Lecture 3: SGD Convergence, Adam, and Linearization

Jincheng Ou

Fall 25

Contents

1 Introduction

This lecture builds upon previous discussions regarding Gradient Descent (GD) and Stochastic Gradient Descent (SGD). It covers three main areas:

1. A rigorous convergence proof for SGD with constant step size in specific settings.
2. The motivation and derivation of Adaptive optimization algorithms (Adam and AdamW).
3. Perspectives on analyzing Neural Networks as linearized models.

2 SGD Convergence Analysis

2.1 Problem Setup

We consider an underdetermined system of linear equations, which is common in over-parameterized models.

$$X\vec{w} = \vec{y} \tag{1}$$

Where X is a wide matrix (full row rank, infinite solutions). We seek the **minimum norm solution** $\vec{w}^*$. The analytic solution is given by:

$$\vec{w}^* = X^T(XX^T)^{-1}\vec{y} \tag{2}$$

We analyze SGD with a batch size of 1 and initialization $\vec{w}_0 = \vec{0}$.

2.2 Change of Basis and Coordinate Systems

To analyze convergence, it is beneficial to switch to error coordinates and align with the Singular Value Decomposition (SVD) of X.

2.2.1 Error Coordinates (q)

We define the error vector relative to the optimal solution:

$$\vec{q}_t = \vec{w}_t - \vec{w}^* \implies X\vec{q}_t = X\vec{w}_t - \vec{y} \tag{3}$$

Since $X\vec{w}^* = \vec{y}$, we essentially solve $X\vec{q} = 0$. The initial condition becomes $\vec{q}_0 = -\vec{w}^*$.

2.2.2 SVD Coordinates (z)

Let the SVD of the data matrix be $X = U\Sigma V^T$. We rotate the error vector into the basis of right singular vectors (V):

$$\vec{z}_t = V^T\vec{q}_t \tag{4}$$

The system equation transforms as follows:

- Since X is wide, $\Sigma = [\Sigma_{diag}|0]$.
- The effective dynamics only concern the non-zero singular value components.
- We can write the transformed system as $\tilde{X}\vec{z} = 0$, focusing only on relevant dimensions .

2.3 SGD Dynamics in z-space

The SGD update rule for a randomly selected data point index i_t (batch size 1) is:

$$\vec{z}_{t+1} = \vec{z}_t - \eta(\vec{x}_{i_t}\vec{x}_{i_t}^T)\vec{z}_t \tag{5}$$

where $\vec{x}_{i_t}$ represents the row of X corresponding to the sampled data point.

To prove convergence, we analyze the squared loss function (Lyapunov function approach) . We examine the evolution of the squared norm of the error:

$$\|\vec{z}_{t+1}\|^2 = \|\vec{z}_t - \eta\nabla_{i_t}\|^2 \tag{6}$$

Expanding this squared term allows us to split the update into two components, often referred to as a linear term (driving decay) and a quadratic term (noise/variance).

2.3.1 Analysis of Terms

We can express the expected loss at the next step conditioned on the current state $\vec{z}_t$:

$$\mathbb{E}[L(\vec{z}_{t+1})|\vec{z}_t] = L(\vec{z}_t) + \mathbb{E}[A] + \mathbb{E}[B] \tag{7}$$

- **Term A (Descent):** Related to $-2\eta\vec{z}_t^T\mathbb{E}[\vec{x}_{i_t}\vec{x}_{i_t}^T]\vec{z}_t$. This term provides the negative drift required for convergence. The expectation $\mathbb{E}[\vec{x}_{i_t}\vec{x}_{i_t}^T]$ recovers the full covariance $\frac{1}{N}X^TX$, which is bounded by singular values σ_{min}^2 and σ_{max}^2 .
- **Term B (Noise):** Quadratic in the update step size η^2. This represents the "bounce" or noise inherent in SGD.

Convergence Condition

Usually, SGD requires decaying step sizes to converge. However, in this realizable setting (where Loss $\to 0$ is possible), SGD with a **constant step size** η converges if η is sufficiently small. Specifically, the descent term A must dominate the noise term B .

3 Adaptive Optimization: From SGD to Adam

3.1 The Problem with Gradient Descent

Standard Gradient Descent (and SGD) scales updates uniformly across all parameters using a single learning rate α.

- **Feature/Bug:** GD takes large steps in directions with large singular values (high curvature) and small steps in directions with small singular values (low curvature) .
- **Consequence:**
  - *Advantage:* Implicit regularization and early stopping often work well.
  - *Disadvantage:* It is hard to choose a learning rate. A rate large enough to make progress in flat directions may cause instability (blowing up) in steep directions .

3.2 Approaches to Normalization

The ideal solution would be to normalize steps using the inverse Hessian or SVD, but this is computationally too expensive . Instead, we use element-wise approximations.

3.2.1 Sign SGD

A crude approximation is to simply use the sign of the gradient.

$$\vec{w}_{t+1} = \vec{w}_t - \eta \cdot \text{sign}(\nabla_{\vec{w}} L(\vec{w})) \tag{8}$$

This is equivalent to normalizing the gradient by its magnitude (element-wise):

$$\text{sign}(g) = \frac{g}{\sqrt{g^2}} \tag{9}$$

Issue: This leads to overshooting and "bouncing" around the optimum once the optimizer gets close .

3.3 Adam (Adaptive Moment Estimation)

Adam combines the idea of normalization (from SignSGD/RMSProp) with Momentum to smooth out the trajectory.

3.3.1 Algorithm Steps

1. **Momentum (First Moment):** Exponential moving average of gradients.

$$\vec{m}_{t+1} = \beta \vec{m}_t + (1-\beta)\nabla_{\vec{w}} L(\vec{w}_t) \tag{10}$$

2. **Squared Gradient (Second Moment):** Exponential moving average of squared gradients.

$$\vec{v}_{t+1} = \gamma \vec{v}_t + (1-\gamma)(\nabla_{\vec{w}} L(\vec{w}_t))^2 \tag{11}$$

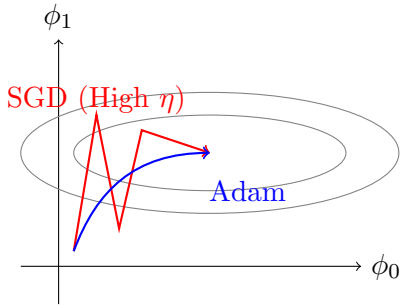
3. **Bias Correction:** Since vectors are initialized to 0, early estimates are biased towards 0 (especially if decay rates β, γ are close to 1) .

$$\hat{\vec{m}}_{t+1} = \frac{\vec{m}_{t+1}}{1-\beta^{t+1}}, \quad \hat{\vec{v}}_{t+1} = \frac{\vec{v}_{t+1}}{1-\gamma^{t+1}} \tag{12}$$

4. **Update Rule:**

$$\vec{w}_{t+1} = \vec{w}_t - \eta \frac{\hat{\vec{m}}_{t+1}}{\sqrt{\hat{\vec{v}}_{t+1}} + \epsilon} \tag{13}$$

3.3.2 Visual Comparison



Adam often takes a smoother path directly to the minimum compared to SGD, which may oscillate or require very small learning rates .

3.4 Memory Cost

Optimization comes with memory trade-offs :

- **SGD:** Requires current weights + gradient ($\times 1$).
- **SGD + Momentum:** Requires weights + velocity vector ($\times 2$).
- **Adam:** Requires weights + m_t + v_t ($\times 3$).

4 AdamW: Weight Decay in Adaptive Methods

Standard L_2 regularization (Ridge Regression) adds a penalty to the loss function: $L_{reg}(\vec{w}) = L(\vec{w}) + \frac{\lambda}{2}\|\vec{w}\|^2$. In standard GD, this results in:

$$\vec{w}_{t+1} = (1 - \eta\lambda)\vec{w}_t - \eta\nabla L(\vec{w}_t) \tag{14}$$

Problem with Adam: In Adam, the gradient magnitude is normalized. If we add the regularization term to the gradient, it gets normalized by $\sqrt{\vec{v}_t}$, meaning the effective regularization strength changes depending on the gradient magnitude .

Solution (AdamW): Decouple the weight decay from the gradient update. Apply weight decay *directly* to the parameters :

$$\vec{w}_{t+1} = \vec{w}_t - \eta\left(\frac{\hat{\vec{m}}_{t+1}}{\sqrt{\hat{\vec{v}}_{t+1}} + \epsilon} + \lambda\vec{w}_t\right) \tag{15}$$

Or essentially: $\vec{w}_{t+1} = (1 - \lambda_{decay})\vec{w}_t - \dots$.

5 Linearization of Neural Networks

To understand deep learning optimization, we often approximate Neural Networks (NNs) as linear models, despite their non-convex nature .

5.1 Perspective 1: Feature Extraction (Linear Probing)

A deep network can be viewed as a feature extractor followed by a linear layer.

- Layers 1 to $L-1$: Compute non-linear features $\phi(\vec{x})$.
- Layer L: Linear model acting on $\phi(\vec{x})$.

Linear Probing: Freezing the early layers and retraining only the final linear layer allows us to reuse computed features . This is mathematically a linear regression on transformed inputs.

5.2 Perspective 2: Taylor Expansion (Lazy Training)

We can linearize the network dynamics using a first-order Taylor expansion around initialization θ_0:

$$f(\vec{x}, \vec{\theta}) \approx f(\vec{x}, \vec{\theta_0}) + \nabla_\theta f(\vec{x}, \theta_0)^T(\vec{\theta} - \vec{\theta_0}) \tag{16}$$

Here, the gradient $\nabla_\theta f$ acts as the "feature" vector .

5.2.1 Lazy Training Assumption

If the parameters θ do not move far from initialization (θ_0) during training (the "Lazy Training" regime), the gradient features $\nabla_\theta f(\vec{x}, \theta_0)$ remain approximately constant .

- The training dynamics become equivalent to solving a linear system using the **Neural Tangent Kernel (NTK)** .
- In the limit of infinite width, GD converges to zero training loss .
- The evolution of the function is governed by:

$$\frac{df(x)}{dt} = -\frac{1}{N}\sum_{i=1}^{N}\underbrace{\nabla f(x,\theta)^T\nabla f(x^{(i)},\theta)}_{K(x,x^{(i)})}\nabla_f L \tag{17}$$