

# Conversations with ChatGPT regarding the theory section

I am now starting a separate project, namely writing a report on all the machine learning methods I have used in the last month or so, and comparing them. Can you help me with starting out on that?

That sounds like a really solid project! A comparative report will both show what you've learned and help you connect the dots between the different methods. Let's sketch out a structure you can follow, then we can start filling in details together.

---

◆ **Suggested Report Structure**

**1. Introduction**

- What is the general problem you've been studying? (Regression, prediction, generalization, etc.)
- Why compare methods? (Bias-variance tradeoff, interpretability, computational efficiency, etc.)
- Brief overview of which methods you'll cover.

---

**2. Methods Overview**

For each method, dedicate a subsection where you describe:

- **Idea:** What problem is it solving?
- **Formula:** Core mathematical definition (keep it clear but concise).
- **Strengths:** When it works well.
- **Weaknesses:** Where it struggles.

From your work, this likely includes:

- **OLS (Ordinary Least Squares)**
- **Ridge Regression**
- **Gradient Descent for OLS/Ridge**
- **(Possibly) Cross-validation / bootstrap as evaluation tools**

↓

# Conversations with ChatGPT regarding the theory section

## 3. Theoretical Comparisons

- Bias and variance of OLS vs Ridge
- Closed-form vs Gradient Descent solution (computational differences, exact vs approximate solution)
- Effect of regularization ( $\lambda$ )

## 4. Experimental Results

- Dataset description (synthetic polynomial + noise, your setup with  $X$  and  $y$ )
- Heatmaps of learning rates vs iterations (gradient descent stability)
- Train/test MSE curves for OLS vs Ridge vs different  $\lambda$
- Any bootstrap estimates you tried

This section is your "evidence": the plots you've already produced.

## 5. Discussion

- Which method generalized better, and why?
- How did regularization affect variance of coefficients?
- What role did hyperparameters (learning rate, iterations,  $\lambda$ ) play?
- Link back to bias–variance tradeoff.

## 6. Conclusion

- Summarize main findings (e.g., "OLS unbiased but high variance; Ridge biased but stabilizes coefficients").
- Practical takeaway: when would you choose each method?



## Conversations with ChatGPT regarding the theory section

I dont want you to start writing up a bunch of stuff, I will dictate what I write about and where I do it, all I want is inputs from you on the way

Perfect  that's actually the best way to do it — you stay in control of the structure and writing, and I'll act as your "sparring partner" for inputs, clarifications, and making things sharper when needed.

So: you tell me which section / idea / method you're working on, what you've drafted (or are planning to draft), and I'll give you feedback, missing angles, or help tighten the math/wording.

👉 Where do you want to start – with OLS, with the introduction, or with laying out the methods list you'll include?

I am starting off with a theory section, I was thinking of starting this way:

The problems we look at in this report are approximating data sets  $y$  in  $\mathbb{R}^n$ , with polynomials

where we have that  $y = f(x) + \text{epsilon}$ , where we  $f$  is a continuous function, usually in the form of a polynomial, a trigonometric function or an exponential function. Rewrite that a bit sharper

Got it  Here's a sharper rewrite of what you drafted — keeping it clean and precise, but still simple:

The problems considered in this report concern approximating data vectors  $y \in \mathbb{R}^n$  using parametric models. We assume that the data are generated according to

$$y = f(x) + \varepsilon,$$

where  $f$  is an unknown continuous function (often taken to be a polynomial, trigonometric, or exponential function), and  $\varepsilon$  is a noise term.

Would you like me to also add a short remark right after that, like "*The goal of the methods studied here is to construct an estimator  $\hat{f}$  that approximates  $f$  as closely as possible while avoiding overfitting to  $\varepsilon$ .*" — or do you want to hold off on motivation now and just keep the bare setup?

# Conversations with ChatGPT regarding the theory section

what do you think about this introduction to the general problem setting

In this project, we are presented with a vector of data points in the form  $y \in \mathbb{R}^n$  which includes some noise  $\epsilon \sim N(0, \sigma^2)$ . Our data vector are generated with respect to some data points  $x \in \mathbb{R}^n$ . We assume that  $y$  can be written as

\$\$ y = f(x) + \epsilon, \$\$

where  $f$  is an unknown continuous function, such as a polynomial, trigonometric or exponential function. Equivalently, we can rephrase this as

\$\$ y = X\beta + \epsilon. \$\$

Where  $X$  is what we call a feature matrix  $X \in \mathbb{R}^{n \times p}$ , whose columns contain the features we use to approximate the data, which in our case will be polynomial orders of  $x$ . We assume there exists what we call a "true parameter", such that  $X\beta = f(x)$ . The goal of our various methods is to construct an estimate  $\hat{\beta}$  of the true parameter, which we get from the noisy data  $y$ . Thus, approximating the true signal  $f(x)$  with a linear model  $\tilde{y} = X\hat{\beta}$ .

This is already a really solid introduction 😊 You've set up the data model, the role of the feature matrix, the true parameter, and the estimation problem very clearly. Let me give you a slightly polished version — keeping everything you wrote, but tightening the flow and wording so it reads smoother:

In this project, we are presented with a data vector  $y \in \mathbb{R}^n$ , which includes random noise  $\epsilon \sim \mathcal{N}(0, \sigma^2)$ . The data are associated with input points  $x \in \mathbb{R}^n$ , and we assume that  $y$  can be expressed as

$$y = f(x) + \epsilon,$$

where  $f$  is an unknown continuous function, such as a polynomial, trigonometric, or exponential function. Equivalently, this model can be written in linear-algebraic form as

$$y = X\beta + \epsilon,$$

where  $X \in \mathbb{R}^{n \times p}$  is the feature matrix, whose columns contain the chosen basis functions (in our case, polynomial powers of  $x$ ). We assume there exists a true parameter vector  $\beta \in \mathbb{R}^p$  such that  $X\beta = f(x)$ . The goal of the methods studied in this report is to construct an estimate  $\hat{\beta}$  of the true parameter from the noisy data  $y$ , and thereby approximate the true signal  $f(x)$  through the linear model

$$\tilde{y} = X\hat{\beta}.$$

👉 This keeps exactly your structure, just cleans up repetition and makes it flow like a report.

Do you want me to also suggest a *final bridging sentence* at the end, something like "*We now turn to different estimation strategies, starting with ordinary least squares*" — or do you prefer to keep the intro standalone?

Squaring the differences between the data points and our model ensures that large differences are penalized, while differences less than 1 have a more negligible contribution. Thus encouraging the model to fit data points reasonably well, while heavily discouraging predictions that deviate substantially from the data points.

That's a really good phrasing — clear and intuitive 😊

If you want to make it just a touch sharper (without making it longer), you could adjust it like this:

"Squaring the differences between the data points and the model ensures that large deviations are penalized heavily, while deviations smaller than 1 contribute only negligibly. This encourages the model to fit data points reasonably well, while strongly discouraging predictions that deviate substantially from the observations."

Would you like me to also suggest how you could follow that up with one sentence that links this intuition to *why OLS specifically uses MSE* (i.e., because it leads to a convex optimization problem with a closed-form solution)?

# Conversations with ChatGPT regarding the theory section

in OLS, ridge and Lasso, are all the cost functions convex?

Yes — all three are convex  Let's go one by one:

## 1. OLS (Ordinary Least Squares)

Cost function:

$$J(\beta) = \frac{1}{n} \|y - X\beta\|^2.$$

- This is a quadratic function in  $\beta$ .
- Quadratics with positive semi-definite Hessians are convex.
- Since  $X^T X$  is positive semi-definite,  $J(\beta)$  is convex.  
 So OLS is convex.

## 2. Ridge Regression

Cost function:

$$J(\beta) = \frac{1}{n} \|y - X\beta\|^2 + \lambda \|\beta\|^2.$$

- This is the OLS cost plus an  $\ell_2$ -norm penalty.
- The  $\ell_2$  norm squared is convex.
- Sum of convex functions is convex.  
 So Ridge is convex.

## 3. Lasso Regression

Cost function:

$$J(\beta) = \frac{1}{n} \|y - X\beta\|^2 + \lambda \|\beta\|_1.$$

- The  $\ell_1$ -norm  $\|\beta\|_1 = \sum_i |\beta_i|$  is convex (even though it's not differentiable everywhere).
- Again, sum of convex functions is convex.  
 So Lasso is convex (but not smooth).

### Summary:

- OLS = convex quadratic, smooth.
- Ridge = convex quadratic + quadratic penalty, smooth.
- Lasso = convex quadratic + absolute-value penalty, convex but not differentiable.

Would you like me to also explain why **convexity matters** (i.e., guarantees global minimum, gradient descent won't get stuck in local minima)?

# Conversations with ChatGPT regarding the theory section

in ols the matrix  $X^T X$  is assumed to be invertible and positive semi-definite right

Close — let's pin this down carefully:

**1.  $X^T X$  is always positive semi-definite**

For any vector  $v \in \mathbb{R}^p$ :

$$v^T (X^T X) v = (Xv)^T (Xv) = \|Xv\|^2 \geq 0.$$

So  $X^T X$  is **symmetric positive semi-definite (PSD)** no matter what.

**2. When is  $X^T X$  invertible?**

- If the columns of  $X$  are **linearly independent** (i.e., full column rank  $p$ ), then  $X^T X$  is actually **positive definite** and invertible.
- If the columns are dependent (e.g. too many features, multicollinearity, or  $p > n$ ), then  $X^T X$  is singular  $\rightarrow$  not invertible.

**3. In OLS**

- The "classic" OLS formula  $(X^T X)^{-1} X^T y$  assumes  $X^T X$  is **invertible**, so yes: it assumes positive definite, not just semi-definite.
- If  $X^T X$  is only PSD and not invertible, then OLS is still defined but you use the **Moore–Penrose pseudoinverse** instead.

So the precise statement is:

- $X^T X$  is **always PSD**.
- In OLS, we usually assume it is **invertible (positive definite)** to get a unique solution.

so essentially, if it is invertible, we can use standard OLS, but it might not be invertible, hence it is more appropriate to use ridge for instance

Exactly You've got it:

- **OLS:** Works great if  $X^T X$  is invertible (full rank). You get a unique, unbiased solution.
- If  $X^T X$  is **not invertible** (e.g. multicollinearity, or more features than data  $p > n$ ), the OLS formula breaks down  $\rightarrow$  infinitely many solutions.
- **Ridge:** Fixes this by adding  $\lambda I$ :  
$$(X^T X + \lambda I)^{-1} X^T y.$$

Since  $\lambda I$  is strictly positive definite, the whole matrix is invertible no matter what. Ridge always has a unique, stable solution.

So yes:

- 👉 When invertibility is guaranteed  $\rightarrow$  OLS works.
- 👉 When invertibility is not guaranteed or the solution is unstable  $\rightarrow$  Ridge (or another regularizer) is more appropriate.

Do you want me to also explain the *numerical stability* angle — why even if  $X^T X$  is technically invertible, Ridge can still be preferable?

## Conversations with ChatGPT regarding the theory section

So I wanted to talk briefly about why one might not want to use OLS. I have a different section later where I go into the bias variance tradeoff in more detail, so I don't want to go too deep into bias and variance stuff here, but maybe mention one sentence of why it plays a part. I am not sure how to write it up though. This is what I have so far.

In a later section, we compute the bias and variance of different regression models. It turns out that under the standard linear model assumptions, OLS has no bias. Although this might make OLS the most natural approach to regression, it is not necessarily always the most effective one. There are a few reasons for this. For one, while the matrix  $(X^T X)^{-1}$  is always positive semi-definite, it can have eigenvalues that are 0 or very small. In the first case, the matrix would not be invertible, making an analytical expression for the optimal parameter impossible. In the second case, computing the inverse may lead to numerical instability, thus making OLS unpractical.

That's a really good draft 🎉 It's sharp, technical, and already sets up the motivation for Ridge nicely. I'd only tweak it slightly for flow and add a light touch of bias-variance (without going into the full tradeoff yet). Here's a refined version:

In a later section, we compute the bias and variance of different regression models. Under the standard linear model assumptions, OLS is unbiased, which might make it appear as the most natural approach to regression. However, it is not always the most effective one. One reason is that while the matrix  $X^T X$  is always positive semi-definite, it can have eigenvalues that are zero or very small. In the first case, the matrix is not invertible, making the OLS solution undefined. In the second case, inverting  $X^T X$  becomes numerically unstable, which can lead to large variance in the estimated coefficients. Thus, despite its unbiasedness, OLS can perform poorly in practice.

That way you:

- keep it compact,
- explain the mathematical/numerical issue,
- sneak in a mention of variance without diving into the full bias-variance discussion.

👉 Would you like me to also suggest a one-sentence bridge to Ridge here (like "This motivates regularization methods such as Ridge regression, which stabilize the solution")?

# Conversations with ChatGPT regarding the theory section

Yes that is what I thought I would mention next,

"This leads us to our next method Ridge regression, which attempts to fix these issues by adding a regularization parameter.

Perfect 🌟 You've already got the right idea — just smooth it a bit so it reads like a natural continuation of your paragraph. For example:

"This motivates the use of Ridge regression, which addresses these issues by adding a regularization term. The added penalty stabilizes the inversion of  $X^T X$  and reduces the sensitivity of the coefficients, at the cost of introducing some bias."

so I was thinking of doing something like this very roughly written up so you get the idea:

As mentioned,  $X^T X$  might have eigenvalues that are very small or equal to zero. An obvious way of correcting for this is by defining a regularization parameter  $\lambda > 0$ , and adding a matrix with  $\lambda$  on its diagonal entries to  $X^T X$ . This is the idea behind the cost function we define for ridge regression "cost function"

Not only does this help with invertibility and numerical stability, it also penalizes large coefficients. The size of the  $\lambda$  we choose will determine how hard we "punish" large coefficients. We will expand on why this is beneficial in a later section (the bias-variance trade off section that I will write later)

How is that introduction, of course it will be refined and corrected for typos and so on, but what do you think about the general idea? Too unprecise and undeveloped?

I think the general idea is spot on 🌟 — it flows naturally from your OLS section, it introduces  $\lambda$  in a concrete way, and it hints at both numerical stability *and* coefficient shrinkage without going too deep into bias-variance (since you'll handle that later).

A couple of thoughts:

- **Clarity:** Saying "*an obvious way of correcting for this*" could be rephrased as "*a natural way to address this*" — that makes it sound less casual.
- **Precision:** It's good to explicitly say that adding  $\lambda I$  ensures  $X^T X + \lambda I$  is positive definite and therefore always invertible.
- **Structure:** Right after introducing the cost function, it might be nice to write one sentence like "*Minimizing this cost leads to the Ridge estimator*" and then give the formula for  $\hat{\beta}_{\text{Ridge}}$ .

So, in short: the structure and level of detail you're aiming for is exactly right for a theory section. It motivates Ridge clearly, connects it to the OLS problem, and sets up the math.

👉 Do you want me to suggest a cleaned-up version of your draft (with the cost function and Ridge solution in there), while keeping the same length and style?

Btw I also think I should add what we mean by generalizable in the section I wrote about cost functions. Do you think this section is enough?

One of the most important aspects in determining if a method is good or not, is whether it is generalizable or not. Meaning, how well the model fits to unseen data. A way of testing this is splitting our data set into a training set and a testing set, where the training usually consists of around two thirds to four fifths of our data. We then compute our estimate based on the training data, and then see how large the cost function is with respect to the testing data.

Of course it needs to be written better, but yeah

Yes 👍 you're on the right track — and it's a good idea to add this, because otherwise "generalizable" might feel like hand-waving. Your draft covers the essentials (unseen data + train/test split). I'd just polish it a bit to make it tighter and more formal:

"An important aspect in evaluating a regression method is its ability to generalize, i.e., how well the model performs on unseen data. A common way to assess this is by splitting the dataset into a training set and a test set, typically allocating two thirds to four fifths of the data for training. The model parameters are estimated on the training set, and the resulting predictor is then evaluated on the test set by computing the cost function on the test data. A method is considered generalizable if it achieves low error not only on the training data but also on the test data."

That keeps your idea, but makes it read report-ready.

👉 Do you want me to also suggest a one-sentence link from here back to MSE (like "This is why MSE is not only minimized during training but also evaluated on a test set")? That would tie your cost function section and the generalizability section together nicely.

# Conversations with ChatGPT regarding the theory section

what do you think of this as an intro

As explained earlier, different methods have different bias and variance.

The bias of an estimator  $\hat{\beta}$  with respect to a true parameter  $\beta$  is given by the difference between its expectation and the true parameter. Namely

$$\text{Bias}(\hat{\beta}) = \mathbb{E}[\hat{\beta}] - \beta.$$

The variance of an estimator measures how much the estimator fluctuates if we compute it on different data sets.

A general expression for the variance of an estimator is given by

$$\text{Var}(\hat{\beta}) = \mathbb{E}[(\hat{\beta} - \mathbb{E}[\hat{\beta}])^2] = \mathbb{E}[\hat{\beta}\hat{\beta}^T] - \mathbb{E}[\hat{\beta}]^2,$$

although this formula is not always easy to work with in practice.

I think that's a very solid intro 🌟 It's clear, mathematical where it needs to be, but still explanatory. A couple of small suggestions to polish it:

1. The first sentence could be a bit more formal: instead of "*different methods have different bias and variance*", maybe say "*different estimation methods lead to different bias and variance properties*."
2. At the end, instead of "*not always easy to work with*", you could make it a bit more academic: "*often impractical to evaluate directly, and specific derivations are usually required*."

Here's a cleaned-up version with those tweaks:

"As explained earlier, different estimation methods lead to different bias and variance properties. The bias of an estimator  $\hat{\beta}$  with respect to the true parameter  $\beta$  is defined as the difference between its expectation and the true parameter, namely

$$\text{Bias}(\hat{\beta}) = \mathbb{E}[\hat{\beta}] - \beta.$$

The variance of an estimator measures how much it fluctuates when computed on different datasets. In general, the variance of an estimator  $\hat{\theta}$  can be expressed as

$$\text{Var}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}]) (\hat{\theta} - \mathbb{E}[\hat{\theta}])^T],$$

although this expression is often impractical to evaluate directly, and closed-form results must be derived for specific estimators."

That way, you sound precise and confident, but still approachable.

👉 Do you want me to also suggest a one-sentence bridge from here to your OLS/Ridge results (like "We now state the bias and variance of OLS and Ridge, which we computed earlier")?

◀ ▶ ⏪ ⏩ ⏴ ⏵

# Conversations with ChatGPT regarding the theory section

so how do I explain what the variance of the model is? as opposed to the variance of the estimator?

Great question  — the difference is subtle but important, and it's good to make it explicit in your report.

## Variance of the estimator ( $\hat{\beta}$ )

- This is what we've already derived:  
$$\text{Var}(\hat{\beta}) = \mathbb{E}[(\hat{\beta} - \mathbb{E}[\hat{\beta}])(\hat{\beta} - \mathbb{E}[\hat{\beta}])^T].$$
- It measures how much the estimated coefficients  $\hat{\beta}$  fluctuate across different training sets.

## Variance of the model / prediction ( $\hat{y}$ )

- Once we have an estimator  $\hat{\beta}$ , our model predicts  
$$\hat{y} = X\hat{\beta}.$$
- The randomness in  $\hat{y}$  comes from the randomness in  $\hat{\beta}$ .
- The variance of the model's predictions is therefore  
$$\text{Var}(\hat{y}) = X \text{Var}(\hat{\beta}) X^T.$$
- This tells us how much the predictions fluctuate across different training sets.

## In words

- **Estimator variance:** how much the coefficients jump around from dataset to dataset.
- **Model variance:** how much the *predictions* jump around from dataset to dataset.

They are directly connected, since prediction is just  $X\hat{\beta}$ .

## Conversations with ChatGPT regarding the theory section

this is what I have now

We see that the OLS variance depends directly on  $(X^T X)^{-1}$ .  
If  $X^T X$  has small eigenvalues, the variance can become very large, which Ridge corrects by adding  $\lambda$ .

From these derivations, we see that while Ridge has more bias than OLS, it also has less variance.  
This leads to the concept of the bias-variance tradeoff. In the OLS setting, this tradeoff can be described by the following equation

```
\begin{aligned} \text{label}{eq:bias_var_eq} \\ |\mathbb{E}| |(y - \hat{y})|^2 = \text{Bias}[\hat{y}]^2 + \text{Var}[\hat{y}] + \sigma^2. \\ \text{text}{appendix} \\ \end{aligned}
```

The bias of the model is the expected deviation of the mean of the model from the true data, meaning  
 $\text{Bias}[\tilde{y}] = \mathbb{E}[|\tilde{y} - \mathbb{E}[\tilde{y}]|^2]$ .

On the other hand the variance term explain how much the predictions fluctuate across different training sets, and is given by

```
\text{Var}[\tilde{y}] = \mathbb{E}[|\tilde{y} - \mathbb{E}[\tilde{y}]|^2] = \frac{1}{n} \sum_i (\tilde{y}_i - \mathbb{E}[\tilde{y}])^2.
```

As usual  $\sigma^2$  denotes the variance of the noise.

Equation `eqref{eq:bias_var_eq}` tells us that a low MSE requires both a low bias and a low variance.  
While the decomposition of the cost function given in equation `eqref{eq:bias_var_eq}` is OLS specific, the same intuition applies more broadly.

Methods without regularization, such as OLS, typically have low bias but can suffer from very high variance.

While methods with regularization introduces some bias, but usually have lower bias.

I now want to say something like, having a very low bias and variance at the same time is not that easy though. In general, bias is closely linked to the model complexity. Meaning how well the model can capture the structure of the data. A high bias will make the model almost useless since it will be very bad at approximating. However, if the complexity of the model is too high, we risk what we call overfitting.  
Meaning that the model  approximate the data well in training, and end up capturing the noise specific to the training data, rather than the general structure of the data. A good model should try to approximate

# Conversations with ChatGPT regarding the theory section

approximating. However, if the complexity of the model is too high, we risk what we call overfitting. Meaning that the model will approximate the data well in training, and end up capturing the noise specific to the training data, rather than the general structure of the data. A good model should try to approximate the true parameter beta, instead of too closely the epsilon part of  $y = X\beta + \epsilon$ , as that would mean that it approximates noise. A model with very little bias can get very big coefficients, which means high variance, which is precisely what Ridge prevents.

I want something like that, now of course that was horribly written, but maybe you could rewrite it better for me, and fill in gaps and clarify where needed

Absolutely  you've got the right ideas lined up — it just needs some polishing and tightening so it reads smoothly and rigorously. Here's a refined version of what you want to say:

Achieving both low bias and low variance simultaneously is generally difficult. In practice, bias is closely tied to the complexity of the model, i.e. how well the model is able to capture the underlying structure of the data. A model with high bias will systematically miss important patterns, making it almost useless for prediction — this is what we call underfitting. On the other hand, if the model is too complex, it risks overfitting: the model not only captures the true signal but also adapts to the random noise specific to the training data. In terms of the data model  $y = X\beta + \epsilon$ , a good model should approximate the true parameter  $\beta$ , while avoiding fitting too closely to the noise term  $\epsilon$ .

Models with very low bias often achieve this by producing large coefficients. However, large coefficients make the estimator highly sensitive to small perturbations in the training data, which translates into high variance. This is precisely the issue that Ridge regression addresses: by penalizing large coefficients, Ridge prevents the estimator from chasing the noise too closely, thereby reducing variance at the expense of introducing a small bias.

This keeps your structure, but:

- introduces *underfitting/overfitting* explicitly,
- ties back to  $y = X\beta + \epsilon$ ,
- explains why large coefficients  $\rightarrow$  variance,
- and makes Ridge the "solution" naturally.

 Do you want me to also help you sketch a short example sentence with polynomials (e.g. degree-2 underfits, degree-20 overfits), so the reader immediately has a concrete picture in mind?

I now have a figure I want to add to the article, it is just a screenshot from a book, how do I add this figure into latex, how do I cite it properly, how do I cite it within my text

Perfect, let's walk through it step by step 

## 1. Add the figure to your LaTeX project

Put the screenshot (say it's called `bias_variance.png`) in the same folder as your `.tex` file.

## 2. Include it with figure environment

`latex`  Copy

```
\begin{figure}[H]
    \centering
    \includegraphics[width=0.8\textwidth]{bias_variance.png}
    \caption{Illustration of the bias-variance tradeoff. Adapted from \cite{hastie2009elements}.}
    \label{fig:bias_variance}
\end{figure}
```

- `width=0.8\textwidth` scales the image (adjust if needed).
- The `\caption{}` is where you both describe and cite the source.
- The `\label{}` lets you reference it later.

If you don't already, make sure to `\usepackage{graphicx}` at the top of your document.

# Conversations with ChatGPT regarding the theory section

## 3. Cite the source properly

In your .bib file (if you're using BibTeX), add something like:

```
bibtex  
@book{hastie2009elements,  
    title={The Elements of Statistical Learning},  
    author={Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome},  
    year={2009},  
    publisher={Springer}  
}
```

 Copy

Then LaTeX will handle formatting it correctly depending on your bibliography style.

## 4. Reference it in your text

Inside the main text, you can write:

```
latex  
As shown in Figure~\ref{fig:bias_variance}, the test error typically decreases with model complexity up to a point, after which overfitting
```

 Copy

This way:

- "Figure (number)" is clickable and correct.
- The source is properly cited.

 That's the clean, academic way: figure → caption cites source → you can also cite it in-text.

 Do you want me to show you how to do this *without* a .bib file (i.e. manual citation), in case you don't want to set up BibTeX?

## how do we rephrase this properly

In a later section, we compute the bias and variance of different regression models.

It turns out that in our experiments, the OLS parameter has no bias. If the input data is highly correlated in specific ways, the OLS parameter can have bias, although we will not touch on that in this report.

Here's a sharper and more precise version of what you wrote:

In a later section, we compute the bias and variance of different regression models.

Under the standard linear model assumptions, the OLS estimator is unbiased.

However, if these assumptions are violated—for example, when the input features are highly correlated (multicollinearity) or when the model is misspecified—the OLS estimator can exhibit bias.

Since our focus here is on the standard case, we will not pursue these complications further in this report.

 This way you keep the technical accuracy, explain *why* bias could appear, and still make it clear you won't go into it.

Do you want me to also make it even shorter and more "report-style" (less explanatory)?

# Conversations with ChatGPT regarding the theory section

The screenshot shows a dark-themed chat interface. The user's messages are in white text on a dark background, while ChatGPT's responses are in white text on a light gray rounded rectangle.

ChatGPT's initial message:

okay I am now going to start explaining gradient descent

I was thinking about starting off with something like this:

In both OLS and Ridge, we have shown closed-form expressions for the optimal parameters. However, it is not always feasible to find the estimators analytically, hence forcing us to use numerical methods. Recall that our cost functions are convex. A natural approach is then to approach the global minimum bla bla gradient descent method jargon. We take the difference and iterate bla bla til we find the minimum.

This method is called gradient descent.

User response:

That's a solid starting point — here's a cleaned-up and more precise version that keeps your style but sharpens the math and flow:

In both OLS and Ridge regression, we have derived closed-form expressions for the optimal parameters. However, in more complex settings it is not always feasible to compute such estimators analytically, which forces us to rely on numerical methods. Since our cost functions are convex, a natural approach is to minimize them iteratively using **gradient descent**.

The basic idea is simple: starting from an initial guess for the parameter vector, we repeatedly update it by moving in the direction of the negative gradient of the cost function.

The gradient tells us the direction of steepest increase, so stepping in the opposite direction gradually moves us toward the minimum. This iterative process continues until convergence, at which point the parameter values are (approximately) optimal.

ChatGPT's message:

This way you (1) motivate *why* we need gradient descent, (2) connect it to convexity, and (3) give the intuition in plain words.

Do you want me to also insert the **general update formula**

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} J(\theta^{(t)})$$

here already, or save the math for just after the introduction?

ChatGPT's message:

I am struggling to formulate this idea precise and concise, without using a figure

The choice of learning rate can affect how good our approximation of the minimum is. If we choose a learning rate that is too large we risk going past the minimum, which will result in an oscillating behavior between two points close to the minimum, until the method stops iterating. Which results in a bad approximation.

help me out

User response:

Got it — here's a concise but precise version of your idea, without relying on a figure:

The choice of learning rate directly affects the quality of the approximation. If the learning rate is too large, the updates may overshoot the minimum, causing the algorithm to oscillate around it instead of converging. If it is too small, the updates will be very slow, requiring many iterations to approach the minimum.

ChatGPT's message:

This keeps the essence:

- too large → overshoot/oscillation, bad approximation,
- too small → slow convergence.

Do you also want me to add one sentence about the "ideal" case (when the learning rate is well-chosen, we converge efficiently to the minimum)?

# Conversations with ChatGPT regarding the theory section

what do you think so far:

In both OLS and Ridge regression, we have presented closed-form expressions for the optimal parameters. However, in some cases it is not always feasible to compute such estimators analytically, forcing us to use numerical methods.

||||

We mentioned earlier that the cost functions we consider in this project are convex. A natural way of minimizing them numerically is starting with an initial guess of the estimator, often the zero vector. We then iteratively update the estimator by moving in small steps in the direction of the negative gradient of the cost function. We continue this iterative process until we see that our estimator begins to converge, at which point we will have found an approximate minimum of the cost function. The main workhorse behind this method is the following fundamental result from multi-variable calculus.

\begin{theorem}

Assume  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is differentiable in a point  $a \in \mathbb{R}^n$ . The gradient  $\nabla f(a)$  will point in the direction where  $f$  has the steepest increase in the point  $a$ , and the slope of  $f$  in this direction is  $\|\nabla f(a)\|$ .

\end{theorem}

This approach is the main idea behind what we call "gradient descent". In gradient descent, the iterative formula for the parameter  $\theta$  for a cost function  $C(X, \theta)$  is given by

\$\$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla C(X, \theta^{(i)})

where the step size  $\eta > 0$  is what we call the "learning rate". In the OLS case, the iterative formula becomes

\$\$\theta^{(i+1)} = \theta^{(i)} - \eta \frac{1}{n} X^T (X \theta^{(i)} - y), \text{ appendix}

while in the Ridge case, the iterative formula is

\$\$\theta^{(i+1)} = \theta^{(i)} - \eta \left( \frac{2}{n} X^T (X \theta^{(i)} - y) + 2\lambda \theta^{(i)} \right), \text{ appendix}

\$\$

# Conversations with ChatGPT regarding the theory section

The choice of learning rate can affect how good our approximation of the minimum is. If we choose a learning rate that is too large we risk going past the minimum, which will result in the algorithm oscillating around the minimum instead of converging. A smaller learning rate generally leads to a better approximation, but may require many iterations to approach the minimum. When doing gradient descent, it is common to define a stopping criterion  $\|\delta\| > 0$ , where the algorithm terminates once the norm of the gradient at the current step becomes less than  $\|\delta\|$ , i.e. when

```
$$
\|\nabla_{\theta} C(X, \theta^{(i)})\| < \|\delta\|.
$$
```

Typically, we choose  $\|\delta\|$  to be a very small number, indicating that further iterations would only make negligible improvements.

This is already very strong 🤩 — it flows well, mixes intuition with math, and your theorem anchors the idea rigorously. Just a few tweaks I'd suggest for clarity and precision:

## Main adjustments

1. "Not always feasible" phrasing → say explicitly why (large datasets, non-closed-form solutions).
2. The theorem → super nice! Maybe tighten the wording a bit.
3. Gradient descent formulas → in the Ridge formula, you wrote  $2\lambda\theta^{(1)}$ , I think you mean  $2\lambda\theta^{(i)}$ .
4. Learning rate and stopping criterion → you nailed the explanation, just smooth it slightly.

## Polished version

In both OLS and Ridge regression, we have presented closed-form expressions for the optimal parameters. However, in more complex problems or very large datasets it is not always feasible to compute such estimators analytically, which forces us to rely on numerical methods.

We have already noted that the cost functions considered in this project are convex. A natural way of minimizing them numerically is to start with an initial guess for the estimator, often the zero vector, and then iteratively update it by moving in small steps in the direction of the negative gradient of the cost function. Repeating this process gradually brings the estimator closer to the minimum, and once the updates become negligible we treat the algorithm as converged.



The key fact underlying this approach is the following result from multivariable calculus:

```
\begin{theorem}
Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be differentiable at a point  $a \in \mathbb{R}^n$ . The gradient  $\nabla f(a)$  points in the direction of the steepest increase of  $f$  at  $a$ , and the slope in this direction equals  $\|\nabla f(a)\|$ .
\end{theorem}
```

This observation forms the basis of *gradient descent*. The iterative update rule for the parameter vector  $\theta$  given a cost function  $C(X, \theta)$  is

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta} C(X, \theta^{(i)}),$$

where the step size  $\eta > 0$  is called the *learning rate*.

In the OLS case, this becomes

$$\theta^{(i+1)} = \theta^{(i)} - \eta \frac{2}{n} X^T (X \theta^{(i)} - y), \quad \text{appendix}$$

while in the Ridge case we obtain

$$\theta^{(i+1)} = \theta^{(i)} - \eta \left( \frac{2}{n} X^T (X \theta^{(i)} - y) + 2\lambda\theta^{(i)} \right). \quad \text{appendix}$$

The choice of learning rate directly affects convergence. If it is too large, the algorithm risks overshooting the minimum and oscillating around it without settling. If it is too small, the convergence will be very slow, requiring many iterations.

To formalize when to stop, it is common to introduce a stopping criterion  $\delta > 0$ . The algorithm terminates once the norm of the gradient becomes smaller than this threshold,

$$\|\nabla_{\theta} C(X, \theta^{(i)})\| < \delta,$$

indicating that further updates would yield only negligible improvements.

# Conversations with ChatGPT regarding the theory section

my reference for the theorem is this book  
<https://www.gyldendal.no/faglitteratur/ingenioerfag/fellesemner/flervariabel-analyse-med-lineær-algebra/p-10017557/>  
how do I reference this?

Cool, I found the necessary info to build a proper reference for that book. Here's how you can cite *Flervariabel analyse med lineær algebra* by Lindstrøm & Hveberg in your bibliography and how to refer to it in your LaTeX document.

**Bibliography (BibTeX) entry**

Add this to your .bib file:

```
bibtex
@book{lindstrom2015flervariabel,
  title      = {Flervariabel analyse med lineær algebra},
  author     = {Tom L. Lindstr{\o}m and Klara Hveberg},
  year       = {2015},
  edition    = {2},
  publisher  = {Gyldendal akademisk},
  address    = {Oslo},
  isbn       = {9788205472402}
}
```

**How to cite in text**

In your LaTeX file, after you've added the BibTeX entry, you can refer to the theorem you used like this:

```
latex
As shown in Lindstrøm & Hveberg \cite{lindstrom2015flervariabel}, the gradient \(\nabla f(a)\) at a point  $a$  gives the direction of
```

Or:

```
latex
According to Lindstrøm & Hveberg \cite[Ch.~X, p.~Y]{lindstrom2015flervariabel}, the gradient at a point determines the direction in which
```

- Replace Ch.~X, p.~Y with the actual chapter/page if you want to be precise.
- The `\cite{lindstrom2015flervariabel}` will produce a reference number or author–year label depending on your bib style.

Let me know if you want me to also draft how to mention this reference in your figure caption or theorem environment (so it's seamless in your theory section)!

# Conversations with ChatGPT regarding the theory section

Isn't it better to cite with a footnote? What is more standard in a report like mine do you think?

Good question 

It really depends on the style and audience of your report:

-  **Scientific/technical reports (math, ML, engineering, etc.)**  
→ Standard is to use **in-text citations** like `\cite{...}` and keep all references in a bibliography at the end. This way, the reader can see where results come from and easily look them up.
-  **Expository or course reports/essays (less formal)**  
→ Sometimes a footnote citation (e.g. "see Lindstrøm & Hveberg") is acceptable, but it's less common in math-heavy writing.
-  **Rule of thumb:**
  - If you're citing a theorem or a definition from a textbook → use `\cite` in-text.
  - If you're making a side comment like "*this explanation follows the one in [book]*" → a footnote is fine.

For your report — since it's mathematical, structured, and already uses equations and references — I'd say:

👉 Stick with **bibliography citations** (`\cite`) for theorems and definitions.

But if you want, you can mix styles:

- Use `\cite{...}` in the main text for core results.
- Add a `\footnote{See Lindstrøm & Hveberg \cite{lindstrom2015flervariabel}}` if you want to keep the flow uncluttered.

Would you like me to show you how to **combine both** (cite in theorem with a footnote that points to the bibliography entry)? That way you get the clean look of a footnote but still keep proper references.

I just write this then:

The main workhorse behind this method is the following fundamental result from multi-variable calculus, translated from `\cite{flervariabelkalkulus}`.

```
\begin{theorem}
  Assume $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable in a point $a \in \mathbb{R}^n$. The gradient $\nabla f(a)$ points in the direction of the steepest increase of $f$ at $a$, and the slope in this direction equals $\|\nabla f(a)\|$.
\end{theorem}
```

Yes  that looks very good — clean, standard, and precise.

A couple of optional refinements you might consider:

1. Instead of "*translated from*", some styles prefer "*adapted from*" or just "*see*", since you aren't giving a literal translation but the mathematical content:  
`latex`   
The main workhorse behind this method is the following fundamental result from multivariable calculus (see `\cite{flervariabelkalkulus}`).
2. You could also move the citation into the theorem environment if you want it directly tied to the statement:  
`latex`   

```
\begin{theorem}[{\cite{flervariabelkalkulus}}]
  Assume $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable at a point $a \in \mathbb{R}^n$. The gradient $\nabla f(a)$ points in the direction of the steepest increase of $f$ at $a$, and the slope in this direction equals $\|\nabla f(a)\|$.
\end{theorem}
```

That way, the reference sits neatly in the theorem header.

Do you want the theorem to look more "*standalone textbook style*" (with the citation in the header), or more "*inline flow*" (with the citation in the surrounding text)?

# Conversations with ChatGPT regarding the theory section

how do I remove the number one from the theorem

Ah, that's a LaTeX styling question 

By default, `\newtheorem{theorem}{Theorem}` numbers your theorems (Theorem 1, Theorem 2, ...). If you want just "Theorem" with no number, define it as *unnumbered*.

In your preamble, instead of:

```
latex
\newtheorem{theorem}{Theorem}
```

use:

```
latex
\newtheorem*{theorem}{Theorem}
```

The \* version tells LaTeX: "don't number these."

So your environment becomes:

```
latex
\begin{theorem}
  Assume $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable at a point $a \in \mathbb{R}^n$.
  The gradient $\nabla f(a)$ points in the direction of the steepest increase of $f$ at $a$,
  and the slope in this direction equals $\|\nabla f(a)\|_2$.
\end{theorem}
```

and it will just display as **Theorem**.

👉 Do you want *all* your theorems unnumbered, or just this one? If only this one, I'll show you how to keep the usual numbered version too.

okay I now want to move into stochastic and momentum gradient descent, I want to motivate them. Which one is more natural to start with do you think?

Nice — good point in your report to transition!

Between **stochastic gradient descent (SGD)** and **momentum gradient descent**, it's more natural to introduce **SGD** first. Here's why:

- **Flow of motivation:**  
You just explained vanilla gradient descent, which requires computing the full gradient  $\nabla_{\theta} C(X, \theta)$  at every step. That's costly when  $n$  (the dataset size) is large. → This naturally motivates the idea: *what if we approximate the gradient using only part of the data?* → leads to **SGD**.
- **Momentum:**  
Momentum addresses a different pain point: gradient descent (and even SGD) can be *slow to converge* or *oscillate in narrow valleys*. Once the reader understands SGD, it's natural to say: *but it can be noisy / unstable — momentum smooths it out*.

👉 So I'd suggest the order:

1. **Vanilla Gradient Descent** (done)
2. **Stochastic Gradient Descent** (motivate: speed + scalability)
3. **Momentum** (motivate: stability + faster convergence, reduces oscillations)

Would you like me to draft a **short transition paragraph** that bridges from your gradient descent section into SGD in a natural way?

## Conversations with ChatGPT regarding the theory section

lets start with what the benefits of SGD over normal GD is, does it help with cost functions that are not globally convex but maybe only locally convex?

Exactly – good instinct 

The main benefits of SGD over standard (batch) gradient descent are:

## 1. Efficiency / scalability

- In full gradient descent, each update requires computing  $\nabla_{\theta} C(X, \theta)$  over the *entire dataset* of size  $n$ .
  - For large  $n$ , this is costly.
  - SGD instead updates using only *one* (or a mini-batch of) randomly chosen data point(s). This makes each update very cheap.

## 2. Escaping local minima / saddle points

- In theory, gradient descent works beautifully if  $\mathcal{C}$  is globally convex (like OLS, Ridge).
  - But in more general ML problems (e.g. neural networks),  $\mathcal{C}$  is usually non-convex. Full-batch gradient descent tends to get stuck in local minima or flat saddle points.
  - SGD introduces randomness into the updates (since each step uses a random sample). This noise can help the algorithm “kick” itself out of shallow local minima or saddle points, exploring the parameter space better.

### **3. Faster progress in practice**

- Even if the updates are noisier, SGD often converges faster *in wall-clock time*, since it can update parameters many times before full gradient descent would even finish computing a single step.

So to answer your question:

- ✓ Yes — while it doesn't guarantee global optimization, SGD is less likely than vanilla GD to get stuck in local minima or saddle points because of its inherent randomness.

Do you want me to phrase this in the same formal/report style you've been writing (like with OLS/Ridge and gradient descent), or do you want a more intuitive, "teaching-style" explanation first that you can refine?

# Conversations with ChatGPT regarding the theory section

what do you think about this

While gradient descent indeed is an effective way to approximate the estimators for both Ridge and OLS, there are situations where the classic gradient descent method can be ineffective. Firstly, in more complex machine learning models, the cost functions is usually not convex. Instead they might have several local minimum and saddle points. In these cases, the gradient descent algorithm can get stuck in one of these points, when there might be a more optimal minimum points somewhere else. Secondly, if we have many data points, the gradient descent algorithm can become very computationally expensive. At each iteration we compute the gradient

$$\nabla_{\theta} C(X, y; \theta),$$

meaning for  $n$  data points, we compute a matrix-vector product with all  $n$  rows of  $X$ .

I think that's a really good start — you've captured the two main issues clearly. I'd only suggest tightening the language a little and fixing some plural/singular slips to make it flow better in your report. Here's a polished version:

While gradient descent is an effective way to approximate the estimators for both Ridge and OLS, there are situations where the classic method can be inefficient or even ineffective.

Firstly, in more complex machine learning models, the cost function is typically non-convex. Such functions may have several local minima and saddle points. In these cases, the gradient descent algorithm can get trapped in one of these points, even though a more optimal minimum exists elsewhere.

Secondly, if the dataset is large, gradient descent can become computationally expensive. At each iteration, the gradient

$$\nabla_{\theta} C(X, y; \theta)$$

must be computed. For  $n$  data points, this requires a matrix–vector product involving all  $n$  rows of  $X$ . Since this operation must be repeated many times, the total cost of batch gradient descent quickly becomes prohibitive when  $n$  is large.

I would also like to say something like this:

Thirdly, our gradient descent depends a lot on our choice of learning rate. Ideally, we would like to be able to adapt or learning rate depending on the situation and so on(expand on this please)

Yes! That's a really nice third point to add. It motivates both momentum and more advanced adaptive methods (like Adam), even if you don't implement them. Here's how you could phrase it in your report style:

Thirdly, the performance of gradient descent depends heavily on the choice of learning rate  $\eta$ . If  $\eta$  is too small, convergence will be very slow; if it is too large, the algorithm may overshoot the minimum or even diverge. Ideally, we would like the learning rate to adapt to the cost landscape: taking larger steps when the gradient is smooth and well-behaved, and smaller steps when the landscape is steep or noisy. Several extensions of gradient descent, such as momentum methods and adaptive learning rate algorithms, have been developed to address this issue.

That ties perfectly into your momentum subsection.

Do you want me to merge all three points (non-convexity, large data, learning rate) into one polished paragraph, so it flows as a single motivation for introducing SGD and momentum?

## Conversations with ChatGPT regarding the theory section

The screenshot shows a dark-themed ChatGPT interface. A user message at the top right reads: "I feel like it makes more sense to start off with momentum based GD, and then mentioned that there are other ways to adaptively change learning rate, which we will show in our implementation section. Such as ADAgrad, RMSprop and ADAM". Below this, a response from ChatGPT says: "Yes, that makes a lot of sense 👍". The user then asks, "Here's why:" followed by a bulleted list of reasons. After the list, the user asks, "That way:" followed by a numbered list of three steps. Finally, the user asks, "Would you like me to draft a polished momentum GD subsection for you now (definition, intuition, update rule, and transition to adaptive methods)?"

I feel like it makes more sense to start off with momentum based GD, and then mentioned that there are other ways to adaptively change learning rate, which we will show in our implementation section. Such as ADAgrad, RMSprop and ADAM

Yes, that makes a lot of sense 👍

Here's why:

- Momentum GD is conceptually closer to standard GD: you still use the full dataset gradient, but you smooth updates to avoid oscillations and speed up convergence.
- That's an easy bridge from vanilla GD → momentum GD.
- Once you've introduced the idea of *modifying the update rule to improve stability/convergence*, it's natural to say: "*There are also other adaptive learning rate methods, such as AdaGrad, RMSprop, and Adam, which we will return to in the implementation section.*"

That way:

1. You first motivate the **limitations of vanilla GD** (your three points).
2. You show **momentum GD** as a concrete, intuitive fix.
3. You *hint* at more advanced adaptive methods without needing to explain them in theory.

Would you like me to draft a polished **momentum GD subsection** for you now (definition, intuition, update rule, and transition to adaptive methods)?

# Conversations with ChatGPT regarding the theory section

just to get something clear, the point of momentum based gradient descent is faster convergence, not escaping local minima?

Exactly ✓ — you've got it.

- Momentum-based GD:
  - Main purpose = faster and smoother convergence.
  - It reduces oscillations (especially in "valley" landscapes where one direction has high curvature and another is flat).
  - Think of it like giving the optimizer "inertia" so it glides along the valley floor instead of zig-zagging.
  - It can incidentally help escape shallow local minima because the momentum carries it forward — but that's not the primary motivation.
- Stochastic GD:
  - Main purpose = make GD feasible for very large datasets (cheaper per update).
  - Secondary effect = the noise from random samples can help escape local minima/saddle points.

So:

- Momentum = stability + speed.
- SGD = scalability + exploration.

👉 Would you like me to add one clarifying sentence like that at the end of your momentum subsection, so your reader doesn't confuse it with SGD?

 MathMLbook.pdf  
PDF  
create a bib reference for this book

Here's a good BibTeX entry for *Mathematics for Machine Learning* (based on the metadata in your PDF and publisher info):

```
bibtex
@book{deisenroth2020mml,
  title    = {Mathematics for Machine Learning},
  author   = {Marc Peter Deisenroth and A. Aldo Faisal and Cheng Soon Ong},
  year     = {2020},
  publisher = {Cambridge University Press},
  address  = {Cambridge},
  url      = {https://mml-book.com},
  isbn     = {9781108455145}
}
```

This matches the official Cambridge University Press publication.

Would you like me to also make a shorter @misc version (pointing only to the free PDF at [mml-book.com](https://mml-book.com)), in case you want to distinguish between citing the official book and citing the online draft?

## Conversations with ChatGPT regarding the theory section

it has to be something like this right??

A way to address problem of choosing a learning rate is using `\text{momentum-based gradient descent}`. The idea behind this is to let each iterative step have memory of the last, and adjust its step size accordingly.

By doing this, we can avoid oscillating around the edges of a minimum point, and instead converge more smoothly more smoothly.

We follow the mathematical formulation of chapter \$7\$ in `\cite{deisenroth2020mml}`.

We define the update between two iterations as  $\Delta \theta^{(i+1)} = \theta^{(i+1)} - \theta^{(i)}$ . We then define a new iterative algorithm

```
$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta} C(X, \theta^{(i)}) + \alpha \Delta \theta^{(i)},$$  
where  $\alpha \in [0,1]$  is the \text{momentum parameter}, and  $\eta > 0$  is the learning rate. Hence,  
$$\Delta \theta^{(i+1)} = \theta^{(i+1)} - \theta^{(i)} = \text{compute?}$$
```

This helps because this and that.

Yes  you're on exactly the right track — you've basically written the structure as the book does, just with your own notation. Let me fill in the missing algebra for

$$\Delta \theta^{(i+1)} = \theta^{(i+1)} - \theta^{(i)}.$$

Starting from your iterative rule:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta} C(X, \theta^{(i)}) + \alpha \Delta \theta^{(i)}.$$

Subtract  $\theta^{(i)}$  from both sides:

$$\Delta \theta^{(i+1)} = -\eta \nabla_{\theta} C(X, \theta^{(i)}) + \alpha \Delta \theta^{(i)}.$$

## Conversations with ChatGPT regarding the theory section

So the final compact system is:

$$\Delta\theta^{(i+1)} = \alpha\Delta\theta^{(i)} - \eta\nabla_{\theta}C(X, \theta^{(i)}),$$

and then

$$\theta^{(i+1)} = \theta^{(i)} + \Delta\theta^{(i+1)}.$$

So your section could read like this:

A way to address the problem of choosing a learning rate is using *momentum-based gradient descent*.

The idea behind this is to let each iterative step have memory of the last, and adjust its step size accordingly.

By doing this, we can avoid oscillating around the edges of a minimum point, and instead converge more smoothly.

We follow the mathematical formulation of Chapter 7 in `\cite{deisenroth2020mml}`.

We define the update between two iterations as

$$\Delta\theta^{(i+1)} = \theta^{(i+1)} - \theta^{(i)}.$$

The iterative algorithm is then given by

$$\theta^{(i+1)} = \theta^{(i)} - \eta\nabla_{\theta}C(X, \theta^{(i)}) + \alpha\Delta\theta^{(i)},$$

where  $\alpha$  is the *momentum parameter*, and  $\eta$  is the learning rate.

Equivalently, the update rule can be written as

$$\Delta\theta^{(i+1)} = \alpha\Delta\theta^{(i)} - \eta\nabla_{\theta}C(X, \theta^{(i)}).$$

This formulation shows explicitly how each update depends both on the current gradient and on the previous update. The momentum term acts like inertia, smoothing oscillations and speeding convergence in narrow valleys of the cost function.

👉 Do you want me to phrase the “inertia / heavy ball” analogy too (like the book), or keep it purely mathematical?



## Conversations with ChatGPT regarding the theory section

this is what I have now,

what do you think?

A way to address problem of choosing a learning rate is using `\textit{momentum-based gradient descent}`. The idea behind this is to let each iterative step have memory of the last, and adjust its step size accordingly.

By doing this, we can avoid oscillating around the edges of a minimum point, and instead converge more smoothly more smoothly.

We follow the mathematical formulation of chapter \$7\$ in `\citet{deisenroth2020mml}`.

We define the update between two iterations as  $\Delta \theta^{(i)} = \theta^{(i)} - \theta^{(i-1)}$ . We then define a new iterative algorithm

```
$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta} C(X, \theta^{(i)}) + \alpha \Delta \theta^{(i)},$$  
where  $\alpha \in [0, 1]$  is the \textit{momentum parameter}, and  $\eta > 0$  is the learning rate.  
Hence with the new iterative algorithm, the update is equal to  
  
$$\Delta \theta^{(i)} = \alpha \Delta \theta^{(i-1)} - \eta \nabla_{\theta} C(X, \theta^{(i-1)}).
```

This is already very solid! You've got the math lined up with the book and a clear explanation. I'd just tighten a few sentences to make it more precise mathematically and avoid ambiguity:

### **Refined version:**

A way to address the problem of choosing a learning rate is using `momentum`. The idea behind this method is to let each iterative step depend partly on the previous step, effectively giving the algorithm a "memory". This damps oscillations near steep directions and accelerates convergence along shallow directions of the cost function. We follow the mathematical formulation of Chapter 7 in [Deisenroth et al. \(2020\)](#).

$$\Delta\theta^{(i)} = \theta^{(i)} - \theta^{(i-1)}.$$

The momentum-based iterative algorithm is then

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta} C(X, \theta^{(i)}) + \alpha \Delta \theta^{(i)},$$

where  $\alpha \in [0, 1]$  is the momentum parameter, and  $\eta > 0$  is the learning rate.

Equivalently, the update can be written as

$$\Delta\theta^{(i)} = \alpha\Delta\theta^{(i-1)} - \eta\nabla_{\theta}C(X, \theta^{(i-1)}).$$

This recursion shows that if successive gradients point in similar directions, the updates accumulate, leading to faster progress. If gradients oscillate (as in steep directions of the cost surface), the momentum term counteracts this oscillation and stabilizes convergence.

That way you explain not just "big update → big step" but the real mechanism: accumulation vs. damping.

👉 Do you want me to also show the effective learning rate interpretation  $(\frac{\eta}{1-\alpha})$  in a sentence? That's a neat compact way of mathematically showing why momentum speeds up flat directions.

# Conversations with ChatGPT regarding the theory section

actually I have a question, how exactly does it help escaping oscillating around minimum points in a valley. As far as I can tell, we have just added an extra term to the normal GD formula. Hence if the previous update was only incremental, I agree that the extra term becomes small, which helps with what exactly? If the previous term was big, meaning we might be on a flat area in the cost optimization landscape, it takes bigger step which helps with convergence, that I understand, but I don't understand the first point that well, can you clear up any misconceptions?

Great question 🌟 — this goes right to the heart of why momentum works. Let's unpack it carefully with the "valley" picture in mind.

**1. The valley problem in plain gradient descent**

Imagine your cost function looks like a narrow valley:

- One direction (along the valley floor) has **shallow curvature** (small eigenvalue of  $X^T X$ ).
- The perpendicular direction (across the valley walls) has **steep curvature** (large eigenvalue).

In plain gradient descent:

- The gradient points somewhat toward the bottom, but with a large component in the steep direction.
- This makes the algorithm zig-zag: it overshoots across the valley walls, then corrects back, oscillating left-right while making only tiny progress down the valley floor.
- Result: **slow convergence**.

**2. What momentum adds**

Momentum modifies the update:

$$\Delta\theta^{(i)} = \alpha\Delta\theta^{(i-1)} - \eta\nabla_{\theta}C(\theta^{(i-1)}).$$

# Conversations with ChatGPT regarding the theory section

TWO things happen:

- **Accumulation in consistent directions:**

If successive gradients point roughly the same way (e.g. down the valley floor), then  $\Delta\theta^{(i-1)}$  and  $-\eta\nabla C(\theta^{(i-1)})$  point in the same direction.  
The momentum term adds to the new update, effectively increasing the step size in that direction.

→ Faster progress along the flat direction.

- **Cancellation in oscillating directions:**

Across the valley walls, the gradient alternates sign at each step (left wall → right wall → left wall).  
That means successive  $-\eta\nabla C(\theta)$  point in opposite directions.  
Here, the momentum term  $\alpha\Delta\theta^{(i-1)}$  partially cancels the new gradient, damping the oscillation.  
→ Smaller swings back and forth.

### 3. Why this helps with oscillations

So the "extra term" is not arbitrary — it acts like a low-pass filter:

- In directions where the gradient flips a lot (oscillation), momentum reduces the effect of those sharp changes.
- In directions where the gradient is consistent (steady slope), momentum amplifies progress.

### Intuition analogy

Think of rolling a ball in a valley:

- Without momentum: the ball is sticky — it follows the exact slope at each point, zig-zagging side to side.
- With momentum: the ball has mass — it keeps rolling in the same general direction, so small opposing pushes (oscillations) don't change its course much, but consistent pushes (downhill along the valley) accelerate it.

So your current intuition about momentum only "making flat regions faster" is correct, but incomplete — the other half is that it damps oscillations by averaging out successive, alternating gradient directions.

👉 Do you want me to show this with a 2D quadratic example (like  $\frac{1}{2}(10x^2 + y^2)$ ) a. ↓ write down how plain GD vs. momentum behave there? That's the canonical "valley" case.

# Conversations with ChatGPT regarding the theory section

lets just use the MSE as an example here, what does the cost landscape look like?

Perfect 🎉 let's pin this down with MSE.

## 1. The MSE cost

Suppose you have training data  $y \in \mathbb{R}^n$ , feature matrix  $X \in \mathbb{R}^{n \times p}$ , and parameters  $\theta \in \mathbb{R}^p$ . The mean squared error is

$$C(\theta) = \frac{1}{n} \|y - X\theta\|^2.$$

## 2. Expanding

Expand it out:

$$C(\theta) = \frac{1}{n} (y^T y - 2y^T X\theta + \theta^T X^T X\theta).$$

Notice the key term:

$$\theta^T (X^T X)\theta.$$

That's quadratic in  $\theta$ .

## 3. The shape of the landscape

- If  $p = 1$  (only one parameter):  $C(\theta)$  is just a parabola in the plane — like we discussed before.
- If  $p = 2$ ;  $C(\theta)$  is a quadratic surface in  $(\theta_1, \theta_2)$ . It's a bowl — an elliptic paraboloid. Its cross-sections are ellipses.
- If  $p > 2$ ; it's the same bowl shape, but in higher dimensions.

Formally, the Hessian is

$$\nabla^2 C(\theta) = \frac{2}{n} X^T X,$$

which is positive semidefinite. This is what makes the surface convex.

## 4. Why it can zig-zag

The eigenvalues of  $X^T X$  determine how "curved" the bowl is in different directions:

- Large eigenvalue = steep direction (narrow valley).
- Small eigenvalue = flat direction (wide valley).

If the ratio of largest to smallest eigenvalue is huge (high condition number), gradient descent zig-zags: it takes small steps along the flat direction while bouncing back and forth across the steep direction.

✓ So, the MSE landscape is always a **convex quadratic bowl**, but depending on  $X$ , it might be "round" (easy to optimize) or a "long narrow valley" (harder, zig-zagging happens).

Do you want me to show you explicitly with equations how the zig-zag arises in the 2D case? Like writing down the update rules for a skinny ellipse-shaped bowl?

# Conversations with ChatGPT regarding the theory section

wait you are getting ahead of yourself, now I wish to talk about the gradient.

recall the theorem

```
\begin{theorem}[Translated from \cite{flervariabelkalkulus}]
Assume $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable in a point $a \in \mathbb{R}^n$. The
gradient $\nabla f(a)$ points in the direction of the steepest increase of $f$ at $a$, and the slope in this
direction equals $\|\nabla f(a)\|$.
\end{theorem}
```

which is the reason why gradient descent works right?

Yes ✓ — that theorem is exactly the reason gradient descent works.

Let me unpack it carefully:

- The gradient  $\nabla f(a)$  tells you the direction in  $\mathbb{R}^n$  where the function  $f$  increases the fastest.
- The magnitude  $\|\nabla f(a)\|$  tells you how steeply it increases in that direction.
- So if you want to minimize  $f$ , you do the opposite: move a little step in the negative gradient direction.

Formally, if we take a small step  $\Delta\theta$  from  $\theta$ , the first-order Taylor expansion gives

$$f(\theta + \Delta\theta) \approx f(\theta) + \nabla f(\theta)^T \Delta\theta.$$

This means the change in  $f$  is controlled by the inner product between  $\nabla f(\theta)$  and the step direction  $\Delta\theta$ .

- If  $\Delta\theta$  points in the same direction as the gradient  $\rightarrow$  inner product positive  $\rightarrow$  function increases.
- If  $\Delta\theta$  points in the opposite direction  $\rightarrow$  inner product negative  $\rightarrow$  function decreases.

Thus, by always stepping in

$$\Delta\theta = -\eta \nabla f(\theta),$$

we guarantee  $f$  decreases (at least locally, for small  $\eta$ ).

# Conversations with ChatGPT regarding the theory section

okay that makes sense, I dont need the proof of why the gradient gives the steepest direction, I will just accept that, but essentially, but if we have  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , evaluated in a point  $a$  (where  $a$  is our initial guess maybe the zero vector.) and  $f$  represents our cost function, then the gradient of  $f$  in  $a$ , will be that specific vector, i.e. the direction that  $f$  grows the steepest, hence by taking the negative, we get the direction where it descends the steepest? right

Exactly ✓, you've nailed the intuition. Let me just restate cleanly in your setting:

- We have a cost function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ,
- We evaluate it at some point  $a$  (say our current guess for the parameter, e.g.  $\theta^{(i)}$ ).
- The gradient  $\nabla f(a)$  is a vector in  $\mathbb{R}^n$ .

**Interpretation:**

- $\nabla f(a)$  points in the direction where  $f$  increases the fastest.
- $-\nabla f(a)$  points in the direction where  $f$  decreases the fastest.
- The size of the gradient,  $\|\nabla f(a)\|$ , tells us the slope of that steepest climb/descent.

So when we do

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla f(\theta^{(i)}),$$

we are literally saying: "move a small step ( $\eta$ ) in the direction of steepest decrease."

That's why gradient descent is guaranteed (for convex  $f$ ) to converge to the global minimum.

Do you want me to also explain how the step size  $\eta$  interacts with this picture — like why too large  $\eta$  overshoots, and too small  $\eta$  converges slowly?

sure but before that I want to try and test my intuition.

So lets say we do a step in the direction of the negative gradient, with a step size eta and we are getting close to the minimum. But our eta is a bit too big, so we jump over the minimum. what exactly happens to the gradient then

Perfect test 🌟 — let's walk through it carefully.

Suppose we are minimizing a nice convex cost function (think of a quadratic bowl in  $\mathbb{R}^2$  for visualization).

1. **We are close to the minimum:**  
At the true minimum, the gradient is exactly zero:  
$$\nabla f(\theta^*) = 0.$$
2. **We take too big a step:**  
If our step size  $\eta$  is too large, then instead of landing closer to the minimum, we overshoot and land on the *other side* of the valley.
3. **What happens to the gradient?**
  - Since we're now on the opposite side, the gradient changes sign (or more generally, its direction flips across the valley).
  - Intuitively: if you were moving downhill on the left side, the gradient pointed right; once you jump to the right side, the gradient now points left.
4. **Next update:**  
The algorithm then tries to move you back across the valley again, but if  $\eta$  is still too big, you overshoot once more.  
→ This creates the classic **oscillation** back and forth around the minimum.
5. **Why small  $\eta$  fixes this:**  
With a smaller  $\eta$ , your steps get shorter, so you land closer to the minimum each time. That way, the oscillations dampen out and you converge smoothly.

Mathematically, in 1D (just for intuition):  
$$f(\theta) = \theta^2, \quad \nabla f(\theta) = 2\theta.$$

**Gradient descent update:**

$$\theta^{(i+1)} = \theta^{(i)} - \eta \cdot 2\theta^{(i)} = (1 - 2\eta)\theta^{(i)}.$$

## Conversations with ChatGPT regarding the theory section

- If  $0 < \eta < 1$ , then  $1 - 2\eta$  is between  $-1$  and  $1$ , so the sequence converges to  $0$ .
- If  $\eta > 1$ , then  $1 - 2\eta < -1$ , and the iterations oscillate with increasing amplitude (diverge).
- If  $0.5 < \eta < 1$ , then  $1 - 2\eta$  is negative but between  $-1$  and  $0$ : you still oscillate, but with shrinking amplitude → convergence with oscillation.

👉 So in your words: when you overshoot the minimum, the gradient flips direction, and the method "bounces" back and forth. If the learning rate is not small enough, the bouncing doesn't settle down.

Would you like me to also show you this behavior with a simple plot of iterations for different  $\eta$  values?

so back to the greater than  $1$  intuition again, it is lower than  $1$ , we will slowly move downwards towards the minimum, since the gradient will slowly decrease, we might still oscillate, but for every time we bounce, we will bounce slightly down, eventually reaching the bottom after enough iterations. If it is  $1$ , we will alternate between the exact same two points for each iteration, and if it is greater than  $1$ , we will begin to climb upwards? since for each step, the gradient will actually become slightly bigger, and we will go further and further away for each iteration?

Yes — you've got it 🤓. Let me walk through it carefully in the simple 1D case:

We had the update rule for  $f(\theta) = \theta^2$ :  
 $\theta^{(i+1)} = (1 - 2\eta)\theta^{(i)}$ .

This is a geometric sequence with ratio  $r = 1 - 2\eta$ .

**Case 1:**  $0 < \eta < 1 \rightarrow |r| < 1$ .

- Each step multiplies  $\theta$  by something with magnitude less than  $1$ .
- If  $0 < \eta < 0.5$ , then  $0 < r < 1$ . The sequence shrinks monotonically toward  $0$  (no oscillation).
- If  $0.5 < \eta < 1$ , then  $-1 < r < 0$ . The sequence alternates signs (oscillates around  $0$ ), but each bounce is smaller, so it converges to  $0$ . ✅

**Case 2:**  $\eta = 1 \rightarrow r = -1$ .

- Then  
 $\theta^{(i+1)} = -\theta^{(i)}$ .
- This makes it alternate between two fixed values forever. Oscillation without decay. ❓

**Case 3:**  $\eta > 1 \rightarrow |r| > 1$ .

- Then the factor  $(1 - 2\eta)$  in magnitude is larger than  $1$ .
- So instead of shrinking,  $\theta$  grows in magnitude at each step.
- The sequence diverges — it shoots off to infinity (while still oscillating if  $r < -1$ ). ❌

So your intuition is right:

- Slightly too big  $\eta \rightarrow$  still converges, but with oscillations.
- Exactly the boundary ( $|r| = 1$ ) → stuck bouncing forever.
- Too big ( $|r| > 1$ ) → blows up, the gradient steps push you *further* away each time.

⚡ Do you want me to generalize this explanation to the multi-dimensional quadratic bowl (with eigenvalues) so you see how the same idea extends?

# Conversations with ChatGPT regarding the theory section

but just so I get it right, the divergence happens since when we bounce back and forwards with step size larger than 1, the gradient will increase slightly each time, as we are moving further from the minimum?

Exactly 🤗 — that's the key intuition.

Let's stay in the 1D example  $f(\theta) = \theta^2$ :

- Near the minimum at  $\theta = 0$ , the gradient is  $\nabla f(\theta) = 2\theta$ .
- Suppose you overshoot past 0. Then  $\theta$  flips sign, but now its magnitude is actually larger than before, because the update rule multiplies by  $|1 - 2\eta|$ .
- Since  $|1 - 2\eta| > 1$  when  $\eta > 1$ , each time you cross,  $|\theta|$  grows.
- A larger  $|\theta|$  means a larger gradient (because gradient =  $2\theta$ ), which means an even bigger step next time.
- So you don't just oscillate — you oscillate with *increasing amplitude*. 🎉

That's why the trajectory diverges.

So:

- If  $\eta < 1$ , the factor  $|1 - 2\eta|$  is less than 1 → oscillations shrink.
- If  $\eta > 1$ , the factor is bigger than 1 → oscillations grow because each overshoot lands further away, increasing the gradient magnitude.

👉 Want me to now show how this intuition generalizes to the multi-dimensional bowl, where "step size too large" means  $\eta$  is too large relative to the largest eigenvalue of  $X^T X$ ?

I just wrote this to finish off the momentum GD, since it will be too much to go into detail, what do you think

With this new iterative formula, depending on the choice of momentum parameter, the step size will decrease when the last update was small. On the other hand, if the last update was large, the step size will increase. This allows us to escape oscillation, and more quickly converge to the minimum. There are other approaches to adaptively change the step size. Such as `textit{ADAM}`, `textit{AdaGrad}` and `textit{RMSProp}`. We will not explain these in detail here, but we will showcase these methods in our implementation section.

I think that's a very solid way to wrap it up 🤗. It stays in line with the rest of your report: mathematical where needed, but not overly technical.

Here's a lightly polished version for flow and precision (without adding bulk):

With this new iterative formula, the influence of past updates depends on the momentum parameter  $\alpha$ . If the previous update was small, the effect is damped, while a large update carries more weight. This reduces oscillations near minima and accelerates convergence in flatter regions of the cost landscape. There are also other approaches that adapt the step size directly, such as `textit{AdaGrad}`, `textit{RMSProp}`, and `textit{Adam}`. We will not explain these in detail here, but they will be demonstrated in our implementation section.

👉 Yours is already fine, this just makes the "why it helps" part a little sharper (oscillations vs. flat regions).

Do you want me to also draft a 1–2 sentence transition into stochastic gradient descent (SGD), so the section flows naturally?

# Conversations with ChatGPT regarding the theory section

SebastianRaschkaYuxi\_2022\_MachineLe...  
PDF  
make a bib reference for this book

Here's a proper BibTeX reference for the book you uploaded:

```
bibtex  
  
@book{raschka2022mlpytorch,  
    title      = {Machine Learning with PyTorch and Scikit-Learn: Develop Machine Learn  
    author     = {Raschka, Sebastian and Liu, Yuxi (Hayden) and Mirjalili, Vahid},  
    year       = {2022},  
    publisher  = {Packt Publishing},  
    address    = {Birmingham, UK},  
    isbn       = {9781801819312}  
}
```

Would you like me to also make an APA style reference for it, in case your report requires that instead of BibTeX?

what do you think about this as an explanation of the steps in bootstrap sampling? Is anything unclear?

The bootstrap method is fairly straightforward. Assume we have a dataset  $\mathbb{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ . With replacement, draw  $n$  input points  $x_i$  from our dataset, and define a bootstrap dataset  $\mathbb{D}' = \{(x'_1, y'_1), \dots, (x'_{n'}, y'_{n'})\}$ , where  $y'_i$  corresponds to input point  $x'_i$ . Meaning this new dataset likely includes duplicates, while leaving other datapoints out. Then compute a parameter  $\hat{\theta}(\theta)$  with respect to the bootstrap dataset, in the same procedure as we would compute  $\hat{\theta}(\theta)$ . Now repeat this process  $k$  times to get  $\{\hat{\theta}(\theta)^{(1)}, \dots, \hat{\theta}(\theta)^{(k)}\}$ . It turns out that if we now make a histogram of the relative frequency for each entry  $\hat{\theta}(\theta)^{(i)}$ , we obtain an estimate of the probability distribution of the coefficient  $\hat{\theta}(\theta)$ .

This is very clear already — the steps are well structured and precise. Just a couple of tweaks will make it more mathematically tight and avoid potential confusion:

- Sampling:** instead of saying "draw  $n$  input points  $x_i$ ", I'd say "draw  $n$  pairs  $(x_i, y_i)$  with replacement". Otherwise, a reader might wonder if you're resampling  $x$  alone (and regenerating  $y$ ), which is not the case.
- Duplicates sentence:** "Meaning this new dataset likely includes duplicates..." is fine, but you can make it sharper: "...so  $\mathbb{D}'$  will typically contain duplicates of some data points and omit others."
- Histogram part:** instead of "relative frequency", I'd use "empirical distribution". It makes it clear you're approximating the sampling distribution.

Here's a tightened-up version of what you wrote:

The bootstrap method is fairly straightforward. Assume we have a dataset  $\mathbb{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ . We generate a bootstrap dataset by drawing  $n$  pairs  $(x_i, y_i)$  with replacement from  $\mathbb{D}$ ,  $\mathbb{D}' = \{(x'_1, y'_1), \dots, (x'_{n'}, y'_{n'})\}$ , so that  $\mathbb{D}'$  will typically contain duplicates of some data points while omitting others. We then compute a parameter estimate  $\hat{\theta}$  from the bootstrap dataset in the same way as we would compute  $\hat{\theta}$ . Repeating this process  $k$  times gives  $\hat{\theta}^{(1)}, \dots, \hat{\theta}^{(k)}$ . By examining the empirical distribution of each coordinate  $\hat{\theta}_j^{(i)}$ , for example through histograms, we obtain an estimate of the probability distribution of the coefficient  $\hat{\theta}_j$ . From this distribution, we can compute statistical properties of  $\hat{\theta}_j$ , such as variance or confidence intervals.

💡 This version keeps your structure but makes it 100% unambiguous.

Do you want me to also add a tiny example (like with a dataset of size  $n = 3$ ) to illustrate duplicates and omissions, or do you want to keep it strictly formal?

# Conversations with ChatGPT regarding the theory section

I have some lecture notes that I want to create a reference of. It is this:

<https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/chapter2.ipynb>

How do I cite this in the text and what is the bib reference?

Yes — you can absolutely cite those lecture notes. Here's how to do it in a clean and standard way (for LaTeX / BibTeX):

**Suggested BibTeX entry**

```
bibtex
@misc{compphysics_chapter2,
    title      = {Machine Learning Lecture Notes, Chapter 2},
    author     = {{CompPhysics / MachineLearning}},
    year       = {2025},
    howpublished = {\url{https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/chapter2.ipynb}},
    note       = {Accessed: 2025-09-24}
}
```

You can adjust the year (if you know when that chapter was last updated) and adjust the "author" field as you see fit (e.g. you could list the instructor or organization).

**How to cite in your text**

You can do something like:

| As shown in the lecture notes (CompPhysics, Chapter 2) \cite{compphysics\_chapter2}, the gradient update can be viewed as ...

Or:

| According to the *Machine Learning Lecture Notes* by CompPhysics \cite{compphysics\_chapter2}, one can derive the gradient descent update ...

Make sure your \cite{compphysics\_chapter2} matches the citation key in your .bib file.

If you like, I can also format a version using a footnote style instead of \cite{...}—whichever fits your document style best.

Could you make one for this as well?

[https://compphysics.github.io/MachineLearning/doc/LectureNotes/\\_build/html/chapteroptimization.html#momentum-based-gd](https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapteroptimization.html#momentum-based-gd)

Here's a suitable BibTeX entry (and how to cite it) for the optimization notes page you linked:

```
bibtex
@misc{compphysics_optimize,
    title      = {Optimization (Chapter): Momentum-based GD and Related Methods},
    author     = {{CompPhysics / MachineLearning}},
    year       = {2025},
    howpublished = {\url{https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapteroptimization.html#momentum-based-gd}},
    note       = {Accessed: 2025-09-24}
}
```

In your LaTeX text you can cite it like:

| As described in the course lecture notes (CompPhysics, Optimization chapter) \cite{compphysics\_optimize}, the momentum-based update blends the current gradient with previous increments to improve convergence.

If you want, I can also suggest a more "clean" citation style (footnote or shortened author) that fits your report style.

# Conversations with ChatGPT regarding the theory section

what do you think about this as an explanation of the steps in bootstrap sampling? Is anything unclear?

The bootstrap method is fairly straightforward. Assume we have a dataset  $\mathbb{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ . With replacement, draw  $n$  input points  $x_{\cdot i}$  from our dataset, and define a bootstrap dataset  $\mathbb{D}' = \{(x'_1, y'_1), \dots, (x'_n, y'_n)\}$ , where  $y'_{\cdot i}$  corresponds to input point  $x'_{\cdot i}$ . Meaning this new dataset likely includes duplicates, while leaving other datapoints out. Then compute a parameter  $\hat{\theta}$  with respect to the bootstrap dataset, in the same procedure as we would compute  $\hat{\theta}$  with respect to the original dataset. Now repeat this process  $k$  times to get  $\hat{\theta}_1, \dots, \hat{\theta}_k$ . It turns out that if we now make a histogram of the relative frequency for each entry  $\hat{\theta}_j$ , we obtain an estimate of the probability distribution of the coefficient  $\hat{\theta}_j$ . From here, we can compute statistical properties of  $\hat{\theta}_j$ , such as variance.

This is very clear already — the steps are well structured and precise. Just a couple of tweaks will make it more mathematically tight and avoid potential confusion:

1. **Sampling:** instead of saying “draw  $n$  input points  $x_i$ ”, I’d say “draw  $n$  pairs  $(x_i, y_i)$  with replacement”. Otherwise, a reader might wonder if you’re resampling  $x$  alone (and regenerating  $y$ ), which is not the case.
2. **Duplicates sentence:** “Meaning this new dataset likely includes duplicates...” is fine, but you can make it sharper: “...so  $\mathbb{D}'$  will typically contain duplicates of some data points and omit others.”\*
3. **Histogram part:** Instead of “relative frequency”, I’d use “empirical distribution”. It makes it clear you’re approximating the sampling distribution.

Here’s a tightened-up version of what you wrote:

The bootstrap method is fairly straightforward. Assume we have a dataset

$$\mathbb{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}.$$

We generate a bootstrap dataset by drawing  $n$  pairs  $(x_i, y_i)$  with replacement from  $\mathbb{D}$ ,

$$\mathbb{D}' = \{(x'_1, y'_1), \dots, (x'_n, y'_n)\},$$

so that  $\mathbb{D}'$  will typically contain duplicates of some data points while omitting others.

We then compute a parameter estimate  $\hat{\theta}$  from the bootstrap dataset in the same way as we would compute  $\hat{\theta}$ .

Repeating this process  $k$  times gives  $\hat{\theta}^{(1)}, \dots, \hat{\theta}^{(k)}$ .

By examining the empirical distribution of each coordinate  $\hat{\theta}_j^{(i)}$ , for example through histograms, we obtain an estimate of the probability distribution of the coefficient  $\hat{\theta}_j$ . From this distribution, we can compute statistical properties of  $\hat{\theta}_j$ , such as variance or confidence intervals.

\* This version keeps your structure but makes it 100% unambiguous.

Do you want me to also add a tiny example (like with a dataset of size  $n = 3$ ) to illustrate duplicates and omissions, or do you want to keep it strictly formal?