

Familiar Problem Space (FPS): A Novel Approach for Brain-like Learning

Zeeshan-ul-hassan Usmani

Florida Institute of Technology
Department of Computer Sciences
Melbourne, Florida, USA

S. M. Akber Zaidi

NED University of Engineering & Technology
Department of CS & Information Systems
Karachi, Pakistan

Abstract - *Lazy learning methods search for the match, while there may be no exact match, so the best match is called for. Unfortunately, there is no general way to search memory for the best match without examining every element of memory. It is proven that brain can not traverse more than 100 neurons in less than 200 milliseconds which we need to solve most of our routine decisions. This paper (inspired by theories of cognition and human brain learning) explains the model of brain learning and then propose a new strategy to learn and classify examples with additional advantages over traditional Back-propagation Neural Network.*

Keywords: Learning Models, Learning by example, Neo-Cortex Learning, Hierarchical Learning, Cognitive Model

1 Introduction

When encountering a new problem situation, a person is reminded of past situations that bear strong similarity to the present problem. The type of reminding experiences serve to retrieve behaviors that were appropriate in earlier problem solving episodes, whereupon past behavior is adapted to meet the demands of the current situation. Similarities among previous and current situations as well as successful application of modified plans can serve as the basis for generalization. This paper explores the capability of this human learning behavior by applying it to machine learning data sets and compares the result with traditional Back-propagation Neural Network algorithm.

Proposed algorithm is trying to improve on four particular areas as they are the disadvantages of eager learning algorithms and Memory Based Reasoning (MBR):

1. Connectionist models/Eager learning methods like ANN with Back-propagation average the values of parameters and may permanently lose some information [3].
2. Speed of prediction on repetitive instances does not improve as evident in real life.

3. Learning is always limited until knowledge-base can learn to extend themselves from experience
4. It is hard to predict incase of incomplete information.

2 How Brain Learns

According to Francis Crick (Co-discoverer of the structure of DNA) – Neuroscience is a lot of data without a theory [5], although it is still a mystery to know how exactly the brain works, we can see and understand the different parts of our brain. For this paper, we are only interested in the learning part: we do not care about respiratory, blood pressure and sexual controls through the brain at this point of time.

The brain does not compute [2], it just maps the new instance with any of previously present instances and follows the same set of commands or controls to perform new tasks with some modifications.

Consider the following example from Hawkin’s book [5] to get an idea of how “computing”, and “using memory” to reach the solution for the same problem, are different. Suppose the task is to catch a moving ball through the air. Someone throws it to you and in a fraction of second you catch it in the air. This is a mundane task for us and we perform it on a daily basis, but it turns out to be very difficult when we want to program a robot to do the same. In this programming task, you have to calculate the speed, angles, velocity, force and weight of the ball and air resistance etc. to make an intelligent guess about the future position of the ball, and you can not wait to perform this calculation - you have to move your robot as soon as the ball is tossed in the air with very poor prediction of its next location. You have to calculate some mathematical equations to find out this answer along with set a of commands that moves the robot arm to catch the ball and you have to consider each and every joint of the robot, arms, legs, fingers etc. This computationally expensive procedure has to be repeated multiple times as the ball reaches to

robot. If the robot performs its calculations first, then it is too late to catch the ball. This program requires thousands, if not millions, of calculations; it can be programmed, but does our brain do this in same way? Nature always comes up with very simple and elegant ways to perform tasks.

Our brain uses memory to catch the ball. When we see the ball thrown towards us, our brain recalls previously stored example(s) of catching a ball along with the set of muscle commands need to perform the task. 'Practice makes perfect' is the best analogy to define the learning of the brain, there is no program in the brain to catch the ball. It is just the past instances where you catch a ball. No computation takes place, no gradient descent moves forward and backward and there is no blame factor for individual neurons. Instead it's just the mapping of a new event to the previously stored one; the old fetched set of commands (Learned behavior) is adjusted to new situations. This is what neurologist called "Invariant Memory" [5].

Brain (Neo-cortex) uses memory to perform tasks [2]. We have memories in computers too, but Neo-cortex is different from computer memory in 4 ways [5]

1. Stores sequence of patterns.
2. Recalls patterns auto-associatively.
3. Stores patterns in an invariant form.
4. Stores patterns in a hierarchy.

Our brain stores examples or instances as sequence of happenings. For example to tell the favorite story one starts from first part, then second, then third and so on. One cannot tell the story in parallel form, neither can one understand. It is not because the language and words are serial, it is because we store them in serial fashion, and if we do not recall them in the same way we cannot retrieve them from our memory. To recall the memories one have to walk through them in sequential manner, in much the same way one experienced it. And this is not restricted to vision only. All instances you get from any senses (smell, touch, hear etc) are stored in the same way. One has to walk through the temporal sequences of how to do things. One pattern evokes another pattern, which evokes another and so on.

The second characteristics of brain learning Auto-associative memory system is one that can recall complete patterns when given only partial or incomplete information. Our brain does the rest, what neurologists call 'filling' [2]. At any time, any part of a sequence can activate the whole. Our brains fill in what it miss with

what it expect to hear. It is a well-established fact [5] that we do not actually hear all the words we perceive.

The third characteristic of learning is to store patterns in 'Invariant Form'. The world and life around us is not perfect at any time, there are always variances and changes occur all the time. One might see several breeds of dogs in life but still able to classify them as dogs - it is because of invariant storage of our brains.

Fourth characteristic is more or less connected with the third one; our brain uses hierarchical framework (we have 6 layers in our Neo-cortex) to store the examples from specific-to-general form. What we perceive is a combination of what we sense and of our brain's memory-derived predictions. Brain makes continuous predictions after each single bit of perceived information. This is how we are able to notice the slightest change in our professor's haircut or a new color painting in the library.

3 Familiar Problem Space (FPS)

The motivation for the proposed algorithm is based on the biology of brain's learning. It is more close to the actual mechanism of brain learning compared to BPNN.

3.1 Algorithm

1. Store all training examples in examples-set-seen-so-far (Positive and Negative matrices), and remove duplications on each hierarchical level.
2. For any new instance, classify it according to following prediction

$$\text{Predict (x) = Max (Instance(n), Instance(n-1), \dots, \text{Instance}(1))}$$

3. Store the new instance in examples-set-seen-so-far with given prediction by step 2.

The algorithm goes like this - make a set of all examples and make two corresponding matrices for negative and positive examples. Store all positive examples in positive and negative examples in negative matrix and remove duplications. The equation in step 2 just reads the matrices where 'predict' is the function to predict the target value for a given instance. 'Max' selects the most common value from the row. Instance (n) represents the example with n parameters, Instance (n-1) means example with n-1 parameters and this list goes to examples with only one parameter.

Instance() compares the given instance with positive and negative n-value parameter examples: if found in either one of the matrix returns the target value

of that matrix (Positive, Negative), if found in both returns Null, and following n-1, n-2 ...1 functions do the same for n-1, n-2... 1 attribute examples. At the end max() selects the most common value (+ve, -ve) and return it as a prediction for that instance. Positive (true) or Negative (false) have priority over Null votes (Null votes does not count). If all values are Null (meaning it is an entirely new instance) you can bias the algorithm to return Null (meaning I do not know) or T (True) or F (False). FPS returns the F (False) for entirely new instance.

3.2 Coding Issues

There could be different ways one prefers to code this algorithm, like the value you choose for representation of Null Row. May be asking the user to input the classification to move on, and the way how one codes the matrices. It could be different matrices for each number of parameters or a single matrix with n parameters while restricting the algorithm to read only n columns when calculating the predict function. One could prefer to delete the instance from both matrices if found exactly the same in both (meaning same example yields positive and negative results). One might want to add another column for counting the number of same examples to weigh more than the target value, which has more number of occurrences, and one can code a column for new and old instances indication. One can represent the same hierarchical abstraction idea in the form of tree structures. Whatever way one chooses to code is ok, as long as it considers the four ways of storing examples in the memory, as the brain does.

4 Results

The training and testing data for test run [1] are presented in Table 1 and Table 2. Proposed algorithm (FPS) will store the training data as presented in Table 3(a) and Table 3(b).

#	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes

#	Outlook	Temp	Humidity	Wind	Play
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No
15	Overcast	Hot	Normal	Strong	Yes
16	Rain	Hot	Normal	Weak	No
17	Sunny	Hot	Normal	Strong	Yes
18	Sunny	Hot	Normal	Weak	Yes

N=4	Overcast	Cool	Normal	Strong
	Overcast	Hot	High	Weak
	Rain	Cool	Normal	Weak
	Rain	Mild	High	Weak
	Rain	Mild	Normal	Weak
	Sunny	Cool	Normal	Weak
N=3	Overcast	Cool	Normal	
	Overcast	Hot	High	
	Rain	Cool	Normal	
	Rain	Mild	High	
	Rain	Mild	Normal	
	Sunny	Cool	Normal	
N=2	Overcast	Cool		
	Overcast	Hot		
	Rain	Cool		
	Rain	Mild		
	Sunny	Cool		
N=1	Overcast			
	Rain			
	Sunny			

N=4	Rain	Cool	Normal	Strong
	Sunny	Hot	High	Weak
	Sunny	Hot	High	Strong
	Sunny	Mild	High	Weak
N=3	Rain	Cool	Normal	
	Sunny	Hot	High	
	Sunny	Mild	High	
N=2	Rain	Cool		
	Sunny	Hot		
	Sunny	Mild		
N=1	Rain			

Table 4 shows the result of Back-propagation Neural Network and Familiar Problem Space (FPS) algorithms on testing data. Positive and Negative matrices keeps updating after each (testing) example. The parameters used for standard BPN implementation [4] are Learning-rate=0.95, Momentum=0.5, Tolerance=0.5, Input=4, Middle=2, Output=1

Table 4: Results		
Desired	BPNN (62.5%)	FPS (75%)
Yes	No	No
Yes	No	Yes
Yes	Yes	Yes
No	No	Yes
Yes	No	Yes
No	No	No
Yes	Yes	Yes
Yes	Yes	Yes

FPS is slightly more efficient in output, compared to BPNN (75% Vs 62.5%) but there are few more interesting observations. The training data has a unique attribute value overcast: if it is present, it can make any instance true. BPNN fails to recognize this generalization while FPS does. FPS has the ability to predict the target value in case of missing information or less number of parameters, while you need to have all input node values for BPNN. FPS is more robust to noisy data since it cancels out the result of noisy or misleading data. FPS maintains sequences, invariant representations, auto-associative memory and hierarchical structures like the brain does. In contrast to BPNN, that construct a general, explicit description of the target function when training examples are provided, FPS simply stores the training examples, therefore training time is almost none. BPNN is not biologically realistic in several ways, like the brain does not have any training period. It learns throughout the life and keeps updating the instance set observed so far, while BPNN has an initial learning period and then there is no update on instance set and no feedback either. Another advantage of FPS over BPNN is that it makes local predictions based on each individual instance; there is no general set of weights for all instance space.

There are few disadvantages of this approach as well, like the cost of classification can be high because all computations take place at the classification part, instead of learning part. Another disadvantage might be the large amount of memory requirements for data, but this can be accommodated by special data structures, and the number of stored instances is far less because of

sequential storing: FPS do not store instances which starts from any values other than outlook (Sunny, Rain, Overcast). Another way to deal with this problem is two techniques of *restriction* (goal restriction and predictor restriction) by retrieving subset of examples from the database as discussed by Craig Stanfill and David Waltz [6]. You can also prioritize different fields over one another to choose the best examples for your prediction.

5 Conclusion & Future Work

FPS is more close to brain learning mechanism. FPS algorithm knows that “it does not know” and that is an important property. It is also available from example one, and there is no wait for learning. No domain expert is required and it is very easy to understand, code and examine why it is performing like this or why not. There is no rule ever created in this approach as opposed to other learning approaches of deduction. The proposed algorithm (FPS) shows better results and fast learning as compared to back-propagation neural networks

Authors would like to extend this work to continuous value output, use of special data structures and comparison of FPS with instance-based learning algorithms like K-Neighbor, locally weighted regression and case-based reasoning [3]. Authors would also like to examine the effects of strict and loose ordering of seen examples, New Vs Old memories, Memory Loss and Permanent Vs Temporary memories over FPS algorithm.

6 References

- [1] Tom Mitchell, *Machine Learning*, McGraw-Hill Education, 1997.
- [2] Robert Sapolsky, *Biology and Human Behavior: The Neurological Origins of Individuality*, Stanford University, Course by Teaching Company, 2000
- [3] S. Russell & P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2nd edition, December 20, 2002
- [4] Timothy Masters, *Practical Neural Network Recipes in C++*, Morgan Kaufmann Academic Press, San Diego, USA, 1993
- [5] Jeff Hawkins and Sandra Blakeslee, *On Intelligence*, Time Book, Henry Halt and Company, New York, USA, 2004
- [6] Craig Stanfill and David Waltz, *Toward Memory-Based Reasoning*, Communications of the ACM, December 1986, Volume 29, Number 12