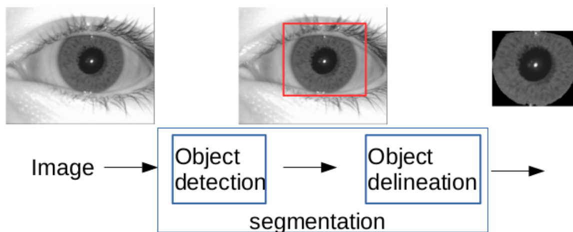
Classification

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

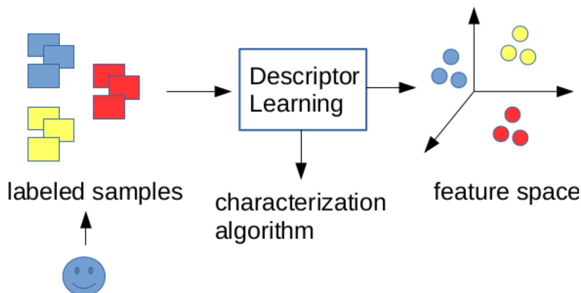
afalcao@ic.unicamp.br

Recall that image analysis requires to learn models for description, detection, delineation, and classification.



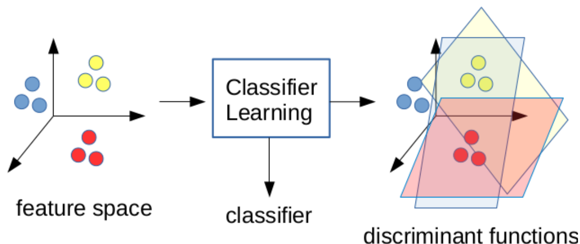
Object (instance) segmentation results from detection and delineation.

We have seen how to learn image description models based on visual dictionaries (with no user annotation) and convolutional layers (with minimal user annotation).



The descriptor aims to create a feature space $\Re^n$ in which images from distinct classes are mapped into separated subspaces of $\Re^n$.

Whenever the classes are linearly separable in $\Re^n$, one can use a single hyperplane per class (e.g., a SVM classifier) to isolate its samples from the others.



Alternatively, a MLP classifier separates them by a collection of hyperplanes per class (i.e., a hyperpolygon).

Among several classifiers,

- Bayesian (parametric) and K-nearest neighbors (non-parametric) [1],
- Optimum-path forest (graph-based) [2, 3, 4],
- Decision trees and random forest [5, 6],
- Support Vector Machines and Multi-Layer Perceptron [5, 6, 7],

we will focus on Multi-Layer Perceptron (MLP), which can learn parameters for description based on convolutional layers and classification, forming a Convolutional Neural Network [8].

Agenda

- Some concepts from Machine Learning.
- The perceptron algorithm.
- The MLP classifier.
- Convolutional Neural Network: construction and use.

Datasets

Let $\mathcal{Z}$ be a dataset – a sample (pixel, superpixel, subimage, image) collection. Each sample $s \in \mathcal{Z}$ is represented by a feature vector $\mathrm{x}(s) \in \Re^n$ and may come from (a) none, (b) one, or (c) multiple classes $\lambda(s) \in \{\omega_k\}_{k=1}^{c}$.

Let $\mathcal{Z}$ be a dataset – a sample (pixel, superpixel, subimage, image) collection. Each sample $s \in \mathcal{Z}$ is represented by a feature vector $\mathrm{x}(s) \in \Re^n$ and may come from (a) none, (b) one, or (c) multiple classes $\lambda(s) \in \{\omega_k\}_{k=1}^{c}$.

- Case (a) defines an open-set problem while the others are closed-set problems.

Datasets

Let $\mathcal{Z}$ be a dataset – a sample (pixel, superpixel, subimage, image) collection. Each sample $s \in \mathcal{Z}$ is represented by a feature vector $\mathrm{x}(s) \in \Re^n$ and may come from (a) none, (b) one, or (c) multiple classes $\lambda(s) \in \{\omega_k\}_{k=1}^c$.

- Case (a) defines an open-set problem while the others are closed-set problems.
- Case (b) defines a single-label problem while (c) defines a multi-label problem.

Datasets

Let $\mathcal{Z}$ be a dataset – a sample (pixel, superpixel, subimage, image) collection. Each sample $s \in \mathcal{Z}$ is represented by a feature vector $\mathrm{x}(s) \in \Re^n$ and may come from (a) none, (b) one, or (c) multiple classes $\lambda(s) \in \{\omega_k\}_{k=1}^{c}$.

- Case (a) defines an open-set problem while the others are closed-set problems.
- Case (b) defines a single-label problem while (c) defines a multi-label problem.

Our focus will be on single-label and closed-set problems.

Classifier: construction and use

- A classifier is built from a training set $\mathcal{Z}_{tr} \subset \mathcal{Z}$.

Classifier: construction and use

- A classifier is built from a training set $\mathcal{Z}_{tr} \subset \mathcal{Z}$.
- The training process may use an auxiliary set $\mathcal{Z}_{vl} \subset \mathcal{Z}$, $\mathcal{Z}_{tr} \cap \mathcal{Z}_{vl} = \emptyset$, named validation set, to optimize the model's hyperparameters (e.g., a network architecture).

Classifier: construction and use

- A classifier is built from a training set $\mathcal{Z}_{tr} \subset \mathcal{Z}$.
- The training process may use an auxiliary set $\mathcal{Z}_{vl} \subset \mathcal{Z}$, $\mathcal{Z}_{tr} \cap \mathcal{Z}_{vl} = \emptyset$, named validation set, to optimize the model's hyperparameters (e.g., a network architecture).
- The final model is then tested on a testing set $\mathcal{Z}_{ts} \subset \mathcal{Z}$, $\mathcal{Z}_{tr} \cap \mathcal{Z}_{vl} \cap \mathcal{Z}_{ts} = \emptyset$.

Classifier: construction and use

- A classifier is built from a training set $\mathcal{Z}_{tr} \subset \mathcal{Z}$.
- The training process may use an auxiliary set $\mathcal{Z}_{vl} \subset \mathcal{Z}$, $\mathcal{Z}_{tr} \cap \mathcal{Z}_{vl} = \emptyset$, named validation set, to optimize the model's hyperparameters (e.g., a network architecture).
- The final model is then tested on a testing set $\mathcal{Z}_{ts} \subset \mathcal{Z}$, $\mathcal{Z}_{tr} \cap \mathcal{Z}_{vl} \cap \mathcal{Z}_{ts} = \emptyset$.
- The process must be repeated multiple times with random splits of $\mathcal{Z}$ into $\mathcal{Z}_{tr}, \mathcal{Z}_{vl}$ and $\mathcal{Z}_{ts}$ to allow statistical analysis.

- When the true labels $\lambda(s)$ are known for all $s \in \mathcal{Z}_{tr}$, the problem is said supervised.

- When the true labels $\lambda(s)$ are known for all $s \in \mathcal{Z}_{tr}$, the problem is said supervised.

- It is semi-supervised when the true labels are known for a subset of $\mathcal{Z}_{tr}$ and unsupervised when the true labels are unknown for all samples in $\mathcal{Z}_{tr}$.

Classifier: construction and use

- When the true labels $\lambda(s)$ are known for all $s \in \mathcal{Z}_{tr}$, the problem is said supervised.
- It is semi-supervised when the true labels are known for a subset of $\mathcal{Z}_{tr}$ and unsupervised when the true labels are unknown for all samples in $\mathcal{Z}_{tr}$.
- In any case, the descriptor maps $\mathcal{Z} \rightarrow \Re^n$ and the classifier maps $\Re^n \rightarrow \{\omega_k\}_{k=1}^{c}$ such that an error occurs when the resulting label $L(s) \in \{\omega_k\}_{k=1}^{c}$ is different from $\lambda(s)$, $s \in \mathcal{Z}$.

Sample selection

Random samples are selected from $\mathcal{Z}$ to compose the training, validation, and testing sets.

Sample selection

Random samples are selected from $\mathcal{Z}$ to compose the training, validation, and testing sets.

- When the true labels of $s \in \mathcal{Z}$ are known *a priori*, if we force a same number of samples per class, the resulting sets will be balanced, but this is not usually the real scenario.

Sample selection

Random samples are selected from $\mathcal{Z}$ to compose the training, validation, and testing sets.

- When the true labels of $s \in \mathcal{Z}$ are known *a priori*, if we force a same number of samples per class, the resulting sets will be balanced, but this is not usually the real scenario.
- Alternatively, a same percentage of samples (stratified sampling) per class creates imbalanced sets whenever $\mathcal{Z}$ is imbalanced.

- Given that $\mathrm{x}(s) = (\mathrm{x}_1(s), \mathrm{x}_2(s), \ldots, \mathrm{x}_n(s)) \in \Re^n$ changes with the random choice of $s \in \mathcal{Z}$, then x is said a random field with probability density function $\rho(\mathrm{x}) \colon \Re^n \rightarrow [0, 1]$ (a manifold in $\Re^{n+1}$).

Sample selection

- Given that $\mathrm{x}(s) = (\mathrm{x}_1(s), \mathrm{x}_2(s), \ldots, \mathrm{x}_n(s)) \in \Re^n$ changes with the random choice of $s \in \mathcal{Z}$, then x is said a random field with probability density function $\rho(\mathrm{x})\colon \Re^n \rightarrow [0, 1]$ (a manifold in $\Re^{n+1}$).

- Likewise, each feature $\mathrm{x}_i(s) \in \Re$, $i \in [1, n]$, changes with the random choice of $s \in \mathcal{Z}$, then x_i is said a random variable.

Sample selection

- Given that $x(s) = (x_1(s), x_2(s), \ldots, x_n(s)) \in \Re^n$ changes with the random choice of $s \in \mathcal{Z}$, then x is said a random field with probability density function $\rho(x)\colon \Re^n \rightarrow [0, 1]$ (a manifold in $\Re^{n+1}$).
- Likewise, each feature $x_i(s) \in \Re$, $i \in [1, n]$, changes with the random choice of $s \in \mathcal{Z}$, then x_i is said a random variable.
- A standard approach is cross validation and the methods can be described for training and validation/testing sets as follows.

Cross validation

Cross validation may be called K-hold-out, K-fold, or $N \times K$-fold [5].

Cross validation

Cross validation may be called K-hold-out, K-fold, or $N \times K$-fold [5].

- K-hold-out: $\mathcal{Z}$ is split K times into $P\%$ of samples for $\mathcal{Z}_{tr}$ and $(100 - P)\%$ for $\mathcal{Z}_{ts}$, $0 < P < 100$. The instances of $\mathcal{Z}_{tr}$ and $\mathcal{Z}_{ts}$ are not statistically independent.

Cross validation

Cross validation may be called K-hold-out, K-fold, or $N \times K$-fold [5].

- K-hold-out: $\mathcal{Z}$ is split K times into $P\%$ of samples for $\mathcal{Z}_{tr}$ and $(100 - P)\%$ for $\mathcal{Z}_{ts}$, $0 < P < 100$. The instances of $\mathcal{Z}_{tr}$ and $\mathcal{Z}_{ts}$ are not statistically independent.

- K-fold: $\mathcal{Z}$ is split into K parts of approximately equal sizes, using each of the parts for testing and the rest for training K times. The instances of $\mathcal{Z}_{ts}$ are statistically independent, but not the instances of $\mathcal{Z}_{tr}$.

Cross validation

Cross validation may be called K-hold-out, K-fold, or $N \times K$-fold [5].

- K-hold-out: $\mathcal{Z}$ is split K times into P% of samples for $\mathcal{Z}_{tr}$ and $(100 - P)$% for $\mathcal{Z}_{ts}$, $0 < P < 100$. The instances of $\mathcal{Z}_{tr}$ and $\mathcal{Z}_{ts}$ are not statistically independent.
- K-fold: $\mathcal{Z}$ is split into K parts of approximately equal sizes, using each of the parts for testing and the rest for training K times. The instances of $\mathcal{Z}_{ts}$ are statistically independent, but not the instances of $\mathcal{Z}_{tr}$.
- $N \times K$-fold: K-fold is repeated N times, usually with $K = 2$.

Effectiveness and confusion matrix

Let n_{ij} be the number of times test samples from class ω_i have been classified into class ω_j for $i, j \in [1, c]$ and m_{ts} samples. A confusion matrix is defined as

$$\left[\begin{array}{c|cccc} & \omega_1 & \omega_2 & \ldots & \omega_c \\ \hline \omega_1 & n_{11} & n_{12} & \ldots & n_{1c} \\ \omega_2 & n_{21} & n_{22} & \ldots & n_{2c} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega_c & n_{c1} & n_{c2} & \ldots & n_{cc} \end{array}\right]$$

Effectiveness and confusion matrix

Let n_{ij} be the number of times test samples from class ω_i have been classified into class ω_j for $i, j \in [1, c]$ and m_{ts} samples. A confusion matrix is defined as

$$\left[\begin{array}{c|cccc} & \omega_1 & \omega_2 & \ldots & \omega_c \\ \hline \omega_1 & n_{11} & n_{12} & \ldots & n_{1c} \\ \omega_2 & n_{21} & n_{22} & \ldots & n_{2c} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega_c & n_{c1} & n_{c2} & \ldots & n_{cc} \end{array}\right]$$

- The total of correct classifications is $\sum_{i=1}^{c} n_{ii}$, being $m_{ts} - \sum_{i=1}^{c} n_{ii}$ the total of misclassifications.

Effectiveness and confusion matrix

Let n_{ij} be the number of times test samples from class ω_i have been classified into class ω_j for $i, j \in [1, c]$ and m_{ts} samples. A confusion matrix is defined as

$$\left[\begin{array}{c|cccc} & \omega_1 & \omega_2 & \dots & \omega_c \\ \hline \omega_1 & n_{11} & n_{12} & \dots & n_{1c} \\ \omega_2 & n_{21} & n_{22} & \dots & n_{2c} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega_c & n_{c1} & n_{c2} & \dots & n_{cc} \end{array}\right]$$

- The total of correct classifications is $\sum_{i=1}^{c} n_{ii}$, being $m_{ts} - \sum_{i=1}^{c} n_{ii}$ the total of misclassifications.
- Several effectiveness measures can be obtained from the confusion matrix (sensitivity, accuracy, specificity, precision, etc). A "good" one is the Cohen's kappa, which is robust to imbalanced classes.

Cohen's kappa

Cohen's kappa κ measures the observed P_o and expected-by-chance P_e agreements between two raters, A (rows) and B (columns) in a confusion matrix.

$$
\begin{aligned}
\kappa &= \frac{P_o - P_e}{1 - P_e}, \\
P_o &= \frac{1}{m_{ts}} \sum_{i=1}^{c} n_{ii}, \\
P_e &= \frac{1}{m_{ts}^2} \sum_{i=1}^{c} N_A(i) N_B(i),
\end{aligned}
$$

where $N_A(i) = \sum_{j=1}^{c} n_{ij}$ and $N_B(i) = \sum_{j=1}^{c} n_{ji}$ are the total of samples raters A and B assign to class ω_i, respectively.

Statistical tests

- Statistical tests provide a formal way to decide if the results of an experiment are significant or accidental [9].

Statistical tests

- Statistical tests provide a formal way to decide if the results of an experiment are significant or accidental [9].

- For example, one can measure the Cohen's kappa $\kappa_i(t)$ of each execution $t = 1, 2, \ldots, T$ of each classifier C_i, $i \in [1, n]$, on T statistically independent sets during cross validation.

Statistical tests

- Statistical tests provide a formal way to decide if the results of an experiment are significant or accidental [9].
- For example, one can measure the Cohen's kappa $\kappa_i(t)$ of each execution $t = 1, 2, \ldots, T$ of each classifier C_i, $i \in [1, n]$, on T statistically independent sets during cross validation.
- A statistical test starts from a null hypothesis, such as all classifiers are equivalent, and verify if it can be rejected at some significance level p (e.g., $p = 0.05$).

Statistical tests

- First, some measure m_o, that indicates differences among the classifiers, is obtained from the experiment. For example, for $n = 2$ classifiers and a 5×2-fold cross validation, one can compute the variances s_t^2 of the differences $\kappa_1(t) - \kappa_2(t)$ of the two folds for $t = 1, 2, \ldots, 5$ and define

$$m_o = \frac{\kappa_1(1) - \kappa_2(1)}{\sqrt{\frac{1}{5}\sum_{t=1}^{5} s_t^2}}$$

Statistical tests

- First, some measure m_o, that indicates differences among the classifiers, is obtained from the experiment. For example, for $n = 2$ classifiers and a 5×2-fold cross validation, one can compute the variances s_t^2 of the differences $\kappa_1(t) - \kappa_2(t)$ of the two folds for $t = 1, 2, \ldots, 5$ and define

$$m_o = \frac{\kappa_1(1) - \kappa_2(1)}{\sqrt{\frac{1}{5}\sum_{t=1}^{5} s_t^2}}$$

- It is shown that m_o (a random variable) satisfies some probability density function $\rho(m_o)$ when the null hypothesis is satisfied. For the example, a t-distribution of five degrees of freedom.

Statistical tests

- The areas below the curve $\rho(m_o)$ are tabulated for each value of m_o, representing the chances p of the null hypothesis be correct.

Statistical tests

- The areas below the curve $\rho(m_o)$ are tabulated for each value of m_o, representing the chances p of the null hypothesis be correct.
- If m_o is observed above a critical value such that $p < 0.05$, for instance, we reject the null hypothesis with less than 5% of chance of being wrong.

Statistical tests

- The areas below the curve $\rho(m_o)$ are tabulated for each value of m_o, representing the chances p of the null hypothesis be correct.
- If m_o is observed above a critical value such that $p < 0.05$, for instance, we reject the null hypothesis with less than 5% of chance of being wrong.
- The most popular tests are student's t-test, Wilcoxon signed-rank test, analysis of variance (ANOVA), Tukey's range test, Nemenyi test, and Friedman test.

The perceptron algorithm

- From a set of discriminant functions $\{g_k(x)\}_{k=1}^{c}$, a classifier can be defined by the selection of $\omega_j \in \{\omega_k\}_{k=1}^{c}$ whose $g_j(x) = \max_{k=1,2,\ldots c}\{g_k(x)\}$.

The perceptron algorithm

- From a set of discriminant functions $\{g_k(\mathrm{x})\}_{k=1}^{c}$, a classifier can be defined by the selection of $\omega_j \in \{\omega_k\}_{k=1}^{c}$ whose $g_j(\mathrm{x}) = \max_{k=1,2,\ldots c}\{g_k(\mathrm{x})\}$.

- Classical approaches estimate the posterior probability $g_k(\mathrm{x}) = P(\omega_k \backslash \mathrm{x})$ based on the Bayes Theorem.

$$P(\omega_k \backslash \mathrm{x}) = \frac{P(\omega_k)\rho(\mathrm{x} \backslash \omega_k)}{\rho(\mathrm{x})},$$

  where $P(\omega_k)$ is the prior probability, the conditional density function $\rho(\mathrm{x} \backslash \omega_k)$ is the likelihood, and $\rho(\mathrm{x})$ is the evidence.

The perceptron algorithm

- From a set of discriminant functions $\{g_k(\mathrm{x})\}_{k=1}^{c}$, a classifier can be defined by the selection of $\omega_j \in \{\omega_k\}_{k=1}^{c}$ whose $g_j(\mathrm{x}) = \max_{k=1,2,\ldots c}\{g_k(\mathrm{x})\}$.

- Classical approaches estimate the posterior probability $g_k(\mathrm{x}) = P(\omega_k \backslash \mathrm{x})$ based on the Bayes Theorem.

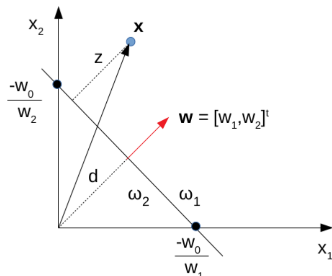
$$P(\omega_k \backslash \mathrm{x}) \quad = \quad \frac{P(\omega_k)\rho(\mathrm{x} \backslash \omega_k)}{\rho(\mathrm{x})},$$

  where $P(\omega_k)$ is the prior probability, the conditional density function $\rho(\mathrm{x} \backslash \omega_k)$ is the likelihood, and $\rho(\mathrm{x})$ is the evidence.

- We will focus on one linear discriminat function per class: $g_k(\mathrm{x}) = \langle \mathrm{w}_k, \mathrm{x} \rangle + w_{k0}$, where $\mathrm{w}_k \in \Re^n$ is a weight vector normal to the hyperplane that separates ω_k from other classes and w_{k0} is the bias.

The perceptron algorithm

For example, a simplification for two classes ($c = 2$) in $\Re^2$ may adopt a single hyperplane with linear discriminant function $g(\mathrm{x}) = \langle \mathrm{w}, \mathrm{x} \rangle + w_0$, such that $g(\mathrm{x}) > 0$ leads to ω_1 and $g(\mathrm{x}) < 0$ leads to ω_2.



$$
\begin{aligned}
d &= \frac{|w_0|}{\|\mathrm{w}\|} \\
z &= \frac{|g(\mathrm{x})|}{\|\mathrm{w}\|}
\end{aligned}
$$

The perceptron algorithm

Let $\mathrm{w}' = [w_0, \mathrm{w}]^t$ and $\mathrm{x}' = [1, \mathrm{x}]^t$, the optimum extended weight vector w^* can be found from $\mathrm{x}'(s)$ of training samples $s \in \mathcal{Z}_{tr}$ based on the minimization of the criterion function

$$J(\mathrm{w}') = \sum_{\mathrm{x}'(s), s \in \mathcal{E}} \delta_s * \langle \mathrm{w}', \mathrm{x}'(s) \rangle,$$

where $\mathcal{E} \subset \mathcal{Z}_{tr}$ contains misclassified samples and δ_s is defined as

$$\delta_s = \begin{cases} -1 & \text{if } \lambda(s) = \omega_1, \\ +1 & \text{if } \lambda(s) = \omega_2. \end{cases}$$

The perceptron algorithm

Let $\mathrm{w}' = [w_0, \mathrm{w}]^t$ and $\mathrm{x}' = [1, \mathrm{x}]^t$, the optimum extended weight vector w^* can be found from $\mathrm{x}'(s)$ of training samples $s \in \mathcal{Z}_{tr}$ based on the minimization of the criterion function

$$J(\mathrm{w}') = \sum_{\mathrm{x}'(s), s \in \mathcal{E}} \delta_s * \langle \mathrm{w}', \mathrm{x}'(s) \rangle,$$

where $\mathcal{E} \subset \mathcal{Z}_{tr}$ contains misclassified samples and δ_s is defined as

$$\delta_s = \begin{cases} -1 & \text{if } \lambda(s) = \omega_1, \\ +1 & \text{if } \lambda(s) = \omega_2. \end{cases}$$

Note that, $J(\mathrm{w}') \geq 0$ and the weight vectors can be updated along with iterations i by

$$\mathrm{w}'(i+1) = \mathrm{w}'(i) - \mu(i) \frac{\partial J(\mathrm{w}')}{\partial \mathrm{w}'} \Big|_{\mathrm{w}' = \mathrm{w}'(i)},$$

where $\mu(i) \in \Re^+$ is a variable learning rate.

The perceptron algorithm

- For linearly separable classes, the perceptron algorithm converges and the choice of $\mu(i)$ controls the speed of convergence.

The perceptron algorithm

- For linearly separable classes, the perceptron algorithm converges and the choice of $\mu(i)$ controls the speed of convergence.

- For instance, one may select $\mu(i) = \frac{c}{i}$, for $i > 0$, and $0 < \mu(0) = c$.

The perceptron algorithm

- For linearly separable classes, the perceptron algorithm converges and the choice of $\mu(i)$ controls the speed of convergence.
- For instance, one may select $\mu(i) = \frac{c}{i}$, for $i > 0$, and $0 < \mu(0) = c$.
- The partial derivative $\frac{\partial J(\mathrm{w}')}{\partial \mathrm{w}'} = \sum_{\mathrm{x}'(s), s \in \mathcal{E}} \delta_s \mathrm{x}'(s)$, then

$$\mathrm{w}'(i+1) = \mathrm{w}'(i) - \mu(i) \sum_{\mathrm{x}'(s), s \in \mathcal{E}} \delta_s \mathrm{x}'(s).$$

The perceptron algorithm

- For linearly separable classes, the perceptron algorithm converges and the choice of $\mu(i)$ controls the speed of convergence.
- For instance, one may select $\mu(i) = \frac{c}{i}$, for $i > 0$, and $0 < \mu(0) = c$.
- The partial derivative $\frac{\partial J(\mathrm{w}')}{\partial \mathrm{w}'} = \sum_{\mathrm{x}'(s), s \in \mathcal{E}} \delta_s \mathrm{x}'(s)$, then

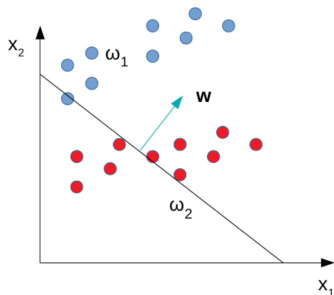
$$\mathrm{w}'(i+1) = \mathrm{w}'(i) - \mu(i) \sum_{\mathrm{x}'(s), s \in \mathcal{E}} \delta_s \mathrm{x}'(s).$$

Assuming linearly separable classes, the perceptron algorithm can be presented as follows.

The perceptron algorithm

1. Choose w' randomly and set $\mu \leftarrow c$, $i \leftarrow 0$, and $\mathcal{E} \leftarrow \emptyset$.
2. Repeat
3. For each $s \in \mathcal{Z}tr$ do
4. If $\delta_s \langle \mathrm{w}', \mathrm{x}'(s) \rangle \geq 0$ then $\mathcal{E} \leftarrow \mathcal{E} \cup \{s\}$.
5. Set $\mathrm{w}' \leftarrow \mathrm{w}' - \mu \sum_{\mathrm{x}'(s), s \in \mathcal{E}} \delta_s \mathrm{x}'(s)$.
6. Update $i \leftarrow i + 1$ and $\mu \leftarrow \frac{\mu}{i}$.
7. Until $\mathcal{E} = \emptyset$.

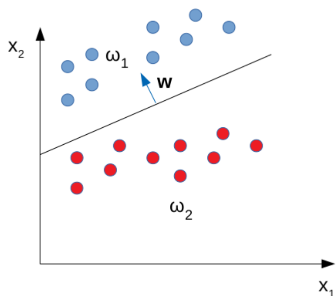
The subsequent adjustments of the weight vector should move the hyperplane as shown.



For nonlinearly separable classes, it is known that this strategy requires two hidden layers to separate classes by hyperpolyhedrons.

The perceptron algorithm

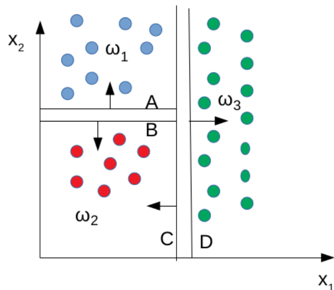
The subsequent adjustments of the weight vector should move the hyperplane as shown.



For nonlinearly separable classes, it is known that this strategy requires two hidden layers to separate classes by hyperpolyhedrons.

Multi-layer perceptron

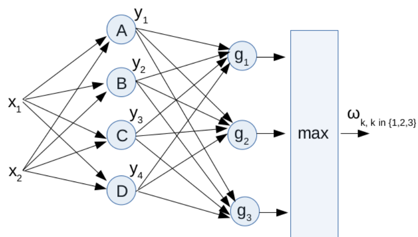
For $c > 2$ classes, whenever the classes are nonlinearly separable, one hidden layer of perceptrons, that activate only for samples $s \in \mathcal{Z}$ whose $x(s)$ is on their positive side, may be enough. However, for a reduced number of perceptrons per layer, more hidden layers are needed.



The perceptrons for class ω_k should define the surfaces of the hyperpolyhedron that separates samples of ω_k from the others.

Multi-layer perceptron

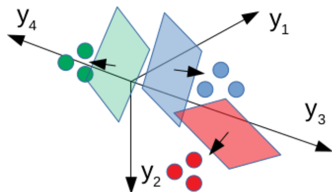
The hidden layer of perceptrons (e.g., A-D) creates a feature space of activations (e.g., y_i, $i = 1, 2, \ldots, 4$) that is higher and sparser than the original space (e.g., it went from $\Re^2$ to $\Re^4$).



Samples of distinct classes are expected to be mapped into different subspaces, such that the decision layer of discriminant functions $\{g_k(\mathrm{x})\}_{k=1}^{c}$ can solve classification by selecting $\omega_j \in \{\omega_k\}_{k=1}^{c}$ whose $g_j(\mathrm{x}) = \max_{k=1,2,\ldots c}\{g_k(\mathrm{x})\}$.

Multi-layer perceptron

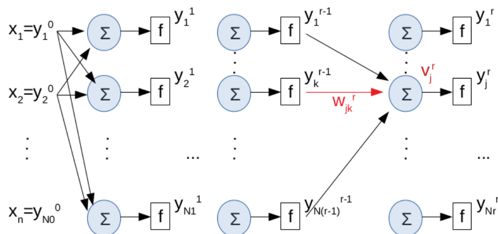
The hidden layer of perceptrons (e.g., A-D) creates a feature space of activations (e.g., y_i, $i = 1, 2, \ldots, 4$) that is higher and sparser than the original space (e.g., it went from $\Re^2$ to $\Re^4$).



Samples of distinct classes are expected to be mapped into different subspaces, such that the decision layer of discriminant functions $\{g_k(\mathrm{x})\}_{k=1}^{c}$ can solve classification by selecting $\omega_j \in \{\omega_k\}_{k=1}^{c}$ whose $g_j(\mathrm{x}) = \max_{k=1,2,\ldots c}\{g_k(\mathrm{x})\}$.

Multi-layer perceptron

Let $\mathbf{w}_j^r = \left[w_{j0}^r, w_{j1}^r, w_{j2}^r, \ldots, w_{jk}^r, \ldots, w_{jN_{(r-1)}}^r \right]^t$ be the weight vector (including bias w_{j0}^r) of a perceptron j at a layer $r \in [1, L]$ of a multi-layer perceptron with L layers, such that w_{jk}^r is the synaptic weight of the connection between perceptron j and a perceptron k from layer $r-1$.



Layer 0 is the input layer that presents $[1, \mathbf{x}]^t$ to the perceptrons of layer 1, $v_j^r = \langle \mathbf{y}^{r-1}, \mathbf{w}_j^r \rangle$, and layer L is the decision layer with $N_L = c$ perceptrons, one per class.

So far, we have considered the ReLU activation f (the McCulloch-Pitts neuron).

$$f(v) = \begin{cases} v & v > 0, \\ 0 & v \leq 0. \end{cases}$$

Multi-layer perceptron

So far, we have considered the ReLU activation f (the McCulloch-Pitts neuron).

$$f(v) = \begin{cases} v & v > 0, \\ 0 & v \leq 0. \end{cases}$$

Other options are continuous differentiable functions (e.g., the family of sigmoid functions and hyperbolic tangent functions). A common example is the logistic function.

$$f(v) = \frac{v}{1 + \exp(-av)},$$

where $a > 0$ is a slope parameter.

Multi-layer perceptron

For w_j^r, each iteration i adjusts its weights by

$$\begin{aligned}
\mathrm{w}_j^r(i+1) &= \mathrm{w}_j^r(i) + \Delta \mathrm{w}_j^r, \\
\Delta \mathrm{w}_j^r &= -\mu \frac{\partial J}{\partial \mathrm{w}_j^r}, \\
J &= \sum_{s \in \mathcal{Z}_{tr}} \mathcal{E}(s)
\end{aligned}$$

for a fixed learning rate μ and error function $\mathcal{E}$.

Multi-layer perceptron

For w_j^r, each iteration i adjusts its weights by

$$
\begin{aligned}
w_j^r(i+1) &= w_j^r(i) + \Delta w_j^r, \\
\Delta w_j^r &= -\mu \frac{\partial J}{\partial w_j^r}, \\
J &= \sum_{s \in \mathcal{Z}_{tr}} \mathcal{E}(s)
\end{aligned}
$$

for a fixed learning rate μ and error function $\mathcal{E}$.

Given the pairs $(x(s), y(s))$, $s \in \mathcal{Z}_{tr}$, with the input and expected output vectors, one can choose $\mathcal{E}(s)$ as

$$
\mathcal{E}(s) = \frac{1}{2}\|y^L(s) - y(s)\|^2 = \frac{1}{2}\sum_{m=1}^{c}(y_m^L(s) - y_m(s))^2 = \frac{1}{2}\sum_{m=1}^{c} e_m^2(s),
$$

where $y^L(s)$ is the estimated output vector.

Multi-layer perceptron

For $\Delta \mathrm{w}_j^r$, we must compute $\frac{\partial J}{\partial \mathrm{w}_j^r} = \sum_{s \in \mathcal{Z}_{tr}} \frac{\partial \mathcal{E}(s)}{\partial \mathrm{w}_j^r}$. By the chain rule,

$$\frac{\partial \mathcal{E}(s)}{\partial \mathrm{w}_j^r} = \frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} \frac{\partial v_j^r(s)}{\partial \mathrm{w}_j^r}.$$

For $\Delta \mathrm{w}_j^r$, we must compute $\frac{\partial J}{\partial \mathrm{w}_j^r} = \sum_{s \in \mathcal{Z}_{tr}} \frac{\partial \mathcal{E}(s)}{\partial \mathrm{w}_j^r}$. By the chain rule,

$$\frac{\partial \mathcal{E}(s)}{\partial \mathrm{w}_j^r} = \frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} \frac{\partial v_j^r(s)}{\partial \mathrm{w}_j^r}.$$

Given that $v_j^r(s) = \sum_{m=0}^{N_{r-1}} w_{jm}^r y_m^{r-1}(s) = \langle \mathrm{w}_j^r, \mathrm{y}^{r-1}(s) \rangle$,

$$\frac{\partial v_j^r(s)}{\partial \mathrm{w}_j^r} = \mathrm{y}^{r-1}(s).$$

Multi-layer perceptron

For $\Delta \mathrm{w}_j^r$, we must compute $\frac{\partial J}{\partial \mathrm{w}_j^r} = \sum_{s \in \mathcal{Z}_{tr}} \frac{\partial \mathcal{E}(s)}{\partial \mathrm{w}_j^r}$. By the chain rule,

$$\frac{\partial \mathcal{E}(s)}{\partial \mathrm{w}_j^r} = \frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} \frac{\partial v_j^r(s)}{\partial \mathrm{w}_j^r}.$$

Given that $v_j^r(s) = \sum_{m=0}^{N_{r-1}} w_{jm}^r y_m^{r-1}(s) = \langle \mathrm{w}_j^r, \mathrm{y}^{r-1}(s) \rangle$,

$$\frac{\partial v_j^r(s)}{\partial \mathrm{w}_j^r} = \mathrm{y}^{r-1}(s).$$

Let us now define $\frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} = \delta_j^r(s)$, such that

$$\Delta \mathrm{w}_j^r = -\mu \sum_{s \in \mathcal{Z}_{tr}} \delta_j^r(s) \mathrm{y}^{r-1}(s).$$

The computation of $\delta_j^r(s)$ starts from $r = L$ and propagates backward for $1 \leq r < L$, deriving the name backpropagation algorithm.

Multi-layer perceptron

The computation of $\delta_j^r(s)$ starts from $r = L$ and propagates backward for $1 \leq r < L$, deriving the name backpropagation algorithm.

For $r = L$ and $1 \leq j \leq c$,

$$
\begin{aligned}
\delta_j^L(s) &= \frac{\partial \mathcal{E}(s)}{\partial v_j^L(s)} = \frac{\partial \left(\frac{1}{2} \sum_{m=1}^{c} \left(f(v_m^L(s)) - y_m(s) \right)^2 \right)}{\partial v_j^L(s)} \\
\delta_j^L(s) &= \left(f(v_j^L(s)) - y_j(s) \right) \frac{\partial f(v_j^L(s))}{\partial v_j^L(s)} = e_j(s) f'(v_j^L(s)) \\
\delta_j^L(s) &= e_j(s) f'(v_j^L(s)).
\end{aligned}
$$

Multi-layer perceptron

For $r < L$ and $1 \leq j \leq N_{r-1}$, $v_j^{r-1}(s)$ affects all $v_k^r(s)$, $k = 1, 2, \ldots, N_r$. Therefore, the chain rule must be applied.

$$\delta_j^{r-1}(s) = \sum_{k=1}^{N_r} \frac{\partial \mathcal{E}(s)}{\partial v_k^r(s)} \frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)} = \sum_{k=1}^{N_r} \delta_k^r(s) \frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)}$$

$$\frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)} = \frac{\partial \left(\sum_{m=0}^{N_{r-1}} w_{km}^r y_m^{r-1}(s) \right)}{\partial v_j^{r-1}(s)} = \frac{\partial \left(\sum_{m=0}^{N_{r-1}} w_{km}^r f(v_m^{r-1}(s)) \right)}{\partial v_j^{r-1}(s)}$$

$$\frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)} = w_{kj}^r \frac{\partial f(v_j^{r-1}(s))}{\partial v_j^{r-1}(s)} = w_{kj}^r f'(v_j^{r-1}(s))$$

$$\delta_j^{r-1}(s) = \left(\sum_{k=1}^{N_r} \delta_k^r(s) w_{kj}^r \right) f'(v_j^{r-1}(s))$$

Multi-layer perceptron

In summary,

$$
\begin{aligned}
\mathrm{w}_j^r(i+1) &= \mathrm{w}_j^r(i) + \Delta \mathrm{w}_j^r, \\
\Delta \mathrm{w}_j^r &= -\mu \sum_{s \in \mathcal{Z}_{tr}} \delta_j^r(s) \mathrm{y}^{r-1}(s) \\
\delta_j^r(s) &= \begin{cases} (f(v_j^r(s)) - y_j^r) f'(v_j^r(s)) & r = L \\ \left(\sum_{k=1}^{N_{r+1}} \delta_k^{r+1}(s) w_{kj}^{r+1} \right) f'(v_j^r(s)) & r < L \end{cases}
\end{aligned}
$$

For the logistic function,

$$
f'(v_j^r(s)) = af(v_j^r(s))(1 - f(v_j^r(s)))
$$

and for ReLU,

$$
f'(v_j^r(s)) = \begin{cases} 1 & v_j^r(s) > 0, \\ 0 & \text{otherwise.} \end{cases}
$$

Backpropagation algorithm

Start from $(\mathrm{x}(s), \mathrm{y}(s))$, $s \in \mathcal{Z}_{tr}$, a given network architecture with random weight initialization, learning rate μ, maximum number $T > 0$ of iterations (epochs), and minimum error $\epsilon > 0$.

01. Set $i \leftarrow 1$.
02. Do
03. Set $\mathcal{E} \leftarrow 0$.
04. For each $s \in \mathcal{Z}_{tr}$ do
05. For $r = 1$ to L and $j = 1$ to N_r do
06. Compute $v_j^r(s)$ and $y_j^r(s) = f(v_j^r(s))$.
07. For $j = 1$ to c do
08. Set $\mathcal{E} \leftarrow \mathcal{E} + \frac{1}{2}(y_j^L(s) - y_j(s))^2$
09. For $r = 1$ to L and $j = 1$ to N_r do
10. Set $\Delta \mathrm{w}_j^r \leftarrow 0$.

Backpropagation algorithm

11. For each $s \in \mathcal{Z}_{tr}$ do
12. For $r = L$ to 1 and $j = 1$ to N_r do
13. Compute $\delta_j^r(s)$ and $\Delta \mathrm{w}_j^r \leftarrow \Delta \mathrm{w}_j^r - \mu \delta_j^r(s) \mathrm{y}^{r-1}(s)$.
14. For $r = 1$ to L and $j = 1$ to N_r do
15. Set $\mathrm{w}_j^r \leftarrow \mathrm{w}_j^r + \Delta \mathrm{w}_j^r$.
16. Set $i \leftarrow i + 1$.
17. While $\mathcal{E} > \epsilon$ and $i \leq T$.

This algorithm is also known as Stochastic Gradient Descendant.

Backpropagation algorithm

- The choice of μ is application-dependent and is crucial to speed-up convergence. Typically, $0.01 \leq \mu \leq 0.6$. One can also update (reduce) μ at every number X of epochs.

Backpropagation algorithm

- The choice of μ is application-dependent and is crucial to speed-up convergence. Typically, $0.01 \leq \mu \leq 0.6$. One can also update (reduce) μ at every number X of epochs.

- A momentum α, typically in $[0.1, 0.8]$, can also be used to reduce oscillation in the criterion function and speed up convergence.

$$
\begin{aligned}
\Delta \mathrm{w}_j^r(i) &= \alpha \Delta \mathrm{w}_j^r(i-1) - \mu \sum_{s \in \mathcal{Z}_{tr}} \delta_j^r(s) \mathrm{y}^{r-1}(s) \\
\mathrm{w}_j^r(i+1) &= \alpha \mathrm{w}_j^r(i) + \Delta \mathrm{w}_j^r(i)
\end{aligned}
$$

Backpropagation algorithm

Cross-entropy is another commonly used criterion function J.

$$J = -\sum_{s\in\mathcal{Z}_{tr}}\sum_{m=1}^{c}\left(y_m(s)\ln y_m^L(s) + (1-y_m(s))\ln(1-y_m^L(s))\right),$$

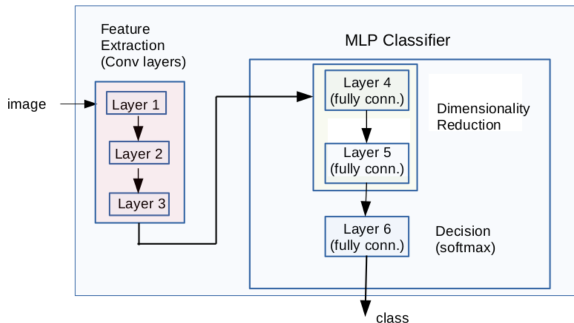
where $y_m^L(s)$ and $y_m(s)$ should be in $[0,1]$. This is usually done by using softmax activation in the decision layer L.

$$y_j^L(s) = \frac{\exp\left(f(v_j^L)\right)}{\sum_{m=1}^{c}\exp\left(f(v_m^L)\right)},$$

$j = 1, 2, \ldots, c$.

Convolutional Neural Networks

The backpropagation algorithm can estimate the weights of the MLP classifier as well as the weights of the convolutional layers.



However, $\delta_j^r(s)$, $j = 1, 2, \ldots, N_r$, tend to zero as $r \rightarrow 1$ and L increases (the vanishing-gradient problem), making it difficult to update the weights of the initial layers.

- Different initialization techniques and activation functions have been used to address the vanishing-gradient problem.

- Different initialization techniques and activation functions have been used to address the vanishing-gradient problem.
- Another problem is overfitting, to which weight dropout and data augmentation have been used as regularization techniques.

- Different initialization techniques and activation functions have been used to address the vanishing-gradient problem.
- Another problem is overfitting, to which weight dropout and data augmentation have been used as regularization techniques.
- For convolutional layers, each pixel j of an image s is a neuron with output $y_j^r(s)$ at a layer r and receptive field defined by the values $y_m^{r-1}(s)$, $m \in \mathcal{A}(j)$, of its adjacent pixels in layer $r-1$. Therefore $\sum_{m=1}^{N_r}$ becomes $\sum_{m \in \mathcal{A}(j)}$.

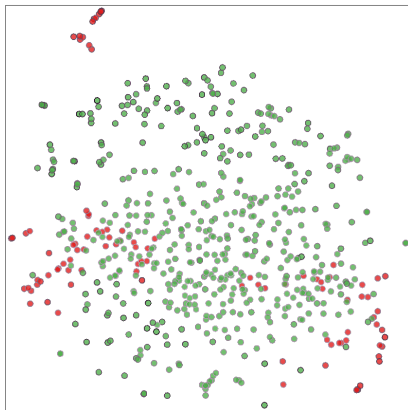
A max-pooling g after activation f implies to substitute $f'(v_j^r(s))$ by $\frac{\partial g(f(v_j^r(s)))}{\partial v_j^r(s)} = \frac{\partial g(f(v_j^r(s)))}{\partial f(v_j^r(s))}\frac{\partial f(v_j^r(s))}{\partial v_j^r(s)} = g'(f(v_j^r(s)))f'(v_j^r(s))$. Then $g(f(v_j^r(s))) = \max_{m\in\mathcal{A}(j)}\{f(v_m^r(s))\}$ can be rewritten as

$$\begin{aligned} g(f(v_j^r(s))) &= \sum_{m\in\mathcal{A}(j)} u_m^r f(v_m^r(s)), \\ u_m^r &= \begin{cases} 1 & k = argmax_{m\in\mathcal{A}(j)}\{f(v_m^r(s))\}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Therefore,

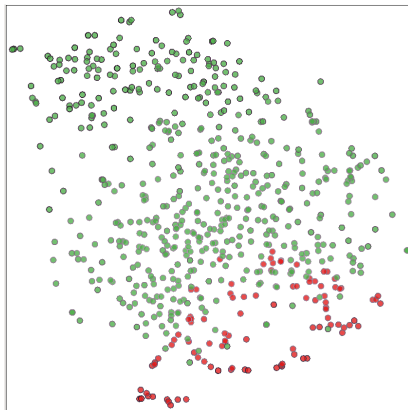
$$g'(f(v_j^r(s))) = \begin{cases} 1 & m = k, \\ 0 & \text{otherwise.} \end{cases}$$

The role of training a CNN is to increase class separation at the outputs of subsequent convolutional layers.



Feature projections (t-SNE) after layers 10, 11, 12, and 13 for larvae of helminth and impurities using VGG-16.

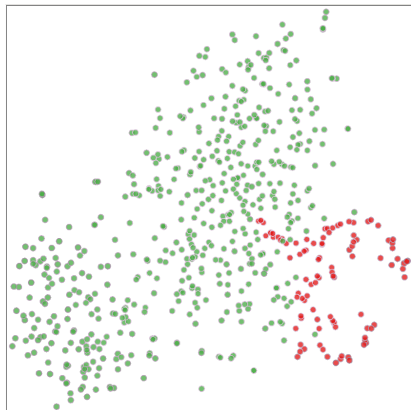
The role of training a CNN is to increase class separation at the outputs of subsequent convolutional layers.



Feature projections (t-SNE) after layers 10, 11, 12, and 13 for larvae of helminth and impurities using VGG-16.

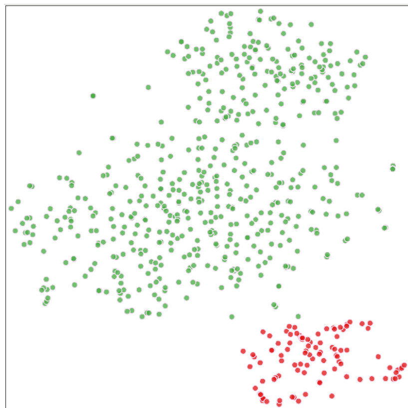
Convolutional Neural Networks

The role of training a CNN is to increase class separation at the outputs of subsequent convolutional layers.



Feature projections (t-SNE) after layers 10, 11, 12, and 13 for larvae of helminth and impurities using VGG-16.

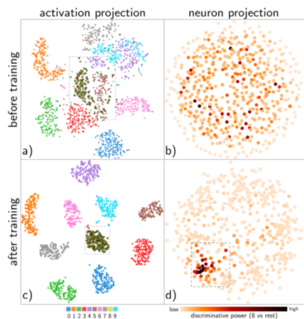
The role of training a CNN is to increase class separation at the outputs of subsequent convolutional layers.



Feature projections (t-SNE) after layers 10, 11, 12, and 13 for larvae of helminth and impurities using VGG-16.

Convolutional Neural Networks

Since convolutional layers make the feature space high and sparse, the fully-connected layers must reduce dimensionality by specializing the neurons that will activate (compose a hyperpolyhedron) to each class in the last hidden layer.



Neuron projections (MDS, right) colored by their discriminative power for class 8 versus the others in a digit dataset.

[1] Richard O. Duda, Peter E. Hart, and David G. Stork.
Pattern Classification (2Nd Edition).
Wiley-Interscience, 2000.

[2] W.P. Amorim, A.X. Falcão, J.P. Papa, and M.H. Carvalho.
Improving semi-supervised learning through optimum connectivity.
Pattern Recognition, 60:72 – 85, 2016.

[3] J.P. Papa, A.X. Falcão, V.H.C. de Albuquerque, and J.M.R.S. Tavares.
Efficient supervised optimum-path forest classification for large datasets.
Pattern Recognition, 45(1):512 – 520, 2012.

[4] J.P. Papa, S.E.N. Fernandes, and A.X. Falcão.
Optimum-path forest based on k-connectivity: Theory and applications.
Pattern Recognition Letters, 87:117 – 126, 2017.
Advances in Graph-based Pattern Recognition.

[5] Ludmila I. Kuncheva.
Combining Pattern Classifiers: Methods and Algorithms.
Wiley-Interscience, 2004.

[6] Konstantinos Koutroumbas and Sergios Theodoridis.
Pattern Recognition.
Elsevier, 2008.

[7] Simon Haykin.
Neural Networks: A Comprehensive Foundation (3rd Edition).
Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2007.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.

[9] David Forsyth.
Probability and Statistics for Computer Scientists.
Springer, 2018.