Modifying Upstart for Use in Multiclass Numerical Domains

Ronnie Fanguy and Miroslav Kubat [*]

Abstract

One of the research topics pursued by scientists specializing in artificial neural networks deals with the question of how to determine a neural network's architecture. In the work reported here, we resurrect Frean's Upstart (1990) that grows the network one neuron at a time. This algorithm is known to have some useful properties; however, it was originally developed only for applications with two classes and with training examples described by boolean attributes. To extend the usefulness of Upstart, we suggest modifications that facilitate the use of this paradigm in multiclass domains with numeric examples.

Keywords.
Pattern recognition, neural networks, architecture, Upstart

Introduction

The widespread use of artificial neural networks in pattern recognition was made possible by algorithms capable of inducing them from sets of pre-classified training examples (Rumelhart & McClelland, 1986). Many practical and theoretical aspects of neural networks, and of the methods used for their training, are now fairly well understood.

One of the research strands receiving attention in the last decade explores methods for automatic design of a neural network's layout. The number of neurons, as well as the number and interconnection of layers, is known to affect the network's learning behavior. Whereas small networks may lack the requisite flexibility, large ones are expensive to train and tend to overfit noisy training data. The conflicting nature of the involved trade-offs has motivated studies of methods to determine the architecture automatically. Techniques developed so far involve logic-based strategies that take advantage of prior knowledge expressed as production rules or decision trees, search-based strategies that rely on AI search (including genetic algorithms), and piecemeal techniques that create the network one neuron at a time. For an overview, see Kubat (2000).

A good example of the "piecemeal" category is Frean's Upstart (1990) that capitalizes on the observation that a single neuron is much cheaper to train than the whole network. In his implementation, each neuron focuses on a somewhat different aspect of the pattern to be learned by using a different labeling of points. The training can be accomplished by perceptron learning (Rosenblatt, 1958), by gradient descent techniques (Widrow and Hoff, 1960), or by some more traditional statistical method (Duda and Hart, 1973).

For domains where training examples are described by vectors of *boolean* attributes, recursively calling Upstart is guaranteed to reach zero error rate on the training set, provided that this set contains only examples from two classes and that they do not conflict (two examples with the same description should not have different class labels). Unfortunately, the restriction to boolean attributes and two-class domains is too severe for many practical domains. This is why we embarked on the search for possible modifications and improvements that would broaden the scope of possible applications of this approach.

To make the paper self-contained, the next section summarizes Frean's original algorithm and briefly dicusses its limitations. After this, we describe our improved version of this technique and report experiments illustrating its behavior in benchmark domains. The final section concludes with a brief discussion.

Frean's Upstart

Description of Upstart

In the machine learning tasks addressed here, the input consists of a set of training examples that have the form of pairs $[\mathbf{x}, c(\mathbf{x})]$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a vector describing an *example*, and $c(\mathbf{x})$ is this example's class label. The variables x_i are referred to as *attributes*. Some of these attributes are boolean, others can be numeric. The space defined by the n-dimensional vectors is called the *instance space*. The training examples have been pre-classified using a function $c(\mathbf{x})$ that acquires the form $c : R^n \to L$, where L is the set of class labels. A machine learning algorithm takes the training set and induces from it a *classifier*, $h : R^n \to L$

[*]R. Fanguy is with the Department of Computer Science, Nicholls State University, Thibodaux, Lousiana, and M.Kubat is with the Department of Electrical & Computer Engineering, University of Miami,Coral Gables, Florida.

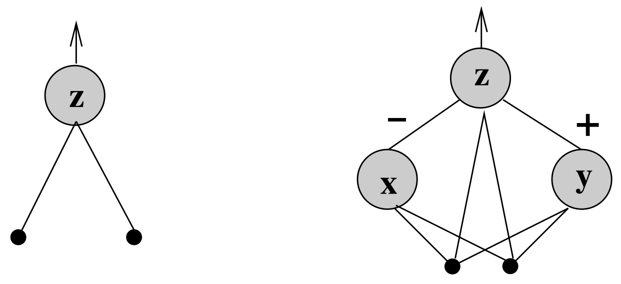

Figure 1: Upstart introduces two "consultant" nodes to rectify the output of the previous node, **z.** Neuron **x** attempts to correct "wrongly ON" states; neuron **y** attempts to correct "wrongly OFF" states.

true label	**z**'s output	state	label for **x**	label for **y**
0	0	correctly OFF	0	0
0	1	wrongly ON	1	0
1	0	wrongly OFF	0	1
1	1	correctly ON	0	0

Table 1: The possible states when **z** predicts the classification of an example, and how this prediction determines the training labels of the two consultant nodes.

with the goal of minimizing the probability of $h(\mathbf{x}) \neq c(\mathbf{x})$ for a randomly drawn **x**.

Let us, for the time being, focus on the binary task with $L = \{0, 1\}$, where 1 denotes *positive* examples of some class and 0 denotes *negative* examples. To start with, Upstart roughly approximates the class by a single neuron, **z.** When used to classify examples, **z** can fail in two different ways. If its output is 1 for a negative example, the neuron is *wrongly ON*; if its output is 0 for a positive example, the neuron is *wrongly OFF.* Frean's idea is to reduce the frequency of these errors by the use of "consultant" neurons (see Figure 1) that attempt to flip-flop **z**'s output whenever an error would occcur.

More specifically, the task of **x** is to change **z**'s output in the case of examples for which **z** is wrongly ONs and the task of **y** is to change **z**'s output in the case of examples for which **z** is wrongly OFF. Ideally, **x** outputs 1 for those and only those examples where **z** is wrongly ON (outputting 0 for all other examples). The link between **x** and **z** then has a large negative weight to reduce **z**'s input accordingly. Likewise, **y** should output 1 for those and only those examples where **z** was wrongly OFF and the link between **y** and **z** is assigned a large positive weight.

The weights of **x** and **y** are induced by a learning algorithm that employs the same training examples that were used for the induction of **z,** but now with changed class labels: when **x** is trained, all examples for which **z** was wrongly ON will be labeled with 1 and the remaining examples will be labeled with 0. When **y** is trained, all examples for which **z** was wrongly OFF will be labeled with 1 and the remaining examples will be labeled with 0. This situation is shown in Table 1.

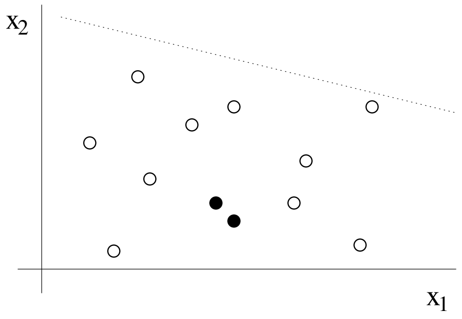

Figure 2: No linear function can separate the dark circles from the light circles. Any attempt to do so will result in more than two misclassifications.

For illustration, suppose that the neuron **z** outputs 1 when presented with a negative example (the wrongly ON situation). When **x** is trained, the example will be labeled with 1; when **y** is trained, the same example will be labeled with 0. Conversely, if **z** outputs 0 for a positive example (the wrongly OFF situation), then the example will be labeled with 0 during the training of **x** and with 1 during the training of **y**. Finally, the label of any example that has been correctly classified by **z** will be 0 for the training of both consultants (**x** and **y**).

If the consultant neurons do not manage to classify the re-labeled examples correctly, Upstart calls the same routine recursively, creating one pair of consultant neurons for **x** and another pair for **y**. This time, the re-labeling will reflect the wrongly ONs and wrongly OFFs of the neurons **x** and **y,** respectively. As long as some wrongly ONs or wrongly OFFs can be observed on the part of any consultant (with respect to the re-labeled examples), Upstart keeps expanding the network, attempting to reach a state where all training examples are classified correctly.

Frean (1990) was able to prove that if all attributes acquire only boolean values, then the procedure just described will result in a neural network that has zero error rate on the training set (unless two examples with the same description have different class labels). This network is created in a finite number of recursive calls.

Upstart's Limitations

From the perspective of real-world applications, Upstart suffers from two major limitations. The first of them is that Frean conceived his idea with only two-class domains in mind, whereas many realistic domains require correct identification of examples from several different classes. Therefore, modifications that would generalize the original algorithm for use in multiclass domains are needed.

The second limitation is perhaps more severe: the algorithm is guaranteed to converge only when all attributes are boolean. In numeric domains, the consultant nodes often fail to improve **z**'s performance. The reason for this deficiency is easy to see. Consider the case from Figure 2 where several training examples are described by two numeric attributes, x_1 and x_2. Most of the examples belong to the class depicted by light circles and only two examples belong to the class depicted by dark circles. The reader can see that no linear function can separate these classes without misclassifying at least two examples.

A likely solution in this particular case is indicated by the dotted line. It turns out that both dark circles find themselves on the same side as the light circles, and are thus misclassified by **z**. Suppose this means that **z** is wrongly ON for these two examples. In an attempt to correct this error, Upstart will train a consultant **x** to output 1 for the two examples and 0 for any other example. However, as any linear function is likely to commit at least the same two errors, adding **x** to the network will not help. As a matter of fact, it can even increase the number of misclassifications. Fanguy (2001) discusses at length the phenomenon of error-rate "oscillations": one consultant neuron increases the error rate, a subsequent consultant decreases it, yet another increases it, and so on. In this event, the learning process does not bring about any improvement, and the size of the neural network may grow infinitely large. Even if the process is halted, the same problem occurs when we attempt to train **y** to correct wrongly OFF errors. This, again, severely limits Upstart's applicability in real-world domains.

Solution

Our solution rests on three simple modifications, described separately in the following paragraphs. The whole algorithm is summarized in Figure 3.

Restricting the training sets used for the induction of consultant nodes

We will explain the principle on a two-class domain. Later in this section, we will show how it generalizes to multiclass domains.

Recall that the task of the consultant **x** is to prevent the output neuron **z** from becoming wrongly ON. Obviously, **z** can be wrongly ON only for an example labeled by **z** as positive, never for an example labeled as negative. The requested behavior is thus achieved if **x** outputs a "1" in response to examples for which **z** is wrongly ON and a "0" otherwise. A key prerequisite for an example to be wrongly ON is that **z** classifies the example as positive. The learning algorithm should take advantage of this fact when training consultant nodes. It is pointless to include the examples **z** classifies as OFF (negative) in the training set of **x**, as **x** is to specialize in correcting the wrongly ON errors of **z**. Following the same logic, the consultant **y** should be trained only using the examples for which **z** predicts OFF. Building on this observation, our first modification is to train each of Upstart's consultant neurons only on the training examples from a single predicted class: **x** is induced from re-labeled examples that **z** classifies as positive and **y** is induced from re-labeled examples that **z** classifies as negative. The new label is "1" if **z** misclassified the example and "0" otherwise.

Restricting the training sets of the consultant neurons reduces computational costs because a smaller number of examples are used for the induction of each node. Moreover, this modification leads to another simplification in using Upstart networks. When **z** predicts that an example is positive, it is only required to use the wrongly ON consultant. There is little point in taking the computational effort to consult with the wrongly OFF consultant when the example is classified as positive anyway. Similarly, **z** is only required to use the wrongly OFF consultant when it predicts that an example is negative. In this way, classifying examples with an Upstart network is computationally less expensive, as we only use a small portion of the overall network.

By using a subset of the training examples for each subsequent consultant, we eliminate the danger of creating an infinitely large network. Although Upstart is still not guaranteed to converge to zero error rate if the attributes are numeric, our experiments indicate that learning approaches zero error rate (on the training set) in a finite-sized network.

Introducing a stopping criterion

For reasons explained earlier, Frean's original version of Upstart can in numeric domain lead to very large (perhaps even infinite) networks unless some reasonable stopping criterion is used. The restriction of training sets to a great extent eliminates this need.

In our implementation, the network growth was stopped either when no more consultants were needed (correct classification of all training examples) or when one of the following two conditions occurred: (1) the number of wrongly ON or wrongly OFF errors is less than some threshold or (2) when the depth of a branch is greater than some maximum depth threshold.

Perhaps due to the nature of the benchmark data we used, this simple approach turned out to be sufficient. Although we did experiment with more sophisticated criteria, we never observed any improvement in terms of classification accuracy. We were able to improve "compactness" of the created networks only at the cost of reduced classification accuracy.

Dealing with Multiclass domains

A single neuron can only separate two classes. To extend the idea to domains with N_C classes, we replaced this neuron with an entire layer of N_C neurons, one for each class. This change increases the number of possible types of error. Instead of just becoming wrongly ON (read: "wrongly positive") or wrongly OFF (read: "wrongly negative"), the generalized output node **z** can now be *wrongly-class-i* for any $i \in [1, N_C]$. As a matter of fact, it can even wrongly predict more than one class.

This means that, in multiclass domains, the information about misclassification is not sufficient to correct the prediction. To address this problem, we make the consultant a "specialist" for the examples assigned to it (e.g., the examples sharing a particular class label prediction by the parent node). Consultant nodes are thus specialists for the examples classified by their parent as class-i.

Figure 3: Modified Upstart.

```
Input:
  1. Set, $T$, of training examples from $N$ classes
  2. List, $L$, of class labels
  3. Depth, $D$, of the current node in the network

z = Upstart(T,L,D).
  1. Train $N$ neurons on $T$. These neurons constitute the
     current node, z.
  2. If $D$ is below a user-set threshold:
     For each class $i$:
       If the number of wrongly-class-$i$ examples exceeds
       a user-set threshold then
         i) Let T' be all examples labeled by z as class-$i$.
         ii) Let L' be the class labels associated with examples in T'
         iii) x = Upstart (T', L', D+1)
         iv) x becomes the wrongly-class-$i$ consultant of z.
```

Again, the algorithm is used recursively.

Using the Classifier

After making a prediction, a node may rely on its consultants to overturn the prediction, if necessary. First, node **z** decides that an example should be assigned class-i. Then, the consultant responsible for class-i either concurs with the decision or returns the corrected class label.

Experiments

The goal of the experiments is to demonstrate that the algorithm described in the previous section can induce a finite neural network that can achieve good classification accuracy in domains with numeric attributes. Moreover, the algorithm can now be used in multiclass domains.

As testbeds, we used benchmark data files from the Machine Learning Database Repository of the University of California, Irvine (Blake and Merz, 1998). Due to the nature of our research, we focused on domains with numeric attributes. Table 2 briefly summarizes the characteristics of the data.

Special attention was devoted to methodology. The benchmark data files are not sufficiently large for statistically safe comparisons of classification accuracies on unseen data. Since the popular t-tests are unreliable in connection with random subsampling of data files of this size, and since N-fold cross-validation on some of the smaller domains would entail very high variations, we opted for a compromise: we used 10-fold cross-validation, repeated 5 times for different partitionings of the data, and then averaged the results. This means that, in each run, 90% examples were used for training and the remaining 10% for testing. The methodology is sound as long as we are interested only in general tendencies and not in precise evaluation of statistical significance.

Table 2: Characteristics of the benchmark data sets used in the experiments.

Dataset	#examples	#attrib.	#classes
BCW	699	9	2
Pima	768	8	2
Aba3	526	10	2
Aba4	594	10	2
Bal2	625	4	2
Bcc1	68	10	2
Derm2	96	34	2
Derm3	108	34	2
Ion	351	34	2
Kbt2	900	15	2
Wdbc	569	31	2
Wpbc2	194	34	2
Wine	178	13	2
zoo	101	17	7
bal	625	4	3
glass	214	10	6
wine	178	13	3
hayes	132	5	3
house	506	13	9
seg	210	19	7

Apart from classical neurons implemented as linear threshold units (LTU), we experimented also with simplified "neurons" that test only a single attribute (SA). In the case of LTU, the neural weights were found using the technique of a pseudoinverse matrix (Duda and Hart, 1973) followed by one epoch of perceptron learning (Rosenblatt, 1958) adapted so that it minimizes a geometric mean of errors committed on individual classes. In the case of SA, we scanned for each attribute all possible binary splits and selected the attribute-split pair with the lowest value of the geometric means of the error rates.

The average classification accuracies (with standard deviations) are listed in Table 3. The classification accuracy is defined as the percentage of correctly classified *testing* examples (unseen during the training phase). The column headed by "LTU" contains the results achieved by the implementation with LTU neurons and the column headed by "SA" contains the results achieved by the implementation with single-attribute neurons. For reference, the table also gives the results achieved by the Quinlan's decision-tree generator See5 run on the same data[1]. The reason we show these results is to make sure that the induced neural network is not inferior to other pattern recognition paradigms.

The modified version of Upstart generates finite neural networks (although not really the classical feedforward neural networks). Moreover, the classification accuracy on unseen data appears to be acceptable. We want to avoid any claims related to the comparisons with decision trees—these

[1]Unlike C4.5, this new version, See5, outputs only results from *pruned* decision trees. The comparison is thus not totally fair because the Upstart networks have not been pruned.

Table 3: Experimental results of classification accuracy on testing data. Averages and standard deviations are from 10-fold cross validation repeated 5 times.

Dataset	LTU	SA	Decision Tr.
BCW	94.4±0.4	96.6±0.2	94.0±0.6
Pima	66.0±0.9	65.4±2.3	74.3±1.2
Aba3	90.9±0.8	89.7±1.2	92.8±0.4
Aba4	89.3±0.7	86.8±0.9	91.6±0.5
Bal2	95.6±0.3	72.9±2.0	84.2±1.0
Bcc1	95.0±2.2	88.0±2.2	85.6±2.0
Derm2	100.0±0.0	100.0±0.0	100.0±0.0
Derm3	99.1±0.1	100.0±0.0	100.0±0.0
Ion	84.7±0.9	87.7±1.5	90.3±0.7
Kbt2	83.4±0.8	86.2±1.0	86.7±0.9
Wdbc	95.0±0.8	93.1±0.5	94.1±0.6
Wpbc2	67.4±2.6	71.3±2.7	73.7±2.0
Wine	99.3±0.3	93.0±0.6	94.4±1.3
zoo	92.7±1.9	93.9±0.9	94.1±1.4
bal	92.4±0.5	70.1±2.3	78.5±0.5
glass	93.1±1.7	95.0±0.3	97.6±0.6
wine	97.2±0.6	90.5±1.9	92.6±2.2
hayes	66.2±3.3	60.5±3.3	79.3±1.4
house	35.9±2.4	42.2±1.4	55.5±1.4
seg	78.4±1.7	75.9±2.0	89.0±1.6

are two different approaches and, after all, none of them is expected to "beat" the other in all domains. Suffice to say that in 6-7 domains, decision trees did clearly better, and in 5-6 domains, the modified Upstart did clearly better. This means that the Upstart's mechanism does not create neural networks with poor classification performance.

Conclusion

The paper proposes a simple modification of Frean's Upstart algorithm for creating neural networks one neuron at a time. The modification facilitates the use of this algorithm in domains where training examples are described by numeric attributes and also in multiclass domains. Upstart addresses the problem of how to automate the design of a neural network's architecture. We believe that the broader use of this paradigm has so far been precluded mainly by its limitation to purely boolean domains. With the extensions presented in this paper, the approach can be easily employed by more realistic applications. The experimental results reported in the previous section are encouraging.

Having shown that the essence of the technique is viable, we hope to have persuaded the reader that Frean's idea indeed deserves more attention. For one thing, it can easily be extended to self-organization of more sophisticated entities. For instance, instead of mere neurons, one can think of networks of more powerful classifiers, such as decision trees. Studies of methods to combine several classifiers into one voting structure have received considerable attention during the last decade. The approach studied here can be understood as an alternative to such approaches as the popular "boosting" (Schapire, 1990).

Development of more advanced mechanisms for stopping the search and for pruning the resulting structure is likely to reveal further advantages of this simple and intuitive approach.

Acknowledgement

The research was partly supported by the grant LEQSF(1999-02)-RD-A-53 from the Louisiana Board of Regents.

References

Blake, C.L. and Merz, C.J. (1998). UCI Repository of machine learning databases `www.ics.uci.edu/~mlearn/MLRepository.html`. University of California Irvine, Department of Information and Computer Science

Duda R.O. and Hart, P.E. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York

Fanguy, R. (2001). *The Upstart Algorithm for Pattern Recognition in Continuous Multiclass Domains.* PhD disertation, University of Louisiana at Lafayette, Center for Advanced Computer Studies.

Frean M. (1990) The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation, 2,* 198–209

Kubat, M. (2000). Designing Neural Network Architectures for Pattern Recognition. *The Knowledge Engineering Review, 15,* 1–20

Rosenblatt, M. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65, 386–408

Rumelhart, D.E. and McClelland, J.L. (1986). *Parallel Distributed Processing*, MIT Bradford Press

Schapire, R.E. (1990). The strength of weak learnability. *Machine Learning, 5,* pp. 197–227.

Widrow, B. and Hoff, M.E. (1960). Adaptive Switching Circuits. *IRE WESCON Convention Record*, 96–104.