Report of the 2024-2025 gap year (Part II)

Department of Mathematical engineering and computer science

VIELZEUF Charles

Particular Approximation of Mean-Field Langevin Dynamics for Deep Learning

Deep Learning theory team trainee student at RIKEN AIP (Tokyo)

Tutor from école des Ponts:
Axel PARMENTIER
In charge of the operational research module
axel.parmentier@enpc.fr

Academic supervisor from école des Ponts:
Eric DUCEAU
In charge of the IMI department
eric.duceau@enpc.fr

Internship supervisor from RIKEN:
Taiji SUZUKI
Deep learning theory team leader
taiji.suzuki@riken.jp

Internship realised from April 2025 to August 2025
At RIKEN, 1-4-1 Nihonbashi, Chuo-ku, Tokyo 103-0027, Japan
October 12, 2025

Fiche de Synthèse 1

Type de stage : stage en laboratoire

Année : 2024-2025

Auteur (Nom, prénom) : VIELZEUF Charles

Formation 2ème année (IMI, GI, SEGF, etc.) : IMI

Titre du rapport : Particular Approximation of Mean-Field Langevin Dynamics for Deep Learning

Titre en français (si titre en langue étrangère) : Approximation par particules de la dynamique de Langevin en champ moyen pour l'apprentissage profond

Organisme d'accueil : RIKEN AIP

Pays d'accueil : Japon

Responsable de stage : PARMENTIER Axel

Mots-clés caractérisant votre rapport (4 à 5 mots max) : apprentissage profond, apprentissage automatique, développement informatique, algorithmique, recherche

Résumé en français

Ce rapport de stage présente mon expérience de quatre mois au sein du **RIKEN, Advanced Institute of Computational Science** à Tokyo (sur les campus de Nihonbashi et Hongo).

Le **RIKEN (理研 – National Research and Development Agency)** est un institut de recherche japonais de renommée mondiale, reconnu pour ses contributions dans divers domaines scientifiques, notamment la biologie, la physique et l'informatique.

Figure 1: SPring-8 (Large Synchrotron Radiation Facility) et SACLA (X-ray Free Electron Laser Facility)

Fondé en 1917, il est l'un des plus grands instituts de recherche au Japon, avec **plus de 3 000 chercheurs** et un budget annuel de plusieurs milliards de yens. [1]
Cette institution est impliqué dans des projets de **recherche fondamentale et appliquée, en collaboration avec des universités, des entreprises et d'autres instituts de recherche à l'échelle nationale et internationale**.
Les domaines couverts sont très variés tels que la biologie, la physique, la chimie, l'informatique et l'ingénierie. J'en avais entendu parler pour ses travaux sur les supercalculateurs, l'apprentissage automatique, l'optimisation et la simulation numérique.
Il est organisé en plusieurs instituts et centres de recherche, chacun se concentrant sur des domaines spécifiques. Une description plus détaillée des différents centres de recherche du RIKEN est disponible sur le site officiel de l'institut: https://www.riken.jp/en/research/labs/index.html .

J'ai effectué mon stage dans l'un de ces centres, le **RIKEN AIP (Advanced Institute of Computational Science)**, qui est constitué de plusieurs équipes de recherche (une vingtaine), chacune se concentrant sur des domaines spécifiques de l'informatique avancée et de l'intelligence artificielle. L'institut a été fondé en 2008 et est dirigé par le professeur **Masashi Sugiyama**, un expert reconnu dans le domaine de l'apprentissage automatique et de l'optimisation.
Il y a quatre grands pôles de recherche au sein du RIKEN AIP, détaillés plus en détails en partie 1.1.1. Celui de la *Deep Learning Theory Team* est le *Generic Technology Research Group*.

J'ai eu l'occasion d'aller assez souvent au centre administratif à *Nihonbashi* qui accueille d'autres équipes et très régulièrement des séminaires, et j'ai travaillé sur le campus de l'*université de Tokyo (Hongo)*.

Figure 2: Entrée sécurisée du centre administratif à Nihonbashi

[1] Voir https://www.riken.jp/en/about/ pour plus de détails

J'ai ainsi eu l'opportunité de travailler au sein de l'équipe de recherche de théorie de l'apprentissage profond (*Deep Learning Theory Team*).

Elle est dirigée par **Taiji Suzuki** (université de Tokyo), et compte environ 20 membres. Le but premier de l'équipe est d'expliquer et de comprendre les modèles d'apprentissage profond, afin de développer des méthodes d'apprentissage plus efficaces et interprétables, la théorie étant très en retard sur l'état de l'art [1]. La recherche est très dynamique avec de nombreuses directions de recherche, notamment récemment sur les transformers, les LLM et les modèles de diffusion..

S'inscrivant dans le cadre plus large du **machine learning** (apprentissage automatique) le **deep learning** (apprentissage profond) est un ensemble de techniques qui utilise des réseaux de neurones profonds pour modéliser des données complexes et pouvoir faire des prédictions sur des données nouvelles sur le principe de la *généralisation*.
Ces dernières années, il a connu une croissance exponentielle, notamment grâce à l'augmentation de la puissance de calcul et à la disponibilité de grandes quantités de données et a permis de réaliser des avancées majeures dans des domaines tels que la vision par ordinateur, le traitement du langage naturel, **NLP (Natural Language Processing)**, et la robotique pour ne citer qu'eux.

Durant mon stage, j'ai commencé par un état de l'art ainsi que des tests divers pour me forger une intuition et avoir un point de départ solide pour pouvoir travailler sur un sujet plus précis qui a été défini en cours de stage. J'ai notamment commencé par implémenter "from scratch" un module python permettant de créer son réseau en superposant des couches paramétrisables, en choisissant parmi plusieurs techniques d'optimisation et plusieurs jeux de données. J'ai pu l'enrichir tout au long de mon stage et il m'a permis de faire des tests lorsque c'était nécessaire.
Ce projet n'est pas designé pour être efficace mais facilement modulable et compréhensible. Les couches principales et quelques exemples d'utilisation sont présentés en partie 1.4.
Le projet est disponible sur le lien https://github.com/charles-vzf/modularyNN.

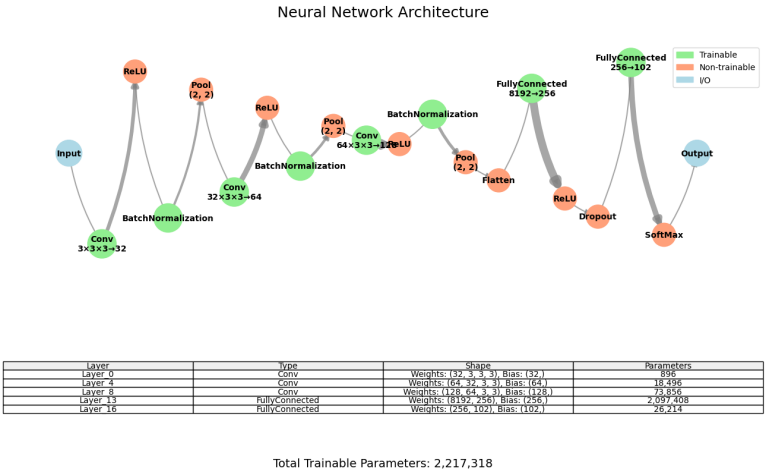


Figure 3: réseau de neurones convolutionel généré en travail préparatoire de modularyNN

Ce projet qui m'a occupé le premier mois m'a notamment permis d'avoir une vision globale de ce qui se fait classiquement en deep learning, car j'ai été amené à tester un bon nombre d'architectures et de techniques d'optimisation.

[1] Voir https://www.riken.jp/en/research/labs/aip/generic_tech/deep_learn_theory/index.html pour plus de détails

L'objectif principal de mon stage a été d'étudier, dans le cadre des dynamiques de Langevin au champ moyen (MFLD), des garanties de **propagation de chaos uniformes en temps**. Le problème se formule comme l'optimisation d'une fonction de perte *régularisée par l'entropie* (définition plus précise ici: **entropy**):

$$\min_{\mu\in\mathcal{P}_2(\mathbb{R}^d)} \Big\{\mathcal{L}(\mu) := F_0(\mu) + \mathbb{E}_{X\sim\mu}[r(X)] + \lambda\,\mathrm{Ent}(\mu)\Big\},$$

avec $r(x) = \lambda'\|x\|_2^2$ ($\lambda' > 0$, fortement convexe) et $F(\mu) = F_0(\mu) + \mathbb{E}_\mu[r(X)]$.

En notant $\mathbf{X}_t = (X_t^1, \ldots, X_t^N)$ les coordonnées des points suivant une dynamique de Langevin au champ moyen en régime sans momentum (N-uplet d'équations différentielles stochastiques), on a :

$$dX_t^i = -\nabla\left(\frac{\delta F(\rho_{\mathbf{x}_t})}{\delta\mu}\right)(X_t^i)\,dt + \sqrt{2\lambda}\,dW_t^i, \quad i = 1, \ldots, N,$$

où $\{W_t^i\}_{t\geq 0}$ sont des mouvements browniens indépendants. Elle satisfait l'équation de Fokker–Planck :

$$\frac{\partial\mu_t^{(N)}}{\partial t} = \lambda\,\nabla\cdot\left(\mu_t^{(N)}\,\log\left(\frac{d\mu_t^{(N)}}{d\mu_*^{(N)}}\right)\right).$$

Sous des hypothèses standard, la solution optimale $\mu^* \in \mathcal{P}_2(\mathbb{R}^d)$ pour la loi $\mu_t^{(N)} = \mathrm{law}(\mathbf{X}_t)$ existe et satisfait la condition d'optimalité (dans le cas **sans momentum**):

$$\mu^* \propto \exp\left(-\frac{1}{\lambda}\,\frac{\delta F(\mu^*)}{\delta\mu}\right).$$

Le résultat central mobilisé (PoC uniforme pour le cas **sans momentum**, [12]) établit, pour N particules, la borne quantitative suivante :

$$\frac{1}{N}\,\mathcal{L}^{(N)}(\mu_t^{(N)}) - \mathcal{L}(\mu^*) \;\leq\; \frac{B}{N} + e^{-2\alpha\lambda t}\,\Delta_0^{(N)},$$

où $\Delta_0^{(N)} := \frac{1}{N}\mathcal{L}^{(N)}(\mu_0^{(N)}) - \mathcal{L}(\mu^*)$. Cette inégalité sépare un *terme d'approximation particulaire* en $O(1/N)$ d'un *terme qui décroît exponentiellement* avec le temps, fournissant des garanties quantitatives fortes et uniformes sur la convergence de la mesure empirique vers la limite de champ moyen.

Je devais ensuite regarder comment se comporte le cas **avec momentum**, où la dynamique est donnée par le système d'équations différentielles stochastiques ou on passe de l'espace des positions à l'espace des phases:

$$d\mu_t = v_t dt \qquad dv_t = -\nabla_\mu U(\mu_t, \mu_t)dt - \gamma v_t dt + \sqrt{2\gamma}dW_t$$

avec μ_t la distribution des points, v_t le momentum, U l'énergie potentielle dépendant de la mesure et W_t un bruit blanc.

Ce stage m'a permis d'acquérir une expertise approfondie en deep learning et optimisation et de contribuer à l'équipe, en fournissant deux bases de code utiles à l'équipe, mais sans toutefois de résultats théoriques nouveaux.

J'ai pu mettre en pratique mes connaissances en deep learning, analyse, machine learning et programmation, en particulier en optimisation convexe et en python. J'ai apprécié le travail en équipe et l'ambiance conviviale.

Je remercie particulièrement Axel Parmentier de m'avoir mis en relation avec Pierre-Louis Poirion, qui m'a lui même redirigé vers Taiji Suzuki, ainsi que toute l'équipe de deep learning theory pour leur accueil et leur soutien durant mon stage.

Abstract

First of all, I would like to thank the entire RIKEN AIP team at Nihonbashi and Tokyo University campus at Hongo for their warm welcome and the nice conversations we had on various subjects, either connex and non-connex to deep learning.

I would like to express my deep gratitude to Taiji, Sonada, Minh, Dake and Hiroko for the time they dedicated to my learning and integration, and the valuable insights they shared throughout my internship. I am also thankful to Axel Parmentier [2] and Pierre-Louis Poirion [3] for introducing me to this opportunity at RIKEN.

With this traineeship, I wanted to discover the world of research, the world of machine learning and Japan; I was not disappointed by any of those. The RIKEN AIP center is really dynamic with lots of strong researchers, various projects and a lot of expertise in machine learning and deep learning. As a common point with Japan, the team is very welcoming and the atmosphere is really nice, with a lot of exchanges and discussions on various topics.

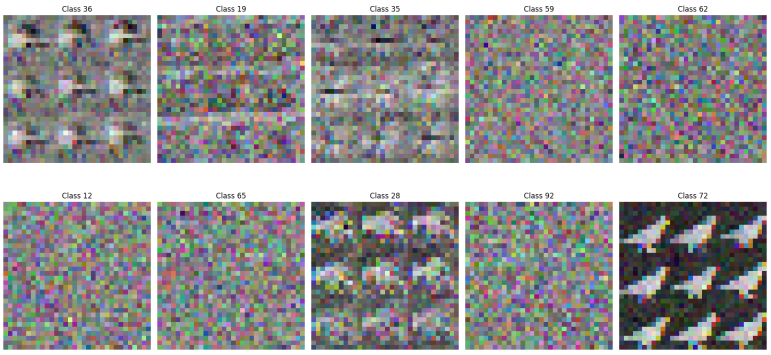
This internship was a really nice opportunity for me to learn, practice coding —especially in python and **SOTA** Deep Learning Techniques— and work alongside a reknowned team in the field.

This report is wider than just my work during the internship because I wanted to keep in mind other connex learnings and was interested by many aspects of the research that were not directly linked to what I primarly did.

I have first worked on the implementation from scratch of a python module that allows to build neural networks by stacking various layers. Not designed for performance but rather for learning purposes, this module allowed me to understand the inner workings of neural networks, including the forward and backward passes, the role of activation functions, the importance of weight initialization and optimization strategies...

Then, I have worked on studying mean-field theory and langevin dynamics. My initial goal was to understand the theory and try to give a generalization of a convergence bound in a specific setting (underdamped case). I started by reproducing some of the previously known results to build an intuition. I unfortunately did not succeed to give a generalization of the results for the underdamped case, as it took me longer than expected to understand the problem and run the experiments.

I am very grateful to Taiji for his help and guidance throughout this internship.



Learned weight patterns by a linear classifier for 10 random classes of Caltech-101 dataset

[2] see https://axelparmentier.github.io/ for more details on his work
[3] see https://sites.google.com/site/plpoirion86/research?authuser=0

Contents

List of Figures

1 Introduction

1.1 About RIKEN

RIKEN (理研), a National Research and Development Agency, is Japan's largest comprehensive research institution renowned for high-quality research in a diverse range of scientific disciplines.
Founded in 1917, initially as a private research foundation, RIKEN has grown rapidly in size and scope, today encompassing a network of world-class research centers and institutes across Japan.
It conducts cutting-edge research across various fields, including physics, chemistry, biology, and computational science. RIKEN is known for its interdisciplinary approach and collaboration with academic institutions and industry partners. The institute is committed to fostering innovation and promoting the application of scientific discoveries to address societal challenges.
RIKEN is also involved in international collaborations and partnerships, contributing to global scientific advancements. The institute plays a significant role in advancing research and technology in Japan and beyond, making it a leading institution in the scientific community.
More can be found on the RIKEN website: https://www.riken.jp/en/about/.

RIKEN has several research centers and laboratories, each focusing on specific areas of research. Below is a small sketch of the main centers classified by big domains:

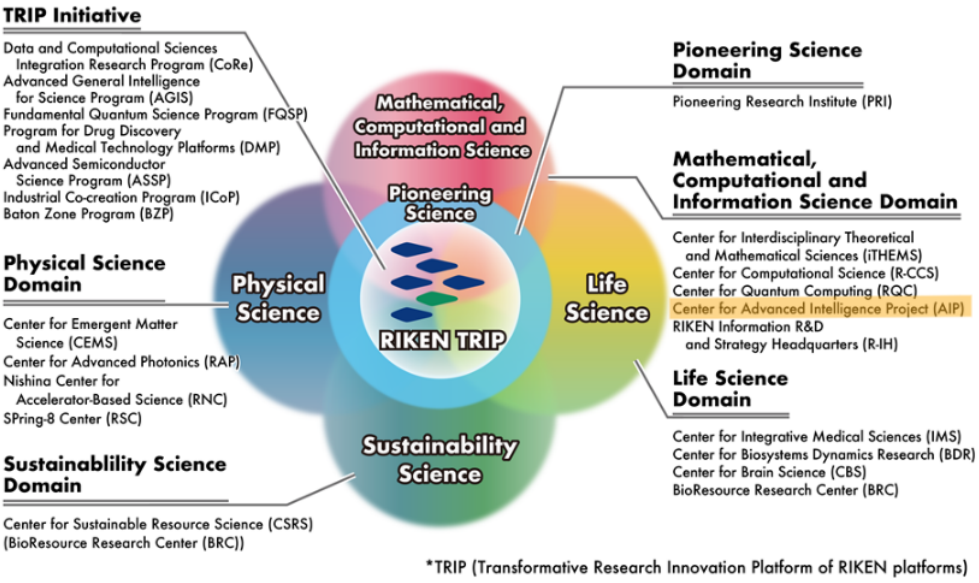


Figure 4: RIKEN's research centers and laboratories

1.1.1 About RIKEN Advanced Intelligence Project

The **RIKEN Center for Advanced Intelligence Project (AIP)** is a research center within RIKEN that focuses on advancing artificial intelligence (AI) and machine learning technologies.
AIP aims to develop innovative AI systems and algorithms that can address complex real-world problems across various domains, including healthcare, finance, robotics, and more.
The center conducts interdisciplinary research, combining expertise from computer science, mathematics, and domain-specific knowledge to create cutting-edge AI solutions.
AIP also collaborates with academic institutions, industry partners, and government organizations to promote the practical application of AI technologies and contribute to societal advancements.

RIKEN AIP is known for its commitment to ethical AI research and responsible AI deployment, ensuring that AI technologies are developed and used in ways that benefit society while addressing potential risks and challenges. more can be found on the RIKEN AIP website: https://aip.riken.jp/about-aip/

The roughly 40 teams at RIKEN AIP are organized into three main research groups, each focusing on different aspects of AI and machine learning:

- **Generic Technology Research Group**: goal is to develop generic AI technologies that can be applied across various domains.
  It includes teams like the Deep Learning Theory Team, tensor learning team, continuous optimization...
- **Goal-Oriented Technology Research Group**: focuses on the societal implications of AI and its applications in various fields.
  It includes teams like the Pathology Informatics Team, Statistical Genetics Team, Natural Language Understanding Team...
- **AI in Society Research Group**: studies the impact of AI and questions like security, big data etc.
  It includes teams like the Decentralized Big Data Team, AI Safety and Reliability Unit, AI Security and Privacy Team...

A complete list of the teams and more details for each of their research areas can be found on the following RIKEN AIP page: https://aip.riken.jp/labs-list/#no3.

1.1.2 About the Deep Learning Theory Team

The *Deep Learning Theory Team* at RIKEN AIP focuses on advancing the theoretical understanding of deep learning algorithms and their applications. More precisely it focuses on advancing theoretical understanding of deep learning and developing new methods and solutions for related machine learning problems based on theory.

It is lead since 2018 by **Taiji Suzuki**, professor at the University of Tokyo, and consists of around 20 members.
According to his words:

> "Our team, deep learning theory team, is studying various kinds of learning systems including deep learning from theoretical viewpoints. We enrich our understandings of complex learning systems, and leverage the insights to construct new machine learning techniques and apply them.
> Especially, machine learning should deal with high dimensional and complicated data, and thus we are studying deep learning and structured sparse learning as methods to deal with such complicated data.
> Moreover, we are also developing efficient optimization algorithms for large and complicated machine learning problems based on such techniques as stochastic optimization."

The team conducts research to improve the **expressivity**, **interpretability**, **robustness**, **generalization** capabilities and **efficiency of deep learning models**, addressing challenges in various domains such as computer vision, natural language processing, reinforcement learning...
The team aims to **bridge the gap between theoretical insights and practical implementations**, contributing to the development of more reliable and effective deep learning systems.
By exploring the mathematical foundations of deep learning, the team seeks to enhance the understanding of how these models learn and generalize from data, ultimately improving their performance in real-world applications.
More can be found on the RIKEN AIP Deep Learning Theory Team website:
https://aip.riken.jp/en/organization/deep-learning-theory-team/

Recent researches concerned many different subjects, including but not limited to SGD dynamics, NTK, mean-field Langevin dynamics, transformers, diffusion models, and many more.

1.2 About Machine Learning and Deep Learning

First of all, **Artificial Intelligence (AI)** is a broad field of computer science that aims to create systems capable of performing tasks that typically require human intelligence, such as understanding natural language, recognizing images, making decisions, and solving complex problems.
The first AI systems were rule-based, relying on explicit programming to define behavior. It is the case for instance of **Kernel methods** like **SVM** (see 1.4.3) or **Gaussian processes**, which are based on mathematical models and statistical principles.
These methods proved effective for specific well-defined problems but struggled to scale to the complexity and dimensionality of real-world applications.

Machine Learning (ML) is a subfield of AI concerned with the development of algorithms that can learn patterns from data and make decisions *without being explicitly programmed.*

Supervised learning constitutes the most prevalent and well-understood category of ML, characterized by training models on labeled datasets where input examples are paired with their corresponding target outputs. The learning process involves the model discovering an optimal mapping function from inputs to outputs by systematically minimizing the discrepancy between its predictions and the ground truth labels. This optimization process (detailed in Section 1.4.2) centers around the minimization of loss functions (explored in Section 1.4.2), which quantify the prediction error and guide the learning algorithm toward better performance.
The trained model's effectiveness is rigorously evaluated on independent test datasets that were not used during training, providing an unbiased assessment of its **generalization capabilities**. The fundamental objective of supervised learning is to achieve strong generalization—the ability to perform accurately on previously unseen data that follows the same underlying distribution as the training set. This generalization capability distinguishes truly useful models from those that merely memorize training examples.
Below are three situations that can be encountered after a training phase:

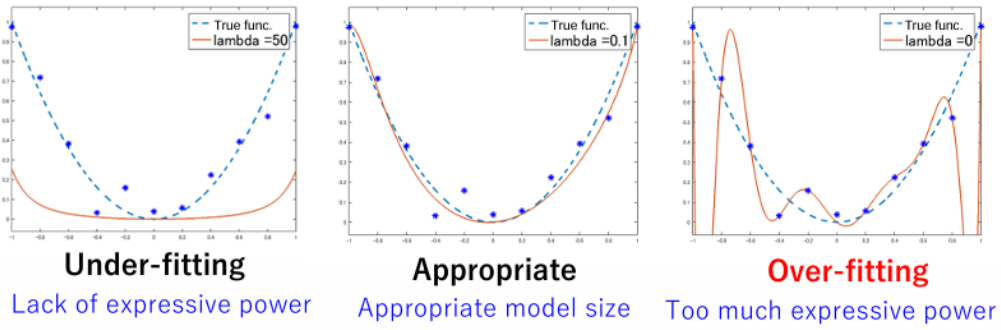


Figure 5: Training scenarios for supervised learning [14]

Unsupervised learning involves training a model on unlabeled data, where the model learns to identify patterns or structures in the data without explicit labels. Common techniques include *clustering* (grouping similar data points) and *dimensionality reduction* (reducing the number of features while preserving important information).
Over the past two decades, ML has evolved a lot, finding applications in fields as diverse as **computer vision, natural language processing (NLP), bioinformatics, robotics**...

A particularly exciting recent development in machine learning has been the emergence of **diffusion model**. These models learn to generate high-quality samples by simulating a gradual denoising process, starting from pure noise and iteratively refining it to produce realistic data. What makes diffusion models especially powerful is their ability to capture complex data distributions while maintaining stable training dynamics. However, adapting these models for specific tasks or aligning them with human preferences through **fine-tuning** presents unique challenges due to the stochastic nature of the generative process. Unlike traditional neural networks where gradients can be computed directly through deterministic forward passes, diffusion models require optimizing objectives that depend on the entire distribution of generated samples. This necessitates specialized optimization techniques such as **implicit diffusion** methods and **Dual Averaging method**s, which can handle the inherent stochasticity and perform distribution-level optimization rather than pointwise loss minimization. These advances enable more effective alignment of generative models with desired outcomes, opening new possibilities for controllable content generation and task-specific adaptation.

Deep Learning (DL), a specialized branch of ML, leverages neural network architectures with multiple layers (see 1.4.2) to automatically learn hierarchical representations from data [10]. DL has demonstrated unprecedented performance in complex tasks like **image classification**, **machine translation** [16], and **generative modeling** [7]. This performance surge has been driven by the availability of large datasets, increased computational power (e.g., GPUs), and architectural innovations.

In recent years, theoretical understanding of DL has become a central concern in the research community. Questions about *generalization, expressivity, optimization landscapes, and implicit biases of gradient-based methods* are actively studied.
For instance, despite being heavily overparameterized, modern neural networks often generalize well, contradicting classical statistical learning theory.

State-of-the-art (SOTA) models such as GPT-4 (already depreciated and Vision Transformers (ViT) have pushed the boundaries of scalability and performance.
However, understanding why these models work so effectively remains a largely open theoretical problem. Recent work focuses on developing mathematical tools to analyze deep networks in the infinite-width regime using neural tangent kernels (NTKs) (see [1] and part 1.4.3) or mean-field theory (see [11] and part 2.2).

Historically, deep learning traces its roots to early work on neural networks in the 1950s and 1980s, beginning with the perceptron and later the development of **multilayer perceptrons (MLPs)**. A pivotal breakthrough came with the reintroduction of the backpropagation algorithm in the 1980s, enabling effective training of deep networks.
In the late 1990s, convolutional neural networks (CNNs) were developed by Yann LeCun, notably the LeNet architecture for handwritten digit recognition.

A major resurgence of interest in DL occurred in 2012 with the success of AlexNet [9] in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which demonstrated the power of deep CNNs trained on GPUs. This success was followed by further innovations such as the ResNet architecture, which introduced residual connections and enabled training of ultra-deep networks.

The landscape shifted again with the introduction of the Transformer architecture [16], leading to a wave of progress in natural language processing and, more recently, computer vision through models like ViT. Modern large-scale models such as GPT-4, PaLM 2, and Gemini 1.5 represented the current state-of-the-art across multiple modalities and are already depreciated, showing how fast the field is evolving.

Major research breakthroughs in DL are typically published in top-tier conferences such as NeurIPS, ICML, ICLR, and CVPR. Pioneering researchers including Geoffrey Hinton, Yoshua Bengio, and Yann LeCun—joint recipients of the 2018 Turing Award—have played foundational roles in shaping the field.

1.3 Objectives of the internship

My objectives during this traineeship (https://aip.riken.jp/aip-osc2-0/) were to learn more about the theoretical aspects of deep learning and machine learning, as well as on the practical aspects of implementing and optimizing algorithms.

During my first month, I worked on the implementation of a **neural network framework in Python** from scratch (see 1.4), as I wanted to start with a solid basis and understand the inner workings of neural networks, which allowed me to learn a lot about various techniques and components used in DL.

I then focused on the mean-field regime and a particular approximation of it (see 2.2), trying to understand the current bounds on speed and accuracy of the approximation, and how it can be used to improve the training of neural networks.
I was asked to look at the underdamped case, which is more challenging than the overdamped case, and try to give a generalization of the results. Unfortunately, I did not succeed to give a generalization of the results for the underdamped case, as it took me longer than expected to understand the problem and run the experiments.

1.4 A first project: modularyNN python module

1.4.1 Overview and Architecture

I started my internship by implementing a project in Python of a **neural network** framework built **from scratch**, without relying on deep learning libraries like *tensorFlow* or *pyTorch*.
My goal was to gain a deeper understanding of the underlying mathematics and algorithms that power modern neural networks. I have also used it to *perform experiments and illustrate theoretical aspects I learned during my internship*.
The *git* repository for this project is available at: https://github.com/charles-vzf/modularyNN.

The **framework** provides a **modular architecture** with fundamental components such as layers (activation functions, core layers (FC, Conv, Pooling, Flatten...), regularization and normalization...), optimizers, and data handling utilities.

I have been able to succesfully implement and try several designs of neural networks, including RNN, CNN, NTK and tested various optimizers and architectures...

The main class *NeuralNetwork*, is an interface for designing a network by adding layers, training, evaluation, and visualization. It is designed to be flexible, allowing users to add custom layers, optimizers, and loss functions. The framework's modular design enables easy extension with new components as well as multiple purposes:

Classification. I started by implementing the framework to perform **classification** tasks. The datasets I used for testing are the following:

Dataset	Train Samples	Test Samples	Sample Shape	Classes
Random dataset	X	X	(X,X)	X
Iris	97	31	(4,)	3
Digit	1168	360	(1, 8, 8)	10
MNIST	45500	14000	(1, 28, 28)	10
CIFAR	39000	12000	(3, 32, 32)	10
Caltech101	5943	1830	(3, 64, 64)	102

Table 1: Datasets used for classification tests

Most of those datasets are well-known and widely used in the machine learning community, I also used synthetic datasets to test some specific features of the framework, with a flexible *random dataset* generator.

Regression. I also implemented the framework to perform **regression** tasks. The dataset I used for testing is a dataset coming from a *eleven strategy* challenge that aimed at predicting the waiting time of three attractions in an amusement park.
Here is an extract of the dataset to illustrate:

Table 2: Extract from `waiting_time_train.csv` [36 000 samples]

DATETIME	ENTITY	ADJ. CAP.	DOWNTIME	WAIT	TTP1	TTP2	TNS	WAIT 2H
2022-02-05 11:45	Water Ride	247.0	0	20				30.0
2019-02-24 10:45	Water Ride	247.0	0	30	375.0		495.0	25.0
2021-07-17 15:45	Pirate Ship	280.5	0	35				35.0
2022-04-03 19:45	Pirate Ship	230.4	0	15	-135.0		195.0	10.0
2021-10-20 10:30	Pirate Ship	153.0	0	15				10.0

Table 3: Extract from `weather_data.csv` [150 000 samples]

Temp	Dew Pt	Feels	Pressure	Humid.	Wind	Rain	Snow	Clouds	DATETIME
12.17	7.68	11.37	1019.0	74.0	3.3			100.0	2018-10-01 00:00
12.00	7.61	11.19	1019.0	74.5	3.26			99.25	2018-10-01 00:15
11.82	7.54	11.02	1019.0	75.0	3.22			98.5	2018-10-01 00:30
11.65	7.46	10.84	1019.0	75.5	3.19			97.75	2018-10-01 00:45

Use example: Linear classifier on the MNIST dataset.

```python
# Imports for the modularyNN framework
from Helpers import Helpers
from Layers.FullyConnected import FullyConnected
from Layers.SoftMax import SoftMax
from Layers.Flatten import Flatten
from Layers.Initializers import He, Constant, UniformRandom
from Optimization.Optimizers import Adam
from Optimization.Loss import CrossEntropyLoss
from Optimization.Constraints import L2_Regularizer

# Loading and displaying MINST dataset
batch_size = 100
mnist = DatasetClasses.MNISTData(batch_size) #loaded train, test and validation
# mnist.print_dataset_info()

def build_linear_classifier(input_dim, output_dim):
    # Create optimizer and regularizer
    learning_rate = 0.01
    optimizer = Adam(learning_rate)  #could use Sgd (similar results)
    regularizer = L2_Regularizer(alpha=0.001)
    optimizer.add_regularizer(regularizer)

    # Initialize the network with He initialization for weights
    weights_initializer = He()
    bias_initializer = Constant(0.1)  # Small constant for bias initialization
    net = NeuralNetwork(optimizer, weights_initializer, bias_initializer)

    # Add layers for linear classifier
    net.append_layer(Flatten()) # First flatten the input images (28x28x1) to a vector
    net.append_layer(FullyConnected(input_dim, output_dim))  # 28*28=784 input, 10 classes output
    net.append_layer(SoftMax()) # SoftMax for converting outputs to probabilities

    # Set loss function
    net.loss_layer = CrossEntropyLoss()

    return net

net = build_linear_classifier(input_dim=784, output_dim=10)
net.data_layer = mnist
net.plot()
```

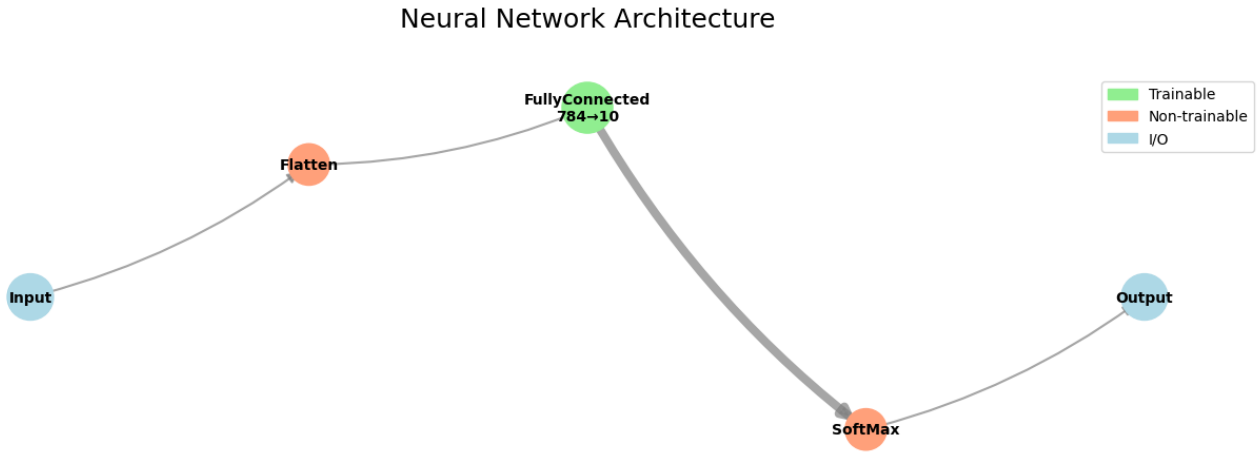


Figure 6: Linear classifier on the MNIST dataset (total trainable parameters: 7850)

```python
# Train the network
net.train(iterations)
# Get the test set
data, labels = net.data_layer.get_test_set()
# Test the network on the test set
results = net.test(data)
accuracy = Helpers.calculate_accuracy(results, labels)
```

```
Validation set detected and will be used for display.
 Iteration |          Loss |  Train Acc (%) |  Val Acc (%) | Elapsed Time
-----------------------------------------------------------------------
         0 |    238.241790 |          31.30 |         31.90 | 0.01s
        10 |     79.728500 |          81.20 |         78.60 | 0.05s
        20 |     40.741727 |          85.40 |         85.20 | 0.09s
        30 |     35.537803 |          86.00 |         85.60 | 0.12s
        40 |     41.187722 |          88.80 |         85.10 | 0.15s
                           [...]
       490 |     46.695422 |          91.90 |         92.40 | 2.00s
       499 |     24.734724 |          91.20 |         91.20 | 2.05s
-----------------------------------------------------------------------
Training completed in 2.07s

On the MNIST dataset, the linear classifier achieves an accuracy of: 91.7% on the
    test set
```

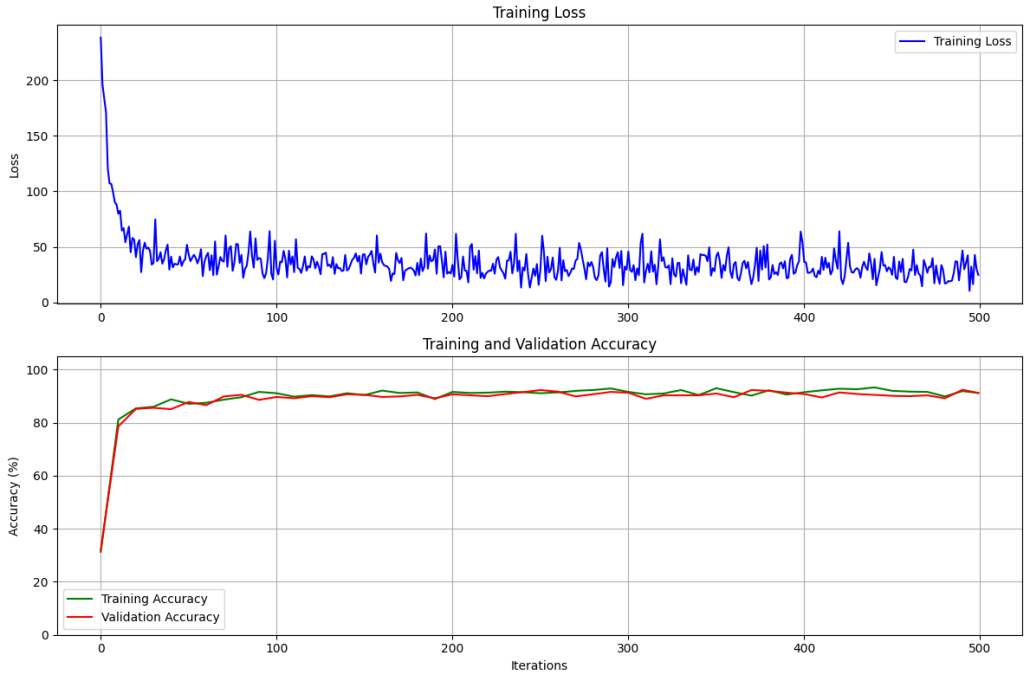


Figure 7: Loss and accuracy during training of the linear classifier on the MNIST dataset

```python
# Visualize the learned weights for each digit
fc_layer = net.layers[1]  # The FullyConnected layer
weights = fc_layer.weights[:-1, :]  # Exclude bias row
plt.figure(figsize=(15, 8))
for i in range(10):
    plt.subplot(2, 5, i+1)
    # Reshape the weights to 28x28 image dimensions
    weight_img = weights[:, i].reshape(28, 28)
    plt.imshow(weight_img, cmap='viridis')
    plt.title(f'Digit {i}')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

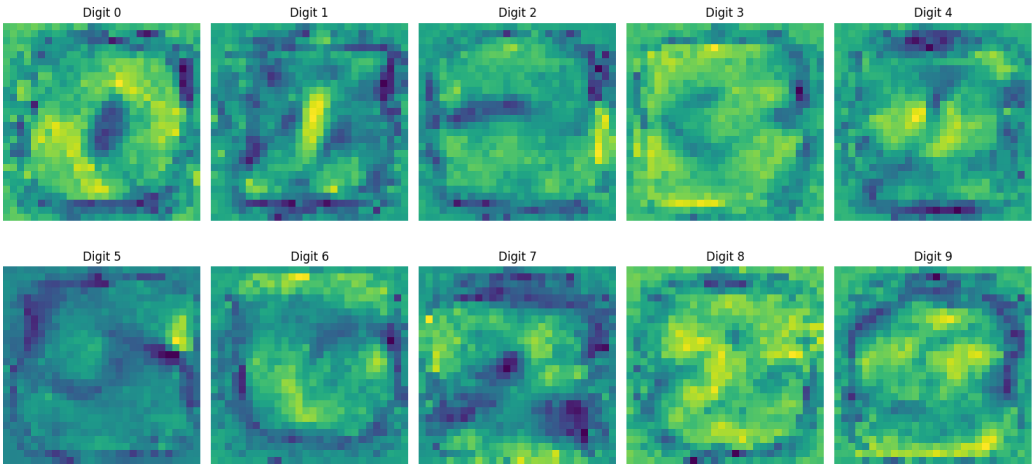


Figure 8: Learned weight patterns for each digit of the linear classifier on the MNIST dataset

More complex examples can by found in the /Models directory like the following convolutional neural network (CNN) designed to classify handwritten digits from the MNIST dataset. Each input is a 28×28 grayscale image and the architecture combines **convolutional layers (see 1.4.2) for feature extraction with fully connected layers (see 1.4.2) for classification**.

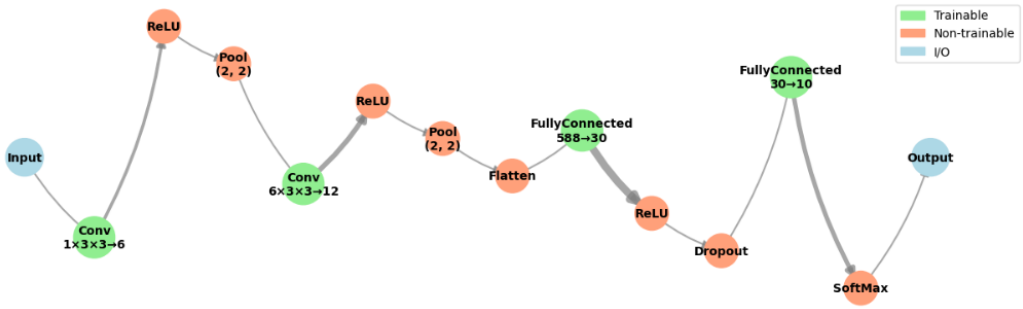


Figure 9: More complex CNN architecture with 18 700 trainable parameters

After a quick training, the model achieves an accuracy of over 92% on the MNIST test set (20%), demonstrating its effectiveness in digit classification.

Adding more convolutional kernels or changing the parameters distribution between *FC* and *conv* layers does not change significantly the final accuracy, but it does change the training time.
Each of the layers are briefly described in the following sections. Basically, what we expect from a network like that is to extract features from the input image by learning some kernels in the two convolutional layers, and then map those extracted features to relevant parameters and finally to the output classes in the fully connected layers.
The poolings and dropout are used to reduce the dimensionality of the data and prevent overfitting, respectively.

1.4.2 Neural Network Basics

Components. The framework implements the standard neural network architecture consisting of interconnected **layers** of **neurons**. Neurons are the basic computational units that receive inputs, apply a linear transformation, and pass the result through a non-linear activation function (a neuron is basically a weight in the network or multiple linked weights).
The main components of the framework include:

- **Layers**: Fully connected, convolutional, pooling, and recurrent layers
- **Activation Functions**: ReLU, Sigmoid, Tanh, Softmax
- **Loss Functions**: Cross-entropy, Mean Squared Error (MSE).
- **Optimizers**: Stochastic Gradient Descent (SGD), SGD with momentum, Adam
- **Regularization Techniques**: L1 and L2 regularization, Dropout, Batch Normalization
- **Data Handling**: Utilities for loading and preprocessing datasets, batching, flatten layer...
- **Testing**: components tests, numerical gradient tests, and training tests

General principle: evaluation and training. The general idea is to use our training set to qualibrate the weights of the network (initialized more or less randomly) in order to minimize the loss function between the predicted and expected outputs (basically fitting the datas while trying not to overfit, see figure below).
The framework uses the classical **backpropagation** logic to compute gradients and update weights. The steps are as follows:

1. **Forward Propagation**: Given an input, the information flows from first layer to output through a series of transformations determined by the layers encountered. It is deterministic and can be performed *on line* (one sample at the time) or *in batch* (multiple samples at once, doing matrix operations).
2. **Loss Calculation**: The difference between predicted and actual values is quantified.
3. **Backward Propagation**: The gradient of the loss is computed and propagated backwards to update parameters of each layer recursively (using the chain rule to compute gradients of the current layer from the preceding layers gradients) and the chosen optimization algorithm to update the parameters using those gradients.

A full path on the training set (either in batch or on line) is called an **epoch**. The training process involves multiple epochs, where the model iteratively refines its parameters to minimize the loss function.
Here is a typical graph of a training process, with the loss function decreasing and the accuracy of the training and validation sets increasing similarly on the first epochs, and then an explosion of the loss function with a drop of the accuracy of the validation set, indicating an overfitting of the model (the learning rate becomes too big after moving in another area, resulting in a gradient explosion).

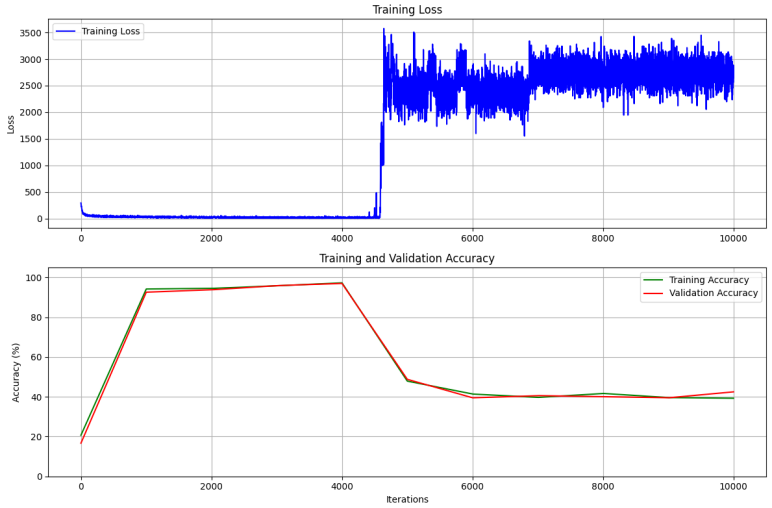


Figure 10: Overfitting and unstable SGD convergence on the MNIST dataset

Loss Functions. Loss functions quantify the discrepancy between predicted and actual values. The framework supports several standard loss functions, summarized in the following table.

Name	**Formula** $L(y)$	**Gradient** $\frac{\partial L(y)}{\partial \hat{y}}$	**Use Cases**
Hinge Loss	$\sum \max(0, 1 - y\hat{y})$	$-y \cdot \mathbb{1}_{(1-y\hat{y}>0)}$	Binary classification (SVMs)
Logistic Loss	$\sum \log(1 + e^{-y\hat{y}})$	$-\frac{y}{1+e^{y\hat{y}}}$	Binary classification (logistic regression)
Cross Entropy (CE)	$-\sum y \log(\hat{y} + \epsilon)$	$-\frac{y}{\hat{y}+\epsilon}$	Classification with softmax
Mean Absolute Error (MAE / L1)	$\frac{1}{n}\sum \lvert y - \hat{y}\rvert$	$\frac{1}{n}\text{sign}(\hat{y} - y)$	Robust regression (outliers)
Mean Squared Error (MSE)	$\frac{1}{n}\sum (y - \hat{y})^2$	$\frac{2}{n}(\hat{y} - y)$	Regression
L2 Loss	$\frac{1}{2n}\sum (y - \hat{y})^2$	$\frac{1}{n}(\hat{y} - y)$	Optimization-friendly squared loss
Exponential Loss	$\sum e^{-y\hat{y}}$	$-y \cdot e^{-y\hat{y}}$	Boosting (e.g., AdaBoost)

Table 4: Summary of supported loss functions, their gradients, and common use cases.

Many other loss functions exists and they all have different purposes and properties. It is easy to add a new one for tests, which I did on some occasions.

Activation Functions. Activation functions introduce non-linearity into the network by transforming the output of each neuron. They determine the output of a neuron given its input. The choice of activation function can significantly impact the network's performance and convergence speed. They are basically the function that transforms the output of a neuron before passing it to the next layer, acting a bit like basis functions in a linear combination allowing to map the input to the output space.
The framework implements several common activation functions:

Activation	**Function** $f(x)$	**Derivative** $f'(x)$	**Typical Use Case**
ReLU	$\max(0, x)$	$\begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$	Most common in hidden layers due to simplicity and avoiding vanishing gradients
Sigmoid	$\frac{1}{1+e^{-x}}$	$f(x)(1 - f(x))$	Binary classification output; less common in hidden layers due to vanishing gradients
TanH	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - \tanh^2(x)$	Used in hidden layers (especially RNNs); zero-centered, but can still saturate
Softmax	$\frac{e^{x_i}}{\sum_j e^{x_j}}$	$\hat{y}_i - y_i$ (w/ cross-entropy)	Final layer for multi-class classification; produces probabilities over classes

Table 5: Summary of common activation functions and their derivatives

Regularizations. Regularization techniques prevent overfitting by adding a penalty term to the loss function. The framework implements common regularization methods:

L1 Regularization (Lasso)
L1 regularization adds the absolute values of the weights to the loss function. This encourages sparsity in the model by pushing some weights to exactly zero, effectively performing feature selection. It is useful when the resulting model is expected to have many input features irrelevant or redundant, with a sparse model desirable (e.g., high-dimensional data like text or genomics).

$$\text{Loss:} \quad L_{reg} = L + \alpha \sum_{i=1}^{n} |w_i| \qquad \text{Gradient:} \quad \frac{\partial L_{reg}}{\partial w_i} = \frac{\partial L}{\partial w_i} + \alpha \cdot \text{sign}(w_i)$$

L2 Regularization (Ridge)
L2 regularization adds the square of the weights to the loss function. This discourages large weights without

necessarily driving them to zero, leading to more stable and smooth models. Suitable when all features are expected to contribute and overfitting is a concern, especially in regression or neural networks.

$$\text{Loss:} \quad L_{reg} = L + \alpha \sum_{i=1}^{n} w_i^2 \qquad \text{Gradient:} \quad \frac{\partial L_{reg}}{\partial w_i} = \frac{\partial L}{\partial w_i} + 2\alpha \cdot w_i$$

Dropout
Dropout is a regularization technique that randomly sets a fraction of the inputs to zero during training, reducing interdependent learning among neurons and helping to prevent overfitting. We have that $y = m \odot x$ where m is a binary mask with elements drawn from a Bernoulli distribution with probability p of being 1, and $\odot$ represents element-wise multiplication.

In term of implementation, the dropout layer uses inverted dropout, where during training a binary mask is applied and scaled to preserve the expected value. In the backward pass, gradients are only propagated through the same active neurons using the stored mask, ensuring proper gradient flow only for retained units.

Batch Normalization. Batch Normalization normalizes the activations of each layer per mini-batch, stabilizing and accelerating training. It is typically used after the convolutional or fully connected layers and before the activation function to improve convergence and reduce sensitivity to initialization. It preserves representational power while mitigating internal covariate shift, allowing higher learning rates.
It operates independently on each *feature channel* [4], normalizing all spatial locations within that channel across the mini-batch. It introduces two learnable parameters per *feature channel*: a **scale** γ **and a shift** β, enabling the network to undo normalization if needed.
For forward pass, given an input x_i in a mini-batch B of size m, it computes the mean μ_B and variance σ_B^2 of the batch:

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i \qquad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$$

Then, it normalizes the input:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad y_i = \gamma \cdot \hat{x}_i + \beta$$

To backpropagate through batch normalization, we compute the gradient of the loss with respect to the input x_i, accounting for the dependency of normalization on all samples in the batch. This involves chain rule application through $\hat{x}_i$ and the normalization function. The gradient w.r.t. input, using saved statistics, is:

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \left(\frac{\partial L}{\partial y_i} \cdot \gamma - \frac{1}{m} \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} \cdot \gamma - \frac{\hat{x}_i}{m} \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} \cdot \gamma \cdot \hat{x}_j \right)$$

The parameters γ and β are updated via gradient descent:

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \cdot \hat{x}_i \qquad \frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}$$

Note that during training, μ_B and σ_B^2 are computed from the mini-batch and used to update exponentially moving averages (running statistics) with a fixed small **momentum** α, to be used in :

$$\mu_{\text{running}} \leftarrow \alpha \cdot \mu_{\text{running}} + (1 - \alpha) \cdot \mu_B \qquad \sigma^2_{\text{running}} \leftarrow \alpha \cdot \sigma^2_{\text{running}} + (1 - \alpha) \cdot \sigma_B^2$$

At inference time, $\hat{x}_i$ is computed using these running statistics instead of the current batch's.

Fully Connected Layer. This layer is often used as final layer combined with a softmax to perform classification tasks, or simply as a connector to change the dimension of the input data.
The fully connected layer performs a linear transformation of its input. For a batch of inputs $X \in \mathbb{R}^{n \times d}$ (where n is the batch size and d is the input dimension), the output $Y \in \mathbb{R}^{n \times m}$ (where m is the output dimension) is computed as:

$$Y = XW + b$$

[4] refers to a specific dimension of the input tensor that holds distinct learned features—such as edges, textures, or patterns—typically corresponding to the depth dimension in convolutional layers (e.g., RGB channels or learned filters).

where $W \in \mathbb{R}^{d \times m}$ is the weight matrix and $b \in \mathbb{R}^m$ is the bias vector.

During backpropagation, given the gradient of the loss with respect to the output $\frac{\partial L}{\partial Y}$, the gradients with respect to the weights and input are computed as:

$$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y} \qquad \frac{\partial L}{\partial b} = \sum_{i=1}^{n} \frac{\partial L}{\partial Y_i} \qquad \frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W^T$$

Convolutional Layer. Convolutional layers extract meaningful local patterns from input data, such as edges or textures in images. It does this by **applying learnable filters over small spatial regions, enabling the network to detect features** regardless of their position. The use of shared weights across the input reduces the number of parameters, making the model more efficient. Stacking multiple convolutional layers allows the network to build hierarchical feature representations. Overall, convolutional layers are essential for recognizing complex structures in data while maintaining computational scalability.
For a 2D input tensor $X \in \mathbb{R}^{n \times c_{in} \times h \times w}$ (batch size, input channels, height, width), the output $Y \in \mathbb{R}^{n \times c_{out} \times h' \times w'}$ is computed as:

$$Y_{i,j,p,q} = \sum_{l=1}^{c_{in}} \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} X_{i,l,s_h \cdot p+m, s_w \cdot q+n} \cdot W_{j,l,m,n} + b_j$$

where:

- $Y_{i,j,p,q}$ is the output at position (i, j, p, q) (batch, output channel, spatial positions)
- $X_{i,l,m,n}$ is the input at position (i, l, m, n) (batch, input channel, spatial positions)
- $W_{j,l,m,n}$ is the weight for output channel j, input channel l, and spatial position (m, n)
- b_j is the bias for output channel j
- s_h and s_w are the stride in height and width dimensions
- k_h and k_w are the kernel height and width

During backpropagation, the gradients for the convolutional layer are computed through the convolution operation with flipped kernels:

$$\frac{\partial L}{\partial X_l} = \sum_{j=1}^{c_{out}} \text{Conv2D}\left(\frac{\partial L}{\partial Y_j}, \text{rot180}(W_{j,l})\right), \quad \frac{\partial L}{\partial W_{j,l}} = \text{Conv2D}\left(X_l, \frac{\partial L}{\partial Y_j}\right), \quad \frac{\partial L}{\partial b_j} = \sum_{i=1}^{n} \sum_{p=1}^{h'} \sum_{q=1}^{w'} \frac{\partial L}{\partial Y_{i,j,p,q}}$$

Here, $\text{rot180}(W_{j,l})$ denotes a 180° rotation of the kernel $W_{j,l}$. The 2D cross-correlation operation is defined as:

$$\text{Conv2D}(A, B)_{p,q} = \sum_{l=1}^{c} \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} A_{l,p+m,q+n} \cdot B_{l,m,n}$$

for an input tensor $A \in \mathbb{R}^{c \times h \times w}$ and kernel $B \in \mathbb{R}^{c \times k_h \times k_w}$. This defines a valid (no padding) cross-correlation over all input channels.

Pooling Layer. Pooling layers are here to **reduce the spatial dimensions of the input feature maps**, effectively downsampling them while retaining the most salient features. They are commonly used in deep learning architectures, especially in computer vision tasks, to reduce the computational burden and improve the model's ability to generalize.

Consider an image processed through several convolutional layers, resulting in feature maps that highlight various patterns like edges or textures. Pooling layers operate on these feature maps by sliding a window (e.g., 2×2 or 3×3) across them and summarizing the information within each window (most common operations are max pooling and average pooling).
These layers are typically inserted after convolutional layers to progressively reduce the spatial dimensions of the feature maps, allowing the network to focus on the most important features and reducing the risk of overfitting, making the network more efficient and less sensitive to minor translations in the input.

Let $X \in \mathbb{R}^{B \times C \times H \times W}$ denote the input tensor, where B is the batch size, C is the number of channels, H and W are the height and width of the feature maps. We note (p_h, p_w) the pooling window size and (s_h, s_w) the stride. The output dimensions are computed as:

$$H_{\text{out}} = \left\lfloor \frac{H - p_h}{s_h} \right\rfloor + 1, \quad W_{\text{out}} = \left\lfloor \frac{W - p_w}{s_w} \right\rfloor + 1.$$

For max pooling, the output at position (i, j) in channel c for sample b is:

$$Y_{b,c,i,j} = \max_{0 \le m < p_h,\, 0 \le n < p_w} X_{b,c,i \cdot s_h + m, j \cdot s_w + n}.$$

For the backward pass of max pooling, since only the maximum value within each pooling window affects the output, the gradient is assigned exclusively to the positions corresponding to these maxima as follows, with:

- $\delta_{b,c,i,j}$ the gradient of the loss with respect to the output at position (b, c, i, j), i.e., the incoming gradient from the next layer (also called the error tensor)
- $Z = \nabla_X \mathcal{L} \in \mathbb{R}^{B \times C \times H \times W}$ the gradient of the loss $\mathcal{L}$ with respect to the input tensor X, we have:

$$Z_{b,c,h,w} = \begin{cases} \delta_{b,c,i,j} & \text{if } (h,w) = \arg \max\limits_{0 \le m < p_h,\, 0 \le n < p_w} X_{b,c,i \cdot s_h + m, j \cdot s_w + n} \\ 0 & \text{otherwise} \end{cases}$$

Recurrent Layer. Recurrent Neural Networks (RNNs) are a class of neural networks specifically designed for processing sequential data, such as time series, natural language, or any other temporally dependent signal. Unlike **feedforward** networks, RNNs introduce cycles in the network architecture by **maintaining a hidden state between two FC layers** that captures information from previous time steps, enabling temporal memory.
The key idea behind RNNs is to **allow information** to persist over time. At each time step, the network receives an input and updates its internal hidden state based on both the new input and the previous hidden state. This design allows RNNs to model dependencies across time, making them particularly effective for tasks such as language modeling, speech recognition, and sequential prediction.

The RNN layer maintains a hidden state $\mathbf{h}_t$ at each time step t, which is computed from the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ following the steps:

$$\mathbf{u}_t := \mathbf{W}_{xh} \cdot [\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_{xh} \quad \rightarrow \quad \mathbf{h}_t = \tanh(\mathbf{u}_t) \quad \rightarrow \quad \mathbf{o}_t := \mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_{hy} \quad \rightarrow \quad \mathbf{y}_t = \sigma(\mathbf{o}_t)$$

where, $\mathbf{W}_{xh}$ and $\mathbf{W}_{hy}$ are the weight matrices for the input-to-hidden and hidden-to-output transformations, respectively. The operator $[\mathbf{x}_t; \mathbf{h}_{t-1}]$ denotes the concatenation of the input and the previous hidden state.

The backward pass computes gradients using *Backpropagation Through Time (BPTT)*. At each time step t, gradients flow backward through the output layer and are propagated through the hidden layer and the input-hidden connections.
Let $\delta^{(t)}$ be the error gradient at the output for time step t:

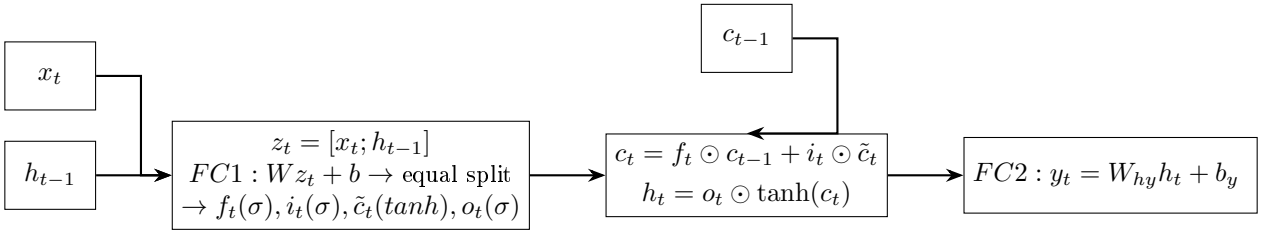
$$\delta_o^{(t)} = \sigma'(\mathbf{o}_t) \odot \frac{\partial \mathcal{L}}{\partial \mathbf{y}_t} \qquad \delta_h^{(t)} = (\delta_o^{(t)} \cdot \mathbf{W}_{hy}^\top + \delta_h^{(t+1)}) \odot \tanh'(\mathbf{u}_t)$$

$$\nabla \mathbf{W}_{xh} += \delta_h^{(t)} \cdot [\mathbf{x}_t; \mathbf{h}_{t-1}]^\top \qquad \nabla \mathbf{W}_{hy} += \delta_o^{(t)} \cdot \mathbf{h}_t^\top$$

The total gradient with respect to the input at each time step is computed by splitting the gradient $\delta_h^{(t)}$ across the input and previous hidden state dimensions.

LSTM Layer. Long Short-Term Memory (LSTM) networks are an extension of recurrent neural networks (RNNs) designed to address the **vanishing gradient problem** through **gating mechanisms**. These gates regulate the flow of information, allowing the model to retain important long-range dependencies over time.

At each time step t, the LSTM receives an input $x_t \in \mathbb{R}^n$, the previous hidden state $h_{t-1} \in \mathbb{R}^m$, and the cell state $c_{t-1} \in \mathbb{R}^m$, producing updated states h_t, c_t, and an output y_t. Here is how the forward pass is done:



Note that the split operation after the first fully connected layer $FC1$ produces four vectors, each corresponding to a **gate**:

- **Forget gate** f_t: Controls how much of the previous cell state c_{t-1} should be kept. Uses a sigmoid activation.
- **Input gate** i_t: Determines how much new information from the candidate cell state $\tilde{c}_t$ should be added to the memory. Uses a sigmoid activation.
- **Candidate (cell input)** $\tilde{c}_t$: Represents new candidate values to add to the memory cell. Uses a tanh activation.
- **Output gate** o_t: Controls how much of the cell state c_t should be exposed through the hidden state h_t. Uses a sigmoid activation.

For the backward pass, we compute the gradients of the loss with respect to the outputs and intermediate states by propagating derivatives backward through the computational graph. Gradients are passed in from the output layer ($\frac{\partial \mathcal{L}}{\partial y_t}$) and then recursively through time.

$$\frac{\partial \mathcal{L}}{\partial h_t} = W_h y^\top \frac{\partial \mathcal{L}}{\partial y_t} + \frac{\partial \mathcal{L}}{\partial h_{t+1}}$$

This combines the gradient from the output layer and the recurrent gradient from the next time step. Next, we backpropagate through the output gate and cell computations:

$$\frac{\partial \mathcal{L}}{\partial o_t} = \frac{\partial \mathcal{L}}{\partial h_t} \odot \tanh(c_t) \odot o_t(1 - o_t) \qquad \frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) + \frac{\partial \mathcal{L}}{\partial c_{t+1}}$$

Here, $\frac{\partial \mathcal{L}}{\partial c_t}$ includes both the influence of the hidden state and the gradient passed through time. Then we compute gradients through the remaining gates:

$$\frac{\partial \mathcal{L}}{\partial f_t} = \frac{\partial \mathcal{L}}{\partial c_t} \odot c_{t-1} \odot f_t(1 - f_t) \qquad \frac{\partial \mathcal{L}}{\partial i_t} = \frac{\partial \mathcal{L}}{\partial c_t} \odot \tilde{c}_t \odot i_t(1 - i_t) \qquad \frac{\partial \mathcal{L}}{\partial \tilde{c}_t} = \frac{\partial \mathcal{L}}{\partial c_t} \odot i_t \odot (1 - \tilde{c}_t^2)$$

These expressions follow from the chain rule applied to the cell state update: $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$, considering the activation functions used in each gate.
Finally, the partial derivatives of the loss with respect to the concatenated input $z_t = [x_t; h_{t-1}]$ are computed via the transpose of the weights of the gate layer:

$$\text{Let} \quad \frac{\partial \mathcal{L}}{\partial z_t} = W^\top \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial f_t} \\ \frac{\partial \mathcal{L}}{\partial i_t} \\ \frac{\partial \mathcal{L}}{\partial \tilde{c}_t} \\ \frac{\partial \mathcal{L}}{\partial o_t} \end{bmatrix} \qquad \text{Then} \quad \left(\frac{\partial \mathcal{L}}{\partial x_t}, \frac{\partial \mathcal{L}}{\partial h_{t-1}} \right) = \text{split} \left(\frac{\partial \mathcal{L}}{\partial z_t} \right)$$

This gives us the gradients to propagate further backward through the input and recurrent paths. Gradients w.r.t. the weights are accumulated at each step, allowing optimization via backpropagation through time (BPTT).

Optimization algorithms. Optimization algorithms update the network parameters based on computed gradients (computed step by step during the backpropagation starting from the loss layer). Each layer is responsible for computing its own gradients, updating its parameters using the choosen optimization algorithm and passing the resulting final gradient to the previous layer.

Weight Initialization Strategies. Proper weight initialization is crucial for training deep networks. The framework implements several strategies as summarized in the following table:

Name	**Distribution**	**Classical Use**
Constant	$W_{i,j} = \text{value}$	Biaises; Testing/debugging; simple baselines
Uniform Random	$W_{i,j} \sim U(0,1)$	Uninformed initialization; not activation-aware
Xavier (Glorot)	$W_{i,j} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}+n_{out}}}\right)$	Tanh or sigmoid activations
He	$W_{i,j} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}}}\right)$	ReLU or variants (LeakyReLU, ELU)

Stochastic Gradient Descent (SGD).
The simplest optimization algorithm that updates weights using the gradient:

$$w_{t+1} = w_t - \eta \cdot \nabla L(w_t)$$

where η is the learning rate and $\nabla L(w_t)$ is the gradient of the full loss (with regularization) with respect to the weights.

SGD with Momentum.
Adds a momentum term to accelerate convergence and reduce oscillation:

$$v_{t+1} = \mu \cdot v_t + \eta \cdot \nabla L(w_t) \qquad w_{t+1} = w_t - v_{t+1}$$

where μ is the momentum coefficient (typically 0.9) and v_t is the velocity vector.

Adam (Adaptive Moment Estimation).
Combines the benefits of *AdaGrad* and *RMSProp* by keeping track of both the first and second moments of the gradients:

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla L(w_t) \qquad v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot (\nabla L(w_t))^2$$

where β_1 and β_2 are decay rates for the first and second moment estimates (typically 0.9 and 0.999), m_t and v_t are the biased first and second moment estimates;

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

$\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected moment estimates, and the update rule is:

$$w_{t+1} = w_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

ϵ is a small constant to prevent division by zero.

1.4.3 Examples

Kernel-based SVM Classifier.

The RBF Kernel layer transforms input vectors into similarity features by computing **Gaussian kernel values** between inputs and **prototype vectors** (also known as support vectors). The RBF kernel computes a similarity score between data points based on their distance in the input space. It assigns high similarity values to points that are close to each other and lower values to points that are farther apart.

The RBF Kernel Function is $K(\mathbf{x}, \mathbf{p}) = \exp(-\gamma \|\mathbf{x} - \mathbf{p}\|^2)$ where $\mathbf{x}$ is the input vector, $\mathbf{p}$ is the prototype vector, γ is a kernel parameter that controls the width of the Gaussian.

For forward pass given a batch input $\mathbf{X} \in \mathbb{R}^{B \times d}$ and prototypes $\mathbf{P} \in \mathbb{R}^{N \times d}$ we get the similarity matrix $\mathbf{K} \in \mathbb{R}^{B \times N}$ with elements: $\mathbf{K}_{ij} = \exp(-\gamma \cdot \|\mathbf{X}_i - \mathbf{P}_j\|^2)$, where $\mathbf{X}_i$ is the i-th input vector and $\mathbf{P}_j$ is the j-th prototype vector.

For the backward pass we compute the gradient of the loss with respect to the kernel matrix $\mathbf{K}$ using the chain rule: $\frac{\partial L}{\partial \mathbf{x}} = \sum_{j=1}^{N} \frac{\partial L}{\partial K_{ij}} \cdot \frac{\partial K_{ij}}{\partial \mathbf{x}}$, by knowing the kernel gradient with respect to input given by: $\frac{\partial K(\mathbf{x},\mathbf{p})}{\partial \mathbf{x}} = -2\gamma \cdot K(\mathbf{x}, \mathbf{p}) \cdot (\mathbf{x} - \mathbf{p})$.

A simple implementation of a nontrainable gaussian radial basis function kernel (RBF Kernel) preceding a fully connected layer with softmax activation and cross-entropy loss already gives really good results on the MNIST dataset (over 90% accuracy achieved by modularyNN, and in litterature over 95%).

Neural Tangent Kernel (NTK).

The Neural Tangent Kernel (NTK) provides a theoretical framework for analyzing the training dynamics of neural networks in the infinite-width limit. Introduced by [1], the NTK connects neural network training with kernel methods by showing that **infinitely wide networks behave like Kernel methods during gradient descent**.
For a neural network $f(\mathbf{x}, \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, the NTK is defined as: $K_{\text{NTK}}(\mathbf{x}, \mathbf{x}') = \nabla_{\boldsymbol{\theta}} f(\mathbf{x}, \boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} f(\mathbf{x}', \boldsymbol{\theta})$. This kernel is symmetric positive semi-definite and uses the gradient of the neural network as a feature map.

In the infinite-width limit, the **NTK remains constant during training**, kept at its random initialization value, allowing neural network optimization to be analyzed through the lens of **kernel ridge regression**. The network's predictions evolve according to:

$$\frac{df(\mathbf{x}, t)}{dt} = \sum_j \frac{\partial f(\mathbf{x}, t)}{\partial \theta_j} \frac{d\theta_j}{dt} = -\eta \sum_j \frac{\partial f(\mathbf{x}, t)}{\partial \theta_j} \sum_i \frac{\partial L}{\partial f(\mathbf{x}_i, t)} \frac{\partial f(\mathbf{x}_i, t)}{\partial \theta_j} = -\eta \sum_i K_{\text{NTK}}(\mathbf{x}, \mathbf{x}_i) \frac{\partial L}{\partial f(\mathbf{x}_i, t)}$$

where η is the learning rate, L is the loss function and $K_{\text{NTK}}(\mathbf{x}, \mathbf{x}_i)$ is the Neural Tangent Kernel value between test point $\mathbf{x}$ and training point $\mathbf{x}_i$. We used the NTK definition and the parameter update rule of gradient descent:

$$K_{\text{NTK}}(\mathbf{x}, \mathbf{x}') = \sum_j \frac{\partial f(\mathbf{x})}{\partial \theta_j} \frac{\partial f(\mathbf{x}')}{\partial \theta_j} \qquad \frac{d\theta_j}{dt} = -\eta \frac{\partial L}{\partial \theta_j} = -\eta \sum_i \frac{\partial L}{\partial f(\mathbf{x}_i, t)} \frac{\partial f(\mathbf{x}_i, t)}{\partial \theta_j}$$

In the limit $t \to \infty$, the network converges to the kernel method solution: $f(\mathbf{x}, \infty) = f(\mathbf{x}, 0) + \sum_i K_{\text{NTK}}(\mathbf{x}, \mathbf{x}_i)\alpha_i$, where $\boldsymbol{\alpha}$ is the solution to the kernel ridge regression problem: $\min_{\boldsymbol{\alpha}} \ \frac{1}{2} \|K_{\text{NTK}}\boldsymbol{\alpha} - \mathbf{y} + \mathbf{f}(0)\|^2 + \frac{\lambda}{2} \boldsymbol{\alpha}^T K_{\text{NTK}} \boldsymbol{\alpha}$ with $(K_{\text{NTK}})_{ij} = K_{\text{NTK}}(\mathbf{x}_i, \mathbf{x}_j)$ and λ the regularization parameter.

This establishes the equivalence between infinite-width neural networks and kernel methods, showing that such networks cannot perform feature learning and are limited to the expressiveness of their initial random features.

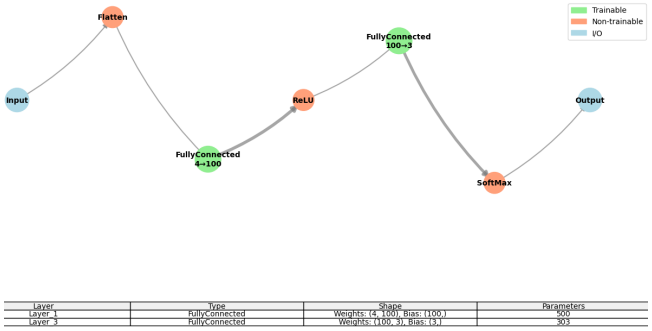
Computationally, solving this kernel ridge regression problem yields the closed-form solution (by taking the gradient of the objective function with respect to $\boldsymbol{\alpha}$ and setting it to zero): $\boldsymbol{\alpha} = (K_{\text{NTK}} + \lambda I)^{-1}\mathbf{r}$ where $\mathbf{r} = \mathbf{y} - \mathbf{f}(0)$ are the residuals. This solution often requires **Singular Value Decomposition**, a matrix factorization technique that decomposes the kernel matrix $K_{\text{NTK}} = U\Sigma V^T$ into orthogonal matrices U, V and diagonal matrix Σ. The SVD-based solution becomes $\boldsymbol{\alpha} = V(\Sigma + \lambda I)^{-1} U^T \mathbf{r}$, providing numerical stability when inverting ill-conditioned kernel matrices by avoiding direct matrix inversion.

Eventhough the NTK computation is **computationally expensive** for practical networks, it provides valuable theoretical insights into why over-parameterized networks train effectively and converge to global minima.

For finite-width networks, the NTK approximation often captures essential training dynamics, with wider networks exhibiting behavior closer to the theoretical infinite-width limit.

Computing the NTK for a Two-Layer ReLU Network on the Iris dataset.

For the specific architecture with input dimension $d_{in} = 4$, hidden dimension h, and output dimension $d_{out} = 3$, we can analytically compute the NTK in the infinite-width limit $h \to \infty$. Here is the network structure we considered with $h = 100$ (similar to the linear classifier example):



Layer	Type	Shape	Parameters
Layer 1	FullyConnected	Weights: (4, 100), Bias: (100,)	500
Layer 3	FullyConnected	Weights: (100, 3), Bias: (3,)	303

Figure 11: Two-layer ReLU network with $h = 100$ hidden units (803 trainable parameters)

The network function is $f(\mathbf{x}) = \mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}$, where σ is the ReLU activation. For analytical tractability, we neglect the biases and consider $f(\mathbf{x}) = \frac{1}{\sqrt{h}}\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x})$ with He initialization $W_{ij}^{(1)} \sim \mathcal{N}(0, \sqrt{\frac{2}{d_{\text{in}}}})$ and $W_{ki}^{(2)} \sim \mathcal{N}(0, \sqrt{\frac{2}{h}})$.

The NTK is given by, with $\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$: $K_{\text{NTK}}(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d_{out}} \nabla_{\boldsymbol{\theta}} f_k(\mathbf{x}) \cdot \nabla_{\boldsymbol{\theta}} f_k(\mathbf{x}')$

- For first-layer weights $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d_{in}}$: $\frac{\partial f_k(\mathbf{x})}{\partial W_{ij}^{(1)}} = \frac{1}{\sqrt{h}} W_{ki}^{(2)} \mathbb{1}[\mathbf{W}_i^{(1)}\mathbf{x} > 0]\, x_j$
- For second-layer weights $\mathbf{W}^{(2)} \in \mathbb{R}^{d_{out} \times h}$: $\frac{\partial f_k(\mathbf{x})}{\partial W_{ki}^{(2)}} = \frac{1}{\sqrt{h}}\sigma(\mathbf{W}_i^{(1)}\mathbf{x})$

The full NTK expression is:

$$K_{\text{NTK}}(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d_{out}} \left[\sum_{i=1}^{h} \sum_{j=1}^{d_{in}} \frac{1}{h} \left(W_{ki}^{(2)}\right)^2 \mathbb{1}[\mathbf{W}_i^{(1)}\mathbf{x} > 0]\mathbb{1}[\mathbf{W}_i^{(1)}\mathbf{x}' > 0] x_j x_j' + \sum_{i=1}^{h} \frac{1}{h}\sigma(\mathbf{W}_i^{(1)}\mathbf{x})\sigma(\mathbf{W}_i^{(1)}\mathbf{x}') \right]$$

In the infinite-width limit $h \to \infty$, by the law of large numbers and the NTK initialization scaling, this random sum converges to its expectation over the weights:

$$K_{\text{NTK}}(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d_{out}} \left[\frac{2}{h} \cdot \mathbf{x}^\top \mathbf{x}' \cdot \mathbb{E}_{w \sim \mathcal{N}\left(0, \sqrt{\frac{2}{d_{in}}} I_{d_{in}}\right)} [\mathbb{1}[w^\top \mathbf{x} > 0]\mathbb{1}[w^\top \mathbf{x}' > 0]] + \mathbb{E}_{w \sim \mathcal{N}\left(0, \sqrt{\frac{2}{d_{in}}} I_{d_{in}}\right)} [\sigma(w^\top \mathbf{x})\sigma(w^\top \mathbf{x}')] \right]$$

where we used that $\mathbb{E}_{W_{ki}^{(2)} \sim \mathcal{N}(0, \sqrt{\frac{2}{h}})}[(W_{ki}^{(2)})^2] = \frac{2}{h}$ and the independence of the weights $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$. The resulting NTK expression consists of two distinct contributions: the first term captures the "tangent" component arising from gradients with respect to the first layer weights, which depends on the correlation between inputs $\mathbf{x}$ and $\mathbf{x}'$ weighted by the probability that both activate the same hidden units, while the second term represents the "neural" component from second layer gradients, corresponding to the standard neural network Gaussian process kernel.

The expectations can be computed analytically. Let $\theta = \arccos\left(\frac{\mathbf{x}^\top \mathbf{x}'}{|\mathbf{x}||\mathbf{x}'|}\right)$ be the angle between $\mathbf{x}$ and $\mathbf{x}'$. The first expectation $\mathbb{E}[\mathbb{1}[w^\top \mathbf{x} > 0]\mathbb{1}[w^\top \mathbf{x}' > 0]]$ represents the probability that both $\mathbf{x}$ and $\mathbf{x}'$ lie in the same half-space defined by the random hyperplane w, which equals $\frac{1}{2} - \frac{1}{2\pi}\theta$. The second expectation $\mathbb{E}[\sigma(w^\top \mathbf{x})\sigma(w^\top \mathbf{x}')]$ corresponds to the ReLU kernel, given by $\frac{1}{2\pi}|\mathbf{x}||\mathbf{x}'|\,[\sin(\theta) + (\pi - \theta)\cos(\theta)]$.

Thus in that case, the complete NTK at the limit (keeping the vanishing first term) is:

$$K_{\text{NTK}}(\mathbf{x}, \mathbf{x}') = \frac{d_{out}}{2\pi} \left[\frac{2}{h} \cdot \mathbf{x}^\top \mathbf{x}' \cdot (\pi - \theta) + |\mathbf{x}||\mathbf{x}'|[\sin\theta + (\pi - \theta)\cos\theta] \right], \qquad \theta = \arccos\left(\frac{\mathbf{x}^\top \mathbf{x}'}{|\mathbf{x}||\mathbf{x}'|}\right)$$

This illustrates a fundamental result of NTK theory: if a sufficiently wide neural network is initialized appropriately and trained with gradient descent on the square loss, it becomes mathematically equivalent to kernel ridge regression with a specific kernel determined by the network architecture and activation function.
I was not able to reproduce this asymptotic behavior with the previously described framework, maybe because of numerical instabilities, neglecting the biases, or implementation/computation errors. Using *PyTorch*, the convergence was improved, but still not perfect[5].

Regression on waiting times in an amusement park.
Using the dataset described in Section 1.4.1, I performed some architectural experiments with the modularyNN framework and compared them with similar architectures implemented in PyTorch. The results are summarized in the following table, with R^2 beeing the coefficient of determination.
It measures the proportion of variance in the target variable that is predictable from the input features defined as: $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$, with y_i the true target values, $\hat{y}_i$ the predicted values, and $\bar{y}$ the mean of the target values.

Model	**Parameters**	**Training (framework)**	**R^2**
Simple_Léger (1 hidden layer, 32 neurons)	∼3k	1000 iter. (modularyNN)	0.5186
Profond_Large (4 layers: 256→128→64→32)	∼50k	1000 iter. (modularyNN)	0.5253
Dropout_Robuste (3 layers + dropout)	∼20k	1000 iter. (modularyNN)	0.3559
Tanh_Centré (2 layers, Tanh activation)	∼15k	1000 iter. (modularyNN)	0.4871
Apprentissage_Rapide (high LR: 0.01)	∼15k	1000 iter. (modularyNN)	0.5280
Sans_Régularisation (no L2)	∼15k	1000 iter. (modularyNN)	0.5114
Étroit_Profond (5 layers: 64→64→32→16→8)	∼8k	1000 iter. (modularyNN)	0.5201
Ensemble_Ready (96→48→24)	∼12k	1000 iter. (modularyNN)	0.5359
Simple (1 hidden layer, 32 neurons)	∼3k	1 epoch (PyTorch)	0.5249
Deep (4 layers: 256→128→64→32)	∼50k	1 epoch (PyTorch)	0.5423
Dropout (3 layers + dropout)	∼20k	1 epoch (PyTorch)	0.4701
LeakyReLU (2 layers, LeakyReLU)	∼15k	1 epoch (PyTorch)	0.5111

Table 6: Comparison of neural network architectures for waiting time prediction

The experimental results demonstrate that both frameworks achieve comparable performance (without the time factor) on this regression task, with R^2 scores having similar ranges. The PyTorch implementation shows slightly more stable training, particularly for the dropout-regularized model. The best performing architecture in both frameworks is the deep model with 4 hidden layers, achieving R^2 scores of 0.5253 (modularyNN) and 0.5423 (PyTorch).

[5]See `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/00_ntk_limit.ipynb` for details.

2 State of the art

I focused on a topic suggested by Prof. T. Suzuki, that consisted in **studying the propagation of chaos for underdamped mean field langevin dynamics**. I tried to adapt the existing technique to the underdamped case, using a **particle approximation method**. I wanted to give a particle approximation error for the underdampled mean field Langevin dynamics.

I started from the following papers: *Propagation of Chaos for Mean-Field Langevin Dynamics and its Application to Model Ensemble (2025)* [12] and *Mean-field underdamped Langevin dynamics and its spacetime discretization (2023)* [6].

Paper 1: "Propagation of Chaos for Mean-Field Langevin Dynamics and its Application to Model Ensemble" (2025) [12]
This paper establishes theoretical foundations for understanding how large ensembles of machine learning models behave when trained using noisy optimization methods. The authors prove that as the number of models in an ensemble grows to infinity, each individual model's behavior becomes independent of the others (propagation of chaos), despite being trained on the same data.
This result provides rigorous justification for ensemble methods and demonstrates that the collective behavior can be approximated by studying a single representative model following mean-field dynamics.

Paper 2: "Mean-field underdamped Langevin dynamics and its spacetime discretization" (2023) [6]
This paper focuses on the mathematical analysis of underdamped Langevin dynamics in the mean-field regime, where momentum effects are explicitly modeled rather than approximated away. The authors develop a comprehensive theory for both the continuous-time dynamics and their discrete approximations, establishing convergence rates and stability conditions. Unlike overdamped variants that ignore inertial effects, the underdamped formulation captures momentum-based acceleration, which is crucial for understanding modern optimization algorithms like momentum-based SGD. The work provides both theoretical convergence guarantees and practical discretization schemes, with applications to sampling from complex distributions and neural network training dynamics.

Research Objective:
My goal was to extend the propagation of chaos results from the first paper to the underdamped setting studied in the second paper. Specifically, I aimed to prove that **underdamped mean-field Langevin dynamics also exhibits propagation of chaos**, and to derive particle approximation error bounds that quantify how well a finite ensemble approximates the infinite-particle limit. This extension is important because many practical optimization algorithms (like momentum SGD) have underdamped characteristics, and understanding their ensemble behavior requires the more complete underdamped analysis rather than the simpler overdamped approximation.

This state-of-the-art part of the report aims at presenting the necessary background on the **Langevin dynamics**, a concept coming from statistical physics. It is a stochastic process that describes the evolution of a system of particles under the influence of both a potential and stochastic forces. It can be studied at the particle level (Langevin equation) or at the distribution level (Fokker-Planck equation), by sudying stochastic differential equations.
A brief presentation of the **mean-field regime** is given, which is a mathematical framework that allows to study the behavior of large systems of interacting "particles" (neural network weights in our case) by simplifying the modelisation of interactions.
Finally, we present the **mean-field Langevin dynamics**, which is a stochastic process that describes the evolution of a system of particles under the influence of both deterministic and stochastic forces, that corresponds to training a neural network through a stochastic gradient descent method with noise.

2.1 Langevin Dynamics

The **Langevin equation** describing a **stochastic process** introduced by Paul Langevin in 1908, describes the dynamics of a particle under both deterministic and stochastic forces, that models the trade-off between inertia, friction (dissipation) and random thermal kicks (fluctuations). It is often written, in the **underdampling dynamics (taking into account the inertia)** as:

$$m\frac{d\mathbf{v}}{dt} = -\gamma\mathbf{v} - \nabla U(\mathbf{x}) + \boldsymbol{\eta}(t),$$

where m, x and v are the particle's mass position and speed, γ is a **damping/friction coefficient**, U is the potential function (energy or loss), with $\nabla U(x)$ beeing the **drift term** driving the system toward lower potential and $\boldsymbol{\eta}(t)$ is a stochastic force modeled as Gaussian white noise with zero mean and covariance

$$\langle \eta_i(t)\eta_j(t')\rangle = 2\gamma k_{\mathrm{B}} T\delta_{ij}\delta(t-t'),$$

where k_{B} is the Boltzmann constant and T the temperature.
To express this equation as a **stochastic differential equation** (SDE), we rewrite the second-order Langevin equation as a system of first-order SDEs:

$$d\mathbf{x}_t = \mathbf{v}_t\,dt, \qquad d\mathbf{v}_t = \left(-\frac{\gamma}{m}\mathbf{v}_t - \frac{1}{m}\nabla U(\mathbf{x}_t)\right)dt + \sqrt{\frac{2\gamma k_{\mathrm{B}}T}{m^2}}\,d\mathbf{W}_t$$

where $\mathbf{W}_t$ denotes a standard **Brownian motion** in $\mathbb{R}^d$. This SDE system captures the same physical dynamics as the original Langevin equation, incorporating inertia, friction, deterministic drift, and stochastic thermal noise.

The particular case of the **overdamped Langevin dynamics** arises when the effects of inertia become negligible compared to friction. Starting from the previous underdamped Langevin equation, we consider the regime where the mass $m \to 0$ or the friction γ is very large. In this limit, the velocity quickly relaxes to a quasi-steady state where inertial effects vanish, so the quantity $\frac{d\mathbf{v}}{dt} \approx 0$. We obtain a first-order stochastic differential equation: $d\mathbf{x}_t = -\frac{1}{\gamma}\nabla U(\mathbf{x}_t)\,dt + \sqrt{\frac{2k_{\mathrm{B}}T}{\gamma}}\,d\mathbf{W}_t$. By introducing the inverse temperature $\beta = \frac{1}{k_{\mathrm{B}}T}$, and absorbing constants into a time-rescaled version, we arrive at the standard form: $dx_t = -\nabla U(x_t)\,dt + \sqrt{2\beta^{-1}}\,dW_t$. This overdamped approximation is widely used in sampling and optimization contexts where inertia plays a negligible role.

In practice, Langevin dynamics are often implemented via there **Euler–Maruyama discretization** (See `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/01_langevin_dyn_fokker.ipynb` for implementation details).:

$$x_{k+1} = x_k - \eta\nabla U(x_k) + \sqrt{2\eta\beta^{-1}}\,\xi_k$$

with $\eta > 0$ is the step size (analogous to a learning rate) and $\xi_k \sim \mathcal{N}(0, I)$ a gaussian noise sampled independently at each step.

Langevin dynamics can be interpreted as a **noisy gradient descent** (when $\beta \to \infty$, the dynamics becomes purely gradient descent). While the gradient term encourages descent toward local minima, the noise helps escape shallow or narrow minima, enabling exploration of the state space.

The **Langevin dynamics** are thus a **stochastic process** used to **sample from a target probability distribution** or to describe the motion of particles under the influence of both deterministic and random forces.

The time evolution of the probability density $p(x,t)$ associated with this **Langevin process** is governed by the **Fokker–Planck equation** (or forward Kolmogorov equation):

$$\frac{\partial p(x,v,t)}{\partial t} = -\nabla_x \cdot (v\, p(x,v,t)) + \nabla_v \cdot (\nabla U(x)\, p(x,v,t) + \gamma v\, p(x,v,t)) + \gamma\beta^{-1}\nabla_v^2 p(x,v,t),$$

In the **overdamped limit**, where the velocity variable is eliminated, this simplifies to:

$$\frac{\partial p(x,t)}{\partial t} = \nabla_x \cdot \left(\nabla U(x)\, p(x,t) + \beta^{-1}\nabla_x p(x,t)\right).$$

Under suitable regularity conditions[6], the process admits a unique *stationary distribution* $\pi(x)$ (i.e. $\frac{\partial \pi(x,t)}{\partial t} = 0$) [7].

In the **underdamped regime**, the stationary distribution over position and velocity is the **Gibbs distribution**:

$$\pi(x,v) = \frac{1}{Z'} \exp\left(-\beta\left(\frac{1}{2}\|v\|^2 + U(x)\right)\right),$$

where Z' normalizes the joint density.
In the **overdamped regime**, where velocity is eliminated due to strong damping, the stationary distribution reduces to the marginal over position:

$$\pi(x) = \frac{1}{Z} e^{-\beta U(x)}, \quad \text{where} \quad Z = \int_{\mathbb{R}^d} e^{-\beta U(x)}\, dx.$$

In the context of **Bayesian inference**, the goal is often to sample from **a posteriori distribution** of the form:

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)\, p(\theta),$$

where $\theta \in \mathbb{R}^d$ are the model parameters, $\mathcal{D}$ is observed data, $p(\theta)$ is the **prior distribution**, and $p(\mathcal{D} \mid \theta)$ is the likelihood. Taking the negative log of this posterior leads to the potential function:

$$U(\theta) = -\log p(\mathcal{D} \mid \theta) - \log p(\theta),$$

up to an additive constant. Langevin dynamics applied to this potential function $U(\theta)$ therefore leads to samples from the Bayesian posterior distribution.
This principle is the foundation of algorithms such as *Stochastic Gradient Langevin Dynamics (SGLD)*, which scale Bayesian sampling to high-dimensional models by using stochastic gradients computed from minibatches of data.

Langevin dynamics offers a theoretically grounded approach to sampling from complex distributions, blending local optimization through gradients with global exploration via injected noise. Its roots in stochastic differential equations and connections to both thermodynamics and Bayesian inference make it a powerful tool in statistical physics and modern machine learning.
For a recent theoretical analysis of convergence guarantees for Langevin Monte Carlo under general functional inequalities, see *Analysis of Langevin Monte Carlo from Poincaré to Log-Sobolev* [3].

[6] Typical conditions include:

- $U(x)$ is differentiable and Lipschitz continuous,
- $U(x) \to \infty$ as $\|x\| \to \infty$ (coercivity),
- $e^{-\beta U(x)}$ is integrable over $\mathbb{R}^d$,
- the dynamics is non-degenerate and ergodic.

[7] See `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/01_langevin_dyn_fokker.ipynb` for implementation details.

2.2 Mean-Field regime

This notion is inspired by statistical physics, where **the behavior of a large number of interacting particles is approximated by the behavior of a single representative particle in a mean-field potential**. In the context of neural networks, a "particle" typically refers to a neuron or a parameter in a large model.

The mean-field regime in neural networks was formalized in the theoretical analysis of over-parameterized models, where the network width tends to infinity. In this setting, the dynamics of training (e.g., via gradient descent) can be described by a deterministic limit.
A foundational contribution in this direction is *"Exponential expressivity in deep neural networks through transient chaos"* by Poole et al. [13], which analyzes signal propagation in deep random networks using mean-field approximations. This work laid the groundwork for understanding how information flows in deep architectures. Later, the work *"Mean Field Theory of Deep Neural Networks: A Perspective"* by Mei, Montanari, and Nguyen [11] developed a rigorous mean-field theory for the training dynamics of neural networks using tools from interacting particle systems. They showed that, in the infinite-width limit, the evolution of parameters follows a partial differential equation (PDE), allowing for precise characterization of the training process.

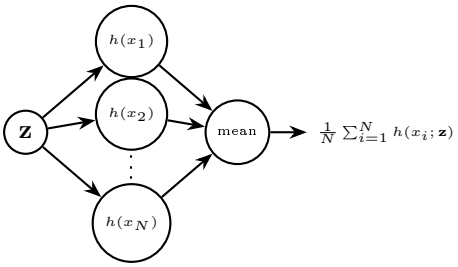
These developments help bridge statistical physics and machine learning, providing a tractable mathematical framework to study large neural networks.

Two-Layer Neural Networks as Particle Systems

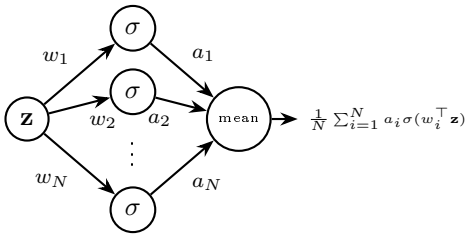
Consider a two-layer neural network with N neurons in the hidden layer which outputs a function of the form:

$$f_N(\theta; \mathbf{z}) = \frac{1}{N} \sum_{i=1}^{N} h(x_i; \mathbf{z})$$

where $\theta = (x_1, \ldots, x_N)$ denotes the collection of all neuron parameters (weights and biases), $x_i \in \mathbb{R}^d$ is the parameter for the i-th neuron, and $\mathbf{z} \in \mathbb{R}^p$ is the input. $h(x_i; \mathbf{z})$ is the output of a single neuron with parameters x_i acting on input $\mathbf{z}$ (e.g., $h(x_i; \mathbf{z}) = a_i \sigma(w_i^\top \mathbf{z})$ for weights w_i and output scale a_i with $x_i = (w_i, a_i)$). The averaging over N neurons corresponds to an empirical mean over the neuron parameters.



Generic case with h as output function.



Mean-aggregated neural network with ReLU activations, parameterized by $x_i = (w_i, a_i)$, where w_i are weight vectors and a_i are post-activation scaling factors.

Figure 12: Two-layer neural network with N hidden neurons.

Define the **empirical distribution of neuron parameters** as $\rho^N := \frac{1}{N} \sum_{i=1}^{N} \delta_{x_i}$, where δ_{x_i} is the Dirac delta measure centered at x_i. Then the network output can be rewritten as:

$$f_N(\theta; \mathbf{z}) = \mathbb{E}_{X \sim \rho^N}[h(X; \mathbf{z})] = \int_{\mathbb{R}^d} h(x; \mathbf{z})\, d\rho^N(x)$$

This reformulation expresses the network output as an **expectation over the empirical distribution**, paving the way for analysis in the infinite-width limit.

The Mean-Field Limit

The mean-field limit describes the behavior of the network as the number of neurons $N \to \infty$; the empirical measure ρ^N converges to a limiting distribution μ over parameter space. This leads to the *mean-field neural network (MFNN)*, under appropriate conditions [8]:

$$f_\infty(\mu; \mathbf{z}) = \mathbb{E}_{X \sim \mu}[h(X; \mathbf{z})]$$

This mean-field limit allows us to represent the neural network output using a continuous distribution over parameters rather than a finite sum. Hence, the model is now parameterized by a probability distribution μ instead of a finite-dimensional parameter vector θ.

Training in the Mean-Field Regime

In the mean-field regime, training the neural network amounts to optimizing a loss functional over the **space of probability measures** $\mathcal{P}(\mathbb{R}^d)$. Let $\mathcal{L}(\mu)$ denote the **population risk** (expected loss over the true data distribution) defined as:

$$\mathcal{L}(\mu) = \mathbb{E}_{(\mathbf{z},y)\sim\mathcal{D}}\left[\ell(f_\infty(\mu; \mathbf{z}), y)\right],$$

where ℓ is a loss function (e.g., squared loss or cross-entropy), and $\mathcal{D}$ is the data (input, output) distribution.

This is a **non-convex functional optimization problem in the space of measures**. To address this, we study the dynamics of a distribution μ_t evolving over a fictitious time variable $t \geq 0$, representing the progression of training under infinitesimal learning rate (i.e., the gradient flow limit of gradient descent). The goal is to minimize $\mathcal{L}(\mu)$ over time by following the steepest descent in the geometry induced by the **Wasserstein-2 metric**.

Under suitable regularity assumptions (detailed below), the evolution of μ_t is governed by the following **continuity equation for mass transport**: $\partial_t \mu_t + \nabla \cdot (\mu_t \nabla V[\mu_t]) = 0$, where $V[\mu] = \nabla\left(\frac{\delta \mathcal{L}}{\delta \mu}\right)$ is derived from the **functional derivative** of $\mathcal{L}$ with respect to μ and is the velocity field. This PDE defines a **Wasserstein gradient flow** of the risk functional $\mathcal{L}(\mu)$, describing how the distribution μ_t evolves to minimize the loss.

Suitable assumptions:

- The neural network has a mean-field structure: the output is expressed as $f_\infty(\mu; \mathbf{z}) = \int \sigma(\mathbf{z}; \theta)\, d\mu(\theta)$, where σ is a smooth activation function.
- The loss function $\ell(f, y)$ is convex and differentiable in f, and the population risk $\mathcal{L}(\mu)$ is **Fréchet differentiable** in the **Wasserstein metric space**
- The measure μ lies in $\mathcal{P}_2(\mathbb{R}^d)$, ensuring finite second moments so the Wasserstein metric is well-defined.
- Gradient descent is approximated by continuous-time dynamics with infinitesimally small learning rate, i.e., the gradient flow approximation.

Recent works have established several theoretical properties of the mean-field regime:

- **Global Convergence:** When the loss function is convex and smooth, the gradient flow of the distribution μ_t converges globally to a minimizer [4, 11].
- **Adaptivity:** The model dynamically adapts its complexity over time as it evolves in distributional space [17].
- **Propagation of Chaos:** For a finite number of neurons N, the parameter evolution under gradient descent approximates the mean-field limit as $N \to \infty$ [11].

This mean-field approach connects neural networks to interacting particle systems and PDEs, providing tools from **statistical physics** and **optimal transport** to analyze training dynamics and generalization.
This regime is particularly useful for understanding the behavior of large neural networks, where the number of neurons N is large, and the empirical distribution of parameters converges to a limiting distribution μ as $N \to \infty$.

[8] The convergence $\rho^N \to \mu$ typically requires:

- The neuron parameters $\{x_i\}_{i=1}^N$ are independent and identically distributed (i.i.d.) at initialization.
- A uniform bound on the moments of the parameters, e.g., $\sup_N \frac{1}{N}\sum_{i=1}^N \|x_i\|^p < \infty$ for some $p \geq 1$, to ensure tightness and convergence in the **space of probability measures** (e.g., under the **Wasserstein-2 metric**)
- The function $h(x; \mathbf{z})$ is continuous (or Lipschitz) in x for each fixed $\mathbf{z}$, so that expectations vary continuously with the distribution.

However, the main restriction is that the neural networks studied are two-layer networks, which is a limitation that comes from the mean-field regime itself by supposing that the pre-activations (inputs to activation functions) across neurons become independent and identically distributed (i.i.d.) as the network width grows, which is not the case for deeper networks.

Some extensions to deeper networks have been proposed, as in [2], where the author introduces Neural Hilbert Ladders to study the mean-field limit of deep neural networks, but this is still an active area of research.

2.3 Mean-field Langevin dynamics (MFLD)

Mean-field Langevin dynamics is a stochastic dynamical system modeling the evolution of N interacting particles $\{X_t^i\}_{i=1}^N$, where each particle is influenced not only by stochastic forces and potential gradients but also by the empirical distribution of the whole system (i.e., a mean-field interaction) and follows a **mean-field Langevin stochastic differential equation (SDE) with interactions**:

$$dX_t^i = V_t^i dt, \qquad dV_t^i = -\gamma V_t^i dt - \nabla V(X_t^i)dt - \frac{1}{N}\sum_{j=1}^{N} \nabla W(X_t^i - X_t^j)dt + \sqrt{2\gamma\beta^{-1}}dB_t^i,$$

where V is an external confining potential, W is a symmetric interaction potential, $\gamma > 0$ is the friction coefficient, $\beta > 0$ is the inverse temperature parameter, and $\{B_t^i\}_{i=1}^N$ are independent standard $\mathbb{R}^d$-valued **Brownian motion**s.
It corresponds to the **mean-field potential** defined as: $U(X_t^i) = V(X_t^i) + \frac{1}{N}\sum_{j=1}^N W(X_t^i - X_t^j)$.

If we introduce, as previously, the time-dependent probability density μ_t over phase space (i.e., the law of a typical particle in the mean-field limit), it allows us to rewrite the mean-field potential in the continuous formulation as:

$$\Phi(x) = V(x) + \int_{\mathbb{R}^d} W(x-y)\,\rho(t,y)\,dy,$$

where $\rho(t,x) = \int_{\mathbb{R}^d} \mu_t(x,v)\,dv$ denotes the spatial marginal of μ_t.

In the limit as $N \to \infty$, the empirical measure of the particle system converges (in law) to a deterministic measure governed by a nonlinear **Vlasov–Fokker–Planck (VFP)** equation. Let $\rho(t,x,v)$ denote the probability density of particles in **phase space**. Then ρ satisfies:

$$\partial_t \rho + v \cdot \nabla_x \rho = \nabla_v \cdot ((\gamma v + \nabla V(x) + (\nabla W * \rho)(x))\,\rho) + \gamma\beta^{-1}\Delta_v \rho$$

where the mean-field interaction (convolution term) is defined by

$$(\nabla W * \rho)(x) = \int_{\mathbb{R}^d \times \mathbb{R}^d} \nabla W(x-y)\rho(t,y,w)dydw.$$

This limit plays a central role in understanding the macroscopic behavior of high-dimensional interacting particle systems, with applications in machine learning, statistical physics, and Bayesian sampling [5].

The concept of **Propagation of Chaos**, originally introduced by Kac [8] and later formalized by Sznitman [15], describes the phenomenon where, as the number of particles $N \to \infty$, the joint law of a system of interacting particles converges to a product of independent, identically distributed (i.i.d.) marginal laws.
That is, although the finite-N system exhibits dependencies due to interactions, these dependencies vanish asymptotically. Formally, if $\mu_t^{(N)} \in \mathcal{P}(\mathbb{R}^{dN})$ denotes the joint law of $(X_t^1, \ldots, X_t^N)$, we say the system exhibits propagation of chaos if for all $k \in \mathbb{N}$,

$$\lim_{N\to\infty} \mu_t^{(N,k)} = \mu_t^{\otimes k}$$

where $\mu_t^{(N,k)}$ is the k-marginal of $\mu_t^{(N)}$, μ_t is the solution of the limiting McKean–Vlasov equation and $\mu_t^{\otimes k}$ is the product measure of μ_t with itself k times ($\mu_t^{\otimes k} = \mu_t(dx_1)\cdots\mu_t(dx_k)$).

One major application of this is **model ensembles**, where models can be viewed as particles in a dynamical system. When training multiple models with stochastic gradient Langevin dynamics (SGLD), their evolution is analogous to Langevin particles with weak interactions (e.g., via regularization or shared priors).

The mean-field framework provides a theoretical justification for approximating the behavior of the entire ensemble by studying a single representative model under the mean-field dynamics.
The propagation of chaos implies that for large ensembles, models behave independently in the limit, which simplifies both analysis and algorithmic design. This has important implications for uncertainty quantification, generalization, and robustness in deep ensemble methods (see `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/03_MFNN.ipynb` for some experiments).

2.4 Propagation of Chaos for Mean Field overdamped Langevin dynamics

Lets focus on the uniform-in-time propagation of chaos (PoC) framework established for **overdamped** mean-field Langevin dynamics in [12]. The following theorem establishes a PoC property for overdamped MFLD, providing improved quantitative bounds on the convergence of the empirical measure to the mean-field limit, with a particle approximation term that does not exhibit exponential dependence on the regularization strength.

We consider the optimization of an **entropy**-regularized convex functional:

$$\min_{\mu\in\mathcal{P}_2(\mathbb{R}^d)} \{\mathcal{L}(\mu) := F_0(\mu) + \mathbb{E}_{X\sim\mu}[r(X)] + \lambda \mathrm{Ent}(\mu)\}$$

where $r : \mathbb{R}^d \to \mathbb{R}$ is $r(x) = \lambda'\|x\|_2^2$ with $\lambda' > 0$ (λ'-strongly convex). We set $F(\mu) = F_0(\mu) + \mathbb{E}_\mu[r(X)]$. A typical example of F_0 is an empirical risk of the two-layer mean-field neural network. [12] assumes that the solution $\mu^* \in \mathcal{P}_2(\mathbb{R}^d)$ exists and satisfies the optimality condition, which is the case under the **Log-Sobolev inequality** in the underdamped case: $\mu^* \propto \exp\left(-\frac{1}{\lambda}\frac{\delta F(\mu^*)}{\delta \mu}\right)$.

Assumptions (overdamped MFLD). From [12]:

1. **Assumption: Regularity of the Empirical Risk $F_0(\mu)$.**
   There exist constants $C_1, C_2 > 0$ such that for all $\mu \in \mathcal{P}_2(\mathbb{R}^d)$ and all $x \in \mathbb{R}^d$, $\left\|\nabla \frac{\delta F_0}{\delta \mu}(\mu)(x)\right\| \le C_1$ and for any $\mu, \mu' \in \mathcal{P}_2(\mathbb{R}^d)$ and $x, x' \in \mathbb{R}^d$, $\left\|\nabla \frac{\delta F_0}{\delta \mu}(\mu)(x) - \nabla \frac{\delta F_0}{\delta \mu}(\mu')(x')\right\| \le C_2\big(W_2(\mu,\mu') + \|x - x'\|_2\big)$.
   This uniform bound controls the magnitude of the velocity field by bounding the spatial gradient of the **functional derivative**. The joint Lipschitz condition in (μ, x) with respect to the **Wasserstein-2 metric** and $\|\cdot\|_2$ ensures stability and well-posedness of the gradient-flow dynamics.

2. **Assumption: Uniform directional LSI.**
   There exists a constant $\alpha > 0$ such that for any $x \in \mathbb{R}^{dN}$ and any $i \in \{1, \dots, N\}$, the **conditional Gibbs distribution** $\nu_{i|-i}(\cdot \mid x^{-i})$ [9] satisfies the **Log-Sobolev inequality** with constant α. In particular, for all $\mu \in \mathcal{P}_2(\mathbb{R}^d)$ absolutely continuous w.r.t. $\nu_{i|-i}(\cdot \mid x^{-i})$,

   $$\mathrm{KL}\big(\mu \,\|\, \nu_{i|-i}(\cdot \mid x^{-i})\big) \le \frac{1}{2\alpha}\, \mathrm{FI}\big(\mu|\, \nu_{i|-i}(\cdot \mid x^{-i})\big)$$

   This assumption ensures that the **conditional Gibbs distribution** is not too concentrated around a single point, as the **Kullback-Leibler (KL) divergence** (quantifying the difference between two probability distributions) is bounded by the **Fisher information** (a measure of the amount of information that an observable random variable carries about an unknown parameter upon which the probability depends).

3. **Assumption: F_0 is differentiable and linearly convex.**
   The nonlinearity of F_0 (key to the PoC analysis in mean-field models) motivates the use of its associated **Bregman divergence**. For $\mu, \mu' \in \mathcal{P}_2(\mathbb{R}^d)$, define $B_{F_0}(\mu, \mu') := F_0(\mu) - F_0(\mu') - \left\langle \frac{\delta F_0}{\delta \mu}(\mu'),\, \mu - \mu' \right\rangle$.

4. **Assumption: Bregman curvature control under single-particle replacement.**
   There exists a constant $B > 0$ such that for any $x = (x^1, \dots, x^N) \in \mathbb{R}^{dN}$, any $z \in \mathbb{R}^d$, and any $i \in \{1, \dots, N\}$, $B_{F_0}\big(\rho_{x^{-i}\cup z},\, \rho_x\big) \le \frac{B}{N^2}$. where $\rho_x := \frac{1}{N}\sum_{j=1}^N \delta_{x^j}$ and $\rho_{x^{-i}\cup z}$ is the empirical measure obtained by replacing x^i by z in x.

[9] where $x^{-i} := (x^1, \dots, x^{i-1}, x^{i+1}, \dots, x^N)$ collects all coordinates except the i-th

Theorem (Continuous-time): Propagation of Chaos for overdamped MFLD. ([12])
Suppose Assumptions 1, 2, 3, and 4 hold, then, the **overdamped** MFLD in continuous-time satisfies, μ^* beeing the optimal distribution,

$$\frac{1}{N}\mathcal{L}^{(N)}(\mu_t^{(N)}) - \mathcal{L}(\mu^*) \leq \frac{B}{N} + \exp(-2\alpha\lambda t)\,\Delta_0^{(N)}$$

with $\Delta_0^{(N)} := \frac{1}{N}\mathcal{L}^{(N)}(\mu_0^{(N)}) - \mathcal{L}(\mu^*)$ represents the initial discrepancy term.

This theorem uses the following **Lemma (Defective LSI)** [12] for his proof; under assumptions 2, 3, and 4, for any $\mu^{(N)} \in \mathcal{P}_2(\mathbb{R}^{dN})$:

$$\frac{\lambda}{N}\mathrm{KL}(\mu^{(N)}||\mu_*^{\otimes N}) + \mathbb{E}_{\mathbf{X}\sim\mu^{(N)}}[B_{F_0}(\rho_{\mathbf{X}}, \mu_*)] = \frac{1}{N}\mathcal{L}^{(N)}(\mu^{(N)}) - \mathcal{L}(\mu^*) \leq \frac{B}{N} + \frac{\lambda}{2\alpha N}\mathrm{FI}(\mu^{(N)}||\mu_*^{(N)})$$

Then, for the distribution $\mu_t^{(N)} = \mathrm{law}(\mathbf{X}_t)$ following the **overdamped MFLD** (N-tuple of SDEs) given by:

$$dX_t^i = -\nabla\left(\frac{\delta F(\rho_{\mathbf{X}_t})}{\delta\mu}\right)(X_t^i)dt + \sqrt{2\lambda}dW_t^i$$

where $\{W_t^i\}_{t\geq 0}$ ($i \in \{1,\ldots,N\}$) are independent standard Brownian motions, it satisfies the following Fokker-Planck equation:

$$\frac{\partial\mu_t^{(N)}}{\partial t} = \lambda\nabla\cdot\left(\mu_t^{(N)}\log\left(\frac{d\mu_t^{(N)}}{d\mu_*^{(N)}}\right)\right)$$

Then, using Lemma 1 and a *standard argument of Langevin dynamics* (see [12]), we get:

$$\frac{d}{dt}\left(\mathcal{L}^{(N)}(\mu_t^{(N)}) - N\mathcal{L}(\mu^*) - B\right) = -\lambda^2\mathrm{FI}(\mu_t^{(N)}||\mu_*^{(N)}) \leq -2\alpha\lambda\left(\mathcal{L}^{(N)}(\mu_t^{(N)}) - N\mathcal{L}(\mu^*) - B\right)$$

Using **Grönwall inequality**, applied to this differential inequality, we obtain the convergence bound in 2.4.

From this result, we see that the overdamped MFLD indeed induces the Propagation of Chaos (PoC) regarding KL-divergence. This is a direct consequence of Lemma 1 and the continuous-time dynamics, demonstrating that the particles $(X_i^N)_{i=1}^N \sim \mu_t^{(N)}$ become independent as $t \to \infty$ and $N \to \infty$. Specifically, the following inequality holds:

$$\frac{1}{N}\mathrm{KL}(\mu_t^{(N)}||\mu_*^{\otimes N}) \leq \frac{B}{\lambda N} + \frac{\Delta_0^{(N)}}{\lambda}\exp(-2\alpha\lambda t)$$

This result provides a quantitative bound on the propagation of chaos for the overdamped MFLD. The left-hand side of Inequality, $\frac{1}{N}\mathrm{KL}(\mu_t^{(N)}||\mu_*^{\otimes N})$, measures the deviation of the N-particle system's joint distribution from a product measure of independent particles, each distributed according to the optimal mean-field distribution μ_*. The inequality demonstrates that this deviation is bounded by two key terms:

1. **Finite-particle approximation error**: $\frac{B}{\lambda N}$ - This term decays with the number of particles N. As $N \to \infty$, this term vanishes, indicating that the system approaches the mean-field limit. We note that the particle approximation term B/N is independent of LSI-constants.

2. **Time-dependent convergence**: $\frac{\Delta_0^{(N)}}{\lambda}\exp(-2\alpha\lambda t)$ - This term decays exponentially with time t. It captures the initial discrepancy from the optimal state and shows that the system converges to equilibrium over time.

The combination of these two terms implies that as both $t \to \infty$ and $N \to \infty$, the KL-divergence goes to zero. This signifies that the particles become independent, and their empirical distribution converges to the optimal mean-field distribution μ_*, which is the core principle of propagation of chaos.
This theorem (2025) improves previous bounds for the overdamped MFLD by removing the LSI-constants dependence.
A similar result holds for the discrete-time version of the theorem, which is given in [12]:
Consider the discrete-time Langevin equation from the finite-particle setting

$$X_{k+1}^i = X_k^i - \eta\nabla\left(\frac{\delta F(\rho_{X_k})}{\delta\mu}\right)(X_k^i) + \sqrt{2\lambda\eta}\xi_k^i$$

where $\eta > 0$ is the step size, $\lambda > 0$ is the regularization parameter, and $\{\xi_k^i\}_{k\geq 0}$ are independent standard Gaussian random variables for each particle $i \in \{1, \ldots, N\}$.

Theorem (Discrete-time): Propagation of Chaos for overdamped MFLD. ([12])
Suppose Assumptions 1, 2, 3, and 4 hold, and let $\eta\lambda' < 1/2$. Then, the **overdamped** MFLD in discrete-time satisfies:

$$\frac{1}{N}\mathcal{L}^{(N)}(\mu_k^{(N)}) - \mathcal{L}(\mu_*) \leq \frac{B}{N} + \frac{\delta_\eta}{\alpha\lambda} + \exp(-\alpha\lambda\eta k)\Delta_0^{(N)}$$

where $\delta_\eta := 8\eta(C_2^2 + \lambda r^2)(\eta C_1^2 + \lambda d) + 32\eta^2\lambda r^2(C_2^2 + \lambda r^2)\left(\mathbb{E}\left[\frac{\|X_0\|_2^2}{N}\right] + \frac{1}{\lambda'}\left(\frac{C_1^2}{4\lambda'} + \lambda d\right)\right)$

The proof uses the same arguments as the continuous-time version, and using a *one step size argument* (see [12] for details). The discrete-time theorem provides similar convergence guarantees as the continuous-time version, with the key differences being:

1. **Step size constraint**: The condition $\eta\lambda' < 1/2$ ensures numerical stability of the discrete-time dynamics.
2. **Discretization error**: The term $\frac{\delta_\eta}{\alpha\lambda}$ captures the additional error introduced by the discrete-time approximation, which depends on the step size η and various problem-dependent constants.
3. **Exponential convergence rate**: The convergence rate $\exp(-\alpha\lambda\eta k)$ is proportional to the step size η, showing that smaller step sizes lead to slower convergence but better approximation accuracy.

This discrete-time result is particularly important for practical implementations, as most numerical algorithms operate in discrete time with finite step sizes. The theorem ensures that the propagation of chaos property is preserved under discretization, providing theoretical justification for using discrete-time Langevin dynamics in practical applications.

3 A particle approximation error for underdamped MFLD

3.1 Purpose and motivation

The purpose of this section is to see what happens in the underdamped case with the convergence rate of $\frac{1}{N}\mathcal{L}^{(N)}(\mu_k^{(N)})$ to $\mathcal{L}(\mu_*)$.

Uunderdamped dynamics, which incorporate momentum terms, often exhibit superior convergence properties compared to their overdamped counterparts, especially in high-dimensional sampling problems and optimization landscapes with complex geometry. The SGDs used in the experiments are mostly underdamped.

The discrepancy between the empirical measure $\mu_t^N = \frac{1}{N}\sum_{i=1}^N \delta_{(X_t^i, V_t^i)}$ and the theoretical mean-field distribution ρ_t defines the **particle approximation error**. This error quantifies how well the finite-particle system approximates the limiting dynamics and is central to both theoretical convergence guarantees and the design of efficient algorithms.
The study of particle approximation errors in mean-field Langevin dynamics represents a crucial step in bridging the theoretical foundations of continuous-time stochastic processes with their practical computational implementations.

As I wanted to have both the theory and experiments to work with, I reproduced some of the previously known results and have published the code on GitHub, which is available at https://github.com/charles-vzf/MFLD-PoC.

3.2 Propagation of Chaos for Mean Field underdamped Langevin dynamics

The goal of this part is to give details on the propagation of chaos for underdamped MFLD, building upon the theorem given in the state-of-the-art section from [12]. Let's start by looking at what's changing in the underdamped case, in term of context, hypotheses and then proof.

Compared to the overdamped setting introduced previously, the underdamped regime **lifts the dynamics to phase space** (x, v) and injects noise only in the velocity coordinate. This leads to different generators, invariant measures, and functional inequalities.

For the distribution $\mu_t^{(N)} = \mathrm{law}(\mathbf{X}_t, \mathbf{V}_t)$ following the **underdamped MFLD** (N-tuple of SDEs for a finite N-particle system) given by:

$$dX_t^i = V_t^i dt \qquad dV_t^i = -\gamma V_t^i dt - \nabla\left(\frac{\delta F(\rho_{\mathbf{x}_t})}{\delta \mu}\right)(X_t^i)dt + \sqrt{2\gamma\lambda}\, dB_t^i \qquad i = 1, \dots, N$$

with friction $\gamma > 0$, and still independent standard Brownian motions $\{B_t^i\}$.

In the mean-field limit, with spatial marginal $\rho(t,x) = \int \mu_t(x,v)\, dv$, the law μ_t solves the Vlasov–Fokker–Planck (VFP) equation

$$\partial_t \mu + v \cdot \nabla_x \mu = \nabla_v \cdot \Big((\gamma v + \nabla_x \tfrac{\delta F(\rho)}{\delta \mu}(x))\, \mu \Big) + \gamma\lambda\, \Delta_v \mu.$$

We deduce the invariant (target) Gibbs measure on phase space and make a similar hypothesis as in the overdamped case:

$$\mu_*(x,v) \propto \exp\Big(-\tfrac{1}{\lambda}\big(\tfrac{1}{2}\|v\|^2 + \tfrac{\delta F(\rho_*)}{\delta \mu}(x)\big)\Big).$$

I will reuse the structural assumptions from the overdamped theorem, with the following underdamped-specific adaptations. I haven't formally prooved it to get the constants but tried to empirically validate it. Some results in the underdamped case already exist but the convergence guaranty could be improved with the new assumptions.

1. **Assumption: Regularity of the empirical risk in position.** Identical to the one in the overdamped case, applied to the position marginals.

2. **Assumption: Hypocoercive functional inequality.** The stationary law μ_* on phase space satisfies a kinetic LSI/Poincaré inequality with constant $\alpha_{\mathrm{kin}} > 0$ arising from:

   - the conditional law in velocity $v \,|\, x$ under μ_* being Gaussian with variance λ, which satisfies a **uniform directional LSI in** v ([12]), independent of x, with constant $1/\lambda$ and
   - an **Log-Sobolev inequality/Poincaré inequality** in x for the positional marginal induced by $\frac{\delta F(\rho_*)}{\delta \mu}(x)$,

- **A defective hypocoercive inequality.** There exists $c_{\mathrm{hyp}}(\gamma) \in (0,1]$ such that, with $\alpha_{\mathrm{kin}} = c_{\mathrm{hyp}}(\gamma)\min\{\alpha_x, 1/\lambda\}$, for any $\mu^{(N)} \ll \mu_*^{(N)}$:
$\frac{\lambda}{N}\,\mathrm{KL}(\mu^{(N)} \,\|\, \mu_*^{\otimes N}) \le \frac{\lambda}{2\alpha_{\mathrm{kin}} N}\,\mathrm{I}_{\mathrm{kin}}(\mu^{(N)} \,\|\, \mu_*^{(N)}) + \frac{1}{N}\,\mathbb{E}_{\mathbf{X}\sim\mu^{(N)}}[B_{F_0}(\rho_{\mathbf{X}}, \mu_*)]$,
where $\mathrm{I}_{\mathrm{kin}}(\mu^{(N)} \,\|\, \mu_*^{(N)}) := \int \|\nabla_{\mathbf{v}} \log \frac{d\mu^{(N)}}{d\mu_*^{(N)}}\|^2 \,\mu^{(N)}$.

3. **Assumption: Differentiability and linear convexity of F_0.** The functional F_0 is differentiable and linearly convex, and its Bregman divergence

$$B_{F_0}(\mu, \mu') := F_0(\mu) - F_0(\mu') - \left\langle \tfrac{\delta F_0}{\delta \mu}(\mu'),\, \mu - \mu' \right\rangle$$

is well-defined for probability measures on $\mathbb{R}^d$ (position space).

4. **Assumption: Single-particle replacement curvature.** There exists $B > 0$ such that for any $x = (x^1, \dots, x^N) \in \mathbb{R}^{dN}$, any $z \in \mathbb{R}^d$, and any index i, with $\rho_x := \frac{1}{N}\sum_{j=1}^N \delta_{x^j}$ and $\rho_{x^{-i}\cup z}$ the empirical measure replacing x^i by z,

$$B_{F_0}\big(\rho_{x^{-i}\cup z},\, \rho_x\big) \le \frac{B}{N^2}.$$

5. **Assumption: Noise/friction parameters.** The friction and temperature parameters satisfy $\gamma > 0$, $\lambda > 0$, and the noise acts only in the velocity coordinate with covariance $2\gamma\lambda I_d\, dt$, consistent with the invariant density $\mu_*(x,v) \propto \exp\big(-\frac{1}{\lambda}(\frac{1}{2}\|v\|^2 + \frac{\delta F(\rho_*)}{\delta \mu}(x))\big)$.

These changes will let us state an underdamped analogue of the defective inequality and the ensuing propagation-of-chaos bound, with rates depending on α, γ, β and the same B/N particle approximation scaling.

Analogy of Theorem by [12] (Continuous-time): Propagation of Chaos for underdamped MFLD.
Assume the analogues of Assumptions 1–5 hold in the kinetic setting, the **underdamped** MFLD in continuous time satisfies:

$$\frac{1}{N}\,\mathcal{L}^{(N)}(\mu_t^{(N)}) - \mathcal{L}(\mu^*) \;\le\; \frac{B}{N} + \exp(-c_{\mathrm{kin}}\, t)\,\Delta_0^{(N)},$$

where $c_{\mathrm{kin}} = c(\alpha_{\mathrm{kin}}, \gamma, \lambda) > 0$ and $\Delta_0^{(N)} := \frac{1}{N}\,\mathcal{L}^{(N)}(\mu_0^{(N)}) - \mathcal{L}(\mu^*)$.

3.3 Experiments

I ran some numerical simulations on mean field dynamics and large neural networks available in the `https://github.com/charles-vzf/MFLD-PoC`.

Weight distribution under mean-field dynamics vs Fokker–Planck.

We simulate a large-particle stochastic dynamics (mean-field regime) and compare the resulting empirical distribution with the solution of the corresponding Fokker–Planck PDE.

Particle scheme (Euler–Maruyama). For step size $\eta > 0$ and i.i.d. $\xi_k^i \sim \mathcal{N}(0, I)$:

$$x_{k+1}^i = x_k^i + \eta\, v_k^i, \qquad v_{k+1}^i = (1 - \gamma\eta)\, v_k^i - \eta\, G(x_k^i; \rho_{x_k}) + \sqrt{2\gamma\lambda\eta}\, \xi_k^i,$$

where $G(x; \rho)$ denotes the positional drift $\nabla_x\big(\frac{\delta F(\rho)}{\delta \mu}\big)(x)$ and ρ_{x_k} is the empirical spatial marginal at step k.

Kinetic solver (finite differences). On a uniform grid $x_i = x_{\min} + i\,\Delta x$, $v_j = v_{\min} + j\,\Delta v$, $t^n = n\,\Delta t$, set $f_{i,j}^n \approx \mu(t^n, x_i, v_j)$ and $a_{i,j}^n := \gamma v_j + G_i^n$ with $G_i^n \approx \partial_x U(x_i)$. Update explicitly by

$$f_{i,j}^{n+1} = f_{i,j}^n - \Delta t \underbrace{\Big[v_j^+ \tfrac{f_{i,j}^n - f_{i-1,j}^n}{\Delta x} + v_j^- \tfrac{f_{i+1,j}^n - f_{i,j}^n}{\Delta x}\Big]}_{\text{upwind in } x} + \Delta t \underbrace{\Big[(a_{i,j}^n)^+ \tfrac{f_{i,j}^n - f_{i,j-1}^n}{\Delta v} + (a_{i,j}^n)^- \tfrac{f_{i,j+1}^n - f_{i,j}^n}{\Delta v}\Big]}_{\text{upwind in } v} + \Delta t\, \gamma\lambda \underbrace{\tfrac{f_{i,j+1}^n - 2f_{i,j}^n + f_{i,j-1}^n}{\Delta v^2}}_{\text{central diffusion in } v},$$

where $b^+ = \max(b, 0)$ and $b^- = \min(b, 0)$. Use no-flux (Neumann) boundary conditions in x and v.

Stability via Courant-Friedrichs-Lewy (CFL). Choose Δt so that

$$\Delta t \;\le\; c \min\Big\{\tfrac{\Delta x}{\max_j |v_j|},\; \tfrac{\Delta v}{\max_{i,j} |a_{i,j}^n|},\; \tfrac{\Delta v^2}{2\gamma\lambda}\Big\}, \quad c \in (0, 1].$$

Below is a plot of the particle distribution under mean-field dynamics and the Fokker–Planck solution. We can see that given an initial position and velocity, the particle distribution is well approximated by the Fokker–Planck solution that evolves over time with a spreading effect due to the noise. (see `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/01_langevin_dyn_fokker.ipynb` for details).

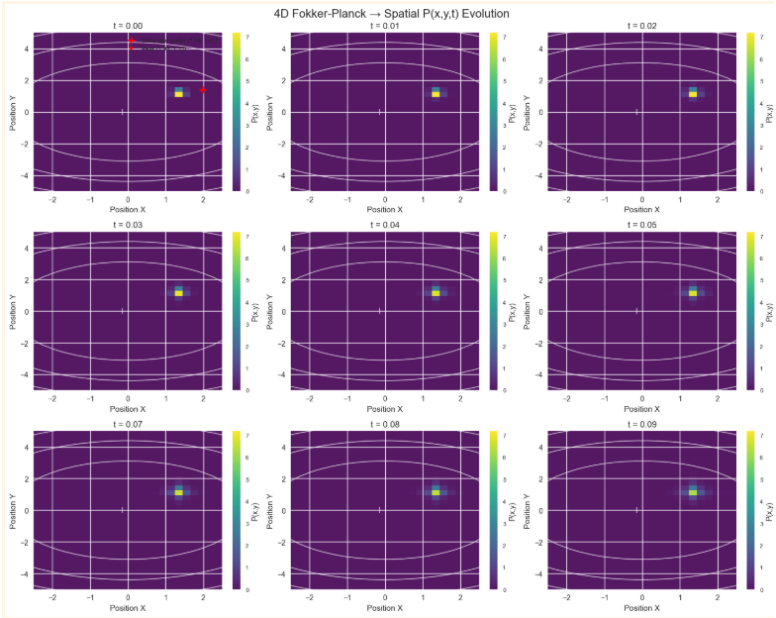


Figure 13: Mean-field particles vs Fokker–Planck: density snapshots and distance over time.

Neural Tangent Kernel (NTK) convergence in the mean-field regime.

Consider a two-layer network with width m: $f_m(x) = \frac{1}{\sqrt{m}} \sum_{i=1}^{m} a_i \, \sigma(w_i^\top x), with (w_i, a_i)$ i.i.d. $\mathcal{N}(0, I) \times \mathcal{N}(0, 1)$, with ReLU or tanh activation. At random initialization, the analytic ReLU NTK on inputs x, x' with angle $\theta = \arccos(\langle x, x' \rangle / (\|x\| \, \|x'\|))$ is

$$K_\infty(x, x') = \underbrace{\frac{\|x\| \, \|x'\|}{2\pi} \big(\sin\theta + (\pi - \theta) \cos\theta \big)}_{\mathrm{NNGP}_{\mathrm{ReLU}}} + \underbrace{\langle x, x' \rangle \, \frac{\pi - \theta}{2\pi}}_{\text{ReLU derivative corr.}} .$$

For a fixed dataset $X = \{x_i\}_{i=1}^n$, the empirical NTK at initialization K_m converges to K_∞ with optimal rate on the Root Mean Square (RMS) norm:

$$\mathbb{E}\Big[\|K_m - K_\infty\|_F^2 \Big] = \mathcal{O}\Big(\frac{n^2}{m}\Big) \quad \Longrightarrow \quad \text{RMS discrepancy}(m) = \frac{\|K_m - K_\infty\|_F}{n} = \mathcal{O}\Big(m^{-1/2}\Big) .$$

In the constant-NTK regime, $\frac{d}{dt}(f_t - y) = -K_\infty (f_t - y) \quad \Rightarrow \quad f_t - y = e^{-\eta K_\infty t}(f_0 - y)$, with rates governed by the spectrum of K_∞. We are able to reproduce the result in `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/02_mean2018.ipynb`.

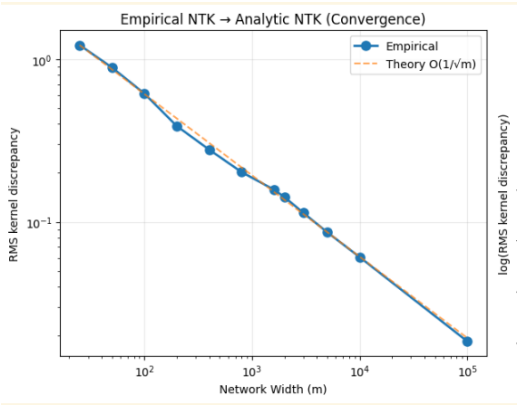


Figure 14: Empirical NTK → analytic NTK: RMS kernel discrepancy vs width. The dashed line shows the theory $\mathcal{O}(m^{-1/2})$.

Training and merging MFNNs.
Let two trained networks have parameters $\theta^{(1)} = \{W_\ell^{(1)}, b_\ell^{(1)}\}_{\ell=1}^L$ and $\theta^{(2)} = \{W_\ell^{(2)}, b_\ell^{(2)}\}_{\ell=1}^L$. We interpolate in weight space with a single mixing coefficient $\alpha \in [0, 1]$, computing the merged parameters as: $\theta(\alpha) = (1 - \alpha)\, \theta^{(1)} + \alpha\, \theta^{(2)}$. The merged predictor is $f_\alpha(x) = f\big(x; \theta(\alpha)\big)$ and predictions use $\hat{y}_\alpha(x) = \arg\max_k \mathrm{softmax}(f_\alpha(x))_k$. This linear mode connectivity often preserves accuracy across α and can improve it near the better model, which is what we observe in the figure below. More details in `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/03_MFNN.ipynb`.

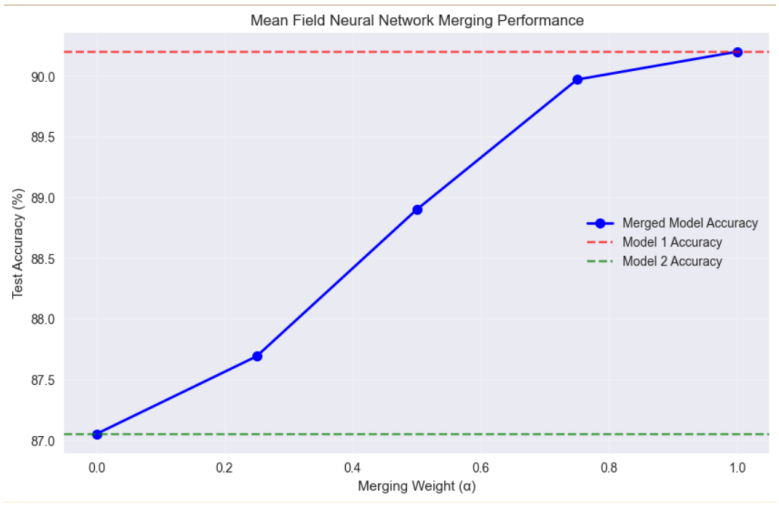


Figure 15: Test accuracy vs merging weight α for the two-model MFNN merge. Dashed lines: individual models.

Specialized ensemble and knowledge distillation.
We train 10 binary specialists $\{f_k\}_{k=0}^{9}$ (one-vs-rest) on MNIST. Each outputs a probability $p_k(x) = \sigma(f_k(x))$. The ensemble produces a 10D score vector $p(x) = [p_0(x), \ldots, p_9(x)]$ and predicts $\hat{y}_{\text{ens}}(x) = \arg\max_{k\in\{0,\ldots,9\}} p_k(x)$. A compact multi-class distribution for training a student is obtained with a softmax over the specialist scores: $q(x) = \text{softmax}(p(x))$. More details in `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/04_MFNN_merging.ipynb`.
A student network with logits (pre-softmax scores) $z_s(x)$ is trained by **knowledge distillation (KD)** with temperature $\tau > 0$ and mixing $\alpha \in [0, 1]$:

$$\mathcal{L}_{\text{KD}} = \alpha\, \text{CE}\big(y,\, z_s(x)\big) + (1-\alpha)\,\tau^2\, \text{KL}\Big(\text{softmax}\big(\tfrac{z_s(x)}{\tau}\big) \,\Big\|\, \text{softmax}\big(\tfrac{q(x)}{\tau}\big)\Big),$$

where $\text{CE}(y, z_s(x)) := -\sum_k y_k \log\big(\text{softmax}(z_s(x))_k\big)$ is the standard cross-entropy (with one-hot labels y). which corresponds to matching softened teacher targets while retaining supervision from hard labels. At inference the student uses $\hat{y}_{\text{stud}}(x) = \arg\max_k \text{softmax}(z_s(x))_k$.

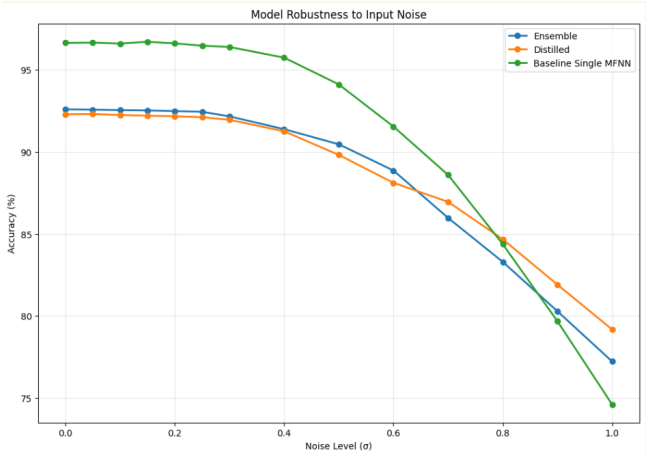


Figure 16: Noise robustness: test accuracy of the merged/student models under additive Gaussian noise $x \mapsto x + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, as a function of σ (or SNR).

Underdamped vs overdamped dynamics.
We reproduce a PoC comparing underdamped and overdamped Langevin dynamics, highlighting the speed–stability trade-off. We use the common temperature notation $\beta^{-1} = \lambda > 0$ (same λ as in the entropy weight). See `https://github.com/charles-vzf/MFLD-PoC/blob/main/notebooks/PoC.ipynb` for details.

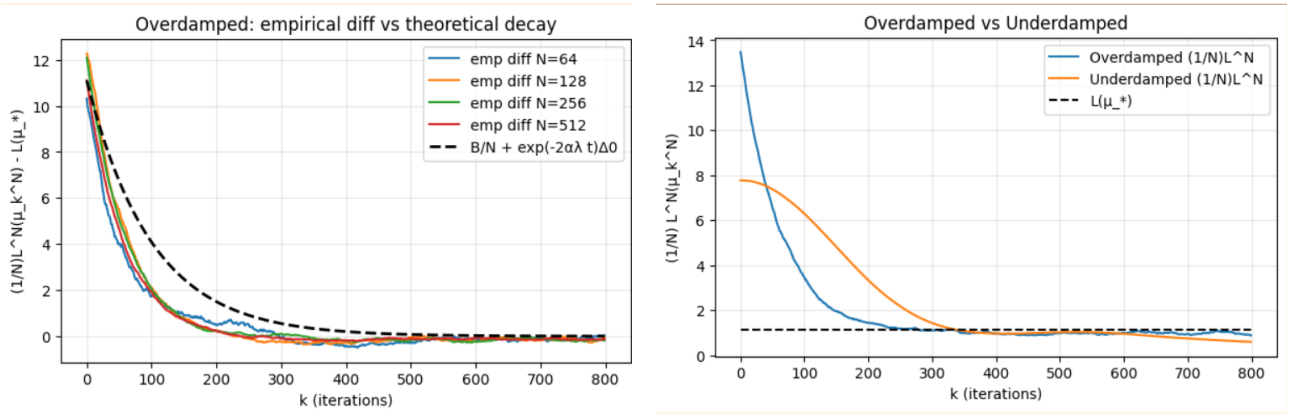


Figure 17: Overdamped and Underdamped (kinetic) Mean Field Langevin Dynamics

We can see that the convergence bounds seems to hold for both dynamics, and the underdamped dynamics shows faster early transport; overdamped offers steadier, more monotone convergence.

Glossary

σ-algebra A **σ-algebra** (or **σ-field**) $\mathcal{F}$ over a set Ω is a collection of subsets of Ω that satisfies:

- $\Omega \in \mathcal{F}$
- If $A \in \mathcal{F}$, then $A^c \in \mathcal{F}$ (closed under complementation)
- If $\{A_n\}_{n=1}^{\infty} \subseteq \mathcal{F}$, then $\bigcup_{n=1}^{\infty} A_n \in \mathcal{F}$ (closed under countable unions)

A σ-algebra represents a collection of events to which we can assign probabilities, and is fundamental in the definition of measurable spaces and random variables.

a posteriori distribution The a posteriori distribution is the probability distribution of a random variable or parameter θ after observing data x. It is obtained using **Bayes' theorem**, which combines the prior distribution $p(\theta)$ and the likelihood $p(\theta \mid x) = \frac{p(x|\theta)\, p(\theta)}{p(x)}$ where $p(x) = \int p(x \mid \theta) p(\theta)\, \mathrm{d}\theta$ is the marginal likelihood. The posterior reflects the updated beliefs about θ after observing data x.

Bayes' theorem Bayes' theorem is a fundamental theorem in probability theory that describes how to update the probability of a hypothesis based on new evidence.
It states that the a posteriori probability of a hypothesis H given observed data D is proportional to the product of the prior probability of H and the likelihood of D given H:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

where $P(D)$ is the marginal likelihood, which can be computed as $P(D) = \sum_H P(D|H)P(H)$ for discrete cases or $P(D) = \int P(D|H)P(H)dH$ for continuous cases.

Bregman divergence A measure of distance between points based on a convex function ϕ, defined as $D_\phi(x, y) = \phi(x) - \phi(y) - \nabla\phi(y)^\top(x - y)$. It generalizes squared Euclidean distance and includes KL divergence as a special case when $\phi(x) = \sum_i x_i \log x_i$.

Brownian motion Also known as a **Wiener process**, it is a fundamental process in stochastic analysis. It serves as the canonical example of a continuous-time martingale and is widely used in the modeling of random phenomena in physics, biology, and mathematical finance. It is a continuous-time stochastic process $\{W(t)\}_{t\geq 0}$ defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, and adapted to a **filtration** $\{\mathcal{F}_t\}_{t\geq 0}$, satisfying the following properties:
- $W(0) = 0$ almost surely, i.e., $\mathbb{P}(W(0) = 0) = 1$
- $W(t)$ has **independent increments**: for all $0 \leq t_0 < t_1 < \cdots < t_n$, the random variables $W(t_1) - W(t_0), \ldots, W(t_n) - W(t_{n-1})$ are independent
- $W(t)$ has **stationary increments** with Gaussian distribution: for $0 \leq s < t$, the increment $W(t) - W(s) \sim \mathcal{N}(0, t - s)$
- $W(t)$ has **almost surely continuous paths**, i.e., $\mathbb{P}(\{t \mapsto W(t) \text{ is continuous on } [0, \infty)\}) = 1$
- $W(t)$ is an $\{\mathcal{F}_t\}$**-adapted** process, and for each $t \geq 0$, $W(t)$ is $\mathcal{F}_t$-measurable .

Caltech101 The Caltech101 dataset is a collection of images used for object recognition tasks in computer vision. It contains 9,146 images of 101 object categories, with each category having a varying number of images.

CIFAR The CIFAR-10 and CIFAR-100 datasets are widely used benchmark datasets in machine learning and computer vision. CIFAR-10 consists of 60,000 32x32 color images in 10 classes, while CIFAR-100 contains 100 classes with 600 images each.

conditional Gibbs distribution For $\mathbf{x}^{-i} = (x^1, \ldots, x^{i-1}, x^{i+1}, \ldots, x^N)$, the conditional Gibbs distribution of the i-th coordinate given the others is the probability measure $\nu_{i|-i}(\,\cdot \mid \mathbf{x}^{-i})$ on $\mathbb{R}^d$ with density

$$\frac{\mathrm{d}\nu_{i|-i}}{\mathrm{d}x}(x \mid \mathbf{x}^{-i}) = \frac{\exp\big(-\frac{N}{\lambda} F(\rho_{\mathbf{x}^{-i}\cup x})\big)}{\int_{\mathbb{R}^d} \exp\big(-\frac{N}{\lambda} F(\rho_{\mathbf{x}^{-i}\cup \bar{x}})\big)\, \mathrm{d}\bar{x}},$$

where $\rho_{\mathbf{x}^{-i}\cup x} := \frac{1}{N}\sum_{j\neq i} \delta_{x^j} + \frac{1}{N}\delta_x$ is the empirical measure obtained by replacing x^i by x.

continuity equation for mass transport A partial differential equation describing how a probability distribution μ_t evolves over time under a velocity field. It expresses conservation of probability mass and takes the form:

$$\partial_t \mu_t + \nabla \cdot (\mu_t v_t) = 0,$$

where μ_t is a time-dependent probability density, and v_t is a velocity field over the domain. This equation ensures that as mass flows through space, none is created or destroyed—it merely moves .

defective Log-Sobolev inequality A relaxed form of the Log-Sobolev inequality allowing an additive defect term: for constants $C_{\mathrm{LS}} > 0$ and $b_{\mathrm{LS}} \geq 0$, a measure μ satisfies

$$\mathrm{Ent}_\mu(f^2) \leq 2C_{\mathrm{LS}} \int \|\nabla f\|_2^2 \, d\mu + b_{\mathrm{LS}} \int f^2 \, d\mu,$$

for smooth f. This inequality still yields entropy dissipation up to a defect and is instrumental in proving uniform-in-time propagation-of-chaos bounds with particle-approximation constants that avoid exponential dependence on regularization.

Digit The Digit dataset is a collection of handwritten digits used for image classification tasks, smaller and simpler than MNIST, making it suitable for educational purposes and quick experiments. It consists of 8x8 pixel grayscale images of digits (0-9) and is often used as a benchmark for evaluating machine learning algorithms.

Dual Averaging method An optimization algorithm that maintains an average of past gradients to determine updates, rather than directly applying the current gradient. It was introduced for convex optimization problems and is especially useful when dealing with large-scale stochastic settings. At iteration t, the dual averaging method computes:

$$g_t = \frac{1}{t} \sum_{i=1}^{t} \nabla f_i(\theta_i) \quad \text{and} \quad \theta_{t+1} = \arg\min_\theta \left\{ \langle g_t, \theta \rangle + \frac{1}{\eta_t} \psi(\theta) \right\}$$

where ψ is a strongly convex regularization function. This method forms the basis for optimization algorithms like AdaGrad and variants of stochastic gradient methods in deep learning and reinforcement learning.

embedding A representation of discrete objects (such as words, nodes, or categories) as vectors in a continuous vector space, typically $\mathbb{R}^d$, where semantic or structural relationships are preserved through geometric properties like distance and angle.

ensemble modeling A machine learning strategy that combines the predictions of multiple models (often called *base learners* or *ensemble members*) to improve generalization performance and robustness. In deep learning, ensembles often consist of independently trained neural networks, each with different initializations, training data subsets, or model architectures. The combined output can be a simple average, weighted average, or a more complex aggregation function. Ensemble modeling helps reduce variance, mitigate overfitting, and enhance predictive uncertainty estimation, particularly in tasks such as classification, regression, and out-of-distribution detection .

entropy For a probability measure μ that is absolutely continuous with respect to the Lebesgue measure (denoted $\mu \ll \mathrm{d}x$), the negative entropy is defined as:

$$\mathrm{Ent}(\mu) = \int \mu(\mathrm{d}x) \log \frac{\mathrm{d}\mu}{\mathrm{d}x}(x)$$

This is the negative of the differential entropy and appears in variational formulations and information geometry. It quantifies the amount of information or disorder in a distribution.

feedforward Neural network architecture where information flows unidirectionally from input to output without cycles. Commonly used for classification and regression tasks.

Fenchel-Young inequality A fundamental inequality in convex analysis stating that for a convex function ϕ and its convex conjugate ϕ^*, we have $\langle x, y \rangle \leq \phi(x) + \phi^*(y)$ for all x, y, with equality if and only if $y \in \partial\phi(x)$ (i.e., y is a subgradient of ϕ at x).

filtration A family of σ-algebras $\{\mathcal{F}_t\}_{t\geq 0}$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ such that $\mathcal{F}_s \subseteq \mathcal{F}_t \subseteq \mathcal{F}$ for all $0 \leq s \leq t$. That is, the information grows (or at least does not decrease) over time.
A stochastic process $\{X(t)\}_{t\geq 0}$ is said to be:

- $\mathcal{F}_t$**-adapted** (or *adapted to the filtration*) if for each $t \geq 0$, the random variable $X(t)$ is $\mathcal{F}_t$-measurable. Intuitively, this means the value of $X(t)$ is known given the information up to time t.
- $\mathcal{F}_t$**-measurable** if the pre-image of any **Borel set** under $X(t)$ belongs to $\mathcal{F}_t$, i.e., for any Borel set $B \subseteq \mathbb{R}$, we have $\{\omega \in \Omega : X(t)(\omega) \in B\} \in \mathcal{F}_t$.

Filtrations are used to model the evolution of information over time in stochastic processes, and are essential in the definition of martingales, stopping times, and adapted processes.

Fisher information A measure of the amount of information that an observable random variable carries about an unknown parameter upon which the probability depends.
Classical definition: For a parameter θ and a probability density function $p(x;\theta)$, the Fisher information $I(\theta)$ is defined as the variance of the score function (the gradient of the log-likelihood function):

$$I(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log p(X;\theta)\right)^2\right] = -\mathbb{E}\left[\frac{\partial^2}{\partial\theta^2}\log p(X;\theta)\right],$$

where X is a random variable with density $p(x;\theta)$. This form plays a central role in statistical inference, especially in the Cramér-Rao bound.
Information-theoretic (variational) Fisher information: For two probability measures μ and ν on a space where $\mu \ll \nu$ (i.e., μ is absolutely continuous with respect to ν), the Fisher information is defined as:

$$\mathrm{FI}(\mu\|\nu) = \int \mathrm{d}\mu(x)\left\|\nabla \log \frac{\mathrm{d}\mu}{\mathrm{d}\nu}(x)\right\|_2^2,$$

where $\frac{\mathrm{d}\mu}{\mathrm{d}\nu}$ is the Radon-Nikodym derivative of μ with respect to ν.

Fokker–Planck equation A partial differential equation describing the time evolution of the probability density function $p(x,t)$ of a stochastic process governed by a stochastic differential equation (SDE). Given the **Itô SDE**

$$dX_t = \mu(X_t)\,dt + \sigma(X_t)\,dW_t,$$

where $\mu(x) \in \mathbb{R}^d$ is the drift vector and $\sigma(x) \in \mathbb{R}^{d\times m}$ the diffusion matrix, the Fokker–Planck equation for the density $p(x,t)$ is:

$$\frac{\partial p}{\partial t} = -\nabla\cdot(\mu(x)p) + \frac{1}{2}\nabla\cdot(D(x)\nabla p),$$

where $D(x) = \sigma(x)\sigma(x)^\top$ is the diffusion tensor. The derivation is based on the conservation of probability: for any region $A \subset \mathbb{R}^d$, the total probability satisfies

$$\frac{d}{dt}\int_A p(x,t)\,dx = -\int_{\partial A}\mathbf{J}(x,t)\cdot n\,dS,$$

where $\mathbf{J}$ is the probability flux and n the outward normal. Applying the divergence theorem gives: $\frac{\partial p}{\partial t} = -\nabla\cdot\mathbf{J}$. The flux $\mathbf{J}(x,t)$ consists of a drift component $\mu(x)p(x,t)$ and a diffusive component given by Fick's law: $-\frac{1}{2}D(x)\nabla p(x,t)$. Substituting yields the Fokker–Planck equation.

framework Software platform providing a foundation for building applications. Includes pre-defined structures, libraries, and tools that simplify development through reusable components.

Fréchet differentiable A function F between Banach spaces X and Y is said to be Fréchet differentiable at a point $x \in X$ if there exists a bounded linear operator $A : X \to Y$ such that

$$\lim_{\|h\|_X\to 0}\frac{\|F(x+h)-F(x)-A(h)\|_Y}{\|h\|_X} = 0.$$

The operator A is called the Fréchet derivative of F at x and is denoted $DF(x)$. This concept generalizes the classical derivative to infinite-dimensional or functional settings and is stronger than Gâteaux differentiability.

functional derivative Given a functional $\mathcal{L} : \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$, the functional derivative $\frac{\delta\mathcal{L}}{\delta\mu}(\theta)$ quantifies how $\mathcal{L}$ changes under perturbations of the measure μ. It is defined via the first-order expansion $\mathcal{L}(\mu+\varepsilon(\nu-\mu)) = \mathcal{L}(\mu) + \varepsilon\int\frac{\delta\mathcal{L}}{\delta\mu}\,d(\nu-\mu) + o(\varepsilon)$.

generalization Generalization is the ability of a machine learning model to perform well on unseen data after being trained on a specific dataset. A model that generalizes effectively can make accurate predictions on new, previously unseen examples, indicating that it has learned the underlying patterns in the data rather than memorizing the training set. We usually use the generalization error evaluating this with a commonly used generalization bound given by:

$$\mathbb{E}_{x\sim\mathcal{D}}[L(f(x),y)] \leq \hat{L}_n(f) + \sqrt{\frac{h(\log(2n/h)+1) - \log(\delta/4)}{n}}$$

where $\hat{L}_n(f)$ is the empirical loss on n training examples, h is the VC-dimension of the hypothesis class, and δ is the confidence parameter. This bound holds with probability at least $1-\delta$.

gradient flow A gradient flow is a dynamical system that describes the evolution of a variable $x(t)$ in a space $\mathcal{X}$ such that it moves in the direction of the steepest descent of an energy or loss functional $\mathcal{E}(x)$. It is formally given by the differential equation $\frac{dx}{dt} = -\nabla\mathcal{E}(x(t))$, where $\nabla\mathcal{E}(x)$ denotes the gradient of $\mathcal{E}$ with respect to x. Gradient flows characterize processes that minimize $\mathcal{E}$ over time.

Gromov's theorem A theorem in geometric group theory which states that a finitely generated group (a group where every element can be expressed as a finite product of elements from a fixed finite subset and their inverses) has polynomial growth (the number of distinct group elements expressible as products of at most k generators grows at most like a polynomial in k) **if and only if** it is virtually nilpotent (it contains a **nilpotent subgroup** of finite index, meaning the number of distinct left cosets of the subgroup in the group is finite).

Grönwall inequality A fundamental tool for bounding solutions to differential or integral inequalities. If $u(t) \leq \phi(t) + \lambda\int_a^t u(s)\,ds$, then

$$u(t) \leq \phi(t) + \lambda\int_a^t \phi(s)e^{\lambda(t-s)}\,ds.$$

In particular, if $\phi(t) = C$, then $u(t) \leq Ce^{\lambda(t-a)}$. Widely used to prove uniqueness and stability in ODEs and stochastic processes.

Gâteaux differentiable A function F between Banach spaces X and Y is said to be Gâteaux differentiable at a point $x \in X$ in the direction $h \in X$ if the limit

$$DF(x)(h) = \lim_{\varepsilon\to 0}\frac{F(x+\varepsilon h) - F(x)}{\varepsilon}$$

exists. The Gâteaux derivative is a linear functional that provides a directional derivative of the function. It is weaker than Fréchet differentiability.

Han's inequality An information-theoretic inequality stating that the joint entropy of n random variables is upper bounded by the sum of their conditional entropies:

$$H(X_1,\ldots,X_n) \leq \sum_{i=1}^{n} H(X_i \mid X_{[n]\setminus\{i\}}).$$

Entropy is defined as $H(X) = -\sum_x p(x)\log p(x)$, joint entropy as $H(X,Y) = -\sum_{x,y} p(x,y)\log p(x,y)$, and conditional entropy as $H(X \mid Y) = -\sum_{x,y} p(x,y)\log p(x \mid y)$. Equality holds for $n = 2$; the inequality is strict in general for $n \geq 3$.

Iris The Iris dataset is a classic dataset in machine learning and statistics, consisting of 150 samples of iris flowers with four features (sepal length, sepal width, petal length, and petal width) and three species (Setosa, Versicolor, and Virginica).

Kernel methods Machine learning algorithms that use kernel functions to compute similarity between data points and implicitly transform data to higher-dimensional spaces. Enable linear algorithms to solve non-linear problems. Commonly used in SVMs and Gaussian processes.

Kullback-Leibler (KL) divergence (also called relative entropy) For probability measures $\mu \ll \nu$,

$$\mathrm{KL}(\mu\|\nu) = \int \log\left(\frac{\mathrm{d}\mu}{\mathrm{d}\nu}\right)\,\mathrm{d}\mu.$$

Intuition: it measures the inefficiency (extra bits) of using ν to model or encode data drawn from μ; it is asymmetric, non-negative, and vanishes iff $\mu = \nu$.

Log-Sobolev inequality A refinement of the classical Sobolev inequality which controls the entropy of a function with respect to its Dirichlet energy. It plays a central role in probability theory, statistical mechanics, and analysis of Markov semigroups.
Let μ be a probability measure on $\mathbb{R}^n$, typically the Gaussian measure. A function f (with suitable regularity and normalization $\int f^2 \, d\mu = 1$) satisfies the logarithmic Sobolev inequality if:

$$\int f^2 \log f^2 \, d\mu \leq 2C \int |\nabla f|^2 \, d\mu,$$

for some constant $C > 0$. In the Gaussian case, with $d\mu(x) = \frac{1}{(2\pi)^{n/2}} e^{-|x|^2/2} dx$, the inequality becomes:

$$\int_{\mathbb{R}^n} f^2 \log f^2 \, d\gamma \leq 2 \int_{\mathbb{R}^n} |\nabla f|^2 \, d\gamma.$$

This inequality implies concentration of measure, hypercontractivity of semigroups, and exponential convergence to equilibrium in entropy.

MNIST The MNIST dataset is a large database of handwritten digits (0-9) commonly used for training and evaluating machine learning models.
It consists of 60,000 training images and 10,000 test images, each 28x28 pixels in size.

nilpotent subgroup A subgroup G of a group H is said to be nilpotent if its lower central series terminates in the trivial subgroup after a finite number of steps.
The lower central series is defined as follows: $G = G_0 \trianglerighteq G_1 \trianglerighteq G_2 \trianglerighteq \cdots \trianglerighteq G_c = \{e\}$ where $G_{i+1} = [G, G_i]$ (the commutator subgroup of G and G_i).
A nilpotent group is one where the commutators eventually become trivial, indicating that the group is "close" to being abelian.

NLP (Natural Language Processing) Field of AI focused on enabling computers to understand, interpret, and generate human language. Develops algorithms and models for meaningful human-computer language interaction.

noise-to-signal ratio The ratio of noise power to signal power in a system, typically expressed as NSR $= \frac{\sigma^2_{\text{noise}}}{\sigma^2_{\text{signal}}}$, where higher values indicate more noise relative to the desired signal. The inverse quantity is the signal-to-noise ratio (SNR).

optimal transport Mathematical theory for finding the most efficient way to transport mass between distributions. Applications include economics, machine learning, and image processing. In ML, used for domain adaptation, generative modeling, and comparing probability distributions.

particle approximation method A sampling-based technique used to approximate complex probability distributions, particularly posterior distributions in Bayesian deep learning, using a finite set of weighted samples called *particles*. The posterior distribution $p(\theta \mid \mathcal{D})$ is approximated by a weighted sum of delta functions:

$$p(\theta \mid \mathcal{D}) \approx \sum_{i=1}^{N} w_i \delta(\theta - \theta_i)$$

where $\theta_i \in \mathbb{R}^d$ are the particles and $w_i \in \mathbb{R}$ are their associated weights. Common in methods like Sequential Monte Carlo (SMC), particle filters, and Stein Variational Gradient Descent (SVGD), these methods are widely applied in Bayesian neural networks for uncertainty quantification and **ensemble modeling**. .

Pinsker's inequality Relates the total variation distance and Kullback–Leibler divergence between two probability distributions P and Q:

$$\|P - Q\|_{\text{TV}} \leq \sqrt{\frac{1}{2} D_{\text{KL}}(P \| Q)}.$$

It provides an upper bound on the statistical distinguishability of P and Q.

Poincaré inequality A functional inequality controlling variance by Dirichlet energy. A probability measure μ on $\mathbb{R}^d$ satisfies a Poincaré inequality with constant $C_P > 0$ if for all smooth f with $\int f\, d\mu = 0$,

$$\mathrm{Var}_\mu(f) \le C_P \int \|\nabla f\|_2^2 \, d\mu.$$

Equivalently, $\int f^2\, d\mu - (\int f\, d\mu)^2 \le C_P \int \|\nabla f\|_2^2\, d\mu$. Poincaré inequalities yield spectral-gap estimates and exponential convergence in $L^2(\mu)$.

Polyak-Łojasiewicz inequality A condition for non-convex functions that enables exponential convergence of gradient descent. A function f satisfies the PL inequality with constant $c > 0$ if $\|\nabla f(x)\|^2 \ge 2c(f(x) - f^*)$ for all x, where f^* is the global minimum. This is weaker than strong convexity but still guarantees global convergence.

Principal Component Analysis (PCA) A linear dimensionality reduction technique that projects data onto the directions of maximum variance by computing the eigenvectors of the covariance matrix $\mathbf{C} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$, where principal components are ordered by their corresponding eigenvalues.

prior distribution The prior distribution is the probability distribution of a random variable before observing any data. It represents the initial beliefs or assumptions about the random variable based on prior knowledge or experience.
The prior distribution is combined with the likelihood of observed data to obtain the a posteriori distribution using **Bayes' theorem**.

pseudo-metric space A generalization of a metric space where the distance between distinct points can be zero. Formally, a set $\mathcal{X}$ with a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\ge 0}$ that is non-negative, symmetric, and satisfies the triangle inequality, but may not distinguish distinct points.

Rademacher complexity A measure of the richness of a class of functions, defined as the expected value of the supremum of the empirical process over a sample of size n. It quantifies how well a class can fit random noise and is used in learning theory to bound generalization error. Formally, for a class $\mathcal{F}$ and sample $S = (x_1, \ldots, x_n)$, the Rademacher complexity is given by:

$$\hat{\mathcal{R}}_n(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i) \right],$$

where σ_i are independent Rademacher variables taking values ± 1 with equal probability.

Radon-Nikodym derivative Given two measures μ and ν such that $\mu \ll \nu$ (i.e., μ is absolutely continuous with respect to ν), the Radon-Nikodym derivative $\frac{\mathrm{d}\mu}{\mathrm{d}\nu}$ is a function that satisfies:

$$\mu(A) = \int_A \frac{\mathrm{d}\mu}{\mathrm{d}\nu}(x)\, \mathrm{d}\nu(x)$$

for all measurable sets A. It represents the density of μ with respect to ν.

Sobolev inequality A fundamental result in functional analysis and partial differential equations that relates norms of a function to norms of its derivatives. In its basic form, the Sobolev inequality provides an embedding of Sobolev spaces into Lebesgue spaces.
Let $u \in W^{1,p}(\mathbb{R}^n)$, the Sobolev space of functions with first weak derivative in L^p. If $1 \le p < n$, then there exists a constant $C > 0$ such that: $\|u\|_{L^{p^*}(\mathbb{R}^n)} \le C\|\nabla u\|_{L^p(\mathbb{R}^n)}$, where $p^* = \frac{np}{n-p}$ is the Sobolev conjugate exponent. For $p = 2$ and $n > 2$, the inequality becomes: $\|u\|_{L^{\frac{2n}{n-2}}(\mathbb{R}^n)} \le C\|\nabla u\|_{L^2(\mathbb{R}^n)}$. Sobolev inequalities are essential in the study of PDE regularity, variational methods, and embedding theorems.

SOTA State Of The Art - the best known method or technique in a specific field at a given time. Used in research papers to refer to the most advanced and effective approaches available.

space of probability measures Denoted $\mathcal{P}(\mathbb{R}^d)$, this is the set of all Borel probability measures on $\mathbb{R}^d$, i.e., measures μ such that $\mu \ge 0$ and $\int_{\mathbb{R}^d} d\mu = 1$.
Often, we consider the subspace $\mathcal{P}_2(\mathbb{R}^d)$, consisting of measures with finite second moments: $\int_{\mathbb{R}^d} \|x\|^2 d\mu(x) < \infty$.

statistical physics Branch of physics using statistical methods to describe large ensembles of particles or systems. Connects microscopic interactions to macroscopic properties like temperature and pressure. Essential in thermodynamics, condensed matter physics, and complex systems.

stochastic differential equation A differential equation that incorporates randomness or noise into its dynamics.
SDEs are used to model systems influenced by random processes, such as financial markets, physical systems with uncertainty, and biological processes.
They are typically expressed in the form $dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t$, where μ is the drift term, σ is the diffusion term, and W_t is a Wiener process (Brownian motion).

stochastic process A **stochastic process** is a collection of random variables $\{X(t)\}_{t\in T}$ defined on a common probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where T is an index set, typically representing time (e.g., $T = \mathbb{N}$ for discrete time or $T = [0, \infty)$ for continuous time). For each fixed $t \in T$, the function $X(t) : \Omega \to \mathbb{R}$ (or more generally $\mathbb{R}^d$) is a random variable.
A stochastic process models the evolution of a random quantity over time and is used in various fields such as probability theory, statistics, physics, and finance.

total variation norm A measure of the maximal difference between two probability distributions P and Q over a measurable space. Defined as:

$$\|P - Q\|_{\mathrm{TV}} = \sup_A |P(A) - Q(A)| = \frac{1}{2}\int |p(x) - q(x)|dx,$$

where p and q are the densities of P and Q, if they exist.

vanishing gradient problem A phenomenon in training deep or recurrent neural networks where gradients shrink exponentially as they are backpropagated through many layers or time steps, effectively preventing weight updates in early layers. In RNNs, this arises from repeated multiplication by the Jacobian $\frac{\partial h_t}{\partial h_{t-1}}$, which can lead to

$$\left\|\frac{\partial \mathcal{L}}{\partial h_1}\right\| \approx \left\|\prod_{t=1}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right\| \to 0$$

when eigenvalues of the transition matrix are less than one. This makes it difficult for the network to learn long-term dependencies.

Wasserstein metric space The space $(\mathcal{P}_2(\mathbb{R}^d), W_2)$, where W_2 is the Wasserstein-2 distance. It provides a geometric structure on the space of probability measures with finite second moments. Gradient flows in this space describe the evolution of probability distributions via transport equations driven by the steepest descent of a functional.

Wasserstein-2 metric A distance function defined on the space of probability measures with finite second moments, denoted $\mathcal{P}_2(\mathbb{R}^d)$. Given two measures $\mu, \nu \in \mathcal{P}_2(\mathbb{R}^d)$, the Wasserstein-2 distance is defined as

$$W_2^2(\mu, \nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \int_{\mathbb{R}^d\times\mathbb{R}^d} \|x - y\|^2 \, d\gamma(x, y),$$

where $\gamma \in \Gamma(\mu, \nu)$ is the set of all couplings of μ and ν, i.e., joint probability measures on $\mathbb{R}^d \times \mathbb{R}^d$ such that the marginals of γ satisfy: $\pi^1_\#\gamma = \mu$ and $\pi^2_\#\gamma = \nu$, where $\pi^1(x, y) = x$ and $\pi^2(x, y) = y$.
The Wasserstein-2 metric equips the space of distributions with a geometric structure that plays a central role in optimal transport theory and variational formulations of gradient flows.

References

[1] C. Hongler A. Jacot, F. Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems (NeurIPS)*, 31, 2018.

[2] Zhengdao Chen. Neural hilbert ladders: Multi-layer neural networks in function space. *Journal of Machine Learning Research*, 25:1–65, 2024. arXiv:2307.01177.

[3] Sinho Chewi, Murat A Erdogdu, Mufan Bill Li, Ruoqi Shen, and Matthew Zhang. Analysis of langevin monte carlo from poincaré to log-sobolev. *arXiv preprint arXiv:2112.12662*, 2022. Published at *Foundations of Computational Mathematics*.

[4] Lénaïc Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in Neural Information Processing Systems*, 2018.

[5] Lénaïc Chizat. Mean-field langevin dynamics: Convergence and applications. Lecture, Center for Intelligent Systems (CIS) at EPFL, 2023. Accessed May 2025.

[6] Qiang Fu and Ashia Wilson. Mean-field underdamped langevin dynamics and its spacetime discretization. *arXiv preprint arXiv:2312.16360*, 2023. Version 5, last revised 6 Feb 2024.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[8] Mark Kac. Foundations of kinetic theory. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, 1954–1955, Vol. III: Contributions to Astronomy and Physics*, pages 171–197, Berkeley and Los Angeles, 1956. University of California Press.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, volume 25, pages 1097–1105, 2012.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[11] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit. *arXiv preprint arXiv:1802.05800*, 2018.

[12] Atsushi Nitanda, Anzelle Lee, Damian Xing Kai Tan, Mizuki Sakaguchi, and Taiji Suzuki. Propagation of chaos for mean-field langevin dynamics and its application to model ensemble. *arXiv preprint arXiv:2502.05784*, 2025.

[13] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Advances in neural information processing systems*, 29, 2016.

[14] Taiji Suzuki. Suzuki's homepage. `https://ibis.t.u-tokyo.ac.jp/suzuki/`. Accessed 2025-05-01.

[15] Alain-Sol Sznitman. Topics in propagation of chaos. In Paul-Louis Hennequin, editor, *École d'Été de Probabilités de Saint-Flour XIX—1989*, volume 1464 of *Lecture Notes in Mathematics*, pages 165–251. Springer, https://doi.org/10.1007/BFb0085169, 1991.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, 2017.

[17] Greg Yang and Edward J. Hu. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. In *Advances in Neural Information Processing Systems*, 2020.