# Project 2

## FYS-STK4155

Håvard Skåli, Erik Røset & Oskar Idland
*University of Oslo, Department of Physics*
(Dated: October 18, 2024)

https://github.com/Oskar-Idland/FYS-STK4155-Projects

## I. INTRODUCTION

The ever-growing complexity of modern data calls for robust machine learning techniques capable of handling non-linear patterns and high-dimensional spaces. While traditional methods, such as Ordinary Least Squares (OLS), Ridge, and Lasso regression, offer valuable insights into model behavior and regularization, their limitations become apparent when addressing more intricate datasets. In our previous project, we explored these regression techniques in-depth, assessing their effectiveness on synthetic and real-world datasets, with particular attention to the bias-variance tradeoff and resampling methods for model validation. However, as data complexity increases, more sophisticated methods, such as neural networks, become necessary to model non-linear relationships and provide superior generalization.

In this project, we extend the concepts introduced earlier by diving into two fundamental areas of machine learning: gradient descent optimization and neural networks. These topics form the backbone of many modern machine learning algorithms and offer powerful tools for tackling a wide range of problems, from regression and classification to complex tasks in pattern recognition and forecasting.

Gradient descent plays a critical role in optimizing the parameters of complex models, especially when closed-form solutions are impractical or non-existent. Through iterative updates, gradient descent methods, including batch, stochastic, and mini-batch variants, minimize loss functions and improve model performance. In particular, understanding the nuances of learning rates, convergence behavior, and methods to prevent overshooting or getting stuck in local minima are central to achieving efficient optimization.

Furthermore, we shift our focus to neural networks, which represent a significant step forward in model complexity and capability. These models, inspired by biological neurons, consist of multiple layers of interconnected nodes that enable the learning of intricate data structures through non-linear activation functions. Backpropagation, coupled with gradient descent, facilitates the adjustment of network weights to minimize prediction errors. In this project, we aim to explore how neural networks can outperform traditional regression methods by capturing non-linear dependencies and providing better generalization on complex datasets.

Through the combination of gradient descent optimization and neural networks, this project seeks to push beyond the limitations of classical machine learning models, addressing the challenges posed by real-world data. By implementing these methods, we hope to gain a deeper understanding of how machine learning models can be optimized and improved for practical applications.

## II. THEORY

In this project, we extend our previous exploration of machine learning techniques into more advanced optimization algorithms and neural networks. Gradient descent methods, along with their various enhancements, are crucial in training neural networks, which are used to model complex, non-linear relationships. While the foundational principles of regression analysis and optimization were addressed in our previous report, here we focus specifically on gradient descent with and without momentum, and tuning methods such as decaying learning rates, AdaGrad, RMSprop, and ADAM. Additionally, the fundamentals of neural networks will be discussed, emphasizing their relevance to the current project.

Gradient Descent Methods

Gradient descent is an iterative optimization algorithm used to minimize a given loss function, $L(\theta)$, by adjusting the parameters $\theta$ of the model. The method operates by computing the gradient of the loss function with respect to the parameters and updating the parameters in the direction that reduces the loss. The general update rule for gradient descent is:

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

where $\eta$ is the learning rate, which controls the size of the update step, and $\nabla_\theta L(\theta_t)$ is the gradient of the loss function with respect to $\theta$ at iteration $t$.

Gradient descent comes in several variants, depending on how the gradient is calculated: - Batch Gradient Descent: Computes the gradient using the entire dataset, which is computationally expensive for large datasets. - Stochastic Gradient Descent (SGD): Updates the parameters using only one data point at a time, making it faster but prone to high variance in the updates. - Mini-batch Gradient Descent: Strikes a balance between batch and

stochastic gradient descent by computing the gradient using a small subset (batch) of the data.

Gradient Descent with Momentum

While vanilla gradient descent can be slow to converge, especially when the gradient is small or when the loss surface contains narrow valleys, momentum can help accelerate convergence. Momentum builds up velocity in directions of consistent gradients and reduces oscillations in the parameter updates. The update rule for gradient descent with momentum is:

$$v_{t+1} = \beta v_t + (1-\beta)\nabla_\theta L(\theta_t)$$

$$\theta_{t+1} = \theta_t - \eta v_{t+1}$$

Here, $v_t$ is the velocity term, and $\beta$ (usually set around 0.9) is the momentum coefficient. Momentum allows the algorithm to continue moving in the same direction if the gradients are consistently pointing that way, leading to faster convergence in directions where the loss function is shallow.

Learning Rate Tuning Methods

Choosing the right learning rate $\eta$ is critical to the success of gradient descent algorithms. If the learning rate is too large, the optimization may overshoot the minimum, while if it is too small, convergence will be very slow. Several adaptive learning rate methods have been proposed to dynamically adjust the learning rate during training:

- Decaying Learning Rate: One simple approach is to gradually decrease the learning rate as training progresses, using a schedule such as:

$$\eta_t = \frac{\eta_0}{1+\lambda t}$$

where $\eta_0$ is the initial learning rate, $t$ is the iteration, and $\lambda$ is the decay rate. This helps ensure that larger updates are made at the start of training when far from the optimum, and smaller, more precise updates are made later.

- AdaGrad (Adaptive Gradient Algorithm): AdaGrad adapts the learning rate for each parameter based on the historical gradients. It assigns a smaller learning rate to frequently updated parameters and a larger rate to less frequently updated ones:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}}\nabla_\theta L(\theta_t)$$

where $G_t$ is the sum of the squares of the past gradients, and $\epsilon$ is a small constant to avoid division by zero. AdaGrad is well-suited for sparse data but may become overly conservative in later iterations due to the cumulative sum of gradients.

- RMSprop: To address AdaGrad's diminishing learning rates, RMSprop uses a moving average of the squared gradients to scale the learning rate. This helps maintain a balance between fast convergence and smooth parameter updates:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}\nabla_\theta L(\theta_t)$$

where $\beta$ (typically 0.9) controls the decay rate of the moving average, and $E[g^2]_t$ is the moving average of the squared gradients.

- ADAM (Adaptive Moment Estimation): ADAM combines the advantages of both AdaGrad and RMSprop by keeping track of both the first moment (mean) and the second moment (uncentered variance) of the gradients:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta L(\theta_t)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta L(\theta_t))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

ADAM is one of the most widely used optimization algorithms today due to its ability to dynamically adjust learning rates and its robustness in practice.

Neural Networks

Neural networks are powerful machine learning models designed to capture complex, non-linear relationships in data. At their core, neural networks consist of multiple layers of interconnected neurons (or nodes). Each neuron computes a weighted sum of its inputs, applies a non-linear activation function, and passes the result to the next layer.

In a feedforward neural network, information flows from the input layer through one or more hidden layers to the output layer. Each layer transforms the input data in a hierarchical manner, allowing the network to learn representations at multiple levels of abstraction. The output of the network can be used for tasks such as classification, regression, or even generating new data.

The training of neural networks involves backpropagation, a process in which the gradient of the loss function is computed with respect to the network's parameters. Backpropagation allows for the efficient calculation of these gradients through the chain rule of calculus, which are then used to update the weights via gradient descent.

Relevant to this project, neural networks can benefit significantly from the advanced gradient descent methods described above. Techniques like momentum, ADAM, and learning rate decay help overcome challenges such as slow convergence and the vanishing gradient problem, especially in deep networks with many layers.

Activation Functions and Relevance to the Project

In this project, the choice of activation function plays a crucial role. The commonly used activation functions include: - ReLU (Rectified Linear Unit): ReLU is defined as $\text{ReLU}(x) = \max(0, x)$, which introduces non-linearity and mitigates the vanishing gradient problem. - Sigmoid: Sigmoid squashes the input to a value between 0 and 1, making it useful for binary classification tasks but prone to vanishing gradients. - Tanh: Similar to sigmoid but maps inputs to the range (-1, 1), which can center data and mitigate some issues caused by sigmoid.

Given the nature of the tasks in this project (such as handling non-linear data structures or optimizing complex loss functions), selecting appropriate activation functions and tuning the network's architecture will be key to achieving robust results.

## III. METHODS & IMPLEMENTATION

## IV. RESULTS & DISCUSSION

## V. CONCLUSION

## REFERENCES

## Appendix A: Code

Link to our GitHub repository: https://github.com/Oskar-Idland/FYS-STK4155-Projects