DSA5106 Deep Learning: Foundations and Techniques

AY2025/26 Sem2 By Zhao Peiduo

Lecture1

A Concrete Definition of Learning A computer program is said to **learn from experience** E with respect to a class of tasks T and performance measure P if its performance on tasks in T, as measured by P, improves with experience E.
Linear Regression as a Canonical Example Task (T):

- Inputs: $x \in \mathbb{R}^d$
- Outputs: $y \in \mathbb{R}$
- Underlying relationship: $y = f^*(x)$
- Goal: predict outputs given inputs

Experience (E):

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N, \quad y_i = f^*(x_i) \text{ or } y_i = f^*(x_i) + \varepsilon_i$$

Hypothesis Space Before defining performance, we specify a hypothesis space:

$$\mathcal{H} = \{\text{collection of candidate functions } f\}$$

Performance (P):

$$\text{Distance}(f, f^*)$$

Learning Objective Learning consists of selecting a hypothesis closest to the true function:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \text{Distance}(f, f^*)$$

Loss Functions and Empirical Risk Distance is defined using a loss function L over the dataset:

$$R_{\text{emp}}(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N L(f(x_i), y_i)$$

Example: Squared Loss

$$L(y', y) = \frac{1}{2}(y' - y)^2$$

Empirical Risk Minimization (ERM) Improving performance with experience is formulated as:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} R_{\text{emp}}(f, \mathcal{D})$$

Least Squares Formulation Define the design matrix and output vector:

$$X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

ERM becomes:

$$\min_w \frac{1}{2N} \|Xw - y\|^2$$

Ordinary Least Squares Solution Setting the gradient to zero yields:

$$X^T(Xw - y) = 0$$

$$\hat{w} = (X^T X)^{-1} X^T y, \quad \hat{f}(x) = \hat{w}^T x$$

Extension to Affine Functions Linear models are often insufficient. A simple extension includes a bias term:

$$\mathcal{H} = \{f(x) = w^T x + b \mid w \in \mathbb{R}^d, \ b \in \mathbb{R}\}$$

Extension to Linear Basis Models To model nonlinear relationships, inputs are mapped into features:

$$\mathcal{H} = \left\{ f(x) = w^T \phi(x) = \sum_{i=1}^m w_i \phi_i(x) \right\}$$

The functions $\phi_1, \ldots, \phi_m$ are called **basis functions** or **feature maps**, chosen *a priori*.
Examples of Basis Functions (1D)

- Polynomial basis: $\phi_j(x) = x^j$
- Gaussian basis: $\phi_j(x) = \exp\left(-\frac{(x-m_j)^2}{2s^2}\right)$
- Sigmoid basis: $\phi_j(x) = \sigma\left(\frac{x-m_j}{s}\right)$, $\sigma(b) = \frac{1}{1+e^{-b}}$

Universality of Linear Basis Models
The capacity of linear basis models depends on **the choice of basis functions** and their number. With appropriate basis functions and $m \to \infty$, linear basis models are **universal**: they can approximate arbitrarily well any target function (under mild conditions). For instance:Fourier series and Weierstrass approximation theorem
Least Squares for Linear Basis Models
The least squares solution is obtained by modifying the design matrix. Original design matrix:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \quad \Rightarrow \quad \Phi = \begin{pmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_N) \end{pmatrix}$$

The empirical risk becomes:

$$R_{\text{emp}}(w) = \frac{1}{2N} \|\Phi w - y\|^2$$

Solution:

$$\hat{w} = (\Phi^T \Phi)^{-1} \Phi^T y$$

Model Capacity and Overfitting Using many basis functions greatly increases model flexibility:

$$\mathcal{H} = \left\{ f(x) = \sum_{j=0}^{99} w_j x^j \;\middle|\; w \in \mathbb{R}^{100} \right\}$$

While such models can fit training data extremely well,they often exhibit **overfitting**: they capture noise rather than the true underlying relationship, leading to poor generalization.
Learning $\neq$ Optimization The goal of learning is not merely minimizing training error, but performing well on **unseen data**.
Empirical Risk Minimization (What we can solve):

$$\min_{f \in \mathcal{H}} R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^N L(f(x_i), f^*(x_i)), \quad x_i \sim \mu$$

Population Risk Minimization (What we want):

$$\min_{f \in \mathcal{H}} R_{\text{pop}}(f) = \mathbb{E}_{x \sim \mu} \left[L(f(x), f^*(x)) \right]$$

The difference between these objectives is known as the **generalization gap**.
Estimating Population Risk
The true data distribution μ is unknown, but we observe samples through the dataset $\mathcal{D}$.
We therefore use a train-test split:

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$$

- $\mathcal{D}_{\text{train}}$: used for empirical risk minimization
- $\mathcal{D}_{\text{test}}$: used for final performance evaluation
- The test set must never influence training

Generalization
Generalization is a central challenge in machine learning.Key strategies to improve generalization include:

- Choice of model architecture
- Regularization
- Data augmentation
- Optimization methods

The Argmax View
Regression:

$$y = w^T \phi(x)$$

Classification (K classes):

$$y_i = w_i^T \phi(x), \quad i = 1, \ldots, K$$

Prediction is obtained by:

$$\hat{y} = \arg\max_{i=1,\ldots,K} y_i$$

This formulation is related to, but not equivalent to, support vector machines.
One-Hot Encoding and Softmax
Class labels are represented as one-hot vectors:

$$y_i = (0, \ldots, 1, \ldots, 0) \in \mathbb{R}^K$$

The model outputs class probabilities:

$$f(x_i) = (a_1, \ldots, a_K) \in \mathbb{R}^K$$

Softmax Function

$$s(z)_k = \frac{\exp(z_k)}{\sum_j \exp(z_j)}$$

Hypothesis space:

$$\mathcal{H} = \left\{ f(x) = s(W\phi(x)) \;\middle|\; W \in \mathbb{R}^{K \times m} \right\}$$

The output can be interpreted as a probability distribution over K classes.

Loss Functions for Classification

Although classification is naturally defined using zero–one loss, it is not suitable for optimization.
Instead, we use surrogate losses:

$$L : \mathbb{R}^K \times \mathbb{R}^K \to \mathbb{R}_+$$

Desired properties:

- $L(y, y') = 0$ if $y = y'$
- $L(y, y')$ is small if $y \approx y'$
- L is differentiable (almost everywhere)

Examples of Loss Functions for Classification

Even in classification, we can apply the squared loss:

$$L(y, y') = \frac{1}{2} \| y - y' \|^2$$

While this loss is simple and differentiable, it is often poorly aligned with the probabilistic interpretation of classification outputs.
In particular:

- It does not strongly penalize confident but incorrect predictions
- It treats probability vectors as Euclidean points rather than distributions
- It may lead to slower or unstable optimization

Cross-Entropy Loss

A more appropriate loss for probabilistic classification is the cross-entropy loss:

$$L(y, y') = -\sum_{k=1}^{K} y'_k \log y_k$$

When y' is a one-hot encoded label, this loss penalizes low predicted probability assigned to the correct class.

General Perspective

Squared loss and cross-entropy are not the only possible choices.
Any differentiable measure of discrepancy between two probability distributions can be used as a surrogate loss, provided it supports efficient optimization.