

# CS559 Machine Learning

## Linear Discriminant Function

Tian Han

Department of Computer Science  
Stevens Institute of Technology

Week 5

## Outline

- Generative and discriminative approach
- Linear Discriminant Function
- The Perceptron Algorithm
- Gradient Descent and its Variants.

# Generative and Discriminative Approach

# Decision Theory for Classification

Recall the decision theory allows us to make optimal decisions in scenarios involving uncertainty. In practice:

## Decision Theory for Classification

Recall the decision theory allows us to make optimal decisions in scenarios involving uncertainty. In practice:

- Given **training data**:  $\{(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_N, t_N)\}$ .

## Decision Theory for Classification

Recall the decision theory allows us to make optimal decisions in scenarios involving uncertainty. In practice:

- Given **training data**:  $\{(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_N, t_N)\}$ .
- **Learning stage**: build model  $p(C_k|\mathbf{x})$ ,  $C_k$  is the  $k$ -th class.

## Decision Theory for Classification

Recall the decision theory allows us to make optimal decisions in scenarios involving uncertainty. In practice:

- Given **training data**:  $\{(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_N, t_N)\}$ .
- **Learning stage**: build model  $p(C_k|\mathbf{x})$ ,  $C_k$  is the  $k$ -th class.
- **Decision stage**: use the obtained posterior  $p(C_k|\mathbf{x})$  (together with the possible risk function) to make optimal decision/classification.

## Discriminative Approach

- Directly learn the **class posterior probability**  $p(C_k|\mathbf{x})$ .
- Logistic regression in last lecture.



# Generative Approach

- Learn the class condition density  $p(\mathbf{x}|C_k)$

# Generative Approach

- Learn the class condition density  $p(\mathbf{x}|C_k)$
- Estimate the class prior  $P(C_k)$

## Generative Approach

- Learn the class condition density  $p(\mathbf{x}|C_k)$
- Estimate the class prior  $P(C_k)$
- Use Bayes formula to get **class posterior**:

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(x)}$$

where

$$p(x) = \sum_{k=1}^K p(\mathbf{x}|C_k)P(C_k)$$

# Linear Discriminant Function

## Two-class Discriminant Function

Recall we use *probabilistic discriminative* approach for classification by directly model the posterior  $p(C_1|\mathbf{x})$  using some parametric form. (e.g., logistic regression,  $p(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$ )

## Two-class Discriminant Function

Recall we use *probabilistic discriminative* approach for classification by directly model the posterior  $p(C_1|\mathbf{x})$  using some parametric form. (e.g., logistic regression,  $p(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$ )

In this lecture, we study *discriminant function* that directly assigns each input vector  $x$  to a specific class.

## Two-class Discriminant Function

Recall we use *probabilistic discriminative* approach for classification by directly model the posterior  $p(C_1|\mathbf{x})$  using some parametric form. (e.g., logistic regression,  $p(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$ )

In this lecture, we study *discriminant function* that directly assigns each input vector  $x$  to a specific class.

The simplest one would be *linear* discriminant function:

$$\begin{aligned} y(\mathbf{x}) &= w_1 x_1 + w_2 x_2 + \cdots + w_D x_D + w_0 \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

where  $\mathbf{w}$  is the weight vector,  $w_0$  is the bias.

## Two-class Discriminant Function

Recall we use *probabilistic discriminative* approach for classification by directly model the posterior  $p(C_1|\mathbf{x})$  using some parametric form. (e.g., logistic regression,  $p(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$ )

In this lecture, we study *discriminant function* that directly assigns each input vector  $x$  to a specific class.

The simplest one would be *linear* discriminant function:

$$\begin{aligned} y(\mathbf{x}) &= w_1 x_1 + w_2 x_2 + \cdots + w_D x_D + w_0 \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

where  $\mathbf{w}$  is the weight vector,  $w_0$  is the bias.

Goal: classify  $\mathbf{x}$ :

$$\mathbf{x} \rightarrow C_1, \text{ if } y(\mathbf{x}) \geq 0$$

$$\mathbf{x} \rightarrow C_2, \text{ if } y(\mathbf{x}) < 0$$



## Geometric Interpretation: dot product

Given two vectors:  $\mathbf{u} = (u_1, \dots, u_D)$  and  $\mathbf{v} = (v_1, \dots, v_D)$ , their dot product:

$$\mathbf{u} \cdot \mathbf{v} = \langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^D u_i v_i$$

- $\mathbf{u}, \mathbf{v}$  in the *same direction*, the dot product grows large and *positive*.
- $\mathbf{u}, \mathbf{v}$  in the *opposite direction*, the dot product grows large and *negative*.
- $\mathbf{u}, \mathbf{v}$  perpendicular, dot product is *zero*.

## Dot Product as Projection

Suppose  $\mathbf{u}$  is **unit vector**, i.e.,  $\|\mathbf{u}\| = 1$ :

Any vector  $\mathbf{v}$  consists of two components:

## Dot Product as Projection

Suppose  $\mathbf{u}$  is **unit vector**, i.e.,  $\|\mathbf{u}\| = 1$ :

Any vector  $\mathbf{v}$  consists of two components:

- A component that in the direction of  $\mathbf{u}$ .
- A component that that's perpendicular to  $\mathbf{u}$ .

## Dot Product as Projection

Suppose  $\mathbf{u}$  is **unit vector**, i.e.,  $\|\mathbf{u}\| = 1$ :

Any vector  $\mathbf{v}$  consists of two components:

- A component that in the direction of  $\mathbf{u}$ .
- A component that that's perpendicular to  $\mathbf{u}$ .

For example:  $\mathbf{u} = (0.8, 0.6)$ ,  $\mathbf{v} = (0.37, 0.73)$ .

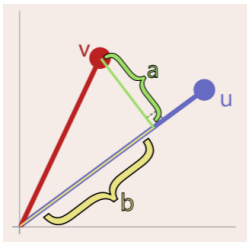


Figure: [from CIML]

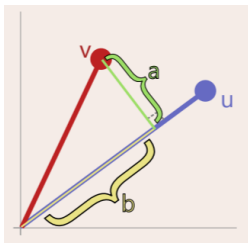
## Dot Product as Projection

Suppose  $\mathbf{u}$  is **unit vector**, i.e.,  $\|\mathbf{u}\| = 1$ :

Any vector  $\mathbf{v}$  consists of two components:

- A component that in the direction of  $\mathbf{u}$ .
- A component that that's perpendicular to  $\mathbf{u}$ .

For example:  $\mathbf{u} = (0.8, 0.6)$ ,  $\mathbf{v} = (0.37, 0.73)$ .



The length of  $b$  is exactly  
 $\mathbf{u} \cdot \mathbf{v} = 0.734$

**Dot product as projections:**

dot product between  $\mathbf{u}$  and  $\mathbf{v}$  is  
the "projection of  $\mathbf{v}$  onto  $\mathbf{u}$ ".

Figure: [from CIML]

## Geometric Interpretation of decision boundary

First, consider linear discriminant function without bias term, that is  $y = \sum_{i=1}^D w_i x_i = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$ .

## Geometric Interpretation of decision boundary

First, consider linear discriminant function without bias term, that is  $y = \sum_{i=1}^D w_i x_i = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$ .

If  $\sum_{i=1}^D w_i x_i > 0$ , classify  $\mathbf{x}$  to class  $C_1$  (i.e., denote as positive class, +1, etc);

## Geometric Interpretation of decision boundary

First, consider linear discriminant function without bias term, that is  $y = \sum_{i=1}^D w_i x_i = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$ .

If  $\sum_{i=1}^D w_i x_i > 0$ , classify  $\mathbf{x}$  to class  $C_1$  (i.e., denote as positive class, +1, etc);

If  $\sum_{i=1}^D w_i x_i < 0$ , classify  $\mathbf{x}$  to class  $C_2$  (i.e., denote as negative class, -1, etc);



## Geometric Interpretation of decision boundary

First, consider linear discriminant function without bias term, that is  $y = \sum_{i=1}^D w_i x_i = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$ .

If  $\sum_{i=1}^D w_i x_i > 0$ , classify  $\mathbf{x}$  to class  $C_1$  (i.e., denote as positive class, +1, etc);

If  $\sum_{i=1}^D w_i x_i < 0$ , classify  $\mathbf{x}$  to class  $C_2$  (i.e., denote as negative class, -1, etc);

The decision boundary:

$$\mathcal{B} = \{\mathbf{x} : \sum_{i=1}^D w_i x_i = 0\}$$

## Geometric Interpretation of decision boundary

First, consider linear discriminant function without bias term, that is  $y = \sum_{i=1}^D w_i x_i = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$ .

If  $\sum_{i=1}^D w_i x_i > 0$ , classify  $\mathbf{x}$  to class  $C_1$  (i.e., denote as positive class, +1, etc);

If  $\sum_{i=1}^D w_i x_i < 0$ , classify  $\mathbf{x}$  to class  $C_2$  (i.e., denote as negative class, -1, etc);

The decision boundary:

$$\mathcal{B} = \{\mathbf{x} : \sum_{i=1}^D w_i x_i = 0\}$$

Think weight as vector  $\mathbf{w}$ , then the decision boundary is simply the plane **perpendicular to  $\mathbf{w}$** . (on board)

## The scale of $\mathbf{w}$ does not matter

From perspective of classification, the *scale* of the weight  $\mathbf{w}$  is irrelevant. Really matters is the **SIGN** of  $\mathbf{w}^T \mathbf{x} = \sum_{i=1}^D w_i x_i$ .

Therefore, its common to work with **normalized** weight vector  $\mathbf{w}$  which have length 1, i.e.,  $\|\mathbf{w}\| = 1$ .

## The scale of $\mathbf{w}$ does not matter

From perspective of classification, the *scale* of the weight  $\mathbf{w}$  is irrelevant. Really matters is the **SIGN** of  $\mathbf{w}^T \mathbf{x} = \sum_{i=1}^D w_i x_i$ .

Therefore, its common to work with **normalized** weight vector  $\mathbf{w}$  which have length 1, i.e.,  $\|\mathbf{w}\| = 1$ .

Question: for any vector  $\mathbf{w}$ , how to make it normalized?

$\mathbf{w}^T \mathbf{x}$  can be considered as one-dimensional data

Value  $\mathbf{w} \cdot \mathbf{x}$  is just the distance of  $\mathbf{x}$  from the origin when projected onto the vector  $\mathbf{w}$ . So can be considered as one-dimensional data.  
(on board)

## Bias term as shift

Previously, without bias term (i.e.,  $w_0 = 0$ ), the threshold would be 0:

- An input  $\mathbf{x}$  with positive projection onto  $\mathbf{w}$  would be classified as **positive**.
- An input  $\mathbf{x}$  with negative projection onto  $\mathbf{w}$  would be classified as **negative**.

## Bias term as shift

Previously, without bias term (i.e.,  $w_0 = 0$ ), the threshold would be 0:

- An input  $\mathbf{x}$  with positive projection onto  $\mathbf{w}$  would be classified as **positive**.
- An input  $\mathbf{x}$  with negative projection onto  $\mathbf{w}$  would be classified as **negative**.

The bias  $w_0$  simply moves this threshold. The role of the bias term  $w_0$  is to *shift* the decision boundary away from origin, in the direction of  $\mathbf{w}$ .

## Bias term as shift

Previously, without bias term (i.e.,  $w_0 = 0$ ), the threshold would be 0:

- An input  $\mathbf{x}$  with positive projection onto  $\mathbf{w}$  would be classified as **positive**.
- An input  $\mathbf{x}$  with negative projection onto  $\mathbf{w}$  would be classified as **negative**.

The bias  $w_0$  simply moves this threshold. The role of the bias term  $w_0$  is to *shift* the decision boundary away from origin, in the direction of  $\mathbf{w}$ .

- If  $w_0$  is positive, the boundary is shift away  $\mathbf{w}$ .
- If  $w_0$  is negative, the boundary is shift towards  $\mathbf{w}$ .



## Multiple classes

Consider  $K$ -class discriminant comprising  $K$  linear functions of the form:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

## Multiple classes

Consider  $K$ -class discriminant comprising  $K$  linear functions of the form:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

Classify  $\mathbf{x}$  to class  $C_k$  if  $y_k(\mathbf{x})$  is largest, i.e.,  $y_k(\mathbf{x}) > y_j(\mathbf{x})$  for all  $j \neq k$ .

## Multiple classes

Consider  $K$ -class discriminant comprising  $K$  linear functions of the form:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

Classify  $\mathbf{x}$  to class  $C_k$  if  $y_k(\mathbf{x})$  is largest, i.e.,  $y_k(\mathbf{x}) > y_j(\mathbf{x})$  for all  $j \neq k$ .

The decision boundary between  $C_k$  and  $C_j$  is given by  $y_k(\mathbf{x}) = y_j(\mathbf{x})$ , i.e.,

$$(\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} + (w_{k0} - w_{j0}) = 0$$

## Geometry of decision region

- If  $\mathbf{x}$  is  $D$ -dimensional, then decision boundary is  $(D - 1)$ -dimensional hyperplane.
- The decision regions of linear discriminant are singly connected and convex.

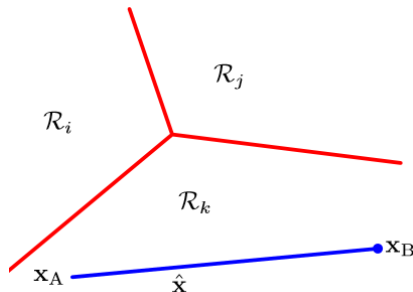


Figure: [Bishop 2006]

# Perceptron Algorithm

## Basic Settings

Given  $N$  observations:  $\{(\mathbf{x}_i, t_i), i = 1, \dots, N\}, t_i \in \{-1, +1\}$

## Basic Settings

Given  $N$  observations:  $\{(\mathbf{x}_i, t_i), i = 1, \dots, N\}$ ,  $t_i \in \{-1, +1\}$

Build model:

$$\begin{aligned} y &= \text{sign}(\mathbf{w}^T \mathbf{x}) \\ &= \begin{cases} +1, & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1, & \text{if } \mathbf{w}^T \mathbf{x} < 0 \end{cases} \end{aligned}$$

- $\mathbf{w} = (w_0, w_1, \dots, w_D)^T$ ,  $\mathbf{x} = (1, x_1, \dots, x_D)^T$

## Basic Settings

Given  $N$  observations:  $\{(\mathbf{x}_i, t_i), i = 1, \dots, N\}$ ,  $t_i \in \{-1, +1\}$

Build model:

$$\begin{aligned} y &= \text{sign}(\mathbf{w}^T \mathbf{x}) \\ &= \begin{cases} +1, & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1, & \text{if } \mathbf{w}^T \mathbf{x} < 0 \end{cases} \end{aligned}$$

- $\mathbf{w} = (w_0, w_1, \dots, w_D)^T$ ,  $\mathbf{x} = (1, x_1, \dots, x_D)^T$

Goal:

- Learn optimal/best  $\mathbf{w}^*$  from  $N$  observation to have minimum classification error.
- Make prediction: given new  $\mathbf{x}$ , predict class using  $\text{sign}(\mathbf{w}^{*T} \mathbf{x})$



## Key Observation

Note that:

$$t_i(\mathbf{w}^T \mathbf{x}_i) > 0 \iff \mathbf{x}_i \text{ is correctly classified}$$

- Correctly classified means  $\mathbf{x}_i$  is in the correct side of the hyperplane defined by  $\mathbf{w}$ .
- Only valid when we define  $t_i \in \{-1, 1\}$ .

## Key Observation

Note that:

$$t_i(\mathbf{w}^T \mathbf{x}_i) > 0 \iff \mathbf{x}_i \text{ is correctly classified}$$

- Correctly classified means  $\mathbf{x}_i$  is in the correct side of the hyperplane defined by  $\mathbf{w}$ .
- Only valid when we define  $t_i \in \{-1, 1\}$ .
- If  $t_i(\mathbf{w}^T \mathbf{x}_i) \leq 0$ , what does it tell you?

## Perceptron Algorithm [Rosenblatt 1957]

1. Initialize weight vector  $\mathbf{w}$ , and set  $\tau = 1$
2. Loop over the training data: given  $(\mathbf{x}_i, t_i)$ , if  $t_i(\mathbf{w}^T \mathbf{x}_i) \leq 0$  (i.e., **mistake has been made**), then update  $\mathbf{w}$ :
  - Mistake on positive,  $\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} + \mathbf{x}_i$
  - Mistake on negative,  $\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} - \mathbf{x}_i$
  - **Equivalently:**  $\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} + t_i \mathbf{x}_i$
3.  $\tau \leftarrow \tau + 1$  (going over whole dataset again and again, until no mistake)

## Error-driven Learning

The perceptron algorithm is error-driven, meaning that:

- If an element  $\mathbf{x}$  is correctly classified with current model  $\mathbf{w}$ , we **do nothing about weight  $\mathbf{w}$** .
- If an element  $\mathbf{x}$  is misclassified using current model, we **update weight  $\mathbf{w}$** :

$$\mathbf{w} = \begin{cases} \mathbf{w} + \mathbf{x} & \text{if } \mathbf{x} \text{ from positive } (t=+1) \text{ was classified to negative } (-1) \\ \mathbf{w} - \mathbf{x} & \text{if } \mathbf{x} \text{ from negative } (t=-1) \text{ was classified to positive } (+1) \end{cases}$$

## Error-driven learning

The goal of update is to adjust the parameters  $\mathbf{w}$  so that they are better for the current example.

## Error-driven learning

The goal of update is to adjust the parameters  $\mathbf{w}$  so that they are better for the current example.

- **1<sup>st</sup> case:**  $\mathbf{x}$  in positive class (+1) was classified as negative class (-1). The correct answer is (+1), which corresponds to:  $\mathbf{w}^T \mathbf{x} > 0$ , but we have  $\mathbf{w}^T \mathbf{x} < 0$ . We update  $\mathbf{w}$  to be  $\mathbf{w}' = \mathbf{w} + \mathbf{x}$ , then:

$$\mathbf{w}'^T \mathbf{x} = (\mathbf{w} + \mathbf{x})^T \mathbf{x} = \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{x} = \mathbf{w}^T \mathbf{x} + \|\mathbf{x}\|^2$$

because  $\|\mathbf{x}\|^2 > 0$ , so  $\mathbf{w}'^T \mathbf{x} > \mathbf{w}^T \mathbf{x}$ .

## Error-driven learning

The goal of update is to adjust the parameters  $\mathbf{w}$  so that they are better for the current example.

- **1<sup>st</sup> case:**  $\mathbf{x}$  in positive class (+1) was classified as negative class (-1). The correct answer is (+1), which corresponds to:  $\mathbf{w}^T \mathbf{x} > 0$ , but we have  $\mathbf{w}^T \mathbf{x} < 0$ . We update  $\mathbf{w}$  to be  $\mathbf{w}' = \mathbf{w} + \mathbf{x}$ , then:

$$\mathbf{w}'^T \mathbf{x} = (\mathbf{w} + \mathbf{x})^T \mathbf{x} = \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{x} = \mathbf{w}^T \mathbf{x} + \|\mathbf{x}\|^2$$

because  $\|\mathbf{x}\|^2 > 0$ , so  $\mathbf{w}'^T \mathbf{x} > \mathbf{w}^T \mathbf{x}$ .

- **2<sup>nd</sup> case:**  $\mathbf{x}$  in negative class (-1) was classified as positive class (+1). The correct answer is (-1), which corresponds to:  $\mathbf{w}^T \mathbf{x} < 0$ , but we have  $\mathbf{w}^T \mathbf{x} > 0$ . We update  $\mathbf{w}$  to be  $\mathbf{w}' = \mathbf{w} - \mathbf{x}$ , then:

$$\mathbf{w}'^T \mathbf{x} = (\mathbf{w} - \mathbf{x})^T \mathbf{x} = \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} = \mathbf{w}^T \mathbf{x} - \|\mathbf{x}\|^2$$

because  $\|\mathbf{x}\|^2 > 0$ , so  $\mathbf{w}'^T \mathbf{x} < \mathbf{w}^T \mathbf{x}$ .

## Illustration

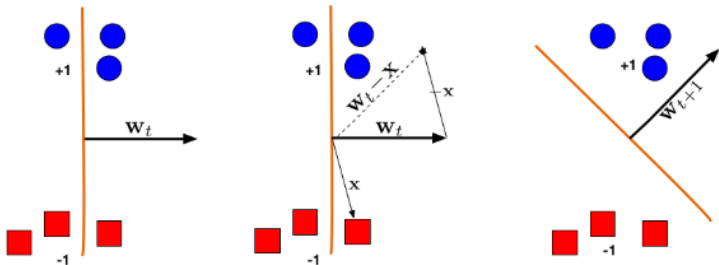


Figure: [Cornell cs4780]



## Perceptron: summary

- Loop over the training data in random order, suppose we get sequence  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k, \dots$

## Perceptron: summary

- Loop over the training data in random order, suppose we get sequence  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k, \dots$
- In  $\tau$  step, if  $\mathbf{x}_k$  is correctly classified, then  $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)}$  otherwise:

$$\mathbf{w}^{(\tau+1)} = \begin{cases} \mathbf{w}^{(\tau)} + \mathbf{x}_k & \text{if } \mathbf{x}_k \text{ is in positive class (+1)} \\ \mathbf{w}^{(\tau)} - \mathbf{x}_k & \text{if } \mathbf{x}_k \text{ is in negative class (-1)} \end{cases}$$

## Perceptron: summary

- Loop over the training data in random order, suppose we get sequence  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k, \dots$
- In  $\tau$  step, if  $\mathbf{x}_k$  is correctly classified, then  $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)}$  otherwise:

$$\mathbf{w}^{(\tau+1)} = \begin{cases} \mathbf{w}^{(\tau)} + \mathbf{x}_k & \text{if } \mathbf{x}_k \text{ is in positive class (+1)} \\ \mathbf{w}^{(\tau)} - \mathbf{x}_k & \text{if } \mathbf{x}_k \text{ is in negative class (-1)} \end{cases}$$

- Convergence theorem: regardless of the initial choice of weights, if the two classes are *linearly separable*, there exists  $\mathbf{w}$  such that:

$$\begin{cases} \mathbf{w}^T \mathbf{x} > 0 & \text{if } \mathbf{x}_k \text{ in positive class} \\ \mathbf{w}^T \mathbf{x} \leq 0 & \text{if } \mathbf{x}_k \text{ in negative class} \end{cases}$$

then the learning rule will find such solution after a finite number of steps.

## Power and Limitation

- The perceptron algorithm can be successful if we have **linearly separable classes**.
- Examples:

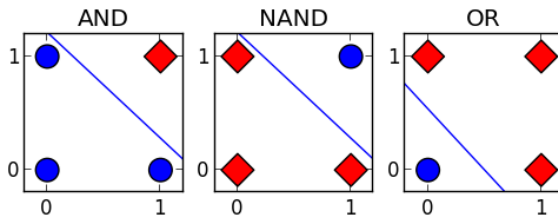


Figure: [Figures from Yue CS559-19S]

## Power and Limitation

Non-linear separable dataset in general cannot be computed by perceptron algorithm. For example, the XOR problem (Minsky 1969).

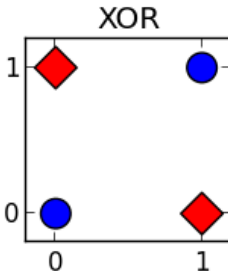


Figure: [Figure from Yue CS559-19S]

# Gradient Descent and its Variants

# Optimization

Recall:

- In previous chapter, we use probabilistic framework to describe models (e.g., **linear regression**, **logistic regression** etc) parameterized by  $\mathbf{w}$  and given training set  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , we learn such model using **maximum likelihood** or **bayesian estimation**.

## Optimization

Recall:

- In previous chapter, we use probabilistic framework to describe models (e.g., **linear regression**, **logistic regression** etc) parameterized by  $\mathbf{w}$  and given training set  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , we learn such model using **maximum likelihood** or **bayesian estimation**.
- Under proper Gaussian assumptions:



# Optimization

Recall:

- In previous chapter, we use probabilistic framework to describe models (e.g., **linear regression**, **logistic regression** etc) parameterized by  $\mathbf{w}$  and given training set  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , we learn such model using **maximum likelihood** or **bayesian estimation**.
- Under proper Gaussian assumptions:
  - For linear regression, **maximize likelihood** is equivalent to **minimize the sum-of-square loss function**

# Optimization

Recall:

- In previous chapter, we use probabilistic framework to describe models (e.g., **linear regression**, **logistic regression** etc) parameterized by  $\mathbf{w}$  and given training set  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , we learn such model using **maximum likelihood** or **bayesian estimation**.
- Under proper Gaussian assumptions:
  - For linear regression, **maximize likelihood** is equivalent to **minimize the sum-of-square loss function**
  - For bayesian linear regression, **maximize the posterior** is equivalent to **minimize the sum-or-square loss and the additional regularization term**.

# Optimization

## Recall:

- In previous chapter, we use probabilistic framework to describe models (e.g., **linear regression**, **logistic regression** etc) parameterized by  $\mathbf{w}$  and given training set  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , we learn such model using **maximum likelihood** or **bayesian estimation**.
- Under proper Gaussian assumptions:
  - For linear regression, **maximize likelihood** is equivalent to **minimize the sum-of-square loss function**
  - For bayesian linear regression, **maximize the posterior** is equivalent to **minimize the sum-or-square loss and the additional regularization term**.
- Learn parameter  $\mathbf{w}$  by minimizing (or maximizing) some loss functions.

## Optimization

- Optimization refers to tasks that minimize (or maximize) the given loss function (or called objective function)  $L(\mathbf{w})$ .

## Optimization

- Optimization refers to tasks that minimize (or maximize) the given loss function (or called objective function)  $L(\mathbf{w})$ .
- Global optimum and local optimum.

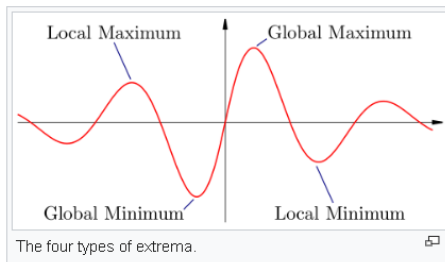


Figure: [From Wikibooks]

## Optimization

- Optimization refers to tasks that minimize (or maximize) the given loss function (or called objective function)  $L(\mathbf{w})$ .
- Global optimum and local optimum.

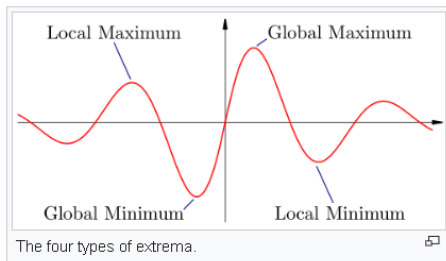


Figure: [From Wikibooks]

- Goal: (1) find the global optimum (feasible when objective function is convex), (2) find the "good" local optimum (in most non-convex problems and deep learning).

## Gradient Descent

- Minimize loss  $L(\mathbf{w})$ , where the underlying model (i.e., hypothesis) is parameterized by  $\mathbf{w}$ .

## Gradient Descent

- Minimize loss  $L(\mathbf{w})$ , where the underlying model (i.e., hypothesis) is parameterized by  $\mathbf{w}$ .
- Procedure:
  - Initialize  $\mathbf{w}_0$
  - $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_{\tau} \nabla L(\mathbf{w})$ , until convergence



## Gradient Descent

- Minimize loss  $L(\mathbf{w})$ , where the underlying model (i.e., hypothesis) is parameterized by  $\mathbf{w}$ .
- Procedure:
  - Initialize  $\mathbf{w}_0$
  - $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_{\tau} \nabla L(\mathbf{w})$ , until convergence
- First-order optimization algorithm. More advanced techniques exist but may need second-order derivative information.

## Gradient Descent

- Minimize loss  $L(\mathbf{w})$ , where the underlying model (i.e., hypothesis) is parameterized by  $\mathbf{w}$ .
- Procedure:
  - Initialize  $\mathbf{w}_0$
  - $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_{\tau} \nabla L(\mathbf{w})$ , until convergence
- First-order optimization algorithm. More advanced techniques exist but may need second-order derivative information.

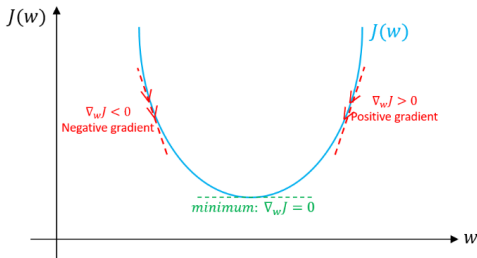


Figure: [From I. Dabbura]

## Key Ingredient: initialization

- Initialization  $\mathbf{w}_0$ :

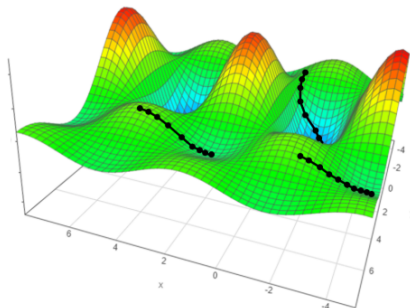


Figure: [From Offconvex.org]

## Key Ingredient: learning rate

- Learning rate  $\eta_T$ :

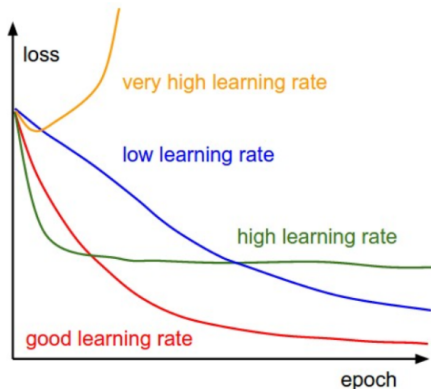


Figure: [From stanford cs231n]

## Variants of gradient descent

Three types based on number of training data used for updating parameter  $\mathbf{w}$ : suppose  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$

## Variants of gradient descent

Three types based on number of training data used for updating parameter  $\mathbf{w}$ : suppose  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$

- Batch gradient descent: use **all** data.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_{\tau} \nabla L(\mathbf{w}; \mathbf{x}_{1:N}, t_{1:N})$$

## Variants of gradient descent

Three types based on number of training data used for updating parameter  $\mathbf{w}$ : suppose  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$

- Batch gradient descent: use **all** data.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_{\tau} \nabla L(\mathbf{w}; \mathbf{x}_{1:N}, t_{1:N})$$

- Mini-batch gradient descent: use **relative small batch** of data (e.g., batch size  $b_s$  could be 16, 64 etc)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_{\tau} \nabla L(\mathbf{w}; \mathbf{x}_{i:b_s}, t_{i:b_s})$$

## Variants of gradient descent

Three types based on number of training data used for updating parameter  $\mathbf{w}$ : suppose  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$

- Batch gradient descent: use **all** data.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_\tau \nabla L(\mathbf{w}; \mathbf{x}_{1:N}, t_{1:N})$$

- Mini-batch gradient descent: use **relative small batch** of data (e.g., batch size  $b_s$  could be 16, 64 etc)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_\tau \nabla L(\mathbf{w}; \mathbf{x}_{i:b_s}, t_{i:b_s})$$

- Stochastic gradient descent: use only **one** example (e.g.,  $\{\mathbf{x}_i, t_i\}$ )

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta_\tau \nabla L(\mathbf{w}; \mathbf{x}_i, t_i)$$



## Pros and Cons

- Batch gradient descent:

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
  - Slow, not infeasible when dataset is large.

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.
- Mini-batch gradient descent:

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.
- Mini-batch gradient descent:
  - + Faster in each iteration.

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.
- Mini-batch gradient descent:
  - + Faster in each iteration.
  - + Add noise to the learning process, improve the generalization.



## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.
- Mini-batch gradient descent:
  - + Faster in each iteration.
  - + Add noise to the learning process, improve the generalization.
    - The learning may go back and forth due to noise, may need learning rate decay near optimum.

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.
- Mini-batch gradient descent:
  - + Faster in each iteration.
  - + Add noise to the learning process, improve the generalization.
    - The learning may go back and forth due to noise, may need learning rate decay near optimum.
- Stochastic gradient descent:

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.
- Mini-batch gradient descent:
  - + Faster in each iteration.
  - + Add noise to the learning process, improve the generalization.
    - The learning may go back and forth due to noise, may need learning rate decay near optimum.
- Stochastic gradient descent:
  - + Shares the similar advantages as mini-batch version.

## Pros and Cons

- Batch gradient descent:
  - + Accurate and unbiased gradient estimation.
  - + Theoretically, it can converge to the global optimum (for convex function) and local optimum.
    - Slow, not infeasible when dataset is large.
    - Easy to trap into local modes.
- Mini-batch gradient descent:
  - + Faster in each iteration.
  - + Add noise to the learning process, improve the generalization.
    - The learning may go back and forth due to noise, may need learning rate decay near optimum.
- Stochastic gradient descent:
  - + Shares the similar advantages as mini-batch version.
    - Can be quite slow to converge, large variance for gradient estimation.

## Illustration

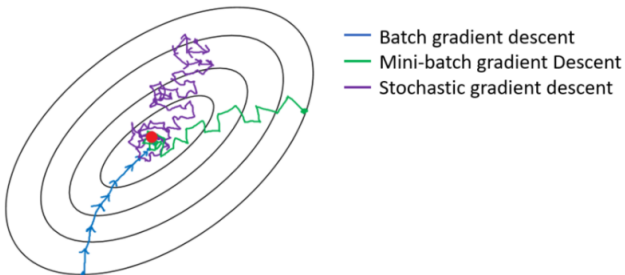


Figure: [From I. Dabbura]

## SGD for linear regression

- Linear regression model:  $y = \mathbf{w}^T \mathbf{x} + \epsilon, \epsilon \sim N(0, I)$

## SGD for linear regression

- Linear regression model:  $y = \mathbf{w}^T \mathbf{x} + \epsilon, \epsilon \sim N(0, I)$
- Given  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , learn  $\mathbf{w}$  to maximize the likelihood  $\prod_{i=1}^N p(t_i | \mathbf{w})$ .

## SGD for linear regression

- Linear regression model:  $y = \mathbf{w}^T \mathbf{x} + \epsilon, \epsilon \sim N(0, I)$
- Given  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , learn  $\mathbf{w}$  to maximize the likelihood  $\prod_{i=1}^N p(t_i | \mathbf{w})$ .
- Equivalently, minimize the sum-of-square loss function  $L(\mathbf{w}) = E_D(\mathbf{w}) = \sum_{i=1}^N (t_i - \mathbf{w}^T \mathbf{x}_i)^2$ .



## SGD for linear regression

- Linear regression model:  $y = \mathbf{w}^T \mathbf{x} + \epsilon, \epsilon \sim N(0, I)$
- Given  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , learn  $\mathbf{w}$  to maximize the likelihood  $\prod_{i=1}^N p(t_i | \mathbf{w})$ .
- Equivalently, minimize the sum-of-square loss function  $L(\mathbf{w}) = E_D(\mathbf{w}) = \sum_{i=1}^N (t_i - \mathbf{w}^T \mathbf{x}_i)^2$ .
- If use Stochastic Gradient Descent (SGD), we consider the loss  $l(\mathbf{w}; \mathbf{x}_i, t_i) = (t_i - \mathbf{w}^T \mathbf{x}_i)^2$  for single data.

## SGD for linear regression

- Linear regression model:  $y = \mathbf{w}^T \mathbf{x} + \epsilon, \epsilon \sim N(0, I)$
- Given  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ , learn  $\mathbf{w}$  to maximize the likelihood  $\prod_{i=1}^N p(t_i | \mathbf{w})$ .
- Equivalently, minimize the sum-of-square loss function  $L(\mathbf{w}) = E_D(\mathbf{w}) = \sum_{i=1}^N (t_i - \mathbf{w}^T \mathbf{x}_i)^2$ .
- If use Stochastic Gradient Descent (SGD), we consider the loss  $l(\mathbf{w}; \mathbf{x}_i, t_i) = (t_i - \mathbf{w}^T \mathbf{x}_i)^2$  for single data.
- SGD:

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta_\tau \nabla l(\mathbf{w}; \mathbf{x}_i, t_i) \\ &= \mathbf{w}^{(\tau)} - 2\eta_\tau (t_i - \mathbf{w}^T \mathbf{x}_i)(-\mathbf{x}_i)\end{aligned}$$

## SGD for logistic regression

- Logistic regression model:

$$p(C_1|\mathbf{x}) = y = \sigma(\mathbf{w}^T \mathbf{x})$$

$$p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$$

## SGD for logistic regression

- Logistic regression model:

$$p(C_1|\mathbf{x}) = y = \sigma(\mathbf{w}^T \mathbf{x})$$

$$p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$$

- Given  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$  and  $t_i \in \{0, 1\}$ , learn  $\mathbf{w}$  to maximize the likelihood

$$\prod_{i=1}^N p(t_i|\mathbf{w}) = \prod_{i=1}^N y_i^{t_i} (1 - y_i)^{1-t_i}.$$

## SGD for logistic regression

- Logistic regression model:

$$p(C_1|\mathbf{x}) = y = \sigma(\mathbf{w}^T \mathbf{x})$$

$$p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$$

- Given  $\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$  and  $t_i \in \{0, 1\}$ , learn  $\mathbf{w}$  to maximize the likelihood

$$\prod_{i=1}^N p(t_i|\mathbf{w}) = \prod_{i=1}^N y_i^{t_i} (1 - y_i)^{1-t_i}.$$

- Equivalently, minimize the cross-entropy loss function:

$$L(\mathbf{w}) = - \sum_{i=1}^N [t_i \log y_i + (1 - t_i) \log(1 - y_i)]$$

$$y_i = \sigma(\mathbf{w}^T \mathbf{x}_i)$$

## SGD for logistic regression

- Use SGD, we consider the loss

$$l(\mathbf{w}; \mathbf{x}_i, t_i) = -[t_i \log y_i + (1 - t_i) \log(1 - y_i)]$$

## SGD for logistic regression

- Use SGD, we consider the loss

$$l(\mathbf{w}; \mathbf{x}_i, t_i) = -[t_i \log y_i + (1 - t_i) \log(1 - y_i)]$$

- SGD:

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta_{\tau} \nabla l(\mathbf{w}; \mathbf{x}_i, t_i) \\ &= \mathbf{w}^{(\tau)} - \eta_{\tau} (\sigma(\mathbf{w}^{(\tau)T} \mathbf{x}_i) - t_i) \mathbf{x}_i\end{aligned}$$

## SGD for perceptron

Q: is it possible to view percetron algorithm in SGD framework?



## SGD for perceptron

Q: is it possible to view percetron algorithm in SGD framework?

A: Yes.

## SGD for perceptron

Q: is it possible to view percetron algorithm in SGD framework?

A: Yes.

- Binary classification:  $y = \text{sign}(\mathbf{w}^T \mathbf{x})$

## SGD for perceptron

Q: is it possible to view perceptron algorithm in SGD framework?

A: Yes.

- Binary classification:  $y = \text{sign}(\mathbf{w}^T \mathbf{x})$
- Loss function: total number of misclassified patterns

$$L(\mathbf{w}) = \sum_{i=1}^N \mathbb{I}(y_i \neq t_i)$$

## SGD for perceptron

Q: is it possible to view perceptron algorithm in SGD framework?

A: Yes.

- Binary classification:  $y = \text{sign}(\mathbf{w}^T \mathbf{x})$
- Loss function: total number of misclassified patterns

$$L(\mathbf{w}) = \sum_{i=1}^N \mathbb{I}(y_i \neq t_i)$$

- However, above loss function cannot be optimized using gradient based learning methods (e.g., SGD).

## SGD for perceptron

Q: is it possible to view perceptron algorithm in SGD framework?

A: Yes.

- Binary classification:  $y = \text{sign}(\mathbf{w}^T \mathbf{x})$
- Loss function: total number of misclassified patterns

$$L(\mathbf{w}) = \sum_{i=1}^N \mathbb{I}(y_i \neq t_i)$$

- However, above loss function cannot be optimized using gradient based learning methods (e.g., SGD).
- *Perceptron criterion*:

$$L(\mathbf{w}) = - \sum_{i=1}^N t_i (\mathbf{w}^T \mathbf{x}_i) \mathbb{I}(y_i \neq t_i)$$

## SGD for perceptron

- Perceptron criterion is differentiable w.r.t  $\mathbf{w}$ , could define the loss:

$$l(\mathbf{w}; \mathbf{x}_i, t_i) = -t_i(\mathbf{w}^T \mathbf{x}_i) \mathbb{I}(y_i \neq t_i)$$

## SGD for perceptron

- Perceptron criterion is differentiable w.r.t  $\mathbf{w}$ , could define the loss:

$$l(\mathbf{w}; \mathbf{x}_i, t_i) = -t_i(\mathbf{w}^T \mathbf{x}_i) \mathbb{I}(y_i \neq t_i)$$

- SGD:

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta_\tau \nabla l(\mathbf{w}; \mathbf{x}_i, t_i) \\ &= \mathbf{w}^{(\tau)} + \eta_\tau t_i \mathbf{x}_i \mathbb{I}(y_i \neq t_i) \end{aligned}$$

## SGD for perceptron

- Perceptron criterion is differentiable w.r.t  $\mathbf{w}$ , could define the loss:

$$l(\mathbf{w}; \mathbf{x}_i, t_i) = -t_i(\mathbf{w}^T \mathbf{x}_i) \mathbb{I}(y_i \neq t_i)$$

- SGD:

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta_\tau \nabla l(\mathbf{w}; \mathbf{x}_i, t_i) \\ &= \mathbf{w}^{(\tau)} + \eta_\tau t_i \mathbf{x}_i \mathbb{I}(y_i \neq t_i) \end{aligned}$$

- Take learning rate  $\eta_\tau = 1$ , we get the perceptron algorithm discussed before.
  - If make mistake on positive example  $\mathbf{x}_i$ ,  $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \mathbf{x}_i$ .
  - If make mistake on negative example  $\mathbf{x}_i$ ,  $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \mathbf{x}_i$ .



## acknowledgement

Part of the slides are based on *A course in Machine Learning*.

Part of the materials are inspired by *Princeton COS495 S16*.

Some figures are from *Cornell cs 4780*.