EE416: Introduction to Image Processing and Computer Vision

Il Yong Chun
Department of Electrical and Computer Engineering, the University of Hawai'i, Mānoa

November 1, 2021

7 Image pattern recognition

7.1 Minimum-distance classifier: Deterministic approach

The **minimum-distance classifier** computes a distance-based measure between a pattern vector and each of the class prototypes. It then assigns the pattern to the class of its closest prototype:

- The prototype vectors of the minimum-distance classifiers usually are the mean vectors of the various pattern classes:
$$\mathbf{m}_c = \frac{1}{M_c} \sum_{\mathbf{x} \in S_c} \mathbf{x}, \quad c = 1, \ldots, C,$$
where M_c is the number of pattern vectors used to compute the cth mean vector, S_c is the cth pattern class, and C is the number of classes.
- If one uses the Euclidean distance to determine similarity, the minimum-distance classifier computes
$$D_c(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_c\|_2, \quad c = 1, \ldots, C.$$
The classifier then assigns a pattern $\mathbf{x}$ to class S_c if
$$D_c(\mathbf{x}) < D_{c'}(\mathbf{x}), \quad c = 1, \ldots, C, c' \neq c.$$
Ties, i.e., $D_c(\mathbf{x}) = D_{c'}(\mathbf{x})$, are resolved arbitrarily.
Selecting the smallest distance is equivalent to evaluating the functions
$$d_c(\mathbf{x}) = \mathbf{m}_c^T \mathbf{x} - \frac{1}{2} \mathbf{m}_c^T \mathbf{m}_c, \quad c = 1, \ldots, C, \tag{1}$$
and assigning $\mathbf{x}$ to class S_c if
$$d_c(\mathbf{x}) > d_{c'}(\mathbf{x}), \quad c = 1, \ldots, C, c' \neq c.$$
For recognition, functions of this form are referred to as **decision** or **discriminant functions**.
- The **decision boundary** separates class S_c from $S_{c'}$ given by the values of $\mathbf{x}$:
$$d_c(\mathbf{x}) = d_{c'}(\mathbf{x}) \tag{2}$$
The decision boundaries for a minimum-distance classifier satisfies
$$d_{c,c'}(\mathbf{x}) = d_c(\mathbf{x}) - d_{c'}(\mathbf{x}) = (\mathbf{m}_c - \mathbf{m}_{c'})^T \mathbf{x} - \frac{1}{2} (\mathbf{m}_c - \mathbf{m}_{c'})^T (\mathbf{m}_c + \mathbf{m}_{c'}) = 0$$
The boundary give by this equation is the perpendicular bisector of the line segment joining $\mathbf{m}_c$ and $\mathbf{m}_{c'}$. For 2D features, i.e., $\{\mathbf{m}_c \in \mathbb{R}^2 : \forall c\}$, the perpendicular bisector is a line; for 3D features, i.e., $\{\mathbf{m}_c \in \mathbb{R}^3 : \forall c\}$, it is a plane; for features larger than 3D, it is called a **hyperplane**.

7.2 Bayes classifier: Statistical approach

This sections introduces a probabilistic approach to pattern recognition. Considering probability is important in pattern recognition, because of the randomness under which pattern classes normally are generated. In particular, we will derive a classification approach that is optimal in the sense that, on average, it yields the lowest probability committing classification errors.

7.2.1 Bayes classifier for arbitrary pattern classes

Conditional average risk.

- $P(S_c|\mathbf{x})$ denotes the probability that a pattern vector $\mathbf{x}$ comes from class S_c.
- The **loss** function $L(a_c|S_{c'})$ is the loss incurred for taking action a_c when the class is $S_{c'}$, $c, c' = 1, \ldots, C$.
  The zero-one loss function is particularly common:

$$L(a_c|S_{c'}) = \begin{cases} 0, & c = c', \\ 1, & c \neq c', \end{cases} \qquad c, c' = 1, \ldots, C. \tag{3}$$

  It assigns no loss to a correct decision and uniform unit loss to an incorrect decision.
- **Conditional average risk** (or loss):

$$R_{c'}(\mathbf{x}) = \sum_{c=1}^{C} L(a_c|S_{c'})P(S_c|\mathbf{x}), \quad c' = 1, \ldots, C, \tag{4}$$

  i.e., an average risk in assigning $\mathbf{x}$ to class $S_{c'}$.

Bayes rule with prior distribution and the class-conditional (or likelihood) density.

- Bayes' rule: $P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$. In plain English, posterior $= \frac{\text{likelihood} \times \text{prior}}{\text{observation}}$.
- The **a priori** or **prior** probability $P(S_c)$ reflects our knowledge of how likely we expect a certain class before we can actually observe.
- The probability $P(S_c|\mathbf{x})$ is a **posterior probability**, i.e., the probability of a certain class given our observation $\mathbf{x}$.
- The **class-conditional pdf** $p(\mathbf{x}|S_c)$ is the pdf for $\mathbf{x}$, given that the class is S_c.
- If we know the prior distribution $P(S_c)$ and the class-conditional density $p(\mathbf{x}|S_c)$, how does this affect the posterior probability $P(S_c|\mathbf{x})$? Using Bayes formula, we observe

$$\begin{aligned} P(S_c, \mathbf{x}) &= P(S_c|\mathbf{x})p(\mathbf{x}) = p(\mathbf{x}|S_c)P(S_c), \\ P(S_c|\mathbf{x}) &= \boxed{??}. \end{aligned} \tag{5}$$

Bayes classifier. Using the Bayes result (5), we rewrite the average risk (4) as

$$R_{c'}(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{c=1}^{C} L(a_c|S_{c'})p(\mathbf{x}|S_c)P(S_c), \quad c' = 1, \ldots, C$$

Dropping the term $1/p(\mathbf{x})$ (because $1/p(\mathbf{x}) > 0$ and common for all $R_{c'}(\mathbf{x})$, $c' = 1, \ldots, C$) gives

$$R_{c'}(\mathbf{x}) = \sum_{c=1}^{C} L(a_c|S_{c'})p(\mathbf{x}|S_c)P(S_c), \quad c' = 1, \ldots, C. \tag{6}$$

If the classifier computes $R_1(\mathbf{x}), R_2(\mathbf{x}), \ldots, R_C(\mathbf{x})$ for each pattern $\mathbf{x}$ and assigns the pattern to the class with the smallest risk, the *total* average risk with respect to all decisions will be minimum. The classifier that minimizes the *total* average risk is called the **Bayes classifier**.

- The Bayes classifier assigns a pattern $\mathbf{x}$ to class S_c if

$$R_c(\mathbf{x}) < R_{c'}(\mathbf{x}), \quad c = 1, \ldots, C, c' \neq c.$$

  where R_c is given as in (6).

Bayes classifier with zero-one loss. We first rewrite the zero-one loss (3) as

$$L(a_c|S_{c'}) = 1 - \delta_{c,c'}$$

where $\delta_{c,c'} = 1$ if $c = c'$, and $\delta_{c,c'} = 0$ if $c \neq c'$. Substituting this into (6) yields

$$\begin{aligned} R_{c'}(\mathbf{x}) &= \sum_{c=1}^{C} (1 - \delta_{c,c'}) p(\mathbf{x}|S_c) P(S_c) \\ &= \boxed{??}, \quad c' = 1, \ldots, C, \end{aligned}$$

where the first term holds by the law of total probability (or partition theorem); see [T2, §2.5.The Law of Total Probability].

- The Bayes classifier assigns a pattern $\mathbf{x}$ to class S_c if

$$\begin{aligned} p(\mathbf{x}) - p(\mathbf{x}|S_c)P(S_c) < p(\mathbf{x}) - p(\mathbf{x}|S_{c'})P(S_{c'}), \quad c = 1, \ldots, C, c' \neq c, \quad \text{or, equivalently,} \\ p(\mathbf{x}|S_c)P(S_c) > p(\mathbf{x}|S_{c'})P(S_{c'}), \quad c = 1, \ldots, C, c' \neq c. \end{aligned}$$

- Thus, the Bayes classifier for a 0-1 loss function computes discriminant functions of the form

$$d_c(\mathbf{x}) = p(\mathbf{x}|S_c)P(S_c), \quad c = 1, \ldots, C, \tag{7}$$

  and assigns a pattern to class S_c if $d_c(\mathbf{x}) > d_{c'}(\mathbf{x})$ for all $c' \neq c$. This is exactly the same processes in (2), but we are now dealing with discriminant functions that have been shown to be optimal in the sense that they minimize the average risk in misclassification.

For the optimality of Bayes discriminant functions to hold, the pdfs of the patterns in each class, $p(\mathbf{x}|S_c)$, and the probability of occurrence of each class, $P(S_c)$, must be known.

- The latter requirement usually is not a problem. For instance, if all classes are equally likely to occur, then $P(S_c) = 1/C$. Even if this condition is not true, these probabilities generally can be inferred from knowledge of the problem.
- Estimating the pdf $p(\mathbf{x}|S_c)$ is more difficult. If the pattern vectors are N-dimensional, then $p(\mathbf{x}|S_c)$ is a function of N variables. If the form of $p(\mathbf{x}|S_c)$ is not known, estimating it requires using multivariate estimation methods. These methods are difficult to apply in practice, especially if the number of representative patterns from each class is not large, or if the pdfs are not well behaved. For theses reasons, uses of the Bayes classifier often are based on assuming an analytical expression for the densify functions.

7.2.2 Bayes classifier for Gaussian pattern classes

If we assume that $p(\mathbf{x}|S_c)$ follows certain distribution, then the pdf estimation problem simplifies to problem of estimating the necessary parameters from sample patterns from each class using training patterns. By far, the most prevalent form assumed for $p(\mathbf{x}|S_c)$ is the Gaussian pdf.

In the N-dimensional case, i.e., $\mathbf{x} \in \mathbb{R}^N$, the Gaussian density of the vectors in the cth pattern class has the form

$$p(\mathbf{x}|S_c) = \frac{1}{(2\pi)^{N/2} \det(\mathbf{V}_c)^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_c)^T \mathbf{V}_c^{-1} (\mathbf{x}-\mathbf{m}_c)} \tag{8}$$

where each density is specified completely by its mean vector $\mathbf{m}_c$ and covariance matrix $\mathbf{V}_c$:

$$\mathbf{m}_c = \mathbb{E}[\mathbf{x}|S_c]$$
$$\mathbf{V}_c = \mathbb{E}[(\mathbf{x}-\mathbf{m}_c)(\mathbf{x}-\mathbf{m}_c)^T|S_c]$$

where $\mathbb{E}[(\cdot)|S_c]$ denotes the expected value of the argument over the patterns of class S_c (i.e., conditional expectation). Approximating the expected value $\mathbb{E}[(\cdot)|S_c]$ by the sample average yields an estimate of the mean vector and covariance matrix (see [wiki]):

- $\mathbf{m}_c = \dfrac{1}{M_c}\displaystyle\sum_{\mathbf{x}\in S_c}\mathbf{x}$,
- $\mathbf{V}_c = \dfrac{1}{M_c}\displaystyle\sum_{\mathbf{x}\in S_c}\mathbf{x}\mathbf{x}^T - \mathbf{m}_c\mathbf{m}_c^T$,

where M_c is the number of sample pattern vectors from class S_c, and the summation is take over these vectors.

Example: Consider 1D patterns ($N = 1$) involving two pattern classes ($C = 2$), governed by Gaussian densities, with means m_1 and m_2, and standard deviations σ_1 and σ_2, respectively. From (7), the Bayesian discriminant functions are in the form of

$$d_c(x) = p(x|S_c)P(S_c) = \frac{1}{\sqrt{2\pi}\sigma_c}e^{-\frac{(x-m_c)}{2\sigma_c^2}}P(S_c), \quad c = 1, 2.$$

Sketch a plot of the pdfs for the two classes:
??

The boundary between the two classes is a single point, x_0, such that $d_1(x_0) = d_2(x_0)$. If the two classes are equally likely to occur, then $P(S_1) = P(S_2) = 1/2$, and the decision boundary is the value of x_0 for which $p(x_0|S_1) = p(x_0|S_2)$. This point is the intersection of the two pdfs.

Considering the exponential form of the Gaussian density (8), taking the log of (7) gives

$$d_c(\mathbf{x}) = \ln\left(p(\mathbf{x}|S_c)P(S_c)\right) = \ln p(\mathbf{x}|S_c) + \ln P(S_c). \tag{9}$$

Because the logarithm is monotonically increasing function, the expression (9) is equivalent to (7) in terms of classification performance. Now substituting (8) into (9) yields

$$d_c(\mathbf{x}) = \ln P(S_c) - \frac{N}{2}\ln 2\pi - \frac{1}{2}\ln\det(\mathbf{V}_c) - \frac{1}{2}(\mathbf{x}-\mathbf{m}_c)^T\mathbf{V}_c^{-1}(\mathbf{x}-\mathbf{m}_c)$$

The second term can be disregarded because it is the same for all classes.

- Thus, the Bayes discriminant functions for Gaussian pattern classes under the condition of a 0-1 loss function is given by

$$d_c(\mathbf{x}) = \ln P(S_c) - \frac{1}{2}\ln\det(\mathbf{V}_c) - \frac{1}{2}(\mathbf{x}-\mathbf{m}_c)^T\mathbf{V}_c^{-1}(\mathbf{x}-\mathbf{m}_c), \quad c = 1, \ldots, C. \tag{10}$$

  The discriminant functions in (10) are **hyperquadrics**, e.g., quadratic functions in N-dimensional space.

Placing a second-order decision boundary between each pair of pattern classes is the best that a Bayes classifier can do for Gaussian patterns.

In other words, no other boundary would yield a lower average risk in classification.

- If covariance matrices are equal, i.e., $\{\mathbf{V}_c = \mathbf{V} : c = 1, \ldots, C\}$, then we can drop all terms that do not depend on c in (10):

$$d_c(\mathbf{x}) = \ln P(S_c) + \mathbf{x}^T\mathbf{V}^{-1}\mathbf{m}_c - \frac{1}{2}\mathbf{m}_c^T\mathbf{V}^{-1}\mathbf{m}_c, \quad c = 1, \ldots, C. \tag{11}$$

This is linear discriminant functions (hyperplanes) for $c = 1, \ldots, C$.

- If in addition, $\mathbf{V} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, and classes are equality likely, i.e., $P(S_c) = 1/C$, $c = 1, \ldots, C$, then we can drop the term $P(S_c)$ (it is the same for all c) in (11):

$$d_c(\mathbf{x}) = \mathbf{x}^T\mathbf{m}_c - \frac{1}{2}\mathbf{m}_c^T\mathbf{m}_c, \quad c = 1, \ldots, C.$$

This is identical to the discriminant functions for a minimum-distance classifier in (1). In other words, the minimum-distance classifier is optimum in the Bayse sense if

1) the pattern classes follow a Gaussian distribution,
2) covariance matrices are equal to the identity matrix, and
3) all classes are equally likely to occur.

Gaussian pattern classes satisfying these conditions are spherical clouds of identical shape in N dimensions, called **hyperspheres**. The minimum distance classifier establishes a hyperplane between every pair of classes, with the property that the hyperplane is the perpendicular bisector of the line segment joining the center of the pair of hyperspheres.

7.2.3 Naive Bayes classifier

If we assume that all of our individual features, x_n, $n = 1, \ldots, N$, are conditional independent given class S_c, then we have

$$p(S_c|\mathbf{x}) \propto \prod_{n=1}^{N} p(x_n|S_c).$$

By using the conditional independence assumption, one can construct a naive Bayes classifier. A naive Bayes classifier has two practical benefits:

- It can circumvent the dimensionality issue in estimating $p(\mathbf{x}|S_c)$ (see §7.2.1. Bayes classifier with zero-one loss).
- It performs with surprising accuracy even in cases violating the underlying independence assumption.

7.3 Neural networks and optimization: Perceptron learning

7.3.1 Quick review: Gradient descent method

The gradient: Recall that the gradient of a function $f(\mathbf{x}) : \mathbb{R}^N \to \mathbb{R}$ at $\mathbf{z}$ is

$$\nabla f(\mathbf{z}) = \left[\frac{\partial f}{\partial x_1}(\mathbf{z}), \ldots, \frac{\partial f}{\partial x_N}(\mathbf{z})\right]^T.$$

- The gradient points in the direction of maximum growth.
- $\nabla f(\mathbf{z})$: the direction of greatest increase of $f(\mathbf{x})$ at $\mathbf{z}$.

- The gradient is perpendicular to the iso-contours of $f(\cdot)$.

Critical point conditions: Let $f(\mathbf{x}) : \mathbb{R}^N \to \mathbb{R}$ be twice continuously differentiable, where $\mathbf{x} \in \mathbb{R}^N$. The point $\mathbf{x}^\star$ is a a local minimum of $f(\mathbf{x})$ iff

- f has zero gradient at $\mathbf{x}^\star$:

$$\boxed{\nabla f(\mathbf{x}^\star) = 0.}$$

- the Hessian of f at $\mathbf{x}^\star$ is positive definite:

$$\boxed{\mathbf{y}^T \nabla^2 f(\mathbf{x}^\star)\mathbf{y} > 0, \quad \text{for all non-zero } \mathbf{y} \in \mathbb{R}^N,}$$

  where $\left[\nabla^2 f(\mathbf{x})\right]_{m,n} = \frac{\partial^2 f}{\partial x_m \partial x_n}(\mathbf{x})$, for $m, n = 1, \ldots, N$.

Gradient descent method: The gradient descent method is a simple minimization technique:

- pick initial estimate $\mathbf{x}^{(0)}$
- follow the negative gradient:

$$\boxed{x^{(i+1)} = x^{(i)} - \alpha \nabla f(x^{(i)}),} \tag{12}$$

  where α is called the **step size** in optimization literature, and called the **learning rate** in machine learning literature. The α value should be carefully chosen:

  - too small α can slow down convergence;
  - too large α may make descent diverge.

7.3.2 Linear discriminant

A single perceptron unit learns a linear boundary between two linearly separable pattern classes. A linear boundary in 2D is a straight line with equation $y = ax + b$. The coefficient b that is not multiplied by a coordinate is often referred to as the **bias**.

For N-dimensional features $\mathbf{x}$, the test would be against **hyperplane** whose equation is

$$\sum_{n=1}^{N} w_n x_n + b = 0$$

or in vector form as

$$\mathbf{w}^T \mathbf{x} + b = 0$$

where $\mathbf{w} \in \mathbb{R}^N$ and $\mathbf{x} \in \mathbb{R}^N$ are column vectors. We refer to $\mathbf{w}$ as a **weight** vector, and to b as a **bias**. The classifier implements the line decision rule

$$d(\mathbf{x}) = \text{sign}(g(\mathbf{x})), \quad g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b.$$

This is called a **perceptron model**; for the perceptron, the **activation function** is a thresholding function, sign$(\cdot)$. Its properties are given as follows:

- The decision boundary is the hyperplane that is perpendicular to the weight vector, and has its distance to the origin as $b/\|\mathbf{w}\|_2$.
- The distance from point $\mathbf{x}$ to the decision boundary is given by $g(\mathbf{x})/\|\mathbf{w}\|_2$.
- $g(\mathbf{x}) = 0$ for points on the plane;
  $g(\mathbf{x}) > 0$ on the side $\mathbf{w}$ points to ("positive side");
  $g(\mathbf{x}) < 0$ on the "negative side".

Suppose that a linearly separable training set, $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$, is given. There exists no error iff

$$y_m = 1 \text{ and } g(\mathbf{x}_m) > 0 \quad \text{or} \quad y_m = -1 \text{ and } g(\mathbf{x}_m) < 0, \quad \forall m,$$

i.e.,

$$y_m \cdot g(\mathbf{x}_m) > 0, \quad \forall m.$$

This allows a concise expression for the situation of "no training error" or "zero empirical risk".

7.3.3 Perceptron learning with gradient descent

Initial learning model: Based on the necessary and sufficient condition for zero empirical risk,

$$\boxed{y_m \cdot (\mathbf{w}^T \mathbf{x}_m + b) > 0, \quad \forall m.}$$

we construct the following reward maximization problem:

$$\underset{\mathbf{w},b}{\text{argmax}} \sum_{m=1}^{M} y_m \cdot (\mathbf{w}^T \mathbf{x}_m + b)$$

or, equivalently, the following loss (or cost) minimization problem:

$$\underset{\mathbf{w},b}{\text{argmin}}\ L(\mathbf{w}, b), \quad \boxed{L(\mathbf{w}, b) \triangleq -\sum_{m=1}^{M} y_m \cdot (\mathbf{w}^T \mathbf{x}_m + b).} \tag{13}$$

Next, we apply the gradient descent method (12) to the loss minimization problem (13). One can derive the gradients straightforwardly:

$$\frac{\partial L(\mathbf{w}, b)}{\partial \mathbf{w}} = \boxed{??}$$
$$\frac{\partial L(\mathbf{w}, b)}{\partial b} = \boxed{??}$$

Using the derived gradients, the perceptron learning algorithm becomes

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \alpha \sum_{m=1}^{M} y_m \mathbf{x}_m$$
$$b^{(i+1)} = b^{(i)} + \alpha \sum_{m=1}^{M} y_m.$$

for $i \geq 0$, where $\mathbf{w}^{(0)}$ and $b^{(0)}$ are arbitrarily chosen. It is necessary (but not sufficient) for α to be in the range $0 < \alpha < 2$ for the above algorithm to converge. (One can prove generalized results for gradient descent methods using the Lipschitz continuity, but the proof is omitted in this lecture. This will be covered in EE616.) A typical range for α is $0.1 < \alpha < 1$.

Observe that

- if a training set is linearly separable, there exists at least a pair $(\mathbf{w}, b)$ such that $L(\mathbf{w}, b) < 0$.
- any minimum that is equal to or better than this will do gradient descent.

Note that one critical problem exists in using the loss $L(\mathbf{w}, b)$ in (13):

- The loss function $L(\mathbf{w}, b)$ in (13) is not bounded below.

- If $L(\mathbf{w}, b) < 0$, it is possible that $L(\mathbf{w}, b) \to -\infty$ when $\mathbf{w}$ or b is/are multiplied by some positive number.
- If the minimum point is at $-\infty$, this is numerically bad.

Modified learning model: One can avoid $L(\mathbf{w}, b)$ in (13) < 0 by restricting attentions to the points that are incorrectly classified. Specifically, at each iteration, we define set of errors

$$E = \{\mathbf{x}_m : y_m(\mathbf{w}^T\mathbf{x}_m + b) < 0\},$$

and construct the loss

$$\boxed{L_E(\mathbf{w}, b) = -\sum_{\mathbf{x}_m \in E} y_m(\mathbf{w}^T\mathbf{x}_m + b).} \tag{14}$$

Note that

- $L_E(\mathbf{w}, b)$ in (14) cannot be negative, since in E, all $y_m(\mathbf{w}^T x_m + b)$ are negative.
- If $L_E(\mathbf{w}, b)$ becomes zero, we know we have the best possible solution (E is empty).

Applying the gradient descent method, the perceptron learning algorithm becomes

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \alpha \sum_{x_m \in E^{(i)}} y_m \mathbf{x}_m$$
$$b^{(i+1)} = b^{(i)} + \alpha \sum_{x_m \in E^{(i)}} y_m.$$

for $i \geq 0$, where $E^{(i)} = \{\mathbf{x}_m : y_m((\mathbf{w}^{(i)})^T\mathbf{x}_m + b^{(i)}) < 0\}$, and $\mathbf{w}^{(0)}$ and $b^{(0)}$ are arbitrarily chosen.

Alternative learning model: For perceptron learning, an alternative loss function is minimizing the MSE between the desired and actual response of the perceptron:

$$\boxed{L_{\text{MSE}}(\tilde{\mathbf{w}}) = \frac{1}{2}\sum_{m=1}^{M}(y_m - \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_m)^2,} \tag{15}$$

where $y_m \in \{+1, -1\}$ is response of perceptron, $\tilde{\mathbf{w}} \triangleq [\mathbf{w}^T, b]^T$, and $\tilde{\mathbf{x}}_m \triangleq [\mathbf{x}_m^T, 1]^T$, $\forall m$. Note that $L_{\text{MSE}}(\tilde{\mathbf{w}})$ in (15) is always positive.

Using the gradient of L_{MSE},

$$\frac{\partial L_{\text{MSE}}(\tilde{\mathbf{w}})}{\partial \tilde{\mathbf{w}}} = \boxed{??}$$

the corresponding perceptron learning algorithm is

$$\tilde{\mathbf{w}}^{(i+1)} = \tilde{\mathbf{w}}^{(i)} + \alpha \sum_{m=1}^{M} \left(y_m - \left(\tilde{\mathbf{w}}^{(i)}\right)^T\tilde{\mathbf{x}}_m\right)\tilde{\mathbf{x}}_m$$

for $i \geq 0$, where $\tilde{\mathbf{w}}^{(0)}$ is arbitrarily chosen.

One limitation of using gradient descent method in perceptron learning – and more generally, neural network training – is that the gradient is computed from entire training samples for each update. For instance in the MSE-based perceptron learning loss, one needs to calculate $(y_m - (\tilde{\mathbf{w}}^{(i)})^T\tilde{\mathbf{x}}_m)\tilde{\mathbf{x}}_m$ for all $m = 1, \ldots, M$. If M is large, then computing gradient from the entire samples requires too much computation. A popular alternative is **stochastic batch gradient descent**:

- Repeat until an approximate minimum is obtained:
  1) Split randomly shuffled $\{1, \ldots, M\}$ into M/B batches with size B; construct $M^{(i)}$, $i = 1, \ldots, M/B$.
  2) For $i = 1, \ldots, M/B$, do:
$$\mathbf{x} \leftarrow \mathbf{x} - \alpha \sum_{m \in M^{(i+1)}} \nabla f_m(\mathbf{x}).$$

where the empirical loss function is given by $\sum_{m=1}^{M} f_m(\mathbf{x})$ with parameters $\mathbf{x}$ to be optimized. When the learning rates α decrease with an appropriate rate, and subject to relatively mild assumptions, stochastic gradient descent converges *almost surely* to a local minimum.