IEMS 402: Statistical Learning **Winter 2024-2025**

Lecture 1-2: Overview

Lecturer: Yiping Lu *Scribes: Josie Charles, Andrew Jin*

-1.1 Introduction to Supervised Learning

The goal in supervised learning is that given a set of data (called the training data) $(x_1, y_1), ..., (x_m, y_m)$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, we want to learn a predictor function $f : \mathcal{X} \to \mathcal{Y}$ (i.e. learn how to predict y given an input x). Usually, we assume (x_i, y_i) are i.i.d. from some distribution D and the function f comes from a particular class of functions, called the hypothesis class $\mathcal{H}$. The learning algorithm outputs a function from the hypothesis class $\mathcal{H}$.

Definition 1.1 (Realizable PAC Learning) A concept class $\mathcal{C}$ of target functions is PAC learnable (with respect to the hypothesis class $\mathcal{H}$) if there exists an algorithm A and a function $m_{\mathcal{C}}^A : (0,1)^2 \to \mathbb{N}$ with the following property:

Assume S=((x_1, y_1),...,(x_m, y_m)) is a sample of i.i.d. examples generated by some arbitrary distribution D such that $y_i = h(x_i)$ for some $h \in \mathcal{C}$ almost surely. If S is the input of A and $m > m_{\mathcal{C}}^A(\epsilon, \delta)$ then the algorithm returns a hypothesis $h_S^A \in \mathbb{H}$ such that, with probability 1- δ (over the choice of the m training examples):

$err(h_S^A) < \epsilon$

The function $m_{\mathcal{C}}^A : (0,1)^2 \to \mathbb{N}$ is called the "sample complexity" of the algorithm, and we aim to know the value of this function.

Once we learn our predictor function, how do we evaluate the quality of our predictor? This is where the Risk function comes in.

-1.2 Risk

We consider a fixed (testing) data distribution $p_{(x,y)}$ on $\mathcal{X} \times \mathcal{Y}$. We define a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, where $\ell(y, z)$ is the loss of predicting z while the true label is y. Then, we can define the *expected risk* (also referred to as *generalization error*, or *testing error* in the textbook) of a function $f : \mathcal{X} \to \mathcal{Y}$, as the expectation of the loss function between the output y and the prediction $f(x)$.

Definition 2.1 (Expected risk) *Given a prediction function $f : \mathcal{X} \to \mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, and a probability distribution p on $\mathcal{X} \times \mathcal{Y}$, the expected risk of f is defined as*

$$\mathcal{R}(f) = \mathbb{E}[\ell(y, f(x))] = \int_{\mathcal{X} \times \mathcal{Y}} \ell(y, f(x)) dp(x, y).$$

The risk depends on the distribution p on (x, y). We sometimes use the notation $\mathcal{R}_p(f)$ to make it explicit. The expected risk is our main performance criterion in this class.

Example 2.2 For classification, we may use the 0-1 loss function, defined

$$\ell(y, f(x)) = \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{otherwise} \end{cases}$$

Example 2.3 For regression, we may use the mean square error loss, defined

$$\ell(y, f(x)) = (y - f(x))^2$$

Be careful with the randomness, or lack thereof, of f: when performing learning from data, f will depend on the random training data, not on the testing data, and thus $\mathcal{R}(f)$ is typically random because of the dependence on the random training data. However, as a function on functions, the expected risk $\mathcal{R}$ is deterministic.

Note that sometimes we consider random predictions; that is, for any x, we output a distribution on y, and then the risk is taken as the expectation over the randomness of the outputs.

Averaging the loss on the training data defines the *empirical risk*, or *training error*.

Definition 2.2 (Empirical risk) *Given a prediction function $f : \mathcal{X} \to \mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, and data $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \ldots, n$, the empirical risk of f is defined as*

$$\hat{\mathcal{R}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)).$$

Note that $\hat{\mathcal{R}}$ is a random function on functions (and is often applied to random functions, with dependent randomness as both will depend on the training data).

The "hat" above the $\mathcal{R}$ in the empirical risk definition ($\hat{\mathcal{R}}$) denotes that we are estimating the risk using *empirical data.* This is a common notation in the field of Machine Learning, and will be used with other variables to similarly denote using empirical data.

-1.2.1 Bayes Risk and Bayes Predictor

Now that we have defined the performance criterion for supervised learning (the expected risk), the main question we tackle here is: What is the best prediction function f (regardless of the training data)?

Using the conditional expectation and its associated law of total expectation, we have

$$\mathcal{R}(f) = \mathbb{E}\left[\ell(y, f(x))\right] = \mathbb{E}\left[\mathbb{E}\left[\ell(y, f(x))\right] | x\right],$$

which we can rewrite, for a fixed $x' \in \mathcal{X}$:

$$\mathcal{R}(f) = \mathbb{E}_{x' \sim p}\left[\mathbb{E}\left[\ell(y, f(x')) | x = x'\right]\right] = \int_{\mathcal{X}} \mathbb{E}\left[\ell(y, f(x')) | x = x'\right] dp(x').$$

To distinguish between the random variable x and a value it may take, we use the notation x'.

From the conditional distribution given any $x' \in \mathcal{X}$ (i.e., $y|x = x'$), we can define the *conditional risk* for any $z \in \mathcal{Y}$, where z is a realized value of the random variable Y (it is a deterministic function of z and x'):

$$r(z|x') = \mathbb{E}\left[\ell(y, z) | x = x'\right],$$

which leads to

$$\mathcal{R}(f) = \int_{\mathcal{X}} r\left(f(x')|x'\right) dp(x').$$

To find a minimizing function $f : \mathcal{X} \to \mathbb{R}$, let us first assume that the set $\mathcal{X}$ is finite: in this situation, the risk can be expressed as a sum of functions that depends on a single value of f; that is, $\mathcal{R}(f) = \sum_{x' \in \mathcal{X}} r(f(x')|x')p(x = x')$. Therefore, *we can minimize with respect to each $f(x')$ independently.* Therefore, a minimizer of $\mathcal{R}(f)$ can be obtained by considering for any $x' \in \mathcal{X}$, the function value $f(x')$ to be equal to a minimizer $z \in \mathcal{Y}$ of $r(z|x') = \mathbb{E}\left[\ell(y, z)|x = x'\right]$. This extends beyond finite sets, as shown next.

We notes that minimizing the expected risk with respect to a function f in a restricted set does not lead to such decoupling.

Proposition -1.1 (Bayes predictor and Bayes risk) *The expected risk is minimized at a Bayes predictor $f^* : \mathcal{X} \to \mathcal{Y}$, satisfying for all $x' \in \mathcal{X}$,*

$$f^*(x') \in \arg\min_{z \in \mathcal{Y}} \mathbb{E}\left[\ell(y, z)|x = x'\right] = \arg\min_{z \in \mathcal{Y}} r(z|x'). \tag{-1.1}$$

The Bayes risk $\mathcal{R}^*$ is the risk of all Bayes predictors and is equal to

$$\mathcal{R}^* = \mathbb{E}_{x' \sim p}\left[\inf_{z \in \mathcal{Y}} \mathbb{E}\left[\ell(y, z)|x = x'\right]\right].$$

Proof We have

$$\mathcal{R}(f) - \mathcal{R}^* = \mathcal{R}(f) - \mathcal{R}(f_*) = \int_X \left[r(f(x')|x') - \min_{z \in \mathcal{Y}} r(z|x')\right] dp(x'),$$

which shows the proposition.

Note that (1) the Bayes predictor is not always unique, but that all lead to the same Bayes risk (e.g., in binary classification when $\mathbb{P}(y = 1|x) = 1/2$); and (2) that the Bayes risk is usually nonzero (unless the dependence between x and y is deterministic). Given a supervised learning problem, the Bayes risk is the optimal performance; we define the excess risk as the deviation with respect to the optimal risk.

Definition 2.3 (Excess risk) *The excess risk of a function $f : X \to Y$ is equal to $\mathcal{R}(f) - \mathcal{R}^*$ (it is always nonnegative because $\mathcal{R}^*$ minimizes the risk).*

Therefore, machine learning could be seen as trivial: *given* the distribution $y|x$ for any x, the optimal predictor is known and given by equation -1.1. The difficulty will be that this distribution is unknown.

-1.2.2 Empirical Risk Minimization

Consider a parameterized family of prediction functions (often referred to as models) $f_\theta : \mathcal{X} \to \mathcal{Y}$ for $\theta \in \Theta$ (typically a subset of a vector space). This class of learning methods aims at minimizing the empirical risk with respect to $\theta \in \Theta$:

$$\hat{\mathcal{R}}(f_\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_\theta(x_i)).$$

This defines an estimator $\hat{\theta} \in \arg\min_{\theta \in \Theta} \hat{\mathcal{R}}(f_\theta)$, and thus a prediction function $f_{\hat{\theta}} : \mathcal{X} \to \mathcal{Y}$.

The most classic example is linear least-squares regression, where we minimize $\frac{1}{n}\sum_{i=1}^n (y_i - \theta^\top \varphi(x_i))^2$, and f is linear in some feature vector $\varphi(x) \in \mathbb{R}^d$ (there is no need for $\mathcal{X}$ to be a vector space). The vector $\varphi(x)$ can be quite large (or even implicit, like in kernel methods). Other examples include neural networks.

- **Pros:**
  (1) Can be relatively easy to optimize (e.g., least-squares with its simple derivation and numerical algebra), many algorithms are available (primarily based on gradient descent);
  (2) Can be applied in any dimension (if a suitable feature vector is available).

- **Cons:**
  (1) can be relatively hard to optimize when the optimization formulation is not convex (e.g., neural networks);
  (2) need a suitable feature vector for linear methods;
  (3) the dependence on parameters can be complex (e.g., neural networks);
  (4) need some capacity control to avoid overfitting; and (5) require to parameterize functions with values in $\{0, 1\}$ (see chapter 4 for the use of convex surrogates).

Risk decomposition. The material in this section will be studied further in more detail in chapter 4.

- **Risk decomposition in estimation error + approximation error:** given any $\hat{\theta} \in \Theta$, we can write the excess risk of $f_{\hat{\theta}}$ as

$$\begin{aligned}\mathcal{R}(f_{\hat{\theta}}) - \mathcal{R}^* &= \left\{ \mathcal{R}(f_{\hat{\theta}}) - \inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'}) \right\} + \left\{ \inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'}) - \mathcal{R}^* \right\} \\ &= \text{estimation error} \quad + \quad \text{approximation error.}\end{aligned}$$

The approximation error $\{\inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'}) - \mathcal{R}^*\}$ is always nonnegative, does not depend on the chosen $f_{\hat{\theta}}$, and depends only on the class of functions parameterized by $\theta \in \Theta$. It is thus always a deterministic quantity, which characterizes the modeling assumptions made by the chosen class of functions. When Θ grows, the approximation error goes down to zero if arbitrary functions can be approximated arbitrarily well by functions f_θ. It is also independent of the number n of observations.

The estimation error $\left\{\mathcal{R}(f_{\hat{\theta}}) - \inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'})\right\}$ is also always nonnegative and is typically random because the function $f_{\hat{\theta}}$ is random. It typically decreases as n increases, and increases when Θ grows.

We can further decompose the estimation error. Let $\theta^* = \arg\min_{\theta' \in \Theta} \mathcal{R}(f_{\theta'})$ (that is, θ^* is the best parameter in Θ for minimizing the risk). Then, the estimation error can be decomposed into generalization error and optimization error.

$$\begin{aligned}\mathcal{R}(f_{\hat{\theta}}) - \mathcal{R}(f_{\theta^*}) &= \left\{ \mathcal{R}(f_{\hat{\theta}}) - \hat{\mathcal{R}}(f_{\hat{\theta}}) \right\} + \left\{ \hat{\mathcal{R}}(f_{\hat{\theta}}) - \hat{\mathcal{R}}(f_{\theta^*}) \right\} + \left\{ \hat{\mathcal{R}}(f_{\theta^*}) - \mathcal{R}(f_{\theta^*}) \right\} \\ &= \text{generalization error} + \text{optimization error} + \text{generalization error.}\end{aligned}$$

We will usually assume optimization error to be zero in this class (note that optimization error is always ≤ 0 because $\hat{\theta}$ minimizes the empirical risk). Then, we can bound generalization error using a uniform bound

$$\mathcal{R}(f_{\hat{\theta}}) - \hat{\mathcal{R}}(f_{\hat{\theta}}) + \hat{\mathcal{R}}(f_{\theta^*}) - \mathcal{R}(f_{\theta^*}) \leq \sup_{\theta' \in \Theta} 2|\mathcal{R}(f_{\theta'}) - \hat{\mathcal{R}}(f_{\theta'})|$$

Difference Between 401 (Statistics) and 402 (Learning)

- Parameter convergence vs Loss convergence
- Parametric Vs Non-parametric

-1.2.3 Design of Loss Function

Max Log Likelihood In statistical modeling and machine learning, the **log-likelihood** is a fundamental concept used to estimate model parameters. It serves as a foundation for designing loss functions, especially in the context of maximum likelihood estimation (MLE).

Given a set of independent and identically distributed (i.i.d.) data points $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and a statistical model parameterized by $\boldsymbol{\theta}$, the **likelihood function** is defined as:

$$\mathcal{L}(\boldsymbol{\theta} \mid \mathcal{D}) = \prod_{i=1}^{n} p(\mathbf{x}_i \mid \boldsymbol{\theta}) \tag{-1.2}$$

where $p(\mathbf{x}_i \mid \boldsymbol{\theta})$ is the probability density (or mass) function of the data point $\mathbf{x}_i$ given the parameters $\boldsymbol{\theta}$.

The goal of MLE is to find the parameter vector $\boldsymbol{\theta}^*$ that maximizes the log-likelihood:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta} \mid \mathcal{D}) \tag{-1.3}$$

In many machine learning algorithms, especially those involving optimization (like neural networks), we design a **loss function** to be minimized. Since MLE seeks to maximize the log-likelihood, minimizing the **negative log-likelihood** (NLL) serves as an appropriate loss function:

$$\mathcal{L}_{\text{loss}}(\boldsymbol{\theta} \mid \mathcal{D}) = -\ell(\boldsymbol{\theta} \mid \mathcal{D}) = -\sum_{i=1}^{n} \log p(\mathbf{x}_i \mid \boldsymbol{\theta}) \tag{-1.4}$$

Example (*Binary Classification with Logistic Regression*) Consider a binary classification problem where each data point $\mathbf{x}_i$ is associated with a binary label $y_i \in \{0, 1\}$. Using logistic regression, the probability of $y_i = 1$ given $\mathbf{x}_i$ and parameters $\boldsymbol{\theta}$ is:

$$p(y_i = 1 \mid \mathbf{x}_i, \boldsymbol{\theta}) = \sigma(\mathbf{x}_i^\top \boldsymbol{\theta}) = \frac{1}{1 + e^{-\mathbf{x}_i^\top \boldsymbol{\theta}}} \tag{-1.5}$$

where $\sigma(z)$ is the sigmoid function.

The likelihood for a single data point is:

$$p(y_i \mid \mathbf{x}_i, \boldsymbol{\theta}) = \sigma(\mathbf{x}_i^\top \boldsymbol{\theta})^{y_i} \left(1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\theta})\right)^{1-y_i} \tag{-1.6}$$

Thus, the log-likelihood for the entire dataset is:

$$\ell(\boldsymbol{\theta} \mid \mathcal{D}) = \sum_{i=1}^{n} \left[y_i \log \sigma(\mathbf{x}_i^\top \boldsymbol{\theta}) + (1 - y_i) \log \left(1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\theta})\right)\right] \tag{-1.7}$$

The corresponding loss function (negative log-likelihood) to be minimized is:

$$\mathcal{L}_{\text{loss}}(\boldsymbol{\theta} \mid \mathcal{D}) = -\ell(\boldsymbol{\theta} \mid \mathcal{D}) = -\sum_{i=1}^{n} \left[y_i \log \sigma(\mathbf{x}_i^\top \boldsymbol{\theta}) + (1 - y_i) \log\left(1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\theta})\right)\right] \tag{-1.8}$$

To simplify computations, especially when dealing with products of probabilities, we take the natural logarithm of the likelihood function, yielding the **log-likelihood**:

$$\ell(\boldsymbol{\theta} \mid \mathcal{D}) = \log \mathcal{L}(\boldsymbol{\theta} \mid \mathcal{D}) = \sum_{i=1}^{n} \log p(\mathbf{x}_i \mid \boldsymbol{\theta}) \tag{-1.9}$$

Example (*Gaussian with Learned Variance Leads to Sparsity*)

$$\begin{aligned}
\ell(\mu, \sigma^2) &= \sum_{i=1}^{n} \log P(y_i|\mu(x_i), \sigma(x_i)^2) \\
&= \sum_{i=1}^{n} \left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma(x_i)^2) - \frac{(y_i - \mu(x_i))^2}{2\sigma(x_i)^2}\right) \\
&= -\frac{n}{2}\ln(2\pi(x_i)) - \underbrace{\frac{n}{2}\ln(\sigma(x_i)^2)}_{\text{sparse regularization}} \quad \underbrace{-\sum_{i=1}^{n}\frac{(y_i - \mu(x_i))^2}{2\sigma(x_i)^2}}_{\text{weighted } \ell_2 \text{ loss}}
\end{aligned}$$

- A weighted ℓ_2 loss, when $\sigma = 0$ need perfectly learn the data

Example (*Max Log Likelihood of Weibull Distribution*) The likelihood function of a Weibull distribution is given by: $p_k(x|\lambda) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$, where $1 > k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter. In our paper, we assume the outbreak ratio follows a Weibull distribution $p(y|x) = p_k(x|\lambda(x)) = \frac{k}{\lambda(x)}\left(\frac{y}{\lambda(x)}\right)^{k-1} e^{-(y/\lambda(x))^k}$ where $x \in \mathbb{R}^d$ denotes the climate conditions while y is the outbreak rate we observed.

To build the new loss function, we deploy the log-likelihood method. The log-likelihood of Weibull distribution is shown as follows:

$$\log p_k(y|\lambda(x)) = -(y/\lambda(x))^k - k\log\lambda(x) + \underbrace{\log(ky^{k-1})}_{\text{not dependent on the prediction } \lambda(x)} \tag{-1.10}$$

We can simplify the full log-likelihood function into following function: $\max_{\lambda(x)} -(y/\lambda(x))^k - k\log\lambda(x) + \underbrace{\log(ky^{k-1})}_{\text{not dependent on parameter } \lambda(x)}$. Since $\log(ky^{k-1})$ is independent term from λ, it does not change as λ changes, thus it can be removed. If we take an inverse of this function and minimize it, it is equivalent to maximize it, which is the goal of the log likelihood. Thus, if our label is y and prediction ($\lambda(x)$) is $\hat{y}$, we define our loss function as $f(y, \hat{y}) = (y/\hat{y})^k + k\log(\hat{y})$ where y is the label provided in the dataset and $\hat{y}$ is the prediction of our model.

Fact $f(y, \lambda)$ attains its minimum at $\lambda = y$.

Convex Surrogate Loss Instead of learning $f : \mathcal{X} \to \{-1, 1\}$, we will thus learn a real-valued function $g : \mathcal{X} \to \mathbb{R}$ and define $f(x) = \text{sign}(g(x))$, where

$$\text{sign}(a) = \begin{cases} 1 & \text{if } a > 0 \\ -1 & \text{if } a < 0. \end{cases}$$

There are several conventions to define a prediction $f(x) \in \{-1, 1\}$ when $g(x) = 0$; a common one is to always choose one of the two labels. In this book, to preserve symmetry between -1 and 1 and to make sure that the loss function can be expressed as a function of $yg(x)$, we consider random predictions; that is, when $g(x) = 0$, the classifier $f(x)$ is sampled uniformly at random in $\{-1, 1\}$, independently from all other random quantities. When computing the loss incurred by the prediction $f(x)$, we will always take the expectation with respect to this random choice. Note that in practice, having $g(x)$ exactly equal to 0 occurs rarely, so the choice of convention makes no visible difference.

The 0–1 risk of function $f = \text{sign} \circ g$, still denoted as $\mathcal{R}(g)$ ($\triangle$ note the slight overloading of notations $\mathcal{R}(g) = \mathcal{R}(\text{sign} \circ g)$), is then equal to, separating between situations where $g(x) = 0$ or not,

$$\mathcal{R}(g) = \mathbb{P}(f(x) \neq y) = \mathbb{E}[1_{g(x)\neq 0} 1_{f(x)\neq y}] + \mathbb{E}[1_{g(x)=0} 1_{f(x)\neq y}]$$

$$= \mathbb{E}[1_{yg(x)<0}] + \frac{1}{2}\mathbb{E}[1_{g(x)=0}] = \mathbb{E}\left[\Phi_{0-1}(yg(x))\right],$$

where $\Phi_{0-1} : \mathbb{R} \to \mathbb{R}$ is defined as

$$\Phi_{0-1}(u) = \begin{cases} 1 & \text{if } u < 0 \\ \frac{1}{2} & \text{if } u = 0 \\ 0 & \text{if } u > 0. \end{cases}$$

A key concept in machine learning is the use of *convex surrogates*, where we replace Φ_{0-1} by another function Φ with better numerical properties (mostly convexity). See the classic examples discussed next and plotted in figure 4.1.

Instead of minimizing the classical risk $\mathcal{R}(g)$ or its empirical version, one then minimizes the Φ-risk (and its empirical version), defined as

$$\mathcal{R}_\Phi(g) = \mathbb{E}[\Phi(yg(x))].$$

In this context, the function g is sometimes called the *score function*.

The critical question tackled in this section is: Does it make sense to convexify the problem? In other words, does it lead to good predictions for the 0–1 loss?

Classical examples. We first review the primary examples used in practice:

- **Quadratic/square loss:** $\Phi(u) = (u-1)^2$, leading to, since we have $y^2 = 1$, $\Phi(yg(x)) = (y - g(x))^2 = (g(x) - y)^2$. We get back least-squares regression, ignore that the labels have to belong to $\{-1, 1\}$, and take the sign of $g(x)$ for the prediction. Note the overpenalization for a large positive value of $yg(x)$ that will not be present for the other losses discussed next (which are nonincreasing).

- **Logistic loss:** $\Phi(u) = \log(1 + e^{-u})$, leading to

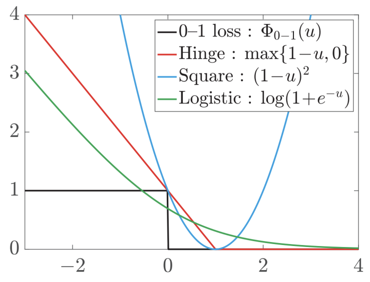
$$\Phi(yg(x)) = \log(1 + e^{-yg(x)}) = -\log\left(\frac{1}{1 + e^{-yg(x)}}\right) = -\log(\sigma(yg(x))),$$

where $\sigma(v) = \frac{1}{1+e^{-v}}$ is the sigmoid function. Note the link with maximum likelihood estimation, where we define the model through

$$\mathbb{P}(y = 1|x) = \sigma(g(x)) \quad \text{and} \quad \mathbb{P}(y = -1|x) = \sigma(-g(x)) = 1 - \sigma(g(x)).$$

The risk is, then, the negative conditional log-likelihood $\mathbb{E}[-\log p(y|x)]$. It is also often called the "cross-entropy loss." See more details about probabilistic methods in chapter 14.

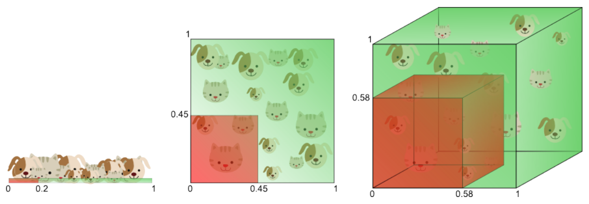
- **Hinge loss:** $\Phi(u) = \max(1 - u, 0)$. With linear predictors, this leads to the support vector machine (SVM), and $yg(x)$ is often called the "margin" in this context. This loss has a geometric interpretation (see section 4.1.2).[2]
- **Squared hinge loss:** $\Phi(u) = \max(1 - u, 0)^2$. This is a smooth counterpart to the regular hinge loss.
- **Exponential loss:** $\Phi(u) = \exp(-u)$. This loss is often used within the boosting framework presented in section 10.3, in particular through the Adaboost algorithm (section 10.3.4).



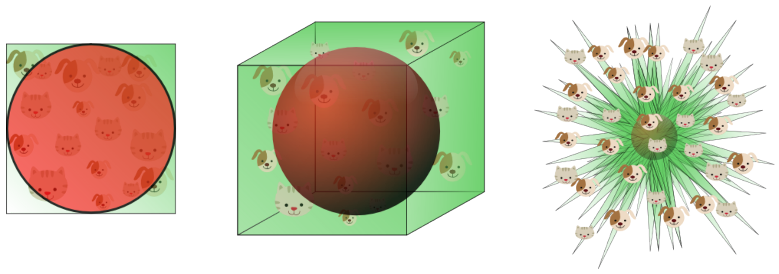
-1.3 Why the success of Deep Learning is a mystery

-1.3.1 Approximation:Curse of Dimensioanality

Let's say we want to train a classifier using only a single feature whose value ranges from 0 to 1. Let's assume that this feature is unique for each cat and dog. If we want our training data to cover 20% of this range, then the amount of training data needed is 20% of the complete population of cats and dogs. Now, if we add another feature, resulting in a 2D feature space, things change; To cover 20% of the 2D feature range, we now need to obtain 45% of the complete population of cats and dogs in each dimension ($0.45^2 = 0.2$). In the 3D case this gets even worse: to cover 20% of the 3D feature range, we need to obtain 58% of the population in each dimension ($0.58^3 = 0.2$).



This shows that the volume of the hypersphere tends to zero as the dimensionality tends to infinity, whereas the volume of the surrounding hypercube remains constant. This surprising and rather counter-intuitive observation partially explains the problems associated with the curse of dimensionality in classification: In high dimensional spaces, most of the training data resides in the corners of the hypercube defining the feature space. As mentioned before, instances in the corners of the feature space are much more difficult to classify than instances around the centroid of the hypersphere. This is illustrated by figure 11, which shows a 2D unit square, a 3D unit cube, and a creative visualization of an 8D hypercube which has $2^8 = 256$ corners:



Number of Parameters to Represent a C^s function in $\mathbb{R}^d$ To approximate a function f in the class $C^s(\mathbb{R}^d)$, which denotes the set of functions with continuous derivatives up to order s, within an error tolerance ϵ, we need to understand how the number of parameters N grows with respect to the function's smoothness s, the input dimension d, and the desired approximation accuracy ϵ.

The primary factors affecting N are:

- **Smoothness s:** Higher smoothness implies that the function has well-behaved derivatives up to order s, allowing for more compact representations of the function with fewer parameters.
- **Dimension d:** As the dimension d of the input space increases, more parameters are generally required to approximate the function accurately, due to the increase in complexity of variations in multiple dimensions.
- **Error Tolerance ϵ:** Smaller error tolerance (i.e., higher accuracy) requires a larger number of parameters, as finer details of the function need to be captured.

The number of parameters N required to achieve an approximation error of at most ϵ can be estimated by:

$$N \approx \left(\frac{1}{\epsilon}\right)^{\frac{d}{s}}$$

This relation is based on the following considerations:

- The term $\frac{1}{\epsilon}$ suggests that as we demand higher accuracy (smaller ϵ), the number of parameters increases.
- The exponent $\frac{d}{s}$ reflects the trade-off between the dimension d and the smoothness s of the function. When s is large (indicating higher smoothness), fewer parameters are needed for the same accuracy, as the function can be approximated more compactly.
- Conversely, if d is large (high-dimensional input), more parameters are required due to the increased complexity in capturing variations across all dimensions.

Proof Sketch: Number of Parameters to Approximate a C^s Function in $\mathbb{R}^d$ with ϵ Error To approximate a function f in $C^s(\mathbb{R}^d)$ with a desired accuracy ϵ, we aim to estimate the number of parameters N required to achieve this level of precision. We use a covering argument and regularity assumptions about f.

1. Partitioning the Domain Consider the input space $\mathbb{R}^d$ and restrict f to a bounded domain $\Omega \subset \mathbb{R}^d$. We divide Ω into smaller subdomains (or cells) of diameter h in each direction. Let M be the number of such subdomains, which scales as:

$$M \sim \left(\frac{1}{h}\right)^d .$$

2. Approximation on Each Subdomain On each subdomain, the function f can be approximated by a polynomial of degree s due to the smoothness assumption $f \in C^s$.

- Since f has continuous derivatives up to order s, a polynomial of degree s provides a good local approximation on each cell with an error that scales with h^s.
- To ensure that the global error does not exceed ϵ, we set $h^s \approx \epsilon$, giving:

$$h \approx \epsilon^{\frac{1}{s}} .$$

3. Estimating the Number of Parameters Each polynomial of degree s in d dimensions has approximately $(s+d)!\,/(s!\,d!\,)$ coefficients. However, for simplicity, we assume the number of parameters per cell is roughly constant with respect to s and d, giving a total of $O(1)$ parameters per cell. The constant here is $\Omega(\exp(d))$

Thus, the total number of parameters N required to approximate f with error ϵ is proportional to the number of cells M:

$$N \approx M \approx \left(\frac{1}{h}\right)^d \approx \left(\frac{1}{\epsilon^{\frac{1}{s}}}\right)^d = \left(\frac{1}{\epsilon}\right)^{\frac{d}{s}} .$$

-1.3.2 Generalization:Benigning Overfitting

-1.3.2.1 Degree of Freedom

- Even though the concept it represents is quite broad, degrees of freedom has a rigorous definition. Suppose that we observe

$$y_i = r(x_i) + \epsilon_i, \quad i = 1, \dots, n,$$

  where the errors ϵ_i, $i = 1, \dots, n$ are uncorrelated with common variance $\sigma^2 > 0$ (note: this is weaker than assuming $\epsilon_i \sim N(0, \sigma^2)$, i.i.d. for $i = 1, \dots, n$). Here we will treat the predictor measurements x_i, $i = 1, \dots, n$ as fixed (equivalently: consider conditioning on the values of the random predictors). Now consider the fitted values $\hat{y}_i = \hat{r}(x_i)$, $i = 1, \dots, n$ from a regression estimator $\hat{r}$. We define the degrees of freedom of $\hat{y}$ (i.e., the degrees of freedom of $\hat{r}$) as

$$\mathrm{df}(\hat{y}) = \frac{1}{\sigma^2} \sum_{i=1}^{n} \mathrm{Cov}(\hat{y}_i, y_i).$$

- It is going to be helpful for some purposes to rewrite the definition of degrees of freedom in matrix notation. This is

$$\mathrm{df}(\hat{y}) = \frac{1}{\sigma^2} \mathrm{tr}\left(\mathrm{Cov}(\hat{y}, y)\right),$$

  where we write $y = (y_1, \dots, y_n) \in \mathbb{R}^n$ and $\hat{y} = (\hat{y}_1, \dots, \hat{y}_n) \in \mathbb{R}^n$.

Examples include:

- Simple average estimator: consider $\hat{y}^{\text{ave}} = (\bar{y}, \ldots, \bar{y})$, where $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$. Then

$$\text{df}(\hat{y}^{\text{ave}}) = \frac{1}{\sigma^2}\sum_{i=1}^n \text{Cov}(\bar{y}, y_i) = \frac{1}{\sigma^2}\sum_{i=1}^n \frac{\sigma^2}{n} = 1,$$

  i.e., the effective number of parameters used by $\text{df}(\hat{y}^{\text{ave}})$ is just 1, which makes sense.

- Identity estimator: consider $\hat{y}^{\text{id}} = (y_1, \ldots, y_n)$. Then

$$\text{df}(\hat{y}^{\text{id}}) = \frac{1}{\sigma^2}\sum_{i=1}^n \text{Cov}(y_i, y_i) = n,$$

  i.e., $\hat{y}^{\text{id}}$ uses n effective parameters, which again makes sense.

Degree of Freedom for Linear Prediction

$$\hat{y}^{\text{linreg}} = X\hat{\beta} = X(X^TX)^{-1}X^Ty.$$

Here X is the $n \times p$ predictor matrix (with x_i along its i-th row). Then, relying on the matrix form of degrees of freedom,

$$\begin{aligned}
\text{df}(\hat{y}^{\text{linreg}}) &= \frac{1}{\sigma^2}\text{tr}\left(\text{Cov}\left(X(X^TX)^{-1}X^Ty, y\right)\right) \\
&= \frac{1}{\sigma^2}\text{tr}\left((X^TX)^{-1}X^T\,\text{Cov}(y,y)\right) \\
&= \text{tr}\left(X(X^TX)^{-1}X^T\right) \\
&= \text{tr}\left(X^TX(X^TX)^{-1}\right) \\
&= p.
\end{aligned}$$

So we have shown that the effective number of parameters used by $\hat{y}^{\text{linreg}}$ is p. This is highly intuitive, since we have estimated p regression coefficients.

Degree of Freedom can Estimate Error

- Remember, as described above, we're assuming the model

$$y_i = r(x_i) + \epsilon_i, \quad i = 1, \ldots, n,$$

  where $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$ satisfies $\mathbb{E}(\epsilon) = 0$ and $\text{Var}(\epsilon) = \sigma^2 I$ (this is a shorter way of writing that $\epsilon_i, i = 1, \ldots, n$ have mean zero, and are uncorrelated with marginal variance $\sigma^2 > 0$). Given an estimator $\hat{r}$ producing fitted values $\hat{y}_i = \hat{r}(x_i), i = 1, \ldots, n$, the expected training error of $\hat{r}$ is

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n (y_i - \hat{y}_i)^2\right].$$

Meanwhile, if

$$y_i' = r(x_i) + \epsilon_i', \quad i = 1, \ldots, n$$

is an independent test sample (important: note here that the predictors measurements $x_i, i = 1, \ldots, n$ are the same—i.e., we are considering these fixed) from the same distribution as our training sample, then

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i' - \hat{y}_i)^2\right]$$

is the expected test error.

- Recall that these two quantities—training and test errors—behave very differently, and it is usually the test error that we seek for the purposes of model validation or model selection. Interestingly, it turns out that (in this simple setup, with $x_i, i = 1, \ldots, n$ fixed) we have the relationship

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i' - \hat{y}_i)^2\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right] + \frac{2\sigma^2}{n}\,\mathrm{df}(\hat{y}).$$

  In words, the expected test error is exactly the expected training error plus a constant factor $\frac{2\sigma^2}{n}$ times the degrees of freedom.

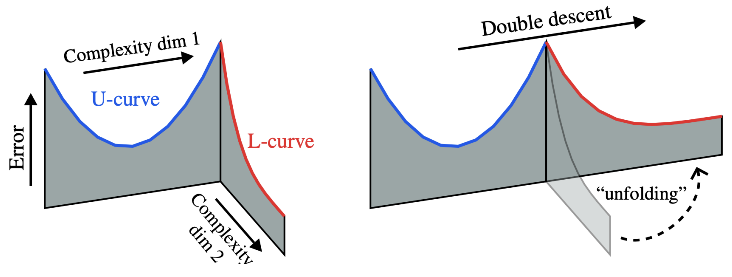
- From this decomposition, we can immediately see that with a larger degrees of freedom, i.e., a more complex fitting method, the test error is going to be larger than the training error. Perhaps more evocative is to rewrite the relationship above as

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i' - \hat{y}_i)^2\right] - \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right] = \frac{2\sigma^2}{n}\,\mathrm{df}(\hat{y}).$$

  We will call the left-hand side, the difference between expected test and training errors, the *optimism* of the estimator $\hat{r}$. We can see that it is precisely equal to $\frac{2\sigma^2}{n}$ times its degrees of freedom; so again, the higher the degrees of freedom, i.e., the more complex the fitting method, the larger the gap between testing and training errors.

-1.3.2.2 Double Descent

Conventional statistical wisdom established a well-understood relationship between model complexity and prediction error, typically presented as a U-shaped curve reflecting a transition between under- and overfitting regimes. However, motivated by the success of overparametrized neural networks, recent influential work has suggested this theory to be generally incomplete, introducing an additional regime that exhibits a second descent in test error as the parameter count p grows past sample size n – a phenomenon dubbed double descent.



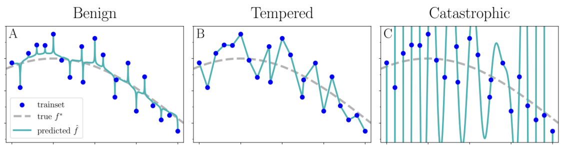
Back to U: Measuring effective parameters folds apparent double descent curves t once careful consideration is given to what is being plotted on the x-axes of their double descent plots, it becomes apparent that there are implicitly multiple, distinct complexity axes along which the parameter count grows. We demonstrate that the second descent appears exactly (and only) when and where the transition between these underlying axes occurs, and that its location is thus not inherently tied to the interpolation threshold $p = n$.

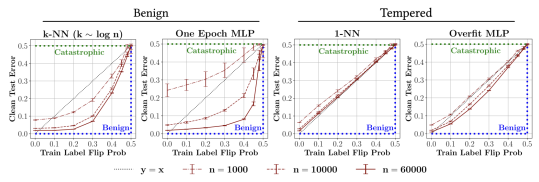
-1.3.2.3 Taxonomy of overfitting

We shall categorize learning procedures in terms of their *asymptotic expected risk*. We handle regression and K-class classification settings separately, due to their different loss scalings. As the number of samples $n \to \infty$, the sequence of expected risks $\{\mathcal{R}_n\}_n$ can behave in three different ways, as listed in Table 1. These three limiting behaviors define our taxonomy.

	Regression	**Classification**
Benign	$\lim_{n\to\infty} \mathcal{R}_n = R^*$	$\lim_{n\to\infty} \mathcal{R}_n = R^*$
Tempered	$\lim_{n\to\infty} \mathcal{R}_n \in (R^*, \infty)$	$\lim_{n\to\infty} \mathcal{R}_n \in (R^*, 1 - \frac{1}{K})$
Catastrophic	$\lim_{n\to\infty} \mathcal{R}_n = \infty$	$\lim_{n\to\infty} \mathcal{R}_n = 1 - \frac{1}{K}$

Table -1.1: Taxonomy of (over)fitting.





-1.3.3 Optimization

Non-convex Landscape

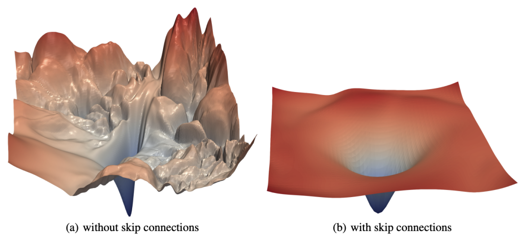


Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

The Polyak–Łojasiewicz (PŁ) condition is a property of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ which implies that the gradient descent method can converge to a global minimum, even if f is not strongly convex.

A differentiable function f satisfies the **PŁ condition** with parameter $\mu > 0$ if, for all $x \in \mathbb{R}^d$,

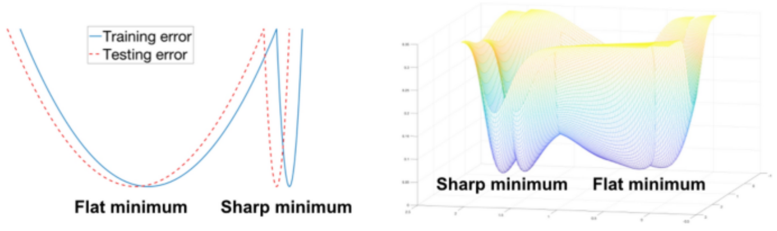
$$\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu\,(f(x) - f^*)\,,$$

where:

- $\|\nabla f(x)\|$ is the norm of the gradient of f at x,
- $f(x)$ is the function value at x,
- $f^* = \min_{x \in \mathbb{R}^d} f(x)$ is the global minimum value of f,
- $\mu > 0$ is a constant that characterizes the ”sharpness” of f around its minimum.

The PŁ condition implies that if f has a large gradient norm, then $f(x)$ is far from its minimum f^*. Consequently, gradient descent can achieve convergence rates similar to those of strongly convex functions even if f is not strongly convex.

Multiple minimum


Training error
Testing error
Flat minimum
Sharp minimum
Sharp minimum
Flat minimum