Title
Accelerating Numerical Simulations with Deep Learning

Author
Kim, Dong Hoon

Publication Date
2020

Peer reviewed|Thesis/dissertation

Accelerating Numerical Simulations with Deep Learning

by

Dong Hoon Kim

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Tarek I. Zohdi, Chair
Professor Per-Olof Persson
Professor Kameshwar Poolla

Summer 2020

Accelerating Numerical Simulations with Deep Learning

Abstract

Accelerating Numerical Simulations with Deep Learning

by

Dong Hoon Kim

Doctor of Philosophy in Mechanical Engineering

University of California, Berkeley

Professor Tarek I. Zohdi, Chair

In many industrial applications, numerical simulations allow us to perform virtual experiments through computers by solving differential equations. However, it often requires us to put large amounts of computational resources, because solving differential equations is computationally expensive and time-consuming in general. This could be more critical when we need to run heavy simulations in real-time applications.

This work introduces a hybrid approach on how to accelerate numerical simulations by applying the fundamental idea of deep learning to the numerical simulations. Deep learning and numerical simulation have proposed two different ways for engineers and scientists to predict and understand the complex behavior of systems. While numerical simulation is a traditional technology that relies on the fundamental laws of nature, deep learning is an emerging technology that is highly data-driven.

In the first half of this dissertation, I will review a basic background on deep learning and nonconvex optimization to help readers easily understand the fundamental concepts. In the second half, I will introduce the two engineering problems on accelerating numerical simulations with both reduced simulation costs and desirable accuracy. The first problem is in rapid process control of multiphase flowing foods. The second problem is to optimize the tool path in the Selective Laser Sintering process.

I wholeheartedly dedicate this dissertation to my beloved parents, Weonjun Kim and Kyungja Lee, who have always loved me unconditionally and have been a constant source of encouragement and support during my entire life. I also dedicate this dissertation to my maternal grandfather, Kwangwoo Lee, who always inspired me to become a creative engineer when I was young.

Contents

List of Figures

List of Tables

Acknowledgments

I have been extremely blessed to have my research advisor by Professor Tarek I. Zohdi. He has not only supported me intellectually and financially, but also encouraged me mentally and emotionally. Every time I made a small progress on research or projects, he always praised and encouraged me by saying 'You are a superman!'. Professor Tarek I. Zohdi is sincerely a real superman for me. I greatly respect his generosity, vision, wisdom, sincerity, and motivation, and these deeply have inspired me during my Ph.D. years. It was a great privilege and honor to work and study under his guidance. Professor Tarek I. Zohdi is truly one of the greatest persons I have ever met in my entire life.

I am extremely grateful to Professor Per-Olof Persson for teaching me numerical methods and answering many naïve mathematical questions in his office hours. I learned a lot of useful knowledge on the numerical methods, which is the core part of my doctorate study, from his classes. I also respect his passion, kindness, smartness, and versatility. He was not only a good teacher but also a great keyboard player. I could not forget the moment when I visited his wonderful concert three years ago with Lun Jiang.

I am deeply indebted to Professor Kameshwar Poolla. He was sincerely a great teacher for me. His lecture on Mathematical Methods in Engineering was one of the greatest and most helpful classes I have ever taken in UC Berkeley. His class was really fun and exciting, and I learned a lot from his class. He was not only good at making students understand, but also a greatly respectful person. Every time I visited his office hour, he kindly answered my questions. I respect his generosity and kindness.

I would also like to extend my deepest gratitude to Professor Grace Gu for encouraging my study on machine learning. She kindly shared good references for studying machine learning two years ago, and this became a necessary part of my doctorate research and dissertation. When I came across her at Etcheverry hall in the evening, she always treated me pleasantly and kindly.

I would like to extend my sincere thanks to Professors George Johnson and Ömer Savaş. They were not only excellent lecturers but also great teachers. They always made themselves available to me and nurtured my intellectual curiosity. Thanks should also go to Professor Grace O'Connell, who always appreciated my diligence when I worked as a reader in her class. I am extremely glad to have learned from or worked with those great professors in the Department of Mechanical Engineering.

Many thanks to Sun Choi, who is an alumnus of the Computational Manufacturing and Materials Research Lab, for mentally supporting me while I was studying at UC Berkeley. When I felt anxious about taking the preliminary exam and the qualifying exam, he encouraged me saying 'Don't worry. I am sure you will pass the first time you take it.' I pay the tribute of praise to his foresight. I also appreciate that he empathized with my graduate life when I felt tired or in a slump.

I would like to thank all the lab members, visiting scholars, and alumni in the CMMRL, including David Fernández-Gutiérrez, Shanna Hays, Erden Yildizdag, Brett Kelly, Mickey Clemon, Nicolas Castrillon, Takashi Maeshima, Simon Schmidt, Roger Isied, Zachary Yun,

Avery rock, Brian Howell, David Alcantara, Kate Edwards, Maxwell Micali, Marc Russell, Zeyad Zaky, Chang Yoon Park, Youngkyu Kim, Payton Goodrich, Lukas Bante, Christoph Schreiber, Timo Schmidt, Christian Zeller, Anna Rehr, Henning Wessels, and Atrin Sarmadi. I greatly appreciate their excellent and insightful feedback and help. Opening the lab door was always fun to me because they made our lab's atmosphere harmonious and pleasant. Many thanks to all for your guidance and friendship.

Especially, many thanks again to Roger Isied and Zachary Yun for working for the COE websites and Shorelight project together. I sincerely feel grateful that I worked with those nice colleagues. I respect Roger Isied's strong responsibility and passion for his work. I was also very pleased to work with Keith Gatto and Walter Campbell for the Shorelight project. They were always cheerful to me. Many thanks again to Nicolas Castrillon, Zachary Yun, Roger Isied, and Avery Rock for providing excellent feedback on my research paper. I must also thank David Fernández-Gutiérrez, Shanna Hays, and Mickey Clemon again for providing me great advice on my research and graduate study.

I can not begin to express my thanks to Zhi-Wei Lin, who was a Master of Engineering student in our lab in 2018-2019. I greatly appreciate his help on simulation work with Star-CCM+ while I was super busy with preparing for my Ph.D. qualifying exam. Without him, I could not have successfully run high fidelity simulations for the ARM project with Siemens. I am also thankful to people at Siemens, Eugene Solowjow, Martin Sehr, and Shashank Tamaskar, whom I worked together for the ARM project in 2019. It was a great pleasure to work with them.

Special thanks to my good friends and confidants Erden Yildizdag, Orhan Ocal, and Milad Shirani. They made my graduate life more fun and exciting. They were not only smart researchers in the lab but also great friends to hang out outside the lab. I also congratulate that Erden Yildizdag became an assistant professor at Istanbul Technical University recently. I pray for good luck on his future research and teaching.

Many thanks to other professors and colleagues in the Department of Mechanical Engineering and the other departments of UC Berkeley. I am deeply grateful to Professor Shaofan Li, in the Department of Civil and Environmental Engineering, for his passionate teaching in his Computational Nano-mechanics class and encouraging me every time. Also, special thanks to Xin Lai, who was a teaching assistant in Professor Shaofan Li's class. He always helped me in the office hour with kindness. Professor Marcel Kristel was always kind and cheerful in his class, and I respect his kindness and generosity. Professors Fai Ma and Mark Mueller always greeted me kindly when I came across them in Etcheverry Hall.

I had the great pleasure of working with Brian Muldoon, Gabriel López, and Travis King as teaching assistants in Professor Grace O'Connell's class. Alejandro Morales Martinez, Çağlar Tamur, Abdulrahman Jbaily, Zacharias Vangelatos, Sen Li, Michael Kelly, Michael Estrada, Joshua Su, Nathaniel Goldberg, Max Chiyu Jiang, Andrew Sanville, Daniel Grieb, Zachary Theroff, Eric Ibarra, Lun Jiang, Jessica Leu, April Novak, Kate Schweidel, Alvin Chi-Chung Li, Ean Hall, Brett Hendrickson, Haris Moazam Sheikh, Sai Mandava, Tina Piracci, Haley Wohlever, Yanhe Huang, Qiuchen Guo, Yara Mubarak, Magda Ntetsika, Guy Bergel, Claire Arthurs, and Nicole Farias always treated me pleasantly and kindly. Many

thanks to all of them. I also appreciate all the staff members of the Department of Mechanical Engineering, including Donna Craig, Isabel Blanco, Reggie Madison, Dan Essley, Rene Viray, Ana Preza-Gregg, and Yawo Akpawu, for always being available and helping me.

I wish to show my sincere gratitude to Professor Phillip Colella in Berkeley Lab. I learned a lot of fancy C++ programming skills from his Software Engineering for Scientific Computing class. I'd like to greatly acknowledge the effort of Fanwei Kong for actively working with me in a team for the final group project of Professor Phillip Colella's class. I also wish to thank Professors Aydın Buluç and Kathy Yelick. Applications of Parallel Computers class from them was really helpful to me. I'd like to recognize the help that I received from Jinkyu Kim and Cecilia Zhang for the project of the parallel computing class. I also gratefully acknowledge the assistance of Philippe Laban and Deepak Pathak in the parallel computing class. Many thanks to all of you.

Also, I very much appreciate some Korean undergraduate students, including Jenny Jungmin Kim, Isabel Yura Hwang, Taeyoung Kim, Jihye Park, Kyunggeun Kim, Howard Hoseok Yoon, Cloud Jaewook Lee, Jane Jang, Yoojeong Do, Amy Jung, Steve Kim, Jiho Kwak, Nayeon Kang, and April Lee, for making my Berkeley life more fun and exciting.

I would like to express my gratitude to some Korean graduate students. Special thanks to Soochan Chung and Yongkeun Choi, who are my trusted confidants, for always supporting me mentally and personally. I am very glad to be in a friendship with such nice friends. Soo Hyun Shin and Euihyun Choi were the nicest roommates ever in my life. Many thanks for being with me and supporting me intellectually and personally during these years.

I am also sincerely grateful to the other Korean graduate students and postdoctoral researchers including Minyoung Kim, Wonjun Jo, Sohee Jung, Yoonsoo Rho, Taesung Park, Hyungtaek Kim, Sunmoon Yoo, Hyungjin Kim, Lynn Yeom, Suhong Moon, Jungpyo Hong, Soomin Woo, Kyungtae Lee, Jaecheol Lee, Kiwoo Shin, Donggun Lee, Joonyoung Kim, Jinkyu Kim, Sunah Moon, Yeojun Kim, Heesoo Kwon, Saemmool Lee, Joonwoo Kim, Suhong Moon, Junseok Lee, Misa Jieun Kwon, Jinkyu Lim, Minsoo Kang, Jichan Chung, Catherine Park, Saehong Park, Sangjae Bae, Hyundong Ha, Jeongseok Son, Changmin Lee, Dongwoo Shin, Ji Min Kim, Yoonjae Park, Sangwon Kim, Wonkee Cho, Edward Kim, Dongguk Shin, Taejoo Ahn, Sangjoon Lee, Jangho Choi, Hotae Lee, Jungpyo Lee, Sareum Kim, Kunmo Kim, Minok Park, and Euiyoung Kim for supporting me personally. Many thanks again to Taesung Park, Suhong Moon, and Junseok Lee for having intellectual discussions in deep learning with me and nurturing my intellectual curiosity.

Last but not least, my sincere gratitude goes to Samsung Scholarship, who supported my graduate study financially. Without their support, I could not have found success. Thanks also to all the other great people whom I might have forgotten to write here for being with me and supporting me personally during my Ph.D. years. I could not have completed this journey without you. Thank you.

Chapter 1

Fundamentals of Deep Learning

1.1 Introduction

Artificial Intelligence (AI) is a technology that realizes human learning ability, reasoning ability, perception ability, and the ability to understand natural languages through computer programs. Since around the 1950s when AI emerged, the AI has consistently demonstrated problem-solving skills and has had a significant effect on the development of science and technology.

In 2016, AlphaGo, an AI developed by Google DeepMind, surprised all over the world by defeating the human champion of the world, Sedol Lee, in the board game Go. Go is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent. There are 2.089×10^{170} cases to locate the go stone on the board while playing [1], and it is impossible to calculate the possibilities to win by using a brute force approach. AlphaGo uses a novel combination of supervised learning from games between human experts and reinforcement learning from games of self-play [2].

Amazon Echo (Figure 1.1), a smart speaker from Amazon, is capable of voice interaction with humans, recommending music that users would like to listen to, and providing real-time information (e.g. weather, traffic, news, etc.). It can also control several smart devices using itself as a home automation system.

Machine Learning (ML), a major branch of AI, is a field of study that gives computers the ability to learn without being explicitly programmed [4]. In explicit programming, developers encapsulate the implementation details associated with a design concept and explicitly express their idea through their programming languages. However, there is a critical limitation of explicit programming when the object of programming has many rules which are difficult to be explicitly programmed (e.g. spam filtering, face recognition, automatic driving, etc.). Beyond the limitations of explicit programming, the ML allows a computer to learn through experience and improve its performance automatically. The ML builds a mathematical model based on sample data, known as *training data*, to learn the patterns of the data and make predictions or decisions without being explicitly programmed to do

Figure 1.1: Echo Dot (3rd Gen) - Smart speaker with Alexa [3]

so. The ML algorithms are used in many applications, such as spam filtering and computer vision.

Deep Learning (DL) is a subset of the ML. The ML *uses* algorithms to parse data, learn from that data, and make informed decisions based on what it has learned. On the other hand, the DL *structures* algorithms in layers itself to learn and make intelligent predictions on its own. The difference between the DL and the ML is illustrated in Figure 1.2. While machine learning still requires *guidance from humans* on feature selection to predict the output, deep learning could extract the feature and learn through its method of computing *by itself*. The overall concepts of the AI, ML, and DL are shown in Figure 1.3. From the next section, we will review the fundamental concepts needed to implement deep learning in our programming language.

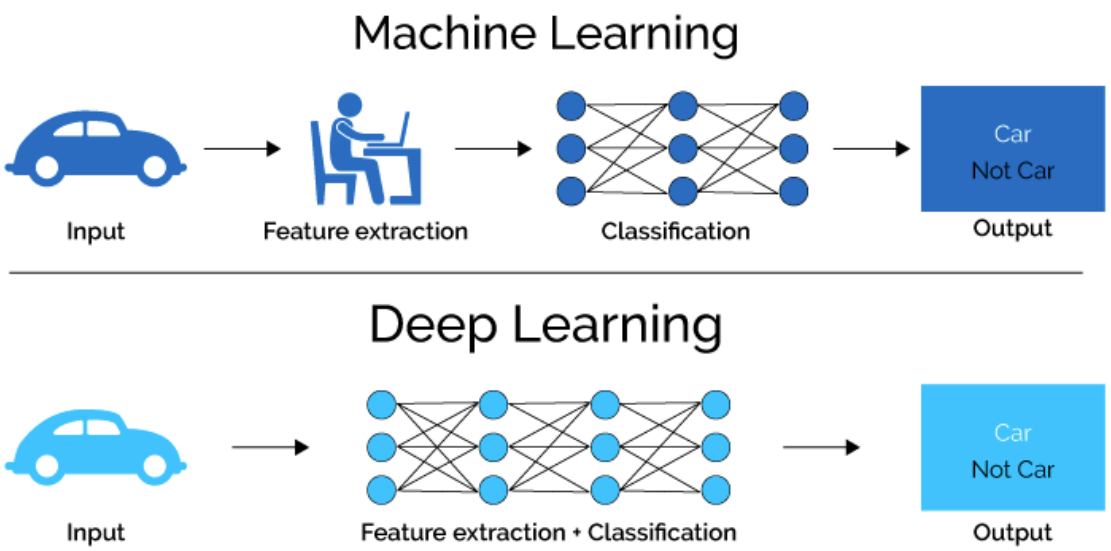


Figure 1.2: Machine Learning vs. Deep Learning [5]

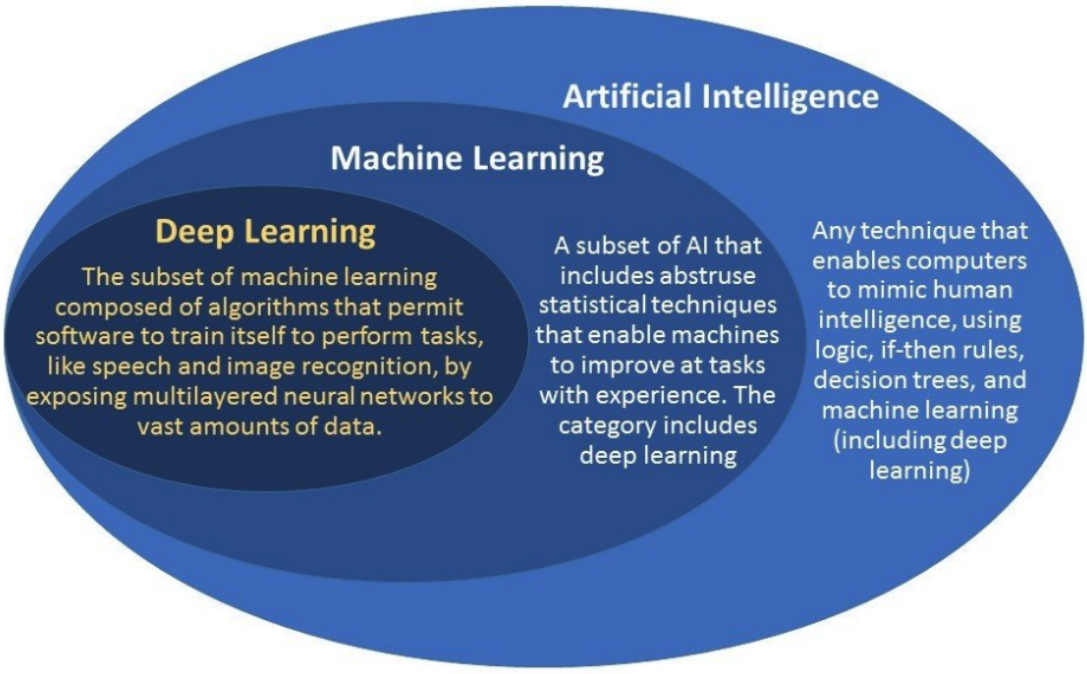


Figure 1.3: Artificial Intelligence, Machine Learning, and Deep Learning [6]

1.2 Machine Learning

Machine Learning systems learn how to combine input to produce useful predictions on never-before-seen data. Machine learning has demonstrated its powerful potentials in various fields such as image recognition, autonomous driving, spam detection, speech recognition, medical prediction, and so on. There are two kinds of machine learning, *Supervised Learning* and *Unsupervised Learning*:

Supervised Learning

1. Learning the *mapping function* from the input data (features) to the output data (labels).
2. The goal is to approximate the mapping function so well that it can predict the output variables for the new input data.
3. Predictive models for classification and regression.

Unsupervised Learning

1. We only have input data and no corresponding output data (label).
2. The goal is to model the underlying structure or distribution in the data in order to learn more about the data.
3. Pattern/structure recognition for clustering, association, dimensionality reduction.

In the upcoming contents, we will only focus on *supervised learning* which is much more common and widely used. Also, I will use the words *machine learning* and *deep learning* only for *supervised learning*. A simple example of supervised learning is shown in Figure 1.4.

There are three core concepts for machine learning: *Features*, *labels*, and a *model*. Features represent input variables, and labels represent the thing we want to make predictions on. A model is a mapping function that defines the relationship between features and labels. The objective of machine learning is to find **the best model which has strong predictive power** on never-before-seen data.

In order to train the model to have good predictive power, we should divide the total data we have into two subsets, which are a *training set* and a *test set*:

- A training set is a subset to **train the model**
- A test set is a subset to **test the performance of the trained model**

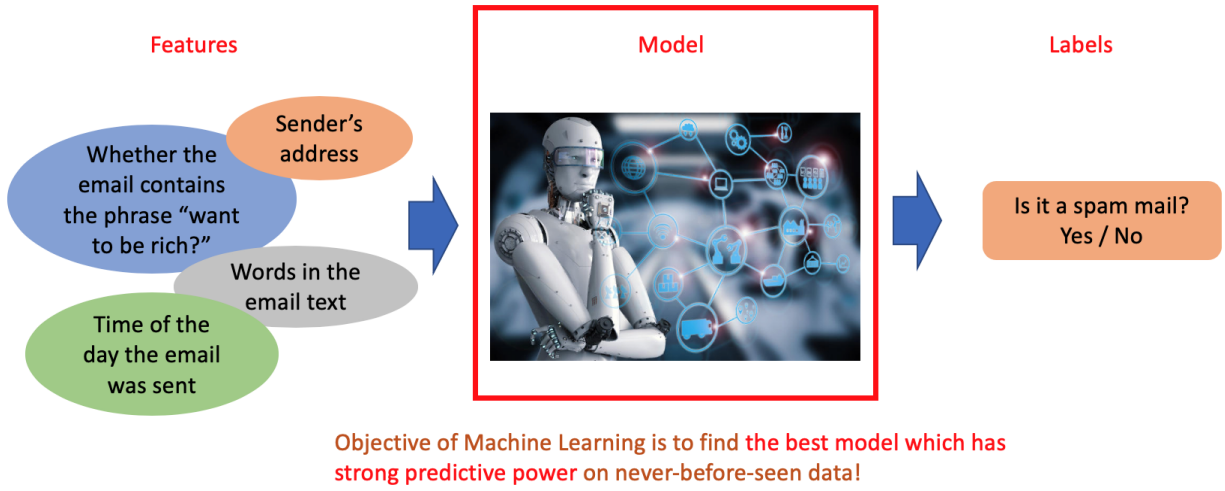


Figure 1.4: Supervised learning

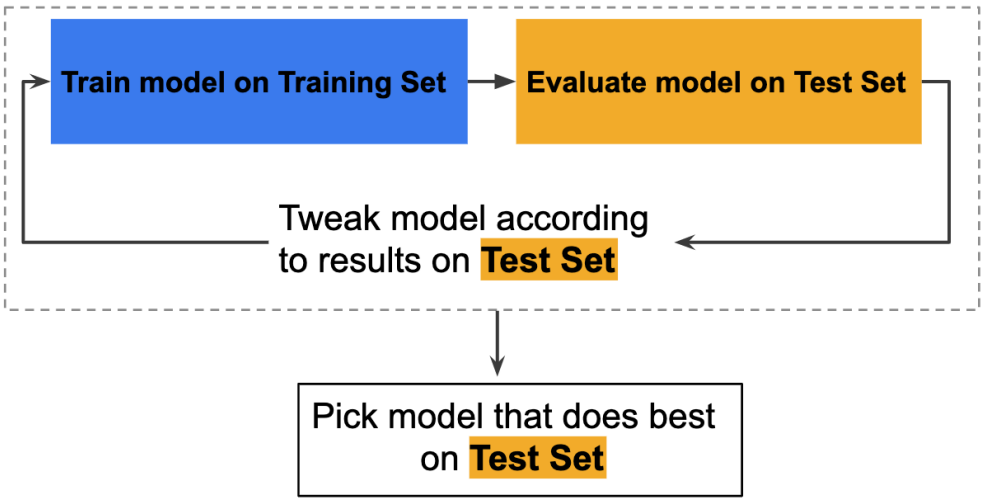


Figure 1.5: A workflow with a training set and a test set [7]

Note that the test set is completely isolated from the training set and the training process. A workflow with the training set and the test set is shown in Figure 1.5. In Figure 1.5, 'Tweak model' means adjusting anything about the model (e.g. changing the learning rate, adding or removing features, designing a completely new model from scratch). At the end of this workflow, we pick the model that does best on the test set. However, dividing the total data set into three subsets (a *training set*, a *validation set*, and a *test set*) could be a better workflow in some cases, as shown in Figure 1.6, because it creates fewer exposures to the

test set while training. Also, we may want to check the performance of the model with the validation set **while training**, so that we could take action as soon as possible if training is not going well [7].

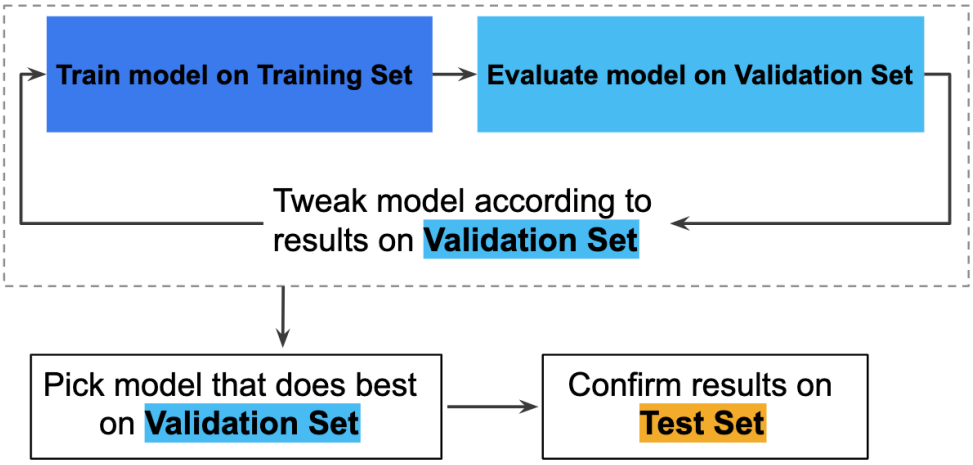


Figure 1.6: A workflow with a training set, a validation set, and a test set [7]

1.3 Regression

Linear Regression

Regression is an approach to find the relationship between input variables (features) and continuous output variables (labels). In machine learning, the objective of the regression is to develop a continuous mapping function that has strong predictive power on never-before-seen data.

Linear regression is the simplest regression method in machine learning. Assuming we are going to use J kinds of input features, the linear model could be expressed as shown in Figure 1.7.

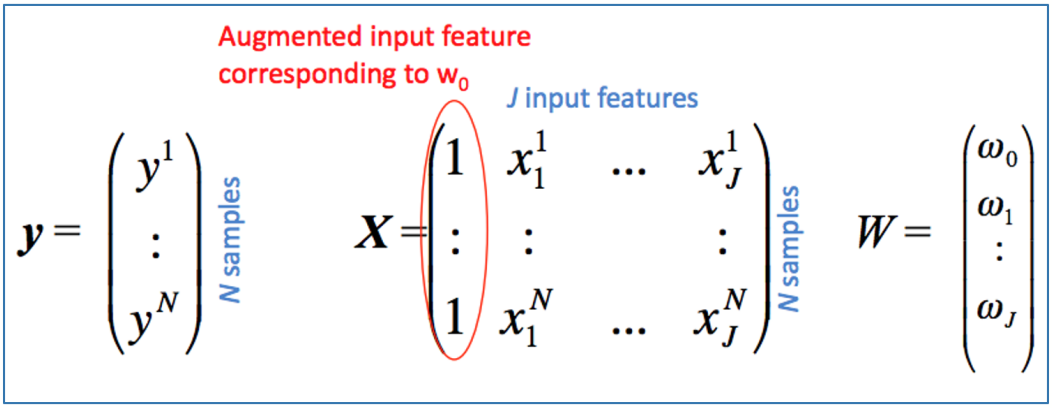


Figure 1.7: Linear regression

In this case, N represents the number of data samples (either for the training, validation, or test set). $\mathbf{y}$ represents labels that we finally want to make predictions on for the test data. $\mathbf{W}$ is a weight for which we eventually want to get. Also, w_0 (the first component of $\mathbf{W}$) is an additive bias.

From Figure 1.7, one can write the *hypothesis function* for the linear model as follows:

$$\mathbf{H}(\mathbf{X}) = \mathbf{X}\mathbf{W} \tag{1.1}$$

In this linear model, we want to minimize the loss function $L(\mathbf{W})$:

$$L(\mathbf{W}) = \|\mathbf{H}(\mathbf{X}) - \mathbf{y}\|_2^2 = \|\mathbf{X}\mathbf{W} - \mathbf{y}\|_2^2 \tag{1.2}$$

In order to minimize the $L(\mathbf{W})$, we use a gradient $\nabla L(\mathbf{W})$ for gradient descent:

$$\nabla L(\mathbf{W}) = 2\mathbf{X}^{\mathbf{T}}(\mathbf{X}\mathbf{W} - \mathbf{y}) \tag{1.3}$$

This represents the gradient of the loss function. Based on this, we correct the weight vector $\mathbf{W}$, to search for optimal weights which minimizes the loss function:

$$\mathbf{W} \leftarrow \mathbf{W} - \alpha\nabla L(\mathbf{W}) \tag{1.4}$$

The method described in Equation 1.4 is called *gradient descent*. The gradient descent is visually illustrated in Figure 1.8. If the gradient (slope) is big, then we could take a big step in a single training step because we are still far from the local minimum of the loss function. However, if the gradient (slope) is small, we need to take a small step carefully to get to the optimal point. We could compare this process with a hiker hiking downhill on a mountain, as shown in Figure 1.9. α in Equation 1.4 represents the training rate (also called the learning rate), which means how much we would multiply to the gradient in every single training step of the training process. We update $\mathbf{W}$ over and over based on Equation 1.4 until the training process ends.

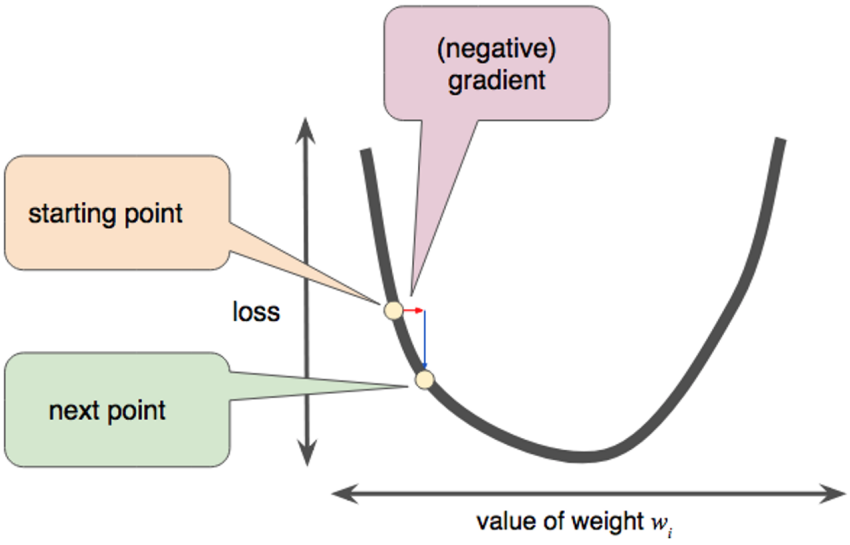


Figure 1.8: A schematic of gradient descent [7]

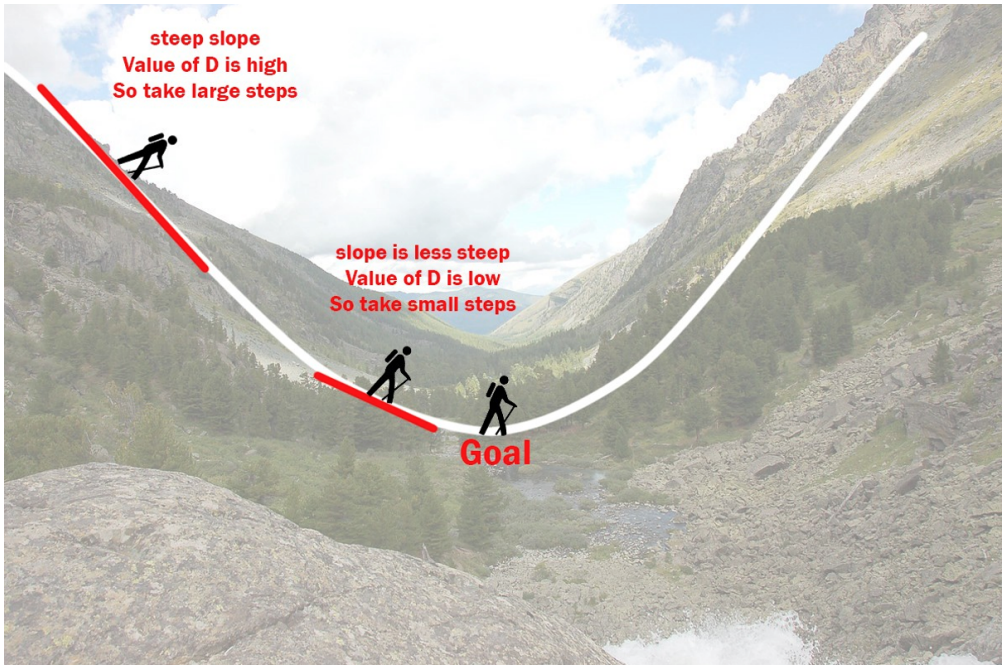


Figure 1.9: A person walking down on a mountain [8]

Nonlinear Regression with Neural Networks

Even though the linear model is simple and computationally cheap enough, many things we want to predict in the world are highly nonlinear. Therefore, the linear model often fails to make good predictions in many cases. A *neural network*, which mimics the structure of biological neurons of the human brain, is used to overcome this limitation (Figure 1.11). Using a neural network, we could perform machine learning with *nonlinear regression*, which could have stronger predictive power in many nonlinear problems. A simple example of the neural network is shown in Figure 1.10.

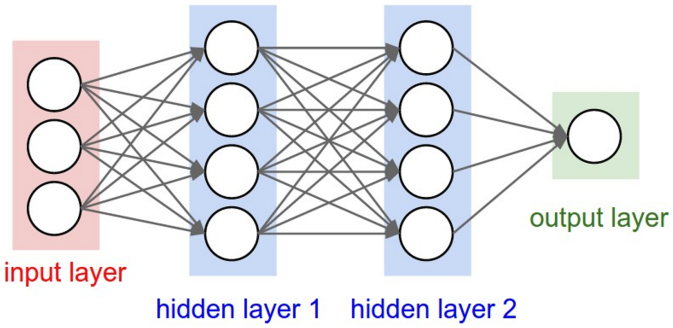


Figure 1.10: A 3-layer neural network with three input features [9]

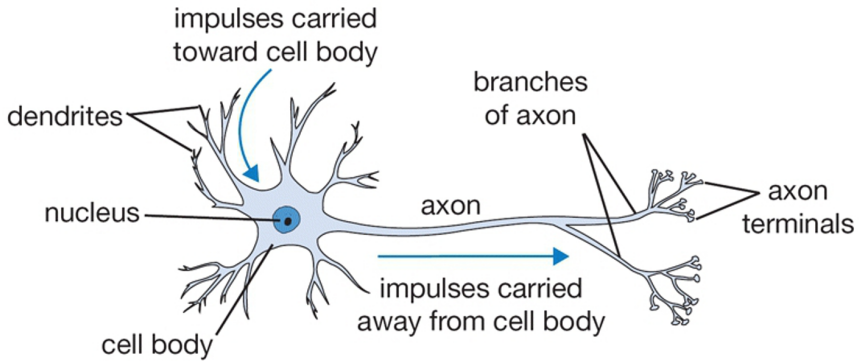
When N by M data (N: the number of data set, M: the number of features) gets into the input layer, M features (N by 1 vectors) enter the corresponding M nodes in the input layer. Passing through the hidden layers, they are **multiplied by weight values** in **'axons'** between the layers,[1] and they are **activated** in the **nodes** by activation functions. *Activation functions* are nonlinear functions that allow the machine learning model to have nonlinear properties. There are a variety of activation functions, as shown in Figure 1.12. A *rectified linear unit (ReLU)* is widely used for activation functions because it could overcome the gradient vanishing problem in the deep neural network, which sigmoid function and hyperbolic tangent function are suffering from.

After passing through all the hidden layers, we need to train the model with gradient descent as we did in linear regression. $\mathbf{y_{output}}$ represents the hypothesis of the model on the input data $\mathbf{X}$, and $\mathbf{y_{target}}$ represents the true label values for the given data set.[2] In Figure 1.13, the objective is to minimize the loss:

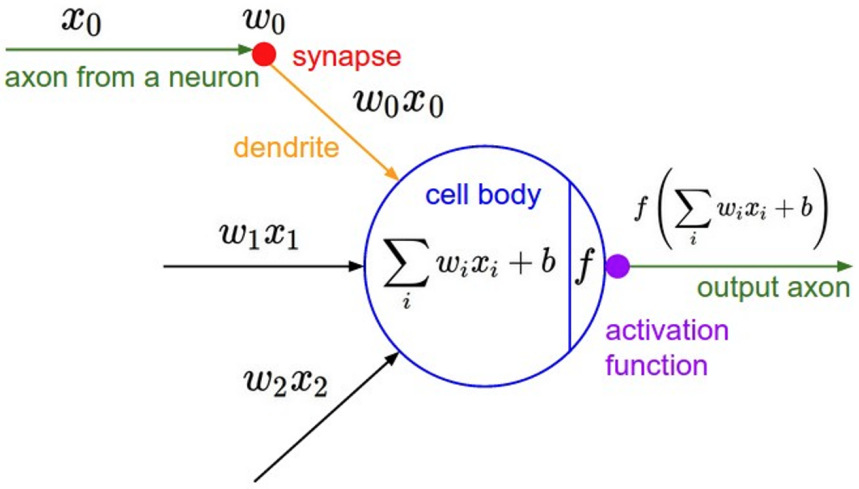
$$E = \frac{1}{N} \|\mathbf{y_{output}} - \mathbf{y_{target}}\|_2^2 \tag{1.5}$$

[1] Additionally, one can have biases in the nodes which are added after the linear multiplication of weights. In this case, biases would be 'learned' along with weight values during the training process.

[2] $\mathbf{y_{output}}$ is equivalent to $\mathbf{H}(\mathbf{X})$, and $\mathbf{y_{target}}$ is equivalent to $\mathbf{y}$ in the linear regression above.



(a) A biological neuron



(b) An artificial neuron

Figure 1.11: A biological neuron vs. An artificial neuron [9]

where N represents the number of data samples. Taking derivative of this yields:

$$\frac{\partial E}{\partial \mathbf{y_{output}}} = \frac{2}{N}(\mathbf{y_{output}} - \mathbf{y_{target}}) \tag{1.6}$$

Having $\mathbf{x}$ as a vector entering to some node and $\mathbf{y}$ as a vector coming out from the node, we could write:

$$\frac{\partial E}{\partial \mathbf{x}} = \frac{d\mathbf{y}}{d\mathbf{x}} \cdot \frac{\partial E}{\partial \mathbf{y}} = \frac{df(\mathbf{x})}{d\mathbf{x}} \cdot \frac{\partial E}{\partial \mathbf{y}} \tag{1.7}$$

where $f(x)$ represents the activation function (ReLU, sigmoid, etc.).

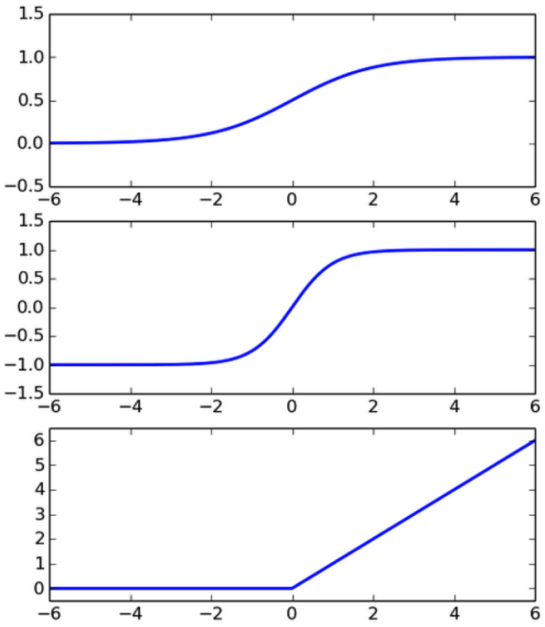


Figure 1.12: Activation functions in nodes

Also, having $\mathbf{y_i}$ as an output vector from i^{th} node of the previous layer and $\mathbf{x_j}$ as an input vector to j^{th} node in the next layer, we could apply the chain rule to write:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial \mathbf{x_j}}{\partial w_{ij}} \cdot \frac{\partial E}{\partial \mathbf{x_j}} = \mathbf{y_i} \cdot \frac{\partial E}{\partial \mathbf{x_j}} \tag{1.8}$$

where w_{ij} represents the weight which is multiplied in an 'axon' connecting i^{th} node of the previous layer and j^{th} node of the next layer.

Based on $\frac{\partial E}{\partial w_{ij}}$ shown in Equation 1.8, we could update the weight values for all the 'axons' by gradient descent:

$$w_{ij} \leftarrow w_{ij} - \alpha \frac{dE}{dw_{ij}} \tag{1.9}$$

where α represents the learning rate.

Also, we could get the derivative of output error with respect to $\mathbf{y_i}$ (output vector of the node in the previous layer):

$$\frac{\partial E}{\partial \mathbf{y_i}} = \sum_{j \in out(i)} \frac{\partial \mathbf{x_j}}{\partial \mathbf{y_i}} \cdot \frac{\partial E}{\partial \mathbf{x_j}} = \sum_{j \in out(i)} w_{ij} \frac{\partial E}{\partial \mathbf{x_j}} \tag{1.10}$$

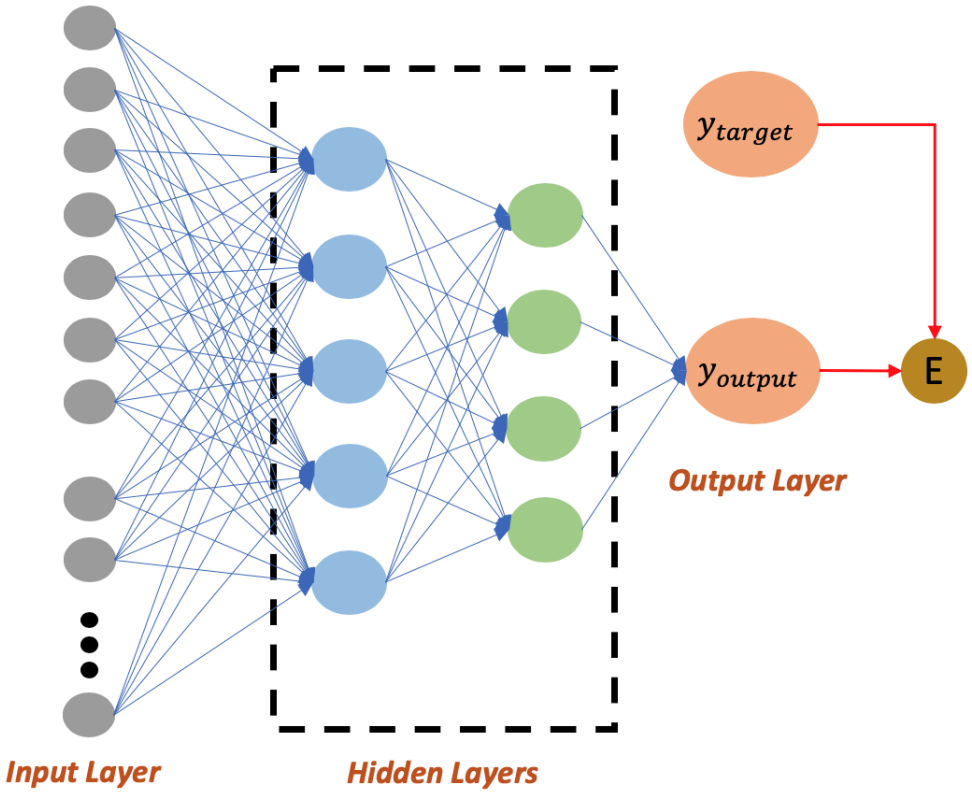


Figure 1.13: A fully connected neural network

The overall process explained above is called **backpropagation**. We start from the error of the output layer, and **backpropagate** the error information to update weight values in 'axons' between all the layers.

1.4 Mini-batches

When training the model, we also have to consider the batch size. *Batch size* represents the number of training examples utilized in **one iteration**. This could be the same as the size of the training data or could be less than training data.

In some applications, the size of the training data set is large enough and it makes the training process with all those data **at once** in each iteration more difficult and time-consuming. Therefore, we could split the training data into *mini-batches* to train the model efficiently, taking advantage of the parallel computing functionality of the graphics processing unit. If the batch size is 1 in gradient descent, it is called *stochastic gradient descent (SGD)*. The word *stochastic* represents selecting a random sample from the total training data. If we perform the iteration over and over, the SGD could effectively get to one of the minima.

However, the SGD still has a problem that it is too noisy because it has much less batch size than full batch iteration. *Mini-batch gradient descent* is a compromise between the full batch iteration and the SGD. Mini-batches typically consist of 10 to 1000 randomly selected examples. Mini-batch gradient descent reduces the noise of the SGD and is more efficient than iteration with the entire batch of the training data. The comparison of gradient descent, mini-batch gradient descent, and stochastic gradient descent is illustrated in Figure 1.14. The **blue** line represents **gradient descent**, the **purple** line represents **stochastic gradient descent (SGD)**, and the **green** line represents **mini-batch gradient descent**. The red circle in the center is a local minimum of the loss function.

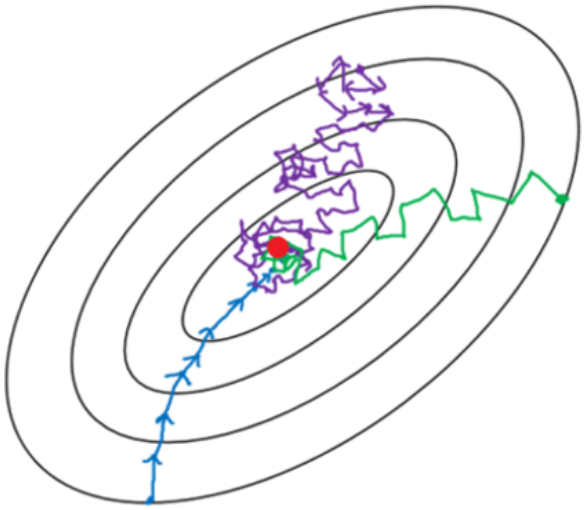


Figure 1.14: Gradient descent, Stochastic gradient descent, and Mini-batch gradient descent

1.5 Regularization

Overfitting

Figure 1.15 illustrates the cases of *underfitting* and *overfitting*. While *underfitting* could be resolved easily by increasing more features to expand the hypothesis space, *overfitting* is a critical problem in machine learning. As mentioned above, we split the total data into the training set and the test set (and the validation set if applicable). We train the model with the training set and make predictions on the test set to check the performance of the trained model. *Overfitting* means the case when the model was over-trained from the training set, so it has weak predictive power on never-before-seen data (the test set). Even though training

error decreases continuously while the training process, test error starts increasing at some point and the model starts losing generalization performance, as shown in Figure 1.16.

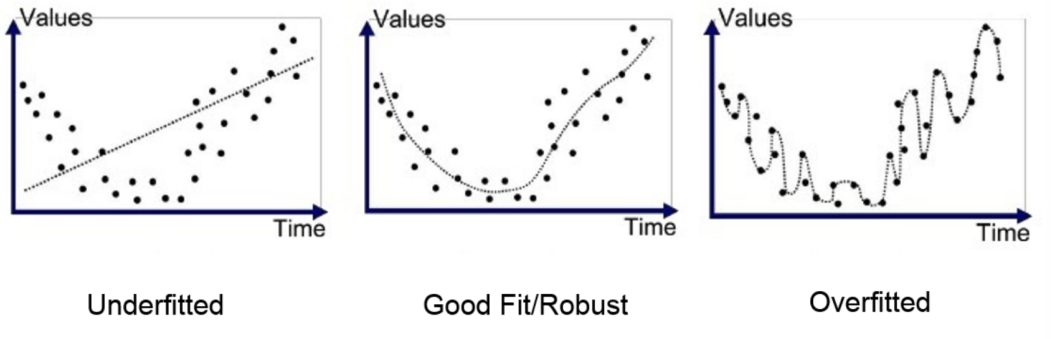


Figure 1.15: Underfitting, Good fitting, and Overfitting [10]

Here are some ways to prevent overfitting:

1. L_1/L_2 regularization
2. Dropout
3. Batch normalization
4. Early stopping (Figure 1.16)[3]
5. Adding more data
6. Reducing architecture complexity to generalize well
7. Adding noise to the input or output [12]
8. Cross-validation [13, 14]

One way to prevent overfitting in a quantified way is to penalize complex models, which is a principle called *regularization* [16, 17]. In other words, instead of simply aiming to minimize loss:

$$\text{minimize}(Loss(Data|Model)) \tag{1.11}$$

We could try to minimize loss and complexity at once:

$$\text{minimize}(Loss(Data|Model) + \lambda \text{complexity}(Model)) \tag{1.12}$$

where λ is a regularization rate. Some ways to **penalize the complexity of the model** are in the following subsections.

[3]For the readers who are interested in theoretical analysis on early stopping, refer to [11].

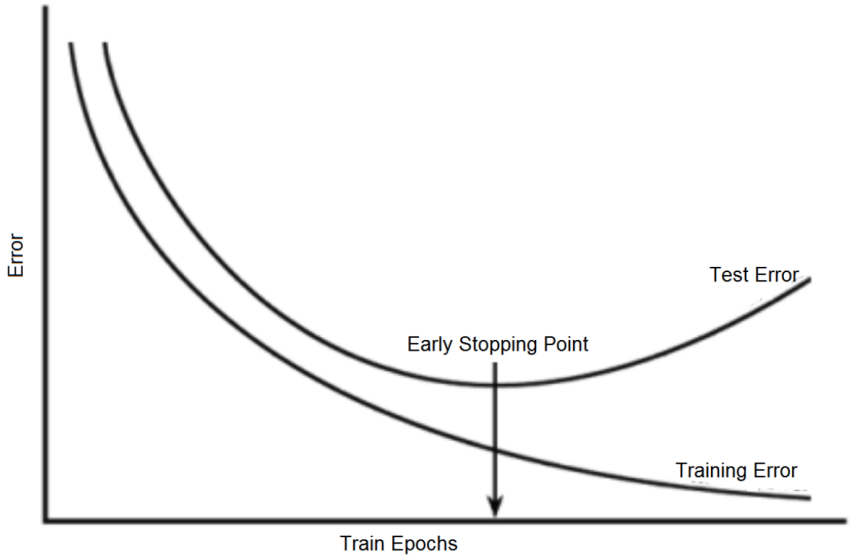


Figure 1.16: Early stopping [15]

L_1 Regularization (Lasso Regression)

L_1 regularization adds **absolute value of magnitude** of coefficient as penalty term to the loss function.

$$L(\mathbf{W}) = \underbrace{\|\mathbf{y} - \mathbf{H}(\mathbf{X})\|_2^2}_{\text{Original loss function}} + \underbrace{\lambda \|\mathbf{W}\|_1}_{\text{Regularization term}} \tag{1.13}$$

where $\mathbf{H}(\mathbf{X})$ represents the hypothesis of the model on the input data $\mathbf{X}$, and $\mathbf{y}$ represents the true label values for the given data set.

Performing L_1 regularization has the following effect on a model:

- It encourages many of the uninformative coefficients of weights in a model to be exactly 0. In other words, it removes the unnecessary features from the model.
- Zeroing out features will save computational cost and may reduce noise in the model.

L_2 Regularization (Ridge Regression)

L_2 regularization adds **squared magnitude** of coefficient as penalty term to the loss function.

$$L(\mathbf{W}) = \underbrace{\|\mathbf{y} - \mathbf{H}(\mathbf{X})\|_2^2}_{\text{Original loss function}} + \underbrace{\lambda \|\mathbf{W}\|_2^2}_{\text{Regularization term}} \tag{1.14}$$

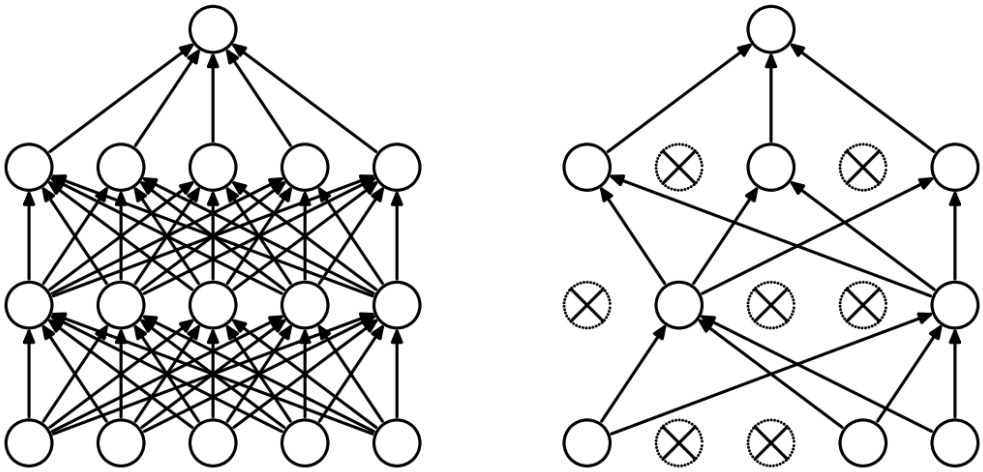
where $\mathbf{H}(\mathbf{X})$ represents the hypothesis of the model on the input data $\mathbf{X}$, and $\mathbf{y}$ represents the true label values for the given data set.

Performing L_2 regularization has the following effect on a model:

- It encourages weight values toward 0 (but not exactly 0).
- It encourages the mean of the weights toward 0, with a normal (bell-shaped or Gaussian) distribution.

Dropout

On top of L_1/L_2 regularization, we could randomly drop the nodes in the neural network as another way of the regularization. *Dropout* is a technique for regularizing neural networks, developed by Hinton et al. [18]. The core idea behind dropout is to randomly set some of the weights in a neural network to 0 during the training process. We set a *keep probability* for dropout, which represents the ratio of the nodes that a weight value is left undisturbed to all the nodes. Figure 1.17 illustrates how dropout works.



(a) Standard Neural Net (b) After applying dropout.

Figure 1.17: *"Dropout: a simple way to prevent neural networks from overfitting"* - Srivastava et al. (2014) [19]

Batch Normalization

Batch normalization has become a necessary technique for neural network training since Ioffe and Szegedy [20] suggested in 2015. Batch normalization is basically one of the ideas

to prevent gradient exploding or gradient vanishing while training the deep neural networks. People tried to resolve these gradient problems by changing the activation function to ReLU, carefully initializing the weights, and using a small learning rate.

However, Ioffe and Szegedy devised a fundamental way to prevent the gradient problems and eventually accelerate the learning speed by **stabilizing** the training process itself, rather than those indirect methods. The overall algorithm of the batch normalization is shown in Figures 1.18 and 1.19.

When training the neural network, data is usually brought as mini-batch units for training. After normalizing the batch by the average and standard deviation **for each feature**, it is scaled by the scale factor and shift factor in the batch normalization layer. The scaled batch then enters a hidden layer that has the activation function. The scale factors and shift factors are trained during backpropagation as weight values in 'axons'. A schematic of the neural network with batch normalization is shown in Figure 1.20.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Figure 1.18: *Batch Normalizing Transform* - Ioffe & Szegedy (2015) [20]

The remarkable advantages of batch normalization are as follows:

1. In traditional deep neural networks, the gradient exploding or vanishing could occur when the learning rate is high, due to the scale of the parameters. When using batch normalization, it is not affected by the scale of the parameters when propagating.

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Figure 1.19: *Training a Batch-Normalized Network* - Ioffe & Szegedy (2015) [20]

Therefore, the learning rate can be greatly increased, and it enables fast learning processes.

2. **Batch normalization itself has a regularization effect.** This allows the model to exclude the weight regularization term (either for L_1 or L_2) and dropout. It also overcomes the disadvantage of dropout by accelerating the learning process (because

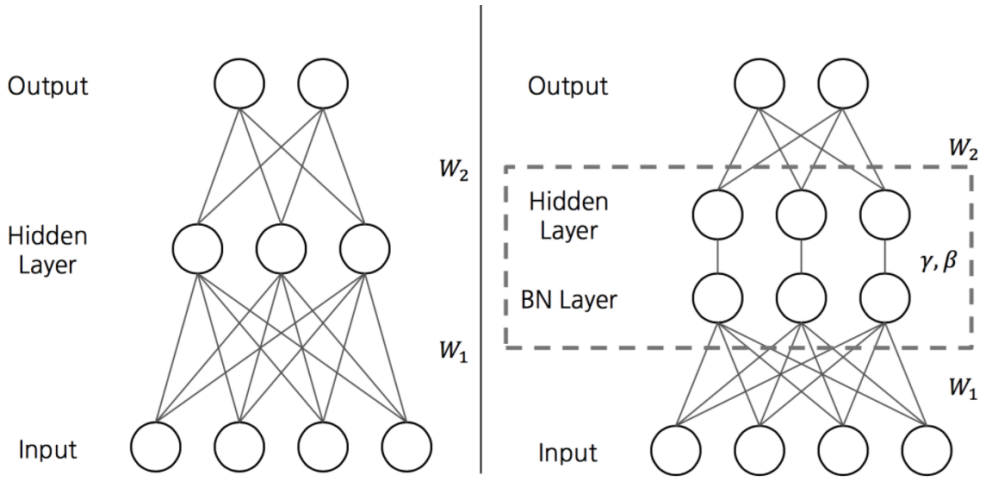


Figure 1.20: Without batch normalization (LEFT) vs. With batch normalization (RIGHT) [21]

learning speed gets slightly slower when we use dropout).

1.6 Classification

Classification represents classifying the data into multiple categories. While labels are continuous values in regression problems, labels are discretized values (classes) in classification problems. The classification technique could be applied to visual recognition and the detection of spam mail or credit card fraudulent transactions. There are two kinds of regression methods for classification: *Logistic regression* and *Softmax regression*.[4]

Logistic Regression

Logistic regression is a regression method for binary classification. In regression problems, the *hypothesis* of the model could potentially have any continuous value from $-\infty$ to ∞. However, in classification problems, we need to manipulate the hypothesis function so that its output could have values between 0 and 1 as probabilities, using the sigmoid function (Figure 1.21). We could express the hypothesis of logistic regression, $\mathbf{H}(\mathbf{X})$, as follows:

$$\mathbf{H}(\mathbf{X}) = \frac{1}{1 + exp(-\mathbf{z})} \tag{1.15}$$

[4]We should distinguish the word *regression* and the phrase *regression methods*. While *regression* is a type of machine learning algorithms [22] as we dealt in section **1.3**, the *regression methods* are mathematical methods for estimating the relationship between input variables and output variables.

where $\mathbf{z}$ represents the output of the function that operates on layers earlier than the sigmoid function. The sigmoid function converts the unscaled output of earlier layers, $\mathbf{z}$, to the new values between 0 and 1.

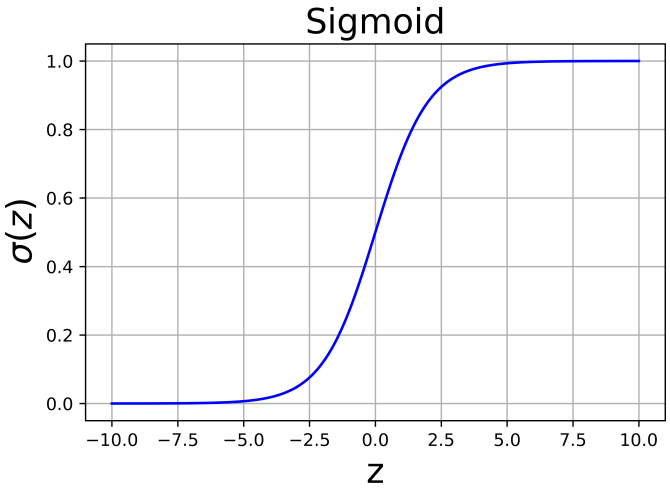


Figure 1.21: A sigmoid function

Note that there are only two classes, **which are 0 and 1**, in logistic regression. The hypothesis of logistic regression will have predicted values between 0 and 1. As we can see in Figure 1.21, the sigmoid function converts the values from $-\infty$ to ∞ into the values from 0 and 1.

We could write log loss function for $\mathbf{X_i}$,[5] the i^{th} data sample of the input $\mathbf{X}$, as follows:

$$\text{Loss} = \begin{cases} -log(H(\mathbf{X_i})) & \text{if } y_i = 1 \\ -log(1 - H(\mathbf{X_i})) & \text{if } y_i = 0 \end{cases} \tag{1.16}$$

where y_i represents the label corresponding to $\mathbf{X_i}$, and $H(\mathbf{X_i})$ represents the logistic hypothesis for $\mathbf{X_i}$. We observe that the loss becomes a minimum when $H(\mathbf{X_i}) = y_i$. Writing Equation 1.16 in the simpler form yields:

$$\text{Loss} = y_i log(H(\mathbf{X_i})) - (1 - y_i) log(1 - H(\mathbf{X_i})) \tag{1.17}$$

Now we could rewrite this form to express the overall loss function $L(\mathbf{W})$ for the data *set*:

$$L(\mathbf{W}) = -\frac{1}{N} \sum_{i=1}^{N} \left(y_i log(H(\mathbf{X_i})) - (1 - y_i) log(1 - H(\mathbf{X_i})) \right) \tag{1.18}$$

[5]For logistic regression with the hypothesis function shown in Equation 1.15, loss function in the mean squared error form is non-convex and not suitable for training. For further detail, refer to [23].

where N is the number of data samples in the data set (either for the training, validation, or test set). $\mathbf{X_i}$ and y_i represent the i^{th} data sample of the input $\mathbf{X}$ and the corresponding label, respectively.

Lastly, we could do logistic regression with $L(\mathbf{W})$, by gradient descent:

$$\mathbf{W} \leftarrow \mathbf{W} - \alpha \frac{\partial}{\partial \mathbf{W}}(L(\mathbf{W})) \tag{1.19}$$

where α is the learning rate.

Softmax Regression

Softmax regression is used when we need to solve multinomial classification problems. In logistic regression, we had only two kinds of labels. However, for multinomial classification problems, we need to classify the input data into more than two classes, as shown in Figure 1.22. Therefore, we need to use another shape of hypothesis function and loss function.

Since softmax regression should output the predicted probabilities for every input data sample to be in each class, the hypothesis $\mathbf{H}(\mathbf{X})$ on the input data $\mathbf{X}$ should have sizes of N by c, where N is the number of the data samples (either for the training, validation, or test set), and c is the number of the total classes.[6]

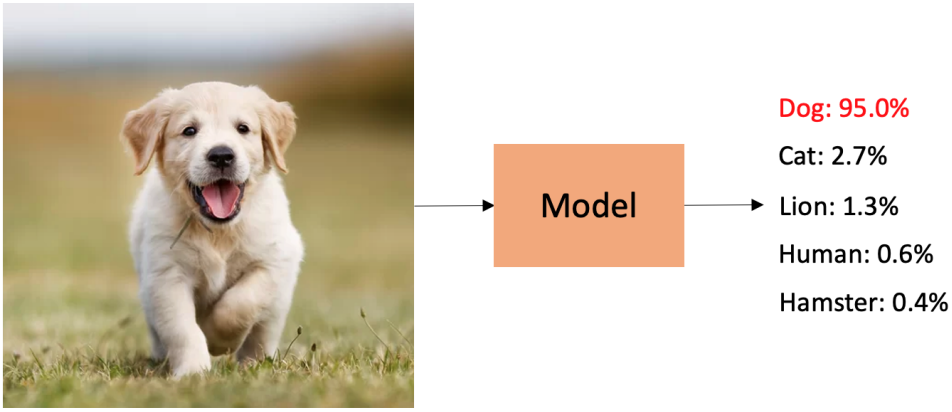


Figure 1.22: An example of multinomial classification

We could express the softmax probability for the m^{th} data sample to belong to the k^{th} class, $H_{m,k}(\mathbf{X})$, as follows:

$$H_{m,k}(\mathbf{X}) = \frac{e^{z_{m,k}}}{\sum_{i=1}^{c} e^{z_{m,i}}} \tag{1.20}$$

[6]This is a remarkable difference from the logistic regression, where $\mathbf{H}(\mathbf{X})$ has a size of N by 1.

where $z_{m,i}$ represents the m^{th} row and the i^{th} column of $\mathbf{z}$, which is the output of the function that operates on layers earlier than the softmax function. In this case, there are c classes in total, from class 1 to class c. $H_{m,k}(\mathbf{X})$ is the **element on the m^{th} row and k^{th} column** of $\mathbf{H}(\mathbf{X})$. Also, Equation 1.20 implies that the sum of the probabilities to be in all the classes should be 1.

For the loss function in multinomial classification, a cross-entropy loss function is used:

$$L(\mathbf{W}) = -\frac{1}{N}\sum_{m=1}^{N}\sum_{k=1}^{c}\left(L_{m,k}\cdot log(H_{m,k}(\mathbf{X}))\right) \tag{1.21}$$

where N refers to the number of the data samples, and $L_{m,k}$ represents the k^{th} column of the one-hot encoded label of m^{th} data sample. For example, if the true label of some data is class 3, $L_{m,(k=3)} = 1$ and $L_{m,(k\neq 3)} = 0$.

Lastly, we could do softmax regression with loss ($L(\mathbf{W})$), by gradient descent:

$$\mathbf{W} \leftarrow \mathbf{W} - \alpha\frac{\partial}{\partial \mathbf{W}}(L(\mathbf{W})) \tag{1.22}$$

where α is the learning rate.

1.7 Convolutional Neural Networks (CNN)

In the 1960s, Hubel and Wiesel [24] worked on the area of Sensory Processing. They experimented by inserting a microelectrode into the primary visual cortex of a partially anesthetized cat so that it can't move. They showed the images of the line at different angles to the cat, as shown in Figure 1.23.[7] Through the micro-electrode, they found out that some neurons fired very rapidly by watching the lines at specific angles, while other neurons responded best to the lines at different angles. Some of these neurons responded to lightness patterns differently, while other neurons focused on detecting motion in a certain direction. The concept of the Convolutional Neural Network was biologically inspired by this experiment. LeCun [25–27], a French-American computer scientist, is known as a founding father of Convolutional Neural Networks.

In deep learning, a *Convolutional Neural Network (CNN)* is one of the core classes of neural networks, which is most commonly applied to image classification and recognition [25–28]. The difference between the fully connected neural network (classical neural network) and the CNN is that the CNN has convolution layers that extract and determine the features from the image itself, in addition to the fully connected layers. A schematic of the CNN is shown in Figure 1.24. Also, a simple application of the CNN to a classification problem is shown in Figure 1.25. Note that the structure of the CNN shown in Figure 1.24 is simply one of the diverse CNN structures it could have. CNNs could have more complex and deeper architectures, as shown in Figures 1.26 and 1.27.

[7] Cat image by CNX OpenStax is licensed under CC BY 4.0; changes made

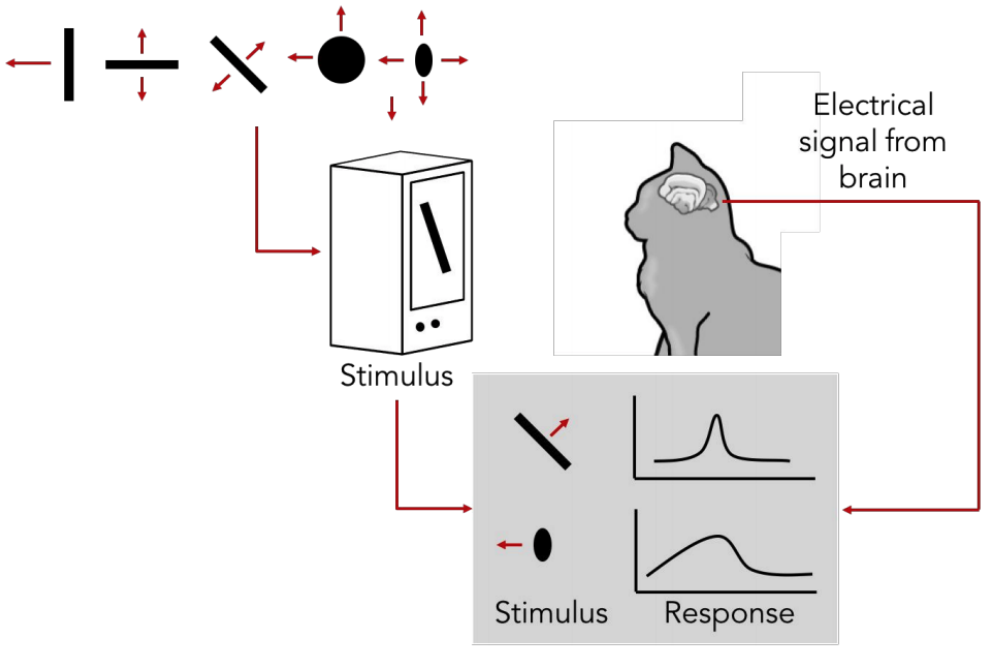


Figure 1.23: The experiment on the visual cortex of a cat - Hubel & Wiesel (1962) [9, 24]

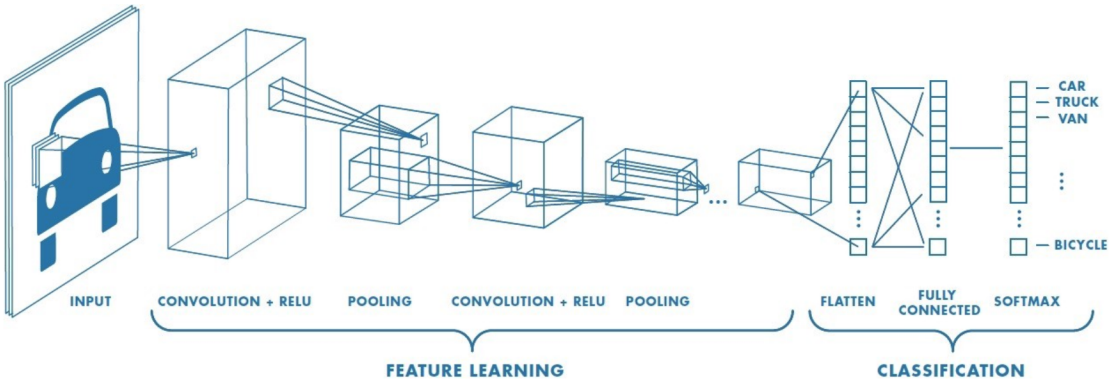


Figure 1.24: A CNN with many convolution layers [29]

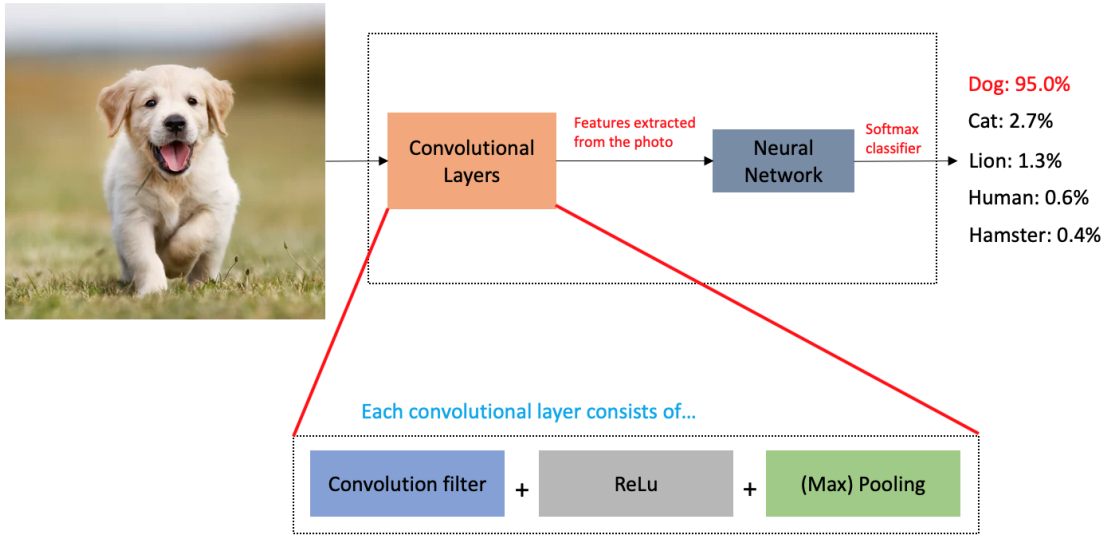


Figure 1.25: An application of the CNN to a classification problem

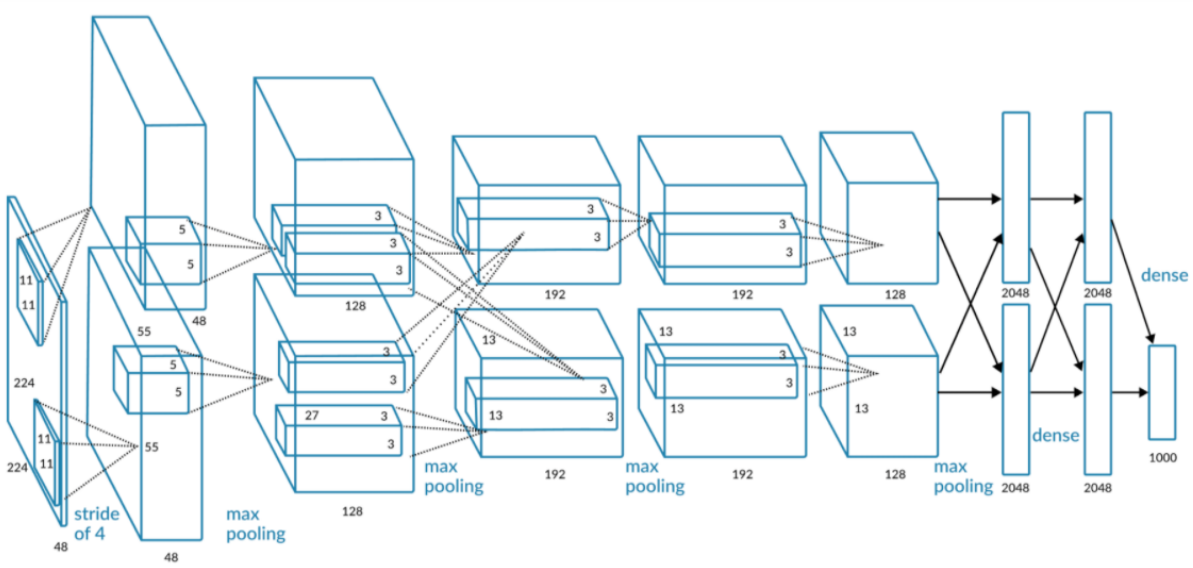


Figure 1.26: AlexNet - Krizhevsky, et al. (2012) [30]

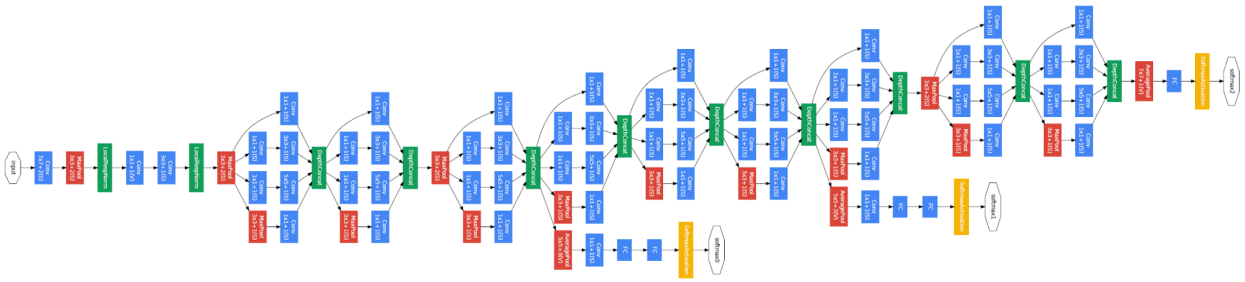


Figure 1.27: GoogLeNet - Szegedy, et al. (2014) [31]

As we could observe in Figure 1.24, the CNN is composed of *convolution layers* which perform feature learning, and a *fully connected neural network* which processes the output of convolution layers. Each convolution layer contains a set of filters (*convolution filters*, *activation function layers*, and *pooling layers*) whose parameters need to be learned along with the weights in the fully connected neural network. The explanations of the components in convolution layers are as follows.

Convolution Filters

Every single convolution layer has its *convolution filters*. As shown in Figure 1.28, we have an input image (input feature) entering the convolution layer and a convolution filter which has scalar weight values as filter elements. We multiply the convolution filter values to the input image values and get the sum of those values, by striding the filter from the left-top to the right bottom of the image by *stride*.[8] This process yields an output image (output feature) as shown in Figure 1.29. The weight values multiplied by convolution filters will be trained over and over during the training process.

Input Feature Map

3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

Convolutional Filter

1	0	0
1	1	0
0	0	1

Figure 1.28: An input feature and and a convolution filter

[8] *Stride* denotes how many steps we are shifting a convolution filter per each step in convolution.

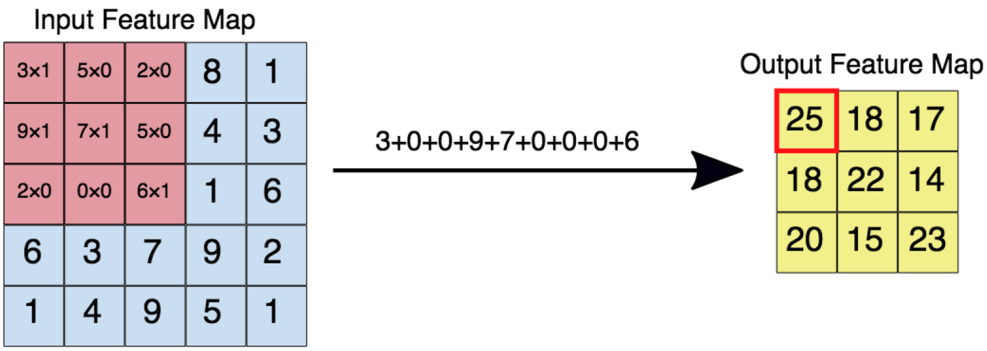


Figure 1.29: An input feature and a corresponding output feature

The output image size from a convolution filter

We could calculate the output image size from a convolution filter. We could assume that we have an input image that has a dimension of 32 by 32 by 3 (3 because of the RGB channel), and 10 different convolution filters which has a dimension of 5 by 5 by 3 (3 because of the RGB channel). We also assume that *stride* is 1 and *padding* is 2.[9] Then the new image will have a dimension of 32 by 32 by 10, based on the following relationship:

$$\text{Output size} = \frac{\text{Input size–Filter size} + 2 \times \text{Padding}}{\text{Stride}} + 1 \tag{1.23}$$

$$\begin{aligned} &\text{The number of filters of the input image} \\ &= \text{The number of channels of the output image} \end{aligned} \tag{1.24}$$

Activation Function Layers

We could give non-linearity to the filtered images by applying *activation functions*, as we did in the fully connected neural network. The activation functions are commonly ReLU (rectified linear units). Other choices might include sigmoid, tangent hyperbolic, leaky ReLU, maxout, and so on. For the readers who are interested in other kinds of activation functions, refer to [9].

Pooling Layers

We use *pooling layers* to detect and extract the summary information from the image. The pooling layers are responsible for reducing the dimension of the convolved feature. The main objective of pooling is to decrease the computational power required to process the data.

[9] *Padding* represents padding 0 values along the sides of the image before entering a convolution filter. For example, M by M by 3 image becomes $(M+4)$ by $(M+4)$ by 3 when padding is 2.

Max-pooling is the most popular pooling method. The concept of max-pooling is illustrated in Figure 1.30. Other pooling functions include the average of a rectangular neighborhood, L^2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel [32].

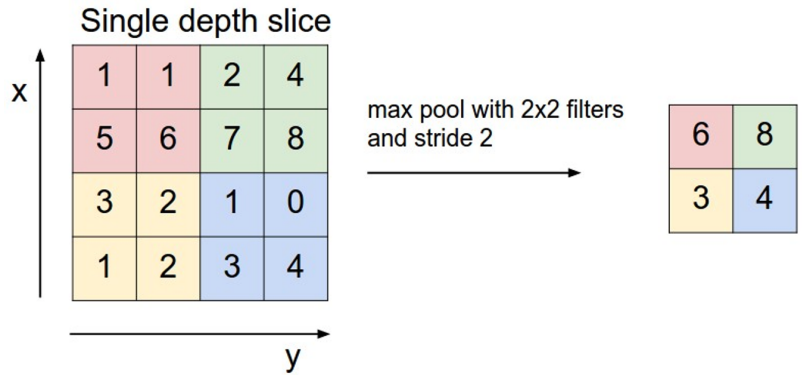


Figure 1.30: Max-pooling: pooling out the maximum value from a rectangular neighborhood

1.8 Recurrent Neural Networks (RNN)

A *Recurrent Neural Network (RNN)* is a class of deep neural networks that can train and predict sequential data. The RNNs have been applied to a variety of fields including voice recognition, speech recognition, machine translation, image/video captioning, and so on. An example structure of the RNN is shown in Figure 1.31.

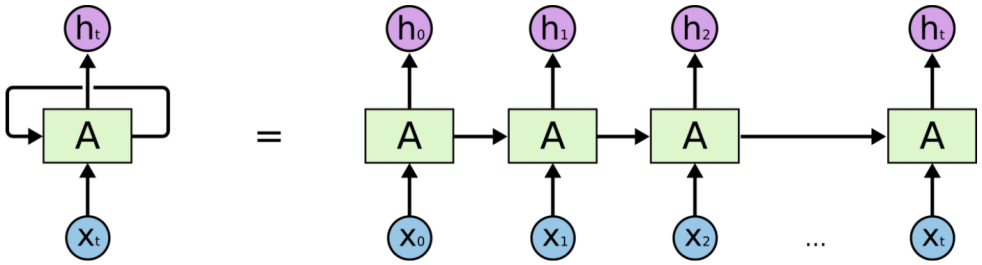


Figure 1.31: An example structure of the RNN [33]

Figure 1.31 shows the simplest structure of the RNN which is called a *Vanilla RNN*. $\mathbf{h}_t$ represents *(hidden) state* at the time t, and the previous *(hidden) state* affects to the next *state*. Every single state consists of a single *hidden vector* $\mathbf{h}$. We associate the new state

with the old state $\mathbf{h}_{t-1}$ and input vector $\mathbf{x}_t$, by some function with parameters $\mathbf{W}$ (which will be trained), as shown in Equation 1.25. For example, tangent hyperbolic could be used for the function to calculate the new state as in Equation 1.26. Based on the new state, we could calculate the output vector $\mathbf{y}_t$ at time t, as shown in Equation 1.27.

$$\mathbf{h}_t = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{1.25}$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t) \tag{1.26}$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t \tag{1.27}$$

Figure 1.32 shows how we train the sequence of the word 'hello'. All the characters appearing in the word 'hello' were one-hot encoded. The RNN should be able to predict that 'ello' will follow the very first character 'h'. Since the output layer still has an error (e.g. 'eolo'), we still need to train the model by **backpropagating** the loss function with softmax.

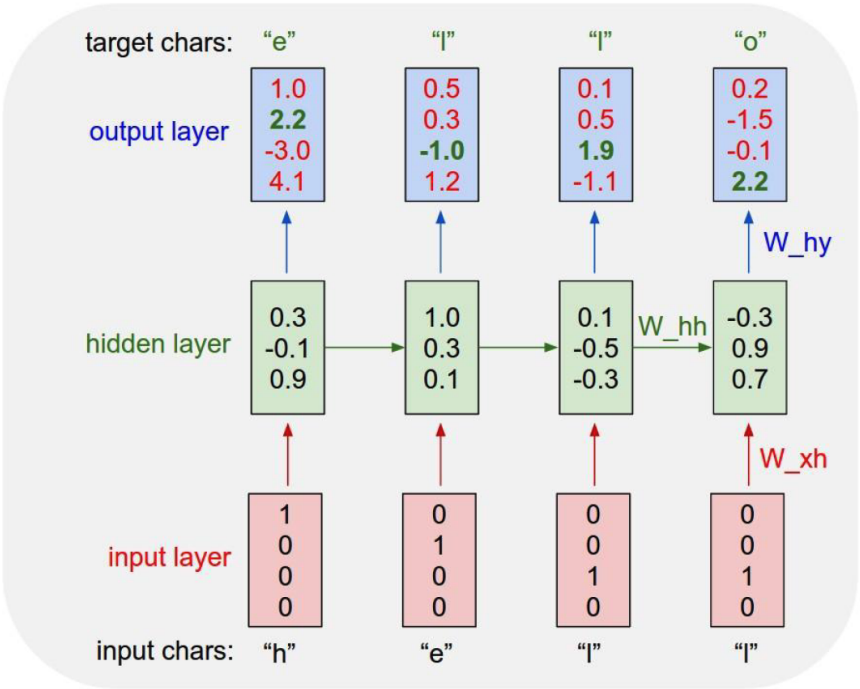


Figure 1.32: An example of the training sequence of word "hello" [9]

In addition to the Vanilla RNN, RNNs offer a lot of flexibility, as shown in Figures 1.33 and 1.34. For example, *one to many* structures could be used in image captioning by converting images into a sequence of words. *Many to one* type could be used in sentiment

classification by interpreting the sequence of words into sentiment. The first *many to many* type could be used for machine translation, and the second *many to many* type could be used to classify videos on frame level. The *multi-layer RNN*, shown in Figure 1.34, is also useful when we would like to train the RNN in a more complex manner.

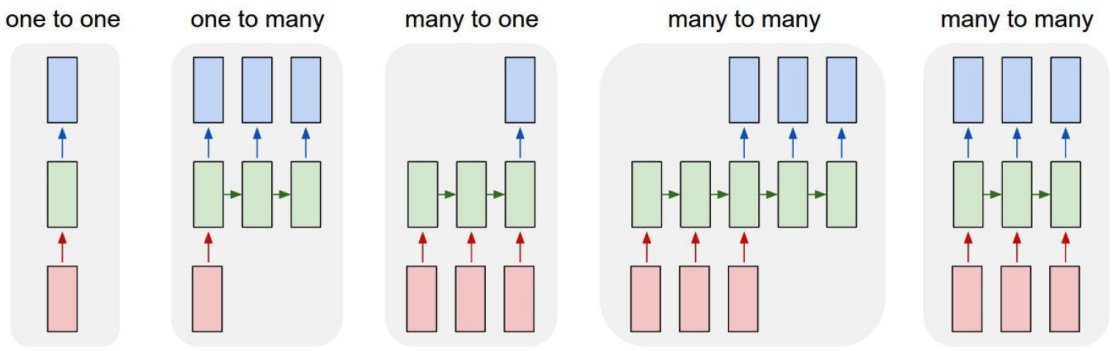


Figure 1.33: Diverse structures of the RNN [9]

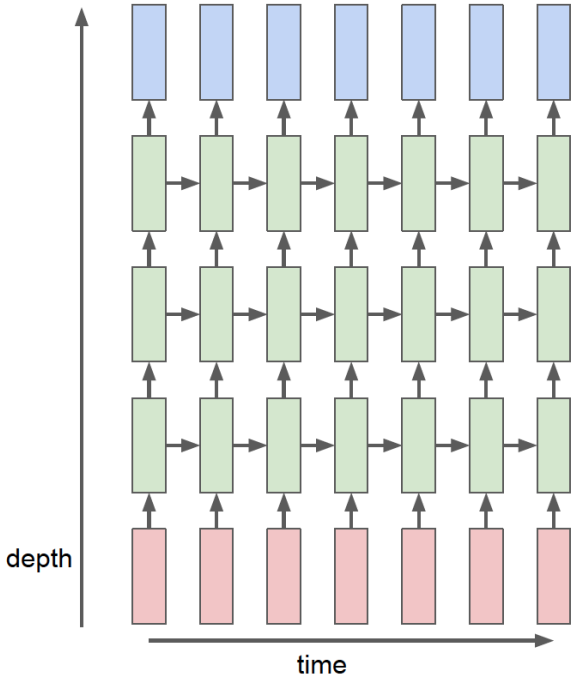


Figure 1.34: A multi-layer RNN [9]

Long Short-Term Memory (LSTM)

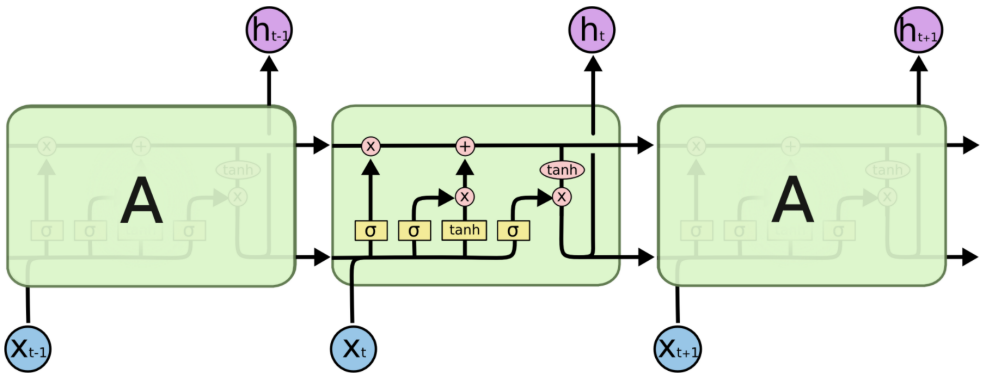


Figure 1.35: Repeating modules in the LSTM containing four interacting layers [33]

Long short-term memory (LSTM) is an advanced architecture of the RNN, which was invented by Hochreiter and Schmidhuber [34] in 1997 and sets accuracy records in multiple application domains. In recent years, employing the LSTM has become a necessary technique in many applications of the RNN. The LSTM is explicitly designed to avoid the long-term dependency problem.

For the long-term dependency problem, we could consider a language model trying to predict the next word based on the previous ones. If we are to predict the last word in "The clouds are in the", we do not need any further context. That is because it is obvious that the last word will be 'sky'. Since the gap between the relevant information and the last word is small enough, traditional RNNs can learn from the recent information and predict the next word.

However, traditional RNNs become unable to learn to connect the information in the cases when we need more previous context far away from the current sentence. For example, we could consider the case when the model needs to predict the last word 'Korean' from the long text "I was born in the United States but I moved to Korea when I was two years old. I have grown up in Korea. (...) I speak fluent". We need the context of Korea, which is further back from the recent sentences, to predict the last word, and traditional RNNs cannot learn to connect the information when the gap is large. The LSTM was proposed to overcome this limitation which traditional RNNs have.

The interacting layers in the LSTM module (Figure 1.35) are as follows:

1. The **forget gate layer** determines whether to throw away the information from the previous cell state.

2. The **input gate layer** determines which values to update.

3. The **tanh layer** creates a vector of the new candidate values that could be added to the state.

4. The **output layer** decides which parts it is going to output.

The LSTM structure mentioned above is simply a standard form of the LSTM. But not all LSTMs have the same structure as above. For example, Gers and Schmidhuber [35], and Cho, et al. [36] proposed different types of LSTM networks. For more detail on the LSTM, refer to [33].

1.9 The Overall Outlook and Further Algorithms

We have reviewed the core algorithms in supervised learning so far. There are also a variety of *unsupervised learning* algorithms. For example, an *Autoencoder* [37] is a type of artificial neural networks that is used to perform feature learning in an unsupervised manner. The objective of using an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction by training the network to ignore signal noise. This kind of idea could be applied to the reduced-order simulations in many applications. Other examples of unsupervised learning include *Deep Generative Models* with *Boltzmann Machines* [38, 39], *Deep Belief Networks (DBN)* [40], *Variational Autoencoders (VAE)* [41], and *Generative Adversarial Networks (GAN)* [42, 43]. Deep generative models are powerful ways of learning any kind of data distribution using unsupervised learning, and it has achieved great success in many fields recently. For the readers who are interested in more detail on both supervised learning and unsupervised learning, refer to [9, 32, 44–48]. *PyTorch* [49] and *TensorFlow* [50], which are the most popular and powerful machine learning frameworks, also provide good tutorials on deep learning on their websites [51, 52].

Chapter 2

Fundamentals of Non-Convex Optimization

2.1 Introduction

Optimization is the selection of the best parameters, which minimizes (or maximizes) the *objective function*, from some set of available candidates. Optimization problems are indispensable in all quantitative disciplines from computer science and engineering to economics and operations research. Even though we could easily solve optimization problems by using gradient-based methods when the objective function is convex, the majority of the optimization problems in engineering applications are highly non-convex.

In convex problems, a local minimum is also the global minimum if it is interior (not on the edge of the objective function). However, there may be multiple local minima in non-convex problems, which makes scientists and engineers difficult to solve for the global minimum. A large number of algorithms proposed for solving non-convex problems are still not capable of making a distinction between locally optimal solutions and globally optimal solutions. However, there are still powerful algorithms to solve non-convex optimization problems, which are capable of guaranteeing convergence in finite time to the highly desirable (near-optimal) solutions. In this chapter, I will review the three representative and powerful non-convex optimization techniques, *Bayesian Optimization*, *Genetic Algorithms*, and *Particle Swarm Optimization*.

2.2 Bayesian Optimization

Hyperparameters

In machine learning, *hyperparameters* are parameters whose values are used to control the learning process. As an example of a deep learning model, the learning rate, mini-batch size, regularization coefficients are typical hyperparameters that are related to learning algorithms

or regularization. Factors that determine the structure of deep learning models (e.g. the number of layers, convolution filter size, etc.) are also considered as hyperparameters.

Hyperparameter optimization refers to the problem of finding the optimal values of hyperparameter in machine learning and deep learning, which are values that must be set in advance to perform learning [32, 53]. Here, the optimal values of the hyperparameter mean the hyperparameters that enable the machine learning model to have the highest generalization performance (e.g. predictive power with a reasonable amount of training data). Hyperparameter optimization is criticial for obtaining good performance in many machine learning and deep learning algorithms [54].

However, since optimal hyperparameters are hard to be determined, many people make decisions depending on their intuition, experience, or publicly known know-how. Therefore, in many cases, the decision of optimal hyperparameters may need trial and error from repeating the learning process several times to find the optimal hyperparameters with limited computational resources.

As the simplest way, we could perform a manual search to find the optimal parameters. We set the candidates of hyperparameters and manually evaluate the corresponding predictive performances of the learning model. Even though the manual search is the most intuitive method for hyperparameter optimization, it still has some problems.

First, the process of searching for the *optimal* hyperparameter is somewhat dependent on luck. For example, there is a graph of performance of the machine learning model (objective function to maximize) in Figure 2.1. The blue line represents the objective function, and red markers correspond to the hyperparameter points we already evaluated the performance. We could evaluate objective function many times until when we are confident about 'discovered' best-performing hyperparameter. If one is lucky, the global maximum point could be evaluated and selected as an optimal point. However, there is less chance to get to the true global maximum by brute force manual search in reality. The example shown in Figure 2.1 is the simplest 1D function example to help readers easily understand the limitation of a manual search. This could become an even more difficult problem when there are a large number of hyperparameters, and the objective function is complex, noisy, or hard-to-evaluate.

The second problem with manual search is that the problem becomes more complicated when we want to search for multiple hyperparameters at once. The best example of this is the relationship between the learning rate and L_2 regularization coefficients. The following equation is a loss function of linear regression with L_2 regularization.

$$L(\mathbf{W}) = \|\mathbf{X}\mathbf{W} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{W}\|_2^2$$

The L_2 regularization term corresponds to the second term in the loss function equation. If the value of λ (regularization rate), which is the L_2 regularization coefficient, is changed on the entire parameter $\mathbf{W}$ space of the deep learning model, the shape of the loss function $L(\mathbf{W})$ also changes. As a result, it can be inferred that the value of the optimal learning rate to achieve optimal performance will also change naturally. As some types of hyperparameters

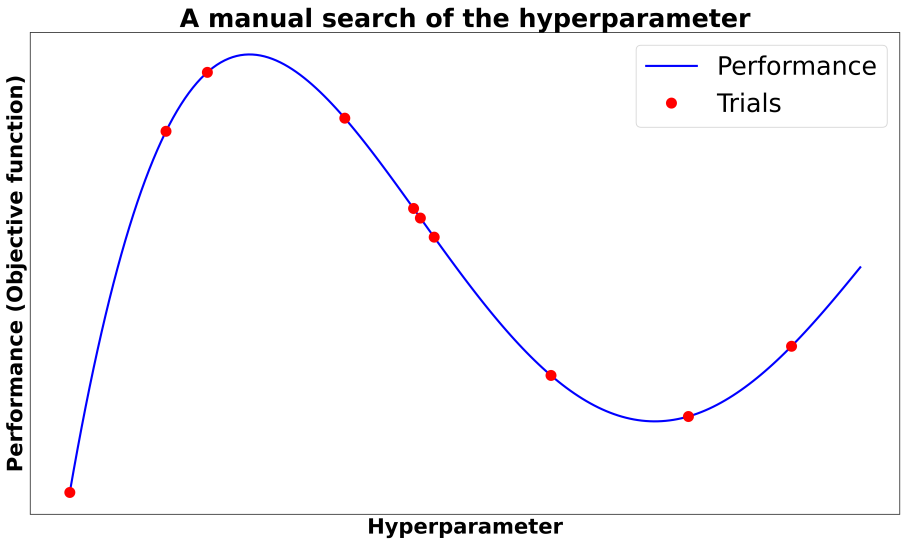


Figure 2.1: A manual search of the hyperparameter

show mutually influencing relationship with each other, it is very difficult to apply this idea to every single hyperparameter when searching for two or more hyperparameters at once.

The third problem of manual search is that we still need to do unnecessary calculations and explorations during trial and error. Even though we take advantage of the *prior knowledge and intuition* on choosing the next candidate hyperparameters, we still may not confident about whether the next candidate hyperparameters are better one until we evaluate the objective function.

Bayesian Optimization is a methodology that allows us to systematically carry out the entire exploration process while simultaneously reflecting *prior knowledge and intuition* when conducting investigations of new hyperparameter values each time. The main purpose of Bayesian optimization is **to tune the hyperparameters** in machine learning and deep learning [55, 56].

Bayesian Optimization

Bayesian Optimization is an approach to search for the global maximum or minimum of an objective function. It is an optimization method that is useful for **objective functions that are complex, noisy, and/or expensive to evaluate**. Bayesian optimization is a powerful strategy for finding the maximum or minimum of objective functions that are expensive to evaluate [55, 57–59].

Bayesian optimization aims to find the optimal solution x^* which maximizes the function

value $f(x)$ when unknown objective function f that takes an input value x is given.[1] It is usually assumed that the expression of the objective function is not known explicitly (i.e. black-box function) and it takes a long time to calculate a function $f(x)$. The main goal is to **efficiently** find the optimal solution x^* that maximizes $f(x)$, by examining the function value for as few input candidates as possible.

There are two essential elements in Bayesian optimization, which are a *surrogate model* and *acquisition functions*. A *surrogate model* is a stochastic estimation of the shape of an unknown objective function, which is constructed based on the real function points $(x_1, f(x_1)), ..., (x_t, f(x_t))$ investigated so far. *Acquisition functions* refer to the functions that recommend the next input value candidate x_{t+1} which is **the most useful for finding the optimal input value** x^*, based on the results of the probabilistic estimation of the objective function. The overall algorithm is shown in Algorithm 1.

Algorithm 1: Bayesian Optimization

for *t=1,2,...* **do**
- **STEP 1:** Obtain the stochastic estimation of surrogate model for the collection of existing points $(x_1, f(x_1)), ..., (x_t, f(x_t))$.
- **STEP 2:** Based on this, select the candidate for the next input value x_{t+1} that maximizes the acquisition function.
- **STEP 3:** Compute the function value, $f(x_{t+1})$, for the input value candidate x_{t+1}.
- **STEP 4:** Add $(x_{t+1}, f(x_{t+1}))$ to the collection of existing function value points.

end

A Surrogate Model

Based on the function points $(x_1, f(x_1)), ..., (x_t, f(x_t))$ investigated so far, a stochastic estimation of the *approximate form* of the unknown objective function is performed. This estimated model is called a *surrogate model*. The most widely used probabilistic model for the surrogate model is a *Gaussian Process (GP)* [55, 60–63]. In addition to the GP, a model that can cover the *uncertainty* in estimating the objective function, based on the input value points of the objective function investigated so far, can be used as a surrogate model. Other kinds of surrogate models that are frequently used include *tree-structured Parzen estimators (TPE)* and *deep neural networks*.

The *Gaussian Process (GP)* is a probability model that represents the probability distribution for functions, unlike the ordinary probability model (which expresses the probability distribution for a specific variable). In the GP, the joint distribution between its components is characterized by the Gaussian distribution, assuming a multivariable Gaussian distribution. The GP is capable of efficient and effective summarization of a large number of functions and smooth transition as more observations become available to the model. The

[1] For simplicity, we confine Bayesian optimization to the 1D case in the contents of this section.

GP expresses the probability distribution for the functions using the mean function μ and the covariance function k as follows.

$$f(x) \sim GP(\mu(x), k(x, x^{'})) \tag{2.1}$$

Based on the function points investigated so far, the GP performs stochastic estimation of the objective function as shown in Figure 2.2.

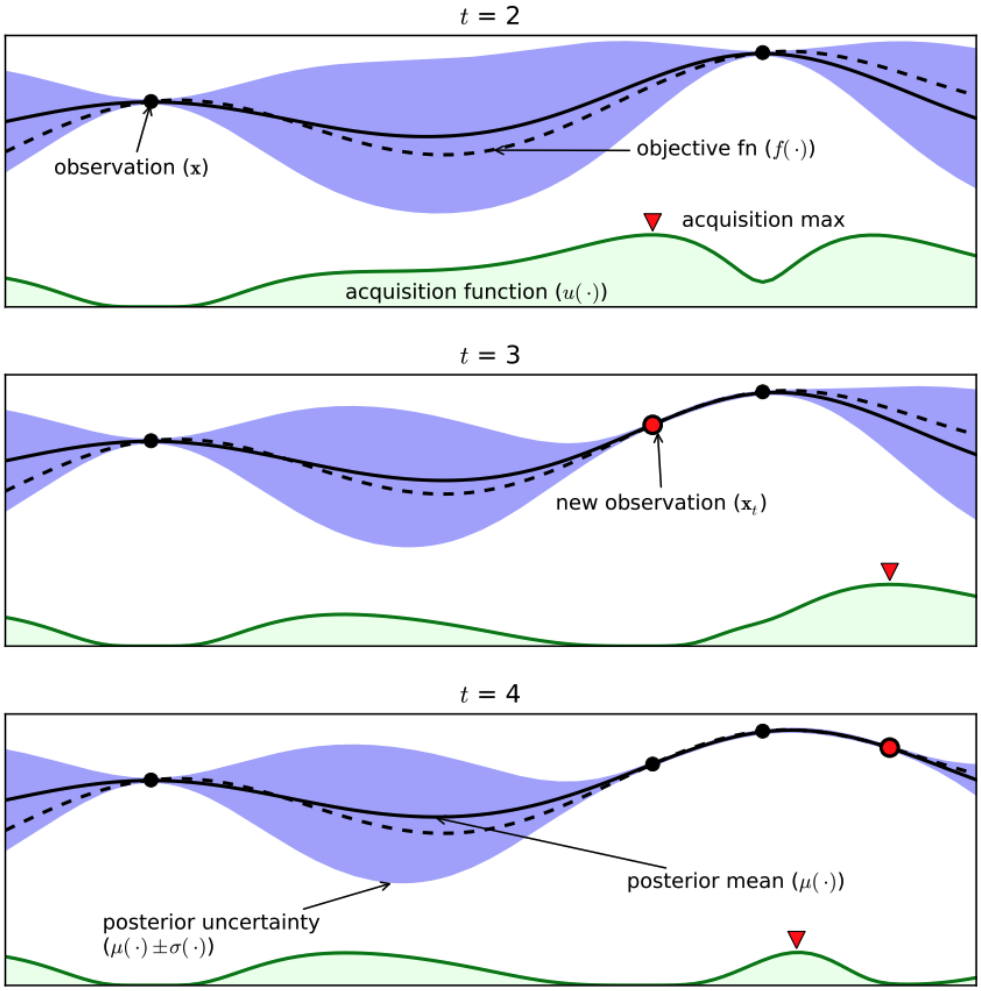


Figure 2.2: An example of Bayesian optimization process using the GP approximation over four iterations - Brochu et al. (2010) [55]

- Black dotted line: Actual objective function
- Black solid line: Estimated mean of the surrogate model

- Shaded blue: Estimated standard deviation of the surrogate model
- Black points: Function point investigated so far
- Green solid line at the bottom: Acquisition Function

In Figure 2.2, the horizontal axis is the input value x, and the vertical axis is the function value $f(x)$. The black solid line indicates an estimated average $\mu(x)$ for all the points, which were estimated based on points $(x_1, f(x_1)), ..., (x_t, f(x_t))$ investigated so far. The blue shaded region corresponds to the standard deviation $\sigma(x)$ for all the points, which were estimated based on points investigated so far. The shape of $\mu(x)$ is determined to pass the points which have already been investigated. Also, we could observe that standard deviation $\sigma(x)$ gets larger if some point gets far away from investigated points, and vice versa.

When $t = 2$ in Figure 2.2, we observe that the blue shaded region is large if it is far away from the two investigated points. On the other hand, as the number of investigated points increases (e.g. $t = 3$ and $t = 4$), the size of the blue shaded region gradually decreases. This implies that the estimation of the objective function gains more certainty, as the number of investigated points increases. As *uncertainty* decreases, the likelihood of finding the input value x^*, which maximizes the objective function, increases.

Acquisition Functions

Based on the stochastic estimates of the objective function from the surrogate model (e.g. GP), the *acquisition function* recommends the next candidate point x_{t+1} on which we evaluate the objective function. x_{t+1} that acquisition function selected is **the most useful** point for finding the optimal input x^* of the objective function.

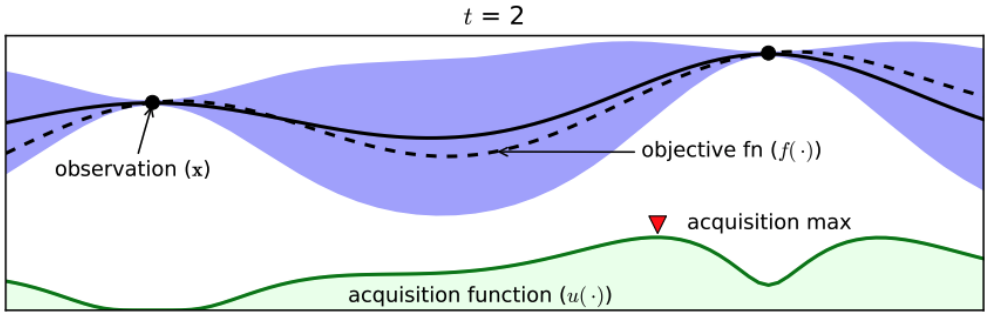


Figure 2.3: An acquisition function - Brochu et al. (2010) [55]

Two points have been investigated so far in $t = 2$, as shown in Figure 2.3. It is reasonable to think that there would be a higher probability to find the optimal point x^* around the point with the higher function value (the point on the right in Figure 2.3). Accordingly, it

is a reasonable strategy to investigate the region around the point with the largest function value, among all the points investigated so far. This form of strategy is called *exploitation*.

On the other hand, the optimal point x^* may exist in the region with large *uncertainty*, and it is hard to guarantee that the estimated mean function in this region would be similar to the true objective function. Therefore, it will be a reasonable strategy to *explore* the *blue shaded* region with large *uncertainty* and select the next candidate point from the region with the largest deviation in the estimated objective function. This form of strategy is called *exploration*.

Both exploration and exploitation strategies are equally important strategies for **effectively** finding the optimal input value x^*, but the two strategies are in a trade-off relationship with each other. Therefore, it is important to properly control the relative strength between exploration and exploitation to successfully find the optimal input value for the true objective function.

There are two kinds of acquisition functions that are widely used. The one is the *Expected Improvement (EI)*, and the other one is the *Probability of Improvement (PI)*. The PI function is the simplest, and the EI function is the most commonly used.[2]

Expected Improvement (EI)

The *Expected Improvement (EI)* function is designed to take both exploration and exploitation strategies and is most often used as an acquisition function. The EI gives us the **usefulness** of the candidate x, based on the **probability of having greater value** than the maximum function value investigated so far ($f(x^+) = \max_i f(x_i)$) and the **magnitude of the difference** between $f(x^+)$ and the function value of the candidate x.

In Figure 2.4, 'PI' represents the Probability of Improvement, which will be explained in the following part of this section. In this case, the maximum value $f(x^+)$ of the points investigated so far is at the rightmost point. For the candidate point x_3 located further to the right, the probability distribution of $f(x_3)$ can be displayed in the form of Gaussian distribution, based on the probability estimation result. The green shaded area represents the probability that the function value of the new candidate point x_3 is greater than $f(x^+)$. The larger the size of this area, the higher the probability that $f(x_3)$ is greater than $f(x^+)$. In other words, if x_3 is adopted as the next input point, it is more likely to obtain a larger function value than the existing three points. Also, the difference between the mean of x_3 ($\mu(x_3)$) and $f(x^+)$ is calculated. Accordingly, the EI value for x_3 is finally calculated considering both **how much likely to obtain a function value that is larger than existing points** and **how big the difference is**, as shown in Equations 2.2 and 2.3. For

[2]There are many other options for the acquisition function, such as Upper Confidence Bound (UCB) and Entropy Search (ES).

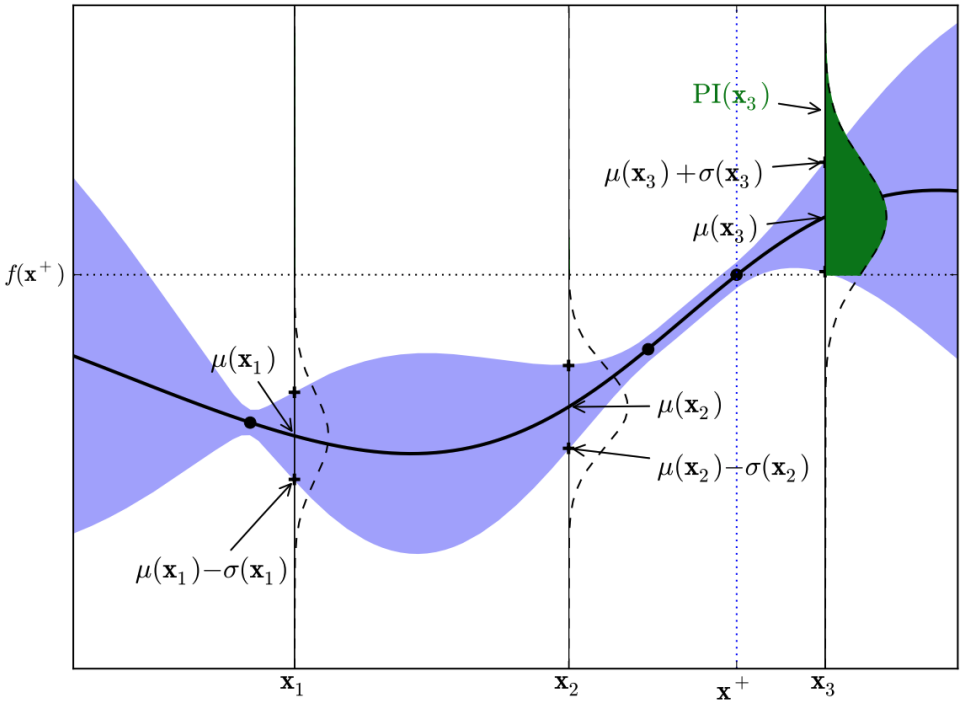


Figure 2.4: An example of visualizing probability to derive a function value greater than the maximum function value $f(x^+)$ - Brochu et al. (2010) [55]

the readers who are interested in the derivation of Equations 2.2 and 2.3, refer to [64, 65].

$$
\begin{aligned}
&EI(x) = \mathbb{E}[\max(f(x) - f(x^+), 0)] \\
&= \begin{cases} (\mu(x) - f(x^+) - \xi)\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases}
\end{aligned} \tag{2.2}
$$

$$
Z = \begin{cases} \frac{\mu(x) - f(x^+) - \xi}{\sigma(x)} & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \tag{2.3}
$$

In Equations 2.2 and 2.3, Φ and ϕ represents the cumulative distribution function (CDF) and the probability density function (PDF), respectively. ξ represents the parameter that balances the strength of exploration and exploitation ($\xi \geq 0$) [66]. Exploration gets dominant as ξ gets bigger, and exploitation gets dominant as ξ gets smaller.

Probability of Improvement (PI)

The *Probability of Improvement (PI)* is an acquisition function that was proposed earlier than the EI in the history of Bayesian optimization. The PI evaluates only the **probability**

to obtain a function value that is greater than the maximum function value of the points investigated so far. The *score* of the new candidate point evaluated by the PI could be described as follows. Φ represents the CDF. Since standard deviation, $\sigma(x)$, represents *uncertainty*, a candidate point with a lower deviation will have a better score.

$$PI(x) = \Phi(\frac{\mu(x) - f(x^+)}{\sigma(x)}) \tag{2.4}$$

Based on the calculated PI value, we add the new point which has the highest PI value.

The Implementation of Bayesian Optimization

The following example illustrates the simple implementation of Bayesian optimization to find the global maximum in the 1D non-convex function. In this case, I used the following configuration.

- Surrogate model: Gaussian Process
- Acquisition function: Probability of Improvement (PI)
- The number of iterations: 100

Also, the objective function is as follows.

$$f(x) = 4(x-2)^2 sin^5(2\pi x) + 1.2^x + Noise \tag{2.5}$$

where $Noise$ is the probability density function of the Gaussian distribution, with $\mu = 0$ (mean) and $\sigma = 0.1$ (standard deviation).

In Figures 2.5 and 2.6, the blue line represents the average of the surrogate model after fitting the investigated points with the Gaussian process (GP) regressor. Red dots represent the **true** points on the objective function that Bayesian optimizer selected and evaluated as **the most useful** points for finding the optimal input value x^*. After investigating the useful points during the iterations as shown in Figure 2.6, it also discovered that $(\mathbf{x}, \mathbf{y}) = (\mathbf{0.241}, \mathbf{13.305})$ is the optimal point of the function which maximizes the objective function value. From Figure 2.6, we could observe that the region around the optimum has much more investigated samplings (red dots) than Figure 2.5 because they have been regarded as **useful** sample points to find the optimum during the optimization process.

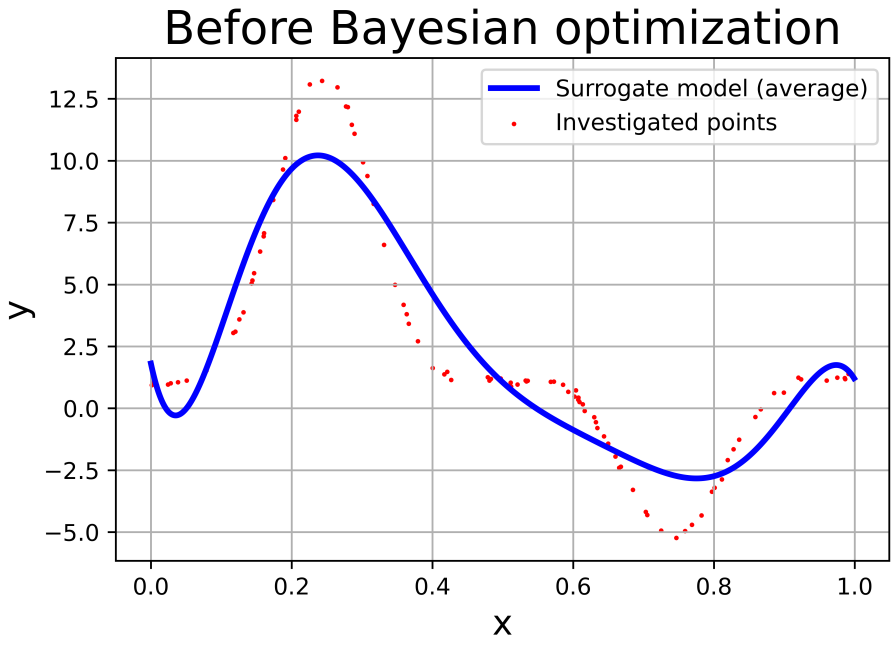


Figure 2.5: All samples (dots) and the surrogate function (line) before Bayesian optimization

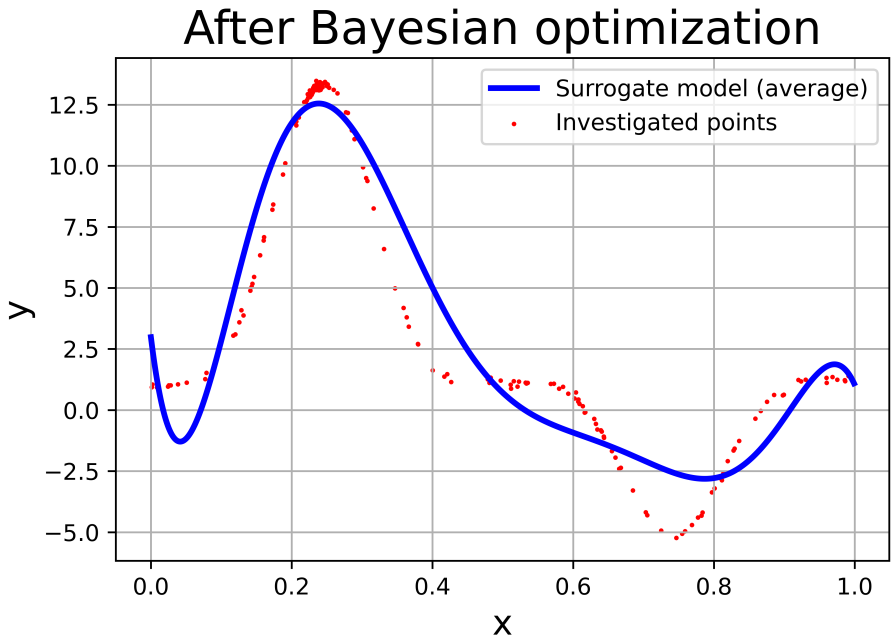


Figure 2.6: All samples (dots) and the surrogate function (line) after Bayesian optimization

2.3 Genetic Algorithms

In many optimization problems, cost functions are often non-convex in design parameter space and often nonsmooth. Their minimization is usually difficult with the direct application of gradient methods. A *Genetic Algorithm (GA)* is a **non-derivative and easy-to-parallelize** optimization method introduced by Holland [67, 68], which has both **affordable computational cost and high optimization performance**. The GA can treat a wide variety of non-convex inverse problems involving various aspects of multiphysics/multiscale phenomena [69–72].

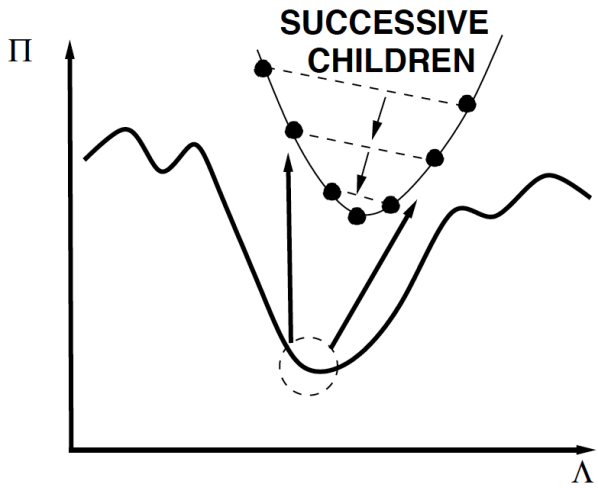


Figure 2.7: The basic action of the genetic algorithm

Suppose we have a multi-variate objective function which we want to minimize:

$$\Pi(\mathbf{\Lambda}) = \Pi(\lambda_1, \lambda_2, ..., \lambda_n) \tag{2.6}$$

According to Zohdi [69], the GA could be described as shown in Algorithm 2. The basic idea of the GA comes from 'mating' the parent strings to obtain the offspring strings by combining the parameters in the parent strings, as shown in Figure 2.7. In Algorithm 2, *performance* represents the evaluated objective function value for each string.[3] For example, if we have an objective function we want to minimize, the smaller objective function value on a certain parameter combination has better *performance*.

If one does not retain the parents in Algorithm 2, inferior performing offspring strings or inferior new strings may replace superior parents. Thus, top parents should be kept for the next generation. This guarantees a monotone reduction in the cost function, and this is critical for proper convergence.

[3]There are N system parameters in this case.

Algorithm 2: Genetic Algorithms

Initialization: Randomly generate a population of S initial strings, $\mathbf{\Lambda}^i$ $(i = 1, 2, 3, ..., S)$. $\longrightarrow \Lambda^i \triangleq \{\lambda_1^i, \lambda_2^i, \lambda_3^i, ..., \lambda_N^i\}$

while *min*$(\Pi) \geq$ *Tolerance* **do**

STEP 1: Compute the *performance* of each genetic string $\Pi(\mathbf{\Lambda}^i)$ $(i = 1, 2, 3, ..., S)$, and rank those strings based on the *performance* values.

STEP 2: Mate the best performing P parent strings to generate C offspring strings. $\mathbf{\Lambda}^{NEW} \triangleq \mathbf{\Phi} \odot \mathbf{\Lambda}^{OLD_1} + (\mathbf{1} - \mathbf{\Phi}) \odot \mathbf{\Lambda}^{OLD_2}$
, where $\mathbf{\Phi} = \{\phi_1, \phi_2, \phi_3, ..., \phi_N\}$, and $0 \leq \phi_k \leq 1$ $(k = 1, 2, ..., N)$.
$\odot$ represents component-wise multiplication.

STEP 3: Replace the worst performing C strings in the old genetic strings with the new child strings obtained in **STEP 2**.

STEP 4:
if *Keeping parents* **then**
Keep the old P parent strings
Generate $S - P - C$ new strings
else
Remove the old P parent strings
Generate $S - C$ new strings
end

end

(Optional) Employ gradient-based methods afterward in the local minima, if the neighbor of the obtained optimal point is smooth enough.

Furthermore, retained parents do not need to be re-evaluated-making the algorithm less computationally less expensive, since these parameter sets do not need to be re-evaluated in the next generation. Numerous studies have shown that the advantage of retaining parents outweighs the advantage of generating new strings, for sufficiently large populations [73–75].

Additionally, if one selects the 'mating' parameters in $\mathbf{\Phi}$ to be greater that one and/or less than zero, one can induce 'mutations' to have characteristics that neither parent possesses. However, this is somewhat redundant with the introduction of new random members of the population in the current algorithm.

The Application of the Genetic Algorithm: Material Optimization

This subsection is the application of the genetic algorithm in multiphase material optimization. The contents of this subsection are from the class project of the *Mechanical Engineering C201 class (2017) from Professor Tarek I. Zohdi*.

When we would like to design new multiphase material (Figure 2.8) by adding particulate material to the matrix (binding) material, we need to find the optimal microscopic properties of particulate materials to get desirable macroscopic responses of the new material (including bulk/shear moduli, electrical conductivity, stress concentration and so on). For simplicity, we confine material optimization to the *two-phase* isotropic materials in this work.[4] The final objective is to optimize the two-phase material using a macro-micro objective function and get the top 10 performing design parameters.

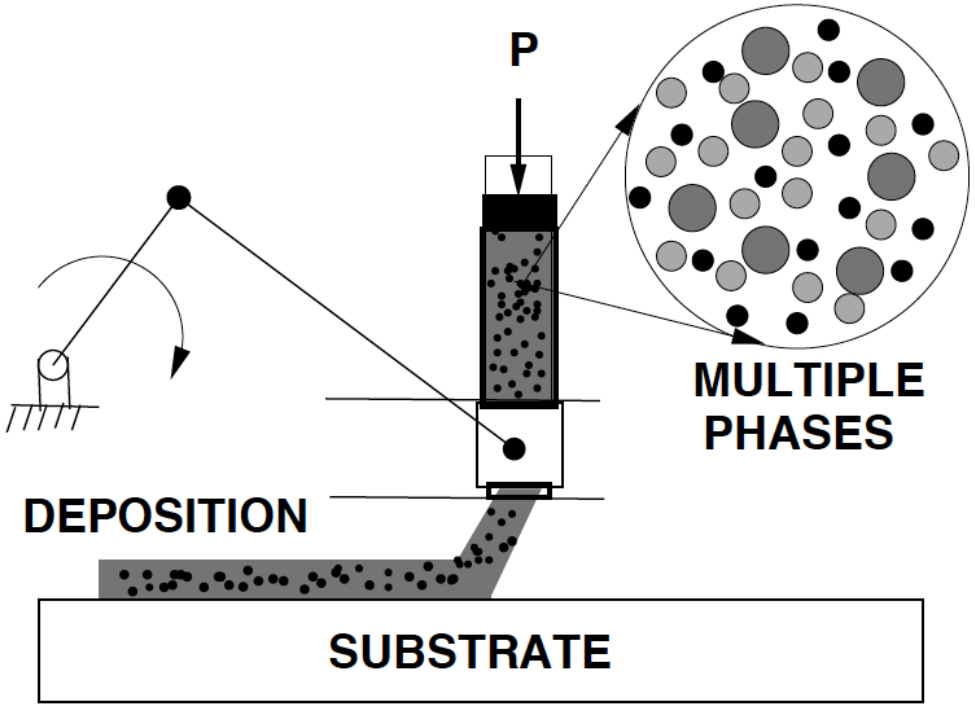


Figure 2.8: 3D printing of a multiphase material [70, 76]

[4]For the readers who are interested in making estimates of the overall properties of a mixture of more than two materials, refer to Zohdi [70].

Effective Property Bounds

Consider the widely used Hashin and Shtrikman bounds [77–79] for multiphase isotropic materials with isotropic effective responses. For the bulk modulus,

$$\kappa^{*,-} \overset{def}{=} \kappa_1 + \frac{v_2}{\frac{1}{\kappa_2-\kappa_1} + \frac{3(1-v_2)}{3\kappa_1+4\mu_1}} \leq \kappa^* \leq \kappa_2 + \frac{1-v_2}{\frac{1}{\kappa_1-\kappa_2} + \frac{3v_2}{3\kappa_2+4\mu_2}} \overset{def}{=} \kappa^{*,+}, \tag{2.7}$$

and for the shear modulus

$$\mu^{*,-} \overset{def}{=} \mu_1 + \frac{v_2}{\frac{1}{\mu_2-\mu_1} + \frac{6(1-v_2)(\kappa_1+2\mu_1)}{5\mu_1(3\kappa_1+4\mu_1)}} \leq \mu^* \leq \mu_2 + \frac{(1-v_2)}{\frac{1}{\mu_1-\mu_2} + \frac{6v_2(\kappa_2+2\mu_2)}{5\mu_2(3\kappa_2+4\mu_2)}} \overset{def}{=} \mu^{*,+}, \tag{2.8}$$

where κ_2 and κ_1 are the bulk moduli and μ_2 and μ_1 are the shear moduli of the respective phases ($(\kappa_2 \geq \kappa_1)$ and $(\mu_2 \geq \mu_1)$). v_2 is the second phase volume fraction.

Such bounds are the tightest known on isotropic effective responses, with isotropic two-phase microstructures, where only the volume fractions and phase contrasts of the constituents are known.

As a model problem, our goal is to computationally design the macroscale effective bulk and shear moduli κ^* and μ^*, using linear combinations of the Hashin-Shtrikman bounds as approximations for the effective moduli $\kappa^* \approx \theta\kappa^{*+} + (1-\theta)\kappa^{*-}$ and $\mu^* \approx \theta\mu^{*+} + (1-\theta)\mu^{*-}$, where $0 \leq \theta \leq 1$.

We can also use the Hashin and Shtrikman bounds for the overall electrical conductivity σ_e^*

$$\langle \sigma_e^{-1}(\boldsymbol{x}) \rangle_\Omega^{-1} \leq \underbrace{\sigma_{1,e} + \frac{v_2}{\frac{1}{\sigma_{2,e}-\sigma_{1,e}} + \frac{1-v_2}{3\sigma_{1,e}}}}_{\sigma_e^{*,-}} \leq \sigma_e^* \leq \underbrace{\sigma_{2,e} + \frac{1-v_2}{\frac{1}{\sigma_{1,e}-\sigma_{2,e}} + \frac{v_2}{3\sigma_{2,e}}}}_{\sigma_e^{*,+}} \leq \langle \sigma_e(\boldsymbol{x}) \rangle_\Omega \tag{2.9}$$

where $\sigma_{2,e} \geq \sigma_{1,e}$, and v_2 is the volume fraction of phase 2. Note that the subscript in σ_e^* is to distinguish between a mechanical stress component and is meant to denote *electric* conductivity.

The original proofs, which are algebraically complicated, can be found in Hashin and Shtrikman [77–79]. We emphasize that in the derivation of the bounds, the body is assumed to be infinite, the micro-structure is isotropic, and that the effective responses are isotropic. We remark that the bounds are the tightest possible, under the previous assumptions that no geometric (micro-topological) information is included.

Stress and Strain Concentration Factors

In addition to the effective (macroscopic) properties, when selecting particulate micro-additives for a base matrix, information about the changes in the otherwise (relatively)

smooth internal fields, corresponding to the matrix material alone, is valuable to characterize a new tailored material's performance. For this work, one way to analytically characterize the smoothness of the microscopic field behavior is via (stress and strain) concentration tensors, which provide a measure of the deviation away from the mean fields throughout the material. We could consider the following identities:

$$\langle \boldsymbol{\epsilon} \rangle_{\Omega} = \frac{1}{|\Omega|}\left(\int_{\Omega_1} \boldsymbol{\epsilon}\, d\Omega + \int_{\Omega_2} \boldsymbol{\epsilon}\, d\Omega\right) = v_1 \langle \boldsymbol{\epsilon} \rangle_{\Omega_1} + v_2 \langle \boldsymbol{\epsilon} \rangle_{\Omega_2} \tag{2.10}$$

and

$$\langle \boldsymbol{\sigma} \rangle_{\Omega} = \frac{1}{|\Omega|}\left(\int_{\Omega_1} \boldsymbol{\sigma}\, d\Omega + \int_{\Omega_2} \boldsymbol{\sigma}\, d\Omega\right) = v_1 \langle \boldsymbol{\sigma} \rangle_{\Omega_1} + v_2 \langle \boldsymbol{\sigma} \rangle_{\Omega_2}. \tag{2.11}$$

By direct manipulation we obtain

$$\begin{aligned} \langle \boldsymbol{\sigma} \rangle_{\Omega} &= v_1 \langle \boldsymbol{\sigma} \rangle_{\Omega_1} + v_2 \langle \boldsymbol{\sigma} \rangle_{\Omega_2} \\ &= v_1 \mathbb{E}_1 : \langle \boldsymbol{\epsilon} \rangle_{\Omega_1} + v_2 \mathbb{E}_2 : \langle \boldsymbol{\epsilon} \rangle_{\Omega_2} \\ &= \mathbb{E}_1 : (\langle \boldsymbol{\epsilon} \rangle_{\Omega} - v_2 \langle \boldsymbol{\epsilon} \rangle_{\Omega_2}) + v_2 \mathbb{E}_2 : \langle \boldsymbol{\epsilon} \rangle_{\Omega_2} \\ &= (\mathbb{E}_1 + v_2(\mathbb{E}_2 - \mathbb{E}_1) : \boldsymbol{C}) : \langle \boldsymbol{\epsilon} \rangle_{\Omega} \end{aligned} \tag{2.12}$$

where

$$\underbrace{\left(\frac{1}{v_2}(\mathbb{E}_2 - \mathbb{E}_1)^{-1} : (\mathbb{E}^* - \mathbb{E}_1)\right)}_{\stackrel{def}{=} \boldsymbol{C}} : \langle \boldsymbol{\epsilon} \rangle_{\Omega} = \langle \boldsymbol{\epsilon} \rangle_{\Omega_2}. \tag{2.13}$$

Thereafter, we may write the following for the variation in the stress: $\boldsymbol{C} : \mathbb{E}^{*-1} : \langle \boldsymbol{\sigma} \rangle_{\Omega} = \mathbb{E}_2^{-1} : \langle \boldsymbol{\sigma} \rangle_{\Omega_2}$, which reduces to

$$\mathbb{E}_2 : \boldsymbol{C} : \mathbb{E}^{*-1} : \langle \boldsymbol{\sigma} \rangle_{\Omega} \stackrel{def}{=} \overline{\boldsymbol{C}} : \langle \boldsymbol{\sigma} \rangle_{\Omega} = \langle \boldsymbol{\sigma} \rangle_{\Omega_2}. \tag{2.14}$$

$\overline{\boldsymbol{C}}$ is known as the stress concentration tensor. Therefore, once either $\overline{\boldsymbol{C}}$ or $\mathbb{E}^*$ are known, the other can be determined.

In the case of isotropy we may write:

$$\overline{C}_{\kappa} \stackrel{def}{=} \frac{1}{v_2}\frac{\kappa_2}{\kappa^*}\frac{\kappa^* - \kappa_1}{\kappa_2 - \kappa_1} \qquad and \qquad \overline{C}_{\mu} \stackrel{def}{=} \frac{1}{v_2}\frac{\mu_2}{\mu^*}\frac{\mu^* - \mu_1}{\mu_2 - \mu_1} \tag{2.15}$$

where $\overline{C}_{\kappa}\langle \frac{tr\boldsymbol{\sigma}}{3} \rangle_{\Omega} = \langle \frac{tr\boldsymbol{\sigma}}{3} \rangle_{\Omega_2}$ and where $\overline{C}_{\mu}\langle \boldsymbol{\sigma}' \rangle_{\Omega} = \langle \boldsymbol{\sigma}' \rangle_{\Omega_2}$. The microstress fields are minimally distorted when $\overline{C}_{\kappa} = \overline{C}_{\mu} = 1$.

For the matrix, since $\langle \boldsymbol{\sigma} \rangle_{\Omega_1} = \frac{\langle \boldsymbol{\sigma} \rangle_{\Omega} - v_2 \langle \boldsymbol{\sigma} \rangle_{\Omega_2}}{v_1}$, we could write:

$$\langle \boldsymbol{\sigma} \rangle_{\Omega_1} = \frac{\langle \boldsymbol{\sigma} \rangle_{\Omega} - v_2 \overline{\boldsymbol{C}} : \langle \boldsymbol{\sigma} \rangle_{\Omega}}{v_1} = \frac{(1 - v_2 \overline{\boldsymbol{C}}) : \langle \boldsymbol{\sigma} \rangle_{\Omega}}{v_1} = \overline{\overline{\boldsymbol{C}}} : \langle \boldsymbol{\sigma} \rangle_{\Omega}. \tag{2.16}$$

Also, in the case of isotropy, we could write:

$$\overline{\overline{C}}_{\kappa} \overset{def}{=} \frac{1}{v_1}(1 - v_2 \overline{C}_{\kappa}) \quad \text{and} \quad \overline{\overline{C}}_{\mu} \overset{def}{=} \frac{1}{v_1}(1 - v_2 \overline{C}_{\mu}). \tag{2.17}$$

Finally, we get the following relations for the deviations in the particulate stress fields.

1. The deviation away from the mean for pressure:

$$|\frac{\langle tr\sigma \rangle_{\Omega_2} - \langle tr\sigma \rangle_{\Omega}}{\langle tr\sigma \rangle_{\Omega_2}}| = |\frac{\overline{C}_{\kappa} - 1}{\overline{C}_{\kappa}}| \tag{2.18}$$

2. The deviation away from the mean for deviatoric stress:

$$\sqrt{\frac{(\langle \boldsymbol{\sigma}' \rangle_{\Omega_2} - \langle \boldsymbol{\sigma}' \rangle_{\Omega}) : (\langle \boldsymbol{\sigma}' \rangle_{\Omega_2} - \langle \boldsymbol{\sigma}' \rangle_{\Omega})}{\langle \boldsymbol{\sigma}' \rangle_{\Omega_2} : \langle \boldsymbol{\sigma}' \rangle_{\Omega_2}}} = |\frac{\overline{C}_{\mu} - 1}{\overline{C}_{\mu}}|, \tag{2.19}$$

3. The deviation of matrix material away from the mean for pressure:

$$|\frac{\langle tr\sigma \rangle_{\Omega_1} - \langle tr\sigma \rangle_{\Omega}}{\langle tr\sigma \rangle_{\Omega_1}}| = |\frac{\overline{\overline{C}}_{\kappa} - 1}{\overline{\overline{C}}_{\kappa}}| \tag{2.20}$$

4. The deviation of matrix material away from the mean for deviatoric stress:

$$\sqrt{\frac{(\langle \boldsymbol{\sigma}' \rangle_{\Omega_1} - \langle \boldsymbol{\sigma}' \rangle_{\Omega}) : (\langle \boldsymbol{\sigma}' \rangle_{\Omega_1} - \langle \boldsymbol{\sigma}' \rangle_{\Omega})}{\langle \boldsymbol{\sigma}' \rangle_{\Omega_1} : \langle \boldsymbol{\sigma}' \rangle_{\Omega_1}}} = |\frac{\overline{\overline{C}}_{\mu} - 1}{\overline{\overline{C}}_{\mu}}|. \tag{2.21}$$

In order to incorporate these deviations into the cost function, we introduce tolarance TOL_{μ} and TOL_{κ} as follows:

$$|\frac{\overline{C}_{\kappa} - 1}{\overline{C}_{\kappa}}| \leq TOL_{\kappa} \quad \text{and} \quad |\frac{\overline{C}_{\mu} - 1}{\overline{C}_{\mu}}| \leq TOL_{\mu}. \tag{2.22}$$

and

$$|\frac{\overline{\overline{C}}_\kappa - 1}{\overline{\overline{C}}_\kappa}| \leq TOL_\kappa \quad \textit{and} \quad |\frac{\overline{\overline{C}}_\mu - 1}{\overline{\overline{C}}_\mu}| \leq TOL_\mu. \tag{2.23}$$

If the normalized deviation exceeds the corresponding tolerance, then the level of violation is incorporated as a multilateral constraint to the macroscopic objectives.

The Objective Function

The objective is to minimize the following macro-micro objective function (cost function):

$$\boxed{\begin{aligned}\Pi = w_1|\frac{\kappa^*}{\kappa^{*,D}} - 1|^p + w_2|\frac{\mu^*}{\mu^{*,D}} - 1|^p + w_3|\frac{\sigma_e^*}{\sigma_e^{*,D}} - 1|^p \\ +\hat{w}_4\left(\left|\frac{\frac{\overline{C}_\kappa - 1}{\overline{C}_\kappa}}{TOL_\kappa}\right| - 1\right)^q + \hat{w}_5\left(\left|\frac{\frac{\overline{C}_\mu - 1}{\overline{C}_\mu}}{TOL_\mu}\right| - 1\right)^q \\ +\hat{w}_6\left(\left|\frac{\frac{\overline{\overline{C}}_\kappa - 1}{\overline{\overline{C}}_\kappa}}{TOL_\kappa}\right| - 1\right)^q + \hat{w}_7\left(\left|\frac{\frac{\overline{\overline{C}}_\mu - 1}{\overline{\overline{C}}_\mu}}{TOL_\mu}\right| - 1\right)^q\end{aligned}}$$

where

1. if $|\frac{\overline{C}_\kappa - 1}{\overline{C}_\kappa}| \leq TOL_\kappa$, then $\hat{w}_4 = 0$,
2. if $|\frac{\overline{C}_\kappa - 1}{\overline{C}_\kappa}| > TOL_\kappa$, then $\hat{w}_4 = w_4$,
3. if $|\frac{\overline{C}_\mu - 1}{\overline{C}_\mu}| \leq TOL_\mu$, then $\hat{w}_5 = 0$,
4. if $|\frac{\overline{C}_\mu - 1}{\overline{C}_\mu}| > TOL_\mu$, then $\hat{w}_5 = w_5$,
5. if $|\frac{\overline{\overline{C}}_\kappa - 1}{\overline{\overline{C}}_\kappa}| \leq TOL_\kappa$, then $\hat{w}_6 = 0$,
6. if $|\frac{\overline{\overline{C}}_\kappa - 1}{\overline{\overline{C}}_\kappa}| > TOL_\kappa$, then $\hat{w}_6 = w_6$,
7. if $|\frac{\overline{\overline{C}}_\mu - 1}{\overline{\overline{C}}_\mu}| \leq TOL_\mu$, then $\hat{w}_7 = 0$,
8. if $|\frac{\overline{\overline{C}}_\mu}{\overline{\overline{C}}_\mu} - 1| > TOL_\mu$, then $\hat{w}_7 = w_7$.

Here the design variables are $\mathbf{\Lambda} = \{\kappa_2, \mu_2\, \sigma_{2,e}, v_2\}$, and their constrained ranges are $\kappa_2^{(-)} \leq \kappa_2 \leq \kappa_2^{(+)}$, $\mu_2^{(-)} \leq \mu_2 \leq \mu_2^{(+)}$, $\sigma_{2,e}^{(-)} \leq \sigma_{2,e} \leq \sigma_{2,e}^{(+)}$ and $v_2^{(-)} \leq v_2 \leq v_2^{(+)}$. There are two characteristics of such a formulation which make the application of standard gradient type minimization schemes (e.g. Newton's method) inapplicable:

1. The incorporation of limits on the micro field behavior, as well as design search space restrictions, renders the objective function not continuously differentiable in design space.

2. The objective function is non-convex, which means that the system Hessian is not positive definite (invertible) throughout design space.

Therefore, the genetic algorithm could be a good option to be applied to this non-convex problem with the restrictions on the design parameters ($\mathbf{\Lambda} = \{\kappa_2, \mu_2\, \sigma_{2,e}, v_2\}$). The parameters used in the simulation are as shown in Table 2.1.

Symbol	Units	Value	Description
κ_1	GPa	80	Bulk modulus (Phase 1)
μ_1	GPa	30	Shear modulus (Phase 1)
$\sigma_{1,e}$	S/m	1.0×10^7	Electrical conductivity (Phase 1)
$\kappa^{*,D}$	GPa	111	Desired bulk modulus
$\mu^{*,D}$	GPa	47	Desired shear modulus
$\sigma_e^{*,D}$	S/m	2.0×10^7	Desired electrical conductivity
TOL_κ, TOL_μ	unitless	0.5	Physical tolerance
κ_2	GPa	$\kappa_1 = \kappa_2^{(-)} \leq \kappa_2 \leq \kappa_2^{(+)} = 10\kappa_1$	Bulk modulus range
μ_2	GPa	$\mu_1 = \mu_2^{(-)} \leq \mu_2 \leq \mu_2^{(+)} = 10\mu_1$	Shear modulus range
$\sigma_{2,e}$	S/m	$\sigma_{1,e} = \sigma_{2,e}^{(-)} \leq \sigma_{2,e} \leq \sigma_{2,e}^{(+)} = 10\sigma_{1,e}$	Electrical conductivity range
v_2	unitless	$0 = v_2^{(-)} \leq v_2 \leq v_2^{(+)} = \frac{2}{3}$	Volume fraction (Phase 2)
w_1, w_2, w_3	unitless	1	Cost function weights
w_4, w_5, w_6, w_7	unitless	0.5	Cost function weights
p, q	unitless	2	Cost function coefficients
θ	unitless	0.5	Hashin-Shtrikman bound combination ratio
TOL	unitless	10^{-6}	Simulation tolerance
-	unitless	100	The number of genetic strings per generation
-	unitless	2	The number of offspring strings per pairs

Table 2.1: Simulation Parameters

For the genetic algorithm, the **two cases** of the genetic algorithm were considered:

- **CASE 1:** *Keeping the top* 10 *parents* after each generation. (80 new genetic strings are created per generation.)

- **CASE 2:** *Not keeping top* 10 *parents* after each generation. (90 new genetic strings are created per generation.)

The plots of the best performing design's objective value for each generation for **CASE 1** and **CASE 2** are shown in Figures 2.9 and 2.10. Note that both axes are in log-log scale.

As we can see, the best performing cost function value decreases monotonically when the top parents are retained. That is because $\Pi(\mathbf{\Lambda}^{opt,I}) \geq \Pi(\mathbf{\Lambda}^{opt,I+1})$, where $\mathbf{\Lambda}^{opt,I+1}$ and $\mathbf{\Lambda}^{opt,I}$ are the best genetic strings from generations $I+1$ and I respectively. There is no such guarantee if the top parents are not retained. As in Figure 2.10, the case of *not keeping parents* shows an increase in the cost function sometimes.

While the non-retention of parents allows newer genetic strings to be evaluated in the next generation, numerous studies have shown that the benefits of parent retention outweigh this advantage when there are sufficiently large populations.

The table with the top 10 performing design values κ_2, μ_2 and v_2 for both cases are in Tables 2.2 and 2.3. Young's modulus and Poisson ratio corresponding to the effective bulk and shear modulus were calculated based on Equations 2.24, 2.25, and 2.26:

$$\kappa = \lambda + \frac{2}{3}\mu = \frac{E^y}{3(1-2\nu)} \tag{2.24}$$

$$\mu = \frac{E^y}{2(1+\nu)} \tag{2.25}$$

$$\frac{\kappa}{\mu} = \frac{2(1+\nu)}{3(1-2\nu)} \tag{2.26}$$

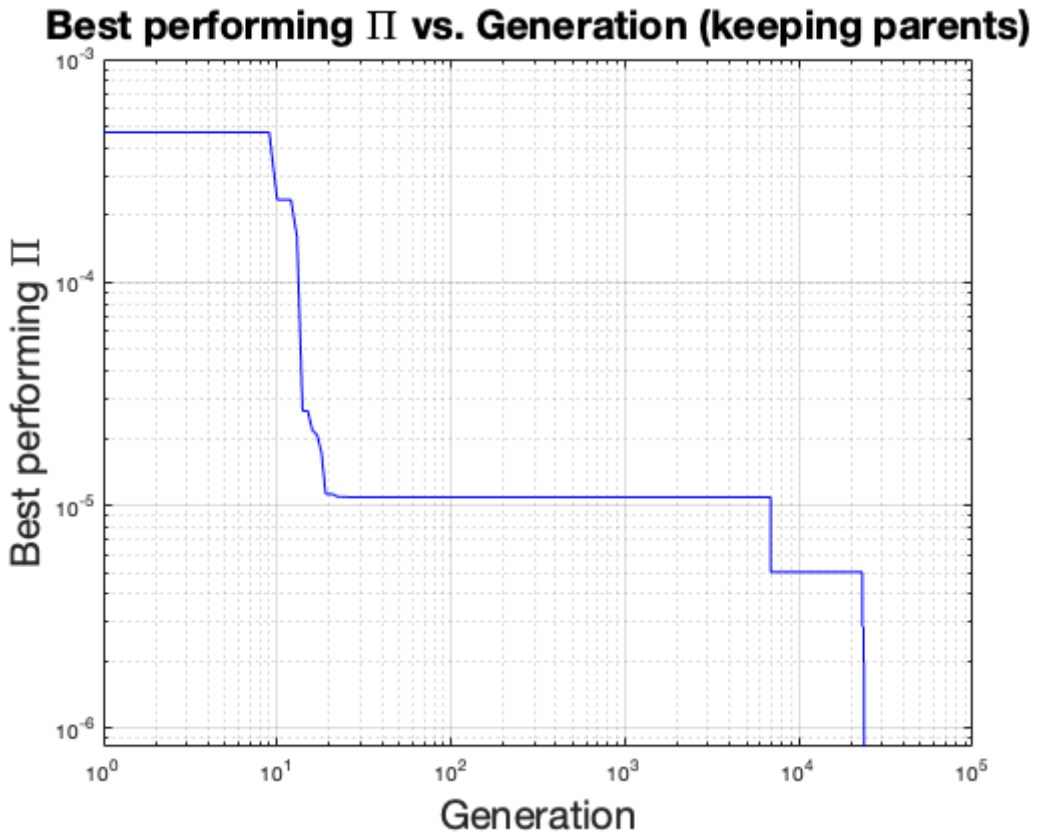


Figure 2.9: Best performing Π vs. Generation (**CASE 1**)

Π	κ_2 (Pa)	μ_2 (Pa)	σ_2 (S/m)	v_2	E (Pa)	ν
8.276E-07	3.369E+11	1.594E+11	8.344E+07	2.447E-01	9.161E+10	3.268E-01
1.958E-06	3.957E+11	1.848E+11	9.689E+07	2.183E-01	9.316E+10	3.260E-01
1.973E-06	3.347E+11	1.572E+11	8.212E+07	2.474E-01	9.148E+10	3.269E-01
2.846E-06	3.218E+11	1.537E+11	8.018E+07	2.525E-01	9.124E+10	3.269E-01
4.332E-06	3.013E+11	1.410E+11	7.415E+07	2.687E-01	9.045E+10	3.275E-01
4.492E-06	3.320E+11	1.568E+11	8.154E+07	2.488E-01	9.145E+10	3.269E-01
4.850E-06	3.380E+11	1.591E+11	8.394E+07	2.446E-01	9.160E+10	3.268E-01
5.023E-06	4.037E+11	1.877E+11	9.757E+07	2.169E-01	9.334E+10	3.259E-01
5.098E-06	3.272E+11	1.563E+11	8.120E+07	2.495E-01	9.141E+10	3.268E-01
5.245E-06	3.351E+11	1.575E+11	8.196E+07	2.475E-01	9.150E+10	3.269E-01

Table 2.2: Top 10 performing parameters (**CASE 1**)

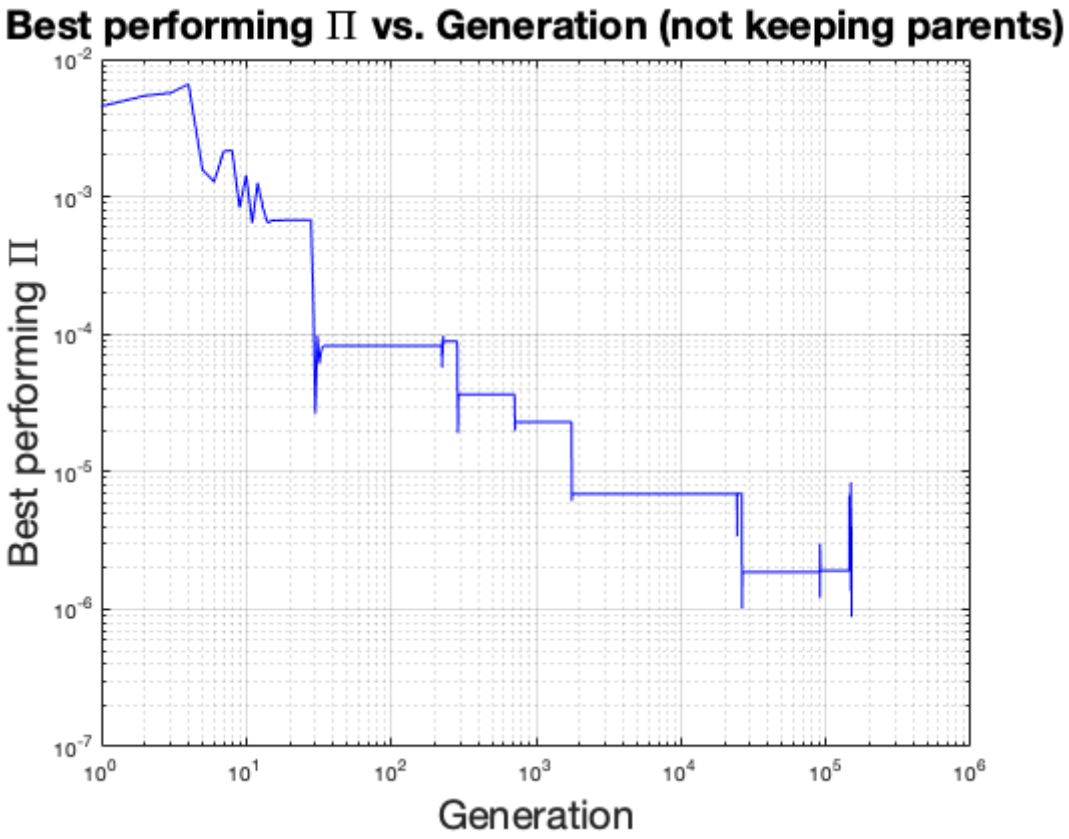


Figure 2.10: Best performing Π vs. Generation (**CASE 2**)

Π	κ_2 (Pa)	μ_2 (Pa)	σ_2 (S/m)	v_2	E (Pa)	ν
8.791E-07	3.442E+11	1.626E+11	8.515E+07	2.413E-01	1.423E+11	3.096E-01
4.034E-06	3.459E+11	1.636E+11	8.579E+07	2.404E-01	1.427E+11	3.094E-01
4.034E-06	3.459E+11	1.636E+11	8.579E+07	2.404E-01	1.427E+11	3.094E-01
7.701E-06	3.449E+11	1.638E+11	8.601E+07	2.401E-01	1.427E+11	3.093E-01
7.701E-06	3.449E+11	1.638E+11	8.601E+07	2.401E-01	1.427E+11	3.093E-01
1.357E-05	3.405E+11	1.628E+11	8.414E+07	2.431E-01	1.423E+11	3.091E-01
1.888E-05	3.403E+11	1.601E+11	8.401E+07	2.423E-01	1.415E+11	3.100E-01
2.308E-05	3.464E+11	1.633E+11	8.578E+07	2.414E-01	1.426E+11	3.096E-01
2.308E-05	3.464E+11	1.633E+11	8.578E+07	2.414E-01	1.426E+11	3.096E-01
3.167E-05	3.418E+11	1.609E+11	8.579E+07	2.417E-01	1.418E+11	3.099E-01

Table 2.3: Top 10 performing parameters (**CASE 2**)

2.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an optimization method in which we iteratively trying to improve the candidate optimal solution. Introduced by Kennedy, Eberhart, and Shi [80, 81], the PSO was biologically inspired by flocking of birds and schooling of fish. The PSO uses particles (corresponding to birds or fish) to find the optimal point of the objective function (either to minimize or to maximize). The PSO is a robust and fast-converging optimization method that is computationally efficient and easy to parallelize.[5] Even though the PSO is not guaranteed to find the global minimum, it still does a solid job in many optimization problems.

Equations 2.27 and 2.28 are the only two equations we need to know in order to implement the PSO. Descriptions of the variables in Equations 2.27 and 2.28 are shown in Table 2.4.

$$x_i^{(k+1)} = x_i^{(k)} + v_i^{(k+1)} \tag{2.27}$$

$$v_i^{(k+1)} = \underbrace{w v_i^{(k)}}_{\text{inertia}} + \underbrace{c_1 r_1 (p_i^{(k)} - x_i^{(k)})}_{\text{cognitive term}} + \underbrace{c_2 r_2 (p^{(k)} - x_i^{(k)})}_{\text{social term}} \tag{2.28}$$

Variable	Definition
$x_i^{(k)}$	Position of the i^{th} particle at k^{th} iteration
$v_i^{(k)}$	Velocity of the i^{th} particle at k^{th} iteration
w	Inertia weight
$p_i^{(k)}$	Best position of the i^{th} particle until k^{th} iteration
$p^{(k)}$	Best position of *all the particles (swarm)* until k^{th} iteration
c_1,c_2	Cognitive parameter and social parameter, respectively
r_1,r_2	Random numbers between 0 and 1
$f_i^{(k)}$	Objective function value at $x_i^{(k)}$
$f_i^{k,best}$	Objective function value at $p_i^{(k)}$
$f^{k,best}$	Objective function value at $p^{(k)}$

Table 2.4: Variables of particle swarm optimization

In the PSO, both $x_i^{(k)}$ and $v_i^{(k)}$ are vectors **having the length of the number of system parameters**. In Equation 2.28, the first term, *inertia*, represents how much the particle would like to keep the current velocity in the next iteration. The second term, *cognitive term*, represents how much the particle wants to consider the difference between

[5] A disadvantage of the PSO is that it can converge prematurely and be trapped into a local minimum, especially with complex problems [82].

its current position and **its own best position** it has ever taken, in calculating the velocity in the next iteration. The third term, *social term*, represents how much the particle wants to consider the difference between its current position and **the best position of all the particles (swarm)** obtained so far, in calculating the velocity in the next iteration.

The remarkable point of the PSO is that every single particle remembers its own *local best position* so far, while all the particles moving together as a *swarm* trying to find the *global best position*. Each particle then considers both the cognitively best direction and the socially best direction to determine the velocity in the next time step. This principle is well described in Equation 2.28.

The basic action of a single particle is shown in Figure 2.11. Also, the overall flowchart of the PSO could be described as shown in Algorithm 3. Note that, in Algorithm 3, the goal is to find the optimal parameter which **minimizes** the objective function. If the objective is to find the parameter which maximizes the objective function, one could simply replace the inequalities in **STEP 1**, $f_i^{(k)} < f_i^{k,best}$ and $f_i^{(k)} < f^{k,best}$, with $f_i^{(k)} > f_i^{k,best}$ and $f_i^{(k)} > f^{k,best}$, respectively.

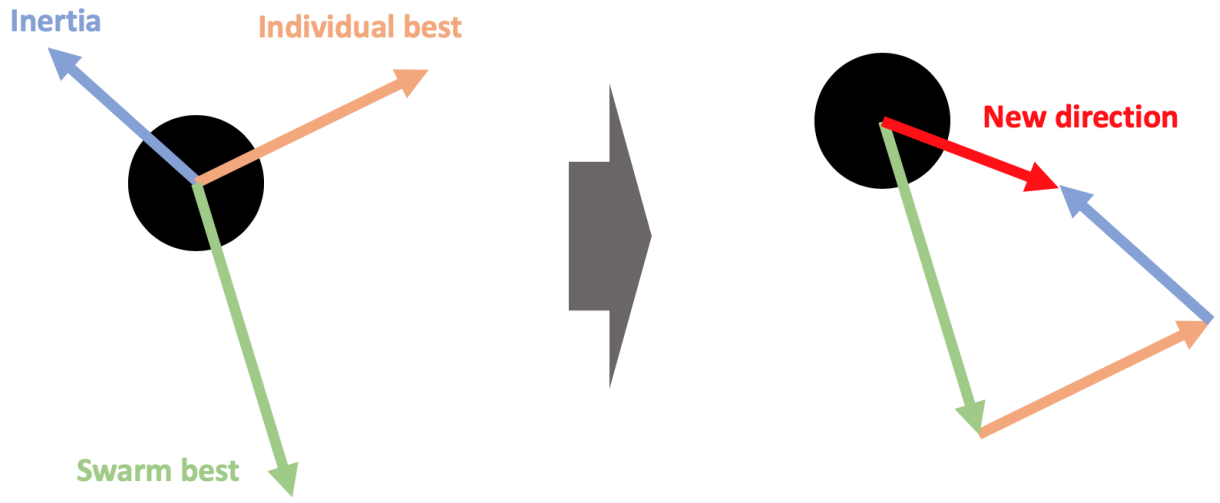


Figure 2.11: The basic action of a particle in particle swarm optimization

Algorithm 3: Particle Swarm Optimization

Initialization: Set the constants (N, $k_{\max}$, w, c_1, and c_2). Randomly initialize the initial positions ($x_i^{(0)}$) and velocities ($v_i^{(0)}$) of the particles ($i = 1, 2, ..., N$).

for *k=0 to* k_{max} **do**

STEP 1: Evaluate the objective function for each particle. Update the local/global best positions and corresponding **fitness** values.

for *i=1 to* N **do**

Evaluate $f_i^{(k)}$

if $f_i^{(k)} < f_i^{k,best}$ **then**

$p_i^{(k)} \leftarrow x_i^{(k)}$

$f_i^{k,best} \leftarrow f_i^{(k)}$

else

pass

end

if $f_i^{(k)} < f^{k,best}$ **then**

$p^{(k)} \leftarrow x_i^{(k)}$

$f^{k,best} \leftarrow f_i^{(k)}$

else

pass

end

end

STEP 2: Update the velocities and positions of the particles for the next iteration.

for *i=1 to* N **do**

$v_i^{(k+1)} = wv_i^{(k)} + c_1 r_1 (p_i^{(k)} - x_i^{(k)}) + c_2 r_2 (p^{(k)} - x_i^{(k)})$

$x_i^{(k+1)} = x_i^{(k)} + v_i^{(k+1)}$

end

STEP 3: Record the global best position and corresponding objective function value of the current k^{th} iteration, $p^{(k)}$ and $f^{k,best}$, respectively.

end

As in the genetic algorithm, one could employ gradient-based methods afterward if the neighbor of the obtained optimal point is smooth enough. Also, if there are upper or lower bound constraints in the searching domain, one could repair the updated positions of the particles to satisfy the bounds.

The Implementation of Particle Swarm Optimization

The following example illustrates the simple implementation of particle swarm optimization to find the global minimum in the 2D non-convex function. In this case, I used the following configuration.

- The number of particles: 100
- The maximum number of iterations: 200
- Inertia weight (w): 0.85
- Cognitive parameter (c_1): 1
- Social parameter (c_2): 2

Also, the objective function we want to minimize is as follows.

$$f(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 2(x - 5x^3 - 5y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}, \tag{2.29}$$

in the range of $-3 \leq x \leq 3$ and $-3 \leq y \leq 3$.

Additionally, one could add a *penalty score* to the objective function if there are any inequality constraints which the objective function is subject to. For example, if there is an inequality constraint $(x-2)^3 - y + 1 \leq 0$ for this problem, the final objective function will look like the following.

$$f(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 2(x - 5x^3 - 5y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2} + \text{Penalty}, \tag{2.30}$$

where

$$\text{Penalty} = \begin{cases} 10^6 & \text{if } (x-2)^3 - y + 1 > 0 \\ 0 & \text{if } (x-2)^3 - y + 1 \leq 0 \end{cases} \tag{2.31}$$

For simplicity, I implemented the particle swarm optimization with the objective function in Equation 2.29 (without any inequality constraint), not in Equation 2.30. The plot of the objective function is shown in Figures 2.12 and 2.13.

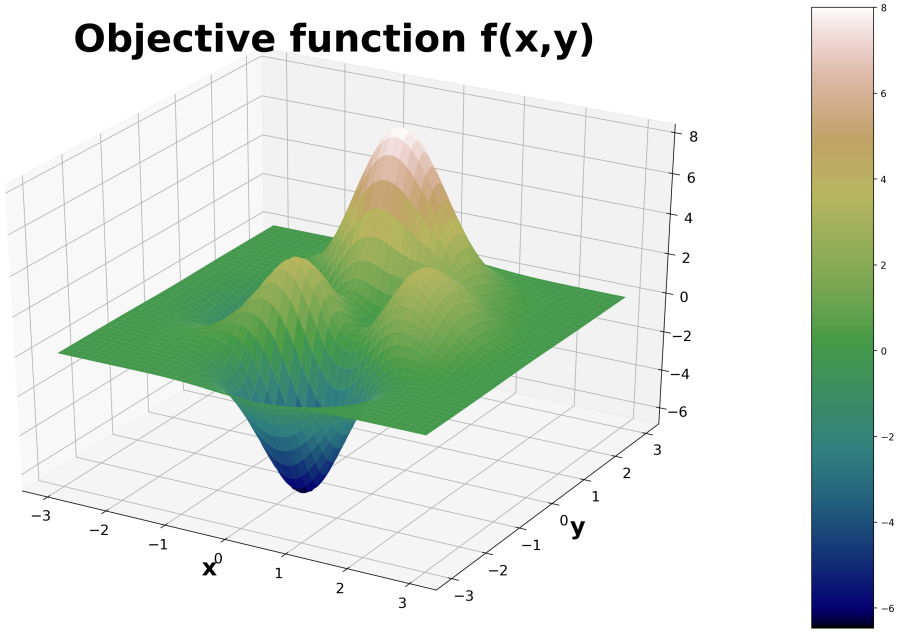


Figure 2.12: The objective function (3D view)

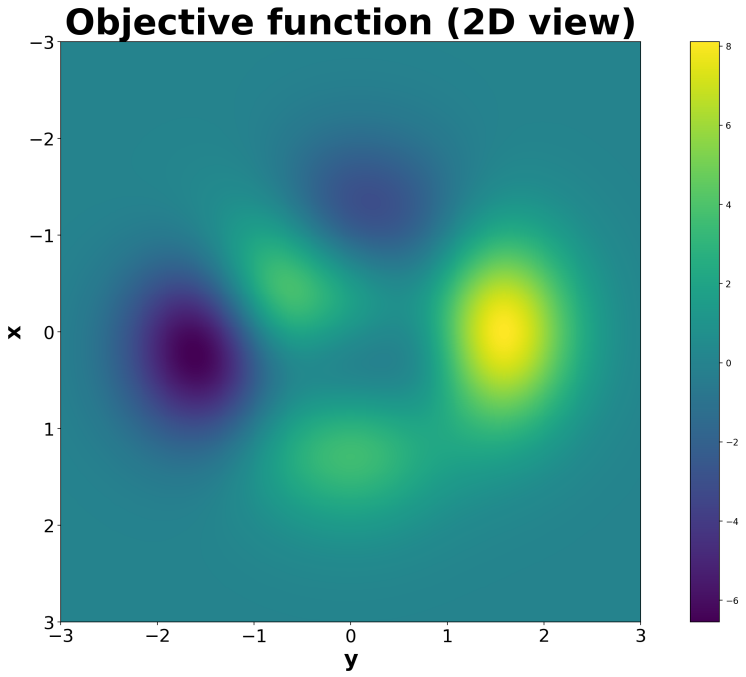


Figure 2.13: The objective function (2D view)

After performing the particle swarm optimization, we could observe that the best performing objective function value monotonically decreases by iterations, as shown in Figure 2.14.

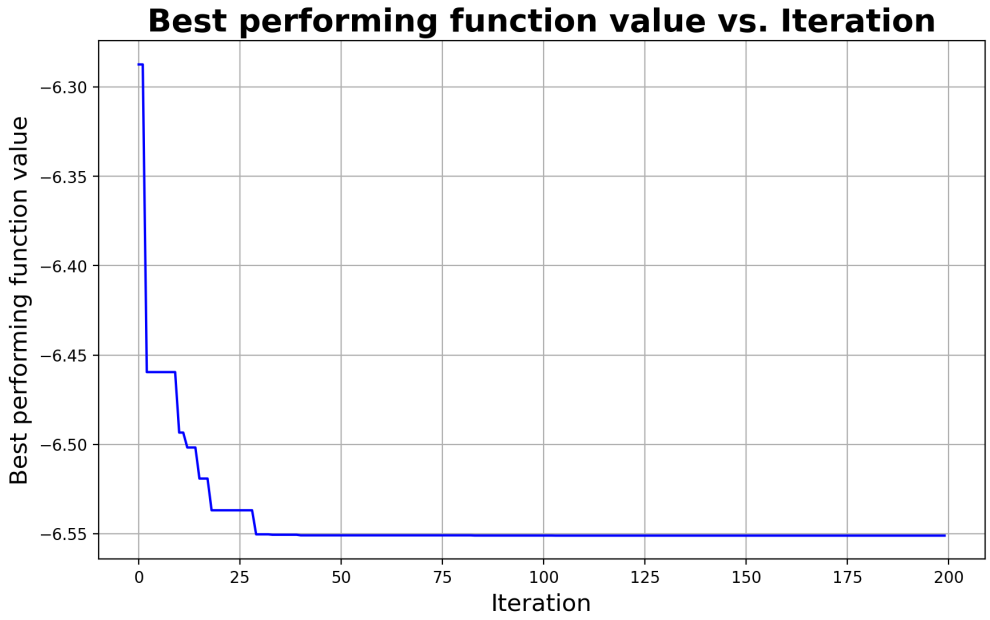


Figure 2.14: Best performing function value vs. Iteration

Also, the positions of the particles before and after the particle swarm optimization are shown in Figures 2.15 and 2.16, respectively. The red dots represent the particles. We could observe that the particles converged on the location where we expected as a global minimum from Figure 2.12. By performing the particle swarm optimization, it also discovered that the *global best position* is $(\mathbf{x}, \mathbf{y}) = (\mathbf{0.228}, \mathbf{-1.625})$, having the objective function value of $\mathbf{-6.551}$.

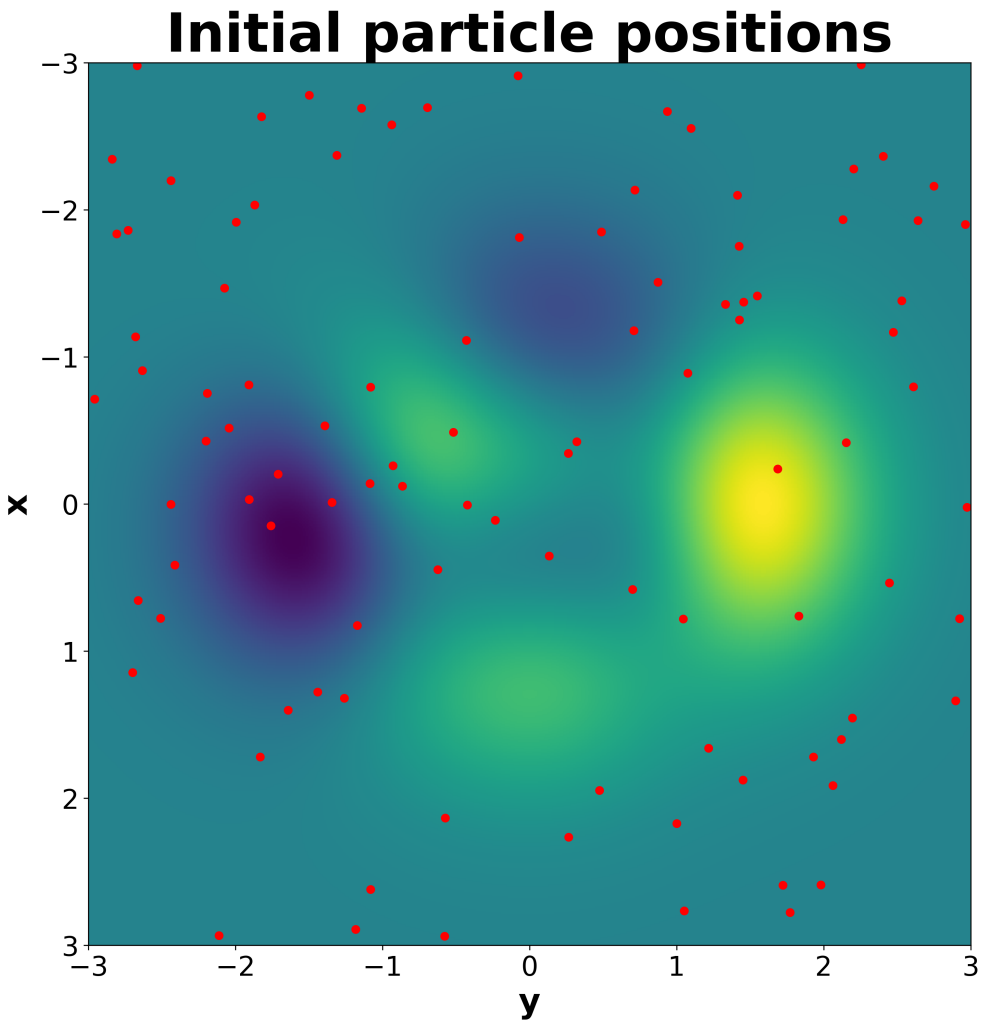


Figure 2.15: Initial positions of the particles

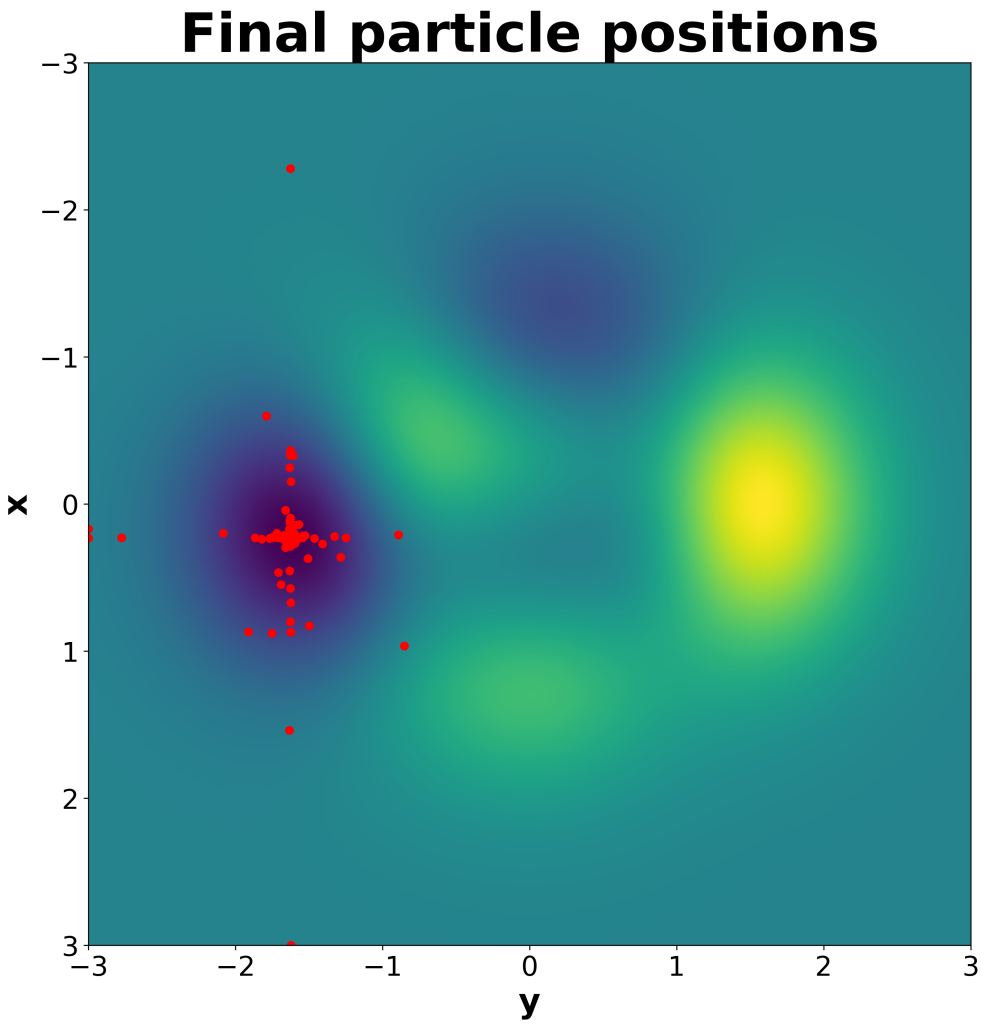


Figure 2.16: Final positions of the particles

2.5 The Overall Outlook and Further Algorithms

We have reviewed three optimization algorithms for solving non-convex optimization problems, *Bayesian Optimization*, *Genetic Algorithms*, and *Particle Swarm Optimization*. They are powerful algorithms that are widely used in a variety of disciplines including computer science, engineering, economics, and business.

Finding the global optimum of a function is usually difficult. That is because analytical methods are not applicable in general, and the numerical approaches often lead to very hard challenges (e.g. ensuring that the *investigated* optimal point is the *true global* optimum). Trying to overcome this limitation, many scientists and engineers have proposed a variety of

optimization algorithms.

Mean-Variance Mapping Optimization (MVMO) is based on the particle swarm optimization principles, but it uses a continuously updated mean and variance of best solutions [83–85]. However, particle swarm optimization or the genetic algorithm sometimes converges more accurately than the MVMO.

Graduated Optimization is an optimization technique that attempts to solve a difficult optimization problem by solving a simplified version of the problem. It then progressively transforms the problem into the complex version of the problem while optimizing, until it becomes equivalent to the given difficult optimization problem [86–88].

The *Ant Colony Optimization Algorithm (ACO)* is a biologically inspired stochastic optimization method for solving problems that can be transformed into finding good paths through graphs [89].

Some people even try to innovate the structures of the existing optimization methods. For example, Dao, et al. [90, 91] provided an innovative framework for designing an effective genetic algorithm structure that can enhance the performance of the genetic algorithm while searching for the global optimal solution.[6]

There are many other algorithms for solving non-convex optimization algorithms including the Artificial Bee Colony Algorithm [93], Cuckoo Search [94], Greedy Algorithms [95], Simulated Annealing [96], and Tabu Search [97, 98]. However, the choice of the 'optimal' optimization algorithm is highly problem dependent. For the readers who are interested in further optimization algorithms, please refer to [99–105].

[6]Dao, the first author of [90, 91], provides good tutorials on many optimization techniques in his blog. Refer to [92] for more detail.

Chapter 3

Modeling, Simulation and Machine Learning for Rapid Process Control of Multiphase Flowing Foods

3.1 Abstract

The contents of this chapter are from the journal paper, "Modeling, simulation and machine learning for rapid process control of multiphase flowing foods" by Kim, D.H., Zohdi, T.I., and Singh, R.P., which was accepted by *Computer Methods in Applied Mechanics and Engineering (VSI: Artificial Intelligence)*.

Across many modern industries, as technologies have matured, the use of more complex processes involving multiphase materials has increased. In the food industry, multiphase fluids are now relatively wide-spread, in particular, because of the desire to have faster throughput for large-scale food production. In many cases involving transport, such materials consist of a fluidized binder material with embedded particles. As one increases the volume fraction of particles, a corresponding increase in effective overall viscosity occurs. Often, during the process, the material must be heated, for example, to ensure food safety, induce pasteurization, sterilization, etc. *For real-time control, this requires rapidly computable models to guide thermal processing, for example by applied electrical induction.* In the present analysis, models are developed for the required heating field (electrically induced) and pressure gradient needed in a pipe to heat a multiphase material to a target temperature and to transport the material with a prescribed flow rate.

3.2 Introduction

Across many industries, new types of particle-laden materials are being developed and utilized. In the development of such materials, the basic philosophy is to select material combinations to produce desired aggregate responses upon deposition onto a substrate or into a

mold. Oftentimes, such materials start in a fluidized form comprised of particles in a solvent or fluidized binder, forming a viscous slurry. However, because of the increasing demands for faster throughput and industrial-scale production of complex particle-laden materials, the determination of accurate pumping pressures is critical to move such fluids through delivery piping systems (Figure 3.1).

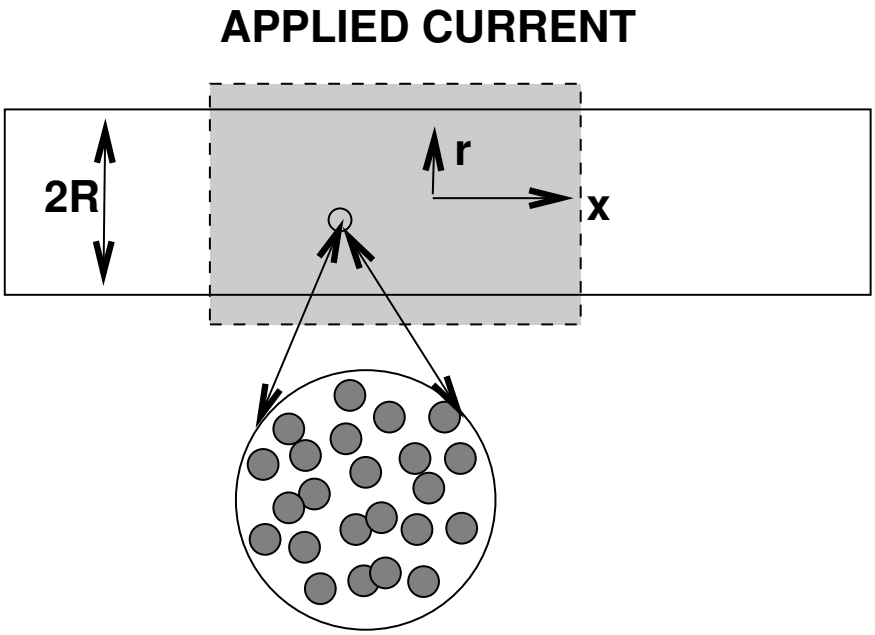


Figure 3.1: Flow of a particle-laden fluid through a pipe in the presence of an applied current (heating)

There have been monumental leaps in manufacturing technologies across many industries. These technologies have the potential to drastically improve the precision in food processing efficiency, food quality, and safety [106]. Some approaches are methodical and systematic, while some are ad-hoc and haphazard. The purpose of this paper is to explore modeling and simulation themes associated with multiphase fluid flow and thermal food processing. Many foods and beverage manufacturers control their continuous fluid processes with PID algorithms, using data from downstream sensors to adjust flow and heating parameters, for example in pasteurization or sterilization processes. As food and beverage manufacturing looks to optimize production efficiencies and energy productivity, machine learning, and other data science tools offer a chance to improve precision and predictive capabilities for real-time process optimization. In particular, new foods, such as plant-based meats, such as Beyond Meat [107] and Impossible Burger [108] present new opportunities and challenges. However, for such procedures to be successful, rapidly computable models are needed to drive these technologies.

3.3 Technological Approaches

The main objective of this work is to develop a relatively simple model for the pressure gradient needed in a pipe and the reduction of a food channel width by fouling, while particle-laden fluids moving through the channel, as a function of (1) the volume fraction of added particles, (2) the pipe radius, (3) the volumetric flow rate, (4) the fluid-induced intensity of the shear stress at the pipe wall and (5) the multiphase fluid viscosity. Things overlaid on this are the induced thermal fields and associated thermal dependency of the materials in the system. This type of physical system has become increasingly more important to the 3D printing industry as well, which is attempting to rapidly print complex electrical inks ("e-inks") of multiphase extruded materials, where the embedded particles endow the cured printed materials with overall (mechanical, electrical, thermal, magnetic, etc.) properties that the pure solvent (particle-free ink) alone does not possess. *This paper intends to adapt and to do further this analysis for food production methods.*

An overall objective of the analysis is to develop semi-analytical expressions that can help guide analysts who are designing manufacturing systems involving fluidized particle-laden foods. Theoretically speaking, one could attempt a large-scale CFD analysis, however, for accurate direct numerical simulation of particle-laden continua, the spatial discretization grids must be extremely fine, with several thousand numerical unknowns needed per particle length-scale. Furthermore, extremely fine time-discretization is required. Thus, for even a small system with several hundred-thousand particles, a proper discretization would require several billion numerical unknowns (see, for example, [109–113]). Although such simulations are possible in high-performance computing centers, their usefulness for rapid daily design analysis for real-time food processing and related processes is minimal. This is even more critical if the models are used to drive real-time control. *Therefore, in this paper we seek to develop simplified approaches.* This work presents reduced-order calculations to predict the radius change of the food channel by fouling, and pressure required to pump a suspension of rigid particles in a fluid through a pipe, under the assumption that the flow is uni-directional and fully developed. Heat is also applied as a function of an externally-applied electric field. It first arrives at a modification of Poiseulle flow through a pipe. The analysis assumes the suspension can be treated as a homogeneous fluid with an effective viscosity μ^*. This is a simplification, in order to develop useful and practical results, without having to resort to overly computationally-intensive numerical methods which seek results from detailed accounting of the micro-scale hydrodynamic interactions between particles in a suspension.

3.4 Fluid through a Pipe of Radius R

We consider fluid that is flowing through an idealized pipe with a circular cross-section of area $A = \pi R^2$.

$$v = v_{max}\left(1 - \left(\frac{r}{R}\right)^q\right), \tag{3.1}$$

where q is now considered a variable. For fully developed laminar flow, $q = 2$, while for increasing q one characterizes progressively turbulent flow ($q \geq 2$). Also, assuming that the overall flow rate is assumed constant Q_o, one can show that

$$v_{max} = \frac{Q_o(q+2)}{\pi R^2 q}, \tag{3.2}$$

and

$$\tau_w = \frac{\mu^* Q_o(q+2)}{\pi R^3}, \tag{3.3}$$

where μ^* represents effective viscosity of multiphase fluid.

We have the following observations: (a) Increasing μ^* or Q_o increases the stress at the wall (τ_w) and (b) Decreasing R increases the stress at the wall (τ_w).

Also, by performing a force balance in the positive x-direction, we obtain

$$-\frac{\Delta P}{\Delta x} = -\frac{\partial P}{\partial x} = \underbrace{\frac{2\mu^*(q+2)}{\pi R^4}}_{C} Q_o \stackrel{def}{=} CQ_o. \tag{3.4}$$

This expression allows us to correlate the pressure applied to a volume of particle-laden to allow it to move as a constant flow rate. If we fix the flow rate Q_o, the multiplier C identifies the pressure gradient needed to achieve a flow rate Q_o.

As the Reynolds number increases, the velocity profile will change from a quadratic ($q = 2$) to a more blunted profile ($q > 2$). The effect of a changing profile is described by representing q as follows.

$$q = \frac{1}{2}\left((\gamma^* + c_2) + \sqrt{(\gamma^* + c_2)^2 + 8\gamma^*}\right), \tag{3.5}$$

where

$$\gamma^* = \frac{2c_1 Q_o \rho^*}{\pi R \mu^*}. \tag{3.6}$$

ρ^* represents effective density of multiphase fluid, respectively. c_1 and c_2 are constants where $0 \leq c_1 << 1$ and $c_2 \approx 2$. For laminar flow ($q = 2$), $c_1 = 0$ and $c_2 = 2$. See [114] for more detail. In the remaining analysis, we will consider turbulent flow ($q > 2$) of the particle-laden fluid, in which the Reynolds number is greater than 4000 [115]. Also, we will assume that the particles are not elongated and that they are well distributed within the base fluid.

Comments on the Turbulent Flow

This subsection describes the reason why the flow we are considering is turbulent flow. We could write the Reynolds number of the multiphase flow through a pipe as follows:

$$Re = \frac{2\rho^* u_{avg} R}{\mu^*} = \frac{2\rho^* R}{\mu^*} \frac{Q_o}{\pi R^2} = \frac{2\rho^* Q_o}{\pi R \mu^*} \tag{3.7}$$

Referring to Equations 3.14 and 3.34 which will be explained further in the following sections, we could write:

$$Re = \frac{2\rho^* Q_o(1 - 2.5v_p)}{\pi R \mu_f} = \frac{2(v_p\rho_p + (1 - v_p)\rho_f)(1 - 2.5v_p)Q_o}{\pi R \mu_f} \tag{3.8}$$

The descriptions of variables in Equation 3.8 are in Table 3.1. From Equation 3.8, we could consider the minimum Reynolds number it can have.

1. Maximum R (biggest when no fouling): 0.1 (m)
2. Maximum viscosity of the fluid μ_f (the maximum is from the initial time when the flow temperature is the lowest): 0.005 $(Pa - S)$
3. Minimum flow rate (Q_o): 0.008 (m^3/s)
4. Minimum of the particle density (ρ_p) : 3000 (kg/m^3)
5. Accordingly,
   $min((v_p\rho_p + (1 - v_p)\rho_f)(1 - 2.5v_p)) = (3000v_p + 2000(1 - v_p))(1 - 2.5v_p)$
6. Minimum of $(3000v_p + 2000(1 - v_p))(1 - 2.5v_p)$ is when $v_p = 0.3$ $(0.05 \leq v_p \leq 0.3)$.

Therefore, we could observe that the minimum Reynolds number of the multiphase flow is:

$$Re_{min} = \frac{2 \cdot (3000 \cdot 0.3 + 2000 \cdot 0.7) \cdot (1 - 2.5 \cdot 0.3) \cdot 0.008}{\pi \cdot 0.1 \cdot 0.005} = 5856.9 > 4000 \tag{3.9}$$

Since it is higher than 4000 [115], it is turbulent flow.

3.5 Induced Thermal Fields via Joule Heating

The heating process in food and beverage production facilities is typically accomplished with steam-heated heat exchangers. Most commonly, low-pressure steam heats a plate and frame or tubular heat exchanger. For thermally sensitive food products, typical industry practice is to heat food in long tubular heat exchangers using flowing hot water, heated via direct steam

injection, maintained at a single static temperature. Both steam-heated (single loop) and hot water heated (double-loop) heating processes are notoriously slow to reach processing temperature or adapt the temperature to changing processing conditions. In particular, industrial-scale double-loop systems can take up to 10 minutes to create a response in end-target heating. Steam-powered heat exchange, in either single loop or double loop systems, is optimized for production in large facilities where processing conditions remain static and unchanging. These systems are optimized for traditional manufacturing operations and do not have the capabilities needed for the future. As this manufacturing sector moves towards shorter production runs and expanding the number of products produced in a single production line, there is an opportunity to find new production efficiencies and increase energy production to boost yields and facility operational effective efficiency (OEE). Joule heating (also known as Ohmic heating) generates heat by passing an electric current through food which has electrical resistance [116]. Heat is generated rapidly and uniformly in the liquid matrix as well as in particulates, producing a higher quality sterile and aseptic product [117]. This heating method is best for foods containing particulates in a weak salt-containing medium due to their high resistance [118]. Joule heating adapts well to computer control and allows for instantaneous control, even though it is currently limited by the sensitivity and response time of a downstream thermocouple. It is for this reason that we select induction heating as a model process. From the first law of thermodynamics, we have the following description of Watts per unit volume:

$$\rho^* C^* \dot{\theta} = H - S \tag{3.10}$$

where θ is the temperature, C^* is specific heat capacity, H represents heating and S represents sinks. For the heating

$$H = a\frac{J^2}{\sigma^*}, \tag{3.11}$$

where a is the absorption coefficient, and S represents all of the losses (conductive, convective, radiative, refrigeration). σ^* is the effective electrical conductivity, and J is the electric current for Joule heating. Discretizing and solving yields

$$\theta(t+\Delta t) = \theta(t) + \frac{\Delta t}{\rho^* C^*}(H - S) = \theta(t) + \frac{\Delta t}{\rho^* C^*}(a\frac{J^2}{\sigma^*} - S(t)). \tag{3.12}$$

If at a given time t, one can solve for the necessary $J(t)$ when $\theta(t+\Delta t) = \theta^*$:

$$J(t) = \sqrt{\frac{\sigma^*}{a}\left(\rho^* C^* \frac{\theta^* - \theta(t)}{\Delta t} + S(t)\right)}. \tag{3.13}$$

3.6 Models for Effective Properties of Particle-Laden Fluids

A key component of the analysis requires the characterization of the effective properties of a particle-laden fluid as a function of the volume fraction of particles and the baseline (interstitial) fluid properties. The density of the particle-laden fluid is actually an "effective density" since it actually is a mixture of materials (particles and interstitial fluid). Effective properties are defined through volume averages. For example, the effective density of the mixture is

$$\rho^* \overset{def}{=} \langle \rho(\boldsymbol{x}) \rangle_V \overset{def}{=} \frac{1}{V} \int_V \rho(\boldsymbol{x})\, dV = \frac{1}{V} \left(\int_{V_f} \rho_f\, dV + \int_{V_p} \rho_p\, dV \right) = \nu_f \rho_f + \nu_p \rho_p \tag{3.14}$$

where ν_f and ν_p are the volume fractions of the fluid and particles, respectively. The volume fractions have to sum to unity: $\nu_f + \nu_p = 1 \Rightarrow \nu_f = 1 - \nu_p$. Similar approaches can be used to calculate various types of properties, such as effective viscosity. However, to calculate it is somewhat more complicated since it requires one to estimate the interaction between the constituents. There are a number of models that provide expressions for the effective viscosity of the fluid containing particles. One of the first models for the effective viscosity of such fluids was developed in 1906 by [119]. It reads as

$$\mu^* = \mu_f (1 + 2.5\nu_p), \tag{3.15}$$

where μ^* is the effective viscosity, μ_f is the viscosity of the fluid and ν_p is the volume fraction of particles. This expression is accurate only for low volume fractions of particles. A more accurate approximation, *in fact, a strict, rigorous, lower bound* can be derived from the well-known Hashin and Shtrikman bounds [77–79] in solid mechanics. Specifically, for linearized elasticity applications, for isotropic materials with isotropic effective (mechanical) responses, the Hashin-Shtrikman bounds (for a two-phase material) are as follows for the effective bulk modulus (κ^*)

$$\kappa^{*,-} \overset{def}{=} \kappa_1 + \frac{\nu_2}{\frac{1}{\kappa_2 - \kappa_1} + \frac{3(1-\nu_2)}{3\kappa_1 + 4\mu_1}} \leq \kappa^* \leq \kappa_2 + \frac{1-\nu_2}{\frac{1}{\kappa_1 - \kappa_2} + \frac{3\nu_2}{3\kappa_2 + 4\mu_2}} \overset{def}{=} \kappa^{*,+} \tag{3.16}$$

and for the effective shear modulus (G^*)

$$G^{*,-} \overset{def}{=} G_1 + \frac{\nu_2}{\frac{1}{G_2 - G_1} + \frac{6(1-\nu_2)(\kappa_1 + 2G_1)}{5G_1(3\kappa_1 + 4G_1)}} \leq G^* \leq G_2 + \frac{(1-\nu_2)}{\frac{1}{G_1 - G_2} + \frac{6\nu_2(\kappa_2 + 2G_2)}{5G_2(3\kappa_2 + 4G_2)}} \overset{def}{=} G^{*,+}, \tag{3.17}$$

where κ_1 (usually the matrix material) and κ_2 (usually the particulate material) are the bulk moduli and G_1 and G_2 are the shear moduli of the respective phases ($\kappa_2 \geq \kappa_1$

and $G_2 \geq G_1$), and where ν_2 is the second phase volume fraction. Such bounds are the tightest possible on isotropic effective responses, with isotropic two-phase microstructures, where only the volume fractions and phase contrasts of the constituents are known (see [79] for a discussion on the optimality of such bounds). Note that no geometric or statistical information is required for the bounds. For an authoritative review of the general theory of random heterogeneous media see [120]. One can take the limit of the particle phase becoming rigid, i.e. the bulk and shear moduli tending towards infinity, $\kappa_2 = \kappa_p \to \infty$ and $G_2 = \mu_p \to \infty$, signifying that the particles are much stiffer than the interstitial fluid, while simultaneously specifying that the interstitial fluid is incompressible, i.e. $\kappa_1/G_1 = \kappa_f/\mu_f \to \infty$ with G_1 being finite. This yields,

$$\mu^* \geq \mu^{*,-} = \mu_f \left(1 + 2.5\frac{\nu_p}{1-\nu_p}\right). \tag{3.18}$$

Equation 3.18 represents the tightest known lower bound on the effective viscosity of a two-phase material comprised of rigid particles in a surrounding incompressible fluid. The bound recaptures the Einstein result in the $\nu_p \to 0$ limit, but is a rigorous lower bound at significant ν_p. *This rigorous lower bound* is extremely accurate up to approximately 20 % volume fraction, which is sufficient for most applications of interest. These bounds have been tested in the numerical analysis literature repeatedly, for example against direct Finite Element calculations found in [113]. We refer the reader to [121] for a more in-depth analysis of the effective viscosity of particle-laden fluids. Refer to [122] for the analysis of the proper application of the non-interaction and the "dilute limit" approximations, and for detailed discussions on the isotropic and anisotropic viscosity of suspensions containing particles of diverse shapes and orientations. It is important to emphasize that [121] is accurate for up to 25-30 % in the case of spherical particles. Furthermore, [121] covers other shapes, including, importantly, mixtures of diverse shapes. Of course, one can employ formulas such as in [121] for more accuracy, however, because the Hashin-Strikman expression is a strict lower bound, $\mu^{*,-} \leq \mu^*$, we consequently generate a strict lower bound for the pressure gradient

$$-\frac{\partial P}{\partial x} \geq \underbrace{\frac{2\mu^{*,-}(q+2)}{\pi R^4}}_{C^-} Q_o \stackrel{def}{=} C^- Q_o. \tag{3.19}$$

Comments on the Volume Fraction

We assume that the multiphase flow we simulate has particle volume fraction which is lower than or equal to 30%. That is because the objective of this work is to develop a relatively simple model for multiphase food processing through a channel. This simple model enables us to perform the rapid computation to efficiently control the thermal processing of foods. More accurate and delicate numerical methods could be applied when the particle volume fraction is more than 30%.

3.7 Approximate Effective Thermal Properties

For illustration purposes, in this model problem, a relatively simplistic definition of the heat capacity is defined through the stored energy at a point, $w = C(\theta - \theta_o)$, where θ_o is a reference temperature, for example, $\theta_o = 0$ degrees Kelvin. In this case, the effective heat capacity for a small body is given by

$$\begin{aligned} \langle w \rangle_V = \langle C\theta \rangle_V &= C^* \langle \theta \rangle_V \stackrel{def}{=} \frac{1}{V} \int_V C(\boldsymbol{x}) \theta(\boldsymbol{x}) \, dV \\ &= \frac{1}{V} \left(\int_{V_f} C_f \theta(\boldsymbol{x}) \, dV + \int_{V_p} C_p \theta(\boldsymbol{x}) \, dV \right) \\ &\approx (\nu_f C_f + \nu_p C_p)\theta \Rightarrow C^* = (\nu_f C_f + \nu_p C_p), \end{aligned} \tag{3.20}$$

provided θ is uniform in the (small body).

The effective density of a mixture for two-phase materials can directly be determined by

$$m = \rho^* V = (\nu_f \rho_f + \nu_p \rho_p) V, \tag{3.21}$$

, while the effective thermal mass is $mC = \rho^* C^* V$.

Thermal Material Behavior

For illustration purposes, in this model problem, the primary properties which change by temperature are viscosity, electrical conductivity, and thermal conductivity. This could be expressed as follows.

$$\mu_f = \mu_{fo} e^{-s_1 \frac{\theta - \theta_o}{\theta_o}}, \tag{3.22}$$

and

$$\sigma_f = \sigma_{fo} e^{-s_2 \frac{\theta - \theta_o}{\theta_o}}, \tag{3.23}$$

and

$$\sigma_p = \sigma_{po} e^{-s_3 \frac{\theta - \theta_o}{\theta_o}}, \tag{3.24}$$

and

$$k_f = k_{fo} e^{-s_4 \frac{\theta - \theta_o}{\theta_o}}, \tag{3.25}$$

and

$$k_p = k_{po} e^{-s_5 \frac{\theta - \theta_o}{\theta_o}}, \tag{3.26}$$

where μ, σ, and k represents viscosity, electrical conductivity, and thermal conductivity, respectively. Subscript f represents fluid carrying particles, and subscript p represents particles. s_1, s_2, s_3, s_4, and s_5 represents thermal softening parameters.

3.8 A Fouling Model

Fouling is a cost-increasing problem for a variety of industries, including aerospace, petrochemicals and especially food [123–126]. The reduction of a food channel width, caused by deposition of thermally modified proteins and other food components, is a major issue in food processing since cleaning or removal of such deposits is crucial for quality and safety issues [127]. Also, fouling of heat exchangers (undesirable depositions on channel surfaces) is directly related to the quality of processed foods and processing cost, because it may require more energy consumption for the pressure gradients and Joule heating. In this section, we adopted the most common fouling model of a heat exchanger by Ebert and Panchal [128], to predict the deposition thickness of the food channel, as shown in equation [3.27]:

$$\frac{dR_f}{dt} = \underbrace{c_1 Re^{-0.66} Pr^{-0.33} e^{-\frac{E_a}{RT_f}}}_{deposition} - \underbrace{c_2 \tau_w}_{suppression} \quad , \tag{3.27}$$

where R_f is the fouling resistance (m^2K/W), Re is the Reynolds number, Pr is the Prandtl number, E_a is the activation energy (J/mol), T_f is the film temperature (K) and τ_w is the shear at the deposit surface (Pa). c_1 and c_2 are fouling parameters determined by regression of experimental data. For flow in a pipe or tube, the Reynolds number is generally defined as $Re = \frac{\rho u D}{\mu}$, where ρ is the density of the fluid, u is the mean velocity of the fluid, D is the hydraulic diameter of the pipe and μ is the dynamic viscosity of the fluid. Also, the Prandtl number is defined as $Pr = \frac{C_p \mu}{k}$, where C_p is the specific heat capacity, μ is the dynamic viscosity of the fluid an k is the thermal conductivity. From fouling resistance obtained by equation [3.27], we can get the deposition thickness, as shown in equation [3.28]

$$d_f = R_f \lambda_f, \tag{3.28}$$

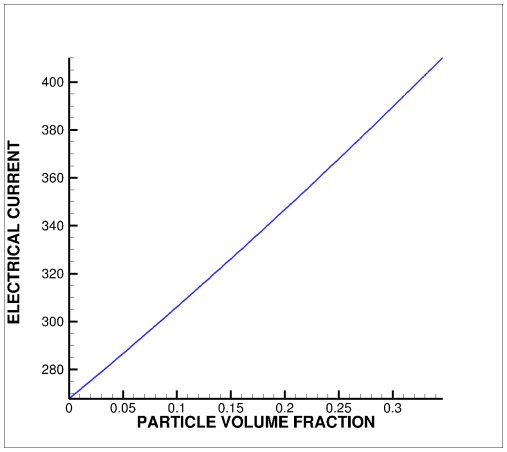
where d_f is the average thickness of the fouling (m), and λ_f is the thermal conductivity of the fouling (W/mK).

3.9 Numerical Experiments

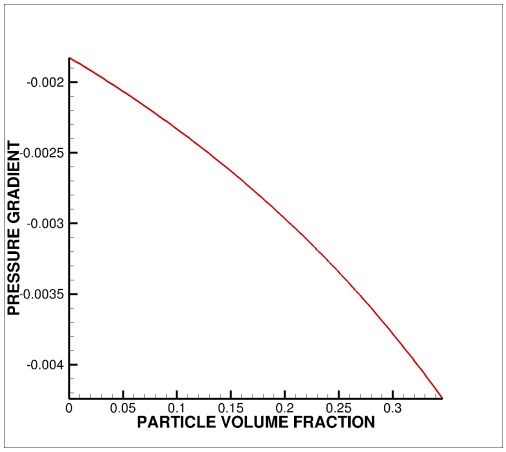
The main object of this section is to show how the pressure gradients, the electric current needed to heat the multiphase food, and the Reynolds number are dependent on the particle volume fraction in the multiphase fluid. We plotted the pressure gradient as a function of of ν_p, with the following parameters: (a) viscosity, $\mu_f = 0.01\,Pa - s$,[1] (b) fluid density:

[1]For reference, the viscosity of water is $\mu_f = 0.001\,Pa - s$ and for honey, $\mu_f = 1\,Pa - s$.

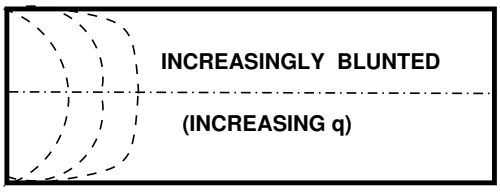
$\rho_f = 1000\,kg/m^3$, (c) particle density: $\rho_p = 2000\,kg/m^3$, (d) flow rate: $Q_o = 0.00001\,m^3/s$, (e) thermal sensitivity: $s_1 = s_2 = s_3 = 1$, and (f) pipe radius: $R = 0.01\,m$. The goal was to raise the temperature from 300 K to 400 K in 1 second. The plots are shown in Figure 3.2. The pressure gradient steadily increases with particle volume fraction. Due to the increase in the particle volume fraction, the viscosity increases, thus the Reynolds number decreases (already quite small). The point of this example was not to illustrate all the encompassing parameter set, but simply to show the explicit dependency of the pressure gradient on the presence of secondary particles.



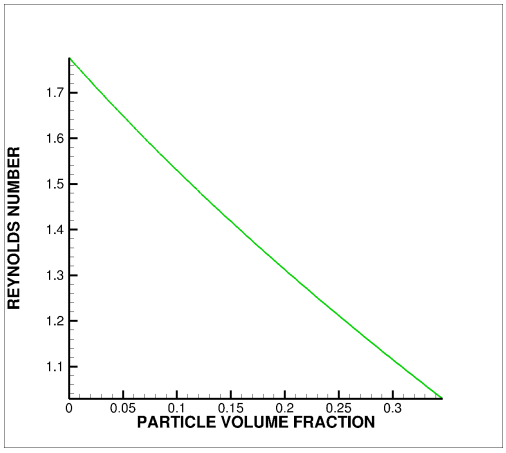
(a) The electric field needed for the volume fraction of particles



(b) The pressure gradient needed ($\frac{\Delta P}{\Delta x}$) as a function of volume fractions of ν_p



(c) Progressive blunting of the velocity profile with increasing q



(d) The resulting Reynolds number

Figure 3.2: The electric current, the pressure gradient, and the Reynolds number by the particle volume fraction

3.10 A Simulation Algorithm

The objective of the simulation is to analyze

- the thermal behavior of food while processing,
- pressure gradient needed in a pipe to heat a multiphase material,
- the change of the radius (caused by fouling) of food channel by time.

First, we consider the electric current for heating the multiphase material, as a function of the time as follows, having a factor α which represents the response rate

$$J = J_0(1 - e^{-\alpha t})sin(2\pi t). \tag{3.29}$$

We obtain the temperature of the next time step (equation 3.12) using a forward Euler scheme.[2] The thermal loss term S can account for any kind of thermal loss. In this example, we assume that the dominant mode of thermal loss to the environment is convection and that the temperature is essentially uniform within a cross-section of the pipe. Convective losses are often modeled by:

$$S = hA(\theta(t) - \theta_a),$$

where h is the convection coefficient, A is the area over which conduction occurs, $\theta(t)$ is the temperature of the object being modeled at a given time, t and θ_a is the temperature of the ambient environment. Since all of our other terms are per unit volume of the slurry, we must alter this relationship to have compatible units. We assume that we are considering an arbitrary length Δx of the pipe. The surface area A will be $2\pi R\Delta x$, and the total volume will be $\pi R^2\Delta x$. Dividing A by volume yields $2/R$. As such, the convective losses per unit volume will be:

$$S = 2h(\theta(t) - \theta_a)/R. \tag{3.30}$$

Note that $S(t)$ models the transfer of heat *out of the system*, so a positive value of S represents a loss of energy. As temperature changes, primary material properties of the mixture change(σ^*, k^* and μ^*), because those properties of fluid and particle change (equation 3.22, 3.23, 3.24, 3.25, and 3.26).

The Hashin-Shtrikman bounds for σ^* (electrical conductivity) are as follows, and σ_f and σ_p were calculated based on Equations 3.23, and 3.24

$$\underbrace{\sigma_f + \frac{v_p}{\frac{1}{\sigma_p - \sigma_f} + \frac{1-v_p}{3\sigma_f}}}_{\sigma^{*,-}} \leq \sigma^* \leq \underbrace{\sigma_p + \frac{1 - v_p}{\frac{1}{\sigma_f - \sigma_p} + \frac{v_p}{3\sigma_p}}}_{\sigma^{*,+}}, \tag{3.31}$$

[2]For the readers who are interested in further numerical methods for solving ordinary differential equations, refer to [129–132]. MATH 228A from the Department of Mathematics at UC Berkeley is also helpful.

Similarly, the Hashin-Shtrikman bounds for k^* (thermal conductivity) are as follows, and k_f and k_p were calculated based on Equations 3.25, and 3.26

$$\underbrace{k_f + \frac{v_p}{\frac{1}{k_p - k_f} + \frac{1-v_p}{3k_f}}}_{k^{*,-}} \leq k^* \leq \underbrace{k_p + \frac{1-v_p}{\frac{1}{k_f - k_p} + \frac{v_p}{3k_p}}}_{k^{*,+}}, \tag{3.32}$$

In the simulation, we take the average of the lower bound and upper bound to approximate σ^* and k^*

$$\sigma^* = \frac{\sigma^{*,-} + \sigma^{*,+}}{2}, \text{ and } k^*. = \frac{k^{*,-} + k^{*,+}}{2} \tag{3.33}$$

Also, widely used effective viscosity estimate is that of Oliver&Ward [133] which is in much better agreement (than Equation 3.18) with experimental data up to $v_p = 0.30$ (see [121, 122] for extensive reviews). It reads as

$$\mu^* \approx \ \mu^{*e} = \frac{\mu_f}{1 - 2.5v_p}. \tag{3.34}$$

We could observe that the ratio of the rigorous lower bound to the Oliver&Ward [133] estimate is always less than unity for finite volume fraction [134]. In the remaining analysis, we will employ Oliver&Ward estimate for the effective viscosity of multiphase flow.

Updating σ^* affects integration of the next time step for obtaining the temperature, and updating μ^* and k^* causes the change of channel radius (Equation 3.27-3.28) and the pressure gradient (Equation 3.4), because they are function of μ^* and k^*. Based on this, we can obtain the new channel radius for the next step, which will be used for the calculation of the temperature, pressure gradient of the next time step. This iteration is done until the temperature becomes greater than or equal to the desired temperature of the food product. The overall flowchart is shown in Figure 3.3, and the simulation parameters are shown in Table 3.1.

3.11 The Genetic Algorithm for Optimization

A genetic algorithm (GA) is a type of non-derivative search and optimization tool, which works differently from classical search and gradient-optimization methods. Because of its broad applicability, ease of use and global perspective, the GA has been increasingly applied to various search and optimization problems in the recent past [135, 136]. Our final objective is to find the optimal parameters we could control for food processing using the GA. In order to do that, we should set the cost function we want to minimize to find a way to process food in a reasonable and efficient way. In this case, we try to minimize the fouling of the food channel after processing (because it gets larger and larger as the food flows), as well as

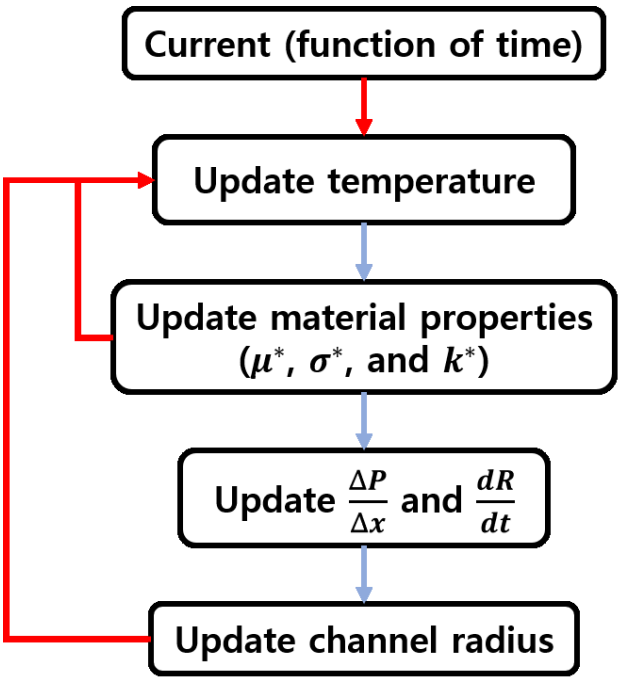


Figure 3.3: The overall flowchart of the simulation

minimize the processing time and the total energy consumption for the process. We applied a penalty if either the food processing time or energy consumption exceeded a preset limit.

The cost function we want to minimize is:

$$\Pi = w_1 |\eta|^2 + \hat{w}_2 \left| \frac{T - T_{limit}}{T_{limit}} \right|^2 + \hat{w}_3 \left| \frac{E_{final} - E_{limit}}{E_{limit}} \right|^2$$

where

1. T: Total processing time,
2. T_{limit}: Limit of the total processing time,
3. E_{final}: Energy consumption for food processing,
4. E_{limit}: Limit of the total energy consumption for processing,
5. if $T \leq T_{limit}$, then $\hat{w}_2 = 0$,
6. if $T > T_{limit}$, then $\hat{w}_2 = w_2$,
7. if $E_{final} \leq E_{limit}$, then $\hat{w}_3 = 0$,

8. if $E_{final} > E_{limit}$, then $\hat{w}_3 = w_3$.

Also, η represents the fouling rate of the channel, as shown in Equation 3.35, and E_{final} is calculated by numerically integrating the power by time during processing time.

$$\eta = \frac{R_0 - R}{R_0} \tag{3.35}$$

Here the design variables we want to optimize are $\mathbf{\Lambda} = \{\alpha, v_p, \sigma_{p0}, \rho_p, C_p, k_{p0}\}$, and their constrained ranges are $\alpha^{(-)} \leq \alpha \leq \alpha^{(+)}$, $v_p^{(-)} \leq v_p \leq v_p^{(+)}$, $\sigma_{p0}^{(-)} \leq \sigma_{p0} \leq \sigma_{p0}^{(+)}$, $\rho_p^{(-)} \leq \rho_p \leq \rho_p^{(+)}$, $C_p^{(-)} \leq C_p \leq C_p^{(+)}$ and $k_{p0}^{(-)} \leq k_{p0} \leq k_{p0}^{(+)}$.

As a consequence of the character of the objective function, we can use the following genetic algorithm:

Algorithm 4: Genetic Algorithm

Initialization: Randomly generate a population of S initial strings, $\mathbf{\Lambda}^i$ $(i = 1, 2, 3, ..., S)$. $\longrightarrow \Lambda^i \triangleq \{\lambda_1^i, \lambda_2^i, \lambda_3^i, ..., \lambda_N^i\}$

while *min*$(\Pi) \geq$ *Tolerance* **do**

STEP 1: Compute the *performance* of each genetic string $\Pi(\mathbf{\Lambda}^i)$ $(i = 1, 2, 3, ..., S)$, and rank those strings based on the *performance* values.

STEP 2: Mate the best performing P parent strings to generate C offspring strings. $\mathbf{\Lambda}^{NEW} \triangleq \mathbf{\Phi} \odot \mathbf{\Lambda}^{OLD_1} + (\mathbf{1} - \mathbf{\Phi}) \odot \mathbf{\Lambda}^{OLD_2}$
, where $\mathbf{\Phi} = \{\phi_1, \phi_2, \phi_3, ..., \phi_N\}$, and $0 \leq \phi_k \leq 1$ $(k = 1, 2, ..., N)$.
$\odot$ represents component-wise multiplication.

STEP 3: Replace the worst performing C strings in the old genetic strings with the new child strings obtained in **STEP 2**.

STEP 4:
if *Keeping parents* **then**
Keep the old P parent strings
Generate $S - P - C$ new strings
else
Remove the old P parent strings
Generate $S - C$ new strings
end

end

(Optional) Employ gradient-based methods afterward in the local minima, if the neighbor of the obtained optimal point is smooth enough.

We remark that the definition of **fitness** of a genetic string in this algorithm indicates the value of the objective function. In other words, the most fit genetic string is simply the one with the smallest objective function. STEPS, which are associated with the genetic part of the overall algorithm, attempt to collect multiple local minima.[3]

By observing Figure 3.4 one sees that if the objective functions are highly nonconvex, there exists a strong possibility that the inferior offspring will replace superior parents. Therefore, retaining the top parents is not only less computationally expensive, since these designs do not have to be reevaluated, but it is also theoretically superior. The minimization of the cost function is guaranteed to be monotone, if the top parents are retained, i.e. $\Pi(\mathbf{\Lambda}^{opt,I}) \geq \Pi(\mathbf{\Lambda}^{opt,I+1})$, where $\mathbf{\Lambda}^{opt,I+1}$ and $\mathbf{\Lambda}^{opt,I}$ are the best genetic strings from generations $I+1$ and I respectively.

Symbol	Units	Value	Description
μ_{f0}	Pa s	.005	Reference temp. viscosity of fluid (Phase 1)
σ_{f0}	$^{-1}$ m^{-1}	0.617	Reference temp. electrical conductivity of fluid (Phase 1)
ρ_f	kg/m^3	2000	Density of fluid (Phase 1)
k_{f0}	W/(m K)	0.45	Reference temp. heat conductivity of fluid (Phase 1)
C_f	J/(kg K)	1600	Specific heat capacity of fluid (Phase 1)
R_0	m	1e1	Initial channel radius
Q_o	m^3/s	8e3, 10e3, 12e3	Flow rate
J_0	A/m^2	2e4	Base electric current
h	W/(m^2 K)	10	Convective heat transfer coefficient
c_1	(m^2 K)/(W s)	5e3	Fouling parameter
c_2	(m^2 K)/(W Pa s)	8e8	Fouling parameter
θ_0	K	300	Initial slurry temperature
θ_a	K	300	Ambient temperature
Δt	s	0.001	Time step size
a	Unitless	0.8	Absorption coefficient
T_{limit}	s	2.5	Limit of the total processing time
E_{limit}	MJ/m^3	600	Limit of the total energy consumption for processing
$s_1,\ s_2,\ s_3$	Unitless	0.1, 0.12, 0.2	Thermal softening parameters
$s_4,\ s_5$	Unitless	0.01, 0.02	Thermal softening parameters
ϕ	Unitless	0.5	Hashin–Shtrikman bound combination weight
α	unitless	$1 = \alpha^{(-)} \leq \alpha \leq \alpha^{(+)} = 10$	Current response rate
v_p	unitless	$0.05 = v_p^{(-)} \leq v_p \leq v_p^{(+)} = 0.3$	Volume fraction (Phase 2)
σ_{p0}	S/m	$0.1 = \sigma_{p0}^{(-)} \leq \sigma_{p0} \leq \sigma_{p0}^{(+)} = 2.5$	Reference temp. electrical conductivity of particles (Phase 2)
ρ_p	kg/m^3	$3000 = \rho_p^{(-)} \leq \rho_p \leq \rho_p^{(+)} = 9000$	Density of particles (Phase 2)
C_p	$J/(kg \cdot K)$	$1000 = C_p^{(-)} \leq C_p \leq C_p^{(+)} = 6000$	Specific heat capacity of particles (Phase 2)
k_{p0}	$W/(m \cdot K)$	$0.11 = k_{p0}^{(-)} \leq k_{p0} \leq k_{p0}^{(+)} = 0.52$	Reference temp. heat conductivity of particles (Phase 2)
$w_1,\ w_2,\ w_3$	unitless	1, 2, 3	Cost function weights
N	unitless	100	The number of genetic strings per generation
-	unitless	2	The number of offspring strings per pairs

Table 3.1: Simulation Parameters

[3] If one selects the 'mating' parameters in $\mathbf{\Phi}$ to be greater that one and/or less than zero, one can induce 'mutations' to have characteristics that neither parent possesses.

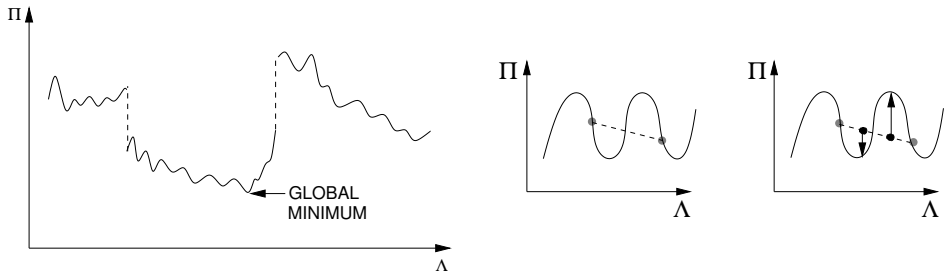
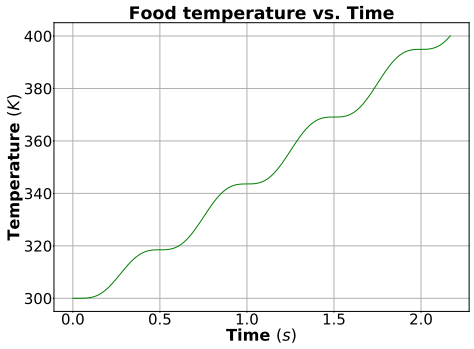


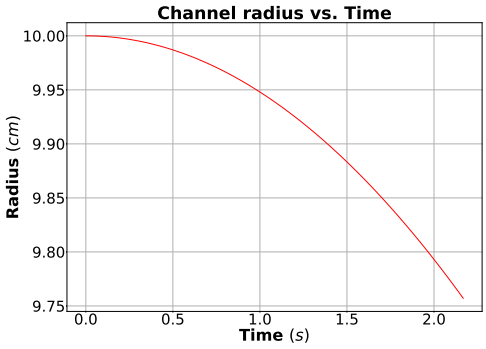
Figure 3.4: LEFT: A characterization of the class of objective functions of interest. RIGHT: A loss of superior older genetic strings if the top parents are not retained.

3.12 Simulation Results

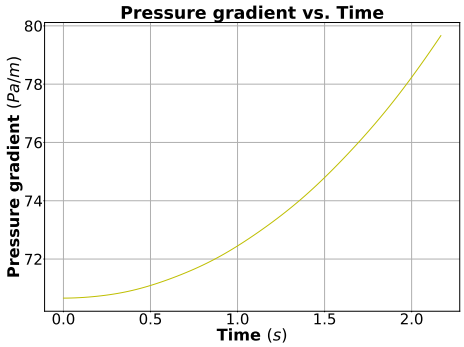
We have developed a relatively simple model for multiphase food processing through a channel and optimized the parameters for processing with numerical simulations and the genetic algorithm. Specifically, with the parameters which were optimized using the genetic algorithm, we could obtain (a) the thermal behavior of multiphase food while processing, (b) the pressure gradient needed in a pipe to heat the multiphase material, (c) the change of the radius (by deposition fouling) of food channel by time and (d) the power needed for processing by time. The plots are shown in Figures 3.4-3.6. Figure 3.4 represents the case when $Q_o = 0.008\,(m^3/s)$, Figure 3.5 represents the case when $Q_o = 0.010\,(m^3/s)$, and Figure 3.6 represents the case when $Q_o = 0.012\,(m^3/s)$.



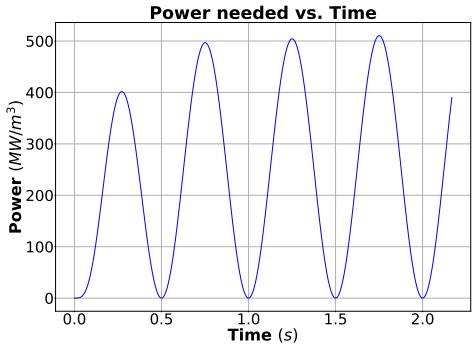
(a) Thermal behavior of the multiphase food



(b) Radius change of channel by time

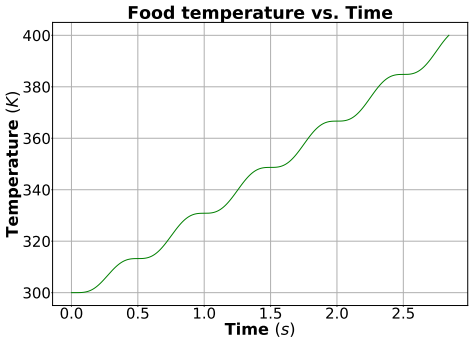


(c) Pressure gradient needed for heating

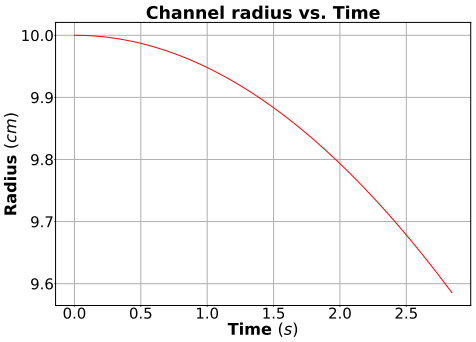


(d) Power needed for processing by time

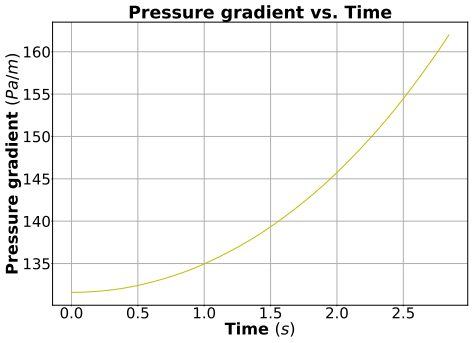
Figure 3.4: CASE 1 ($Q_o = 0.008\,(m^3/s)$)



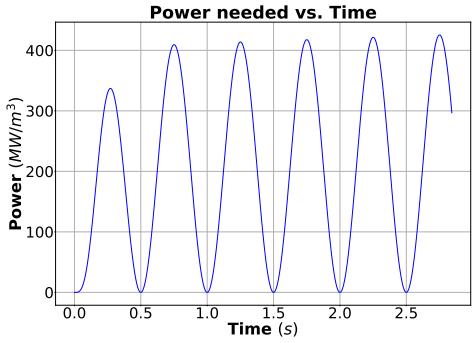
(a) Thermal behavior of the multiphase food



(b) Radius change of channel by time

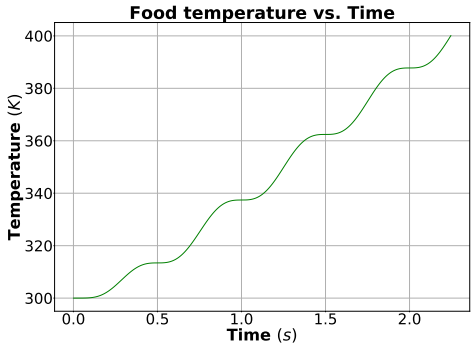


(c) Pressure gradient needed for heating

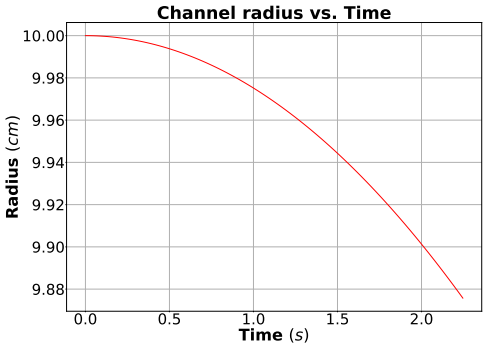


(d) Power needed for processing by time

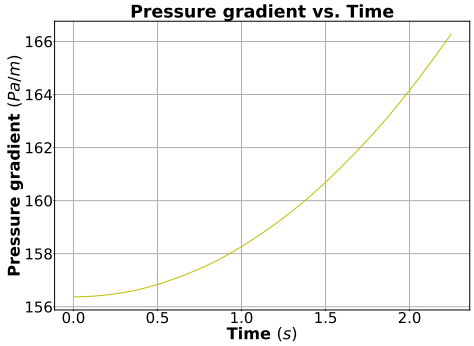
Figure 3.5: CASE 2 ($Q_o = 0.010\,(m^3/s)$)



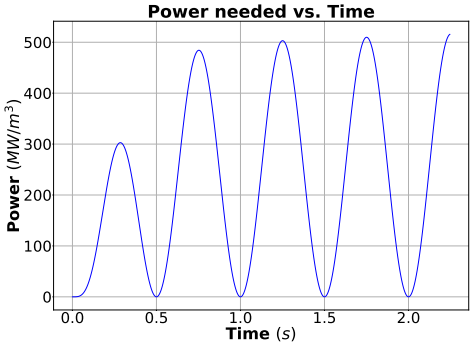
(a) Thermal behavior of the multiphase food



(b) Radius change of channel by time



(c) Pressure gradient needed for heating



(d) Power needed for processing by time

Figure 3.6: CASE 3 ($Q_o = 0.012\,(m^3/s)$)

3.13 Prediction of Deposition Fouling on the Channel

The unintended accumulation of the food on a channel wall while processing can cause deposition fouling. The fouling on the food delivery channel is a critical issue for processing, because it directly affects manufacturing costs [123–125]. It becomes harder to process the food precisely in a desirable way, as channel radius gets smaller and smaller. Therefore, predicting the possible changes in the radius of the food channel is important to control food quality. We applied a machine learning technique to predict the fouling rate of the food channel after final food processing time to get to the desired food temperature.

To obtain the channel fouling rate for combinations of the parameters, we still need to solve differential equations using numerical methods, as we already did in the genetic algorithm of the previous section. It might be computationally expensive when we use elaborate numerical methods (Discrete element methods, Finite element methods, etc.) and/or there are a large number of parameter sets to simulate. Even though numerical simulations are

computationally expensive in general, it gives us the data that is useful for generating a predictive model.

Therefore, we applied a machine learning technique to predict the fouling rate of the channel at the final processing time with reduced computation time and desirable accuracy, when specific parameter sets are chosen while the other operation conditions (the initial channel radius, the electric current, the ambient temperature) are fixed.

Machine learning (ML) is a subdivision of artificial intelligence, which allows computers to learn from the past data so that it could detect patterns and make predictions from noisy and complex data sets [32, 137–140]. The ML approach deals with the design of algorithms to learn from machine-readable data and make predictions on future unknown data [141]. Also, there has been some research on generating prediction models using machine learning to solve a variety of engineering problems such as material design, computer vision, pattern recognition, and spam filtering [142–148], including those in computational mechanics [149–151].

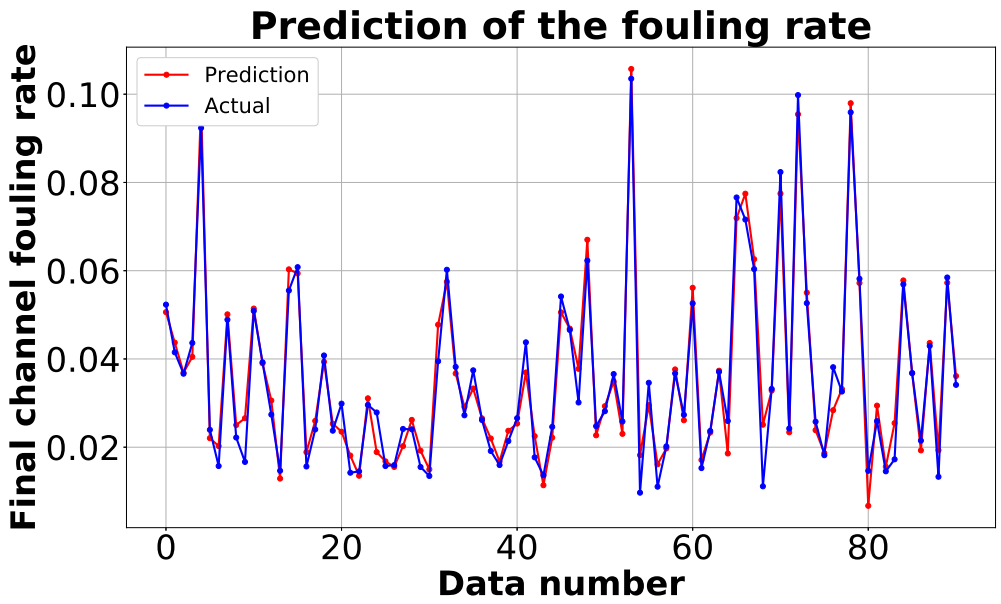
In our approach, we used a fully connected neural network to develop the model for predicting the final fouling rate of the channel. The neural network is a biologically-inspired machine learning model which enables nonlinear learning process based on how neurons communicate and learn in living things. We used 9010 parameter combinations for the total data. All the parameter combinations (features) and corresponding fouling rates (labels) were normalized to have the range between 0 and 1, before training the neural network. We implemented our machine learning model with PyTorch (1.1.0 version), which is an efficient machine learning framework for Python and competent in both usability and speed [49]. We used batch normalization, which is one of the ways to accelerate neural network training, by reducing the internal covariance shift, allowing us to use much higher learning rates [20]. Xavier initialization was used for weight initialization in order to obtain substantially faster convergence [152]. Also, an Adam optimizer was used for the optimization, which is computationally efficient and has little memory requirements [153]. The configuration of the Machine Learning model is as follows.

- Learning rate: 0.001
- Training epochs: 100
- Batch size: 50
- The number of hidden layers: 2
- The number of nodes in hidden layers: 40, 20
- Activation function: ReLu (Rectified Linear Unit)
- Weight initialization: Xavier uniform
- Loss function: Mean Squared Error

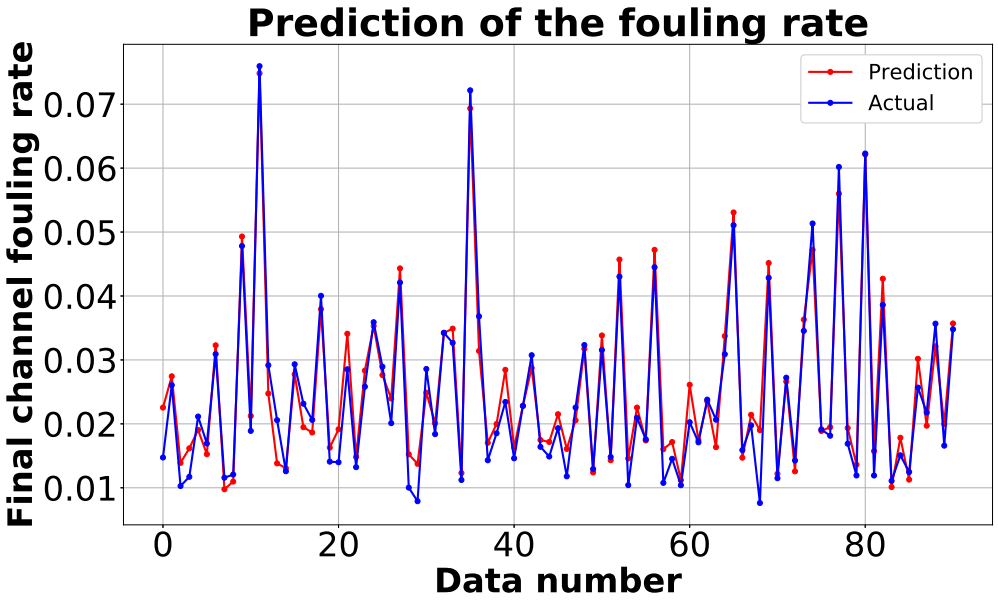
- Optimization Method: Adam optimizer
- Division of the data: 20% for training, 79% for validation, and 1% for testing

With the trained model, we were able to desirably predict the fouling rate of the channel (Equation 3.35) for the test data set, even though we trained our model with a small portion of the total data (20% for training, 79% for validating, and 1% for testing), with considerable accuracy, as shown in Figure 3.6. There are three cases with varying flow rates (Figure 3.6). **Data number** represents the combinations of $\mathbf{\Lambda} = \{\alpha, v_p, \sigma_{p0}, \rho_p, C_p, k_{p0}\}$ of the test data. The parameter combinations of the test data represented by **Data number** is shown in Table 3.4 - 3.6. Note that the test data is brand new data for the trained model, which means they were completely isolated from the training data set and training process.

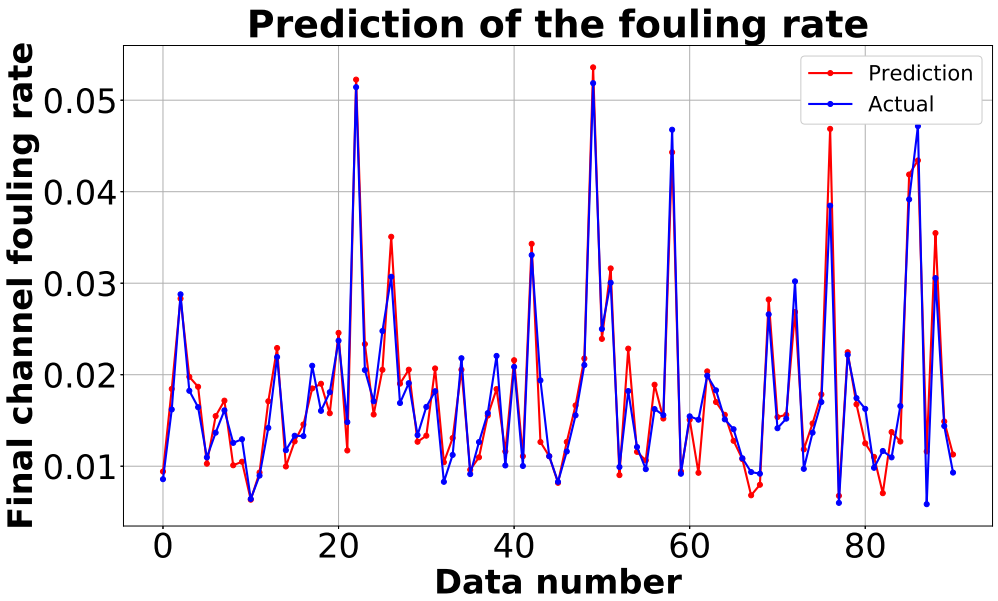
Also, our model reduced computational cost significantly (as shown in Table 3.2) while ensuring the desired accuracy (as shown in Table 3.3). The total time represents the entire simulation time of the machine learning model, including numerical simulation for the number of training data (20% of the total data), the training time, and the testing (prediction) time. The acceleration represents how many times faster it is than direct exhaustive simulations to calculate the fouling rate for 9010 combinations of parameters.



(a) $Q_o = 0.008(m^3/s)$



(b) $Q_o = 0.010(m^3/s)$



(c) $Q_o = 0.012(m^3/s)$

Figure 3.6: The prediction of the food channel fouling rate

Flow rate(m^3/s)	Training (s)	Prediction (s)	Total time (s)	Acceleration
0.008	13.74	0.0008	172.90	4.60
0.01	12.46	0.001	174.12	4.64
0.012	12.22	0.0008	171.39	4.64

Table 3.2: Computation time comparison

Flow rate(m^3/s)	RMS error of the fouling rate
0.008	0.00408
0.01	0.00318
0.012	0.00246

Table 3.3: Root mean squared error of prediction

This implies that the machine learning model could learn and detect the pattern of the fouling rate (which includes the process of solving differential equations) and predict the fouling of the channel, without having any mathematical or physical knowledge to solve differential equations and physics problems. This could illustrate the potential predictive power of machine learning with the low computational cost because we do not need to do numerical simulation for all the combinations which may be included in the genetic algorithm process.

3.14 The Overall Outlook

Currently, a modest level of modern technologies has been implemented in food production. For example, sensors, cameras, telecommunications have not been widely deployed. Furthermore, the cost of specialized equipment has been prohibitive and the development of coherent, easy-to-use, rapid data fusion/management systems across different platforms is lacking. Additionally, while control systems exist, they simply are too slow to be useful in deployed mobile computing platforms in harsh environments. The long term mission of this research is to integrate and implement convergent research in the development of smart, robust, and inexpensive systems that are easy to maintain, upgrade, and deploy, incorporating state-of-the-art technologies. A key to much of this work is the transfer of advances in the fields of Advanced Manufacturing and Computational Science to food production. In particular, digital twins, which refers to a digital replica of physical systems, blend artificial intelligence, machine learning, and software analytics with data to create living digital computer models that can update and change in tandem with their physical counterparts. We will seek to enable real-time simulation of processing devices to operate in tandem with their

deployed response. Updates to the digital twin are made continuously in near real-time, which necessitates rapid wireless communication, hyperspectral cameras and sensor fusion, and rapid simulation of process behavior. The digital twin concept should quickly ascertain fault behavior by utilizing the best available data. Today, there is no shortage of simulation codes; however, the fundamental limitations are real-time accuracy and deployable in-the-field use in harsh environments. A core issue across all domains of application is extreme flexibility - the ability of a system to adapt to rapid changes in the environment and system capabilities by autonomously modifying tasks and then apply various problem-solving approaches. In this context, a goal will be to make fundamental advances in several coupled autonomy-related fields to increase functional flexibility, while being constrained by the multiple aspects of variable system capabilities and operation in complex environments. These methods should be benchmarked and validated against an extensive suite of experimental tests.

Table 3.4: Parameter combinations of the test data ($Q_o = 0.008\,(m^3/s)$)

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
0	1.549	0.15	2.456	7759.404	3433.3	0.218
1	3.847	0.294	0.335	6105.349	4259.458	0.508
2	3.895	0.245	0.819	4181.942	5092.967	0.266
3	1.064	0.125	0.506	6140.392	5263.828	0.193
4	2.562	0.297	2.351	3775.15	3415.132	0.438
5	8.099	0.237	0.121	3727.23	2749.679	0.195
6	4.487	0.226	0.157	5545.354	3138.772	0.328
7	6.373	0.186	1.66	7784.213	5910.001	0.268
8	4.621	0.091	1.949	6439.95	4414.465	0.188
9	7.89	0.277	0.12	4903.641	1327.825	0.143
10	5.276	0.264	1.484	7035.64	3099.573	0.27
11	5.347	0.165	1.009	8632.634	5082.983	0.461
12	7.999	0.212	1.315	6683.187	1967.633	0.44
13	8.607	0.051	1.426	8422.895	3126.233	0.406
14	4.689	0.181	2.007	7584.325	5890.057	0.373
15	6.837	0.271	1.947	3265.004	4077.301	0.422
16	4.067	0.101	0.14	7968.876	3444.474	0.436
17	6.099	0.144	1.381	5948.694	4144.484	0.279
18	6.815	0.24	1.905	3973.045	3267.218	0.346
19	4.376	0.136	0.722	7860.504	2820.899	0.5
20	9.257	0.197	0.173	3092.617	2956.472	0.158
21	5.528	0.122	0.209	4989.043	5837.073	0.14
22	5.655	0.071	0.528	6727.897	5773.078	0.116
23	6.002	0.191	2.224	3335.19	3355.906	0.306
24	7.627	0.064	0.304	4873.36	1369.402	0.167
25	7.68	0.106	1.054	3243.908	4748.334	0.216
26	8.307	0.097	1.901	7089.897	2308.456	0.237
27	2.666	0.153	1.32	4057.853	1012.664	0.255
28	4.896	0.116	2.103	5193.494	5475.022	0.146
29	6.049	0.061	2.233	6832.804	4226.989	0.133
30	9.659	0.077	0.493	7077.536	3192.111	0.243
31	7.775	0.255	1.57	8566.101	1376.947	0.472
32	8.81	0.283	1.545	4493.745	5214.075	0.235
33	2.054	0.268	1.047	7779.834	1362.59	0.191
34	8.589	0.216	1.123	7107.375	3363.087	0.17
35	9.497	0.283	1.739	4630.313	1993.417	0.271
36	5.772	0.136	1.529	4635.428	4075.312	0.472

Continued on next page

Table 3.4 – continued from previous page

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
37	3.959	0.249	0.567	4696.297	1548.019	0.328
38	5.166	0.06	0.816	7087.657	5301.154	0.308
39	6.712	0.158	0.832	7776.77	2382.714	0.327
40	7.988	0.19	0.145	3634.772	1554.625	0.236
41	1.044	0.117	0.37	6138.783	4653.459	0.239
42	4.365	0.054	2.375	3587.32	5727.176	0.362
43	9.887	0.063	0.317	8819.026	5592.158	0.225
44	4.351	0.225	0.663	3033.798	5027.076	0.194
45	3.4	0.292	1.52	8851.484	2095.329	0.236
46	3.999	0.292	0.947	6391.476	4732.204	0.139
47	1.701	0.155	0.129	7762.79	5396.288	0.306
48	2.909	0.281	2.455	3551.428	2084.967	0.373
49	9.847	0.167	0.844	5712.906	4155.826	0.477
50	1.418	0.056	0.512	3089.134	5009.892	0.255
51	2.694	0.169	2.128	8402.041	2136.059	0.171
52	3.747	0.117	1.019	4391.483	4364.756	0.517
53	4.239	0.3	2.043	5246.343	4470.875	0.412
54	9.979	0.077	0.273	4239.395	2739.448	0.396
55	4.874	0.261	0.328	8498.197	5401.833	0.24
56	6.398	0.056	0.301	3232.287	3332.41	0.373
57	7.663	0.078	0.154	3122.636	2411.005	0.214
58	1.514	0.14	0.238	6034.478	5289.729	0.509
59	9.996	0.19	0.883	6812.561	5578.166	0.165
60	1.607	0.251	1.41	4849.983	3990.129	0.139
61	9.718	0.12	0.854	4394.153	2442.501	0.378
62	4.92	0.099	1.503	3621.493	4953.384	0.446
63	6.286	0.195	0.854	8390.804	5191.483	0.287
64	2.437	0.089	1.775	5797.98	1149.577	0.481
65	1.329	0.191	1.449	8568.115	5527.728	0.315
66	5.432	0.257	2.285	6038.271	4253.427	0.269
67	4.804	0.278	2.287	8215.119	3030.96	0.115
68	8.814	0.184	0.157	3068.411	2843.516	0.151
69	6.651	0.144	2.44	6823.313	3277.01	0.413
70	9.965	0.28	1.788	7031.894	3760.994	0.472
71	3.354	0.086	1.932	5018.222	3000.592	0.382
72	1.169	0.235	1.463	6540.024	4345.719	0.497
73	3.062	0.275	2.028	3219.607	3852.654	0.195
74	4.217	0.249	0.704	3738.223	3115.538	0.263
75	5.827	0.074	2.087	4486.627	4704.963	0.519

Continued on next page

Table 3.4 – continued from previous page

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
76	7.767	0.244	0.141	3435.857	1243.581	0.134
77	1.346	0.154	0.224	6231.153	4732.696	0.224
78	1.27	0.241	1.288	8043.79	4691.051	0.481
79	5.676	0.24	1.278	8149.764	3998.937	0.463
80	5.215	0.081	0.316	4160.237	1821.958	0.512
81	5.495	0.149	1.72	6061.707	3496.811	0.309
82	9.217	0.153	0.451	5250.311	1459.664	0.467
83	7.969	0.069	1.825	7418.46	4914.177	0.509
84	3.173	0.183	2.44	8438.922	3141.411	0.483
85	1.882	0.141	2.28	4967.85	1073.279	0.38
86	5.487	0.156	1.35	3969.385	2644.987	0.238
87	6.69	0.281	1.166	8996.767	1632.565	0.362
88	5.602	0.092	0.341	7741.338	3433.192	0.111
89	4.442	0.284	1.206	8580.623	2926.173	0.357
90	1.573	0.082	1.427	4317.658	5299.801	0.179

Table 3.5: Parameter combinations of the test data ($Q_o = 0.010\,(m^3/s)$)

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
0	6.14	0.131	2.336	3580.736	1074.287	0.405
1	7.366	0.194	1.319	7920.128	3488.167	0.441
2	6.546	0.069	0.595	7439.564	1296.572	0.142
3	8.069	0.197	0.273	7197.536	2553.029	0.394
4	6.992	0.198	0.91	3704.36	5351.689	0.308
5	6.506	0.169	0.215	5782.7	5007.166	0.494
6	1.283	0.075	1.633	7819.798	4286.152	0.518
7	8.999	0.064	2.009	4702.403	2703.609	0.232
8	6.913	0.057	2.295	6070.889	3640.574	0.211
9	5.313	0.28	1.504	5434.051	4517.909	0.285
10	8.343	0.149	1.713	7696.903	3796.667	0.119
11	4.251	0.268	2.251	5347.55	5358.302	0.366
12	5.399	0.195	0.924	4955.521	5341.165	0.495
13	1.573	0.084	0.16	3933.561	1358.369	0.427
14	5.526	0.059	1.299	4417.878	5746.938	0.233
15	7.907	0.199	1.525	8347.193	5045.599	0.15
16	7.913	0.184	1.262	4081.202	5150.931	0.258
17	8.825	0.118	0.127	3679.186	3709.769	0.261
18	8.208	0.244	1.45	4764.49	4500.437	0.434
19	6.495	0.238	0.22	6225.841	4326.061	0.2
20	3.545	0.159	0.58	3533.284	1859.182	0.337
21	3.646	0.201	2.372	6478.118	3440.068	0.12
22	3.986	0.195	0.497	7797.545	1876.565	0.137
23	1.525	0.121	0.24	5680.408	5489.909	0.301
24	2.546	0.202	1.015	8919.161	3901.178	0.376
25	1.336	0.071	1.671	6119.824	2968.495	0.328
26	2.328	0.061	2.496	3945.234	4892.474	0.199
27	8.521	0.226	1.592	8861.208	4454.62	0.456
28	9.33	0.188	0.625	7597.352	1331.154	0.181
29	9.309	0.083	0.142	4597.605	3840.298	0.396
30	4.506	0.211	2.116	6237.805	1367.773	0.402
31	7.365	0.152	1.846	8430.727	1885.953	0.377
32	6.613	0.216	2.199	8504.861	2341.794	0.421
33	4.45	0.238	1.446	3544.603	4181.313	0.34
34	4.609	0.087	0.288	7393.452	1554.853	0.288
35	9.616	0.261	2.172	7403.373	5379.083	0.386
36	1.071	0.089	1.857	8608.928	1252.925	0.439

Continued on next page

Table 3.5 – continued from previous page

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
37	3.446	0.193	0.418	4291.864	3091.729	0.117
38	6.782	0.106	2.198	8922.436	3786.966	0.397
39	4.383	0.166	2.492	5042.57	1773.285	0.472
40	6.799	0.254	0.11	6127.599	1292.772	0.196
41	5.138	0.134	1.987	7621.284	2952.805	0.371
42	7.023	0.175	2.296	3309.995	4505.566	0.489
43	7.086	0.267	0.218	4201.604	2153.164	0.159
44	5.792	0.145	1.29	3216.131	1738.549	0.497
45	8.795	0.155	2.01	5458.378	2902.507	0.334
46	5.576	0.206	0.772	4151.183	1507.298	0.179
47	5.532	0.167	1.711	8743.86	2311.082	0.309
48	9.374	0.248	1.838	8682.585	2009.389	0.359
49	6.109	0.078	2.381	3636.448	2012.176	0.296
50	2.166	0.203	1.099	8702.51	3044.366	0.334
51	5.297	0.077	2.366	4051.802	3998.746	0.518
52	3.328	0.271	2.136	7263.344	2043.585	0.261
53	4.575	0.101	0.154	6912.569	2418.829	0.19
54	8.081	0.151	2.457	7083.64	2776.945	0.17
55	3.908	0.177	0.548	6635.572	3127.318	0.314
56	6.428	0.264	1.733	6713.824	2894.822	0.515
57	8.357	0.123	0.296	6581.155	2697.933	0.323
58	9.003	0.214	0.194	5955.788	1192.977	0.253
59	9.802	0.055	0.202	6481.557	5289.709	0.308
60	2.432	0.103	2.243	4260.709	3420.331	0.171
61	7.717	0.12	1.827	6263.439	2399.915	0.37
62	3.475	0.131	1.634	7523.677	4489.278	0.256
63	7.867	0.129	0.271	3663.054	1972.714	0.347
64	5.743	0.271	2.386	4490.073	1391.131	0.26
65	1.977	0.193	2.254	6120.876	4469.23	0.458
66	2.757	0.152	0.42	6308.862	1341.053	0.27
67	4.742	0.183	1.68	4413.228	1015.702	0.484
68	9.889	0.096	0.274	3253.247	1898.977	0.44
69	6.368	0.265	0.808	7856.217	5156.008	0.455
70	9.919	0.095	1.355	5642.983	1646.385	0.291
71	2.827	0.221	0.558	8795.148	4688.409	0.265
72	4.534	0.184	1.4	4462.572	1254.301	0.134
73	1.227	0.166	1.265	8281.254	3170.687	0.14
74	6.591	0.297	1.538	4557.043	5111.217	0.25
75	5.312	0.147	1.817	5661.513	1222.957	0.455

Continued on next page

Table 3.5 – continued from previous page

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
76	3.909	0.118	1.692	3771.867	3305.802	0.194
77	9.697	0.269	2.317	7709.971	5539.29	0.173
78	4.905	0.12	0.937	3135.147	5807.623	0.158
79	7.611	0.111	0.419	5335.717	5490.597	0.245
80	2.27	0.282	0.806	7178.22	5990.579	0.375
81	7.148	0.119	0.147	3693.587	2416.442	0.33
82	2.208	0.291	1.094	3544.886	3700.504	0.196
83	9.935	0.068	1.739	6036.328	2055.888	0.206
84	3.848	0.103	1.569	6220.79	1088.941	0.513
85	4.786	0.068	0.35	4669.444	3201.769	0.516
86	3.297	0.186	2.295	3256.407	1832.094	0.442
87	8.355	0.163	1.026	3105.065	5858.134	0.487
88	9.78	0.23	1.363	8354.454	4339.494	0.312
89	8.423	0.21	1.314	4572.152	3396.524	0.126
90	3.571	0.208	2.002	4743.436	4432.712	0.221

Table 3.6: Parameter combinations of the test data ($Q_o = 0.012\,(m^3/s)$)

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
0	6.4	0.067	0.522	7876.243	5294.28	0.195
1	8.78	0.24	1.116	3315.163	3814.936	0.241
2	1.915	0.132	2.017	5458.866	5279.052	0.474
3	5.979	0.146	2.487	4069.539	4539.445	0.317
4	4.064	0.115	2.412	3081.456	4120.004	0.327
5	3.761	0.085	0.371	3162.713	5162.42	0.458
6	2.31	0.069	0.104	4964.014	5683.006	0.335
7	4.44	0.151	0.919	8144.161	5741.744	0.308
8	1.221	0.209	0.376	8897.054	1056.524	0.116
9	6.439	0.21	0.847	5777.023	1814	0.321
10	8.849	0.288	0.183	7840.757	1727.002	0.258
11	7.487	0.171	0.55	8558.128	3965.964	0.121
12	7.588	0.169	1.783	6494.871	2374.835	0.338
13	8.981	0.216	0.99	7794.138	4990.806	0.472
14	8.75	0.139	1.734	5713.009	2484.315	0.17
15	4.216	0.093	0.442	6046.996	5614.149	0.493
16	5.844	0.282	0.22	3935.321	4672.811	0.256
17	4.761	0.191	1.158	7752.158	3894.048	0.431
18	4.763	0.269	2.057	3853.139	1174.468	0.152
19	5.056	0.214	1.258	4200.577	2981.031	0.413
20	6.769	0.206	1.161	8467.196	5783.529	0.388
21	9.673	0.101	0.352	5057.071	2211.53	0.189
22	3.21	0.297	2.194	3280.192	4266.42	0.318
23	9.994	0.273	1.797	4081.603	2730.073	0.178
24	4.713	0.188	1.656	5510.669	3933.104	0.159
25	1.086	0.188	0.581	5109.942	1802.562	0.365
26	9.754	0.291	1.306	8539.635	4946.283	0.269
27	6.067	0.3	0.765	3245.378	1794.394	0.517
28	3.226	0.283	0.529	7603.214	4977.732	0.179
29	7.635	0.114	2.404	6106.222	4012.162	0.235
30	8.139	0.288	0.121	3781.991	1329.161	0.147
31	8.736	0.26	1.823	8281.504	1041.569	0.359
32	8.915	0.206	0.463	6300.075	2275.214	0.257
33	6.199	0.123	1.633	8890.819	2525.629	0.197
34	5.162	0.235	1.417	4398.862	4607.633	0.146
35	8.868	0.062	1	3597.699	5313.489	0.27
36	3.883	0.146	0.627	6571.866	1986.721	0.379

Continued on next page

Table 3.6 – continued from previous page

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
37	2.454	0.153	0.578	3642.078	4549.263	0.276
38	1.263	0.145	0.207	4141.563	3678.891	0.372
39	5.341	0.089	0.959	4360.097	2066.179	0.438
40	1.107	0.183	0.205	8058.297	4336.652	0.302
41	7.514	0.088	2.165	7392.249	1587.091	0.498
42	4.528	0.294	1.271	5270.921	4601.011	0.234
43	8.451	0.133	0.204	5234.141	1257.345	0.195
44	9.383	0.076	0.184	6001.8	1902.392	0.341
45	9.766	0.11	1.323	5491.346	3151.914	0.112
46	8.075	0.121	1.084	4987.955	5017.741	0.142
47	8.623	0.141	1.427	5743.662	5370.107	0.447
48	8.812	0.177	2.22	4669.98	5199.864	0.291
49	4.039	0.26	2.427	5501.073	4214.584	0.494
50	3.25	0.168	1.554	7151.826	4625.882	0.491
51	1.659	0.296	1.132	6315.573	1746.152	0.305
52	5.033	0.06	2.072	5041.384	2139.78	0.286
53	5.723	0.265	1.25	8936.12	4446.507	0.13
54	7.419	0.195	1.261	6695.9	1515.017	0.249
55	6.283	0.054	0.949	4657.896	4149.641	0.511
56	8.674	0.191	2.111	4430.844	3077.662	0.261
57	3.723	0.136	2.064	4707.604	1925.808	0.329
58	1.435	0.266	1.987	6817.481	2040.755	0.43
59	8.294	0.198	0.958	7924.672	1850.88	0.135
60	5.616	0.16	2.158	7039.122	3272.005	0.15
61	8.249	0.225	0.383	4251.902	1873.476	0.314
62	7.484	0.2	1.954	7625.207	2127.711	0.507
63	8.496	0.267	0.382	3791.214	5095.26	0.49
64	9.224	0.147	1.351	8413.807	3864.131	0.358
65	4.129	0.081	2.221	7633.587	2446.464	0.506
66	3.505	0.059	1.365	5032.053	3032.907	0.415
67	3.215	0.109	0.23	7915.023	1055.042	0.471
68	5.79	0.059	1.71	6376.457	2151.38	0.213
69	2.214	0.178	1.516	5894.345	4671.929	0.35
70	3.621	0.098	1.716	8745.389	4080.627	0.309
71	2.539	0.179	0.465	6227.274	3531.866	0.225
72	9.306	0.274	1.967	3629.099	5095.89	0.135
73	7.608	0.055	2.16	8401.329	4750.897	0.487
74	2.708	0.066	2.313	6694.385	3688.736	0.117
75	2.993	0.15	1.284	3730.68	4412.211	0.292

Continued on next page

Table 3.6 – continued from previous page

Data	α	v_p	$\sigma_{p0}(\Omega^{-1}m^{-1})$	$\rho_p(kg/m^3)$	$C_p(J/(kgK))$	$k_{p0}(W/(mK))$
76	2.232	0.258	2.491	8767.063	4597.317	0.194
77	6.972	0.251	0.256	7734.183	1171.318	0.185
78	4.931	0.171	1.98	4670.762	4427.269	0.394
79	7.116	0.168	2.029	8174.577	3866.464	0.174
80	7.671	0.241	0.107	4285.157	1597.274	0.142
81	4.819	0.116	0.28	3542.585	1363.044	0.156
82	4.388	0.225	0.355	7705.786	3898.758	0.179
83	3.838	0.129	1.132	3071.931	3060.07	0.182
84	7.787	0.223	0.106	3666.485	1475.339	0.133
85	3.691	0.283	2.366	7904.586	3782.818	0.199
86	1.286	0.195	1.802	3339.908	5903.523	0.456
87	5.092	0.114	0.203	3557.497	1379.287	0.166
88	2.214	0.265	1.511	4430.614	3331.978	0.242
89	6.272	0.207	0.895	7031.337	3987.298	0.195
90	7.152	0.068	0.624	3125.018	3860.56	0.437

Chapter 4

Tool Path Optimization of the Selective Laser Sintering Process using Deep Learning

4.1 Abstract

The contents of this chapter are from the journal paper, "Tool path optimization of Selective Laser Sintering process using Deep Learning" by Kim, D.H., and Zohdi, T.I., which was submitted to the *Journal of Manufacturing Science and Engineering, ASME*.

Advancements in additive manufacturing (3D printing) have enabled researchers to create complex structures, offering a new class of materials that can surpass their individual constituent properties. Selective Laser Sintering (SLS) is one of the most popular additive manufacturing techniques and uses laser power to bond powdered material into intricate structures. It is one of the fastest additive manufacturing processes for printing functional, durable prototypes, or end-user parts. It is also widely used in many industries, due to its ability to easily make complex geometries with little to no added manufacturing effort. In the SLS process, tool path selection is important because it is directly related to the integrity of a 3D printed structure. In this research, we focus on how to obtain an optimal tool path for the SLS process from a numerical simulation. Also, we apply a Deep Learning technique to accelerate the simulation of the SLS processes, while obtaining accurate simulation results.

4.2 Introduction

Selective Laser Sintering (SLS) is an additive manufacturing technique that uses a laser as a power source to sinter powdered material,[1] by directing the laser automatically at points

[1]PA12 is a widely used thermoplastic material for Selective Laser Sintering which is suitable for many applications [154]

in space, dictated by a 3-D model.[2] The original SLS process was invented in 1986 at the University of Texas at Austin, with increasing applications as the technology matures [161]. Many researchers have proposed various ways to simulate the SLS process, to figure out the mechanical/thermal behavior of the sintered material [162]. Dong et al. [163] developed a transient three-dimensional Finite Element (FE) model to simulate the temperature evolution during the SLS process. Kolossov et al. [164] created a three-dimensional finite element model for SLS processes, considering the non-linear behavior of thermal conductivity and specific heat due to temperature changes and phase transformations. Matsumoto et al. [165] proposed a FE model for calculating the temperature and stress distribution in a single layer of metallic powder in selective laser processing. Simchi [166] and Simchi and Pohl [167] used experimental results to observe microstructural evolution and densification during laser sintering of metal powders. Gusarov and Kruth [168] employed a radiation transfer model to calculate absorptances and deposited energy profiles while processing thin layers of metallic powder, and provided an analytical equation of laser penetration as a function of particle size and powder bed density. This work was followed by a finite difference simulation of heat transfer during Selective Laser Melting [169]. There also have been Discrete Element Models, which is a good option for simulation of additive manufacturing processes, for modeling and simulation of the laser processing of the powdered particles [70, 170–173]. That work developed a coupled discrete element-finite difference model of SLS process [174].

As mentioned, tool path selection in additive manufacturing is important because it is directly related to the durability of the 3D printed structure [74]. This research aims to obtain an optimal tool path for the SLS process from numerical simulations. During laser processing, temperature gradients occur within a 3-D printed structure, which can cause unintended residual stresses. This is one of the major reasons for premature failure in parts printed by additive manufacturing. In order to find an optimal laser path which minimizes temperature gradients among a variety of tool paths, we first need to solve a dynamic programming problem to find all the possible laser paths for the laser grid configuration of a given geometry. After obtaining all possible laser paths, we could use numerical methods, such as the Finite Difference Method, the Discrete Element Method, the Finite Element Method, etc.,[3] to calculate the thermal gradients or residual stresses of a printed geometry.

However, when the simulation system becomes excessively large, it can become computationally expensive to perform exhaustive simulations to calculate the temperature gradients for large numbers of possible paths. Accordingly, in order to efficiently obtain an optimal laser path for SLS processes, we apply Deep Learning techniques, which allow computers to learn and detect patterns from noisy/complex data sets, and then to extract discovered patterns to make predictions for future unknown data. Based on this idea, we can accelerate the numerical simulations with high accuracy, as well as low computational cost. In order

[2]For the readers who are interested in comprehensive background of additive manufacturing, refer to [155–160].

[3]For the readers who are interested in further numerical methods for solving partial differential equations, refer to [129, 132, 175–177]. MATH 228A/B from the Department of Mathematics at UC Berkeley is also helpful.

to efficiently process multi-dimensional data, we employ the Convolutional Neural Network (CNN) to solve this manufacturing problem.

4.3 Technological Approaches

The main objective of this work is to develop a deep learning model to efficiently predict the optimal tool path of the SLS process, which minimizes the average thermal gradient. In order to achieve this, we first obtain all the possible laser paths needed to print the objective geometry. We then solve a dynamic programming problem (which will be explained further in the next section). After that, we model and simulate the SLS process in order to calculate the average thermal gradient. Finally, based on the obtained simulation data, we construct a deep learning model to predict the optimal laser path for the SLS process. The overall flowchart is shown in Figure 4.1.

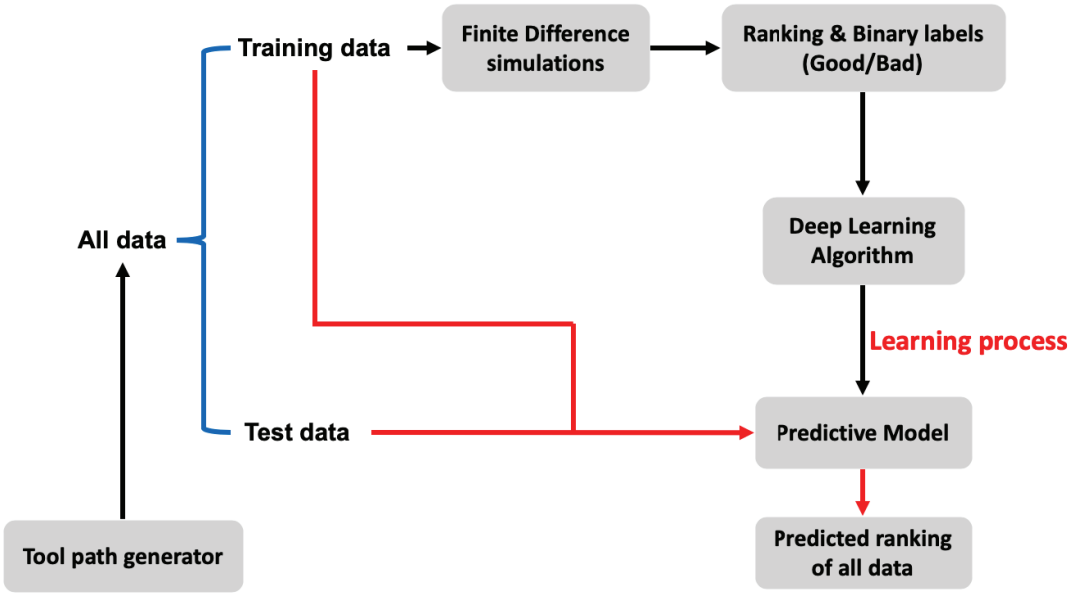


Figure 4.1: The overall algorithm flowchart

4.4 Tool Path Generation using Dynamic Programming

In this section, we will focus on finding all the possible laser paths within the discretized domain. Examples of the possible laser paths for a simple 4 by 4 grid is shown in Figure 4.2. After discretizing the domain into a structured grid, we can set the starting point of the laser from any of the nodes in the grid. In order to simplify the problem, we set the following restrictions on the laser path:

- The laser should visit all the nodes in the grid.
- The laser should visit each node only once.
- The laser cannot jump over nodes.

The main problem we encounter is that, as the laser moves from node to node, every single path obtained so far branches down to the new points, as shown in Figure 4.3. For example, if we have connected the points (0, 4, 8, 12, 13, 14, 10) so far, then there are three possible next points 9,6, and 11. Therefore, (0, 4, 8, 12, 13, 14, 10) branches down to (0, 4, 8, 12, 13, 14, 10, 9), (0, 4, 8, 12, 13, 14, 10, 6), and (0, 4, 8, 12, 13, 14, 10, 11) in the next step. We have to accumulate the cases to search for all the possible tool paths.

Due to this difficulty, we apply a dynamic programming technique to efficiently solve for the laser path. Dynamic programming is a technique that simplifies a complicated problem by breaking it down into simpler sub-problems in a recursive manner. Dynamic programming was developed by Richard Bellman in the 1950s and has found applications in numerous fields, ranging from aerospace engineering to economics.[4]

Memoization is used primarily to solve the dynamic programming problem by a top-down approach. This accelerates computer programs by storing the results of expensive function calls and returning the cached result, whenever the same inputs occur again. In order to develop the tool path finder which could be applied for various kinds of complex geometries, we employ memoization in order to store the previous movements of paths, so that all the possible paths can be found efficiently.

There could be many possible starting points for the laser paths and the number of starting points is equal to the number of points in the laser grid. However, since the laser grid obtained from the geometry could be symmetric in either the x-axis, y-axis, or $y = x$ line, one can reduce the number of starting points to remove symmetrically overlapping laser paths. For example, one can start from only three points marked in red, as shown in Figure 4.4, because the given laser grid in Figure 4.4 is symmetric about the x-axis, y-axis, and $y = x$ line. In other words, one does not need to consider all nodes as starting points.

[4]For the readers who are interested in comprehensive background of dynamic programming, refer to [178–181].

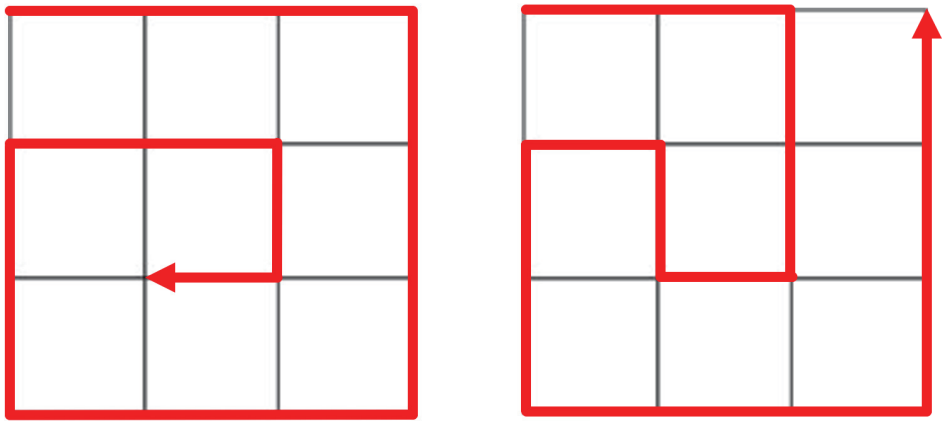


Figure 4.2: Examples of the possible laser paths in a 4 by 4 laser grid

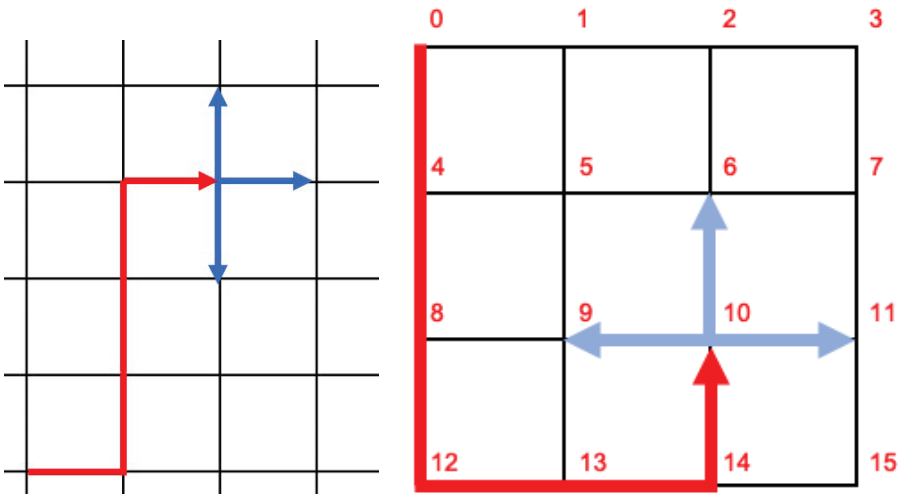


Figure 4.3: Laser path finding (RED: Accumulated path points, BLUE: New possible points)

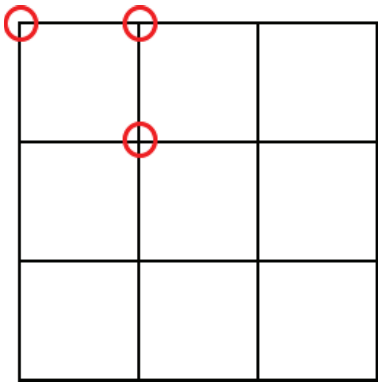


Figure 4.4: Starting points considering the symmetric laser grid

The Trick for Finding Paths

This subsection deals with the trick I used in the path finding process. Even though we can apply a dynamic programming technique to find all the possible laser paths, every single path obtained so far branches off to the possible new points. Also, those new points are added back to the *current paths* to create *new current paths*. They branch off again to the new points in the next time steps. This process is iterated over and over until we obtain all the final paths visiting all the nodes in a given laser grid.

For this reason, performing calculations to obtain all the possible paths could be computationally expensive when the size of the laser grid is large. To overcome this limitation, one could perform early detection of the paths to be failed and remove them from the candidate path set in advance. This work could prevent path finder from a huge amount of unnecessary calculations. The examples of the failed paths are shown in Figure 4.5. In Figure 4.5, red lines represent *current paths* and blue arrows represent possible next points. As we can observe, they can not be included in the set of possible laser paths no matter what direction of blue arrows they choose, since either paths or boundary of the geometry already *encompassed* some node points in the laser grid.

To implement this, we could create a circular linked list (*flag list*) in either clockwise or counterclockwise direction along with boundary nodes, which indicates whether the boundary nodes of the laser grid are touched by the path. After that, we *flag-up* each boundary node when the path touches them. If flags have any interval between '1's (*flagged-up*), that is a failed path. This is applied in the same way for inner boundaries for hollow geometries. Also, a similar principle could be applied for detecting the *self-colliding* paths.

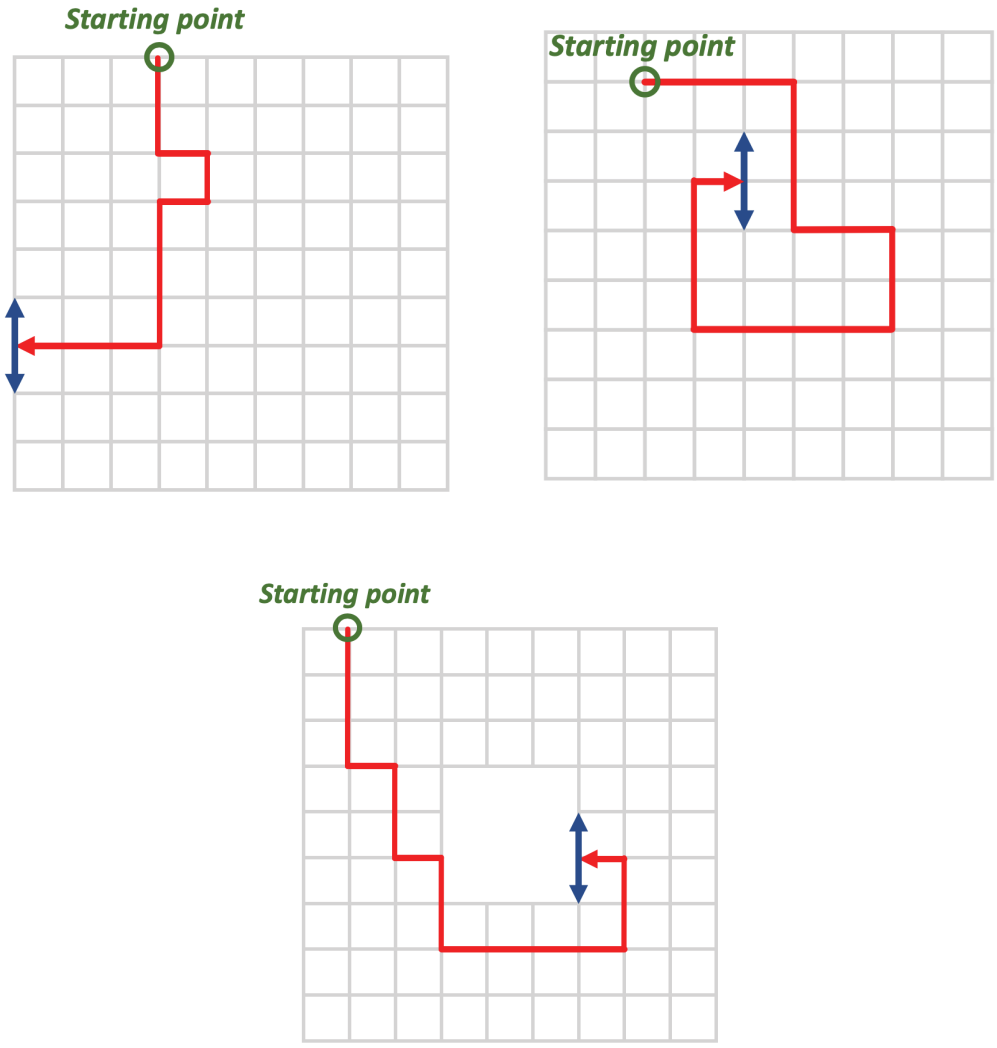


Figure 4.5: Failed paths to be removed from the candidate path set

4.5 Modeling and Simulation of the SLS Process

After obtaining all of the possible laser paths, one needs to mathematically model the laser sintering process in order to perform numerical simulations for given laser paths. A schematic of the laser processing is shown in Figure 4.6. As for the grid configuration, we adopt a material grid that is three times finer and includes the laser grid nodes, as shown in Figure 4.7. In Figure 4.7, yellow circles represent the area covered by laser. While the laser grid points are the positions through which a laser moves as time progresses, the material grid

points are the points where numerical simulations are actually performed to evaluate the thermal gradients. The simulation parameters are shown in Table 4.1.

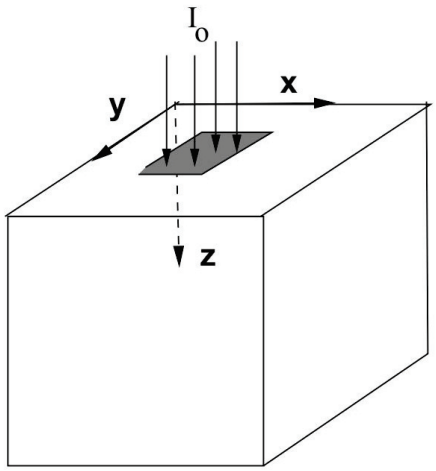


Figure 4.6: A schematic and the coordinate system of the laser processing

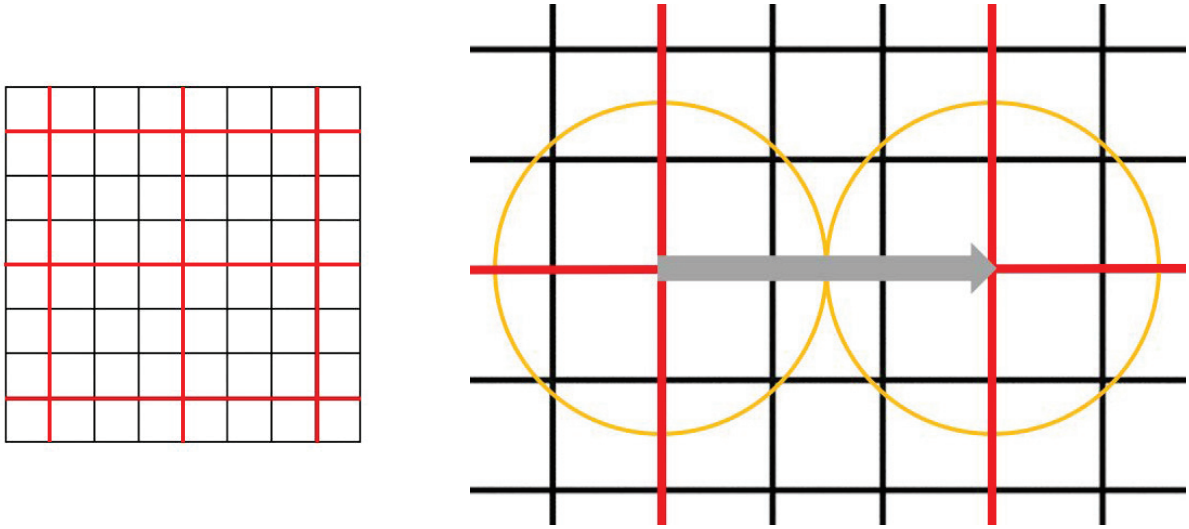


Figure 4.7: Grid configuration (RED: The laser grid, BLACK: The material grid)

The governing equation is as follows:

$$\rho C \frac{\partial \theta}{\partial t} = \nabla \cdot (K \nabla \theta) + I_{abs}, \tag{4.1}$$

where

- θ: Temperature (K)
- ρ: Density (kg/m^3)
- C: Specific heat capacity $(J/(kgK))$
- K: Heat conductivity (W/mK)
- I_{abs}: Absorbed laser energy within the laser zone (W/m^3)

Assuming that we have a constant thermal conductivity k for simplicity, the boundary conditions are as follows:

- Top surface: Convection

$$-k\frac{d\theta}{dz} = h(\theta_\infty - \theta) \tag{4.2}$$

- Other surfaces: Adiabatic

$$q_s = -k\frac{d\theta}{dx} = 0 \tag{4.3}$$

$$q_s = -k\frac{d\theta}{dy} = 0 \tag{4.4}$$

$$q_s = -k\frac{d\theta}{dz} = 0, \tag{4.5}$$

where θ_∞ is the ambient temperature (K), and q_S is heat flux at the surfaces (W/m^2). As for energy absorption I_{abs}, we use the Beer-Lambert penetration model for a Gaussian laser, where

$$I_{abs}(r,z) = I_0 e^{-\beta z} e^{\frac{-2r^2}{w^2}} \tag{4.6}$$

with

$$I_0 = \frac{2P}{\pi w^2}, \tag{4.7}$$

where β is an optical extinction coefficient [182], P is the power of the laser, and w is the laser radius. The optical extinction coefficient for a powder bed is modeled according to the theory of Gusarov et al. [168, 169]. Also, we can rewrite Equation 4.1 as follows:

$$\rho C\frac{\partial\theta}{\partial t} = k\big(\frac{\partial^2\theta}{\partial x^2} + \frac{\partial^2\theta}{\partial y^2} + \frac{\partial^2\theta}{\partial z^2}\big) + I_{abs}(r,z). \tag{4.8}$$

When we discretize Equation 4.8 using finite difference method with a forward Euler scheme for time integration,[5] we obtain:

$$\theta(t+\Delta t,x,y,z)=\theta(t,x,y,z)+\Delta t\Big[\frac{k}{\rho C}\big(A_x+A_y+A_z\big)+\frac{1}{\rho C}I_{abs}(r,z)\Big], \tag{4.9}$$

where

$$A_x=\frac{\partial^2\theta}{\partial x^2}=\frac{\theta(t,x+\Delta x,y,z)-2\theta(t,x,y,z)+\theta(t,x-\Delta x,y,z)}{\Delta x^2} \tag{4.10}$$

and

$$A_y=\frac{\partial^2\theta}{\partial y^2}=\frac{\theta(t,x,y+\Delta y,z)-2\theta(t,x,y,z)+\theta(t,x,y-\Delta y,z)}{\Delta y^2} \tag{4.11}$$

and

$$A_z=\frac{\partial^2\theta}{\partial z^2}=\frac{\theta(t,x,y,z+\Delta z)-2\theta(t,x,y,z)+\theta(t,x,y,z-\Delta z)}{\Delta z^2}. \tag{4.12}$$

The convective boundary condition [4.2] on the top surface is discretized as follows

$$h(\theta_\infty-\theta)+k\frac{\theta_1-\theta_0}{\Delta z}=0, \tag{4.13}$$

where θ_1 is the temperature at the outermost inner node, which is adjacent to the boundary node, and θ_0 is the temperature at the boundary node.

After obtaining the final temperature values throughout the domain, we get the average thermal gradient by calculating the average of the root mean squares of the temperature gradients throughout the inner nodes of the 3D printed layer, as shown in Equation [4.14]. For each gradient component, we take a central difference from the adjacent nodes.

$$\text{Average thermal gradient}=\frac{1}{N}\sum_{i\in Z}\sqrt{\left(\frac{\partial\theta}{\partial x}\right)_i^2+\left(\frac{\partial\theta}{\partial y}\right)_i^2+\left(\frac{\partial\theta}{\partial z}\right)_i^2}, \tag{4.14}$$

where Z is a printing zone sintered by laser, and N is the number of material grid nodes included in the printing zone. Also, in this work, we assume that we are printing a single layer, which has a thickness of $4\times\Delta L$. The temperature gradient values are then used for the labeling of the training/test data in the next section.

[5] Since the heat equation is *stiff*, explicit methods including the forward Euler method have the strict CFL (Courant–Friedrichs–Lewy) condition [183, 184]. Therefore, implicit methods including the Crank-Nicholson method [185] would be a good choice to relax the CFL condition. For more detail, refer to the chapter 9 of [129].

Symbol	Units	Value	Description
θ_∞	K	300	Ambient temperature
θ_0	K	300	Initial temperature
h	W/m^2K	10	Convection coefficient
K	$W/(mK)$	0.22	Thermal conductivity of PA12
ρ	kg/m^3	1100	Density of PA12
C	$J/(kgK)$	1590	Specific heat capacity of PA12
w	m	0.0025	Laser radius
v	mm/s	10	Laser scanning speed
P	W	200	Power term in I_{abs}
β	$1/m$	80	Optical extinction coefficient
ΔL	m	0.00167	Material grid gap size ($= \Delta x = \Delta y = \Delta z$)

Table 4.1: Simulation Parameters

4.6 Numerical Experiments

We implemented the simulation of the SLS process to print the example geometry shown in Figure 4.8. We consider two example laser paths, even though there could be numerous possible laser paths for this geometry and the corresponding laser grid. The possible starting points on the laser grid, considering the symmetry of the geometry, are also shown in Figure 4.8. There were 66,464 possible laser paths in total for this geometry and corresponding laser grid configuration, which were found by dynamic programming. The examples of laser paths are shown in Figure 4.9. Also, the temperature plots of the powder bed for the corresponding laser paths at the final processing time is shown in Figure 4.10.

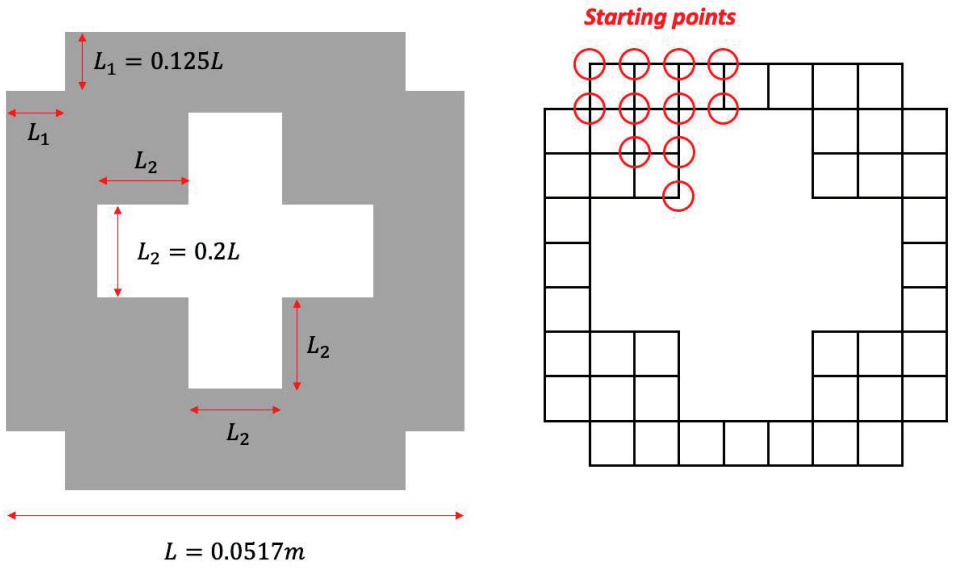
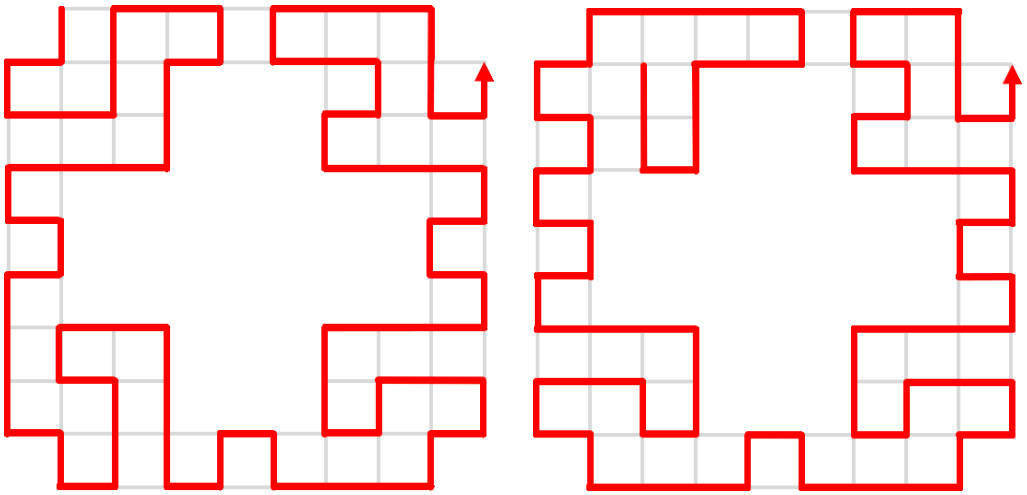


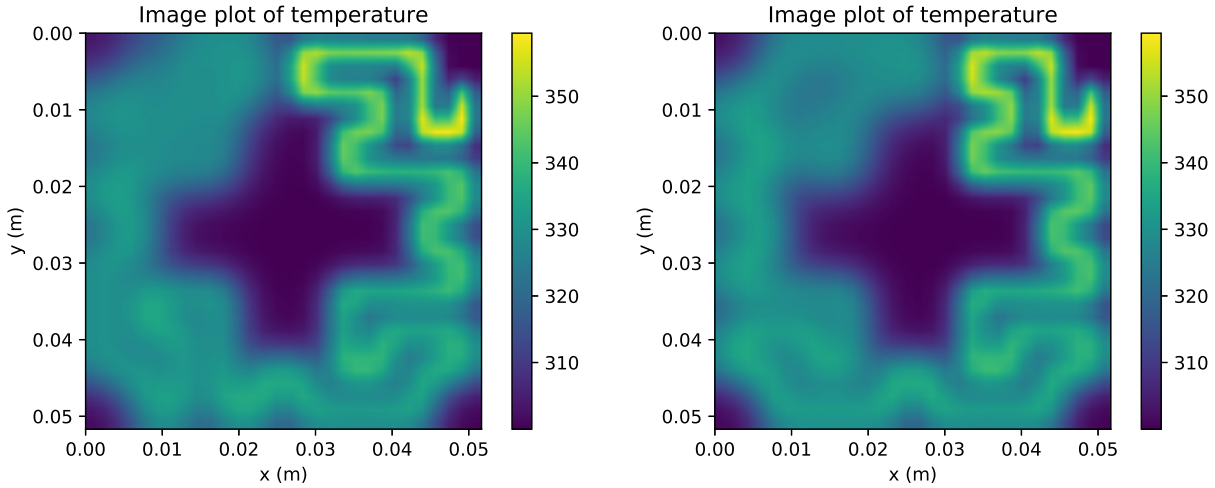
Figure 4.8: LEFT: Example geometry, RIGHT: Corresponding laser grid



(a) Example path 1 (b) Example path 2

Figure 4.9: Examples of the possible laser paths for the given geometry

In the same way, we can perform the numerical simulations for all the other laser paths to obtain the corresponding average thermal gradients described in Equation 4.14. The data distribution of the temperature gradients for 66464 laser paths is shown in Figure 4.11.



(a) Example path 1

(b) Example path 2

Figure 4.10: Temperature plots of the top surfaces at the final processing time (37.5 (s))

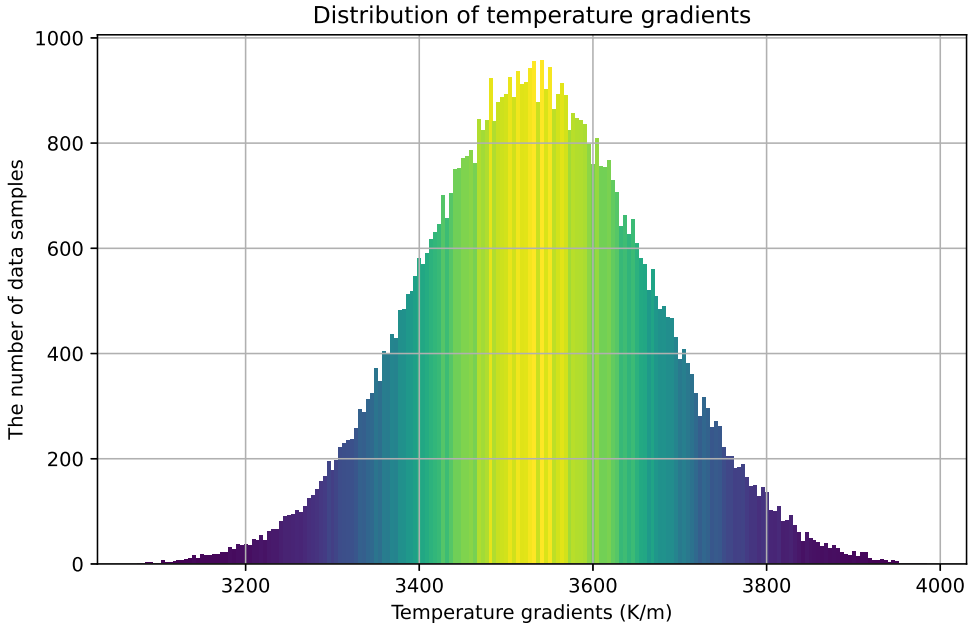


Figure 4.11: Distribution of the temperature gradients

Comments on Temporal Discretization

Temporal discretization refers to the discretization in the time domain for transient problems as laser processing simulation in this work. For temporal discretization, we should also consider how many time intervals will be included in every single material grid interval. If temporal discretization is too coarse, it will give less accurate results even though the simulation is fast. On the other hand, if temporal discretization is too fine, the simulation will consume more computational power despite more accurate results. Therefore, it is important to determine the optimal balance between these two in numerical simulations.

As mentioned above, the temperature gradient is a critical property in the SLS process because it is directly related to the unintended residual stress that could cause premature failure of a 3D printed structure. Therefore, we need to ensure reasonable accuracy for the thermal gradient calculation.

We could do a temporal discretization based on the following steps.

1. Randomly select a possible laser path for the given geometry.
2. With the finest time step,[6] run the simulation to obtain the temperature gradients for the path. Consider them as *true* temperature gradients ($GRAD_{true}$).
3. Select the coarsest time step considering the CFL condition.[7]
4. Run the simulation to obtain the temperature gradients ($GRAD$) and calculate the relative maximum error ($Error$) of the temperature gradients: $Error = max(\left|\frac{GRAD_{true} - GRAD}{GRAD_{true}}\right|)$.
5. Iterate Step 4 with finer time steps until $Error$ falls below the $Tolerance$ ($Error \leq Tolerance$).[8] If this is achieved, set this as a time step size for simulations.

4.7 Preprocessing of the Laser Paths

In deep learning, a Convolutional Neural Network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to image classification and recognition [25–28]. The CNN processes data that has a known grid-like topology [32]. Our next step is to convert the laser paths we found into gray-scale path map images to train the deep learning model (CNN). The way to convert the laser paths (time data) into the image (space data) is shown in Figure 4.12. The path maps (as shown in Figure 4.12) also help to visually check

[6] A large number of time steps between material grid points.

[7] The Courant–Friedrichs–Lewy (CFL) condition which is a necessary condition for convergence when solving time-dependent partial differential equations. For more detail, refer to [183, 184], or MATH 228A from the Department of Mathematics at UC Berkeley.

[8] For example, one could set the $Tolerance$ as 2-3%.

that each node was visited only once by looking at each row. Node numbering goes from the left to the right, and from the top to the bottom, starting from zero. On the converted path map image, we identify the white points based on the laser path. In other words, the white squares correspond to the laser location at each time step. For example, if the second point of the laser path is node number 4, then the time-stamp identifier for this point is '1' (since node numbering starts from zero), and the spatial identifier of this point is '4'.

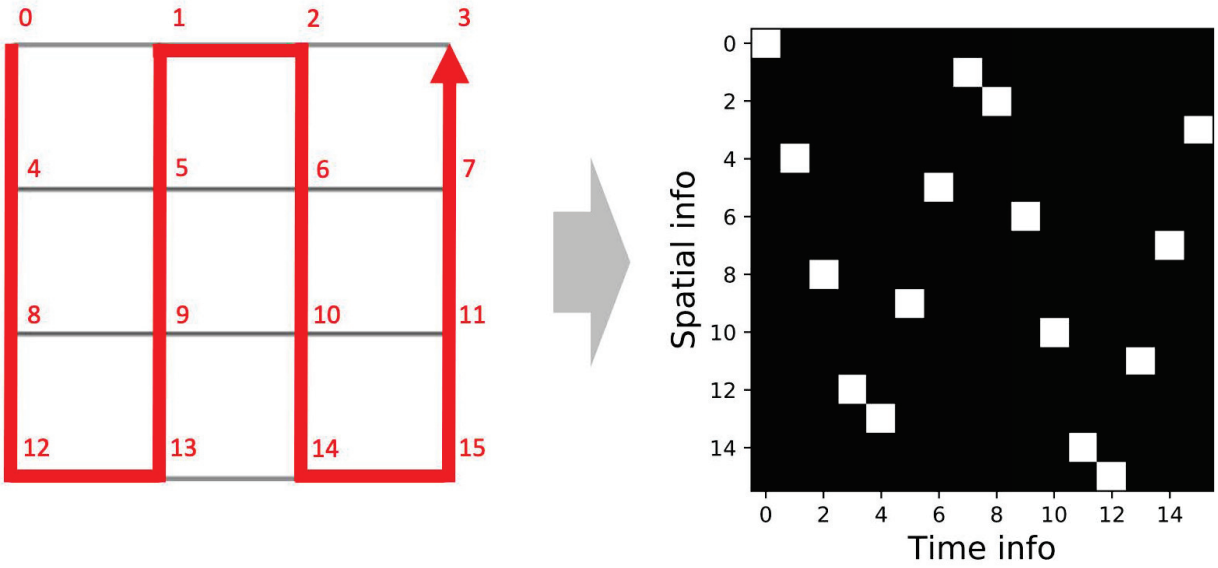
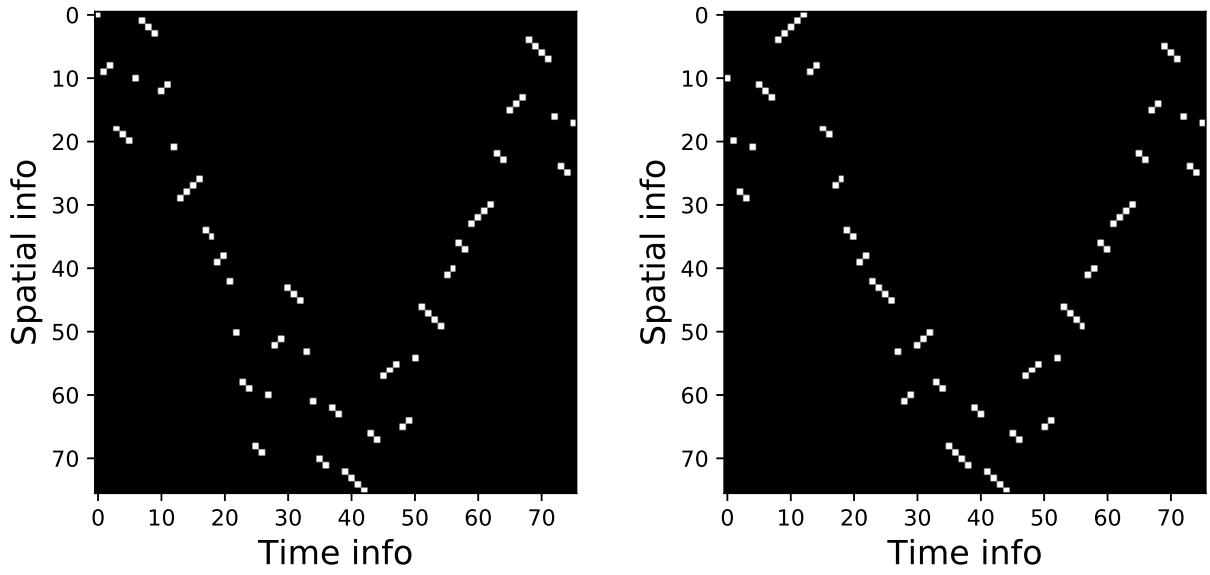


Figure 4.12: Preprocessing of path data

Since the number of the laser grid nodes of the given geometry is 76, every single laser path is preprocessed into 76 by 76 gray-scale images. The preprocessing of the previous example paths for the given geometry is shown in Figure 4.13.



(a) Image of example path 1

(b) Image of example path 2

Figure 4.13: Preprocessing of the paths into path map images

4.8 A Deep Learning Model to Predict the Optimal Tool Path

Deep learning is a branch of artificial intelligence based on a biologically-inspired learning process based on how neurons communicate and learn in living things, allowing computers to learn from the past data so that it could detect patterns and make predictions from noisy and complex data sets [32, 137–140]. The deep learning approach deals with the design of algorithms to learn from machine-readable data. Also, there has been some research on generating predictive models to solve a variety of engineering problems such as material design, computer vision, pattern recognition, and spam filtering [142–148, 186], including those in computational mechanics [187]. In our approach, we applied a deep learning algorithm to efficiently and accurately predict the optimal laser paths for SLS. We implemented our deep learning model with PyTorch (1.1.0 version), which is an efficient deep learning framework for Python and competent in both usability and speed [49].

In the previous section, we converted laser paths into grayscale images. Those images were processed in our deep learning model. Also, every path map image was ranked before training with the deep learning model, based on the average thermal gradient described in equation 4.14. The lower the average thermal gradient, the better the path. After that, we gave each path two kinds of labels: half of the group was labeled as ‘good’ (a label of ‘1’)

and the other half was labeled as 'bad' (a label of '0').

The architecture of the CNN that we used for training and prediction is shown in Figure 4.14. In order to reduce architectural complexity, we used a simple CNN structure having three convolutional layers, which was followed by a fully connected neural network. In Figure 4.14, *None* represents the number of data samples, and *FC Neural Network* represents the fully connected neural network. For initialization of the parameters, Xavier initialization was used for the weight initialization, in order to obtain substantially faster convergence [152]. The configuration of the hyper-parameters is shown below.

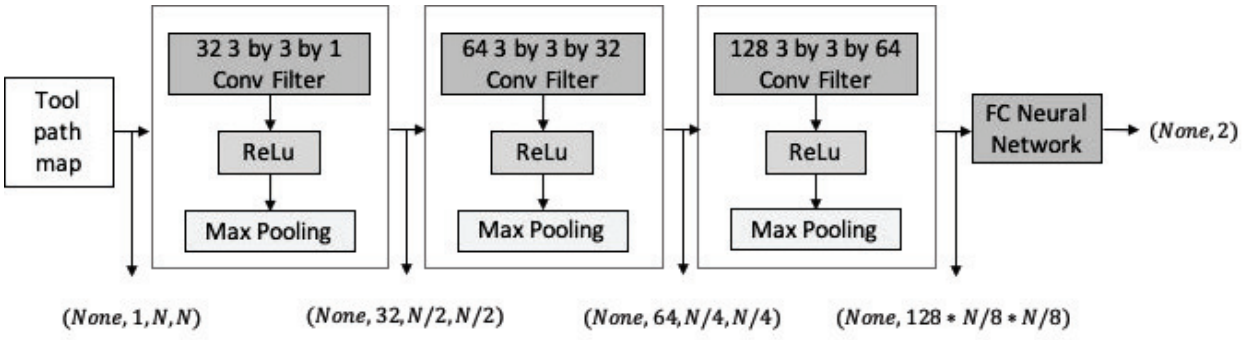


Figure 4.14: CNN architecture

- Convolution filter size: (3,3)
- Pooling: max-pooling, (2,2)
- Padding: 1
- Stride: 1
- Learning rate: 0.001
- Training epochs: 25
- Batch size: 200
- The number of hidden layers in a FC neural network: 1
- The number of nodes in a hidden layer: 800
- Activation function: ReLu (Rectified Linear Unit)
- Weight initialization: Xavier uniform
- Loss function: cross entropy loss

- Optimization method: Adam optimizer

After passing through the CNN, we used the softmax function to convert neural network output numbers to probabilities (of being either a good path or a bad path) for each image. We not only used the softmax function to calculate the cross entropy loss function while training, but also used it as a probability extractor to predict the probability of being a good path for each path map image in a test data set. The softmax probability function is described in equation 4.15.

$$P_\theta(y^{(m)} = i) = \frac{e^{z_i^{(m)}}}{\sum_{k=0}^{1} e^{z_k^{(m)}}}, \tag{4.15}$$

where i is the label of either 1 (good) or 0 (bad), $z_i^{(m)}$ is the output number of m^{th} image data for the label i from the CNN, and $P_\theta(y^{(m)} = i)$ is the predicted probability for m^{th} image to have label i. Also, for multi-class classification, we used the cross entropy loss as in Equation 4.16.

$$L = -\frac{1}{N}\sum_{m=1}^{N}\sum_{i=0}^{1} P(y^{(m)} = i) \cdot log P_\theta(y^{(m)} = i), \tag{4.16}$$

where N refers to the number of data samples (path map images), and $P(y^{(m)} = i)$ is the actual probability (either 0 or 1) for m^{th} image to have label i.

In order to train the CNN with the cross entropy loss function, an Adam optimizer was used for the optimization, which is computationally efficient and has little memory requirements [153].

Also, note that the test data is brand new data for the trained model, which means they were completely isolated from the training data set and training process. The training set was **randomly** selected from all data sets.

4.9 Simulation Results

With the deep learning model described above, we predicted what the optimal laser path should be, by ranking them based on the probability of being a good path, as obtained by the softmax probability extractor in Equation 4.15. We compared the predicted results from the CNN model with that of the linear model. The configuration of the linear prediction model is shown below.

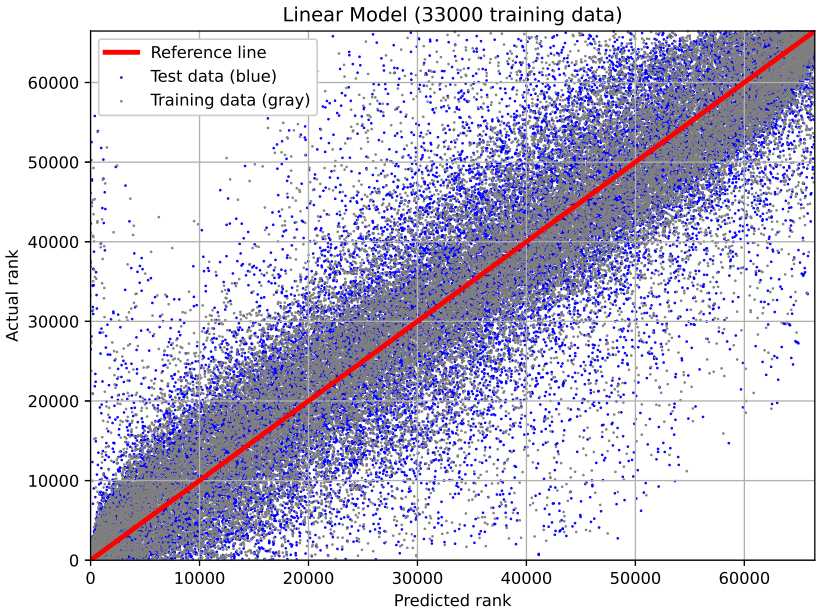
- Learning rate: 0.001
- Training epochs: 25

- Batch size: 200
- Weight initialization: Xavier uniform
- Loss function: cross entropy loss
- Optimization method: Adam optimizer

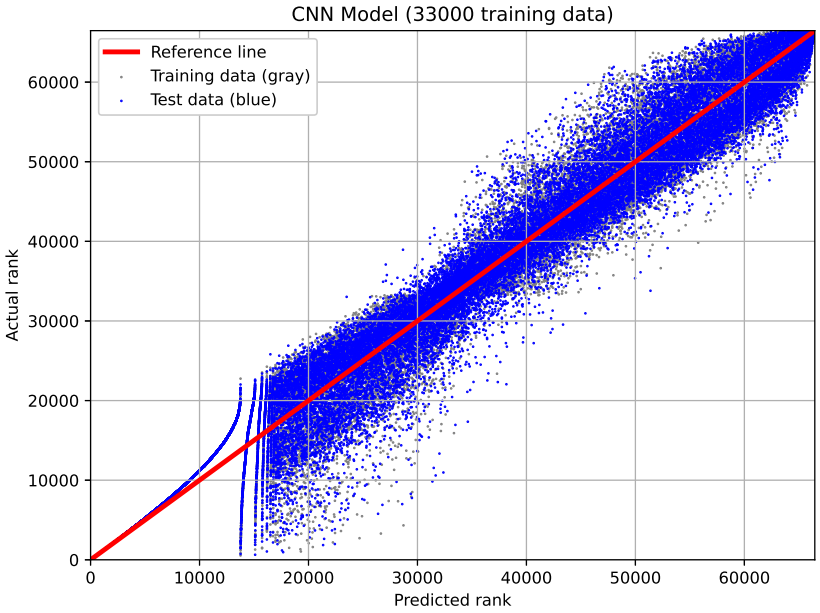
We first trained both the CNN model and the linear model with 33,000 point training data set (49.7% of 66464 total data points). In this case, the rest of the data (33464 paths) are what we actually make the predictions for. The ranking of the predicted results is shown in Figure 4.15. Also, a closeup on the top-ranked path for Figures 4.15a and 4.15b are shown in Figures 4.16a and 4.16b, respectively.

We observe that the linear model could not capture the actual optimal laser path well, as shown in Figures 4.15a and 4.16a. That is because the system is highly nonlinear. Even though the linear model is the least expensive machine learning model with the lowest computational cost, we need to account for the accuracy. However, we can see that the CNN model captures the highest-ranked laser path quite well, as shown in Figures 4.15b and 4.16b.

We also attempted to train both the CNN model and the linear model with quite small data sets: 4000 training data points (**only 6.02%** of 66464 total data points). In this case, the rest of the data (62464 paths, **93.98%**) are used to make the predictions. A ranking of the predicted results is shown in Figure 4.17. Also, a closeup on the top-ranked path for Figures 4.17a and 4.17b are shown in Figures 4.18a and 4.18b, respectively.

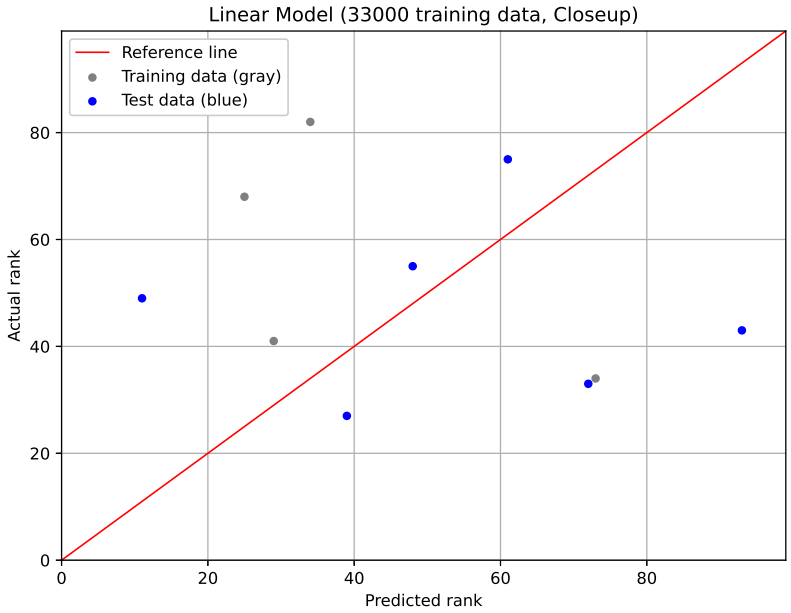


(a) Linear model

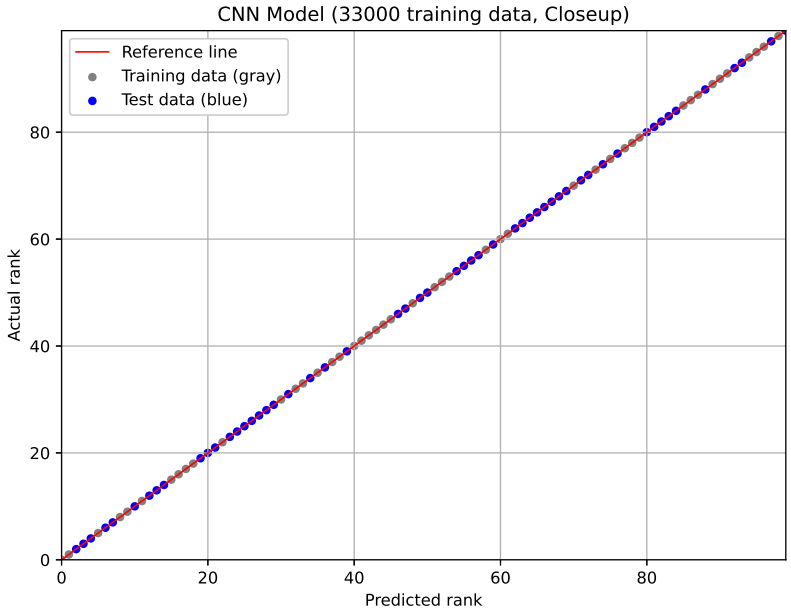


(b) CNN model

Figure 4.15: Ranking prediction (33000 training data)

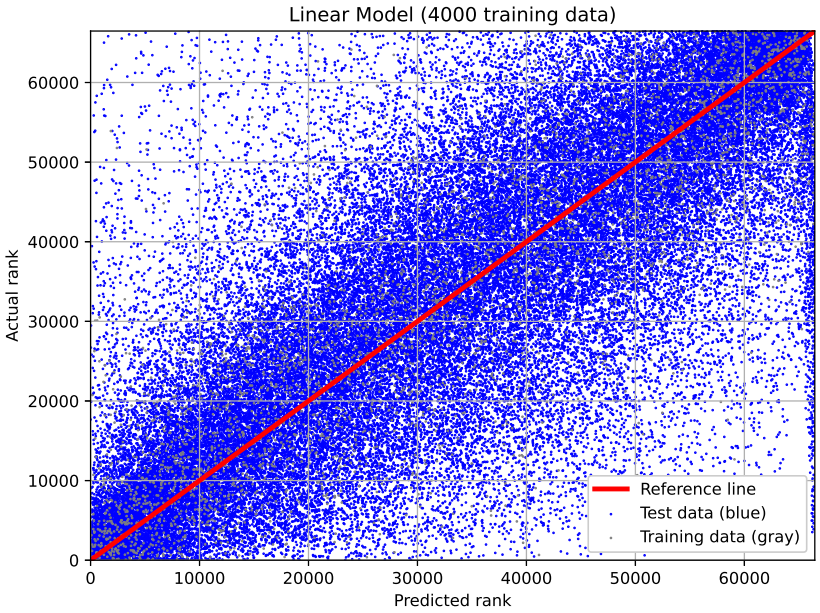


(a) Linear model

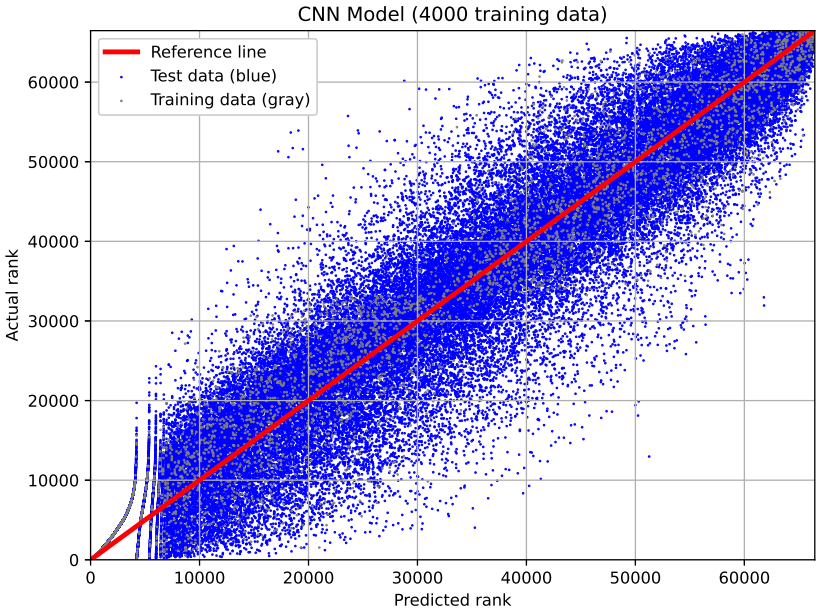


(b) CNN model

Figure 4.16: Ranking prediction (33000 training data, Closeup)

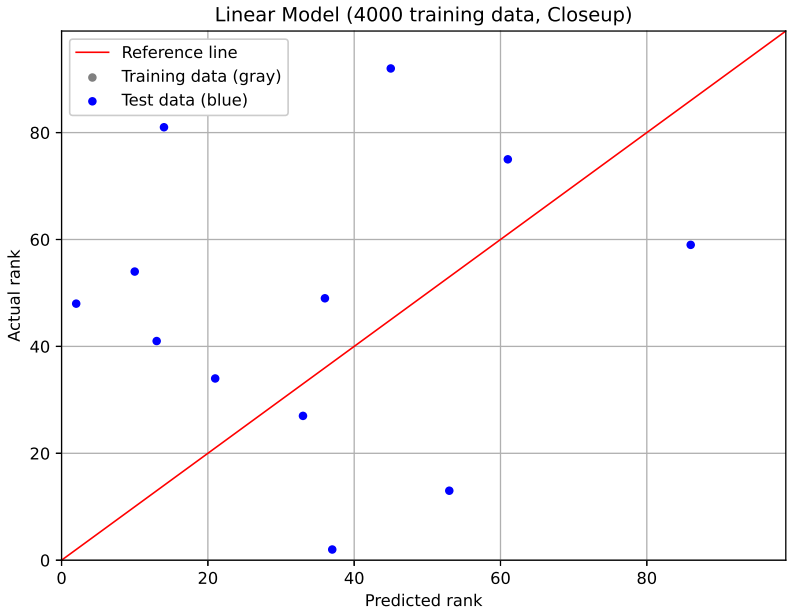


(a) Linear model

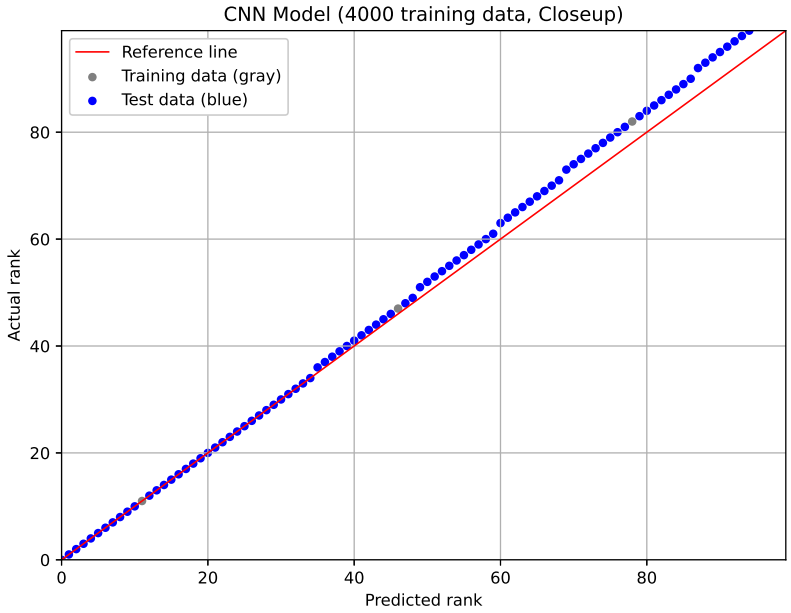


(b) CNN model

Figure 4.17: Ranking prediction (4000 training data)



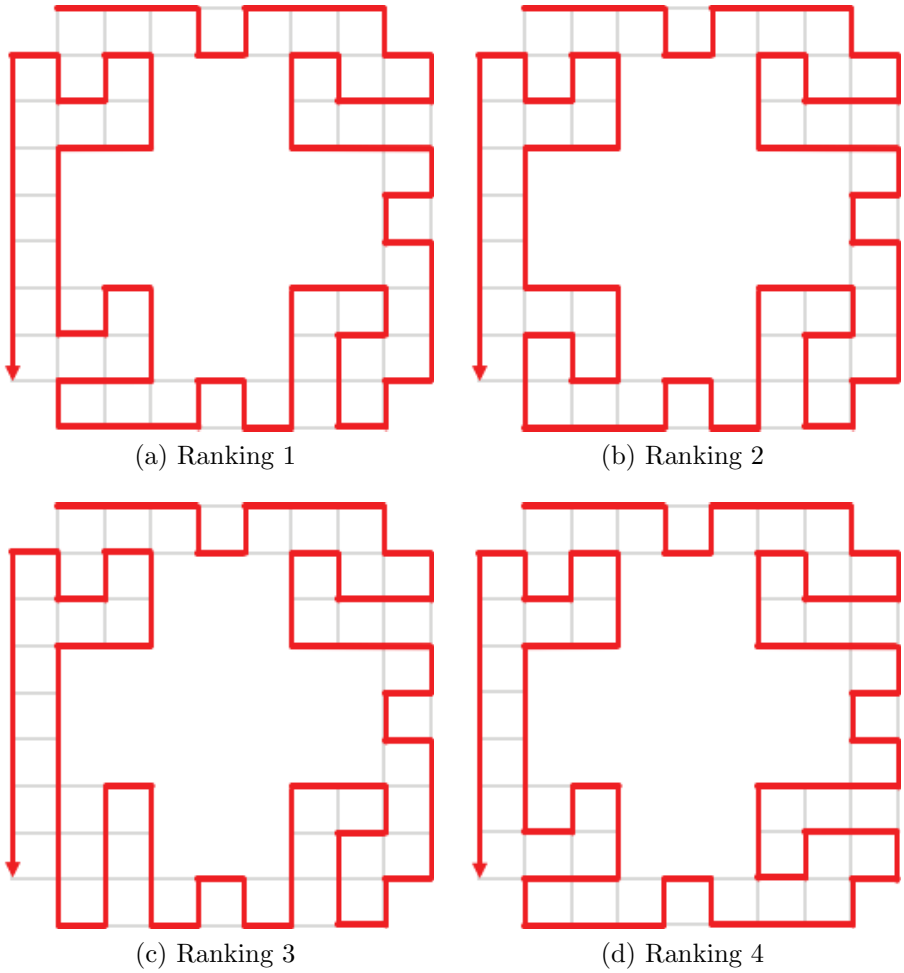
(a) Linear model



(b) CNN model

Figure 4.18: Ranking prediction (4000 training data, Closeup)

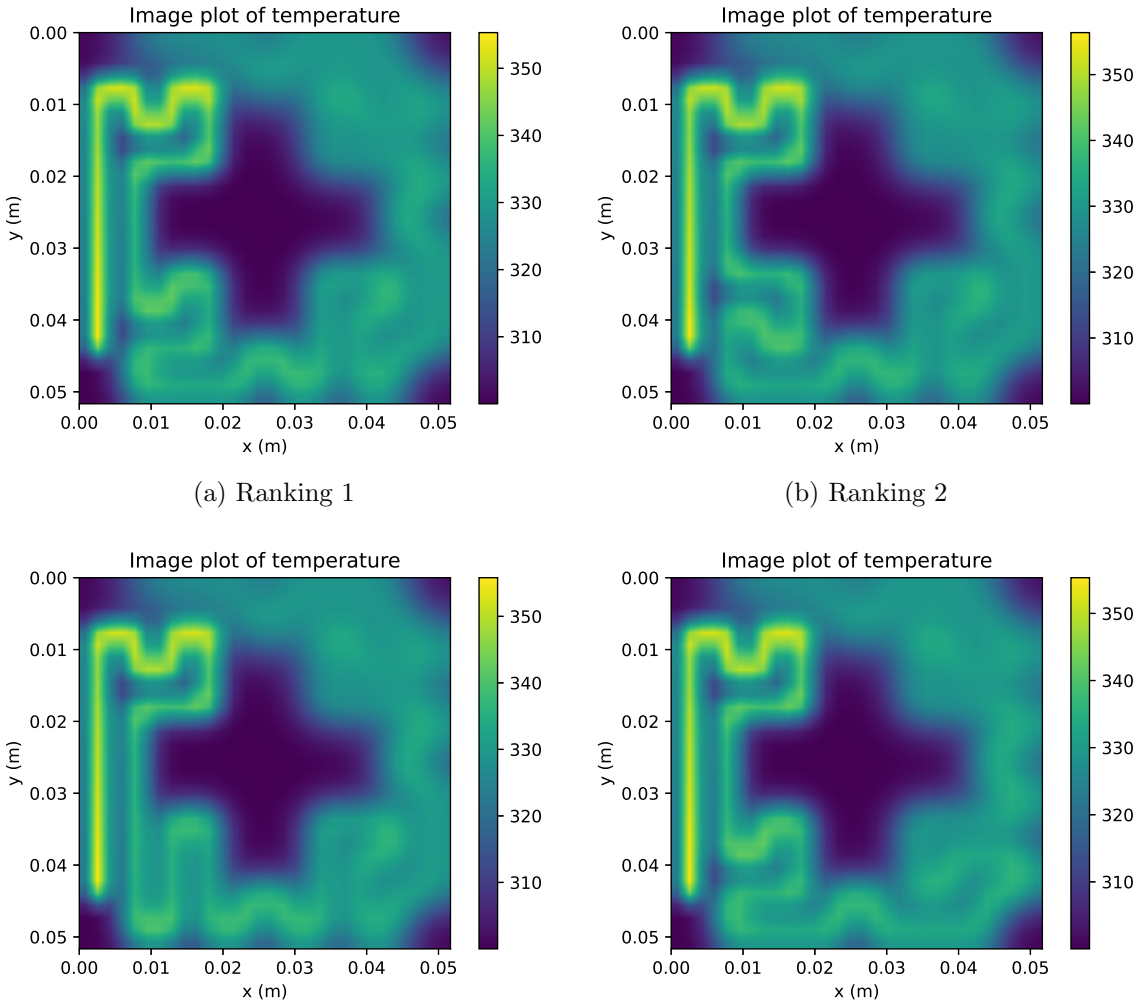
We clearly observe that the linear model could not capture the actual optimal laser path pattern, as shown in Figures 4.17a and 4.18a. This implies that the linear model becomes less useful when the data set is even small. However, the CNN model still captures the high-ranked laser paths reasonably well, in spite of a much smaller training set, as shown in Figures 4.17b and 4.18b. The CNN model is quite successful in capturing the top 35 paths, even though 34 paths of them were from the test data set. The top 4 paths which were chosen by the linear model are shown in Figure 4.19, and the corresponding plots of the temperature at the top surface of the powder bed are shown in Figure 4.20. Also, the top 4 paths which were chosen by the CNN model are shown in Figure 4.21, and the corresponding plots of the temperature at the top surface of the powder bed are shown in Figure 4.22.



(a) Ranking 1 (b) Ranking 2

(c) Ranking 3 (d) Ranking 4

Figure 4.19: Top 4 paths of the linear model



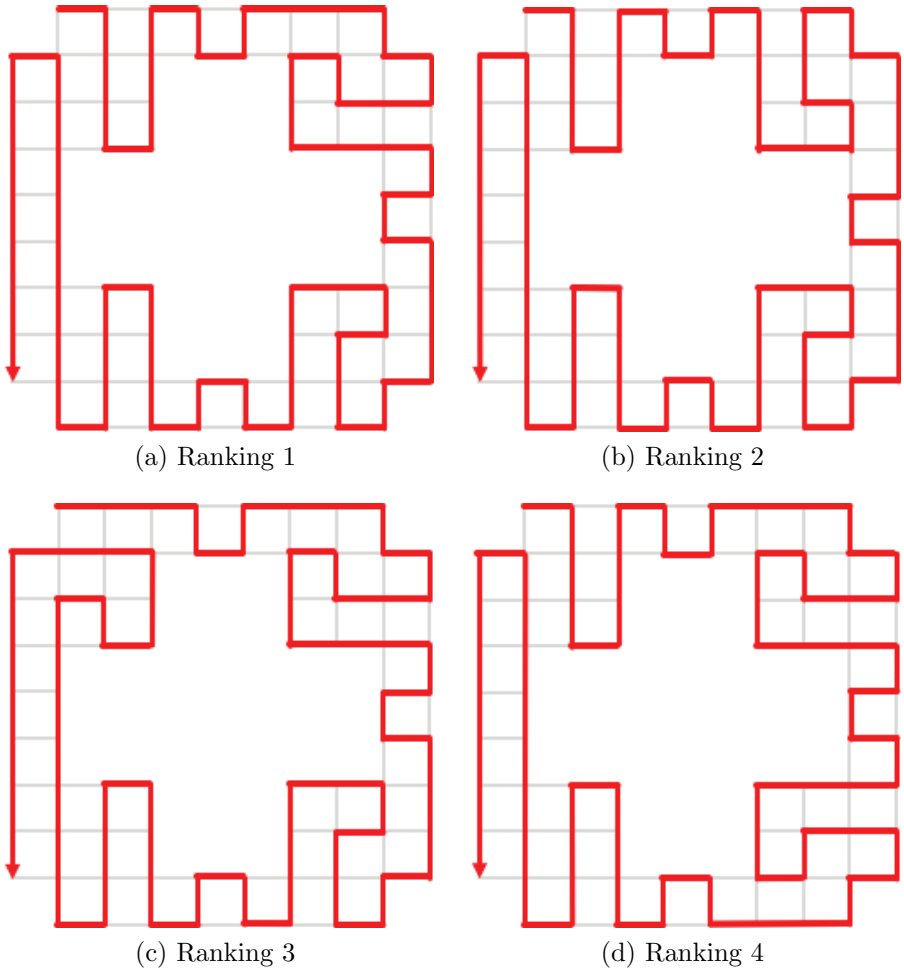
(a) Ranking 1

(b) Ranking 2

(c) Ranking 3

(d) Ranking 4

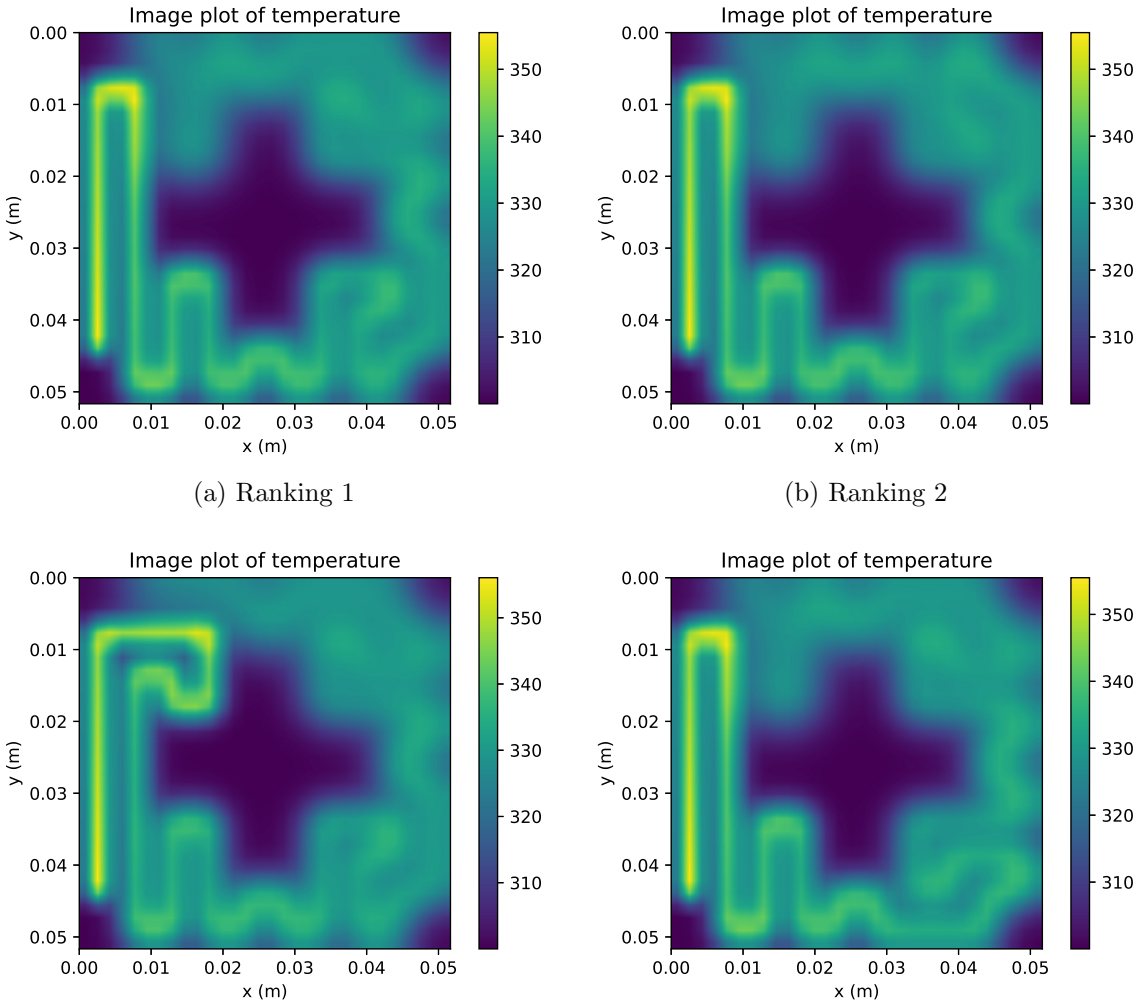
Figure 4.20: Temperature plots at the top surfaces (Top 4, linear model)



(a) Ranking 1 (b) Ranking 2

(c) Ranking 3 (d) Ranking 4

Figure 4.21: Top 4 paths of the CNN model



(a) Ranking 1

(b) Ranking 2

(c) Ranking 3

(d) Ranking 4

Figure 4.22: Temperature plots at the top surfaces (Top 4, CNN model)

The computing times for all the above models are shown in Table 4.2. The computer employed for the entire simulation was a *MacBook Pro (Retina, 15-inch, Mid 2014)*, and the corresponding CPU was a *2.8 GHz Intel Core i7*. The exhaustive simulation time for all the 66,464 laser paths was 38,109.53 (s). The total time represents the entire simulation time, including every single path simulation for the number of training data points, the training time, and the prediction time. The acceleration represents how many times faster it is than direct exhaustive simulations for 66,464 possible paths.

As we see in the table, the CNN model with 4000 training data (**only 6.02%** of 66464 total data) was more than 10 times faster than a brute force simulation for all the possible laser paths needed to calculate the thermal gradients, with the desired accuracy. This implies

Model	Training data	Training (s)	Prediction (s)	Total time (s)	Acceleration
CNN	33000	3872.44	747.45	23541.63	1.619
CNN	4000	473.13	752.91	3519.58	10.828
Linear	33000	12.58	0.52	18934.84	2.013
Linear	4000	1.98	0.47	2295.99	16.598

Table 4.2: Computation time comparison

that the deep learning model can learn the pattern of the preferable paths (which includes the process of solving differential equations) and predict the optimal laser paths accurately and efficiently, without knowing geometry and without having any mathematical or physical knowledge to solve differential equations and optimization problems.

4.10 The Overall Outlook

As this work illustrated, optimal laser paths can be accurately predicted using a Deep Learning technique, even with a very small amount of training data and binary information (good/bad path). Furthermore, a Deep Learning simulation using the CNN was significantly faster than a brute force simulation. These results illustrate the potential of Deep Learning for tool path optimization, in particular the ability to learn the patterns of tool paths and rebuild detailed path performances.

Clearly, Deep Learning techniques can be applied to many other fields of numerical simulations, well beyond simple tool path optimization for additive manufacturing, and deliver reduced simulation costs while ensuring desirable accuracy [187, 188]. However, there will always be trade-offs between computational costs and accuracy. A more computational effort may be required if we use complex and delicate deep learning models, yielding good predictive capabilities, as opposed to simple deep learning models, with limited predictive power. Determining the optimal balance between these competing interests is an ongoing issue throughout this field.

Bibliography

[1] John Tromp and Gunnar Farnebäck. "Combinatorics of go". In: *International Conference on Computers and Games*. Springer. 2006, pp. 84–99.

[2] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[3] *Amazon Echo Dot*. https://amazon.com/.

[4] Arthur L Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229.

[5] *What is machine learning?* https://lawtomated.com/a-i-technical-machine-vs-deep-learning/.

[6] *What is the difference between AI, machine learning and deep learning?* https://www.geospatialworld.net/blogs/difference-between-ai%EF%BB%BF-machine-learning-and-deep-learning/.

[7] *Machine Learning Crash Course*. https://developers.google.com/machine-learning/crash-course/.

[8] *Linear Regression using Gradient Descent*. https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931.

[9] *Stanford CS231n - Fei-Fei Li, et al.* http://cs231n.stanford.edu/.

[10] *Concept of Overfitting and Underfitting*. https://medium.com/@0885angjain/concept-of-overfitting-and-underfitting-353d73cf4a8.

[11] Steve Smale and Ding-Xuan Zhou. "Learning theory estimates via integral operators and their approximations". In: *Constructive approximation* 26.2 (2007), pp. 153–172.

[12] *Deep Learning Basics - Lecture 4: Regularization II*. https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/DL_lecture4_regularization_II.pdf.

[13] David M Allen. "The relationship between variable selection and data agumentation and a method for prediction". In: *technometrics* 16.1 (1974), pp. 125–127.

[14] Mervyn Stone. "Cross-validatory choice and assessment of statistical predictions". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 36.2 (1974), pp. 111–133.

[15] *Early stopping*. https://www.oreilly.com/library/view/hands-on-transfer-learning/9781788831307/41172567-9482-4cad-ac87-1cfbd46026df.xhtml.

[16] Peter Bühlmann and Sara Van De Geer. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media, 2011.

[17] *Regularization in Machine Learning*. https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a.

[18] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[19] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[20] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[21] *Understand Neural Networks & Model Generalization*. https://towardsdatascience.com/understand-neural-networks-model-generalization-7baddf1c48ca.

[22] Ethem Alpaydin. *Introduction to Machine Learning. [Sl]*. 2010.

[23] *Why not Mean Squared Error(MSE) as a loss function for Logistic Regression?* https://towardsdatascience.com/why-not-mse-as-a-loss-function-for-logistic-regression-589816b5e03c.

[24] David H Hubel and Torsten N Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1 (1962), p. 106.

[25] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[26] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[27] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[28] MV Valueva et al. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation". In: *Mathematics and Computers in Simulation* (2020).

[29] *A Comprehensive Guide to Convolutional Neural Networks*. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[31] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[33] *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[34] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[35] Felix A Gers and Jürgen Schmidhuber. "Recurrent nets that time and count". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Vol. 3. IEEE. 2000, pp. 189–194.

[36] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[37] Mark A Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243.

[38] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines". In: *Cognitive science* 9.1 (1985), pp. 147–169.

[39] Geoffrey E Hinton. "Boltzmann machine". In: *Scholarpedia* 2.5 (2007), p. 1668.

[40] Geoffrey E Hinton. "Deep belief networks". In: *Scholarpedia* 4.5 (2009), p. 5947.

[41] Diederik P Kingma and Max Welling. "An introduction to variational autoencoders". In: *arXiv preprint arXiv:1906.02691* (2019).

[42] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[43] Yunjey Choi et al. "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8789–8797.

[44] *Deep Learning for Natural Language Processing*. http://cs224d.stanford.edu/.

[45] *Machine Learning - Andrew Ng*. https://www.coursera.org/learn/machine-learning.

[46] *Deep Learning Zero To All (Korean)*. https://github.com/deeplearningzerotoall.

[47] *Deep Learning Tutorial*. http://deeplearning.stanford.edu/tutorial/.

[48] *Andrej Karpathy's YouTube*. https://www.youtube.com/channel/UCPk8m_r6fkUSYmvgCBwq-sw.

[49] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems*. 2019, pp. 8026–8037.

[50] Martín Abadi et al. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[51] *PyTorch*. https://pytorch.org/.

[52] *TensorFlow*. https://www.tensorflow.org/.

[53] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 281–305.

[54] Jost Tobias Springenberg et al. "Bayesian optimization with robust Bayesian neural networks". In: *Advances in neural information processing systems*. 2016, pp. 4134–4142.

[55] Eric Brochu, Vlad M Cora, and Nando De Freitas. "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv preprint arXiv:1012.2599* (2010).

[56] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[57] Jason Brownlee. *Statistical methods for machine learning: Discover how to transform data into knowledge with Python*. Machine Learning Mastery, 2018.

[58] Bobak Shahriari et al. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[59] *How to Implement Bayesian Optimization from Scratch in Python*. https://machinelearningmastery.com/what-is-bayesian-optimization/.

[60] *Fernando Nogueira, bayesian-optimization: A Python implementation of global optimization with gaussian processes*. https://github.com/fmfn/BayesianOptimization.

[61] *Cognex Deep Learning Lab-KOR Research Blog*. https://research.sualab.com/.

[62] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[63] Peter I Frazier. "A tutorial on bayesian optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[64] *Expected Improvement for Bayesian Optimization: A Derivation*. http://ash-aldujaili.github.io/blog/2018/02/01/ei/.

[65] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. "The application of Bayesian methods for seeking the extremum". In: *Towards global optimization* 2.117-129 (1978), p. 2.

[66] Daniel James Lizotte. *Practical bayesian optimization*. University of Alberta, 2008.

[67] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[68] David E Goldberg and John Henry Holland. "Genetic algorithms and machine learning". In: (1988).

[69] TI Zohdi. "An explicit macro-micro phase-averaged stress correlation for particle-enhanced composite materials in loaded structures". In: *International Journal of Engineering Science* 109 (2016), pp. 1–13.

[70] Tarek I Zohdi. *Modeling and Simulation of Functionalized Materials for Additive Manufacturing and 3D Printing: Continuous and Discrete Media: Continuum and Discrete Element Methods*. Vol. 60. Springer, 2017.

[71] TI Zohdi. "Electrodynamic machine-learning-enhanced fault-tolerance of robotic free-form printing of complex mixtures". In: *Computational Mechanics* 63.5 (2019), pp. 913–929.

[72] TI Zohdi. "A machine-learning framework for rapid adaptive digital-twin based fire-propagation simulation in complex environments". In: *Computer Methods in Applied Mechanics and Engineering* 363 (2020), p. 112907.

[73] TI Zohdi. "Genetic design of solids possessing a random–particulate microstructure". In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 361.1806 (2003), pp. 1021–1043.

[74] TI Zohdi. "Dynamic thermomechanical modeling and simulation of the design of rapid free-form 3D printing processes with evolutionary machine learning". In: *Computer Methods in Applied Mechanics and Engineering* 331 (2018), pp. 343–362.

[75] TI Zohdi. "The Game of Drones: Rapid agent-based machine-learning models for multi-UAV path planning". In: *Computational Mechanics* 65.1 (2020), pp. 217–228.

[76] Tarek I Zohdi. "CM Approaches: Estimation and Optimization of the Effective Properties of Mixtures". In: *Modeling and Simulation of Functionalized Materials for Additive Manufacturing and 3D Printing: Continuous and Discrete Media*. Springer, 2018, pp. 31–42.

[77] Z Hashin and S Shtrikman. "On some variational principles in anisotropic and nonhomogeneous elasticity". In: *Journal of the Mechanics and Physics of Solids* 10.4 (1962), pp. 335–342.

[78] Zvi Hashin and Shmuel Shtrikman. "A variational approach to the theory of the elastic behaviour of multiphase materials". In: *Journal of the Mechanics and Physics of Solids* 11.2 (1963), pp. 127–140.

[79] Zvi Hashin. "Analysis of composite materials—a survey". In: (1983).

[80] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95-International Conference on Neural Networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.

[81] Yuhui Shi and Russell Eberhart. "A modified particle swarm optimizer". In: *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*. IEEE. 1998, pp. 69–73.

[82] Zeineb Abdmouleh et al. "Review of optimization techniques applied for the integration of distributed generation from renewable energy sources". In: *Renewable Energy* 113 (2017), pp. 266–280.

[83] István Erlich, Ganesh K Venayagamoorthy, and Nakawiro Worawat. "A mean-variance optimization algorithm". In: *IEEE Congress on Evolutionary Computation*. IEEE. 2010, pp. 1–6.

[84] *Mean Variance Mapping Optimization Algorithm*. https://pypi.org/project/MVMO/.

[85] Jaime C Cepeda et al. "Mean–Variance Mapping Optimization Algorithm for Power System Applications in DIgSILENT PowerFactory". In: *PowerFactory Applications for Power System Analysis*. Springer, 2014, pp. 267–295.

[86] Neil A Thacker and Timothy F Cootes. "Vision through optimization". In: *BMVC Tutorial Notes* (1996).

[87] Andrew Blake and Andrew Zisserman. *Visual reconstruction*. MIT press, 1987.

[88] Hossein Mobahi and John W Fisher. "On the link between gaussian homotopy continuation and convex envelopes". In: *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer. 2015, pp. 43–56.

[89] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization". In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[90] Son Duy Dao, Kazem Abhary, and Romeo Marian. "An innovative framework for designing genetic algorithm structures". In: *Expert Systems with Applications* 90 (2017), pp. 196–208.

[91] Son Duy Dao, Kazem Abhary, and Romeo Marian. "Maximising Performance of Genetic Algorithm Solver in Matlab." In: *Engineering Letters* 24.1 (2016).

[92] *Solving Optimization Problems*. https://learnwithpanda.com.

[93] Dervis Karaboga. *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Technical report-tr06, Erciyes university, engineering faculty, computer . . ., 2005.

[94] Xin-She Yang and Suash Deb. "Cuckoo search via Lévy flights". In: *2009 World congress on nature & biologically inspired computing (NaBIC)*. IEEE. 2009, pp. 210–214.

[95] Paul E Black. "Greedy algorithm". In: *Dictionary of Algorithms and Data Structures* 2 (2005), p. 62.

[96] Martin Pincus. "Letter to the editor—a Monte Carlo method for the approximate solution of certain types of constrained optimization problems". In: *Operations research* 18.6 (1970), pp. 1225–1228.

[97] Fred Glover. "Future paths for integer programming and links to ar tifi cial intelli g en ce". In: *Computers operations research* 13.5 (1986), pp. 533–549.

[98] Fred Glover and Manuel Laguna. "Tabu search". In: *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[99] Kaisa Miettinen. *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media, 2012.

[100] Matthias Ehrgott. *Multicriteria optimization*. Vol. 491. Springer Science & Business Media, 2005.

[101] Philip E Gill, Walter Murray, and Margaret H Wright. *Practical optimization*. SIAM, 2019.

[102] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[103] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Vol. 28. Princeton University Press, 2009.

[104] Kalyanmoy Deb. "Multi-objective optimization". In: *Search methodologies*. Springer, 2014, pp. 403–449.

[105] Singiresu S Rao. *Engineering optimization: theory and practice*. John Wiley & Sons, 2019.

[106] R Paul Singh and Dennis R Heldman. *Introduction to food engineering*. Gulf Professional Publishing, 2001.

[107] *Beyond Meat*. https://www.beyondmeat.com/.

[108] *Impossible Foods*. https://impossiblefoods.com/.

[109] Alessandro Leonardi et al. "Coupled DEM-LBM method for the free-surface simulation of heterogeneous suspensions". In: *Computational Particle Mechanics* 1.1 (2014), pp. 3–13.

[110] Eugenio Oñate et al. "Lagrangian analysis of multiscale particulate flows with the particle finite element method". In: *Computational Particle Mechanics* 1.1 (2014), pp. 85–102.

[111] B Avci and P Wriggers. "A DEM-FEM coupling approach for the direct numerical simulation of 3D particulate flows". In: *Journal of Applied Mechanics* 79.1 (2012).

[112] T Zohdi. "Embedded electromagnetically sensitive particle motion in functionalized fluids". In: *Computational Particle Mechanics* 1.1 (2014), pp. 27–45.

[113] Tarek I Zohdi and Peter Wriggers. *An introduction to computational micromechanics*. Springer Science & Business Media, 2008.

[114] Tarek I Zohdi. "On simple scaling laws for pumping fluids with electrically-charged particles". In: *International Journal of Engineering Science* 123 (2018), pp. 73–80.

[115] Osborne Reynolds. "XXIX. An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels". In: *Philosophical Transactions of the Royal society of London* 174 (1883), pp. 935–982.

[116] Hosahalli S Ramaswamy et al. *Ohmic heating in food processing*. CRC press, 2014.

[117] Peter John Fellows. *Food processing technology: principles and practice*. Elsevier, 2009.

[118] K Shiby Varghese et al. "Technology, applications and modelling of ohmic heating: a review". In: *Journal of food science and technology* 51.10 (2014), pp. 2304–2317.

[119] Albert Einstein. "A new determination of molecular dimensions". In: *Ann. Phys.* 19 (1906), pp. 289–306.

[120] Salvatore Torquato and HW Haslach Jr. "Random heterogeneous materials: microstructure and macroscopic properties". In: *Appl. Mech. Rev.* 55.4 (2002), B62–B63.

[121] Mark Kachanov and Behrouz Abedian. "On the isotropic and anisotropic viscosity of suspensions containing particles of diverse shapes and orientations". In: *International Journal of Engineering Science* 94 (2015), pp. 71–85.

[122] Igor Sevostianov and Mark Kachanov. "Effective properties of heterogeneous materials: Proper application of the non-interaction and the "dilute limit" approximations". In: *International Journal of Engineering Science* 58 (2012), pp. 124–128.

[123] C Sandu and RK Singh. "Energy increases in operation and cleaning due to heat-exchanger fouling in milk pasteurization". In: *Food technology (Chicago)* 45.12 (1991), pp. 84–91.

[124] J Visser and Th JM Jeurnink. "Fouling of heat exchangers in the dairy industry". In: *Experimental Thermal and Fluid Science* 14.4 (1997), pp. 407–424.

[125] H Petermeier et al. "Hybrid model of the fouling process in tubular heat exchangers for the dairy industry". In: *Journal of Food Engineering* 55.1 (2002), pp. 9–17.

[126] Hatice Ozlem Ozden and Virendra M Puri. "Computational analysis of fouling by low energy surfaces". In: *Journal of food engineering* 99.3 (2010), pp. 250–256.

[127] Adel Fickak, Ali Al-Raisi, and Xiao Dong Chen. "Effect of whey protein concentration on the fouling and cleaning of a heat transfer surface". In: *Journal of food engineering* 104.3 (2011), pp. 323–331.

[128] W Ebert and CB Panchal. *Analysis of Exxon crude-oil-slip stream coking data*. Tech. rep. Argonne National Lab., IL (United States), 1995.

[129] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[130] E Hairer, S Paul Nørsett, and G Wanner. "Solving Ordinary Differential Equations I, Nonstiff Problems. 1993". In: *Springer-Verlag, Berlin, DOI* 10 (), pp. 978–3.

[131] Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*. Springer Berlin Heidelberg, 1996.

[132] Arieh Iserles. *A first course in the numerical analysis of differential equations*. 44. Cambridge university press, 2009.

[133] DR Oliver and Stacey G Ward. "Relationship between relative viscosity and volume concentration of stable suspensions of spherical particles". In: *Nature* 171.4348 (1953), pp. 396–397.

[134] TI Zohdi. "An upper bound on the particle-laden dependency of shear stresses at solid–fluid interfaces". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2211 (2018), p. 20170332.

[135] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*. Vol. 16. John Wiley & Sons, 2001.

[136] Mohamed G Sahab, Vassili V Toropov, and Amir H Gandomi. "Optimum Design of Composite Concrete Floors Using a Hybrid Genetic Algorithm". In: *Handbook of Neural Computation*. Elsevier, 2017, pp. 581–589.

[137] RS Mitchell, JG Michalski, and TM Carbonell. *An artificial intelligence approach*. Springer, 2013.

[138] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[139] Pat Langley et al. "Selection of relevant features in machine learning". In: *Proceedings of the AAAI Fall symposium on relevance*. Vol. 184. 1994, pp. 245–271.

[140] Igor Kononenko and Matjaz Kukar. *Machine learning and data mining*. Horwood Publishing, 2007.

[141] David J Lary et al. "Machine learning in geosciences and remote sensing". In: *Geoscience Frontiers* 7.1 (2016), pp. 3–10.

[142] Grace X Gu, Chun-Teh Chen, and Markus J Buehler. "De novo composite design based on machine learning algorithm". In: *Extreme Mechanics Letters* 18 (2018), pp. 19–28.

[143] Chun-Teh Chen and Grace X Gu. "Machine learning for composite materials". In: *MRS Communications* 9.2 (2019), pp. 556–566.

[144] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[145] Thiago S Guzella and Walmir M Caminhas. "A review of machine learning approaches to spam filtering". In: *Expert Systems with Applications* 36.7 (2009), pp. 10206–10222.

[146] Igor Santos et al. "Machine-learning-based mechanical properties prediction in foundry production". In: *2009 ICCAS-SICE*. IEEE. 2009, pp. 4536–4541.

[147] Arun Mannodi-Kanakkithodi et al. "Machine learning strategy for accelerated design of polymer dielectrics". In: *Scientific reports* 6 (2016), p. 20952.

[148] Ghanshyam Pilania et al. "Data-Based Methods for Materials Design and Discovery: Basic Ideas and General Methods". In: *Synthesis Lectures on Materials and Optics* 1.1 (2020), pp. 1–188.

[149] Masoud Sarveghadi et al. "Development of prediction models for shear strength of SFRCB using a machine learning approach". In: *Neural Computing and Applications* 31.7 (2019), pp. 2085–2094.

[150] Florent Pled et al. "Neural network prediction of cortical bone damage using a stochastic computational mechanical model". In: *3rd International Conference on Uncertainty Quantification in Computational Sciences and Engineering (UNCECOMP 2019)*. 2019.

[151] Christian Soize. "A probabilistic learning on manifolds as a new tool in machine learning and data science with applications in computational mechanics". In: *UNCECOMP 2019, 3nd International Conference on Uncertainty Quantification in Computational Sciences and Engineering, and COMPDYN 2019, 7th International Conference on Computational Methods in Structural Dynamics and Earthquake Engineering*. 2019.

[152] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[153] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[154] Saleh Ahmed Aldahash. "Optimum manufacturing parameters in selective laser sintering of PA12 with white cement additives". In: *The International Journal of Advanced Manufacturing Technology* 96.1-4 (2018), pp. 257–270.

[155] Kaufui V Wong and Aldo Hernandez. "A review of additive manufacturing". In: *International scholarly research notices* 2012 (2012).

[156] William E Frazier. "Metal additive manufacturing: a review". In: *Journal of Materials Engineering and performance* 23.6 (2014), pp. 1917–1928.

[157] Tuan D Ngo et al. "Additive manufacturing (3D printing): A review of materials, methods, applications and challenges". In: *Composites Part B: Engineering* 143 (2018), pp. 172–196.

[158] Harry Bikas, Panagiotis Stavropoulos, and George Chryssolouris. "Additive manufacturing methods and modelling approaches: a critical review". In: *The International Journal of Advanced Manufacturing Technology* 83.1-4 (2016), pp. 389–405.

[159] Sunpreet Singh, Seeram Ramakrishna, and Rupinder Singh. "Material issues in additive manufacturing: A review". In: *Journal of Manufacturing Processes* 25 (2017), pp. 185–200.

[160] John J Lewandowski and Mohsen Seifi. "Metal additive manufacturing: a review of mechanical properties". In: *Annual review of materials research* 46 (2016).

[161] William M Steen and Jyotirmoy Mazumder. *Laser material processing*. springer science & business media, 2010.

[162] Kai Zeng, Deepankar Pal, and Brent Stucker. "A review of thermal analysis methods in laser sintering and selective laser melting". In: *Proceedings of Solid Freeform Fabrication Symposium Austin, TX*. Vol. 60. 2012, pp. 796–814.

[163] L Dong et al. "Three-dimensional transient finite element analysis of the selective laser sintering process". In: *Journal of materials processing technology* 209.2 (2009), pp. 700–706.

[164] Serguei Kolossov et al. "3D FE simulation for temperature evolution in the selective laser sintering process". In: *International Journal of Machine Tools and Manufacture* 44.2-3 (2004), pp. 117–123.

[165] M Matsumoto et al. "Finite element analysis of single layer forming on metallic powder bed in rapid prototyping by selective laser processing". In: *International Journal of Machine Tools and Manufacture* 42.1 (2002), pp. 61–67.

[166] A Simchi. "Direct laser sintering of metal powders: Mechanism, kinetics and microstructural features". In: *Materials Science and Engineering: A* 428.1-2 (2006), pp. 148–158.

[167] A Simchi and H Pohl. "Effects of laser sintering processing parameters on the microstructure and densification of iron powder". In: *Materials Science and Engineering: A* 359.1-2 (2003), pp. 119–128.

[168] AV Gusarov and J-P Kruth. "Modelling of radiation transfer in metallic powders at laser treatment". In: *International Journal of Heat and Mass Transfer* 48.16 (2005), pp. 3423–3434.

[169] AV Gusarov et al. "Model of radiation and heat transfer in laser-powder interaction zone at selective laser melting". In: *Journal of heat transfer* 131.7 (2009).

[170] Rishi Ganeriwala and Tarek I Zohdi. "Multiphysics modeling and simulation of selective laser sintering manufacturing processes". In: *Procedia Cirp* 14 (2014), pp. 299–304.

[171] Tarek I Zohdi. "Rapid simulation of laser processing of discrete particulate materials". In: *Archives of Computational Methods in Engineering* 20.4 (2013), pp. 309–325.

[172] Tarek I Zohdi. "Additive particle deposition and selective laser processing-a computational manufacturing framework". In: *Computational Mechanics* 54.1 (2014), pp. 171–191.

[173] TI Zohdi. "A direct particle-based computational framework for electrically enhanced thermo-mechanical sintering of powdered materials". In: *Mathematics and Mechanics of Solids* 19.1 (2014), pp. 93–113.

[174] Rishi Ganeriwala and Tarek I Zohdi. "A coupled discrete element-finite difference model of selective laser sintering". In: *Granular Matter* 18.2 (2016), p. 21.

[175] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[176] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*. Vol. 31. Cambridge university press, 2002.

[177] Tarek I Zohdi, Zohdi, and Ditzinger. *A Finite Element Primer for Beginners*. Springer, 2018.

[178] Dimitri P Bertsekas et al. *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA, 1995.

[179] Richard Bellman. "Dynamic programming". In: *Science* 153.3731 (1966), pp. 34–37.

[180] Ronald A Howard. "Dynamic programming and markov processes." In: (1960).

[181] Richard E Bellman and Stuart E Dreyfus. *Applied dynamic programming*. Princeton university press, 2015.

[182] Jeff Hecht. *Understanding lasers: an entry-level guide*. John Wiley & Sons, 2018.

[183] Richard Courant, Kurt Friedrichs, and Hans Lewy. "On the partial difference equations of mathematical physics". In: *IBM journal of Research and Development* 11.2 (1967), pp. 215–234.

[184] Carlos A De Moura and Carlos S Kubrusly. "The Courant–Friedrichs–Lewy (CFL) Condition". In: *AMC* 10 (2013), p. 12.

[185] John Crank and Phyllis Nicolson. "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type". In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 43. 1. Cambridge University Press. 1947, pp. 50–67.

[186] Yongtae Kim et al. "Designing adhesive pillar shape with deep learning-based optimization". In: *ACS Applied Materials & Interfaces* (2020).

[187] Atsuya Oishi and Genki Yagawa. "Computational mechanics enhanced by deep learning". In: *Computer Methods in Applied Mechanics and Engineering* 327 (2017), pp. 327–351.

[188] Jackson K Wilt, Charles Yang, and Grace X Gu. "Accelerating Auxetic Metamaterial Design with Deep Learning". In: *Advanced Engineering Materials* (2020), p. 1901266.