Machine Learning with Templates

Michael Stephen Fiske
Aemea Institute, San Francisco, CA, USA
mf@aemea.org

Abstract— *New methods are presented for the machine recognition and learning of categories, patterns, and knowledge. A probabilistic machine learning algorithm is described that scales favorably to extremely large datasets, avoids local minima problems, and provides fast learning and recognition speeds. Templates may be created using an evolutionary algorithm described here, constructed with other machine learning methods, designed by a human expert or synthesized using a combination of these methods. Each template has a prototype and matching function which can help improve generalization. These methods have applications in bioinformatics, financial data mining, goal-based planners, handwriting recognition, machine vision, natural language processing / understanding, search engines, strategy such as business and games and voice recognition.*

Keywords: evolution, machine learning, pattern recognition, polymorphous, template

1. Introduction

New machine learning methods and knowledge representation have sought to provide superior technology applications. In this regard, Machine Learning with Templates pertains to recognizing categories and predictive modelling, which can be important components of advanced software technology.

Machine Learning with Templates was designed to use the advantages of bottom-up and top-down methods. Over the last few decades, some practitioners in AI, cognitive psychology, machine learning and neurobiology have observed that some cognitive tasks are better suited to bottom-up methods, while other tasks are better performed by top-down methods (See [1], [7], [10], [11], [18], [19], [20]).

An example of a bottom-up representation is a feedforward neural network that uses a gradient descent learning algorithm ([1], [12]), applied to handwritten digit recognition [17]. Other types of tasks use top-down methods. For example, IBM's Deep Blue software program plays chess [15]. Hammond's *CHEF* program creates new cooking recipes by adapting old recipes [23].

2. Summary of Useful Properties

Machine Learning with Templates has some useful properties.

1) Categorization is probabilistic and polymorphous (See [5], [24], and pages 26-31 in [7]).
2) Learning algorithm 5.1 is extremely fast. It takes less than one minute – for a 5 Ghz Intel Pentium 4 computer – to build templates that successfully recognize handwritten letters with an error rate less than 0.5%. Some neural network training times for handwriting recognition take considerably more time. Further, the design of the neural network architecture may take human researchers many weeks.
3) Learning algorithm 5.1 does not use a *greedy* optimization algorithm such as gradient descent [1], so it avoids local minima problems. On extremely large datasets, locally greedy algorithms may not adequately train in a practical amount of time.
4) Each template has a prototype and matching function which can help improve generalization. In some applications, the use of prototypical examples and templates designed by a clever human expert can substantially increase machine learning accuracy and speed.
5) Recognition algorithm 4.1 is extremely fast. It scales well on huge datasets and large numbers of categories because it exploits *exponential elimination*. As an example, consider the task of recognizing Chinese characters [25]. Some estimates state that there are 180 million distinct categories of Chinese characters. When the learning algorithm builds a set of templates that on average eliminate $\frac{1}{3}$ of the remaining Chinese character categories, then one trial of the recognition algorithm on average uses only 46 randomly chosen templates to reduce 180 million possible Chinese categories to a single category. (46 is the largest natural number n satisfying inequality $180,000,000 * (\frac{2}{3})^n < 2$.)
6) Machine Learning with Templates is flexible enough to create useful applications in bioinformatics, financial data mining, goal-based planners, handwriting recognition, information retrieval, machine vision, natural language processing / understanding and voice recognition.

3. Definitions and Template Structure

The space $\mathcal{C}$ is called a category space. Sometimes a space is a mathematical set, but a space may have more structure [21]. If the category space is about concepts that are living

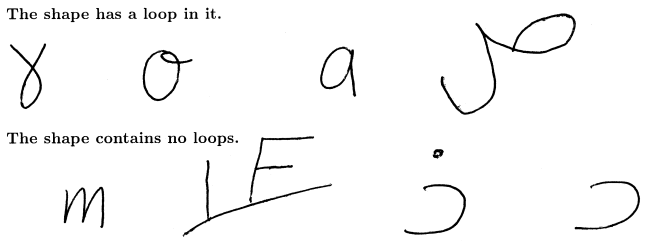

Fig. 1: Shapes with loops and no loops

creatures, then typical members of $\mathcal{C}$ are the categories: *dog*, *cat*, *animal*, *mammal*, *lizard*, *starfish*, *tree*, *barley*. A different category space is the letters of our alphabet $\{a, b, c, \ldots, y, z\}$. Another abstract type of category space may be the functional purpose of genes. One category is any gene that influences eye color; another category is any gene that codes for enzymes used in the liver; and another category is any gene that codes for membrane proteins used in neurons. In short, the structure of the category space depends upon what set of possibilities that you want the software to retrieve, recognize or categorize.

The example space $\mathcal{E}$ represents every conceivable example. Consider a software program that recognizes handwritten letters used in the English language [17]. The example space is every handwritten *a*; ... ; and every handwritten *z*. The set of training examples $\{e_1, e_2, \ldots, e_m\}$ is a subset of the example space $\mathcal{E}$.

The function $G : \mathcal{E} \rightarrow \mathcal{P}(\mathcal{C})$ is an ideal map or *ideal target function* [20], where $\mathcal{P}(\mathcal{C})$ is the power set of $\mathcal{C}$. In this case, each example e in $\mathcal{E}$ can be classified as lying in 0, 1 or multiple categories. The goal is to construct a function $g : \mathcal{E} \rightarrow \mathcal{P}(\mathcal{C})$ so that $g(e) = G(e)$ for every $e \in \mathcal{E}$. The example e = *tiger* lies in the categories *cat*, *animal* and *mammal*. In other words, $G(e) = \{cat, animal, mammal\}$. For some applications, it may impossible or extremely difficult to explicitly describe G with a mathematical formula or representation. In other implementations, the results of the template recognition algorithm can interpret e as having a probability $p(c)$ of lying in category c for each $c \in \mathcal{C}$. In these cases, the goal is to build $g : \mathcal{E} :\rightarrow [0, 1]^{\mathcal{C}}$ close to G.

Templates are similar to classifiers [20], but have additional structure. Templates are used to distinguish between two different categories of patterns, information or knowledge. For example, if the two different categories are the letters *a* and *m*, then *the shape has a loop in it* is a useful template because it distinguishes *a* from *m*. (See figure 1.)

Distinct from classifiers, templates have prototype and matching functions. Let $\{T_1, T_2, \ldots, T_n\}$ denote a collection of templates, called the template set. Associated to each template, there is a corresponding template value function $T_i : \mathcal{E} \rightarrow \mathcal{V}$, where $\mathcal{V}$ is a template value space. There are no restrictions made on the structure of $\mathcal{V}$, which may be a subset of the real line, a subset of the integers, a discrete set, a manifold, or even a function space.

Each template T_i has a corresponding prototype function $T_i : \mathcal{C} \rightarrow \mathcal{P}(\mathcal{V})$. The prototype function is constructed during the learning phase. If c is a category in $\mathcal{C}$, then $T_i(c)$ equals the set of prototypical template values that one expects for all examples e that lie in category c. Intuitively, the prototype function represents how template T_i generalizes to every example e.

For each $(template, category)$ pair, denoted as (T_i, c), there is a matching function $M_{(T_i,c)} : \mathcal{V} \rightarrow \mathcal{S}$, where $\mathcal{S}$ is the similarity space. The matching function determines if the template value is similar enough to its set of prototypical values. In general, the similarity space $\mathcal{S}$ is the range of the matching function, and can be the unit interval $[0, 1]$, a subset of $\mathbb{R}^n$, a discrete set, a manifold or a function space.

Example 3.1: Boolean Matching Function
Let $\mathcal{V} = \{0, 1\}$. Choose similarity space $\mathcal{S} = \{0, 1\}$. If template value v_1 is similar to its prototypical set $T_i(c)$, then $M_{(T_i,c)}(v_1) = 1$. If v_1 is not similar enough to its prototypical set $T_i(c)$, then $M_{(T_i,c)}(v_1) = 0$. Choose two categories $\{c_1, c_2\}$ and two templates $\{T_1, T_2\}$. Define prototypical functions for T_1 and T_2 as $T_1(c_1) = \{1\}$ and $T_1(c_2) = \{0, 1\}$. $T_2(c_1) = \{0, 1\}$ and $T_2(c_2) = \{0\}$. There are four distinct matching functions $M_{(T_1,c_1)}, M_{(T_2,c_1)}, M_{(T_1,c_2)}$ and $M_{(T_2,c_2)}$.

1) $M_{(T_1,c_1)}(0) = 0$ AND $M_{(T_1,c_1)}(1) = 1$
2) $M_{(T_2,c_1)}(0) = 1$ AND $M_{(T_2,c_1)}(1) = 1$
3) $M_{(T_1,c_2)}(0) = 1$ AND $M_{(T_1,c_2)}(1) = 1$
4) $M_{(T_2,c_2)}(0) = 1$ AND $M_{(T_2,c_2)}(1) = 0$

4. Template Recognition

The template recognition algorithm categorizes an example e from $\mathcal{E}$. When finished, for each category c in $\mathcal{C}$, there is a corresponding category score s_c, which measures to what extent the algorithm believes example e is in category c.

Algorithm 4.1: Template Recognition Algorithm

Allocate memory. Read learned templates $\{T_1, \ldots, T_n\}$ from long-term memory.

Initialize every category score s_c to zero.
Outer loop: m trials.
{
  Initialize set R equal to $\mathcal{C}$.
  Inner loop: choose ρ templates randomly.
  {
    Choose template T_k with probability p_k.
    For each category $c \in R$
      if $M_{(T_k,c)}(T_k(e)) = 0$, then set $R := R - \{c\}$.
      (Remove category c from R.)
  }
  For each category c remaining in R, category score $s_c := s_c + 1$.
}
Initialize A to the empty set.

For each category c in C, if $(\frac{s_c}{m}) > \theta$, $A := A \cup \{c\}$.
The answer is A. The example e is in the categories that are in A.

Comments on the Template Recognition Algorithm.

1) Each template T_i has a corresponding probability p_i of being chosen during the inner loop. m is the number of trials in the outer loop. ρ is the number of templates randomly chosen in each trial. θ is a number in the interval $[0, 1]$ that is the acceptable category threshold.
2) In some applications, the category threshold is not used. Each value $\frac{s_c}{m}$ is interpreted as the probability that example e lies in category c.
3) If only the best category is returned as an answer, rather than multiple categories, then do this by replacing the step For each category c in $\mathcal{C}$, if $(\frac{s_c}{m}) > \theta$, $A := A \cup \{c\}$. Instead, search for the maximum category score if there are a finite number of categories. The answer is the category or categories that have the maximum category score.
4) If template space $\mathcal{V}$ and similarity space $\mathcal{S}$ have more structure, then the test if $M_{(T_k,c)}(T_k(e)) = 0$ inside the inner loop may be replaced by the test if $T_k(e)$ is not close to prototype value $T_k(c)$. In this case, matching function $M_{(T_k,c)}$ measures the closeness of $T_k(e)$ and $T_k(c)$.
5) In some applications, the inner loop may be exited if R has only one element left (i.e., $|R| = 1$) before all ρ templates have been applied.

5. Template Learning

The initial part of the learning phase constructs the templates from simple building blocks, using the examples and the categories to guide the construction. Templates can be built with evolution, another machine learning method [1], by a human expert with domain expertise or by a combination of these methods. In the next section, evolution algorithm 6.1 builds template value functions $T_i : \mathcal{E} \to \mathcal{V}$ from a collection of building block functions $\{f_1, f_2, \ldots, f_r\}$. The rest of the learning builds the matching functions $M_{(T_i,c)}$, constructs the prototype functions $T_i : \mathcal{C} \to \mathcal{P}(\mathcal{V})$, computes the probabilities p_i, and in some cases sets the category threshold θ.

The learning starts with a collection of training examples along with their categories. Depending on the type of template values, there are different methods for constructing the prototype and matching functions $M_{(T_i,c)}$. For clarity, the template values used are boolean (i.e., $\mathcal{V} = \{0, 1\}$). In the boolean case, $M_{(T_i,c)}(v) = 1$ if v lies in the prototypical set $T_i(c)$; $M_{(T_i,c)}(v) = 0$ if v does not lie in the prototypical set $T_i(c)$. In a more general description of the algorithm, the template value space $\mathcal{V}$ may be the interval $[0, 1]$, the circle S^1, or another manifold, a function space, a space of algorithms or even a measurable space (e.g., [6], [22]). When $\mathcal{V}$ is a metric space [21], the matching function $M_{(T_k,c)}$ may use $\mathcal{V}$'s metric to measure the closeness of $T_k(e)$ and $T_k(c)$ as described in recognition comment 4.

Algorithm 5.1: Template Learning Algorithm

Allocate memory for the templates.
Read from memory template set $\{T_1, T_2, \ldots, T_n\}$.
(The templates read are user-created, created by evolution or an alternative method as in [16].)
Outer loop: iterate thru each template T_k.
{
  Initialize $X := T_k(c_1)$.
  Initialize $A := X$.
  Inner loop: iterate thru each category c_i.
  {
    Set $E_{c_i} :=$ all learning examples in c_i.
    Build prototype function T_k as follows:
    Set $T_k(c_i) := \cup\{v\}$ for each $v = T_k(e)$ and $e \in E_{c_i}$.
    Set $A := A \cup T_k(c_i)$.
    Build matching function $M_{(T_k,c_i)}$.
    (See above for boolean case.)
  }
  If $(A == X)$ remove T_k from the template set.
}
Store the remaining templates.
Set each probability $p_k = \frac{1}{m}$ where m is the number of remaining templates.

Comments on the Template Learning Algorithm.

1) The Inner loop assumes that there are a finite number of categories.
2) In some cases, instead of a category threshold, each score s_c is interpreted as the probability that e lies in category c. In other cases, the category threshold θ is empirically determined.
3) For a fixed template T_k, there should be at least one pair of categories (c_i, c_j) such that $T_k(c_i) \neq T_k(c_j)$. Otherwise, template T_k can not separate any categories, so T_k should be removed.
4) In some applications, non-uniform probabilities p_k can be selected based on template T_k's ability to separate categories, T_k's computing speed or another property.

6. Designing Templates with Evolution

The use of evolutionary methods for optimizing processes and algorithms was first introduced by [2], [3], [4] and [9] and were further developed in [12], [13] and [14]. Building upon this prior work, this section presents an evolutionary method to design the template value functions $T_i : \mathcal{E} \to \mathcal{V}$.

Building blocks are composed to build a useful element. In some cases, the building blocks are a collection of functions $f_\lambda : X \longrightarrow X$, where $\lambda \in \Lambda$, X is a set and Λ is an index set. In some cases, X is the set of computable real numbers. In a handwriting recognition application, X is the

rational numbers and the binary functions $f_1 = +$, $f_2 = -$, $f_3 = *$, and $f_4 = /$ are sufficient for the building blocks. The index set, $\Lambda = \{1, 2, 3, 4\}$, has four elements. In some cases, the index set may be infinite. For example, consider the functions $f_{(k,b)} : \mathbb{Z} \longrightarrow \mathbb{Z}$ such that $f_{(k,b)}(x) = p_k x + b$ where $b, k \in \mathbb{N}$ and p_k is the kth prime (i.e., $p_1 = 2, p_2 = 3, \ldots$).

Bit-sequences $[b_1 b_2 b_3, \ldots b_n]$, where $b_k \in \{0, 1\}$ encode functions composed from building block functions. $[10110]$ is a bit-sequence of length 5. The expression $\{f_1, f_2, f_3, \ldots, f_r\}$ denotes the building block functions, where $r = 2^K$ for some K. Then K bits uniquely represent one building block function. The arity of a function is the number of arguments that it requires. For example, the arity of the real-valued quadratic function $f(x) = x^2$ is 1. The arity of the projection function, $P_i : X^n \longrightarrow X$, is n, where $P_i(x_1, x_2, \ldots, x_n) = x_i$.

Define the function $\vee$ as $\vee(x, y, z) = x$ if ($x \geq y$ AND $x \geq z$), else $\vee(x, y, z) = y$ if ($y \geq x$ AND $y \geq z$), else $\vee(x, y, z) = z$. Consider the functions, $\{+, -, *, \vee\}$. Each sequence of two bits uniquely corresponds to one of these functions: $00 \longleftrightarrow +$ $01 \longleftrightarrow -$ $10 \longleftrightarrow *$ $11 \longleftrightarrow \vee$

Bit-sequence $[00, 01]$ encodes the function $+(-(x_1, x_2), x_3) = (x_1 - x_2) + x_3$, which has arity 3. For the general case, consider the building block functions $\{f_1, f_2, f_3, \ldots, f_r\}$, where $r = 2^K$. Any bit-sequence $[b_1\ b_2\ \ldots\ b_K\ b_{K+1}\ b_{K+2}\ \ldots b_{2K} \ldots b_{aK+1}\ b_{aK+2} \ldots b_{(a+1)K}]$ with length $(a+1)K$ is a composition of the building block functions, $\{f_1, f_2, f_3, \ldots, f_r\}$. The composition of these r building block functions are encoded in a similar way, as described for functions $\{+, -, *, \vee\}$.

The distinct categories are $\{C_1, C_2, \ldots, C_N\}$. The population size of each generation is m. For each i, where $1 \leq i \leq N$, E_{C_i} is the set of all learning examples that lie in category C_i. The symbol γ is an acceptable level of performance for a template. The symbol Q is the number of distinct templates whose fitness must be greater than γ. The symbol $p_{crossover}$ is the probability that two templates chosen for the next generation will be crossed over. The symbol $p_{mutation}$ is the probability that a template will be mutated.

The main evolution steps are summarized. For each category pair (C_i, C_j), $i < j$, the building blocks $\{f_1, f_2, f_3, \ldots, f_r\}$ are used to build a population of m templates. This is accomplished by choosing m multiples of K, $\{l_1, l_2, \ldots, l_m\}$. For each l_i, a bit sequence of length l_i is constructed. These m bit sequences represent the m templates, $\{T_1{}^{(i,j)}, T_2{}^{(i,j)}, T_3{}^{(i,j)}, \ldots, T_m{}^{(i,j)}\}$. The superscript (i, j) represents that these templates are evolved to distinguish examples chosen from E_{C_i} and E_{C_j}. The fitness of each template is determined by how well the template can distinguish examples chosen from E_{C_i} and E_{C_j}. Using crossover and mutation, the population of bit-sequences are evolved until there are at least Q templates which have a fitness greater than γ. When this happens, choose the Q best templates from the population that distinguish categories C_i and C_j. Store these Q best templates in a distinct set $\mathcal{T}$ of templates that are used in the template learning algorithm.

Algorithm 6.1: Building Templates with Evolution

```
Set T equal to the empty set.
For each i in {1, 2, 3, ..., N}
For each j in {i+1, i+2, ..., N}
{
  Initialize population A_(i,j) = {T_1^(i,j), ... T_m^(i,j)}
  Set q := 0.
  while (q < Q)
  {
    Set G := ∅.
    while (|G| < m)
    {
      For the next generation, randomly choose
      templates T_a^(i,j) and T_b^(i,j) from A_(i,j) where
      the probability is proportional to the
      template's fitness.

      Randomly choose a number r in [0, 1].

      If (r < p_crossover), then crossover templates
      templates T_a^(i,j) and T_b^(i,j).

      Randomly choose numbers s_a, s_b in [0, 1].

      If (s_a < p_mutation), mutate template T_a^(i,j).
      If (s_b < p_mutation), mutate template T_b^(i,j).

      Set G := G ∪ {T_a^(i,j), T_b^(i,j)}.
    }
    Set A_(i,j) := G.

    For each template T_a^(i,j) in A_(i,j), evaluate
    T_a^(i,j)'s ability to distinguish examples
    from categories C_i and C_j.

    Store this ability as the fitness of T_a^(i,j)

    Set q equal to the number of templates
    with fitness greater than γ.
  }
  Based on fitness, choose the Q best
  templates from A_(i,j) and add them to T.
}
```

Comments on Building Templates with Evolution.

1) The fitness ϕ_a of template $T_a{}^{(i,j)}$ is computed by a weighted average of three criteria.
   a) The ability of a template to distinguish examples in E_{C_i} from examples in E_{C_j}
   b) The amount of memory used by the template.
   c) The average amount of time to compute the template value function on E_{C_i} and E_{C_j}.

   A quantitative measure for criterion (a) depends on the topology of the template value space $\mathcal{V}$. If $\mathcal{V} = \{0, 1\}$, the ability of template $T_a{}^{(i,j)}$ to distinguish examples in E_{C_i} from examples in E_{C_j} equals

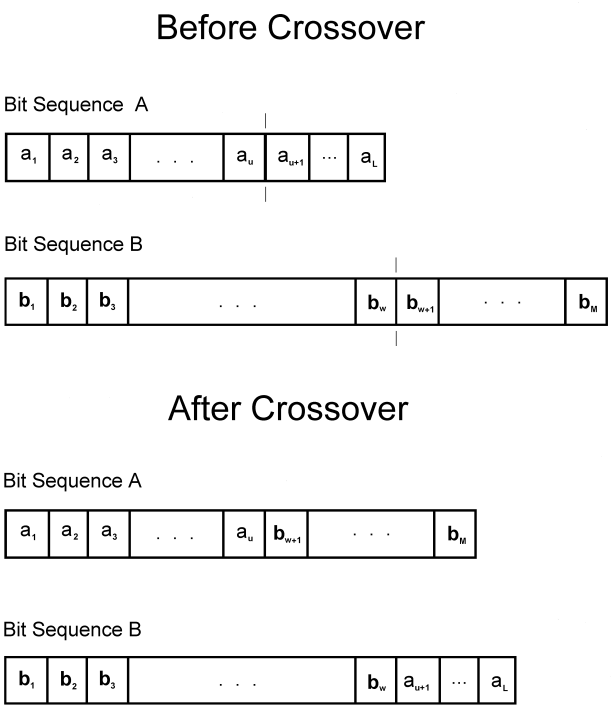

Fig. 2: Unbounded Crossover

$\frac{1}{|E_{C_i}||E_{C_j}|} \sum_{e_j \in E_{C_j}} \sum_{e_i \in E_{C_i}} |{T_a}^{(i,j)}(e_i) - {T_a}^{(i,j)}(e_j)|$.
When $\mathcal{V} \neq \{0,1\}$, then $\mathcal{V}$ has a metric $\mathcal{D}$, which measures the distance between two points in the template value space $\mathcal{V}$. In this case, the ability of template ${T_a}^{(i,j)}$ to distinguish examples E_{C_i} and E_{C_j} equals
$\frac{1}{|E_{C_i}||E_{C_j}|} \sum_{e_j \in E_{C_j}} \sum_{e_i \in E_{C_i}} \mathcal{D}({T_a}^{(i,j)}(e_i),\ {T_a}^{(i,j)}(e_j))$.

2) Figure 2 shows a crossover between bit-sequences $A = [a_1 a_2 a_3 \ldots a_L]$ and $B = [b_1 b_2 b_3 \ldots b_M]$. L and M are each multiples of K, where $r = 2^K$ and the functions are $\{f_1, f_2, f_3, \ldots, f_r\}$. In general $L \neq M$. Two natural numbers u and w are randomly chosen such that $1 \leq u \leq L$, $1 \leq w \leq M$ and $u + M - w$ is a multiple of K. The multiple of K condition assures that after crossover, the length of each bit sequence is a multiple of K. Each bit sequence after crossover is interpreted as a composition of functions $\{f_1, f_2, f_3, \ldots, f_r\}$. The numbers u and w identify the crossover locations on A and B, respectively. After crossover, bit-sequence A is $[a_1 a_2 a_3 \ldots a_u b_{w+1} b_{w+2} \ldots b_M]$, and bit-sequence B is $[b_1 b_2 b_3 \ldots b_w a_{u+1} a_{u+2} \ldots a_L]$.

3) Before mutation, the bit-sequence is $[b_1 b_2 b_3 \ \ldots b_n]$. A mutation randomly selects k and assigns b_k the value $1 - b_k$.

4) In the current generation, the collection $\{\phi_1, \phi_2, \ldots, \phi_m\}$ represents the fitnesses of the templates $\{{T_1}^{(i,j)}, {T_2}^{(i,j)}, \ldots, {T_m}^{(i,j)}\}$. The probability that template ${T_a}^{(i,j)}$ is chosen for the next generation is $\frac{\phi_a}{\sum_{k=1}^{m} \phi_k}$.

5) $p_{crossover}$ usually ranges from 0.3 to 0.7.

6) $p_{mutation}$ is usually less than 0.1.

7. UCI Machine Learning Tests

Testing against the UCI Machine Learning Repository `http://archive.ics.uci.edu/ml/` is in progress. A subsequent publication will cover these results.

8. Acknowledgements

I would like to thank Michael Jones, David Lewis, A. Mayer, Lutz Mueller, Don Saari and Francesca Saglietti for their helpful advice.

References

[1] Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

[2] W.W. Bledsoe. The use of biological concepts in the analytical study of systems. *ORSA-TIMS National Meeting*, San Francisco, CA, 1961.

[3] G.E.P. Box. Evolutionary operation: A method for increasing industrial production. *Journal of the Royal Statistical Society*, C, 6(2), pp. 81-101. 1957.

[4] R.J. Bremermann. Optimization through evolution and recombination. *Self-organizing systems.*, pp. 93-106. Washington, D.C., Spartan Books, 1962.

[5] I. Dennis, J.A. Hampton, and S.E.G. Lea. New problem in concept formation. *Nature*. **243**, pp. 101-102, 1973.

[6] Abbas Edalat. A computable approach to measure and integration theory. *Information and Computation*. **207**, Issue 5, May 2009, Pages 642-659

[7] Gerald Edelman. *Neural Darwinism*. Basic Books, Inc., 1987.

[8] Michael S. Fiske. *Machine Learning*. U.S. Patent 7,249,116. 2003. `http://www.aemea.com/7,249,116`.

[9] G.J. Friedman. Digital simulation of an evolutionary process. *General Systems Yearbook*, **4**, pp. 171-184. 1959.

[10] Dedre Gentner. The mechanisms of analogical learning. *Similarity and Analogical Reasoning*. Edited by Stella Vosniadou and Andrew Ortony, pp. 199-241. Cambridge University Press, 1989.

[11] Dedre Gentner, Arthur B. Markman. Structure mapping in analogy and similarity. *American Psychologist*. **52**(1), pp. 45-56, January 1997.

[12] John Hertz, Anders Krogh and Richard G. Palmer. *Introduction To The Theory of Neural Computation*. pp. 124-129. Addison-Wesley Publishing Company. Redwood City, California, 1991.

[12] John H. Holland. Genetic Algorithms and the optimal allocation of trials *SIAM Journal of Computing*. 2(2), pp. 88-105. 1973.

[13] John H. Holland. Genetic algorithms and adaptation. *Technical Report No. 34, Ann Arbor, University of Michigan*. Department of Computer and Communication Sciences, 1981.

[14] John H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Perseus Books, 1995.

[15] Feng-hsiung Hsu, Thomas Anantharaman, Murray Campbell, and Andreas Nowatzyk. A Grandmaster Chess Machine. *Scientific American*, October edition, 1990.

[16] J.R. Koza. *Genetic Programming: On the Programming of Computer by Means of Natural Selection* Cambridge, MA. MIT Press, 1992.

[17] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. *Adv. in Neural Information Processing Systems*, 2, D. Touretzky (Editor). Morgan Kaufmann, 1990.

[18] David Marr. *Vision*. W.H. Freeman & Company, 1982.

[19] Marvin Minsky. Logical vs. Analogical or Symbolic vs. Connectionist or Neat vs. Scruffy *Artificial Intelligence at MIT. Expanding Frontiers*, Patrick H. Winston (Ed.), Volume 1, MIT Press, 1990. Online at `http://bit.ly/L9Mobg`.

[20] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[21] James R. Munkres. *Topology*. Prentice-Hall, 1975.

[22] H.L. Royden. *Real Analysis*. 3rd Edition. Prentice-Hall, 1988.

[23] Roger Schank, Christopher Riesbeck. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, ISBN 0-89859-767-6, 1989.

[24] Ludwig Wittgenstein. *Philosophical Investigations*. 1953.

[25] Su-Ling Yeh, Jing-Ling Li, Tasuto Takeuchi, Vincent Sun, and Wen-Ren Liu. The role of learning experience on the perceptual organization of Chinese Characters. *Visual Cognition*, 10(6), pp. 729-764. 2003.