Perceptron and Logistic Regression

Jindřich Libovický (reusing materials by Milan Straka)

October 13, 2025

Charles University
ÚFAL

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Today's Lecture Objectives

After this lecture you should be able to

- Think about binary classification using **geometric intuition** and use the **perceptron algorithm**.
- Define the **main concepts of information theory** (entropy, cross-entropy, KL-divergence) and prove their properties.
- Derive training objectives using the **maximum likelihood principle**.
- Implement and use **logistic regression** for binary classification with SGD.

Perceptron

Binary Classification

Binary classification is a classification in two classes.

The simplest way to evaluate classification is **accuracy**, which is the ratio of input examples that were classified correctly – i.e., where the predicted class and the target class match.

To extend linear regression to binary classification, we might seek a **threshold** and then classify an input as negative/positive depending on whether $y(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w} + b$ is smaller/larger than a given threshold.

Zero value is usually used as the threshold, both because of symmetry and also because the **bias** parameter acts as a trainable threshold anyway.

The set of points with prediction 0 is called a **decision boundary**.

Geometric Intuition

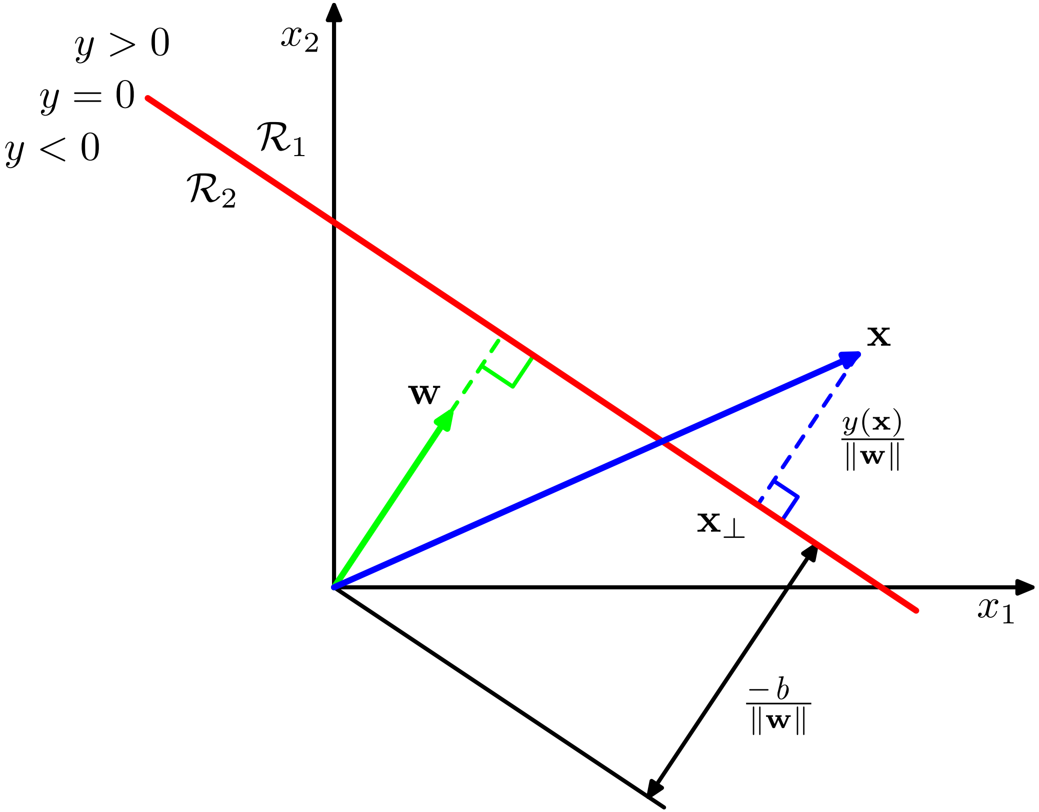


Figure 4.1 of Pattern Recognition and Machine Learning.

Perceptron

The perceptron algorithm is probably the oldest one for training weights of a binary classification. Assuming the target value $t \in \{-1, +1\}$, the goal is to find weights $\boldsymbol{w}$ such that for all train data,

$$\operatorname{sign}(y(\boldsymbol{x}_i; \boldsymbol{w})) = \operatorname{sign}(\boldsymbol{x}_i^T \boldsymbol{w}) = t_i,$$

or equivalently,

$$t_i y(\boldsymbol{x}_i; \boldsymbol{w}) = t_i \boldsymbol{x}_i^T \boldsymbol{w} > 0.$$

Note that a set is called **linearly separable**, if there exists a weight vector $\boldsymbol{w}$ such that the above equation holds.

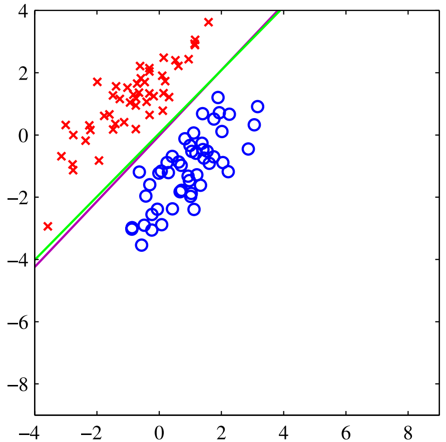


Figure 4.4 of Pattern Recognition and Machine Learning.

Perceptron

The perceptron algorithm was invented by Rosenblatt in 1958.

Input: Linearly separable dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{-1, +1\}^N$).
Output: Weights $\boldsymbol{w} \in \mathbb{R}^D$ such that $t_i \boldsymbol{x}_i^T \boldsymbol{w} > 0$ for all i.

- $\boldsymbol{w} \leftarrow \mathbf{0}$
- until all examples are classified correctly, process example i:
  - $y \leftarrow \boldsymbol{x}_i^T \boldsymbol{w}$
  - if $t_i y \leq 0$ (incorrectly classified example):
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + t_i \boldsymbol{x}_i$

We will prove that the algorithm always arrives at some correct set of weights $\boldsymbol{w}$ if the training set is linearly separable.

Proof of Perceptron Convergence

Let $\boldsymbol{w}_*$ be some weights correctly classifying (separating) the training data, and let $\boldsymbol{w}_k$ be the weights after k nontrivial updates of the perceptron algorithm, with $\boldsymbol{w}_0$ being 0.

We will prove that the angle α between $\boldsymbol{w}_*$ and $\boldsymbol{w}_k$ decreases at each step. Note that

$$\cos(\alpha) = \frac{\boldsymbol{w}_*^T \boldsymbol{w}_k}{\|\boldsymbol{w}_*\| \cdot \|\boldsymbol{w}_k\|}.$$

Proof of Perceptron Convergence

Assume that the maximum norm of any training example $\|\boldsymbol{x}\|$ is bounded by R, and that γ is the minimum margin of $\boldsymbol{w}_*$, so for each training example $(\boldsymbol{x}, t)$, $t\boldsymbol{x}^T\boldsymbol{w}_* \geq \gamma$.

First consider the dot product of $\boldsymbol{w}_*$ and $\boldsymbol{w}_k$:

$$\boldsymbol{w}_*^T\boldsymbol{w}_k = \boldsymbol{w}_*^T(\boldsymbol{w}_{k-1} + t_k\boldsymbol{x}_k) \geq \boldsymbol{w}_*^T\boldsymbol{w}_{k-1} + \gamma.$$

By iteratively applying this equation, we get

$$\boldsymbol{w}_*^T\boldsymbol{w}_k \geq k\gamma.$$

Now consider the length of $\boldsymbol{w}_k$:

$$\|\boldsymbol{w}_k\|^2 = \|\boldsymbol{w}_{k-1} + t_k\boldsymbol{x}_k\|^2 = \|\boldsymbol{w}_{k-1}\|^2 + 2t_k\boldsymbol{x}_k^T\boldsymbol{w}_{k-1} + \|\boldsymbol{x}_k\|^2.$$

Because $\boldsymbol{x}_k$ was misclassified, we know that $t_k\boldsymbol{x}_k^T\boldsymbol{w}_{k-1} \leq 0$, so $\|\boldsymbol{w}_k\|^2 \leq \|\boldsymbol{w}_{k-1}\|^2 + R^2$. When applied iteratively, we get $\|\boldsymbol{w}_k\|^2 \leq k \cdot R^2$.

Proof of Perceptron Convergence

Putting everything together, we get

$$\cos(\alpha) = \frac{\boldsymbol{w}_*^T \boldsymbol{w}_k}{\|\boldsymbol{w}_*\| \cdot \|\boldsymbol{w}_k\|} \geq \frac{k\gamma}{\sqrt{kR^2}\|\boldsymbol{w}_*\|}.$$

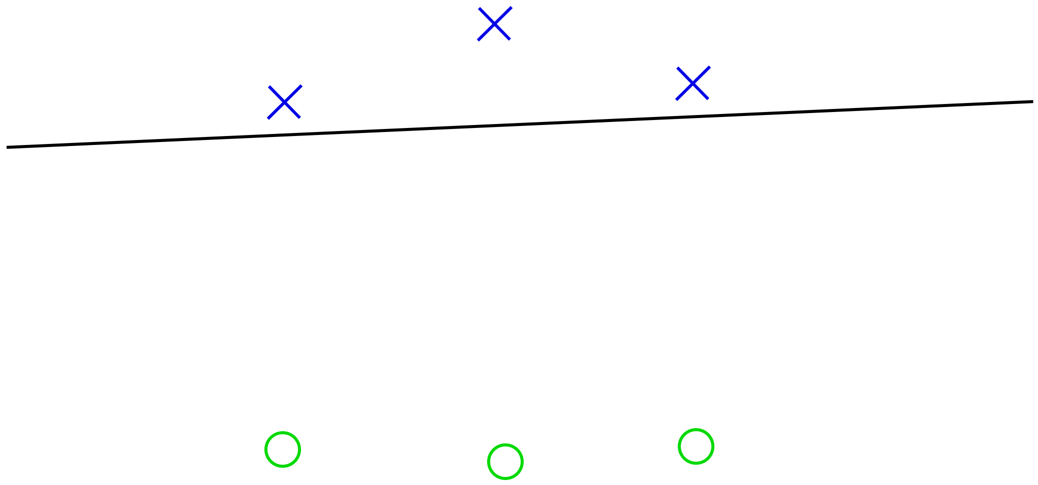
Therefore, the $\cos(\alpha)$ increases during every update. Because the value of $\cos(\alpha)$ is at most one, we can compute the upper bound on the number of steps when the algorithm converges as

$$1 \geq \frac{\sqrt{k}\gamma}{\sqrt{R^2}\|\boldsymbol{w}_*\|} \text{ or } k \leq \frac{R^2\|\boldsymbol{w}_*\|^2}{\gamma^2}.$$

Perceptron Issues

Perceptron has several drawbacks:

- If the input set is not linearly separable, the algorithm never finishes.
- The algorithm performs only prediction, it is not able to return the probabilities of predictions.
- Most importantly, Perceptron algorithm finds *some* solution, not necessarily a good one, because once it finds some, it cannot perform any more updates.

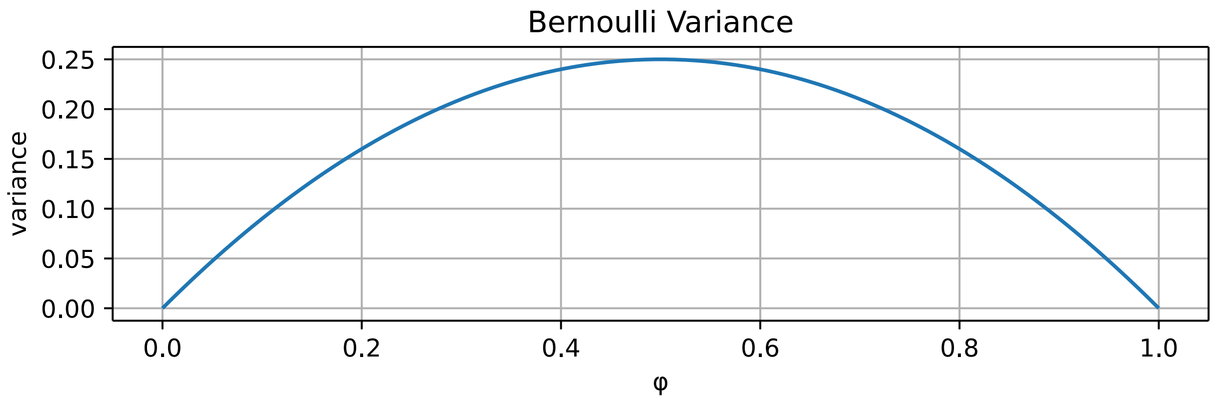


Basics of Probability

Bernoulli Distribution

The Bernoulli distribution is a distribution over a binary random variable. It has a single parameter $\varphi \in [0, 1]$, which specifies the probability that the random variable is equal to 1.

$$P(x) = \varphi^x (1 - \varphi)^{1-x}$$
$$\mathbb{E}[x] = \varphi$$
$$\mathrm{Var}(x) = \varphi(1 - \varphi)$$



Categorical Distribution

Extension of the Bernoulli distribution to random variables taking one of K different discrete outcomes. It is parametrized by $\boldsymbol{p} \in [0,1]^K$ such that $\sum_{i=0}^{K-1} p_i = 1$.

We represent outcomes as vectors $\in \{0,1\}^K$ in **one-hot encoding**. Therefore, an outcome $x \in \{0, 1, \ldots, K-1\}$ is represented as a vector

$$\mathbf{1}_x \stackrel{\text{def}}{=} \big([i = x]\big)_{i=0}^{K-1} = \big(\underbrace{0, \ldots, 0}_{x}, 1, \underbrace{0, \ldots, 0}_{K-x-1}\big).$$

The outcome probability, mean, and variance are very similar to the Bernoulli distribution.

$$P(\boldsymbol{x}) = \prod\nolimits_{i=0}^{K-1} p_i^{x_i}$$

$$\mathbb{E}[x_i] = p_i$$

$$\mathrm{Var}(x_i) = p_i(1 - p_i)$$

Information Theory

Self Information

Amount of **surprise** when a random variable is sampled.

- Should be zero for events with probability 1.
- Less likely events are more surprising.
- Independent events should have **additive** information.

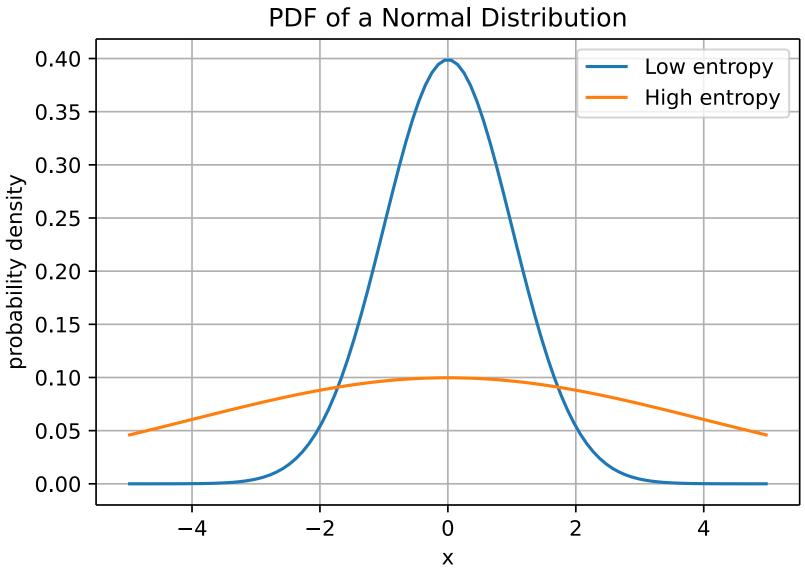
$$I(x) \stackrel{\text{def}}{=} -\log P(x) = \log \frac{1}{P(x)}$$

Entropy

Amount of **surprise** in the whole distribution.

$$H(P) \stackrel{\text{def}}{=} \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)]$$

- for discrete P: $H(P) = -\sum_x P(x) \log P(x)$
- for continuous P: $H(P) = -\int P(x) \log P(x)\,\mathrm{d}x$



Because $\lim_{x\to 0} x \log x = 0$, for $P(x) = 0$ we consider $P(x) \log P(x)$ to be zero.

Note that in the continuous case, the continuous entropy (also called *differential entropy*) has slightly different semantics, for example, it can be negative.

For binary logarithms, the entropy is measured in **bits**.
However, from now on, all logarithms are *natural logarithms* with base *e* (and then the entropy is measured in units called **nats**).

Cross-Entropy

$$H(P, Q) \stackrel{\text{def}}{=} -\mathbb{E}_{\mathrm{x} \sim P}[\log Q(x)]$$

Gibbs Inequality

- $H(P, Q) \geq H(P)$
- $H(P) = H(P, Q) \Leftrightarrow P = Q$

Proof: Consider $H(P) - H(P, Q) = \sum_x P(x) \log \frac{Q(x)}{P(x)}$.

Using the fact that $\log x \leq (x - 1)$ with equality only for $x = 1$, we get

$$\sum_x P(x) \log \frac{Q(x)}{P(x)} \leq \sum_x P(x) \left(\frac{Q(x)}{P(x)} - 1\right) = \sum_x Q(x) - \sum_x P(x) = 0.$$

For the equality to hold, $\frac{Q(x)}{P(x)}$ must be 1 for all x, i.e., $P = Q$.

Kullback-Leibler Divergence (KL Divergence)

Sometimes also called **relative entropy**.

$$D_{\mathrm{KL}}(P\|Q) \stackrel{\text{def}}{=} H(P,Q) - H(P) = \mathbb{E}_{\mathrm{x}\sim P}[\log P(x) - \log Q(x)]$$

- consequence of Gibbs inequality: $D_{\mathrm{KL}}(P\|Q) \geq 0$, $D_{\mathrm{KL}}(P\|Q) = 0$ iff $P = Q$
- generally $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$

Normal (or Gaussian) Distribution

A distribution over real numbers, parametrized by mean μ and variance σ^2:

$$\mathcal{N}(x;\mu,\sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

For standard values $\mu = 0$ and $\sigma^2 = 1$ we get $\mathcal{N}(x;0,1) = \sqrt{\frac{1}{2\pi}} e^{-\frac{x^2}{2}}$.

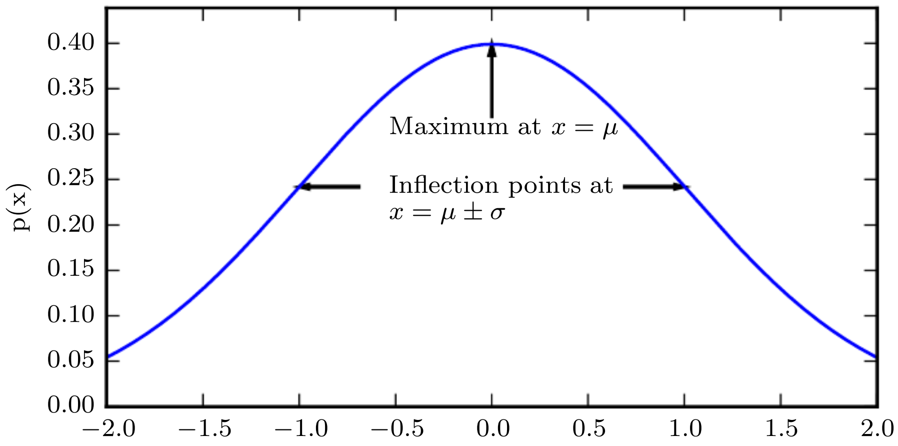


Figure 3.1 of "Deep Learning" book, https://www.deeplearningbook.org.

Central Limit Theorem

The sum of independent identically distributed random variables with finite non-zero variance converges to normal distribution.

Principle of Maximum Entropy

Given a set of constraints, a distribution with maximal entropy fulfilling the constraints can be considered the most general one, containing as little additional assumptions as possible.

Considering distributions with a **given mean and variance**, it can be proven (using variational inference) that such a distribution with **maximum entropy** is exactly the normal distribution.

Maximum Likelihood Estimation

Maximum Likelihood Estimation

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be training data drawn independently from the data-generating distribution p_{data}.

We denote the **empirical data distribution** as $\hat{p}_{\text{data}}$, where

$$\hat{p}_{\text{data}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \frac{\big|\{i : \boldsymbol{x}_i = \boldsymbol{x}\}\big|}{N}.$$

Let $p_{\text{model}}(\mathbf{x}; \boldsymbol{w})$ be a family of distributions.

- If the weights are fixed, $p_{\text{model}}(\mathbf{x}; \boldsymbol{w})$ is a probability distribution.
- If we instead consider the fixed training data $\boldsymbol{X}$, then

$$L(\boldsymbol{w}) = p_{\text{model}}(\boldsymbol{X}; \boldsymbol{w}) = \prod\nolimits_{i=1}^{N} p_{\text{model}}(\boldsymbol{x}_i; \boldsymbol{w})$$

is called the **likelihood**. Note that even if the value of the likelihood is in range $[0, 1]$, it is not a probability, because the likelihood is not a probability distribution.

Maximum Likelihood Estimation

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be training data drawn independently from the data-generating distribution $p_\textrm{data}$. We denote the empirical data distribution as $\hat p_\textrm{data}$ and let $p_\textrm{model}(\mathbf{x}; \boldsymbol{w})$ be a family of distributions.

The **maximum likelihood estimation** of $\boldsymbol{w}$ is:

$$\begin{aligned}
\boldsymbol{w}_\textrm{MLE} &= \arg\max_{\boldsymbol{w}} p_\textrm{model}(\boldsymbol{X}; \boldsymbol{w}) = \arg\max_{\boldsymbol{w}} \prod\nolimits_{i=1}^N p_\textrm{model}(\boldsymbol{x}_i; \boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum\nolimits_{i=1}^N -\log p_\textrm{model}(\boldsymbol{x}_i; \boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \mathbb{E}_{\mathbf{x} \sim \hat p_\textrm{data}} [-\log p_\textrm{model}(\boldsymbol{x}; \boldsymbol{w})] \\
&= \arg\min_{\boldsymbol{w}} H(\hat p_\textrm{data}(\mathbf{x}), p_\textrm{model}(\mathbf{x}; \boldsymbol{w})) \\
&= \arg\min_{\boldsymbol{w}} D_\textrm{KL}(\hat p_\textrm{data}(\mathbf{x}) \| p_\textrm{model}(\mathbf{x}; \boldsymbol{w})) + H(\hat p_\textrm{data}(\mathbf{x}))
\end{aligned}$$

MLE can be easily generalized to the conditional case, where our goal is to predict t given $\boldsymbol{x}$:

$$\begin{aligned}
\boldsymbol{w}_{\text{MLE}} &= \arg\max_{\boldsymbol{w}} p_{\text{model}}(\boldsymbol{t}|\boldsymbol{X};\boldsymbol{w}) = \arg\max_{\boldsymbol{w}} \prod\nolimits_{i=1}^{N} p_{\text{model}}(t_i|\boldsymbol{x}_i;\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum\nolimits_{i=1}^{N} -\log p_{\text{model}}(t_i|\boldsymbol{x}_i;\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(\mathbf{x},\mathrm{t})\sim\hat{p}_{\text{data}}}[-\log p_{\text{model}}(t|\boldsymbol{x};\boldsymbol{w})] \\
&= \arg\min_{\boldsymbol{w}} H(\hat{p}_{\text{data}}(\mathrm{t}|\mathbf{x}), p_{\text{model}}(\mathrm{t}|\mathbf{x};\boldsymbol{w})) \\
&= \arg\min_{\boldsymbol{w}} D_{\text{KL}}(\hat{p}_{\text{data}}(\mathrm{t}|\mathbf{x})\|p_{\text{model}}(\mathrm{t}|\mathbf{x};\boldsymbol{w})) + H(\hat{p}_{\text{data}}(\mathrm{t}|\mathbf{x}))
\end{aligned}$$

where the conditional entropy is defined as $H(\hat{p}_{\text{data}}) = \mathbb{E}_{(\mathbf{x},\mathrm{t})\sim\hat{p}_{\text{data}}}[-\log(\hat{p}_{\text{data}}(t|\boldsymbol{x};\boldsymbol{w}))]$ and the conditional cross-entropy as $H(\hat{p}_{\text{data}}, p_{\text{model}}) = \mathbb{E}_{(\mathbf{x},\mathrm{t})\sim\hat{p}_{\text{data}}}[-\log(p_{\text{model}}(t|\boldsymbol{x};\boldsymbol{w}))]$.

The resulting *loss function* is called **negative log-likelihood** (**NLL**), or **cross-entropy**, or **Kullback-Leibler divergence**.

Logistic Regression

Logistic Regression

An extension of perceptron, which models the conditional probabilities of $p(C_0|\boldsymbol{x})$ and of $p(C_1|\boldsymbol{x})$. Logistic regression can in fact handle also more than two classes, which we will see in the next lecture.

Logistic regression employs the following parametrization of the conditional class probabilities:

$$p(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{x}^T\boldsymbol{w} + b)$$
$$p(C_0|\boldsymbol{x}) = 1 - p(C_1|\boldsymbol{x}),$$

where σ is a **sigmoid function**

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

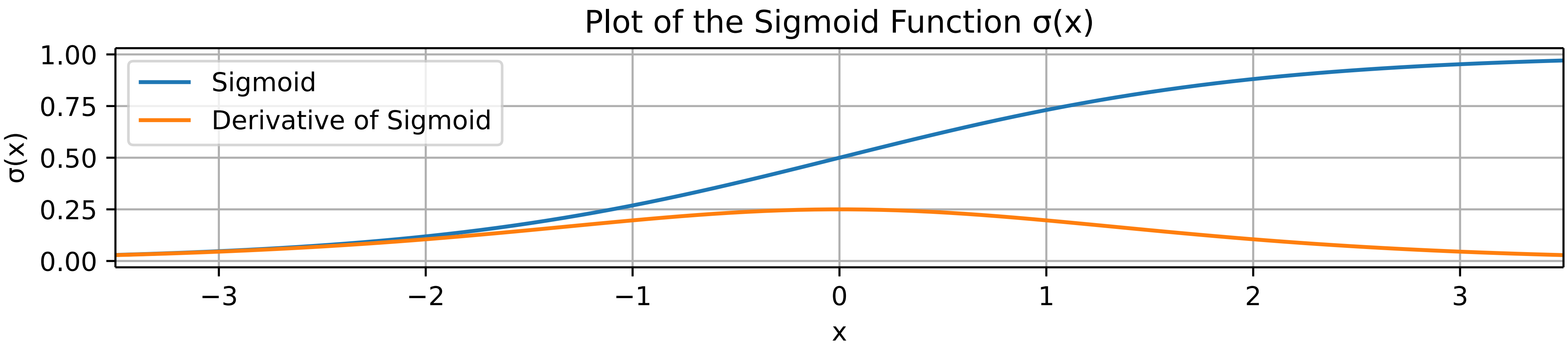
It can be trained using the SGD algorithm.

Sigmoid Function

The sigmoid function has values in range $(0, 1)$, is monotonically increasing and it has a derivative of $\frac{1}{4}$ at $x = 0$.

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

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$



We denote the output of the "linear part" of logistic regression as

$$\bar{y}(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w},$$

and the overall prediction as

$$y(\boldsymbol{x}; \boldsymbol{w}) = \sigma(\bar{y}(\boldsymbol{x}; \boldsymbol{w})) = \sigma(\boldsymbol{x}^T \boldsymbol{w}).$$

To train logistic regression, we use MLE (the maximum likelihood estimation). Its application is straightforward, given that $p(C_1|\boldsymbol{x}; \boldsymbol{w})$ is directly the model output $y(\boldsymbol{x}; \boldsymbol{w})$.

Therefore, the loss for a minibatch $\mathbb{X} = \{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_N, t_N)\}$ is

$$E(\boldsymbol{w}) = \frac{1}{N} \sum_i -\log(p(C_{t_i}|\boldsymbol{x}_i; \boldsymbol{w})).$$

Input: Input dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{0, +1\}^N$), learning rate $\alpha \in \mathbb{R}^+$.

- $\boldsymbol{w} \leftarrow \boldsymbol{0}$ or we initialize $\boldsymbol{w}$ randomly
- until convergence (or patience runs out), process a minibatch of examples $\mathbb{B}$:
  - $\boldsymbol{g} \leftarrow \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \nabla_{\boldsymbol{w}} \Big(-\log \big(p(C_{t_i}|\boldsymbol{x}_i; \boldsymbol{w})\big)\Big)$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{g}$

Practical note

Everything we learned about **features** and L^2 **regularization** holds for logistic regression too.

😮

Today's Lecture Objectives

After this lecture you should be able to

- Think about binary classification using **geometric intuition** and use the **perceptron algorithm**.
- Define the **main concepts of information theory** (entropy, cross-entropy, KL-divergence) and prove their properties.
- Derive training objectives using the **maximum likelihood principle**.
- Implement and use **logistic regression** for binary classification with SGD.