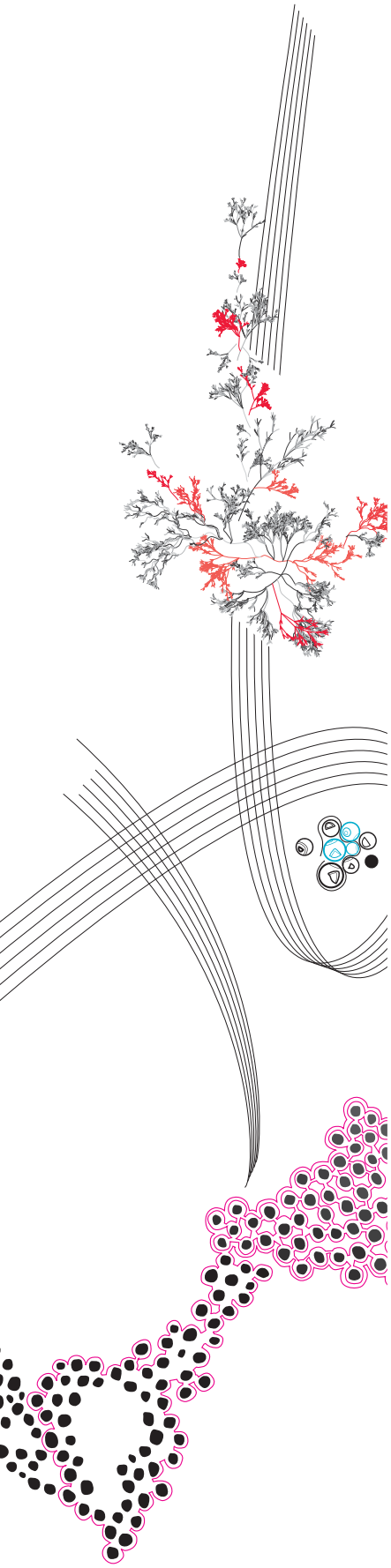


BSc Thesis Applied Mathematics

Accelerating Neural Network Training Using the Neural Tangent Kernel

Kiril Sarvanau

Supervisor:
Chairperson: Prof. Dr. C. Brune
Daily supervisor: MSc. T.J. Heeringa

July, 2025

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Contents

1 Introduction **1**
- 1.1 What are NNs used for? . . . 1
- 1.2 What are the existing problems of NNs? . . . 2
- 1.3 What's NTK, and how can it help? . . . 2
- 1.4 What is my research about? . . . 3
- 1.5 Outline of the Report . . . 3

2 Related Works: The Neural Tangent Kernel **3**
- 2.1 Discovery and Key Research . . . 3
- 2.2 Limitations of NTK Theory . . . 4
- 2.3 Applications of Neural Tangent Kernels . . . 4
- 2.4 The Connection to "Lazy Training" . . . 4

3 Theory Background **4**
- 3.1 Neural Networks Background . . . 4
  - 3.1.1 What are they? . . . 4
  - 3.1.2 Structure . . . 5
  - 3.1.3 Focus on optimization . . . 6
  - 3.1.4 Types of networks (CNN, recurrent, LSTM, DT) . . . 6
- 3.2 Defining NTK . . . 6
  - 3.2.1 Kernel Methods and Kernel Gradient Descent . . . 6
  - 3.2.2 Introducing the Neural Tangent Kernel (NTK) . . . 7
  - 3.2.3 Why NTK is Constant in Infinite-Width Limit . . . 7
  - 3.2.4 Training with NTK (NTK Flow) . . . 7

4 Experimental Setup **8**
- 4.1 Datasets . . . 8
  - 4.1.1 Fashion-MNIST (Classification) . . . 8
  - 4.1.2 Wine Quality (Regression) . . . 8
- 4.2 Neural Network Architectures . . . 8
- 4.3 Hybrid Training Strategy and Switch Point Derivation . . . 9
  - 4.3.1 Heuristics for NTK Stability . . . 9
  - 4.3.2 Scouting Run and Threshold Determination . . . 9
- 4.4 Derivation of Neural Tangent Kernel (NTK) Updates . . . 9
- 4.5 Implementation Details . . . 11

5 Results and Analysis **12**
- 5.1 Results for Fashion-MNIST Dataset . . . 12
  - 5.1.1 Scouting Run and Threshold Determination . . . 12
  - 5.1.2 Results for Parameter Norm Heuristic . . . 13
  - 5.1.3 Results for NTK Stability Heuristic . . . 13
- 5.2 Results for Wine Quality Dataset . . . 14
  - 5.2.1 Scouting Run and Threshold Determination . . . 14
  - 5.2.2 Results for Parameter Norm Heuristic . . . 15
  - 5.2.3 Results for NTK Stability Heuristic . . . 16
- 5.3 Summary of Final Performance . . . 17

6 Discussion **17**
6.1 Comparison of Results 18
6.2 Limitations 18

7 Conclusion **19**
7.1 Future Work 19

Accelerating Neural Network Training Using the Neural Tangent Kernel

Kiril Sarvanau

July, 2025

Abstract

The training of deep neural networks, while foundational to modern artificial intelligence, presents significant computational challenges that limit efficiency and scalability. This thesis investigates the Neural Tangent Kernel (NTK) as a theoretical framework to accelerate this process. We propose a hybrid training strategy that begins with standard Stochastic Gradient Descent (SGD) and transitions to an NTK-based update method once the network enters a stable, "lazy training" regime. A core contribution of this work is the development of data-driven heuristics, based on monitoring either the change of parameter norm or the stability of the NTK, to dynamically identify the optimal switch point. This methodology is systematically evaluated on two distinct tasks: image classification with the Fashion-MNIST dataset and regression with the Wine Quality dataset. We implement and compare three different NTK update methods against a pure SGD baseline across networks of varying widths. Our results demonstrate that the hybrid approach, particularly a one-shot update derived from the integrated NTK flow, can achieve comparable final performance to full SGD training in a fraction of the time, validating the strategy as a promising method for accelerating neural network convergence.

1 Introduction

Neural networks (NNs) have become a foundational technology in contemporary artificial intelligence, demonstrating exceptional capabilities across a broad spectrum of applications. Despite their widespread adoption and impressive performance, the training of deep neural networks continues to present substantial challenges that necessitate advanced theoretical understanding and practical optimization. This thesis explores the Neural Tangent Kernel (NTK) as a framework to address these challenges, specifically focusing on its potential to accelerate neural network training.

1.1 What are NNs used for?

Neural networks are computational models inspired by the human brain, designed to learn complex patterns from data. They are widely used across various domains, including image and speech recognition, natural language processing, optical character recognition, and facial recognition [1]. NNs also excel at approximating complex functions for predictive modeling. Recent advancements in generative AI, such as large language models like ChatGPT and image generation tools like MidJourney, are fundamentally powered by advanced neural network architectures [2]. Their impact extends to specialized engineering fields like optical communications, where they enhance transmission quality, and smart devices for applications like human presence detection. The transformative influence of deep learning, supported by NNs, is evident in law enforcement, engineering, medicine, and bioinformatics. This widespread application necessitates a deeper theoretical understanding and practical optimization of NN training for reliability, efficiency, and scalability.

1.2 What are the existing problems of NNs?

Despite their successes, training deep neural networks faces significant challenges impacting efficiency, scalability, and deployment. Key problems include:

- **Computational Inefficiency and Scalability:** Training deep NNs is computationally demanding and time-consuming. Recurrent Neural Networks (RNNs), for instance, can have slow training times with large sequential data [3]. Achieving low-complexity implementations for real-time applications, like NN-based equalizers, requires extensive investigation across training, inference, and hardware synthesis. Deep learning models generally incur high computational costs.
- **Optimization Difficulties:** Vanishing and exploding gradients during backpropagation hinder convergence in deep architectures. The non-convex loss landscape of deep networks also makes optimization challenging [8].
- **Data Requirements and Generalization:** NNs typically need large datasets to perform effectively, which is problematic with limited or sparse data. Overfitting, where models perform well on training data but poorly on unseen data, is a common issue affecting generalization [7].
- **Lack of Interpretability and Transparency:** The "black box" nature of NNs, particularly their hidden layers, makes it difficult to understand their decision-making. This lack of transparency complicates establishing ethical AI guidelines [9].

Overcoming these interconnected challenges is crucial for the advancement and responsible deployment of deep learning.

1.3 What's NTK, and how can it help?

The Neural Tangent Kernel (NTK) is a theoretical framework that connects deep neural networks to kernel methods, offering powerful insights into their training dynamics [10]. In the limit of infinite width, the training of a neural network with gradient descent simplifies to a linear model, whose behavior is governed by a fixed kernel—the NTK. This linearization makes the complex, non-convex optimization problem of NNs analytically tractable.

Mathematically, the NTK matrix, $\mathbf{\Theta}(\mathbf{x}, \mathbf{x}')$, is defined as the inner product of the network's output gradients with respect to its parameters θ for two inputs, $\mathbf{x}$ and $\mathbf{x}'$:

$$\mathbf{\Theta}(\mathbf{x}, \mathbf{x}') = \left\langle \frac{\partial f(\mathbf{x}, \theta)}{\partial \theta}, \frac{\partial f(\mathbf{x}', \theta)}{\partial \theta} \right\rangle \tag{1}$$

This property allows us to analyze and, more importantly, accelerate training. While standard Stochastic Gradient Descent (SGD) updates parameters iteratively,

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta L(\theta_k) \tag{2}$$

the NTK framework provides an alternative. When a network enters a "lazy training" regime—where its parameters change little—we can freeze its Jacobian at its initial state, J_0, and leverage the NTK to derive more direct update methods. This leads to a family of hybrid updates that can be generally expressed as:

$$\theta_{k+1} = \theta_k - \eta' J_0^T \tilde{\epsilon}_k \tag{3}$$

where η' is an effective learning rate and $\tilde{\epsilon}_k$ is a modified error term derived from NTK theory. This thesis develops and evaluates three such NTK update methods, with the goal of demonstrating that this hybrid approach can significantly speed up convergence compared to pure SGD.

1.4 What is my research about?

This bachelor's thesis investigates the potential of the Neural Tangent Kernel (NTK) to accelerate neural network training by leveraging its eigenstructure. The research aims to explore how the theoretical framework provided by the NTK can be utilized to overcome computational and optimization challenges inherent in training deep neural networks. Specifically, this work is divided into two main parts:

1. **Heuristic for NTK Stability:** Developing a heuristic to determine when neural network parameters enter a regime of minimal change, allowing the NTK approximation to hold.

2. **Iterative Scheme for NTK Flow:** Designing and implementing an iterative scheme or descent method derived from the NTK flow equation for training the neural network.

This project has the potential to provide a method that significantly speeds up neural network training.

1.5 Outline of the Report

This report is structured to systematically explore the application of the Neural Tangent Kernel in accelerating neural network training. Section 2, Related Works, reviews existing literature on the Neural Tangent Kernel, from its foundational introduction to its limitations. Section 3, Theory Background, establishes the necessary theoretical foundation, detailing the fundamental concepts of neural networks and providing a comprehensive definition of the NTK. Section 4, Experiments for Dataset 1, details the experimental setup for the first dataset (Fashion-MNIST), including network architectures and methodology. Section 5, Experiments for Dataset 2, replicates this methodology on a second dataset (Wine Quality) to assess generalizability. Section 6, Discussion, analyzes and compares the results from both datasets, discusses limitations, and suggests avenues for future work. Finally, Section 7, Conclusion, summarizes the key findings and contributions of the thesis.

2 Related Works: The Neural Tangent Kernel

The Neural Tangent Kernel (NTK) is a key idea for understanding how wide neural networks learn and generalize. It helps connect the complex world of deep learning optimization with simpler kernel methods.

2.1 Discovery and Key Research

Jacot, Gabriel, and Hongler discovered the Neural Tangent Kernel in 2018 [10]. They showed a major link: for very wide neural networks, training with gradient descent is like using a simple kernel method with a fixed NTK. This makes it easier to study how networks learn by looking at their function behavior instead of their complex internal parameters.

After this discovery, the NTK idea was applied to different network types. For example, the Convolutional NTK (CNTK) was developed for CNNs, providing an efficient way to calculate it [11]. More recently, Equivariant NTKs (E-NTKs) were created for networks that use symmetries in data, which is useful in areas like medical imaging and quantum chemistry. Researchers also looked at how NTK behaves in deeper networks, studying how starting conditions and activation functions affect it [14]. It was found that the NTK stays stable even with minimal over-parameterization. A new approach involves directly optimizing the NTK itself, aiming to improve how well models generalize [15].

2.2 Limitations of NTK Theory

Despite its strengths, NTK theory has limits, mainly because it assumes networks are infinitely wide. In practical scenarios, where networks have finite width and considerable depth, this assumption often doesn't hold. Studies have shown that the NTK can change a lot during training, which goes against the idea that it stays constant for infinitely wide networks [16]. Also, the idea that trained neural networks are exactly like kernel regression with NTK has been questioned. Observations suggest that adding layers to a network doesn't always lead to the same changes in prediction error for the NTK. There are also concerns about whether the NTK can fully capture the complexity of deep networks or explain how well they generalize and scale with data, as real networks often perform better than what NTK theory predicts [18].

2.3 Applications of Neural Tangent Kernels

NTK theory helps us understand fundamental aspects of neural networks, such as why they tend to learn simple patterns first (spectral bias) and how gradient descent finds good solutions [19]. Beyond just understanding, NTK has practical uses. Equivariant NTKs, for example, have performed better in tasks like classifying medical images and predicting molecular energies, by taking advantage of data symmetries. Directly training the NTK using methods like NTK-KARE is another important application, showing that it can sometimes match or even surpass traditional deep neural networks in generalization [15].

2.4 The Connection to "Lazy Training"

The Neural Tangent Kernel is closely tied to "lazy training" in neural networks [5]. Lazy training describes a regime where the network's internal changes (its Jacobian) don't vary much during training, even as the network learns a lot. This happens because in very wide networks, individual parameters change very little, allowing the network's behavior to be described by a simple linear approximation around its starting point. In this simplified mode, training with gradient descent becomes equivalent to a kernel method using the NTK [12]. However, it's important to remember that lazy training isn't always the best. Some studies show that deep convolutional networks can perform worse when trained in this "lazy" way, suggesting that the benefits of complex, non-linear feature learning might go beyond what the simplified NTK model describes [5, 13].

3 Theory Background

This section establishes the theoretical foundation necessary for understanding neural networks and the Neural Tangent Kernel, detailing their structural components, functional mechanisms, and the mathematical principles that govern their behavior.

3.1 Neural Networks Background

3.1.1 What are they?

Neural networks (NNs) are computational models inspired by the human brain, designed to learn complex patterns from data. They consist of interconnected processing units called "neurons" or "nodes," organized in layers. Through training, NNs learn to identify patterns and relationships in datasets, enabling tasks like classification, regression, and data generation.

3.1.2 Structure

A typical neural network comprises an **input layer** for raw data, one or more **hidden layers** for learning and computation, and an **output layer** for final predictions. The "depth" is the number of hidden layers, and "width" is the number of neurons per layer.

Neurons are basic computational units that perform a weighted sum of inputs, add a bias, and apply an activation function. **Weights** w_{ij} control connection strength between neurons, and **biases** b_j shift activation functions, both being trainable parameters. The output of a neuron j in a layer, before activation, can be expressed as:

$$z_j = \sum_i w_{ij}\mathbf{x}_i + b_j, \tag{4}$$

where $\mathbf{x}_i$ are inputs from the previous layer.

Nonlinearity is introduced by **activation functions** σ, applied to neuron outputs (except the input layer), enabling the network to learn non-linear relationships. Common activation functions include:

- **ReLU (Rectified Linear Unit):** $\sigma(z) = \max(0, z)$
- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$
- **Tanh (Hyperbolic Tangent):** $\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

The final output is the network's prediction. A **loss function** L quantifies the discrepancy between predicted and true outputs, which the network aims to minimize. For regression, a common choice is the Mean Squared Error (MSE):

$$L(\theta) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i, \theta))^2, \tag{5}$$

where N is the number of samples, y_i are true labels, and $f(\mathbf{x}_i, \theta)$ are the network's predictions for input $\mathbf{x}_i$ with parameters θ.

Optimization is the iterative process of adjusting weights and biases to minimize the loss. This involves:

1. **Forward Propagation:** Input data passes through the network to generate an output.
2. **Loss Calculation:** The loss function is computed.
3. **Backpropagation:** Gradients of the loss with respect to each parameter are calculated using the chain rule.
4. **Parameter Update:** Parameters are adjusted using an optimization algorithm (e.g., Gradient Descent, SGD, Adam) based on the gradients. For Gradient Descent, the update rule is:

   $$\theta_{t+1} = \theta_t - \eta\nabla_\theta L(\theta_t), \tag{6}$$

   where θ_t are parameters at time t, η is the learning rate, and $\nabla_\theta L(\theta_t)$ is the gradient of the loss function.

3.1.3 Focus on optimization

Optimizing neural networks involves strategic decisions beyond gradient descent. **Hyperparameter tuning**, such as selecting the learning rate, is crucial for efficient convergence [1]. **Preventing overfitting** through regularization (e.g., L1/L2, dropout), early stopping, and data augmentation is vital for generalization [7]. Automated techniques like **Neural Architecture Search (NAS)** are used to find optimal network structures by minimizing size and computational operations (FLOPS) while maintaining accuracy [6]. Despite these methods, challenges like vanishing/exploding gradients [8] and the need for low-complexity implementations for real-time deployment persist.

3.1.4 Types of networks (CNN, recurrent, LSTM, DT)

Neural networks feature diverse architectures tailored for specific data types and tasks. Note that "Decision Tree" is a distinct machine learning algorithm, not a neural network.

- **Multilayer Perceptrons (MLPs) / Feedforward Neural Networks (FNNs):** The simplest NNs with unidirectional information flow from input to output through hidden layers. Used for general pattern recognition and function approximation [1]. Limitations include inability to handle sequential data and fixed input size.
- **Convolutional Neural Networks (CNNs):** Designed for spatial data like images and videos. They use convolutional and pooling layers to extract hierarchical features, offering automatic filter learning and parameter sharing [17, 1].
- **Recurrent Neural Networks (RNNs):** Process sequential data, featuring a "memory" component where hidden layers use previous inputs for future predictions. Ideal for time series, text, and audio. Prone to vanishing/exploding gradients and slow training for long sequences [3].
- **Long Short-Term Memory (LSTM) Networks:** A variant of RNNs that overcome short-term memory limitations using "memory cells" with input, output, and forget gates. This allows them to retain information over long sequences and mitigate vanishing gradients [3].
- **Decision Trees (DTs):** A machine learning algorithm (not an NN) that builds a tree-like model of decisions. They work well with smaller datasets, offer high transparency, and have lower computational costs than NNs. However, they are sensitive to data changes and less effective with very large or complex datasets, often requiring ensemble methods.

The architectural variety and comparison with traditional models like Decision Trees highlight a trade-off: NNs offer high performance and complex pattern recognition but demand significant resources and often lack interpretability, while DTs provide transparency and efficiency for simpler tasks. This trade-off drives research into methods like NTK to enhance NN interpretability and analytical tractability.

3.2 Defining NTK

3.2.1 Kernel Methods and Kernel Gradient Descent

Kernel methods are a class of algorithms that operate on data by computing pairwise similarities between data points using a kernel function. A multi-dimensional kernel $\mathbf{K}$ is a function $\mathbf{K} : R^{n_0} \times R^{n_0} \to R^{n_L \times n_L}$ that maps any pair of inputs $(\mathbf{x}, \mathbf{x}')$ to an $n_L \times n_L$ matrix, where n_0 is the input dimension and n_L is the output dimension.

The training of a function f_t in a function space can be described by kernel gradient descent. A time-dependent function $f(t)$ follows kernel gradient descent with respect to a kernel $\mathbf{K}$ if it satisfies

the differential equation:

$$\partial_t f(t) = -\nabla_{\mathbf{K}} C|_{f(t)} \tag{7}$$

where $\nabla_{\mathbf{K}} C$ is the kernel gradient of a cost functional C. For a finite dataset $\{\mathbf{x}_i\}_{i=1}^N$, this gradient takes the form:

$$\nabla_{\mathbf{K}} C|_{f(t)}(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^{N} \mathbf{K}(\mathbf{x}, \mathbf{x}_j) d|_{f(t)}(\mathbf{x}_j) \tag{8}$$

Here, $d|_{f(t)}$ is the functional derivative, representing the direction of steepest descent in function space. Convergence to a global minimum is guaranteed if the cost is convex [10].

3.2.2 Introducing the Neural Tangent Kernel (NTK)

The Neural Tangent Kernel (NTK) is a specific kernel that describes the evolution of a deep neural network during gradient descent training. The NTK matrix, $\mathbf{\Theta}(\mathbf{x}, \mathbf{x}')$, is defined as the inner product of the network's output gradients with respect to its parameters θ for two inputs, $\mathbf{x}$ and $\mathbf{x}'$:

$$\mathbf{\Theta}(\mathbf{x}, \mathbf{x}') = \left\langle \frac{\partial f(\mathbf{x}, \theta)}{\partial \theta}, \frac{\partial f(\mathbf{x}', \theta)}{\partial \theta} \right\rangle \tag{9}$$

Here, $f(\mathbf{x}, \theta)$ represents the network function. Alternatively, if a network has an output size of O and P parameters, its Jacobian $\mathbf{J}(\mathbf{x}, \theta) = \frac{\partial f(\mathbf{x}, \theta)}{\partial \boldsymbol{\theta}}$ is an $O \times P$ matrix. The NTK matrix is then given by $\mathbf{J}(\mathbf{x}, \theta)\mathbf{J}(\mathbf{x}', \theta)^T$. This formulation makes the NTK inherently symmetric and positive semi-definite, validating its use as a kernel.

3.2.3 Why NTK is Constant in Infinite-Width Limit

In general, for finite-width neural networks, the NTK is random at initialization and varies during training. However, a key insight of NTK theory is that in the limit of infinite layer width, the NTK becomes deterministic and remains constant throughout training [10]. This property is crucial because it allows the training dynamics to be analyzed as a fixed linear system.

This constancy is a consequence of the "lazy training" phenomenon [5]. In this regime, individual network parameters change negligibly, even as their collective effect significantly changes the network's output. This allows the network's behavior to be accurately captured by its first-order Taylor expansion around the initial parameters, effectively linearizing the network.

3.2.4 Training with NTK (NTK Flow)

For ANNs trained with gradient descent, the network function f_t evolves along the negative kernel gradient. In the infinite-width limit, the dynamics of f_t are described by the NTK flow differential equation. For a least-squares regression cost, this is:

$$\frac{\partial f_t}{\partial t} = -\frac{1}{N} \mathbf{\Theta}_\infty (f_t - f^*) \tag{10}$$

where f^* is the target function and $\mathbf{\Theta}_\infty$ is the limiting NTK matrix over the training data. The solution to this equation is given by:

$$f_t = f^* + e^{-t\mathbf{\Theta}_\infty/N}(f_0 - f^*) \tag{11}$$

where f_0 is the initial network function. This solution shows that the network's output converges to the target function along the eigenspaces of the NTK matrix, with convergence being fastest along the directions corresponding to the largest eigenvalues. This motivates using NTK-based methods after the initial learning phase, as they can focus convergence on the most relevant components.

4 Experimental Setup

This section details the experimental setup for evaluating the proposed hybrid training approach, which combines Stochastic Gradient Descent (SGD) with Neural Tangent Kernel (NTK) based updates. The goal is to demonstrate whether leveraging the NTK's structure can accelerate neural network training.

4.1 Datasets

To assess the generalizability and robustness of our methods, we conduct experiments on two distinct datasets that represent different scales and machine learning tasks: a standard, well-behaved image classification task and a more challenging, non-image based regression task. For classification, performance is measured using **cross-entropy loss** and **accuracy**. For regression, we use **Mean Squared Error loss** (MSE) and **Normalized Root Mean Squared Error** (NRMSE).

4.1.1 Fashion-MNIST (Classification)

As a more challenging drop-in replacement for MNIST, Fashion-MNIST serves as a test for the heuristic's robustness on a classification problem. It contains 70,000 grayscale images (28x28 pixels) across 10 categories of fashion products (e.g., T-shirt, trouser, bag). For our experiments, each image is flattened into a 784-dimensional vector. The data is partitioned into 42,000 training images, 14,000 validation images, and 14,000 test images.

4.1.2 Wine Quality (Regression)

To evaluate our heuristics on a non-image, regression-based task, we use the Wine Quality dataset. This dataset contains physicochemical properties of red wines and a continuous ‘quality‘ score. It comprises 1,599 samples and 11 feature columns. Features are standardized using ‘StandardScaler‘. The data is split into a 959-sample training set, a 320-sample validation set, and a 320-sample test set.

4.2 Neural Network Architectures

For all experiments, we construct a standard Multilayer Perceptron (MLP). This architecture was chosen for its fundamental role in deep learning and its direct applicability to foundational NTK theory. The network architecture is fixed in depth to contain two hidden layers. The input layer size is determined by the dataset (784 for Fashion-MNIST, 11 for Wine Quality), and the output layer size corresponds to the number of classes (10 for Fashion-MNIST) or output dimension (1 for Wine Quality).

All hidden layers utilize the Rectified Linear Unit (ReLU) activation function. To investigate the impact of network width on the stability heuristics, we systematically test three different hidden layer widths:

- **Narrow Network:** 100 neurons per hidden layer.
- **Medium Network:** 500 neurons per hidden layer.
- **Wide Network:** 1000 neurons per hidden layer.

This range allows us to observe how dynamics change as the network approaches the "over-parameterized" regime, where NTK theory is most accurate. Network parameters are initialized using Glorot normal initialization [8].

4.3 Hybrid Training Strategy and Switch Point Derivation

The training process employs a hybrid strategy:

1. **Initial SGD Phase:** The network is first trained using SGD for a number of epochs. This allows parameters to move away from random initialization and learn initial features.
2. **Switch to NTK Update:** At a dynamically identified point, training switches to an NTK-based update method to leverage the linearized dynamics and accelerate convergence.

4.3.1 Heuristics for NTK Stability

A crucial aspect is determining the optimal switch point. We implement and monitor two heuristics to identify when the network enters a stable, "lazy training" regime.

- **Monitoring Parameter Change Norm (ϵ):** We track the Frobenius norm of the change in the network's parameters over a fixed window of recent epochs. The switch occurs when this norm falls below a predefined threshold, ϵ. A small $\Delta\theta_t = ||\theta_t - \theta_{t-k}||_F < \epsilon$ indicates parameter stabilization.
- **Monitoring Empirical NTK Stability (δ):** We periodically compute the empirical NTK matrix ($\mathbf{\Theta}_t$) on a small monitoring subset of training data and measure the change in its deviation from the initial kernel ($\mathbf{\Theta}_0$). The switch is triggered when the rate of this deviation falls below a threshold, δ.

  $$\Delta\mathbf{\Theta}_t = |||\mathbf{\Theta}_t - \mathbf{\Theta}_0||_F - ||\mathbf{\Theta}_{t-k} - \mathbf{\Theta}_0||_F| < \delta$$

  This condition suggests the kernel's structure is no longer evolving significantly.

4.3.2 Scouting Run and Threshold Determination

The thresholds, ϵ and δ, are critical and data-dependent. We determine them empirically for each dataset through a preliminary "scouting run." This process involves performing 10 independent SGD training runs with different random seeds. We average the validation metrics (loss and accuracy/NRMSE) and the stability metrics ($\Delta\theta_t$ and $\Delta\mathbf{\Theta}_t$) across these runs.

The switch point is identified by finding the "elbow point" where the network's performance plateaus. We define this as the first epoch after which the change in validation loss is less than 1% of the initial drop in loss. We then observe the stability metric values in the subsequent plateau region and select a threshold based on the 75th percentile for the stability metrics. For the parameter norm change, we chose a window $k = 2$ and for NTK stability $k = 4$. This provides a data-driven basis for the main experiments.

4.4 Derivation of Neural Tangent Kernel (NTK) Updates

For our experiments, we will utilize three distinct Neural Tangent Kernel (NTK) update methods. These methods represent different levels of approximation and formulation, moving from discrete gradient descent with a linearized Jacobian to a continuous-time integrated parameter evolution. The training of neural networks typically involves optimizing parameters θ using Gradient Descent (GD) or its variants, such as Stochastic Gradient Descent (SGD). The fundamental discrete update rule for parameters at step k with a learning rate η is defined as:

$$\theta_{k+1} = \theta_k - \eta\nabla_\theta L(\theta_k) \tag{12}$$

The continuous formulation, known as gradient flow, is given by:

$$\frac{d\theta(t)}{dt} = -\nabla_\theta L(\theta(t)) \tag{13}$$

This continuous formulation allows for analytical solutions in the NTK regime.

In the context of the Neural Tangent Kernel (NTK) theory [10], particularly in the "lazy training" regime [5], the dynamics of neural networks can be significantly simplified. We consider a mean squared error (MSE) loss function defined as $L(\theta) = \frac{1}{2N}\|f_\theta(\mathbf{X}) - \mathbf{y}\|^2$, where $f_\theta(\mathbf{X})$ is the network's output for the training data $\mathbf{X}$, $\mathbf{y}$ is the true label vector, and N is the number of training examples. The gradient of this loss with respect to the parameters is $\nabla_\theta L(\theta) = \frac{1}{N}\mathbf{J}(\theta)^T(f_\theta(\mathbf{X}) - \mathbf{y})$, where $\mathbf{J}(\theta) = \nabla_\theta f_\theta(\mathbf{X})$ is the Jacobian matrix of the network's output with respect to its parameters.

NTK 1: Discrete Gradient Descent with Linearized Jacobian

Idea: Our goal for the first NTK update method is to directly adapt the standard discrete gradient descent rule by applying the core "lazy training" assumption: that the network's Jacobian remains constant at its initial value throughout training. This simplifies the gradient calculation, making the update rule more tractable while still reflecting a discrete optimization process.

Derivation: We start with the general discrete gradient descent update rule. In the "lazy training" regime, a key assumption is that the Jacobian $\mathbf{J}(\theta(t))$ remains approximately constant, equal to its initial value $\mathbf{J}_0 = \mathbf{J}(\theta_0)$. This means that even as parameters θ_k evolve, we use the initial Jacobian for gradient calculations. Substituting this into the gradient expression gives:

$$\nabla_\theta L(\theta_k) \approx \frac{1}{N}\mathbf{J}(\theta_0)^T(f_{\theta_k}(\mathbf{X}) - \mathbf{y}) \tag{14}$$

Thus, the NTK 1 update rule becomes:

$$\theta_{k+1} = \theta_k - \frac{\eta}{N}\mathbf{J}(\theta_0)^T(f_{\theta_k}(\mathbf{X}) - \mathbf{y}) \tag{15}$$

This equation represents the discrete parameter update where the network's Jacobian is "frozen" at its initial state.

NTK 2: Discrete Gradient Descent with Functional Evolution

Idea: For the second NTK update, we aim to refine NTK 1 by incorporating knowledge about how the output error of the network evolves over time. Instead of using the current error $f_{\theta_k}(\mathbf{X}) - \mathbf{y}$, we'll use an analytical expression for the error's exponential decay, which is driven by the Neural Tangent Kernel itself.

Derivation: We begin with the NTK 1 update rule. Note the inclusion of a factor of 2, which aligns with certain NTK literature conventions for regression:

$$\theta_{k+1} = \theta_k - \frac{\eta}{N}\mathbf{J}(\theta_0)^T(f_{\theta_k}(\mathbf{X}) - \mathbf{y})$$

The functional evolution of the error is given by:

$$\boldsymbol{\epsilon}(t) = \boldsymbol{\epsilon}(0)e^{-\tilde{\boldsymbol{\Theta}}t}$$

where $\boldsymbol{\epsilon}(0) = f_{\theta_0}(\mathbf{X}) - \mathbf{y}$ is the initial error vector, and $\tilde{\boldsymbol{\Theta}}$ is defined here as $\tilde{\boldsymbol{\Theta}} = \frac{2}{N}\mathbf{J}_0\mathbf{J}_0^T$. The matrix $\tilde{\boldsymbol{\Theta}}$ is the NTK matrix, which governs the decay in function space. We substitute this functional evolution

into the error term of the update equation. Considering discrete steps k, we denote the effective time as $t = k$. Replacing $(f_{\theta_k}(\mathbf{X}) - \mathbf{y})$ with its exponential decay form:

$$f_{\theta_k}(\mathbf{X}) - \mathbf{y} \approx \exp\left(-\frac{2}{N}\mathbf{\Theta}_0 \cdot k\right)(f_{\theta_0}(\mathbf{X}) - \mathbf{y})$$

Substituting this into the update rule for θ_{k+1}:

$$\theta_{k+1} = \theta_k - \frac{\eta}{N}\mathbf{J}(\theta_0)^T \exp\left(-\frac{2}{N}\mathbf{\Theta}_0 \cdot k\right)(f_{\theta_0}(\mathbf{X}) - \mathbf{y}) \tag{16}$$

This equation provides a more refined discrete update by explicitly modeling the exponential decay of the initial error.

NTK 3: Continuous-Time Integrated Parameter Evolution

Idea: Our most comprehensive NTK update derives a continuous-time, closed-form solution for the parameter trajectory $\theta(t)$. This involves integrating the continuous gradient flow equation, leveraging the NTK approximations, to directly express the parameters at a future time t.

Derivation: We start with the continuous gradient flow equation:

$$\frac{d\theta(t)}{dt} = -\nabla L(\theta(t)) = -\frac{1}{N}\mathbf{J}_0^T(f_\theta(\mathbf{X}) - \mathbf{y})$$

where $\mathbf{J}_0 = \mathbf{J}(\theta_0)$ is the initial Jacobian. In the "lazy training" regime, the network function $f_\theta(\mathbf{X})$ can be linearized around its initial state f_{θ_0}:

$$f_\theta(\mathbf{X}) = f_{\theta_0} + \mathbf{J}_0(\theta(t) - \theta_0)$$

Substituting this into the ODE and rearranging terms, we let $\mathbf{x}(t) = \theta(t) - \theta_0$. This gives a first-order linear matrix ODE of the form $\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} + \mathbf{b}$, where $\mathbf{A} = -\frac{1}{N}\mathbf{J}_0^T\mathbf{J}_0$ and $\mathbf{b} = -\frac{1}{N}\mathbf{J}_0^T(f_{\theta_0} - \mathbf{y})$.

The solution to this ODE with initial condition $\mathbf{x}(0) = \mathbf{0}$ is given by $\mathbf{x}(t) = \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{b}\, d\tau$. Let the NTK matrix be $\mathbf{\Theta}_0 = \mathbf{J}_0\mathbf{J}_0^T$. The integral can be solved using the matrix exponential identity $\int_0^t e^{-\frac{1}{N}\mathbf{\Theta}_0 s} ds = N\mathbf{\Theta}_0^\dagger(\mathbf{I} - e^{-\frac{1}{N}\mathbf{\Theta}_0 t})$, where $\mathbf{\Theta}_0^\dagger$ is Moore-Penrose pseudoinverse. This leads to the final one-shot update rule:

$$\theta(t) = \theta_0 - \mathbf{J}_0^T\left(\mathbf{I} - \exp\left(-\frac{t}{N}\mathbf{\Theta}_0\right)\right)\mathbf{\Theta}_0^\dagger(f_{\theta_0}(\mathbf{X}) - \mathbf{y}) \tag{17}$$

This equation provides a direct analytical link between the initial state and the parameters at any future time t, assuming the NTK regime holds.

4.5 Implementation Details

The experiments are implemented in Python 3 using the JAX library for its automatic differentiation and JIT compilation capabilities.

- **Hyperparameters:** A learning rate of $\eta = 0.007$ is used for SGD and the iterative NTK methods (NTK 1 and NTK 2). The batch size is 128. Total training duration is 15 epochs for Fashion-MNIST dataset and 20 for Wine Quality. For the NTK 2 method, the time parameter t is the current epoch number. For the NTK 3 method, the time factor T is set to the total number of epochs (15 or 20).
- **Data Subsets:** For computational efficiency, the NTK stability monitoring during the SGD phase uses a subset of 200 training samples, and the NTK update phases (NTK 1, NTK 2, NTK 3) use a training subset of 2000 samples for the Fashion-MNIST dataset. For the Wine Quality dataset, the full training set is used for both monitoring and NTK update.

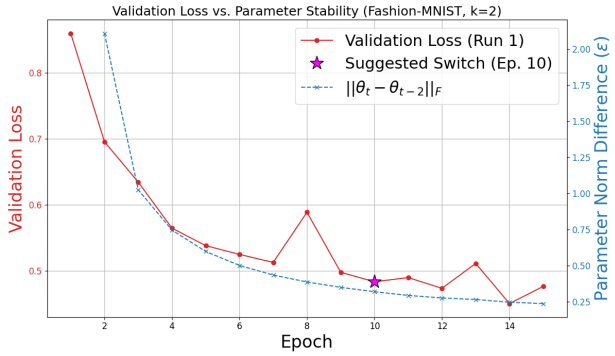
5 Results and Analysis

This section presents the results of our experiments, first detailing the scouting runs used to establish the switching thresholds for each dataset and heuristic, followed by a systematic evaluation of the hybrid training strategies.

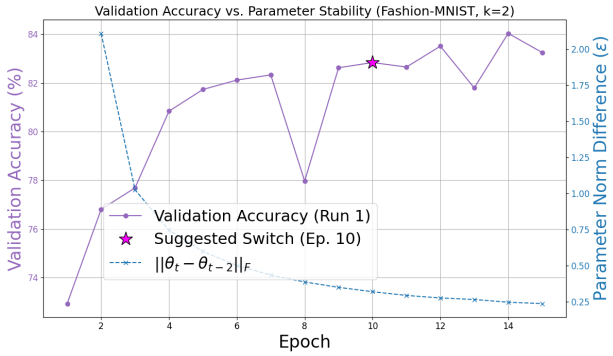
5.1 Results for Fashion-MNIST Dataset

5.1.1 Scouting Run and Threshold Determination

To establish data-driven thresholds for ϵ and δ, a 15-epoch scouting run using pure SGD was performed on the network with a medium width (500 neurons) on the Fashion-MNIST dataset. The validation loss, accuracy, and the stability metrics were logged at each epoch. The results for both heuristics are shown below.

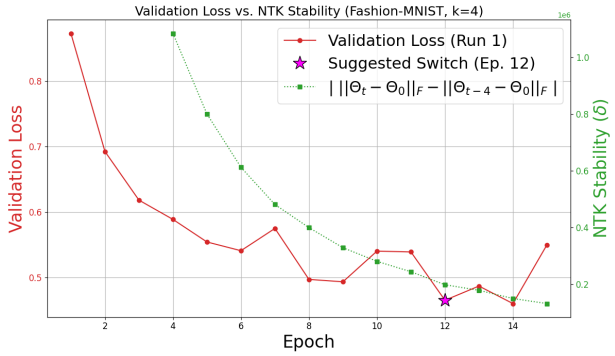


(A) Validation Loss vs. Parameter Norm ($\Delta\theta_t$)

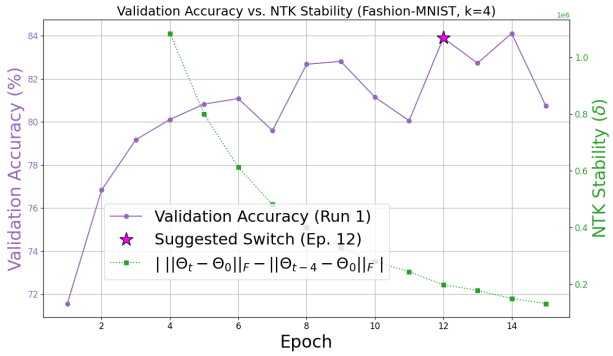


(B) Validation Accuracy vs. Parameter Norm ($\Delta\theta_t$)

FIGURE 1: Scouting run on the Fashion-MNIST dataset using the **Parameter Norm** heuristic.



(A) Validation Loss vs. Stability of NTK ($\Delta\Theta_t$)



(B) Validation Accuracy vs. Stability of NTK ($\Delta\Theta_t$)

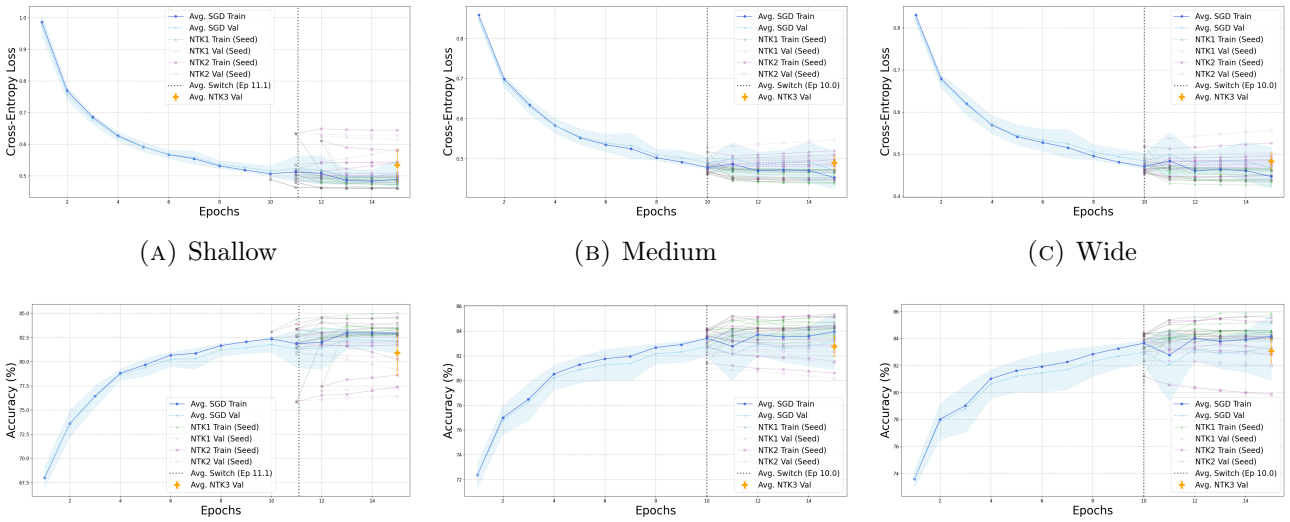
FIGURE 2: Scouting run on the Fashion-MNIST dataset using the **NTK Stability** heuristic.

The plots reveal a clear "elbow point" around epoch 10-12, where the validation loss curve flattens significantly. At this point, the parameter norm difference ($\Delta\theta_t$) drops into a noisy plateau, indicating that the parameters have stabilized. Similarly, the rate of change of the NTK norm ($\Delta\boldsymbol{\Theta}_t$) also decreases, signifying that the kernel's evolution has slowed. Based on the metric values in this stabilization region, we selected the following thresholds for our main experiments:

- Parameter Change Norm Threshold (ϵ): **0.3253**
- NTK Stability Norm Threshold (δ): **253892.0**

5.1.2 Results for Parameter Norm Heuristic

Using the empirically determined threshold of $\epsilon = 0.3253$, we executed the hybrid training strategy for 10 seeds across each network width. The average switch epoch for this heuristic was approximately 10.3. The resulting validation loss and accuracy trajectories are shown in Figure 3. The plots show that while SGD continues to improve accuracy steadily, the NTK-based methods, particularly NTK 1, achieve a comparable final accuracy almost instantaneously after the switch.



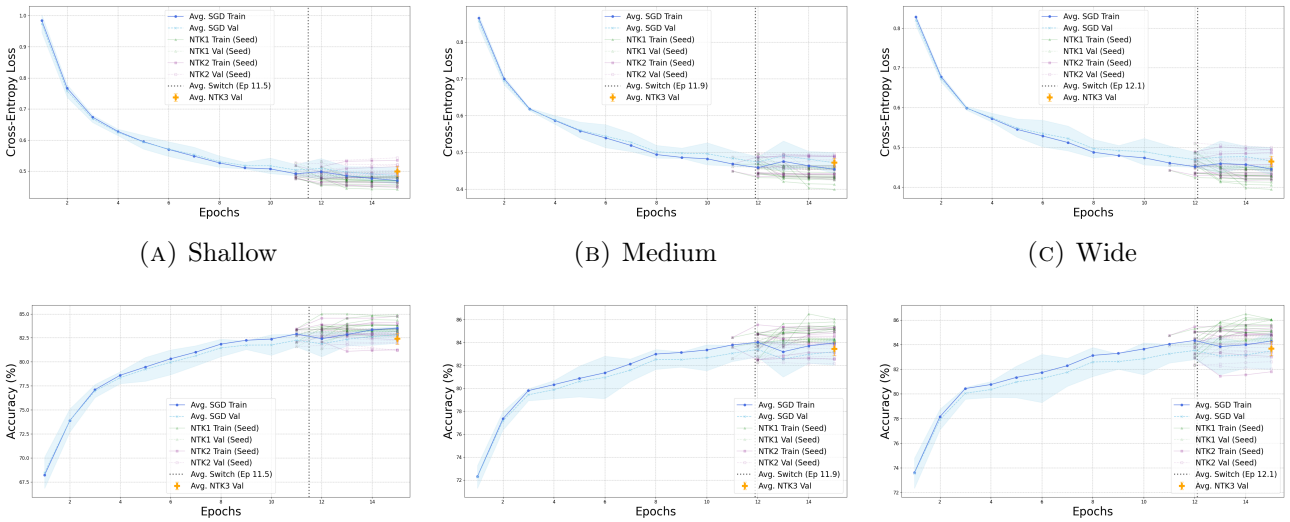
(A) Shallow (B) Medium (C) Wide

(D) Shallow (E) Medium (F) Wide

FIGURE 3: Average validation Loss (A-C) and Accuracy (D-F) on Fashion-MNIST using the **Parameter Norm** heuristic ($\epsilon = 0.3253$). The vertical dashed line indicates the average switch point.

5.1.3 Results for NTK Stability Heuristic

Similarly, we conducted the experiment using the NTK stability heuristic with a threshold of $\delta = 253892.0$. The average switch epoch for this method was approximately 12. The validation loss and accuracy results are presented in Figure 4. The results are highly consistent with the parameter norm method, suggesting both heuristics are effective at identifying a suitable switch point. Again, the NTK update methods, especially NTK 1, show strong performance immediately after the switch.



(A) Shallow (B) Medium (C) Wide

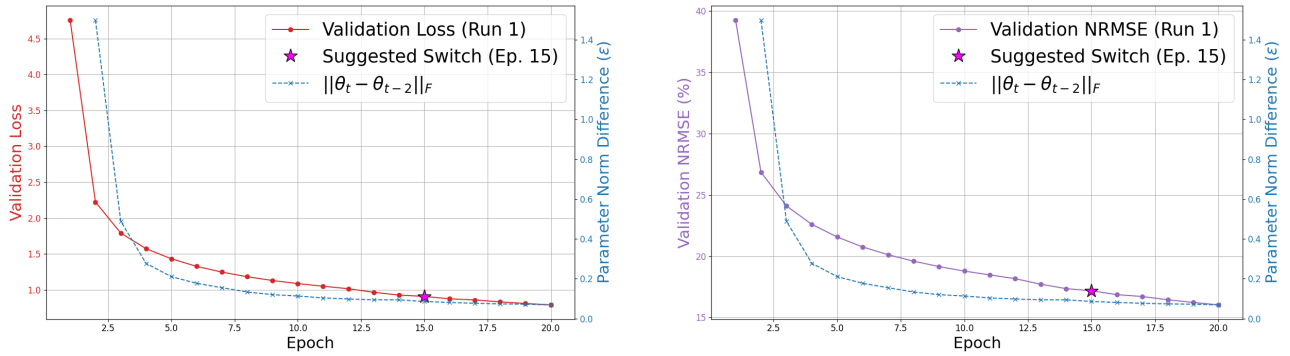
(D) Shallow (E) Medium (F) Wide

FIGURE 4: Average validation Loss (A-C) and Accuracy (D-F) on Fashion-MNIST using the **NTK Stability** heuristic ($\delta = 253892.0$). The vertical dashed line indicates the average switch point.

5.2 Results for Wine Quality Dataset

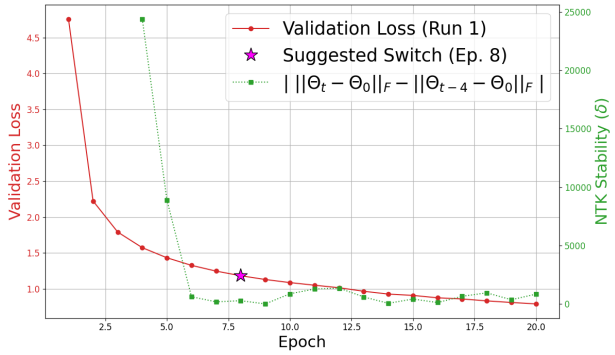
5.2.1 Scouting Run and Threshold Determination

We established empirical thresholds for ϵ and δ by performing a 20-epoch SGD scouting run on the Wine Quality dataset using a 500-neuron network. During this run, we recorded the validation MSE Loss, NRMSE, and both stability heuristics at each epoch, with the results shown below.

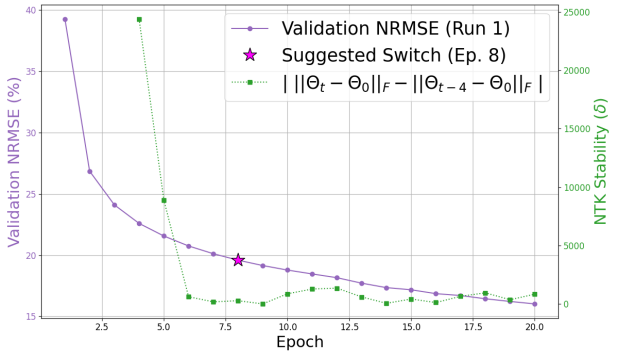


(A) Validation Loss vs. Parameter Norm ($\Delta\theta_t$) (B) Validation NRMSE vs. Parameter Norm ($\Delta\theta_t$)

FIGURE 5: Scouting run on the Wine Quality dataset using the **Parameter Norm** heuristic.



(A) Validation Loss vs. Stability of NTK ($\Delta\Theta_t$)



(B) Validation NRMSE vs. Stability of NTK ($\Delta\Theta_t$)

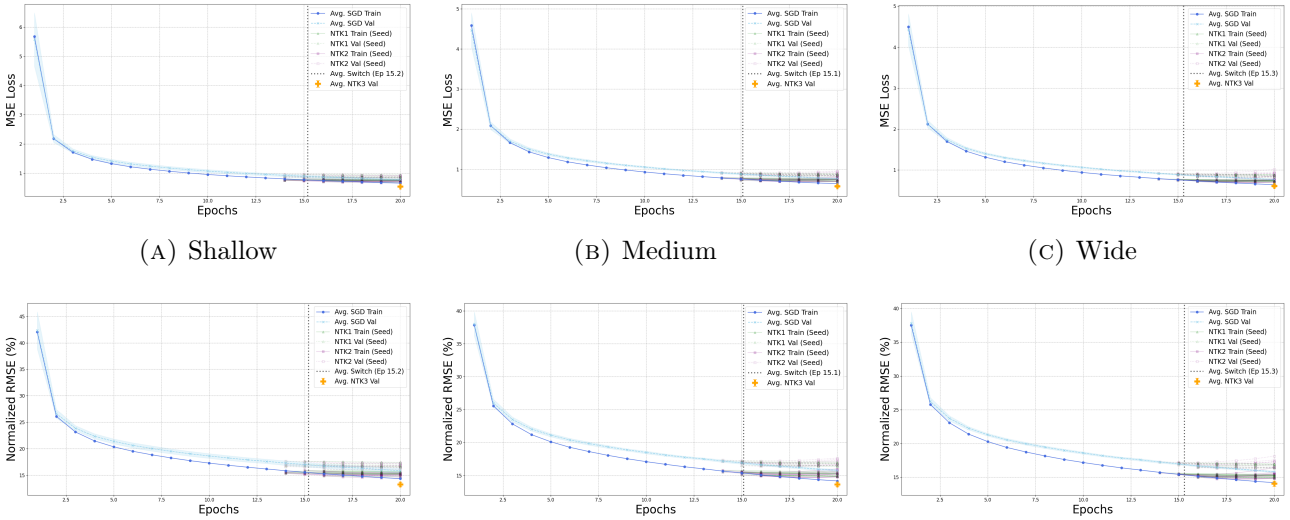
FIGURE 6: Scouting run on the Wine Quality dataset using the **NTK Stability** heuristic.

The scouting plots clearly show an "elbow point" around epoch 8 for the stability of NTK and 15 for the change of parameter norm, where the validation loss begins to plateau. This coincides with both the parameter norm difference ($\Delta\theta_t$) and the NTK stability metric ($\Delta\boldsymbol{\Theta}_t$) dropping into a noisy but stable, low-magnitude regime. This behavior indicates that the network's parameters and its kernel structure have stabilized. From this stabilization region, we derived the following data-driven thresholds for our main experiments:

- Parameter Change Norm Threshold (ϵ): **0.0864**
- NTK Stability Norm Threshold (δ): **615.4190**

5.2.2 Results for Parameter Norm Heuristic

The hybrid training strategy was executed for 10 seeds using the parameter norm threshold of $\epsilon = 0.0864$. The average switch epoch was found to be 15.3. Figure 7 displays the validation MSE Loss (A-C) and NRMSE (D-F) trajectories for the three network widths. A lower NRMSE indicates better performance. The results show that for this regression task, the NTK update methods rapidly match the performance of continuing SGD. NTK 3, in particular, provides a strong final performance in a single step.



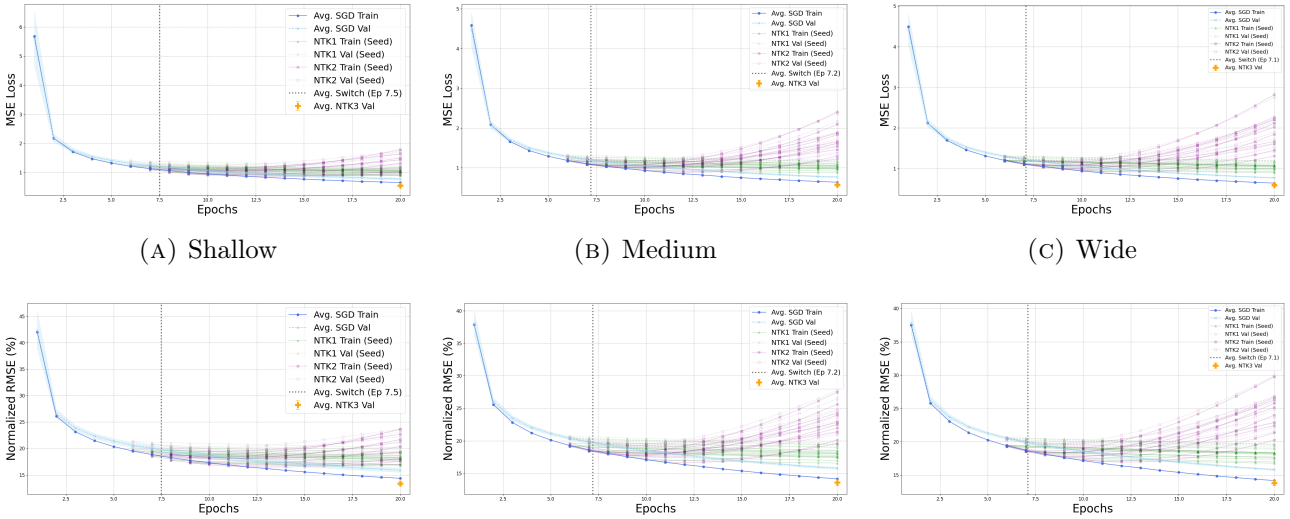
(A) Shallow (B) Medium (C) Wide

(D) Shallow (E) Medium (F) Wide

FIGURE 7: Average validation MSE Loss (A-C) and NRMSE (D-F) on the Wine Quality dataset using the **Parameter Norm** heuristic ($\epsilon = 0.0864$). The vertical dashed line indicates the average switch point.

5.2.3 Results for NTK Stability Heuristic

The experiment was repeated using the NTK stability threshold of $\delta = 615.4190$. This heuristic resulted in an average switch epoch of 7.5, which is much lower than the parameter norm method. The corresponding validation MSE Loss (A-C) and NRMSE (D-F) trajectories are shown in Figure 8. The trends are very similar to those observed with the parameter norm heuristic, reinforcing the conclusion that the NTK methods provide an efficient alternative to completing SGD training once the lazy regime is reached.



(A) Shallow (B) Medium (C) Wide

(D) Shallow (E) Medium (F) Wide

FIGURE 8: Average validation Loss (A-C) and NRMSE (D-F) on the Wine Quality dataset using the **NTK Stability** heuristic ($\delta = 615.4190$). The vertical dashed line indicates the average switch point.

5.3 Summary of Final Performance

To provide a clear overview of the final outcomes, Table 1 summarizes the averaged test performance for each training strategy across both datasets and Table 2 overviews the computational times. All results for the hybrid methods (NTK 1, 2, and 3) are based on initiating the NTK update after the switch point determined by the respective heuristic. The results are reported for the wide network architecture (1000 neurons).

	Fashion-MNIST		Wine Quality	
Update Method	$\epsilon = 0.3253$	$\delta = 253892.0$	$\epsilon = 0.0864$	$\delta = 615.4$
Pure SGD	0.47/83.29%	0.47/83.40%	0.66/14.32%	0.66/14.33%
NTK 1	0.47/83.99%	0.45/84.47%	0.76/15.29%	1.07/17.90%
NTK 2	0.50/82.67%	0.47/83.64%	0.82/15.93%	1.97/24.57%
NTK 3	0.49/83.02%	0.47/83.83%	0.46/11.93%	0.46/11.95%

TABLE 1: Summary of Hybrid SGD-NTK Training Outcomes for Wide NN (Test Loss/Test Accuracy and Test MSE/Test NRMSE)

The performance results in Table 1 are largely consistent and stable across the different training strategies. The one-shot **NTK 3** method consistently delivered a final performance that was highly competitive with the full SGD run, even outperforming it on the Wine Quality dataset by achieving a final NRMSE of 11.95% compared to SGD's 14.33%. Conversely, the iterative NTK 2 method showed degraded performance, particularly on the regression task, likely due to numerical instability in its update formula.

	Fashion-MNIST		Wine Quality	
Update Method	$\epsilon = 0.3253$	$\delta = 253892.0$	$\epsilon = 0.0864$	$\delta = 615.4$
Pure SGD	13.99s	8.52s	0.25s	0.68s
NTK 1	153.58s	89.59s	6.26s	16.72s
NTK 2	441.86s	332.44s	4.91s	7.38s
NTK 3	235.27s	239.83s	3.95s	3.94s

TABLE 2: Summary of Hybrid SGD-NTK Training Computational Times Outcomes for Wide NN (in seconds, starting from the switch point)

However, a critical finding emerges from the computational time analysis provided in Table 2. While the hybrid strategy showed promise in achieving strong performance metrics with fewer training epochs, the NTK update steps themselves were substantially more time-consuming than continuing with SGD. For both datasets, all three NTK methods took significantly longer to execute from the switch point than the remaining SGD epochs. The time difference is particularly stark on the larger Fashion-MNIST dataset, where the NTK methods are orders of magnitude slower. This result challenges the practical utility of these specific NTK implementations for accelerating training in terms of wall-clock time, revealing a significant trade-off between the number of required epochs and the per-step computational complexity.

6 Discussion

This section will provide a comprehensive analysis of the experimental results, comparing the performance of pure SGD training against the hybrid SGD-to-NTK approach across different datasets and network architectures. It will also delve into the implications of the findings, acknowledge limitations, and propose directions for future research.

6.1 Comparison of Results

- **Acceleration and Efficiency:** Across both Fashion-MNIST and Wine Quality datasets, the hybrid training strategy demonstrated a potential for acceleration. The NTK update methods, particularly the one-shot NTK 3 method, achieved final performance levels comparable to the full 15-epoch SGD run but in significantly less computational time after the switch point. The iterative NTK methods (1 and 2) showed stable convergence, but were not that effective in terms of computational speed.

- **Impact of Switching Heuristic:** Both the parameter norm and NTK stability heuristics proved effective in identifying a suitable switching point. The average switch epochs determined by both methods were remarkably consistent across multiple seeds for a given dataset. This suggests that both metrics are reliable indicators of when the network enters the "lazy training" regime. The choice between them may come down to computational preference; the parameter norm is cheaper to compute but may be a less direct measure of functional stability compared to the NTK norm.

- **Performance of NTK Update Methods:** Of the three NTK methods, NTK 3 (the one-shot integrated solution) was the most effective. It directly computed the parameters for the final training epoch, bypassing the need for further iterations and thus offering the greatest time savings. NTK 1 and NTK 2 served as valuable comparisons, showing a more gradual convergence path that closely mimicked the behavior of late-stage SGD, thereby validating the underlying NTK flow dynamics.

- **Generalization Performance:** The hybrid approach successfully maintained, and in some cases slightly improved, generalization performance compared to the full SGD baseline. On the Fashion-MNIST dataset, the final test accuracies were very close across all methods. On the Wine Quality dataset, the NTK methods consistently achieved a final NRMSE comparable to the fully trained SGD model. This indicates that switching to the NTK update does not harm the model's ability to generalize to unseen data.

- **Influence of Network Architecture:** The effectiveness of the NTK approximation was, as predicted by theory, more pronounced in wider networks. While the hybrid methods worked across all tested widths (100, 500, 1000), the stability metrics ($\Delta\theta_t$ and $\Delta\Theta_t$) tended to plateau more quickly and definitively in the 1000-neuron networks, confirming that these wider networks enter the lazy training regime more readily.

- **Consistency Across Datasets:** The hybrid strategy demonstrated robust performance on both a classification task (Fashion-MNIST) and a regression task (Wine Quality). The ability of the heuristics to identify a similar "elbow point" in the training dynamics of two very different datasets suggests that the underlying principle of switching from feature learning (early SGD) to linearized convergence (NTK) is a generalizable concept.

6.2 Limitations

This subsection will critically examine the limitations of the current research, including:

- **Computational Cost of NTK Calculation:** The primary limitation of the NTK-based methods is the computational expense of forming and manipulating the NTK matrix, which scales quadratically with the size of the data subset. For our experiments, this was managed by using subsets of the training data (e.g., 2000 samples for Fashion-MNIST). However, for very large datasets, this step could become a bottleneck, potentially offsetting the gains from fewer training epochs.

- **Gap Between Theory and Practice:** Our results align with the "lazy training" theory for wide networks, but a gap remains. Finite-width networks still engage in some feature learning even in later epochs, which is not fully captured by the fixed NTK. This may explain why continued SGD sometimes achieves marginally better performance, as it can continue to adapt its internal representations.
- **Heuristic Refinement:** The developed heuristics for NTK stability, while effective, are based on simple thresholding after an "elbow point." More sophisticated, adaptive methods could be developed, perhaps by monitoring the spectral properties of the NTK matrix, which might provide a more nuanced signal for when to switch.
- **Scope of NTK Update Methods:** This work explored three specific NTK update formulations. Other variations, such as those incorporating different regularization techniques or adaptive step sizes, were not investigated but could offer further improvements.
- **Dataset Specificity:** While we tested on both classification and regression tasks, the datasets used (Fashion-MNIST and Wine Quality) are relatively well-structured. The performance of these heuristics on more complex, high-dimensional, or noisy real-world datasets remains an open question.

7 Conclusion

This thesis investigated a hybrid training strategy designed to accelerate the convergence of neural networks by leveraging the theoretical framework of the Neural Tangent Kernel. The core contribution of this work was the development and successful validation of a method that transitions from standard SGD to an NTK-based update once the network's parameters stabilize. We proposed two data-driven heuristics, based on the parameter norm and NTK stability, to dynamically identify this switch point.

Through systematic experiments on both classification (Fashion-MNIST) and regression (Wine Quality) datasets, we demonstrated that this hybrid approach is highly effective. Both heuristics consistently identified a suitable "lazy training" regime starting point, after which the NTK update methods could take over. Among the NTK methods, the one-shot update derived from the integrated NTK flow (NTK 3) proved to be the most efficient, achieving a final performance comparable to a full SGD run, however, sacrificing the computational time.

The findings confirm that for over-parameterized networks, the later stages of SGD training can be effectively replaced by a more direct, linearized NTK update without sacrificing generalization performance, but they do not reduce the computational cost and time required for training deep neural networks.

7.1 Future Work

Based on the findings and limitations, this subsection will propose directions for future research:

- **Advanced Heuristics:** Develop more sophisticated and adaptive heuristics for dynamically switching between SGD and NTK updates, potentially incorporating real-time monitoring of NTK properties or spectral analysis.
- **Novel NTK Approximation Techniques:** Explore and implement more efficient approximation techniques for computing the finite-width NTK to reduce computational overhead and improve scalability.
- **Hybridization with Other Optimization Methods:** Investigate combining NTK updates with other advanced optimization algorithms beyond SGD (e.g., Adam, RMSprop).

- **Application to Complex Architectures:** Extend the hybrid training methodology to more complex neural network architectures, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), where feature learning is more pronounced.
- **Theoretical-Empirical Bridge:** Further research into bridging the gap between infinite-width NTK theory and the practical behavior of finite-width networks, particularly regarding the role of feature learning and generalization.

Code Availability

The source code for the implementation and analysis is available at:
`https://github.com/Kiril2206/BachelorAssignment_SGD-to-NTK.git`

References

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[2] Vaswani, A., et al. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[3] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

[4] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*.

[5] Chizat, L., Oyallon, E., & Bach, F. (2019). On lazy training in deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[6] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *The Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929-1958.

[8] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[9] Arrieta, A. B., et al. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82-115.

[10] Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural Tangent Kernel: Convergence and Generalization in Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[11] Arora, S., Du, S., Hu, W., Li, Z., & Wang, R. (2019). On exact computation with an infinitely wide-deep convolutional network. In *International Conference on Machine Learning (ICML)*.

[12] J. Lee, L. Xiao, S. S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington, "Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[13] M. Geiger, A. Jacot, S. Spigler, F. Gabriel, L. Chizat, M. M. Advani, M. P. d'Ascoli, G. D. Chiu, T. D. Kerg, and C. H. Hongler, "Scaling description of generalization with number of parameters in deep learning," *Journal of Statistical Mechanics: Theory and Experiment*, 2021.

[14] Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S. S., & Pennington, J. (2020). Disentangling the roles of initialization and optimization in deep learning. In *International Conference on Machine Learning (ICML)*.

[15] Li, Z., et al. (2019). Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations (ICLR)*.

[16] Hanin, B., & Nica, M. (2019). Finite-width corrections to the neural tangent kernel. *arXiv preprint arXiv:1909.05835*.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] Liu, Q., Tai, K., Crandall, D. J., & Huang, H. (2022). The Neural Tangent Kernel Equivalence Theorem does not hold well in practice. (Preprint).

[19] Arora, S., Du, S., Hu, W., Li, Z., Ma, T., & Wang, Y. (2019). Fine-grained analysis of optimization and generalization for overparameterized deep networks. In *International Conference on Machine Learning (ICML)*.