matrix P is introduced, leading to the auxiliary functions $y = P^{-1}\psi$, $\tilde{H} = PHP$, and $\tilde{S} = PSP$. The new minimization problem is,

$$\left[\langle y | \tilde{H} | y \rangle - \lambda (\langle y | \tilde{S} | y \rangle - 1) \right]. \tag{1.71}$$

The gradient of Equation (1.71) is

$$g^{(0)} = (\tilde{H} - \lambda \tilde{S}) y^{(0)}. \tag{1.72}$$

Making this gradient initially orthonormal to the starting vector so that,

$$\left\langle g^{(0)} \middle| \tilde{S} \middle| y^{(0)} \right\rangle = 0, \tag{1.73}$$

the Lagrange multiplier can be written as,

$$\lambda = \frac{\langle y^{(0)} | \tilde{S} \tilde{H} | y^{(0)} \rangle}{\langle y^{(0)} | \tilde{S}^2 | y^{(0)} \rangle}. \tag{1.74}$$

$Pg^{(0)}$ is then orthonormalized by explicitly orthonormalizing ψ_j, and the conjugate gradient $h^{(0)}$ is introduced with an initial value set to the gradient $g^{(0)}$, as well as the normalized direction $n^{(0)}$. This direction is written as,

$$n^{(0)} = \frac{h^{(0)}}{\langle h^{(0)} | \tilde{S} | h^{(0)} \rangle^{1/2}}. \tag{1.75}$$

The minimum of $\langle y^{(1)} | \tilde{H} | y^{(1)} \rangle$ is search for along the direction $y^{(1)}$ where,

$$y^{(1)} = y^{(0)} \cos\theta + n^{(0)} \sin\theta. \tag{1.76}$$

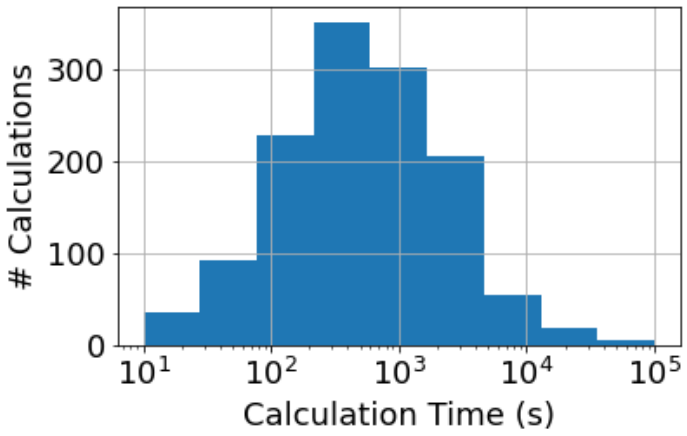


Figure 1.4: Distribution computational time of 1,300 DFT calculations. Calculations were done on a compute node with 2 Intel 2.1GHz Xeon E5-2620v4 processors using 16 cores and 64GB RAM.

in order to ensure the constraint on the norm is enforced [1]. The minimum is found analytically by,

$$\theta = \frac{1}{2}\arctan\left(\frac{2\,\mathrm{Re}\,\langle y^{(0)}|\tilde{H}|n^{(0)}\rangle}{\epsilon^{(0)} - \langle n^{(0)}|\tilde{H}|n^{(0)}\rangle}\right). \tag{1.77}$$

Each iteration, the next conjugate gradient is calculated from the previous using the Polak-Ribiere formula [28],

$$h^{(n)} = g^{(n)} + \gamma^{(n-1)}h^{(n-1)} \tag{1.78}$$

where,

$$\gamma^{(n-1)} = \frac{\langle g^{(n)} - g^{(n-1)}|\tilde{S}|g^{(n)}\rangle}{\langle g^{(n-1)}|\tilde{S}|g^{(n-1)}\rangle}. \tag{1.79}$$

$h^{(n)}$ is re-orthogonalized to $y^{(n)}$ in the next step.

1.5.3 Computational Benchmark

Depending on the size of the system and level of accuracy of the exchange-correlation functional, density functional theory can vary greatly in computational time. Figure 1.4 quantifies some of these times, using calculations done of 1,300 transition

metal oxides with a mean size of 30 atoms. The calculations had a mean time of 1,635 seconds, the fastest taking 5 seconds and the slowest taking 20 hours. These calculations were run using VASP with a GGA exchange correlation functional.

Chapter 2

Neural Network Modeling For Density Functional Theory

In this section we go over what machine learning is, some common machine learning methods, and how it is currently applied for modeling atomic systems. Machine learning can be divided into two categories: supervised and unsupervised learning. This research uses supervised learning techniques. Supervised machine learning involves learning a function to map an input to an output. Given a mapping between independent variable $\boldsymbol{x}$ and dependent variable $\boldsymbol{y}$,

$$\boldsymbol{y} = f(\boldsymbol{x}), \tag{2.1}$$

supervised machine learning attempts to approximate the mapping f given a set of input and output pairs.

2.1 Input-Output Maps

Two common examples of input-output maps in machine learning are regression and classification. In regression, the input is mapped to a numerical output so that $\boldsymbol{y}$ is a quantitative variable. A common example of this is linear regression. Linear regression learns a mapping with an affine relationship assumed between the input and

output. Linear regression has two parameters, w and b where,

$$\boldsymbol{y} = \boldsymbol{w}\boldsymbol{x} + \boldsymbol{b}. \tag{2.2}$$

An example of linear regression is seen in Figure 2.1, where a line of best fit is found for the given data points. In order to learn the best fit mapping, a common approach is to minimize the mean square of the error. Given N data points, the function to minimize, also called the cost function or loss function, is,

$$E(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{N} \sum_{n=1}^{N} \left[\boldsymbol{y}_n - (\boldsymbol{w}\boldsymbol{x}_n + \boldsymbol{b})\right]^2. \tag{2.3}$$

The cost function E is minimized with respect to the parameters $\boldsymbol{w}$ and $\boldsymbol{b}$. Generalizing this as a problem with n data points of k dimensions each, the linear model in Equation (2.2) for a given data point $\boldsymbol{x_i}$ becomes,

$$y_i = b_i + w_1 x_{1i} + w_2 x_{2i} + \ldots + w_k x_{ki}. \tag{2.4}$$

Defining a matrix X as a $k \times n$ matrix containing all n data points,

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{21} & \ldots & x_{k1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{2n} & \ldots & x_{kn} \end{bmatrix}, \tag{2.5}$$

and a column vector for both $\boldsymbol{y}$ and the parameters $\boldsymbol{\beta}$,

$$\boldsymbol{\beta} = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_k \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \tag{2.6}$$

the cost function from Equation (2.3) can be rewritten as,

$$E(\boldsymbol{\beta}) = \frac{1}{N} ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2. \tag{2.7}$$

Regression is useful for both prediction and inferring relationships. It can be used to predict a y-value for a never before seen x-value. It can also discover relationships

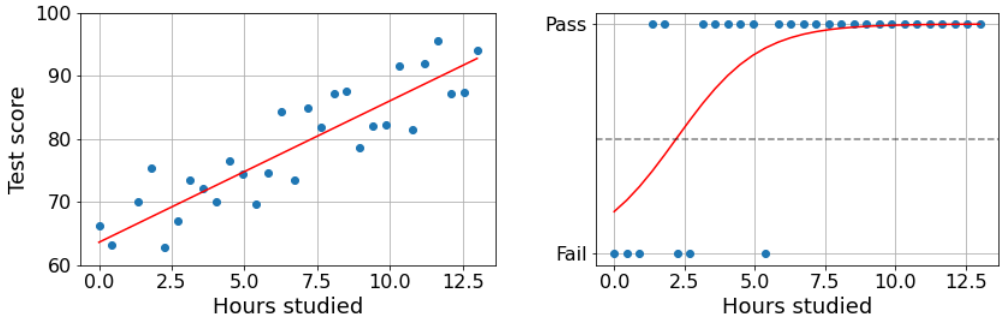


Figure 2.1: Left: Linear regression. The red line minimizes the error function in Equation (2.3) to find the line that best fits the relationship between hours studied and test grade.. Right: Logistic regression gives the percent chance of getting a passing (above 70) grade on the test. A 50% chance is indicated by the dashed horizontal line.

between variables, such as determining the rate of change w between x and y in linear regression. This can have important interpretations for real problems. For example, given the number of hours each student in a 30-student class studied for a test and their final score on the test, linear regression could be used to determine a relationship between these two values. From this regression, the number of hours of studying needed to get a certain score can be inferred. From Figure 2.1, we can infer that if a student wants a score of 80, they must study for about seven hours.

In classification, the input is mapped to a qualitative variable rather than a quantitative variable, such as a label. $\boldsymbol{y}$ is a numerical code corresponding to a label. Consider the last example from the previous subsection. If it was only known if a student passed or failed their test, this data could be represented as 1 for pass, 0 for fail. A basic algorithm for classification problems is the perceptron. The perceptron is an adaptation of linear regression to classification, done by applying a step function to the its output. The output of a perceptron is,

$$\begin{cases} 1 & \boldsymbol{X}\boldsymbol{\beta} > 0 \\ 0 & \boldsymbol{X}\boldsymbol{\beta} \leq 0 \end{cases}. \tag{2.8}$$

The perceptron is the basis of the neural network, which will be seen next. A more practical algorithm for classification with a statistical interpretation is logistic regres-

sion. Logistic regression, like linear regression, returns a continuous function based on its input. Unlike linear regression, the output is made to be between 0 and 1, the two values for binary labels. This is done by applying a logistic function rather than a step function as done by the perceptron. This function becomes,

$$h(\boldsymbol{X}) = \frac{e^{\boldsymbol{X\beta}}}{1 + e^{\boldsymbol{X\beta}}}. \tag{2.9}$$

In Figure 2.1, we see that this function gives us a continuous output, rather than discrete labels. This can be viewed as the probability that x falls within the label where $y = 1$, or $\Pr(y = 1|x)$. These two techniques for input-output mapping are simple models that, while not mapping each point exactly, give a general idea of the trend of the data points. This method will work similarly with both a small are large number of input-output pairs. For cases with more complicated, highly nonlinear, mappings with many more data points available there are more sophisticated methods that can give us better results. Neural networks are a machine learning method that accomplish this.

2.2 Neural Networks

Neural networks are a machine learning algorithm that are able to learn highly nonlinear mappings between an input and output. The machine learning methods described so far are considered shallow learning methods. Given a set of inputs, they directly map to the output from these features. These methods are interpretable, as the importance of each feature can be measured directly by the magnitude of its corresponding coefficient. Neural networks are deep learning methods. Deep learning methods have multiple layers, using the input of the previous layer as the input to the next. In these intermediate layers, latent features are learned from the data which are then used by the final layer to approximate the mapping. Deep learning has the tradeoff of being less interpretable, as feature importance is not as obvious, but has potential to be more accurate with large amounts of data.

The Feed-forward neural network is the simplest form of the neural network. More domain-specialized neural networks are based off of the feed-forward network ar-

chitecture. Feed-forward neural networks consist of many units connected in an acyclic computational graph. Each node of this graph is a unit similar to that seen in the perceptron where its output represented as an affine function of its input, with a nonlinear function is applied to this. The output $\boldsymbol{h}^{(1)}$ of the first layer is,

$$\boldsymbol{h}^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}), \tag{2.10}$$

where $g^{(1)}$ is a nonlinear function known as the activation function. In the case of the perceptron, the activation function is a step function. The output of second layer is then,

$$\boldsymbol{h}^{(2)} = g^{(2)}(\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}). \tag{2.11}$$

This pattern generalizes such that layer n will have the output,

$$\boldsymbol{h}^{(n)} = f^{(n)}(\boldsymbol{h}^{(n-1)}) = g^{(n)}(\boldsymbol{W}^{(n)}\boldsymbol{h}^{(n-1)} + \boldsymbol{b}^{(n)}). \tag{2.12}$$

The output of a neural network can be generalized as a composition of functions such that,

$$\hat{f}(\boldsymbol{x}) = f^{(n)}(f^{(n-1)}(f^{(n)}(...f^{(2)}(f^{(1)}(\boldsymbol{x}))))). \tag{2.13}$$

2.2.1 Motivation For Neural Networks

Neural networks bring to the table some advantages that make them as popular as they are for practical applications today. First and foremost, their parameter space allows them to approximate a mapping of arbitrary complexity and dimension to any desired degree of accuracy, given certain conditions on the activation function [13][19]. What gives them an advantage over other techniques such as a basis expansion is network depth. The number of terms needed in order to represent a function using k basis functions in an n-dimensional space is k^n; an exponential increase with dimensionality. In a multi-layer neural network, the composition of functions across layers reduces the number of parameters needed. It is shown by Eldan and Shamir that to approximate a certain function that requires an exponential number of input nodes with respect to dimensions for a 2-layer neural network, the same function can be approximated with

a 3-layer network with only a polynomial number of input nodes [8]. Another benefit of neural networks is that there is no need for feature engineering. Feature engineering is a common practice in machine learning where the inputs to a model are carefully chosen or created. Neural networks have the ability to learn features from the data, removing user effort and bias from the equation. The input to each hidden layer is the output of the previous layer, which can be seen as a feature learned from its input. Later, a powerful example of this is seen with convolutional neural networks and their application to image processing.

2.2.2 Activation Functions

Each layer of the neural network has an activation function g as seen in Equation (2.12). Without this function, the output would simply be a linear function of the inputs, regardless of the number of layers and nodes. In the perceptron, the activation function is the step function, where the output is thresholded at zero. This step function, however, is not continuous and has no non-zero value for its gradient, which makes the minimization process in training hard to do. This function is instead replaced by a continuous function that behaves similarly: The logistic function seen in logistic regression. This function, also known as the sigmoid function, is more widely replaced by the hyperbolic tangent function tanh, which, as seen in Figure 2.2, is a scaled up version of the sigmoid, bound by $(-1, 1)$ rather than $(0, 1)$. tanh has a stronger gradient, meaning its derivative is steeper. This is beneficial in the training process, as the gradient is used to guide the optimization. However, both of these functions still suffer from the issue of the gradient approaching zero the farther away the function's input gets from 0. This is known as the vanishing gradient problem.

Another common activation function is the rectified linear unit (ReLu), which is one of the most commonly used activation functions. The ReLu is a piecewise linear function, which is applies a nonlinear transformation to its input. It takes the form,

$$g(\boldsymbol{z}) = \max(\boldsymbol{z}, 0). \tag{2.14}$$

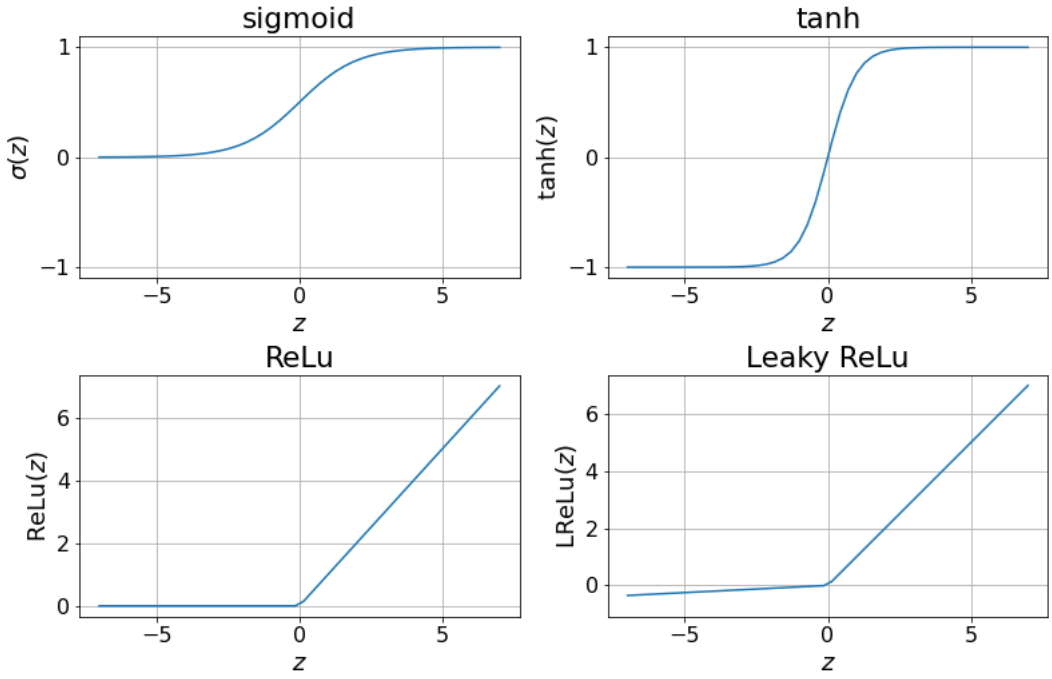


Figure 2.2: Activation functions. It can be seen that $tanh(z)$ is a rescaled version of the sigmoid function $\sigma(z)$, which both mimic the behavior of the step function in a continuous form. The ReLu is a piecewise linear function, which applies a nonlinear transformation to z.

Because it is nearly a linear function, ReLu retains qualities that make it efficient to optimize with gradient-based methods, such as a constant derivative, as long as parameters are initialized with nonzero values [10]. It can be seen that the ReLu will also suffer from a vanishing gradient as its input becomes negative, so there is a change made to mitigate this, giving us the leaky ReLu. The leaky ReLu has a small linear coefficient for negative z, rather than zero so that,

$$g(z) = \begin{cases} 0.01z & z < 0 \\ z & z \geq 0 \end{cases}. \tag{2.15}$$

For the output layer, the activation function is simply linear in the case of regression, and softmax in the case of classification. The softmax function normalizes each output

so they sum to 1, since they each represent the possibility of a given label over the rest. The softmax function for each output node is,

$$g(z)_i = \frac{e^{z_i}}{\sum_{i=i}^{N} e^{z_i}}. \tag{2.16}$$

Consider a case of classification with three possible output classes A, B, and C with corresponding z values in their output nodes of $z_A = 0.72$, $z_A = 0.54$, and $z_A = .22$. The softmax function will transform these into 0.41, 0.34, and 0.25. This means the given data point has a 41% chance of being class A, 34% of being class B, and 25% chance of being class C.

There are many variations of the neural network designed for specialized tasks. One that has been important in the growing popularity of machine learning is the convolutional neural network, which will be described next.

2.2.3 Convolutional Neural Network

The convolutional neural network is a specialized adaptation of the neural network for applications in signal processing. These networks employ a convolutional layer, rather than just the fully connected layers seen so far. Convolutional layers apply a convolution to its input with a kernel that is learned in the training process. A convolutional layer learns a kernel that creates features from its input, which typically is an image. A convolutional network can either pass these features to fully connected layers to perform a regression or classification task, or can use this as a structured, high-dimensional output, giving more flexibility in tasks these networks can be used for. First, the convolution is introduced and examples are given as to why it is useful in image processing.

A convolution is an integral that gives the overlap between two functions as one, called the kernel ($w(a)$), is shifted over the other ($x(t)$) [3]. This can be expressed as,

$$(x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da. \tag{2.17}$$

Figure 2.3: Examples of convolutional filters applied to an image of an apple. From left to right is the original image, then that image convolved with a sharpening kernel, a blurring kernel, and an edge detection kernel called the Sobel filter.

In discrete form, which is how it is used in practice on discrete data points, the equation becomes,

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{2.18}$$

An image can be treated as a three-dimensional tensor, $m \times n$ spatial dimensions, with one grid point for each pixel. A color image has three channels, one each for red, blue, and green. These channels are the third dimension of the tensor. Pixel values range between 0 and 255. To convolve a two-dimensional kernel $\boldsymbol{K}$ with a two-dimensional image $\boldsymbol{I}$, the operation is written as,

$$(\boldsymbol{I} * \boldsymbol{K})(i,j) = \sum_m \sum_n \boldsymbol{I}(m,n)\boldsymbol{K}(i-m, j-n). \tag{2.19}$$

The convolution is commutative, so $\boldsymbol{K} * \boldsymbol{I} = \boldsymbol{I} * \boldsymbol{K}$. In practice, it is simpler to perform $K * I$. Many machine learning libraries implement the a cross-correlation instead of a convolution [10], which is the same as the convolution but with a flipped kernel such that,

$$(K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{2.20}$$

Figure 2.3 gives a few examples of applications of convolution in image processing. A few common examples of convolutional kernels used in image processing are blurring and sharpening kernels, as also seen in 2.3. A blurring kernel takes the average over the

values within the kernel size's distance from the pixel. A 3×3 blurring kernel is,

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{2.21}$$

A sharpening kernel does the inverse, by removing correlation from neighboring pixels. This takes the form,

$$\frac{1}{9}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{2.22}$$

These kernels sum to one, making them a weighted average over the pixels they operate on.

The convolutional layer is the building block of the convolutional network. The convolutional layer's optimizable parameter is its kernel, which learns to create features from the input images. A standard convolutional layer is made up of three parts; the convolution, activation, and pooling. In the initial stage, we perform a convolution of the input $\boldsymbol{x}$ with the kernel $\boldsymbol{w}$ and add a bias $\boldsymbol{b}$ to it.

$$\boldsymbol{z}(\boldsymbol{x}) = \boldsymbol{w} * \boldsymbol{x} + \boldsymbol{b}. \tag{2.23}$$

Kernels typically are three-dimensional like the image, so the convolution gives us an output with one channel. Each layer can have multiple kernels, so the convolution with each kernel is stacked in the output, as shown in Figure 2.4. Next, output is passed through an activation function, such as the ReLu. Third, pooling may be applied. Pooling is effectively downsampling the output image. A common technique for this is max-pooling, or taking the maximum value from an area as the value to represent the entire area. For example, if we wanted to downsample a 4×4 output by a factor of two, it would be resized to 2×2, taking on the max values of each 2×2 quadrant of the original output. The reason for this is two-fold. For one, it reduces the computational cost of later layers while minimally affecting performance. Once a feature is located, its

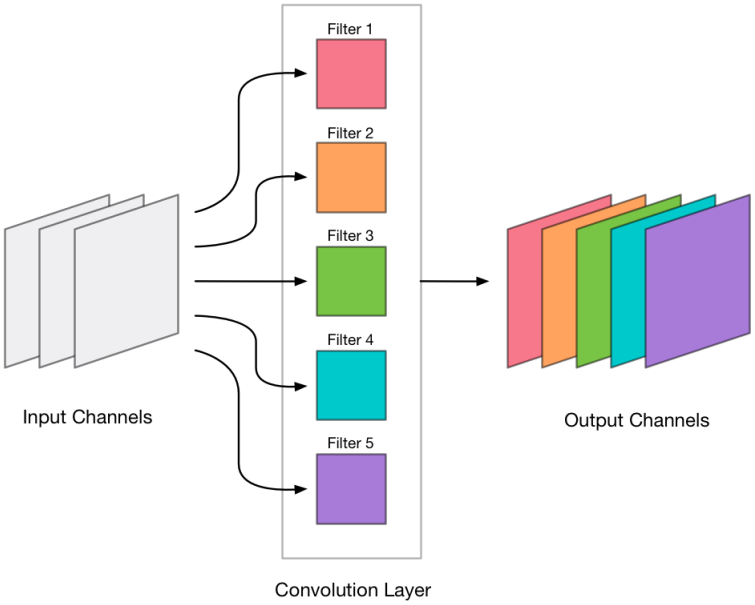


Figure 2.4: A convolutional layer with six kernels, also called filters. Each kernel is convolved with the three-dimensional input tensor, outputting a single channel for each of these operations.

exact location is not as important as its relative location to others. This then helps to prevent overfitting, making the output invariant to small translations in the input [10].

2.3 Assessing Accuracy

We've seen that neural networks can represent input-output mappings. But how is the accuracy of this representation evaluated? In this subsection, methods for model validation and bias-variance trade-off are discussed.

As mentioned earlier, the cost function of a machine learning model is minimized with respect to its parameters. The cost function describes a quantity that we want to minimize, such as an accuracy metric. In the linear regression seen, the cost function is mean squared error of the model output and the true output. Given the true output y and the output of a model $\hat{y}_n = \hat{y}(\boldsymbol{x}_n; \boldsymbol{\beta}_n)$ with parameters $\boldsymbol{\beta}$, the mean

squared error for N data points is,

$$MSE(\boldsymbol{x};\boldsymbol{\beta}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2. \tag{2.24}$$

Mean squared error becomes more sensitive to error as the error increases, penalizing large differences more than small differences. Mean squared error is a convex function given that $y, \hat{y} \in (-\infty, \infty)$. This accuracy metric works well for regression, as it compares two numerical values. For different tasks, different accuracy metrics are necessary. Assessing the accuracy of a classification problem using mean squared error would not guarantee finding a minimum since it is not guaranteed to be convex with binary labels where $y \in \{0, 1\}$ and $\hat{y} \in [0, 1]$. An alternative approach is to maximize the likelihood. In binary classification, the likelihood is the probability that x falls within the label where $y = 1$, or $\Pr(y = 1|x)$. The total likelihood is written as the product of the likelihood of each of N points,

$$L(\boldsymbol{x}) = \prod_{n=1}^{N} \Pr(y_n|\boldsymbol{x}_n). \tag{2.25}$$

Writing this in terms of the output of the model we get,

$$L(\boldsymbol{x};\boldsymbol{\beta}) = \prod_{n=1}^{N} \hat{y}_n^{y_n}. \tag{2.26}$$

In practice, it is easier to take the logarithm of the likelihood to get a sum instead of a product. Instead of maximizing it, cost functions are typically minimized, so this is then made negative. The negative log-likelihood cost function is,

$$L(\boldsymbol{x};\boldsymbol{\beta}) = -\sum_{n=1}^{N} y_n \log \hat{y}_n. \tag{2.27}$$

To evaluate the model using an accuracy metric, an entire data set is not evaluated all together. The simplest method for evaluating a machine learning model is to split the available data set into a training and testing set. Most of the data will be used to train the model, while the remaining data points, which have not been seen in the training process, are then assessed for accuracy. Simply evaluating the accuracy

of the model on the training set can result in a model that does not generalize well to unseen data, which is known as an overfit model. In Figure 2.5, we see two different regression models fit on a training set from the hours studied and test performance example. The linear model on the left performs worse than the polynomial fit on the right if looking at the mean square error of the training set. However, the mean squared error of the test data is much lower for the linear model.

The polynomial fit in this case has a high variance, meaning it is able to vary more with changes in the input. Though this can be a good thing by allowing more complicated functions to be modeled, without a sufficient amount of data points it will tend to model the noise in the small data set, which results in overfitting [17]. The linear model, on the other hand, has a low variance, allowing it to perform similarly on the test and training test. However, it is not as flexible in the functions it can model, leading to a higher error on the data it is trained on compared to the polynomial fit. This inflexibility leads to what is known as the bias. The bottom row of Figure 2.5 visualizes the relationship of this bias-variance tradeoff. It can be seen that the polynomial fit gives lower error on the training set than linear regression, but on the testing set this performance does not translate as well as linear regression. A key challenge in machine learning is choosing a model that has sufficiently low variance and bias for the task, allowing both an accurate and generalizable fit. The next subsection addresses methods to help reach this balance and prevent overfitting.

2.4 Training

In order to get meaningful values for the weights in these machine learning models we have seen so far, they must be trained. The goal of training a machine learning model is to minimize its cost function (E) with respect to its parameters ($\boldsymbol{\beta}$) given a set of input-output pairs $\{\boldsymbol{x}, \boldsymbol{y}\}$. This leaves us with the minimization problem,

$$\min_{\boldsymbol{\beta}} E(\boldsymbol{x}|\boldsymbol{y}; \boldsymbol{\beta}). \tag{2.28}$$

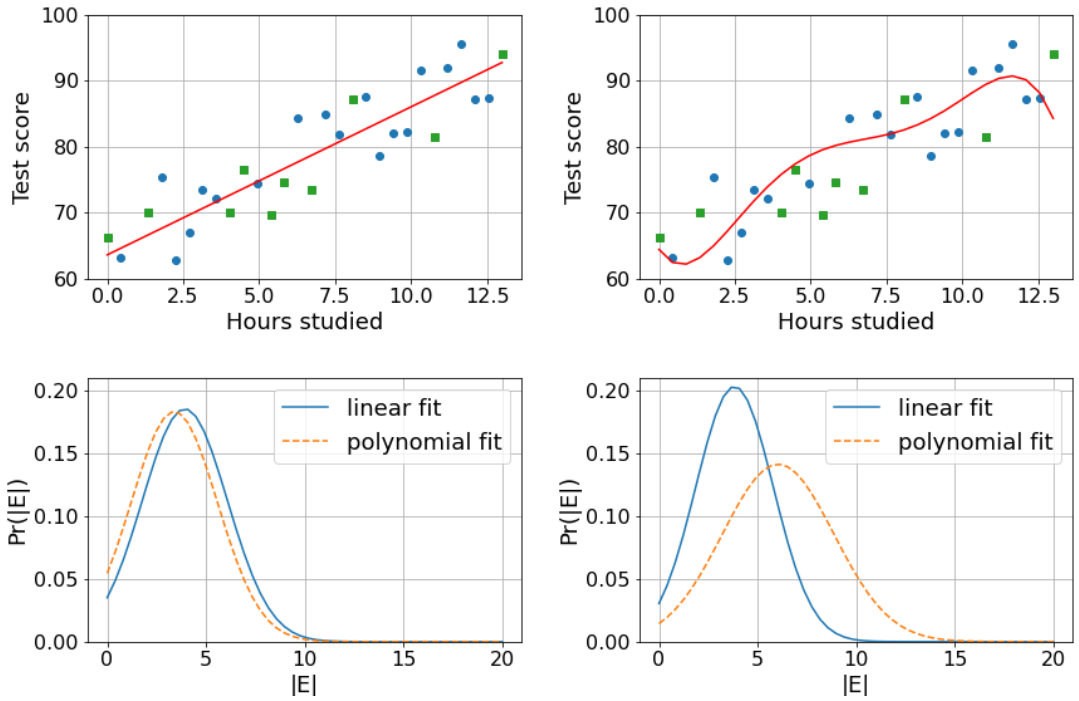


Figure 2.5: Top: The data from Figure 2.1 is split into a training set (blue circles) and a test set (green squares). The training data is fit with linear regression on the left and a fifth-degree polynomial on the right. Bottom: Probability distribution of the absolute error of the training set on the left and test set on the right. The polynomial fit achieves lower mean error (E) on the training set but performs much differently on the test set, while linear regression performs consistently on both. The polynomial overfits the training data, making it unable to generalize well.

Here, some basic concepts of optimization and a family of methods used to train neural networks is covered. To solve a minimization problem such as Equation (2.28), a local minimum $\boldsymbol{\beta}^*$ is found. Since it is non computationally feasible to search the entire parameter space of the function, there is no guarantee of finding a global minimum unless the function being minimized is convex, meaning there is only one local minimum. At a local minimum, the gradient of the cost function will be zero, and its hessian will be positive semidefinite [22]. These give us the necessary condition of local minimization. Considering the minimization of function $f(\boldsymbol{x})$, the necessary conditions are,

$$\nabla f(\boldsymbol{x}^*) = 0, \quad \nabla^2 f(\boldsymbol{x}^*) \geq 0. \tag{2.29}$$

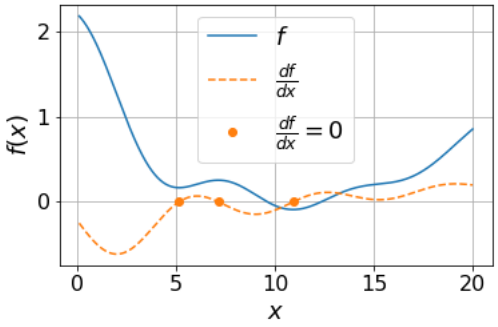


Figure 2.6: A plot of a function and its derivative. It can be seen that the maxima and minima of f correspond to the points where its derivative are equal to zero. Only the points where $\frac{df}{dx}$ is increasing are the minima.

For the case of a convex loss function we can find an analytical solution to the minimum, such as with linear regression. The cost function for squared error is,

$$E(\boldsymbol{\beta}) = ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2. \tag{2.30}$$

Taking the partial derivative with respect to $\boldsymbol{\beta}$ and equating it to zero,

$$\frac{\partial E}{\partial \boldsymbol{\beta}} = 2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0, \tag{2.31}$$

we can rearrange to arrive at what is known as the normal equation for least squares,

$$\boldsymbol{\beta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.32}$$

For many cost functions in practice, there may not be a closed form solution for the minimum or it may be inefficient to find, as matrix operations of high dimension can be computationally expensive. For these cases, there are iterative methods that are used for minimization. Line search is one of these methods, and is the basis of the most commonly used neural network optimization methods. Line search updates the parameters iteratively such that,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \alpha_k \boldsymbol{p}_k, \tag{2.33}$$

where α is a step length and p is a step direction. First, the search direction is determined, then a suitable step length is found. Generally, $\boldsymbol{p}$ is in a descent direction, meaning the function being minimized is decreasing along the direction, so $\boldsymbol{p}_k^T \nabla E_k < 0$. To determine the step length, we want to take a step that sufficiently decreases the cost function, but isn't computationally inefficient. A popular set of conditions used for determining the step size are the Wolfe conditions. These first require a sufficient decrease by the condition,

$$E(\boldsymbol{\beta}_k + \alpha \boldsymbol{p}_k) \leq E(\boldsymbol{\beta}) + c_1 \alpha \nabla E_k^T \boldsymbol{p}_k, \tag{2.34}$$

where $c_1 \in (0, 1)$. The step size is then prevented from being unreasonably small by the second condition,

$$E(\boldsymbol{\beta}_k + \alpha \boldsymbol{p}_k)^T \boldsymbol{p}_k \geq c_2 E_k^T \boldsymbol{p}_k, \tag{2.35}$$

where $c_2 \in (c_1, 1)$. Typically, c_1 is chosen to be close to 0, such as 10^{-4}, and c_2 is chosen between 0.1 and 0.9, depending on the direction method [22]. Since our goal in optimization is to approach a minimum, our step direction should be in a direction of descent. The simplest direction of descent is the one of steepest descent, or the negative gradient. This direction, also known as gradient descent, updates the parameters at each iteration so that,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \alpha \nabla E(\boldsymbol{x}; \boldsymbol{\beta}). \tag{2.36}$$

Following the direction of descent, this method will eventually arrive at a minimum of the cost function. In a case of a convex function, where there is a single global minimum, this gets a satisfactory result. In the case of a non-convex cost function, as with neural networks, this can lead to gradient descent converging on an unsatisfactory local minimum. We can rewrite E as the sum of the mean squared error of N training points,

$$E(\boldsymbol{x}|\boldsymbol{y}; \boldsymbol{\beta}) = \frac{1}{N} \sum_{n=1}^{N} \left[\boldsymbol{y}_n - \hat{\boldsymbol{y}}(\boldsymbol{x}_n; \boldsymbol{\beta}) \right]^2. \tag{2.37}$$

One approach that can be taken to help prevent the optimization from falling into a local minimum is instead updating $\boldsymbol{\beta}$ with only a small, random subset of training data

each iteration so that,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \alpha \nabla E(\boldsymbol{x'}|\boldsymbol{y'};\boldsymbol{\beta}), \quad \{\boldsymbol{x'}, \boldsymbol{y'}\} \subseteq \{\boldsymbol{x}, \boldsymbol{y}\}. \tag{2.38}$$

This is called stochastic gradient descent. In Figure 2.7, we see an example of batch gradient descent where the entire set of training points is used in each update, and an example of stochastic gradient descent used to minimize the cost function for linear regression. Although stochastic gradient descent doesn't follow the direction of steepest descent exactly, it still arrives at the same minimum. With an incomplete gradient being used in each step of stochastic gradient descent, line search will not give a step size satisfactory for the optimization problem over the complete training set. In practice, the common approach for the step length is to choose a constant value. The step length, also called the learning rate, can also be given a decay rate so steps become more refined as training goes on. The step length can be linearly decayed until it reaches a minimum size,

$$\alpha_k = \left(1 - \frac{\lambda}{\tau}\right)\alpha_0 + \frac{\lambda}{\tau}\alpha_\tau. \tag{2.39}$$

where λ is a decay coefficient and τ is a fixed iteration number where the decay reaches the minimum step length. As seen in Figure 2.8, decaying learning rate allows for more precise steps toward the end of training, getting closer to the minimum point. Sufficient conditions on the step length for stochastic gradient descent to converge are (see [10]),

$$\sum_{k=1}^{\infty} \alpha_i = \infty, \quad \sum_{k=1}^{\infty} \alpha_i^2 < \infty. \tag{2.40}$$

The most popular neural network optimization methods today are modifications of stochastic gradient descent. One of these methods is momentum. Momentum aims to speed up the optimization process by multiplying the learning rate by a factor of the previous iteration's step. Each iteration, $\boldsymbol{\beta}$ is updated as,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \boldsymbol{v}, \tag{2.41}$$

with,

$$\boldsymbol{g} = \nabla E(\boldsymbol{x'}|\boldsymbol{y'};\boldsymbol{\beta}), \tag{2.42}$$

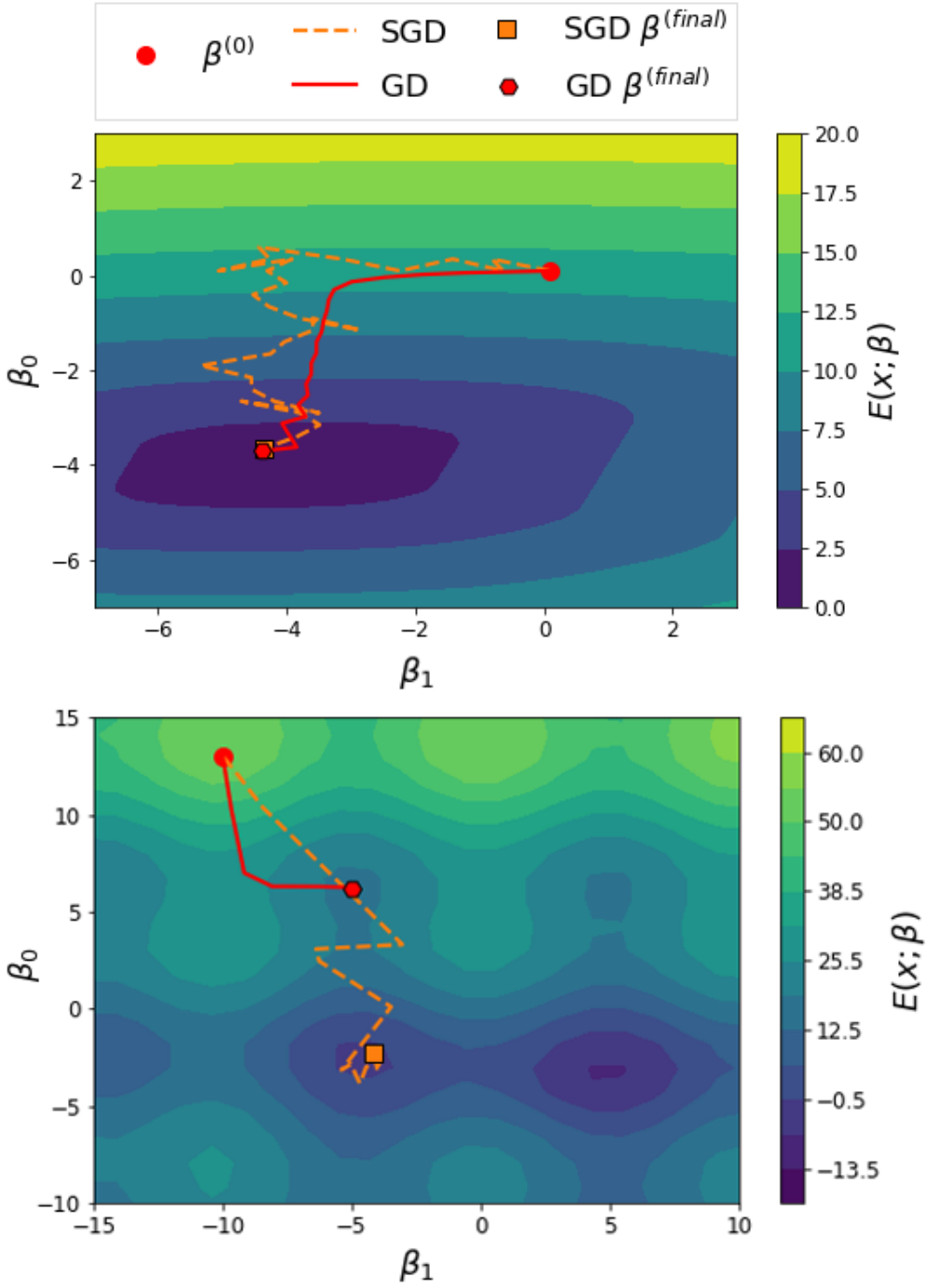


Figure 2.7: Top: State space of batch gradient descent and stochastic gradient descent used to train linear regression, where $y = -4x - 4$. Here, stochastic gradient descent is updating each step with the gradient of only one data point at a time. This leads to a path that does not follow the direction of steepest descent as closely as batch gradient descent, but arrives at the same minimum. Each method uses the constant learning rate $\alpha = 0.1$. Bottom: The two gradient descent methods used on a non-convex loss function. It can be seen that stochastic gradient descent does not get stuck in the smaller local minima as batch gradient descent does due to the variation in its direction.

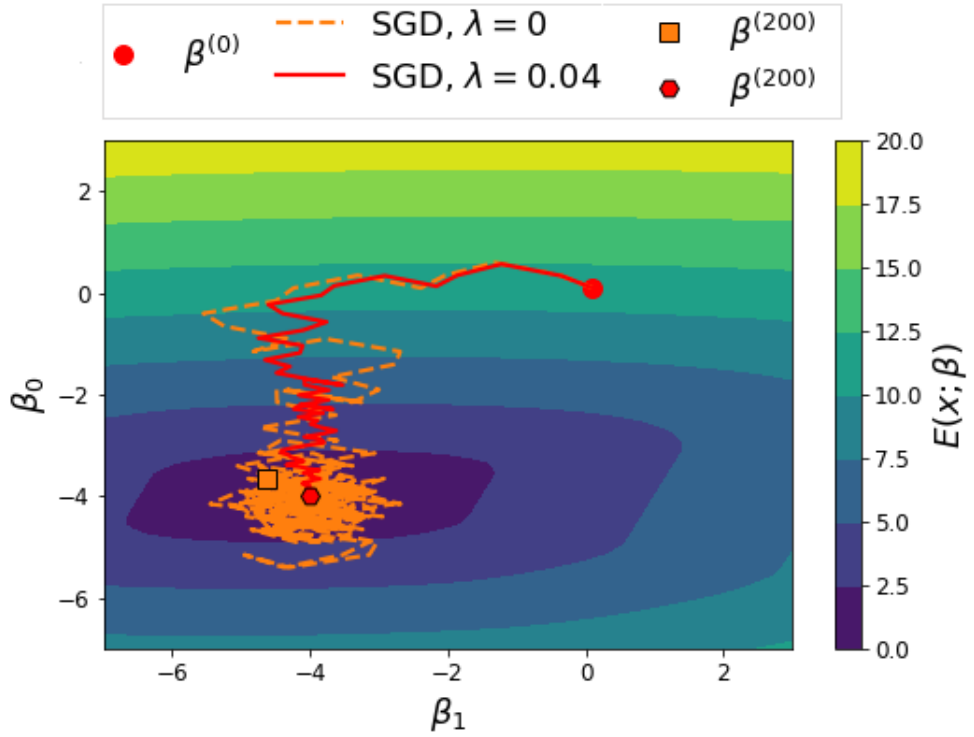


Figure 2.8: Training of the linear regression problem from Figure 2.7 using stochastic gradient descent with and without learning rate decay. An initial step size of $\alpha = 0.25$ is used. As each converges on the minimum, it can be seen the learning rate decay allows a more precise minimization. The larger step size does not allow the flexibility to get as close to the minimum, leading to the oscillations seen crossing over it.

$$\boldsymbol{v} = \eta \boldsymbol{v} - \alpha \boldsymbol{g}, \tag{2.43}$$

where $\eta \in [0, 1)$ is a predetermined parameter [10], and $\boldsymbol{v}$ must be initialized. Momentum is a physical analogy for the velocity of a ball rolling down a hill in this method. As it rolls down the hill, much like the optimization descends toward a minimum, the ball will gain speed. The larger η is, the more the previous iteration will affect the next one. This momentum also helps to escape local minima, as seen in Figure 2.9. Another family of gradient descent methods used to speed up training are adaptive learning rate algorithms. These methods use a separate learning rate for each parameter and update them throughout the optimization process. One of the best performing adaptive methods is the Adam optimizer [18]. Adam, short for adaptive moments, updates individual

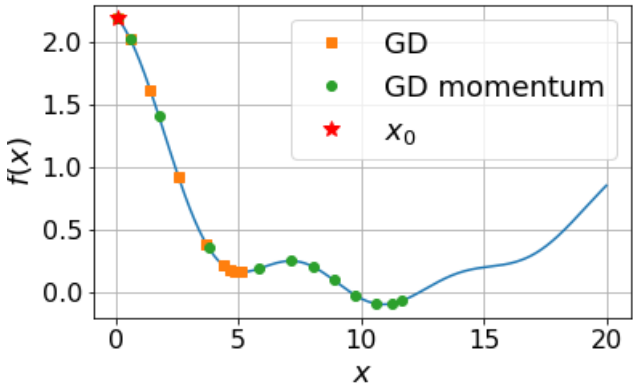


Figure 2.9: Gradient descent with momentum gradually increases the step size during the descent, allowing it to escape the local minimum at 5, which normal gradient descent converges to.

learning rates based on previous learning rates in an exponential moving average. Each parameter is updated similarly to momentum, where,

$$\boldsymbol{v} = \eta \boldsymbol{v} - \frac{\alpha}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}, \tag{2.44}$$

and the $\boldsymbol{r}$, the accumulation of the gradient, is updated as,

$$\boldsymbol{r} = \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}, \tag{2.45}$$

and $\odot$ is the element-wise product. Although Adam exhibits faster convergence to a minimum, it has been shown to not reach an optimal solution as well as regular stochastic gradient descent, which is more likely to reach a value close to the global minimum [37] [32].

2.4.1 Training Neural Networks & Backpropagation

We've seen that optimization algorithms for machine learning rely on the gradient of the cost function with respect to the parameters. In the case of the neural network, the multi-layer computational graph makes the process not as straightforward, and an efficient approach must be taken. In the forward propagation of a neural

network, the input $\boldsymbol{x}$ is passed through each node of the first layer, from where its output flows to each node of the next hidden layer, continuing until the output $\boldsymbol{y}$ is produced. In order to get the gradient, this path is followed in reverse; from the output to each individual node. This process is called back-propagation. To understand how back-propagation is implemented, we will first consider the chain rule of calculus, which allows us to find the partial derivative of the neural network's output with respect to any given weight. For two functions $y = g(x)$ and $z = f(y)$, their composition would be, $z = f(g(x))$. The chain rule tells us that,

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{2.46}$$

For the case of vector input and output $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$, this rule can be generalized to,

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}. \tag{2.47}$$

In vector notation this becomes,

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z. \tag{2.48}$$

$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is a $n \times m$ Jacobian matrix of the above function $g(\boldsymbol{x})$. We can use this result to compute the gradient of the neural network output with respect to each parameter, but working backwards through the computational graph we see many expressions are frequently reused, such as, say, the gradient of a final layer node. It would be computationally efficient to do approach this in a way that does not recalculate these repeated subexpressions.

Consider the computational graph in Figure 2.10. The input is w, and the output is z, with a function f applied three times in between these value. The gradient of z with respect to w, is found by,

$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}. \tag{2.49}$$

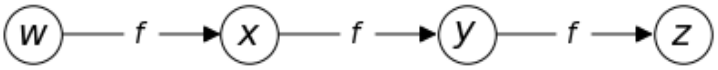


Figure 2.10: A simple computational graph. Equation (2.50) describes back-propagation used to get the gradient of x with respect to w.

In terms of the function f, this becomes,

$$\frac{\partial z}{\partial w} = f'(y)f'(x)f'(w) \tag{2.50}$$

$$= f'(f(f(w)))f'(f(w))f'(w). \tag{2.51}$$

In this case, back-propagation will compute $f(w)$ only once, saving us a function evaluation. In the case of a deep neural network with many neurons each layer, this saves a significant amount of computation and is a significant reason that neural networks are computationally practical.

2.4.2 Regularization Methods

In training, it is necessary to prevent overfitting the training data. As seen in the section on assessing the accuracy of a machine learning model, a model with a larger number of parameters can easily overfit without a substantial amount of data. Regularization is a method to reduce a fit's variance, making it more able to generalize to new data. The two common techniques that will be addressed in this section do this by putting constraints on the fit parameters in the cost function [17]. These methods are L1 and L2 regularization. Consider again the case of linear regression. L2 regularization constrains the parameters $\boldsymbol{\beta}$ by penalizing their 2-norm, turning the cost function from Equation (2.30) into,

$$E(\boldsymbol{\beta}) = ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2 + \lambda||\boldsymbol{\beta}||_2^2, \tag{2.52}$$

where λ is a positive coefficient. In this setting, $\boldsymbol{\beta}^*$ becomes,

$$\boldsymbol{\beta}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.53}$$

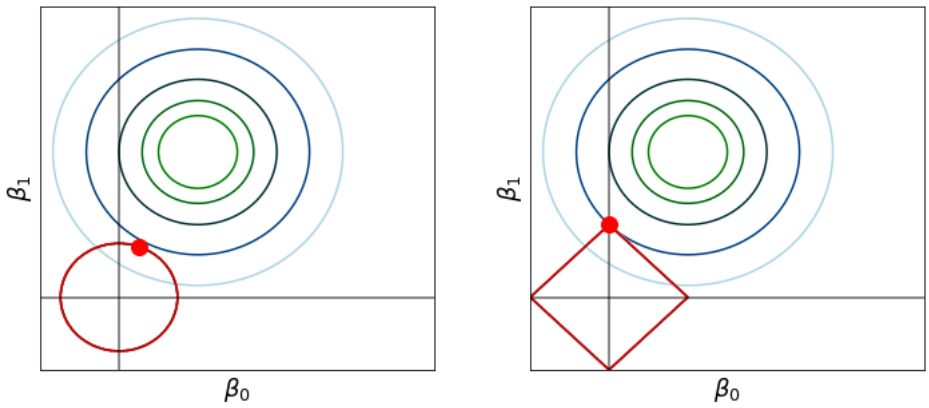


Figure 2.11: L1 and L2 regularization visualized. The red contour line can be seen as the bound placed on the two parameters β_0 and β_1 by the regularization, L2 on the left and L1 on the right. The blue/green contour shows the minimum parameter values for the cost function by itself. The red point shows where the regularized cost function is minimized.

The matrix $(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}$ is changed by a constant factor on the diagonal from its original form [10]. Penalizing the L2 norm effectively puts an upper bound on the square of the magnitude of the parameters, which is proportional to λ. This is visualized in Figure 2.11. In L1 regularization, the magnitude of the parameters is penalized rather than the square of the magnitude. The previous cost function with this form of regularization becomes,

$$E(\boldsymbol{\beta}) = ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2 + \lambda||\boldsymbol{\beta}||_1. \tag{2.54}$$

This type of regularization enforces sparsity on the parameters, meaning less important parameters tend towards zero. For this reason, L1 regularization is useful for deciding feature importance and ultimately, feature selection.

2.5 Transfer Learning

Neural networks are able to approximate any mapping given a sufficient number of nodes and layers, but without a sufficient amount of training data, this approx-

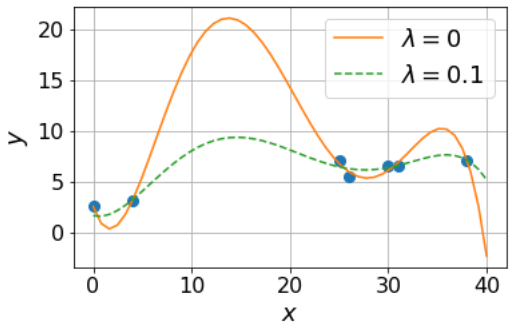


Figure 2.12: Comparison of polynomial regression on a small number of data points without regularization and with L2 regularization where $\lambda = 0.1$.

imation will probably not generalize well to other data. This can be a problem in applications where data is costly to collect, generate, or label. A method to limit the amount of new data needed to create accurate neural networks is transfer learning [26]. Transfer learning takes the latent features learned by a trained neural network and applies them to a similar task. Ideally these latent features are transferable, circumventing the need to re-learn them for the new task [14]. A common method to performing transfer learning is freezing the hidden layers of a trained neural network, re-initializing its output layer, and re-training the network with the new data. This method is visualized in Figure 2.13. Transfer learning in this manner can be thought of as using the hidden layer of a trained neural network as the input features to a single-layer neural network. Another technique for transfer learning is fine-tuning, which only involves initializing a new neural network with the trained model's weights, allowing all of the weights to be adjusted for the new data during training.

Transfer learning is widespread in the domain of computer vision, where it has been shown that features extracted from convolutional neural networks are generalizable and perform better than traditional feature extraction techniques used in image processing [31]. An example of this is seen in using a neural network pre-trained on the ImageNet dat aset for melanoma identification in medical imaging. ImageNet is a large-scale, publicly available data set consisting of 14 million images of common ob-

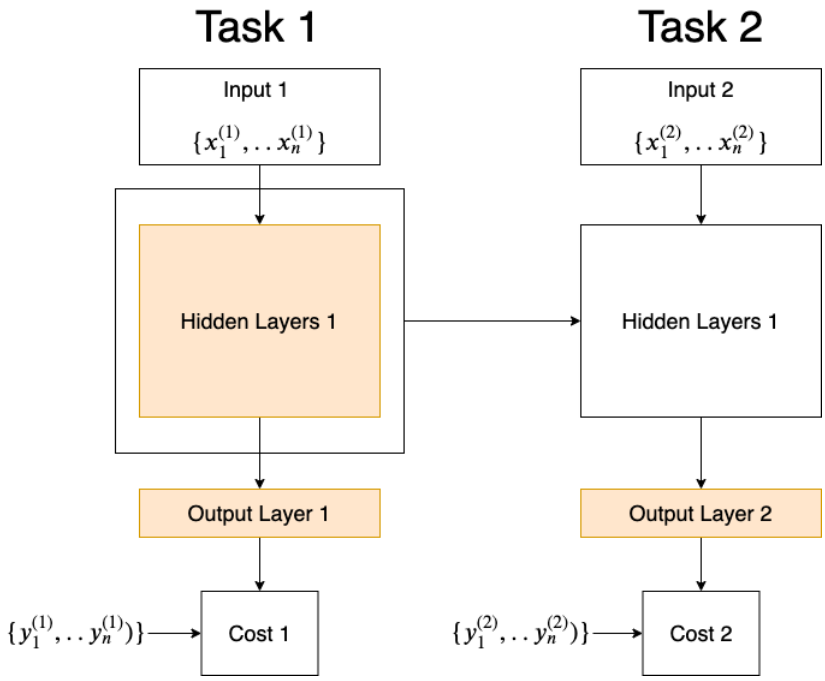


Figure 2.13: A visualization of transfer learning. A neural network is trained for source Task 1 given a data set of input-output pairs $\{x^{(1)}, y^{(1)}\}$. The shaded boxes indicate portions whose weights are updated during training. The trained hidden layer is then used to train a model for target Task 2, a similar task to the source task with a different domain, $\{x^{(2)}, y^{(2)}\}$. In this model, the hidden layers are frozen and only the output layer's parameters are optimized.

jects spanning 22,000 unique labels [34]. A convolutional neural network pre-trained on ImageNet with its final layer trained with a data set of roughly 2,000 images for melanoma identification out-performed a convolutional neural network trained directly on the same data set across three different identification tasks, achieving up to 10% better accuracy. [21]. Furthermore, it was also shown that transfer learning from a smaller, more related data set of 35,000 retinopathy images was not as effective as with ImageNet, showing the robustness of the features learned from the exhaustive ImageNet data set.

2.6 Neural Networks For Modeling Atomic Systems

Discovery of material properties for a wide range of applications, such as electronics and medicine, rely on computational methods like density functional theory. As seen in Section 1, density functional theory calculations can be computationally expensive, so accelerating these is necessary for efficient high-throughput screening of materials. High-throughput screening relies on computationally finding properties of a wide configuration space of materials rather than carefully picking a smaller subset to test. With the increase of computing power available to researchers, this approach has become more feasible, although it still has room for further speed-up. Machine learning methods such as neural networks are a good candidate to do this. Once a neural network is trained, the prediction calculation is very fast and consistent in its computational performance, taking milliseconds to make a prediction that may take DFT minutes or hours to calculate. Additionally, their ability to represent highly non-linear relationships gives them the potential to model a material's properties to a high degree of accuracy. If able to represent the mappings between a material's structure and these properties to relatively close to chemical accuracy, neural networks can be an effective surrogate model to improve computational performance of density functional theory calculations.

Deep learning has helped to make recent advances in prediction of chemical properties. Previously, shallow machine learning approaches relied on input features manually engineered for specific applications. Deep learning methods circumvent this feature engineering approach by learning features directly from the data, given constraints consistent with the underlying physics. In this section, an overview of deep learning methods used for predicting chemical properties of materials is given.

2.6.1 Atomistic Neural Networks

One of the first developments for neural networks specialized for modeling atomic systems was made by Behler and Parrinello in 2007, where they introduced

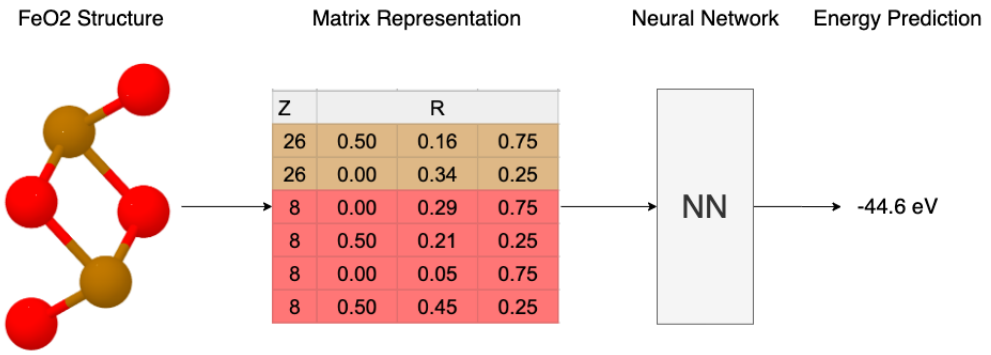


Figure 2.14: Atomistic neural network approach to predicting properties. The structure of a material can be represented by a matrix consisting of each atom's atomic number (Z) and its Cartesian coordinates (R) with respect to a reference point. This representation will be used as the input to the neural network, which maps it to a target property, such as an energy.

a method to represent individual contributions from atoms to total molecular energy, allowing models that scale well with system size [15]. A standard, fully-connected feed-forward neural network is limited in its ability to model atomic systems because of its fixed input size. The atomistic neural network applies the same weights to the transformed input for each atom, allowing a flexible input size. Outputs for each of these atoms, seen as their individual contribution to the total energy, are then summed to get the molecule's total energy.

Another limitation of a standard neural network is representation of the rotational invariance of a molecule. Regardless of its orientation, a molecule's properties should remain constant, which is a hard task for a neural network to model with a limited amount of data. To solve this problem, the Cartesian coordinates of the atomic positions are transformed into rotationally invariant symmetry functions. These symmetry functions, which describe the local environment of each atom, consist of both radial and angular symmetry functions. The radial symmetry functions for an n-atom system are written as a summation of gaussians with parameters η and R_s,

$$G_i^R = \sum_{j \neq i}^{n} e^{-\eta(R_{ij}-R_s)^2} f_c(R_{ij}). \tag{2.55}$$

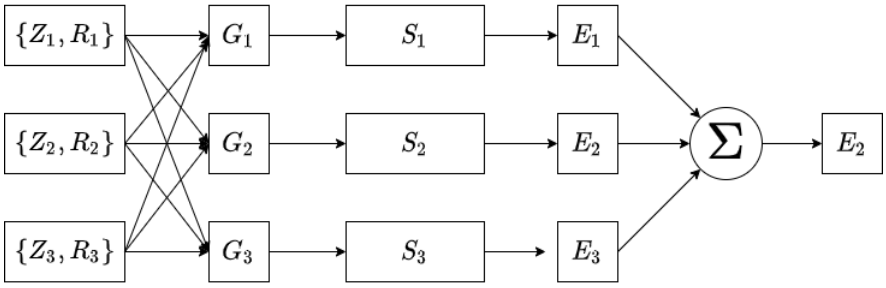


Figure 2.15: The atomistic neural network proposed by Behler and Parrinello, shown with a three atom system modeling the total energy E. Each atom, from its atomic number and position, are embedded into symmetry functions G, describing its local environment relative to its neighbors. These symmetry functions are then each passed to a subnet S, each of which is a neural network with the same weights. The result of this subnet is the individual contribution to the total energy from each atom, which is then aggregated to get the total energy.

f_c is a cutoff function, limiting the number of symmetry functions to an immediate vicinity of the atom. It is given as,

$$f_c(R_{ij}) = \begin{cases} 0.5\left(\cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1\right) & R_{ij} \leq R_c, \\ 0 & R_{ij} > R_c. \end{cases} \tag{2.56}$$

The angular symmetry functions, which ensure a smooth decay to zero for large interatomic distances, has the form,

$$G^A = 2^{1-\xi} \sum_{j,k\neq i}^{n} (1 + \lambda \cos\theta_{ijk})^{\xi} e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} f_c(R_{ij}) f_c(R_{jk}) f_c(R_{ik}), \tag{2.57}$$

where $\theta_{ijk} = \frac{R_{ij}R_{ik}}{R_{ij}R_{ik}}$, and ξ is another parameter. The number of symmetry functions used is a model hyperparameter, where each unique symmetry function has different set parameter values. The symmetry function for each atom is then passed through a neural network, whose output is a single value identified as the atom's individual contribution to the total energy. These individual atomic energies are then summed to reach the total system's energy.

2.6.2 SchNet

SchNet is an improvement on the previous atomistic neural network. It retains the approach of atom-wise contributions and rotationally invariant representation, and adds more powerful tools used in modern neural networks, such as the convolution [35].

Convolutional layers are the state of the art for machine learning with spatial data, but typically these are discretized, such as pixels of an image. Molecular structure does not lie on a grid such as these signals. Although it can be discretized, it requires choosing a proper interpolation scheme and typically a large number of grid points for proper representation that can capture subtle positional changes of atoms. Continuous-filter convolutional layers are implemented in SchNet, getting around this problem by applying a convolution element-wise. Given feature representations of n objects $\boldsymbol{X}^l = (\boldsymbol{x}_1^l, ..., \boldsymbol{x}_n^l)$ at locations $\boldsymbol{R}^l = (\boldsymbol{r}_1^l, ..., \boldsymbol{r}_n^l)$, the output of continuous convolutional layer l at position $\boldsymbol{r}_i$ is,

$$\boldsymbol{x}_i^{l+1} = (\boldsymbol{X}^l * \boldsymbol{W}^l)_i = \sum_{j=1}^{n} \boldsymbol{x}_j^l \odot \boldsymbol{W}^l(\boldsymbol{r}_i - \boldsymbol{r}_j), \tag{2.58}$$

where $\odot$ is the element-wise product. In the continuous-filter convolutional layer, the positions, the distances $d_{ij} = |\boldsymbol{r}_i - \boldsymbol{r}_j|$ are expanded with radial basis functions,

$$e_k(\boldsymbol{r}_i - \boldsymbol{r}_j) = \exp\big(-\gamma |d_{ij} - \mu_k|^2\big), \tag{2.59}$$

located at centers $0\mathring{A} \leq \mu_k \leq 30\mathring{A}$ with $\gamma = 10\mathring{A}$. Introducing this additional nonlinearity causes filter to be less correlated, since the network after initialization is close to linear. This speeds up the beginning of the training process, which may plateau otherwise [35].

In addition to the radial basis functions, each atom of a system is represented by an embedding unique to its atomic number. This embedding is a vector of a predefined length F that is refined through each layer l of the network. The feature representations for an n-atom system for a layer are $X^l = (\boldsymbol{x}_1^l, ..., \boldsymbol{x}_n^l)$, with $\boldsymbol{x}_i^l \in \mathbb{R}^F$. Each feature vector is initialized randomly for each Z_i such that,

$$\boldsymbol{x}_i^0 = \boldsymbol{a}_{Z_i}, \tag{2.60}$$

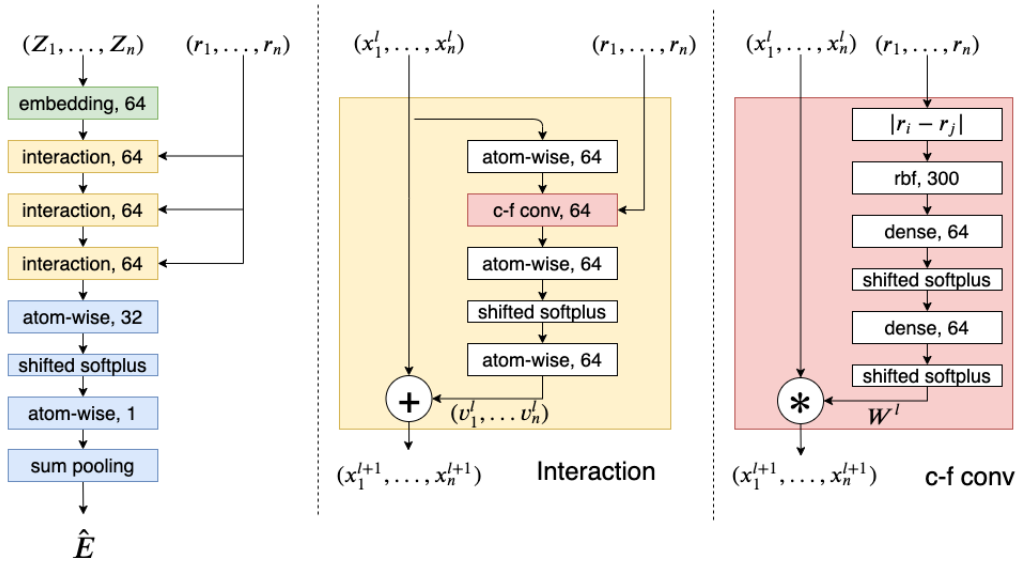


Figure 2.16: Architecture of SchNet using a feature size of 64 and three interaction blocks. The interaction block is shown in the middle, and the continuous-filter convolutional layer on the right.

and is refined during training.

2.6.2.1 Architecture

The SchNet architecture consists of the previously described features in blocks called the interaction blocks. Each interaction block refines the feature representations, which are then passed to a final set of atom-wise layers and are pooled to reach the output value. Figure 2.16 shows the full form of this output. Rather than each interaction block being a composition of the previous, as typically done with neural network layers, each uses a residual connection. The features are updated each layer as,

$$\boldsymbol{x}_i^{l+1} = \boldsymbol{x}_i^l + \boldsymbol{v}_i^l. \tag{2.61}$$

This connection helps to prevent overfitting, as it is easier for $\boldsymbol{v}_i^l$ to become zero in the training process if the next layer is unnecessary by minimizing the residual between

$\boldsymbol{x}_i^{l+1}$ and $\boldsymbol{x}_i^l$. Without a residual connection, the next layer would be updated as,

$$\boldsymbol{x}_i^{l+1} = f^l(\boldsymbol{x}_i^l), \tag{2.62}$$

requiring f^l to learn the identity function, which is a non-trivial task. The activation function used is the shifted softplus. It is defined as,

$$ssp(x) = ln(0.5e^x + 0.5). \tag{2.63}$$

This function is a smooth approximation of the ReLu, and $ssp(0) = 0$ in order to improve convergence of the network.

The atom-wise layer seen in the architecture applies an affine transformation to the features of each atom separately. This layer shares the same weights throughout every atom, giving the output,

$$\boldsymbol{x}_n^{l+1} = \boldsymbol{W}^l \boldsymbol{x}_i^l + \boldsymbol{b}^l \tag{2.64}$$

for atom i. The atom-wise layer is a concept seen in the atomistic neural network discussed earlier. The sharing of weights across atoms allows for the network to scale with the size of the system properly.

2.6.2.2 Results

Results of SchNet were given for the QM9 data set. QM9 is a database of roughly 130,000 molecules consisting of up to nine atoms of C, H, O, N, and F. Each molecule has fifteen chemical properties, including free energy G and total energy U_0 [33][30]. The authors focus on the results of the total energy calculations. Table 2.1 gives the resulting mean absolute error of evaluating the validation set with an SchNet model trained using different amounts of training data.

2.6.3 Transfer Learning For Modeling Molecular Properties

Neural networks such as SchNet achieve impressive accuracy on large data sets such as QM9. However, this level of accuracy may not be attainable in applications

N	MAE (eV)
50,000	0.026
100,000	0.015
110,462	0.013

Table 2.1: Validation set Mean absolute error of SchNet on the U_0 from the QM9 dataset with varying amounts of training data. For 50,000 data points and up, error remains within chemical accuracy.

without a large amount of data for a specific task available. Transfer learning, which has shown promise in other fields, is a potential candidate to improve the effectiveness of SchNet with these smaller data sets. Transfer learning has been used to improve molecular property prediction for very small data sets, with as few as 19 observations [38]. In that approach, 1,000 neural networks pre-trained for various tasks were used for transfer learning on predicting heat capacity and thermal conductivity of polymers, showing considerable improvement over directly trained networks for a variety of base data sets. This method may be viable for very small data sets, but with hundreds or thousands of data points, training many networks without discretion can become computationally expensive, so it would be desirable to train fewer neural networks with a higher confidence in their transferability.

Another application of transfer learning seen in tandem with density functional theory is in improving the accuracy of low-fidelity DFT results. Coupled cluster methods such as coupled cluster single double triple (CCSD(T)) are a generally more accurate technique for solving many-body wave functions than density functional theory but are much more computationally expensive. Demonstrated in [36], a modified Behler-Parrinello atomistic network trained on a large data set of DFT calculations for organic molecules was used for transfer learning with a smaller set of accurate coupled cluster calculations of similar molecules. The accuracy of the resulting model's predictions approached that of CCSD(T) calculations, surpassing the accuracy of DFT [36].

Chapter 3

Predicting Free Energy Of Transition Metal Oxides With SchNet

The goal of using machine learning in place of density functional theory is to speed up calculations, which is especially useful for high-throughput screening of materials. The goal of this work is to apply machine learning for screening of transition metal oxides. Transition metal oxides are used for solar energy conversion. However, poor conductivity and electron-hole separation limits their carrier conductivity. It has been shown that appropriate doping (adding of impurity) of these materials may improve their utility. An important property to be found in these doped transition metal oxides is a low defect formation energy. The defect formation energy is the difference between the total free energy of the pure transition metal oxide and that of the impure, doped transition metal oxide. Although there are other properties of importance, this work focuses on the learning of the mapping between transition metal oxides and their free energy.

To get the speedup of using a neural network for screening materials, some examples are needed to train it, adding an overhead computational cost if these are not materials with properties already available and DFT is needed to get them. Because of this, it is desirable to have a training method that is data-efficient; able to learn an accurate mapping with relatively little data as opposed to tens or hundreds of thousands

of examples as demonstrated so far. This chapter describes how a data set of transition metal oxide DFT calculations is put together and reports the results of using this data for training SchNet networks with a variety of methods.

3.1 Data Set Creation

In order to create a data set of transition metal oxides, an open-source data base of DFT results was used. AFLOW makes outputs from DFT calculations directly available, including over 3 million material compounds currently [6]. From the AFLOW database, a small set of 517 transition metal oxides including Iron, Titanium, Vanadium, and Oxygen was created. These materials each have between 2 and 110 atoms, with a mean of 12. An extension of this data set was then created, introducing an additional 146 transition metal oxides consisting of Chromium and Manganese.

A second data set was created for the purpose of transfer learning use the Materials Project database. The Materials Project curates a database of materials with information regarding their structure and properties [24]. This subset consisted of materials made up of 87 elements, including those of the target data set, Ti, Fe, V, and O. Materials of the same unit cell formula as the transition metal oxide data set were excluded for the sake of preventing overlap between the two data sets. This means that materials with the same composition as one in the transition metal oxide set, even if they had a unique geometry, were not included in this data set. This data set includes 50,000 materials used for training, and 10,000 for validation. A smaller subset of this data set was created, also excluding Mn and Cr for the purpose of transfer learning with the extended transition metal oxide data set.

3.2 SchNet Direct Training Results

To get a performance benchmark for the Transition metal oxide (TMO) data set, SchNet was trained on it on the full training set. Two different model architectures and regularization schemes were used. The architecture used for training on the QM9

N	arch 1	arch 1+L2	arch 2	arch 2+L2
400	0.563	**0.551**	1.175	1.005

Table 3.1: Validation set mean absolute error in eV of SchNet models trained on the full TMO training set. Arch 1 is the larger architecture described and arch 2 is the smaller architecture. L2 signifies the inclusion of L2 regularization with a coefficient of 10^{-3} in the loss during training.

data set by the creators of SchNet was first considered, with and without L2 regularization on all weights. This architecture consisted of six interaction blocks, 128 length embedding vectors and 128-filter convolutional layers. Since the TMO data set is considerably smaller than QM9, a smaller architecture was also trained with and without regularization in an attempt to prevent potential overfitting. This architecture had four interaction blocks, with embeddings of length 30 and convolutional layers with 30 filters. For each model trained, Adam optimizer is used to minimize the mean squared error with an initial learning rate of 10^{-4}, which is reduced by a factor 0.8 when training loss plateaus for 25 epochs until a minimum learning rate of 10^{-6} is reached. The results of the validation set evaluated by these models are reported in Table 3.1. As expected, with the small amount of data and a more diverse data set, the error is significantly higher than what is achieved with the QM9 data set. The model with the best mean absolute error was the original architecture with an L2 regularization coefficient of 10^{-3}. Out of the validation set, only 47 of the 117 predictions were within chemical accuracy. This best-performing model is used to compare with transfer learning models in the following sections.

3.3 Transfer Learning With SchNet

In order to improve the results of SchNet with the small data set, transfer learning was implemented in three different schemes with the Materials Project data set used for training the initial network:

1. Initialize the network with pre-trained weights, fine-tune all by training on the

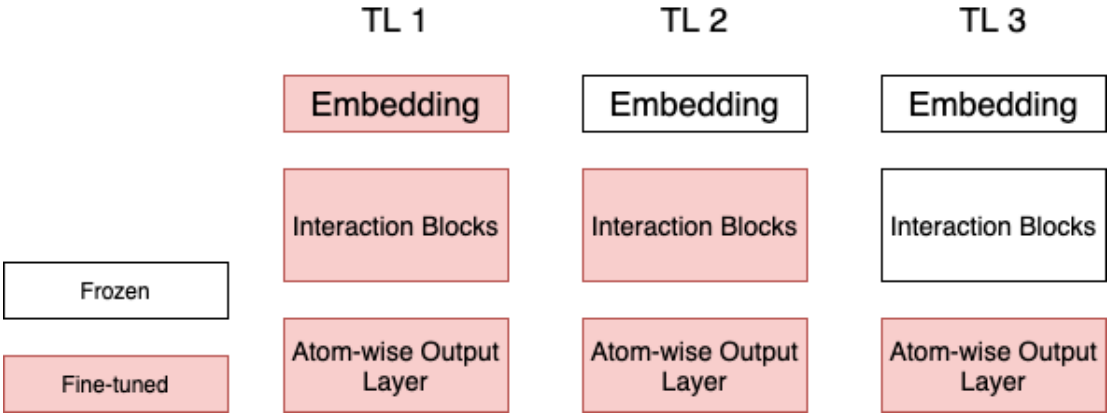


Figure 3.1: Visualization of the three transfer learning schemes used. The embedding layer, the interaction blocks, and the output layer are either frozen or fine-tuned.

TMO data set.

2. Initialize the network with pre-trained weights, fine-tune all except the embedding layer, which stays frozen in training.

3. Initialize the network with pre-trained weights, fine-tune only the output layer during training.

The transfer learning methods are demonstrated on two target transition metal oxide data sets. First, all three of the methods are used on the initial TMO data set. The data was split into a training set of 400 and a validation set of 117. Training data sizes of 400, 200, and 100 are used. For the 200-length data set training, the 400 are split into two sets, and a separate neural network is trained with each. The same is done with the 100-length training set, where four neural networks are trained. The mean absolute error of each of the same-length data set networks is then averaged to get the given results. This is done to ensure consistent results, since the data set is small and all portions may not be entirely representative of each other. Next, the two successful methods are compared to direct training on the extended TMO data set, including two additional transition metal elements and 146 additional data points. On the extended set, training is done similarly except with training set sizes of 500, 250, and 125 and

N	No TL	TL1	TL2	TL3
100	0.939	**0.929**	0.943	1.191
200	0.762	**0.691**	0.703	0.847
400	0.551	**0.477**	0.482	0.686

Table 3.2: Validation set mean absolute error in eV of each of the transfer learning schemes along with the best performing direct training method across the different-sized splits of the original TMO data set. The bold numbers are the best result for the row.

a validation set of 163 samples. The best performing direct training method, using L2 regularization with a coefficient of 10^{-3}, is compared to the transfer learning results for each data set.

3.3.1 Results

This subsection provides the results of the two experiments outlined above. First The original TMO data set is trained with each method, then the best methods are considered again for the slightly larger and more diverse extended data set.

3.3.1.1 TMO Data Set

For the transition metal oxide data set consisting of three transition metals, transfer learning methods 1 and 2 proved to be the most effective, achieving lower mean absolute error on the validation set than the best direct training model for the 400 and 200 size training sets, and similar error for the 100 size training sets. The third transfer learning scheme performed worse than the direct training. The mean absolute errors are reported in Table 3.2. A significant difference was seen between the number of validation predictions within chemical accuracy of the DFT value for the successful transfer learning methods compared to direct training, with over 20% more for each training data set size. These results are visualized in Figure 3.2.

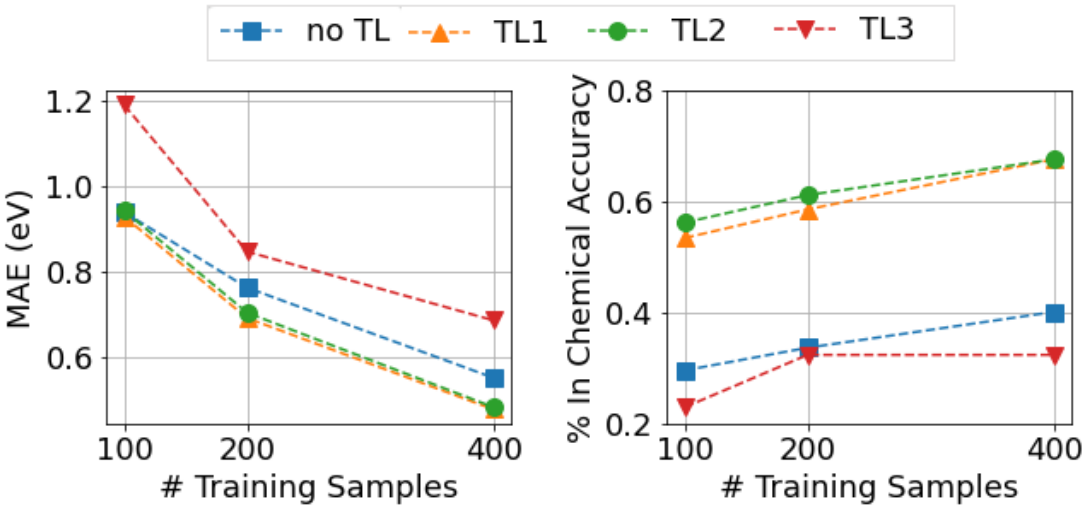


Figure 3.2: Results of the three best transfer learning methods compared to best direct training method without transfer learning with the original TMO data set. Left: Mean absolute error of the validation set evaluation for the models trained on each of the training set sizes. Right: Percent of evaluations within chemical accuracy of the value computed with DFT.

3.3.1.2 Extended Data set

For the extended TMO data set, only TL1 and TL2 transfer learning methods are considered in the comparison with direct training. For models trained on training sets of size 125 and 250, both transfer learning methods achieve lower mean absolute error than direct training, while models trained on the largest training set all performed similarly. Like the smaller TMO data set, the transfer learning methods both get significantly more predictions from the validation set within chemical accuracy of their DFT-calculated value than the directly trained model. These results are reported in Table 3.3 and Figure 3.3.

3.3.2 Error Analysis

In these two experiments, transfer learning methods 1 and 2 perform good as or better than direct training in mean absolute error. However, even in the two cases with similar mean absolute error, a much larger fraction of the transfer learning

N	No TL	TL1	TL2
125	1.355	1.083	**1.067**
250	0.827	0.753	**0.751**
500	**0.730**	0.744	**0.730**

Table 3.3: Validation set mean absolute error in eV of the previously best performing direct training and transfer learning methods applied to the extended data TMO data set.

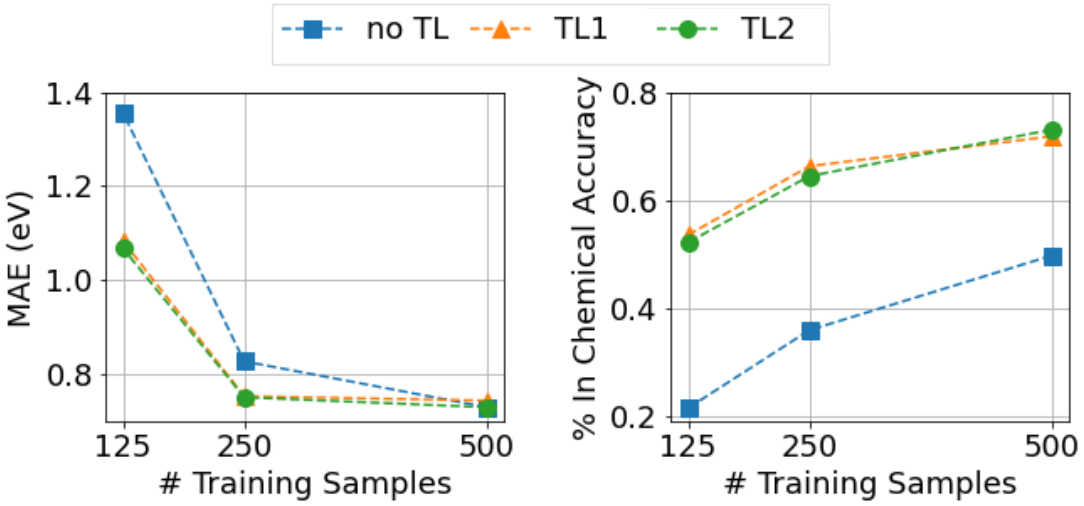


Figure 3.3: Results of the two best transfer learning methods compared to best direct training method without transfer learning with the extended TMO data set. Left: Mean absolute error of the validation set evaluation for the models trained on each of the training set sizes. Right: Percent of evaluations within chemical accuracy of the value computed with DFT.

predictions are within chemical accuracy than the direct model predictions. The reason for this was explored further, and it was found that in addition to the lower error, the transfer learning predictions also shared a higher proportion of the higher errors than the directly trained model, leading to similar mean error. This is shown for the TL1 model in both the 100 length training sets for the original data set and the 500 length training set for the extended data set in Figure 3.5. The cumulative error distributions of the best transfer learning method for the larger and smaller training set size of both the original and extended TMO data is explored in Figure 3.6, along with relative errors

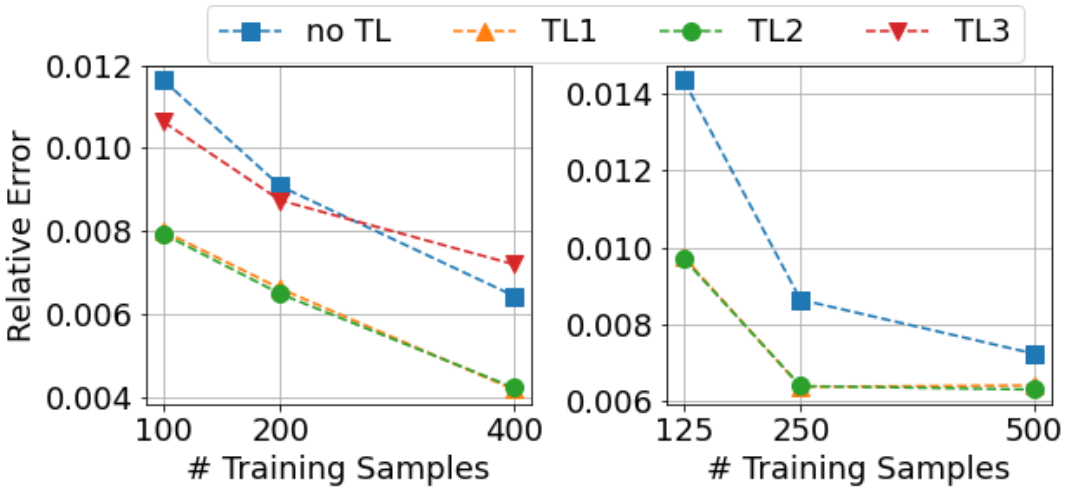


Figure 3.4: Relative error for each method on the original and extended TMO data set. Left: Original data set. Right: Extended data set.

N	No TL	TL
125	7182	214495
250	5008	210736
500	3626	209751

Table 3.4: Total computational time in seconds to train direct and transfer learning models per size of training set. Transfer learning models also take into account the training time of their source model, accounting for the large difference between methods.

of all methods in Figure 3.4.

3.4 Computational Cost Comparison

As seen in Chapter 1, density functional theory calculations can be computationally expensive, with the average calculation from a set of 1,300 transition metal oxides taking over 27 minutes. Using a neural network in place of DFT allows for accelerated predictions after the overhead cost of training and generation of data. With a trained SchNet model, predicting the free energy of a new transition metal oxide is in the order of milliseconds, while running DFT calculations takes similar time as the previous calculations. The overhead cost of training the neural network can be insignificant

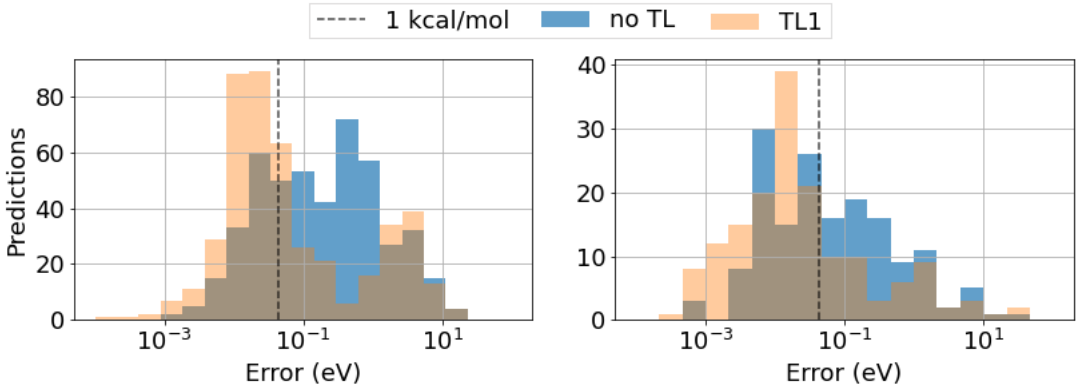


Figure 3.5: Error distribution in cases where TL1 and direct training had similar MAE with a significant gap in predictions within chemical accuracy. In each case, TL1 errors make up a larger portion of both the ends of the distribution, with the larger values making a significant impact to the MAE. Left: Predictions from models trained on 100 length training sets of original TMO data set. Right: Predictions from models trained on 500 length training set of extended TMO data set.

for larger scale screening of materials. Table 3.4 gives the training times for direct and transfer learning models. SchNet was implemented using the PyTorch machine learning framework in python [27]. Training was done with an NVIDIA Titan RTX GPU. While both direct and transfer training times were similar, we must also consider the training of the source model used to initialize the transfer learning models, which leads to the large discrepancy between the two. For the largest training set, the total training time was 2.2 times that of the mean DFT calculation time for direct training, and 128.2 times for transfer learning. If hundreds or thousands of materials are to be screened, neural networks allow a large savings in computational time, as visualized in Figure 3.7.

To give an example of this speedup, DFT was done on a simple, two-atom TiO molecule using the Quantum Espresso DFT code on a 2014 Mac Mini with 1.4 GHz Intel Core i5 processor and 4GB of RAM. This took 87 seconds. Evaluating the same material with the neural network took 4.6 milliseconds on the same computer. While being a big speedup, this does not capture the more significant speedup seen with materials with more atoms. While DFT calculations for larger transition metal

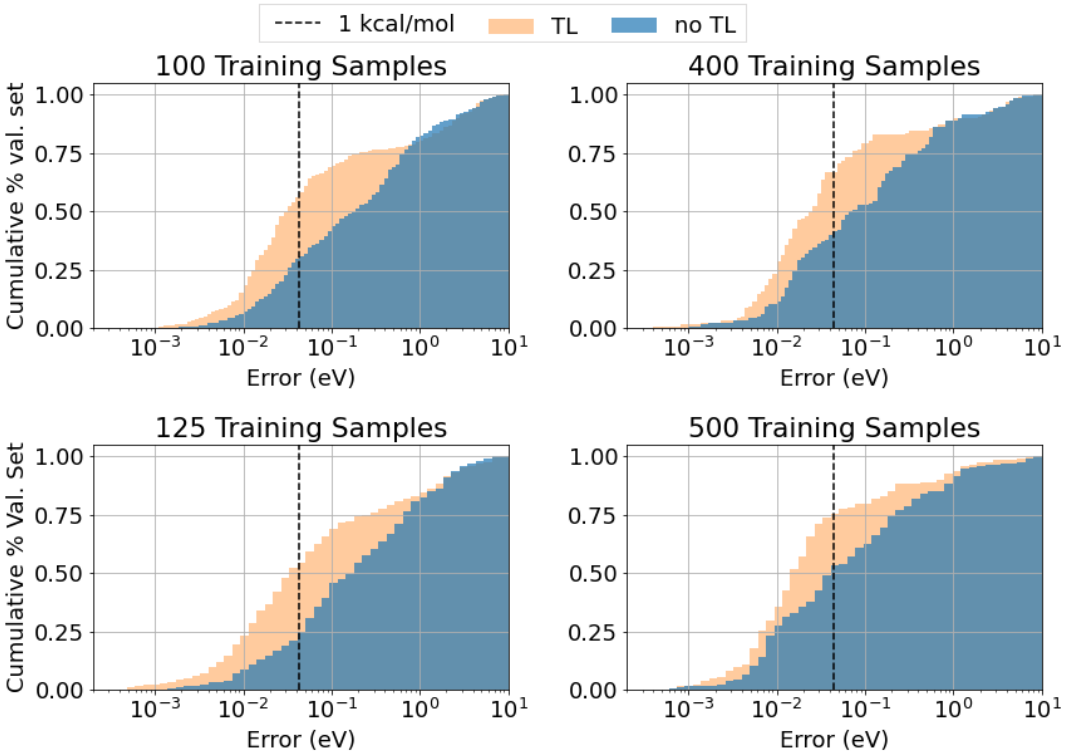


Figure 3.6: Top: Cumulative distribution of validation set error for direct training and TL1 methods trained with the 100 and 400 length training sets of the TMO data. In each case, over 20% more of the predictions from transfer learning are within chemical accuracy (1 kcal/mol). Bottom: Cumulative distribution of validation set error for direct training and TL2 methods trained with the 125 and 500 length training sets of the extended TMO data.

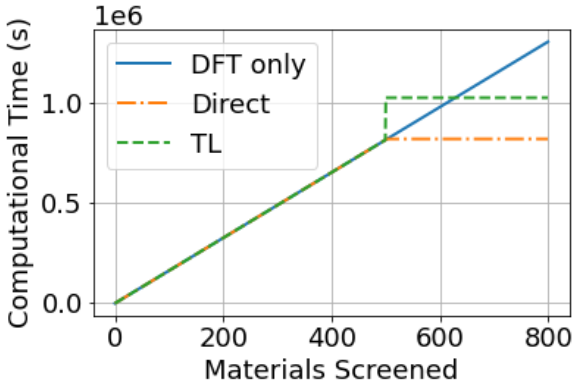


Figure 3.7: Total hypothetical computational time for screening properties with only DFT and neural networks. Using 500 samples for training, using either direct training or transfer learning would save 135 or 78 hours respectively in computational time over DFT in predicting properties of just 300 more materials.

oxides took as long as 20 hours in the benchmark, the longest SchNet evaluation time was 350 milliseconds.

3.5 Summary

In two experiments, it is demonstrated that using transfer learning for training SchNet models with small data sets can improve the mean absolute error of predictions compared to direct training methods. Furthermore, even in cases with marginal differences in mean absolute error, transfer learning models made a higher percentage of predictions within chemical accuracy, but on the other end of the spectrum had outlying predictions of greater error than that of the directly trained models.

Transfer learning methods using a source model trained on a data set with the same elements as the target data set and fine-tuning the weights proved to be the most effective transfer learning approaches. Method 1 did this, while method 2 only froze the weights of the initial embedding layer, fine-tuning the rest. These two methods performed similarly on all tests, while the other approach proved to be less effective than the direct training.

Using neural networks for predicting material properties allows for accelerated screening of materials, as evaluating with a neural network for a single material takes milliseconds while density functional theory can take minutes to hours. Here, it was shown that transfer learning can allow for training more accurate models than direct training with the same amount of data. Though it requires further investigation, this could mean training models of similar accuracy as direct training would require less data for the target data set, reducing the number of new density functional theory calculations needed to be done to create training data. The source data set used for transfer learning consisted of 50,000 materials, but was all available through public data sets, meaning no extra computational cost was used in its creation. The extra computational cost associated with transfer learning was in training the source model, which could be reused in practice since it consisted of a wide range of elements.

To further validate the efficacy of these methods, experiments with a larger target data set would be necessary. Although the goal was to learn accurate representations with little data, 500 training samples is still considered relatively small for neural network optimization, and the effects of this can be seen by the very poor performance on some outliers. A larger data set would reduce the effect these outliers have on the mean error, giving a more reliable metric less dependent on the noise of the data set.

Bibliography

[1] T. A. Arias, M. C. Payne, and J. D. Joannopoulos. Ab initio molecular-dynamics techniques extended to large-length-scale systems. *Phys. Rev. B*, 45(4):1538–1549, January 1992.

[2] P. Blaha, H. Hofstätter, O. Koch, R. Laskowski, and K. Schwarz. Iterative diagonalization in augmented plane wave based methods in electronic structure calculations. *J. Comput. Phys.*, 229(2):453–460, January 2010.

[3] R.N. Bracewell. *The Fourier Transform and its Applications.* McGraw-Hill Kogakusha, Ltd., Tokyo, second edition, 1978.

[4] A. Chandran. *A performance study of Quantum ESPRESSO's diagonalization methods on cutting edge computer technology for high-performance computing.* PhD thesis, 2017.

[5] M. Crouzeix, B. Philippe, and M. Sadkane. The Davidson Method. *SIAM J. Sci. Comput.*, 15(1):62–76, January 1994.

[6] S. Curtarolo, W. Setyawan, S. Wang, J. Xue, K. Yang, R. H. Taylor, L. J. Nelson, G. L. W. Hart, S. Sanvito, ¡. Buongiorno-Nardelli, N. Mingo, and O Levy. AFLOWLIB.ORG: A distributed materials properties repository from high-throughput ab initio calculations. *Computational Materials Science*, 58:227–235, June 2012.

[7] P. A. M. Dirac. *The Principles of Quantum Mechanics.* Clarendon Press, 1981.

[8] R. Eldan and O. Shamir. The power of depth for feedforward neural networks. *CoRR*, abs/1512.03965, 2015.

[9] B. Ghojogh, F. Karray, and M. Crowley. Eigenvalue and Generalized Eigenvalue Problems: Tutorial. March 2019.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] D. Griffiths and D. Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Press, August 2018.

[12] P. Hohenberg and W. Kohn. Inhomogeneous Electron Gas. *Phys. Rev.*, 136(3B):B864–B871, November 1964.

[13] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989.

[14] M. Huh, P. Agrawal, and A. A. Efros. What makes ImageNet good for transfer learning? *arXiv:1608.08614 [cs]*, December 2016. arXiv: 1608.08614.

[15] M. Parrinello J. Behler. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Physical Review Letters*, 98(14):146401, April 2007.

[16] P. Duclos J. M. Combes and R. Seiler. The Born-Oppenheimer Approximation. In G. Velo and A. S. Wightman, editors, *Rigorous Atomic and Molecular Physics*, pages 185–213. Springer US, Boston, MA, 1981.

[17] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY, 2013.

[18] D.P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. arXiv: 1412.6980.

[19] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, January 1993.

[20] R. M. Martin. *Electronic Structure: basic theory and practical methods*. Cambridge, 2004.

[21] A. Menegola, M. Fornaciali, R. Pires, F. V. Bittencourt, S. Avila, and E. Valle. Knowledge transfer for melanoma screening with deep learning. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pages 297–300, Melbourne, Australia, April 2017. IEEE.

[22] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, December 2006.

[23] F. Nogueira, A. Castro, and M. A. L. Marques. A Tutorial on Density Functional Theory. In *A Primer in Density Functional Theory*, volume 620. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[24] A. Jainand S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, and K. a. Persson. The Materials Project: A materials genome approach to accelerating materials innovation. *APL Materials*, 1(1):011002, 2013.

[25] N. Bonini M. Calandra R. Car C. Cavazzoni D. Ceresoli G. L. Chiarotti M. Cococcioni I. Dabo A. D. Corso S. de Gironcoli S. Fabris G. Fratesi R. Gebauer U. Gerstmann C. Gougoussis A. Kokalj M. Lazzeri L. Martin-Samos N. Marzari F. Mauri R. Mazzarello S. Paolini A. Pasquarello L. Paulatto C. Sbraccia S. Scandolo G. Sclauzero A. P. Seitsonen A. Smogunov P. Umari P. Giannozzi, S. Baroni and R.M. Wentzcovitch. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *J. Phys. Condens. Matter*, 21(39):395502, September 2009.

[26] S. J. Pan and Q. Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[28] E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge*, 3(16):35–43, 1969.

[29] J. A. Pople, P. M. W. Gill, and B. G. Johnson. Kohn—Sham density-functional theory within a finite basis set. *Chem. Phys. Lett.*, 199(6):557–560, November 1992.

[30] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

[31] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

[32] S. J. Reddi, S. Kale, and S. Kumar. On the Convergence of Adam and Beyond. *arXiv:1904.09237 [cs, math, stat]*, April 2019. arXiv: 1904.09237.

[33] L. Ruddigkeit, R. van Deursen, L.C. Blum, and J.-L.Reymond. Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17. *Journal of Chemical Information and Modeling*, 52(11):2864–2875, November 2012. Publisher: American Chemical Society.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, December 2015.

[35] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems 30*, pages 991–1001. Curran Associates, Inc., 2017.

[36] J. Smith, B. Nebgen, R. Zubatyuk, N. Lubbers, C. Devereux, K. Barros, S. Tretiak, O. Isayev, and A. Roitberg. Outsmarting Quantum Chemistry Through Transfer Learning. January 2018.

[37] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *Advances in Neural Information Processing Systems 30*, pages 4148–4158. Curran Associates, Inc., 2017.

[38] H. Yamada, C. Liu, S. Wu, Y. Koyama, S. Ju, J. Shiomi, Morikawa, and R. Yoshida. Predicting Materials Properties with Little Data Using Shotgun Transfer Learning. *ACS Central Science*, 5(10):1717–1730, October 2019. Publisher: American Chemical Society.