Predicting Frictional Properties of Graphene Kirigami Using Molecular Dynamics and Neural Networks

Designs for a negative friction coefficient.

Mikkel Metzsch Jensen

Thesis submitted for the degree of
Master in Computational Science: Materials Science
60 credits

Department of Physics
Faculty of mathematics and natural sciences

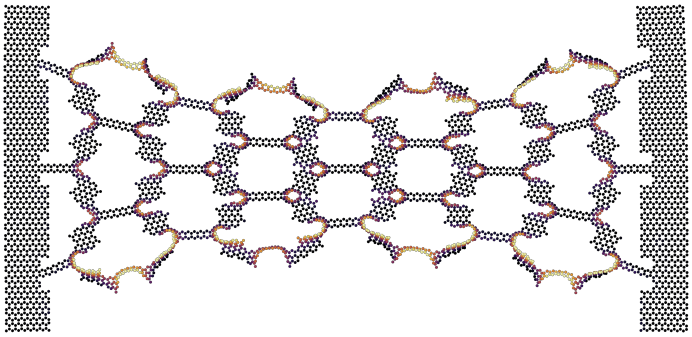
UNIVERSITY OF OSLO

Spring 2023

Predicting Frictional Properties of Graphene Kirigami Using Molecular Dynamics and Neural Networks

Designs for a negative friction coefficient.

Mikkel Metzsch Jensen



Predicting Frictional Properties of Graphene Kirigami Using Molecular Dynamics and Neural Networks

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Basic introduction

Various theoretical models and experimental results propose different governing mechanisms for friction at the nanoscale.

More detailed background

We consider a graphene sheet modified with Kirigami-inspired cuts and under the influence of strain. Prior research has demonstrated that this system exhibits out-of-plane buckling, which could result in a decrease in contact area when sliding on a substrate. According to asperity theory, this decrease in contact area is expected to lead to a reduction of friction.

General problem

However, to the best of our knowledge, no previous studies have investigated the friction behavior of a nanoscale Kirigami graphene sheet under strain.

Summarize main result: "here we show"

Here we show that specific Kirigami designs yield a non-linear dependency between kinetic friction and the strain of the sheet.

General context

Using molecular dynamics simulation, we have found a non-monotonic increase in friction with strain. We found that the friction-strain relationship does not show any clear dependency on contact area which contradicts asperity theory. Our findings suggest that the effect is associated with the out-of-plane buckling of the graphene sheet and we attribute this to a commensurability effect. By mimicking a load-strain coupling through tension, we were able to utilize this effect to demonstrate a negative friction coefficient on the order of -0.3 for loads in the range of a few nN. In addition, we have attempted to use machine learning to capture the relationship between Kirigami designs, load, and strain, with the objective of performing an accelerated search for new designs. While this approach produced some promising results, we conclude that further improvements to the dataset are necessary in order to develop a reliable model.

Broader perspective

We anticipate our findings to be a starting point for further investigations of the underlying mechanism for the frictional behavior of a Kirigami sheet. For instance, the commensurability hypothesis could be examined by varying the sliding angle in simulations. We propose to use an active learning strategy to extend the dataset for the use of machine learning to assist these investigations. If successful, further studies can be done on the method of inverse design. In summary, our findings suggest that the application of nanoscale Kirigami can be promising for developing novel friction-control strategies.

353 words

Various numerical models and experimental results propose different governing mechanisms for friction at the nanoscale. We consider a graphene sheet modified with Kirigami-inspired cuts and under the influence of strain. Prior research has demonstrated that this system exhibits out-of-plane buckling, which could result in a decrease in contact area when sliding on a substrate. According to asperity theory, this decrease in contact area is expected to lead to a reduction of friction. However, to the best of our knowledge, no previous studies have investigated the friction behavior of a nanoscale Kirigami graphene sheet under strain. Here we show that specific Kirigami designs yield a non-linear dependency between kinetic friction and the strain of the sheet. Using molecular dynamics simulation, we have found a non-monotonic increase in friction with strain. We found that the friction-strain relationship does not show any clear dependency on contact area which contradicts asperity theory. Our findings suggest that the effect is associated with the out-of-plane buckling of the graphene sheet and we attribute this to a commensurability effect. By mimicking a load-strain coupling through tension, we were able to utilize this effect to demonstrate a negative friction coefficient on the order of -0.3 for loads in the range of a few nN. In addition, we have attempted to use machine learning to capture the relationship between Kirigami designs, load, and strain, with the objective of performing an accelerated search for new designs. While this approach produced some promising results, we conclude that further improvements to the dataset are necessary in order to develop a reliable model. We anticipate our findings to be a starting point for further investigations of the underlying mechanism for the frictional behavior of a Kirigami sheet. For instance, the commensurability hypothesis could be examined by varying the sliding angle in simulations. We propose to use an active learning strategy to extend the dataset for the use of machine learning to assist these investigations. If successful, further studies can be done on the method of inverse design. In summary, our findings suggest that the application of nanoscale Kirigami can be promising for developing novel friction-control strategies.

Acknowledgments

The task of writing a master's thesis is a demanding and extensive project which I could not have been done without the support of many good people around me. First of all, I want to thank my supervisors Henrik Andersen Sveinsson and Anders Malthe-Sørenssen for the assistance in this thesis work. I am especially grateful for the weekly meetings with Henrik and the inspiring discussion being had as we unraveled the discoveries related to the topic of this thesis. I remember that I initially asked for an estimate of how much time he had available for supervision and the answer was something along the lines of "There are no limits really, just send me an email and we figure it out". This attitude captures the main experience I have had working with Henrik and I am profoundly grateful for the time and effort you have put into this project. I hope that you do not regret said statement, too much, because I have certainly been taken advantage of it. I also want to thank Even Marius Nordhagen for technical support regarding the use of the computational cluster, in times when the computer did exactly what it was told and was being silly. In that context, I also want to acknowledge the University of Oslo (CCSE?) for making these resources available.

I would like to express my gratitude to all the parties involved in making it possible for me to write my thesis from Italy. I am particularly grateful for the flexibility shown by my supervisors and for the support of Anders Kvellestad, who allowed me to work remotely as a group teacher. I would also like to thank Scuola Normale Superiore for providing me with access to their library.

I realize that it is a commonly used cliche to express gratitude for the support of loved ones. However, I want to highlight the exceptional role played by my fiancé, Ida. She deserves the main credit for helping me maintain a healthy state of mind, and she has provided me with a solid foundation for a fulfilling life that enables me to pursue secondary objectives, such as an academic career. I look forward to spending the rest of my life with you.

Throughout this thesis, I have used a formal "we" mainly as a customary habit related to the formalities of scientific writing in a team. However, I have come to the realization that this approach feels more appropriate since I have not been working on this project alone. I have found support all the way from colleagues and friends at the University of Oslo, to my family residing in Denmark, and my life partner sleeping beside me every night here in Italy. They constitute the "good people around me" who have made this thesis possible.

List of Symbols

F_N Normal force (normal load)

Acronyms

CNN Convolutional Neural Network. 12, 13, 14

EMA Exponetial Moving Average. 10

GA Genetic Algorithm. 17, 18

GAN Generative Adversarial Networks. 2, 27

MD Molecular Dynamics. 1, 2, 3, 4, 7, 23, 24, 25, 26

ML Machine Learning. 2

MSE Mean Squared Error. 8

RMSProp Root Mean Square Propagation. 10, 12

SGD Stochastic gradient descent. 9

Contents

Chapter 1

Introduction

1.1 Motivation

Friction is the force that prevents the relative motion of objects in contact. From our everyday life, we recognize it as the inherent resistance to sliding motion. Some surfaces appear slippery and some rough, and we know intuitively that sliding down a snow-covered hill is much more exciting than its grassy counterpart. Without friction, it would not be possible to walk across a flat surface, lean against the wall without falling over or secure an object by the use of nails or screws [p. 5] [1]. It is probably safe to say that the concept of friction is integrated into our everyday life to such an extent that most people take it for granted. However, the efforts to control friction date back to the early civilization (3500 B.C.) with the use of the wheel and lubricants to reduce friction in translational motion [2]. Today, friction is considered a part of the wider field *tribology* derived from the Greek word *tribos* meaning "rubbing" and includes the science of friction, wear and lubrication [2]. The most compelling motivation to study tribology is ultimately to gain full control of friction and wear for various technical applications. Especially, reducing friction is of great interest as this has advantages for energy efficiency. It has been reported that tribological problems have a significant potential for economic and environmental improvements [3]:

> "On global scale, these savings would amount to 1.4% of the GDP annually and 8.7% of the total energy consumption in the long term." [4].

On the other hand, the reduction of friction is not the only sensible application for tribological studies. Controlling frictional properties, besides minimization, might be of interest in the development of a grasping robot where finetuned object handling is required. While achieving a certain "constant" friction response is readily obtained through appropriate material choices, we are yet to unlock the full capabilities to alter friction dynamically on the go. One example from nature inspiring us to think along these lines are the gecko feet. More precisely, the Tokay gecko has received a lot of attention in scientific studies aiming to unravel the underlying mechanism of its "togglable" adhesion properties. Although geckos can produce large adhesive forces, they retain the ability to remove their feet from an attachment surface at will [5]. This makes the gecko able to achieve a high adhesion on the feet when climbing a vertical surface while lifting them for the next step remains relatively effortless. For a grasping robot, we might consider an analog frictional concept of a surface material that can change from slippery to rough on demand depending on specific tasks; Slippery and smooth when interacting with people and rough and firmly gripping when moving heavy objects.

In recent years an increasing amount of interest has gone into the studies of the microscopic origin of friction, due to the increased possibilities in surface preparation and the development of nanoscale experimental methods. Nano-friction is also of great concern for the field of nano-machining where the frictional properties between the tool and the workpiece dictate machining characteristics [3]. With concurrent progress in computational capacity and development of Molecular Dynamics (MD), numerical investigations serve as an invaluable tool for getting insight into the nanoscale mechanics associated with friction. This simulation-based approach can be considered as a "numerical experiment" enabling us to create and probe a variety of high-complexity systems which are still out of reach for modern experimental methods.

In materials science such MD-based numerical studies have been used to explore the concept of so-called *metamaterials* where the material compositions are designed meticulously to enhance certain physical properties [6–11]. This is often achieved either by intertwining different material types or removing certain regions completely. In recent papers by Hanakata et al. [6, 7], numerical studies have showcased that the mechanical properties of a graphene sheet, yield stress and yield strain, can be altered through the introduction of so-called *Kirigami* inspired cuts into the sheet. Kirigami is a variation of origami where the paper is cut additionally to being folded. While these methods originate as an art form, aiming to produce various artistic objects, they have proven to be applicable in a wide range of fields such as optics, physics, biology, chemistry and engineering [12]. Various forms of stimuli enable direct 2D to 3D transformations through folding, bending, and twisting of microstructures. While original human designs have contributed to specific scientific applications in the past, the future of this field is highly driven by the question of how to generate new designs optimized for certain physical properties. However, the complexity of such systems and the associated design space makes for seemingly intractable[1] problems ruling out analytic solutions.

Earlier architecture design approaches such as bioinspiration, looking at gecko feet for instance, and Edisonian, based on trial and error, generally rely on prior knowledge and an experienced designer [9]. While the Edisonian approach is certainly more feasible through numerical studies than real-world experiments, the number of combinations in the design space rather quickly becomes too large for a systematic search, even when considering the computation time on modern-day hardware. However, this computational time constraint can be relaxed by the use of machine learning (ML) which has proven successful in the establishment of a mapping from the design space to physical properties of interest. This gives rise to two new styles of design approaches: One, by utilizing the prediction from a trained network we can skip the MD simulations altogether resulting in an *accelerated search* of designs. This can be further improved by guiding the search accordingly to the most promising candidates, for instance, as done with the *genetic algorithm* based on mutation and crossing of the best candidates so far. Another more sophisticated approach is through generative methods such as *Generative Adversarial Networks* (GAN) or diffusion models. The latter is being used in state-of-the-art AI systems such as OpenAI's DALL·E2 [13] or Midjourney [14]. By working with a so-called *encoder-decoder* network structure, one can build a model that reverses the prediction process. This is often referred to as *reverse design*, where the model predicts a design from a set of physical target properties. In the papers by Hanakata et al. [6, 7] both the *accelerated search* and the *inverse design* approach was proven successful to create novel metamaterial Kirigami designs with the graphene sheet.

Hanakata et al. attributes the variation in mechanical properties to the non-linear effects arising from the out-of-plane buckling of the sheet. Since it is generally accepted that the surface roughness is of great importance for frictional properties it can be hypothesized that Kirigami-induced out-of-plane buckling can also be exploited for the design of frictional metamaterials. For certain designs, we might hope to find a relationship between the stretching of the sheet and frictional properties. If significant, this could give rise to an adjustable friction behavior beyond the point of manufacturing. For instance, the grasping robot might apply such a material as artificial skin for which stretching or relaxing of the surface could result in a changeable friction strength.

In addition, the Kirigami graphene properties can be explored through a potential coupling between the stretch and the normal load, through a nanomachine design, with the aim of altering the friction coefficient. This invites the idea of non-linear friction coefficients which might in theory also take on negative values. The latter would constitute a rare property only found a few cases. These are mainly for the unloading phase of adhesive surfaces [15] or the loading phase of particular heterojunction materials [16, 17].

To the best of our knowledge, Kirigami has not yet been implemented to alter the frictional properties of a nanoscale system. However, in a recent paper by Liefferink et al. [18] it is reported that macroscale Kirigami can be used to dynamically control the macroscale roughness of a surface through stretching. They reported that the roughness change led to a changeable frictional coefficient by more than one order of magnitude. This supports the idea that Kirigami designs can be used to alter friction, but we believe that taking this concept to the nanoscale would involve a different set of governing mechanisms and thus contribute to new insight in this field.

[1] In computer science we define an *intractable* problem as a problem with no *efficient* algorithm to solve it nor any analytical solutions. The only way to solve such problems is the *brute-force* approach, simply trying all possible solutions, which is often beyond the capabilities of computational resources.

1.2 Goals

In this thesis, we investigate the prospects of altering the frictional properties of a graphene sheet through the application of Kirigami-inspired cuts and stretching of the sheet. With the use of molecular dynamics (MD) simulations, we evaluate the frictional properties of various Kirigami designs under different physical conditions. Based on the MD results, we investigate the possibility to use machine learning for the prediction of frictional properties and subsequently using the model for an accelerated search of new designs. The main goals of the thesis can be summarized as follows.

1. Design an MD simulation procedure to evaluate the frictional properties of a Kirigami graphene sheet under specified physical conditions.

2. Develop a numerical framework to generate various Kirigami designs, both by seeking inspiration from macroscale designs and by the use of a random walk based algorithm.

3. Investigate the frictional behavior under varying load and stretch for different Kirigami designs.

4. Develop and train a machine learning model to predict the MD simulation result and perform an accelerated search of new designs with the scope of optimizing certain frictional properties.

1.3 Contributions

By working towards the goals outlined above (Sec. 1.2), I have discovered a non-linear relationship between the kinetic friction and the strain for certain Kirigami patterns. This phenomenon was found to be associated with the out-of-plane buckling of the Kirigami sheet but with no clear relationship to the contact area or the tension in the sheet. I found that this method does not provide any mechanism for a reduction in friction, in comparison to a non-cut sheet, but the straining of certain Kirigami sheets allows for a non-monotonic increase in friction. The relationship to normal load was proven negligible in this context and I have demonstrated that a coupled system of load and strain (through sheet tension) can exhibit a negative friction coefficient in certain load ranges. Moreover, I have created a dataset of roughly 10,000 data points for assessing the employment of machine learning and accelerated search of Kirigami designs. I have found, that this approach might be useful, but it requires an extended dataset in order to produce reliable results for a search of new designs.

During our investigations, I have built three numerical tools, beyond the regular scripts associated with data analysis, which can be found on Github [19]. The tools are summarized in the following.

- I have written a LAMMPS-based [20] tool for simulating and measuring the frictional properties of a graphene sheet sliding on a substrate. The code is generally made flexible concerning the choice of sheet configuration, system size, simulation parameters and MD potentials, which makes it applicable for further studies within this topic. I have also built an automized procedure to carry out multiple simulations under varying parameters by submitting jobs to a computational cluster via an ssh connection. This was done by adding minor additions to the python package developed by E. M. Nordhagen [21].

- I have generated a Python-based tool for generating Kirigami patterns and exporting these in a compatible format with the simulation software. The generation of molecular structures is done with the use of ASE [22]. Our software includes two classes of patterns inspired by macroscale designs and a random walk algorithm which allow for a variety of different designs through user-defined biases and constraints. Given our system size of choice, the first two pattern generators are capable of generating on the order of 10^8 unique designs while the random walk generator allows for significantly more.

- I have built a machine learning tool based on Pytorch [23] which includes setting up the data loaders, a convolutional network architecture, a loss function, and general algorithms for training and validating the results. Additionally, I have written several scripts for performing grid searches and analyzing the model predictions in the context of the frictional properties of graphene.

All numerical implementations used for this thesis have been originally developed for the purpose with the exception of the usage of libraries as mentioned above and commonly known Python libraries such as Numpy and Matplotlib.

1.4 Thesis structure

The thesis is divided into two parts. In Part I we introduce the relevant theoretical background, and in Part II we present the numerical implementations and the results of this thesis.

Part I contains a description of the theoretical background related to Friction (**??**), Molecular Dynamics (**??**) and Machine Learning (Chapter 2). In **??** we fill formulate our research questions n the light of the friction theory.

In Part II, we begin by presenting our definition and setting up of the systemm in **??**. This includes the MD simulations and the generation of Kirigami designs. This is followed by a pilot study in **??** where we evaluate the simulation results for various physical conditions and compare a non-cut sheet to two different Kirigami designs. Further explorations of the Kirigami configurations are carried out in **??** which includes the generation of a dataset and the employment of machine learning and an accelerated search. We use the results from the pilot study to demonstrate the possibility to achieve a negative friction coefficient for a system with coupled load and strain in **??**. Finally, we summarize our results and provide an outlook for further studies in Chapter 3. Additional figures are shown in **??**, **??** and **??**.

Part I

Background Theory

Chapter 2

Machine Learning

We will use machine learning to predict the friction resulting from the straining and loading of a given Kirigami sheet. To this end, we will generate data through MD simulations that will serve as the ground truth for training a machine learning model. The advantage of using machine learning for this purpose is that it can significantly speed up the exploration of new configurations compared to full MD simulations. However, there is no guarantee that the machine learning model can accurately capture the physical mechanisms governing our system. Hence, a key objective is to assess the viability of this approach in the study of Kirigami friction, which we will pursue using traditional machine-learning methods. In this chapter, we introduce the key concept behind machine learning and some of the concepts and techniques relevant to our implementation. For the numerical implementation, we will use the machine learning framework PyTorch [23]

2.1 Neural network

The neural network, or more precisely the *feed-forward dense neural network*, is one of the original concepts in machine learning arising from the attempt of mimicking the way neurons work in the brain [24, 25] brain. The neural network can be considered in terms of three major parts: The input layer, the so-called *hidden layers* and finally the output layer as shown in Fig. 2.1. The input is described as a vector $\mathbf{x} = x_0, x_1, \ldots, x_{n_x}$ where each input x_i is usually denoted as a *feature*. The input features are densely connected to each of the *nodes* in the first hidden layer as indicated by the straight lines in Fig. 2.1. Each line represents a weighted connection that can be adjusted to configure the importance of that feature. Similar dense connections are present throughout the hidden layers to the final output layer. For a given note $a_j^{[l]}$ in layer l the input from all the nodes in the previous layer $l-1$ are processed as

$$a_j^{[l]} = f\left(\sum_i w_{ij}^{[l]} a_i^{[l-1]} + b_j^{[l]}\right),$$

where $w_{ij}^{[l]}$ is the weight connection node $a_i^{[l-1]}$ of the previous layer to the node $a_j^{[l]}$ in the current layer. Note that having the weight belong to layer l as opposed to $l-1$ is simply a notation choice. $b_j^{[l]}$ denotes a bias and $f(\cdot)$ is the so-called *activation function*. The activation function provides a non-linear mapping of the input to each node. Without this, the network will only be capable of approximate linear functions [24]. Two common activation functions are the *sigmoid*, mapping the input to the range $(0, 1)$, and the *ReLU* which cuts off negative contributions

$$\text{Sigmoid:} \quad f(z) = \frac{1}{1+e^{-z}}, \qquad \text{ReLU:} \quad f(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}.$$

Often the same activation function is used throughout the network, except for the output layer where the activation function is usually omitted or the sigmoid is used for classification tasks. The whole process of sending data through the model is called *forward propagation* and constitutes the mechanism for mapping an input $\mathbf{x}$ to the model output $\hat{\mathbf{y}}$. In order to get useful predictions we must *train* the model which involves tuning the model parameters, i.e. the weight and biasses.

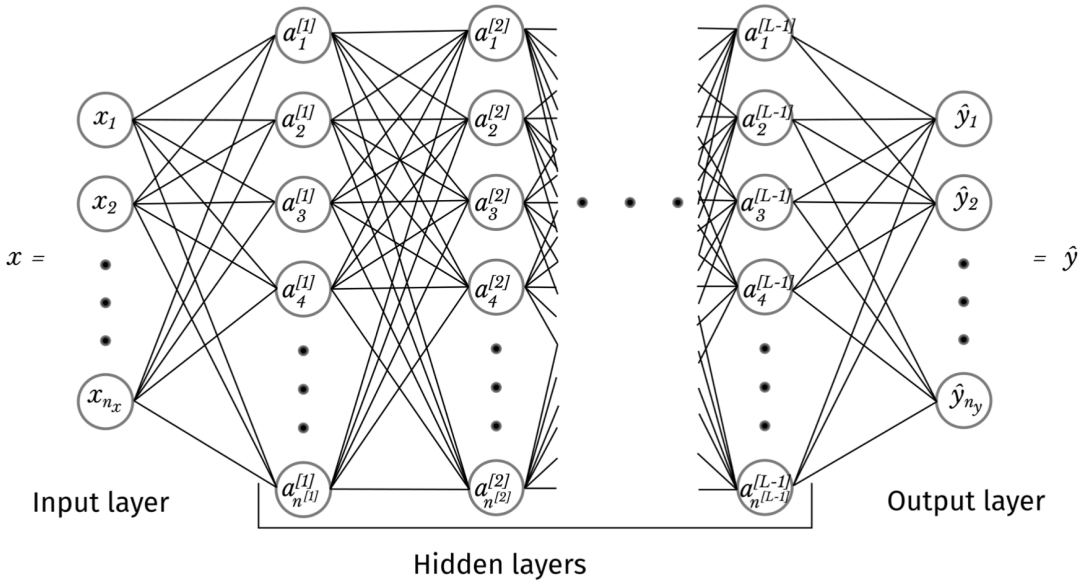


Figure 2.1: Illustration of a general feed-forward dense neural network with n_x input features and n_y outputs. Reproduced from [26].

The model training relies on two core concepts: *backpropagation* and *gradient descent* optimization. First, we define the error associated with a model prediction, otherwise known as the *loss*, through the *loss function* $L(\hat{\mathbf{y}}, \mathbf{y})$ that evaluates the model output $\hat{\mathbf{y}}$ against the ground truth $\mathbf{y}$. For a continuous scalar output, we might simply use the mean squared error (MSE)

$$L_{\text{MSE}} = \frac{1}{n_y} \sum_{i=1}^{n_y} (y_i - \hat{y}_i)^2. \tag{2.1}$$

For a binary classification problem, meaning that the output is True or False (1 or 0), a common choice is binary cross entropy (BSE)

$$L_{\text{BSE}} = -\sum_{i=1}^{n_y} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right] = \sum_{i=1}^{n_y} \begin{cases} -\log(\hat{y}_i), & y_i = 1 \\ -\log(1 - \hat{y}_i), & y_i = 0. \end{cases} \tag{2.2}$$

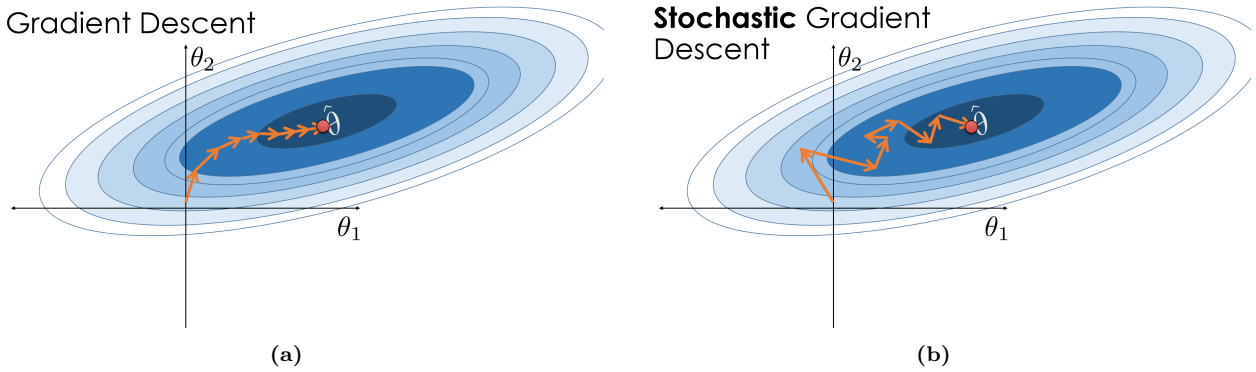
Without going into details with the derivation we can convince ourselves that the error is minimized for the correct prediction and maximized for the worst prediction. When $y_i = 1$ we get the negative loss contribution $-\log(\hat{y}_i)$ where a correct prediction $\hat{y}_i \to 1$ yields $L_i \to 0$. For a wrong prediction $\hat{y}_i \to 0$ the loss contribution will diverge $L_i \to \infty$. Similar applies to the case of $y_i = 0$ with opposite directions.

Given a loss function, we can calculate the loss gradient $\nabla_\theta L$ with respect to each of the weights and biases in the model. This is called *backpropagation* since we follow the propagation of the errors as we go backward through the model layers calculating the gradient using the chain rule. These gradients express how each parameter is connected to the loss and the overall idea is then to "nudge" each parameter in the right direction to reduce the loss. We usually denote a full cycle of forward propagation, backpropagation and an update of all model parameters as one *epoch*. We calculate the updated parameter θ_t for epoch t using the *gradient descent* method

$$\theta_t = \theta_{t-1} - \eta \nabla_\theta L(\theta_t). \tag{2.3}$$

Gradient descent is analog to taking a step in parameter space in the direction that yields the biggest decrease in the loss. If we imagine a simplified case with only two parameters θ_1 and θ_2 we can think of these as longitude and latitude coordinates on a map and the loss being the terrain height. The gradient descent steps in the direction perpendicular to the contour lines shaped by the loss function terrain as shown in Fig. 2.2. Notice,

however, that state-of-the-art models in general contain on the order of 10^6–10^9 parameters [27] which poses some challenges for the visualization. The length of each step is proportional to the gradient norm and the learning rate η. There are three main flavors to the gradient descent: Batch, stochastic and mini-batch gradient descent. In *batch gradient descent* we simply calculate the gradient based on the entire dataset by averaging the contribution from each data point before updating the parameters. This gives the most robust estimate of the gradient and thus the most direct path through parameter space in terms of minimizing the loss function as indicated in Fig. 2.2a. However, for big datasets, this calculation can be computationally heavy as it must carry the entire dataset in memory at once. A solution to this issue is provided by *stochastic gradient descent* (SGD) which considers only one data point at a time. Each data point is chosen randomly and the parameters are updated based on the corresponding gradient. This leads to more frequent updates of the parameters and a more "noisy" path through parameter space as shown in Fig. 2.2b. Under some circumstances, this might compromise the precision. However, the presence of noise can increase the likelihood of avoiding local minima in parameter space. The *mini-batch gradient descent* serves as a middle ground between the above-mentioned methods by dividing the full dataset into a subset of mini-batches. Each parameter update is then based on the gradient within a mini-batch. By choosing a suitable batch size we get the robustness of the (full) batch gradient descent and the computational efficiency and resistance to local minima of the SGD method.



(a) (b)

Figure 2.2: Qualitative illustration of the gradient descent method for a simplified problem with only two parameters θ_1 and θ_2. The blue shade and lines indicate the contour map of the loss function with the darker shade denoting a lower loss. (a) The batch gradient descent method resulting in a relatively straight path toward the optimal parameters. (b) The stochastic gradient descent method resulting in a more noisy path toward the optimal parameters. Reproduced from [28].

2.1.1 Optimizers

The name *optimizers* covers a variety of gradient descent methods. In our study, we will use the ADAM (adaptive moment estimation) optimizer [29]. ADAM combines several "tricks in the book" which we will introduce in the following.

One considerable extension of the gradient descent scheme is by the introduction of a momentum term m_t such that we get

$$\theta_t = \theta_{t-1} - m_t, \qquad m_t = \alpha m_{t-1} + \eta \nabla_\theta L(\theta_t), \tag{2.4}$$

with $m_0 = 0$. If we introduce the shorthand $g_t = \nabla_\theta L(\theta_t)$ we find

$$\begin{aligned} m_1 &= \alpha m_0 + \eta g_1 = \eta g_1 \\ m_2 &= \alpha m_1 + \eta g_2 = \alpha^1 \eta g_1 + \eta g_2 \\ m_3 &= \alpha m_2 + \eta g_3 = \alpha^2 \eta g_1 + \alpha \eta g_2 + \eta g_3 \\ &\vdots \\ m_t &= \eta \left(\sum_{k=1}^{t} \alpha^{t-k} g_k \right). \end{aligned} \tag{2.5}$$

Hence m_t is a weighted average of the gradients with an exponentially decreasing weight. This act as a memory of the previous gradients and aid to pass local minima and to some degree plateaus in the parameter space. It also provides a general steadiness to the descent which counteracts the transition from batch to mini-batch gradient descent. A variation of momentum can be achieved with the introduction of the exponential moving average (EMA) which builds on the recursion

$$\begin{aligned}
\mathrm{EMA}(g_1) &= \alpha \overbrace{\mathrm{EMA}(g_0)}^{\equiv 0} + (1-\alpha) g_1 \\
\mathrm{EMA}(g_2) &= \alpha \mathrm{EMA}(g_1) + (1-\alpha) g_2 \\
&\vdots \\
\mathrm{EMA}(g_t) &= \alpha \mathrm{EMA}(g_{t-1}) + (1-\alpha) g_t = \sum_{k=0}^{t} \alpha^{t-k} (1-\alpha) g_t,
\end{aligned}$$

which is similar to that of momentum Eq. (2.5), but with the explicit weighting by $(1-\alpha)$. The second moment of the exponential moving average is utilized in the root mean square propagation method (RMSProp) which is motivated by the issue of passing long loss plateaus in the parameter space. Since the size of the updates are proportional to the norm of the gradient

$$\theta_{t+1} = \theta_t - \eta g_t \implies ||\theta_{t+1} - \theta_t|| = \eta ||g_t||,$$

we might get the idea of normalizing the gradient step by the norm $|||g_t|$. However, this does not immediately solve the problem of long plateaus as we need to consider multiple past gradients, but this can be done with the use of the EMA. When reentering a steep region again we need to "quickly" downscale the gradient steps which can be achieved more efficiently by using the squared norm $||g_t||^2$ for the EMA which makes it more sensitive to outliers. From this motivation the RMSProp update scheme is given

$$\theta_t = \theta_{t-1} - \eta \frac{g_t}{\sqrt{\mathrm{EMA}(||g_t||^2) + \epsilon}}, \tag{2.6}$$

where ϵ is simply a small number to avoid division by zero issues.

ADAM merges the idea of first order EMA for the momentum m_t, and the second order EMA v_t for gradient normalization similar to the root mean square propagation technique in Eq. (2.6)

$$\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2.
\end{aligned}$$

Since m_t and v_t are initially set to zero ADAM introduces the scaling terms $(1-\beta_1^t)$ and $(1-\beta_2^t)$ to correct for a bias towards zero. The ADAM scheme is given [29]

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \qquad \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{2.7}$$

2.1.2 Weight decay

By adding a so-called *regularization* to the loss function we can penalize high magnitudes of the model parameters, usually intended for the model weights. This is motivated by the idea of preventing overfitting during training, which we will address in more detail in Sec. 2.3. The most common way to regularize the loss function is by the use of L2 regularization, adding the squared l^2 norm $||\theta||_2^2$, where $||\theta||_2 = \sqrt{\theta_1^2 + \theta_2^2 + \ldots}$, to the model. The loss and gradient then become

$$L_{l^2}(\theta) = L(\theta) + \frac{1}{2}\lambda ||\theta||_2^2 \tag{2.8}$$

$$\nabla_\theta L_{l2}(\theta) = \nabla_\theta L(\theta) + \lambda\theta, \tag{2.9}$$

where $\lambda \in [0, 1]$ is the weight decay parameter. The name *weight decay* relates to the fact that some practitioners only apply this penalty to the weights in the model, but we will include the biases as well (standard in PyTorch).

Following the original gradient descent scheme Eq. (2.9) we get

$$\theta_{t+1} = \theta_t - \eta g_t - \eta\lambda\theta_t = \theta_t \underbrace{(1-\eta\lambda)}_{\text{weight decay}} - \eta g_t. \tag{2.10}$$

We notice that choosing a large weight decay ($\lambda \to 1$) will downscale the model parameters while choosing a low weight decay ($\lambda \to 0$) yields the original gradient descent scheme. Note that we will use the weight decay principle in combination with ADAM. In Eq. (2.10) we have simply used the original gradient descent scheme Eq. (2.3) since this makes it easier to demonstrate the consequences of introducing the L2 regularization into the loss function Eq. (2.8).

2.1.3 Parameter distributions

In order to get optimal training conditions it has been found that the initial state of the weight and biases are important [30]. First of all, we must initialize the weight by sampling from some distribution. If the weights are set to equal values the gradient across a layer would be the same. This results in a complexity reduction as the model can only encode the same values across the layer. Further, we want to consider the gradient flow during training. Especially for deep networks, networks with many layers, we must pay attention to the problem of *vanishing* or *exploding* gradients. If we for instance consider the sigmoid activation function and its derivative

$$f(z) = \frac{1}{1+e^{-z}}, \qquad f'(z) = \frac{df(z)}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{e^z}{(1+e^z)^2},$$

we notice that for large and small input values z we get $f(z \to \pm\infty) \to 0$. However, even a small finite gradient can vanish throughout a deep network as the calculation of the gradient involves the chain rule. This gives rise to a gradient that potentially gets smaller and smaller for each layer it passes in the backpropagation. A similar problem can be found with the ReLU activation function which contributes toward a gradient of zero for inputs $z < 0$. This can be mitigated by the so-called leaky RelU which maps the $z < 0$ to a small negative slope $a < 0$ as $f(z) = az$. On the other hand, we have exploding gradients, which are simply a result of the chain rule gradient calculation. For a sufficiently deep network, the gradient can grow exponentially large and sometimes result in a numerical overflow. One approach to mitigate this is with the use of gradient clipping, where all gradients above a certain value are manually set to a predefined maximum number. While there exist techniques to accommodate the problem of vanishing or exploding gradients as mentioned above, they both benefit from a properly initialized set of weights. That is, we want the gradients across a given layer to have a zero mean while the variance is similar between layers in the model. This balanced gradient flow is more likely to happen if we initialize the weight by the same criteria [30]. The specific actions to achieve this depend on the model architecture, including the choice of activation functions. For instance, using the ReLU activation functions it was found that the node standard deviation will depend on the number of input nodes from the previous layer $N^{[l-1]}$ as $\sqrt{N^{[l-1]}/2}$ [31]. Thus we can simply generate the weights from a zero mean normal distribution $N \sim (0, 2/N^{[l-1]})$ with the standard deviation $\sqrt{2/N^{[l-1]}}$ to ensure a balanced initialization of weights. This is part of the Kaiming initialization scheme which is standard in Pytorch. The bias is initialized from a similar consideration.

Batch normalization is another technique that can help reduce the issue of poor gradient flow. Furthermore, it can benefit by speeding up convergence and making the training process more stable [32]. In general, model parameters are modified throughout training meaning that the range of values coming from a previous layer will shift (internal covariate shift), even though the same training data is fed through the network repeatedly. By scaling the input for a given layer, for each mini-batch, we can mitigate this problem and make for a more standardized input range. This often results in a faster training convergence. For layer l we calculate the mean $\mu^{[l]}$ and variance $(\sigma^{[l]})^2$ across the layer with nodes $x_1^{[l]}, x_2^{[l]}, \ldots, x_m^{[l]}$ for each mini-batch of size m as

$$\mu^{[l]} = \frac{1}{m}\sum_i^m x_i, \qquad (\sigma^{[l]})^2 = \frac{1}{m}\sum_i^m (x^i - \mu)^2.$$

We then perform a normal scaling of the inputs within the batch

$$\hat{x}_i^{[l]} = \frac{x_i^{[l]} - \mu^{[l]}}{\sqrt{(\sigma^{[l]})^2 + \epsilon}},$$

where ϵ is a small number to ensure numerical stability (similar to what we used for RMSProp gradient descent). In the final step, the input values are rescaled as

$$\tilde{x_i} = \gamma^{[l]}\hat{x}_i^{[l]} + \beta^{[l]}$$

with trainable parameters γ and β.

2.1.4 Learning rate decay strategies

Until now we have assumed a constant learning rate, but many training schemes use a changing learning rate beyond the adaptiveness included in the optimizers covered so far. Under some circumstances, it can be beneficial to start with a higher learning rate to speed up the initial part of training and then lower the learning rate for the final gradient descent [33]. One straightforward strategy is a step-wise learning rate decay where the learning rate is reduced by a factor $\gamma \in (0, 1)$ every K steps. A more smooth change can be achieved by for instance a polynomial decay $\eta_t = \eta_0/t^\alpha$ for $\alpha > 0$. More advanced approaches use multiple cycles of increasing and decreasing cycles. We will mainly concern ourselves with a one-cycle policy for which we start at an intermediate value, increase toward a maximum bound and then decrease toward a final lower learning rate bound. We do this by following a cosine function that is shifted and scaled to increase towards the maximum bound for the first 30% of the training length and decrease toward the lower learning rate bound for the remaining epochs.

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) build upon many of the same concepts as introduced with the feed-forward neural network in Sec. 2.1. The difference lies in its specialization for a spatially correlated input, such as pixels in an image. In a dense neural network, every node is connected to each of the nodes from the previous layers which is not ideal for image recognition. For instance, if we want the model to recognize images of animals the dense network will be very sensitive to where that animal is placed within the frame. The CNN is motivated by the idea of capturing spatial relations in the input, but without being sensitive to the relative placement within the input, i.e. being translational invariant. This is achieved by having a so-called *kernel* or *filter* which slides over the images[2] as it processes the input. The overall flow of data for a typical convolutional network including a final fully connected neural network is illustrated in Fig. 2.3.

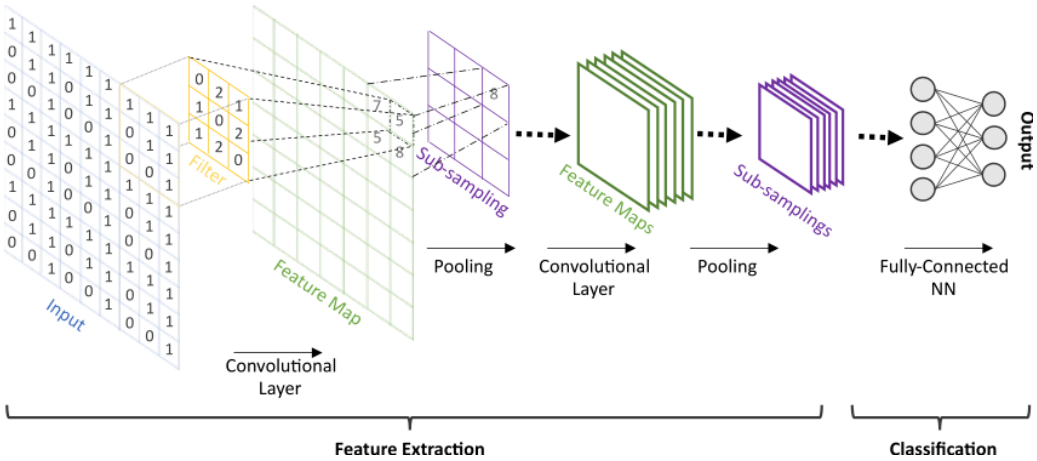


Figure 2.3: Representation of a Convolutional Neural Network (CNN). The CNN performs automatic spatial feature extraction from images by successively applying feature filters that create feature maps (Convolutional layer) and compressing these maps (Pooling layer). Based on the final feature maps, a fully-connected neural network does a prediction, which can be a classification or regression. Figure and caption reproduced from [34].

[2]Note, that we will be using the word "image" as a reference for a spatially dependent input, but in reality, it does not have to be an actual image in the classical sense.

A convolutional layer contains multiple kernels, each consisting of a set of trainable weights and a bias. Each kernel will produce a separate output channel to the resulting *feature map* layer. The kernel has a 2D spatial size, specific to the model architecture, and a depth that matches the number of input channels to the layer. For instance, a typical RGB image will have three channels, while the number of channels usually increases for each layer in the model. The kernel lines up with the image and calculates the feature map output as a dot product between the weights in the kernel and the aligning subset of the input. This is done for each input channel and summed up with the addition of a bias as illustrated in Fig. 2.4b. The kernel then slides over by a step size given by the *stride* parameter and repeats the calculation. Choosing a stride of 2 or higher results in a reduction of the output spatial size. If we want to preserve the spatial size we must keep a stride of one and additionally apply *padding* to the input images, such that we can achieve one kernel position for each input "pixel". The spatial size of the feature map is given as

$$N_d^{[l]} = \left\lfloor \frac{N_d^{[l-1]} - F_d + 2P}{S} + 1 \right\rfloor, \tag{2.11}$$

for padding P, stride S, spatial size of the kernel filter F_d, spatial size of the input $N_d^{[l-1]}$, for dimension $d = \{x, y\}$ and layer l. The *down-sampling* is often done through a pooling layer. A pooling layer is reminiscent of a kernel, but instead of calculating the output as a dot product, it calculates the mean (mean pooling) or the max value (max pooling) of the values within its scope. For instance, by using a max pooling of size 2×2 and stride 2 we essentially half the dimensions of the image as dictated by Eq. (2.11). CNNs will often use repeating series of convolution (applying a kernel), pooling and then an activation function. Most architectures aim to down-sample the spatial input while increasing the number of channels throughout the model layers. This results in a smaller set of features extracted from the input which then can then be fed into a dense network, or *fully connected* connected neural network, as also illustrated in Fig. 2.3. The convolution part aims to handle the transition from a spatial input into some internal features. For a model which recognizes animals, we would perhaps think of features such as the number of legs, size, color and so on. In practice, the network will not give readily interpreted features for the processing in the fully connected layer, but the concept is still the same.

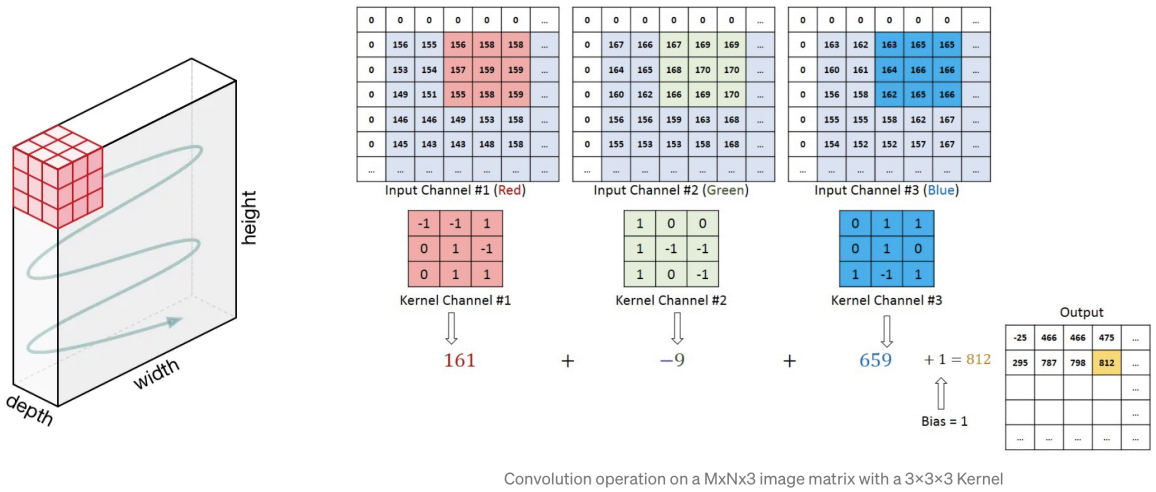


Figure 2.4: Illustration of the convolution procedure for a 3×3 kernel matching with a depth of 3 matching an RGB input with 3 channels (a) The kernel goes through the various positions aligning with the input as sketched with the arrow. The precise number of positions along this path depends on the stride parameter and the choice of padding. (b) A specific example of the calculations involved in the convolution process. Here a rim of zero padding is included. For each kernel position, a dot product is calculated between the aligning input and the kernel weights for each corresponding channel. This results in a number for each channel which is then summed up and added with a bias to constitute the final output in the feature map. Reproduced from [35].

For a CNN, we often consider the *receptive field*. The receptive field relates to the spatial size of the input that affects a given node in the feature map at a given layer in the model. Often this term is used in consideration of the output nodes. Fig. 2.5 illustrates the receptive field for a 1D representation of a CNN with repetitive use of a kernel of width 1 and stride 1. Going from the output and backward, we see that the output layers are connected to two nodes in the previous layer. Each of these nodes is connected to two nodes in the layer before that, however with one of them being the same due to the stride of 1. By back-tracking to the input layer we see that this corresponds to a receptive field of $D = 5$. This means that a single output node is only affected by the 5 input nodes within its receptive field. By increasing the filter size and the stride the receptive field will grow a lot faster than shown in this example. If we assume a constant filter size throughout the network F, a stride S_l from layer $l-1$ to l we get that the receptive field D_l with respect to a certain layer and in a given spatial dimension is

$$D_l = D_l + \left[(F_l - 1) \cdot \prod_{i=1}^{l-1} S_i \right], \tag{2.12}$$

with $D_0 = 1$ and $l = 0$ as the input layer. Note that by convention, the product of zero elements is 1, such that for the first layer, the product is 1. Eq. (2.12) apply for both spatial dimensions.

The receptive field is important in understanding the connectivity in the model since the output will be completely independent of the inputs and feature maps outside the receptive field. Furthermore, we differentiate between the theoretical receptive field and the effective receptive field. The effective receptive field will have a Gaussian distribution within the theoretical receptive field because the nodes in the center of the receptive field will have more connections leading to the output, as seen in Fig. 2.5. Thus, in practice, the effective receptive field will be smaller than the theoretical. Implementations like dilated convolutions, which make the filter expand in circumference and skip positions within the filter, can be used to further increase the effective field.

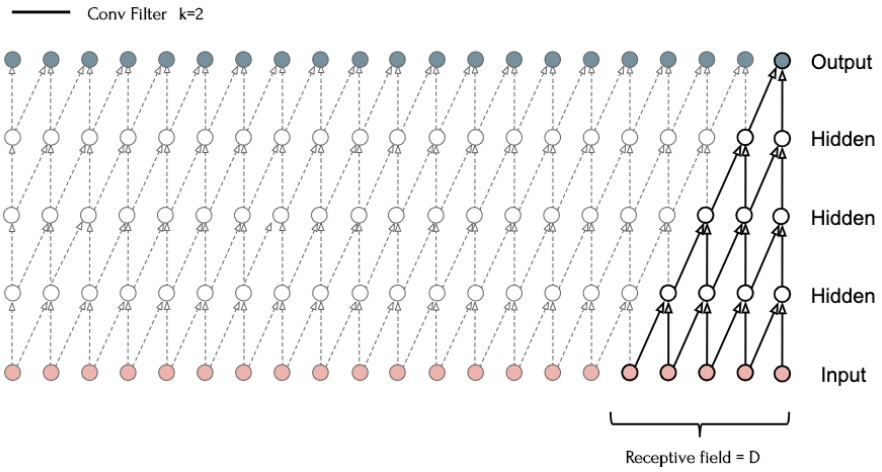


Figure 2.5: An illustration of the receptive field D with respect to an output node in a 1D convolutional network. This example uses a width of 2 for the kernel and a stride of 1. Reproduced from [36]

2.2.1 Training, validation and test data

So far, we have simply considered the concept of *training data* as a means to update the model parameters. Yet, we want to evaluate the model performance as it improves. The problem arises immediately from the fact that a complex model can fit about any function. More precisely, it has been proven that a deep convolutional neural network is universal (follows the universal approximation theorem), meaning that can approximate any continuous function to an arbitrary accuracy when the depth of the network is large enough [37]. Thus for a complex model, it is just a matter of time before the model eventually finds a good approximation for the training data. However, we want the model to learn general trends and not to "memorize" all the data points which are known as *overfitting*. While the predictions for the training data can grow arbitrarily good in most cases, the performance on unseen data within the domain will yield poor performance in the case of overfitting. The

common way to address this issue is by putting aside a subset of the data, the so-called *validation* data, which we use to validate the model performance during and after training. By keeping this *validation* set separate from the training data we can get a more reliable performance estimate for the model. Random partitioning is crucial for ensuring an equal distribution of data across both sets. To strike a balance between the quality of training and validation, a commonly used partitioning ratio is usually around 20:80 in favor of the training set. Other techniques exist which aim to optimize the data used for sparse data situations, like cross-validation and bootstrap right?, but we will not consider such methods for this thesis. A third data set that is often forgotten is the *test* set. While the validation set should be kept unseen from the model training, the test set should be kept unseen from the model developer. As we choose the model architecture and hyper-parameters. We define a hyper-parameter as a variable to be set prior to the actual application of the learning algorithm, one that is not selected by the learning algorithm itself [38]. This includes parameters such as learning rate, momentum and weight decay, but not the weights and biases as these are updated by the learning algorithm. When adjusting the hyper-parameters we will use the performance on the validation set as a guiding metric. Hence, our choices can eventually lead to a higher level of overfitting through the hyper-parameter choices. Hence, we should denote a test set for the final evaluation of our model which has not been considered before the end. Formally, this is the only reliable performance metric for the model.

2.3 Overfitting and underfitting

Underfitting and overfitting represent a crucial balance going on when training a model. This concept is highly related to the model complexity and the chosen hyper-parameters. The textbook visualization of underfitting and overfitting is shown in Fig. 2.6a. As we begin to train or model both the training and validation loss is decreasing. At some point, the model will start to pick up, not only the general data trends but also specific trends in the training data. This marks the transition into the overfitting regime where the validation loss will increase again, even though the training loss is steadily declining. *Early stopping* can be utilized to detect this transition and stop the training in an attempt to hit the sweet spot between under- and overfitting. We will use a variation of this which is to store the best model based on validation performance. For this approach, we let the training finish but only keep the model corresponding to the best validation score. In principle, we can "get lucky" and find the model settings at a state that is specially overfitted for the validation set, but we consider this highly unlikely when having a reasonable amount of data and a complex model with many model parameters. The underfitting and overfitting phenomena can also be thought of as a function of the complexity and not just training time. For a certain amount of epochs a simple model will yield underfitting and an overly complex model will yield overfitting, and this can be expected to follow a similar qualitative trend as in Fig. 2.6a with the substitution for *model complexity* on the x-axis. Fig. 2.6b visualizes the concept of underfitting and overfitting in terms of the complexity regarding the fitting of a second-order polynomial. We see how a simple linear function will make a crude approximation for the true curve. An overly complex model will pick up the noise in the data and miss the general trend. However, the problem is that we do not know the true curve. If we did, we would not need machine learning to approximate it in the first place. Without having additional insight into the governing source of the data the overfitting case seems to produce the most confident fit for all we know.

2.4 Hypertuning

The training of a machine learning model revolves around tuning the model parameters such as weights and biases. However, as mentioned already, a handful of *hyper-parameters* remains for us to decide. First of all, we need to choose an architecture for the model. This includes high-level considerations, for instance, whether to use a neural network or a convolutional network, but also lower-level considerations, such as the depth and the width of the model, i.e. how many layers and how many nodes/channels. In addition, we have to define and consider the loss function and the optimizer which come with hyper-parameters such as learning rate, momentum and weight decay. This extensive list of choices makes the designing of a functional machine learning procedure more complicated than simply hitting "run" for the learning algorithm. As N. Smith [33] puts it: "Setting the hyper-parameters remains a black art that requires years of experience to acquire". In the following, we will review a general approach for choosing the learning rate, momentum and weight decay hyper-parameters based on the findings of [33]. The traditional approach is to perform a *grid search*, trying out different combinations of hyper-parameters different training sessions, but this might rather quickly become computationally expensive

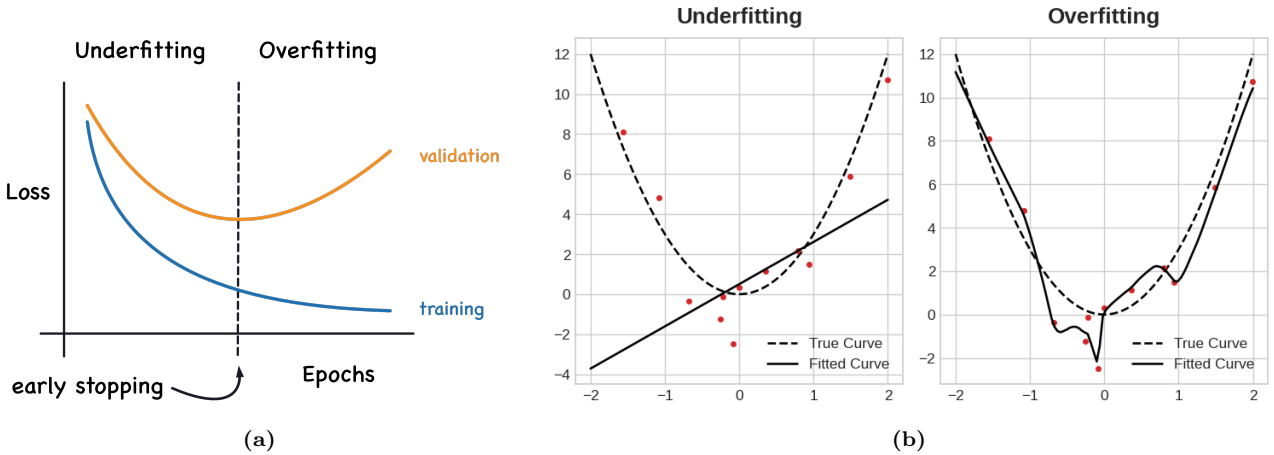


Figure 2.6: TMP

and ineffective. In addition, hyper-parameters will depend on the training data, the model architecture and not at least each other, which make it difficult to narrow down the choice one by one. N. Smith points to the fact that validation loss can be examined early on for clues of either underfitting or overfitting.

The learning rate is often regarded as the most important hyper-parameter to tune [38]. Typical values are in the range $[10^{-6}, 1]$. Instead of simply running a grid search, we can perform a so-called *learning rate range test* (LR range test). One then specifies the minimum and maximum learning rate boundaries and a learning rate step size. A minimum and maximum bound of 10^{-7} to 10 will most likely cover an appropriate range, but the test will reveal this immediately. The idea is then to vary the learning rate throughout the given range in small steps during a short pre-training. We will vary the learning rate for each iteration, i.e. each parameter update following a mini-batch, and thus we can run this test for a few epochs, or even a single one, depending on the number of mini-batches. The learning rate can be varied in a linear increasing or decreasing manner which is found to produce similar results [39]. We chose the linear increasing version for simplicity. For small learning rates, the model will converge slowly. As the learning rate approaches an appropriate value the convergence will accelerate which we see as a drop in the validation loss. Eventually, the convergence will stop and the validation loss will pass a minimum for which it will begin to diverge. This general behavior can be understood for the simplified 1D example of finding the minima of a second-order polynomial as shown in Fig. 2.7. Small learning rates will step in the right direction, but for very small values this will result in a slow convergence. If the learning rate becomes too large, we will effectively step past the minimum. Each following step will overshoot the minimum more and more (the step is proportional to the gradient os the loss) leading to a diverging trend. The point of divergence can be used as an upper bound for the learning rates when considering a cyclic learning rate scheme. The steepest decline of the validation loss can be used as an estimate for the best constant learning rate choice [33].

Next, we consider the choice of momentum. Momentum and learning rates are found to affect each other considerably. From the gradient descent scheme with momentum Eq. (2.4) we see that the momentum parameter α and the learning rate η have a similar effect on the parameter update

$$\theta_t = \theta_{t-1} - \eta g_t - \alpha m_{t-1},$$

since m_t is a moving average of the gradient g_t as well. Like the learning rate, we want to set the momentum value as high as possible without causing instabilities in the training. However, it is found that these values are highly related. N. Smith [33] reports that a momentum range test is not useful to find the right momentum. Instead, he suggests doing a few short runs with different values of momentum, such as 0.99, 0.97, 0.95, and 0.9, to determine a suitable choice. By including momentum in the LR range test we can balance the learning rate accordingly. Moreover, for a cyclic learning rate scheme he reports that a cycling momentum scheme, reversed with respect to the learning rate, is beneficial. When the learning rate increase toward the upper bound the learning rate should decrease toward the lower bound and vice versa. Choosing a lower momentum of 0.80–0.85 often gives similar stabile results [33].

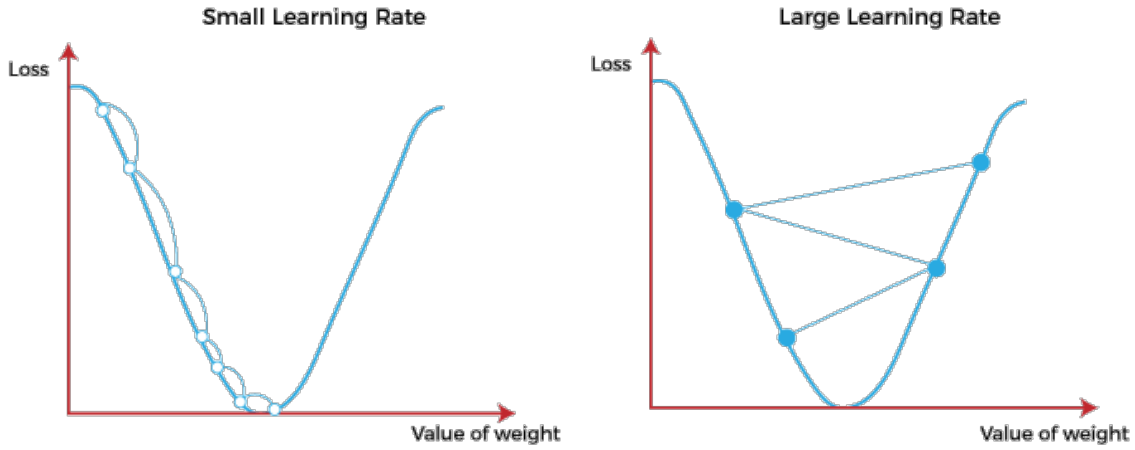


Figure 2.7: TMP

Finally, we address weight decay. N. Smith [33] reports that weight decay is different from learning rate and momentum by the fact that weight decay is better chosen as a constant value as opposed to a cyclic scheme. However, the weight decay is dependent on the model complexity, learning rate and momentum choice and this can often be dialed in after setting those. We can estimate a suitable choice by doing a rough grid search for values such as 0, 10^{-6}, 10^{-5} and 10^{-4} for complex architectures and 10^{-4}, 10^{-3} and 10^{-2} for more shallow architectures. Choosing the weight decay on the scale of exponential exponents will often provide good enough precision in practice.

2.5 Prediction explanation

On a final note, we present a simple method for providing some insight into the prediction from a convolutional neural network. The high complexity of deep learning models limits our ability to gain insight into the decision-making process behind a prediction beyond the input data. This is known as the *black box* problem. A lot of effort is currently being developed for making more transparent models, like decision trees with interpretable rules, and numerical tools for unpacking the inner workings of the model. We will consider a gradient based method called *Grad-CAM* [40] which aim to highlight some of the important features from the input image. The algorithm is based on the idea use the gradients for a certain feature map with respect to the loss.

First, we forward propagate the input through the model and decide on a feature map of interest. We then calculate the gradients for the feature map with respect to the loss of a certain target output. For a classification task, one would often choose the predicted class, the class with the highest score, as the target output. The gradients can then be used as an estimate of which part of the feature maps is most important for the prediction. a ReLU activation layer is then applied to keep only the positive contributions. Since the convolutional layers preserve spatial information we can rescale the heatmap provided by the feature map gradient to make an input-sized heatmap allowing for an overlaid visualization of the on the input image. This provides a visual clue of which part of the image the prediction is most strongly based on. We can do this for different depths of the model and even combine the results for multiple layers. Fig. 2.8 show an exemplary use, where the Grad-CAM analysis reveals the difference between a biased and unbiased prediction model for the task of predicting professions. The biased model shows to be considering the person more than the actual objective clues given by relevant equipment and work-related clothing

2.6 Accelerated search using genetic algorithm

For the scope of finding new Kirigami designs which exhibit certain frictional properties, we are interested in utilizing a trained machine-learning model for further exploration. This reverses the design process as one has to find the right input to achieve a certain output. A possible strategy is to explore a range of inputs and use the model predictions as a guiding metric. One approach to this is the genetic algorithm (GA) which is inspired by

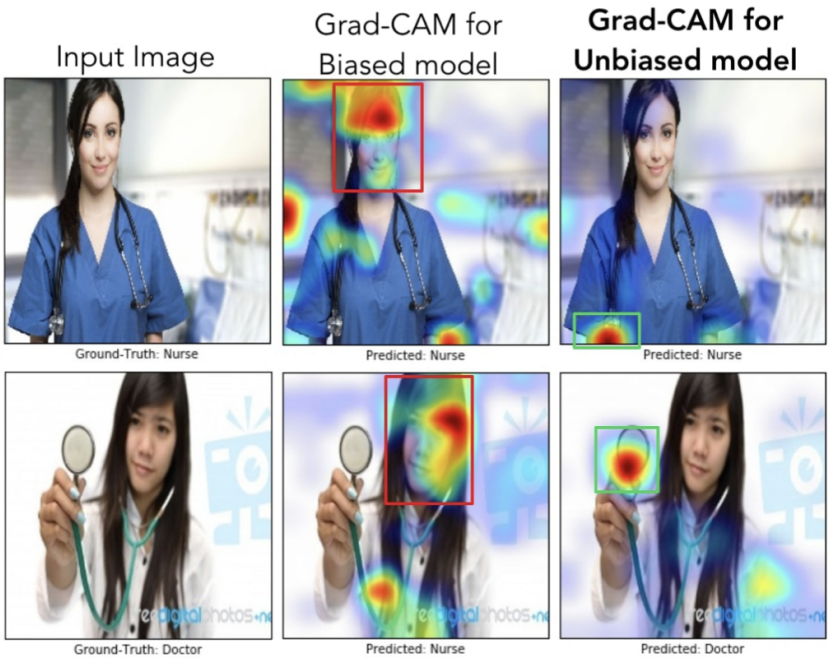


Fig. 8: In the first row, we can see that even though both models made the right decision, the biased model (model1) was looking at the face of the person to decide if the person was a nurse, whereas the unbiased model was looking at the short sleeves to make the decision. For the example image in the second row, the biased model made the wrong prediction (misclassifying a doctor as a nurse) by looking at the face and the hairstyle, whereas the unbiased model made the right prediction looking at the white coat, and the stethoscope.

Figure 2.8: TMP [40]

biological evolution and mimics the Darwin theory of the survival of the fittest. [41]. GA is a population-based algorithm for which the basic elements are chromosome representation, fitness selection and biological-inspired operators. The chromosomes represent the genes for each individual in the population and typically take the form of a binary string. Each position within the chromosome is called a *locus* and has two possible values (0 or 1). A fitness function is defined to assign a score for all chromosomes based on some optimization objective. This plays a role for the biologically inspired operators for which the main ones are selection, mutation and crossover. Selection is the process of selecting chromosomes based on their fitness score for further processing. In mutation, some of the loci within a chromosome are flipped and in crossover, chromosomes are merged to create offspring. GA has been implemented in many areas such as the traveling salesman problem [42], function optimization [43], adaptive agents in stock markets [44] and airport scheduling [45]. Wang et al. [46] note that a general drawback is a need for expertise when choosing parameters that match specific applications. They propose an accelerated genetic algorithm based on a Markov chain transition probability matrix to perform a guided search that reduces the number of parameter choices one has to make. The following introduction of this method is thus based on [46].

We define the binary population matrix $A_{ij}(t)$ at generation t, consisting of N rows denoting chromosomes $i \in \{0, 1, \ldots, N\}$ and L columns denoting the loci $j \in \{0, 1, \ldots L\}$. For our application, we let the locus represent an atom in the Kirigami pattern matrix which is flattened to fit the format of the population matrix. We carry forward the binary values with 0 meaning a removed atom and 1 a present atom. By the use of a fitness function $f(t)$, we sort the population matrix row-wise in descending order by fitness score, i.e. $f_i(t) \leq f_k(t)$ for $i \geq k$. In the spirit of Markov chains, we assume that some transitions probability exists for the transition between the current state $A(t)$ and the next state $A(t+1)$. We assume that this transition probability only takes into

account the mutation process, and thus we omit operators like crossover. For each generation, the chromosomes are sorted according to the fitness function and the chromosome at the i^{th} fittest place is assigned a ranking score $r_i(t)$ by some monotonic increasing ranking scheme. We take this to be

$$r_i(t) = \begin{cases} (i-1)/N', & i-1 < N' \\ 1, & \text{else} \end{cases}$$

with $N' = N/2$ from [46]. We assign a row mutation probability $a_i(t)$ meaning that the probability for a mutation will increase towards the lower fitness scores. For the considerations of mutation with respect to each locus in the columns of $A_{ij}(t)$, we define the count of 0's and 1's as $C_0(j)$ and $C_1(j)$ respectively. These are normalized as

$$n_0(j,t) = \frac{C_0(j)}{C_0(j)+C_1(j)}, \quad n_1(j,t) = \frac{C_1(j)}{C_0(j)+C_1(j)}.$$

We can thus describe the state of the j^{th} locus column as the state vector $\mathbf{n}(j,t) = (n_0(j,t), n_1(j,t))$. In order to direct the current population to a preferred state for locus j we consider the highest weight $W_i = 1 - r_i$ among the chromosomes for the case of the locus being 0 or 1 respectively. This corresponds to the targets

$$C_0'(j) = \max\{W_i | A_{ij} = 0;\ i = 1, \ldots, N\}$$
$$C_1'(j) = \max\{W_i | A_{ij} = 1;\ i = 1, \ldots, N\}.$$

These are normalized

$$n_0(j,t+1) = \frac{C_0'(j)}{C_0'(j)+C_1'(j)}, \quad n_1(j,t+1) = \frac{C_1'(j)}{C_0'(j)+C_1'(j)}. \tag{2.13}$$

to produce the target state vector $\mathbf{n}(j,t+1) = (n_0(j,t+1), n_1(j,t+1))$. This will serve as a direction for each locus to evolve in and thus we can formulate the Markov chain as

$$\begin{bmatrix} n_0(j,t+1) \\ n_1(j,t+1) \end{bmatrix} = \begin{bmatrix} P_{00}(j,t) & P_{10}(j,t) \\ P_{01}(j,t) & P_{11}(j,t) \end{bmatrix} \begin{bmatrix} n_0(j,t) \\ n_1(j,t) \end{bmatrix},$$

where the matrix represents the transition matrix. Since the probability must sum to one for the rows in the transition matrix we get

$$P_{00}(j,t) = 1 - P_{01}(j,t), \quad P_{11}(j,t) = 1 - P_{10}(j,t).$$

These conditions allow us to solve for the transition probability $P_{10}(j,t)$ in terms of the single variable $P_{00}(j,t)$

$$P_{10}(j,t) = \frac{n_0(j,t+1) - P_{00}(j,t) n_0(j,t)}{n_1(j,t)} \tag{2.14}$$
$$P_{01}(j,t) = 1 - P_{00}(j,t) \tag{2.15}$$
$$P_{11}(j,t) = 1 - P_{10}(j,t) \tag{2.16}$$

The remaining part is to define $P_{00}(j,t)$. We adopt the choice from [46] and start from $P_{00}(j,t=0) = 0.5$ and choose $P_{00}(j,t) = n_0(j,t)$ for the following generations. Thus for a locus $A_{ij}(t)$ we mutate it, changing the binary value, by the probability

$$p_{ij}(t) = \begin{cases} a_i(t) P_{01}(t), & A_{ij}(t) = 0 \\ a_i(t) P_{10}(t), & A_{ij}(t) = 1 \end{cases} \tag{2.17}$$

In summary, each generation update involves the following steps.

1. For generation t calculate the fitness score $f_i(t)$ of each chromosome i and sort the population matrix $A_{ij}(t)$ row-wise according to a descending score.
2. From a defined ranking scheme $r_i(t)$ set the chromosome mutation probability to $a_i(t) = r_i(t)$ and the weighting of each row $W_i(t) = 1 - r_i(t)$.
3. Calculate the target states Eq. (2.13) and the transition probabilities using Eq. (2.14) to (2.16) and $P_{00}(j,t=0) = 0.5$, $P_{00}(j,t>0) = n_0(j,t>0)$.
4. Mutate (flip) each locus $A_{ij}(t)$ by the p_{ij} given by Eq. (2.17).

Notice that this algorithm treats every locus as an independent gene. Thus, we do not incorporate any effects from spatial dependencies in the Kirigami pattern matrix.

2.6.0.1 Repair function

A numerical scheme for repairing the Kirigami matrix to correspond to a non-detached sheet. This was implemented in order to get candidates that are not immediately marked as a rupture. But it might be better placed in the Random walk section even though we did not make this algorithm before creating the random walk configurations. .

Part II

Simulations

Chapter 3

Summary

In this thesis, we have studied the nanoscale friction of a Kirigami graphene sheet under the influence of strain using molecular dynamics (MD) simulations. We have developed a numerical tool for generating diverse Kirigami designs which we have utilized to create a dataset for the frictional behavior depending on Kirigami pattern, strain, and loading. Our findings suggest that the frictional behavior of a Kirigami sheet is highly dependent on the geometry of the pattern and the strain conditions. We observed that the out-of-plane buckling can be associated with a non-linear friction-strain relationship which can be utilized to demonstrate a negative friction coefficient in a system with coupled load and strain. Moreover, we have investigated the possibility to use machine learning on this dataset and we have attempted an accelerated search for the optimization of various friction properties. Our findings imply that machine learning can be feasible for this approach, but additional data is required to establish a more reliable foundation for the prediction on new Kirigami patterns. In this chapter, we will provide a summary of our findings and draw conclusions based on the results obtained. Finally, we will suggest some topics for further research.

3.1 Summary and conclusions

3.1.1 Designing an MD simulation

We have designed an MD simulation for the examination of friction for a graphene sheet sliding on a silicon substrate. The key system features were the introduction of the pull blocks, defined as the end regions of the sheet with respect to the sliding direction, which was utilized for applying normal load and sliding the sheet. The pull blocks were made partly rigid and used to employ a thermostat as well. Through an analysis of the friction forces retrieved from sliding simulations, we have established a standardized metric for kinetic friction. In particular, we measured the force exerted by the substrate on the full sheet, including the pull blocks, with respect to the sliding direction. We then determined the kinetic friction as the force mean value of the last half of the simulation. The uncertainties were estimated based on the fluctuations in the running mean. We found that the assessment of static friction was ambiguous for our simulation and did not pursue this further. From the analysis of the force traces, friction force vs. time, we identify the friction behavior in our simulation domain as being in the smooth sliding regime. This is attributed to the choice of a relatively high sliding speed (20 m/s) and infinitely stiff springs for the tethering of the sheet. This was supported by a demonstration of a transition to the stick-slip regime through the use of softer springs and a decrease in sliding speed. By conducting a more systematic investigation of the effects of temperature, sliding speed, spring constant and timestep, we identified a set of default values based on numerical stability and computational cost. During this process, we aimed to select the variables that would maintain relatively stable measures for friction with moderate perturbations around these default values. We found that friction increased with temperature which is in disagreement with the Prandtl–Tomlinson model and most experimental results. However, this agrees with the predictions from the Frenkel-Kontorova models and other MD studies which attribute this to ballistic sliding. In the absence of clear indications from the investigation regarding an appropriate temperature, we opted for the standard choice of room temperature, 300 K. Furthermore, we found friction to increase with velocity as expected from other studies and the Prandtl–Tomlinson model, with some signs of phonon resonance at certain sliding speeds as well which aligns with the predictions from the Frenkel-Kontorova models. We chose a rather high velocity

of 20 m/s mainly for the consideration of computational costs. For the spring constant, we found decreasing friction with increasing stiffness of the springs. This is associated with the transition from a stick-slip-influenced regime toward smooth sliding and can be attributed to an underlying change in commensurability. This is predicted by the Frenkel-Kontorova models and supported by both numerical and excperimental results. The choice of an infinitely stiff spring was made from an assessment of the variation in friction with perturbations in the spring constant value. Finally, we confirmed that a timestep of 1 fs provides reasonable numerical stability. However, based on fluctuations with timestep we find that the uncertainty in the simulations might be higher than first estimated. For the non-strained Kirigami sheet, these fluctuations were on the order of ± 0.017 nN for the evaluation of the kinetic friction.

3.1.2 Generating Kirigami patterns

In order to investigate the effects of Kirigami design we have created a numerical tool for generating various patterns. By defining an indexing system for the hexagonal lattice structure we were able to define the Kirigami designs as a 2D binary matrix for numerical implementation. We have selected two macroscale designs, which we denote the *Tetrahedron* and *Honeycomb* patterns, based on their ability to exhibit out-of-plane buckling when subjected to strain. By digitalizing the designs to match the hexagonal graphene lattice, we found that the characteristic design features can be translated to the nanoscale, as we observed similar out-of-plane buckling in MD simulations. Through our numerical tool we were able to create an ensemble of perturbed unique variations which yielded approximately 1.35×10^5 and 2.025×10^6 unique configurations for the Tetrahedron and Honeycomb patterns respectively. When considering the possibility to translate the periodic patterns on the sheet, the number of possible patterns can be increased by approximately a factor of 100. To introduce some random design features, we have developed a tool for generating Kirigami patterns based on random walks. The tool includes mechanisms such as bias, avoidance of existing cuts, preference for maintaining a direction, and procedures for repairing the sheet for simulation purposes. In general, we found that the capabilities of the numerical tools for generating Kirigami designs exceeded our computational resources with regard to performing MD simulation under different loads and strains for each of the designs. Our MD-based dataset only utilized a subset of configurations with 9660 data points based on 216 Kirigami configurations (Tetrahedron: 68, Honeycomb: 45, Random walk: 100, Pilot study: 3). Hence, we argue that the Kirigami generative tool can be valuable for further studies on an extended dataset.

3.1.3 Friction control using Kirigami design and strain

We have investigated the frictional behavior of the Tetrahedron and Honeycomb patterns in comparison to a non-cut sheet under various strains and loads. Initially, we observed that straining the Kirigami sheets in a vacuum resulted in an out-of-plane buckling. When adding the substrate to the simulation this translated into a decreasing contact area with strain. We found the Honeycomb sheet to exhibit the most significant buckling with a corresponding reduction of relative contact to approximately 43%, whereas the non-cut sheet did not produce any significant buckling in comparison. For the Kirigami sheets, we found that friction initially increased with strain, which made for increasing friction with decreasing contact area. As the strain continued to increase the friction-strain curve exhibited highly non-linear trends with strong negative slopes as well (see **??**). During the full strain, the contact area was decreasing monotonically except for a slight increase just before rupturing. These results contradict the asperity theory hypothesis of decreasing friction with decreasing contact area, which is also supported by the predictions of the 2D Frenkel-Kontorova models suggesting increasing friction with (contacting) system size. Thus, we conclude that the contact area is not a governing mechanism for the friction-strain relationship observed. From the investigation of the friction-load relationship, we found that both friction and contact area increased slightly with load, but we were not able to find significant evidence of any linear relationship between friction and contact area as predicted by asperity theory. The non-cut sheet did not show any dependency on the strain as both the friction and the contact area remained constant with strain. Thus our findings suggest that a changing contact area and the strain-induced friction effects might be associated with an underlying mechanism related to the buckling of the sheet. When analyzing the independent effect of the non-strained Kirigami sheets, we found a slight increase in friction between the non-cut sheet and the Kirigami sheets. However, this increase was one order of magnitude lower than the friction changes induced by the strain in combination. Therefore, we can conclude that the observed friction behavior cannot be attributed solely to the effects of the non-strained Kirigami sheet or the tension induced by the strain in a non-cut sheet.

When considering the friction dependency with load, we generally found a weak dependency corresponding to a friction coefficient on the order of 10^{-4}–10^{-5} even though we could not confirm any clear relationship. This is best attributed to the superlubric state of the graphene sheet as seen in other studies as well. The slope of the friction-load curve was not considerably affected by the straining of the Kirigami sheet which led us to the conclusion that strain-induced effects are dominant in comparison to any load-related effects.

When considering our findings in light of previous related results we find it plausible that the governing mechanism for the observed friction effects is related to commensurability as predicted by the Frenkel-Kontorova models. Since we have observed extremely low friction in our system, we argue that the non-deformed sheet corresponds to an incommensurable configuration. This is supported by the fact that we were not able to lower the friction below the original starting point. Additionally, this also aligns with the observation that the introduction of the non-strained Kirigami designs increased friction slightly. Since the Kirigami designs correspond to the removal of atoms from the sheet, it can be hypothesized to relax the incommensurability to some extent. When the Kirigami sheet buckles during stretching, it allows for a considerable rearrangement of the atoms in contact with the substrate. Hence, it may transition in and out of commensurable configurations, which could explain the non-linear trend for the friction-strain curve. One way to test this hypothesis is to alter the simulation conditions such that the non-strained sheet starts in a commensurable phase. This might be achieved by a softening of the springs for tethering and a lowering of the sliding speed since this was found to yield stick-slip behavior which can be associated with a commensurable phase. Another way is to reorient the sheet or change the sliding direction as reported in both numerical and experimental results. Then we might find a transition from a commensurable to an incommensurable case during the initial straining which would result in a lowering in friction with respect to the starting point.

3.1.4 Capturing trends with machine learning

By utilizing the numerical tool for generating Kirigami designs we have created a MD-based dataset for the frictional behavior depending on Kirigami design, load and strain. The dataset reveals some general correlations with mean friction, such as a positive correlation to strain (0.77) and porosity (0.60), and a negative correlation to contact area (-0.67). These results align with the finding in the pilot study, suggesting that the change in friction is associated with cuts in the sheet (porosity) and a changing contract area indicating out-of-plane buckling.

By defining the friction property metrics: $\min F_{\text{fric}}$, $\max F_{\text{fric}}$, $\max \Delta F_{\text{fric}}$ and max drop (maximum decrease in friction with strain), we investigated the top design candidates within our dataset. From these results, we found no indication of the possibility to reduce friction with the Kirigami approach since the non-cut sheet provided the overall lowest friction. Furthermore, among the top candidates, we found that a flat friction-strain profile is mainly associated with little decrease in the contact area and vice versa. These observations are consistent with the results of the pilot study and support the hypothesis that commensurability plays a key role in governing the behavior of the system. In terms of the maximum properties, we observed an improvement compared to the values obtained in the pilot study, with the Honeycomb patterns exhibiting the highest scores. This indicates that the dataset contains some relevant information for optimizing these properties since it includes examples of design improvements.

For the machine learning investigation, we have implemented a VGGNet-inspired convolutional neural network with a deep "stairlike" architecture: C32-C64-C128-C256-C512-C1024-D1024-D512-D256-D128-D64-D32, for convolutional layers C with the number denoting channels and fully connected (dense) layers D with the number denoting nodes. The final model contains 1.3×10^7 parameters and was trained using the ADAM optimizer for a cyclic learning rate and momentum scheme. We trained the network for a total of 1000 epochs while saving the best model during training based on the validation score. The model validation performance gives a mean friction R^2 score of $\sim 98\%$ and a rupture accuracy of $\sim 96\%$. However, we got lower scores for a selected subset of the Tetrahedon ($R^2 \sim 88.7\%$) and Honeycomb ($R^2 \sim 96.6$) pattern based on the top 10 max drop property scores respectively. The scores obtained were lower, even though the selected configurations were partly included in the training data and the hyperparameter selection favored the performance on this selected set. Thus we conclude that these selected configurations, associated with a highly non-linear friction-strain curve, represent a bigger challenge for machine learning prediction. One interpretation is that these involve the most complex dynamics and perhaps that this is not readily distinguished from the behavior of the other configurations which constitute the majority of the dataset. By evaluating the ability of the model to rank the dataset based on property scores, we found that it was able to effectively represent the top three scores for the maximum categories.

However, the ranking for the minimum friction property was lacking, which we attribute to the requirement of higher prediction precision that the model did not meet To obtain a more accurate evaluation of the model's performance, we generated a test set using MD simulations for some of the random walk based suggestions obtained from the accelerated search. The results showed a significantly worse performance compared to the validation set, with a two-order of magnitude higher loss and a negative R^2 score for the mean friction property. The negative R^2 score suggests that the model's predictions were worse than simply predicting the mean value of the true data. However, by reevaluating the hypertuning and choosing solely based on validation loss, we still found poor results on the test set. This suggests that the inadequate performance is not solely due to a biased hypertuning process, but rather because our original dataset did not cover a sufficiently diverse range of Kirigami configurations. The validation scores indicate that the use of machine learning is a feasible approach for addressing this problem, as we were able to identify some general trends in the data. Nonetheless, from the test scores, it is evident that further improvements to the dataset are necessary in order to develop a reliable model.

3.1.5 Accelerated search for Kirigami patterns

Using the machine learning model we performed two types of accelerated search. One by evaluating the property scores of an extended dataset and another with the use of the genetic algorithm approach. For the extended dataset search, we used the developed pattern generators to generate 1.35×10^6 Tetrahedron, 2.025×10^7 Honeycomb and 10^4 Random walk patterns. The search results for the minimum friction property indicate a preference for low cut density. This aligns with the overall observation that the dataset does not provide any suggestions for a further reduction in friction.

The search for the maximum properties resulted in some minor score increases, but the suggested candidates were mainly overlapping with the original dataset. By investigating the sensitivity to translations of the Tetrahedron and Honeycomb patterns, we observed significant variations in the model's predictions with only minor translations. This can be attributed to a physical dependency since these translations affect the edge of the sheet. However, given the poor model performance on the test set, we believe it is more likely that this variation is due to an insufficiency in the model caused by the limitations of the dataset.

In our investigation of the genetic algorithm approach, we used a starting population that was based on the results from the extended dataset accelerated search, as well as some randomly generated initializations with different porosity values. However, this approach did not provide any noteworthy indication for new design structures worth more investigation. In general, the initialization of the population itself proved to be a more promising strategy than the genetic algorithm. We acknowledge that further effort could potentially yield useful results with the genetic algorithm approach. However, we believe that the current lack of promising results can be attributed to the uncertainty of the model which where the reason for not pursuing this any further.

By considering the Grad-CAM explanation method, we observed that the model predictions were often substantially reliant on the top and bottom edges of the Kirigami configurations. This was unexpected since these edges are not true edges but are connected to the pull blocks used in the simulation. Despite the model's uncertain predictions, we speculate that this may be due to the thermostat effects from the pull blocks. Therefore, we note this as a feature worth investigating in the simulations.

3.1.6 Negative friction coefficient

Based on our initial investigations of the Kirigami sheet, we have discovered a highly non-linear friction-strain relationship. By proposing a linear coupling between load and strain with ratio R, we found that these results suggest the possibility to utilize the negative slope on the friction-strain curve to achieve a negative friction coefficient. Based on the decrease in friction from the top to bottom of these curves, using the Tetrahedron $(7, 5, 1)$ and Honeycomb $(2, 2, 1, 5)$ pattern from the pilot study, we estimate that the average coefficient within this range will be $-R12.75\,\mathrm{nN}$ for the Tetrahedron pattern and $-R \cdot 2.72\,\mathrm{nN}$ for the Honeycomb pattern.

To investigate this hypothesis, we conducted a simulation with a coupling between load and sheet tension, mimicking a nanomachine attached to the sheet, using the Kirigami configurations from the Pilot study. We observed that the non-linear behavior in the friction-strain curve also translated into a non-linear friction-load relationship for the coupled system. Additionally, we found that the Honeycomb pattern exhibited a non-linear strain-tension curve which resulted in an almost discontinuous increase in friction for the initial increase in load. We attribute this feature to an unfolding process visually confirmed from the simulation frames. For the coupled system with a load-to-tension ratio of 6, we found regions in the friction-load curve with significant negative

slopes. By considering the maximum and minimum points for such regions we estimated the average friction coefficient to be -0.31 in the load range $F_N = [4.65, 6.55]$ nN for the Tetrahedron pattern and -0.38 in the range $F_N = [0.71, 4.31]$ nN for the Honeycomb pattern. These results can be scaled by adjusting the load-to-tension ratio.

Based on our investigations, we have found that the combination of Kirigami cuts and strain has significant potential for controlling friction. Specifically, we have demonstrated that by enforcing a coupling between load and strain through tension, it is possible to achieve a negative friction coefficient. Therefore, we believe that this approach could be promising for developing novel friction-control strategies.

3.2 Outlook

In this thesis, we have demonstrated that certain Kirigami designs exhibit non-linear friction behavior with strain. This discovery was made through an exploration of different designs, which invites further investigation into the underlying mechanisms of this phenomenon. To this end, it would be valuable to choose only one or two selected designs, such as the Tetrahedron and Honeycomb patterns, and study the effects of various physical parameters on the friction-strain curve.

First of all, we suggest an investigation of how the friction-strain curve depends on temperature, sliding speed, spring constant, and loads for an increased range $F_N > 100nN$. This is especially interesting in the context of physical conditions leading to a stick-slip behavior since our study takes a basis in smooth sliding friction. Moreover, it would be valuable to verify that the choices for relaxation time, pauses, interatomic potentials and substrate material are not critical for the qualitative observations found for the friction-strain relationship. Especially the Adaptive Intermolecular Reactive Empirical Bond Order (AIREBO) potential for the modeling of the graphene sheet might be of interest. In this context, it might also be useful to investigate the effects of excluding adhesion from the simulations.

In order to investigate the hypothesis of commensurability as a governing mechanism we suggest an investigation of the friction-strain curve for different scan angles. If commensurability is an important factor, we hypothesize that the friction-strain curve will exhibit different qualitative shapes at varying scan angles. Additionally, it may be interesting to investigate the friction-strain relationship under a uniform load to gain insight into how the loading distribution affects the out-of-plane buckling and associated commensurability effect.

Another topic worth exploring is the impact of scale and edge effects. This includes an investigation of scaling the ratio of the sheet area to the sheet edge length. However, the machine learning predictions also suggest a study of Kirigami-induced edge effects as we translate the patterns on the sheet. With this regard, we would also suggest a more detailed study of the effect of the thermostat in the pull blocks which is suggested to have a possible importance when judging from the Grad-CAM analysis.

Regarding the machine learning approach, our findings indicate that there is a significant need to expand the dataset. In order to get more insight into this issue one could use unsupervised clustering techniques like the t-Distributed Stochastic Neighbor Embedding (t-SNE) to visualize the distribution of Kirigami configurations in the dataset. Another valuable approach is the active learning method similar to that used by Hanakata et al. [6]. That is, we extend the dataset using the top candidates of a machine learning-driven accelerated search and repeat the process of training the model, searching for new candidates and extending the dataset. This provides a direction for the extension of the dataset which could lead to a more efficient approach to address the dataset problem. We note that one can also create a dataset based on a fixed Kirigami design and vary the physical parameters to support the investigations mentioned above. For both variations, we believe that the results could benefit from a consideration of more advanced model architectures and machine learning techniques. For instance, we suggest increasing the receptive field in the convolutional part of the model, by the use of bigger strides or dilated convolution. If we can develop a reliable machine learning model, it would invite further studies of inverse design methods such as GAN or diffusion models.

Appendices

Bibliography

[1]E. Gnecco and E. Meyer, *Elements of friction theory and nanotribology* (Cambridge University Press, 2015).

[2]Bhusnan, "Introduction", in *Introduction to tribology* (John Wiley & Sons, Ltd, 2013) Chap. 1, 1–?

[3]H.-J. Kim and D.-E. Kim, "Nano-scale friction: a review", International Journal of Precision Engineering and Manufacturing **10**, 141–151 (2009).

[4]K. Holmberg and A. Erdemir, "Influence of tribology on global energy consumption, costs and emissions", Friction **5**, 263–284 (2017).

[5]B. Bhushan, "Gecko feet: natural hairy attachment systems for smart adhesion – mechanism, modeling and development of bio-inspired materials", in *Nanotribology and nanomechanics: an introduction* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), pp. 1073–1134.

[6]P. Z. Hanakata, E. D. Cubuk, D. K. Campbell, and H. S. Park, "Accelerated search and design of stretchable graphene kirigami using machine learning", Phys. Rev. Lett. **121**, 255304 (2018).

[7]P. Z. Hanakata, E. D. Cubuk, D. K. Campbell, and H. S. Park, "Forward and inverse design of kirigami via supervised autoencoder", Phys. Rev. Res. **2**, 042006 (2020).

[8]L.-K. Wan, Y.-X. Xue, J.-W. Jiang, and H. S. Park, "Machine learning accelerated search of the strongest graphene/h-bn interface with designed fracture properties", Journal of Applied Physics **133**, 024302 (2023).

[9]Y. Mao, Q. He, and X. Zhao, "Designing complex architectured materials with generative adversarial networks", Science Advances **6**, eaaz4169 (2020).

[10]Z. Yang, C.-H. Yu, and M. J. Buehler, "Deep learning model to predict complex stress and strain fields in hierarchical composites", Science Advances **7**, eabd7416 (2021).

[11]A. E. Forte, P. Z. Hanakata, L. Jin, E. Zari, A. Zareei, M. C. Fernandes, L. Sumner, J. Alvarez, and K. Bertoldi, "Inverse design of inflatable soft membranes through machine learning", Advanced Functional Materials **32**, 2111610 (2022).

[12]S. Chen, J. Chen, X. Zhang, Z.-Y. Li, and J. Li, "Kirigami/origami: unfolding the new regime of advanced 3D microfabrication/nanofabrication with "folding"", Light: Science & Applications **9**, 75 (2020).

[13]OpenAI, *Dall-e2*, 2023.

[14]Midjourney, *Midjourney*, 2023.

[15]Z. Deng, A. Smolyanitsky, Q. Li, X.-Q. Feng, and R. J. Cannara, "Adhesion-dependent negative friction coefficient on chemically modified graphite at the nanoscale", Nature Materials **11**, 1032–1037 (2012).

[16]B. Liu, J. Wang, S. Zhao, C. Qu, Y. Liu, L. Ma, Z. Zhang, K. Liu, Q. Zheng, and M. Ma, "Negative friction coefficient in microscale graphite/mica layered heterojunctions", Science Advances **6**, eaaz6787 (2020).

[17]D. Mandelli, W. Ouyang, O. Hod, and M. Urbakh, "Negative friction coefficients in superlubric graphite–hexagonal boron nitride heterojunctions", Phys. Rev. Lett. **122**, 076102 (2019).

[18]R. W. Liefferink, B. Weber, C. Coulais, and D. Bonn, "Geometric control of sliding friction", Extreme Mechanics Letters **49**, 101475 (2021).

[19]M. Metzsch, *Tuning Frictional Properties of Kirigami Altered Graphene Sheets using Molecular Dynamics and Machine Learning*.

[20]A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales", Comp. Phys. Comm. **271**, 108171 (2022).

[21]E. M. Nordhagen, *LAMMPS simulator*.

[22]A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, "The atomic simulation environment—a python library for working with atoms", Journal of Physics: Condensed Matter **29**, 273002 (2017).

[23]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: an imperative style, high-performance deep learning library", in *Advances in neural information processing systems 32* (Curran Associates, Inc., 2019), pp. 8024–8035.

[24]J. Lederer, *Activation functions in artificial neural networks: a systematic overview*, 2021.

[25]P. Shankar, "A review on artificial neural networks", **3**, 166–169 (2022).

[26]A. Binder, *Lecture materials. in5400 - machine learning for image analysis*, [Online; accessed 08/05/2023], 2022.

[27]N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, *The computational limits of deep learning*, 2022.

[28]J. G. Sam Lau and D. Nolan, *Principles and techniques of data science, 11.4. stochastic gradient descent*, [Online; accessed 08/05/2023], 2020.

[29]D. P. Kingma and J. Ba, *Adam: a method for stochastic optimization*, 2017.

[30]T. Salimans and D. P. Kingma, *Weight normalization: a simple reparameterization to accelerate training of deep neural networks*, 2016.

[31]K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: surpassing human-level performance on imagenet classification*, 2015.

[32]S. Ioffe and C. Szegedy, *Batch normalization: accelerating deep network training by reducing internal covariate shift*, 2015.

[33]L. N. Smith, *A disciplined approach to neural network hyper-parameters: part 1 – learning rate, batch size, momentum, and weight decay*, 2018.

[34]B. Cunha, C. Droz, A. Zine, S. Foulard, and M. Ichchou, *A review of machine learning methods applied to structural dynamics and vibroacoustic*, 2022.

[35]S. Saha, *A comprehensive guide to convolutional neural networks — the eli5 way*, [Online; accessed 08/05/2023], 2018.

[36]L. Guerdan, *Diving into temporal convolutional networks*, [Online; accessed 08/05/2023], 2019.

[37]G. Cybenko, "Approximation by superpositions of a sigmoidal function", Mathematics of Control, Signals and Systems **2**, 303–314 (1989).

[38]Y. Bengio, "Practical recommendations for gradient-based training of deep architectures", in *Neural networks: tricks of the trade: second edition*, edited by G. Montavon, G. B. Orr, and K.-R. Müller (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012), pp. 437–478.

[39]L. N. Smith, *Cyclical learning rates for training neural networks*, 2017.

[40]R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: visual explanations from deep networks via gradient-based localization", International Journal of Computer Vision **128**, 336–359 (2019).

[41]S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future", Multimedia Tools and Applications **80**, 8091–8126 (2021).

[42]R Jiang, K. Szeto, Y. Luo, and D. Hu, "Distributed parallel genetic algorithm with path splitting scheme for the large traveling salesman problems", in Proceedings of conference on intelligent information processing, 16th world computer congress (2000), pp. 21–25.

[43]K. Szeto, K. Cheung, and S. Li, "Effects of dimensionality on parallel genetic algorithms", in Proceedings of the 4th international conference on information system, analysis and synthesis, orlando, florida, usa, Vol. 2 (1998), pp. 322–325.

[44]K. Y. Szeto and L. Fong, "How adaptive agents in stock market perform in the presence of random news: a genetic algorithm approach", in Intelligent data engineering and automated learning—ideal 2000. data mining, financial engineering, and intelligent agents: second international conference shatin, nt, hong kong, china, december 13–15, 2000 proceedings 2 (Springer, 2000), pp. 505–510.

[45]K. L. Shiu and K. Y. Szeto, "Self-adaptive mutation only genetic algorithm: an application on the optimization of airport capacity utilization", in Intelligent data engineering and automated learning–ideal 2008: 9th international conference daejeon, south korea, november 2-5, 2008 proceedings 9 (Springer, 2008), pp. 428–435.

[46]G. Wang, C. Chen, and K. Y. Szeto, "Accelerated genetic algorithms with markov chains", in *Nature inspired cooperative strategies for optimization (nicso 2010)*, edited by J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor (Springer Berlin Heidelberg, Berlin, Heidelberg, 2010), pp. 245–254.