Machine Learning and Optimization

Lecture 4: Classification and Cross-Entropy Loss

CSCI 4140/6960 — Spring 2026

Course website: `cs.rpi.edu/~gittea/.../mlandopt.html`

Learning Objectives

By the end of this lecture, you should be able to:

1. **Derive** classification loss functions from probabilistic models using maximum likelihood
2. **Understand** entropy, cross-entropy, and KL divergence as measures of distributional (dis)similarity
3. **Explain** the equivalence between minimizing NLL, cross-entropy, and KL divergence
4. **Apply** this framework to derive binary cross-entropy (BCE) loss
5. **Extend** the framework to multi-class classification via softmax and categorical cross-entropy

Lecture Roadmap

Motivation and Problem Setup

Probabilistic Modeling for Binary Classification

Bridge: From Probabilistic Models to Optimization

Information Theory: Entropy, Cross-Entropy, and KL Divergence

Multi-Class Classification: Softmax and Categorical Cross-Entropy

Regularization for Classification

Summary and Connections

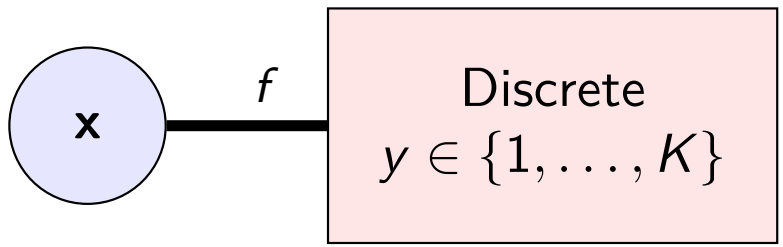
What is Classification?

Regression (Lecture 3): Predict continuous output $y \in \mathbb{R}$

- House prices, temperature, stock returns

Classification: Predict discrete output y from a finite set

- **Binary**: $y \in \{0, 1\}$ (or sometimes $y \in \{-1, +1\}$)
- **Multi-class**: $y \in \{1, 2, \ldots, K\}$ for $K \geq 3$

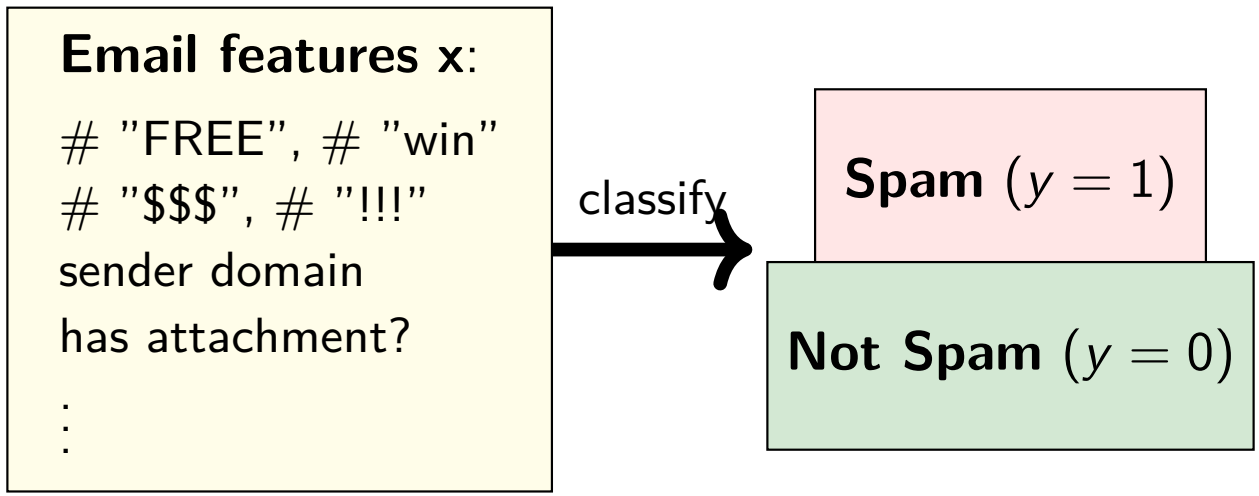


Assign to one of K categories

Difference from regression: Output space is *discrete*, not continuous

Example 1: Binary Classification — Spam Detection

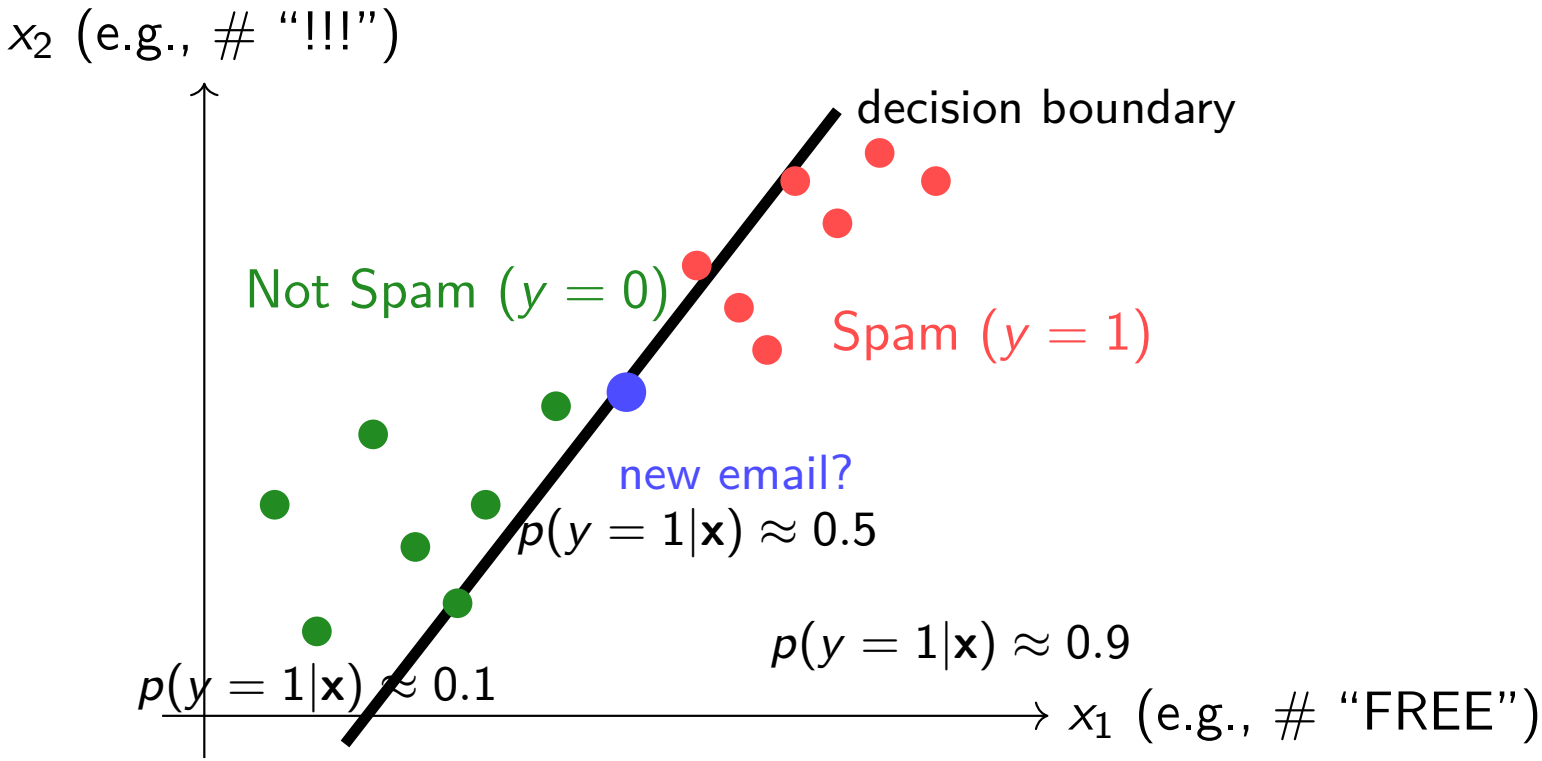
Task: Determine if an email is spam or not spam



Goal: Learn $f(\mathbf{x})$ that predicts $y \in \{0, 1\}$ from features $\mathbf{x}$

Running example: We'll use spam detection throughout for binary classification

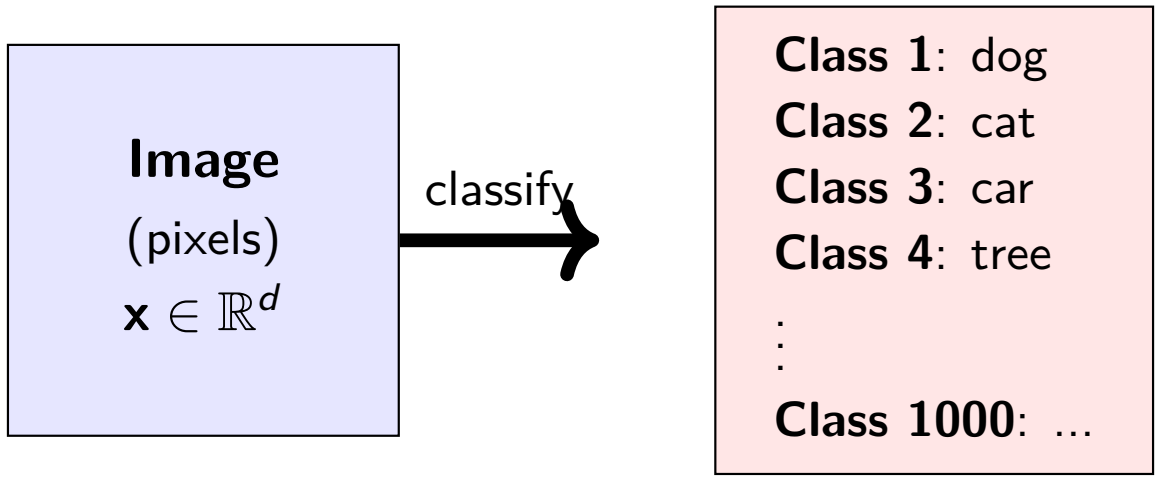
Visualizing Binary Classification: Spam Detection



Question: How do we learn probabilities and the decision boundary from training data?

Example 2: Multi-Class Classification — ImageNet

Task: Classify images into one of $K = 1000$ object categories



Goal: Learn $f(\mathbf{x})$ that predicts $y \in \{1, 2, \ldots, 1000\}$ from image $\mathbf{x}$

Running example: We'll use ImageNet throughout for multi-class classification

The Bernoulli Model for Binary Classification

Setup: $y \in \{0, 1\}$ (spam detection: 0 = not spam, 1 = spam)

Model: Assume $y \mid \mathbf{x}$ follows a Bernoulli distribution

$$p_{\boldsymbol{\theta}}(y \mid \mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x})^y (1 - p_{\boldsymbol{\theta}}(\mathbf{x}))^{1-y}$$

where $p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{P}(y = 1 \mid \mathbf{x})$ is the probability of class 1.

Why this model?

- Natural for binary outcomes (like Gaussian was natural for continuous)
- Gives us probabilities, not just labels
- Parameter $\boldsymbol{\theta}$ determines the distribution

Question: How do we parametrize $p_{\boldsymbol{\theta}}(\mathbf{x})$ as a function of $\mathbf{x}$?

We need: $p_{\boldsymbol{\theta}} : \mathbb{R}^d \to (0, 1)$ (features to valid probability)

From Log-Odds to the Logistic Sigmoid

Parametrization idea: Model the **log-odds** as a linear function

$$\log \frac{p(\mathbf{x})}{1 - p(\mathbf{x})} = \mathbf{w}^T \mathbf{x}$$

Why log-odds?

- Ranges over all of $\mathbb{R}$ (no constraints)
- Linear model in log-odds $\Rightarrow$ interpretable coefficients
- Standard approach in statistics (logit transformation)

Solving for $p(\mathbf{x})$:

$$\begin{aligned}
\log\left(p(\mathbf{x})/(1 - p(\mathbf{x}))\right) &= \mathbf{w}^T \mathbf{x} \\
p(\mathbf{x})/(1 - p(\mathbf{x}) &= e^{\mathbf{w}^T \mathbf{x}} \\
p(\mathbf{x}) &= (1 - p(\mathbf{x}))e^{\mathbf{w}^T \mathbf{x}} \\
p(\mathbf{x}) &= e^{\mathbf{w}^T \mathbf{x}} - p(\mathbf{x})e^{\mathbf{w}^T \mathbf{x}} \\
p(\mathbf{x})(1 + e^{\mathbf{w}^T \mathbf{x}}) &= e^{\mathbf{w}^T \mathbf{x}} \\
p(\mathbf{x}) &= \frac{e^{\mathbf{w}^T \mathbf{x}}}{1 + e^{\mathbf{w}^T \mathbf{x}}} = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}
\end{aligned}$$

The Logistic Sigmoid Function

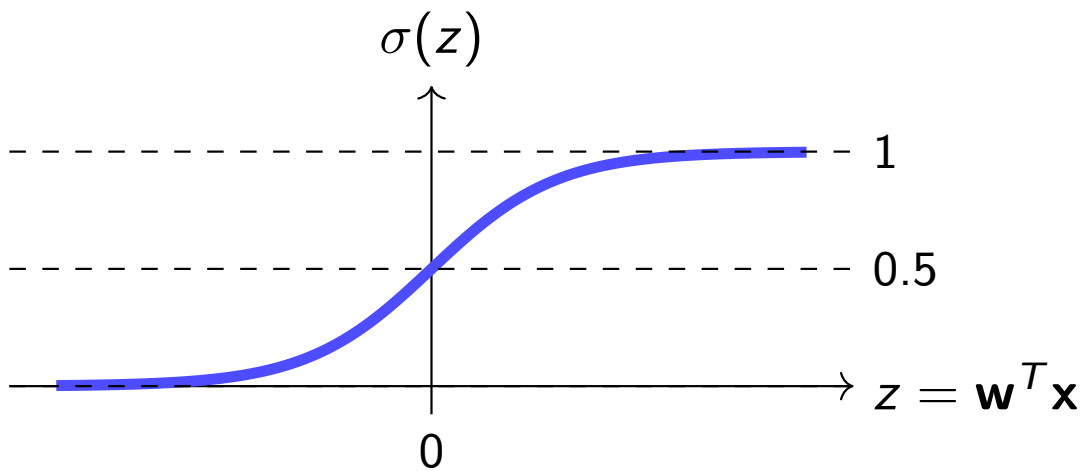
Definition: The **logistic sigmoid** is

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$$

Our model: $p_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$, with parameters $\boldsymbol{\theta} = \mathbf{w} \in \mathbb{R}^d$

Properties of sigmoid:

- Maps $\mathbb{R} \to (0, 1)$ (valid probability)
- $\sigma(0) = 0.5$ (decision boundary)
- $\sigma(z) \to 1$ as $z \to \infty$ (confident in class 1)
- $\sigma(z) \to 0$ as $z \to -\infty$ (confident in class 0)
- Smooth and differentiable everywhere
- Useful identity: $1 - \sigma(z) = \sigma(-z)$



Deriving the Loss: From Likelihood to NLL

Given: Training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $y_i \in \{0, 1\}$
Model: $p_{\mathbf{w}}(y \mid \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})^y(1 - \sigma(\mathbf{w}^T\mathbf{x}))^{1-y}$

Step 1: Likelihood (probability of observing the data)

$$L(\mathbf{w}) = \prod_{i=1}^n p_{\mathbf{w}}(y_i \mid \mathbf{x}_i) = \prod_{i=1}^n \sigma(\mathbf{w}^T\mathbf{x}_i)^{y_i}(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))^{1-y_i}$$

Step 2: Log-likelihood (take log, product becomes sum)

$$\ell(\mathbf{w}) = \log L(\mathbf{w}) = \sum_{i=1}^n \log p_{\mathbf{w}}(y_i \mid \mathbf{x}_i)$$

$$= \sum_{i=1}^n \left[y_i \log \sigma(\mathbf{w}^T\mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))\right]$$

Step 3: Negative log-likelihood (NLL) — this is our loss!

$$\mathcal{L}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^n \left[y_i \log \sigma(\mathbf{w}^T\mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))\right]$$

This is **binary cross-entropy (BCE) loss**!

Binary Cross-Entropy (BCE) Loss

Definition: The **binary cross-entropy loss** is:

$$\mathcal{L}_{\text{BCE}}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log \sigma(\mathbf{w}^T\mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}^T\mathbf{x}_i))\right]$$

Compact notation: Let $p_i = \sigma(\mathbf{w}^T\mathbf{x}_i)$. Then for each example:

$$\ell_i = -y_i \log p_i - (1-y_i)\log(1-p_i)$$

Interpretation per example:

- If $y_i = 1$ (spam): loss is $-\log p_i$ (encourages $p_i \to 1$, i.e., high probability of spam)
- If $y_i = 0$ (not spam): loss is $-\log(1-p_i)$ (encourages $p_i \to 0$, i.e., low probability of spam)

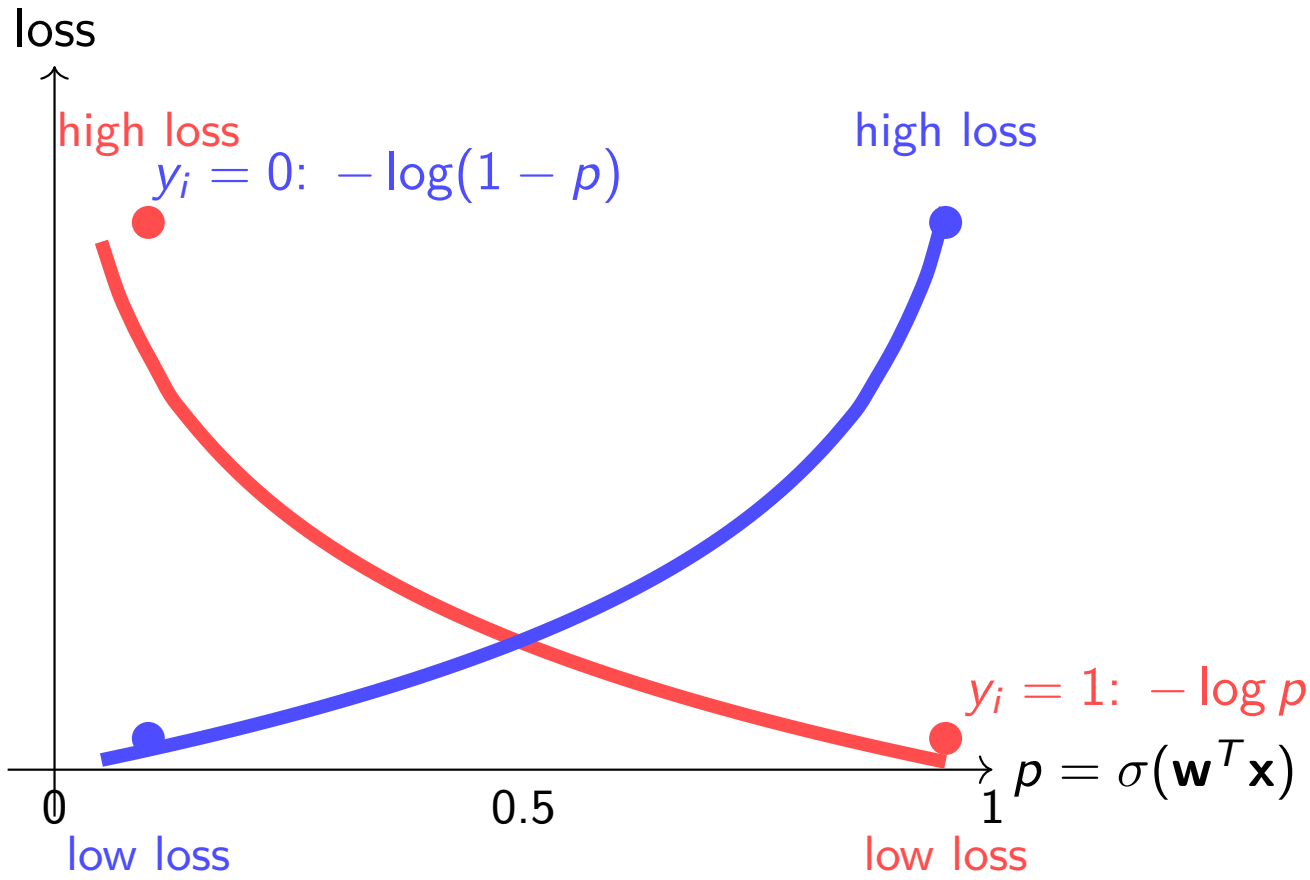
The optimization problem:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \mathcal{L}_{\text{BCE}}(\mathbf{w})$$

This is **maximum likelihood estimation (MLE)** for the Bernoulli model!

Visualizing Binary Cross-Entropy Loss

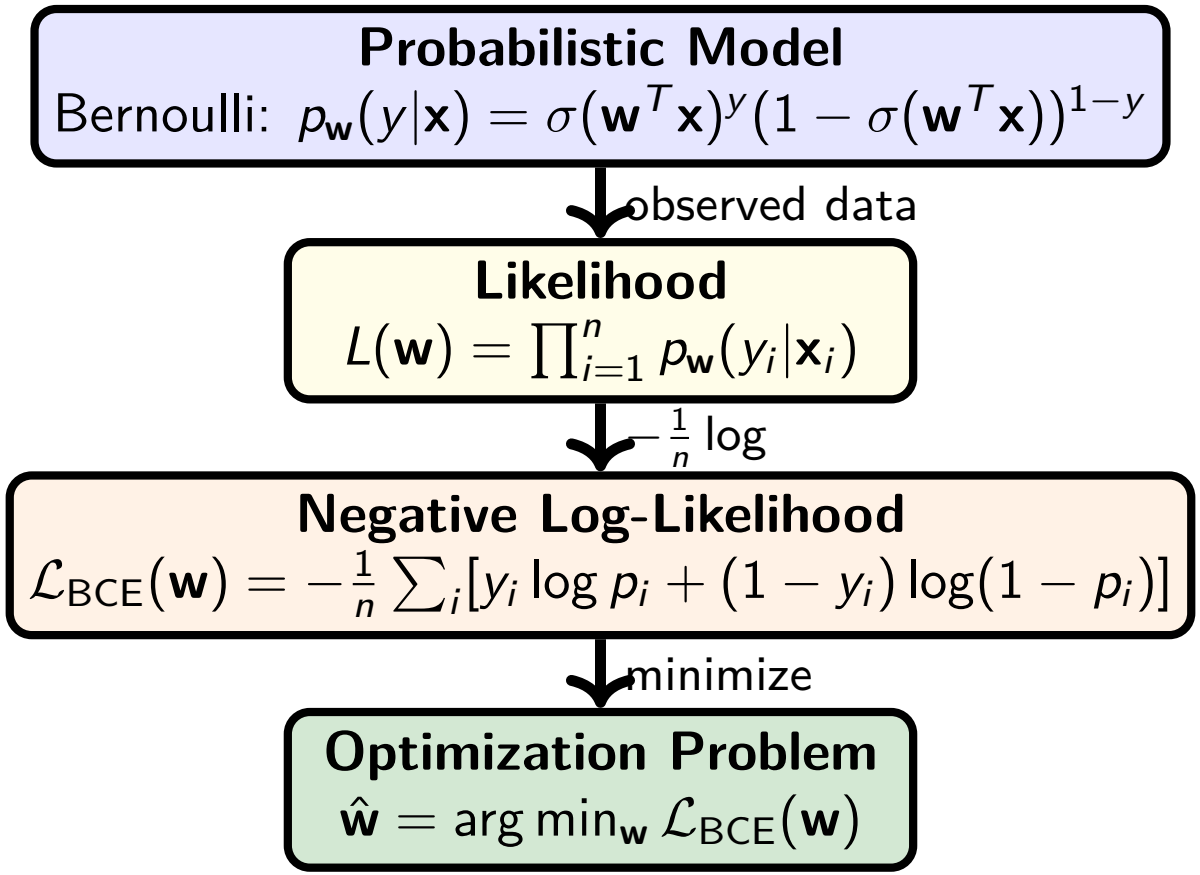
Per-example loss as function of predicted probability:



Key observations:

- Loss is 0 when prediction is perfectly confident and correct
- Loss $\to \infty$ when prediction is perfectly confident and *wrong*
- Heavily penalizes confident wrong predictions

Summary: Binary Classification Framework



This same recipe will work for multi-class!

But first, let's understand *why* this loss has the right form using information theory...

The General Recipe: Probabilistic Model → Loss Function

We just saw this recipe for binary classification. It's completely general:

1. **Choose a probabilistic model** for $p_\theta(y \mid \mathbf{x})$
   - Binary: Bernoulli ⇒ sigmoid
   - Multi-class: Categorical ⇒ softmax (coming soon!)
   - Regression: Gaussian ⇒ linear model (Lecture 3)
2. **Write down the likelihood** over the dataset

$$L(\theta) = \prod_{i=1}^{n} p_\theta(y_i \mid \mathbf{x}_i)$$

3. **Take the log** ⇒ product becomes sum

$$\log L(\theta) = \sum_{i=1}^{n} \log p_\theta(y_i \mid \mathbf{x}_i)$$

4. **Negate and average** ⇒ empirical risk (loss)

$$\mathcal{L}(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log p_\theta(y_i \mid \mathbf{x}_i)$$

5. **Minimize** $\mathcal{L}(\theta)$ ⇒ optimization problem!

Why Does This Recipe Work?

This gives us the MLE view.

But there's a deeper question:

Why is MLE the "right" loss?

Answer: There's another perspective from **information theory** that explains why this loss has the right structure.

Information theory gives us:

- A way to measure "closeness" between probability distributions
- A principled notion of what it means for a model to "match" the data
- Insight into why we use $-\log$ (not squared error, not absolute error)

Next: We'll develop the concepts of entropy, cross-entropy, and KL divergence, then see how they connect back to the loss we just derived.

Surprise and Information Content

Question: How do we measure the "surprise" of observing outcome k?

Consider a discrete random variable with outcomes $\{1, \ldots, K\}$ and probabilities $p_1, \ldots, p_K$.

Intuition:

- Rare events (p_k small) are more surprising
- Common events (p_k large) are less surprising
- Surprise should combine additively for independent events

Definition: The **information content** or **surprise** of observing outcome k is:

$$I(k) = -\log_2 p_k \quad \text{(measured in bits)}$$

Why $-\log$?

- $p_k = 1$ (certain): $-\log_2(1) = 0$ bits (no surprise)
- $p_k = 1/2$: $-\log_2(1/2) = 1$ bit (one coin flip worth of uncertainty)
- $p_k = 1/4$: $-\log_2(1/4) = 2$ bits (two coin flips)
- p_k small: $-\log_2 p_k$ large (very surprising)

Entropy: Expected Surprise

Definition: The **entropy** of distribution p is the *expected surprise*:

$$H(p) = \mathbb{E}_{k\sim p}[I(k)] = \mathbb{E}_{k\sim p}[-\log_2 p_k] = -\sum_{k=1}^{K} p_k \log_2 p_k$$

(Note: In practice we use natural log ln, then units are "nats" instead of "bits")

Interpretation:

- Average number of bits needed to encode samples from p
- Measures the inherent uncertainty in the distribution
- Shannon: This is the fundamental limit of compression

Examples:

- **Deterministic:** $p = [1, 0, 0]$ has $H(p) = 0$ (no surprise, always same)
- **Uniform:** $p = [1/3, 1/3, 1/3]$ has $H(p) = \log_2 3 \approx 1.58$ bits (maximum uncertainty)

Key property: For fixed K, entropy is maximized by the uniform distribution.

Cross-Entropy: Using the Wrong Code

Setup: True distribution is p, but we use a code designed for distribution q

When outcome k occurs (which happens with probability p_k):

- True surprise: $-\log p_k$ bits (optimal code for p)
- Surprise when using wrong code: $-\log q_k$ bits (code designed for q)

Definition: The **cross-entropy** $H(p, q)$ is the *expected surprise when encoding samples from p using a code designed for q*:

$$H(p, q) = \mathbb{E}_{k \sim p}[-\log q_k] = -\sum_{k=1}^{K} p_k \log q_k$$

Key property: Cross-entropy is minimized when $q = p$:

$$H(p, q) \geq H(p, p) = H(p)$$

with equality iff $p = q$.

Interpretation: Average bits needed to encode samples from p using a code optimized for q. We always pay at least $H(p)$ bits on average, with additional cost if $q \neq p$.

KL Divergence: The Extra Cost of Using the Wrong Code

Question: How much *extra* coding cost do we pay by using q instead of p?

Definition: The **Kullback-Leibler (KL) divergence** is:

$$D_{\mathrm{KL}}(p\|q) = H(p,q) - H(p) = \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k}$$

Interpretation:

- $H(p,q)$: bits needed using code for q
- $H(p)$: bits needed using optimal code for p
- $D_{\mathrm{KL}}(p\|q)$: *extra* bits wasted by using wrong code

Key properties:

- $D_{\mathrm{KL}}(p\|q) \geq 0$ always (Gibbs' inequality)
- $D_{\mathrm{KL}}(p\|q) = 0$ iff $p = q$
- Not symmetric: $D_{\mathrm{KL}}(p\|q) \neq D_{\mathrm{KL}}(q\|p)$ in general
- Not a metric (doesn't satisfy triangle inequality)

Crucial observation: Since $H(p)$ doesn't depend on q:

$$\arg\min_q H(p,q) = \arg\min_q D_{\mathrm{KL}}(p\|q)$$

Cross-Entropy and KL Divergence in Machine Learning

Connection to classification:

In machine learning:

- p = true data distribution (from observed labels)
- $q = p_\theta$ = model's predicted distribution (parametrized by θ)

Goal: Find θ that minimizes $H(p, p_\theta)$ or equivalently $D_{\mathrm{KL}}(p \| p_\theta)$
This makes the model distribution p_θ as close as possible to the true distribution p.

Why this matters:

- Minimizing cross-entropy $\Leftrightarrow$ Making model match reality
- Provides principled loss function for classification
- Generalizes naturally from binary to multi-class
- Explains why we use $-\log$ (information-theoretic optimality)

Next: We'll see the exact connection to the losses we derived earlier.

Understanding δ_{y_i}: The True Label Distribution

Question: For a training example $(\mathbf{x}_i, y_i)$, what is the "true distribution" p?

When we observe label $y_i \in \{1, \ldots, K\}$, the true distribution is **deterministic**:

$$\delta_{y_i}(k) = \begin{cases} 1 & \text{if } k = y_i \\ 0 & \text{otherwise} \end{cases}$$

This is a **point mass** or **Dirac delta** at y_i: all probability on the observed label.

One-hot encoding: Represent δ_{y_i} as a vector $\mathbf{e}_{y_i} \in \mathbb{R}^K$:

$$\mathbf{e}_{y_i} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 1 \text{ in position } y_i$$

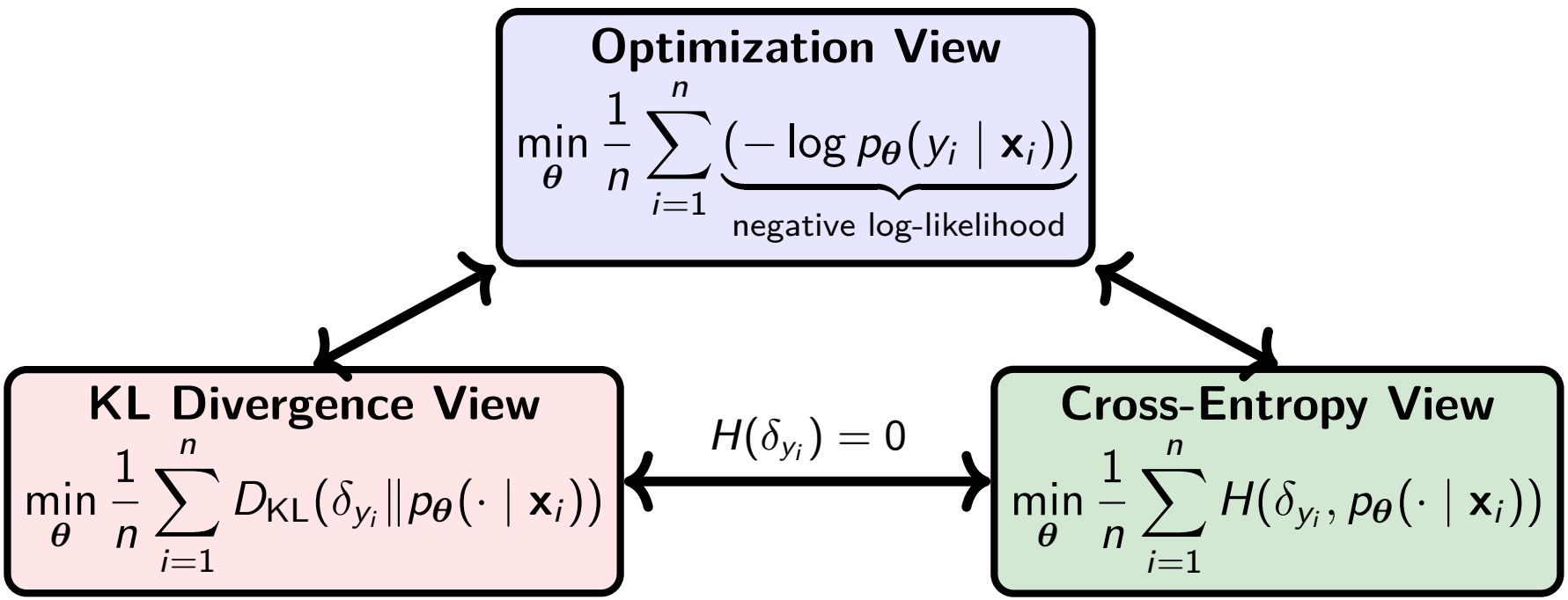
Note: $H(\delta_{y_i}) = 0$ (deterministic distribution has zero entropy)

The Fundamental Equivalence for Classification

Setup:

- Training data: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $y_i \in \{1, \ldots, K\}$
- Parametric model: $p_{\boldsymbol{\theta}}(y \mid \mathbf{x})$
- For each example i, true distribution is $p_i^{\text{true}} = \delta_{y_i}$ (one-hot at y_i)

Three equivalent optimization problems:



Proof of Equivalence

For a single example $(\mathbf{x}_i, y_i)$ with true distribution δ_{y_i}:

Cross-entropy:

$$\begin{aligned} H(\delta_{y_i}, p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}_i)) &= -\sum_{k=1}^{K} \delta_{y_i}(k) \log p_{\boldsymbol{\theta}}(k \mid \mathbf{x}_i) \\ &= -\log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i) \end{aligned}$$

since $\delta_{y_i}(k) = 1$ if $k = y_i$ and 0 otherwise (only one term survives).

KL divergence:

$$\begin{aligned} D_{\mathrm{KL}}(\delta_{y_i} \| p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}_i)) &= H(\delta_{y_i}, p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}_i)) - H(\delta_{y_i}) \\ &= -\log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i) - 0 \\ &= -\log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i) \end{aligned}$$

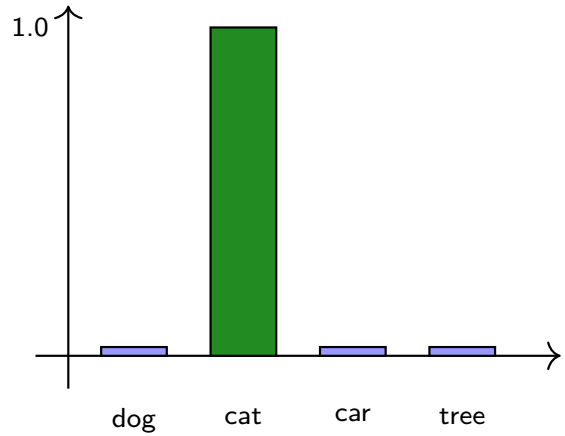
since $H(\delta_{y_i}) = 0$ (deterministic distribution has zero entropy).

All three views give the same loss: $-\log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i)$

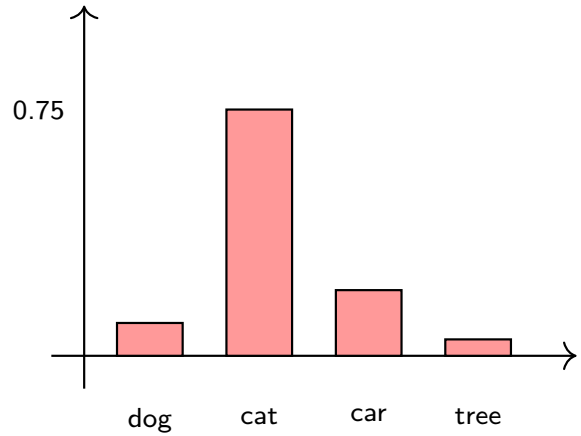
Visualizing True vs Model Distributions

Example: Image of a cat, true label $y_i = 2$ (cat), with $K = 4$ classes

True $\delta_{y_i=2}$

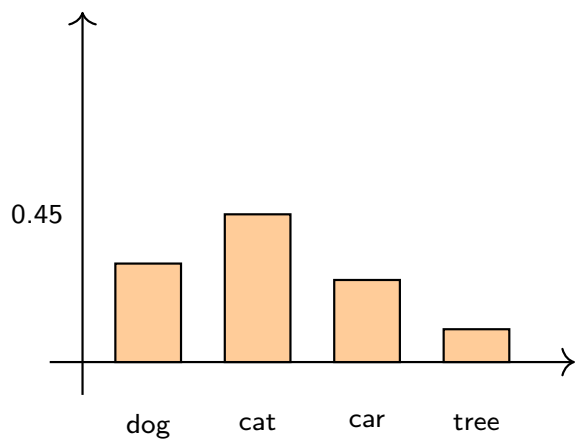


Model 1 (good)



Loss $= -\ln(0.75) \approx 0.29$

Worse model (less certain about correct class):



Loss $= -\ln(0.45) \approx 0.80$

Connecting Back to Binary Classification

For binary classification ($K = 2$), we have:

- True distribution: δ_{y_i} where $y_i \in \{0, 1\}$
- Model distribution: $p_{\mathbf{w}}(y \mid \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})^y(1 - \sigma(\mathbf{w}^T\mathbf{x}))^{1-y}$

The loss per example:

$$\begin{aligned}\ell_i &= -\log p_{\mathbf{w}}(y_i \mid \mathbf{x}_i) \\ &= H(\delta_{y_i}, p_{\mathbf{w}}(\cdot \mid \mathbf{x}_i)) \\ &= D_{\mathrm{KL}}(\delta_{y_i} \| p_{\mathbf{w}}(\cdot \mid \mathbf{x}_i)) \\ &= -y_i \log \sigma(\mathbf{w}^T\mathbf{x}_i) - (1 - y_i) \log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))\end{aligned}$$

This is exactly the BCE loss we derived earlier!

The information-theoretic view explains *why* BCE is the right loss: it measures how well our model's probability distribution matches the true (observed) distribution.

From Binary to Multi-Class: Requirements

Binary case: Two outcomes, model $p(y = 1 \mid \mathbf{x})$ as single probability

Multi-class: K outcomes, need to model $p(y = k \mid \mathbf{x})$ for $k = 1, \ldots, K$

Requirements for valid probability distribution:

- $p(y = k \mid \mathbf{x}) > 0$ for all k (valid probabilities)
- $\sum_{k=1}^{K} p(y = k \mid \mathbf{x}) = 1$ (normalization)
- Should reduce to binary case when $K = 2$

These constraints define the **probability simplex**:

$$\Delta^{K-1} = \left\{ \mathbf{p} \in \mathbb{R}^K : p_k \geq 0 \text{ for all } k, \sum_{k=1}^{K} p_k = 1 \right\}$$

Naive approaches don't work:

- K independent sigmoids: Outputs don't sum to 1
- One-vs-all then normalize: No principled probabilistic interpretation

Solution: The **softmax** transformation: $\mathbb{R}^K \to \Delta^{K-1}$

The Softmax Transformation

Definition: The **softmax function** maps scores (logits) to probabilities:

$$\text{softmax}(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } k = 1, \ldots, K$$

where $\mathbf{z} = [z_1, \ldots, z_K]^T \in \mathbb{R}^K$ are the **logits** (unnormalized log-probabilities).

Properties:

- Output: $\text{softmax}(\mathbf{z}) \in \Delta^{K-1}$ (valid probability distribution)
- Preserves order: if $z_i > z_j$ then $\text{softmax}(\mathbf{z})_i > \text{softmax}(\mathbf{z})_j$
- Smooth and differentiable everywhere
- Translation invariant: $\text{softmax}(\mathbf{z} + c\mathbf{1}) = \text{softmax}(\mathbf{z})$ for any c

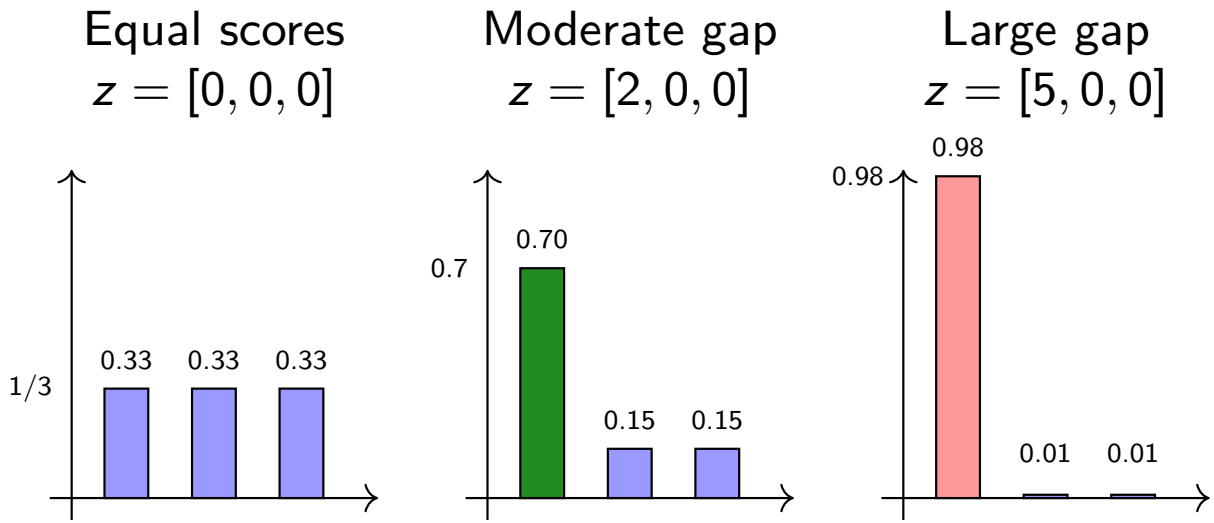
Why "softmax"?

- Smooth approximation to max (one-hot encoding)
- As score gaps increase, probability concentrates on highest score
- When $K = 2$: $\text{softmax}([z, 0]) = [\sigma(z), \sigma(-z)]$ (reduces to sigmoid!)

Visualizing Softmax Concentration

How does softmax behave as score differences change?

Fix two scores at 0, vary the third:



Key insights:

- Equal scores $\Rightarrow$ Uniform distribution (maximum uncertainty)
- Moderate gap $\Rightarrow$ Clear preference, but others still non-zero
- Large gap $\Rightarrow$ Nearly one-hot (approaching deterministic)

Softmax smoothly interpolates from uniform to deterministic as confidence grows.

This is exactly the behavior we want for classification!

The Categorical Model for Multi-Class Classification

Setup: $y \in \{1, \ldots, K\}$ (ImageNet: $K = 1000$ classes)

Model: Assume $y \mid \mathbf{x}$ follows a categorical distribution with probabilities given by softmax:

$$p_{\boldsymbol{\Theta}}(y = k \mid \mathbf{x}) = \text{softmax}(\boldsymbol{\Theta}\mathbf{x})_k = \frac{e^{\boldsymbol{\theta}_k^T \mathbf{x}}}{\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T \mathbf{x}}}$$

Parameters: $\boldsymbol{\Theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K]^T \in \mathbb{R}^{K \times d}$

- Each class k has its own weight vector $\boldsymbol{\theta}_k \in \mathbb{R}^d$
- Logit for class k: $z_k(\mathbf{x}) = \boldsymbol{\theta}_k^T \mathbf{x}$
- Softmax converts logits to probabilities

Names for this model:

- **Softmax regression**
- **Multinomial logistic regression**
- **Multi-class logistic regression**

All refer to the same model!

One-Hot Encoding and Multi-Class Labels

Setup: For training example $(\mathbf{x}_i, y_i)$ where true label is $y_i = 2$ (cat) and $K = 4$ classes

The model's loss on example i is $-\log p_{\boldsymbol{\Theta}}(y_i \mid \mathbf{x}_i)$. One-hot encoding lets us write this compactly as $-\log(\mathbf{e}_{y_i}^T p_{\boldsymbol{\Theta}}(y_i \mid \mathbf{x}_i))$ for any K.

Representing the true label:
The true distribution δ_{y_i} is a **one-hot vector**:

$$\mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \text{dog} \\ \text{cat} \leftarrow \text{true class} \\ \text{car} \\ \text{tree} \end{matrix}$$

All probability on the observed class, zero elsewhere.

Extracting the model's prediction:
Model outputs probability vector:

$$\mathbf{p} = \begin{bmatrix} 0.1 \\ 0.6 \\ 0.25 \\ 0.05 \end{bmatrix} \begin{matrix} \text{dog} \\ \text{cat} \\ \text{car} \\ \text{tree} \end{matrix}$$

Inner product picks out the probability assigned to the true class:

$$\mathbf{e}_2^T \mathbf{p} = [0, 1, 0, 0] \cdot \mathbf{p} = 0.6$$

Deriving Multi-Class Loss: From Likelihood to NLL

Given: Training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $y_i \in \{1, \ldots, K\}$

Model: $p_{\boldsymbol{\Theta}}(y = k \mid \mathbf{x}) = \frac{e^{\boldsymbol{\theta}_k^T \mathbf{x}}}{\sum_{j=1}^K e^{\boldsymbol{\theta}_j^T \mathbf{x}}}$

Step 1: Likelihood

$$L(\boldsymbol{\Theta}) = \prod_{i=1}^n p_{\boldsymbol{\Theta}}(y_i \mid \mathbf{x}_i) = \prod_{i=1}^n \frac{e^{\boldsymbol{\theta}_{y_i}^T \mathbf{x}_i}}{\sum_{j=1}^K e^{\boldsymbol{\theta}_j^T \mathbf{x}_i}}$$

Step 2: Log-likelihood

$$\log L(\boldsymbol{\Theta}) = \sum_{i=1}^n \log p_{\boldsymbol{\Theta}}(y_i \mid \mathbf{x}_i) = \sum_{i=1}^n \left[\boldsymbol{\theta}_{y_i}^T \mathbf{x}_i - \log \left(\sum_{j=1}^K e^{\boldsymbol{\theta}_j^T \mathbf{x}_i} \right) \right]$$

Step 3: Negative log-likelihood (NLL)

$$\mathcal{L}_{\text{CE}}(\boldsymbol{\Theta}) = -\frac{1}{n} \sum_{i=1}^n \left[\boldsymbol{\theta}_{y_i}^T \mathbf{x}_i - \log \left(\sum_{j=1}^K e^{\boldsymbol{\theta}_j^T \mathbf{x}_i} \right) \right]$$

This is **categorical cross-entropy loss**!

Categorical Cross-Entropy Loss

Definition: The **categorical cross-entropy loss** is:

$$\boxed{\mathcal{L}_{\text{CE}}(\boldsymbol{\Theta}) = -\frac{1}{n}\sum_{i=1}^{n}\left[\boldsymbol{\theta}_{y_i}^T\mathbf{x}_i - \log\left(\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T\mathbf{x}_i}\right)\right]}$$

Equivalently:

$$\mathcal{L}_{\text{CE}}(\boldsymbol{\Theta}) = -\frac{1}{n}\sum_{i=1}^{n}\log p_{\boldsymbol{\Theta}}(y_i \mid \mathbf{x}_i)$$

Interpretation of each term:

- $\boldsymbol{\theta}_{y_i}^T\mathbf{x}_i$: Reward for logit of true class (want this large)
- $\log\left(\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T\mathbf{x}_i}\right)$: Normalization penalty (log-partition function)

The optimization problem:

$$\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}\in\mathbb{R}^{K\times d}} \mathcal{L}_{\text{CE}}(\boldsymbol{\Theta})$$

Connecting to Information Theory

For multi-class, the three views are equivalent:

1. **Optimization view (NLL):**

$$\ell_i = -\log p_{\boldsymbol{\Theta}}(y_i \mid \mathbf{x}_i) = -\log \frac{e^{\boldsymbol{\theta}_{y_i}^T \mathbf{x}_i}}{\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T \mathbf{x}_i}}$$

2. **Cross-entropy view:**

$$\begin{aligned}\ell_i = H(\delta_{y_i}, p_{\boldsymbol{\Theta}}(\cdot \mid \mathbf{x}_i)) &= -\sum_{k=1}^{K} \delta_{y_i}(k) \log p_{\boldsymbol{\Theta}}(k \mid \mathbf{x}_i) \\ &= -\log p_{\boldsymbol{\Theta}}(y_i \mid \mathbf{x}_i)\end{aligned}$$

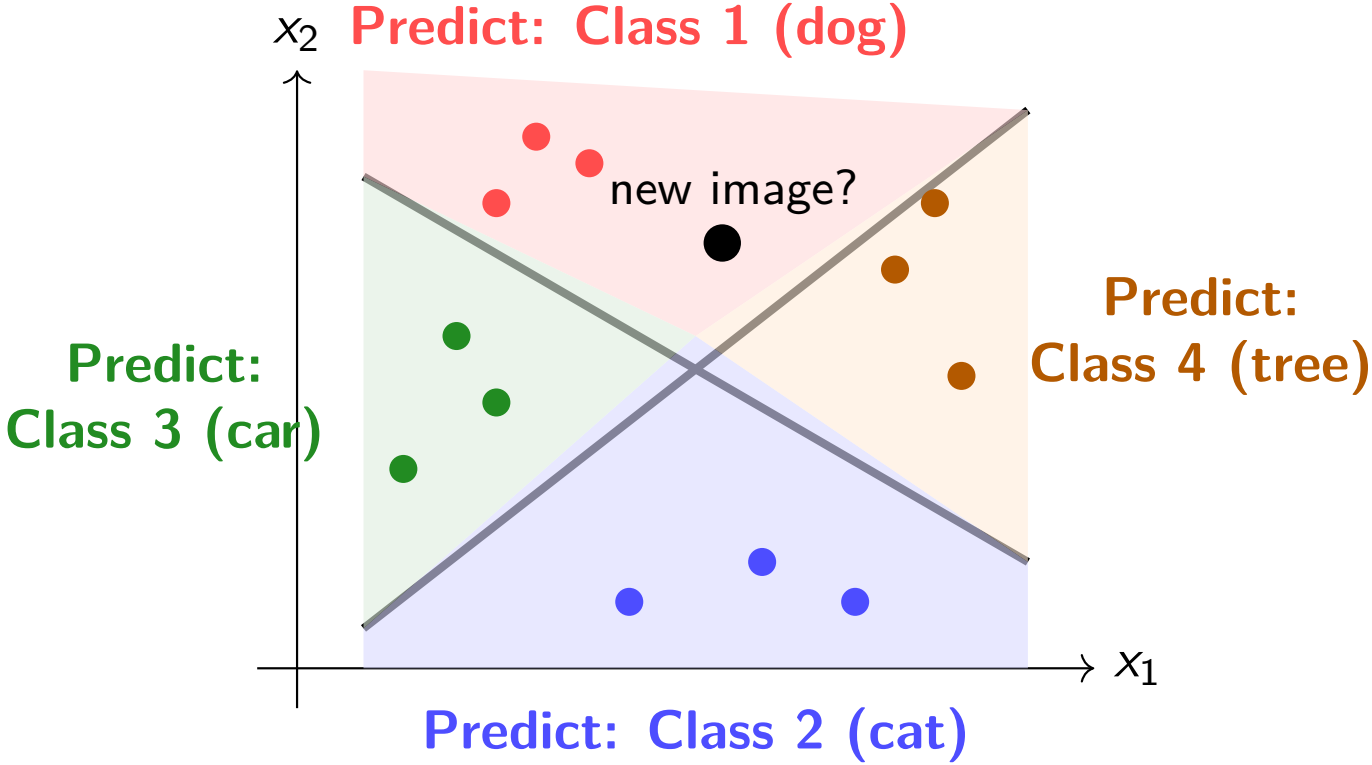
3. **KL divergence view:**

$$\ell_i = D_{\mathrm{KL}}(\delta_{y_i} \| p_{\boldsymbol{\Theta}}(\cdot \mid \mathbf{x}_i)) = H(\delta_{y_i}, p_{\boldsymbol{\Theta}}(\cdot \mid \mathbf{x}_i)) - 0$$

Same equivalence as binary case, just with K classes instead of 2!

Geometric interpretation

Decision rule: Predict $\hat{y} = \arg\max_k \boldsymbol{\theta}_k^T \mathbf{x} = \arg\max_k p(y = k \mid \mathbf{x})$
Boundaries occur where two classes have equal probability.



Each decision boundary is where two classes have equal probability: $\boldsymbol{\theta}_i^T \mathbf{x} = \boldsymbol{\theta}_j^T \mathbf{x}$. This is a hyperplane! Then the region of inputs $\mathbf{x}$ that gets classified as class k is the intersection of half-spaces $\cap_{i \neq k} \{\mathbf{x} \mid (\boldsymbol{\theta}_k - \boldsymbol{\theta}_i)^T \mathbf{x} \geq 0\}$.

Example: ImageNet Classification

Training data: n images with labels $y_i \in \{1, \ldots, 1000\}$

Model: $p_{\boldsymbol{\Theta}}(y = k \mid \mathbf{x}) = \frac{e^{\boldsymbol{\theta}_k^T \mathbf{x}}}{\sum_{j=1}^{1000} e^{\boldsymbol{\theta}_j^T \mathbf{x}}}$

where $\mathbf{x}$ is a feature representation of the image (e.g., raw pixels, or representation from a neural network).

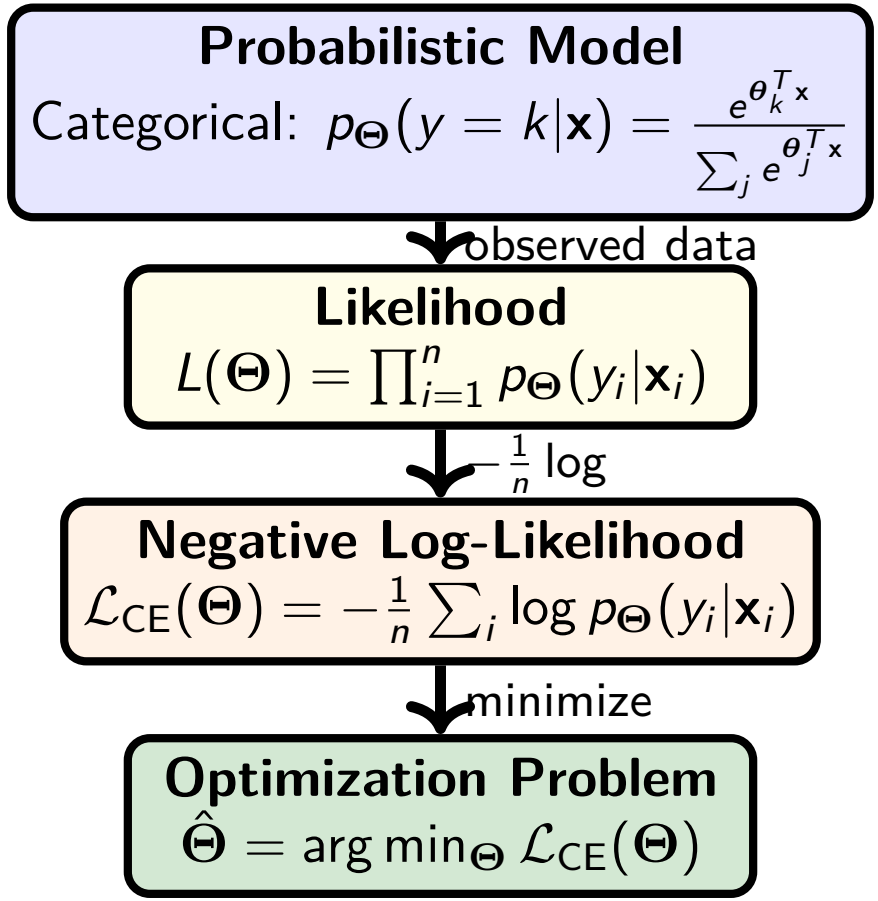
Optimization:

$$\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}} -\frac{1}{n} \sum_{i=1}^{n} \left[\boldsymbol{\theta}_{y_i}^T \mathbf{x}_i - \log \left(\sum_{k=1}^{1000} e^{\boldsymbol{\theta}_k^T \mathbf{x}_i} \right) \right]$$

Prediction for new image $\mathbf{x}_{\text{new}}$:

1. Compute probabilities: $\mathbf{p}_{\text{new}} = \text{softmax}(\hat{\boldsymbol{\Theta}} \mathbf{x}_{\text{new}})$
2. Predict class: $\hat{y} = \arg\max_{k \in \{1,\ldots,1000\}} p_k$
3. Report confidence: Can show top-5 classes with highest probabilities

Summary: Multi-Class Classification Framework



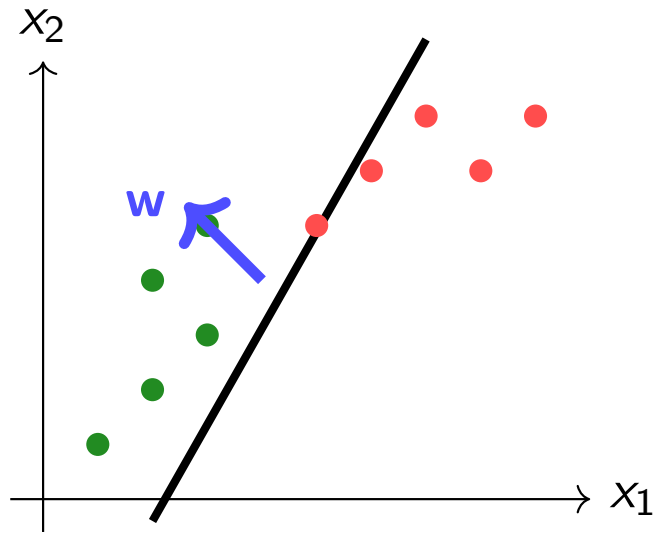
This is exactly the same recipe as binary classification!

Just replace: Bernoulli $\rightarrow$ Categorical, sigmoid $\rightarrow$ softmax, 2 classes $\rightarrow$ K classes

Why Regularization is Necessary

Problem for both binary and multi-class:

If training data is linearly separable, the loss can be minimized by making weights arbitrarily large: $\|\mathbf{w}\| \to \infty$ or $\|\mathbf{\Theta}\|_F \to \infty$



Perfect separation exists

As $\|\mathbf{w}\| \to \infty$:

$\sigma(\mathbf{w}^T\mathbf{x}) \to 0$ or 1

Consequence:

- Model becomes overconfident (probabilities $\to$ 0 or 1)
- Poor generalization to new data
- Numerical instability

Regularization for Binary and Multi-Class

Solution: Add ℓ_2 regularization to prevent unbounded weights

Binary classification:

$$\boxed{\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left[\mathcal{L}_{\text{BCE}}(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \right]}$$

where $\|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} w_j^2$

Multi-class classification:

$$\boxed{\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}} \left[\mathcal{L}_{\text{CE}}(\boldsymbol{\Theta}) + \frac{\lambda}{2} \|\boldsymbol{\Theta}\|_F^2 \right]}$$

where $\|\boldsymbol{\Theta}\|_F^2 = \sum_{k=1}^{K} \sum_{j=1}^{d} \theta_{kj}^2$ is the Frobenius norm (sum of squared entries)

Effect:

- Prevents arbitrarily large weights
- Encourages simpler, smoother decision boundaries
- Improves generalization (less overfitting)
- Same motivation as soft-margin SVM from Lecture 2!

Regularization: Probabilistic Interpretation

Recall from Lecture 3: Regularization = MAP estimation with prior

Binary case:

- Prior: $\mathbf{w} \sim \mathcal{N}(0, \frac{1}{\lambda} I)$ (Gaussian prior on weights)
- MAP estimate: $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} [p(\mathbf{w}) \prod_i p(y_i | \mathbf{x}_i, \mathbf{w})]$
- Taking $-\log$: $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} [\mathcal{L}_{\text{BCE}}(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2]$

Multi-class case:

- Prior: Each $\boldsymbol{\theta}_k \sim \mathcal{N}(0, \frac{1}{\lambda} I)$ independently
- MAP estimate gives: $\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}} [\mathcal{L}_{\text{CE}}(\boldsymbol{\Theta}) + \frac{\lambda}{2} \|\boldsymbol{\Theta}\|_F^2]$

Hyperparameter λ:

- λ large: Strong regularization, simpler model, may underfit
- λ small: Weak regularization, complex model, may overfit
- Choose via cross-validation (next lecture!)

Comparison: Binary vs Multi-Class

	Binary	**Multi-Class**
Label space	$y \in \{0, 1\}$	$y \in \{1, \ldots, K\}$
Distribution	Bernoulli	Categorical
Transformation	Sigmoid ($\sigma(z)$)	Softmax (softmax($\mathbf{z}$))
Parameters	$\mathbf{w} \in \mathbb{R}^d$	$\boldsymbol{\Theta} \in \mathbb{R}^{K \times d}$
Loss	BCE	Categorical CE
Regularization	$\frac{\lambda}{2}\|\mathbf{w}\|_2^2$	$\frac{\lambda}{2}\|\boldsymbol{\Theta}\|_F^2$
Names	Logistic regression	Softmax regression

Both share:

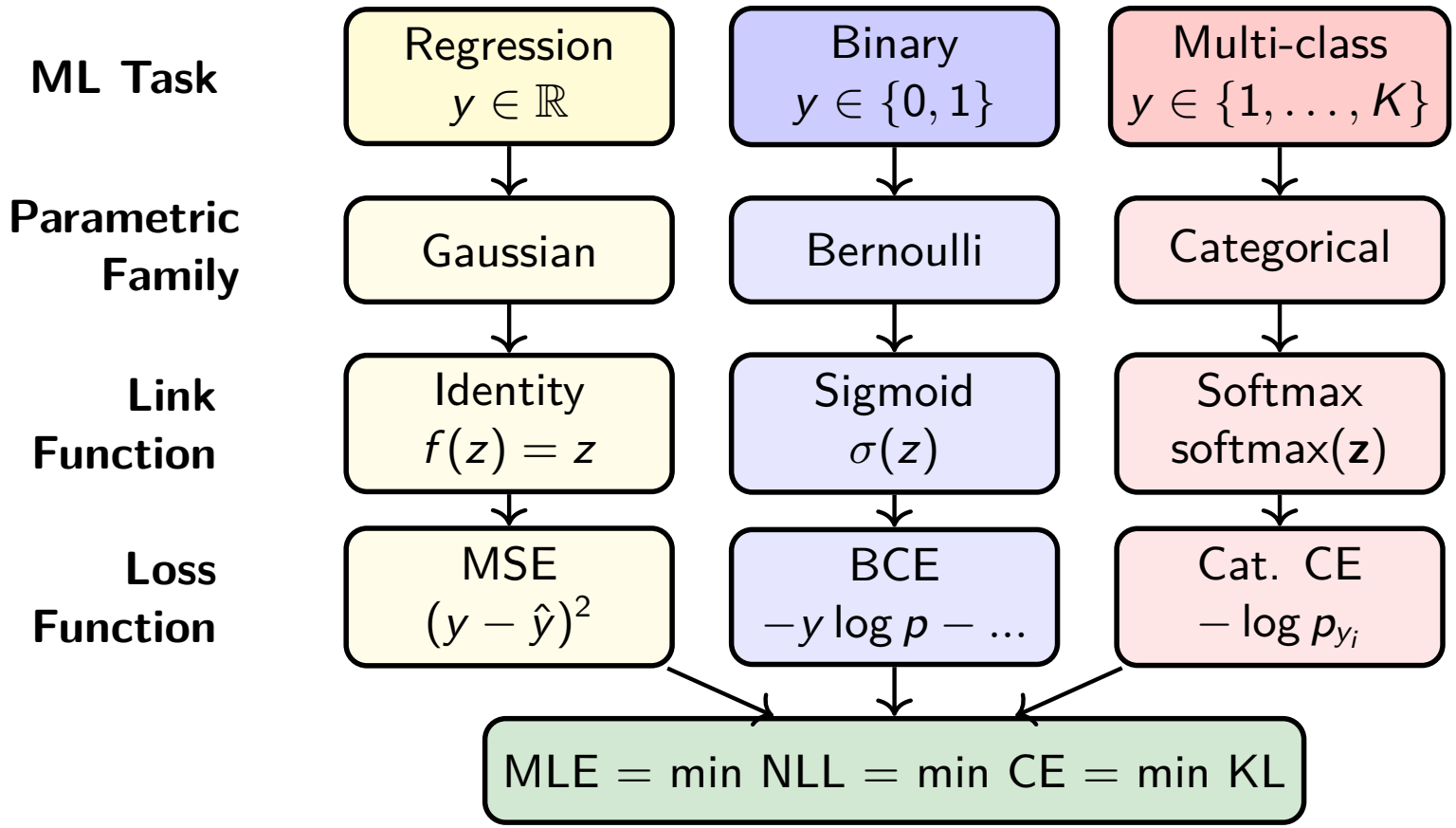
- Derived from maximum likelihood (MLE)
- Equivalent to minimizing cross-entropy / KL divergence
- Need regularization for linearly separable data
- Produce calibrated probability estimates

Recap

1. **Probabilistic modeling for classification:**
   - Binary: Bernoulli model with sigmoid transformation
   - Multi-class: Categorical model with softmax transformation
   - Both derived from maximum likelihood principle
2. **Information theory provides the unifying framework:**
   - Entropy $H(p)$ measures uncertainty in distribution
   - Cross-entropy $H(p, q)$ measures cost of using wrong code
   - KL divergence $D_{\mathrm{KL}}(p\|q)$ measures distributional dissimilarity
3. **Three equivalent perspectives on classification loss:**
   - Optimization: minimize negative log-likelihood
   - Statistics: maximum likelihood estimation
   - Information theory: minimize KL divergence / cross-entropy
4. **Regularization is necessary:**
   - Prevents unbounded weights for separable data
   - Probabilistic interpretation: MAP with Gaussian prior
   - Same form for both binary and multi-class

Linear Models

So far, we've consider multiple linear models, where $\mathbf{z} = \mathbf{w}^T\mathbf{x} + b$



All paths lead to the same principle: Maximum Likelihood Estimation (or MAP)!

What We Haven't Covered (Yet)

Important topics for future lectures:

1. **How to actually solve these optimization problems**
   - Computing gradients of BCE and categorical CE
   - Gradient descent and stochastic gradient descent
   - Convergence analysis and step size selection
2. **Model evaluation and selection**
   - Train/validation/test split methodology
   - Choosing regularization parameter λ
   - Evaluation metrics: accuracy, precision, recall, F1, AUC
3. **Nonlinear models and other tasks**
   - Replacing raw features $\mathbf{x}$ with features $\mathbf{x}_{\text{rep}}$ obtained from neural networks
   - Handling other tasks like prediction of sequences

Questions?

Appendix: Logistic Loss (Alternative Binary Formulation)

Alternative parametrization for binary: Use $y \in \{-1, +1\}$ instead of $\{0, 1\}$

Claim: For $y \in \{-1, +1\}$:

$$p_{\mathbf{w}}(y \mid \mathbf{x}) = \sigma(y \cdot \mathbf{w}^T \mathbf{x})$$

Proof:

- If $y = +1$: $p_{\mathbf{w}}(y = +1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$
- If $y = -1$: $p_{\mathbf{w}}(y = -1 \mid \mathbf{x}) = 1 - \sigma(\mathbf{w}^T \mathbf{x}) = \sigma(-\mathbf{w}^T \mathbf{x})$

This gives the logistic loss:

$$\ell_i = -\log \sigma(y_i \cdot \mathbf{w}^T \mathbf{x}_i) = \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i})$$

Logistic loss for dataset:

$$\boxed{\mathcal{L}_{\text{logistic}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i})}$$

Equivalent to BCE, just different label encoding!

Appendix: BCE vs Logistic Loss Comparison

	BCE Loss	**Logistic Loss**
Label encoding	$y \in \{0, 1\}$	$y \in \{-1, +1\}$
Formula	$-y \log p - (1 - y) \log(1 - p)$	$\log(1 + e^{-y\mathbf{w}^T\mathbf{x}})$
Connection to CE	Explicit	Implicit
Extends to multi-class	Naturally via softmax	Less natural
Use when	Probabilistic interpretation matters	Simpler formula preferred
Common in	Deep learning libraries	Optimization literature

Why BCE is often preferred:

- Explicit connection to cross-entropy and information theory
- Extends naturally to multi-class (categorical cross-entropy)

Connection to SVM (Lecture 2)

SVM (Lecture 2):

- Used hinge loss: $\max(0, 1 - y(\mathbf{w}^T\mathbf{x}))$ for $y \in \{-1, +1\}$
- Focus on decision boundary and margin
- No probabilistic interpretation
- Soft-margin: add regularization $\frac{1}{2}\|\mathbf{w}\|_2^2$

Logistic Regression (Today):

- Use logistic/BCE loss: $\log(1 + e^{-y(\mathbf{w}^T\mathbf{x})})$ or $-y \log p - (1 - y) \log(1 - p)$
- Probabilistic model $\Rightarrow$ likelihood $\Rightarrow$ loss
- Outputs interpretable probabilities
- Regularization: add $\frac{\lambda}{2}\|\mathbf{w}\|_2^2$ (same form!)

Key differences:

- SVM: margin-based, geometric interpretation
- Logistic: probabilistic, information-theoretic interpretation
- Both are convex optimization problems with similar regularization
- Logistic extends naturally to multi-class via softmax

Connection to Regression (Lecture 3)

Regression (Lecture 3):

- Gaussian likelihood $\Rightarrow$ squared loss for regression
- Gaussian prior $\Rightarrow$ ridge regularization (ℓ_2)
- Maximum likelihood and MAP estimation

Classification (Today):

- Bernoulli/Categorical likelihood $\Rightarrow$ cross-entropy loss
- Gaussian prior $\Rightarrow$ ℓ_2 regularization (same!)
- Extended MLE/MAP framework to discrete outputs

Unified view: Different probabilistic models and loss functions all emerge from the same principles:

Probabilistic model + Maximum likelihood $\Rightarrow$ Loss function

Gaussian prior + MAP $\Rightarrow$ Regularization