Lecture 1, 2, 3: Deep Learning Optimization

Jincheng Ou

Fall 25

Contents

1 Introduction, Function Approximation, and Neural Architectures

1.1 The Machine Learning Problem

The fundamental goal of machine learning and deep learning is **function approximation**. Consider a true, unknown function $f : \mathbb{R}^d \to \mathbb{R}$. We do not have access to f directly. Instead, we possess a dataset of samples:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$$

where $y_i \approx f(x_i)$. The objective is to learn a parameterized model $f_\theta(x)$ that approximates the true function $f(x)$ on unseen data.

1.2 Machine Learning Paradigms

1.2.1 Supervised Learning

Data comes as pairs (x, y).

- **Regression:** $y \in \mathbb{R}$ (real-valued). Example: Predicting house prices.
- **Classification:** $y \in \{0, 1\}$ or discrete classes. Example: Cat vs. Dog.
- **Localized Annotation (DL Extension):** y represents pixel-level labels (Semantic Segmentation). Critical for tasks like self-driving cars.

1.2.2 Unsupervised Learning

Data is only x. The goal is to find structure.

- **Dimensionality Reduction:** finding variation (e.g., PCA in classical ML; Autoencoders in DL).
- **Clustering:** Grouping similar points (e.g., K-Means).
- **Density Estimation:** estimating $p(x)$. In DL, this relates to **Generative Models** (generating new samples from the distribution).
- **Learned Embeddings:** Mapping discrete objects (words) to vectors (e.g., Word2Vec in NLP).

1.2.3 Generative and Foundation Models

- **Generative Models:** Sampling from a learned distribution $p(x)$ to create new data (e.g., generating images of cats).
- **Foundation Models:** A paradigm shift from "training specific models for specific tasks" to training massive models on diverse data to learn general representations, which are then **fine-tuned** for specific downstream tasks.

1.3 Piecewise Linear Approximation

One intuitive way to approximate a complex non-linear function is via **piecewise linear approximation**.

- A function can be approximated by connecting discrete points ("elbows") with lines.
- To parameterize this for a computer, we use a sum of basic units (neurons).
- A single unit with a "kink" can be represented as:

$$g_\theta(x) = \max(0, wx + b) \tag{1}$$

This is known as the **ReLU (Rectified Linear Unit)** activation function.

By summing multiple shifted and scaled ReLUs, $F(x) = b_{out} + \sum_i w_i \text{ReLU}(w_i' x + b_i')$, we can construct an arbitrary piecewise linear function with as many "kinks" as needed to approximate the target curve.

1.4 Neural Network Architecture

In block diagram form, a simple neural network layer consists of: 1. **Linear Transformation:** Affine transform $z = Wx + b$. 2. **Non-linearity (Activation):** $h = \sigma(z)$, e.g., $\sigma(\cdot) = \text{ReLU}(\cdot)$.

For a multi-layer network (Deep Learning):

$$x \xrightarrow{W_1, b_1} z_1 \xrightarrow{\text{ReLU}} h_1 \xrightarrow{W_2, b_2} z_2 \ldots \to \hat{y}$$

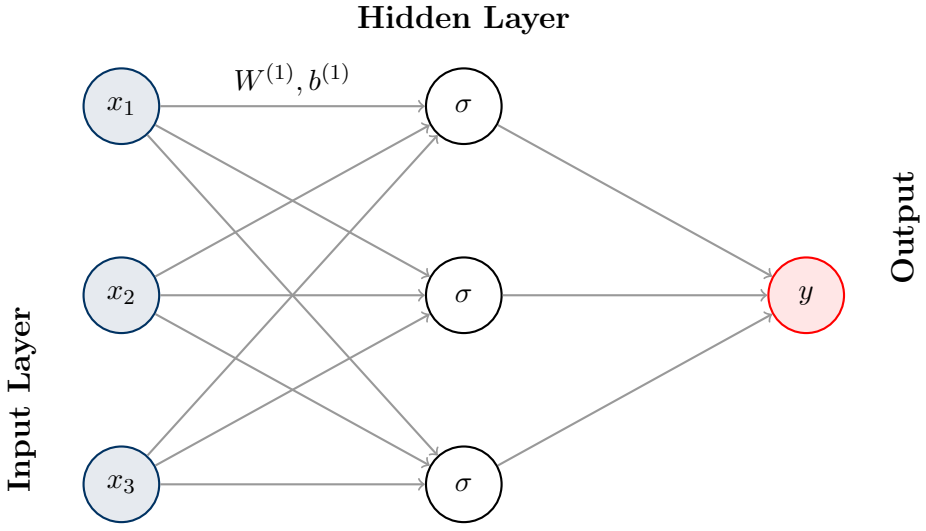

Figure 1: Visualizing a Shallow Neural Network: Input → Affine+ReLU → Output

2 Optimization Fundamentals

2.1 Empirical Risk Minimization (ERM)

The goal is to find parameters θ that minimize a loss function. Given training data $\{(x_i, y_i)\}_{i=1}^n$, we want to find:

$$\hat{\theta} = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f_\theta(x_i)) \tag{2}$$

Challenges in Optimization:

1. **Unknown Distribution:** We assume $(x, y) \sim p(x, y)$, but we don't know p. We only have training samples.
2. **Generalization:** Minimizing training loss does not guarantee low test loss.
3. **Overfitting:** The model fits noise in the training data, leading to poor performance on test data.

2.2 Gradient Descent (GD)

To minimize $\mathcal{L}(\theta)$, we iteratively update parameters in the direction of the negative gradient.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{3}$$

where η is the learning rate.

2.2.1 Derivation and Analysis for Least Squares

Consider linear regression with squared loss:

$$\mathcal{L}(w) = \|Xw - y\|_2^2 = (Xw - y)^T (Xw - y)$$

Gradient derivation:

$$\nabla_w \mathcal{L}(w) = 2X^T(Xw - y)$$

The GD update rule becomes:

$$w_{t+1} = w_t - 2\eta X^T(Xw_t - y) \tag{4}$$

Derivation: GD Convergence Dynamics

Let w^* be the optimal solution such that $X^T(Xw^* - y) = 0$. We define the error vector $e_t = w_t - w^*$. Subtracting w^* from both sides of the update rule:

$$w_{t+1} - w^* = w_t - w^* - 2\eta(X^T X w_t - \underbrace{X^T y}_{X^T X w^*})$$

$$e_{t+1} = e_t - 2\eta X^T X (w_t - w^*)$$

$$e_{t+1} = (I - 2\eta X^T X) e_t$$

This is a linear recurrence relation. For convergence, the eigenvalues of the matrix $(I - 2\eta X^T X)$ must have magnitude less than 1. Let $\lambda_{\max}$ and $\lambda_{\min}$ be the maximum

and minimum eigenvalues of X^TX (which correspond to squared singular values σ^2).
Condition for stability:

$$|1 - 2\eta\lambda_{\max}| < 1 \implies \eta < \frac{1}{\lambda_{\max}}$$

Key Insight: Convergence rate depends on the **condition number** $\kappa = \lambda_{\max}/\lambda_{\min}$. GD takes big steps in directions of large curvature (large singular values) and small steps in directions of low curvature.

Optimal Learning Rate: $\eta^* = \frac{1}{\lambda_{min}+\lambda_{max}}$.

2.3 Regularization and Implicit Bias

2.3.1 Explicit Regularization (Ridge Regression)

To prevent overfitting, we add a penalty term:

$$\hat{w}_{Ridge} = \arg\min_w \|Xw - y\|_2^2 + \lambda\|w\|_2^2$$

Closed-form solution using SVD ($X = U\Sigma V^T$):

$$w^* = \sum_i \frac{\sigma_i}{\sigma_i^2 + \lambda}(u_i^T y)v_i \tag{5}$$

Interpretation:

- If $\sigma_i^2 \gg \lambda$: coefficient $\approx 1/\sigma_i$ (standard inverse).
- If $\sigma_i^2 \ll \lambda$: coefficient $\approx \sigma_i/\lambda \approx 0$ (suppressed).
- Ridge effectively filters out directions corresponding to small singular values (low variance/noise).

2.3.2 Implicit Regularization via Early Stopping

Consider Gradient Descent initialized at $w_0 = 0$. In the SVD basis (rotated coordinates), the update for the i-th component $\tilde{w}_{t,i}$ is:

$$\tilde{w}_{t+1,i} = (1 - 2\eta\sigma_i^2)\tilde{w}_{t,i} + 2\eta\sigma_i\tilde{y}_i$$

The explicit solution at step t is:

$$w_t = \sum_i \frac{1}{\sigma_i}(1 - (1 - 2\eta\sigma_i^2)^t)(u_i^T y)v_i \tag{6}$$

Comparison to Ridge:

- For large σ_i, $(1 - 2\eta\sigma_i^2)^t \to 0$ quickly. The component converges to $1/\sigma_i$.
- For small σ_i, $(1 - 2\eta\sigma_i^2) \approx 1$. The term decays very slowly.

Early Stopping as Regularization

If we stop GD at time $t < \infty$ (Early Stopping), components corresponding to small singular values have not yet built up. Thus, **Early Stopping acts as an implicit regularizer**, effectively mimicking Ridge Regression by suppressing small singular value directions (noise) without an explicit penalty term.

2.4 Stochastic Gradient Descent (SGD)

GD is computationally expensive ($O(n)$ per step) and gets stuck in local minima in non-convex landscapes.

The Algorithm: Instead of the full gradient $\nabla\mathcal{L}(\theta) = \frac{1}{n}\sum \nabla\mathcal{L}_i(\theta)$, we estimate it using a **mini-batch** $\mathcal{B}$:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla\mathcal{L}_i(\theta_t) \tag{7}$$

Properties:

- Unbiased estimator of full gradient: $\mathbb{E}[\nabla_{SGD}] = \nabla_{GD}$.
- Adds **noise**, which helps escape local minima and saddle points.
- Computational cost is constant with respect to dataset size n.

2.5 Momentum

To dampen oscillations (especially in directions with high curvature/low singular values) and accelerate convergence:

Idea: Use a moving average of past gradients (Low-pass filter).

$$z_{k+1} = \beta z_k + (1-\beta)\nabla\mathcal{L}(w_k) \tag{8}$$

$$w_{k+1} = w_k - \eta z_{k+1} \tag{9}$$

where $\beta \in [0, 1)$ is the momentum parameter (typically 0.9). This accumulates velocity in consistent directions and cancels out noise in oscillating directions.

3 Advanced Optimization

3.1 SGD Convergence Proof (Constant Step Size)

Consider $Xw = y$ where $d > n$ (wide matrix, infinite solutions). This is the **Over-parameterized / Interpolation Regime** where the loss can actually reach exactly zero. Surprisingly, SGD can converge with a *constant* step size in this regime.

Derivation: SGD Convergence (Underdetermined System)

Setup: Let $X \in \mathbb{R}^{n \times d}$ where $d > n$ (wide). Assume full row rank. Let w^* be the minimum norm solution. Define error $q_t = w_t - w^*$. We want to solve $Xq = 0$. Using SVD basis $z = V^T q$, the relevant dynamics occur in the top n dimensions.

Update Rule (Batch size 1): At step t, randomly sample index i_t. The update is a projection step:

$$z_{t+1} = z_t - 2\eta(x_{i_t}x_{i_t}^T)z_t \tag{10}$$

Lyapunov Function Analysis: We define a Lyapunov function (Energy) $V(z) = \|Xz\|^2$ (Squared Loss). We want to show expected decrease:

$$\mathbb{E}[V(z_{t+1})|z_t] \leq (1-\rho)V(z_t)$$

Expanding $V(z_{t+1}) = \|Xz_{t+1}\|^2$:

$$\|X(z_t - 2\eta x_{i_t}x_{i_t}^T z_t)\|^2 \tag{11}$$

Let A be the linear term and B be the quadratic term of the expansion. Taking expectation over i_t (knowing $\mathbb{E}[x_{i_t}x_{i_t}^T] = \frac{1}{n}X^TX$):

- **Descent Term (A):** Comes from linear part.

  $$\mathbb{E}[A] \propto -\eta\sigma_{min}^2 V(z_t)$$

  This provides the negative drift (convergence).

- **Noise Term (B):** Comes from quadratic part $\eta^2(\dots)$. Bounded by the max norm of data $\|x\|_{max}^2$ and σ_{max}.

Result: For sufficiently small η (specifically $\eta < \frac{2}{\sigma_{max}^2 + \|x\|_{max}^2}$), the descent term dominates the noise term. The system converges exponentially to 0 error. Note the dependence on **data norms** $\|x_i\|^2$.

3.2 Adaptive Optimization: Adam

3.2.1 Motivation: The Scaling Problem

Standard GD scales every parameter update by the same scalar η.

- If the condition number $\kappa = \lambda_{max}/\lambda_{min}$ is large, GD bounces in steep directions (high λ) and crawls in flat directions (low λ).
- Ideal world: Normalize by curvature (Newton's method / inverse Hessian), but this is computationally prohibitive ($O(d^3)$).

3.2.2 SignSGD

A crude approximation: Just look at the sign.

$$w_{t+1} = w_t - \eta \cdot \text{sign}(\nabla\mathcal{L}) \tag{12}$$

This normalizes magnitude but causes "chattering" (oscillation) around the optimum.

3.2.3 Adam (Adaptive Moment Estimation)

Adam combines **Momentum** (smoothing) and **RMSProp** (Adaptive Scaling, normalizing by second moment).

Algorithm:

1. Update biased first moment (Momentum):
$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla\mathcal{L}$$
2. Update biased second moment (Uncentered Variance):
$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla\mathcal{L})^2 \quad \text{(element-wise square)}$$
3. Bias correction (crucial for early steps to fix "Slow Start"):
$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}}, \quad \hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}}$$
4. **Parameter Update:**
$$w_{t+1} = w_t - \eta \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon} \tag{13}$$

Interpretation: The term $\frac{1}{\sqrt{\hat{v}}}$ approximates $1/|g|$, effectively normalizing the step size so it does not depend on the raw magnitude of the gradient, similar to SignSGD but smoothed.

3.2.4 AdamW

Standard L2 regularization ($\lambda\|\theta\|^2$) in the loss does not work well with Adam's adaptive scaling because the weight decay gradient gets normalized by v_t. **AdamW** decouples weight decay, applying it directly to the parameter update:

$$w_{t+1} = \text{AdamStep}(w_t) - \eta\lambda w_t$$

3.3 Linear Perspectives on Neural Networks

Neural networks are non-convex, but we can analyze them via linear approximations.

3.3.1 Linear Probing

Train a deep network, freeze the weights of layers $1 \dots L-1$ (treating them as a fixed feature extractor $\phi(x)$), and only train the final linear layer. This reduces the problem to convex logistic/linear regression on learned features.

3.3.2 Neural Tangent Kernel (NTK) / Linearization

Consider the first-order Taylor expansion of the network output $f(x, \theta)$ around initialization θ_0:

$$f(x, \theta) \approx f(x, \theta_0) + \nabla_\theta f(x, \theta_0)^T(\theta - \theta_0) \tag{14}$$

- In the "Lazy Training" regime (extremely wide networks), weights θ hardly move from θ_0.
- The network behaves like a linear model with features given by the gradient $\nabla_\theta f(x, \theta_0)$.
- This allows the analysis of convergence and generalization using linear theory (Kernel Ridge Regression).