Data-dependencies and Learning in Artificial Systems

Palem GopalaKrishna KRISHNA@CSE.IITB.AC.IN
Research Scholar, Computer Science & Engineering, Indian Institute of Technology - Bombay, Mumbai, India

Abstract

Data-dependencies play an important role in the performance of learning algorithms. In this paper we analyze the concepts of data dependencies in the context of artificial systems. When a problem and its solution are viewed as points in a system configuration, variations in the problem configurations can be used to study the variations in the solution configurations and vice versa. These variations could be used to infer solutions to unknown instances of problems based on the solutions to known instances, thus reducing the problem of learning to that of identifying the relations among problems and their solutions. We use this concept in constructing a formal framework for a learning mechanism based on the relations among data attributes. As part of the framework we provide metrics – *quality* and *quantity* – for data samples and establish a knowledge conservation theorem. We explain how these concepts can be used in practice by considering an example problem and discuss the limitations.

1. Introduction

Two instances of a function can only differ in their arguments, i.e. the input data. When a function is sensitive to the data it is operating upon, even a slight variation in the nature of data can cause large variations in the path of execution. This property of being sensitive to data is termed as *data-dependency* which poses critical restrictions on the applicability of algorithms themselves.

The success of any data-dependent learning algorithm highly depends on the nature of the data samples it learns from. A well designed algorithm with mismatched data is unlikely to succeed in generalization. Thus a careful analysis of the size and quality of the input data samples is vital for the success of every learning algorithm. While there exists sufficient number of metrics for learning in traditional systems in this regard (Kearns, 1990; Angluin, 1992), there exists almost none for learning in artificial systems, where the typical requirements would be *action selection* and *planning* implemented through agents (Wilson, 1994; Bryson, 2003). These agents would act as deterministic systems and thus demand non-probabilistic metrics with data-independent algorithms.

Data-independence essentially means that the path of execution (the series of instructions carried out) is independent of the nature of the input data. In other words, when an algorithm is said to be data-independent, all instances of the algorithm would follow the same execution path no matter what the input data is. We can understand this with the following example. Consider an algorithm to search a number in a given array of numbers. Such an algorithm would typically look like below.

```
int Search(int Array[], int ArrLen, int Number) {
    for( int i=0; i < ArrLen; ++i )
        if( Array[i] == Number)
            return i;
    return -1;
}
```

The above procedure sequentially scans a given array of numbers to find if a given number is present in the array. It returns the index of the number if it finds a match and -1 otherwise. The time complexity of this algorithm is $O(1)$ in the best case and $O(n)$ in the average and worst cases. However, if we change the iterator construct from $for(i = 0; i <$ ArrLen; $++i)$ to $for(i=$ArrLen-1; $i \geq 0; --i)$, then the performances would vary from best to worst and vice versa.

On the other hand consider the following data-independent version of the same code.

```
int Search1(int Array[], int ArrLen, int Number) {
    int nIndex = -1;
    for(int i=0; i < ArrLen; ++i) {
        int bEqual = (Number == Array[i]);
        nIndex = bEqual * i + !bEqual * nIndex;
    }
    return nIndex;
}
```

Search1 is same as *Search* with the mere exception that we have replaced the non-deterministic *if* statement with a series of deterministic arithmetic constructs that in the end produce same results. The advantage with this replacement is that the path of execution is deterministic and independent of the input array values, thus facilitating us to reorder or even parallelize the individual iterations. This is possible because no $(i+1)$th iteration depends on the results of ith iteration, unlike the case of *search* where the $(i+1)$th iteration would be processed only if the ith iteration fails to find a match.

Demanding a time complexity of $O(n)$ in all cases, it might appear that *Search1* is inferior to *Search* in performance. However, for this small cost of performance we are gaining two invaluable properties that are crucial for our present discussion: *stability* and *predictability*.

It is a well-known phenomenon in the practice of learning algorithms that the performance of learner is highly affected by the order of the training data samples, making the learner unstable and at times unpredictable. In this regard, what relation could one infer between the stability of the learner and the dependencies among data samples? How does such relation affect the performance of learner? Can these dependencies be analyzed in a formal framework to assist the learning? These are some of the issues that we try to address in the following.

2. Learning in Artificial Systems

By an *artificial system* we essentially mean a man-made system that has a software module, commonly known as *agent*, as one of its components. The artificial system itself could be a software program such as a simulation program in a digital computer, or it could be a hardware system such as an autonomous robot in the real world. And there could be more than one agent in an artificial system. The system can use the agents in many ways as to steer the course of simulation or to process the environmental inputs (or events) and take the necessary action etc.... Additionally, the functionality of agents could be static, i.e. does not change with experience, or it could be dynamic, varying with experience. The literature addressing these can be broadly classified into two classes, namely the theories that study the agents as pre-programmed units (such as (Reynolds, 1987; Ray, 1991)), and the theories that consider the agents as *learning units* which can adjust their functionality based on their experience (e.g. (Brooks, 1991; Ramamurthy et al., 1998; Cliff & Grand, 1999)). The present discussion falls into the second category. We discuss a learning mechanism for agents based on the notion of data-dependencies.

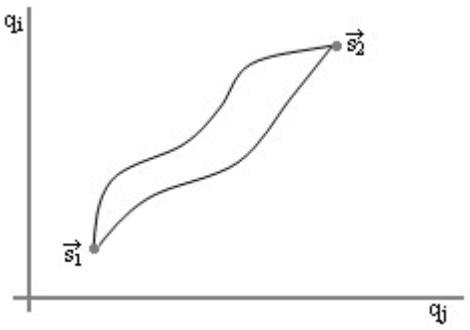

Figure 1. Different paths indicate different algorithms to solve a task instance in configuration space

Consider an agent that is trying to accomplish a task, such as solving a maze or sorting the events based on priority etc..., in an artificial system.

Assume that the instantaneous configuration (the current state) of artificial system is described by n generalized coordinates $q_1, q_2, \ldots, q_n$, which corresponds to a particular point in a Cartesian hyperspace, known as the *configuration space*, where the q's form the n coordinate axes. As the state of the system changes with time, the system point moves in the configuration space tracing out a curve that represents "the path of motion of the system".

In such a configuration space, a task is specified by a set of system point pairs representing the initial and final configurations for different instances of the task. An instance of the task is said to be *solvable* if there exists an algorithm that can compute the final configuration from its initial configuration. The task is said to be solvable if there exists an algorithm that can solve all its instances.

Each instance of the algorithm solving an instance of the task represents a path of the system in the configuration space between the corresponding initial and final system points. If there exists more than one algorithm to solve the task then naturally there might exist more than one path between the two points.

The goal of an agent that is trying to learn a task in such a system is to observe the algorithm instances and infer the algorithm. In this regard, all the information that the agent would get from an algorithm instance is just an initial-final configuration pair along with a series of configuration changes that lead from initial configuration to final configuration. The agent would not be aware of the details of the underlying process

that is responsible for these changes, and has to infer the process purely based on the observations.

The agent is said to have "learned the task" if it can perform the task on its own, by moving the system from any given initial configuration to the corresponding final configuration in the configuration space. It should be noted that the procedure used (the algorithm inferred) by the agent may not be the same as the original algorithm from whose instances it has learned.

It should also be noted that the notion of learning the task, as described above, does not allow any probabilistic or approximate solutions. The agent should be able to perform the task correctly under all circumstances. An additional constraint that we put on the agent is that it should infer the algorithm from as few algorithmic instances as possible. This is important for agents of both real world systems and simulation systems alike, for in case of agents observing samples from real world environment it may not be possible to pickup as many samples as they want, and in case of simulated environments each sample instance incurs an execution cost in terms of time and other resources and hence should be traded sparingly.

We formalize these concepts in the following.

2.1. A Formal Framework

Consider an agent that it trying to learn a task T in a system S whose configuration space is given by

$$\mathcal{C}(S) = \{\vec{s_1}, \vec{s_2}, \ldots, \vec{s_N}\},$$

where each $\vec{s_i}$ is a system point represented with $n-$coordinates $\{q_{i_1}, q_{i_2}, \ldots, q_{i_n}\}$.

Let A be an algorithm to solve the task T, and $A_1, A_2, \ldots, A_k$ be the instances of A solving the instances $T_1, T_2, \ldots, T_k$ of T respectively.

In the configuration space each T_i is represented by a pair of system points $(\vec{s}_{i_1}, \vec{s}_{i_2})$, and the corresponding A_i by a path between those system points.

Let I, F be two operators that when applied to an algorithm instance A_i, yield the corresponding initial and final system points respectively, such as $I(A_i) = \vec{s}_{i_1}$ and $F(A_i) = \vec{s}_{i_2}$. We also define the corresponding set versions of these operators $\vec{I}$ and $\vec{F}$ as following. For all $A' \subseteq \{A_1, A_2, \ldots, A_k\}$,

$$\vec{I}(A') = \{I(A_i) \mid A_i \in A'\},$$

and

$$\vec{F}(A') = \{F(A_i) \mid A_i \in A'\}.$$

The goal of the agent is to perform T, by mimicking or modeling A, inferring A's details from a subset of its instances.

By following the tradition of learning algorithms, let us call A as the *target concept*, and the subset of its instances $D = \{D_1, D_2, \ldots, D_d\} \subseteq \{A_1, A_2, \ldots, A_k\}$ as the *training set* or *data samples*, and the agent as the *learner*. We use the symbol L to denote the learner.

At any instance during the phase of learning the set D can be partitioned into two subsets O, O' such that

$$(O \cup O' = D) \ \wedge \ (O \cap O' = \emptyset).$$

The set $O \subseteq D$ denotes the set of data samples that the learner has already seen, and the set $O' \subseteq D$ denotes the set of data samples the learner has yet to see. Learning progresses by presenting the learner with an unseen sample $D_i \in O'$, and marking it as seen, by moving it to the set O. Starting from $O = \emptyset, O' = D$, this process of transition would be repeated till it becomes $O = D, O' = \emptyset$.

In this process, each data sample D_i decreases the ignorance of the learner L about the target concept A, and hence could be assigned some specific *information content* value that indicates how much additional information L can gain from D_i about A.

We can determine the information content values of data samples by establishing the concept of a *zone,* where we treat an ensemble of system points that share a common relation as a single logical entity.

Definition. A set $Z \subseteq \mathcal{C}(S)$ defines a zone if there exists a function $f : \mathcal{C}(S) \rightarrow \{0, 1\}$ such that for each $\vec{s_i} \in \mathcal{C}(S)$:

$$f(s_i) \ = \ \begin{cases} 1 & \text{if } \vec{s_i} \in Z, \\ 0 & \text{if } \vec{s_i} \notin Z. \end{cases}$$

The function f is called the characteristic function of Z.

For the configuration space $\mathcal{C}(S) = \{\vec{s_1}, \vec{s_2}, \ldots, \vec{s_m}\}$, we can construct an equivalent *zone-space* $\mathcal{Z}(S) = \{Z_1, Z_2, \ldots, Z_r\}$, such that the following holds.

$$\forall \vec{s_i} \, \exists Z_i \, [\vec{s_i} \in Z_i] \ \wedge \ \forall_{i=1}^{r} \forall_{j=i+1}^{r} \, [Z_i \cap Z_j = \emptyset] \ \wedge \ \forall_{i=1}^{r} \, [Z_i \neq \emptyset] \ \wedge \ \cup_{i=1}^{r} Z_i = \mathcal{C}(S).$$

The zone-space can be viewed as a $m-$dimensional space with each Z_i being a point in it, where m is some function of n whose value depends upon and hence would be decided by the nature of T. Let the range of ith coordinate of this $m-$dimensional space be $[0, r_i]$.

If we use the notation $|P|$ to indicate the size of any set P, then we could represent the volume of the zone-space as

$$vol(\mathcal{Z}(S)) = |\mathcal{Z}(S)| = \prod_{i=1}^{m} r_i.$$

Define an operator $\nabla : \mathcal{C}(S) \rightarrow \mathcal{Z}(S)$ that when applied to a system point in the configuration space yields the corresponding zone in the zone-space. Similarly, let $\vec{\nabla}$ be the corresponding set version of this operator defined as, for all $S' \subseteq \mathcal{C}(S)$,

$$\vec{\nabla}(S') = \{\nabla(\vec{s_i}) \mid \vec{s_i} \in S'\}.$$

At any instance the *knowledge* of L about A depends on the set of samples it has seen till then, and the information content of a data sample depends on whether the sample has already been seen by L or not.

To define formally, the knowledge of the learner, after having seen a set of samples $O \subseteq D$, is given by

$$\mathcal{K}_O(L) \quad = \quad \sum_{Z \in \vec{\nabla}(\vec{I}(O))} |Z| \,.$$

The information content of any data sample $D_i \in D$, after the learner has seen a set of samples $O \subseteq D$, is given by

$$IC_O(D_i) = \begin{cases} |\nabla(I(D_i))| & \text{if } \forall D_j \in O \quad [\nabla(I(D_i)) \neq \nabla(I(D_j))]; \\ 0 & \text{otherwise;} \end{cases}$$

and the information content of all data samples would be given by

$$\overrightarrow{IC}_O(D) \quad = \quad \sum_{Z \in \vec{\nabla}(\vec{I}(D-O))} |Z| \,.$$

It should be noted that the above definitions measure the information content of data samples relative to the state of the learner and satisfy the limiting conditions $\mathcal{K}_\emptyset(L) = 0$, $\overrightarrow{IC}_D(D) = 0$.

The process of learning is essentially a process of transfer of information from data samples to the learner, resulting in a change in the state of the learner. When these changes are infinitesimal, spanning many steps, the transformation process satisfies the condition that the line integral

$$\mathcal{L} = \int_{\emptyset}^{D} E \;\; do, \tag{2.1}$$

where $E = \overrightarrow{IC}_O(D) - \mathcal{K}_O(L)$, has a stationary value.

This is known as the Hamilton's principle, which states that out of all possible paths by which the learner could move from $\mathcal{K}_\emptyset(L)$ to $\mathcal{K}_D(L)$, it will actually travel along that path for which the value of the line integral (2.1) is stationary. The phrase "stationary value" for a line integral typically means that the integral along the given path has same value to within first-order infinitesimals as that along all neighboring paths (Goldstein, 1980; McCauley, 1997).

We can summarize this by saying that the process of learning is such that the *variation* of the line integral $\mathcal{L}$ is zero.

$$\delta\mathcal{L} = \delta \int_{\emptyset}^{D} E \;\; do = 0.$$

Thus we can formulate the following conservation theorem.

Theorem 1. *The sum $\mathcal{K}_O(L) + \overrightarrow{IC}_O(D)$ is conserved for all $O \subseteq D$.*

Proof. We shall prove this by establishing that $\mathcal{K}_{O_i}(L) + \overrightarrow{IC}_{O_i}(D) = \mathcal{K}_{O_j}(L) + \overrightarrow{IC}_{O_j}(D)$ for all $O_i, O_j \subseteq D$.

Consider $O_1, O_2 \subseteq D$ such that $|O_2| - |O_1| = 1$. Let $O_2 - O_1 = \{D_i\}$. To calculate the information content value of D_i, we need to consider two cases.

Case 1. $\nabla(I(D_i)) = \nabla(I(D_j))$ for some $D_j \in O_1$. In such case, $\vec{\nabla}(\vec{I}(O_2)) = \vec{\nabla}(\vec{I}(O_1))$, and hence $IC_{O_1}(D_i) = IC_{O_2}(D_i) = 0$.

$$\begin{aligned} \mathcal{K}_{O_2}(L) &= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(O_2))} |Z| \\ &= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(O_1))} |Z| \\ &= \mathcal{K}_{O_1}(L). \\ \overrightarrow{IC}_{O_2}(D) &= \overrightarrow{IC}_{O_1}(D) - IC_{O_1}(D) \\ &= \overrightarrow{IC}_{O_1}(D). \\ \mathcal{K}_{O_2}(L) + \overrightarrow{IC}_{O_2}(D) &= \mathcal{K}_{O_1}(L) + \overrightarrow{IC}_{O_1}(D). \end{aligned}$$

Case 2. $\nabla(I(D_i)) \neq \nabla(I(D_j))$ for all $D_j \in O_1$. In such case, $IC_{O_1}(D_i) = |\nabla(I(D_i))| \,.$

$$\begin{aligned} \overrightarrow{IC}_{O_2}(D) &= \overrightarrow{IC}_{O_1}(D) - IC_{O_1}(D_i) \\ &= \overrightarrow{IC}_{O_1}(D) - |\nabla(I(D_i))| \,. \\ \mathcal{K}_{O_2}(L) &= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(O_2))} |Z| \\ &= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(O_1 + \{D_i\}))} |Z| \\ &= \sum_{Z \in \vec{\nabla}(\vec{I}(O_1))} |Z| + \sum_{Z \in \{\nabla(I(D_i))\}} |Z| \\ &= \mathcal{K}_{O_1}(L) + |\nabla(I(D_i))| \,. \\ \mathcal{K}_{O_2}(L) + \overrightarrow{IC}_{O_2}(D) &= \mathcal{K}_{O_1}(L) + |\nabla(I(D_i))| + \\ &\quad \overrightarrow{IC}_{O_1}(D) - |\nabla(I(D_i))| \\ &= \mathcal{K}_{O_1}(L) + \overrightarrow{IC}_{O_1}(D). \end{aligned}$$

Thus whenever $|O_2| - |O_1| = 1$, it holds that

$$\mathcal{K}_{O_2}(L) + \overrightarrow{IC}_{O_2}(D) = \mathcal{K}_{O_1}(L) + \overrightarrow{IC}_{O_1}(D).$$

Now consider two sets $O_i, O_j \subseteq D$ such that $|O_j| - |O_i| = l$, $l > 1$. Let $O_j - O_i = \{D_{j_1}, \ldots, D_{j_l}\}$. We can construct sets $P_1, \ldots, P_{l-1}$ such that

$$P_1 = O_i \cup \{D_{j_1}\}, \ldots, P_{l-1} = O_i \cup \{D_{j_1}, \ldots, D_{j_{l-1}}\}.$$

Then it holds that

$$|P_1| - |O_i| = |P_2| - |P_1| = \cdots = |O_j| - |P_{l-1}| = 1.$$

However, we have proved that

$$\mathcal{K}_{O_2}(L) + \overrightarrow{IC}_{O_2}(D) = \mathcal{K}_{O_1}(L) + \overrightarrow{IC}_{O_1}(D)$$

whenever $|O_2| - |O_1| = 1$, and hence it follows that

$$\begin{aligned}
\mathcal{K}_{O_i}(L) + \overrightarrow{IC}_{O_i}(D) &= \mathcal{K}_{P_1}(L) + \overrightarrow{IC}_{P_1}(D),\\
\mathcal{K}_{P_1}(L) + \overrightarrow{IC}_{P_1}(D) &= \mathcal{K}_{P_2}(L) + \overrightarrow{IC}_{P_2}(D),\\
&\vdots\\
\mathcal{K}_{P_{l-1}}(L) + \overrightarrow{IC}_{P_{l-1}}(D) &= \mathcal{K}_{O_j}(L) + \overrightarrow{IC}_{O_j}(D),
\end{aligned}$$

and thereby, $\mathcal{K}_{O_i}(L) + \overrightarrow{IC}_{O_i}(D) = \mathcal{K}_{O_j}(L) + \overrightarrow{IC}_{O_j}(D)$.

Hence the sum $\mathcal{K}_O(L) + \overrightarrow{IC}_O(D)$ is conserved for all $O \subseteq D$. □

An important consequence of this theorem is that irrespective of the order of individual samples that L chooses to learn from, the gain in its knowledge would always be equal to the corresponding loss in the information content of the data samples.

$$\mathcal{K}_{O_j}(L) - \mathcal{K}_{O_i}(L) = \overrightarrow{IC}_{O_i}(D) - \overrightarrow{IC}_{O_j}(D).$$

We now define two metrics – *quality* and *quantity* – for the data samples to denote the notions of *necessity* and *sufficiency*.

The metric *quality* measures the relative information strength of individual samples, defined as

$$quality(D) = \frac{|D| - \left|\vec{\nabla}(\vec{I}(D))\right|}{|D|} \times 100\%.$$

Ideally a data sample set should have this value to be 100%. Smaller values indicate the presence of unnecessary samples that do not contribute to learning.

Similarly we define the *quantity* of data samples as

$$quantity(D) = \frac{\left|\vec{\nabla}(\vec{I}(D))\right|}{|\mathcal{Z}(S)|} \times 100\%.$$

This is a sufficiency measure and hence a value less than 100% indicates the insufficiency of data samples to complete the learning.

Theorem 2. *The knowledge of the learner, after completing the learning over data samples D having $quantity(D) = 100\%$, would be equal to the volume of the configuration space $|\mathcal{C}(S)|$.*

Proof. When the $quantity(D) = 100\%$,

$$\left|\vec{\nabla}(\vec{I}(D))\right| = |\mathcal{Z}(S)|.$$

Since $\vec{\nabla}(\vec{I}(D)) \subseteq \mathcal{Z}(S)$,

$$\left|\vec{\nabla}(\vec{I}(D))\right| = |\mathcal{Z}(S)| \;\Rightarrow\; \vec{\nabla}(\vec{I}(D)) = \mathcal{Z}(S).$$

From theorem 1 we have,

$$\mathcal{K}_D(L) + \overrightarrow{IC}_D(D) = \mathcal{K}_\emptyset(L) + \overrightarrow{IC}_\emptyset(D).$$

Since $\mathcal{K}_\emptyset(L) = 0$ and $\overrightarrow{IC}_D(D) = 0$,

$$\begin{aligned}
\mathcal{K}_D(L) &= \overrightarrow{IC}_\emptyset(D)\\
&= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(D-\emptyset))} |Z|\\
&= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(D))} |Z|\\
&= \textstyle\sum_{Z \in \mathcal{Z}(S)} |Z|\\
&= |\mathcal{C}(S)|.
\end{aligned}$$

Thus when $quantity(D) = 100\%$, $\mathcal{K}_D(L) = |\mathcal{C}(S)|$. □

Theorem 3. *The target concept can not be learnt with less than $|\mathcal{Z}(S)|$ number of data samples.*

Proof. Consider a data sample set $D = \{D_1, \ldots, D_d\}$ having $quantity(D) = 100\%$ and $|D| < |\mathcal{Z}(S)|$.

Let $P = \mathcal{Z}(S) - \vec{\nabla}(\vec{I}(D)) = \{P_1, \ldots, P_l\}$, $l \geq 1$. Assume that L has learned the target concept completely from D. Then, by theorem 2, $\mathcal{K}_D(L) = |\mathcal{C}(S)|$, and by theorem 1,

$$\mathcal{K}_D(L) + \overrightarrow{IC}_D(D) = \mathcal{K}_\emptyset(L) + \overrightarrow{IC}_\emptyset(D).$$

Since $\mathcal{K}_\emptyset(L) = 0$ and $\overrightarrow{IC}_D(D) = 0$, it leads to

$$\begin{aligned}
|\mathcal{C}(S)| &= \overrightarrow{IC}_\emptyset(D)\\
&= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(D-\emptyset))} |Z|\\
&= \textstyle\sum_{Z \in \vec{\nabla}(\vec{I}(D))} |Z|\\
&= \textstyle\sum_{Z \in (\mathcal{Z}(S)-P)} |Z|\\
&= \textstyle\sum_{Z \in \mathcal{Z}(S)} |Z| - \sum_{Z \in P} |Z|\\
&= |\mathcal{C}(S)| - \textstyle\sum_{Z \in P} |Z|.
\end{aligned}$$

This is not possible unless $P = \emptyset$, in which case it would become $\mathcal{Z}(S) = \vec{\nabla}(\vec{I}(D))$, and $|D| \geq |\mathcal{Z}(S)|$. Hence proved. □

2.2. A Learning Mechanism Based on Data-dependencies

Consider a task instance T_1 with end points $(\vec{s}_1, \vec{s}_2)$ in the configuration space. If we express T_1 as a point function f, then we could write $\vec{s}_2 = f(\vec{s}_1)$. An algorithm instance A_1 that solves T_1 would typically implement the functionality of f thereby representing a path between $\vec{s}_1$ and $\vec{s}_2$. If there exists more than one way to implement f, then there exists more than one path between $\vec{s}_1$ and $\vec{s}_2$. Such a set of paths might be denoted by $f(\vec{s}_1, \alpha)$ with $f(\vec{s}_1, 0)$ representing some arbitrary path chosen to be treated as reference path.

Further, if we select some function $\eta(\vec{x})$ that vanishes at $\vec{x} = \vec{s}_1$ and $\vec{x} = \vec{s}_2$, then a possible set of varied paths is given by

$$f(\vec{x}, \alpha) = f(\vec{x}, 0) + \alpha\, \eta(\vec{x}).$$

It should be noted that all these varied paths terminate at the same end points, that is, $f(\vec{x}, \alpha) = f(\vec{x}, 0)$ for all values of α.

However, when we try to consider another task instance T_2 to be represented with these variations, we need to make them less constrained. The tasks T_1 and T_2 would not have the same end points in the configuration space and hence there would be a variation in the coordinates at those points. We can, however, continue to use the same parameterization as in the case of single task instance, and represent the family of possible varied paths by

$$f_i(\vec{x}, \alpha) = f_i(\vec{x}, 0) + \alpha\, \eta_i(\vec{x}),$$

where α is an infinitesimal parameter that goes to zero for some assumed reference path. Here the functions η_i do not necessarily have to vanish at the end points, either for the reference path or for the varied paths. Upon close inspection, one could realize that the variation in these family of paths is composed of two parts.

1. Variations within a task instance due to different algorithmic implementations.
2. Variations across task instances due to different initial system point configurations.

The learner can overcome the first type of variations by observing that the end points, and their corresponding zones, are invariant to the paths between them. In this regard, all the system points that belong to the same initial, final zone pair could be learned with a single algorithm instance. However, for the second type of variations, the learner may not be able to overcome them without any prior knowledge of the task. All the different instances of the task would have different corresponding zones for their end points and hence they need to be remembered as they are.

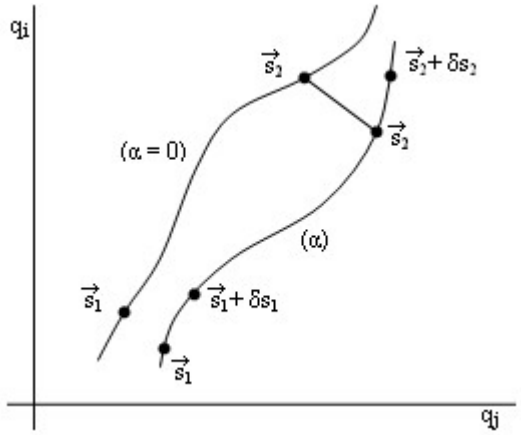

Figure 2. Schematic illustration of path variations across task instances in configuration space

Below we present a mechanism that uses these concepts of variations in the configurations paths to infer solutions to the unknown problem instances based on the solutions to the known problem instances. This results in a learning like behavior where the known problem-solution configuration pairs form the training set samples. Such samples could be collected by the following procedure.

1. For a given problem identify the appropriate configuration space and number of dimensions.
2. Express the solution as a logical relation R_s in terms of coordinates of the configuration space.
3. Use R_s to identify an appropriate characteristic function F_s to form a solution zone.
4. Use F_s in deciding the characteristic functions for other zones and the number of dimensions for zone-space.
5. Define the operator ∇ to map the system points from configuration space to the zones in zone-space.
6. Define an appropriate variation operator δ in the configuration space such that variations in the known problem configurations would give clue to the variations in the solution configurations, such as, $solution(x + \delta x) = solution(x) + \delta x$.
7. Construct the sample problem-solution configuration pairs by using any traditional algorithm. The

samples should be such that all zones are represented.

Once we have all the required data samples with us, the training procedure is simple and straightforward in that all that is needed is to mark each of the sample problem configurations as the reference configuration for the corresponding zone and remembering the respective solution configurations for those references. We can use a memory lookup table to store these reference solutions. The procedure is as follows.

For each data sample $D_i = (\vec{p_i}, \vec{s_i})$
{
  Let $Z = \nabla(p_i)$;
  RefProbConfig$[Z] = \vec{p_i}$;
  RefSolConfig$[Z] = \vec{s_i}$;
}

Once the training is over, we can compute the solution configuration $\vec{s}$ for any given problem configuration $\vec{p}$ in the configuration space as follows.

1. For the given problem configuration $\vec{p}$, apply the operator ∇ and find the zone $Z = \nabla(\vec{p})$;
2. Get the reference problem configuration $\vec{p_i}$ = RefProbConfig$[Z]$, and compute the variation $\delta(\vec{p}, \vec{p_i})$;
3. Compute the required solution configuration from the reference solution configuration by applying the variation parameter as:

$$\vec{s} = \text{RefSolConfig}[Z] + \delta(\vec{p}, \vec{p_i});$$

2.3. An Example Problem

To explain how these concepts of variations in the configuration paths could be used in practice, we consider an example problem of *sorting*. We outline a procedure that implements sorting based on the concepts we have discussed till now.

The reason behind choosing sorting as opposed to any other alternative is that the problem of sorting has been well studied and well understood, and requires no additional introduction. However, it should be noted that our interest here is, rather to explain how zones can be constructed and used for the sorting problem, than to propose a new sorting algorithm; and hence we do not consider any performance comparisons. In fact, the procedure we outline below runs with $O(n^2)$ time complexity requiring $O(2^{n^2})$ memory, thus any performance comparisons would be futile.

To start with, we can consider the task of sorting as being represented by its instances such as $\{(3,5,4),\ (3,4,5)\}$, where the second element $(3,4,5)$ represents the *sorted result* of first element $(3,5,4)$. We can consider these elements as points in a $3-$dimensional space.

Thus in general given an array of n integers to be sorted, we can form a system with $n-$coordinate axes resulting in an $n-$dimensional configuration space. If we assume that each element of the array can take a value in the range $[0, N]$, where N is some maximum integer value, then there would be a total of N^n system points in the configuration space. That is, following our notation from section 2.1, $|\mathcal{C}(S)| = N^n$. To construct the corresponding zone-space for this configuration space, consider the following mathematical specification for sorting,

$$\forall_{i=1}^{n} \forall_{j=1}^{n} \left[\, i < j \ \Rightarrow\ a[i] < a[j] \,\right],$$

where a is an array with n integers. This specification represents a group of conditions that need to be satisfied by the array if it has to considered as being in sorted order. Now, we can use this specification in identifying the following.

1. Number of dimensions of zone-space: The specification involves two quantifiers $\forall_{i=1}^{n}$ and $\forall_{j=1}^{n}$, with an additional constraint $i < j$. Thus the valid values could be $i = 1, \ldots, n, j = i+1, \ldots, n$, resulting in a group of $n \times (n-1)/2$ conditions to be accounted for. Each condition would form one coordinate axis in the zone-space and hence we have $n \times (n-1)/2$ axes.
2. Range of each axis of zone-space: Since each axis is formed out of the condition $(a[i] < a[j])$, with various values of i, j representing various axes, the range of each axis would be defined by the number of possible conditions $(a[i] < a[j]), (a[i] = a[j])$ and $(a[i] > a[j])$, which is *three*. Hence the range of each axis $r_i = 3$.
3. Operator ∇ : Each zone is a point in zone-space with $n \times (n-1)/2$ coordinates. To find these coordinate values we need to evaluate $n \times (n-1)/2$ conditions (one for each axis) as below.

```
for(int i=0,r=0; i<n; ++i)
    for(int j=i+1; j<n; ++j,++r)
      ZCoord[r]=((a[i]=a[j])?0:((a[i]<a[j])?1:2));
```

4. Variation operator δ : We implement the variation operator using the differences between relative array positions of numbers before and after sorting. We can use the array indexing and de-indexing

operations for this purpose. For example, sorting $a = \{3, 5, 4\}$ produces $\vec{a} = \{3, 4, 5\}$, which gives us a variation in the indices of elements from $(0, 1, 2)$ to $(0, 2, 1)$. Thus we can use our variation operator to express $\vec{a}$ as, $\vec{a} = \{\, a[0],\, a[2],\, a[1]\,\}$.

Once we have these necessary operators with us, we can start assigning the reference $(unsorted, sorted)$ configuration pairs for each zone by using any traditional sorting algorithm such as *heapsort* or *quicksort*, as shown below.

```
for(int i=0; i < nSamples; ++i)
{
    GetZCoord(Unsorted[i], ZCoord);
    quicksort(Unsorted[i], Sorted[i]);
    SetRefConfig(ZCoord, Unsorted[i], Sorted[i]);
}
```

It should be noted that we have $3^{n\times(n-1)/2}$ zones in the zone-space and hence we need so many sample $(unsorted, sorted)$ pairs as well. However, once we complete the training with all those samples, we can use the following procedure to sort any of the N^n possible arrays.

```
void LSort(int nArray[], int nSize, int nSorted[])
{
  GetZCoord( nArray, ZCoord );
  GetRefConfig( ZCoord, RefProb, RefSol);
  for(int i=0; i < nSize; ++i)
    nSorted[i] = nArray[RefSol[i]];
}
```

2.4. Limitations

Having presented the mechanism for learning based on the concepts of variations in the system configuration paths, here we discuss the limitations of this approach.

Disadvantages:

1. As could be easily understood, the concepts of *configuration space* and *zone-space* form the central theme of this approach. However, it may not be always possible to come up with appropriate configuration space or zone-space for any given problem. In fact, for many tasks such as face recognition etc... we readily do not have any clues for logical relations among the data attributes. This is one of the biggest limitations of this approach.
2. To present the learner with some sample configurations, we assumed the existence of an algorithm that could solve the task at hand. However, this assumption may not hold at all times. Once again, face recognition is an example.
3. The memory requirements are too high. We have already seen that we need $3^{n\times(n-1)/2}$ samples to correctly learn the sorting task.

However, given the goal of mimicking a human being and the scope of abstract concepts the agents have to learn from human beings, and given the virtually unlimited number of problem instances that could be solved by this learning mechanism, the memory requirements should not become a problem at all (note that the memory requirements do not depend on N but on n, so there is no upper limit to the number of problem instances that can be solved correctly). Further advantages are as follows.

Advantages:

1. Independent of the order of training data samples. In this method, the learner is invariant to the order in which it receives the data samples. All that a learner does with a data sample is, compute the corresponding zone and mark the sample as a reference for that zone. This process clearly is independent of the order of the data samples and hence gives the same results in all circumstances. It should be noted that the traditional learning algorithms does not guarantee any such invariance.
2. Additional samples do not create any bias. If there exists more than one sample per zone, the characteristic functions of zones guarantee that they all would produce the same results as that of first sample. Hence the training would not be biased by the presence of additional samples. Further, a sample could be repeated as many times as one wants without affecting the training results. This is useful for situations where a robot might be learning from real world, where some typical observations (such as the changing traffic lights, flow of vehicles etc...) would get repeated more frequently compared with some rare observations (such as earth quakes or accidents etc...). Traditional learning algorithms fail to provide unbiased results in such situations.
3. Non-probabilistic metrics and accurate results. To meet the demands of artificial systems, the metrics we have devised are completely deterministic and are void of any probabilistic assumptions and thus can be adapted to any suitable system.

4. Expandable to multi-task learning. Though we have concentrated on learning a single task in this discussion, there is nothing in this method that could prevent the learner from learning more than one task at the same time. For example, once an agent learns to sort in ascending order ($SASC$), it can further learn to sort in descending order ($SDSC$) simply by computing the new variation operator δ_{SDSC} directly from (δ_{SASC}), instead of from new sample problem-solution configuration pairs. This saves the training time and cost for $SDSC$. However, to implement this feature the agent should be informed of the relation between the tasks a priori. Smart agents that can automatically recognize the relation among tasks based on their configuration spaces should be an interesting option to explore further in this direction.
5. Knowledge transfer. All the knowledge of the learner is represented in terms of reference configurations for individual zones. Any learner who has access to these reference configurations can perform equally well as the owner of the knowledge itself, without the need to go through all the training again. This could lead to the concept of *tradable knowledge* resources for agents.
6. Perfect partial learning. Just as additional samples do not create bias, lack of samples also would not create problems for learning. A training set with *quantity* less than 100% would still give correct results as long as the problem instance at hand is from one of the learnt zones. That is, whatever the agent learns, it learns perfectly. This feature comes handy to implement low cost *bootstrapping robots* with reduced features and functionality which can be used as "data sample suppliers" for other full-blown implementaions. This concept of bootstrapping robots is one of the fundamental concepts of artificial life study in that it might invoke the possibility of self-replicating robots (Freitas & Gilbreath, 1980; Freitas & Merkle, 2004).

3. Conclusions

The notion of data-independence for an algorithm speaks for constant execution paths across all its instances. A variation in execution path is generally attributable to the variations in the nature of data. When a problem and its solution are viewed as points in a system configuration, variations in the problem configurations can be used to study the variations in the solution configurations and vice versa. These variations could be used to infer solutions to unknown instances of problems based on the solutions to the known instances.

This paper analyzed the problem of data-dependencies in the learning process and presented a learning mechanism based on the relations among data attributes. The mechanism constructs a Cartesian hyperspace, namely the configuration space, for any given task, and finds a set of paths from the initial configuration to final configuration that represents different instances of the task. As part of the learning process the learner gradually gains information from data samples one by one, till all data samples were processed. Once such a transfer of information is complete, the learner can solve any instance of the task without any restrictions.

The mechanism presented is independent of the order of data samples and has the flexibility to be expandable to multi-task learning. However, the practicality of this approach may be hindered by the lack of appropriate algorithms that could provide sample instances. Further study to eliminate such bottlenecks could make this a perfect choice to implement learning behavior in artificial agents.

References

Angluin, D. (1992). Computational learning theory: survey and selected bibliography. *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing* (pp. 351–369). New York: ACM Press.

Balmer, M., Cetin, N., Nagel, K., & Raney, B. (2004). Towards truly agent-based traffic and mobility simulations. *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 60–67). Washington, DC, USA: IEEE Computer Society.

Brooks, R. A. (1991). Intelligence without reason. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)* (pp. 569–595). San Mateo, CA, USA: Morgan Kaufmann publishers Inc.

Brugali, D., & Sycara, K. (2000). Towards agent oriented application frameworks. *ACM Computing Surveys*, *32*, 21–27.

Bryson, J. J. (2003). Action selection and individuation in agent based modelling. *Proceedings of Agent 2003: Challenges of Social Simulation*.

Cliff, D., & Grand, S. (1999). The creatures global digital ecosystem. *Artificial Life*, *5*, 77–93.

Collins, J. C. (2001). *On the compatibility between physics and intelligent organisms* (Technical Report DESY 01-013). Deutsches Elektronen-Synchrotron DESY, Hamburg.

Decugis, V., & Ferber, J. (1998). Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. *AGENTS '98: Proceedings of the second international conference on Autonomous agents* (pp. 354–361). New York, NY, USA: ACM Press.

Franklin, S. (2005). A "consciousness" based architecture for a functioning mind. In D. N. Davis (Ed.), *Visions of mind*, chapter 8. IDEA Group INC.

Freitas, R. A., & Gilbreath, W. P. (Eds.). (1980). *Advanced automation for space missions*, Proceedings of the 1980 NASA/ASEE Summer Study. National Aeronautics and Space Administration and the American Society for Engineering Education. Santa Clara, California: NASA Conference Publication 2255.

Freitas, R. A., & Merkle, R. C. (2004). *Kinematic self-replicating machines*. Georgetown, TX: Landes Bioscience.

Goldstein, H. (1980). *Classical mechanics*. Addison-Wesley Series in Physics. London: Addision-Wesley.

Kamareddine, F., Monin, F., & Ayala-Rincón, M. (2002). On automating the extraction of programs from proofs using product types. *Electronic Notes in Theoretical Computer Science*, *67*, 1–21.

Katsuhiko, T., Takahiro, K., & Yasuyoshi, I. (2002). Translating multi-agent autoepistemic logic into logic program. *Electronic Notes in Theoretical Computer Science*, *70*, 1–18.

Kearns, M. J. (1990). *The computational complexity of machine learning*. ACM Distinguished Dissertation. Massachusetts: MIT Press.

Lau, T., Domingos, P., & Weld, D. S. (2003). Learning programs from traces using version space algebra. *K-CAP '03: Proceedings of the international conference on Knowledge capture* (pp. 36–43). New York, USA: ACM Press.

Laue, T., & Röfer, T. (2004). A behavior architecture for autonomous mobile robots based on potential fields. *RoboCup 2004*. Springer-Verlag.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*.

Lopez, R., & Armengol, E. (1998). Machine learning from examples: Inductive and lazy methods. *Data & Knowledge Engineering*, *25*, 99–123.

Maes, P. (1989). How to do the right thing. *Connection Science Journal*, *1*.

McCauley, J. (1997). *Classical mechanics*. Cambridge University Press.

Moses, Y. (1992). Knowledge and communication: A tutorial. *TARK '92: Proceedings of the 4th conference on Theoretical aspects of reasoning about knowledge* (pp. 1–14). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Raedt, L. D. (1997). Logical settings for concept-learning. *Artificial Intelligence*, *95*, 187–201.

Ramamurthy, U., Franklin, S., & Negatu, A. (1998). Learning concepts in software agents. *From Animals to Animats 5: Proceedings of The Fifth International Conference on Simulation of Adaptive Behavior*. Cambridge: MIT Press.

Ray, T. S. (1991). *Artificial life ii*, chapter An Approach to the Synthesis of Life. Newyork: Addison-Wesley.

Ray, T. S. (1994). Evolution, complexity, entropy and artificial reality. *Physica D*, *75*, 239–263.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, *21*, 25–34.

Schmidhuber, J. (2000). *Algorithmic theories of everything* (Technical Report IDSIA-20-00 (Version 2.0)). Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Manno-Lugano, Switzerland.

Wilson, S. W. (1994). Zcs: a zeroth level classifier system. *Evolutionary Computation*, *2*, 1–18.

Zurek, W. H. (1989). Algorithmic randomness and physical entropy. *Physical Review A*, *40*, 4731–4751.