# From Theory to Implementation: Gradient Descent Optimization for Over-Parameterized Neural Networks

Yuanxin Zhang (yz6201) - *Mathematics*

**Abstract**

In deep learning training, a common phenomenon occurs where randomly initialized first-order methods (such as gradient descent) can achieve zero training loss, even when the objective function is non-convex and non-smooth. In our paper, we demonstrate this phenomenon using a two-layer fully connected Relu-activated neural network. We specifically choose the Relu-activated function for its non-convex and non-smooth properties. Additionally, we introduce variables to represent the number of hidden nodes ($m$) and the number of training data points ($n$), while utilizing the quadratic loss function. In this report, we first prove that if no two inputs are parallel and there is sufficient overparameterization (i.e., a large enough $m$), randomly initialized gradient descent causes the quadratic loss function to converge to a global minimum at a linear convergence rate. Next, we perform numerical experiments to validate our theoretical proof, and finally, we discuss potential improvements to the current experiment on a broader scale. Our research is mainly inspired by Simon's research, but have a further implementation based on his proof. [1]

## 1 Introduction

The field of deep learning has witnessed a remarkable phenomenon in training neural networks using first-order methods, particularly gradient descent. Surprisingly, even when dealing with non-convex and non-smooth objective functions, these randomly initialized methods have been observed to achieve zero training loss. This intriguing behavior challenges the previous assumption that the over-parameterization of neural networks is solely responsible for their ability to fit all training labels.

The traditional analysis employed in convex optimization fails to address the challenges posed by non-smooth and non-convex optimization objective functions. Consequently, alternative approaches are required to understand and explain the phenomenon observed in deep learning training.

In light of this, our research aims to investigate and provide insights into this intriguing phenomenon. Specifically, we focus on understanding why randomly initialized first-order methods, such as gradient descent, can achieve zero training loss despite the presence of non-convex and non-smooth objective functions. By delving into the behavior of these methods, we seek to bridge the gap between the existing assumptions and the observed phenomenon, offering a deeper understanding of the underlying mechanisms at play.

In the following sections, we will present our comparative analysis, followed by the theoretical proof of convergence for randomly initialized gradient descent. Subsequently, we will describe the numerical experiments conducted to validate our theoretical claims. Finally, we will discuss the implications of our research and propose avenues for further improvement in optimizing the training of over-parameterized neural networks using gradient descent optimization.

By addressing the limitations of traditional analysis techniques and exploring the perplexing behavior of randomly initialized first-order methods, we aim to contribute to the advancement of deep learning theory and provide valuable insights for practitioners in this rapidly evolving domain.

## 2 State-of-the-art results

In this section, we compare our findings with the state-of-the-art approaches in the field. We consider various analysis techniques that have been employed to study the convergence and dynamics of first-order methods in non-convex optimization landscapes.

**landscape analysis:** This approach aims to identify the geometric properties of the optimization landscape. Previous work in this area has established the following theorem: if the objective function is smooth and all local minima are global, and for every saddle point, there exists a negative curvature, then stochastic gradient

descent (SGD) can find a global minimum in polynomial time. While this result provides valuable insights into the behavior of SGD, it does not directly address the specific phenomenon of randomly initialized first-order methods achieving zero training loss in non-convex and non-smooth objective functions. [2]

**Analysis of algorithm dynamics:** convergence results are obtained by analyzing the dynamics of first-order methods. In this context, a theorem has been established under the assumptions of Gaussian input distribution and labels generated according to a planted neural network. It has been shown that randomly initialized (stochastic) gradient descent can successfully learn various structures, including a ReLU, a single convolutional filter, a convolutional neural network with one filter and one output layer, and a residual network with a small spectral norm weight matrix. These findings demonstrate the effectiveness of randomly initialized first-order methods in fitting over-parameterized neural networks with specific architectures. [3]

**Other analysis techniques:** In addition to the aforementioned approaches, researchers have explored other analysis techniques. For instance, optimal transport theory has been applied to analyze continuous-time gradient descent on over-parameterized models. However, the results in this context are limited to the formal level, with the requirement of the second layer being infinitely wide for their findings on ReLU-activated neural networks. [4] Furthermore, some studies have analyzed SGD for optimizing population loss and have shown that the dynamics can be captured by a partial differential equation in a suitable scaling limit. However, it remains unclear why first-order methods, specifically, can effectively minimize the empirical risk.

While these state-of-the-art approaches have contributed valuable insights into the behavior of first-order methods in non-convex optimization, they do not fully explain the specific phenomenon of randomly initialized first-order methods achieving zero training loss in non-convex and non-smooth objective functions. Therefore, our research aims to bridge this gap by providing a novel theoretical framework and conducting numerical experiments specifically focused on understanding and verifying the convergence behavior of randomly initialized gradient descent for over-parameterized neural networks.

In the next sections, we will present our theoretical proof and numerical experiments, which not only provide empirical evidence but also offer a deeper understanding of the observed phenomenon. By investigating the convergence behavior of randomly initialized gradient descent and verifying our theoretical claims, we aim to contribute to the advancement of deep learning theory and provide valuable insights for practitioners seeking to optimize the training process of over-parameterized neural networks using gradient descent optimization.

# 3 Methodology

## 3.1 background

First, we consider the two-layer fully connected neural networks with rectified linear unit(ReLU) activation as follows.

$$f(\boldsymbol{w}, a, x) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma(\boldsymbol{w}_r^T x) \tag{1}$$

where we $x \in \mathcal{R}^d$ is the input training data with $d$ features; $w_r \in \mathcal{R}^d$ is the weight vector of the first layer; $a_r \in \mathcal{R}$ is the output weight; $\sigma(\cdot)$ is the Relu activation function where $\sigma(z) = z$ if $z \geq 0$ and $\sigma = 0$ if $z \leq 0$. And we are using the quadratic loss function for the optimization process. Now that we have the training dataset $(x_i, y_i{}_{i=1}^n$, we aim at minimizing:

$$L(\boldsymbol{w}, a) = \sum_{i=1}^{n} \frac{1}{2} (f(\boldsymbol{w}, a, x_i) - y_i)^2 \tag{2}$$

Then, we apply randomly initialized gradient descent to optimize the first layer (The second layer is fixed).

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - \eta \frac{\partial L(\boldsymbol{w}(k), a)}{\partial \boldsymbol{w}(k)} \tag{3}$$

where $\eta > 0$ is the learning rate.

$$\frac{\partial L(\boldsymbol{w}, a)}{\partial w_r} = \frac{1}{\sqrt{m}} \sum_{i=1}^{n} (f(\boldsymbol{w}, a, x_i) - y_i) a_r x_i \mathbb{I}_{\{} w_r^T x_i \geq 0\} \tag{4}$$

In our paper, we should assume no two inputs are parallel, and in reality, this assumption is always true. Meanwhile, m ought to be close to infinite to maintain overparameterization in the neural networks. $a$ and $\boldsymbol{w}(0)$ should be randomly initialized.

## 3.2 Continuous Time Analysis

In order to understand discrete algorithms and analyze the optimization process of neural networks, continuous time analysis serves as a valuable tool. By studying the dynamics of neural network parameters over time, we can gain insights into the behavior and convergence properties of optimization algorithms.
In this context, we focus on a significant ordinary differential equation (ODE) that plays a crucial role in our analysis. The ODE is defined as:

$$\frac{d\boldsymbol{w}_r(t)}{dt} = -\frac{\partial L(\boldsymbol{w}(t), a)}{\partial w_r(t)} \tag{5}$$

Here $\boldsymbol{w}(t)$ denotes the vector of network parameters at time $t$, $\nabla_{w_r(t)} L(\boldsymbol{w}(t), a)$ is the gradient of the loss function with respect to the parameters at time $t$, and $\frac{d\boldsymbol{w}_r(t)}{dt}$ is the time derivative of the parameters.
Intuitively, the differential equation describes how the network parameters change over time as the algorithm seeks to minimize the loss function. The gradient of the loss function points in the direction of the steepest descent, so the negative gradient term represents the direction in which the parameters should be updated to minimize the loss.
By solving this ODE, the authors are able to derive a continuous-time trajectory of the network parameters, which they can use to analyze the convergence properties of the algorithm.
Next, we will define $u_i(t) = f(\boldsymbol{w}(t), a, x_i)$ the predicted value in terms of input data $x_i$ at time $t$ and we let $\mathbf{u}(t) \in \mathcal{R}^n$ be the predicted vector at time $t$.
First, We define the Gram matrix.

**Definition 3.1.** *Matrix $H^\infty \in \mathcal{R}^{n*n}$ with $H_{ij}^\infty = \mathbb{E}_{w\ N(0,I)}[\boldsymbol{x}_i^T \boldsymbol{x}_j \mathbb{I}_{\{} w^T x_i \geq 0, w^T x_j \geq 0\}]$.*

$H^\infty$ is the Gram matrix induced by the Relu activation function and the random initialization. Next, we will show how we get the definition of $H^\infty$ by calculating the dynamics of each prediction.

*Proof.* From the previous definition, we have the predicted value $u_i(t)$.

$$\frac{du_i(t)}{dt} = \sum_{r=1}^{m} \langle \frac{\partial f(\boldsymbol{w}(t), a, x_i)}{\partial w_r(t)}, \frac{dw_r(t)}{dt} \rangle = \sum_{j=1}^{n} (y_j - y_i) \sum_{r=1}^{m} \langle \frac{\partial f(\boldsymbol{w}(t), a, x_i)}{\partial w_r(t)}, \frac{\partial f(\boldsymbol{w}(t), a, x_j)}{\partial w_r(t)} \rangle = \sum_{j=1}^{n} (y_j - u_j) \boldsymbol{H}_{ij}(t) \tag{6}$$

Therefore, we have that

$$\boldsymbol{H}_{ij}(t) = \frac{1}{m} \boldsymbol{x}_i^T \boldsymbol{x}_j \sum_{r=1}^{m} \mathbb{I}_{\{} x_i^T \boldsymbol{w}_r(t) \geq 0, x_j^T \boldsymbol{w}_r(t) \geq 0\} \tag{7}$$

And we have the dynamics of predictions as :

$$\frac{d\mathbf{u}(t)}{dt} = \boldsymbol{H}(t)(\boldsymbol{y} - \mathbf{u}(t)) \tag{8}$$

□

Though the Gram matrix changed, it's still close to $\boldsymbol{H}^\infty$ in terms of the Equation 7, and the gram matrix ($\boldsymbol{H}^\infty$) is the fundamental property that determines the convergence rate.(Equation 8). Now we offer an assumption in terms of our previous definition.

**Assumption 3.2.** *$\lambda_0 \triangleq \lambda_{min}(\boldsymbol{H}^\infty) \geq 0$ for any Gram matrix $\boldsymbol{H}^\infty$, where $\lambda_0$ is the least eigenvalue of the Gram matrix*

The reason why the assumption is important is to ensure the well-posedness of the optimization problem. If the least eigenvalue is zero or negative, then the Gram matrix is not invertible, and the optimization problem doesn't have a unique solution. This means that there might be multiple local minima, or that the objective function may not be well-defined. By assuming that the least eigenvalue is strictly positive, it guarantees there is a unique global minimum that can be found by the optimization algorithm. This allows the authors to prove convergence guarantees for gradient descent in the over-parameterized setting.
For the initialization phase, we could always have if no two inputs are parallel, the least eigenvalue is strictly positive.
Now, let's conclude the main theorem in our paper.

**Theorem 3.3.** *(Convergence Rate of Gradient Flow). Suppose assumption 3.2 holds, for all* $i \in [n], \|x_i\|_2 = 1$, *and* $|y_i| \leq C$ *for some constant* $C$. *If we have the hidden nodes* $m = \Omega(\frac{n^6}{\lambda^4\sigma^3})$, *and we have i.i.d. initialize* $\boldsymbol{w}_r$ $N(0, I)$, $a_r$ $unif[-1, 1]$ *for* $r \in [m]$, *then with probability at least* $1 - \delta$ *over the initialization (which is Gaussian initialization), we have*

$$\|\boldsymbol{u}(t) - \boldsymbol{y}\|_2^2 \geq exp(-\lambda_0 t)\|\boldsymbol{u}(0) - \boldsymbol{y}\|_2^2$$

Here we normalize all input data for simplicity. And in real data, the label $y_i$s are always bounded at an interval. m is defined as $\Omega(\frac{n^6}{\lambda^4\sigma^3})$, which depends on the number of samples $n$, $\lambda_0$ and the failure probability *delta*. In general, we could say that overparameterization ($m = \text{poly}(n, \frac{1}{\lambda_0}, \frac{1}{\delta})$) guarantees that the gradient descent with random initialization has a global minimum. Meanwhile, the specific convergence rate depends on $\lambda_0$ but is not relevant to the number of hidden nodes $m$.
Now that we've already had a main theory for the phenomenon of gradient descent optimization for over-parameterized neural networks, we will show when $m \to +\infty$, (1) at initialization $\|\boldsymbol{H}(0) - \boldsymbol{H}^\infty\|_2 \to 0$ and (2) for all $t > 0$, $\|\boldsymbol{H}(t) - \boldsymbol{H}(0)\|_2 \to 0$. And in the equation 8, we've proved that $\boldsymbol{u}(t)$is correlated with $\boldsymbol{H}(t)$. Therefore, we could get $\boldsymbol{u}(t)$ is characterized by $H^\infty$.
So the following lemma shows that $\boldsymbol{H}(0)$ converges to $\boldsymbol{H}^\infty$, and $\boldsymbol{H}(0)$ has a lower bounded least eigenvalue.

**Lemma 3.4.** *If* $m = \Omega(\frac{n^2}{\lambda_0^2} log(\frac{n}{\sigma}))$, *we have with probability at least* $1-\delta$, $\|\boldsymbol{H}(0)-\boldsymbol{H}^\infty\|_2 \leq \frac{\lambda_0}{4}$ *and* $\lambda_{min}(\boldsymbol{H}(0)) \geq \frac{3}{4}\lambda_0$.

Then we show that $\forall \boldsymbol{w} \to \boldsymbol{w}(0)$, the induced Gram Matrix $\boldsymbol{H} \to \boldsymbol{H}(0)$ and has a lower bounded least eigenvalue.

**Lemma 3.5.** *If* $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$ *are i.i.d generated from N(0,I), (Gaussian initialization on* $\boldsymbol{w}$*), with probability at least* $1-\delta$, *for any set of weight vectors* $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m \in \mathbb{R}^d$ *that satisfy for any* $r \in [m]$,$\|\boldsymbol{w}_r(0)-\boldsymbol{w}_r\|_2 \leq \frac{c\sigma\lambda_0}{n^2} \triangleq R$ *for some* $c > 0$, *we have* $\|\boldsymbol{H} - \boldsymbol{H}(0)\|_2 < \frac{\lambda_0}{4}$ *and* $\lambda_{min}(\boldsymbol{H}) > \frac{\lambda}{2}$

Lemma 3.4 and lemma 3.6 denote the case that the least eigenvalue of $\boldsymbol{H}$ is lower bounded. Now let's see that if the least eigenvalue of $\boldsymbol{H}(t)$ is still lower bounded, then the loss function has a linear convergence rate, and $\boldsymbol{w}_r(t) \to \boldsymbol{w}(0), \quad \forall r \in [m]$.

**Lemma 3.6.** *If* $\forall \boldsymbol{H}(s) \in t$, $\lambda_{min}(\boldsymbol{H}(s)) \geq \frac{\lambda_0}{2}$, *then we have* $\|\boldsymbol{y} - \boldsymbol{u}(t)\|_2^2 \leq exp(-\lambda_0 t)\|\boldsymbol{y} - \boldsymbol{u}(0)\|_2^2$ *and* $\forall r \in [m]$, $\|\boldsymbol{w}_r(t) - \boldsymbol{w}_r(0)\|_2 \leq \frac{\sqrt{n}\|\boldsymbol{y}-\boldsymbol{u}(0)\|_2}{\sqrt{m}\lambda_0} \triangleq R'$

Then we will expand this lemma to the general case for all $t \geq 0$

**Lemma 3.7.** *If* $R' < R$, *we have* $\forall t > 0$, $\lambda_{min}(\boldsymbol{H}(t)) \geq \frac{1}{2}\lambda_0$, *for all* $r \in [m]$, $\|\boldsymbol{y}-\boldsymbol{u}(t)\|_2^2 \leq exp(-\lambda_0 t)\|\boldsymbol{y}-\boldsymbol{u}(0)\|_2^2$

Lemma 3.6 and 3.7 expand the proof from $\boldsymbol{H}$ to $\boldsymbol{H}(t)$ in the continuous time analysis. The four lemmas above have rigorous proof that overparameterization could guarantee gradient descent to find a global minimum. And the convergence rate only depends on $\lambda_0$ but is independent of the number of hidden nodes $m$. Next, we will verify our theory by numerical experiments.

## 3.3 Numerical Experiments Estimation

To validate and support the theoretical findings, we conducted a numerical experiment using synthetic data. This experiment aims to provide empirical evidence that aligns with the theoretical results obtained.
**Initialization:**
We begin by setting up the experiment with specific initialization parameters. The following values were used:

- Number of epochs: 100
- Learning rate: 0.1
- Number of training samples: $n = 1000$
- Number of input features: $d = 1000$
- Output labels ($y$): Generated from a one-dimensional standard Gaussian distribution

- Number of parameters: $m = 500, 1000, 2000, 4000, 8000$

First, we will verify how overparameterization will influence the convergence rates. In our theory, we think that if $m$ becomes larger, the overparameterization will be stronger, therefore, the convergence rates will be linear but faster. The code can be seen in the appendix 6.
Second, we will also verify the relation between the amount of overparameterization and the maximum distances between weight vectors and their initialization. We try to verify that at a given iteration $k$, the formula $\max_{r\in[m]} \|\boldsymbol{w}_r(k) - \boldsymbol{w}_r(0)\|_2$. This verifies lemma 3.7. The purpose of this experiment is to gain insights into the behavior of the algorithm and to validate the theoretical results presented in the paper. By analyzing the maximum distance from initialization, the authors are able to observe how many parameters move during the optimization and whether they reach a satisfactory level of accuracy.

# 4 Results

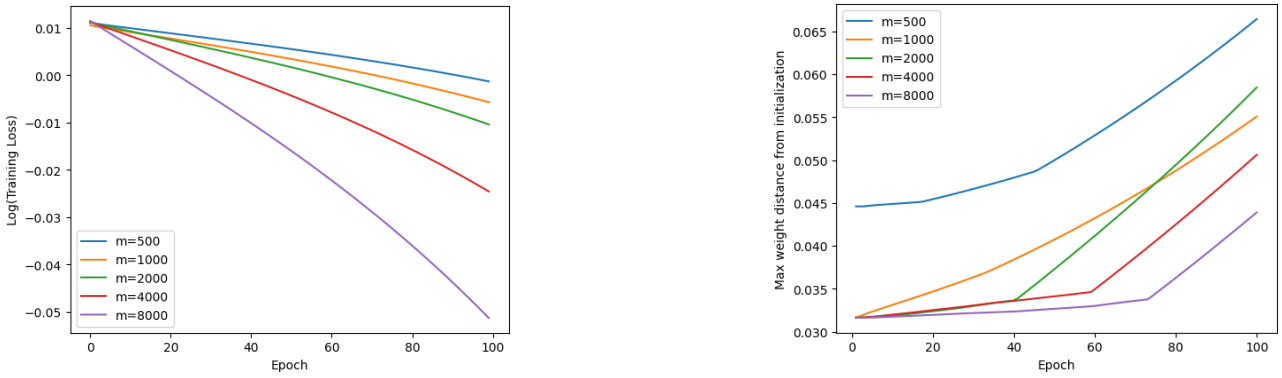


Figure 1: Numerical experiments on synthetic data

From Figure 4, we can observe that all the lines depicting the convergence process exhibit a clear decreasing trend, which aligns with the expected behavior according to machine learning training rules. Additionally, the convergence rate trend is consistently linear across different values of $m$. This observation suggests that as the number of hidden nodes $m$ increases, the convergence rate becomes faster. Consequently, our numerical experiments provide empirical evidence supporting the hypothesis that over-parameterization has a positive influence on the convergence rate of the optimization process. These findings demonstrate the potential benefits of incorporating over-parameterization techniques in machine learning models to expedite convergence and enhance training efficiency.
In Figure 4, we present an insightful observation regarding the effect of increasing the number of hidden nodes, denoted as $m$, on the behavior of the optimization process. Specifically, we examine the maximum distance from the initialization.
As we increase the value of $m$, we observe a compelling trend: Figure 4 highlights another significant finding: the maximum distance from the initialization decreases as $m$ becomes larger. This result implies that as we increase the number of hidden nodes, the weights of the neural network gradually adjust and move closer to their optimal values. The diminishing maximum distance from the initialization indicates a convergence towards a more refined and accurate solution. The decreasing maximum distance demonstrates the ability of the optimization process to effectively navigate the high-dimensional weight space and converge to a globally optimal configuration.
Overall, these results provide valuable insights into the impact of over-parameterization, represented by the increase in the number of hidden nodes, on the convergence behavior of the optimization process. The diminishing maximum distance from the initialization signifies the benefits of over-parameterization in promoting stability, robustness, and improved convergence properties in machine learning models. These findings further reinforce the notion that over-parameterization can be a beneficial technique for enhancing the performance and optimization capabilities of neural networks.

# 5 Discussion

In this project, we have explored the convergence properties of gradient descent in over-parameterized neural networks. Our key conclusion is that with over-parameterization, gradient descent provably converges to the global minimum of the empirical loss at a linear convergence rate. This finding has significant implications for the optimization process in neural networks and provides a deeper understanding of their convergence behavior. The key proof supporting our conclusion revolves around the observation that over-parameterization ensures the Gram matrix remains positive definite for all iterations during the optimization process. This property guarantees the linear convergence of gradient descent, as it indicates a well-behaved optimization landscape without spurious local minima that impede convergence. By establishing the positive definiteness of the Gram matrix, we provide theoretical evidence for the convergence of gradient descent in over-parameterized networks. Looking ahead, there are several potential future directions that can build upon our current findings:

1. Generalization of deep neural networks: While our approach has focused on shallow neural networks, it would be interesting to investigate whether similar convergence properties hold for deep neural networks. Exploring the behavior of gradient descent in deeper architectures can provide valuable insights into the optimization landscape and convergence guarantees in more complex models.

2. Refinement of the analysis: Our current analysis relies on the assumption that the Gram matrix has a lower bounded least eigenvalue. Future research could utilize advanced tools from probability theory and matrix perturbation theory to further analyze the behavior of the Gram matrix and potentially tighten the convergence rate bounds. This could lead to a better understanding of the necessary number of hidden nodes for convergence.

3. Convergence rates of accelerated methods: While gradient descent exhibits linear convergence in over-parameterized networks, accelerated optimization methods such as Nesterov acceleration or adaptive learning rate schemes could potentially further improve convergence rates. Investigating the convergence properties of these accelerated methods in the context of over-parameterization and establishing convergence rate bounds would be an interesting avenue for future research.

4. Generalization to other loss functions: While our analysis focused on the convergence of gradient descent for the empirical loss, extending our approach to other loss functions can provide insights into their convergence rates. By proving convergence rates for different loss functions, we can gain a more comprehensive understanding of the optimization landscape and potentially identify loss functions that exhibit faster convergence properties.

In summary, our investigation into the convergence properties of gradient descent in over-parameterized neural networks has shed light on the underlying dynamics of optimization in these networks. The provable convergence to the global minimum at a linear convergence rate has important implications for training deep learning models and provides a foundation for future research in the field. By exploring the suggested future directions, we can further enhance our understanding of optimization in neural networks and potentially develop more efficient training algorithms with improved convergence properties.

# 6 Appendix

Here is code A1, which shows how we corroborate synthetic data for numerical experiments.

```
import numpy as np
import pandas as pd

# Set the random seed for reproducibility
np.random.seed(1234)

# Generate n=1000 data points from a d=1000 dimensional unit sphere
X = np.random.normal(size=(1000, 1000))
X /= np.linalg.norm(X, axis=1, keepdims=True)

```

```
    # Generate labels from a one-dimensional standard
    # Gaussian distribution
    y = np.random.normal(size=1000)

    # Convert the data and labels to a Pandas DataFrame
    df = pd.DataFrame(data=X)
    df['y'] = y

    # Save the data and labels to a CSV file
    df.to_csv('data.csv', index=False)

    print("Data saved to 'data.csv'")
```

Here is code A2, which denotes how the amount of overparameterization affects convergence rates.

```
    import torch
    import torch.nn as nn
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt

    # Load data
    df = pd.read_csv("/content/drive/My Drive/ds&ml/data.csv")
    X = df.iloc[:, :-1].values
    y = df.iloc[:, -1].values

    # Convert data and labels to tensors
    X = torch.from_numpy(X).float()
    y = torch.from_numpy(y).float()

    # Define neural network architecture
    # Define neural network architecture
class Net(nn.Module):
    def __init__(self, m):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(X.shape[1], m)
        torch.nn.init.normal_(self.fc1.weight)
        torch.nn.init.normal_(self.fc1.bias)
        self.fc2 = nn.Linear(m, 1)
        torch.nn.init.normal_(self.fc2.weight)
        torch.nn.init.normal_(self.fc2.bias)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    criterion = nn.MSELoss()
    # Train neural network for each value of m
    m_list = [500, 1000, 2000, 4000, 8000]
    epochs = 100
    learning_rate = 0.1
    train_loss_list = []
    sign_diff_list = []
```

```
    max_distance_list = []

    for m in m_list:
        net = Net(m)
        optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)
        train_loss = []
        sign_diff = []
        max_distance = 0.0
        for epoch in range(epochs):
            # Forward pass
            y_pred = net(X).squeeze()
            loss = criterion(y_pred,y)
            train_loss.append(loss.item())

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        train_loss_list.append(train_loss)
    new_net = Net(m)
    new_net.load_state_dict(torch.load('saved_weights.pth'))
    plt.figure(figsize=(10, 5))
    for i, m in enumerate(m_list):
        plt.subplot(1, 2, 1)
        plt.plot(range(epochs), np.log(train_loss_list[i]), label=f"m={m}")
        plt.xlabel("Epoch")
        plt.ylabel("Log(Training Loss)")
        plt.legend()
    torch.save(net.state_dict(), 'saved_weights.pth')
```

Here is code A3, which tests the relationship between the amount of over-parameterization and the maximum of distances between weight vectors and their initialization.

```
    import torch
    import torch.nn as nn
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt

    # Load data
    df = pd.read_csv("/content/drive/My Drive/ds&ml/data.csv")
    X = df.iloc[:, :-1].values
    y = df.iloc[:, -1].values

    # Convert data and labels to tensors
    X = torch.from_numpy(X).float()
    y = torch.from_numpy(y).float()

    # Define neural network architecture
    class Net(nn.Module):
    def __init__(self, m):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(X.shape[1], m)
        torch.nn.init.normal_(self.fc1.weight)
```

```
        torch.nn.init.normal_(self.fc1.bias)
        self.fc2 = nn.Linear(m, 1)
        torch.nn.init.normal_(self.fc2.weight)
        torch.nn.init.normal_(self.fc2.bias)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    def max_weight_distance_from_initial(self):
    max_distance = 0
    if self.w_initial is None:  # Check if initial weights are stored
        self.w_initial = []  # Store initial weights
        for param in self.parameters():
            self.w_initial.append(param.detach().clone())
            # Clone and detach the initial weights
    for init_param, param in zip(self.w_initial, self.parameters()):
        distance = torch.max(torch.abs(param - init_param)).item()
        if distance > max_distance:
            max_distance = distance
    return max_distance

    criterion = nn.MSELoss()
    m_list = [500, 1000, 2000, 4000, 8000]
    epochs = 100
    learning_rate = 0.1
    max_weight_distance_list = []

    for m in m_list:
        net = Net(m)
        optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)
        max_weight_distance = np.zeros(epochs)  # initialize with zeros
        for epoch in range(epochs):
            # Forward pass
            y_pred = net(X).squeeze()
            loss = criterion(y_pred, y)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Calculate max weight distance at iteration k=50
            max_distance = net.max_weight_distance_from_initial()
            max_weight_distance[epoch] = max_distance  # append to array

        max_weight_distance_list.append(max_weight_distance)

    # Plot max weight distance vs epoch for each value of m
    plt.figure()
    for i, m in enumerate(m_list):
        plt.plot(range(1, epochs+1), max_weight_distance_list[i],
        label=f"m={m}")
```

```
77        plt.legend()
78        plt.xlabel("Epoch")
79        plt.ylabel("Max weight distance from initialization")
80        plt.show()
```

# References

[1] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," 2019.

[2] D. Soudry and Y. Carmon, "No bad local minima: Data independent training error guarantees for multilayer neural networks," 2016.

[3] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with relu activation," 2017.

[4] L. Chizat and F. Bach, "On the global convergence of gradient descent for over-parameterized models using optimal transport," 2018.