Lecture 4: Convexity, Optimization, and Loss Functions

Justo E. Karell

CS 583: Deep Learning

Introduction

Convexity and optimization are foundational concepts in the training of neural networks. Understanding convexity helps in grasping the behavior of loss functions, while optimization techniques are crucial for adjusting network parameters to minimize these loss functions. This lecture delves into the mathematical underpinnings of convexity, explores various optimization algorithms, and demonstrates the optimization process through a detailed example, highlighting how loss function values change over iterations.

1. Convex Functions and Sets

1.1. Convex Sets

A set $S \subseteq \mathbb{R}^n$ is called **convex** if, for any two points $\mathbf{x}, \mathbf{y} \in S$ and any $\theta \in [0, 1]$, the following holds:

$$\theta\mathbf{x} + (1 - \theta)\mathbf{y} \in S$$

Example: The set of points inside a circle is convex, whereas the set forming a crescent moon is non-convex.

1.2. Convex Functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is **convex** if its domain is a convex set and, for all $\mathbf{x}, \mathbf{y} \in \text{dom}(f)$ and $\theta \in [0, 1]$:

$$f(\theta\mathbf{x} + (1 - \theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1 - \theta)f(\mathbf{y})$$

Properties:

- **Local Minimum is Global Minimum**: In convex functions, any local minimum is also a global minimum.
- **First-Order Condition**: If f is differentiable, then for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$:

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$$

- **Second-Order Condition**: If f is twice differentiable, it is convex if and only if its Hessian matrix $\mathbf{H}$ is positive semi-definite for all $\mathbf{x}$.

Example: The quadratic function $f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$ is convex if and only if $\mathbf{A}$ is positive semi-definite.

2. Loss Functions

2.1. Role of Loss Functions

The **loss function** quantifies the difference between the network's predictions and the actual target values. It serves as the objective function to be minimized during training. The choice of loss function impacts the optimization landscape and, consequently, the ease with which optimization algorithms can find the global minimum.

2.2. Common Loss Functions

- **Mean Squared Error (MSE)** for regression tasks:

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_{\text{true},i})^2$$

  - *Convexity*: The MSE loss function is convex with respect to the model parameters when the model is linear.

- **Binary Cross-Entropy** for binary classification:

$$L_{\text{binary}} = -\left[y_{\text{true}}\log(y) + (1 - y_{\text{true}})\log(1 - y)\right]$$

  - *Convexity*: The binary cross-entropy loss is convex with respect to the model's output y but may not be convex with respect to the model parameters in non-linear models.

- **Categorical Cross-Entropy** for multi-class classification:

$$L_{\text{categorical}} = -\sum_{i=1}^{k} y_{\text{true},i}\log(y_i)$$

  - *Convexity*: Similar to binary cross-entropy, it is convex in y_i but not necessarily in the parameters.

2.3. Loss Functions and Convexity

In neural networks, the loss function's convexity with respect to the parameters is crucial for optimization:

- **Convex Loss Functions**: Guarantee that any local minimum is a global minimum, simplifying optimization.
- **Non-Convex Loss Functions**: Present multiple local minima and saddle points, making optimization more challenging.

Due to the non-linear activation functions and complex architectures in neural networks, the overall loss function becomes **non-convex** with respect to the network parameters.

3. Optimization in Neural Networks

3.1. Optimization Algorithms

Several optimization algorithms are employed to minimize the loss function:

- **Gradient Descent**:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta\frac{\partial L}{\partial \mathbf{W}}, \quad \mathbf{b} \leftarrow \mathbf{b} - \eta\frac{\partial L}{\partial \mathbf{b}}$$

  where η is the learning rate.

- **Stochastic Gradient Descent (SGD)**: Uses a single or a few training examples to compute the gradient at each step, providing faster iterations.
- **Adam (Adaptive Moment Estimation)**: Combines momentum and adaptive learning rates for more efficient and stable training.
- **RMSprop**: Adjusts the learning rate based on a moving average of squared gradients.

3.2. Backpropagation

Backpropagation is the algorithm used to compute the gradients of the loss function with respect to the network's weights and biases. It involves two main phases:

- **Forward Pass**: Compute the outputs of each layer and the final prediction.
- **Backward Pass**: Compute the gradients of the loss with respect to each parameter by applying the chain rule.

3.3. Loss Function Values Over Iterations

During training, we aim to minimize the loss function. By updating the network parameters using the computed gradients, we adjust the model to reduce the loss. Tracking the loss function values over iterations provides insight into the optimization process and the model's learning progress.

4. Example: Training a Simple Neural Network

Let's walk through the training process of a simple neural network with one hidden layer using gradient descent. We will observe how the loss function changes over multiple iterations.

4.1. Network Architecture and Parameters

- **Input Vector**: $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \in \mathbb{R}^2$
- **Weights for Hidden Layer**:

$$\mathbf{W}_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.4 \\ -0.5 & 0.3 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

- **Biases for Hidden Layer**:

$$\mathbf{b}_1 = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} \in \mathbb{R}^2$$

- **Weights for Output Layer**:

$$\mathbf{W}_2 = \begin{bmatrix} w_{31} & w_{32} \end{bmatrix} = \begin{bmatrix} 0.7 & -0.3 \end{bmatrix} \in \mathbb{R}^{1\times 2}$$

- **Bias for Output Layer**: $b_2 = 0.05 \in \mathbb{R}$
- **Activation Function**: Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$
- **Learning Rate**: $\eta = 0.1$
- **Target Output**: $y_{\text{true}} = 1$

4.2. Iteration 1

Step 1: Forward Propagation

Compute Weighted Sum for Hidden Layer

$$\begin{aligned}
\mathbf{z}_1 &= \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \\
&= \begin{bmatrix} 0.2 & 0.4 \\ -0.5 & 0.3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} \\
&= \begin{bmatrix} (0.2)(1) + (0.4)(2) + 0.1 \\ (-0.5)(1) + (0.3)(2) - 0.2 \end{bmatrix} \\
&= \begin{bmatrix} 0.2 + 0.8 + 0.1 \\ -0.5 + 0.6 - 0.2 \end{bmatrix} \\
&= \begin{bmatrix} 1.1 \\ -0.1 \end{bmatrix}
\end{aligned}$$

Apply Activation Function (Sigmoid)

$$\begin{aligned}
\mathbf{h} &= \sigma(\mathbf{z}_1) \\
&= \begin{bmatrix} \sigma(1.1) \\ \sigma(-0.1) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{1+e^{-1.1}} \\ \frac{1}{1+e^{0.1}} \end{bmatrix} \\
&\approx \begin{bmatrix} 0.750 \\ 0.475 \end{bmatrix}
\end{aligned}$$

Compute Weighted Sum for Output Layer

$$\begin{aligned}
z_2 &= \mathbf{W}_2\mathbf{h} + b_2 \\
&= \begin{bmatrix} 0.7 & -0.3 \end{bmatrix} \begin{bmatrix} 0.750 \\ 0.475 \end{bmatrix} + 0.05 \\
&= (0.7)(0.750) + (-0.3)(0.475) + 0.05 \\
&= 0.525 - 0.1425 + 0.05 \\
&= 0.4325
\end{aligned}$$

Apply Activation Function (Sigmoid)

$$\begin{aligned}
y &= \sigma(z_2) \\
&= \frac{1}{1 + e^{-0.4325}} \\
&\approx 0.606
\end{aligned}$$

Step 2: Compute Loss

Using Mean Squared Error (MSE):

$$\begin{aligned}
L &= \frac{1}{2}(y - y_{\text{true}})^2 \\
&= \frac{1}{2}(0.606 - 1)^2 \\
&= \frac{1}{2}(-0.394)^2 \\
&\approx 0.0777
\end{aligned}$$

Step 3: Backpropagation

Compute Gradient of Loss w.r.t. Output Activation

$$\begin{aligned}\delta_2 &= \frac{\partial L}{\partial z_2} \\ &= (y - y_{\text{true}}) \cdot \sigma'(z_2) \\ &= (0.606 - 1) \cdot \sigma(z_2)(1 - \sigma(z_2)) \\ &= (-0.394) \cdot 0.606 \cdot 0.394 \\ &\approx -0.094\end{aligned}$$

Compute Gradients for Output Layer Weights and Bias

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}_2} &= \delta_2 \cdot \mathbf{h}^T \\ &= -0.094 \cdot \begin{bmatrix}0.750 & 0.475\end{bmatrix} \\ &= \begin{bmatrix}-0.0705 & -0.0447\end{bmatrix}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial b_2} &= \delta_2 \\ &= -0.094\end{aligned}$$

Compute Gradient of Loss w.r.t. Hidden Layer Activations

$$\begin{aligned}\delta_1 &= (\mathbf{W}_2^T \delta_2) \circ \sigma'(\mathbf{z}_1) \\ &= \begin{bmatrix}0.7 \\ -0.3\end{bmatrix} \cdot (-0.094) \circ \begin{bmatrix}\sigma'(1.1) \\ \sigma'(-0.1)\end{bmatrix} \\ &= \begin{bmatrix}-0.0658 \\ 0.0282\end{bmatrix} \circ \begin{bmatrix}0.750 \cdot 0.250 \\ 0.475 \cdot 0.525\end{bmatrix} \\ &= \begin{bmatrix}-0.0658 \cdot 0.1875 \\ 0.0282 \cdot 0.2494\end{bmatrix} \\ &\approx \begin{bmatrix}-0.0123 \\ 0.0070\end{bmatrix}\end{aligned}$$

Here, $\circ$ denotes element-wise multiplication.

Compute Gradients for Hidden Layer Weights and Biases

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}_1} &= \delta_1 \cdot \mathbf{x}^T \\ &= \begin{bmatrix}-0.0123 \\ 0.0070\end{bmatrix} \begin{bmatrix}1 & 2\end{bmatrix} \\ &= \begin{bmatrix}-0.0123 \times 1 & -0.0123 \times 2 \\ 0.0070 \times 1 & 0.0070 \times 2\end{bmatrix} \\ &= \begin{bmatrix}-0.0123 & -0.0246 \\ 0.0070 & 0.0140\end{bmatrix}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{b}_1} &= \delta_1 \\ &= \begin{bmatrix}-0.0123 \\ 0.0070\end{bmatrix}\end{aligned}$$

Step 4: Update Weights and Biases

Using Gradient Descent:

$$\begin{aligned}\mathbf{W}_2 &\leftarrow \mathbf{W}_2 - \eta \cdot \frac{\partial L}{\partial \mathbf{W}_2} \\ &= \begin{bmatrix}0.7 & -0.3\end{bmatrix} - 0.1 \cdot \begin{bmatrix}-0.0705 & -0.0447\end{bmatrix} \\ &= \begin{bmatrix}0.7 + 0.00705 & -0.3 + 0.00447\end{bmatrix} \\ &= \begin{bmatrix}0.70705 & -0.29553\end{bmatrix}\end{aligned}$$

$$\begin{aligned}b_2 &\leftarrow b_2 - \eta \cdot \frac{\partial L}{\partial b_2} \\ &= 0.05 - 0.1 \times (-0.094) \\ &= 0.05 + 0.0094 \\ &= 0.0594\end{aligned}$$

$$\begin{aligned}\mathbf{W}_1 &\leftarrow \mathbf{W}_1 - \eta \cdot \frac{\partial L}{\partial \mathbf{W}_1} \\ &= \begin{bmatrix}0.2 & 0.4 \\ -0.5 & 0.3\end{bmatrix} - 0.1 \cdot \begin{bmatrix}-0.0123 & -0.0246 \\ 0.0070 & 0.0140\end{bmatrix} \\ &= \begin{bmatrix}0.2 + 0.00123 & 0.4 + 0.00246 \\ -0.5 - 0.0007 & 0.3 - 0.0014\end{bmatrix} \\ &= \begin{bmatrix}0.20123 & 0.40246 \\ -0.5007 & 0.2986\end{bmatrix}\end{aligned}$$

$$\begin{aligned}\mathbf{b}_1 &\leftarrow \mathbf{b}_1 - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_1} \\ &= \begin{bmatrix}0.1 \\ -0.2\end{bmatrix} - 0.1 \cdot \begin{bmatrix}-0.0123 \\ 0.0070\end{bmatrix} \\ &= \begin{bmatrix}0.1 + 0.00123 \\ -0.2 - 0.0007\end{bmatrix} \\ &= \begin{bmatrix}0.10123 \\ -0.2007\end{bmatrix}\end{aligned}$$

Step 5: Iteration Summary

Updated Parameters After Iteration 1

$$\mathbf{W}_1 = \begin{bmatrix}0.20123 & 0.40246 \\ -0.5007 & 0.2986\end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix}0.10123 \\ -0.2007\end{bmatrix}$$

$$\mathbf{W}_2 = \begin{bmatrix}0.70705 & -0.29553\end{bmatrix}, \quad b_2 = 0.0594$$

$$y \approx 0.606, \quad L \approx 0.0777$$

4.3. Iteration 2

Repeating the steps for the second iteration:

Step 1: Forward Propagation

Compute Weighted Sum for Hidden Layer

$$\begin{aligned}\mathbf{z}_1 &= \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \\ &= \begin{bmatrix} 0.20123 & 0.40246 \\ -0.5007 & 0.2986 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.10123 \\ -0.2007 \end{bmatrix} \\ &= \begin{bmatrix} 0.20123 + 0.80492 + 0.10123 \\ -0.5007 + 0.5972 - 0.2007 \end{bmatrix} \\ &= \begin{bmatrix} 1.10738 \\ -0.1042 \end{bmatrix}\end{aligned}$$

Apply Activation Function (Sigmoid)

$$\begin{aligned}\mathbf{h} &= \sigma(\mathbf{z}_1) \\ &= \begin{bmatrix} \frac{1}{1+e^{-1.10738}} \\ \frac{1}{1+e^{0.1042}} \end{bmatrix} \\ &\approx \begin{bmatrix} 0.7516 \\ 0.47397 \end{bmatrix}\end{aligned}$$

Compute Weighted Sum for Output Layer

$$\begin{aligned}z_2 &= \mathbf{W}_2\mathbf{h} + b_2 \\ &= \begin{bmatrix} 0.70705 & -0.29553 \end{bmatrix} \begin{bmatrix} 0.7516 \\ 0.47397 \end{bmatrix} + 0.0594 \\ &= (0.70705)(0.7516) + (-0.29553)(0.47397) + 0.0594 \\ &= 0.53177 - 0.14016 + 0.0594 \\ &= 0.45101\end{aligned}$$

Apply Activation Function (Sigmoid)

$$\begin{aligned}y &= \sigma(z_2) \\ &= \frac{1}{1 + e^{-0.45101}} \\ &\approx 0.6109\end{aligned}$$

Step 2: Compute Loss

$$\begin{aligned}L &= \frac{1}{2}(y - y_{\text{true}})^2 \\ &= \frac{1}{2}(0.6109 - 1)^2 \\ &= \frac{1}{2}(-0.3891)^2 \\ &\approx 0.0757\end{aligned}$$

Step 3: Backpropagation

Compute Gradient of Loss w.r.t. Output Activation

$$\begin{aligned}\delta_2 &= (y - y_{\text{true}}) \cdot \sigma'(z_2) \\ &= (-0.3891) \cdot 0.6109 \cdot 0.3891 \\ &\approx -0.0924\end{aligned}$$

Compute Gradients for Output Layer Weights and Bias

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}_2} &= \delta_2 \cdot \mathbf{h}^T \\ &= -0.0924 \cdot \begin{bmatrix}0.7516 & 0.47397\end{bmatrix} \\ &= \begin{bmatrix}-0.0695 & -0.0438\end{bmatrix}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial b_2} &= \delta_2 \\ &= -0.0924\end{aligned}$$

Compute Gradient of Loss w.r.t. Hidden Layer Activations

$$\begin{aligned}\delta_1 &= (\mathbf{W}_2^T \delta_2) \circ \sigma'(\mathbf{z}_1) \\ &= \begin{bmatrix}0.70705 \\ -0.29553\end{bmatrix} \cdot (-0.0924) \circ \begin{bmatrix}\sigma'(1.10738) \\ \sigma'(-0.1042)\end{bmatrix} \\ &= \begin{bmatrix}-0.0654 \\ 0.0273\end{bmatrix} \circ \begin{bmatrix}0.7516 \cdot 0.2484 \\ 0.47397 \cdot 0.5260\end{bmatrix} \\ &= \begin{bmatrix}-0.0654 \cdot 0.1867 \\ 0.0273 \cdot 0.2490\end{bmatrix} \\ &\approx \begin{bmatrix}-0.0122 \\ 0.0068\end{bmatrix}\end{aligned}$$

Compute Gradients for Hidden Layer Weights and Biases

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}_1} &= \delta_1 \cdot \mathbf{x}^T \\ &= \begin{bmatrix}-0.0122 \\ 0.0068\end{bmatrix} \begin{bmatrix}1 & 2\end{bmatrix} \\ &= \begin{bmatrix}-0.0122 & -0.0244 \\ 0.0068 & 0.0136\end{bmatrix}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{b}_1} &= \delta_1 \\ &= \begin{bmatrix}-0.0122 \\ 0.0068\end{bmatrix}\end{aligned}$$

Step 4: Update Weights and Biases

$$\begin{aligned}\mathbf{W}_2 &\leftarrow \mathbf{W}_2 - \eta \cdot \frac{\partial L}{\partial \mathbf{W}_2} \\ &= \begin{bmatrix}0.70705 & -0.29553\end{bmatrix} - 0.1 \cdot \begin{bmatrix}-0.0695 & -0.0438\end{bmatrix} \\ &= \begin{bmatrix}0.70705 + 0.00695 & -0.29553 + 0.00438\end{bmatrix} \\ &= \begin{bmatrix}0.71400 & -0.29115\end{bmatrix}\end{aligned}$$

$$\begin{aligned}b_2 &\leftarrow b_2 - \eta \cdot \frac{\partial L}{\partial b_2} \\ &= 0.0594 + 0.00924 \\ &= 0.06864\end{aligned}$$

$$
\begin{aligned}
\mathbf{W}_1 &\leftarrow \mathbf{W}_1 - \eta \cdot \frac{\partial L}{\partial \mathbf{W}_1} \\
&= \begin{bmatrix} 0.20123 & 0.40246 \\ -0.5007 & 0.2986 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} -0.0122 & -0.0244 \\ 0.0068 & 0.0136 \end{bmatrix} \\
&= \begin{bmatrix} 0.20123 + 0.00122 & 0.40246 + 0.00244 \\ -0.5007 - 0.00068 & 0.2986 - 0.00136 \end{bmatrix} \\
&= \begin{bmatrix} 0.20245 & 0.40490 \\ -0.50138 & 0.29724 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{b}_1 &\leftarrow \mathbf{b}_1 - \eta \cdot \frac{\partial L}{\partial \mathbf{b}_1} \\
&= \begin{bmatrix} 0.10123 \\ -0.2007 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} -0.0122 \\ 0.0068 \end{bmatrix} \\
&= \begin{bmatrix} 0.10123 + 0.00122 \\ -0.2007 - 0.00068 \end{bmatrix} \\
&= \begin{bmatrix} 0.10245 \\ -0.20138 \end{bmatrix}
\end{aligned}
$$

Step 5: Iteration Summary

Updated Parameters After Iteration 2

$$
\mathbf{W}_1 = \begin{bmatrix} 0.20245 & 0.40490 \\ -0.50138 & 0.29724 \end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix} 0.10245 \\ -0.20138 \end{bmatrix}
$$

$$
\mathbf{W}_2 = \begin{bmatrix} 0.71400 & -0.29115 \end{bmatrix}, \quad b_2 = 0.06864
$$

$$
y \approx 0.6109, \quad L \approx 0.0757
$$

4.4. Loss Function Values Over Iterations

Iteration	Output y	Loss L	Change in Loss
1	0.6060	0.0777	–
2	0.6109	0.0757	−0.0020

Table 1: Loss function values over iterations

As observed, the loss function decreases over iterations, indicating that the optimization algorithm is successfully reducing the error between the predicted output and the true value.

5. Convexity in Optimization

5.1. Convex vs. Non-Convex Optimization

- **Convex Optimization**:
  - The loss function is convex.
  - Any local minimum is a global minimum.

  - Optimization algorithms are guaranteed to converge to the global minimum.

- **Non-Convex Optimization**:
  - The loss function is non-convex.
  - Multiple local minima and saddle points may exist.
  - Optimization algorithms may converge to local minima, which might not be optimal.

5.2. Implications for Neural Networks

Neural networks typically involve non-convex optimization due to:

- **Non-Linear Activation Functions**: Introduce non-linearity into the model.
- **Complex Architectures**: Multiple layers and interconnected neurons create a high-dimensional parameter space.

Strategies to Address Non-Convexity:

- **Good Initialization**: Helps in starting the optimization from a favorable point.
- **Advanced Optimization Algorithms**: Methods like Adam and RMSprop adapt learning rates to navigate complex loss landscapes.
- **Regularization Techniques**: Prevent overfitting and can smooth the loss surface.
- **Batch Normalization and Dropout**: Improve training stability and generalization.